# Training RNNs via Forward Propagation Through Time (FPTT)

Preparation Phase Presentation (to be completed)

July 7, 2023

Yicheng Zhang, master student in Embedded Systems

TU/e EINDHOVEN UNIVERSITY OF TECHNOLOGY

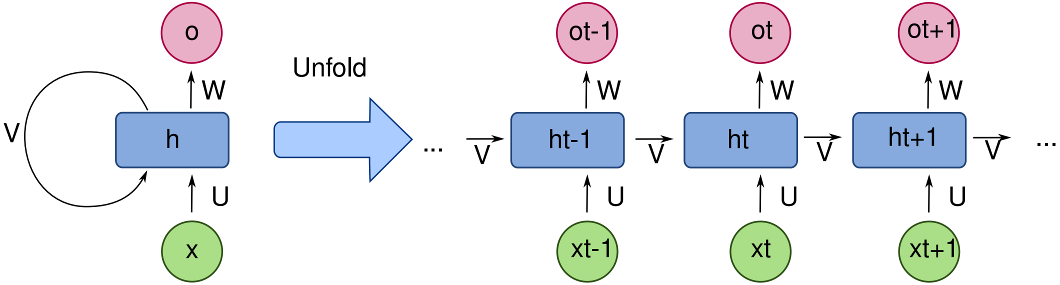



Figure: Recurrent Neural Network and Unrolling

# Recall: RNNs and Applications

RNNs: Recurrent Neural Networks

1. Neurons in the same layer have self-loops so that they can remember
   i.e., the output of the last time step is also the input for now
2. Solve tasks with dependencies distributed over time steps (Sequences)

Typical Sequential Applications:

1. Sequence-to-Sequence (Seq2Seq): At every time steps, the output is collected and used.
   e.g. Language Modelling like PTB
2. Terminal Prediction: Only the output at the final time step is the result
   e.g. Classifications like S-MNIST, CIFAR-10

# Training RNNs by BPTT and Problems

Backpropagation Trough Time (BPTT): Apply Backpropagation (BP) on an unrolled RNN

Example:

$$\mathbf{W}_{hh} \leftarrow \mathbf{W}_{hh} - \eta \frac{\partial L}{\partial \mathbf{W}_{hh}} = \mathbf{W}_{hh} - \eta \cdot \frac{\partial \sum_{t=1}^{T} L^{(t)}}{\partial \mathbf{W}_{hh}} = \mathbf{W}_{hh} - \eta \cdot \sum_{t=1}^{T} \frac{\partial L^{(t)}}{\partial \mathbf{W}_{hh}}$$

$$\frac{\partial L^{(t)}}{\partial \mathbf{W}_{hh}} = \frac{\partial L^{(t)}}{\partial y^{(t)}} \cdot \frac{\partial y^{(t)}}{\partial \mathbf{h}^{(t)}} \cdot \left(\sum_{k=1}^{t} \frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{h}^{(k)}} \cdot \frac{\partial \mathbf{h}^{(k)}}{\partial \mathbf{W}_{hh}}\right) = \frac{\partial L^{(t)}}{\partial y^{(t)}} \cdot \frac{\partial y^{(t)}}{\partial \mathbf{h}^{(t)}} \cdot \left(\sum_{k=1}^{t} \left(\prod_{i=k+1}^{t} \frac{\partial \mathbf{h}^{(i)}}{\partial \mathbf{h}^{(i-1)}}\right) \cdot \frac{\partial \mathbf{h}^{(k)}}{\partial \mathbf{W}_{hh}}\right)$$

Problems:

1. Computation Cost: $\Omega(c(T)T)$
   Any state depends on all states before it
2. Memory Cost: $\Omega(T)$
   Need to save states generated at all time step

# Online Formulation: the first try

To make the update online:

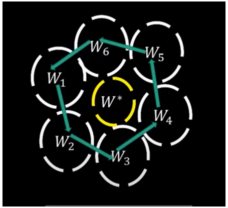
$$\mathbf{W}_{hh}^{(t+1)} \leftarrow \mathbf{W}_{hh}^{(t)} - \eta \cdot \frac{\partial L^{(t)}}{\partial \mathbf{W}_{hh}}$$

Improvements:

1. Save Memory Cost: Weights/Biases are updated at every time step, then the memory space for the state of that time step can be released
2. Save Computation Cost: Since memory space only keeps one state, the production is no done/possible

New Problems: Lacks stability (BPTT unrolling is time invariant); SGD is prone to stray

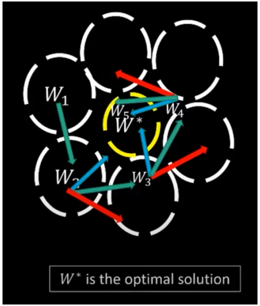
Therefore, an **extra constraint** is necessary.






TU/e

# Intuition of FPTT: Online Formulation + Constraints




$$\mathbf{W}_{hh}^{(t+1)} \leftarrow \mathbf{W}_{hh}^{(t)} - \eta \cdot \frac{\partial (L'^{(t)} + R^{(t)})}{\partial \mathbf{W}_{hh}}$$

$$R^{(t)} = \frac{\alpha}{2} \cdot ||w_{hh} - \bar{\mathbf{W}}_{hh}^{(t)} - \frac{1}{2\alpha} \cdot \frac{\partial L'^{(t-1)}}{\partial \mathbf{W}_{hh}^{(t)}}||^2$$

$$\bar{\mathbf{W}}_{hh}^{(t+1)} \leftarrow \frac{1}{2}(\bar{\mathbf{W}}_{hh}^{(t)} + \mathbf{W}_{hh}^{(t+1)}) - \frac{1}{2\alpha} \cdot \frac{\partial L'^{(t)}}{\partial \mathbf{W}_{hh}^{(t+1)}}$$

$$(\nabla_{l_t}(W_{t+1}) - \nabla_{l_{t-1}}(W_t) + \alpha(W_{t+1} - \overline{W_t}) = 0)$$

$R^{(t)}$ in Cost Function:

Force smaller distance between weights and the running means of weights, to help stability and convergence

# Mathematical foundations

**Proposition 1.** *In the Algorithm 2, suppose, the sequence $W_t$ is bounded and converges to a limit point $W_\infty$. Further assume the loss function $\ell_t$ is smooth and Lipschitz. Let the cumulative loss be $F = \frac{1}{T}\sum_{t=1}^{T} \nabla \ell_t(W_\infty)$ after $T$ iterations* [2]. *It follows that $W_\infty$ is a stationary point of Eq. 3, i.e.,* $\lim_{T\to\infty} \frac{\partial F}{\partial W}(W_\infty) = 0$.

$$\frac{\partial L}{\partial W} = \sum_{i=1}^{N}\sum_{t=1}^{T} \frac{\partial \ell(y_t^i, \hat{y}_i^t)}{\partial W}$$
$$= \frac{1}{NT}\sum_{i=1}^{N}\sum_{t=1}^{T} \frac{\partial \ell(y_t^i, \hat{y}_i^t)}{\partial \hat{y}_i^t} \frac{\partial \hat{y}_i^t}{\partial h_t^i} \sum_{j=1}^{t} \Big( \prod_{s=j}^{t} \frac{\partial h_s^i}{\partial h_{s-1}^i} \Big) \frac{\partial h_{j-1}^i}{\partial W} \tag{3}$$

Summary: FPTT is effectively identical to BPTT

# Generalization

**FPTT-K**: generalized FPTT, split the sequence into coarser grain

- e.g. a sequential application of 784 steps
  FPTT (default): 784 sub-sequences each has length 1
  FPTT-14: 14 sub-sequences each has length 56 (14x56=784)
- a trade-off option between computation cost, update frequency and Memory Storage (see later slide)

**Auxiliary Loss**: enables FPTT for Terminal Predictions ($\beta = \frac{t}{T}$)

$$\ell_t = \beta \ell_t^{CE} + (1-\beta)\ell_t^{Div}$$

$$\ell_t^{CE} = -\sum_{\bar{y} \in \mathcal{Y}} \mathbf{1}_{\bar{y}=y} \log \hat{P}(\bar{y}); \quad \ell_t^{Div} = -\sum_{\bar{y} \in \mathcal{Y}} Q(\bar{y}) \log \hat{P}(\bar{y})$$

Defined by intuition: the prediction first approaches the prediction made in last epoch to keep stable, then gets close to the real label

# Comparisons-1

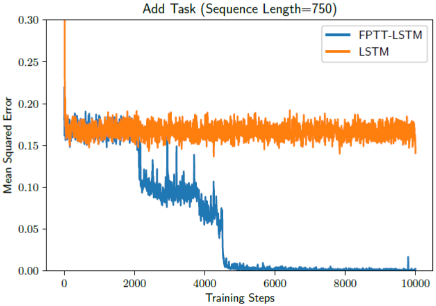
Part 1: Gradient Update, Parameter Update and Memory Storage

| Algorithm | Gradient Update | parameter Update | Memory Storage |
|---|---|---|---|
| BPTT | $\Omega(c(T)T)$ | $\Omega(1)$ | $\Omega(T)$ |
| RTRL | $\Omega(c(T)T^2)$ | $\Omega(T)$ | $\Omega(T)$ |
| e-prop / OSTL | $\Omega(c(1)T)$ | $\Omega(T)$ | $\Omega(1)$ |
| FPTT | $\Omega(c(1)T)$ | $\Omega(T)$ | $\Omega(1)$ |
| FPTT-K | $\Omega(c(K)T)$ | $\Omega(K)$ | $\Omega(T/K)$ |

# Comparisons-2

Part 2: Better Generalization on tasks by FPTT

1. enabling model to learn longer sequence than BPTT can do
2. higher accuracy than BPTT can achieve




| Algorithm | S-MNIST | PS-MNIST | CIFAR-10 |
|---|---|---|---|
| BPTT | 97.71% | 88.91% | 60.11% |
| FPTT | 98.67% | 94.75% | 71.03% |

Table: Accuracy Comparison



TU/e

# To be done

1. introduction of E-Propagation (e-prop)
2. FPTT on spiking neurons
3. results of own experiments
4. proposal of FPTT hardware implementation (diagram)