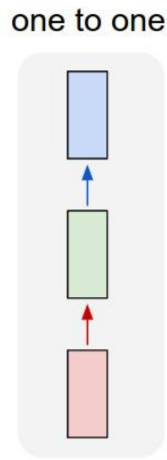
Lecture 13. Recurrent Neural Networks and Continuous-depth Neural Networks

Bao Wang
Department of Mathematics
Scientific Computing and Imaging Institute
University of Utah
Math 5750/6880, Fall 2023



$$f(\boldsymbol{x}, \boldsymbol{W}) = \boldsymbol{W}_n \sigma\Big(\boldsymbol{W}_{n-1}\sigma\Big(\cdots \boldsymbol{W}_2 \sigma\big(\boldsymbol{W}_1 \boldsymbol{x}\big)\cdots\Big)\Big)$$

Image classification.

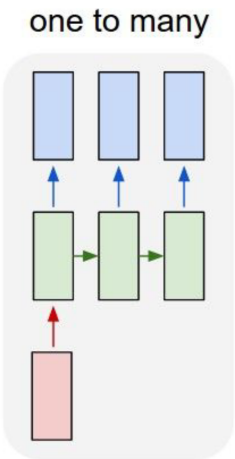
Recurrent Neural Networks

Learning sequences with varying sequence length!

Why existing ConvNets are insufficient?

- How to build a machine learning model for variable sequence length inputs and outputs?

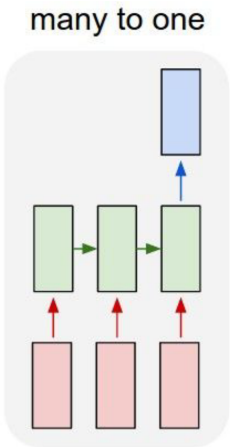
- One to many



- Image captioning (image $\rightarrow$ sequence of words)

Recurrent neural networks – II (Process sequences)

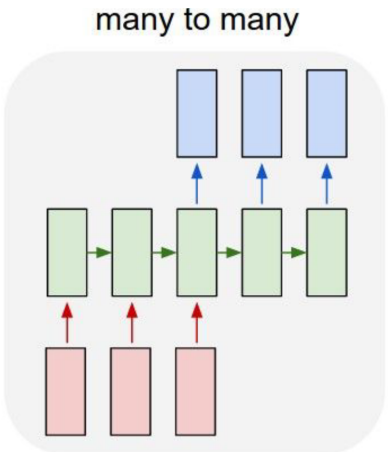
- Many to one



Review (X)	Rating (Y)
"This movie is fantastic! I really like it because it is so good!"	★★★★☆
"Not to my taste, will skip and watch another movie"	★★☆☆☆
"This movie really sucks! Can I get my money back please?"	★☆☆☆☆

- Sentiment classification/sequence of video frames $\rightarrow$ action class

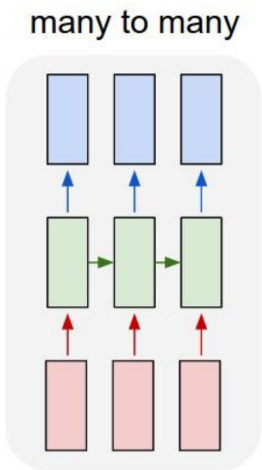
- Many to many



- Machine translation, dialogue
- Sequence of video frames → caption

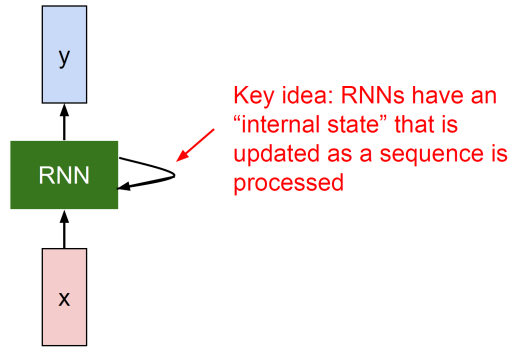
Recurrent neural networks – IV (Process sequences)

- Many to many



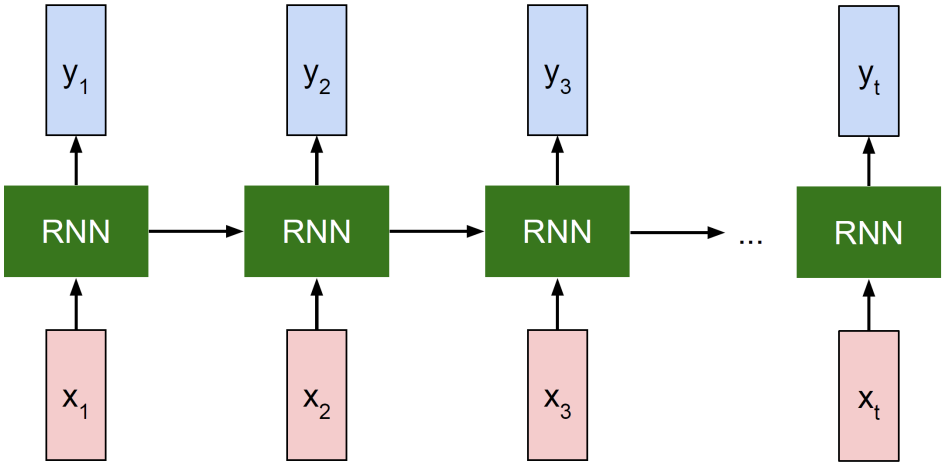
- Video captioning (sequence of video frames → caption)

- Let us start with a task that takes a variable input and produces an output at every step. **Many-to-many.**



- RNN internal state: $\boldsymbol{h}$. Encoding the sequence information.

Unrolled RNN



- RNN internal state: $\boldsymbol{h}_t$ is a function of $\boldsymbol{h}_{t-1}$ and $\boldsymbol{x}_t$.
- Two ingredients: Hidden state updates and output generation.

RNN hidden state update

We can process a sequence of vectors $\boldsymbol{X} = [\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots]$ by applying a recurrence formula at every time step:

$$\boldsymbol{h}_t = f_{\boldsymbol{W}}\left(\boldsymbol{h}_{t-1}, \boldsymbol{x}_t\right),$$

where

- $\boldsymbol{h}_t$ is the new state
- $f_{\boldsymbol{W}}(\cdot, \cdot)$ is a function with parameters $\boldsymbol{W}$
- $\boldsymbol{h}_{t-1}$ is the old state
- $\boldsymbol{x}_t$ is the input vector at some time step

RNN output generation

We output $\boldsymbol{y}_t$ at every time-step t by using the following formula

$$\boldsymbol{y}_t = f_{\boldsymbol{W}_o}(\boldsymbol{h}_t),$$

where

- $\boldsymbol{y}_t$ is the output
- $f_{\boldsymbol{W}_o}(\cdot)$ is a function with parameters $\boldsymbol{W}_o$
- $\boldsymbol{h}_t$ is the new state

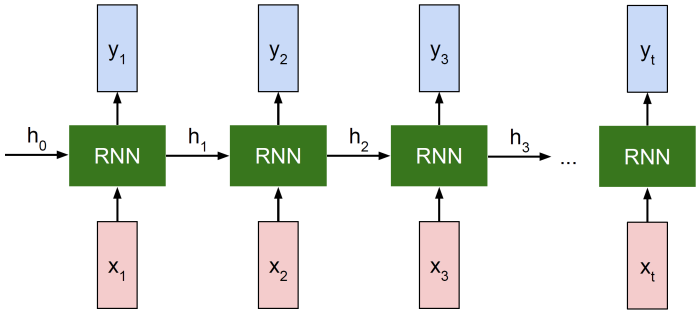
RNN

We can process a sequence of vectors $\boldsymbol{X} = [\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots]$ by applying a recurrence formula at every time step:

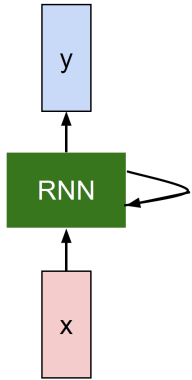
$$\boldsymbol{h}_t = f_{\boldsymbol{W}}(\boldsymbol{h}_{t-1}, \boldsymbol{x}_t),$$

output $\boldsymbol{y}_t$ at time-step t by using

$$\boldsymbol{y}_t = f_{\boldsymbol{W}_o}(\boldsymbol{h}_t).$$



$$\boldsymbol{h}_t = f_{\boldsymbol{W}}\left(\boldsymbol{h}_{t-1}, \boldsymbol{x}_t\right)$$



Note that the same function and the same set of parameters are used at every time step.

Simple recurrent neural network

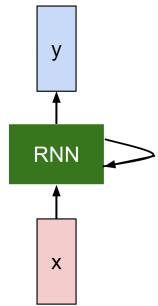
Let $\boldsymbol{h}_t = f_{\boldsymbol{W}}(\boldsymbol{h}_{t-1}, \boldsymbol{x}_t)$ be

$$\boldsymbol{h}_t = \tanh\Big(\boldsymbol{W}_{hh}\boldsymbol{h}_{t-1} + \boldsymbol{W}_{xh}\boldsymbol{x}_t\Big).$$

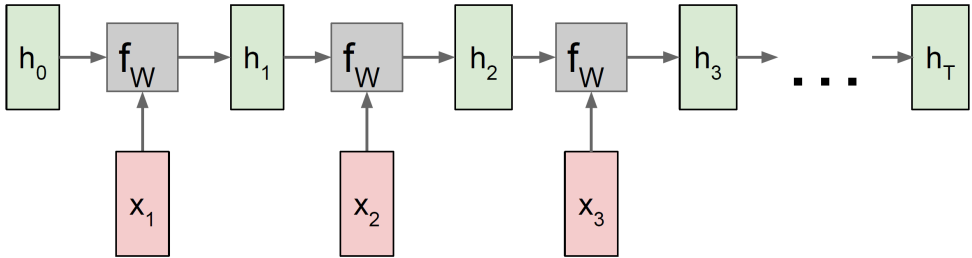
Therefore the simple RNN model can be formulated as

$$\boldsymbol{h}_t = \tanh\Big(\boldsymbol{W}_{hh}\boldsymbol{h}_{t-1} + \boldsymbol{W}_{xh}\boldsymbol{x}_t\Big)$$

$$\boldsymbol{y}_t = \boldsymbol{W}_{hy}\boldsymbol{h}_t \quad \text{or} \quad \boldsymbol{y}_t = \mathrm{softmax}(\boldsymbol{W}_{hy}\boldsymbol{h}_t)$$

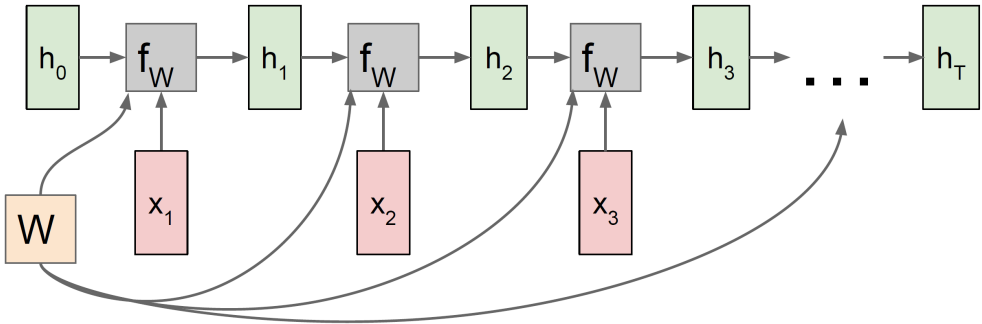


RNN: Computational graph



$$\boldsymbol{h}_t = f_{\boldsymbol{W}}(\boldsymbol{h}_{t-1}, \boldsymbol{x}_t)$$

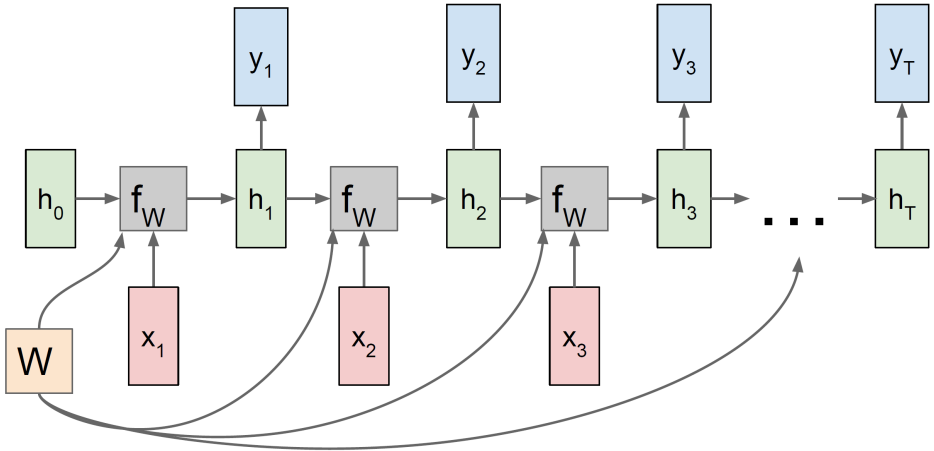
RNN: Computational graph



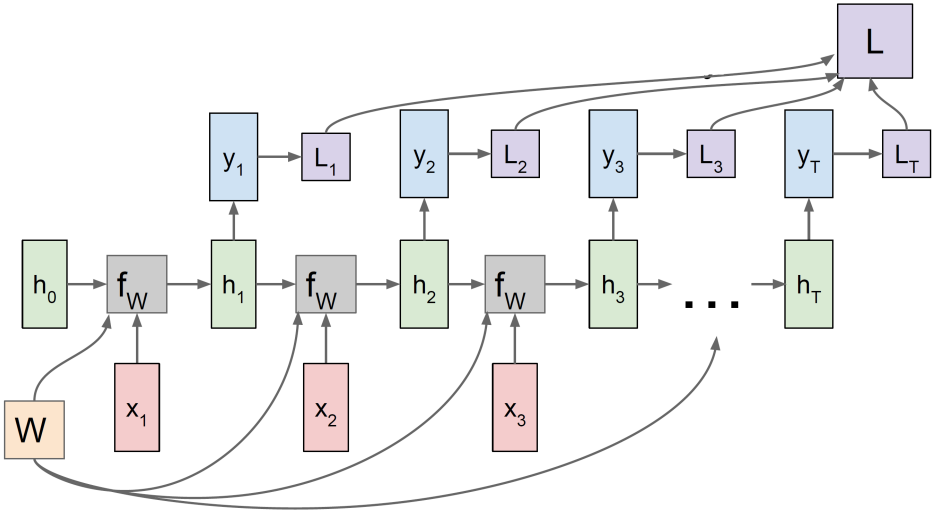
$$\boldsymbol{h}_t = f_{\boldsymbol{W}}(\boldsymbol{h}_{t-1}, \boldsymbol{x}_t)$$

Re-use the same weight matrix at every time-step.

RNN: Computational graph: Many-to-many

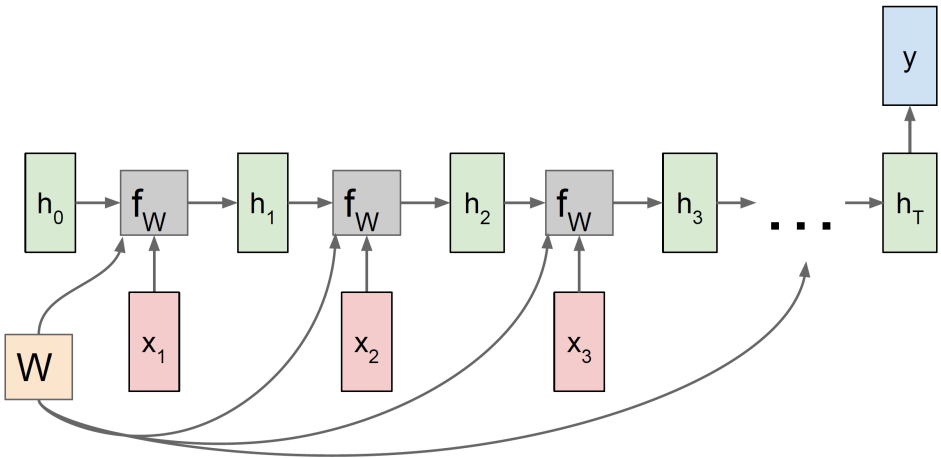


RNN: Computational graph: Many-to-many

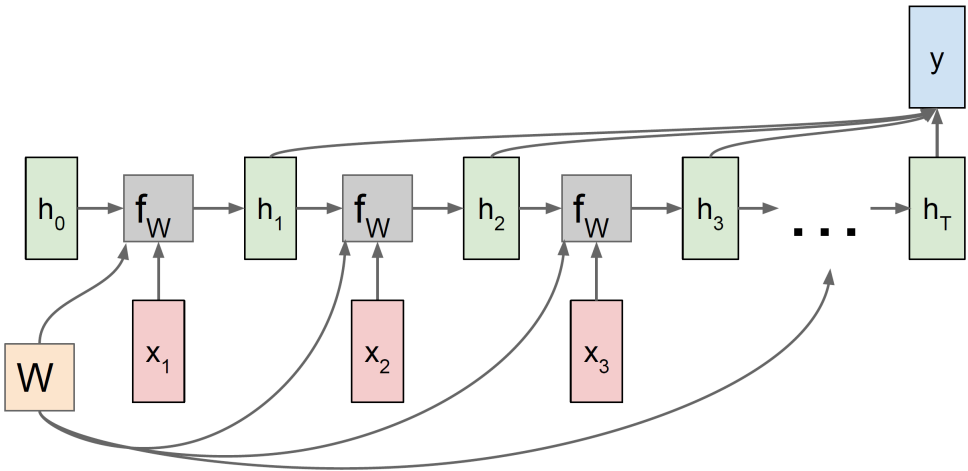


The loss is the sum of the loss at each slot.

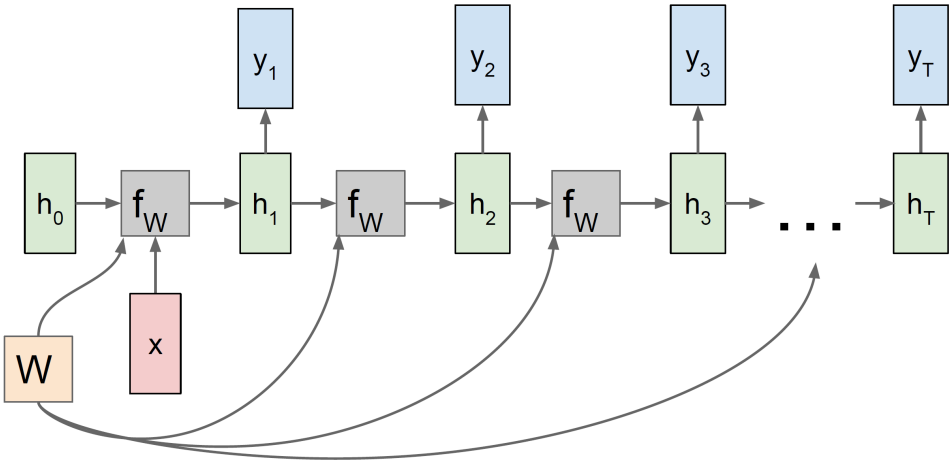
RNN: Computational graph: Many-to-one – I



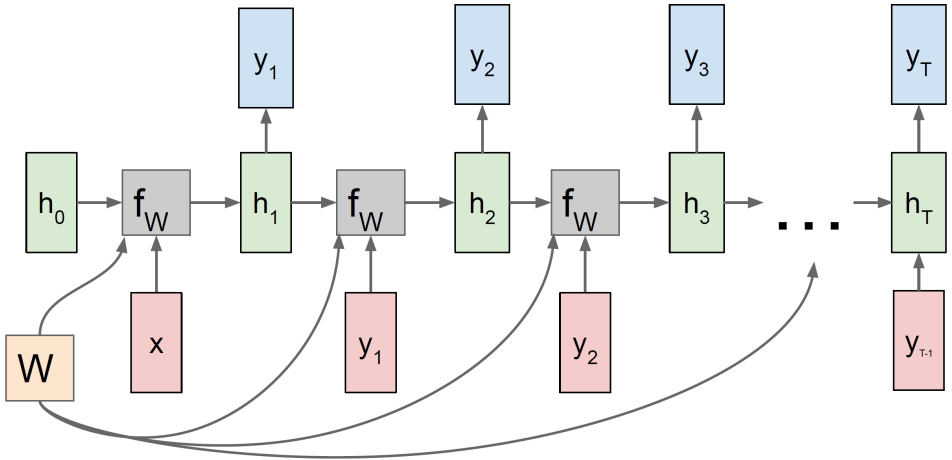
RNN: Computational graph: Many-to-one – II



RNN: Computational graph: One-to-many – I

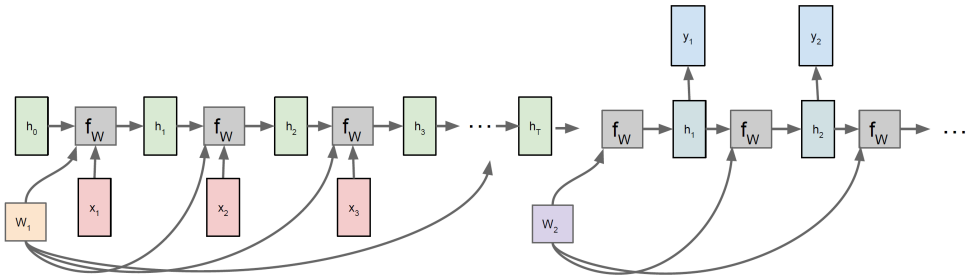


RNN: Computational graph: One-to-many – II



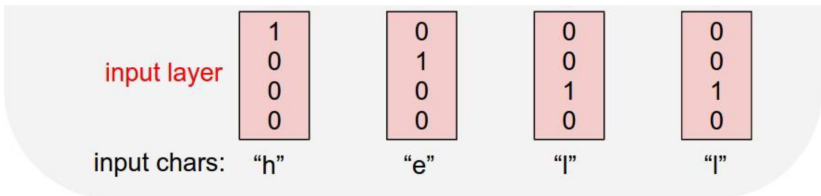
Sequence to sequence modeling: Many-to-one + One-to-many

- **Many-to-one:** Encode input sequence $(\boldsymbol{x}_1, \cdots, \boldsymbol{x}_T)$ in a single vector $\boldsymbol{h}_T$.
- **One-to-many:** Produce output sequence $(\boldsymbol{y}_1, \boldsymbol{y}_2, \cdots)$ from single input vector $\boldsymbol{h}_T$.

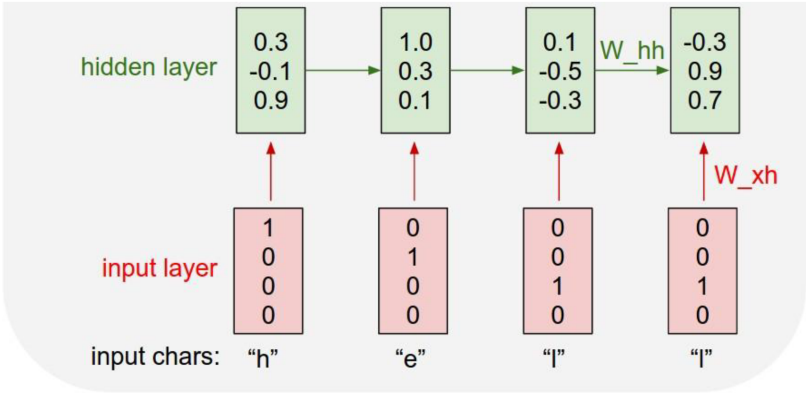


Example: Character-level language model

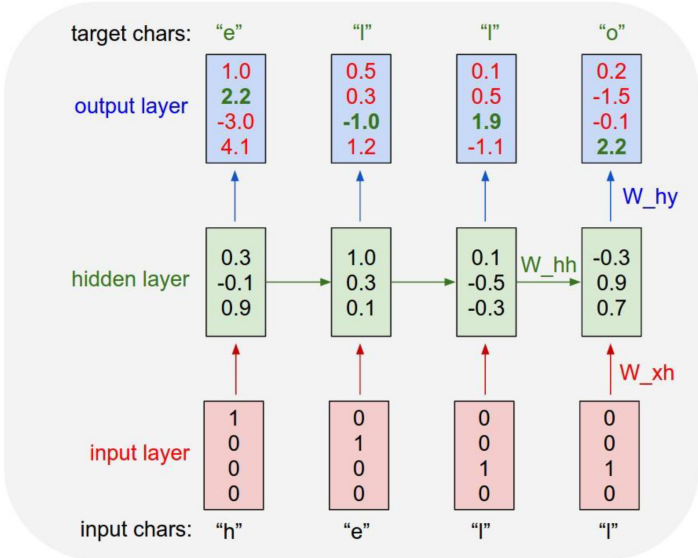
- Vocabulary: [h,e,l,o]
- Example training sequence: “hello”
- Task: predict the next character



$$\boldsymbol{h}_t = \tanh\left(\boldsymbol{W}_{hh}\boldsymbol{h}_{t-1} + \boldsymbol{W}_{xh}\boldsymbol{x}_t\right)$$

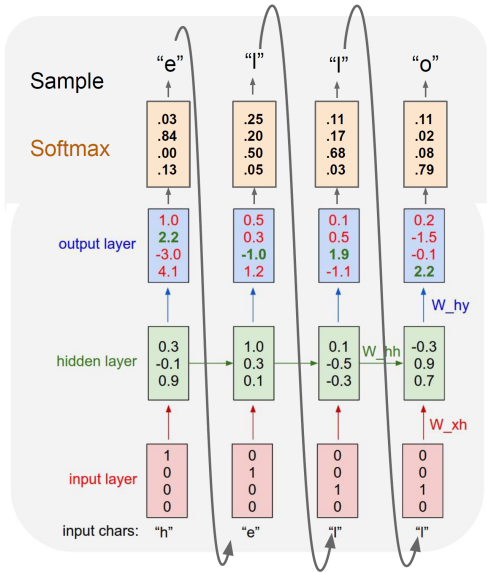


Training


target chars:
“e”
“l”
“l”
“o”
output layer
1.0
2.2
-3.0
4.1
0.5
0.3
-1.0
1.2
0.1
0.5
1.9
-1.1
0.2
-1.5
-0.1
2.2
W_hy
hidden layer
0.3
-0.1
0.9
1.0
0.3
0.1
0.1
-0.5
-0.3
W_hh
-0.3
0.9
0.7
W_xh
input layer
1
0
0
0
0
1
0
0
0
0
1
0
0
0
1
0
input chars:
“h”
“e”
“l”
“l”

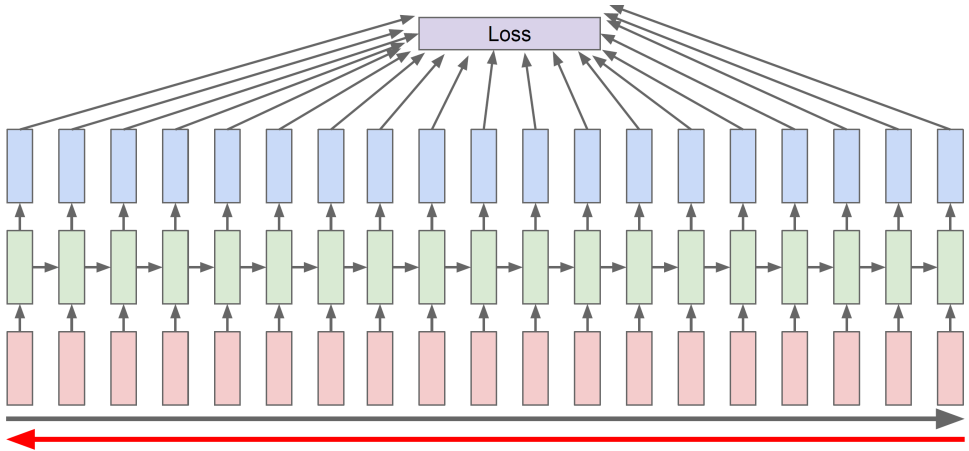
Testing

At test-time sample one characters at a time, and then feed back to model to predict the next.



Backpropagation through time

Forward through the entire sequence to compute loss, then backward through the entire sequence to compute the gradient. (We will discuss the algorithm later). — Note that the weight is shared!



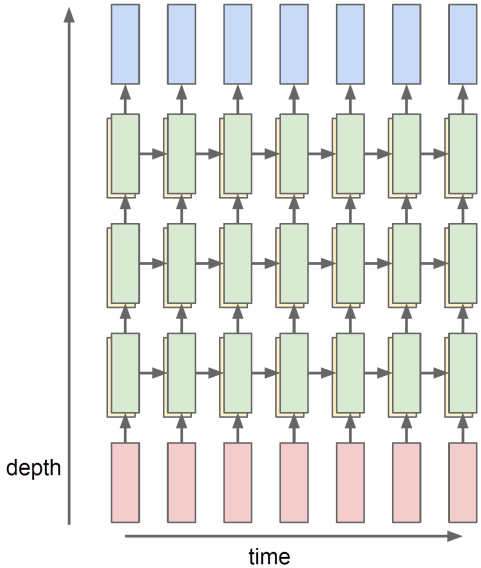
RNN Advantages:

- Can process any length input
- Computation for step t can (in theory) use information from many steps back
- Model size does not increase for longer input
- Same weights applied on every timestep, so there is symmetry in how inputs are processed.

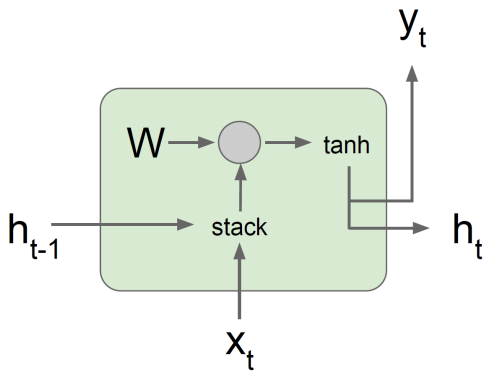
RNN Disadvantages:

- Recurrent computation is slow
- In practice, difficult to access information from many steps back

Multilayer RNNs

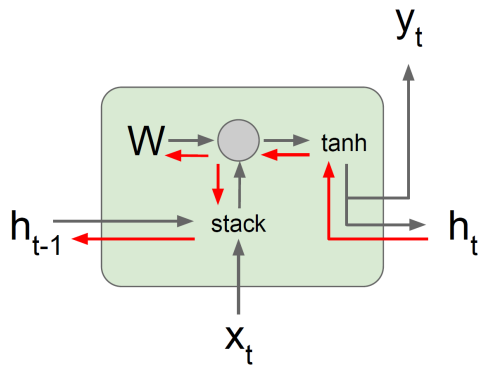


$$\boldsymbol{h}_t = \tanh\left(\boldsymbol{W}_{hh}\boldsymbol{h}_{t-1} + \boldsymbol{W}_{xh}\boldsymbol{x}_t\right) = \tanh\left((\boldsymbol{W}_{hh}\ \boldsymbol{W}_{hx})\begin{pmatrix}\boldsymbol{h}_{t-1}\\ \boldsymbol{x}_t\end{pmatrix}\right) := \tanh\left(\boldsymbol{W}\begin{pmatrix}\boldsymbol{h}_{t-1}\\ \boldsymbol{x}_t\end{pmatrix}\right)$$



BPTT

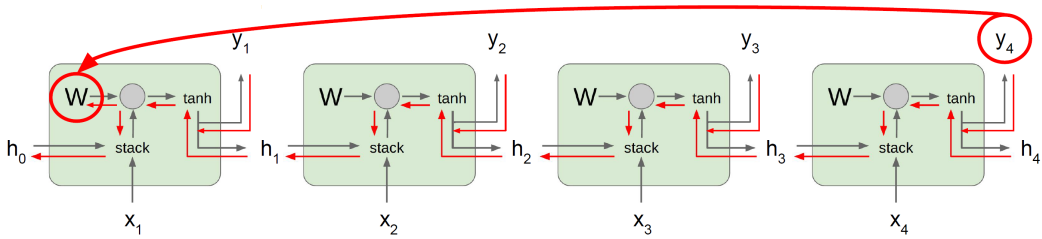
$$\boldsymbol{h}_t = \tanh\left(\boldsymbol{W}_{hh}\boldsymbol{h}_{t-1} + \boldsymbol{W}_{xh}\boldsymbol{x}_t\right) = \tanh\left((\boldsymbol{W}_{hh}\ \boldsymbol{W}_{hx})\begin{pmatrix}\boldsymbol{h}_{t-1}\\ \boldsymbol{x}_t\end{pmatrix}\right) := \tanh\left(\boldsymbol{W}\begin{pmatrix}\boldsymbol{h}_{t-1}\\ \boldsymbol{x}_t\end{pmatrix}\right)$$



$$\frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{h}_{t-1}} = \tanh'\left(\boldsymbol{W}_{hh}\boldsymbol{h}_{t-1} + \boldsymbol{W}_{xh}\boldsymbol{x}_t\right)\boldsymbol{W}_{hh}$$

Backpropagation from $\boldsymbol{h}_t$ to $\boldsymbol{h}_{t-1}$ multiplies by $\boldsymbol{W}$ (actually $\boldsymbol{W}_{hh}^\top$).

BPTT



Note that

$$\frac{\partial L}{\partial \boldsymbol{W}} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial \boldsymbol{W}},$$

where

$$\frac{\partial L_T}{\partial \boldsymbol{W}} = \frac{\partial L_T}{\partial \boldsymbol{h}_T} \cdot \frac{\partial \boldsymbol{h}_T}{\partial \boldsymbol{h}_{T-1}} \cdots \frac{\partial \boldsymbol{h}_1}{\partial \boldsymbol{W}} = \frac{\partial L_T}{\partial \boldsymbol{h}_T} \Big(\prod_{t=2}^{T} \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{h}_{t-1}} \Big) \frac{\partial \boldsymbol{h}_1}{\partial \boldsymbol{W}}$$

Vanishing and exploding gradient issue in BPTT

$$\frac{\partial L_T}{\partial \boldsymbol{W}} = \frac{\partial L_T}{\partial \boldsymbol{h}_T} \cdot \frac{\partial \boldsymbol{h}_T}{\partial \boldsymbol{h}_{T-1}} \cdots \frac{\partial \boldsymbol{h}_1}{\partial \boldsymbol{W}} = \frac{\partial L_T}{\partial \boldsymbol{h}_T} \Big(\prod_{t=2}^{T} \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{h}_{t-1}} \Big) \frac{\partial \boldsymbol{h}_1}{\partial \boldsymbol{W}}.$$

Since

$$\frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{h}_{t-1}} = \tanh' \Big(\boldsymbol{W}_{hh} \boldsymbol{h}_{t-1} + \boldsymbol{W}_{xh} \boldsymbol{x}_t \Big) \boldsymbol{W}_{hh},$$

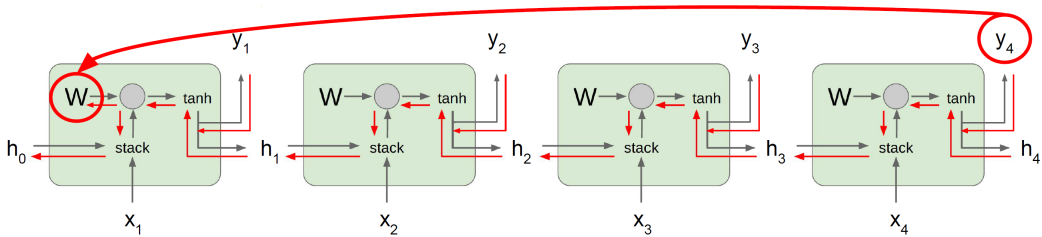
therefore

$$\frac{\partial L_T}{\partial \boldsymbol{W}} = \frac{\partial L_T}{\partial \boldsymbol{h}_T} \cdot \frac{\partial \boldsymbol{h}_T}{\partial \boldsymbol{h}_{T-1}} \cdots \frac{\partial \boldsymbol{h}_1}{\partial \boldsymbol{W}} = \frac{\partial L_T}{\partial \boldsymbol{h}_T} \Big(\prod_{t=2}^{T} \tanh' \left(\boldsymbol{W}_{hh} \boldsymbol{h}_{t-1} + \boldsymbol{W}_{xh} \boldsymbol{x}_t \right) \Big) \boldsymbol{W}_{hh}^{T-1} \frac{\partial \boldsymbol{h}_1}{\partial \boldsymbol{W}}.$$

Note that $\tanh' \left(\boldsymbol{W}_{hh} \boldsymbol{h}_{t-1} + \boldsymbol{W}_{xh} \boldsymbol{x}_t \right) < 1$ a.s. Hence, $\frac{\partial L_T}{\partial \boldsymbol{W}} \to 0$ as $T \to \infty$.

Vanishing gradient: bottleneck for learning long-term dependencies.

Vanishing and exploding gradient issue in BPTT



If we assume there is no nonlinearity, we have

$$\frac{\partial L_T}{\partial \boldsymbol{W}} = \frac{\partial L_T}{\partial \boldsymbol{h}_T} \boldsymbol{W}_{hh}^{T-1} \frac{\partial \boldsymbol{h}_1}{\partial \boldsymbol{W}}.$$

- Largest singular value $> 1 \Rightarrow$ exploding gradients. [Gradient clipping, i.e., scaling gradient if its norm is greater than 1.]
- Largest singular value $< 1 \Rightarrow$ vanishing gradients. [Change the RNN architecture.]

Long-short Term Memory (LSTM)

Hochreiter and Schmidhuber, "Long Short Term Memory", Neural Computation 1997.

Vanilla RNN:

$$\boldsymbol{h}_t = \tanh\left(\boldsymbol{W}\begin{pmatrix}\boldsymbol{h}_{t-1}\\ \boldsymbol{x}_t\end{pmatrix}\right)$$

LSTM:

$$\begin{pmatrix}\boldsymbol{i}\\ \boldsymbol{f}\\ \boldsymbol{o}\\ \boldsymbol{g}\end{pmatrix} = \begin{pmatrix}\sigma\\ \sigma\\ \sigma\\ \tanh\end{pmatrix}\boldsymbol{W}\begin{pmatrix}\boldsymbol{h}_{t-1}\\ \boldsymbol{x}_t\end{pmatrix},$$

$$\boldsymbol{c}_t = \boldsymbol{f}\odot\boldsymbol{c}_{t-1} + \boldsymbol{i}\odot\boldsymbol{g}$$

$$\boldsymbol{h}_t = \boldsymbol{o}\odot\tanh(\boldsymbol{c}_t)$$

where $\odot$ represents entry-wise product.

LSTM

$$\begin{pmatrix} \boldsymbol{i} \\ \boldsymbol{f} \\ \boldsymbol{o} \\ \boldsymbol{g} \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} \boldsymbol{W} \begin{pmatrix} \boldsymbol{h}_{t-1} \\ \boldsymbol{x}_t \end{pmatrix},$$

$$\boldsymbol{c}_t = \boldsymbol{f} \odot \boldsymbol{c}_{t-1} + \boldsymbol{i} \odot \boldsymbol{g}$$

$$\boldsymbol{h}_t = \boldsymbol{o} \odot \tanh(\boldsymbol{c}_t)$$

where

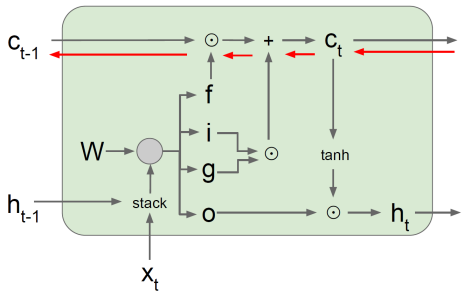
- $\boldsymbol{i}$: input gate, whether to write to cell
- $\boldsymbol{f}$: forget gate, whether to erase cell
- $\boldsymbol{o}$: output gate, how much to reveal cell
- $\boldsymbol{g}$: how much to write to cell

LSTM

$$\begin{pmatrix} \boldsymbol{i} \\ \boldsymbol{f} \\ \boldsymbol{o} \\ \boldsymbol{g} \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} \boldsymbol{W} \begin{pmatrix} \boldsymbol{h}_{t-1} \\ \boldsymbol{x}_t \end{pmatrix},$$

$$\boldsymbol{c}_t = \boldsymbol{f} \odot \boldsymbol{c}_{t-1} + \boldsymbol{i} \odot \boldsymbol{g}$$

$$\boldsymbol{h}_t = \boldsymbol{o} \odot \tanh(\boldsymbol{c}_t)$$



Remarks

LSTM does not guarantee that there is no vanishing/exploding gradient, but it does provide an easier way for the model to learn long-term dependencies in practice.

Momentum Recurrent Neural Networks

T. Nguyen, R. Baraniuk, A. Bertozzi, S. Osher, B. Wang, MomentumRNN: Integrating momentum into RNNs, NeurIPS, 2020.

Heavy-ball method

$$\boldsymbol{x}_t = \boldsymbol{x}_{t-1} - s\nabla f(\boldsymbol{x}_{t-1}) + \mu(\boldsymbol{x}_{t-1} - \boldsymbol{x}_{t-2}),$$

let $\boldsymbol{p}_t = \boldsymbol{x}_{t-1} - \boldsymbol{x}_t$, we can rewrite the heavy-ball method as

$$\boldsymbol{p}_t = \mu\boldsymbol{p}_{t-1} + s\nabla f(\boldsymbol{x}_{t-1}); \quad \boldsymbol{x}_t = \boldsymbol{x}_{t-1} - \boldsymbol{p}_t.$$

Heavy-ball method vs. RNN

Heavy-ball method:

$$\boldsymbol{p}_t = \mu \boldsymbol{p}_{t-1} + s\nabla f(\boldsymbol{x}_{t-1}); \quad \boldsymbol{x}_t = \boldsymbol{x}_{t-1} - \boldsymbol{p}_t.$$

Also, the recurrence equation of RNN can be written as

$$\boldsymbol{h}_t = \sigma\big(\boldsymbol{U}\boldsymbol{h}_{t-1} + \boldsymbol{W}\boldsymbol{x}_t\big),$$

let $\phi(\cdot) = \sigma(\boldsymbol{U}(\cdot))$ and $\boldsymbol{u}_t = \boldsymbol{U}^{-1}\boldsymbol{W}\boldsymbol{x}_t$, we can rewrite the above equation as

$$\boldsymbol{h}_t = \phi(\boldsymbol{h}_{t-1} + \boldsymbol{u}_t).$$

What if we treat $-\boldsymbol{u}_t$ in RNN as $\nabla f(\boldsymbol{x}_t)$ in the heavy-ball method? Also, regard $\phi(\cdot)$ as a projection.

Momentum RNN

$$\boldsymbol{p}_t = \mu \boldsymbol{p}_{t-1} - s\boldsymbol{u}_t; \quad \boldsymbol{h}_t = \phi(\boldsymbol{h}_{t-1} - \boldsymbol{p}_t),$$

where $\mu \geq 0, s > 0$ are two hyperparameters.

Let $\boldsymbol{v}_t := -\boldsymbol{U}\boldsymbol{p}_t$, we arrive at the following Momentum RNN

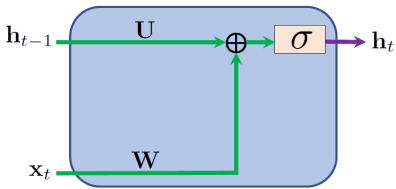
$$\boldsymbol{v}_t = \mu \boldsymbol{v}_{t-1} + s\boldsymbol{W}\boldsymbol{x}_t; \quad \boldsymbol{h}_t = \sigma\big(\boldsymbol{U}\boldsymbol{h}_{t-1} + \boldsymbol{v}_t\big).$$

Momentum RNN

Recurrent Cell

$$\mathbf{h}_t = \phi(\mathbf{U} \times \mathbf{h}_{t-1} + \mathbf{W} \times \mathbf{x}_t)$$

gradient

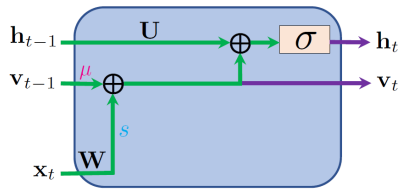


Momentum Cell

$$\mathbf{v}_t = \mu \times \mathbf{v}_{t-1} + s \times \mathbf{W} \times \mathbf{x}_t$$

momentum stepsize

$$\mathbf{h}_t = \sigma(\mathbf{U} \times \mathbf{h}_{t-1} + \mathbf{v}_t)$$



RNN BPTT:

$$\frac{\partial L}{\partial \boldsymbol{h}_t} = \frac{\partial L}{\partial \boldsymbol{h}_T} \cdot \frac{\partial \boldsymbol{h}_T}{\partial \boldsymbol{h}_t} = \frac{\partial L}{\partial \boldsymbol{h}_T} \cdot \prod_{k=t}^{T-1} \frac{\partial \boldsymbol{h}_{k+1}}{\partial \boldsymbol{h}_k} = \frac{\partial L}{\partial \boldsymbol{h}_T} \cdot \prod_{k=t}^{T-1} (\boldsymbol{D}_k \boldsymbol{U}^\top),$$

where $\boldsymbol{D}_k = \mathrm{diag}(\sigma'(\boldsymbol{U}\boldsymbol{h}_k + \boldsymbol{W}\boldsymbol{x}_{k+1}))$ is a diagonal matrix with $\sigma'(\boldsymbol{U}\boldsymbol{h}_k + \boldsymbol{W}\boldsymbol{x}_{k+1})$ being its diagonal entries.

Momentum RNN overcomes vanishing gradients

Momentum RNN:

$$\boldsymbol{v}_t = \mu \boldsymbol{v}_{t-1} + s\boldsymbol{W}\boldsymbol{x}_t; \quad \boldsymbol{h}_t = \sigma\big(\boldsymbol{U}\boldsymbol{h}_{t-1} + \boldsymbol{v}_t\big),$$

which can be rewritten as

$$\boldsymbol{h}_t = \sigma\bigg(\boldsymbol{U}(\boldsymbol{h}_{t-1} - \mu\boldsymbol{h}_{t-2}) + \mu\sigma^{-1}(\boldsymbol{h}_{t-1}) + s\boldsymbol{W}\boldsymbol{x}_t\bigg),$$

where $\sigma^{-1}(\cdot)$ is the inverse function of $\sigma(\cdot)$. We compute $\partial L/\partial \boldsymbol{h}_t$ as follows

$$\frac{\partial L}{\partial \boldsymbol{h}_t} = \frac{\partial L}{\partial \boldsymbol{h}_T} \cdot \frac{\partial \boldsymbol{h}_T}{\partial \boldsymbol{h}_t} = \frac{\partial L}{\partial \boldsymbol{h}_T} \cdot \prod_{k=t}^{T-1} \frac{\partial \boldsymbol{h}_{k+1}}{\partial \boldsymbol{h}_k} = \frac{\partial L}{\partial \boldsymbol{h}_T} \cdot \prod_{k=t}^{T-1} \widehat{\boldsymbol{D}}_k[\boldsymbol{U}^\top + \mu\boldsymbol{\Sigma}_k], \qquad (1)$$

where $\widehat{\boldsymbol{D}}_k = \mathrm{diag}(\sigma'(\boldsymbol{U}(\boldsymbol{h}_k - \mu\boldsymbol{h}_{k-1}) + \mu\sigma^{-1}(\boldsymbol{h}_k) + s\boldsymbol{W}\boldsymbol{x}_{k+1}))$ and $\boldsymbol{\Sigma} = \mathrm{diag}((\sigma^{-1})'(\boldsymbol{h}_k))$. For mostly used σ, e.g., sigmoid and tanh, $(\sigma^{-1}(\cdot))' > 1$ and $\mu\boldsymbol{\Sigma}_k$ dominates $\mathrm{U}^\top$.

Neural Ordinary Differential Equations

R. Chen, Y. Rubanova, J. Bettencourt, and D. Duvenaud, Neural Ordinary Differential Equations, NeurIPS 2018.

- RNNs: Information carried by the time interval between different frames is missing.

- Neural ODE:

$$\frac{d\boldsymbol{h}(t)}{dt} = f(\boldsymbol{h}(t), t, \theta),$$

where $f(\boldsymbol{h}(t), t, \theta)$ is a neural network, e.g., $f(\boldsymbol{h}(t), t, \theta) = \theta_2 \sigma(\theta_1 \boldsymbol{h}(t))$.

• **Forward propagation:** starting from the input $\boldsymbol{h}(0)$, we define the output layer $\boldsymbol{h}(T)$ to be the solution to the above ODE initial value problem at some time T.

• $\boldsymbol{h}(T)$ can be computed by a black-box numerical ODE solver, which evaluates the hidden unit dynamics f whenever necessary to determine the solution with the desired accuracy.

• Forward Euler solver:

$$\boldsymbol{h}_{k+1} = \boldsymbol{h}_k + sf(\boldsymbol{h}_k, t_k, \theta)$$

• **Loss function:** consider the scalar-valued loss function $L(\cdot)$, whose input is the result of an ODE solver

$$L(\boldsymbol{h}(T)) = L\Big(h(0) + \int_0^T f(\boldsymbol{h}(t), t, \theta)dt\Big) = L\big(\mathrm{ODESolve}(\boldsymbol{h}(0), f, 0, T, \theta)\big).$$

Adjoint state: $\boldsymbol{a}(t) := \frac{\partial L}{\partial \boldsymbol{h}(t)}$

Theorem 1.

$$\frac{dL}{d\theta} = -\int_T^0 \boldsymbol{a}(t)^\top \frac{\partial f(\boldsymbol{h}(t), t, \theta)}{\partial \theta} dt,$$

where $\boldsymbol{a}(t)$ satisfies the following adjoint ODE

$$\frac{d\boldsymbol{a}(t)}{dt} = -\boldsymbol{a}(t)^\top \frac{\partial f(\boldsymbol{h}(t), t, \theta)}{\partial \boldsymbol{h}}.$$

Eliminate the use of backpropagation.

Proof.

- Let $\boldsymbol{h}(t)$ follow the ODE $\frac{d\boldsymbol{h}(t)}{dt} = f(\boldsymbol{h}(t), t, \theta)$, where θ are the parameters.

- We will prove that if we define an adjoint state

$$\boldsymbol{a}(t) := \frac{dL}{d\boldsymbol{h}(t)}, \tag{2}$$

then it follows the following ODE

$$\frac{d\boldsymbol{a}(t)}{dt} = -\boldsymbol{a}(t)^\top \frac{\partial f(\boldsymbol{h}(t), t, \theta)}{\partial \boldsymbol{h}(t)}. \tag{3}$$

For ease of notation, we denote vectors as row vectors, whereas the vectors in the previous part are column vectors.

• The adjoint state is the gradient w.r.t. the hidden state at time t. In neural networks, the gradient of a hidden layer $\boldsymbol{h}_t$ depends on the gradient from $\boldsymbol{h}_{t+1}$ by chain rule

$$\frac{dL}{d\boldsymbol{h}_t} = \frac{dL}{d\boldsymbol{h}_{t+1}} \frac{d\boldsymbol{h}_{t+1}}{d\boldsymbol{h}_t}. \tag{4}$$

• With a continuous hidden state, we can write the transformation after an ϵ change in time as (note $\dot{\boldsymbol{h}}(t) = f(\boldsymbol{h}(t), t, \theta)$, implying $\boldsymbol{h}(t) = \int f(\boldsymbol{h}(t), t, \theta) dt$)

$$\boldsymbol{h}(t+\epsilon) = \int_t^{t+\epsilon} f(\boldsymbol{h}(t), t, \theta) dt + \boldsymbol{h}(t) := T_\epsilon(\boldsymbol{h}(t), t) \tag{5}$$

and chain rule can also be applied ($\partial \boldsymbol{h}(t+\epsilon)/\partial \boldsymbol{h}(t) = \partial T_\epsilon(\boldsymbol{h}(t), t)/\partial \boldsymbol{h}(t)$)

$$\frac{dL}{d\boldsymbol{h}(t)} = \frac{dL}{d\boldsymbol{h}(t+\epsilon)} \frac{d\boldsymbol{h}(t+\epsilon)}{d\boldsymbol{h}(t)} \quad \text{or} \quad \boldsymbol{a}(t) = \boldsymbol{a}(t+\epsilon) \frac{\partial T_\epsilon(\boldsymbol{h}(t), t)}{\partial \boldsymbol{h}(t)}. \tag{6}$$

- The proof of (3) follows from the definition of derivative:

$$
\begin{aligned}
&\frac{d\boldsymbol{a}(t)}{dt} \\
&= \lim_{\epsilon \to 0^+} \frac{\boldsymbol{a}(t+\epsilon) - \boldsymbol{a}(t)}{\epsilon} \\
&= \lim_{\epsilon \to 0^+} \frac{\boldsymbol{a}(t+\epsilon) - \boldsymbol{a}(t+\epsilon)\partial(T_\epsilon(\boldsymbol{h}(t)))/(\partial \boldsymbol{h}(t))}{\epsilon} \text{ (by (6))} \\
&= \lim_{\epsilon \to 0^+} \frac{\boldsymbol{a}(t+\epsilon) - \boldsymbol{a}(t+\epsilon)\frac{\partial}{\partial \boldsymbol{h}(t)}\left(\boldsymbol{h}(t) + \epsilon f(\boldsymbol{h}(t), t, \theta) + \mathcal{O}(\epsilon^2)\right)}{\epsilon} \text{ (Taylor series of } \boldsymbol{h}(t+\epsilon) = T_\epsilon(\boldsymbol{h}(t), \epsilon)) \\
&= \lim_{\epsilon \to 0^+} \frac{\boldsymbol{a}(t+\epsilon) - \boldsymbol{a}(t+\epsilon)\left(\boldsymbol{I} + \epsilon \frac{\partial f(\boldsymbol{h}(t), t, \theta)}{\partial \boldsymbol{h}(t)} + \mathcal{O}(\epsilon^2)\right)}{\epsilon} \qquad (7) \\
&= \lim_{\epsilon \to 0^+} \frac{-\epsilon \boldsymbol{a}(t+\epsilon)\frac{\partial f(\boldsymbol{h}(t), t, \theta)}{\partial \boldsymbol{h}(t)} + \mathcal{O}(\epsilon^2)}{\epsilon} \\
&= \lim_{\epsilon \to 0^+} -\boldsymbol{a}(t+\epsilon)\frac{\partial f(\boldsymbol{h}(t), t, \theta)}{\partial \boldsymbol{h}(t)} + \mathcal{O}(\epsilon) \\
&= -\boldsymbol{a}(t)\frac{\partial f(\boldsymbol{h}(t), t, \theta)}{\partial \boldsymbol{h}(t)}
\end{aligned}
$$

• Next, we compute gradient w.r.t. θ. We can generalize (3) to obtain gradient w.r.t. θ and t. We view θ ($\theta(t) = \theta$) and t ($t(t) = t$) as states with constant differential equations and write

$$\frac{\partial \theta(t)}{\partial t} = 0, \quad \frac{dt(t)}{dt} = 1. \tag{8}$$

• We can then combine these with $\boldsymbol{h}$ to form an augmented state with corresponding differential equation and adjoint state,

$$\frac{d}{dt} \begin{pmatrix} \boldsymbol{h} \\ \theta \\ t \end{pmatrix} (t) = f_{aug}([\boldsymbol{h}, \theta, t]) := \begin{pmatrix} f([\boldsymbol{h}, \theta, t] \\ 0 \\ 1 \end{pmatrix}, \tag{9}$$

- The Jacobian of f_{aug} has the form

$$\frac{\partial f_{aug}(t)}{\partial [\boldsymbol{h}, \theta, t]} = \begin{pmatrix} \frac{\partial f}{\partial \boldsymbol{h}} & \frac{\partial f}{\partial \theta} & \frac{\partial f}{\partial t} \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} (t) \tag{10}$$

where 0 is a matrix of zeros with the appropriate dimensions.

- We plug this into (3) to obtain

$$\frac{d\boldsymbol{a}_{aug}(t)}{dt} = -[\boldsymbol{a}(t)\ \boldsymbol{a}_\theta(t)\ \boldsymbol{a}_t(t)] \frac{\partial f_{aug}}{\partial [\boldsymbol{h}, \theta, t]}(t) = -\left[\boldsymbol{a}\frac{\partial f}{\partial \boldsymbol{h}}\ \boldsymbol{a}\frac{\partial f}{\partial \theta}\ \boldsymbol{a}\frac{\partial f}{\partial t}\right](t).$$

where

$$\boldsymbol{a}_{aug} := \begin{pmatrix} \boldsymbol{a} \\ \boldsymbol{a}_\theta \\ \boldsymbol{a}_t \end{pmatrix},\ \boldsymbol{a}_\theta(t) := \frac{dL}{d\theta(t)},\ \boldsymbol{a}_t(t) := \frac{dL}{dt(t)}.$$

• The first element is the adjoint differential equation (3), as expected. The second element can be used to obtain the total gradient w.r.t. the parameters, by integrating over the full interval and setting $\boldsymbol{a}_\theta(T) = 0$ (L is independent of $\theta(T)$).

$$\frac{dL}{d\theta} = \boldsymbol{a}_\theta(0) = -\int_T^0 \boldsymbol{a}(t)\frac{\partial f(\boldsymbol{h}(t), t, \theta)}{\partial \theta} dt. \tag{11}$$

Advantages of Neural ODEs

• **Memory efficiency:** Recall that in backpropagation we need to store any intermediate quantities of the forward pass, which is expensive in memory footprint. Later, we will see that in training neural ODEs we do not need to store these intermediate quantities.

• **Continuous time-series models:** Neural ODE can learn time-series that are irregularly-observed in time, which is significantly different from RNNs.

Computational bottleneck of neural ODEs

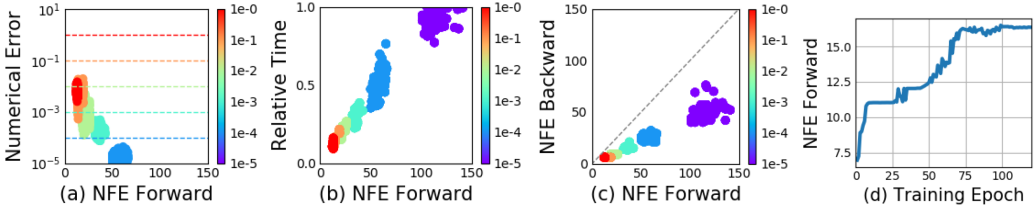


Figure 3: Statistics of a trained ODE-Net. (NFE = number of function evaluations.)

NFEs can be excessively high in training neural ODEs, resulting in complicated models.

As training goes, the stiffness of NODE keeps increasing!

Learning bottleneck of neural ODEs

Lemma 1. For neural ODEs, we have

$$\frac{\partial L}{\partial \boldsymbol{h}_t} = \frac{\partial L}{\partial \boldsymbol{h}_T}\frac{\partial \boldsymbol{h}_T}{\partial \boldsymbol{h}_t} = \frac{\partial L}{\partial \boldsymbol{h}_T}\exp\Big\{-\int_T^t \frac{\partial f}{\partial \boldsymbol{h}}(\boldsymbol{h}(s), s, \theta)ds\Big\},$$

the adjoint state $\partial L/\partial \boldsymbol{h}_t$ vanishes quickly, making neural ODE cannot learn long-term dependencies.

Note that if $\int_T^t \frac{\partial f}{\partial \boldsymbol{h}}(\boldsymbol{h}(s), s, \theta)$ is large, then $\frac{\partial L}{\partial \boldsymbol{h}_t} \approx 0$, i.e., vanishing gradient.

Remark. BPTT for training the RNN model $\boldsymbol{h}_t = \sigma(\boldsymbol{U}\boldsymbol{h}_{t-1} + \boldsymbol{W}\boldsymbol{x}_t)$:

$$\frac{\partial L}{\partial \boldsymbol{h}_t} = \frac{\partial L}{\partial \boldsymbol{h}_T}\cdot\frac{\partial \boldsymbol{h}_T}{\partial \boldsymbol{h}_t} = \frac{\partial L}{\partial \boldsymbol{h}_T}\cdot\prod_{k=t}^{T-1}\frac{\partial \boldsymbol{h}_{k+1}}{\partial \boldsymbol{h}_k} = \frac{\partial L}{\partial \boldsymbol{h}_T}\cdot\prod_{k=t}^{T-1}(\boldsymbol{D}_k\boldsymbol{U}^\top),$$

where $\boldsymbol{D}_k = \mathrm{diag}(\sigma'(\boldsymbol{U}\boldsymbol{h}_k + \boldsymbol{W}\boldsymbol{x}_{k+1}))$ is a diagonal matrix with $\sigma'(\boldsymbol{U}\boldsymbol{h}_k + \boldsymbol{W}\boldsymbol{x}_{k+1})$ being its diagonal entries.

Heavy-ball Neural Ordinary Differential Equations

H. Xia*, V. Suliafu*, H. Ji, T. Nguyen, A. Bertozzi, S. Osher, B. Wang, Heavy-ball Neural Ordinary Differential Equations, NeurIPS 2021.

Drawbacks of neural ODEs

- Computationally expensive: requiring solving an ODE in both forward and backward propagation.

- Cannot learn long-range dependencies: For neural ODEs, we have

$$\frac{\partial L}{\partial \boldsymbol{h}_t} = \frac{\partial L}{\partial \boldsymbol{h}_T}\frac{\partial \boldsymbol{h}_T}{\partial \boldsymbol{h}_t} = \frac{\partial L}{\partial \boldsymbol{h}_T}\exp\Big\{-\int_T^t \frac{\partial f}{\partial \boldsymbol{h}}(\boldsymbol{h}(s), s, \theta)ds\Big\},$$

the adjoint state $\partial L/\partial \boldsymbol{h}_t$ vanishes quickly, making neural ODE cannot learn long-term dependencies.

Recap: Derivation of the heavy-ball ODE

$$\boldsymbol{x}^{t+1} = \boldsymbol{x}^t - \eta \nabla f(\boldsymbol{x}^t) + \theta(\boldsymbol{x}^t - \boldsymbol{x}^{t-1}),$$

Let $\boldsymbol{m}^{t+1} := (\boldsymbol{x}^{t+1} - \boldsymbol{x}^t)/\sqrt{\eta}$ and let $\theta := 1 - \gamma\sqrt{\eta}$, where $\gamma \geq 0$ is another hyperparameter. Then we have

$$\frac{\boldsymbol{x}^{t+1} - \boldsymbol{x}^t}{\sqrt{\eta}} = -\sqrt{\eta}\nabla f(\boldsymbol{x}^t) + (1 - \gamma\sqrt{\eta})\frac{\boldsymbol{x}^t - \boldsymbol{x}^{t-1}}{\sqrt{\eta}},$$

therefore, we can rewrite the heavy-ball method as

$$\boldsymbol{m}^{t+1} = (1 - \gamma\sqrt{\eta})\boldsymbol{m}^t - \sqrt{\eta}\nabla f(\boldsymbol{x}^t);\ \ \boldsymbol{x}^{t+1} = \boldsymbol{x}^t + \sqrt{\eta}\boldsymbol{m}^{t+1}.$$

Let $\eta \to 0$; we obtain the following system of first-order ODEs

$$\frac{d\boldsymbol{X}(t)}{dt} = \boldsymbol{M}(t); \ \frac{d\boldsymbol{M}(t)}{dt} = -\gamma \boldsymbol{M}(t) - \nabla f(\boldsymbol{X}(t)),$$

which can be further written as

$$\ddot{\boldsymbol{X}}(\tau) + \gamma \dot{\boldsymbol{X}}(\tau) + \nabla f(\boldsymbol{X}(\tau)) = 0.$$

What if we parameterize $-\nabla f(\boldsymbol{X}(\tau))$ by a neural network?

Heavy-ball neural ODE can accelerate forward and backward propagation.

Heavy-ball neural ODE (HBNODE)

$$\frac{d^2\boldsymbol{h}(t)}{dt^2} + \gamma \frac{d\boldsymbol{h}(t)}{dt} = f(\boldsymbol{h}(t), t, \theta), \quad \gamma \geq 0. \tag{12}$$

The above HBNODE is equivalent to the following system of first-order neural ODEs

$$\frac{d\boldsymbol{h}(t)}{dt} = \boldsymbol{m}(t); \quad \frac{d\boldsymbol{m}(t)}{dt} = -\gamma \boldsymbol{m}(t) + f(\boldsymbol{h}(t), t, \theta). \tag{13}$$

Let us compare the stiffness of the following two ODE models

$$\frac{d\boldsymbol{h}(t)}{dt} = \boldsymbol{A}\boldsymbol{h}(t)$$

and

$$\begin{aligned} \frac{d\boldsymbol{h}(t)}{dt} &= \boldsymbol{m}(t) \\ \frac{d\boldsymbol{m}(t)}{dt} &= -\gamma \boldsymbol{m}(t) + \boldsymbol{A}\boldsymbol{h}(t), \end{aligned} \quad \text{i.e.,} \quad \frac{d}{dt}\begin{pmatrix} \boldsymbol{h}(t) \\ \boldsymbol{m}(t) \end{pmatrix} = \underbrace{\begin{pmatrix} 0 & \boldsymbol{I} \\ \boldsymbol{A} & -\gamma \boldsymbol{I} \end{pmatrix}}_{:=\boldsymbol{B}} \begin{pmatrix} \boldsymbol{h}(t) \\ \boldsymbol{m}(t) \end{pmatrix}$$

The stiffness of ODEs is defined by the ratio between the largest and the smallest eigenvalues (in absolute value). We consider the case when $\boldsymbol{A}$ is positive definite.

Stiffness of the linearized models

Let the eigenvalues and the corresponding eigenvectors of $\boldsymbol{A} \in \mathbb{R}^{d\times d}$ be $\{\lambda_i, \boldsymbol{v}_i\}_{i=1}^{d}$. We assume the eigenvectors of $\boldsymbol{B}$ has the following form

$$\begin{pmatrix} \boldsymbol{v}_i \\ c\boldsymbol{v}_i \end{pmatrix}, \quad \text{where } c \text{ is a constant,}$$

then we have

$$\begin{pmatrix} 0 & \boldsymbol{I} \\ \boldsymbol{A} & -\gamma \boldsymbol{I} \end{pmatrix} \begin{pmatrix} \boldsymbol{v}_i \\ c\boldsymbol{v}_i \end{pmatrix} = \begin{pmatrix} c\boldsymbol{v}_i \\ \lambda_i \boldsymbol{v}_i - \gamma c \boldsymbol{v}_i \end{pmatrix}$$

Let $\tilde{\lambda}$ be the eigenvalue of $\boldsymbol{B}$, then we have

$$\frac{c}{1} = \tilde{\lambda} = \frac{\lambda_i - \gamma c}{c}$$

Thus:

$$c^2 + \gamma c - \lambda_i = 0 \implies c = \tilde{\lambda} = \frac{\gamma \pm \sqrt{\gamma^2 + 4\lambda_i}}{2}$$

Each eigenvalue λ_i of the matrix $\boldsymbol{A}$ correspond to two eigenvalues of the matrix $\boldsymbol{B}$, given by

$$\tilde{\lambda}_{i,1} = \frac{\gamma + \sqrt{\gamma^2 + 4\lambda_i}}{2}, \quad \text{and} \quad \tilde{\lambda}_{i,2} = \frac{\gamma - \sqrt{\gamma^2 + 4\lambda_i}}{2}.$$

We assume the eigenvalues of $\boldsymbol{A}$ are sorted as follows:

$$0 < \lambda_{\min} \leq \lambda_2 \leq \lambda_3 \leq \ldots \leq \lambda_{\max}.$$

Therefore, we have

$$\tilde{\lambda}_{\max} = \frac{\gamma + \sqrt{\gamma^2 + 4\lambda_{\max}}}{2} \quad \text{and} \quad |\tilde{\lambda}_{\min}| = \left| \frac{\gamma - \sqrt{\gamma^2 + 4\lambda_{\min}}}{2} \right|.$$

Thus the stiffness ratio of the matrix $\boldsymbol{B}$ satisfies

$$S(\boldsymbol{B}) = \frac{\gamma + \sqrt{\gamma^2 + 4\lambda_{\max}}}{|\gamma - \sqrt{\gamma^2 + 4\lambda_{\min}}|} \underbrace{\leq}_{\text{let } \gamma = 0} \sqrt{\frac{\lambda_{\max}}{\lambda_{\min}}} = \sqrt{S(\boldsymbol{A})}.$$

Analysis of the adjoint state of the heavy-ball neural ODEs.

Theorem 2. The adjoint state $\boldsymbol{a_h}$ and $\boldsymbol{a_m}$ of the following HBNODE

$$\frac{d\boldsymbol{h}(t)}{dt} = \boldsymbol{m}(t); \quad \frac{d\boldsymbol{m}(t)}{dt} = -\gamma \boldsymbol{m}(t) + f(\boldsymbol{h}(t), t, \theta),$$

satisfies the following first-order ODEs

$$\frac{d\boldsymbol{a_h}}{dt} = \boldsymbol{a_m}(t)\frac{\partial f}{\partial \boldsymbol{h}}; \quad \frac{d\boldsymbol{a_m}}{dt} = -\boldsymbol{a_h}(t) + \gamma \boldsymbol{a_m}(t). \tag{14}$$

Proof of Theorem 2.

• First, recall the adjoint state $\boldsymbol{a}(t)$ of the neural ODE $\frac{d\boldsymbol{h}(t)}{dt} = f(\boldsymbol{h}(t), t, \theta)$ satisfy the following ODE

$$\frac{d\boldsymbol{a}(t)}{dt} = -\boldsymbol{a}(t)\frac{\partial f(\boldsymbol{h}(t), t, \theta)}{\partial \boldsymbol{h}(t)}.$$

• Let $\boldsymbol{n}(t) = [\boldsymbol{h}(t)\ \boldsymbol{m}(t)]$ and $\boldsymbol{a}(t) = [\boldsymbol{a}_{\boldsymbol{h}}(t)\ \boldsymbol{a}_{\boldsymbol{m}}(t)]$, then we can rewrite HBNODE as

$$\frac{d}{dt}[\boldsymbol{h}(t),\ \ \boldsymbol{m}(t)] = [\boldsymbol{m}(t),\ -\gamma \boldsymbol{m}(t) + f(\boldsymbol{h}(t), t, \theta)]$$

• Therefore, we have

$$\frac{d}{dt}[\boldsymbol{a}_{\boldsymbol{h}}\ \ \boldsymbol{a}_{\boldsymbol{m}}] = -[\boldsymbol{a}_{\boldsymbol{h}}\ \ \boldsymbol{a}_{\boldsymbol{m}}] \begin{bmatrix} \frac{\partial \boldsymbol{m}}{\partial \boldsymbol{h}} & \frac{\partial \boldsymbol{m}}{\partial \boldsymbol{m}} \\ \frac{\partial\left(-\gamma \boldsymbol{m}(t) + f(\boldsymbol{h}(t), t, \theta)\right)}{\partial \boldsymbol{h}} & \frac{\partial\left(-\gamma \boldsymbol{m}(t) + f(\boldsymbol{h}(t), t, \theta)\right)}{\partial \boldsymbol{m}} \end{bmatrix}$$

i.e.,

$$\frac{d\boldsymbol{a}_{\boldsymbol{h}}}{dt} = \boldsymbol{a}_{\boldsymbol{m}}(t)\frac{\partial f}{\partial \boldsymbol{h}}; \quad \frac{d\boldsymbol{a}_{\boldsymbol{m}}}{dt} = -\boldsymbol{a}_{\boldsymbol{h}}(t) + \gamma \boldsymbol{a}_{\boldsymbol{m}}(t).$$

Theorem 3. The adjoint state $\boldsymbol{a}(t) := \partial\mathcal{L}/\partial\boldsymbol{m}(t)$ for the HBNODE (12) satisfies the following HBODE with the same damping parameter γ as that in (12),

$$\frac{d^2\boldsymbol{a}(t)}{dt^2} - \gamma\frac{d\boldsymbol{a}(t)}{dt} = \boldsymbol{a}(t)\frac{\partial f}{\partial \boldsymbol{h}}(\boldsymbol{h}(t), t, \theta). \tag{15}$$

Remark. Note that we solve the adjoint equation (15) from time $t = T$ to $t = t_0$ in the backward propagation. By letting $\tau = T - t$ and $\boldsymbol{b}(\tau) = \boldsymbol{a}(T - \tau)$, we can rewrite (15) as follows (note that $\frac{d\boldsymbol{a}(t)}{dt} = \frac{d\boldsymbol{a}(T-\tau)}{dt} = -\frac{d\boldsymbol{a}(T-\tau)}{d\tau} = -\frac{d\boldsymbol{b}(\tau)}{d\tau}$),

$$\frac{d^2\boldsymbol{b}(\tau)}{d\tau^2} + \gamma\frac{d\boldsymbol{b}(\tau)}{d\tau} = \boldsymbol{b}(\tau)\frac{\partial f}{\partial \boldsymbol{h}}(\boldsymbol{h}(T - \tau), T - \tau, \theta). \tag{16}$$

The adjoint of HBNODE is also a HBNODE with the same damping parameter.

Theorem 4. The adjoint states $\frac{\partial L}{\partial \boldsymbol{h}_t}$ and $\frac{\partial L}{\partial \boldsymbol{m}_t}$ of the following HBNODE

$$\frac{d\boldsymbol{h}(t)}{dt} = \boldsymbol{m}(t); \quad \frac{d\boldsymbol{m}(t)}{dt} = -\gamma \boldsymbol{m}(t) + f(\boldsymbol{h}(t), t, \theta). \tag{17}$$

satisfy

$$\begin{bmatrix} \frac{\partial \mathcal{L}}{\partial \boldsymbol{h}_t} & \frac{\partial \mathcal{L}}{\partial \boldsymbol{m}_t} \end{bmatrix} = \begin{bmatrix} \frac{\partial \mathcal{L}}{\partial \boldsymbol{h}_T} & \frac{\partial \mathcal{L}}{\partial \boldsymbol{m}_T} \end{bmatrix} \exp\Big\{ - \underbrace{\int_T^t \begin{bmatrix} 0 & \boldsymbol{I} \\ \frac{\partial f}{\partial \boldsymbol{h}} & -\gamma \boldsymbol{I} \end{bmatrix} ds}_{:=\boldsymbol{M}} \Big\}. \tag{18}$$

Proof of Theorem 4.

- Recall for neural ODE $\frac{d\boldsymbol{h}(t)}{dt} = f(\boldsymbol{h}(t), t, \theta)$, we have

$$\frac{\partial L}{\partial \boldsymbol{h}_t} = \frac{\partial L}{\partial \boldsymbol{h}_T}\frac{\partial \boldsymbol{h}_T}{\partial \boldsymbol{h}_t} = \frac{\partial L}{\partial \boldsymbol{h}_T}\exp\Big\{-\int_T^t \frac{\partial f}{\partial \boldsymbol{h}}(\boldsymbol{h}(s), s, \theta)ds\Big\},$$

- Let $\boldsymbol{n}(t) = [\boldsymbol{h}(t)\ \boldsymbol{m}(t)]$ and $\boldsymbol{a}(t) = [\boldsymbol{a}_{\boldsymbol{h}}(t)\ \boldsymbol{a}_{\boldsymbol{m}}(t)]$, then we can rewrite HBNODE as

$$\frac{d}{dt}[\boldsymbol{h}(t),\ \ \boldsymbol{m}(t)] = [\boldsymbol{m}(t),\ -\gamma\boldsymbol{m}(t) + f(\boldsymbol{h}(t), t, \theta)]$$

- Therefore, we have

$$\begin{aligned}\begin{bmatrix}\frac{\partial \mathcal{L}}{\partial \boldsymbol{h}_t} & \frac{\partial \mathcal{L}}{\partial \boldsymbol{m}_t}\end{bmatrix} &= \begin{bmatrix}\frac{\partial \mathcal{L}}{\partial \boldsymbol{h}_T} & \frac{\partial \mathcal{L}}{\partial \boldsymbol{m}_T}\end{bmatrix}\exp\Big\{-\int_T^t \begin{bmatrix}\frac{\partial \boldsymbol{m}}{\partial \boldsymbol{h}} & \frac{\partial \boldsymbol{m}}{\partial \boldsymbol{m}} \\ \frac{\partial(-\gamma\boldsymbol{m}+f)}{\partial \boldsymbol{h}} & \frac{\partial(-\gamma\boldsymbol{m}+f)}{\partial \boldsymbol{m}}\end{bmatrix} ds\Big\} \\ &= \begin{bmatrix}\frac{\partial \mathcal{L}}{\partial \boldsymbol{h}_T} & \frac{\partial \mathcal{L}}{\partial \boldsymbol{m}_T}\end{bmatrix}\exp\Big\{-\underbrace{\int_T^t \begin{bmatrix}0 & \boldsymbol{I} \\ \frac{\partial f}{\partial \boldsymbol{h}} & -\gamma\boldsymbol{I}\end{bmatrix} ds}_{:=\boldsymbol{M}}\Big\}.\end{aligned}$$

Proposition 1. The eigenvalues of the matrix $-\boldsymbol{M}$ can be paired so that the sum of each pair equals $(t - T)\gamma$.

Remark: The eigenvalues cannot all be very small, thus the adjoint will not vanish.

Proof of Proposition 1. Let $\boldsymbol{F} = \frac{1}{t-T}\int_T^t \frac{\partial f}{\partial \boldsymbol{h}}(\boldsymbol{h}(s), s, \theta)ds$ and $\boldsymbol{H} = \frac{1}{t-T}\boldsymbol{M}$, then we have the following equation

$$\boldsymbol{H} = \frac{1}{t-T}\boldsymbol{M} = \begin{bmatrix} 0 & \boldsymbol{I} \\ \boldsymbol{F} & -\gamma \boldsymbol{I} \end{bmatrix}. \tag{19}$$

Then the characteristic polynomial of $\boldsymbol{H}$ satisfies

$$\begin{aligned} ch_{\boldsymbol{H}}(\lambda) &= \det(\lambda \boldsymbol{I} - \boldsymbol{H}) = \det \begin{bmatrix} \lambda \boldsymbol{I} & -\boldsymbol{I} \\ -\boldsymbol{F} & (\lambda+\gamma)\boldsymbol{I} \end{bmatrix} \\ &= det(\lambda(\lambda+\gamma)\boldsymbol{I} - \boldsymbol{F}) = -ch_{\boldsymbol{F}}(\lambda(\lambda+\gamma)). \end{aligned} \tag{20}$$

We can write the characteristic polynomial of $\boldsymbol{F}$ as

$$ch_{\boldsymbol{F}}(\lambda) = \prod_{i=1}^{n}(\lambda - \lambda_{\boldsymbol{F},i}),$$

therefore

$$ch_{\boldsymbol{H}}(\lambda) = -ch_{\boldsymbol{JF}}(\lambda(\lambda+\gamma)) = -\prod_{i=1}^{n}(\lambda(\lambda+\gamma) - \lambda_{\boldsymbol{F},i}). \tag{21}$$

Therefore, the eigenvalues of $\boldsymbol{H}$ appear in n pairs with each pair satisfying the quadratic equation

$$\lambda(\lambda+\gamma) - \lambda_{\boldsymbol{F},i} = 0. \tag{22}$$

By Vieta's formulas, the sum of these pairs are all $-\gamma$. Therefore, the eigenvalues of $\boldsymbol{M}$ comes in n pairs and the sum of each pair is $-(t-T)\gamma$.

Remark on vanishing adjoint states

- By Schur decomposition, there exists an orthogonal matrix $\boldsymbol{Q}$ and an upper triangular matrix $\boldsymbol{U}$, where the diagonals of $\boldsymbol{U}$ are eigenvalues of $\boldsymbol{Q}$ ordered by their real parts, s.t.

$$-\boldsymbol{M} = \boldsymbol{Q}\boldsymbol{U}\boldsymbol{Q}^\top \Longrightarrow \exp\{-\boldsymbol{M}\} = \boldsymbol{Q}\exp\{\boldsymbol{U}\}\boldsymbol{Q}^\top. \tag{23}$$

- Let $\boldsymbol{v}^\top := \begin{bmatrix} \frac{\partial L}{\partial \boldsymbol{h}_T} & \frac{\partial L}{\partial \boldsymbol{m}_T} \end{bmatrix} \boldsymbol{Q}$,

$$\begin{bmatrix} \frac{\partial L}{\partial \boldsymbol{h}_t} & \frac{\partial L}{\partial \boldsymbol{m}_t} \end{bmatrix} = \begin{bmatrix} \frac{\partial L}{\partial \boldsymbol{h}_T} & \frac{\partial L}{\partial \boldsymbol{m}_T} \end{bmatrix} \exp\{-\boldsymbol{M}\} = \begin{bmatrix} \frac{\partial L}{\partial \boldsymbol{h}_T} & \frac{\partial L}{\partial \boldsymbol{m}_T} \end{bmatrix} \boldsymbol{Q}\exp\{\boldsymbol{U}\}\boldsymbol{Q}^\top = \boldsymbol{v}^\top \exp\{\boldsymbol{U}\}\boldsymbol{Q}^\top. \tag{24}$$

- By taking the ℓ_2 norm in (24) and dividing both sides by $\left\| \begin{bmatrix} \frac{\partial L}{\partial \boldsymbol{h}_T} & \frac{\partial L}{\partial \boldsymbol{m}_T} \end{bmatrix} \right\|_2$, we arrive at

$$\frac{\left\| \begin{bmatrix} \frac{\partial L}{\partial \boldsymbol{h}_t} & \frac{\partial L}{\partial \boldsymbol{m}_t} \end{bmatrix} \right\|_2}{\left\| \begin{bmatrix} \frac{\partial L}{\partial \boldsymbol{h}_T} & \frac{\partial L}{\partial \boldsymbol{m}_T} \end{bmatrix} \right\|_2} = \frac{\left\| \boldsymbol{v}^\top \exp\{\boldsymbol{U}\}\boldsymbol{Q}^\top \right\|_2}{\|\boldsymbol{v}^\top \boldsymbol{Q}^\top\|_2} = \frac{\|\boldsymbol{v}^\top \exp\{\boldsymbol{U}\}\|_2}{\|\boldsymbol{v}\|_2} = \|\boldsymbol{e}^\top \exp\{\boldsymbol{U}\}\|_2, \tag{25}$$

i.e., $\left\| \begin{bmatrix} \frac{\partial L}{\partial \boldsymbol{h}_t} & \frac{\partial L}{\partial \boldsymbol{m}_t} \end{bmatrix} \right\|_2 = \left\| \boldsymbol{e}^\top \exp\{\boldsymbol{U}\} \right\|_2 \left\| \begin{bmatrix} \frac{\partial L}{\partial \boldsymbol{h}_T} & \frac{\partial L}{\partial \boldsymbol{m}_T} \end{bmatrix} \right\|_2$ where $\boldsymbol{e} = \boldsymbol{v}/\|\boldsymbol{v}\|_2$.

Remark on vanishing adjoint states

• For a given constant $a > 0$, we can group the upper triangular matrix $\exp\{\boldsymbol{U}\}$ as follows

$$\exp\{\boldsymbol{U}\} := \begin{bmatrix} \exp\{\boldsymbol{U}_L\} & \boldsymbol{P} \\ 0 & \exp\{\boldsymbol{U}_V\} \end{bmatrix}, \tag{26}$$

where the diagonal of $\boldsymbol{U}_L$ ($\boldsymbol{U}_V$) contains eigenvalues of $-\boldsymbol{M}$ that are no less (greater) than $(t - T)a$.

• Then, we have $\|\boldsymbol{e}^\top \exp\{\boldsymbol{U}\}\|_2 \geq \|\boldsymbol{e}_L^\top \exp\{\boldsymbol{U}_L\}\|_2$ where the vector $\boldsymbol{e}_L$ denotes the first m columns of $\boldsymbol{e}$ with m be the number of columns of $\boldsymbol{U}_L$. By choosing $0 \leq \gamma \leq 2a$, for every pair of eigenvalues of $-\boldsymbol{M}$ there is at least one eigenvalue whose real part is no less than $(t - T)a$. Therefore, $\exp\{\boldsymbol{U}_L\}$ decays at a rate at most $(t - T)a$, and the dimension of $\boldsymbol{U}_L$ is at least $n \times n$.