CS639 Deep Learning for NLP
Recurrent Neural Networks

Junjie Hu

WISCONSIN
UNIVERSITY OF WISCONSIN–MADISON

Slides adapted from Graham, Sergey, Chris
https://junjiehu.github.io/cs639-spring26/

Goals for Today

- Review: Sequential Data

- **Recurrent Neural Network**

- **Vanishing/Exploding Gradients & LSTM**

- Applications of RNN

- Variants of RNN

NLP and Sequential Data

- NLP is full of sequential data

  - Words in sentences

  - Characters in words

  - Sentences in discourse

  - ...

Long-distance Dependencies in Language

- Agreement in number, gender, etc.

He does not have very much confidence in **himself**.
She does not have very much confidence in **herself**.

- Selectional preference

The **reign** has lasted as long as the life of the **queen**.
The **rain** has lasted as long as the life of the **clouds**.

Can be Complicated!

- What is the referent of "it"?

The trophy would not fit in the brown suitcase because it was too **big**.

Trophy

The trophy would not fit in the brown suitcase because it was too **small**.

Suitcase

(from Winograd Schema Challenge:
http://commonsensereasoning.org/winograd.html)

Recurrent Neural Networks

What if we have variable-size inputs?

$$x_1 = (x_{1,1}, x_{1,2}, x_{1,3}, x_{1,4})$$

$$x_2 = (x_{2,1}, x_{2,2}, x_{2,3})$$

$$x_3 = (x_{3,1}, x_{3,2}, x_{3,3}, x_{3,4}, x_{3,5})$$

- Simple idea: zero-pad up to length of longest sequence

$$x_1 = (x_{1,1}, x_{1,2}, x_{1,3}, x_{1,4}, 0)$$

Concatenate word vectors

transform

predict

label

Feedforward Networks
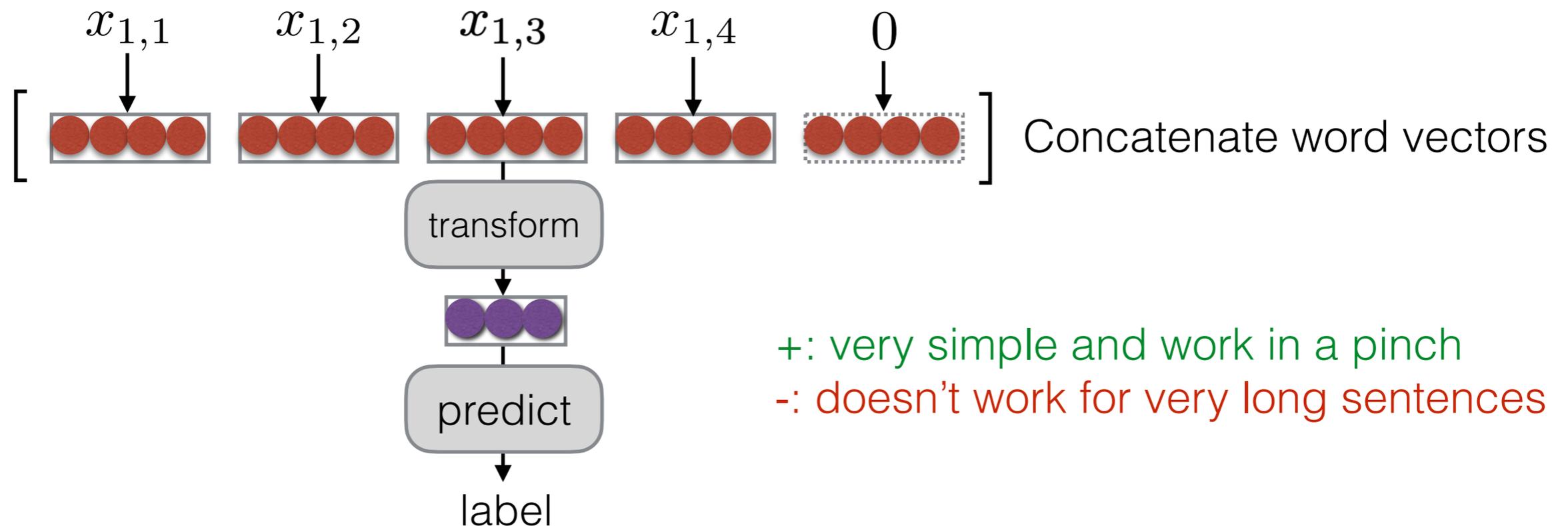
+: very simple and work in a pinch
-: doesn't work for very long sentences

One input per layer?

$$x_1 = (x_{1,1}, x_{1,2}, x_{1,3}, x_{1,4})$$

$$x_2 = (x_{2,1}, x_{2,2}, x_{2,3})$$

$$x_3 = (x_{3,1}, x_{3,2}, x_{3,3}, x_{3,4}, x_{3,5})$$

- Another idea: feed one word to each feedforward layer
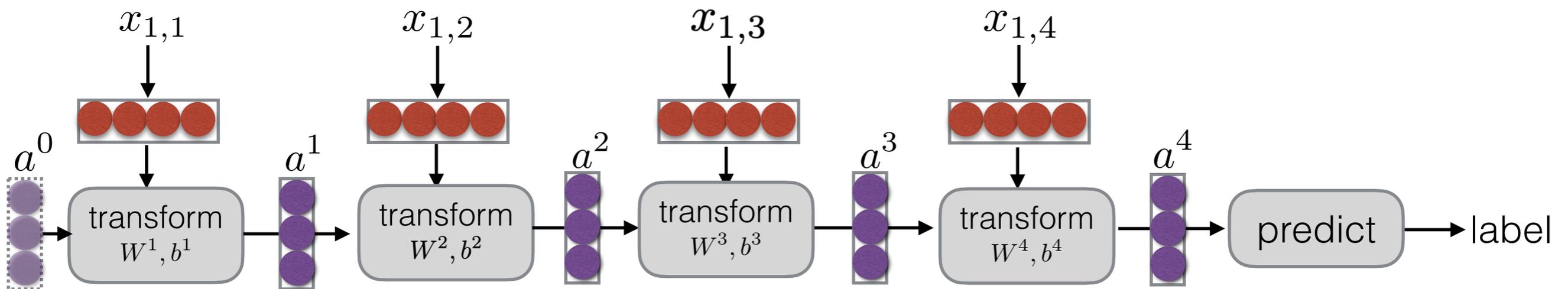
$$\bar{a}^{\ell-1} = \begin{bmatrix} a^{\ell-1} \\ x_{i,t} \end{bmatrix} \qquad z^\ell = W^\ell \bar{a}^{\ell-1} + b^\ell \qquad a^\ell = \sigma(z^\ell)$$
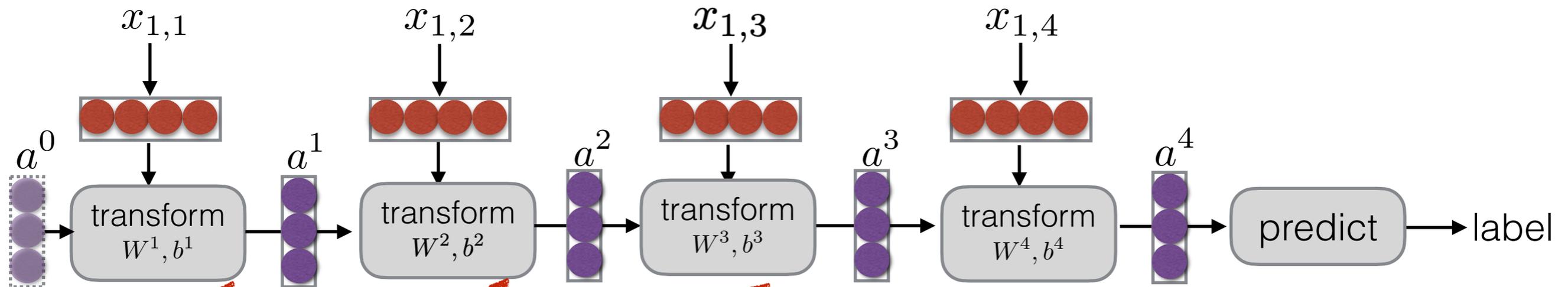
+: do not need to pad sentences
+: each FF layer is much smaller than the giant FF layer for concatenated inputs
+: shorter sentence requires fewer layers
-: Obvious question: what happens to the long sentences?

8

Can we share weight matrics?

$$\bar{a}^{\ell-1} = \begin{bmatrix} a^{\ell-1} \\ x_{i,t} \end{bmatrix} \qquad z^{\ell} = W^{\ell}\bar{a}^{\ell-1} + b^{\ell} \qquad a^{\ell} = \sigma(z^{\ell}) \qquad W = W^{\ell}, b = b^{\ell}, \ \forall \ell$$

+: we can have as many "layers" as we want!

This is called a **recurrent** neural network (RNN) — "variable-depth" network.
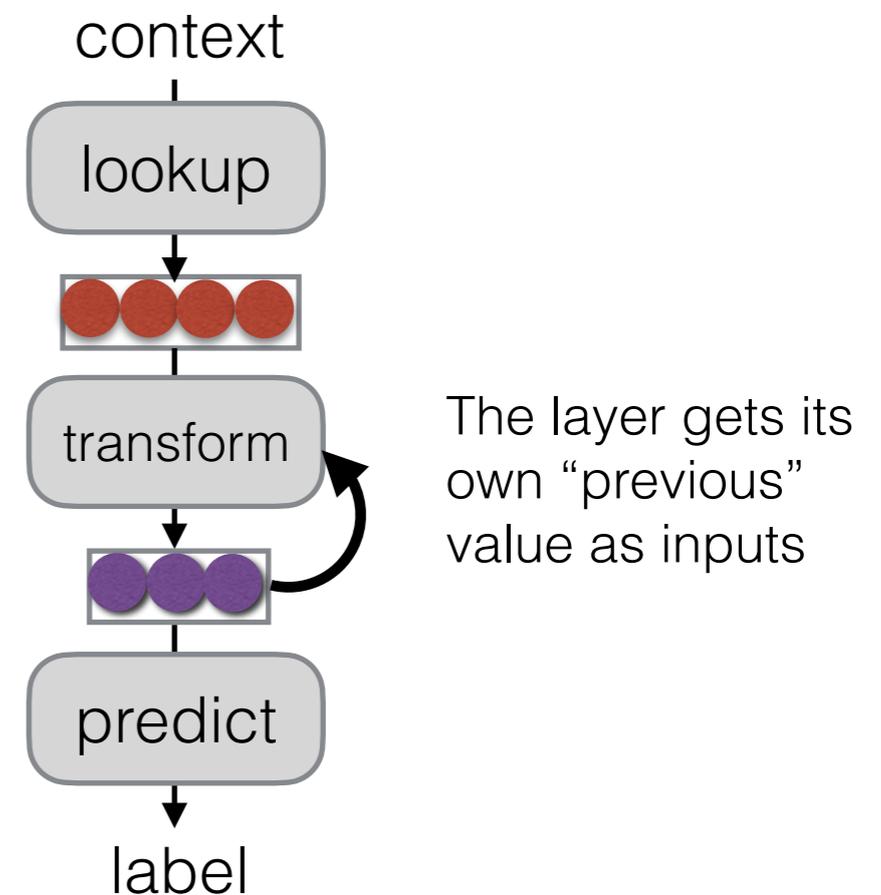
Recurrent Neural Networks
(Elman 1990)

- Tools to "remember" information along the "time dimension"

- RNNs are just NNs that share weights across layers, take an input each layer, and have a variable number of layers.

Feed-forward NN

context

lookup

transform

predict

label

Recurrent NN

context

lookup

transform

predict

label

The layer gets its own "previous" value as inputs
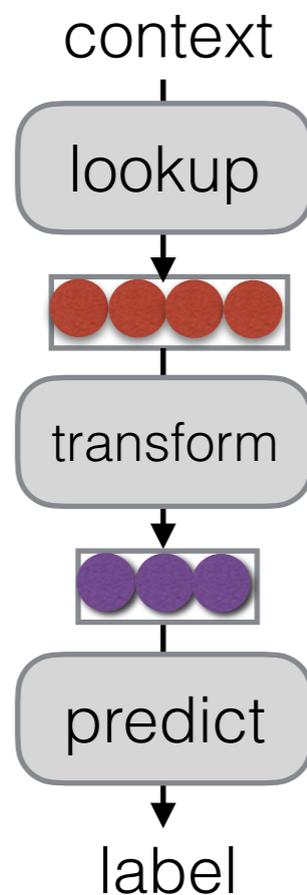
Unrolling in Time

- What does processing a sequence look like?
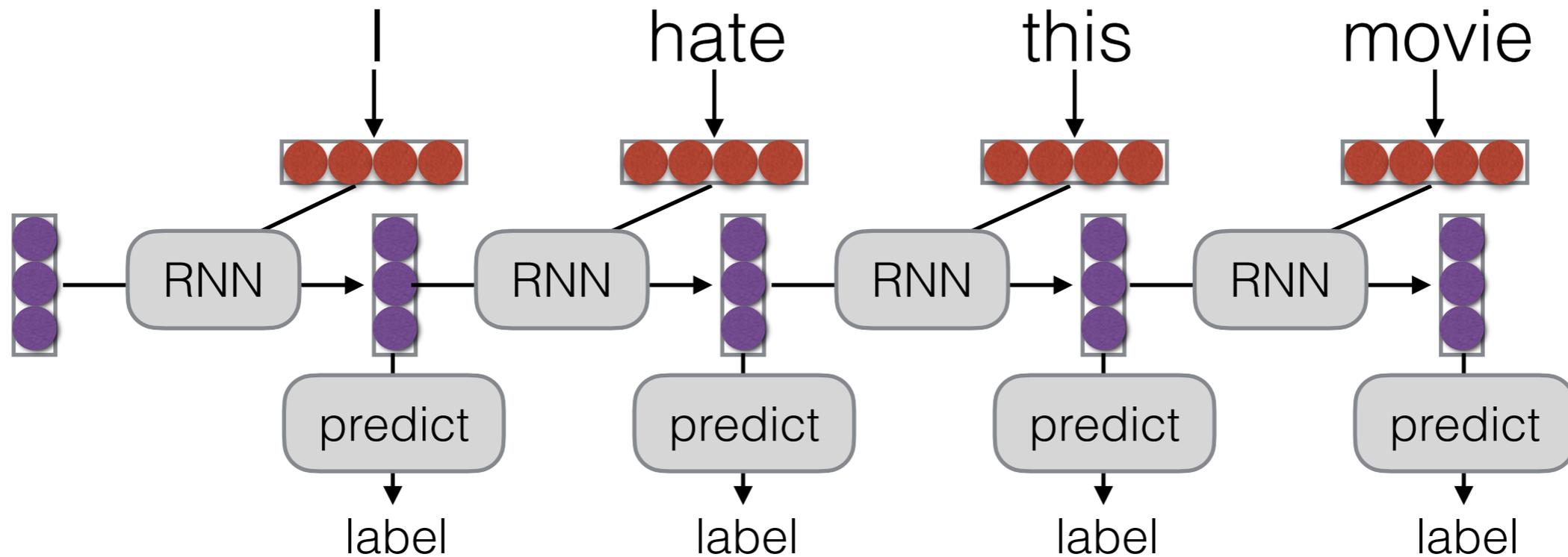
Training RNNs
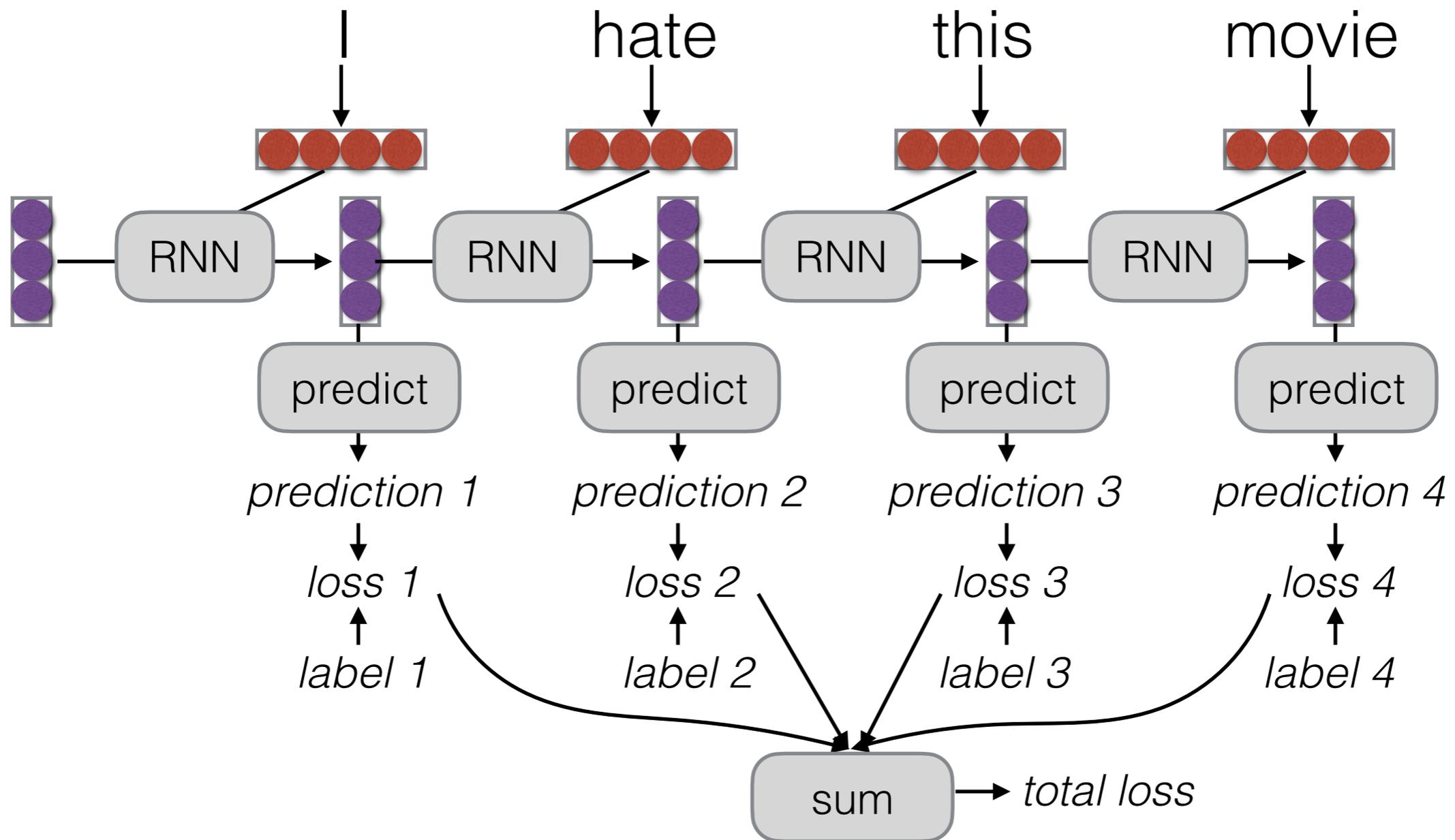
RNN Training

- The unrolled graph is a well-formed directed acyclic graph (DAG)—we can run backprop

total loss
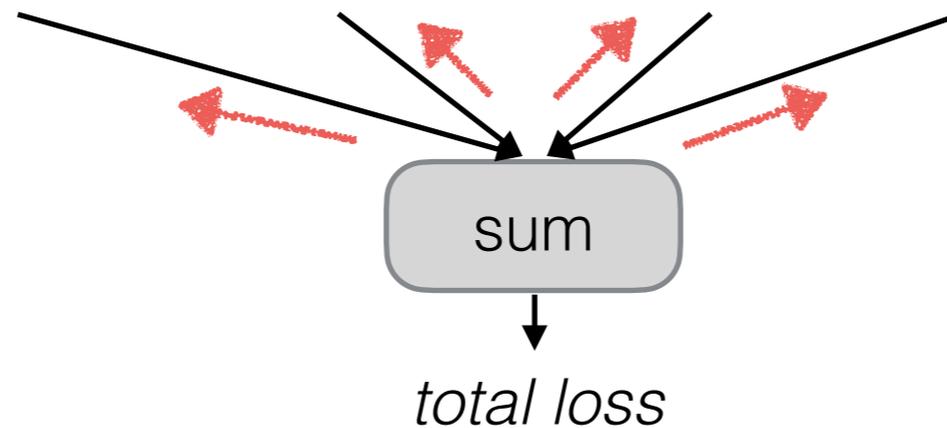
- Parameters are tied across time, derivatives are aggregated across all time steps

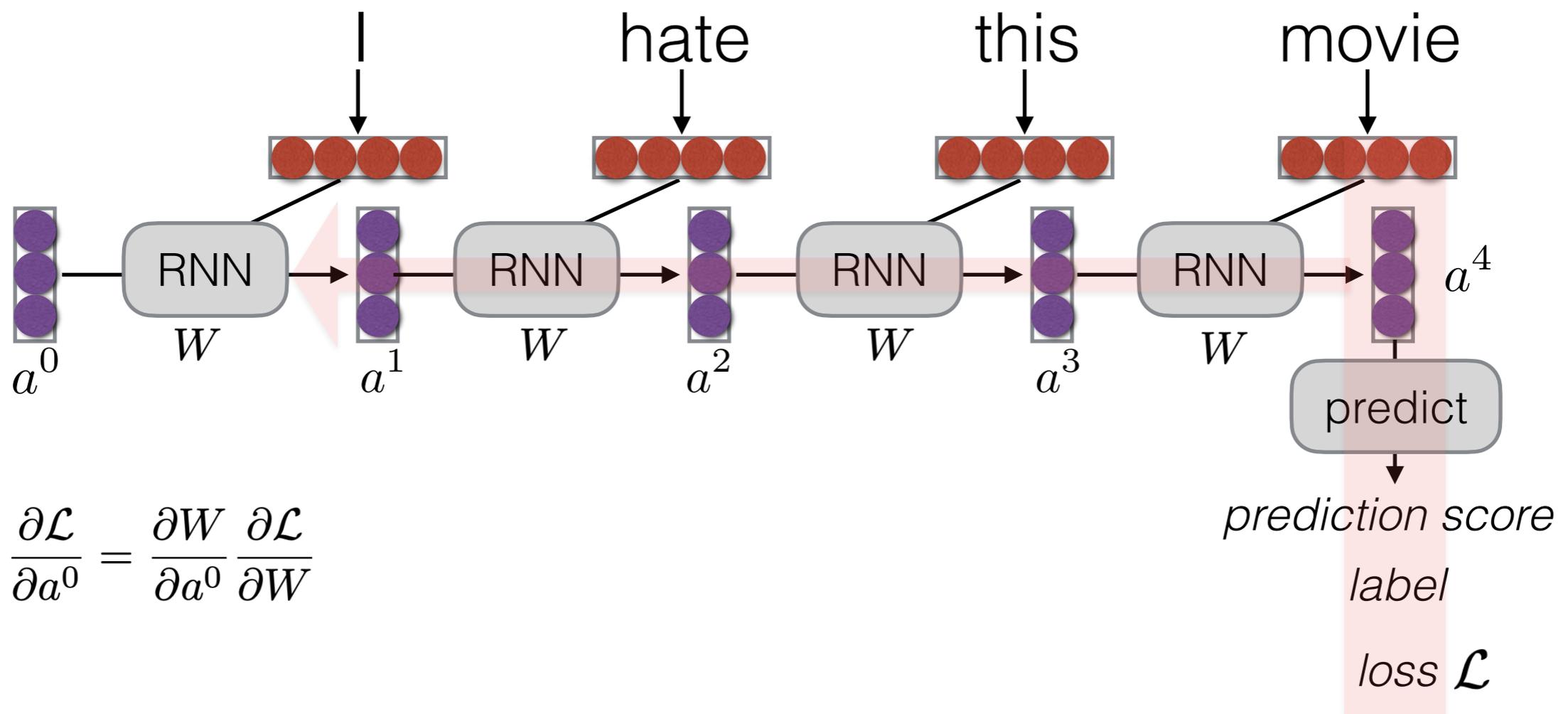- This is historically called "backpropagation through time" (BPTT)
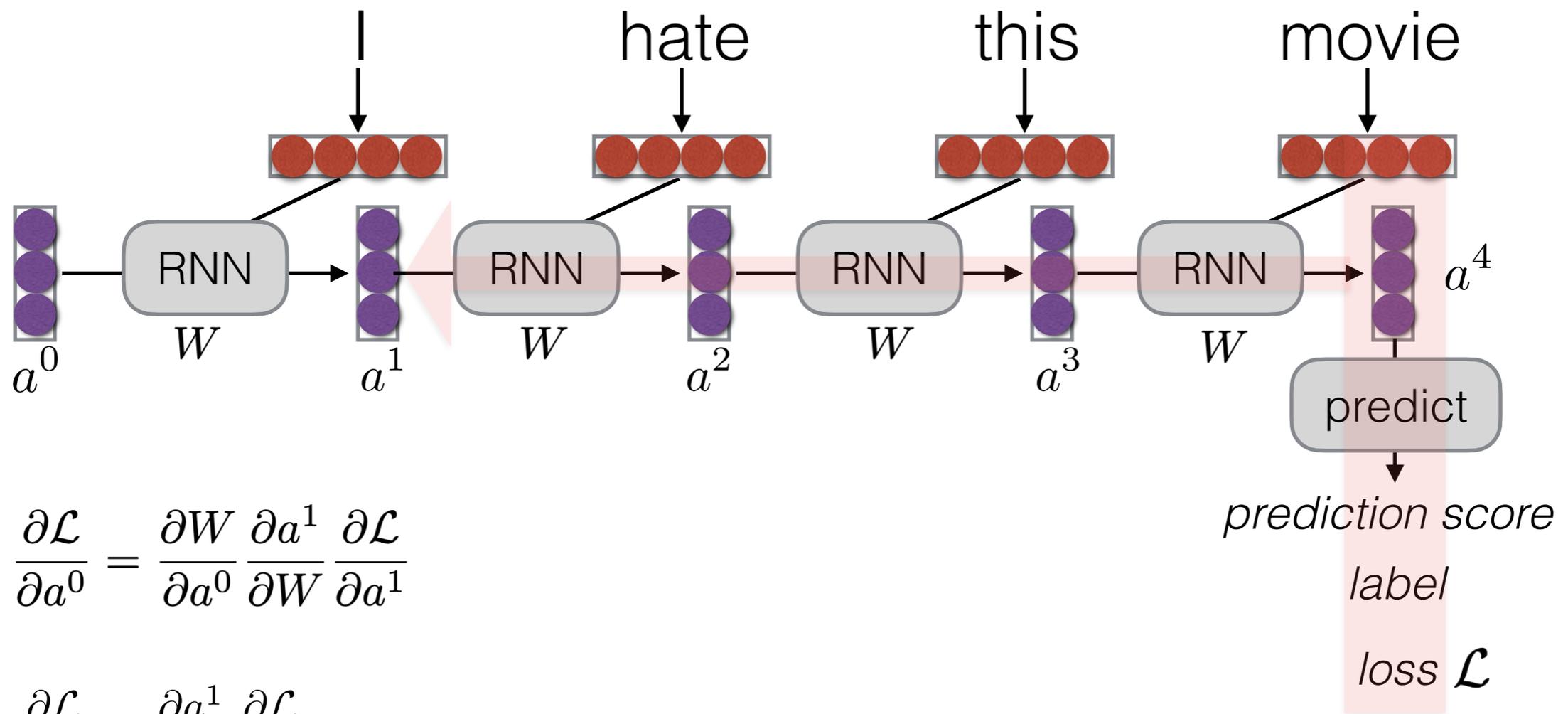
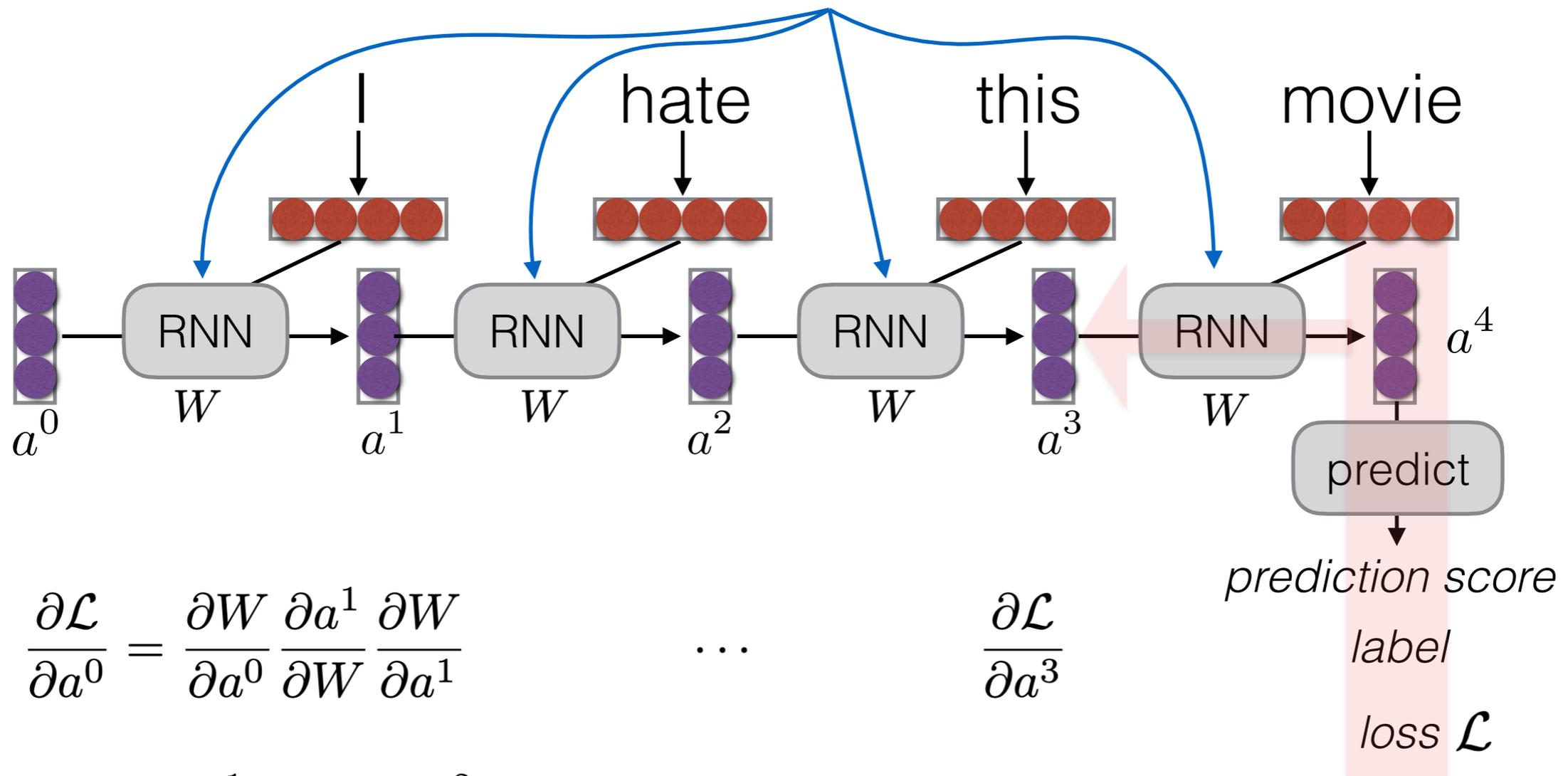# Forward pass

Parameters *W* are shared!

Backward pass

$$\frac{\partial \mathcal{L}}{\partial a^0} = ?$$
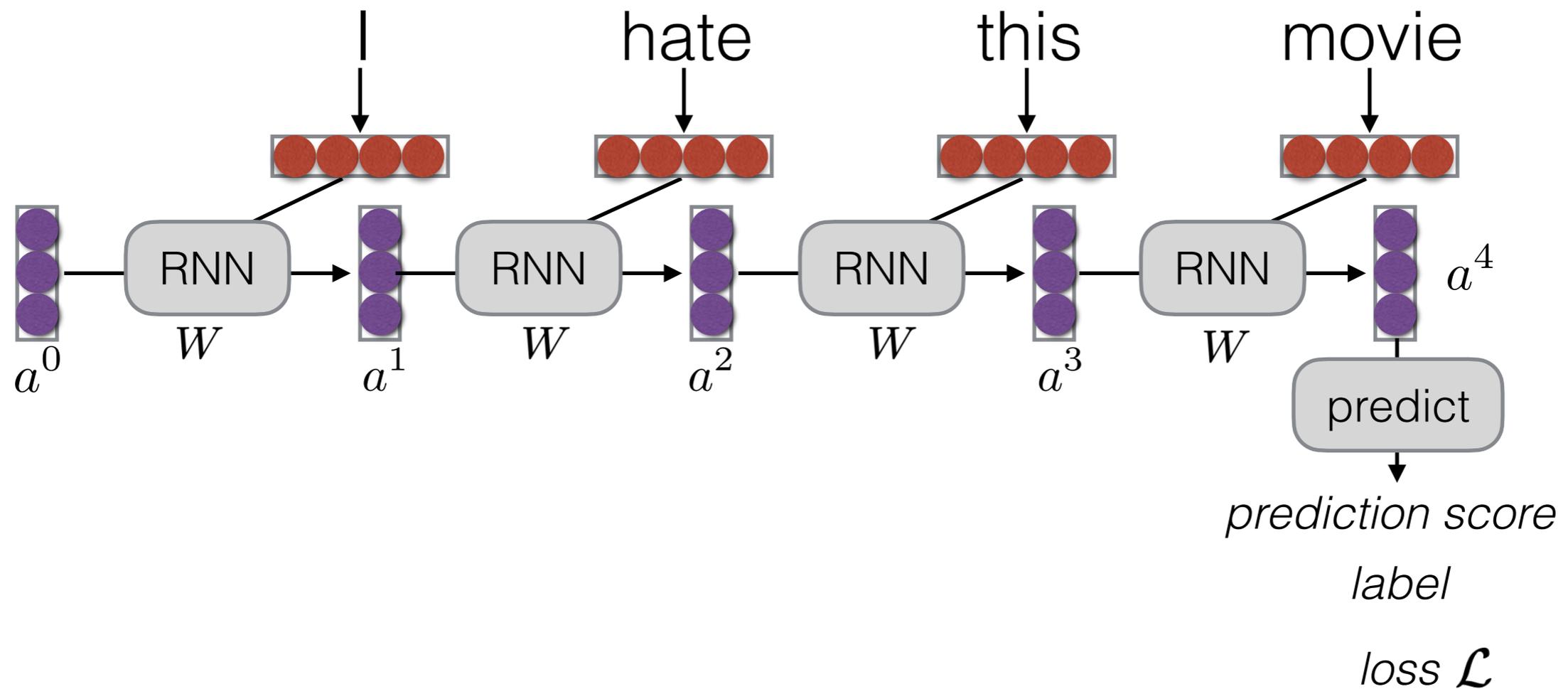
Backward pass
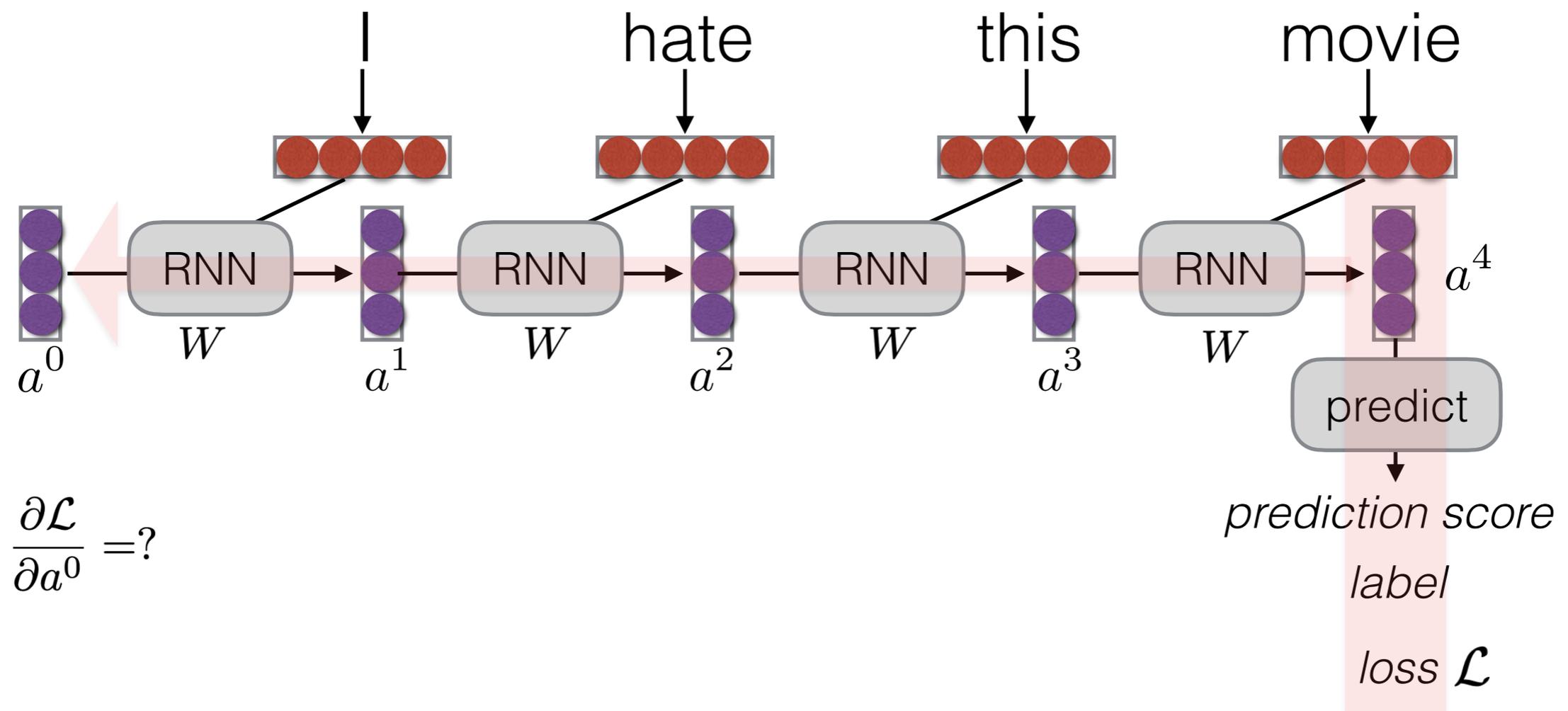
$$\frac{\partial \mathcal{L}}{\partial a^0} = \frac{\partial W}{\partial a^0} \frac{\partial \mathcal{L}}{\partial W}$$

Backward pass

$$\frac{\partial \mathcal{L}}{\partial a^0} = \frac{\partial W}{\partial a^0} \frac{\partial a^1}{\partial W} \frac{\partial \mathcal{L}}{\partial a^1}$$

$$\frac{\partial \mathcal{L}}{\partial W} = \frac{\partial a^1}{\partial W} \frac{\partial \mathcal{L}}{\partial a^1}$$
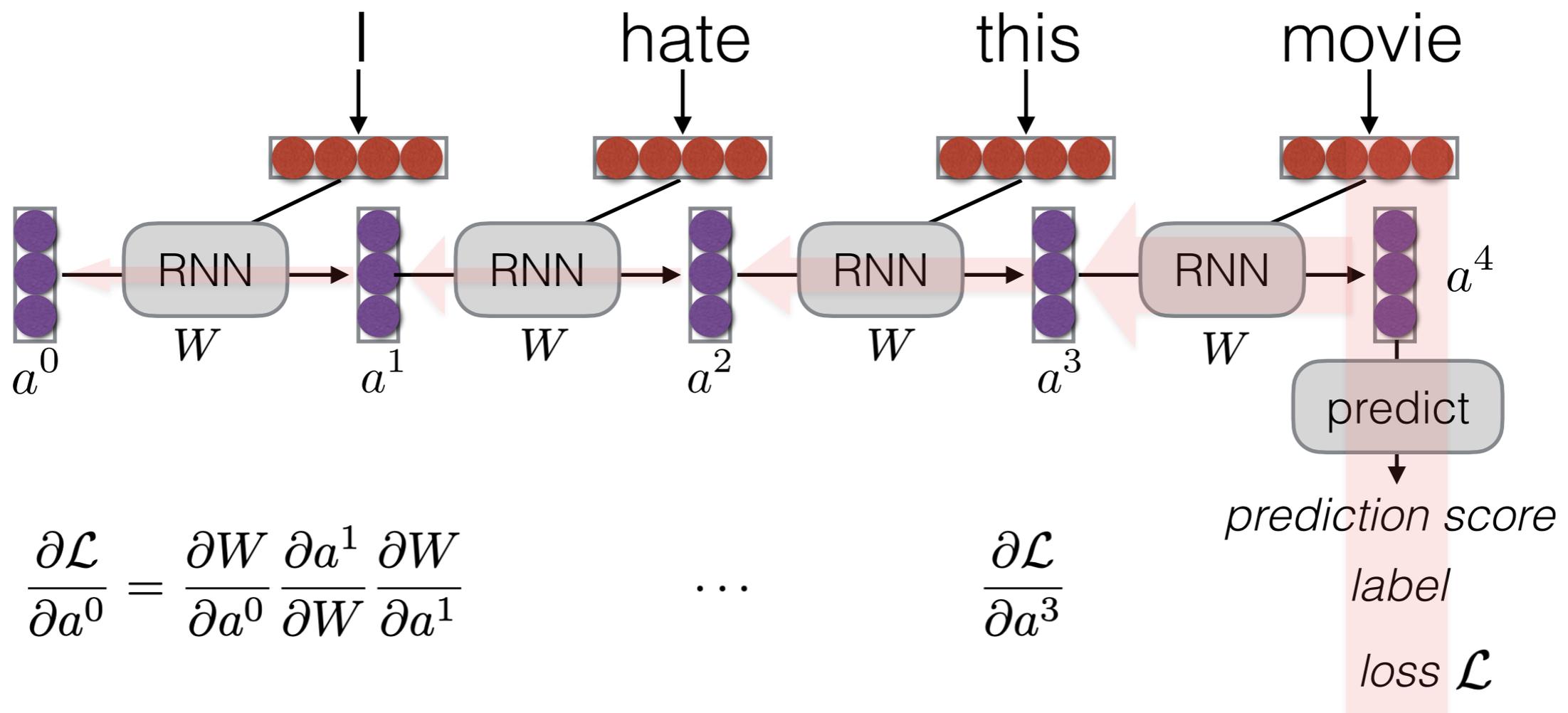
Backward pass

Parameters are shared! Derivatives are accumulated.

$$\frac{\partial \mathcal{L}}{\partial a^0} = \frac{\partial W}{\partial a^0} \frac{\partial a^1}{\partial W} \frac{\partial W}{\partial a^1} \qquad \cdots \qquad \frac{\partial \mathcal{L}}{\partial a^3}$$

$$\frac{\partial \mathcal{L}}{\partial W} = \frac{\partial a^1}{\partial W} \frac{\partial \mathcal{L}}{\partial a^1} + \frac{\partial a^2}{\partial W} \frac{\partial \mathcal{L}}{\partial a^2} + \cdots$$

What happens if these gradients are small or large?

Vanishing/Exploding gradient:
The gradient signal gets smaller (or larger) as it backpropagates further.

Vanishing/Exploding Gradients

Why is vanishing gradient a problem?

- Why? "Squashed" by **non-linearities** (e.g., tanh) or **multiplication of small weights** in matrices.
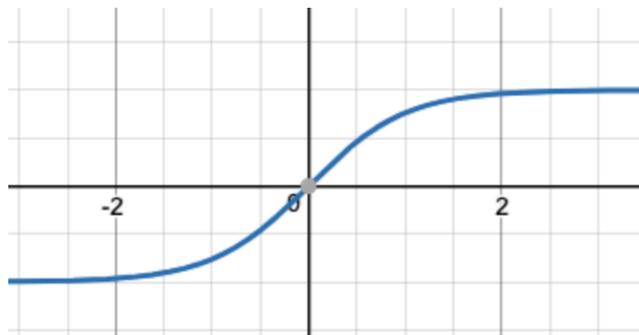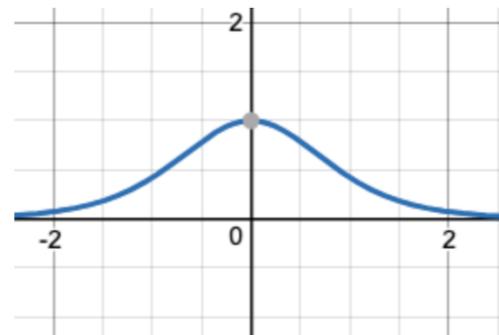
$$\frac{\partial \mathcal{L}}{\partial a^0} = \frac{\partial \mathcal{W}}{\partial a^0} \frac{\partial a^1}{\partial W} \frac{\partial W}{\partial a^1} \cdots$$

Small matrix multiplication

$$f(x) = \tanh(x)$$

$$\frac{\partial f}{\partial x} = 1 - \tanh(x)^2$$

Easy to get derivatives less than 1

Why is vanishing gradient a problem?

- Why? "Squashed" by **non-linearities** (e.g., tanh) or **multiplication of small weights** in matrices.

$$\frac{\partial \mathcal{L}}{\partial a^0} = \frac{\partial W}{\partial a^0} \frac{\partial a^1}{\partial W} \frac{\partial W}{\partial a^1} \qquad \dots \qquad \frac{\partial \mathcal{L}}{\partial a^3}$$

Gradient signal from far away is lost because it's much smaller than **gradient signal from close-by**. So, model parameters are updated only with respect to **near effects**, not **long-term effects**.

A Solution:
Long Short-term Memory
(Hochreiter and Schmidhuber 1997)

- **Basic idea:** make **additive** connections between time steps

- Addition does not modify the last step's gradient, no vanishing

- Gates to control the information flow

- Two types of memory to summarize previous contexts at different paces.

Long short-term memory

- In practice, LSTM works much better than a vanilla RNN
  - Long term memory cell: a_{t-1} <=> Short term memory state: h_{t-1}

$$a_{t-1}$$

"cell state"

$$h_{t-1}$$

RNN output at
previous time step

Long short-term memory

- In practice, LSTM works much better than a vanilla RNN

  - Long term memory cell: a_{t-1} <=> Short term memory state: h_{t-1}

  - Forget gate f_t: how much memory do we want to forget

  - Input gate i_t: how much information do we add to the memory

  - g_t: compute new values we want to add to the memory cell

  - Output gate o_t : how much information do we reflect in the next state

a_{t-1}

"cell state"

$f_t \in [0, 1]$

"forget gate"

$i_t \in [0, 1]$

σ

σ

h_{t-1}

RNN output at
previous time step

W

tanh

$g_t \in [-1, 1]$

σ

$o_t \in [0, 1]$

x_t
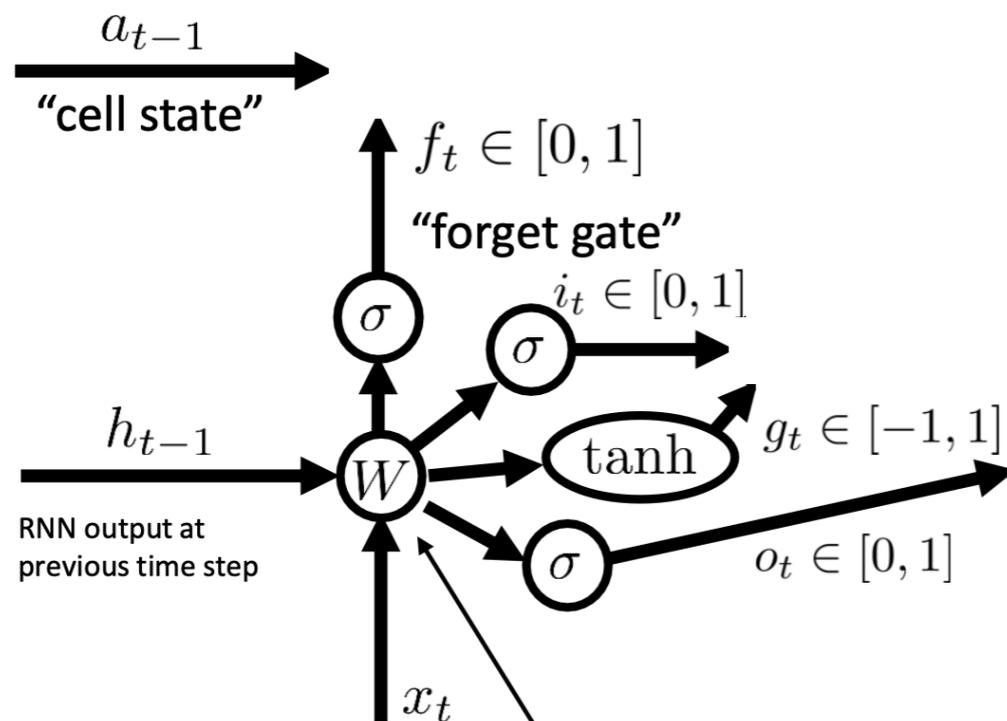
$$W \begin{bmatrix} h_{t-1} \\ x_t \end{bmatrix} + b = \begin{bmatrix} \bar{f}_t \\ \bar{i}_t \\ \bar{g}_t \\ \bar{o}_t \end{bmatrix}$$

$f_t = \sigma(\bar{f}_t)$

$i_t = \sigma(\bar{i}_t)$

$g_t = \tanh(\bar{g}_t)$

$o_t = \sigma(\bar{o}_t)$

σ : sigmoid

$f_t, i_t, g_t, o_t \in \mathbb{R}^d$

$W \in \mathbb{R}^{4d \times 2d}$ in LSTM is 4x larger in
dimensionality than $W \in \mathbb{R}^{d \times 2d}$ in RNN

24

Long short-term memory

- In practice, LSTM works much better than a vanilla RNN
    - Long term memory cell: a_{t-1} <=> Short term memory state: h_{t-1}
    - Forget gate f_t: how much memory do we want to forget
    - Input gate i_t: how much information do we add to the memory
    - g_t: compute new values we want to add to the memory cell
    - Output gate o_t : how much information do we reflect in the next state

a_{t-1}

"cell state"

$f_t \in [0, 1]$

"forget gate"

$i_t \in [0, 1]$

σ

σ

$\times$

$+$

$\times$

tanh

a_t

tanh

$g_t \in [-1, 1]$

$\times$

h_t

h_{t-1}

W

RNN output at previous time step

σ

$o_t \in [0, 1]$

x_t

$$W \begin{bmatrix} h_{t-1} \\ x_t \end{bmatrix} + b = \begin{bmatrix} \bar{f}_t \\ \bar{i}_t \\ \bar{g}_t \\ \bar{o}_t \end{bmatrix}$$
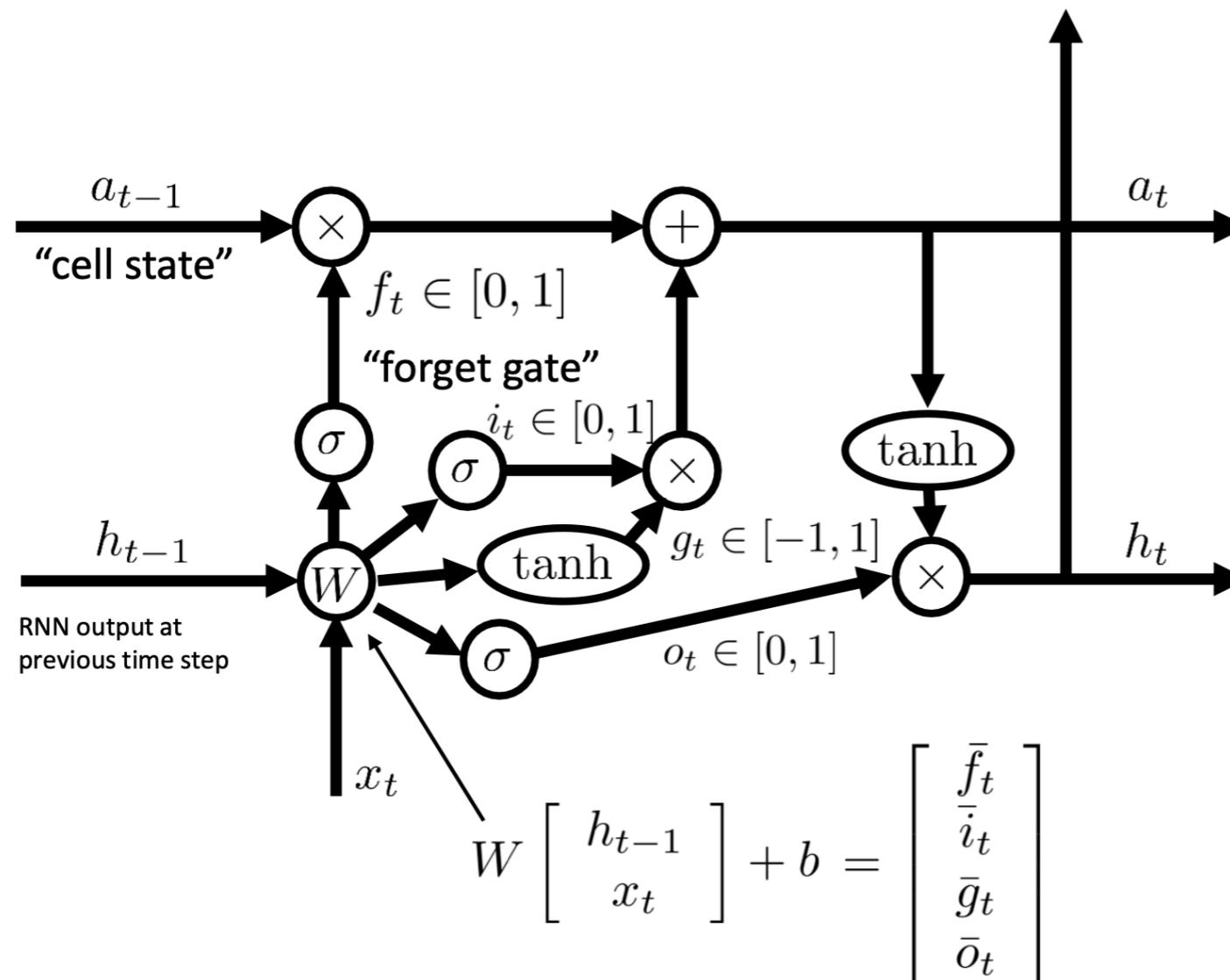
$$a_t = f_t \otimes a_{t-1} + i_t \otimes g_t$$

$$h_t = o_t \otimes \tanh(a_t)$$

where $\otimes$ is element-wise product

25

Long short-term memory
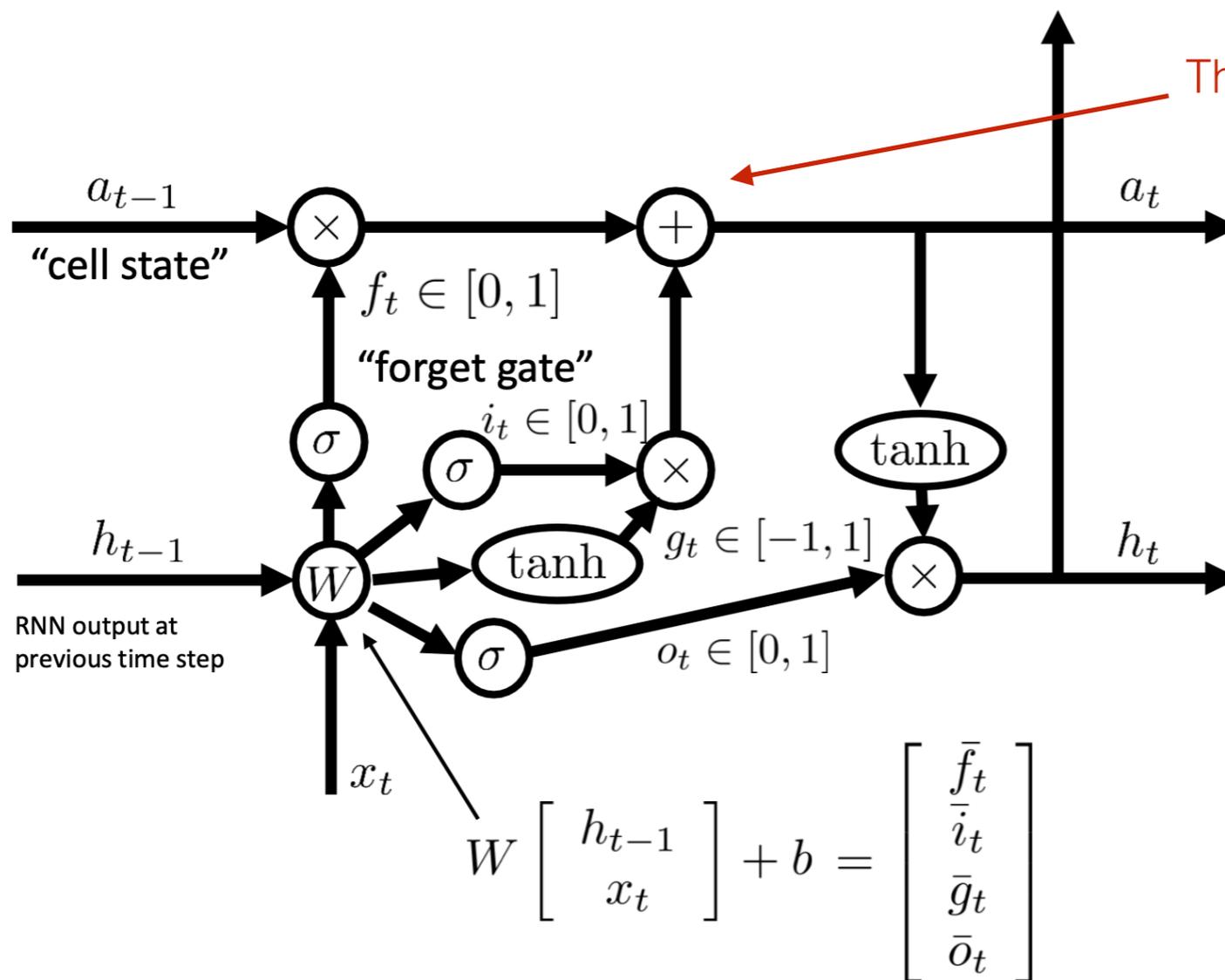
This addition is the secret part!

$$a_t = f_t \otimes a_{t-1} + i_t \otimes g_t$$

changes very little step to step!
"long term memory"

$$h_t = o_t \otimes \tanh(a_t)$$

changes all the time (multiplicative)!
"short term memory"

Why is exploding gradient a problem?

- If the gradient becomes too big, then the SGD update step becomes too big:

$$\theta \leftarrow \theta - \eta \frac{\partial \mathcal{L}}{\partial \theta}$$

- This can cause **bad updates**, if we take too large a step and obtain weird and bad parameters.

- In the worst case, this will result in **Inf** or **NaN** error in the network (then you have to restart training from an earlier checkpoint).

Gradient clipping: solution for exploding gradient

- If the norm of the gradient is greater than some threshold, scale it down before applying SGD update

- Intuition: take a step in the same direction, but a smaller step

Algorithm 1 Pseudo-code for norm clipping

$\hat{\mathbf{g}} \leftarrow \frac{\partial \mathcal{E}}{\partial \theta}$

if $\|\hat{\mathbf{g}}\| \geq threshold$ **then**

$\qquad \hat{\mathbf{g}} \leftarrow \frac{threshold}{\|\hat{\mathbf{g}}\|} \hat{\mathbf{g}}$

end if

Aside: distributional shift (or exposure bias)

- During training, we feed ground-truth words as the context to predict the next word

- During testing, we feed the model's predicted words as the context to predict the next word

I think therefore I am

I like machine learning

I am not just a neural network

Training data

Example: text generation

think: 0.3   therefore: 0.8   I: 0.8
like: 0.3   machine: 0.1   learning: 0.0
am: 0.4   not: 0.1   just: 0.2

I   think   therefore

Training phrase

think: 0.6   hippo: 0.6
like: 0.3   paintbrush: 0.3
drive: 0.1   California: 0.1
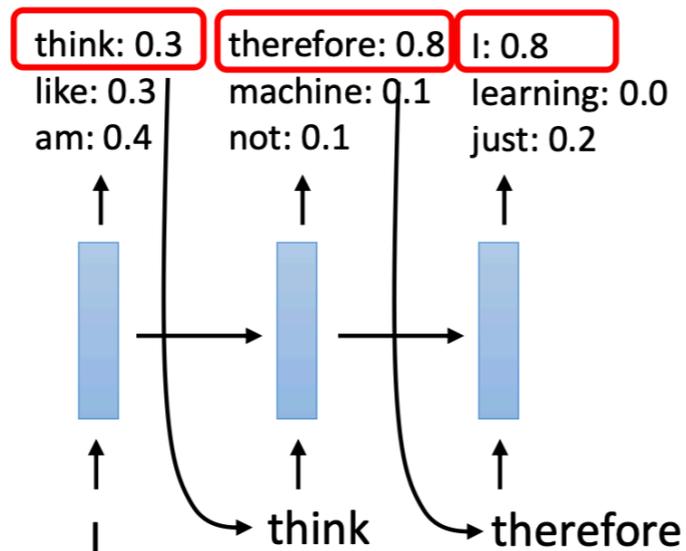
I   drive
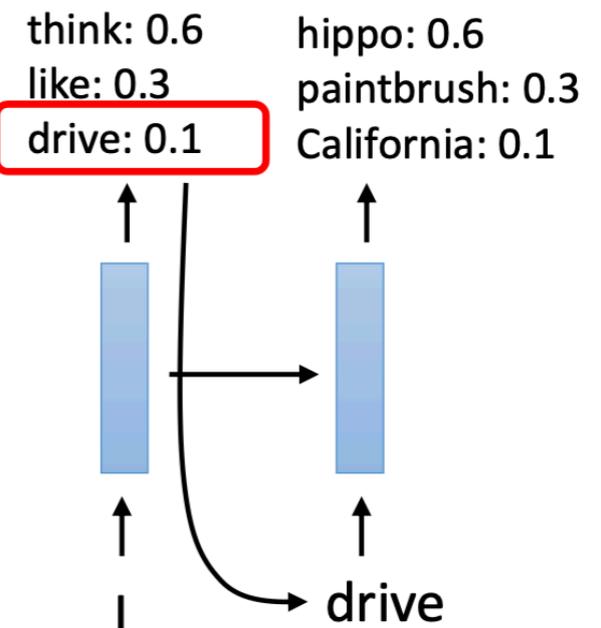
Testing phrase

Distributional shift: input distribution shifts from true strings (at training) to predicted strings (at test time)
Even **one** random mistake can completely scramble the output!

Aside: scheduled sampling

- An old trick from reinforcement learning adapted to training RNNs

  - Start by feeding in ground-truth tokens as input, since model predictions are mostly nonsense.

  - In the middle of training, randomly decide whether to give the network a ground-truth token or its own previous prediction.

  - At the end of training, mostly feed in the model's own predictions, to mitigate distribution shift.
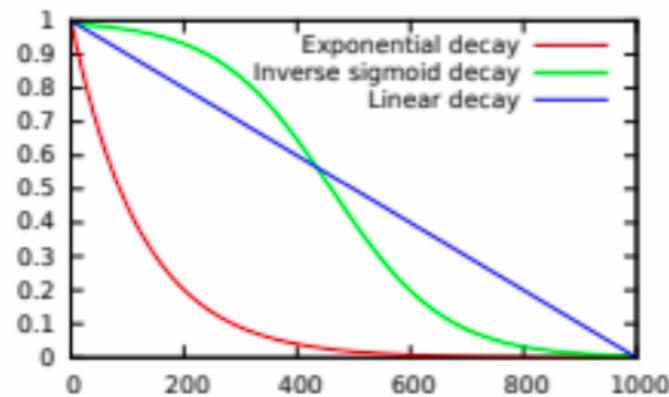
Schedules for probability of using ground-truth token
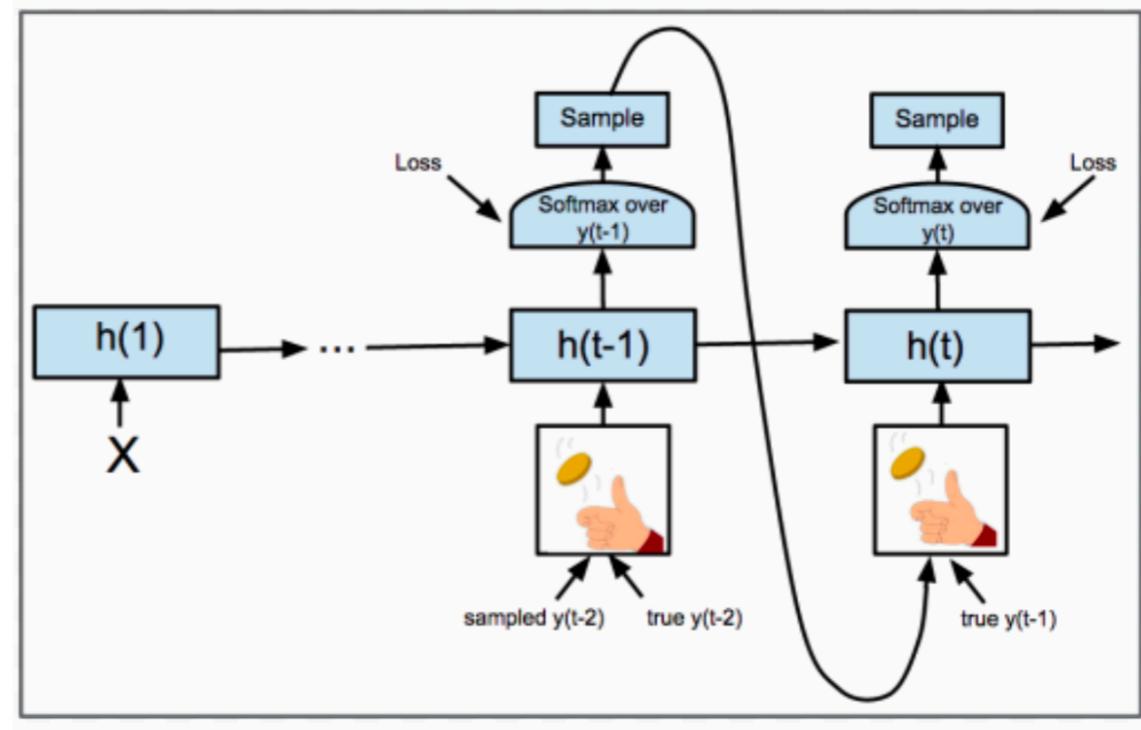
Applications of RNNs

What Can RNNs Do?

- Represent a sentence

  - Read whole sentence, make a prediction

- Represent a context within a sentence

  - Read context up until that point

Different ways to use RNNs

one to one	one to many	many to one	many to many	many to many
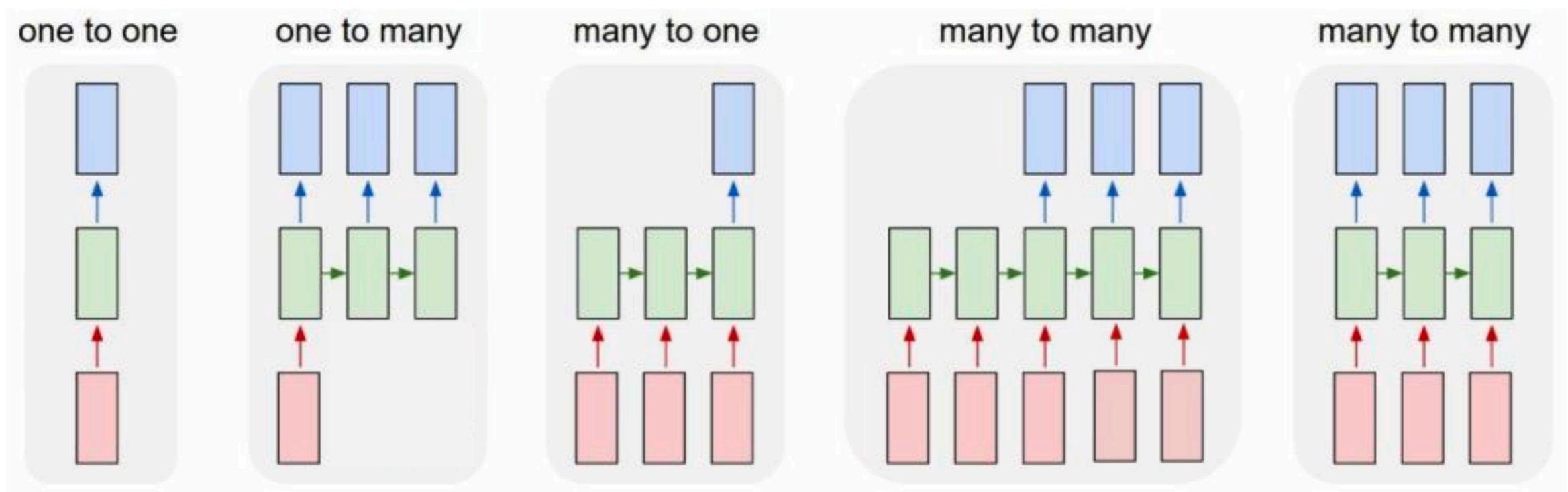
e.g., image captioning e.g., sentence classification   e.g., machine translation   e.g., sentence tagging (POS, NER)

Encoding Sentences
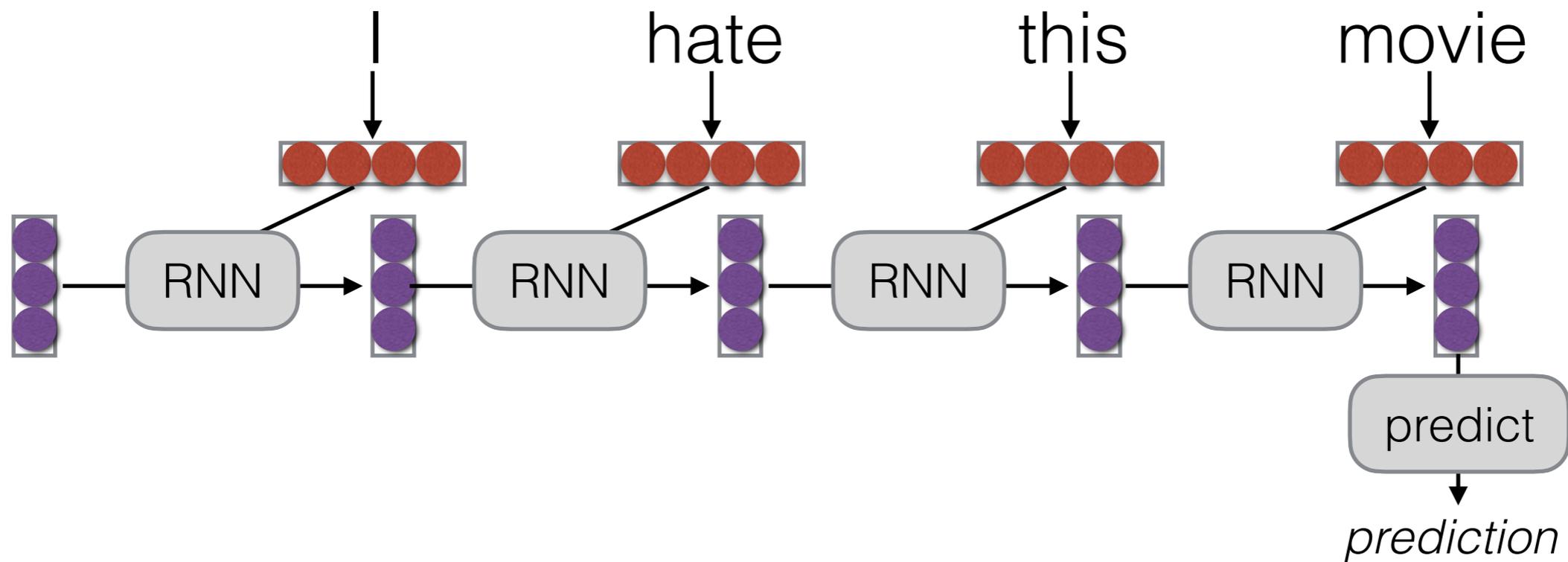
- Binary or multi-class prediction

- Sentence representation for retrieval, sentence comparison, etc.

Representing Contexts
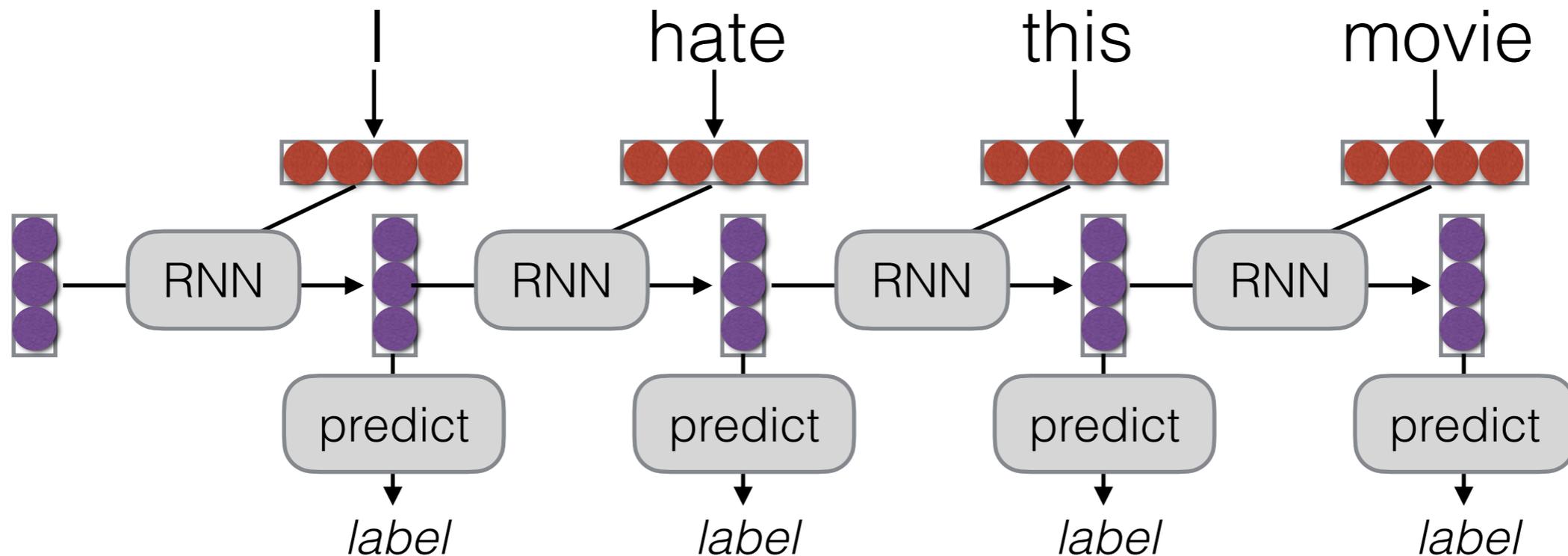
- Sequence labeling

- Calculating representations for parsing, etc.

Bi-RNNs

- A simple extension, run the RNN in both directions
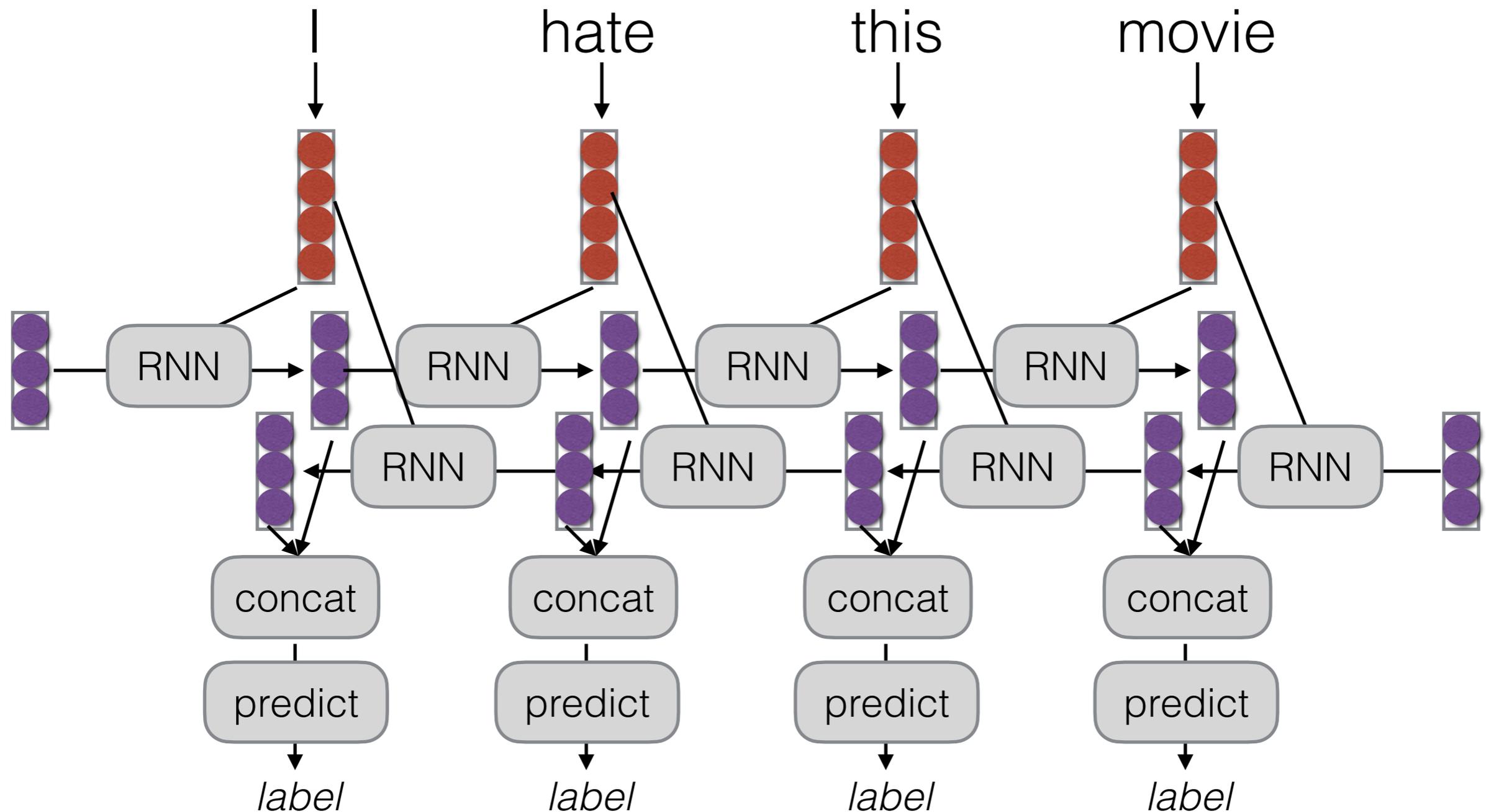
e.g. Language Modeling
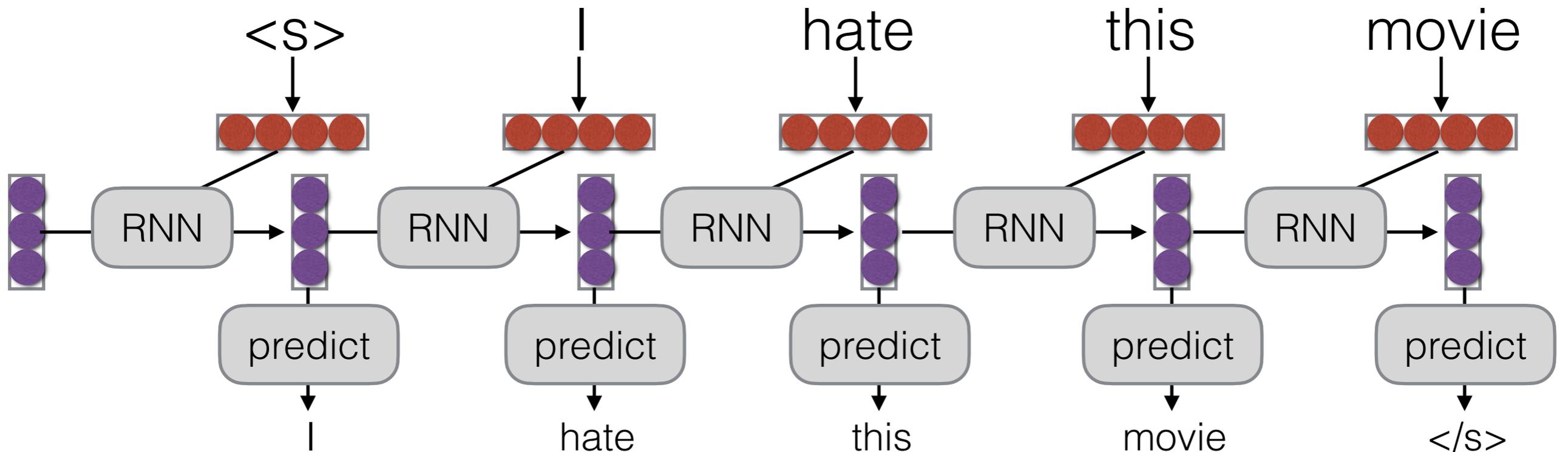
- Language modeling is like a tagging task, where each tag is the next word!

- Note: this is an autoregressive model

Understanding RNNs

What can LSTMs Learn? (1)
(Karpathy et al. 2015)

- Additive connections make single nodes surprisingly interpretable

Text color corresponds to *tahn(c)*, where -1 is red and +1 is blue, where c is one element in the memory cell.

What can LSTMs Learn? (2)
(Shi et al. 2016, Radford et al. 2017)

Count length of sentence

Sentiment

Notably, difficult for GRUs!
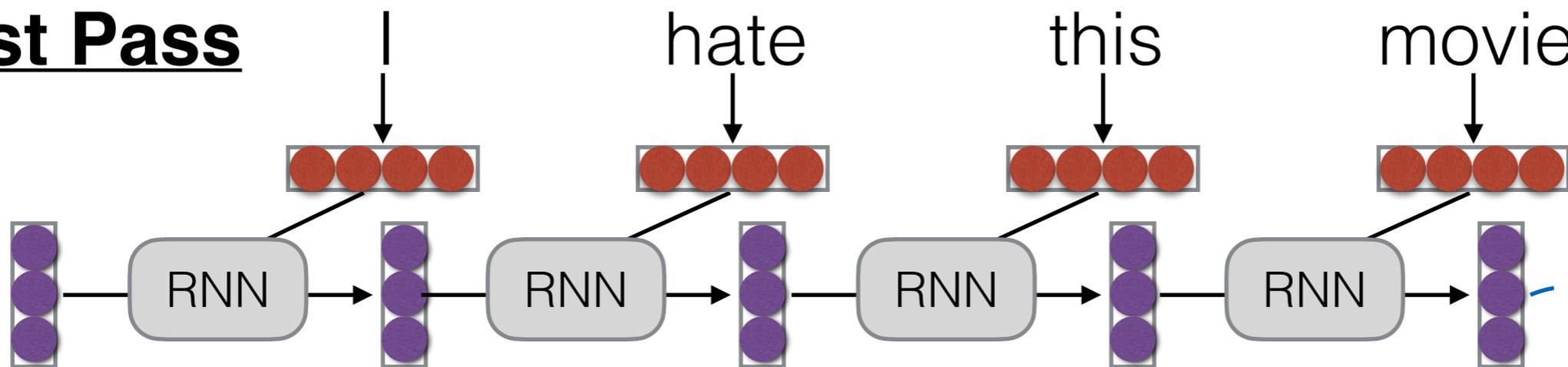
Handling Long Sequences

Handling Long Sequences

- Sometimes we would like to capture long-term dependencies over long sequences

- e.g. words in full documents

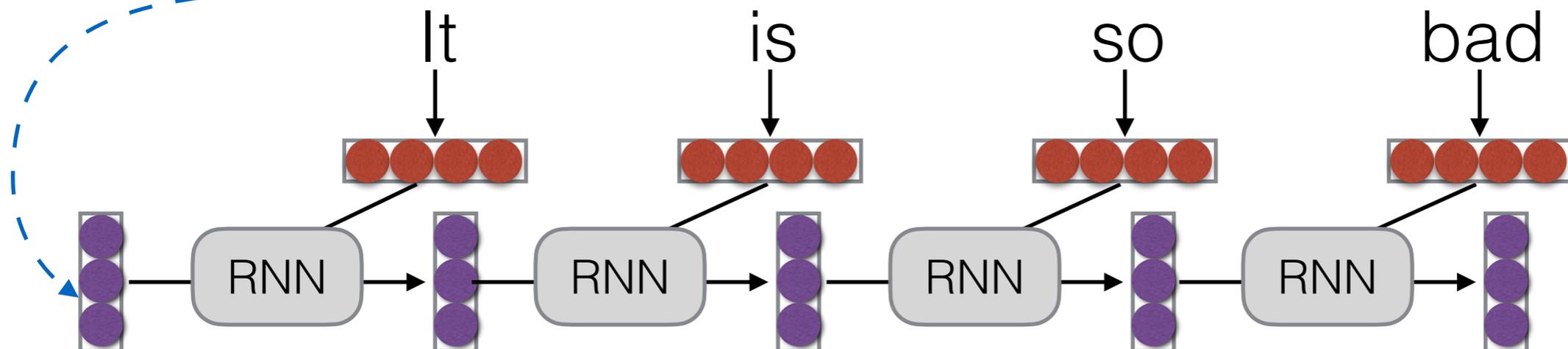- However, this may not fit on (GPU) memory

Truncated BPTT

- Backprop over shorter segments, initialize w/ the state from the previous segment

1st Pass

I   hate   this   movie

state only, no backprop

2nd Pass

It   is   so   bad

RNN Variants

Gated Recurrent Units
(Cho et al. 2014)

- A simpler version that preserves the additive connections

$$z_t = \sigma_g(W_z x_t + U_z h_{t-1} + b_z)$$
$$r_t = \sigma_g(W_r x_t + U_r h_{t-1} + b_r)$$
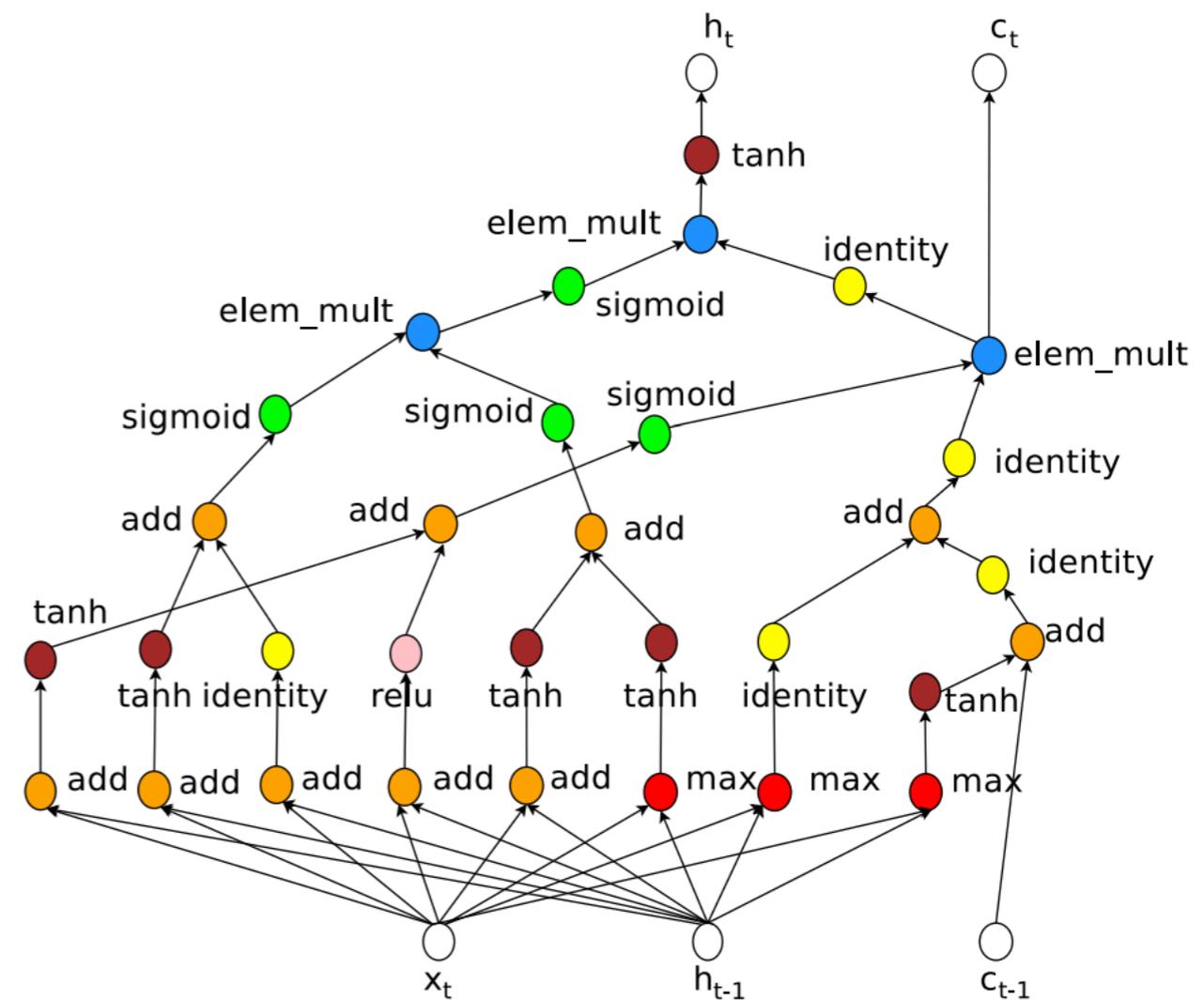$$h_t = \boxed{(1 - z_t)} \circ h_{t-1} + \boxed{z_t} \circ \sigma_h(W_h x_t + U_h(r_t \circ h_{t-1}) + b_h)$$

Additive   or   Non-linear

- **Note:** GRUs cannot do things like simply count

Extensive Architecture Search for LSTMs
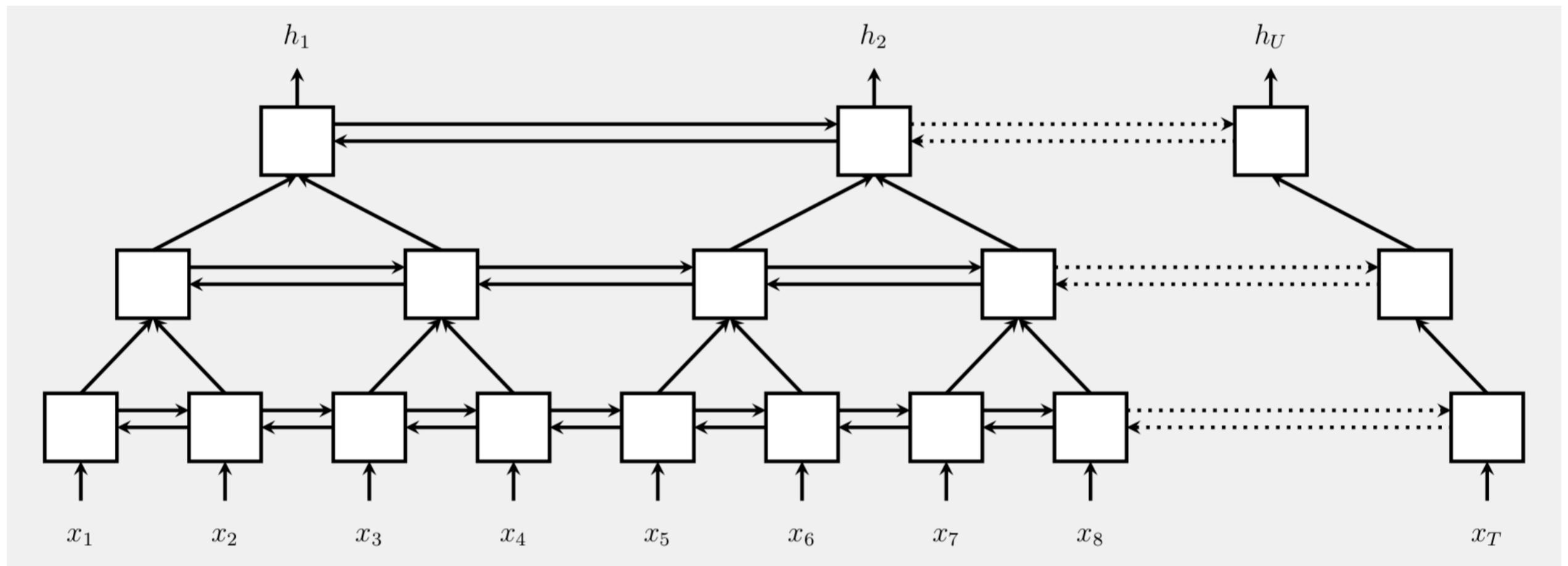(Greffen et al. 2015, Zoph and Le 2017)

- Many different types of architectures tested for LSTMs (manually or automatically)

- **Conclusion:** basic LSTM quite good, other variants (e.g. coupled input/forget gates) reasonable

Multi-scale/Pyramidal RNN
(e.g. Chan et al. 2015)

- Have different RNNs of different scales

Questions?