Deep Spiking Neural Network, Deep Liquid State Machine e Deep Echo State Network

André Furlan and Josué Ivan Gomes da Silva

Universidade Estadual Paulista Júlio de Mesquita Filho

2023

Deep Echo State Network

In 2001, two works were proposed independently and at different times and countries. However, both dealt with the concept of Reservoir Computing, with slightly different models. Echo State Networks were presented in a more general way, in a mathematical and engineering approach, demonstrating the use of Reservoir Computing for time series prediction tasks. While in the same year the Liquid State Machine model was proposed by Wolfgang Maass, Thomas Natschläger and Henry Markram, reaching a result very similar to the ESN.

Liquid State Machine

"Liquid State Machines can be a type of Reservoir Computing(RC) model, similar to Echo State Networks(ESNs), they can leverage the dynamics of a recurrent neural network with randomly connected neurons to process sequential and time-series data efficiently. LSMs were introduced as an extension of ESNs, incorporating additional principles inspired by biological neural networks." (SANKHALA, 2023)

Echo State Networks

Echo State Networks were proposed by Herbert Jaeger in his work "The 'echo state' approach to analyzing and training recurrent neural networks". In this work, he demonstrates a dynamics reservoir architecture with just 3 layers:

- ▶ Input Layer
- ▶ Reservoir Layer
- ▶ Output Layer

The objective of an Echo State Network is to maximize W_{out} based on the dynamics generated by the reservoir, recovering data from previous stages and allowing the identification of patterns based on the given history.
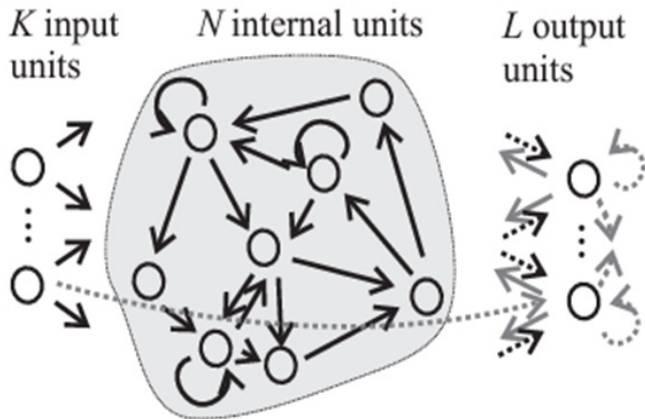
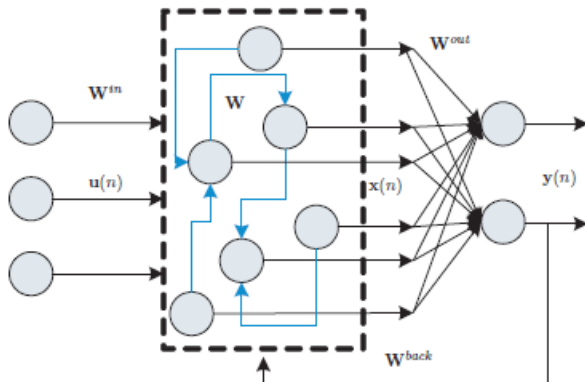Figure: The Jaeger model - Source:(JAEGER, 2001)

Figure: Source: (BOCCATO et al., 2011)

According to (BOCCATO et al., 2011) Consider a discrete-time recurrent neural network with K input units, N internal units and L outputs, as depicted in Figure 1. The vector $u(n) = [u_1(n)...u_K(n)]^T$ denotes the activations of the input units, which are transmitted to the internal neurons by means of a linear combination. The weight matrix $W^{in} \in R^{N \times K}$ specifies the coefficients of such combinations.

The internal layer, also called dynamical reservoir, is composed of fully-connected nonlinear units whose activations, given by $x(n) = [x_1(n)...x_N(n)]^T$, correspond to the network states, and are updated as follows:

$x(n + 1) = f(\mathbf{W}^{in}\mathbf{u}(n + 1) + \mathbf{W}\mathbf{x}(n) + \mathbf{W}^{back}\mathbf{y}(n))$,

where $\mathbf{W} \in R^{N \times N}$ brings the weights of the connections within the reservoir, $\mathbf{W}^{back} \in R^{N \times L}$ contains the weights associated with the connections that project the output samples back to the internal units, and $\mathbf{f}(\cdot) = (f_1(\cdot), ..., f_N(\cdot))$ denotes the activation functions of the internal units.

Finally, the outputs of the network, represented by the vector $y(n) = [y_1(n)...y_L(n)]^T$, are determined by the following expression: $y(n+1) = f^{out}(\mathbf{W})^o ut\mathbf{x}(n+1)$, where $\mathbf{W}^{out} \in R^{L \times N}$ is the output weight matrix and $\mathbf{f}(\cdot) = (f_1^{out}(\cdot), \dots, f_L^{out}(\cdot))$ especifies the activation functions of the output units.

According to (BOCCATO et al., 2011) Jaeger demonstrated two sufficient conditions regarding echo states:

- **The first condition** determines that for an RNN to present the echo state property, the largest singular value of the internal weight matrix W must be less than unity.
- **The second condition** establishes the non-existence of echo states in terms of the spectral radius (the largest among the absolute eigenvalues) of the internal weight matrix W: if $||W||| > 1$, then the network has no echo states

ESNs have gained notoriety and have been used to solve the most diverse problems. But the main focus of use is on:

- ▶ Time series forecasting.
- ▶ Signal processing.
- ▶ Pattern Recognition.
- ▶ Natural Language Processing (NLP).

The model used for testing is accessible on the website
⟨https://mantas.info/code/simple_esn/⟩ (LUKOSEVICIUS, 2012) from which it was originally derived. This model is designed to generate a neural network that, when provided with an audio file containing a melodic sequence as input, produces an output that closely resembles the original audio in its predictions. The resulting prediction is saved in a new file for comparison.

unesp

Based on the research carried out, it was demonstrated in the works read that Echo State Networks enable substantial gains when applied to prediction tasks involving historical data, such as time series prediction and Natural Language Processing. They are capable of recovering short-term memory states and reducing the possibility of overfitting and gradient explosion.

Deep Liquid State Machine

Liquid Neural Networks, also known as Liquid State Machines (LSMs), were first introduced by Wolfgang Maass (MAASS; NATSCHLäGER; MARKRAM, 2002). The primary departure from traditional Artificial Neural Networks (ANNs) lies in their **dynamic and recurrent architecture**. While conventional ANNs consist of fixed layers of interconnected neurons, LNNs employ a collection of interconnected neurons that are **constantly changing**. This dynamic behavior enables LNNs to process continuous streams of data, **typically (but not limited)** in the form of spike trains, and it maps streams of inputs into streams of outputs. These outputs depend on the previous states created by the streaming data (MAASS,).

Also, according to (HASANI et al., 2020), the inspiration comes from the nervous system of the nematode *C. elegans*, which, despite having just **302 neurons**, exhibits complex behavior.

LSM is a model for **adaptive computing systems**. The network's structure **architecture is diverse**: Some may use one to several full/partial connected layers of classic, spike or any other type of neuron, other can use an ESN taking advantage of the reservoir computing.

Than, **what differentiates the liquid networks???** The catch here is the **forward propagation** that allows the model to learn **without** the need of retraining. In this model, the network's parameters change over time according to a set of nested differential equations **while it is already in use**, meaning that **this model does not necessarily require retraining to adapt**.

The word "liquid" can sometimes be taken literally, as in this work (FERNANDO; SOJAKKA, 2003), where a bucket of water has been used to represent the liquid part of the network, as seen in Figure 3.
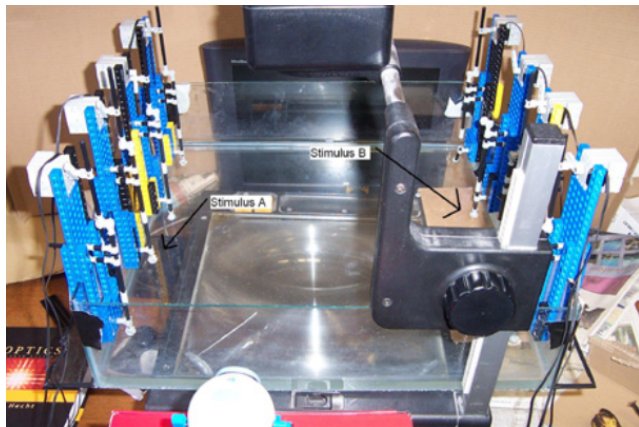

Figure: A literal liquid network. Source: (FERNANDO; SOJAKKA, 2003)

Characteristics

- ▶ **Adaptability**: Their dynamic nature enables them to respond dynamically to varying data distributions, making them well-suited for tasks involving non-stationary data.
- ▶ **Robustness**: LNNs have shown improved robustness against noise and input variations. The fluid-like behavior allows them to self-adjust and filter out irrelevant information, leading to enhanced generalization capabilities.
- ▶ **Exploration of Solution Space**: LNNs encourage solution space exploration by providing flexibility in the network's structure. The dynamic connectivity patterns enable the network to explore diverse pathways, potentially discovering novel solutions to complex problems.
- ▶ **Reduced Overfitting**: Due to their continuous learning capabilities, LNNs are less prone to overfitting, which often occurs in static networks, resulting in more accurate and generalizable models.

To describe a LNN, we can use the equation 1 (HASANI et al., 2022) where at a time step t, $x(t)^{D \times 1}$ defines the hidden state of a layer with D cells, and $I(t)^{M \times 1}$ is the input to the system with M features, $w_\tau^{D \times 1}$ is a time-constant parameter vector, $A^{D \times 1}$ is a bias vector, f is a neural network parametrized by θ and $\odot$ is the Hadamard product. The dependence of f(.) on x(t) denotes the possibility of having recurrent connections.

$$\frac{d\mathbf{x}}{dt} = -[w_\tau^{D \times 1} + f(\mathbf{x^{D \times 1}}, \mathbf{I^{M \times 1}}, \theta)] \odot \mathbf{x^{D \times 1}}(t) + A^{D \times 1} \odot f(\mathbf{x^{D \times 1}}, \mathbf{I^{M \times 1}}, \theta) \qquad (1)$$

To illustrate, a simplified neuron is show below at the listing 1:

```
1  class LiquidTimeConstantNeuron:
2      def __init__(self):
3          self.time_constant = 1.0
4
5      def adjust_time_constant(self, input_data):
6          # Adjust the time constant based on input data
7          self.time_constant = 1.0 + input_data
8
9      def update_state(self, state, time_step):
10         # Update the state based on the time constant and time step
11         return state - (state / self.time_constant) + np.sin(time_step)
12
13 # Simulate over time
14 duration = 10.0
15 time = np.arange(0, duration, 0.1)
16 states = [0.0]
17
18 for t in time[1:]:
19     input_data = np.sin(t)  # Simulated input data (can vary over time)
20     ltc_neuron.adjust_time_constant(input_data)
21     new_state = ltc_neuron.update_state(states[-1], t)
22     states.append(new_state)
```
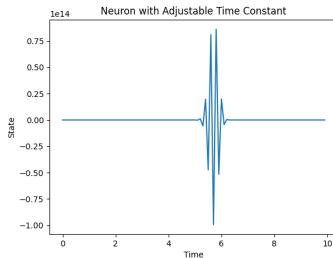
Listing: Liquid neuron

Figure: The liquid neuron of the listing 1 reacting to a continuous sinusoidal signal

Then finally a pyTorch implementation (TYAGI, 2023):

```python
import torch
import torch.nn as nn

class ESN(nn.Module):
    def __init__(self, input_size, reservoir_size, output_size):
        super(ESN, self).__init__()
        self.reservoir_size = reservoir_size
        self.W_in = nn.Linear(input_size, reservoir_size)
        self.W_res = nn.Linear(reservoir_size, reservoir_size)
        self.W_out = nn.Linear(reservoir_size, output_size)

    def forward(self, input):
        reservoir = torch.zeros((input.size(0), self.reservoir_size))
        for i in range(input.size(1)):
            input_t = input[:, i, :]
            reservoir = torch.tanh(self.W_in(input_t) + self.W_res(reservoir))
        output = self.W_out(reservoir)
        return output

# Example usage
input_size = 10
reservoir_size = 100
output_size = 1

model = ESN(input_size, reservoir_size, output_size)
```

Deep Spiking Neural Network

Spiking Neural Networks (SNNs) mimic the workings of the brain by utilizing action potentials, in contrast to continuous values transmitted between neurons.

The term "Spiking" originates from the behavior of biological neurons, which sporadically emit action potentials, creating voltage spikes that convey information (KASABOV, 2019). Figure 5 illustrates these spikes.

An SNN **is not** a one-to-one simulation of neurons. Instead, it approximates certain computational capabilities of specific biological properties. Some studies explore the nonlinearity of dendrites and other neuron features (JONES; KORDING, 2020), yielding remarkable results in the classification.
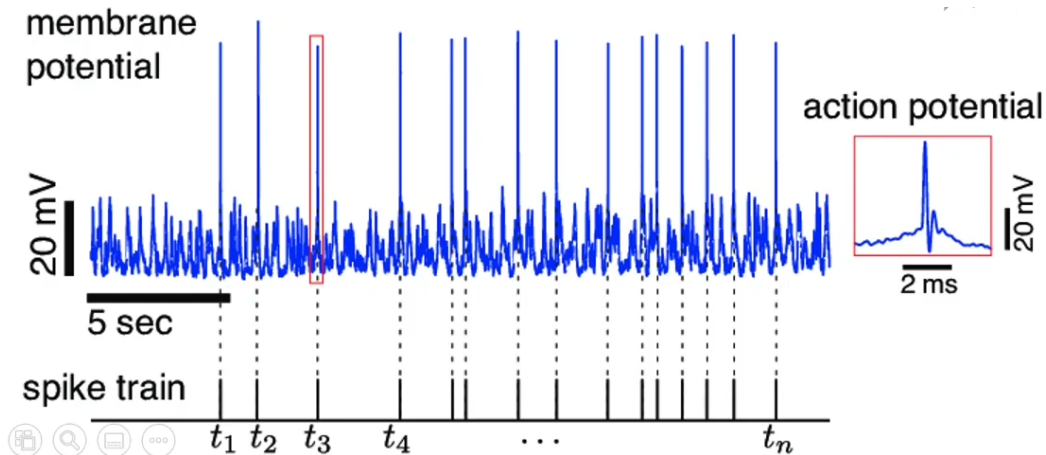
Figure: Spikes from a noisy signal. Source (GOODMAN et al., 2022)

General characteristics

SNNs possess several noteworthy characteristics that distinguish them from traditional machine learning techniques, including classical neural networks. These distinctions encompass (KASABOV, 2019):

- ▶ Proficiency in modeling temporal, spatial-temporal, or spectro-temporal data.
- ▶ Effectiveness in capturing processes involving various time scales.
- ▶ Seamless integration of multiple modalities, such as sound and vision, into a unified system.
- ▶ Aptitude for predictive modeling and event prediction.
- ▶ Swift and highly parallel information processing capabilities.
- ▶ Streamlined information processing.
- ▶ Scalability, accommodating structures ranging from a few tens to billions of spiking neurons.
- ▶ Minimal energy consumption when implemented on neuromorphic platforms.

Spiking Neural Networks (SNNs) are often considered power-efficient for several reasons:
Event-Driven Processing: SNNs are inherently event-driven. Instead of constantly updating neuron activations and synapse weights as in traditional artificial neural networks (ANNs), SNNs only transmit spikes (action potentials) when a neuron's membrane potential reaches a certain threshold. This event-driven approach reduces the amount of computation required and can lead to significant energy savings.

Sparse Activity: SNNs tend to exhibit sparse activity, meaning that only a small percentage of neurons are active at any given time. This sparsity reduces the number of computations that need to be performed, which is especially beneficial for hardware implementations where most of the energy consumption comes from active components.

Low Precision: SNNs can often work with lower precision than ANNs. While ANNs typically use high-precision floating-point numbers for neuron activations and synaptic weights, SNNs can use lower precision fixed-point or binary representations. Lower precision computations require less energy to perform.

Neuromorphic Hardware: SNNs can be efficiently implemented on specialized neuromorphic hardware, which is designed to mimic the energy-efficient behavior of biological neural systems. These hardware platforms are optimized for the event-driven nature of SNNs, further reducing power consumption.

Energy-Aware Learning Rules: SNNs can employ learning rules that take into account energy efficiency. For example, some learning rules prioritize strengthening or weakening synapses based on their contribution to network activity, which can lead to more energy-efficient learning.

Spike Encoding: SNNs can encode information in the timing and frequency of spikes, which can be a highly efficient way to represent and process data, particularly for event-based sensors like vision sensors or auditory sensors.

unesp

In order to emulate such behavior, let's begin with a simple model: The "Leaky Integrate and Fire neuron" (LIF). The LIF model describes the evolution of membrane potential which the potential decay over time.

$$\tau \cdot \frac{dV}{dt} = -V \qquad (2)$$

When a neuron receives a spike, the membrane potential V increases according to a synaptic weight w.

$$V = V + w \qquad (3)$$

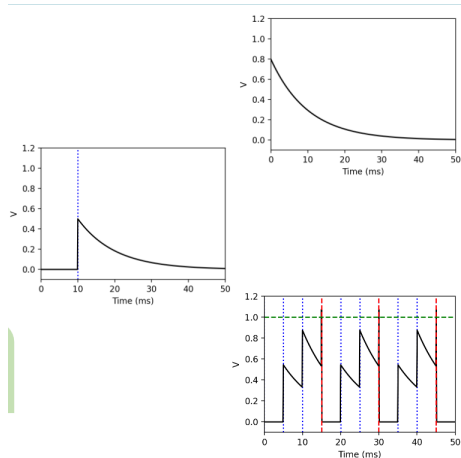Such behaviors are depicted in Figure 6.

Figure: Evolution of a Spike. Source (GOODMAN et al., 2022)

Theory: Training

As shown in Figure 6, when a neuron reaches a certain threshold, it resets ($V = 0$). Note that $V = 0$ is just one of the possible values of reset, after this the neuron may or may not enter in a refractory period. **These, an another values cited ahead, can be parameterized or learned**.

How do SNNs get trained? Well, this is still an open question. An SNN neuron has an activation-function behavior that is more relatable to a **step function**. Therefore, in principle, we can't use gradient descent-based solutions because this kind of function **is not** differentiable (KASABOV, 2019).

But there are some insights out there that may shed some light on this subject: While some *in vivo/in vitro* observations show that brains, in general, learn by strengthening/weakening and adding/removing synapses or even by creating new neurons or other cumbersome methods like RNA packets, there are some more acceptable ones like the ones in the slide (KASABOV, 2019):

Theory: Training

- ▶ Spike Timing-Dependent Plasticity (STDP): The idea is that if a pre-synaptic neuron fires **before** the post-synaptic one, there is a strengthening in connection, but if the post-synaptic neuron fires before, then there is a weakening.

- ▶ Surrogate Gradient Descent: The technique **approximates** the step function by using another mathematical function, which is differentiable (like a sigmoid), in order to train the network. These approximations are used only **in the backward pass**, while keeping the step function in the forward pass (KASABOV, 2019).

- ▶ Evolving Algorithms: Use the selection of the fittest throughout many generations of networks.

- ▶ Reservoir/Dynamic Computing: **Echo state networks** or **Liquid state machines** respectively. These will be discussed further in this presentation.

Before entering into demonstration we need to understand how the exposed theory can be implemented into our neuron, let's see a python code:

```python
class Neuron:
    """
    Leak integrate and fire neuron — LIF

    """
    def __init__(self, tau=10, threshold=1):

        # Initial membrane voltage
        self.voltage = 0

        # The smaller tau is the faster the voltage decays
        # When tau is large the neuron acts as an intergrator summing its inputs
        # and firing when a certain threshold is reached.
        # When tau is small the neuron acts as a coincidence detector, firing a
        # spike only when two or more input arrive simultaneosly.
        self.tau = tau

        # The threshold above which the neuron fires
        self.threshold = threshold

        # Time step for decaying (i still don't known what this really is)
        # Bigger the number faster the decay
        self.timeStep = 0.5
```

```
26            # The rate by which the membrane voltage decays each time step
27            self.alpha = np.exp(-self.timeStep/self.tau)
28
29        def set_tau(self, tau):
30            self.tau = tau
31            self.alpha = np.exp(-self.timeStep/self.tau)
32
33        def fire_spike(self):
34            if self.voltage > self.threshold:
35                self.voltage = 0
36                return 1
37            return 0
38
39        def add_synaptic_weight(self, weigth):
40            # Membrane voltage integration
41            self.voltage += weigth
42
43        def iterate(self):
44            # Membrane voltage leak
45            self.voltage = max(self.voltage * self.alpha, 0)
46            return self.fire_spike()
```

And now the simulation (code available in repository):

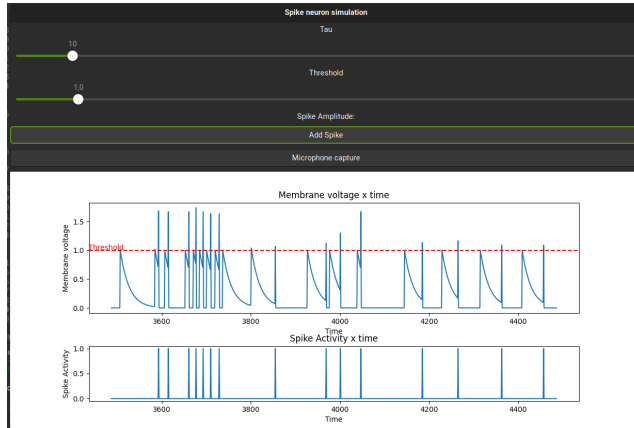unesp

Figure: A simulation a of spiking neuron

📄 BOCCATO, L. et al. An echo state network architecture based on volterra filtering and pca with application to the channel equalization problem. In: *The 2011 International Joint Conference on Neural Networks*. [S.l.: s.n.], 2011. p. 580–587.

📄 FERNANDO, C.; SOJAKKA, S. Pattern recognition in a bucket. In: BANZHAF, W. et al. (Ed.). *Advances in Artificial Life*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003. p. 588–597. ISBN 978-3-540-39432-7.

📄 GOODMAN, D. et al. *Spiking Neural Network Models in Neuroscience - Cosyne Tutorial 2022*. Zenodo, 2022. Disponível em: ⟨https://doi.org/10.5281/zenodo.7044500⟩.

📄 HASANI, R. et al. *Liquid Time-constant Networks*. 2020.

📄 HASANI, R. et al. Closed-form continuous-time neural networks. *Nature Machine Intelligence*, v. 4, n. 11, p. 992–1003, November 2022. ISSN 2522-5839. Disponível em: ⟨https://doi.org/10.1038/s42256-022-00556-7⟩.

📄 JAEGER, H. The echo state approach to analysing and training recurrent neural networks-with an erratum note. *Bonn, Germany: German National Research Center for Information Technology GMD Technical Report*, v. 148, January 2001.

📄 JONES, I. S.; KORDING, K. P. *Can Single Neurons Solve MNIST? The Computational Power of Biological Dendritic Trees*. 2020. Disponível em: ⟨https://arxiv.org/abs/2009.01269⟩.

📄 KASABOV, N. K. *Time-space, spiking neural networks and brain-inspired artificial intelligence*. [S.l.]: Springer, 2019.

📄 LUKOSEVICIUS, M. A practical guide to applying echo state networks. In: MONTAVON, G.; ORR, G. B.; MüLLER, K.-R. (Ed.). *Neural Networks: Tricks of the Trade, 2nd Edition*. Springer, 2012, (LNCS, v. 7700). p. 659–686. ISBN 978-3-642-35288-1. Simple source code samples available. Disponível em: ⟨https://mantas.info/get-publication/?f=Practical_ESN.pdfhttp://link.springer.com/chapter/10.1007%2F978-3-642-35289-8_36https://www.ai.rug.nl/minds/uploads/PracticalESN.pdf/code/simple_esn/⟩.

📄 MAASS, W. Liquid state machines: Motivation, theory, and applications.
In: _____. *Computability in Context*. [s.n.]. p. 275–296. Disponível em:
⟨https://www.worldscientific.com/doi/abs/10.1142/9781848162778_0008⟩.

📄 MAASS, W.; NATSCHLäGER, T.; MARKRAM, H. Real-time computing without
stable states: A new framework for neural computation based on perturbations.
Neural Computation, v. 14, n. 11, p. 2531–2560, 2002.

📄 SANKHALA, V. Reservoir computing - echo state networks and liquid state
machines. In: . [S.l.: s.n.], 2023. v. 1.

📄 TYAGI, B. *Liquid Neural Networks: Revolutionizing AI with Dynamic Informa-
tion Flow*. 2023. Medium article. Disponível em: ⟨https://tyagi-bhaumik.medium.com/
liquid-neural-networks-revolutionizing-ai-with-dynamic-information-flow-30e27f1cc912⟩.

unesp