APEX 数据和知识管理实验室
DATA & KNOWLEDGE MANAGEMENT LAB

Recurrent Neural Network Introduction

Lantao Yu
2016 / 7 / 21

Outline

- Background
- Deep Learning Models
- Training

Sharing Parameters

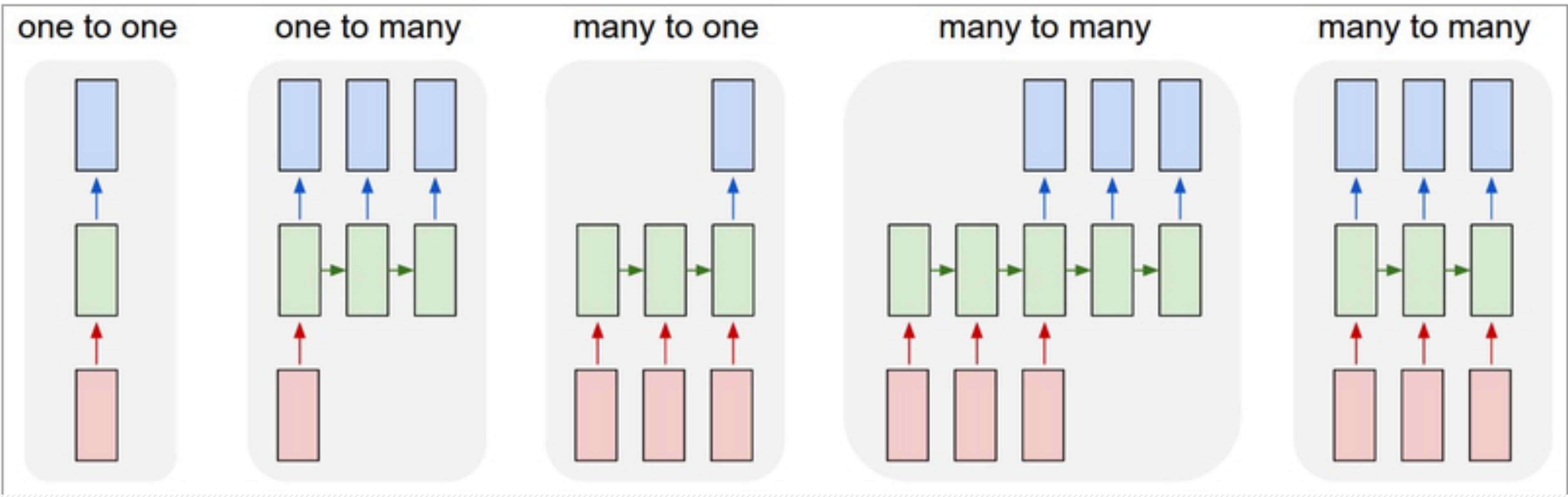
- RNN shares the same weights across several time steps

- makes it possible to extend and apply the model to examples of different forms(different lengths, here)

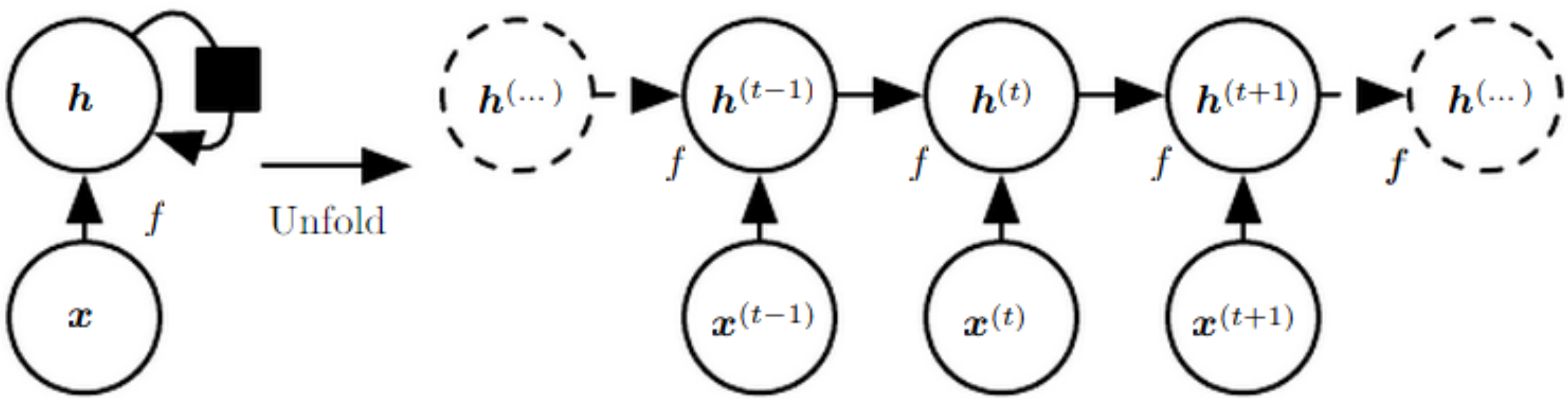
- Generalize across different forms of data

- Less parameters

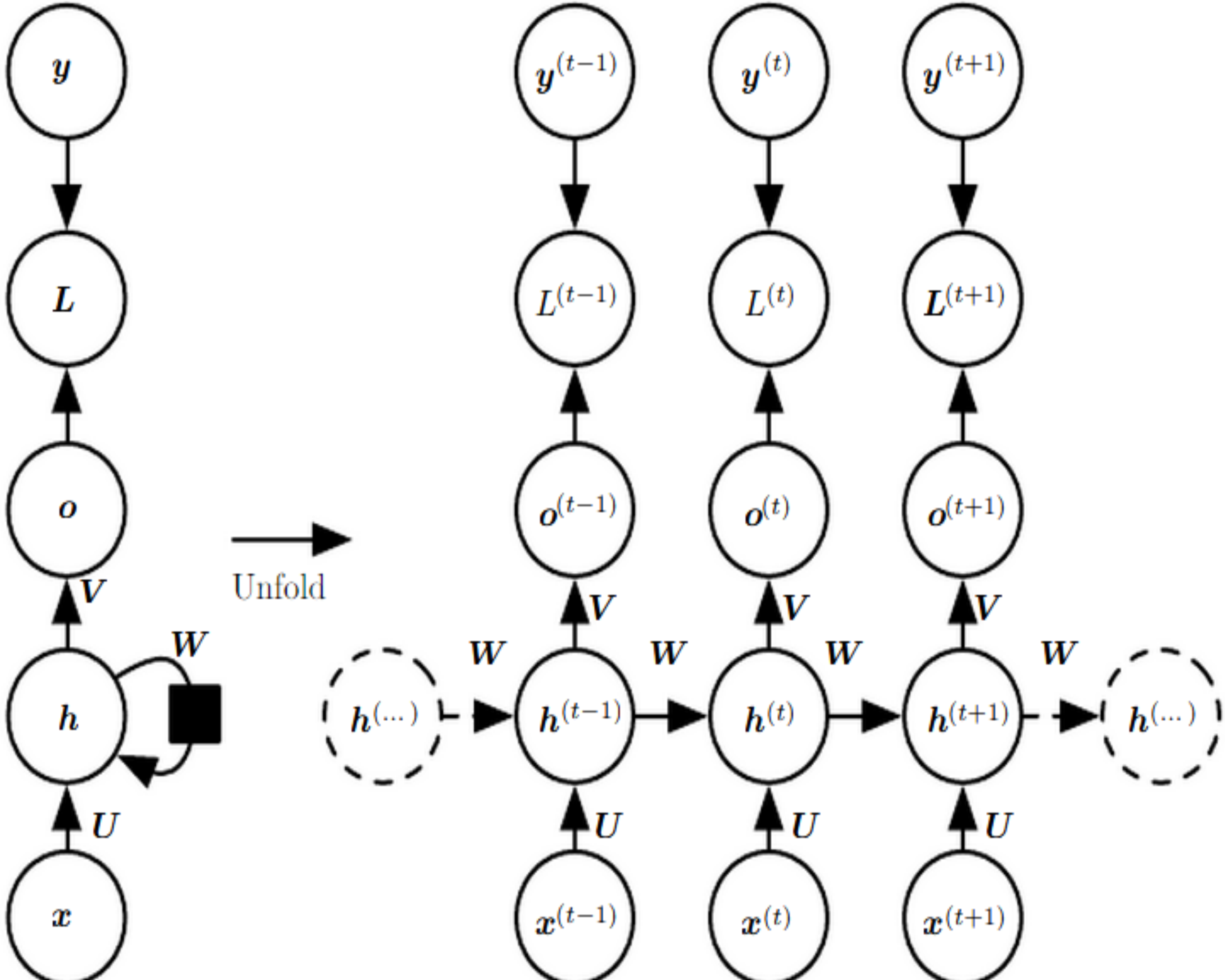
RNN offer a lot of flexibility



Vanilla Recurrent Networks without output

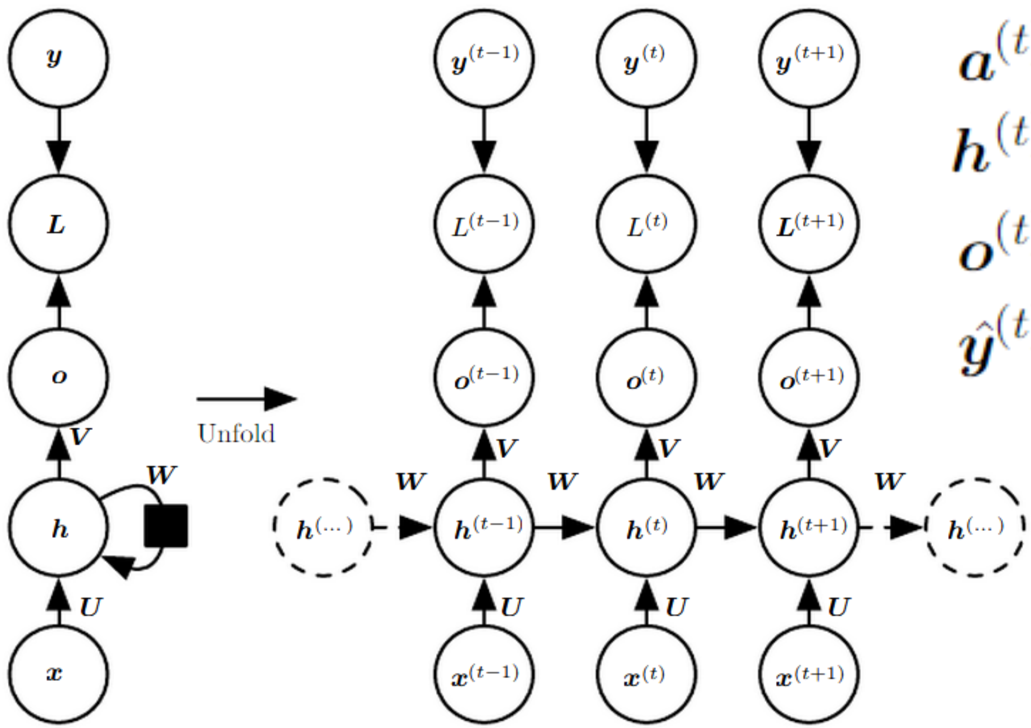


$$
\begin{aligned}
\boldsymbol{h}^{(t)} &= g^{(t)}(\boldsymbol{x}^{(t)}, \boldsymbol{x}^{(t-1)}, \boldsymbol{x}^{(t-2)}, \dots, \boldsymbol{x}^{(2)}, \boldsymbol{x}^{(1)}) \\
&= f(\boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t)}; \boldsymbol{\theta})
\end{aligned}
$$



y
L
o
V
W
h
U
x
Unfold
y^(t−1)
y^(t)
y^(t+1)
L^(t−1)
L^(t)
L^(t+1)
o^(t−1)
o^(t)
o^(t+1)
V
V
V
W
W
W
W
h^(...)
h^(t−1)
h^(t)
h^(t+1)
h^(...)
U
U
U
x^(t−1)
x^(t)
x^(t+1)

Vanilla Recurrent Networks



$$
\begin{aligned}
\boldsymbol{a}^{(t)} &= \boldsymbol{b} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t)} \\
\boldsymbol{h}^{(t)} &= \tanh(\boldsymbol{a}^{(t)}) \\
\boldsymbol{o}^{(t)} &= \boldsymbol{c} + \boldsymbol{V}\boldsymbol{h}^{(t)} \\
\hat{\boldsymbol{y}}^{(t)} &= \operatorname{softmax}(\boldsymbol{o}^{(t)})
\end{aligned}
$$

Negative log-likelyhood loss

$$L\left(\{\boldsymbol{x}^{(1)},\ldots,\boldsymbol{x}^{(\tau)}\},\{\boldsymbol{y}^{(1)},\ldots,\boldsymbol{y}^{(\tau)}\}\right)$$

$$=\sum_t L^{(t)}$$

$$=-\sum_t \log p_{\text{model}}\left(y^{(t)} \mid \{\boldsymbol{x}^{(1)},\ldots,\boldsymbol{x}^{(t)}\}\right),$$

Computing the Gradient

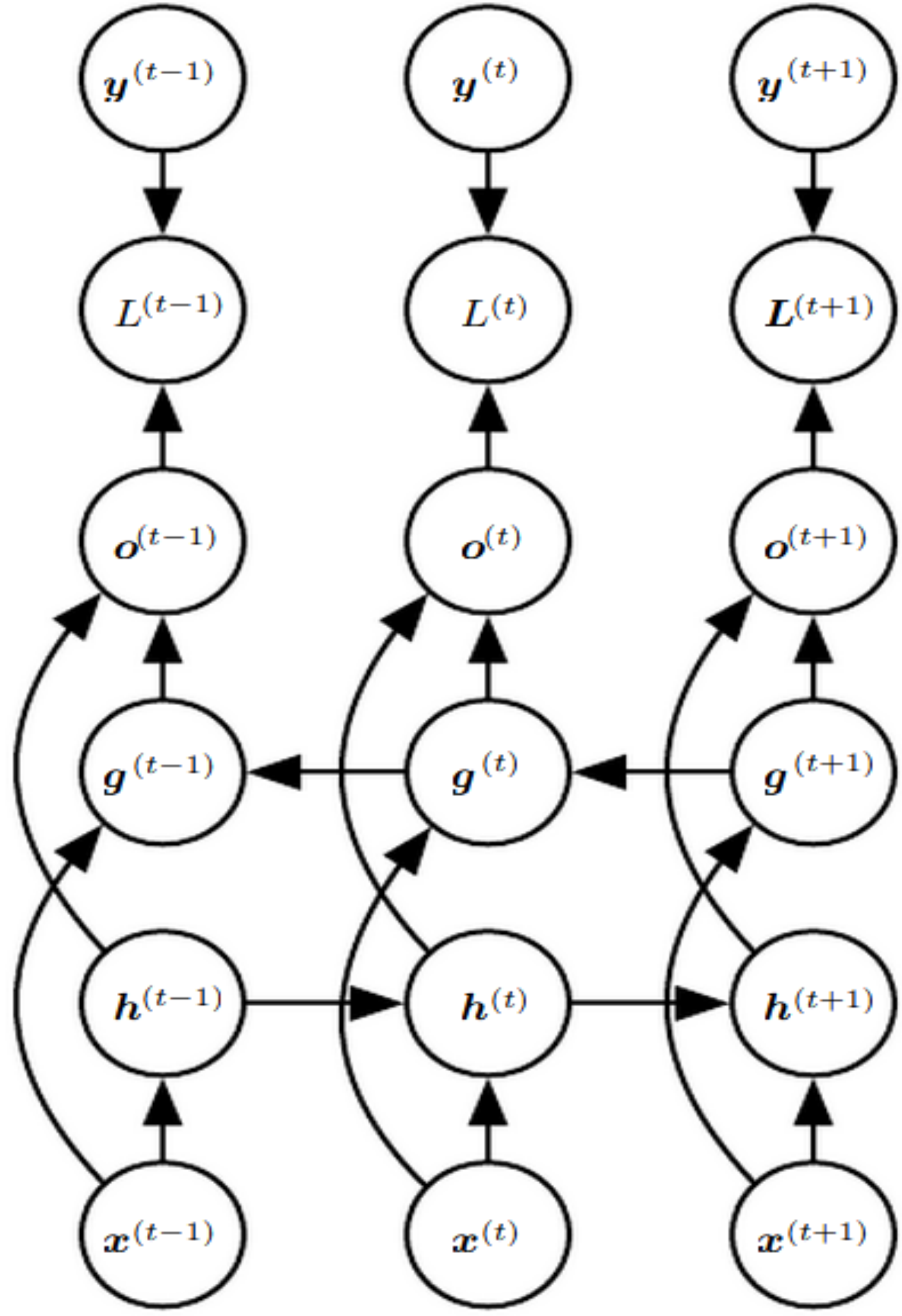
$$(\nabla_{\boldsymbol{o}^{(t)}} L)_i = \frac{\partial L}{\partial o_i^{(t)}} = \frac{\partial L}{\partial L^{(t)}} \frac{\partial L^{(t)}}{\partial o_i^{(t)}} = \hat{y}_i^{(t)} - \mathbf{1}_{i,y^{(t)}}.$$

$$\nabla_{\boldsymbol{h}^{(t)}} L = \left(\frac{\partial \boldsymbol{h}^{(t+1)}}{\partial \boldsymbol{h}^{(t)}}\right)^\top (\nabla_{\boldsymbol{h}^{(t+1)}} L) + \left(\frac{\partial \boldsymbol{o}^{(t)}}{\partial \boldsymbol{h}^{(t)}}\right)^\top (\nabla_{\boldsymbol{o}^{(t)}} L)$$

$$= \boldsymbol{W}^\top (\nabla_{\boldsymbol{h}^{(t+1)}} L) \operatorname{diag}\left(1 - \left(\boldsymbol{h}^{(t+1)}\right)^2\right) + \boldsymbol{V}^\top (\nabla_{\boldsymbol{o}^{(t)}} L)$$

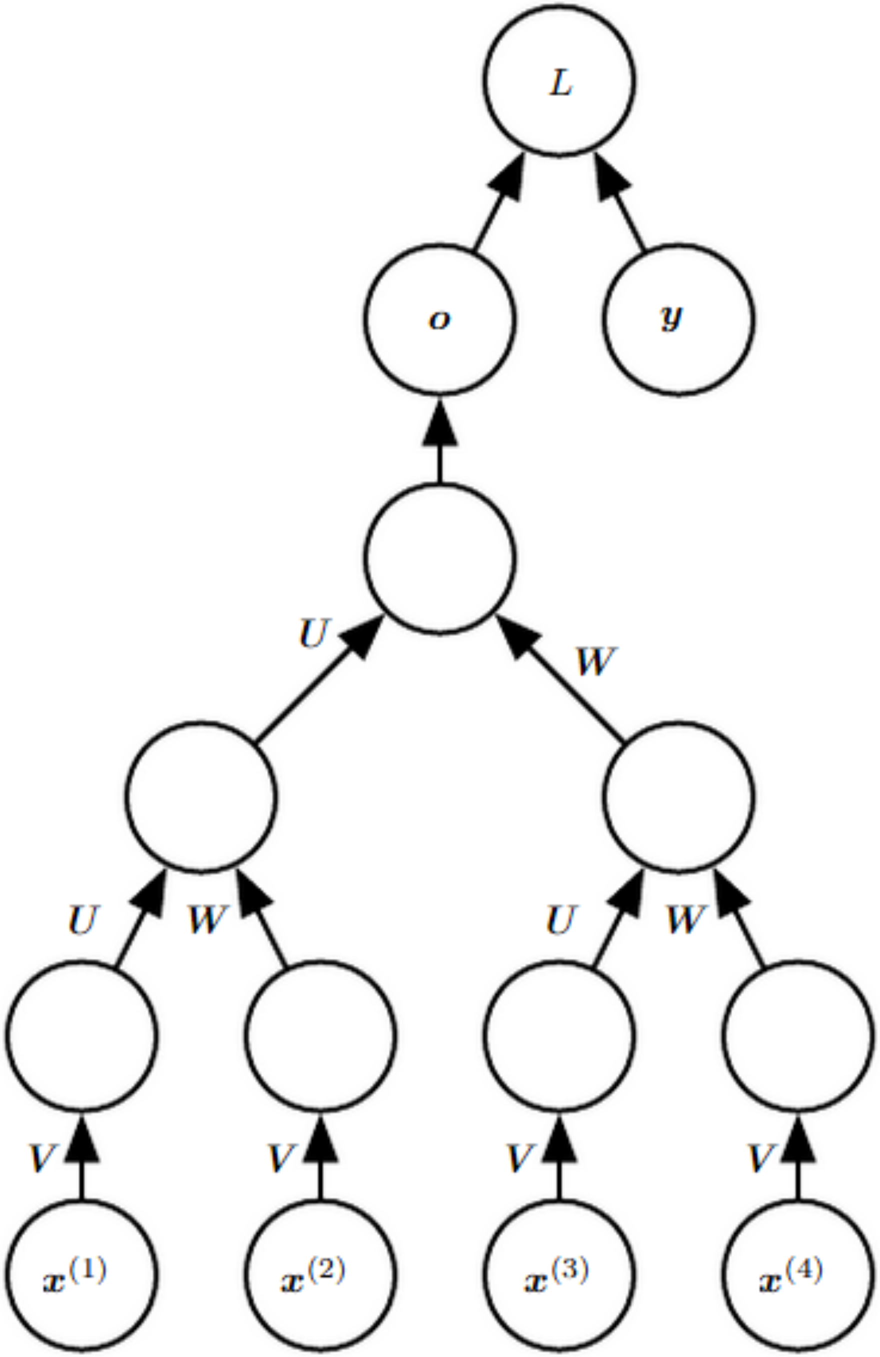
$$
\begin{aligned}
\nabla_{\boldsymbol{c}} L &= \sum_t \left(\frac{\partial \boldsymbol{o}^{(t)}}{\partial \boldsymbol{c}}\right)^\top \nabla_{\boldsymbol{o}^{(t)}} L = \sum_t \nabla_{\boldsymbol{o}^{(t)}} L \\
\nabla_{\boldsymbol{b}} L &= \sum_t \left(\frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{b}^{(t)}}\right)^\top \nabla_{\boldsymbol{h}^{(t)}} L = \sum_t \operatorname{diag}\left(1 - \left(\boldsymbol{h}^{(t)}\right)^2\right) \nabla_{\boldsymbol{h}^{(t)}} L \\
\nabla_{\boldsymbol{V}} L &= \sum_t \sum_i \left(\frac{\partial L}{\partial o_i^{(t)}}\right) \nabla_{\boldsymbol{V}} o_i^{(t)} = \sum_t \left(\nabla_{\boldsymbol{o}^{(t)}} L\right) \boldsymbol{h}^{(t)^\top} \\
\nabla_{\boldsymbol{W}} L &= \sum_t \sum_i \left(\frac{\partial L}{\partial h_i^{(t)}}\right) \nabla_{\boldsymbol{W}^{(t)}} h_i^{(t)} \\
&= \sum_t \operatorname{diag}\left(1 - \left(\boldsymbol{h}^{(t)}\right)^2\right) \left(\nabla_{\boldsymbol{h}^{(t)}} L\right) \boldsymbol{h}^{(t-1)^\top} \\
\nabla_{\boldsymbol{U}} L &= \sum_t \sum_i \left(\frac{\partial L}{\partial h_i^{(t)}}\right) \nabla_{\boldsymbol{U}^{(t)}} h_i^{(t)} \\
&= \sum_t \operatorname{diag}\left(1 - \left(\boldsymbol{h}^{(t)}\right)^2\right) \left(\nabla_{\boldsymbol{h}^{(t)}} L\right) \boldsymbol{x}^{(t)^\top}
\end{aligned}
$$

Bi-directional RNN



Recursive Networks

A variable-size sequence x(1),x(2), . . . , x(t) can be mapped to a fixed-size representation (the output o)



Long Short-Term Memory

Examples:

Predict the last word in the text :

"I grew up in France

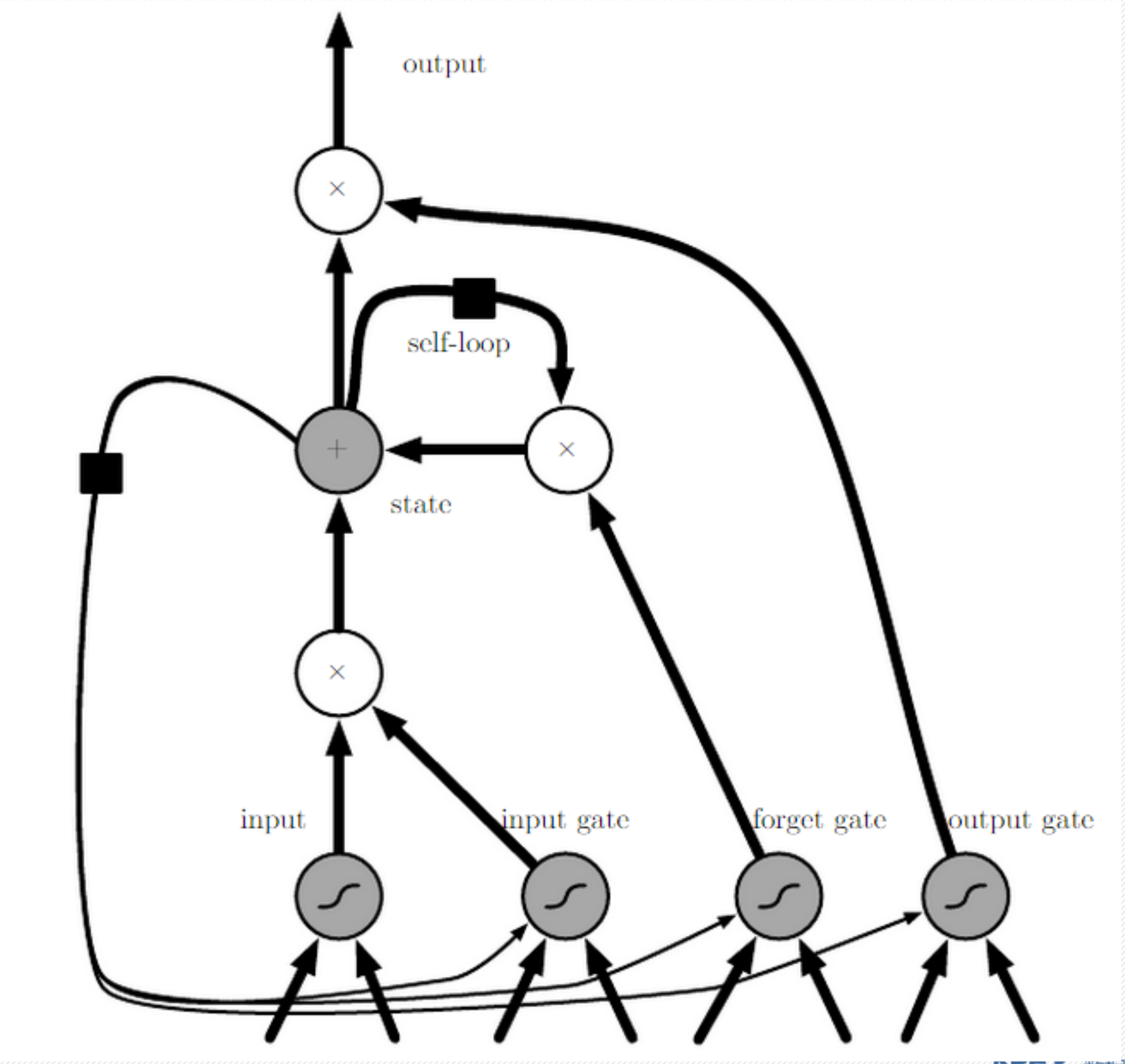
...

...

...

I speak fluent French."

Long


output
self-loop
state
input
input gate
forget gate
output gate

LSTM Functions

Forget Gate

$$f_i^{(t)} = \sigma\left(b_i^f + \sum_j U_{i,j}^f x_j^{(t)} + \sum_j W_{i,j}^f h_j^{(t-1)}\right)$$

External Input Gate

$$g_i^{(t)} = \sigma\left(b_i^g + \sum_j U_{i,j}^g x_j^{(t)} + \sum_j W_{i,j}^g h_j^{(t-1)}\right)$$

Output Gate

$$q_i^{(t)} = \sigma\left(b_i^o + \sum_j U_{i,j}^o x_j^{(t)} + \sum_j W_{i,j}^o h_j^{(t-1)}\right)$$

LSTM Functions

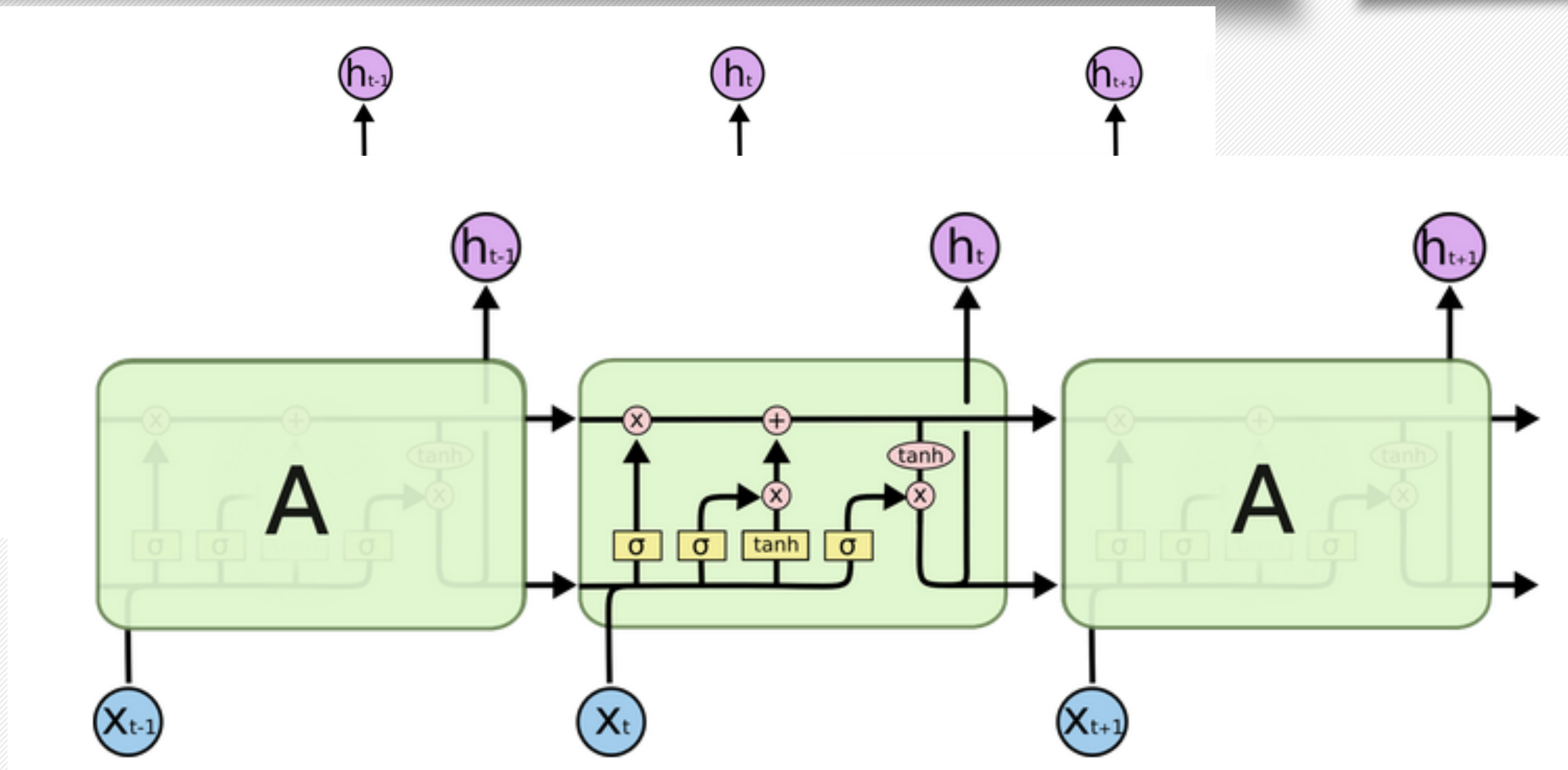
Internal Memory cell

$$s_i^{(t)} = f_i^{(t)} s_i^{(t-1)} + g_i^{(t)} \sigma \left(b_i + \sum_j U_{i,j} x_j^{(t)} + \sum_j W_{i,j} h_j^{(t-1)} \right)$$

Output

$$h_i^{(t)} = \tanh\left(s_i^{(t)}\right) q_i^{(t)}$$

The core idea of LSTM



Gated Recurrent Units

- Update gate:

$$u_i^{(t)} = \sigma\left(b_i^u + \sum_j U_{i,j}^u x_j^{(t)} + \sum_j W_{i,j}^u h_j^{(t)}\right)$$

- Reset gate:

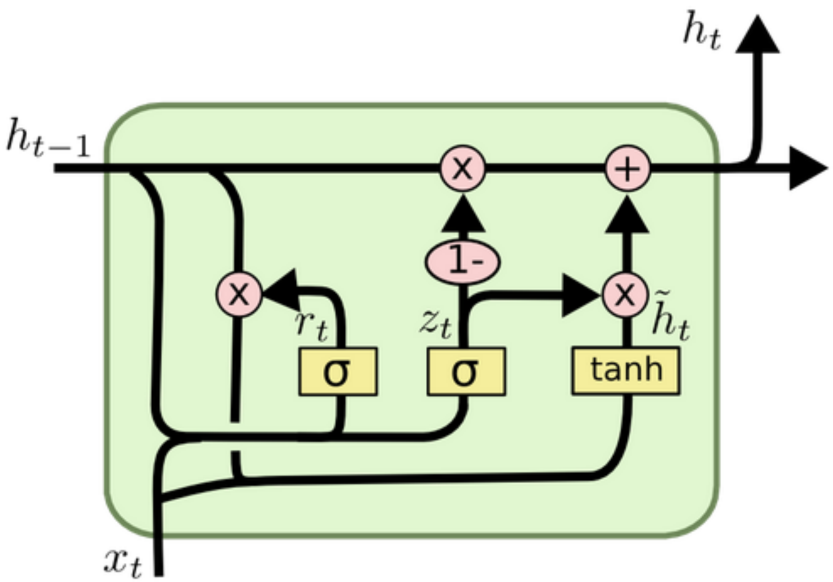
$$r_i^{(t)} = \sigma\left(b_i^r + \sum_j U_{i,j}^r x_j^{(t)} + \sum_j W_{i,j}^r h_j^{(t)}\right)$$

Gated Recurrent Units

Update Equation

$$h_i^{(t)} = u_i^{(t-1)} h_i^{(t-1)} + (1 - u_i^{(t-1)}) \sigma \left(b_i + \sum_j U_{i,j} x_j^{(t-1)} + \sum_j W_{i,j} r_j^{(t-1)} h_j^{(t-1)} \right)$$

Gated Recurrent Units



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

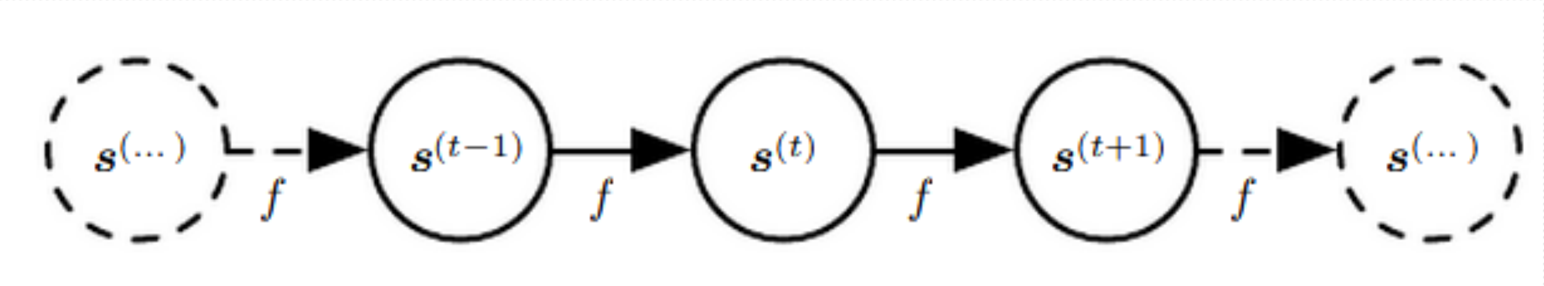
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

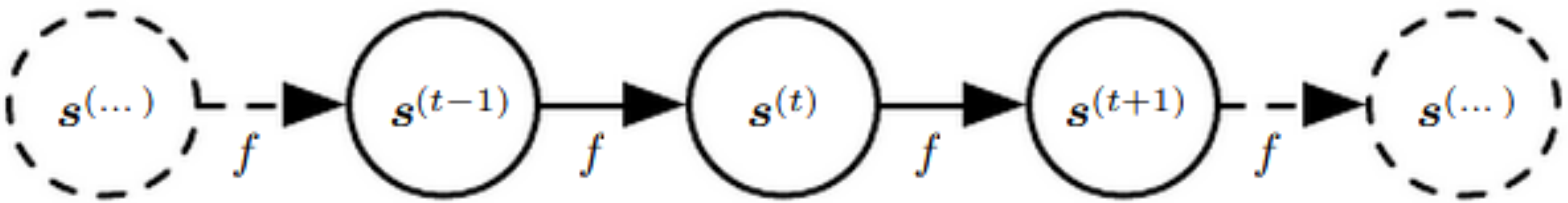
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Long Term Dependency

- Example:





$$\boldsymbol{h}^{(t)} = \boldsymbol{W}^{\top}\boldsymbol{h}^{(t-1)}$$

$$\boldsymbol{h}^{(t)} = \left(\boldsymbol{W}^{t}\right)^{\top}\boldsymbol{h}^{(0)}$$

$$\boldsymbol{W}^{t} = \left(\boldsymbol{V}\mathrm{diag}(\boldsymbol{\lambda})\boldsymbol{V}^{-1}\right)^{t} = \boldsymbol{V}\mathrm{diag}(\boldsymbol{\lambda})^{t}\boldsymbol{V}^{-1}$$

$$\boldsymbol{h}^{(t)} = \boldsymbol{Q}^{\top}\boldsymbol{\Lambda}^{t}\boldsymbol{Q}\boldsymbol{h}^{(0)}$$

Clipping Gradients

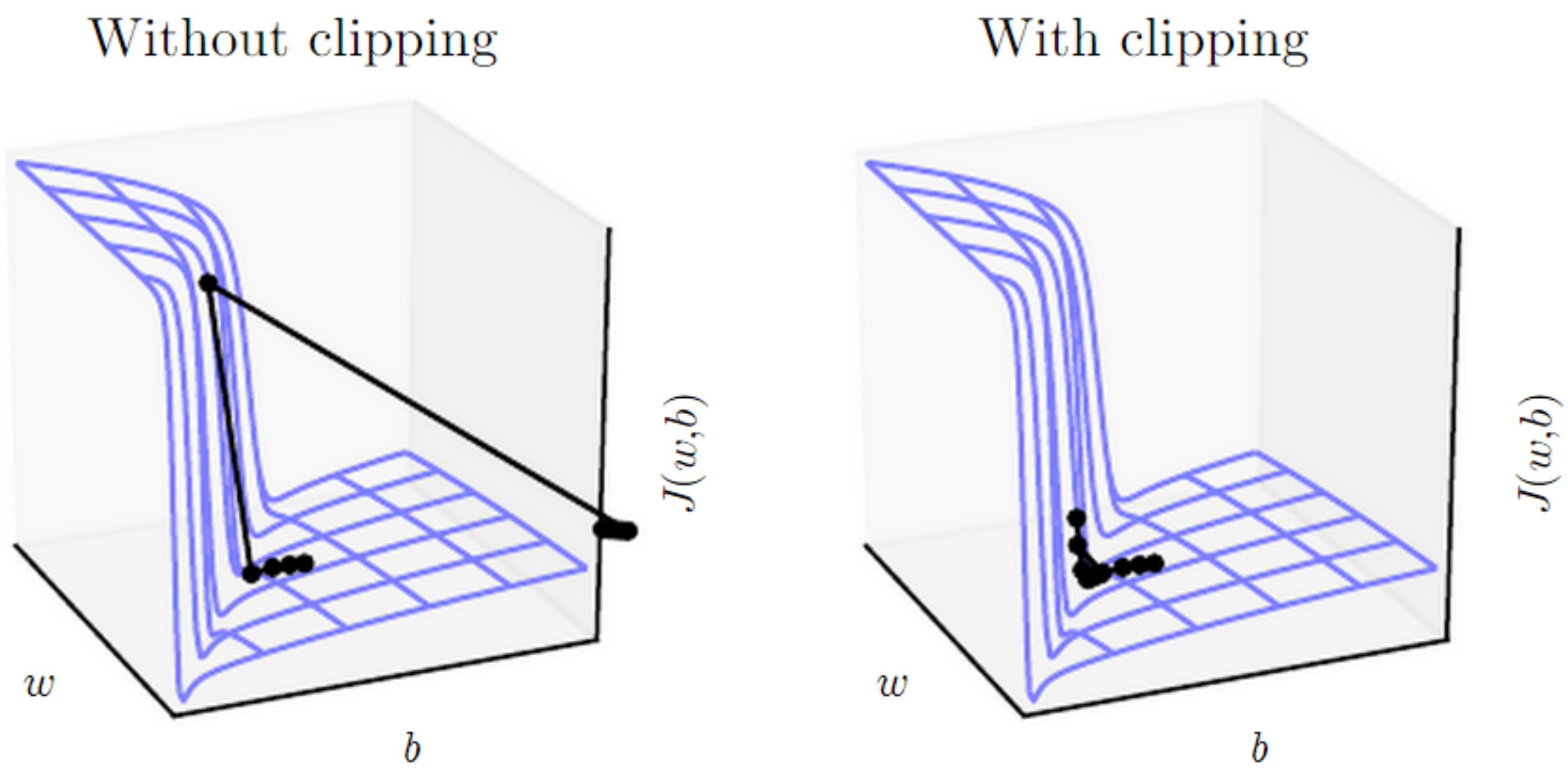
- Element-wise clipping

- clip the norm $||\boldsymbol{g}||$

$$\text{if } ||\boldsymbol{g}|| > v$$

$$\boldsymbol{g} \leftarrow \frac{\boldsymbol{g}v}{||\boldsymbol{g}||}$$

Clipping Gradients



RNN Research

- **Attention Mechanism**
- **Grid LSTM**
- **Generative Models**
- **...**

Impleme

```
# Input Gate
i = tf.sigmoid(
    tf.matmul(x, self.Wi) +
    tf.matmul(previous_hidden_state, self.Ui) + self.bi
)

# Forget Gate
f = tf.sigmoid(
    tf.matmul(x, self.Wf) +
    tf.matmul(previous_hidden_state, self.Uf) + self.bf
)

# Output Gate
o = tf.sigmoid(
    tf.matmul(x, self.Wog) +
    tf.matmul(previous_hidden_state, self.Uog) + self.bog
)

# New Memory Cell
c_ = tf.nn.tanh(
    tf.matmul(x, self.Wc) +
    tf.matmul(previous_hidden_state, self.Uc) + self.bc
)

# Final Memory cell
c = f * c_prev + i * c_

# Current Hidden state
current_hidden_state = o * tf.nn.tanh(c)
```

Summary

- RNNs allow a lot of flexibility in architecture design
- Vanilla RNNs are simple but don’t work very well
- Common to use LSTM or GRU: their additive interactions improve gradient flow
- Backward flow of gradients in RNN can explode or vanish. Exploding is controlled with gradient clipping. Vanishing is controlled with additive interactions (LSTM)
- Better/simpler architectures are a hot topic of current research
- Better understanding (both theoretical and empirical) is needed.