Neural Differential Equations for Learning to Program Neural Nets Through Continuous Learning Rules

Istituto Dalle Molle di studi sull'intelligenza artificiale
USI/SUPSI
IDSIA
جامعة الملك عبدالله للعلوم والتقنية
King Abdullah University of Science and Technology

Kazuki Irie[1] Francesco Faccio[1] Jürgen Schmidhuber[1, 2]

[1] The Swiss AI Lab, IDSIA, USI & SUPSI Switzerland
[2] AI Initiative, KAUST Saudi Arabia

Neural ODEs for Sequence Processing

- Original **Neural ODEs**: continuous-depth version of (feedforward) residual nets
- There are extensions to **process sequences**, e.g.,

Neural Controlled Differential Equations *Kidger et al. 2020*

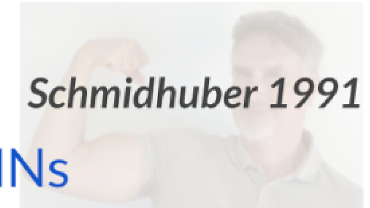
Hidden state $\boldsymbol{h}(t) \in \mathbb{R}^d$ *Differentiable control signal* $\boldsymbol{x}(t) \in \mathbb{R}^{d_{\text{in}}}$

$$\begin{aligned}\boldsymbol{h}(t) &= \boldsymbol{h}(t_0) + \int_{s=t_0}^{t} \boldsymbol{F}_\theta(\boldsymbol{h}(s))d\boldsymbol{x}(s) \\ &= \boldsymbol{h}(t_0) + \int_{s=t_0}^{t} \boldsymbol{F}_\theta(\boldsymbol{h}(s))\boldsymbol{x}'(s)ds\end{aligned}$$

"vanilla" RNN in the continuous-time domain

- *Good empirical performance (outperform other ODE based sequence processors), but*
- *Scalability limitation:* $\mathbb{R}^d \rightarrow \mathbb{R}^{d \times d_{\text{in}}}$

Fast Weight Programmers (FWPs)



Schmidhuber 1991

- NN that learns to program other NNs by rapidly generating weight changes
- Outer product version: linear Transformer
- General purpose sequence processor *Katharopoulos et al. 2020 etc.*

Example: *DeltaNet* *Schlag et al. 2021*

At each step n

- *Input* $\boldsymbol{x}_n \in \mathbb{R}^{d_{\text{in}}}$ $\quad \beta_n, \boldsymbol{q}_n, \boldsymbol{k}_n, \boldsymbol{v}_n = \boldsymbol{W}_{\text{slow}}\boldsymbol{x}_n$
- *Fast Weight Matrix* $\quad \boldsymbol{W}_n = \boldsymbol{W}_{n-1} + \sigma(\beta_n)(\boldsymbol{v}_n - \boldsymbol{W}_{n-1}\phi(\boldsymbol{k}_n)) \otimes \phi(\boldsymbol{k}_n)$
- *Output* $\boldsymbol{y}_n \in \mathbb{R}^{d_{\text{out}}}$ $\quad \boldsymbol{y}_n = \boldsymbol{W}_n\phi(\boldsymbol{q}_n)$

To hear more about FWPs: Visit our poster on Friday "Memory in Artificial and Real Intelligence" WS

You only have 2 min?

We introduce continuous-time counterparts of Fast Weight Programmers (FWP)/ linear Transformers by combining FWPs with Neural ODEs

- → *We obtain a new type of **Neural ODE/CDE based sequence processors, that***
- → *Conceptually **scale better** than existing Neural CDE models*
- → *Empirically **outperform** existing Neural CDE based models*

We propose multiple model variations, depending on

- → *Smoothness of input control signals, and*
- → *Different learning rule parameterisations (Hebb, Oja, Delta)*

General Idea:

Discrete-time Weight Update

$$\beta_n, \boldsymbol{q}_n, \boldsymbol{k}_n, \boldsymbol{v}_n = \boldsymbol{W}_{\text{slow}}\boldsymbol{x}_n$$

$$\boldsymbol{W}_n = \boldsymbol{W}_{n-1} + \sigma(\beta_n)(\boldsymbol{v}_n - \boldsymbol{W}_{n-1}\phi(\boldsymbol{k}_n)) \otimes \phi(\boldsymbol{k}_n)$$

Continuous-time Counterpart

$$\boldsymbol{W}(t) = \boldsymbol{W}(t_0) + \int_{s=t_0}^{t} \boldsymbol{F}_\theta(\boldsymbol{W}(s), \boldsymbol{x}(s))ds$$

Code: github.com/IDSIA/neuraldiffeq-fwp

Continuous-Time FWPs

NCDE FWPs

Differentiable Input Control Signal $\boldsymbol{x}(t) \in \mathbb{R}^{d_{\text{in}}}$

Forward pass: ODE solver
Backward pass: Continuous adjoint method

State (Fast Weight Matrix)

$$\underbrace{\boldsymbol{W}(t)}_{\in \mathbb{R}^{d_{\text{out}} \times d_{\text{key}}}} = \boldsymbol{W}(t_0) + \int_{s=t_0}^{t} \boldsymbol{F}_\theta(\boldsymbol{W}(s), \boldsymbol{x}(s), \boldsymbol{x}'(s))\boldsymbol{x}'(s)ds$$

Output

$$\underbrace{\boldsymbol{y}(T)}_{\in \mathbb{R}^{d_{\text{out}}}} = \begin{cases} \boldsymbol{W}(T)^\top \boldsymbol{W}_q\boldsymbol{x}(T) & \text{Hebb and Oja} \\ \boldsymbol{W}(T)\boldsymbol{W}_q\boldsymbol{x}'(T) & \text{Delta} \end{cases}$$

Query

Key *Value* *Outer Product*

$$= \underbrace{\sigma(\beta(s))}_{\text{Learning rate}} \begin{cases} \boldsymbol{W}_k\boldsymbol{x}(s) \otimes \boldsymbol{W}_v\boldsymbol{x}'(s) & \text{Hebb} \\ (\boldsymbol{W}_k\boldsymbol{x}(s) - \boldsymbol{W}(s)^\top \boldsymbol{W}_v\boldsymbol{x}'(s)) \otimes \boldsymbol{W}_v\boldsymbol{x}'(s) & \text{Oja} \\ (\boldsymbol{W}_v\boldsymbol{x}(s) - \boldsymbol{W}(s)\boldsymbol{W}_k\boldsymbol{x}'(s)) \otimes \boldsymbol{W}_k\boldsymbol{x}'(s) & \text{Delta} \end{cases}$$

Key properties

- ***Scalable**: outer product-based vector field*
- ***Expressive**: Sum all rank-1 updates in the continuous-time domain, then use the resulting weight matrix to compute the output (i.e., sum before matrix multiplication)*
  vs. basic NCDEs with a rank-1 vector field: scalable but not expressive** **(rank-1 mat. used in isolation)
- *Good empirical performance (Transformer!)*

Direct NODE FWPs

(piece-wise) Continuous/Bounded Input Control Signal $\boldsymbol{x}(t) \in \mathbb{R}^{d_{\text{in}}}$

- Similar idea to (left) but w/o derivative of control signal
- Theoretically NCDEs are more powerful (Kidger et al. 2020), but
- With our parameterisations, performance gap is small

$$\boldsymbol{W}(t) = \boldsymbol{W}(t_0) + \int_{s=t_0}^{t} \boldsymbol{F}_\theta(\boldsymbol{W}(s), \boldsymbol{x}(s))ds$$

$$\boldsymbol{q}(T) = \boldsymbol{W}_q\boldsymbol{x}(T) \quad \textit{Query}$$

Output $\quad \boldsymbol{y}(T) = \boldsymbol{W}(T)\boldsymbol{q}(T)$

$$\boldsymbol{F}_\theta(\boldsymbol{W}(s), \boldsymbol{x}(s)) = \sigma(\beta(s)) \begin{cases} \boldsymbol{k}(s) \otimes \boldsymbol{v}(s) & \text{Hebb-style} \\ \boldsymbol{v}(s) \otimes (\boldsymbol{k}(s) - \boldsymbol{W}(s)^\top \boldsymbol{v}(s)) & \text{Oja-style} \\ (\boldsymbol{v}(s) - \boldsymbol{W}(s)\boldsymbol{k}(s)) \otimes \boldsymbol{k}(s) & \text{Delta-style} \end{cases}$$

Learning rate *Key* *Value*

$$[\beta(s), \boldsymbol{k}(s), \boldsymbol{v}(s)] = \boldsymbol{W}_{\text{slow}}\boldsymbol{x}(s) \qquad \boldsymbol{W}_{\text{slow}} \in \mathbb{R}^{(1+d_{\text{key}}+d_{\text{out}}) \times d_{\text{in}}}$$

Time Series Classification Tasks

Speech Commands & PhysioNet Sepsis

Type	Model	Speech Commands	PhysioNet Sepsis	
			OI	no-OI
Direct NODE	GRU-ODE	47.9 (2.9)	85.2 (1.0)	77.1 (2.4)
	Hebb	82.8 (1.1)	**90.4 (0.4)**	82.9 (0.7)
	Oja	**85.4 (0.9)**	88.9 (1.4)	82.9 (0.5)
	Delta	81.5 (3.8)	89.8 (1.0)	**84.5 (2.9)**
CDE	NCDE	89.8 (2.5)	88.0 (0.6)	77.6 (0.9)
	Hebb	89.5 (0.3)	89.9 (0.6)	**85.7 (0.3)**
	Oja	90.0 (0.7)	**91.2 (0.4)**	85.1 (2.5)
	Delta	**90.2 (0.2)**	90.9 (0.2)	84.5 (0.7)

- *FWPs **outperform** the existing ODE/CDE baselines*
- *No clear winner among different learning rules*

EigenWorms

- *long sequences (> 4000 timesteps)*
- *The delta rule outperform others*

Model	Sig-Depth	Step	Test Acc. [%]
NRDE	2	4	83.8 (3.0)
Hebb	2	4	45.6 (5.9)
Oja			46.7 (7.5)
Delta			**87.7 (1.9)**
NCDE	1	4	66.7 (11.8)
Hebb	1	4	41.0 (6.5)
Oja			49.7 (9.9)
Delta			**91.8** (3.4)

Overall:

- *FWPs outperform the best existing Neural ODE/CDE based models, but*
- *There exist **discrete-time models** that perform equally well or better, e.g., LEM for Eigenworms GRU-D for PhysioNet Sepsis no-OI*

Model-based Reinforcement Learning

What if we need to directly work with irregularly sampled discrete inputs? FWP analogs to Latent ODE-RNNs

$$\begin{aligned}\boldsymbol{u}_n &= \text{ODESolve}(\boldsymbol{f}_{\theta_1}, \boldsymbol{h}_{n-1}, t_{n-1}, t_n) \\ \boldsymbol{h}_n, \boldsymbol{W}_n &= \text{FWP}([\boldsymbol{x}_n, \boldsymbol{u}_n], \boldsymbol{W}_{n-1}; \theta_2)\end{aligned}$$

- ***Setting:** MuJoCo **with irregularly timed observations (semi-MDP; repeated actions)***
- ***FWPs perform as well or better than Latent ODE-RNNs***

(a) Swimmer (b) Hopper