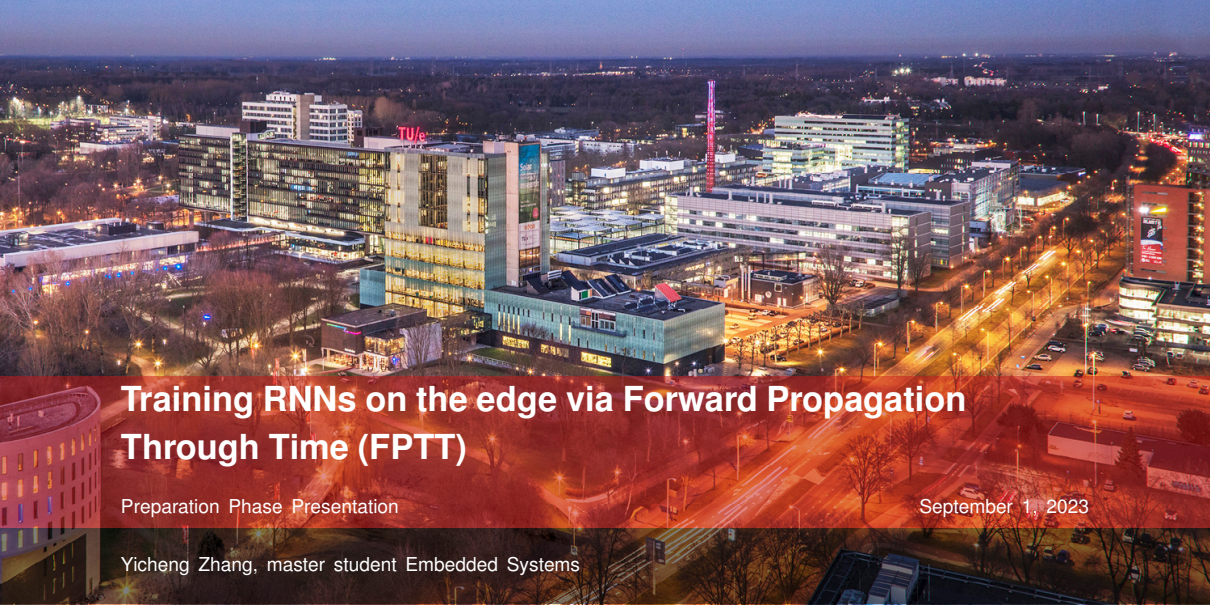


Training RNNs on the edge via Forward Propagation Through Time (FPTT)

Preparation Phase Presentation

September 1, 2023

Yicheng Zhang, master student Embedded Systems

TU/e EINDHOVEN UNIVERSITY OF TECHNOLOGY

Edge-AI: inference and training on the edge

Edge-AI is the implementation of artificial intelligence in an edge-computing environment. This allows computations to be done close to where data is actually collected, rather than at a centralized cloud server.

Advantages of **inference** on the edge:

- reduced latency in response: no need to communicate with the cloud
- privacy: no data sharing with the external
- lower cost: save the power in data transfer and efficient processing on edge devices
- ...

Training is as necessary as inference on the edge.

Training on the edge: What, Why and How

What?

1. lightweight partial re-training of the pre-trained model on edge devices (focus)
   - for customization and personalization
   - continuous adaption to new features in data
2. training from scratch

Why not still GPU?

- accessing the GPU: issues of huge delay, privacy and security...
- computations on the GPU: training of small batch sizes leads to low utilization on GPU

How/What needs to be solved?

- training algorithm is usually demanding in computation and memory, while edge devices have limited resources

Training RNNs on the edge by FPTT: Prospects and Concerns

Forward Propagation Through Time (FPTT) is **a novel training algorithm** for the Recurrent Neural Network (RNN) based on the classic Backpropagation Through Time (BPTT).

Compared to BPTT, FPTT has the following prospects

- has the lower computation cost
- requires less memory
- leads to better performance

but still has a few concerns

- computation cost is reduced but still heavy
- more frequent memory access is required which is power-consuming

Overall, it is a promising algorithm suitable for on-edge training.

Goal: to develop a system-level solution for training on the edge with FPTT

Related Work:

Edge TPU + TensorFlow Lite (Google, 2018 onwards)

- ASIC chip designed to run machine learning (ML) models for edge computing
- is capable of accelerating forward-pass operations with TensorFLow Lite
- is possible to perform lightweight transfer learning (but only for the last classifier layer)
- commercial, with board products

TinyOL (IJCNN 2021) link

- TinyOL (TinyML with Online-Learning), which enables incremental on-device training on streaming data.
- based on the concept of online learning and is suitable for constrained IoT devices
- autoencoder NN running on an Arduino Nano 33 BLE Sense board with Cortex™-M4 CPU

Related Work: On-Device Training Under 256KB Memory (NeurIPS'22)

Software solution link

- Quantization-Aware Scaling
  calibrate the gradient scales and stabilize 8-bit quantized training
- Sparse Update
  skip the gradient computation of less important layers $\rightarrow$ save memory
- Tiny Training Engine
  prunes the backward computation graph to support sparse updates
  offload the runtime auto-differentiation to compile time

Platform: Cortex M7 microcontroller STM32F746 (320KB SRAM/1MB Flash)

Results

- training memory from 303MB (PyTorch) to 149KB with the same transfer learning accuracy, leading to 2300x reduction
  (with MobilenetV2-w0.35, batch size 1 and resolution 128x128,
  the dataset is a tiny ML application VWW (Visual Wake Words))

Section 2: FPTT theory

- RNN and applications
- Training by BPTT and Problems
- BPTT online formualtion
- FPTT: regularization penalty
- algorithm generalization
- Comparison

FPTT theory: RNNs and Applications

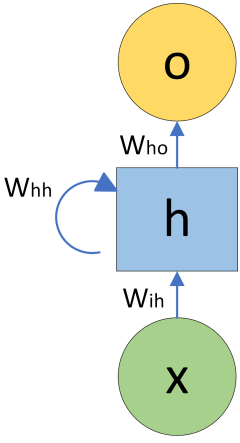
Recurrent Neural Network Speciality

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

Typical Sequential Applications

1. Sequence Modeling (Seq2Seq): At every time step, 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 to be considered
   e.g. Classification tasks like S-MNIST, pixel CIFAR-10

FPTT theory: Training RNNs by BPTT and Problems

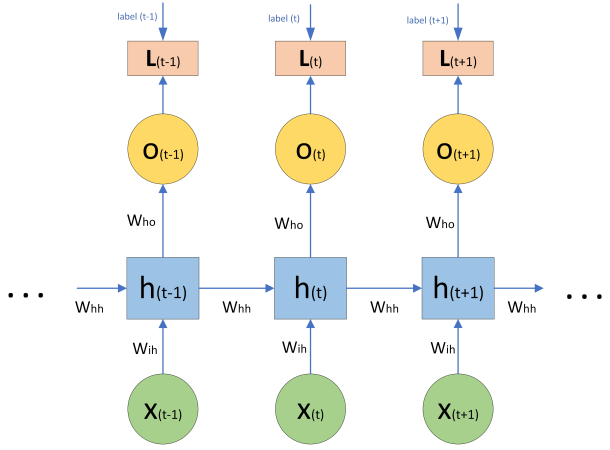


$$h_t = \sigma_h(x_t \cdot W_{ih} + h_{t-1} \cdot W_{hh}) \quad (1)$$

$$o_t = \sigma_o(h_t \cdot W_{ho}) \quad (2)$$

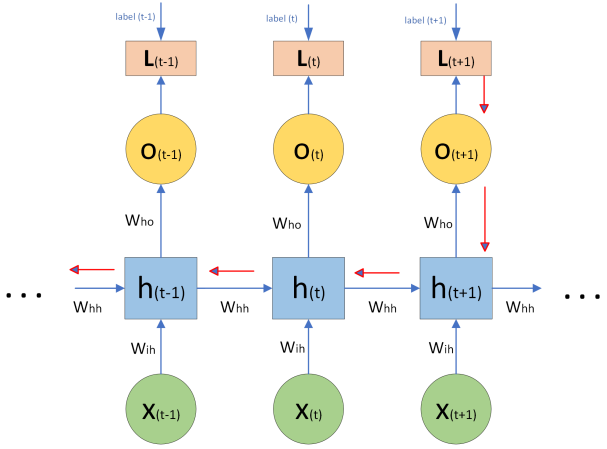
FPTT theory: Training RNNs by BPTT and Problems

After forward pass for T time steps, we have a sequence of network states of length T.



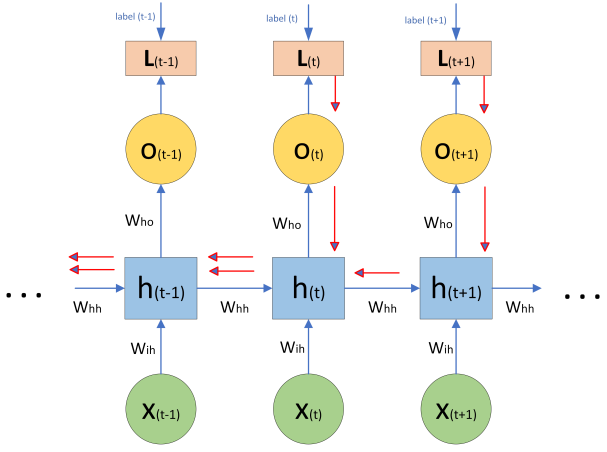
TU/e

FPTT theory: Training RNNs by BPTT and Problems



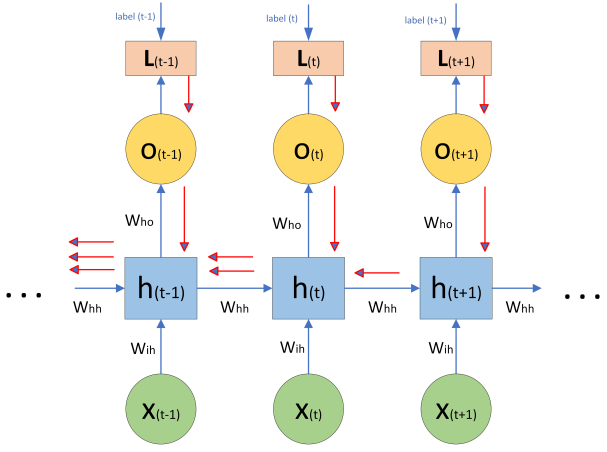
TU/e

FPTT theory: Training RNNs by BPTT and Problems



TU/e

FPTT theory: Training RNNs by BPTT and Problems



TU/e

FPTT theory: Training RNNs by BPTT and Problems

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

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

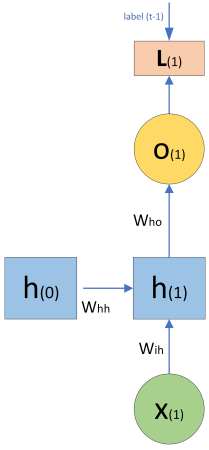
$$\frac{\partial L^{(t)}}{\partial \mathbf{W}_{hh}} = \frac{\partial L^{(t)}}{\partial o^{(t)}} \cdot \frac{\partial o^{(t)}}{\partial \mathbf{h}^{(t)}} \cdot \left(\sum_{k=1}^{t} \frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{h}^{(k)}} \cdot \frac{\partial \mathbf{h}^{(k)}}{\partial \mathbf{W}_{hh}}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

Problems:

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

FPTT theory: BPTT Online Formulation: the first try

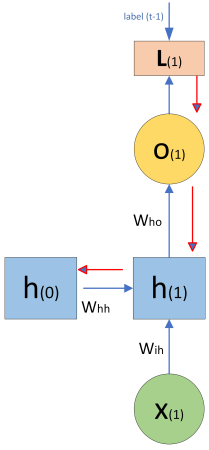
Interleave forward and backward passes ($t = 1$ forward)



TU/e

FPTT theory: BPTT Online Formulation: the first try

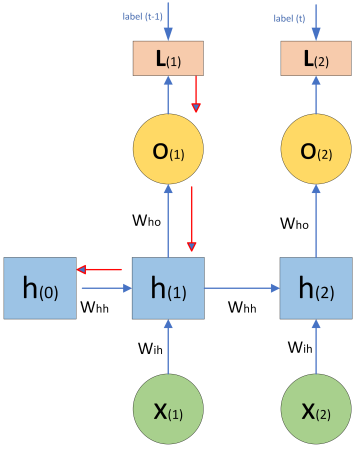
Interleave forward and backward passes ($t = 1$ backward)



TU/e

FPTT theory: BPTT Online Formulation: the first try

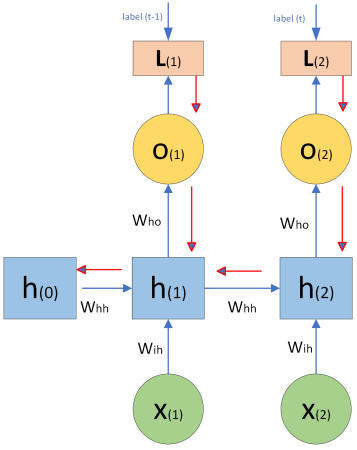
Interleave forward and backward passes ($t = 2$ forward)



TU/e

FPTT theory: BPTT Online Formulation: the first try

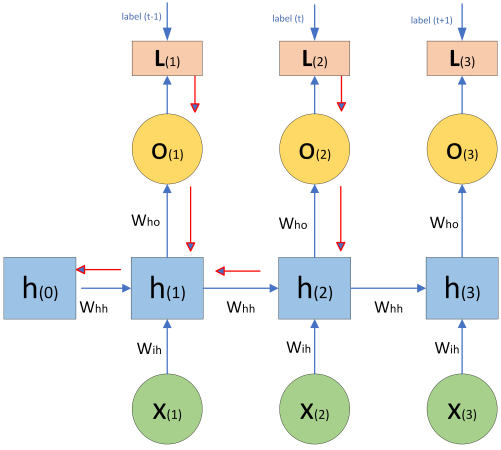
Interleave forward and backward passes ($t = 2$ backward)



TU/e

FPTT theory: BPTT Online Formulation: the first try

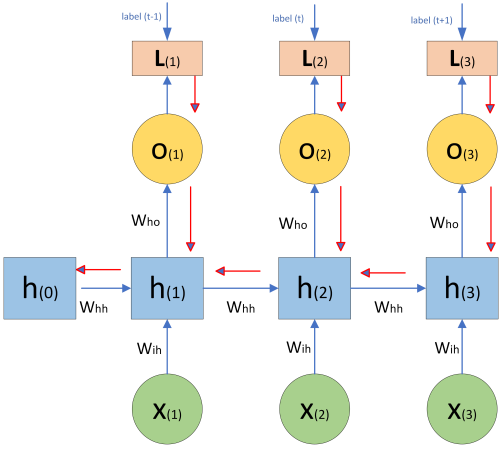
Interleave forward and backward passes ($t = 3$ forward)



TU/e

FPTT theory: BPTT Online Formulation: the first try

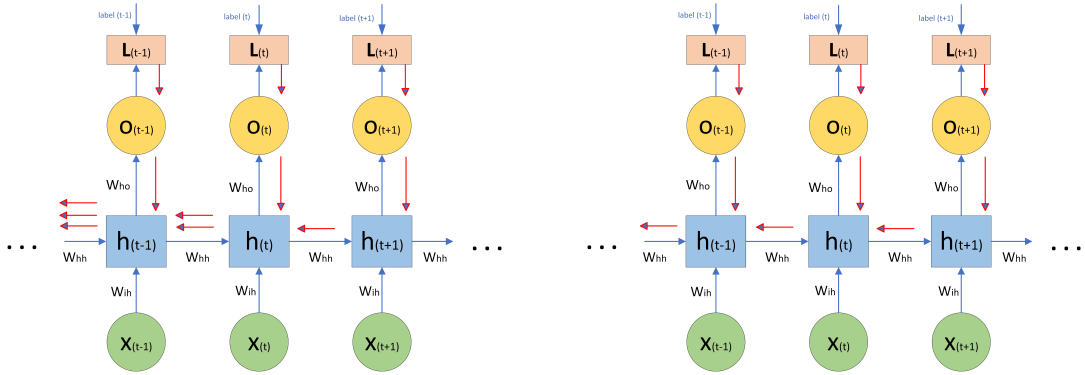
Interleave forward and backward passes ($t = 3$ backward)



TU/e

FPTT theory: BPTT Online Formulation: the first try

BPTT vs. BPTT online formulation



TU/e

FPTT theory: BPTT Online Formulation: the first try

Update after every time step:

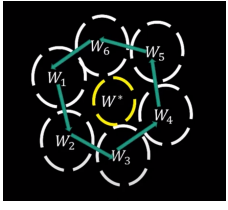
$$\mathbf{W}_{hh}^{(t+1)} \leftarrow \mathbf{W}_{hh}^{(t)} - \eta \cdot \frac{\partial L^{(t)}}{\partial \mathbf{W}_{hh}}$$

What has changed?

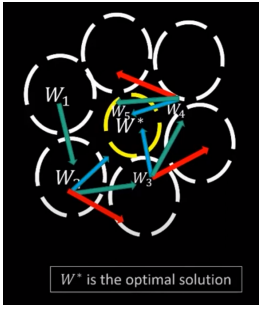
- Save Computation Cost: backpropagate one time step for each $L(t)$
- Save Memory Cost: accessed network states can be released
- frequent update (memory access)

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

Therefore, an **extra constraint** is proposed



FPTT theory: Intuition (BPTT Online Formulation + the Constraint)



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

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

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

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

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

FPTT theory: Mathematical foundations

Proposition 1. *In the Algorithm 2, suppose, the sequence* W_t *is bounded and converges to a limit point* W_∞*. Further assume the loss function* ℓ_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_∞ *is a stationary point of Eq. 3, i.e.,* $\lim_{T\to\infty} \frac{\partial F}{\partial W}(W_\infty) = 0$.

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

Summary: By FPTT, W_t can converge a limited point W_∞, which is a stationary point that BPTT can find

FPTT theory: Algorithm 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

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

$$l_t^{CE} = -\sum_{\overline{y} \in y} \mathbf{1}_{\overline{y}=y} log\hat{P}(\overline{y}); \quad l_t^{Div} = -\sum_{\overline{y} \in y} Q(\overline{y}) log\hat{P}(\overline{y}); \quad \beta = \frac{t}{T}$$

Defined by intuition: the prediction first approaches the prediction made in the last epoch to

FPTT theory: Comparisons-1

Table: Computational cost for gradient, parameter update & memory storage overhead[2]

Algorithm	Gradient Updates	Parameter Updates	Memory Storage
BPTT	$\Omega(c(T)T)$[1]	$\Omega(1)$	$\Omega(T)$
FPTT	$\Omega(c(1)T)$	$\Omega(T)$	$\Omega(1)$
FPTT-K	$\Omega(c(K)T)$	$\Omega(K)$	$\Omega(T/K)$
E-Prop	$\Omega(c(1)T)$	$\Omega(T)$	$\Omega(1)$

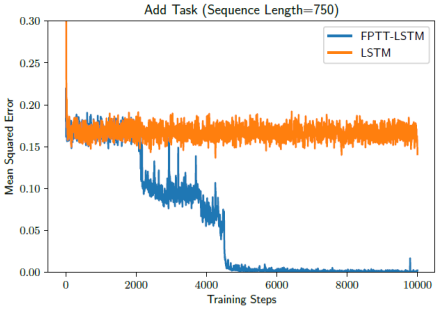
[1] on the sequence-to-sequence task

[2] reference: spiking FPTT

FPTT theory: 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 on a single LSTM network

Section 2: Summary

Forward Propagation Through Time (Time)

- BPTT online formulation
  - saves memory and computation
  - increases update frequency
- regularization penalty $R(t)$
  - for stabilization

Section 3: FPTT Exploration towards hardware

- the simplified network configuration
- Effect of $R(t)$
- deterministic backward pass
- FPTT routine

Purpose:
Make a bridge between Pytorch and hardware,
i.e. a deterministic program can be evaluated in terms of workload

Explorations: the simplified network configuration

Table: Configuration: The default vs The simplified

Configuration	Network Structure	Network Size	Optimizer	Dropout/Clips gradient
the Default	Input-LSTM-FC-FC	1-128-256-10	Adam	with
the Simplified	Input-LSTM-FC	1-128-10	SGD [1]	without

[1] With Momentum

Table: Accuracy: The default vs The simplified [1]

Configuration	S-MNIST	PS-MNIST	CIFAR-10
the default	98.67%	94.75%	71.03%
the simplified [2]	98.44%	94.74%	60.04%
accuracy loss	$<$1%	$<$1%	$\approx$10%

[1] K used here for both versions is 10.

[2] SGD must be with Momentum, otherwise the accuracy loss can be more than 50%

Explorations: the effect of $R(t)$

To figure out whether $R(t)$ is necessary for every K on every data set, so that to

- further understand the algorithm
- determine $R(t)$ to be optional/mandatory data path in hardware

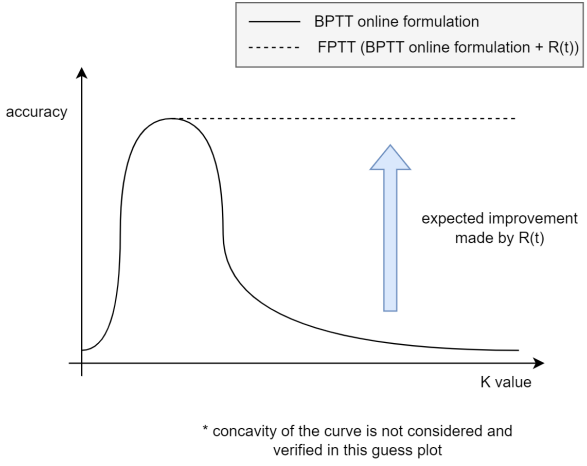


Figure: Hypothesis plot of K vs accuracy

Explorations: the effect of $R(t)$

Table: Actual accuracy[1] achieved at epoch 100 by eight K values on S-MNIST task (784 steps)

Loss Function	K=1	K=2	K=4	K=7	K=14	K=98	K=392	K=784
clf[2]	11.35%	42.90%	39.70%	96.62%	98.28%	96.39%	90.89%	83.91%
clf[2]+ oracle	11.35%	97.46%	90.77%	97.69%	98.47%	96.63%	90.65%	80.14%
clf[2]+ oracle + R[3]	11.35%	84.93%	95.39%	97.52%	98.2%	95.99%	91.92%	83.40%

[1] learning rate is 0.01, and it decays at epoch [50, 75, 90] by 0.1

[2] classification loss

[3] regularization penalty $R(t)$

$R(t)$ can be skipped sometimes, so it is better to make it an option in hardware.

Explorations: the backward pass of LSTM

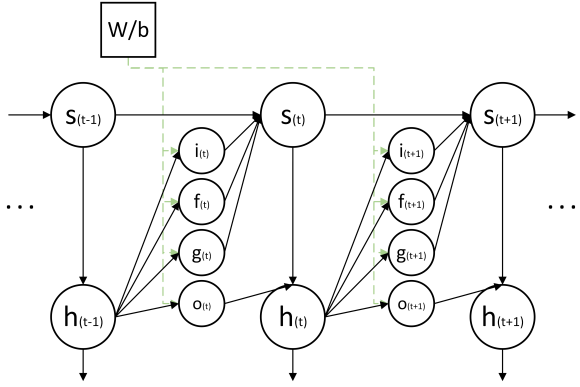


Figure: LSTM unrolling

Explorations: core computations of FPTT

1. Evaluate 3 to get the gradient of entire loss function w.r.t. parameters;

$$\frac{\partial L(t)}{\partial W} = \nabla_{l_t}(W_t) - \lambda_t + \alpha(W_t - \overline{W}_t) \tag{3}$$

2. update W_{t+1} by gradient descent;

$$W_{t+1} = W_t - \eta \frac{\partial L(t)}{\partial W} \tag{4}$$

3. according to 5, update the λ by new weights W_{t+1} and current running mean $\overline{W}_t$;

$$\lambda_{t+1} = \lambda_t - \alpha(W_{t+1} - \overline{W}_t) \tag{5}$$

4. according to 6, update the $\overline{W}$ by λ_{t+1}, W_{t+1} and current $\overline{W}_t$.

$$\overline{W}_{t+1} = \frac{1}{2}(\overline{W}_t + W_{t+1}) - \frac{1}{2\alpha}\lambda_{t+1} \tag{6}$$

Explorations: the outcome – a baseline program

Table: The result of self-design FPTT training script on MNIST task

The way of processing	self-designed version	Pytorch version
streaming by rows	94%	98%
streaming by pixels	78%	88%

Although can be improved further, the program is eligible to be analyzed for hardware architecture

Section 4: Hardware Proposal

- direct mapping of baseline program to hardware system
- system analysis: memory, workload and time
- design choice of the system
- FPTT logic design: subsystem and PE

Hardware Proposal: direct mapping of the baseline program

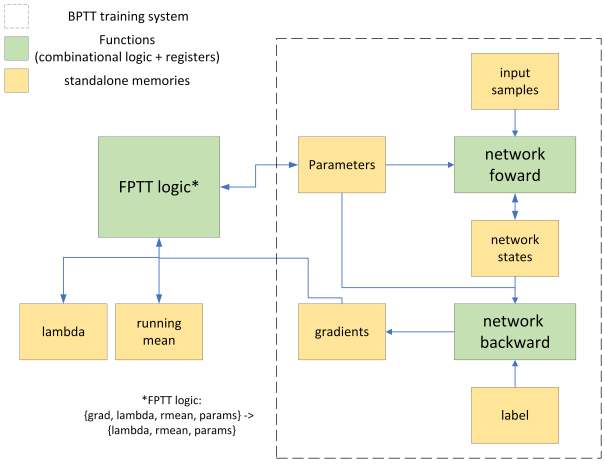


Figure: Dataflow of the system

TU/e

Hardware Proposal: analysis of memory requirement

Table: Estimated memory requirement of the system

Name		Size	Note
Memory	Parameters	265 KB	((128*1+1+128*128+128)*4+(10*128+10))*4B
	Lambda	265 KB	identical to the above
	Running Mean	265 KB	identical to the above
	Gradients	265 KB	identical to the above
	Input Sample [1]	400 B	28*100*4 B
	Label[1]	440 B	(100 + 10)*4B
	Network States[1]	8600 KB	28*(128*6 + 10*2)*100 *4B
Register[2]	Network Forward[1]	200 KB	128*4*100*4B
	Network Backward[1]	500 KB	(128*10 + 10*1)*100*4B
Total		$\approx$ **10.2MB**	$\approx$ 1.2 MB/3 MB if batch size is 1/10

[1] depends on the batch size

[2] tentative values here, will depend on pipeline design

Hardware Proposal: analysis of workload and latency

Table: Estimated operation counts required by each function block

Function Block	MUL/DIV	ADD/SUB	non-linear activation
FPTT logic	5.7 G	9.1 G	N/A
Network forward	3185.2 G	3167.4 G	3 G
Network backward	5609.4 G	5470.9 G	12 G

Table: Execution time of each function block of Python program

Function Block	Execution Time (sec)	percentage
FPTT logic	0.000367	0.11%
Network forward	0.138306	39.73%
Network backward	0.209401	60.16%

Hardware Proposal: design choice

General purpose architecture

- pros
  - flexibility, various NN models can be easily deployed on it
  - can showcase FPTT on multiple NN models
- cons
  - the overhead of ISA, like control path
  - relatively low efficiency in parallel operations

ASIC, specialized accelerator

- pros
  - efficient, no overhead of ISA
  - can be optimized extremely towards the target application
- cons
  - fixed with one application
  - complexity of the logic design of NN backward pass

Hardware Proposal: tentative FPTT logic design of subsystem

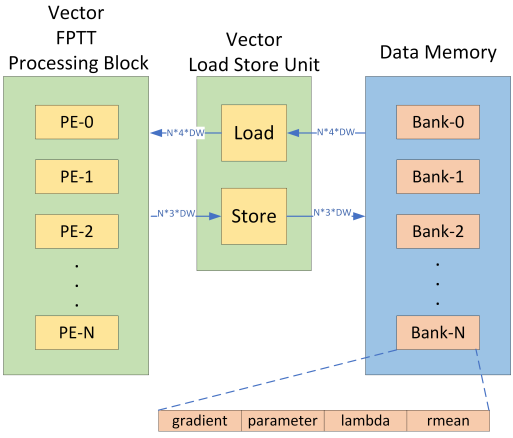


Figure: Tentative FPTT subsystem

TU/e

Hardware Proposal: tentative FPTT logic design of PE

Table: baseline scheduling of FPTT logic on hardware

cycle	operations can be done in parallel	
	op1	op2
1	c0_0 = grad – lbd	c0_1 = param – rmean
2	c1_0 = lr * c0_0	c1_1 = lr_a * c0_1
3	(P) c2_0 = parameter – c1_0 – c1_1	
4	c3_0 = c2_0 – rmean	c3_1 = rmean + c2_0
5	c4_0 = alpha * c3_0	c4_1 = 0.5 * c3_1
6	(L) c5_0 = lbd – c4_0	
7		c6_1 = ita * c5_0
8		(R) c7_1 = c4_1 – c6_1

Conclusion

Work has been done so far:

1. Specified topic: lightweight partial re-training on edge devices
2. Expected outcome: system-level solution based on FPTT
3. Preparations:
   - FPTT theory understanding
   - FPTT exploration towards hardware
   - hardware proposal

Next Step and Plan

Table: Timeline of next steps

Time/Weeks	Task	Deliverables
2	in-depth study of potential general-purpose architectures	document
4	Logic design of the scalable system	RTL files
2	Design space exploration	document
3	Physical design of the system	netlist
2	Performance evaluation and comparison	document
3	Final thesis writing and presentation preparation	report and slides