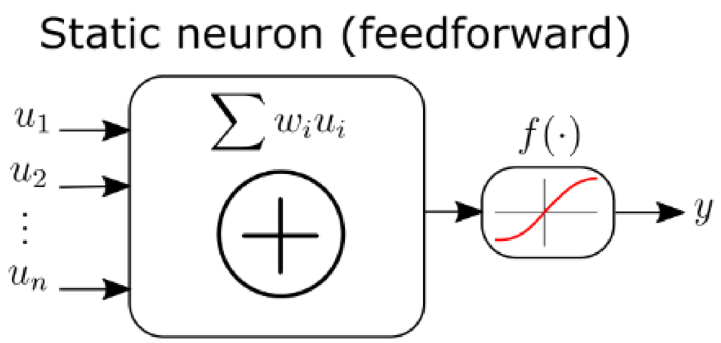
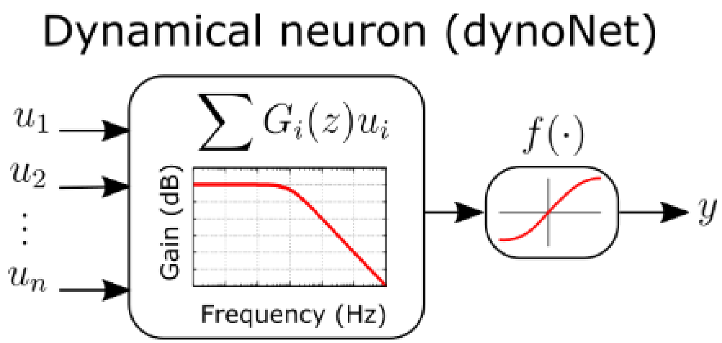
dynoNet: linear dynamical blocks in deep learning

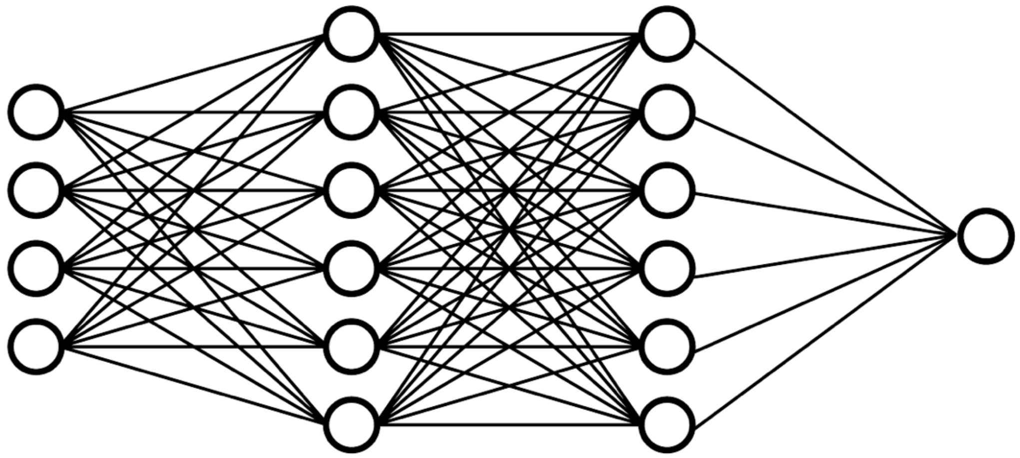
Dario Piga, Marco Forgione, Dalle Molle Institute for Artificial Intelligence, Lugano, Switzerland

dynoNet: main idea

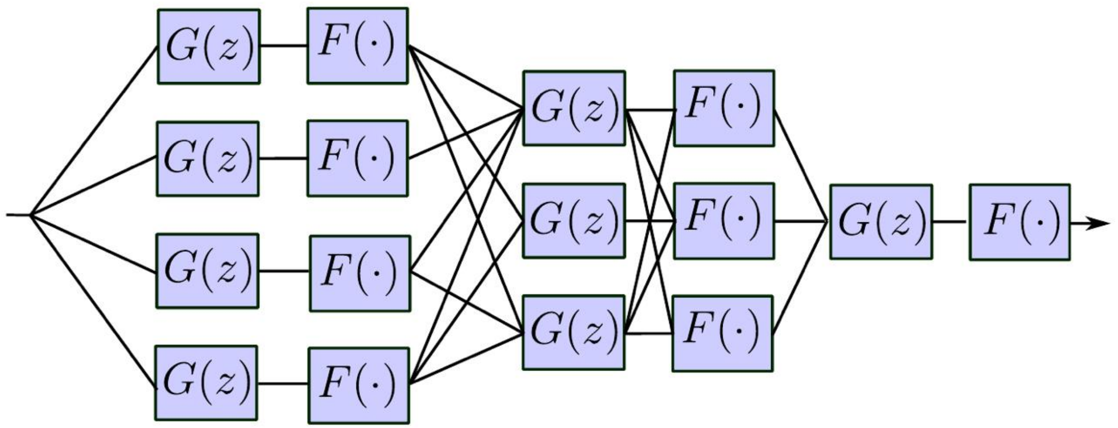




feedforward NN



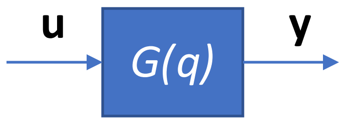
dynoNet



Forgione, M., & Piga, D. (2021). dynoNet: A neural network architecture for learning dynamical systems. International Journal of Adaptive Control and Signal Processing, 35(4), 612-626.

dynoNet: LTI operator

LTI linear operator

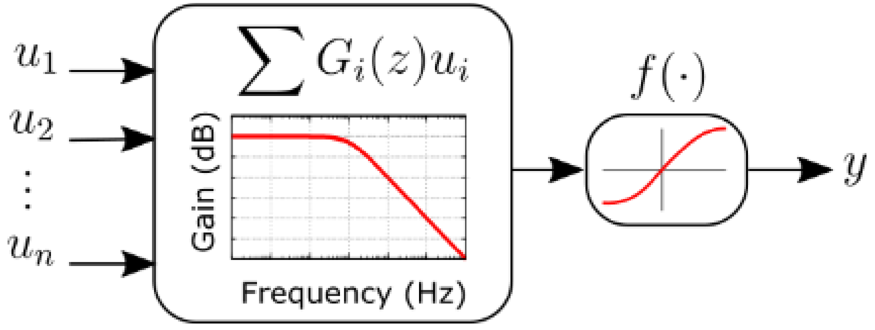


$$G(q^{-1}, \theta)$$

$$q^{-1}\mathbf{y}_t = \mathbf{y}_{t-1}$$

$$\mathbf{y}_t = \frac{B(q^{-1}, b)}{A(q^{-1}, a)}\mathbf{u}_t$$

Dynamical neuron (dynoNet)



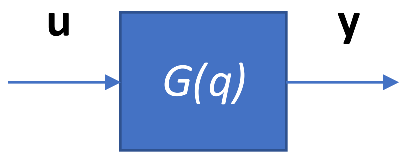
$$\underbrace{\left(1 - a_1 q^{-1} - \ldots - a_{n_a} q^{-n_a}\right)}_{A(q^{-1}, a)}\mathbf{y}_t = \underbrace{\left(b_0 + b_1 q^{-1} - \ldots - b_{n_b} q^{-n_b}\right)}_{B(q^{-1}, b)}\mathbf{u}_t$$

$$\mathbf{y}_t = a_1 \mathbf{y}_{t-1} + \cdots + a_{n_a}\mathbf{y}_{t-n_a} + b_0 \mathbf{u}_t + b_1 \mathbf{u}_{t-1} + \cdots + b_{n_b}\mathbf{u}_{t-n_b}$$

recurrence equation

Integration in a DL framework

What do we need to integrate the LTI dynamic block in a neural network architecture?

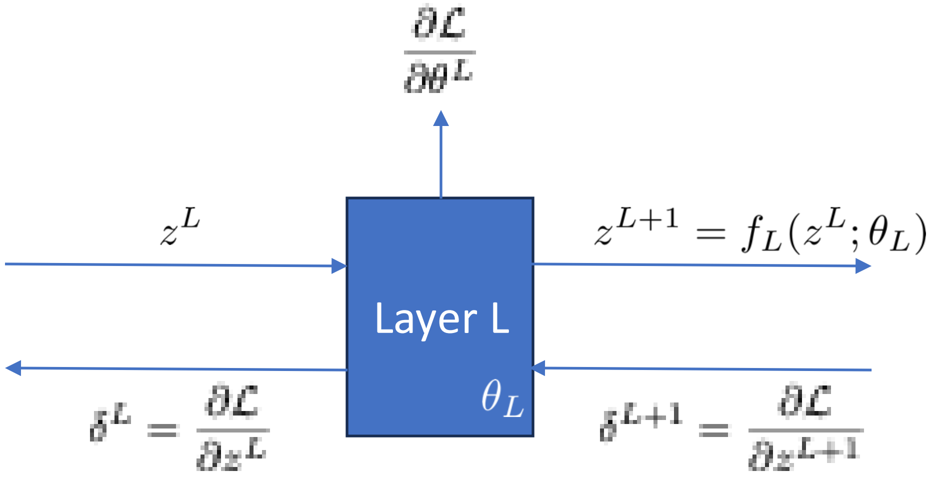


$$\mathbf{y}_t = G(q^{-1}; a, b)\mathbf{u}_t$$

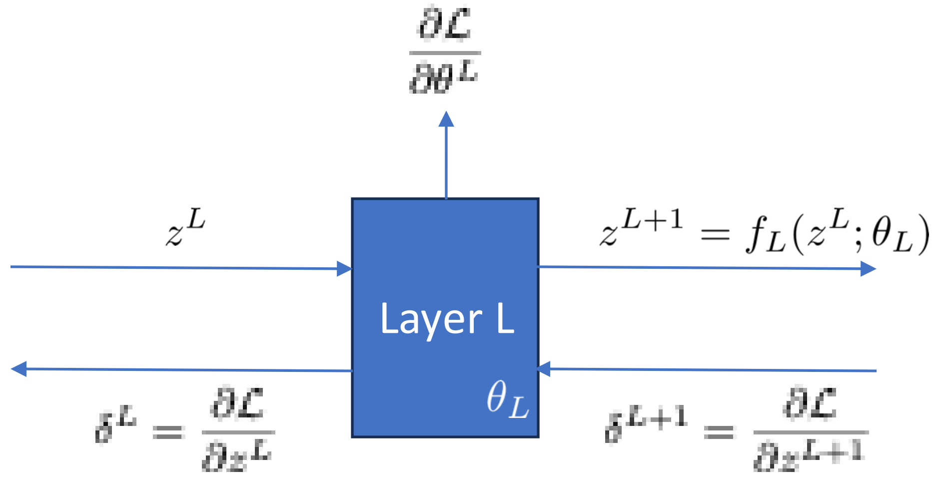
We need derivatives!

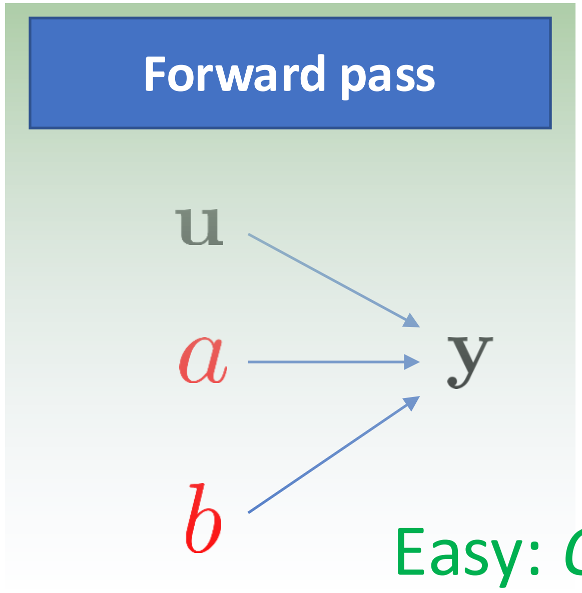
$$\frac{\partial \mathcal{L}}{\partial a} \quad \frac{\partial \mathcal{L}}{\partial b}$$

More in general, we need to transform the *G* operator into a differentiable layer!

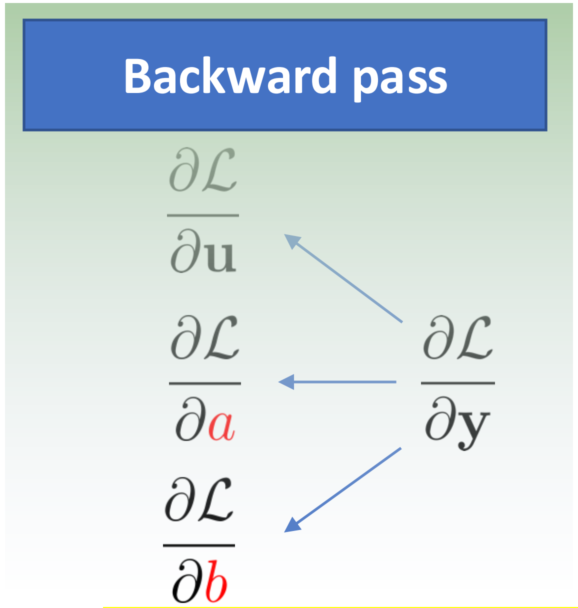


Transforming G into a differentiable layer





Easy: *O(T)*



$$\mathbf{y}_t = a_1 \mathbf{y}_{t-1} + \cdots + a_{n_a} \mathbf{y}_{t-n_a} + b_0 \mathbf{u}_t + b_1 \mathbf{u}_{t-1} + \cdots + b_{n_b} \mathbf{u}_{t-n_b}$$

Differentiating dynamical blocks

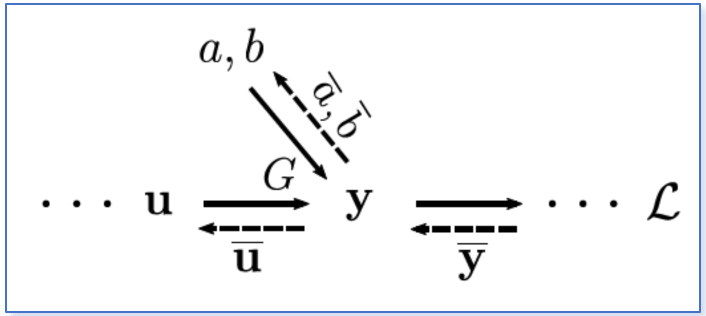
Backward pass

$$\frac{\partial \mathcal{L}}{\partial a} = \sum_{t=0}^{T} \frac{\partial \mathcal{L}}{\partial \mathbf{y}_t} \frac{\partial \mathbf{y}_t}{\partial a}$$

$$\frac{\partial \mathcal{L}}{\partial b} = \sum_{t=0}^{T} \frac{\partial \mathcal{L}}{\partial \mathbf{y}_t} \frac{\partial \mathbf{y}_t}{\partial b}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{u}_\tau} = \sum_{t=0}^{T} \frac{\partial \mathcal{L}}{\partial \mathbf{y}_t} \frac{\partial \mathbf{y}_t}{\partial \mathbf{u}_\tau}$$

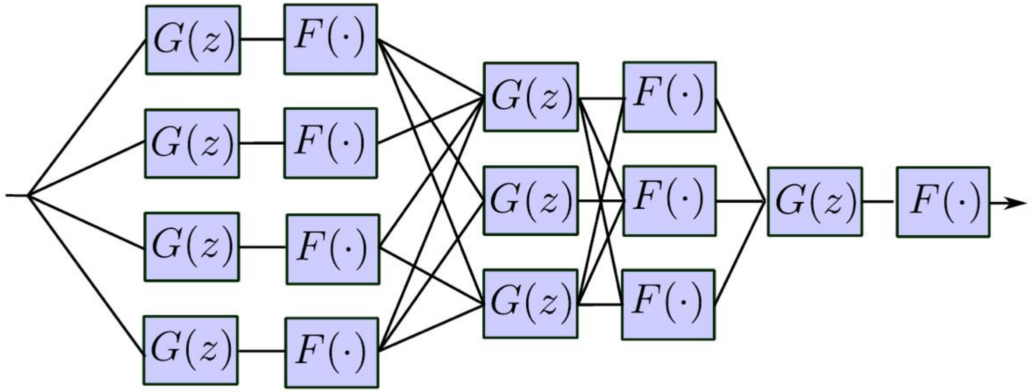
here's what we really need



$$\frac{\partial \mathcal{L}}{\partial \mathbf{u}} \leftarrow \frac{\partial \mathcal{L}}{\partial \mathbf{y}}$$

$$\frac{\partial \mathcal{L}}{\partial a} \leftarrow \frac{\partial \mathcal{L}}{\partial \mathbf{y}}$$

$$\frac{\partial \mathcal{L}}{\partial b} \leftarrow \frac{\partial \mathcal{L}}{\partial \mathbf{y}}$$



Backward pass

$$\boxed{\frac{\partial \mathbf{y}_t}{\partial a_1} = ???}$$

$$\mathbf{y}_t = a_1 \mathbf{y}_{t-1} + \cdots + a_{n_a} \mathbf{y}_{t-n_a} + b_0 \mathbf{u}_t + b_1 \mathbf{u}_{t-1} + \cdots + b_{n_b} \mathbf{u}_{t-n_b}$$

$$\frac{\partial \mathbf{y}_t}{\partial a_1} = \mathbf{y}_{t-1} + a_1 \frac{\partial \mathbf{y}_{t-1}}{\partial a_1} + \cdots + a_{n_a} \frac{\partial \mathbf{y}_{t-n_a}}{\partial a_1}$$

recurrence equation

Complexity: O(T)

Well-known derivation, see e.g. the classic book:

Ljung, L. System Identification: Theory for the User. Prentice-hall, Inc. 2nd Edition

Backward pass

$$\boxed{\frac{\partial \mathbf{y}_t}{\partial \mathbf{u}_\tau} = ???}$$

$$\mathbf{y}_t = \frac{B(q^{-1}, b)}{A(q^{-1}, a)}\mathbf{u}_t = G(q^{-1})\mathbf{u}_t$$

$$\mathbf{y}_t = a_1\mathbf{y}_{t-1} + \cdots + a_{n_a}\mathbf{y}_{t-n_a} + b_0\mathbf{u}_t + b_1\mathbf{u}_{t-1} + \cdots + b_{n_b}\mathbf{u}_{t-n_b}$$

$$\mathbf{y}_t = \sum_{\tau=0}^{t} \mathbf{g}_{t-\tau}\mathbf{u}_\tau$$

Impulse response

$$\frac{\partial \mathbf{y}_t}{\partial \mathbf{u}_\tau} = \begin{cases} \mathbf{g}_{t-\tau} & \text{if } \; t \geq \tau \\ 0 & \text{if } \; t < \tau \end{cases}$$

$$\mathbf{y}_0 = b_0\mathbf{u}_0$$

$$\mathbf{y}_1 = a_1\mathbf{y}_0 + b_0\mathbf{u}_1 + b_1\mathbf{u}_0 = a_1 b_0\mathbf{u}_0 + b_0\mathbf{u}_1 + b_1\mathbf{u}_0 = (a_1 b_0 + b_1)\mathbf{u}_0 + b_0\mathbf{u}_1$$

$$\mathbf{y}_2 = a_1\mathbf{y}_1 + a_2\mathbf{y}_0 + b_0\mathbf{u}_2 + b_1\mathbf{u}_1 + b_2\mathbf{u}_0 = \left(a_1(a_1 b_0 + b_1) + a_2 b_0 + b_2\right)\mathbf{u}_0 + (a_1 b_0 + b_1)\mathbf{u}_1 + b_0\mathbf{u}_2$$

$$\vdots$$

Backward pass: computational complexity

$$\frac{\partial \mathbf{y}_t}{\partial \mathbf{u}_\tau} = \begin{cases} \mathbf{g}_{t-\tau} & \text{if} \quad t \geq \tau \\ 0 & \text{if} \quad t < \tau \end{cases}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{u}_\tau} = \sum_{t=0}^{T} \frac{\partial \mathcal{L}}{\partial \mathbf{y}_t} \frac{\partial \mathbf{y}_t}{\partial \mathbf{u}_\tau}$$

Complexity: O(T), τ=0,…,T ← *O(T²)*

$$\frac{\partial \mathcal{L}}{\partial \mathbf{u}_\tau} = \sum_{t=\tau}^{T} \frac{\partial \mathcal{L}}{\partial \mathbf{y}_t} \mathbf{g}_{t-\tau}$$

Change of notation →

$$\bar{\mathbf{u}}_t = \sum_{\tau=t}^{T} \bar{\mathbf{y}}_\tau \mathbf{g}_{\tau-t}$$

The latter looks a lot like a convolution product involving $\boldsymbol{g}$ and $\bar{\boldsymbol{y}}$, which would be easy to compute…
It is equivalent to filtering through G!

Backward pass: computational complexity

Can we also compute $\bar{\mathbf{u}}_t = \sum_{\tau=t}^{T} \bar{\mathbf{y}}_\tau \mathbf{g}_{\tau-t}$ recursively?

$$\bar{\mathbf{u}}_t = \sum_{\tau=t}^{T} \bar{\mathbf{y}}_\tau \mathbf{g}_{\tau-t} \longleftrightarrow \mathbf{flip}(\bar{\mathbf{u}})_t = \sum_{h=0}^{t} \mathbf{flip}(\bar{\mathbf{y}})_h \mathbf{g}_{t-h}$$

$$\updownarrow$$

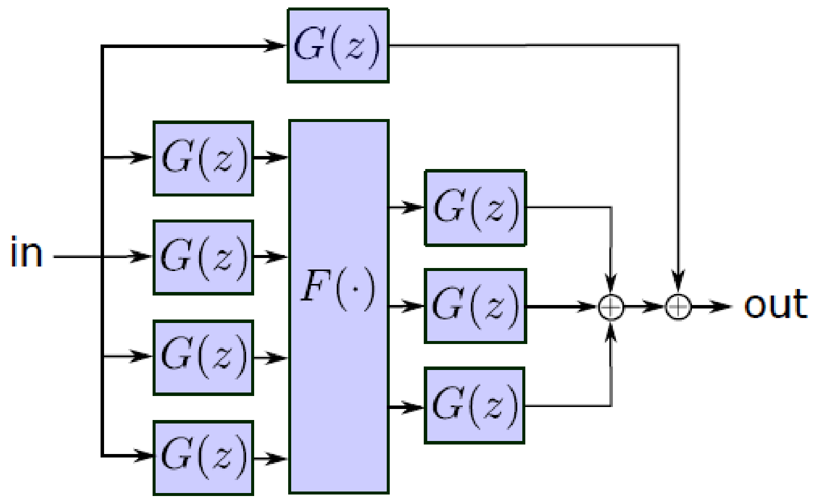
$$\mathbf{flip}(\bar{\mathbf{u}})_t = \frac{B(q^{-1}, b)}{A(q^{-1}, a)} \mathbf{flip}(\bar{\mathbf{y}})_t = G(q^{-1}) \mathbf{flip}(\bar{\mathbf{y}})_t$$

Complexity: O(T)

PyTorch implementation

PyTorch implementation of the *G*-block in the repository: *https://github.com/forgi86/dynonet*

dynoNet architecture



Python code

```
class CustomDynonet(torch.nn.Module):
    def __init__(self):
        super().__init__()
        self.G1 = MimoLinearDynamicalOperator(in_channels=1, out_channels=4, n_a=2, n_b=2, n_k=1)
        self.F = MimoStaticNonLinearity(in_channels=4, out_channels=3)
        self.G2 = MimoLinearDynamicalOperator(in_channels=3, out_channels=1, n_a=2, n_b=3)
        self.Glin = MimoLinearDynamicalOperator(in_channels=1, out_channels=1, n_a=2, n_b=2, n_k=1)

    def forward(self, u):
        x = self.G1(u)
        x = self.F(x)
        x = self.G2(x)
        y = x + self.Glin(u)
        return u

model = CustomDynonet()
batch_u = torch.randn(32, 1000, 1)
batch_y = model(batch_u)
batch_y.shape

torch.Size([32, 1000, 1])
```

Any gradient-based optimization algorithm can be used to train the network, with gradients readily obtained through back-propagation

dynoNet in TorchAudio

TORCHAUDIO.FUNCTIONAL.LFILTER

TorchAudio

```
torchaudio.functional.lfilter(
    waveform: Tensor,
    a_coeffs: Tensor,
    b_coeffs: Tensor,
    clamp: bool = True,
    batching: bool = True
) → Tensor  [SOURCE]
```

Perform an IIR filter by evaluating difference equation, using differentiable implementation developed independently by *Yu et al.* [Yu and Fazekas, 2023] and *Forgione et al.* [Forgione and Piga, 2021].

Devices CPU, CUDA | Properties Autograd, TorchScript

Other architectures with linear layers

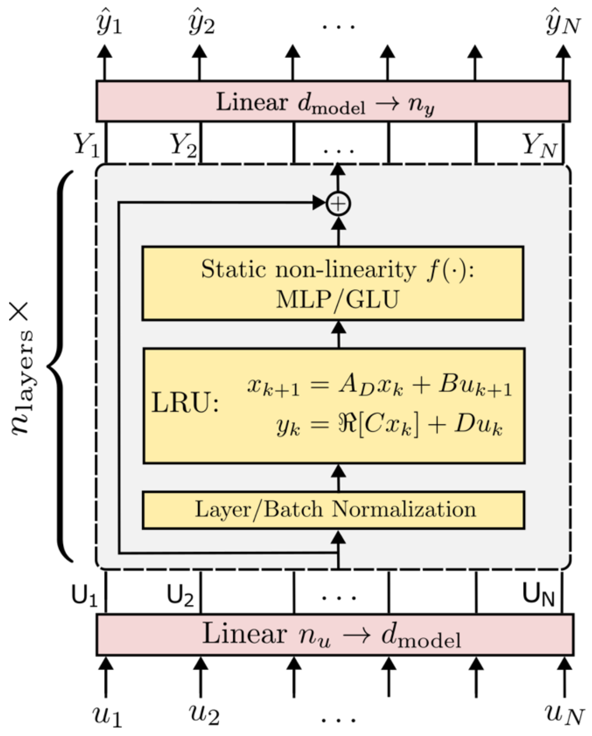
Several new models with linear dynamical blocks: S4, S5, LRU, Mamba, …

- Flexible and expressive
- Fast to train and simulate
- Stability almost for free
- Potential for analysis and explainability

Keyword: structured state-space sequence models (S4).

Variations in the linear dynamical block:

- Continuous/discrete-time
- State-space/transfer function
- Time-domain/frequency-domain
- Time-invariant/time-varying



There's room to apply system theory!

M. Forgione, M. Mejari, and D. Piga [Model order reduction of deep structured state-space models: A system-theoretic approach](), CDC 2024

dynoNet: conclusions

Key takeaways

- Differentiable dynamic neuron
- Extension of block-oriented models with arbitrary interconnections
- Training through back-propagation at a cost *O(T)*. No specialized algorithm required

Current work

- System analysis and model reduction through linear tools

M. Forgione, D. Piga, *dynoNet: A neural network architecture for learning dynamical systems*, IJACSP, 2021

https://github.com/forgi86/dynonet