

CPEN 455: Deep Learning

Lecture 6: Recurrent Neural Networks

Qi Yan, Muchen Li, Jiahe Liu, Qihang Zhang, Renjie Liao

University of British Columbia

Winter, Term 1, 2026

Outline

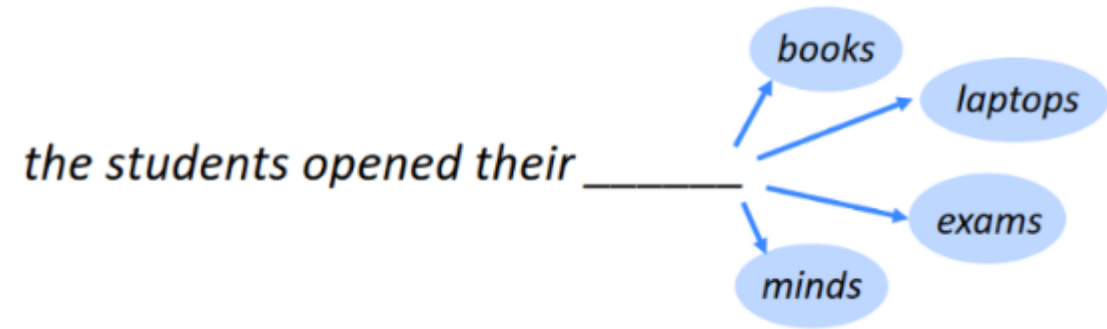
- Recurrent Neural Networks (RNNs)
 - Motivations
 - RNN basics
 - Example: Character-level language model
- Back Propagation Through Time (BPTT)
 - Vanishing and exploding gradients
- Long Short-Term Memory (LSTM)
- Truncated BPTT

Outline

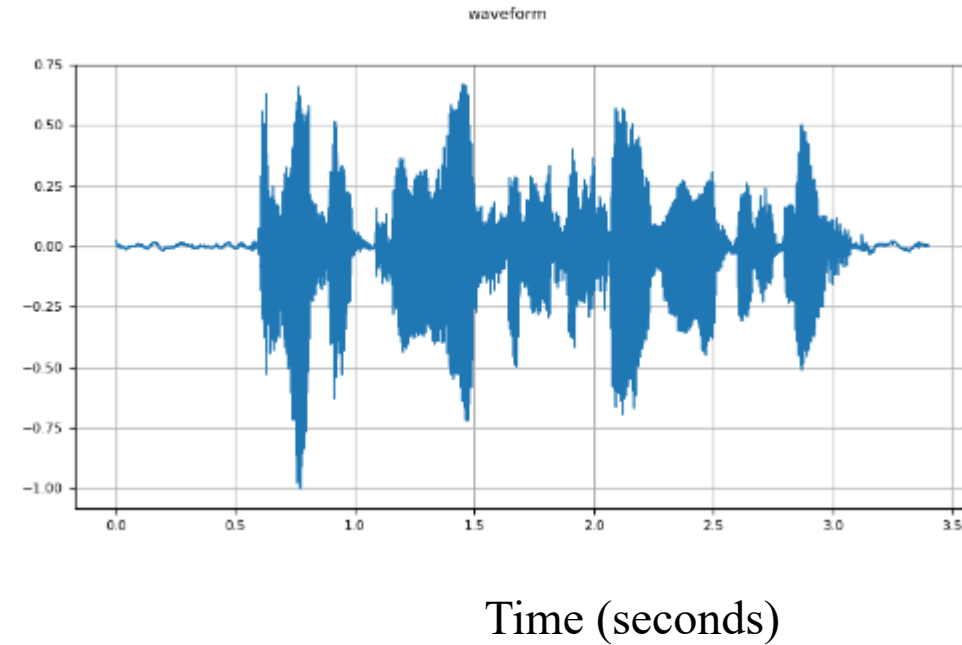
- Recurrent Neural Networks (RNNs)
 - **Motivations**
 - RNN basics
 - Example: Character-level language model
- Back Propagation Through Time (BPTT)
 - Vanishing and exploding gradients
- Long Short-Term Memory (LSTM)
- Truncated BPTT

Sequential Data

- Text generation
- Stock price prediction
- Speech recognition
- Music generation



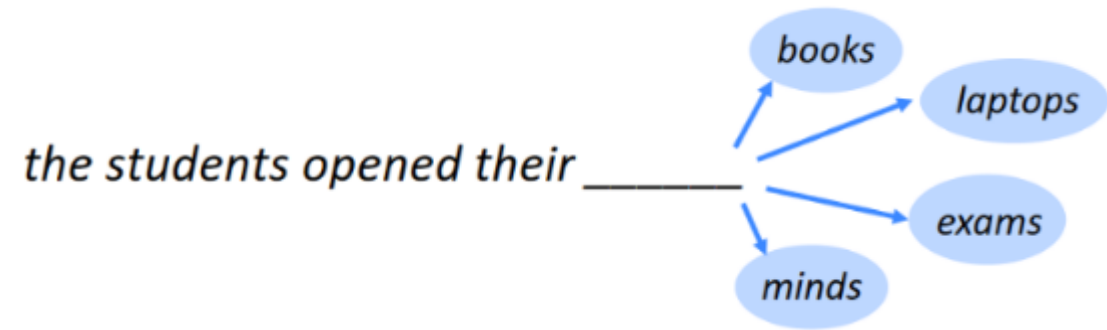
Speech recognition



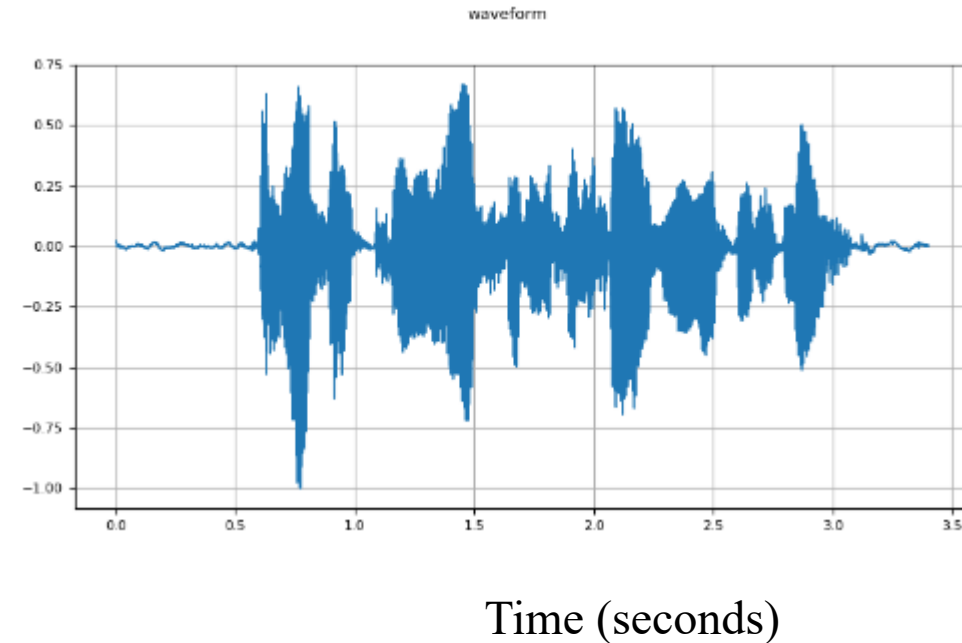
Audio samples form an ordered sequence.

Sequential Data

- Text generation
 - Stock price prediction
 - Speech recognition
 - Music generation
-
- A flattened fixed-window FCN
 - Fixes the input window length



Speech recognition

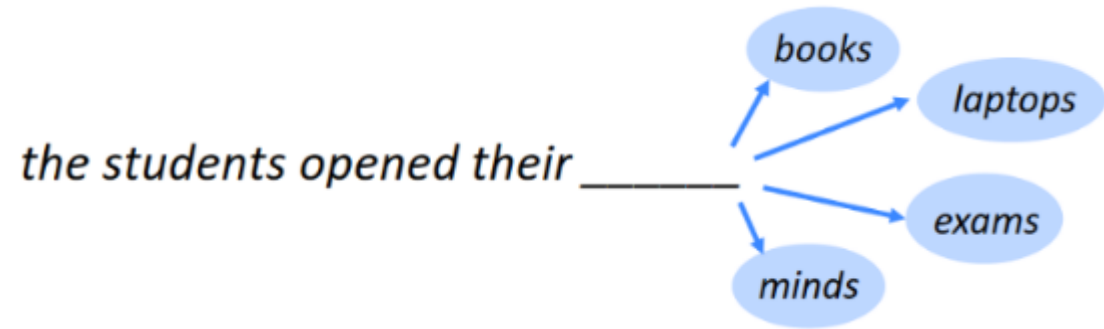


Audio samples form an ordered sequence.

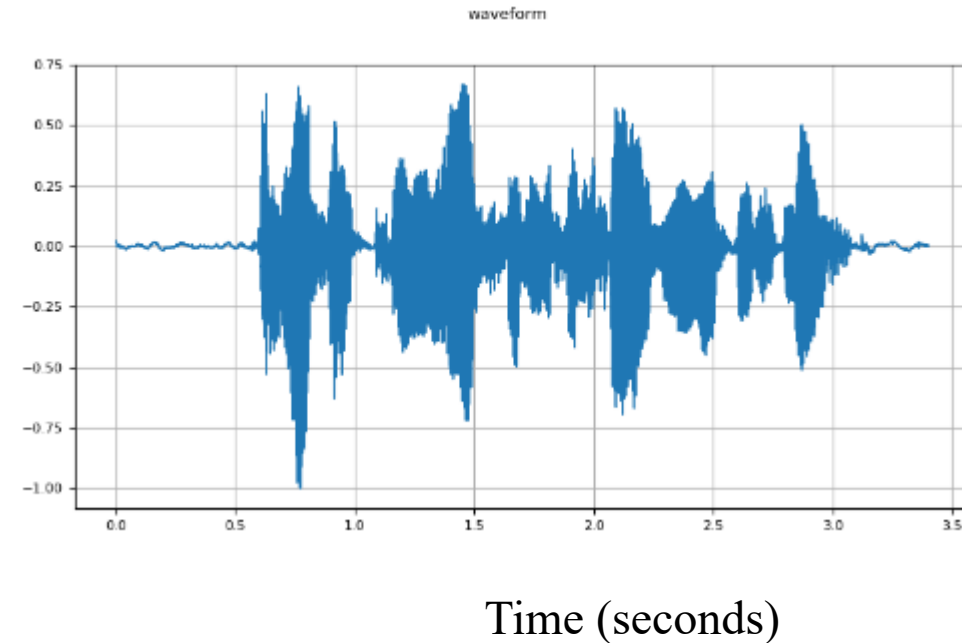
Sequential Data

- Text generation
 - Stock price prediction
 - Speech recognition
 - Music generation
-
- A flattened fixed-window FCN
 - Fixes the input window length

Sequences may have varying lengths.
Temporal CNNs can also handle them with
an appropriate head.



Speech recognition



Audio samples form an ordered sequence.

Applications

- One-to-many:
 - Image captioning

One image



Generated caption

“A green car parked in front of a yellow building.”



A sequence of output tokens

Applications

- One-to-many:
 - Music generation

A primer note can seed an entire melody [18].



Notes appear in temporal order.

[Play the RNN music demo](#)

Applications

- Many-to-one:
 - Sentiment classification

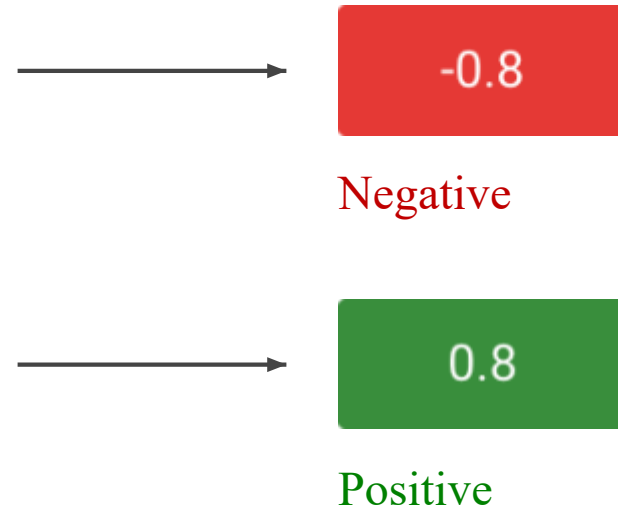
Each text produces one sentiment prediction.

Input text (excerpts)

“I'm extremely angry and disappointed for the result.”

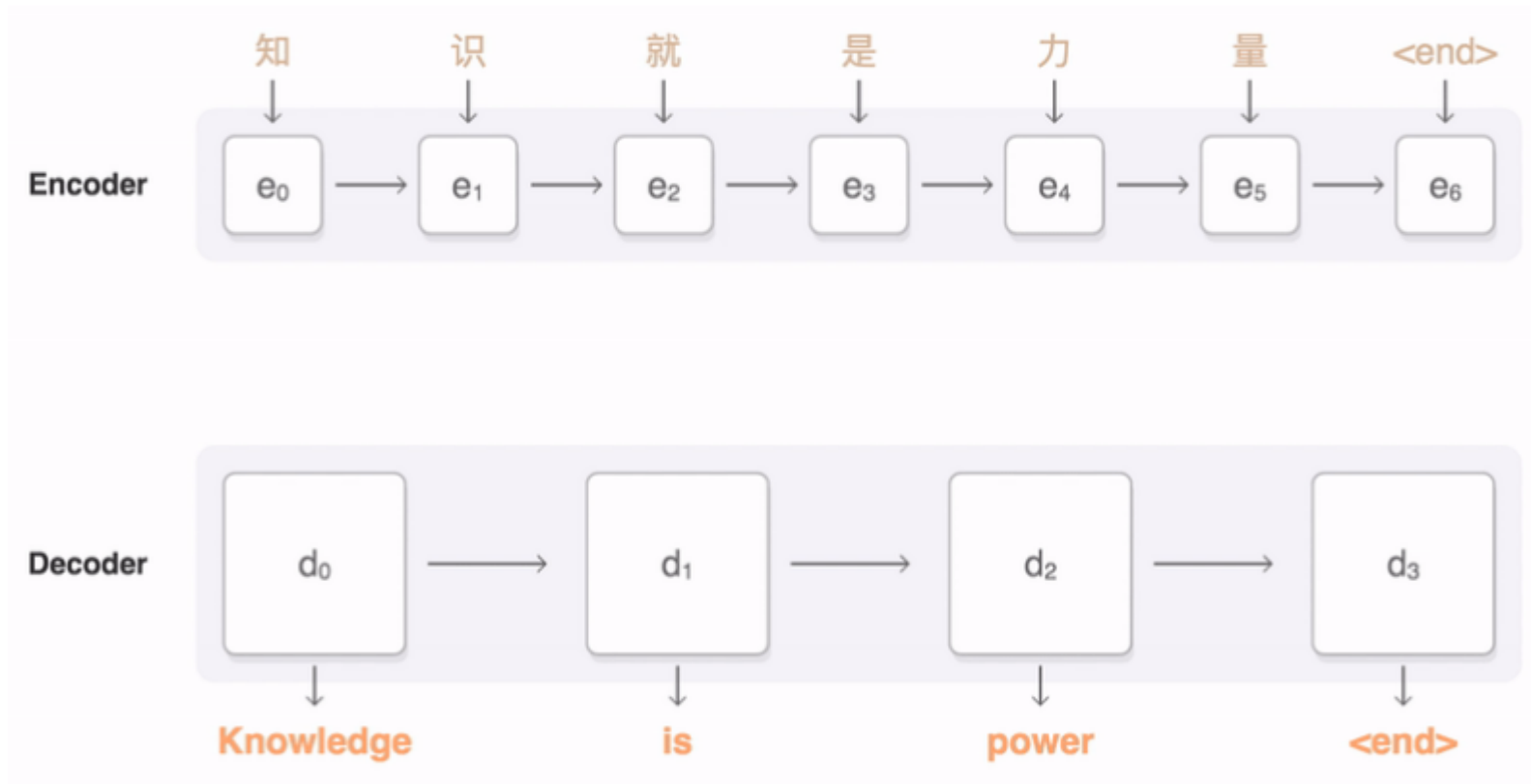
“I'm happy to see our team was working very hard and show professional attitude.”

Sentiment score



Applications

- Many-to-many:
 - Machine translation



Applications

- Many-to-many:
 - Video captioning

Input: a sequence of video frames



Output: a sequence of words

→ “A man is doing stunts
on his bike.”



Time

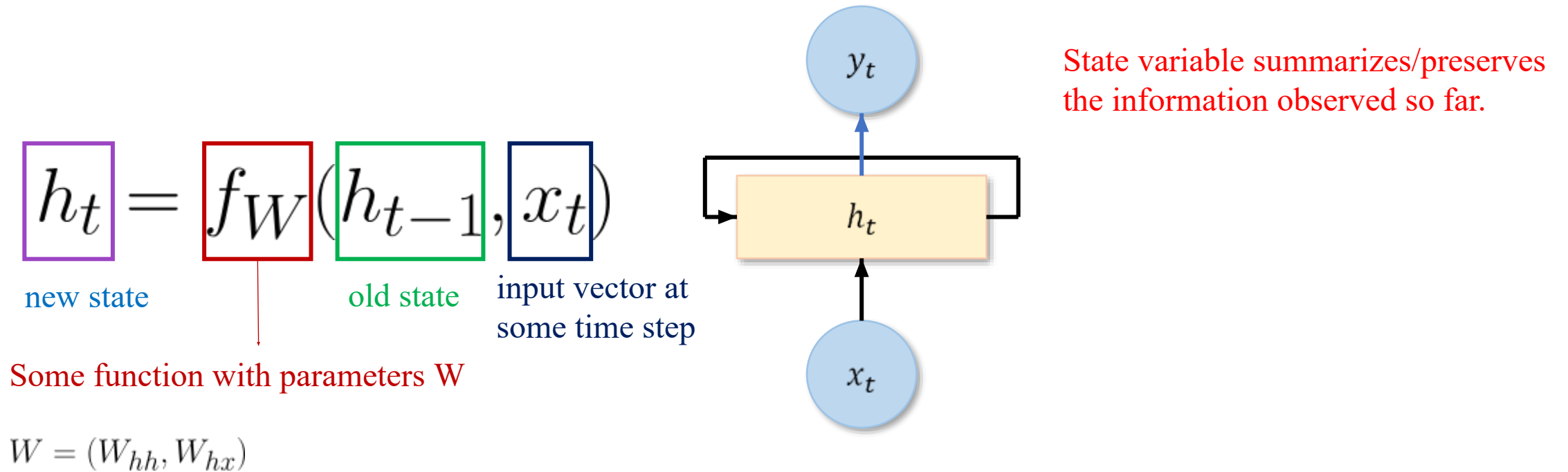
The frame and word sequences can have different lengths.

Outline

- Recurrent Neural Networks (RNNs)
 - Motivations
 - **RNN basics**
 - Example: Character-level language model
- Back Propagation Through Time (BPTT)
 - Vanishing and exploding gradients
- Long Short-Term Memory (LSTM)
- Truncated BPTT

RNN Cell Unit

Recurrent neural networks (RNNs) are networks with loops, allowing information to persist [17].



RNN Readout

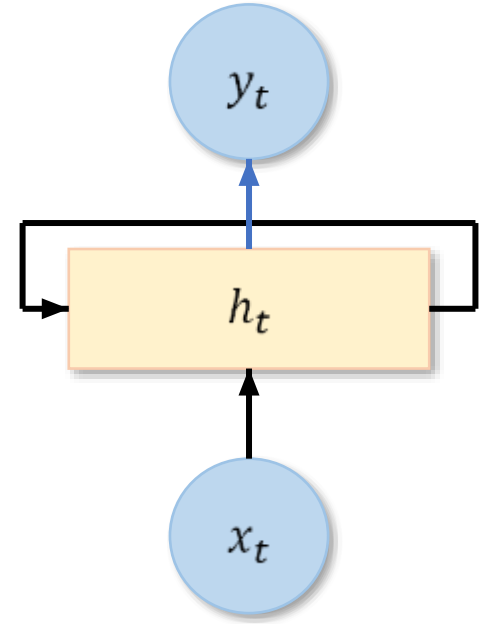
We can process a sequence of vectors $\mathbf{x}$ by applying a **recurrence formula** at every time step:

$$\boxed{y_t} = \boxed{f_{W_{hy}}}\left(\boxed{h_t}\right)$$

output

new state

Readout weights: W_{hy}



Formula of RNN (Vanilla)

The weights are shared at all time steps. Biases are omitted.

$$h_t = f_W(h_{t-1}, x_t)$$



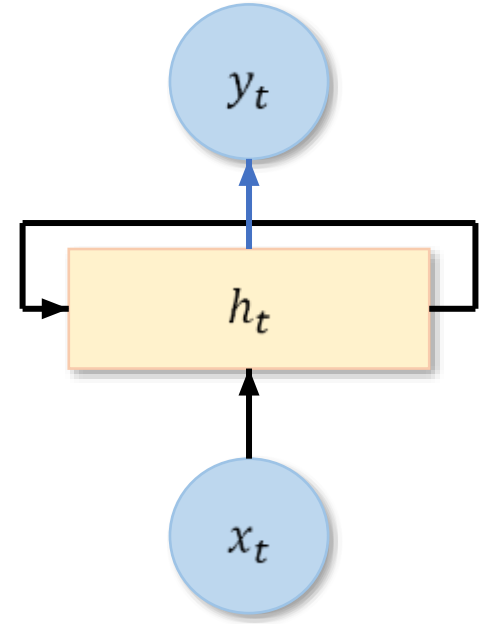
$$h_t = \tanh(W_{hh}h_{t-1} + W_{hx}x_t)$$

$$y_t = W_{hy}h_t$$

Dimensions (input d, hidden h, output q):

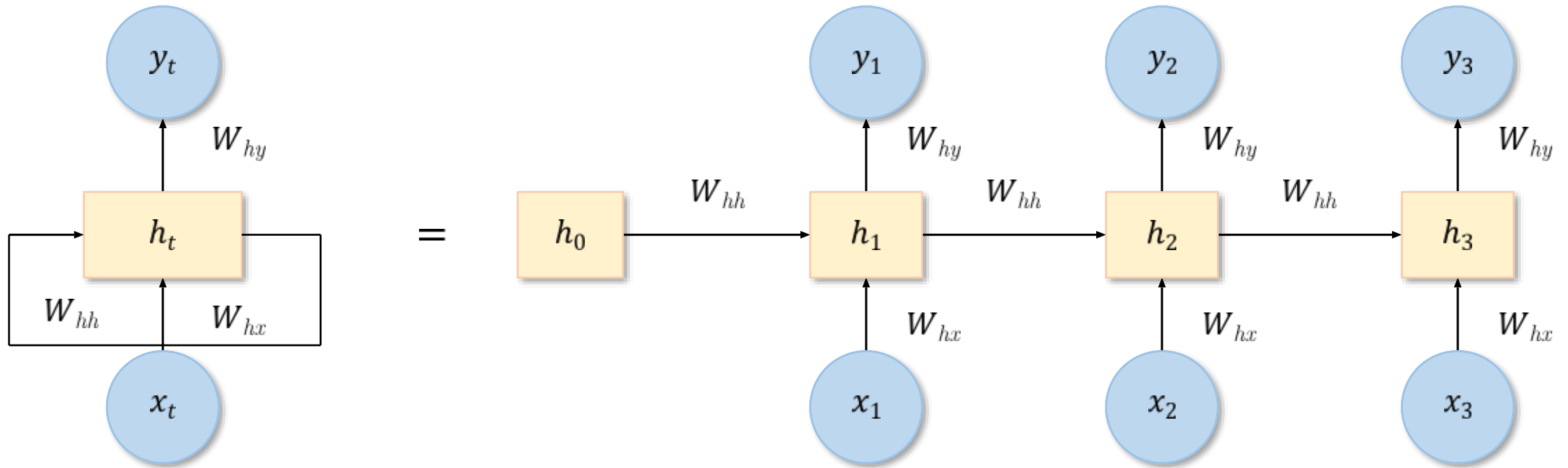
$$x_t \in \mathbb{R}^d, \quad h_t \in \mathbb{R}^h, \quad y_t \in \mathbb{R}^q, \quad h_0 = 0$$

$$W_{hx} \in \mathbb{R}^{h \times d}, \quad W_{hh} \in \mathbb{R}^{h \times h}, \quad W_{hy} \in \mathbb{R}^{q \times h}$$



Parameter sharing

RNNs reuse one recurrent cell across a sequence.



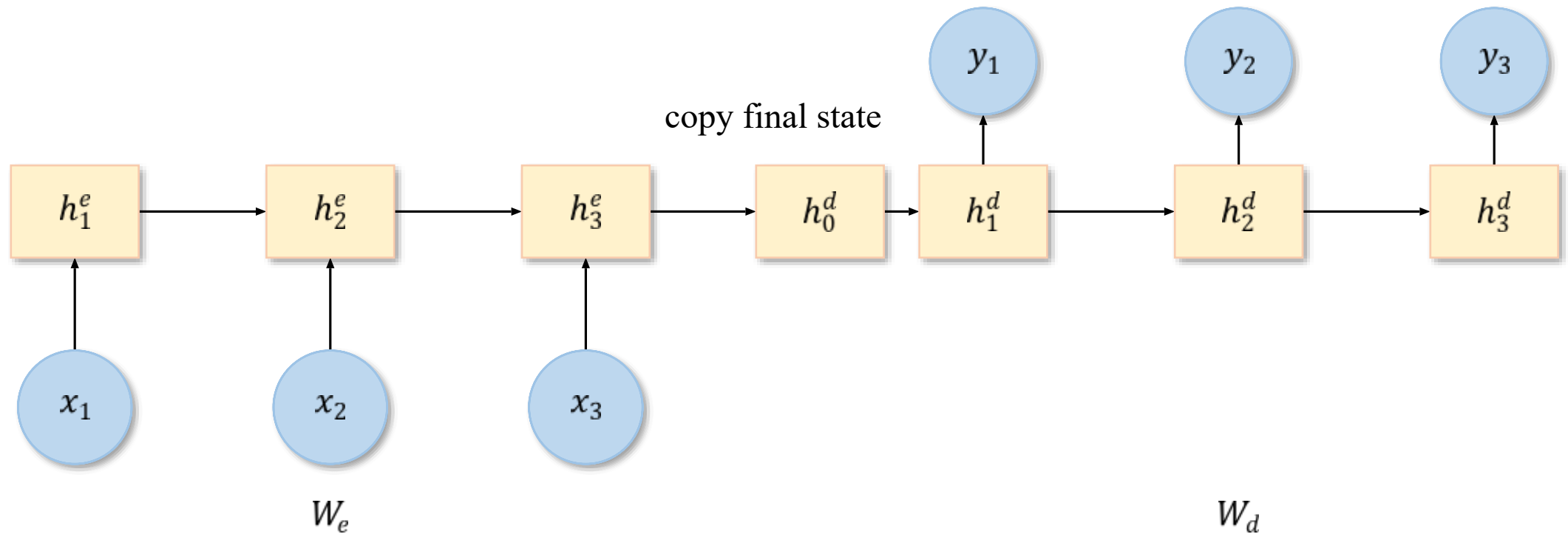
The same function and the same weights are used at every time step.

RNN Computational Graph

Many-to-Many: Many-to-One + One-to-Many

Encoder

Decoder



one set of encoder weights

one set of decoder weights

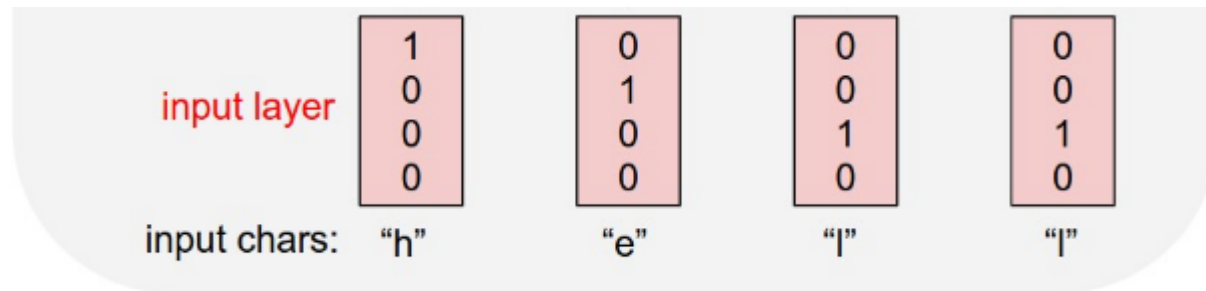
Outline

- Recurrent Neural Networks (RNNs)
 - Motivations
 - RNN basics
 - **Example: Character-level language model**
- Back Propagation Through Time (BPTT)
 - Vanishing and exploding gradients
- Long Short-Term Memory (LSTM)
- Truncated BPTT

Example: Character-level Language Model

Vocabulary:[h,e,l,o]

Example training sequence :
“hello”

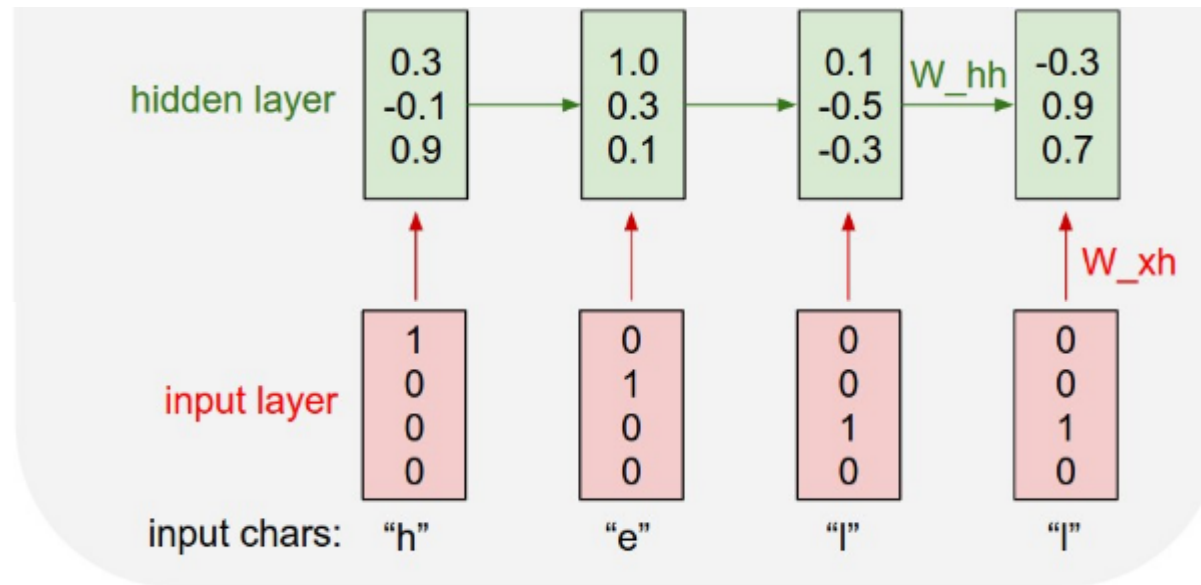


Example: Character-level Language Model

$$h_t = \tanh(W_{hh}h_{t-1} + W_{hx}x_t)$$

Vocabulary:[h,e,l,o]

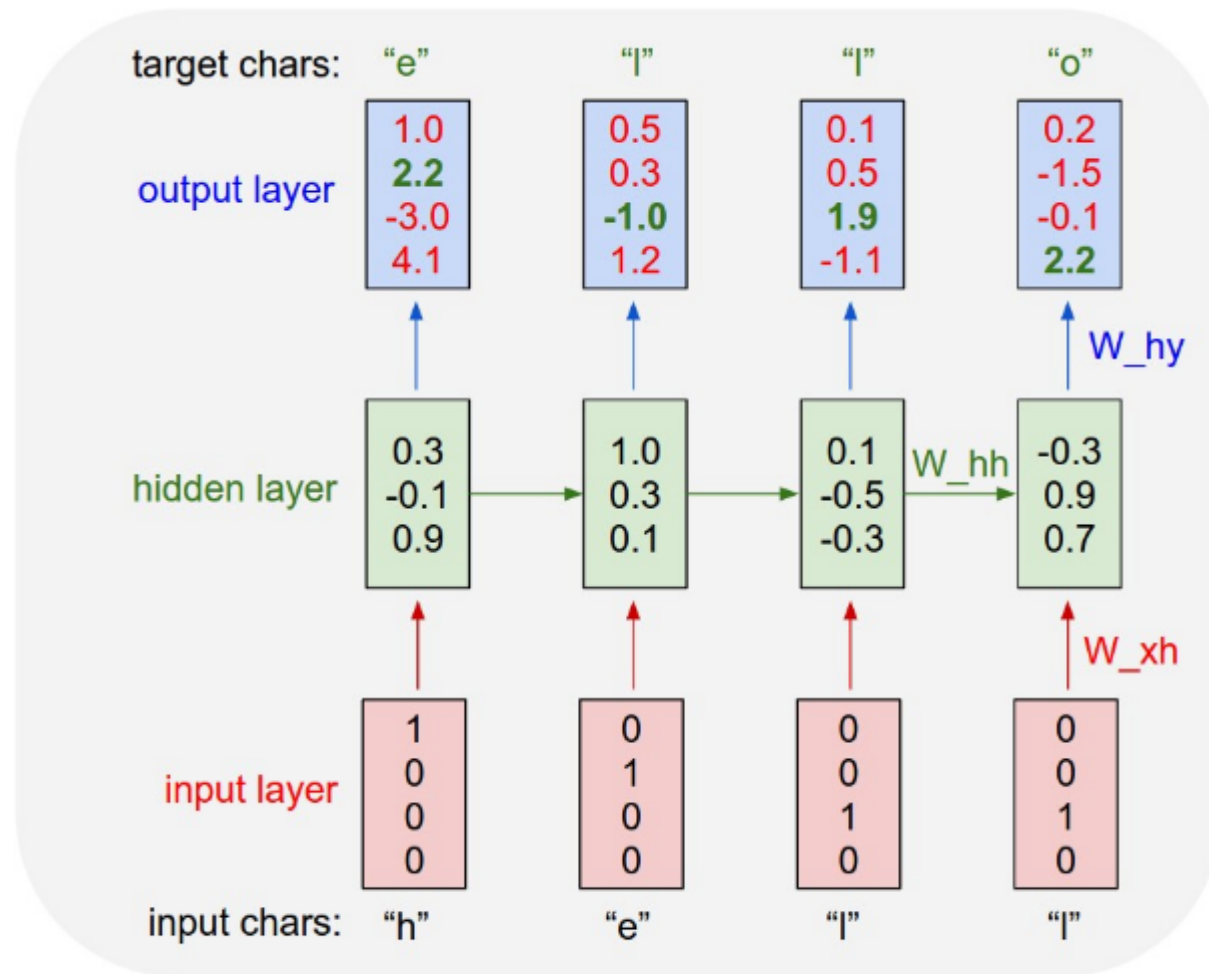
Example training sequence :
“hello”



Example: Character-level Language Model

Vocabulary:[h,e,l,o]

Example training sequence :
“hello”



Example: Character-level Language Model

The readout produces one score (logit) for each character.

$$p_t = \text{softmax}(y_t).$$

Input “h” → predict “e”; input “e” → predict “l”; ...

Softmax turns these scores into a probability distribution over the vocabulary.

Example: Character-level Language Model

The readout produces one score (logit) for each character.

$$p_t = \text{softmax}(y_t).$$

Input “h” → predict “e”; input “e” → predict “l”; ...

Train with cross-entropy; the next-character target is one-hot:

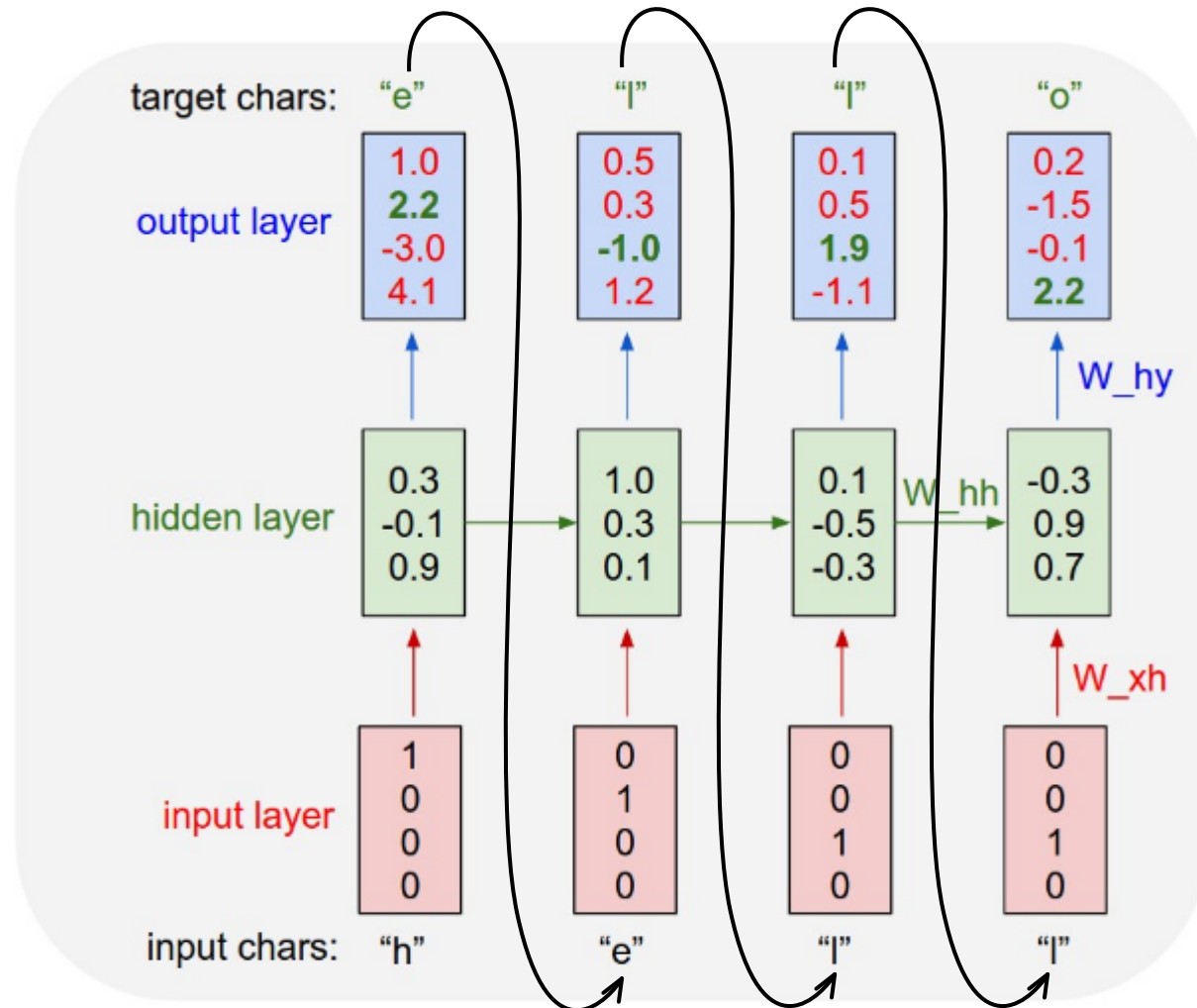
$$L = -\frac{1}{T} \sum_{t=1}^T \bar{y}_t^\top \log p_t.$$

Teacher forcing: feed the true previous character during training.

Example: Character-level Language Model

Vocabulary:[h,e,l,o]

At test-time sample characters
one at a time, feed back to
model



Outline

- Recurrent Neural Networks (RNNs)
 - Motivations
 - RNN basics
 - Example: Character-level language model
- **Back Propagation Through Time (BPTT)**
 - Vanishing and exploding gradients
- Long Short-Term Memory (LSTM)
- Truncated BPTT

Back Propagation Through Time (BPTT)

- Applying backpropagation in RNNs is called *backpropagation through time* [3].
- This procedure requires us to expand (or unroll) the computational graph of an RNN one time step at a time.

BPTT: An Example

Use the same vanilla RNN as in the introduction.

Bias terms are omitted; the initial state is fixed.

$$h_t = \tanh(W_{\text{hh}}h_{t-1} + W_{\text{hx}}x_t),$$

$$y_t = W_{\text{hy}}h_t.$$

BPTT: An Example

Use the same vanilla RNN as in the introduction.

Bias terms are omitted; the initial state is fixed.

$$h_t = \tanh(W_{\text{hh}}h_{t-1} + W_{\text{hx}}x_t),$$

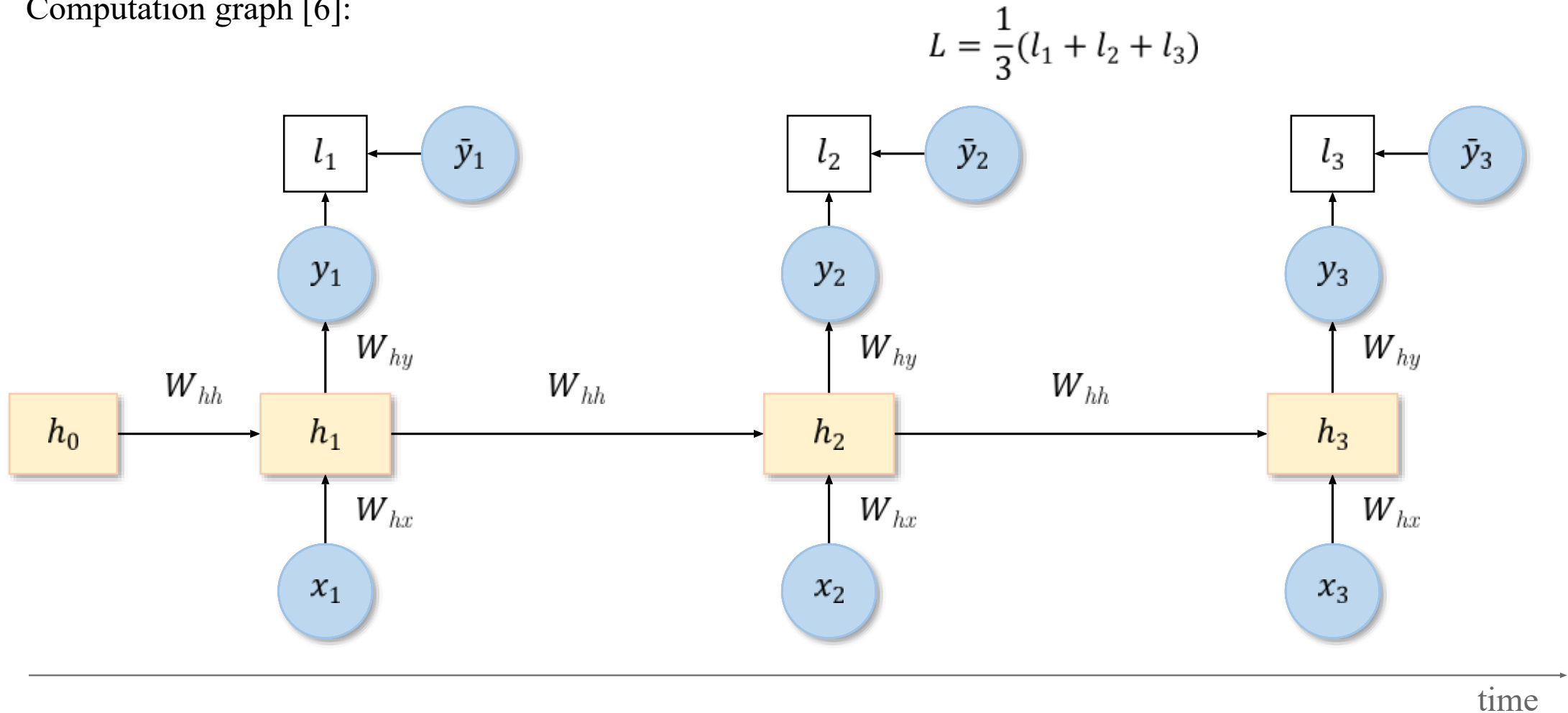
$$y_t = W_{\text{hy}}h_t.$$

Loss at each time step and the mean objective:

$$l_t = l(y_t, \bar{y}_t), \quad L = \frac{1}{T} \sum_{t=1}^T l_t$$

BPTT: An Example

Computation graph [6]:

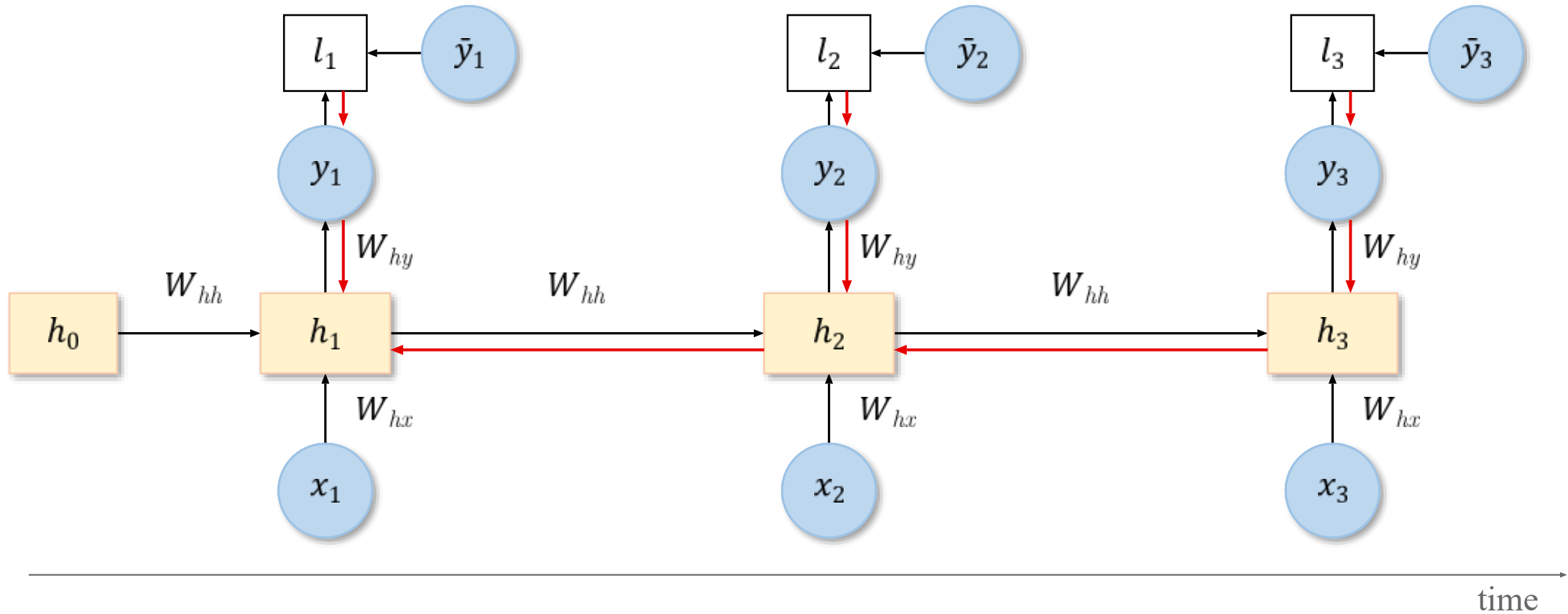


The same nonlinear RNN and weights are reused at every time step.

BPTT: An Example

Key components in gradient computation:

$$\partial L / \partial W_{hx}, \quad \partial L / \partial W_{hh}, \quad \partial L / \partial W_{hy}$$



Backpropagation follows the dependencies in reverse.

BPTT: An Example

First, differentiate the mean loss with respect to each prediction:

$$q_t = \nabla_{y_t} L = \frac{\partial L}{\partial y_t} = \frac{1}{T} \nabla_{y_t} l(y_t, \bar{y}_t) \in \mathbb{R}^q$$

We write vector gradients as column vectors.

BPTT: An Example

First, differentiate the mean loss with respect to each prediction:

$$q_t = \nabla_{y_t} L = \frac{\partial L}{\partial y_t} = \frac{1}{T} \nabla_{y_t} l(y_t, \bar{y}_t) \in \mathbb{R}^q$$

We write vector gradients as column vectors.

Then sum the contributions to the shared readout matrix across time:

$$\frac{\partial L}{\partial W_{\text{hy}}} = \sum_{t=1}^T q_t h_t^\top$$

BPTT: An Example

At the final time step, the hidden-state gradient comes from the readout:

$$r_T = \nabla_{h_T} L = \frac{\partial L}{\partial h_T} = \left(\frac{\partial y_T}{\partial h_T} \right)^\top q_T = W_{\text{hy}}^\top q_T$$

For the recurrent path, use the forward Jacobian:

$$J_t = \frac{\partial h_t}{\partial h_{t-1}} = D_t W_{\text{hh}}, \quad D_t = \text{diag}(1 - h_t \odot h_t)$$

BPTT: An Example

At the final time step, the hidden-state gradient comes from the readout:

$$r_T = \nabla_{h_T} L = \frac{\partial L}{\partial h_T} = \left(\frac{\partial y_T}{\partial h_T} \right)^\top q_T = W_{\text{hy}}^\top q_T$$

For the recurrent path, use the forward Jacobian:

$$J_t = \frac{\partial h_t}{\partial h_{t-1}} = D_t W_{\text{hh}}, \quad D_t = \text{diag}(1 - h_t \odot h_t)$$

At earlier steps, add the current readout and the future-state contribution:

$$r_t = \nabla_{h_t} L = \frac{\partial L}{\partial h_t} = W_{\text{hy}}^\top q_t + J_{t+1}^\top r_{t+1}, \quad t < T$$

BPTT: An Example

At the final time step, the hidden-state gradient comes from the readout:

$$r_T = \nabla_{h_T} L = \frac{\partial L}{\partial h_T} = \left(\frac{\partial y_T}{\partial h_T} \right)^\top q_T = W_{\text{hy}}^\top q_T$$

For the recurrent path, use the forward Jacobian:

$$J_t = \frac{\partial h_t}{\partial h_{t-1}} = D_t W_{\text{hh}}, \quad D_t = \text{diag}(1 - h_t \odot h_t)$$

At earlier steps, add the current readout and the future-state contribution:

$$r_t = \nabla_{h_t} L = \frac{\partial L}{\partial h_t} = W_{\text{hy}}^\top q_t + J_{t+1}^\top r_{t+1}, \quad t < T$$

Repeated substitution gives (the product is identity when $s=t$):

$$r_t = \sum_{s=t}^T J_{t+1}^\top \cdots J_s^\top W_{\text{hy}}^\top q_s$$

BPTT: An Example

Pass the hidden-state gradient through tanh:

$$z_t = W_{\text{hh}}h_{t-1} + W_{\text{hx}}x_t, \quad \delta_t = \nabla_{z_t} L = \frac{\partial L}{\partial z_t} = D_t r_t$$

Then sum each shared weight's contributions across time:

$$\frac{\partial L}{\partial W_{\text{hx}}} = \sum_{t=1}^T \delta_t x_t^\top,$$

$$\frac{\partial L}{\partial W_{\text{hh}}} = \sum_{t=1}^T \delta_t h_{t-1}^\top.$$

BPTT: An Example

Algorithm 1 BPTT for the vanilla RNN

- 1: **Inputs:** $x_{1...T}, \bar{y}_{1...T}, h_{0...T}, y_{1...T}$ (from the forward pass)
 - 2: $q_t \leftarrow \frac{1}{T} \nabla_{y_t} l(y_t, \bar{y}_t)$ for every t
 - 3: $r_{T+1} \leftarrow 0, J_{T+1} \leftarrow 0$
 - 4: **for** $t = T$ **down to** 1 **do**
 - 5: $D_t \leftarrow \text{diag}(1 - h_t \odot h_t), J_t \leftarrow D_t W_{\text{hh}}$
 - 6: $r_t \leftarrow W_{\text{hy}}^\top q_t + J_{t+1}^\top r_{t+1}$
 - 7: $\delta_t \leftarrow D_t r_t$
 - 8: **end for**
 - Calculate gradients with respect to the shared weights:
 - 9: $\frac{\partial L}{\partial W_{\text{hy}}} = \sum_{t=1}^T q_t h_t^\top$
 - 10: $\frac{\partial L}{\partial W_{\text{hx}}} = \sum_{t=1}^T \delta_t x_t^\top$
 - 11: $\frac{\partial L}{\partial W_{\text{hh}}} = \sum_{t=1}^T \delta_t h_{t-1}^\top$
-

BPTT: An Example

Long-range gradients contain products of recurrent Jacobians:

$$r_t = \sum_{s=t}^T J_{t+1}^\top \cdots J_s^\top W_{\text{hy}}^\top q_s$$

BPTT: An Example

Long-range gradients contain products of recurrent Jacobians:

$$r_t = \sum_{s=t}^T J_{t+1}^\top \cdots J_s^\top W_{\text{hy}}^\top q_s$$

Use matrix norms to bound the accumulated effect:

$$\frac{\partial h_s}{\partial h_t} = J_s \cdots J_{t+1},$$

$$\left\| \frac{\partial h_s}{\partial h_t} \right\|_2 \leq \prod_{k=t+1}^s \|J_k\|_2.$$

A uniform bound below 1 makes this Jacobian product decay with the time lag.

BPTT: An Example

Long-range gradients contain products of recurrent Jacobians:

$$r_t = \sum_{s=t}^T J_{t+1}^\top \cdots J_s^\top W_{\text{hy}}^\top q_s$$

Use matrix norms to bound the accumulated effect:

$$\frac{\partial h_s}{\partial h_t} = J_s \cdots J_{t+1},$$

$$\left\| \frac{\partial h_s}{\partial h_t} \right\|_2 \leq \prod_{k=t+1}^s \|J_k\|_2.$$

A uniform bound below 1 makes this Jacobian product decay with the time lag.

Large factors can amplify gradients; saturated tanh can shrink them.

$$\|J_t\|_2 \leq \|D_t\|_2 \|W_{\text{hh}}\|_2, \quad 0 \leq \|D_t\|_2 \leq 1$$

Vanishing and Exploding Gradients

In a scalar example, repeated Jacobian factors give:

$$0.5^{10} \approx 9.77 \times 10^{-4},$$

$$1.5^{10} \approx 57.7.$$

Gradient norm clipping limits the global gradient norm [10]:

$$g_{\text{clip}} = \begin{cases} g \min\left(1, \frac{c}{\|g\|_2}\right), & g \neq 0, \\ 0, & g = 0, \end{cases} \quad c > 0$$

Clipping addresses exploding gradients. Gates can help preserve long-range gradients.

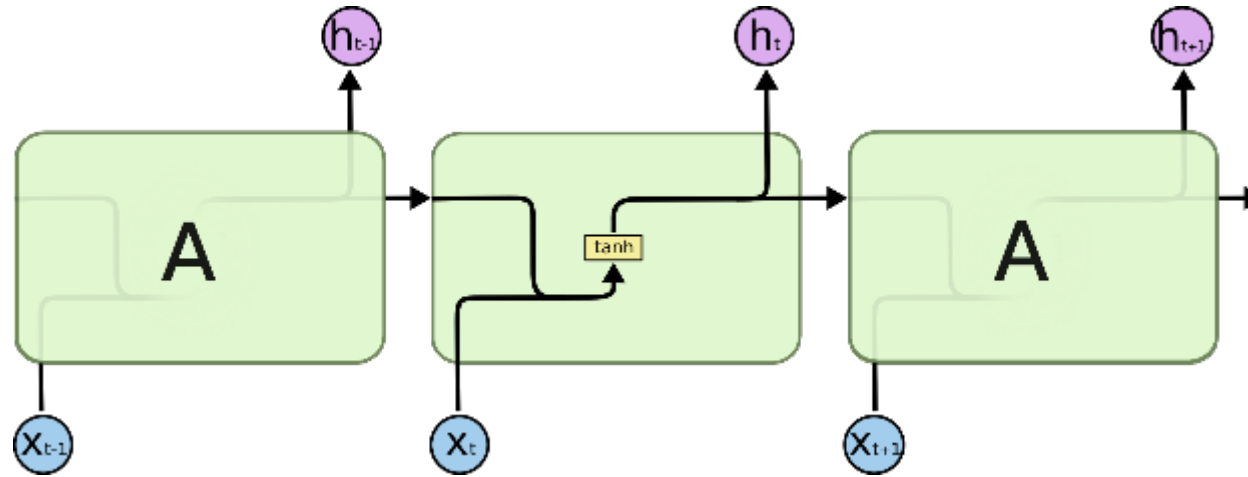
Outline

- Recurrent Neural Networks (RNNs)
 - Motivations
 - RNN basics
 - Example: Character-level language model
- Back Propagation Through Time (BPTT)
 - Vanishing and exploding gradients
- **Long Short-Term Memory (LSTM)**
- Truncated BPTT

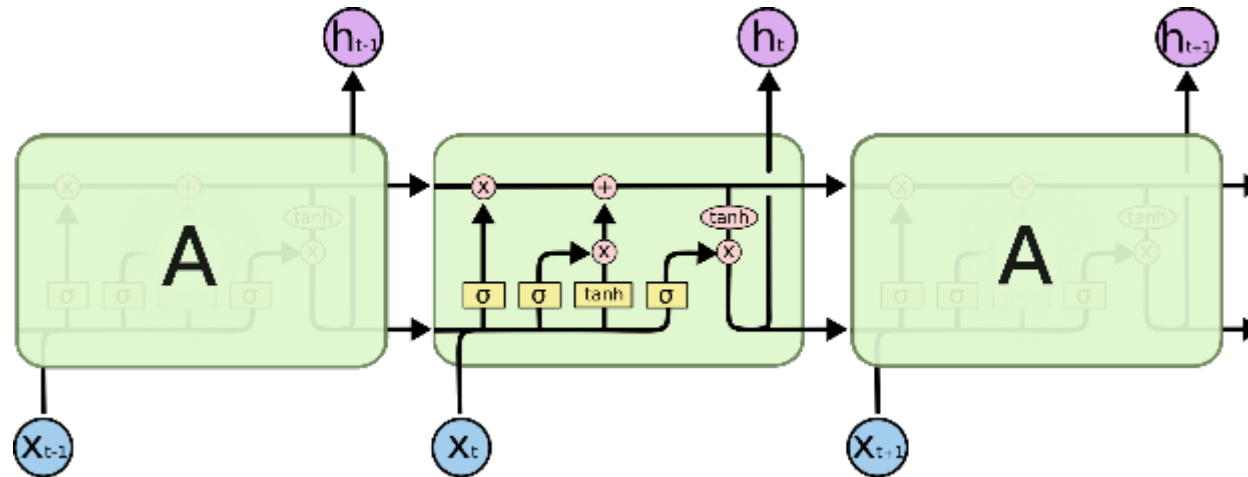
Vanilla RNN vs LSTM

LSTM networks use gated memory to help learn long-term dependencies [8].

RNN

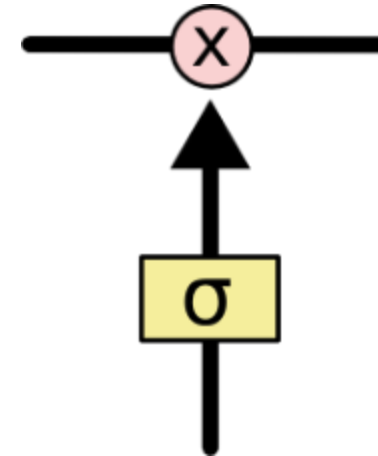
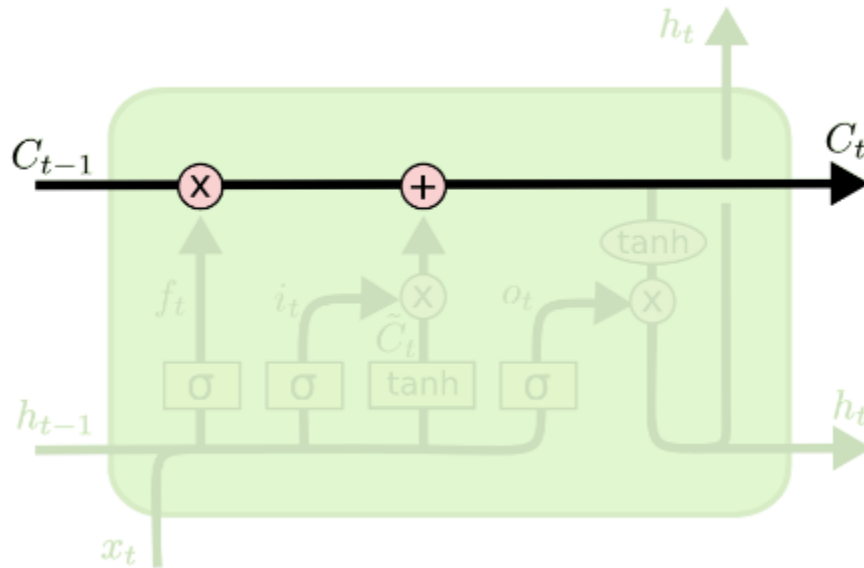


LSTM



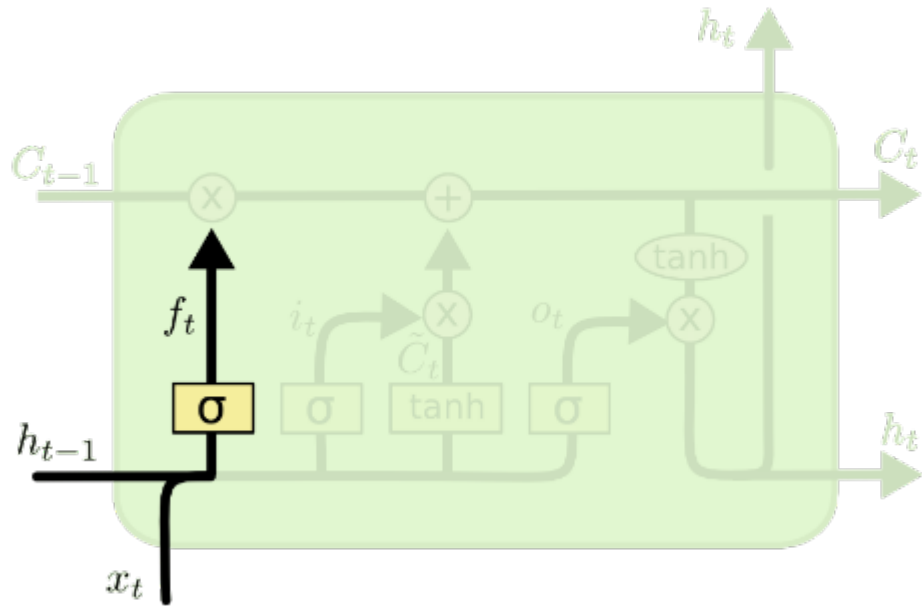
The Core Idea – Cell State

- The cell state is like a conveyor belt. It runs straight down the entire chain, with only some minor linear interactions.
- Gates are a way to optionally let information through. They are composed out of a sigmoid neural net layer and a pointwise multiplication operation.



Step-by-Step LSTM Walk Through

- Forget gate layer:

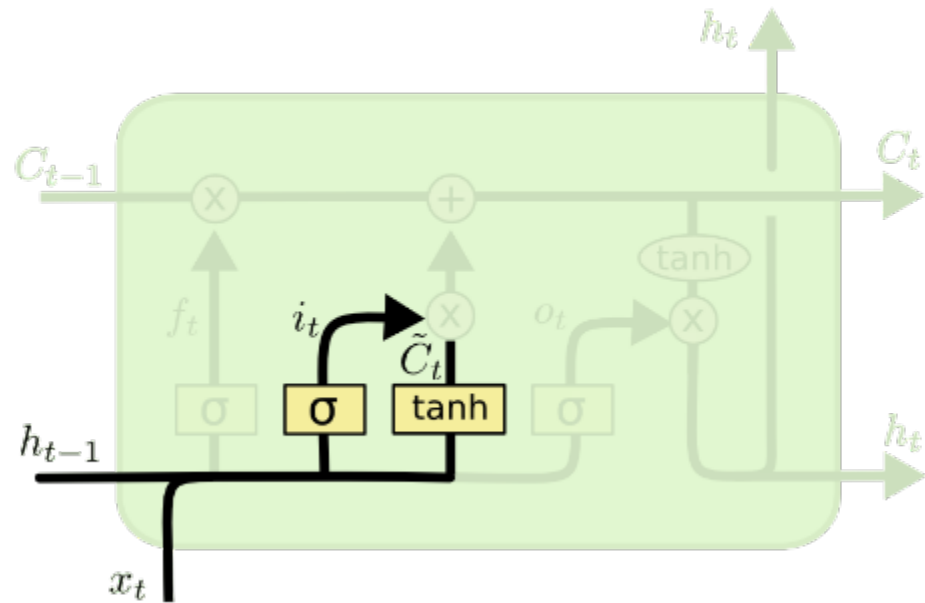


$$f_t = \sigma (W_f \cdot [h_{t-1}, x_t] + b_f)$$

Soft gate: values between 0 and 1

Step-by-Step LSTM Walk Through

- Input gate layer + tanh layer:

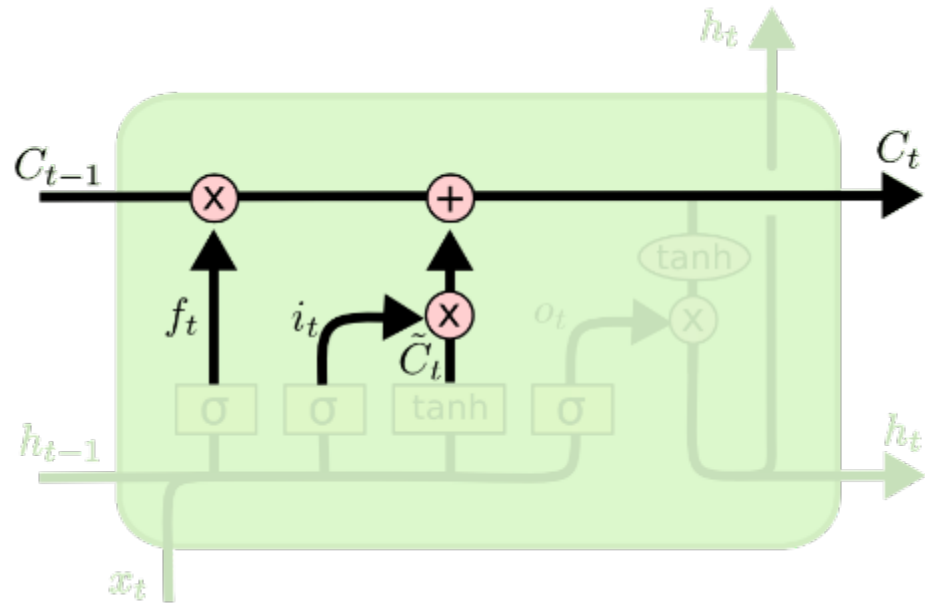


$$i_t = \sigma (W_i \cdot [h_{t-1}, x_t] + b_i)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

Step-by-Step LSTM Walk Through

- Update Cell States:

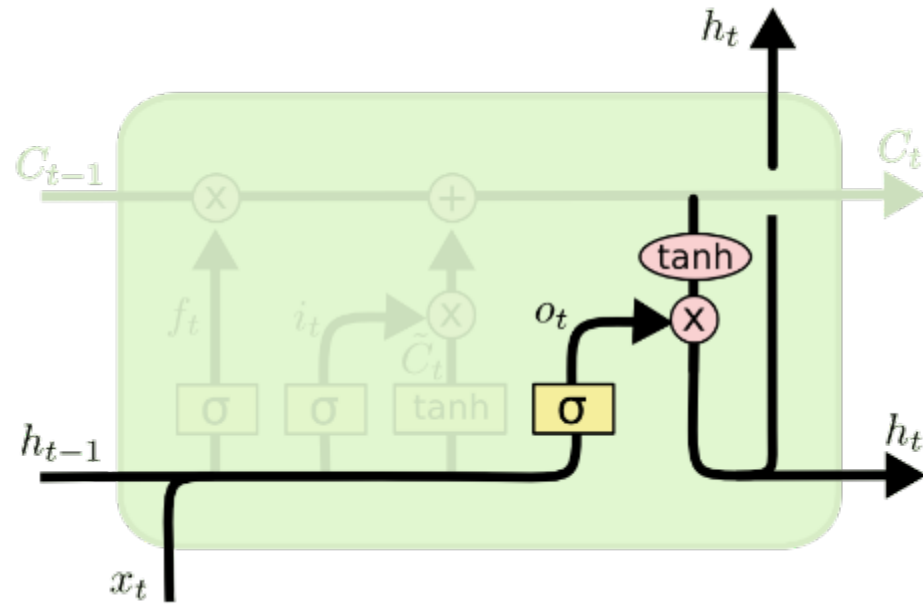


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Multiplication in the diagram is elementwise.

Step-by-Step LSTM Walk Through

- Output:



$$o_t = \sigma (W_o [h_{t-1}, x_t] + b_o)$$

$$h_t = o_t * \tanh (C_t)$$

Multiplication in the diagram is elementwise.

LSTM: The Direct Cell-state Path

The cell state has an additive update, with a gated path from the previous state.

$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t, \quad \left. \frac{\partial C_t}{\partial C_{t-1}} \right|_{\text{direct cell path}} = \text{diag}(f_t).$$

Along this direct path (holding the gate values fixed):

$$f = 0.99 : \quad f^{100} \approx 0.366,$$

$$f = 0.5 : \quad f^{100} \approx 7.9 \times 10^{-31}.$$

Forget gates near 1 help retain information and gradients over many steps.

This is one path through the graph; it is not the full LSTM derivative.

LSTM: A Scalar Example

A gate controls how much to keep, write, or reveal.

Keep 90% of the old cell state; write 20% of the candidate:

$$C_{t-1} = 2, \quad f_t = 0.9, \quad i_t = 0.2, \quad \tilde{C}_t = 0.5,$$

$$C_t = 0.9(2) + 0.2(0.5) = 1.9.$$

LSTM: A Scalar Example

A gate controls how much to keep, write, or reveal.

Keep 90% of the old cell state; write 20% of the candidate:

$$C_{t-1} = 2, \quad f_t = 0.9, \quad i_t = 0.2, \quad \tilde{C}_t = 0.5,$$
$$C_t = 0.9(2) + 0.2(0.5) = 1.9.$$

Reveal 80% of the squashed cell state as the hidden state:

$$o_t = 0.8, \quad h_t = 0.8 \tanh(1.9) \approx 0.765.$$

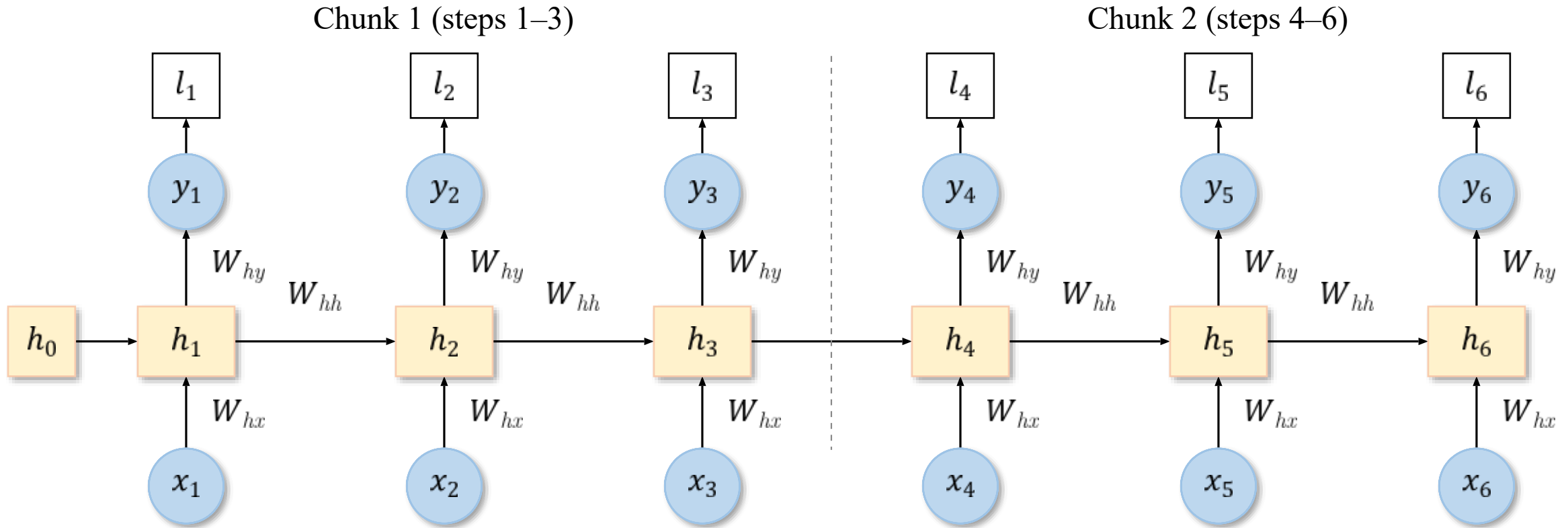
Cell state stores information; hidden state exposes it to the rest of the network.

Outline

- Recurrent Neural Networks (RNNs)
 - Motivations
 - RNN basics
 - Example: Character-level language model
- Back Propagation Through Time (BPTT)
 - Vanishing and exploding gradients
- Long Short-Term Memory (LSTM)
- **Truncated BPTT**

Truncated BPTT

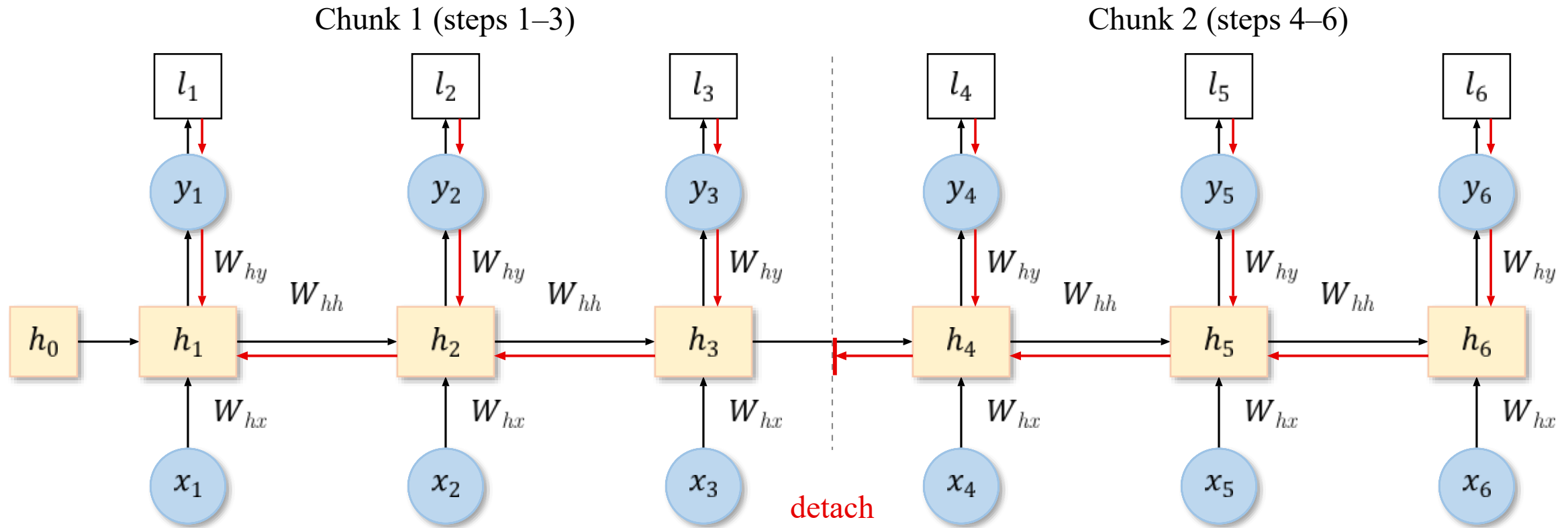
Split a long sequence into chunks; carry the hidden state forward.



Forward pass: reuse the hidden-state value at the boundary.

Truncated BPTT

Split a long sequence into chunks; carry the hidden state forward.



Forward pass: reuse the hidden-state value at the boundary.

Backward pass: stop the gradient at the chunk boundary.

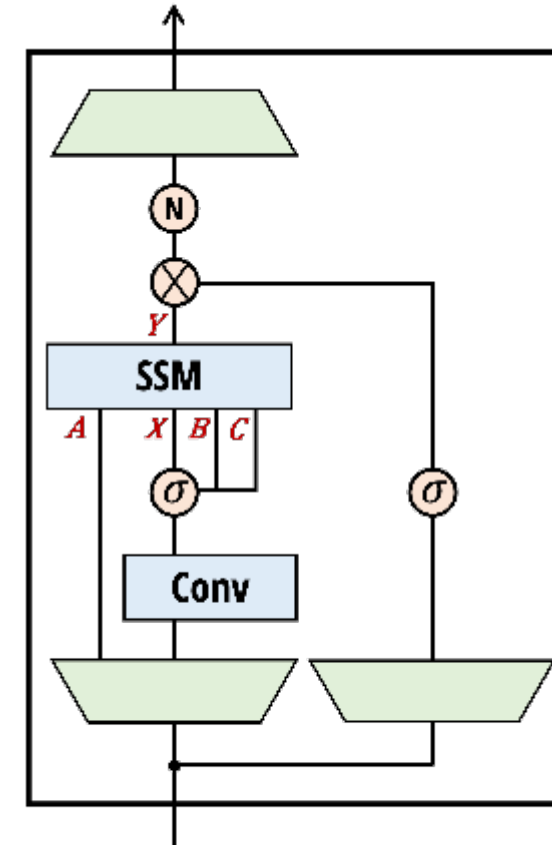
BPTT Summary

- Unroll the recurrence and sum shared-weight gradients.
- Long-range gradients may vanish or explode.
- Gradient clipping limits explosions; LSTM adds gated memory.
- TBPTT carries state values and truncates gradient history.

Recurrence Beyond Vanilla RNNs

Example: Mamba-2 [9]

- Compact recurrent state for streaming.
- Structured algorithms support parallel training.
- Gating and compact state also appear in modern sequence models.



Parallel Mamba Block

SSM = state-space model

References

- [1] <https://www.auroria.io/language-models-are-few-shot-learners-analyzing-gpt-3-meta-learning-nlp/>
- [2] <https://karpathy.github.io/2015/05/21/rnn-effectiveness/>
- [3] Werbos, Paul J.. “Backpropagation Through Time: What It Does and How to Do It.” *Proc. IEEE* 78 (1990): 1550-1560.
- [4] Truncation: Jaeger, H. (2002). Tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF and the “echo state network” approach.
- [5] Randomized Truncation: Tallec, C., & Ollivier, Y. (2017). Unbiasing truncated backpropagation through time. ArXiv:1705.08209.
- [6] <https://d2l.ai/images/rnn-bptt.svg>
- [7] Olah, Christopher. “Understanding LSTM Networks.” 2015.
<https://colah.github.io/posts/2015-08-Understanding-LSTMs/>
- [8] Hochreiter, Sepp, and Jürgen Schmidhuber. “Long Short-Term Memory.” *Neural Computation*, 1997.
<https://doi.org/10.1162/neco.1997.9.8.1735>
- [9] Dao, Tri, and Albert Gu. “Transformers are SSMs: Generalized Models and Efficient Algorithms Through Structured State Space Duality.” ICML, 2024.
<https://arxiv.org/abs/2405.21060>
- [10] Pascanu, Razvan, Tomas Mikolov, and Yoshua Bengio. “On the difficulty of training recurrent neural networks.” ICML, 2013.
<https://proceedings.mlr.press/v28/pascanu13.html>

References

- [11] Moto Hira. “Audio I/O.” TorchAudio 2.5 documentation.
https://docs.pytorch.org/audio/2.5.0/tutorials/audio_io_tutorial.html
- [12] Microsoft. “Florence-2: sample inference notebook.”
https://huggingface.co/microsoft/Florence-2-large/blob/main/sample_inference.ipynb
- [13] Ian Simon and Sageev Oore. “Performance RNN: Generating Music with Expressive Timing and Dynamics.”
Magenta Blog, 2017.
<https://magenta.withgoogle.com/performance-rnn>
- [14] Google Cloud. “Analyze message sentiment.” Agent Assist documentation.
<https://docs.cloud.google.com/agent-assist/docs/sentiment-analysis>
- [15] Quoc V. Le and Mike Schuster. “A Neural Network for Machine Translation, at Production Scale.”
Google Research, 2016.
<https://research.google/blog/a-neural-network-for-machine-translation-at-production-scale/>
- [16] Subhashini Venugopalan et al. “Sequence to Sequence – Video to Text.” ICCV, 2015.
<https://vsubhashini.github.io/s2vt.html>
- [17] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams.
“Learning representations by back-propagating errors.” Nature 323 (1986): 533–536.
<https://doi.org/10.1038/323533a0>
- [18] Magenta. “Melody RNN.” Official model documentation.
https://github.com/magenta/magenta/blob/main/magenta/models/melody_rnn/README.md

Questions?

Appendix: Further Reading

- **Randomized Truncated BPTT**
 - General RNN derivatives
 - Deterministic truncation
 - Randomized estimator and unbiasedness
 - Uniform-cut example
 - Comparison of truncation strategies

BPTT Analysis for general RNNs

RNN layer:

$$\begin{aligned}h_t &= f(x_t, h_{t-1}, w_h), \\ y_t &= g(h_t, w_y),\end{aligned}$$

Hence we have a chain of values:

$$\{\dots, (x_{t-1}, h_{t-1}, y_{t-1}), (x_t, h_t, y_t), \dots\}$$

The objective averages the losses over all time steps:

$$L(x_1, \dots, x_T, \bar{y}_1, \dots, \bar{y}_T, w_h, w_y) = \frac{1}{T} \sum_{t=1}^T l(y_t, \bar{y}_t).$$

BPTT Analysis for general RNNs

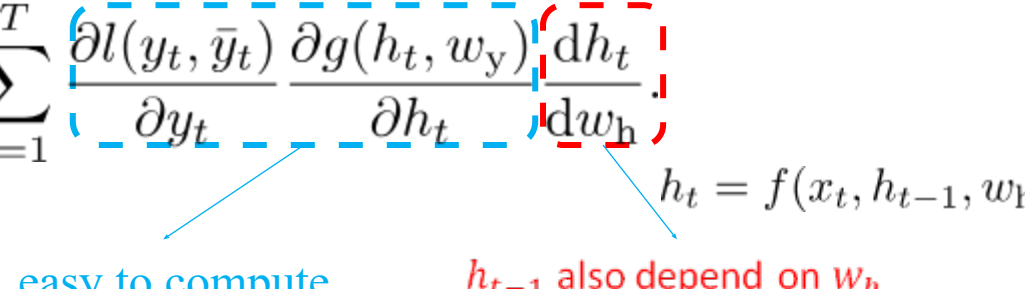
Let's look at backpropagation, by chain rule, we have:

$$\begin{aligned}\frac{dL}{dw_h} &= \frac{1}{T} \sum_{t=1}^T \frac{\partial l(y_t, \bar{y}_t)}{\partial w_h} \\ &= \frac{1}{T} \sum_{t=1}^T \frac{\partial l(y_t, \bar{y}_t)}{\partial y_t} \frac{\partial g(h_t, w_y)}{\partial h_t} \frac{dh_t}{dw_h}.\end{aligned}$$

BPTT Analysis for general RNNs

Let's look at backpropagation, by chain rule, we have:

$$\begin{aligned}\frac{dL}{dw_h} &= \frac{1}{T} \sum_{t=1}^T \frac{\partial l(y_t, \bar{y}_t)}{\partial w_h} \\ &= \frac{1}{T} \sum_{t=1}^T \left[\frac{\partial l(y_t, \bar{y}_t)}{\partial y_t} \frac{\partial g(h_t, w_y)}{\partial h_t} \frac{dh_t}{dw_h} \right]\end{aligned}$$

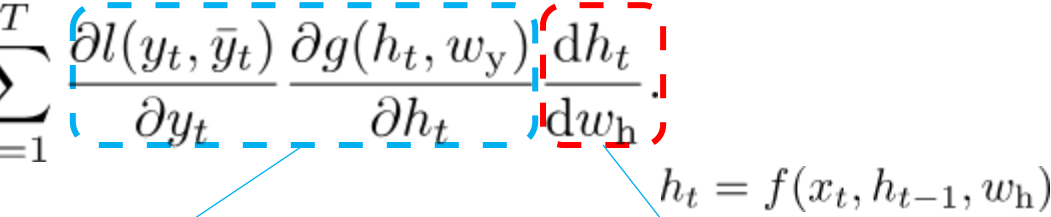

easy to compute h_{t-1} also depend on w_h

$h_t = f(x_t, h_{t-1}, w_h)$

BPTT Analysis for general RNNs

Let's look at backpropagation, by chain rule, we have:

$$\begin{aligned}\frac{dL}{dw_h} &= \frac{1}{T} \sum_{t=1}^T \frac{\partial l(y_t, \bar{y}_t)}{\partial w_h} \\ &= \frac{1}{T} \sum_{t=1}^T \left[\frac{\partial l(y_t, \bar{y}_t)}{\partial y_t} \frac{\partial g(h_t, w_y)}{\partial h_t} \frac{dh_t}{dw_h} \right]\end{aligned}$$



easy to compute h_{t-1} also depend on w_h

$$\frac{dh_t}{dw_h} = \frac{\partial f(x_t, h_{t-1}, w_h)}{\partial w_h} + \frac{\partial f(x_t, h_{t-1}, w_h)}{\partial h_{t-1}} \frac{dh_{t-1}}{dw_h}.$$

BPTT Analysis for general RNNs

Let's simplify the notation:

$$a_t = \frac{dh_t}{dw_h}, \quad B_t = \frac{\partial f(x_t, h_{t-1}, w_h)}{\partial w_h}, \quad J_t = \frac{\partial f(x_t, h_{t-1}, w_h)}{\partial h_{t-1}},$$
$$a_t = B_t + J_t a_{t-1}, \quad a_0 = 0.$$

Hold the initial state fixed when differentiating the shared parameters.

BPTT Analysis for general RNNs

Let's simplify the notation:

$$a_t = \frac{dh_t}{dw_h}, \quad B_t = \frac{\partial f(x_t, h_{t-1}, w_h)}{\partial w_h}, \quad J_t = \frac{\partial f(x_t, h_{t-1}, w_h)}{\partial h_{t-1}},$$
$$a_t = B_t + J_t a_{t-1}, \quad a_0 = 0.$$

Hold the initial state fixed when differentiating the shared parameters.

$$a_t = B_t + \sum_{i=1}^{t-1} J_t J_{t-1} \cdots J_{i+1} B_i.$$

BPTT Analysis for general RNNs

Let's simplify the notation:

$$a_t = \frac{dh_t}{dw_h}, \quad B_t = \frac{\partial f(x_t, h_{t-1}, w_h)}{\partial w_h}, \quad J_t = \frac{\partial f(x_t, h_{t-1}, w_h)}{\partial h_{t-1}},$$
$$a_t = B_t + J_t a_{t-1}, \quad a_0 = 0.$$

Hold the initial state fixed when differentiating the shared parameters.

$$a_1 = B_1,$$

$$a_2 = B_2 + J_2 B_1,$$

$$a_3 = B_3 + J_3 B_2 + J_3 J_2 B_1.$$

BPTT Analysis for general RNNs

Let's simplify the notation:

$$a_t = \frac{dh_t}{dw_h}, \quad B_t = \frac{\partial f(x_t, h_{t-1}, w_h)}{\partial w_h}, \quad J_t = \frac{\partial f(x_t, h_{t-1}, w_h)}{\partial h_{t-1}},$$
$$a_t = B_t + J_t a_{t-1}, \quad a_0 = 0.$$

Hold the initial state fixed when differentiating the shared parameters.

$$a_t = B_t + \sum_{i=1}^{t-1} J_t J_{t-1} \cdots J_{i+1} B_i.$$

For vector states, the newest Jacobian is on the left; the multiplication order matters.

BPTT Analysis for general RNNs

$$a_t = B_t + \sum_{i=1}^{t-1} J_t J_{t-1} \cdots J_{i+1} B_i.$$

Explicitly expanding all temporal paths is costly; BPTT reuses intermediate gradients.

Long sequences require more stored activations, and distant gradient contributions can grow or decay.

Truncated BPTT

Keep the most recent τ terms in the hidden-state sensitivity.

$$\hat{a}_t^{(\tau)} = B_t + \sum_{i=\max(1, t-\tau+1)}^{t-1} J_t \cdots J_{i+1} B_i, \quad \tau \geq 1.$$

Treat the earlier boundary state as constant: carry its value forward, but stop the gradient there.

This saves computation and activation storage, but omits dependencies crossing the boundary.

Randomized Truncated BPTT

Choose independent truncation multipliers while keeping the forward trajectory fixed.

$$P(\xi_t = 0) = 1 - \pi_t, \quad P(\xi_t = \pi_t^{-1}) = \pi_t, \quad 0 < \pi_t \leq 1.$$

A zero multiplier cuts the recurrent gradient; surviving paths receive a compensation factor.

$$\hat{a}_t = B_t + \sum_{i=1}^{t-1} \left(\prod_{j=i+1}^t \xi_j \right) J_t J_{t-1} \cdots J_{i+1} B_i.$$

The random estimate is denoted with a hat; the exact derivative has no hat.

Randomized Truncated BPTT

With independent multipliers, the random estimate is correct in expectation.

$$P(\xi_t = 0) = 1 - \pi_t, \quad P(\xi_t = \pi_t^{-1}) = \pi_t, \quad 0 < \pi_t \leq 1.$$

$$\mathbb{E}[\xi_t] = 1, \quad \mathbb{E}[\xi_i \xi_j] = \mathbb{E}[\xi_i] \mathbb{E}[\xi_j] \quad (i \neq j).$$

$$\begin{aligned} \mathbb{E}[\hat{a}_t] &= \mathbb{E} \left[B_t + \sum_{i=1}^{t-1} \left(\prod_{j=i+1}^t \xi_j \right) J_t \cdots J_{i+1} B_i \right] \\ &= B_t + \sum_{i=1}^{t-1} \mathbb{E} \left[\prod_{j=i+1}^t \xi_j \right] J_t \cdots J_{i+1} B_i \\ &= B_t + \sum_{i=1}^{t-1} \left(\prod_{j=i+1}^t \mathbb{E}[\xi_j] \right) J_t \cdots J_{i+1} B_i = a_t. \end{aligned}$$

Randomized Truncated BPTT

Choose one cut uniformly; let the mask indicate which earlier contributions survive.

$$K \sim \text{Uniform}\{1, \dots, T\}, \quad m_i = \mathbf{1}_{\{K \leq i\}},$$
$$P(m_i = 1) = \frac{i}{T}, \quad \mathbb{E}[m] = \left[\frac{1}{T}, \frac{2}{T}, \dots, \frac{T-1}{T} \right].$$

Earlier contributions are less likely to survive the cut.

The mask entries share the same cut and are correlated.

Randomized Truncated BPTT

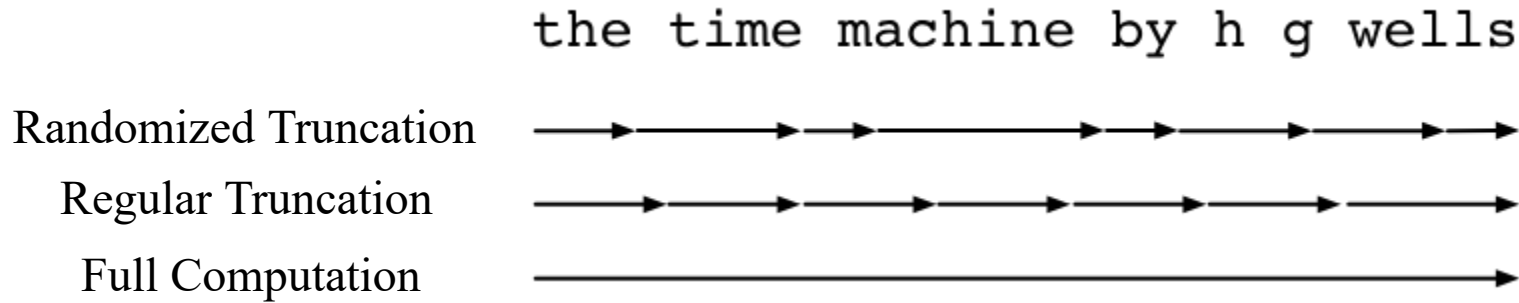
Rescale surviving contributions by their inverse inclusion probabilities.

$$K \sim \text{Uniform}\{1, \dots, T\}, \quad m_i = \mathbf{1}_{\{K \leq i\}}, \quad c_i = \frac{T}{i}, \quad \mathbb{E}[c_i m_i] = 1.$$

$$\begin{aligned} \mathbb{E}[\hat{a}_T] &= \mathbb{E} \left[B_T + \sum_{i=1}^{T-1} c_i m_i J_T \cdots J_{i+1} B_i \right] \\ &= B_T + \sum_{i=1}^{T-1} \mathbb{E}[c_i m_i] J_T \cdots J_{i+1} B_i \\ &= \frac{dh_T}{dw_h} = a_T. \end{aligned}$$

Only the marginal inclusion probabilities are needed for this expectation.

Comparing Strategies



- Randomized truncation that partitions the text into segments of varying lengths.
- Regular truncation that breaks the text into subsequences of the same length. This is what we have been doing in RNN experiments.
- Full BPTT retains the complete unrolled graph, requiring more activation storage for long sequences.