Mathematics Foundations for Computer Science

RNN's BACKPROPAGATION THROUGH TIME

Lecturers: Dr. Nguyen An Khuong
Presented By Le Thi Hong Cuc

AGENDA

1. INTRODUCTION: FROM TRADITIONAL NNS TO RNNS & BPTT
2. MATHEMATICAL FORMULATION
3. CONCRETE EXAMPLE
4. TRAINING CHALLENGES & SOLUTIONS
5. Q&A

Motivation: Why RNNs and BPTT?

- Traditional Neural Networks handle fixed-length, i.i.d. inputs → struggles with sequential data
- Many tasks involve sequences (text, speech, time series) with temporal dependencies
- RNNs process sequences by maintaining hidden states evolving through time
- Training RNNs requires special gradient computations → Backpropagation Through Time

Recurrent Neural Networks (RNNs) Basics

- Hidden state h_t encodes historical context up to time step t
- General RNN Update Rule:

$$\mathbf{h}_t = f(\mathbf{x}_t, \mathbf{h}_{t-1}, \mathbf{w}_h)$$

- For a vanilla RNN (simple):

$$\mathrm{h_t} = \phi(\mathrm{W}_{\chi\mathrm{h}}\mathrm{x_t} + \mathrm{W_{hh}}\mathrm{h_{t-1}} + \mathrm{b_h})$$

- Weights shared across all time steps → parameter sharing
- Processing sequences of varying length

Forward Pass in RNN

- At each time step t:
  - Compute hidden state: $\mathbf{h}_t = \phi(\mathbf{x}_t \mathbf{W}_{xh} + \mathbf{h}_{t-1} \mathbf{W}_{hh} + \mathbf{b}_h)$
  - Compute output: $\mathbf{o}_t = \mathbf{h}_t \mathbf{W}_{hq} + \mathbf{b}_q$
- Here, ϕ is a nonlinearity (e.g., tanh),

  $\mathbf{W}_{xh}$: input-to-hidden weights, $\mathbf{W}_{hh}$: hidden-to-hidden (recurrent) weights.
- Sequence Unrolling:

  The RNN is “unrolled” through all T time steps, so each output depends on current input and all previous hidden states.

Loss Function (Mean Squared Error) in RNNs:

- If targets y_t are given for each time step, the total loss (over sequence length T) is:

$$L = \frac{1}{T} \sum_{t=1}^{T} l(\mathbf{y}_t, o_t).$$

- This loss quantifies the discrepancy between model predictions and actual targets across all time steps.
- For each sequence, forward pass computes all hidden states and outputs, then loss is summed or averaged over sequence.

1. Introduction: From Traditional NNs to RNNs & BPTT

The Problem with Time Dependency and Need for BPTT

- **Time Dependency in RNNs:**
  The hidden state at each step t (h_t) depends recursively on the previous hidden state $(\mathrm{h_{t-1}})$, which itself depends on earlier states, and so on.
  This makes RNNs fundamentally dependent on temporal context.
- **Why Standard Backpropagation Fails:**
  - Unlike feedforward networks, RNNs require learning from dependencies across time.
    Standard backpropagation cannot account for how earlier inputs affect later outputs, because it only updates weights based on a single forward pass without unrolling time.

What is Backpropagation Through Time (BPTT)?

- BPTT = backpropagation applied to the unrolled RNN computational graph over T time steps.
- The recurrent RNN is expanded (“unrolled”) into a deep feedforward network:
  - Each “layer” of this network corresponds to one time step ($t = 1, 2, \ldots, T$).
  - All these layers share the same parameters (weights).
- Gradients with respect to shared weights accumulate contributions from every time step:

$$\frac{\partial L}{\partial W} = \sum_{t=1}^{T} \frac{\partial L}{\partial W\,(t)}$$

$$\text{or, equivalently,}$$

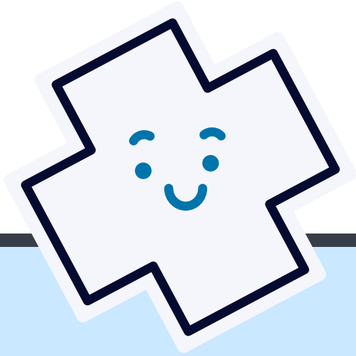
$$\frac{\partial L}{\partial W} = \frac{\partial L}{\partial W^{(1)}} + \frac{\partial L}{\partial W^{(2)}} + \cdots + \frac{\partial L}{\partial W^{(T)}}$$

Applications of RNNs & BPTT

- Natural Language Processing:
  - Language modeling & text generation
  - Machine translation
  - Sentiment analysis, NER, dialogue systems
- Speech Processing:
  - Speech recognition & synthesis
- Time Series Analysis:
  - Forecasting & anomaly detection
- Computer Vision:
  - Image & video captioning
  - Human action recognition
- Other Domains:
  - Music generation, medical predictions, drug design, robot control

How Computer Science Problems Are Solved

- BPTT is the main algorithm for training Recurrent Neural Networks (RNNs) on sequential data.
- Core idea: “Unroll” the recurrent network through all time steps, creating a deep feedforward graph to allow gradients to flow back across the entire sequence.
- Training steps:
  - Forward pass: Compute hidden states & outputs for each time step.
  - Error calculation in BPTT: Sum (or average) the errors between outputs and targets at all time steps.
  - Backward pass: Accumulate gradients from all time steps for shared weights.



Forward Pass in BPTT

- Data is processed sequentially, one time step at a time.
- At each time step t:
  - Hidden state update:

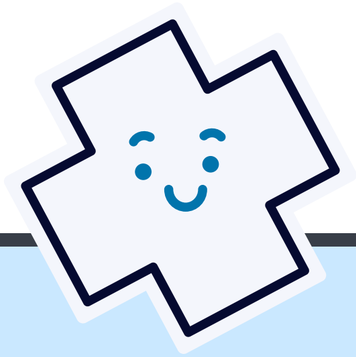
$$\mathbf{h}_t = \phi(\mathbf{x}_t \mathbf{W}_{xh} + \mathbf{h}_{t-1} \mathbf{W}_{hh} + \mathbf{b}_h)$$

- where:
  - x_t: input vector at time step t
  - W_{xh}: input-to-hidden weight matrix
  - h_{t-1}: hidden state from the previous time step (t−1t−1)
  - W_{hh}: hidden-to-hidden (recurrent) weight matrix
  - b_h: bias vector for the hidden state
  - $\phi(\cdot)$: activation function
  - h_t: hidden state at time step t
- Output Computation:

$$\mathbf{o}_t = \mathbf{h}_t \mathbf{W}_{hq} + \mathbf{b}_q$$

- where:
  - $\mathrm{W_{hq}}$: hidden-to-output weight matrix
  - b_q: bias vector for the output
  - o_t: output at time step t
- This process continues until the entire sequence is processed.

2. Mathematical Formulation



ERROR CALCULATION AND STARTING THE BACKWARD PASS

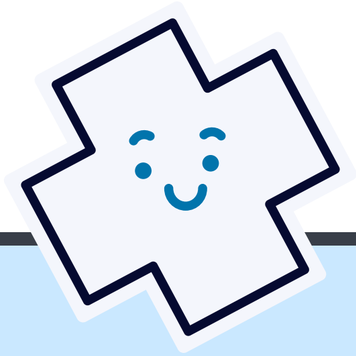
After the forward pass, the model's output is compared to the target at each time step.
Total sequence loss:

$$L = \frac{1}{T} \sum_{t=1}^{T} l(\mathbf{y}_t, o_t).$$

l: loss function per time step

- The loss is used to compute gradients for all model parameters (weights, biases).
- Intermediate activations and hidden states are cached to make the backward pass efficient and to avoid redundant computations.

2. Mathematical Formulation



BACKWARD PASS

- BPTT applies the chain rule to propagate gradients from the end of the sequence back to the start.
- Each parameter's gradient aggregates its effects over all time steps:
  - For example, for $\mathbf{W}_{xh}$:

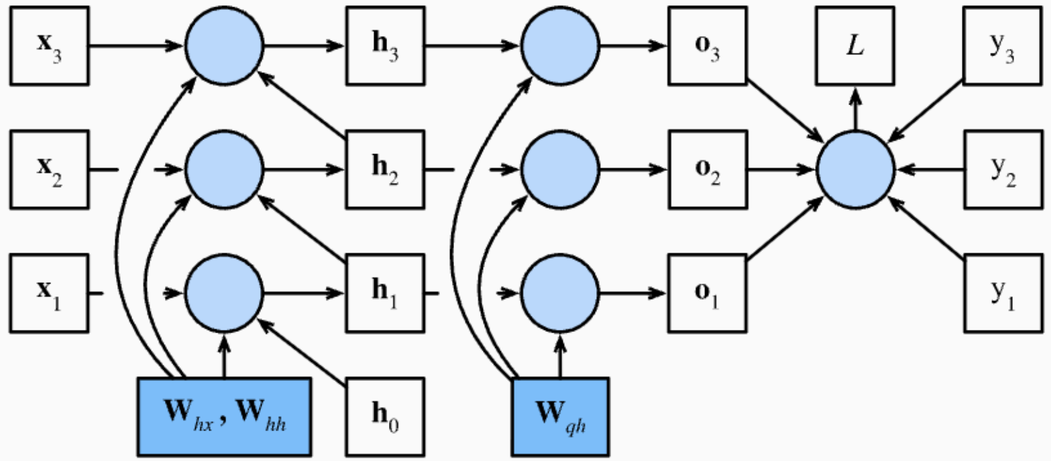
$$\frac{\partial L}{\partial \mathbf{W}_{xh}} = \sum_{t=1}^{T} \frac{\partial L}{\partial \mathbf{h}_t} \frac{\partial \mathbf{h}_t}{\partial \mathbf{W}_{xh}}$$

  - This allows parameters to learn temporal dependencies.
- Problems for long sequences:
  - Vanishing gradients: gradients become too small to update weights for long-term dependencies.
  - Exploding gradients: gradients grow too large, causing instability—commonly handled by gradient clipping.
  - Advanced RNN variants (LSTM, GRU) were developed to address these issues.

2. Mathematical Formulation

COMPUTATIONAL GRAPH AND DEPENDENCIES

- **Hidden state $\mathbf{h}_t$** depends on
  - $\mathbf{x}_t$, $\mathbf{h}_{t-1}$
  - Weights: $\mathbf{W}_{hx}$, $\mathbf{W}_{hh}$
- **Output $\mathbf{o}_t$** depends on
  - $\mathbf{h}_t$
  - Weight: $\mathbf{W}_{qh}$
- **Loss L** aggregates all $l(\mathbf{o}_t, y_t)$



2. Mathematical Formulation

GRADIENT CALCULATION: OUTPUT LAYER

- **Gradient of loss w.r.t. output at time t:**

$$\frac{\partial L}{\partial \mathbf{o}_t} = \frac{1}{T} \frac{\partial l(\mathbf{o}_t, y_t)}{\partial \mathbf{o}_t}$$

- **Gradient w.r.t. $\mathbf{W}_{qh}$:**

$$\frac{\partial L}{\partial \mathbf{W}_{qh}} = \sum_{t=1}^{T} \frac{\partial L}{\partial \mathbf{o}_t} \mathbf{h}_t^\top$$

2. Mathematical Formulation

GRADIENT CALCULATION: HIDDEN STATES

- **Final time step ($t = T$):**

$$\frac{\partial L}{\partial \mathbf{h}_T} = \mathbf{W}_{qh}^\top \frac{\partial L}{\partial \mathbf{o}_T}$$

- **Intermediate ($t < T$):**

$$\frac{\partial L}{\partial \mathbf{h}_t} = \mathbf{W}_{hh}^\top \frac{\partial L}{\partial \mathbf{h}_{t+1}} + \mathbf{W}_{qh}^\top \frac{\partial L}{\partial \mathbf{o}_t}$$

- *→ Hidden states receive gradients both from output at time t and future hidden states!*

2. Mathematical Formulation

FULL RECURRENT EXPANSION & NUMERICAL ISSUES

- **Expanding recursively:**

$$\frac{\partial L}{\partial \mathbf{h}_t} = \sum_{i=t}^{T} (\mathbf{W}_{hh}^\top)^{T-i} \mathbf{W}_{qh}^\top \frac{\partial L}{\partial \mathbf{o}_{T+t-i}}$$

- **Problem**: Repeated matrix multiplication can cause
  - **Vanishing gradients:** eigenvalues < 1
  - **Exploding gradients:** eigenvalues > 1

2. Mathematical Formulation

GRADIENTS W.R.T. INPUT-TO-HIDDEN AND HIDDEN-TO-HIDDEN WEIGHTS

- For $\mathbf{W}_{hx}$:

$$\frac{\partial L}{\partial \mathbf{W}_{hx}} = \sum_{t=1}^{T} \frac{\partial L}{\partial \mathbf{h}_t} \mathbf{x}_t^\top$$

- For $\mathbf{W}_{hh}$:

$$\frac{\partial L}{\partial \mathbf{W}_{hh}} = \sum_{t=1}^{T} \frac{\partial L}{\partial \mathbf{h}_t} \mathbf{h}_{t-1}^\top$$

- **Reuse $\frac{\partial L}{\partial \mathbf{h}_t}$ in both!**

3. Concrete Example

Detail computations in step-by-step with a concrete step-by-step example for a simple vanilla RNN over a short sequence

Problem Statement & Setup

Step-by-Step Example — Simple Vanilla RNN

- Sequence length $T = 3$ (3 time steps)
- Input dimension = 1 (scalar per step), hidden dimension = 1 (single neuron for simplicity), output dimension = 1
- Inputs: $x_1 = 0.5$, $x_2 = 0.1$, $x_3 = 0.4$
- Targets: $y_1 = 0.4$, $y_2 = 0.2$, $y_3 = 0.1$
- Activation functions: hidden state $h_t = \tanh(z_t)$, where $z_t = W_{hh}h_{t-1} + W_{xh}x_t$; output $o_t = W_{hy}h_t$ (linear output)
- Loss: Mean Squared Error $L = \frac{1}{2}\sum_{t=1}^{T}(o_t - y_t)^2$
- Initial hidden state $h_0 = 0$

Assume initial weights:

- $W_{xh} = 0.6$
- $W_{hh} = 0.9$
- $W_{hy} = 0.8$

3. Concrete Example

Step 1: Forward Pass (Unroll through time)

We compute the hidden states h_t and outputs o_t for each time step.

- $t = 1$:

$$z_1 = W_{hh}h_0 + W_{xh}x_1 = 0.9 \times 0 + 0.6 \times 0.5 = 0.3$$

$$h_1 = \tanh(0.3) \approx 0.2913$$

$$o_1 = W_{hy}h_1 = 0.8 \times 0.2913 = 0.2330$$

- $t = 2$:

$$z_2 = W_{hh}h_1 + W_{xh}x_2 = 0.9 \times 0.2913 + 0.6 \times 0.1 = 0.2622 + 0.06 = 0.3222$$

$$h_2 = \tanh(0.3222) \approx 0.3115$$

$$o_2 = 0.8 \times 0.3115 = 0.2492$$

- $t = 3$:

$$z_3 = W_{hh}h_2 + W_{xh}x_3 = 0.9 \times 0.3115 + 0.6 \times 0.4 = 0.2803 + 0.24 = 0.5203$$

$$h_3 = \tanh(0.5203) \approx 0.4775$$

$$o_3 = 0.8 \times 0.4775 = 0.3820$$

3. Concrete Example

Step 2: Compute Loss

$$L = \frac{1}{2}[(0.2330 - 0.4)^2 + (0.2492 - 0.2)^2 + (0.3820 - 0.1)^2]$$

$$= \frac{1}{2}[0.0279 + 0.0024 + 0.0789] = \frac{1}{2} \times 0.1092 = 0.0546$$

3. Concrete Example

Step 3: Backward Pass

For each t,

$$\frac{\partial L}{\partial o_t} = o_t - y_t$$

- $t = 3$: $\delta_{o_3} = 0.3820 - 0.1 = 0.2820$
- $t = 2$: $\delta_{o_2} = 0.2492 - 0.2 = 0.0492$
- $t = 1$: $\delta_{o_1} = 0.2330 - 0.4 = -0.1670$

Gradient w.r.t W_{hy}:

$$\frac{\partial L}{\partial W_{hy}} = \sum_{t=1}^{3} \delta_{o_t} h_t$$

Calculate each term:

- $t = 1 : -0.1670 \times 0.2913 = -0.0486$
- $t = 2 : 0.0492 \times 0.3115 = 0.0153$
- $t = 3 : 0.2820 \times 0.4775 = 0.1347$

Summing:

$$\frac{\partial L}{\partial W_{hy}} = -0.0486 + 0.0153 + 0.1347 = 0.1014$$

3. Concrete Example

Step 4: Backpropagate through Time to Hidden Layers

We now find $\frac{\partial L}{\partial h_t}$ for all t, then backpropagate the error through h_t to weights W_{hh} and W_{xh}.

Initialize:

$$\delta h_t = \frac{\partial L}{\partial h_t} = \frac{\partial L}{\partial o_t}\frac{\partial o_t}{\partial h_t} + \text{contribution from } h_{t+1}$$

For output:

$$\frac{\partial o_t}{\partial h_t} = W_{hy} = 0.8$$

We also backpropagate through the recurrent connection. The key recursive relation for error flowing backward is:

$$\delta h_t = \delta o_t \cdot W_{hy} + \delta h_{t+1} \cdot W_{hh} \cdot (1 - h_t^2)$$

Where $(1 - h_t^2)$ is the derivative of $\tanh$.

Calculate starting from $t = 3$:

- $t = 3$:

$$\delta h_3 = \delta o_3 \times 0.8 = 0.2820 \times 0.8 = 0.2256$$

- $t = 2$:

$$\delta h_2 = \delta o_2 \times 0.8 + \delta h_3 \times 0.9 \times (1 - 0.3115^2)$$

$$= 0.0492 \times 0.8 + 0.2256 \times 0.9 \times (1 - 0.0970) = 0.0394 + 0.2256 \times 0.9 \times 0.9030$$

$$= 0.0394 + 0.1830 = 0.2224$$

- $t = 1$:

$$\delta h_1 = \delta o_1 \times 0.8 + \delta h_2 \times 0.9 \times (1 - 0.2913^2)$$

$$= (-0.1670) \times 0.8 + 0.2224 \times 0.9 \times (1 - 0.0849) = -0.1336 + 0.2224 \times 0.9 \times 0.9151$$

$$= -0.1336 + 0.1832 = 0.0496$$

Step 5: Gradients for W_{hh} and W_{xh}

Recall:

$$z_t = W_{hh}h_{t-1} + W_{xh}x_t$$

$$h_t = \tanh(z_t)$$

Gradients wrt weights come from chain rule:

$$\frac{\partial L}{\partial W_{hh}} = \sum_{t=1}^{3} \delta h_t \cdot (1 - h_t^2) \cdot h_{t-1}$$

$$\frac{\partial L}{\partial W_{xh}} = \sum_{t=1}^{3} \delta h_t \cdot (1 - h_t^2) \cdot x_t$$

3. Concrete Example

Step 5: Gradients for W_{hh} and W_{xh}

Calculate for each t:

- $t = 1$:

$$\delta_t = \delta h_1 (1 - h_1^2) = 0.0496 \times (1 - 0.2913^2) = 0.0496 \times 0.9151 = 0.0454$$

- $t = 2$:

$$\delta_t = 0.2224 \times (1 - 0.3115^2) = 0.2224 \times 0.9030 = 0.2008$$

- $t = 3$:

$$\delta_t = 0.2256 \times (1 - 0.4775^2) = 0.2256 \times (1 - 0.2280) = 0.2256 \times 0.7720 = 0.1743$$

3. Concrete Example

Step 5: Gradients for W_{hh} and W_{xh}

Calculate for each t:

- $t = 1$:

$$\delta_t = \delta h_1 (1 - h_1^2) = 0.0496 \times (1 - 0.2913^2) = 0.0496 \times 0.9151 = 0.0454$$

- $t = 2$:

$$\delta_t = 0.2224 \times (1 - 0.3115^2) = 0.2224 \times 0.9030 = 0.2008$$

- $t = 3$:

$$\delta_t = 0.2256 \times (1 - 0.4775^2) = 0.2256 \times (1 - 0.2280) = 0.2256 \times 0.7720 = 0.1743$$

3. Concrete Example

Step 5: Gradients for W_{hh} and W_{xh}

Now sum for each weight:

- W_{hh}:

$$\sum \delta_t \times h_{t-1}$$

Recall $h_0 = 0$, so terms:

$$t = 1 : 0.0454 \times 0 = 0$$
$$t = 2 : 0.2008 \times 0.2913 = 0.0585$$
$$t = 3 : 0.1743 \times 0.3115 = 0.0543$$
$$\frac{\partial L}{\partial W_{hh}} = 0 + 0.0585 + 0.0543 = 0.1128$$

3. Concrete Example

Step 5: Gradients for W_{hh} and W_{xh}

- W_{xh}:

$$\sum \delta_t \times x_t$$

$$t = 1 : 0.0454 \times 0.5 = 0.0227$$

$$t = 2 : 0.2008 \times 0.1 = 0.0201$$

$$t = 3 : 0.1743 \times 0.4 = 0.0697$$

$$\frac{\partial L}{\partial W_{xh}} = 0.0227 + 0.0201 + 0.0697 = 0.1125$$

Challenges in BPTT Training

- Vanishing gradients: Gradients shrink exponentially with time steps due to repeated multiplication by derivatives less than 1 and weight matrices with eigenvalues less than 1.
- Exploding gradients: Gradients grow exponentially due to weight matrices with eigenvalues greater than 1.
- Both cause learning difficulties: forgetfulness or unstable updates.
- Gradients tend to align with dominant eigenvectors - losing diversity.

Practical Solutions

- Gradient Clipping:
- Prevents exploding gradients by limiting gradient norm to a threshold:
- Stabilizes training, but doesn’t fix vanishing gradients.
- LSTM & GRU Architectures:
- LSTM: Uses memory cells and gates (forget, input, output) to preserve gradients, learn long-term dependencies.
- GRU: Simplified LSTM with update & reset gates; fewer parameters, similar benefits.
- Truncated BPTT:
- Backpropagate over fixed number of steps (not entire sequence) to reduce computation and limit gradient path length.

Summary

- BPTT enables effective training of RNNs by unrolling and applying chain rule through time.
- It accounts for temporal dependencies by accumulating gradients from all time steps.
- Faces vanishing/exploding gradient problems due to long chains of product derivatives.
- Solutions like gradient clipping, truncated BPTT, and gated architectures effectively address these challenges.

References

[1] A. Zhang, Z. Lipton, M. Li, and A. Smola, “Dive into Deep Learning: 9.7 Backpropagation Through Time,” [Online]. Available: [1] A. Zhang, Z. Lipton, M. Li, and A. Smola, “Dive into Deep Learning: 9.7 Backpropagation Through Time,” [Online]. Available: BPTT enables effective training of RNNs by unrolling and applying chain rule through time.

[2] “Back Propagation Through Time - RNN,” GeeksforGeeks, [Online]. Available: BPTT enables effective training of RNNs by unrolling and applying chain rule through time.

[3] “Backpropagation Through Time Explained With Derivations,” Quark.ai, [Online]. Available: BPTT enables effective training of RNNs by unrolling and applying chain rule through time.

[4] “Backpropagation Through Time,” Wikipedia, [Online]. Available: BPTT enables effective training of RNNs by unrolling and applying chain rule through time.

[5] “CS230 RNN Cheatsheet,” Stanford University, [Online]. Available: BPTT enables effective training of RNNs by unrolling and applying chain rule through time.

[6] S. Raschka, “L15.4 BPTT Overview,” YouTube, [Online]. Available: BPTT enables effective training of RNNs by unrolling and applying chain rule through time.

[7] “LSTM: Derivation of Backpropagation Through Time,” GeeksforGeeks, [Online]. Available: BPTT enables effective training of RNNs by unrolling and applying chain rule through time.

[8] “Let's Understand the Problems with RNNs,” [Online]. Available: BPTT enables effective training of RNNs by unrolling and applying chain rule through time.

[9] “Long Short-Term Memory,” Wikipedia, [Online]. Available: BPTT enables effective training of RNNs by unrolling and applying chain rule through time.

[10] S. Soni, “RNN from Basic to Advanced,” Medium, [Online]. Available: BPTT enables effective training of RNNs by unrolling and applying chain rule through time.

[11] J. Starmer, “RNNs Clearly Explained – StatQuest,” YouTube, [Online]. Available: BPTT enables effective training of RNNs by unrolling and applying chain rule through time.

Q&A

Thank You!