Neural Natural Language Processing

Lecture 4: Recurrent neural networks for natural language processing

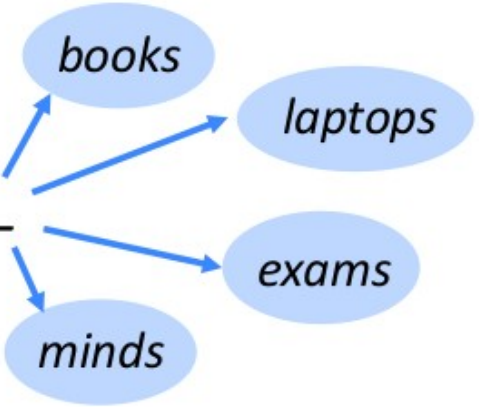
Plan of the lecture

- **Part 1**: Language modeling.
- **Part 2**: Recurrent neural networks.
- **Part 3**: Long-Short Term Memory (LSTM).
- **Part 4**: LSTMs for sequence labelling.
- **Part 5**: LSTMs for text categorization.

Language Models (LMs)

- **Language Modeling** is the task of predicting what word comes next.

the students opened their _____



- More formally: given a sequence of words $\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(t)}$, compute the probability distribution of the next word $\boldsymbol{x}^{(t+1)}$:

$$P(\boldsymbol{x}^{(t+1)} | \ \boldsymbol{x}^{(t)}, \ldots, \boldsymbol{x}^{(1)})$$

Probabilistic Multiclass Classifier with Variable length input

where $\boldsymbol{x}^{(t+1)}$ can be any word in the vocabulary $V = \{\boldsymbol{w}_1, \ldots, \boldsymbol{w}_{|V|}\}$

- A system that does this is called a **Language Model**.

Source: Abigail See. CS224N/Ling284 slides: http://web.stanford.edu/class/cs224n/

Language Models (LMs)

- You can also think of a Language Model as a system that assigns probability to a piece of text.

- For example, if we have some text $\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(T)}$, then the probability of this text (according to the Language Model) is:

$$P(\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(T)}) = P(\boldsymbol{x}^{(1)}) \times P(\boldsymbol{x}^{(2)} | \boldsymbol{x}^{(1)}) \times \cdots \times P(\boldsymbol{x}^{(T)} | \boldsymbol{x}^{(T-1)}, \ldots, \boldsymbol{x}^{(1)})$$

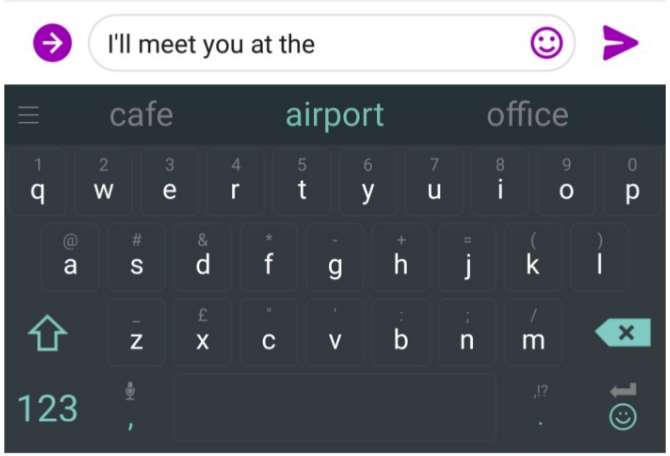
$$= \prod_{t=1}^{T} P(\boldsymbol{x}^{(t)} | \boldsymbol{x}^{(t-1)}, \ldots, \boldsymbol{x}^{(1)})$$

This is what our LM provides

Source: Abigail See. CS224N/Ling284 slides: http://web.stanford.edu/class/cs224n/

Language Models are useful for

- Estimation of [conditional] probability of a sequence P(*x*), P(*x*|*s*)
  - Ranking hypothesis
    - Speech Recognition
    - Machine Translation
- Generation of texts from P(*X*), P(*X*|*s*)
  - Autocomplete / autoreply
  - Generate translation / image caption
  - Neural poetry
- Unsupervised Pretraining



Source: Abigail See. CS224N/Ling284 slides: http://web.stanford.edu/class/cs224n/

n-gram Language Modeling

- First we make a simplifying assumption: $\boldsymbol{x}^{(t+1)}$ depends only on the preceding *n-1* words.

n-1 words

$$P(\boldsymbol{x}^{(t+1)}|\boldsymbol{x}^{(t)},\ldots,\boldsymbol{x}^{(1)}) = P(\boldsymbol{x}^{(t+1)}|\boldsymbol{x}^{(t)},\ldots,\boldsymbol{x}^{(t-n+2)}) \quad \text{(assumption)}$$

prob of a n-gram

prob of a (n-1)-gram

$$= \frac{P(\boldsymbol{x}^{(t+1)},\boldsymbol{x}^{(t)},\ldots,\boldsymbol{x}^{(t-n+2)})}{P(\boldsymbol{x}^{(t)},\ldots,\boldsymbol{x}^{(t-n+2)})} \quad \text{(definition of conditional prob)}$$

- **Question:** How do we get these *n*-gram and (*n*-1)-gram probabilities?
- **Answer:** By counting them in some large corpus of text!

$$\approx \frac{\text{count}(\boldsymbol{x}^{(t+1)},\boldsymbol{x}^{(t)},\ldots,\boldsymbol{x}^{(t-n+2)})}{\text{count}(\boldsymbol{x}^{(t)},\ldots,\boldsymbol{x}^{(t-n+2)})} \quad \text{(statistical approximation)}$$

Source: Abigail See. CS224N/Ling284 slides: http://web.stanford.edu/class/cs224n/

n-gram Language Modeling

Suppose we are learning a 4-gram Language Model.

~~*as the proctor started the clock, the*~~ *students opened their* _____

discard

condition on this

$$P(\boldsymbol{w}|\text{students opened their}) = \frac{\text{count}(\text{students opened their } \boldsymbol{w})}{\text{count}(\text{students opened their})}$$

For example, suppose that in the corpus:

- "students opened their" occurred 1000 times
- "students opened their books" occurred 400 times
  - → P(books | students opened their) = 0.4
- "students opened their exams" occurred 100 times
  - → P(exams | students opened their) = 0.1

Should we have discarded the "proctor" context?

Source: Abigail See. CS224N/Ling284 slides: http://web.stanford.edu/class/cs224n/

Problems of *n*-gram LMs

- Small fixed-size context
  - *n*>5 hardly can be used in practice
- Lots of storage space to keep n-gram counts
- Sparsity of data

  Most ngrams (both probable and improbable) never occur even in very large train corpus

  => cannot compare them

  - The cat caught a frog on Monday → The kitten will catch a toad/*house on Friday
  - Tezguino is an alcoholic beverage. It is made from corn and consumed during festivals. Tezguino makes us _

Neural Language Models: Motivation

- Neural net-based language models turn out to have many advantages over the *n*-gram language models:
  - neural language models don't need smoothing
  - they can handle much longer histories
    - recurrent architectures
  - they can generalize over contexts of similar words
    - word embeddings / distributed representations
- **(+)** a neural language model has much **higher predictive accuracy** than an *n*-gram language model!
- **(–)** neural net language models are **strikingly slower to train** than traditional language models

Source: https://web.stanford.edu/~jurafsky/slp3/7.pdf

Neural Language Model based on FFNN by Bengio et al. (2003)

- **Input**: at time *t* a representation of some number of previous words $(w_{t-1}, w_{t-2}, \text{etc.})$
  - Similarly to the *n*-gram model approximates the probability of a word given the entire prior context

$$P(w_t|w_1^{t-1})$$

  - ...by approximating based on the *N* previous words

$$P(w_t|w_1^{t-1}) \approx P(w_t|w_{t-N+1}^{t-1})$$

Source: https://web.stanford.edu/~jurafsky/slp3/7.pdf

Neural Language Model based on FFNN by Bengio et al. (2003)

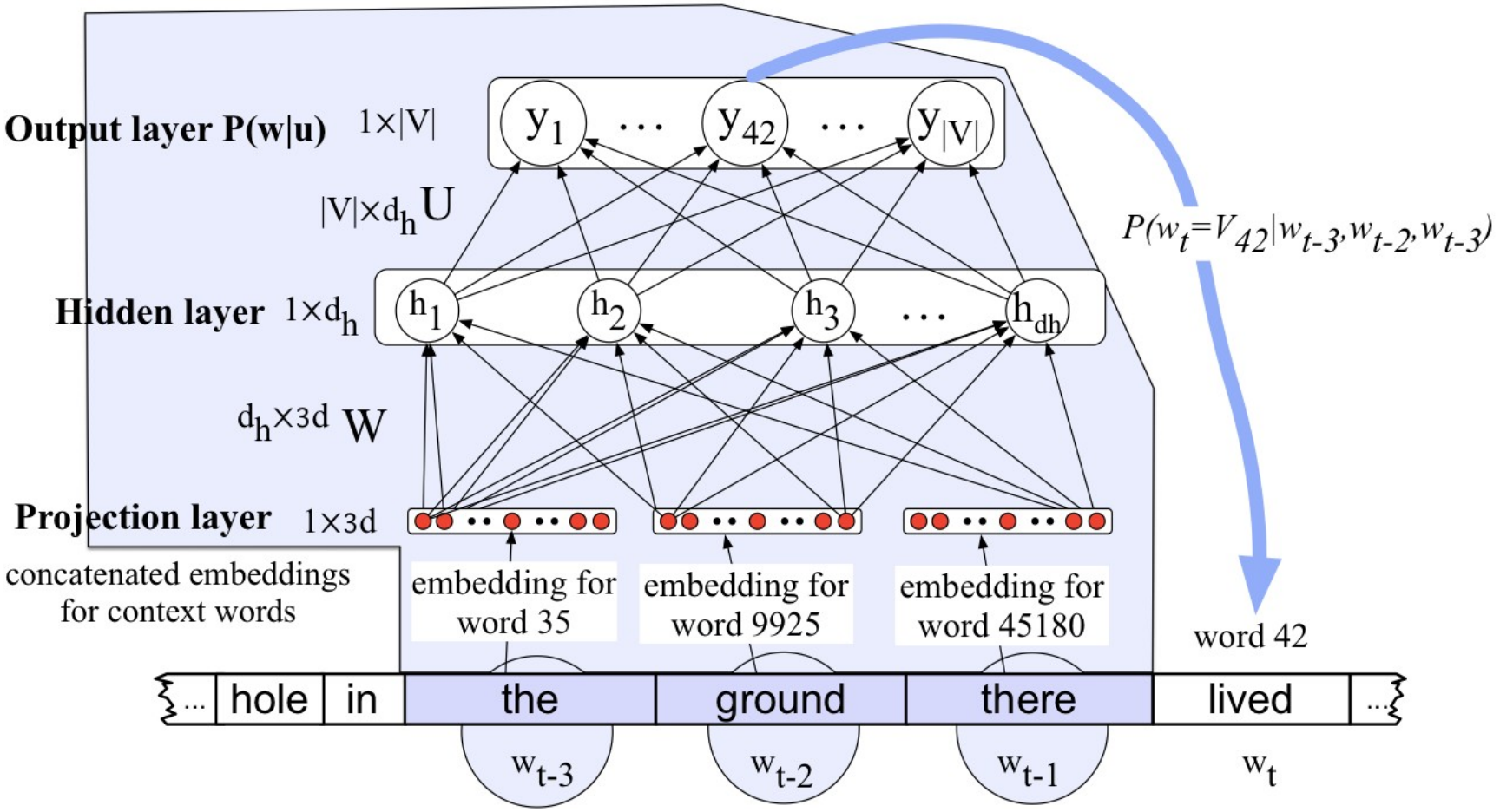
- Representing the prior context as embeddings:
  - rather than by exact words (*n*-gram LMs)
  - allows neural LMs to generalize to unseen data:

- “I have to make sure when I get home to feed the cat.”
  - “feed the dog”
  - cat ↔ dog, pet, hamster, ...

Source: https://web.stanford.edu/~jurafsky/slp3/7.pdf

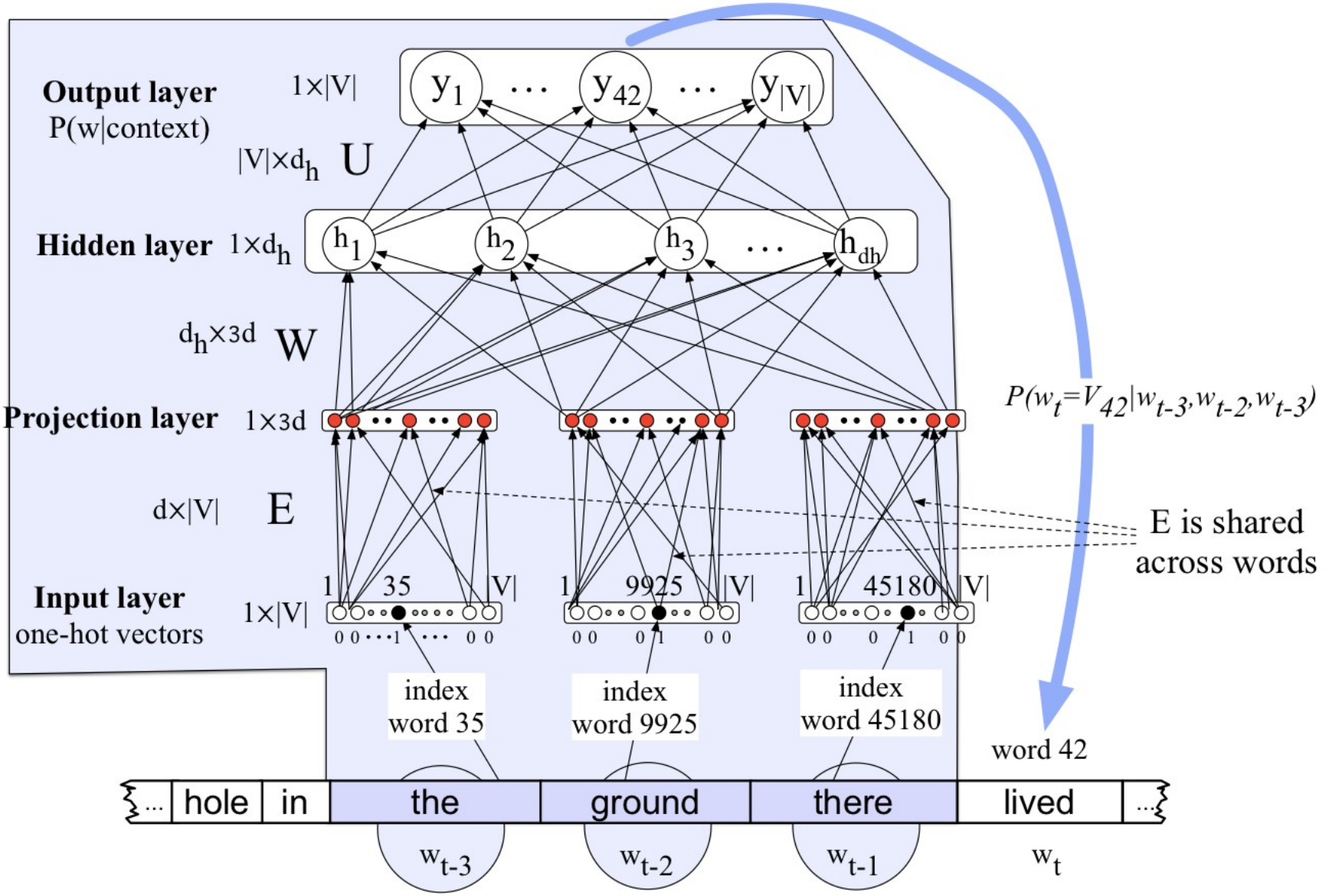
Neural Language Model based on FFNN by Bengio et al. (2003)

- A moving window at time *t* with an embedding vector representing each of the *N*=3 previous words:



Source: https://web.stanford.edu/~jurafsky/slp3/7.pdf

Neural Language Model based on FFNN: no pre-trained embeddings



Source: https://web.stanford.edu/~jurafsky/slp3/7.pdf

Neural Language Model based on FFNN: Training

- At each word w_t, the cross-entropy (negative log likelihood) loss is:

$$L = -\log p(w_t | w_{t-1}, \ldots, w_{t-n+1})$$

- The gradient for this loss is:

$$\theta_{t+1} = \theta_t - \eta \frac{\partial -\log p(w_t | w_{t-1}, \ldots, w_{t-n+1})}{\partial \theta}$$

Source: https://web.stanford.edu/~jurafsky/slp3/7.pdf

Plan of the lecture

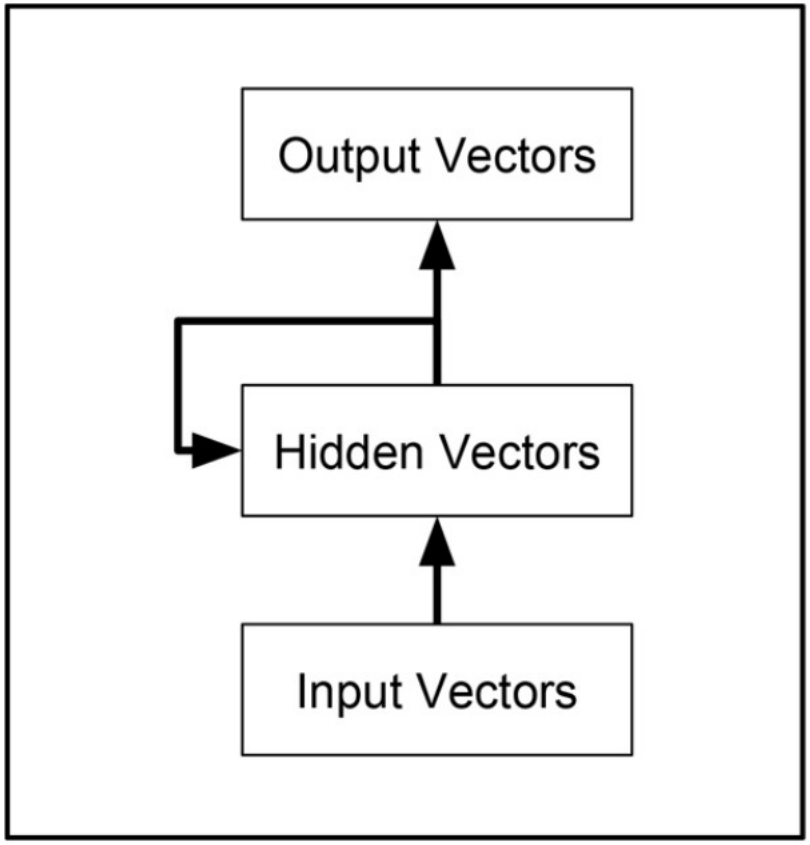
- **Part 1**: Language modeling.
- **Part 2**: Recurrent neural networks.
- **Part 3**: Long-Short Term Memory (LSTM).
- **Part 4**: LSTMs for sequence labelling.
- **Part 5**: LSTMs for text categorization.

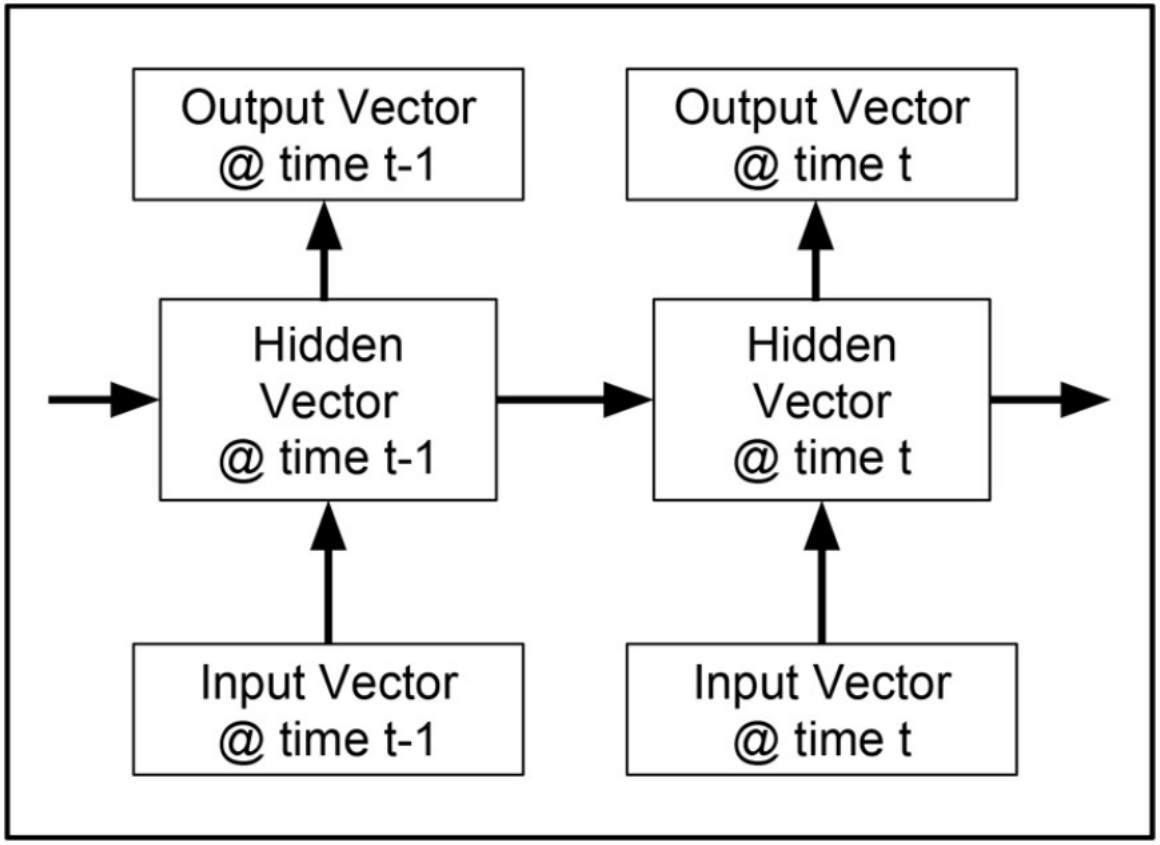
Language Modeling with a fixed context: issues

- The sliding window approach is problematic for a number of reasons:
  - limits the context from which information can be extracted;
  - anything outside the context window has no impact on the decision being made.
- **Recurrent Neural Networks (RNNs)**:
  - dealing directly with the temporal aspect of language;
  - handle variable length inputs without the use of arbitrary fixed-sized windows.

Elman (1990) Recurrent Neural Network (RNN)

- Recurrent networks model sequences:
  - The goal is to learn a **representation of a sequence**;
  - Maintaining a hidden state vector that captures the **current state of the sequence**;
  - Hidden state vector is computed from both a **current input** vector and the **previous hidden state** vector.

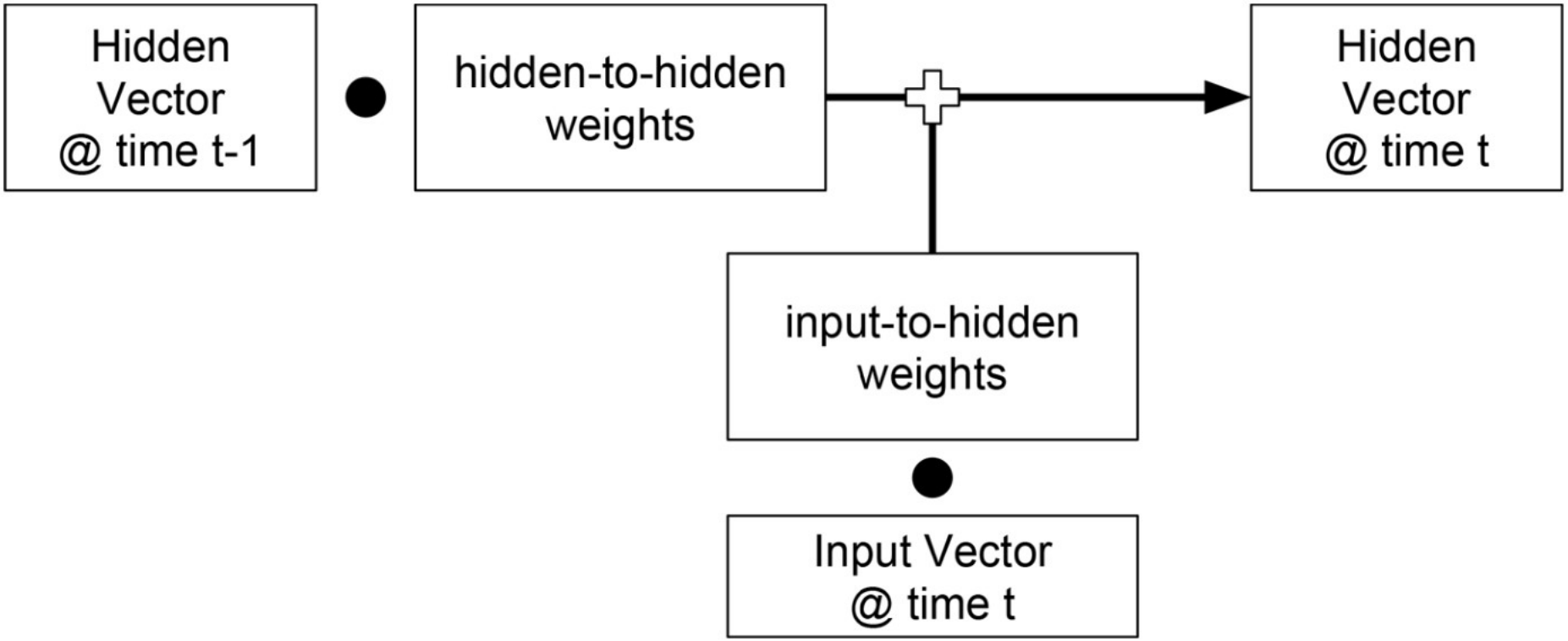




Source: Rao D. & McMahan (2019): Natural Language Processing with PyTorch: Build Intelligent Language Applications Using Deep Learning.

Elman (1990) Recurrent Neural Network (RNN)

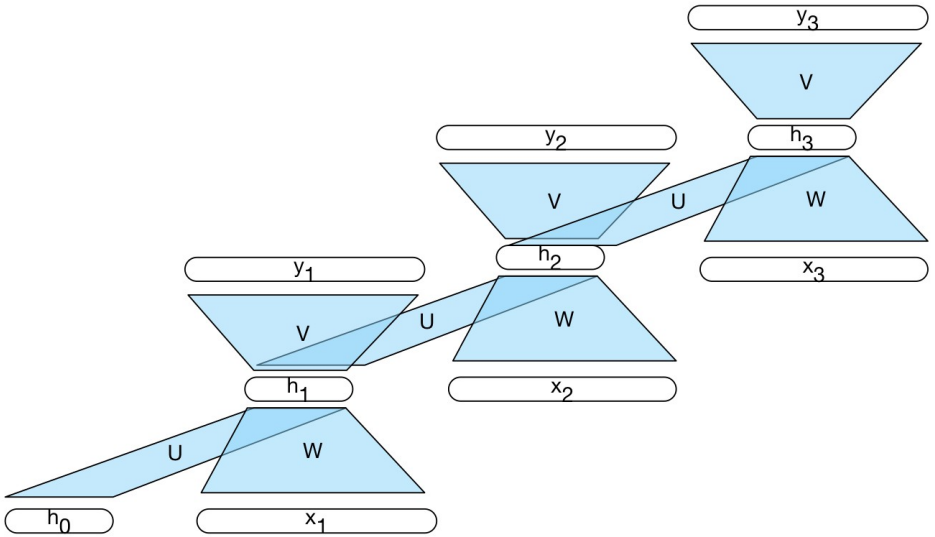
- **Input vector from the current time step** and the **hidden state vector from the previous time step** are mapped to the **hidden state vector of the current time step**:



Source: Rao D. & McMahan (2019): Natural Language Processing with PyTorch: Build Intelligent Language Applications Using Deep Learning.

Elman (1990) Recurrent Neural Network (RNN)

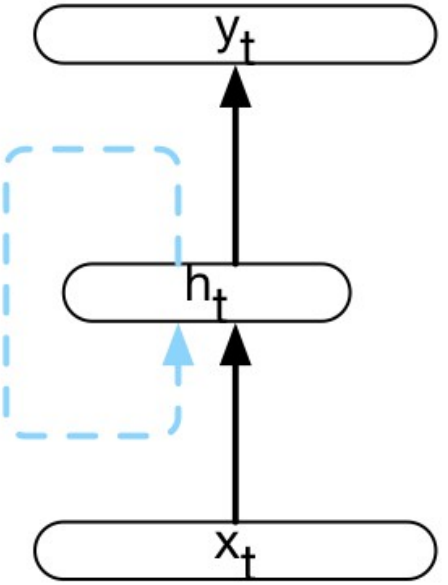
- Hidden-to-hidden and input to hidden weights are shared across the different time steps.
- Weights will be adjusted so that the RNN is learning how to incorporate incoming information and maintain a state representation summarizing the input seen so far;
- RNN does not have any way of knowing which time step it is on;
- RNN is learning how to transition from one time step to another and maintain a state representation that will minimize its loss.

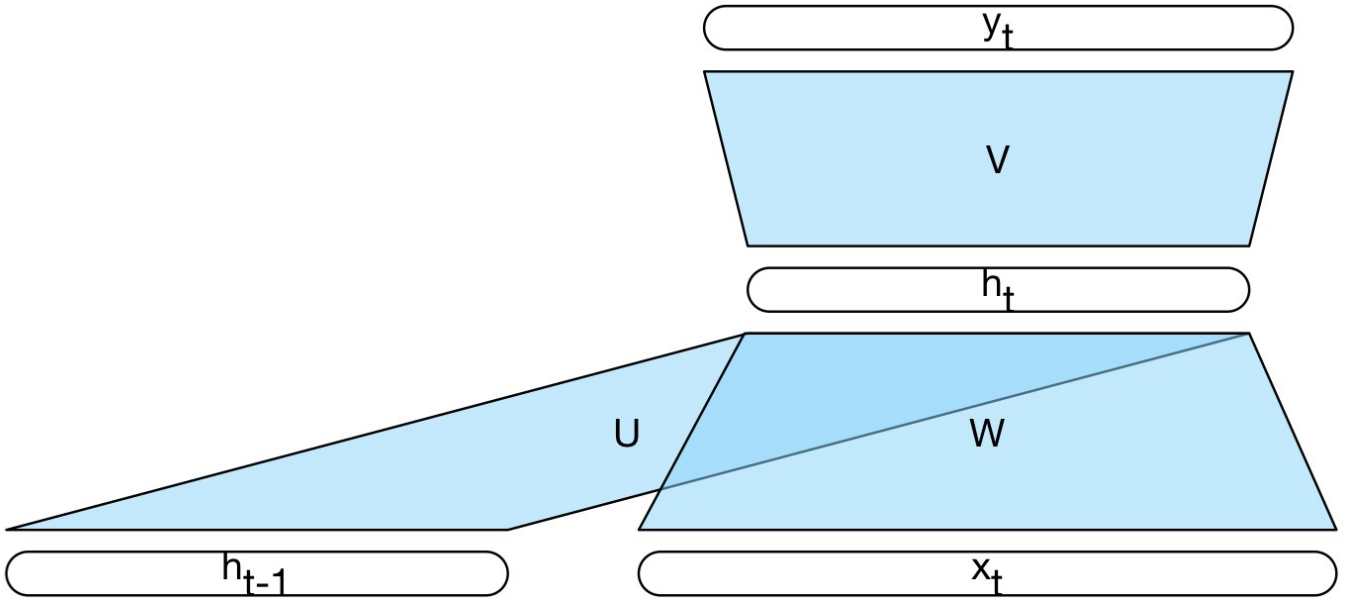


Source: Rao D. & McMahan (2019): Natural Language Processing with PyTorch: Build Intelligent Language Applications Using Deep Learning. And https://web.stanford.edu/~jurafsky/slp3/9.pdf

Elman (1990) or “Simple” RNN

- input vector representing the current input element x_t
- hidden units h_t
- output y_t





$$h_t = g(Uh_{t-1} + Wx_t)$$
$$y_t = f(Vh_t)$$
$$y_t = softmax(Vh_t)$$

Source: https://web.stanford.edu/~jurafsky/slp3/9.pdf

Forward inference in a simple recurrent network

- The matrices U, V and W are shared across time, while new values for *h* and *y* are calculated with each time step.

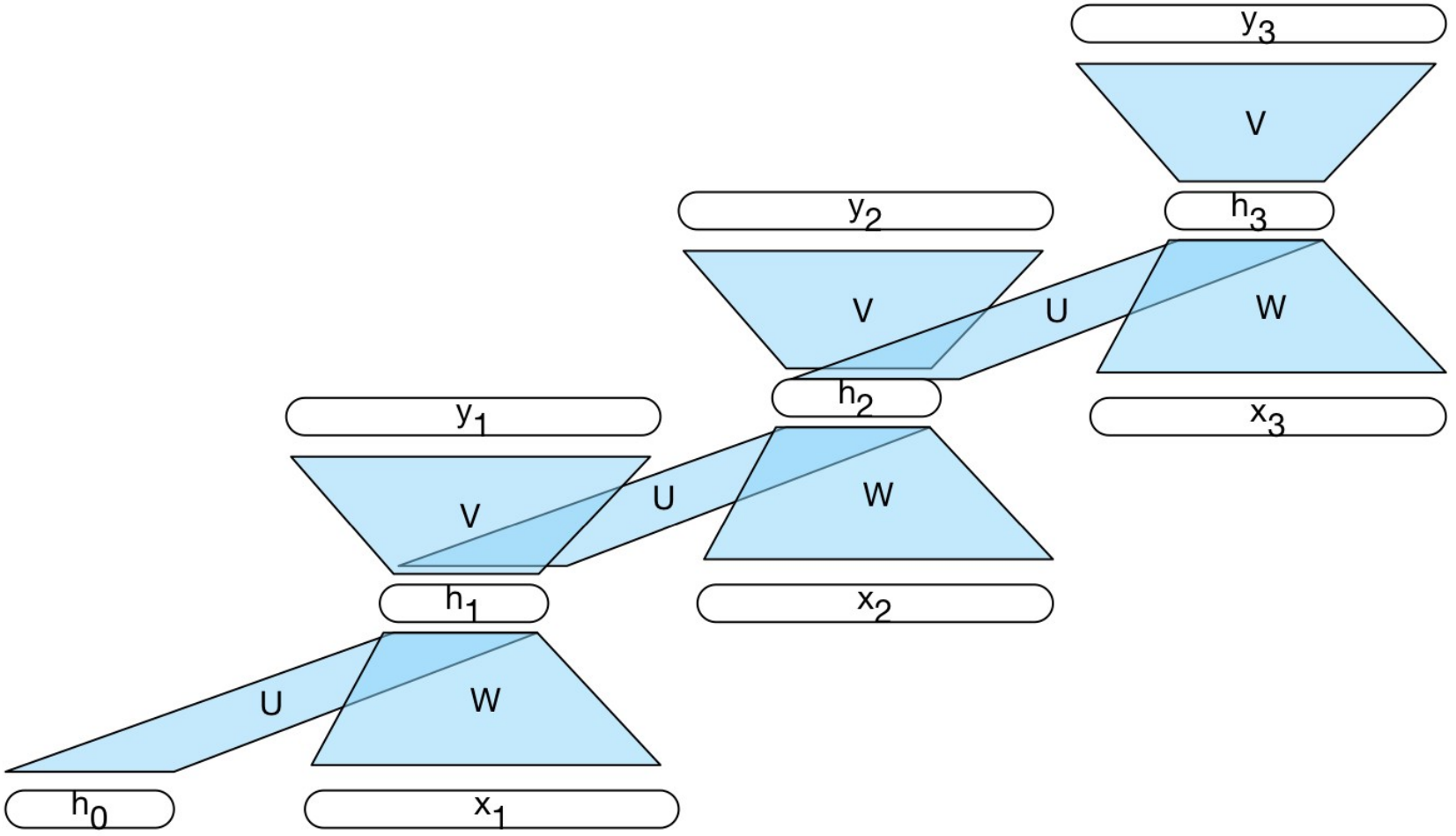
function FORWARDRNN(x, *network*) **returns** output sequence y

$h_0 \leftarrow 0$
for $i \leftarrow 1$ **to** LENGTH(x) **do**
    $h_i \leftarrow g(U\ h_{i-1} + W\ x_i)$
    $y_i \leftarrow f(V\ \ h_i)$
return y

Source: https://web.stanford.edu/~jurafsky/slp3/9.pdf

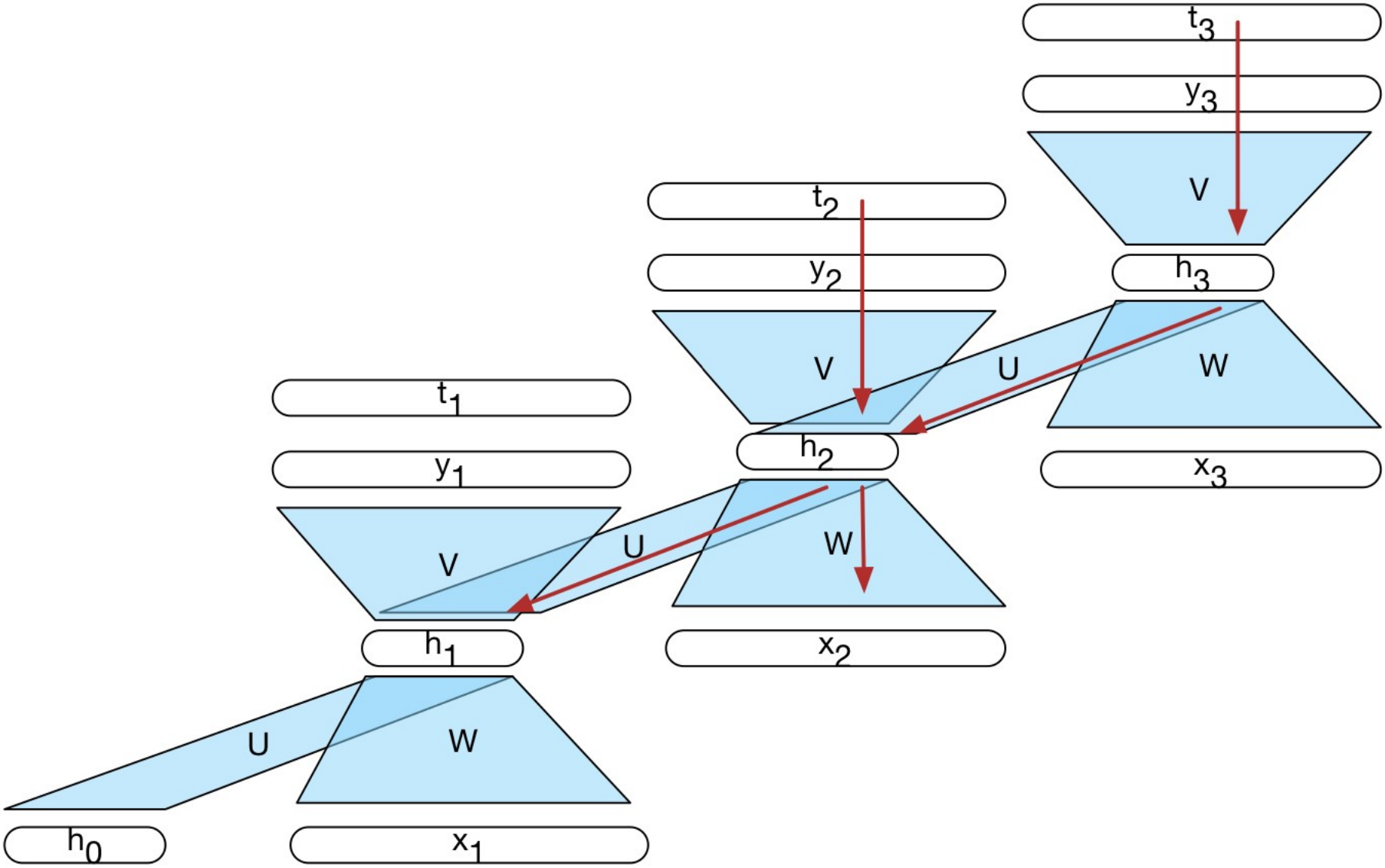
A simple recurrent neural network shown unrolled in time

- Network layers are copied for each time step, while the weights U, V and W are shared in common across all time steps.



Source: https://web.stanford.edu/~jurafsky/slp3/9.pdf

Training: backpropagation through time (BPTT)



Source: https://web.stanford.edu/~jurafsky/slp3/9.pdf

BPTT: backpropagation through time (Werbos, 1974; Rumelhart et al. 1986)

- Gradient of the output weights V:

$$\frac{\partial L}{\partial V} = \frac{\partial L}{\partial a}\frac{\partial a}{\partial z}\frac{\partial z}{\partial V} \qquad \delta_{out} = \frac{\partial L}{\partial a}\frac{\partial a}{\partial z} \qquad \frac{\partial L}{\partial V} = \delta_{out} h_t$$

$$\delta_{out} = L' g'(z)$$

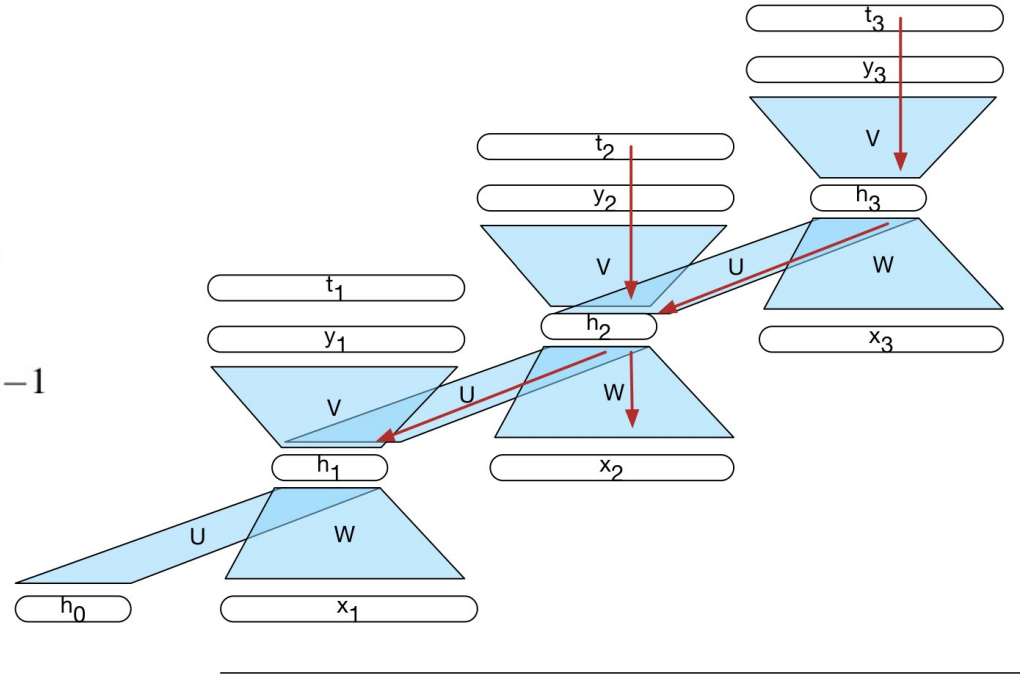
- Gradient of the W and U weights:

$$\delta_h = g'(z) V \delta_{out} + \delta_{next}$$

$$\frac{dL}{dW} = \frac{dL}{dz}\frac{dz}{da}\frac{da}{dW} \qquad \frac{\partial L}{\partial W} = \delta_h x_t$$

$$\frac{dL}{dU} = \frac{dL}{dz}\frac{dz}{da}\frac{da}{dU} \qquad \frac{\partial L}{\partial U} = \delta_h h_{t-1}$$

$$\delta_{next} = g'(z) U \delta_h$$



Source: https://web.stanford.edu/~jurafsky/slp3/9.pdf

Optimization

- Loss is differentiable w.r.t. parameters

  => use backprop+SGD

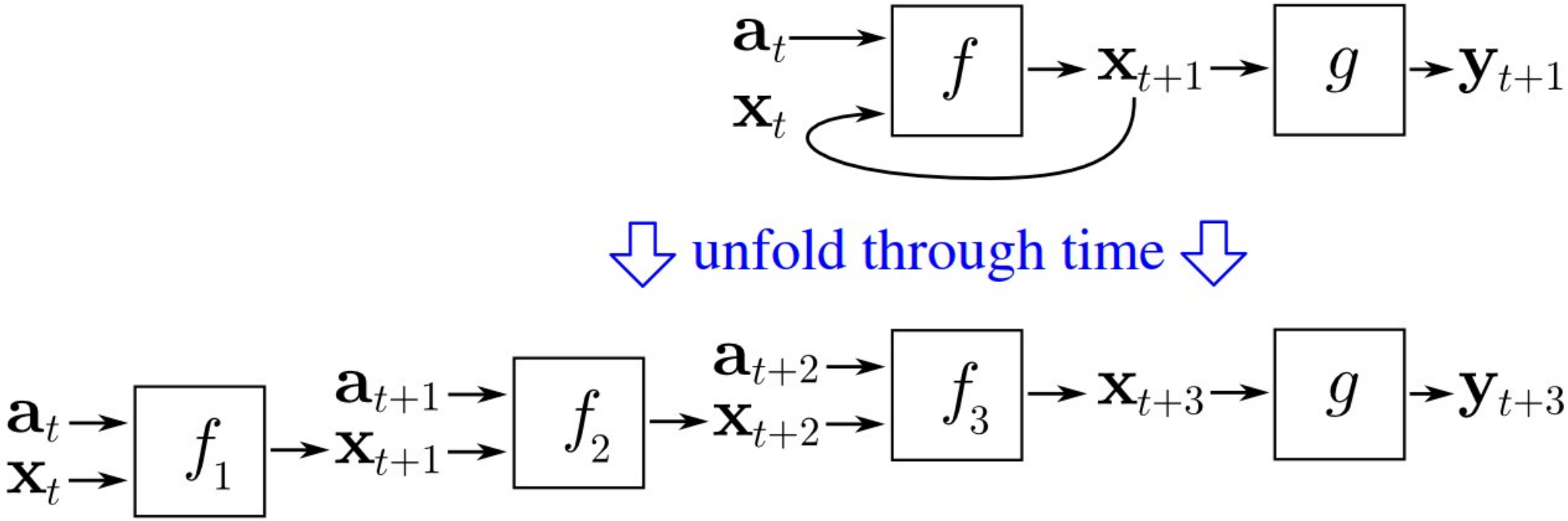
  - BPTT – backpropagation through time

    Similar to FFNN (#layers = #words) with shared weights (same weights in all layers)

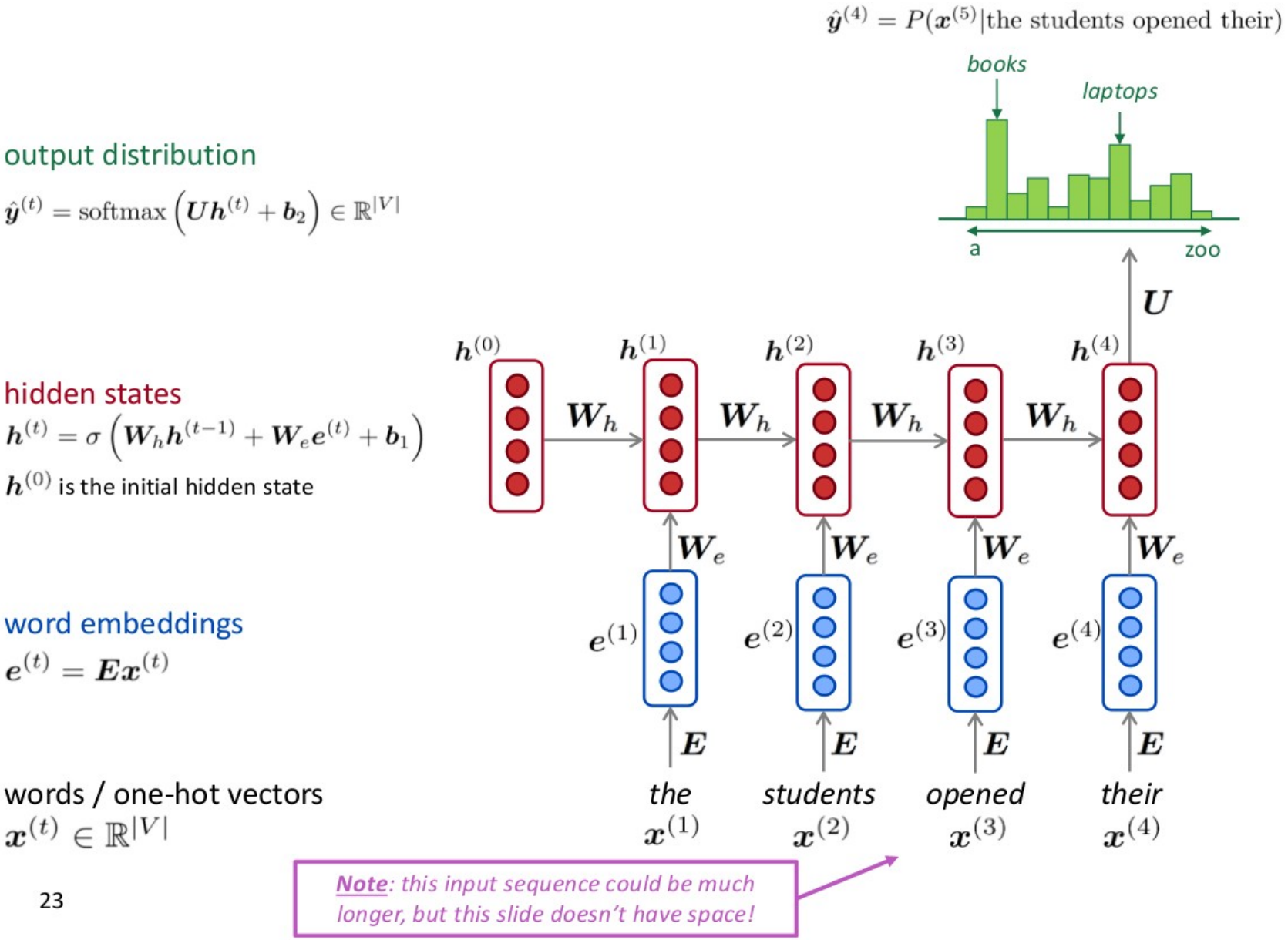
- Truncated BPTT is used in practice
  - Forward-backward pass on segments of seqlen (50-500) words
  - Little better to use final hidden state from the previous segment as initial hidden state for the next segment (0 for the first segment)

Unrolled Networks as Computation Graphs

- With modern computational frameworks explicitly unrolling a recurrent network into a deep feedforward computational graph is practical for word-by-word approaches to sentence-level processing.

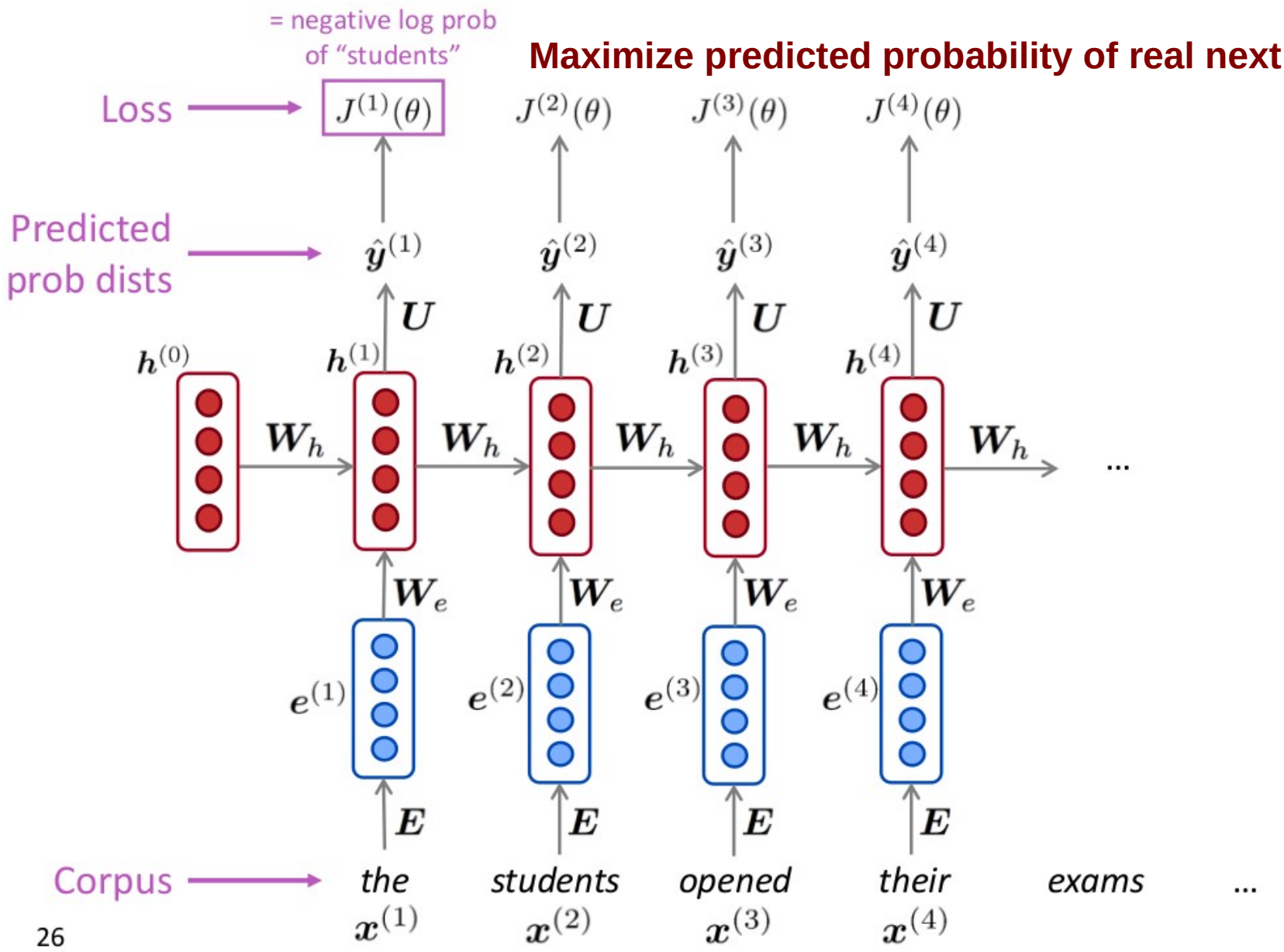


A RNN Language Model



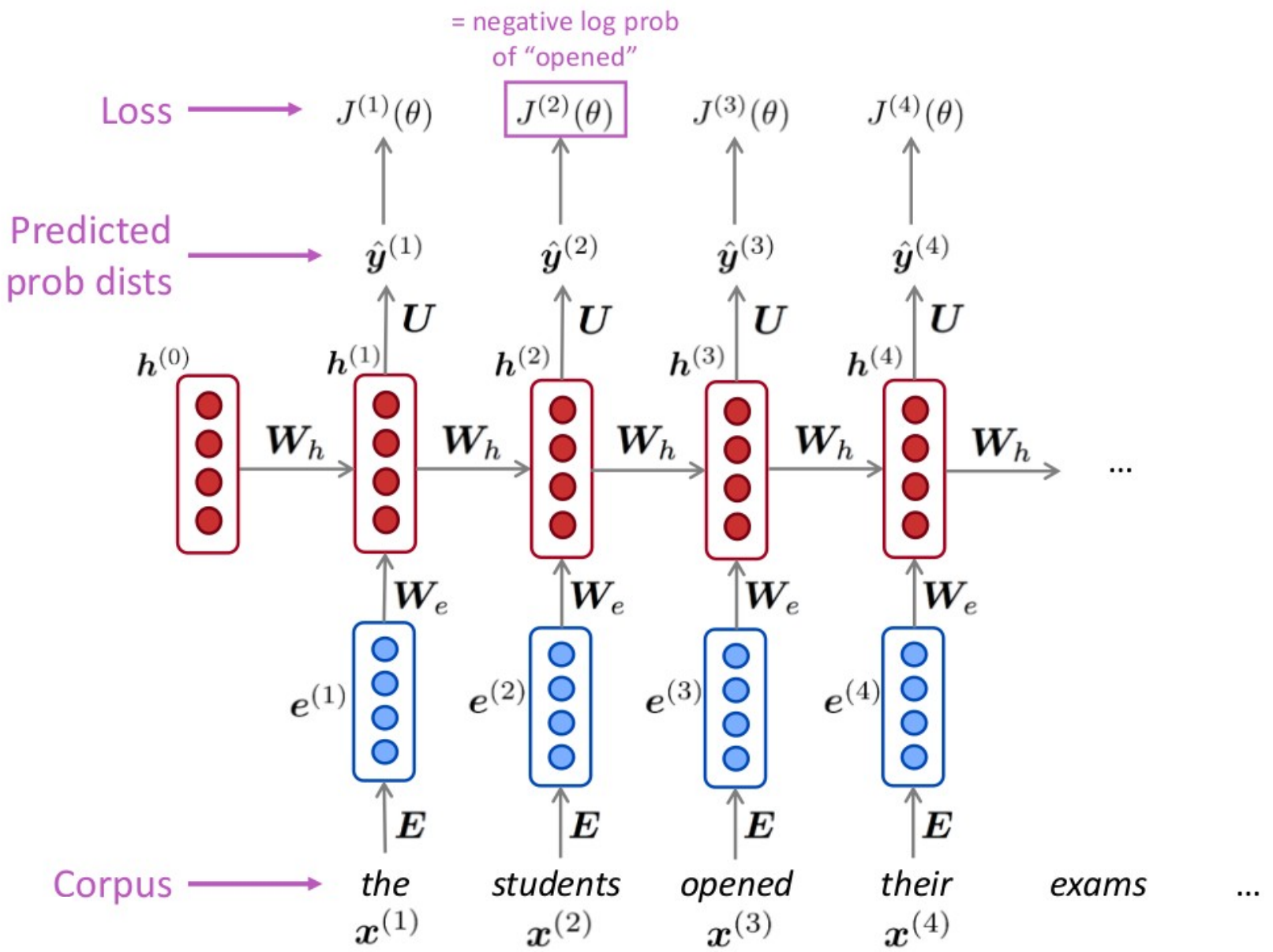
Source: Abigail See. CS224N/Ling284 slides: http://web.stanford.edu/class/cs224n/

Training a RNN Language Model



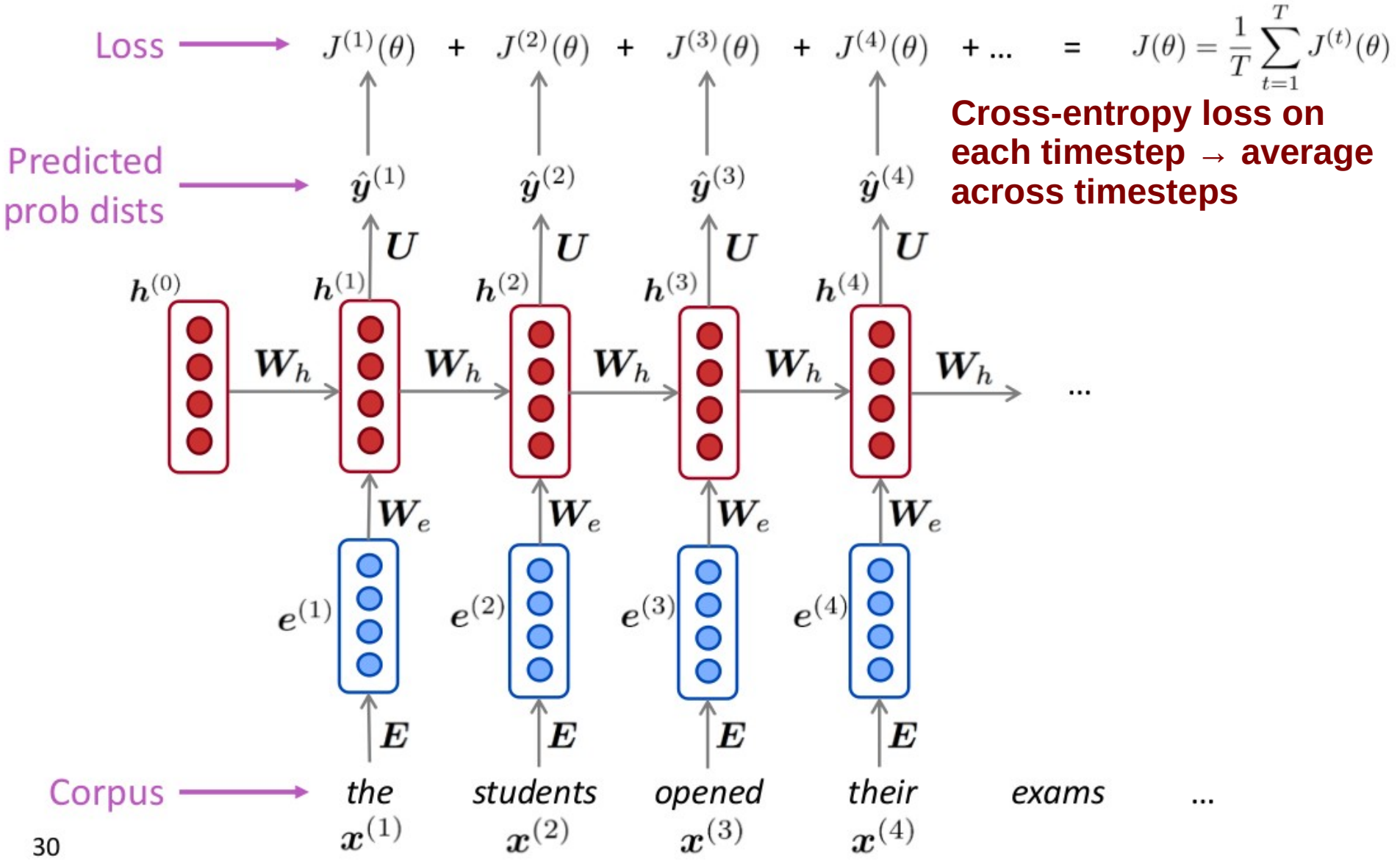
Source: Abigail See. CS224N/Ling284 slides: http://web.stanford.edu/class/cs224n/

Training a RNN Language Model



Source: Abigail See. CS224N/Ling284 slides: http://web.stanford.edu/class/cs224n/

Training a RNN Language Model

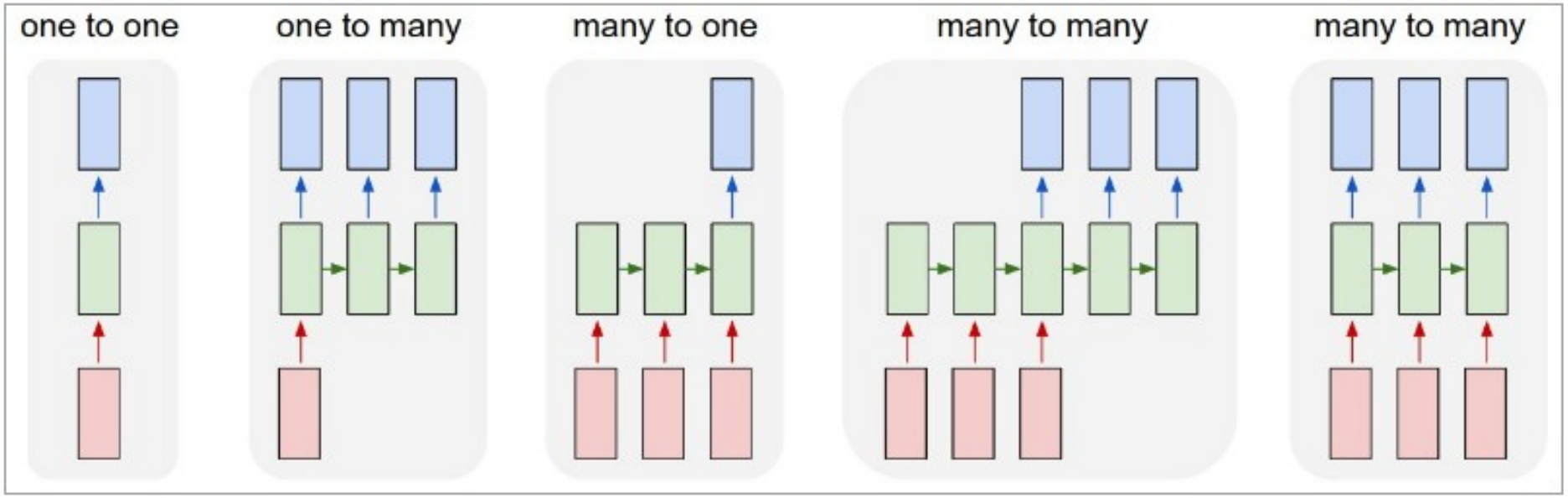


Cross-entropy loss on each timestep → average across timesteps

Source: Abigail See. CS224N/Ling284 slides: http://web.stanford.edu/class/cs224n/

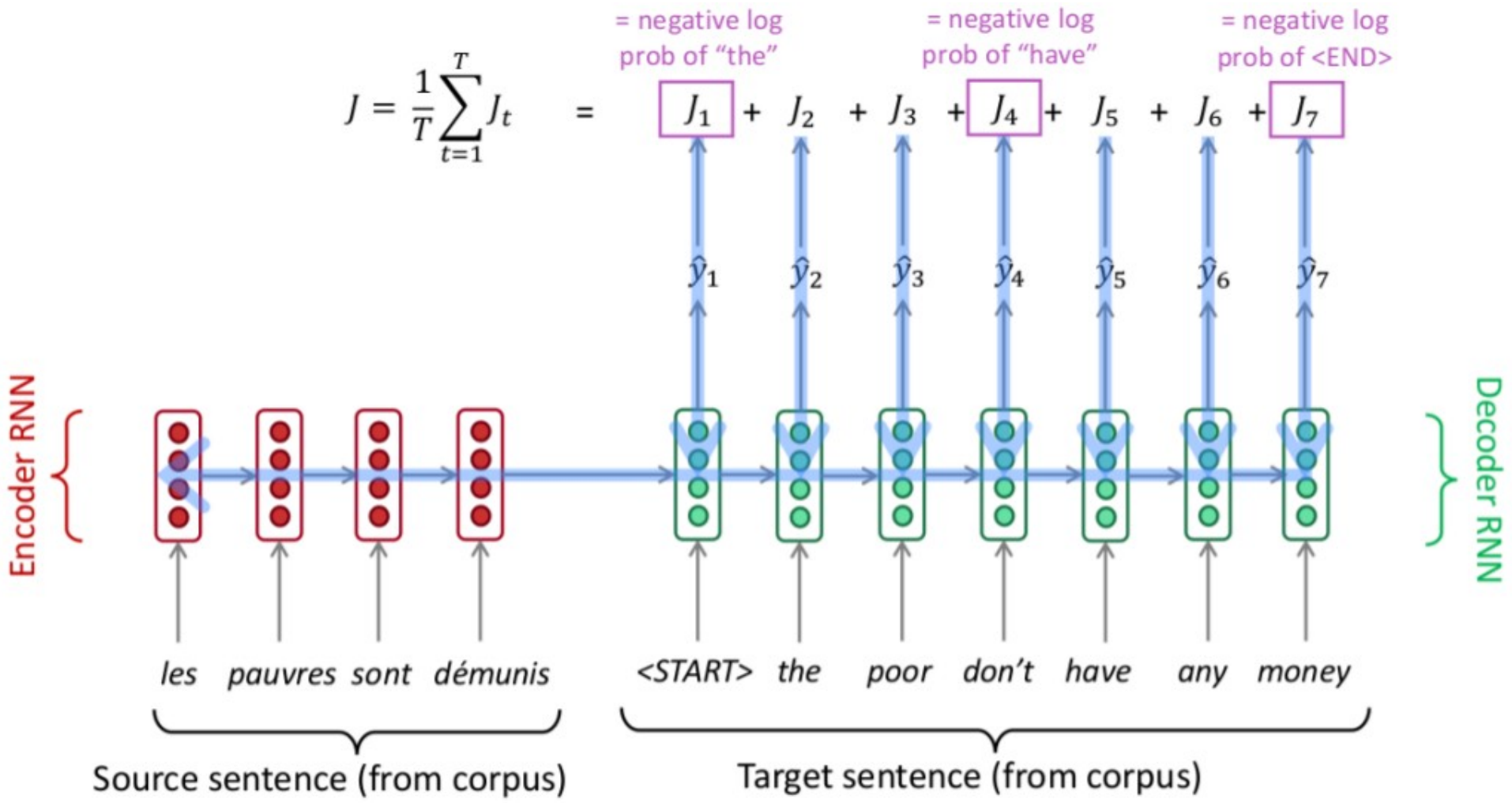
Applications of Recurrent NNs

- **1 → 1**: FFNN
- **1 → many**: conditional generation (image captioning)
- **many → 1**: text classification
- **many → many**:
  - Non-aligned: sequence transduction (machine translation, summarization)
  - Aligned: sequence tagging (POS, NER,Argument Mining, ...)



Source: Karpathy. The Unreasonable Effectiveness of Recurrent Neural Networks
http://karpathy.github.io/2015/05/21/rnn-effectiveness/

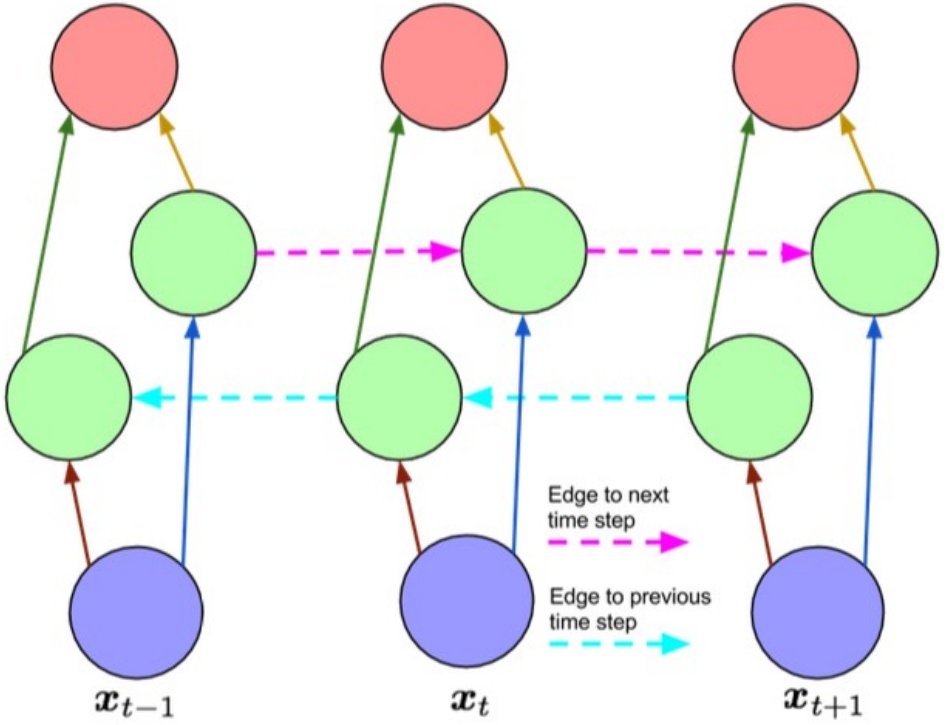
seq2seq



Source: Abigail See. CS224N/Ling284 slides: http://web.stanford.edu/class/cs224n/

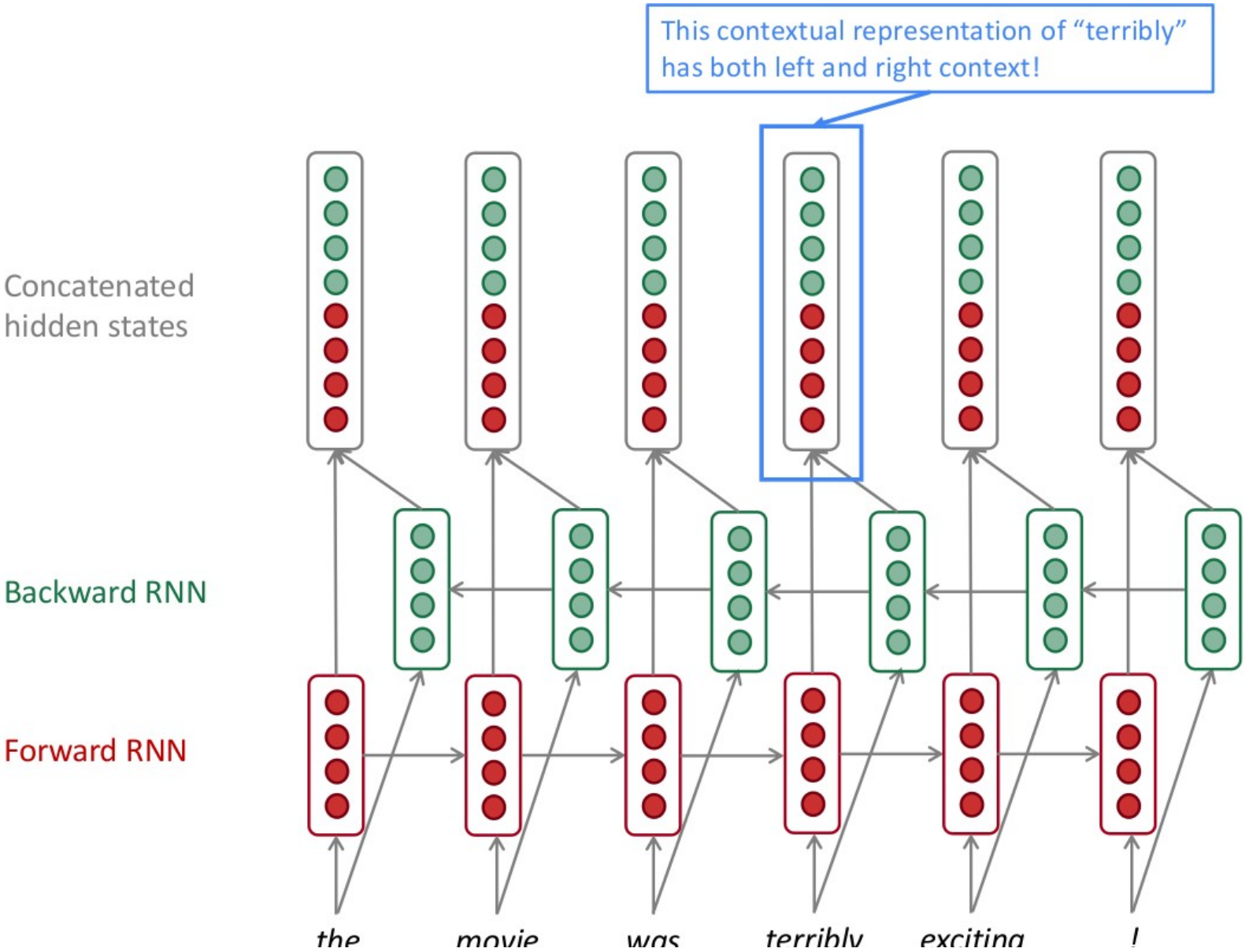
Bidirectional RNNs

- Idea: if we are tagging whole sentences, we can use context representations from the ‘past’ and from the ‘future’ to predict the ‘current’ label
- Not applicable in an online incremental setting.
- LSTM cells and bidirectional networks can be combined into Bi-LSTMs



Bidirectional recurrent network, unfolded in time

Bidirectional RNNs



Source: Abigail See. CS224N/Ling284 slides: http://web.stanford.edu/class/cs224n/

Bidirectional RNNs

On timestep t:

This is a general notation to mean "compute one forward step of the RNN" – it could be a vanilla, LSTM or GRU computation.

Forward RNN $\overrightarrow{\boldsymbol{h}}^{(t)} = \boxed{\mathrm{RNN}_{\mathrm{FW}}}(\overrightarrow{\boldsymbol{h}}^{(t-1)}, \boldsymbol{x}^{(t)})$

Backward RNN $\overleftarrow{\boldsymbol{h}}^{(t)} = \mathrm{RNN}_{\mathrm{BW}}(\overleftarrow{\boldsymbol{h}}^{(t+1)}, \boldsymbol{x}^{(t)})$

Generally, these two RNNs have separate weights

Concatenated hidden states $\boxed{\boldsymbol{h}^{(t)}} = [\overrightarrow{\boldsymbol{h}}^{(t)}; \overleftarrow{\boldsymbol{h}}^{(t)}]$

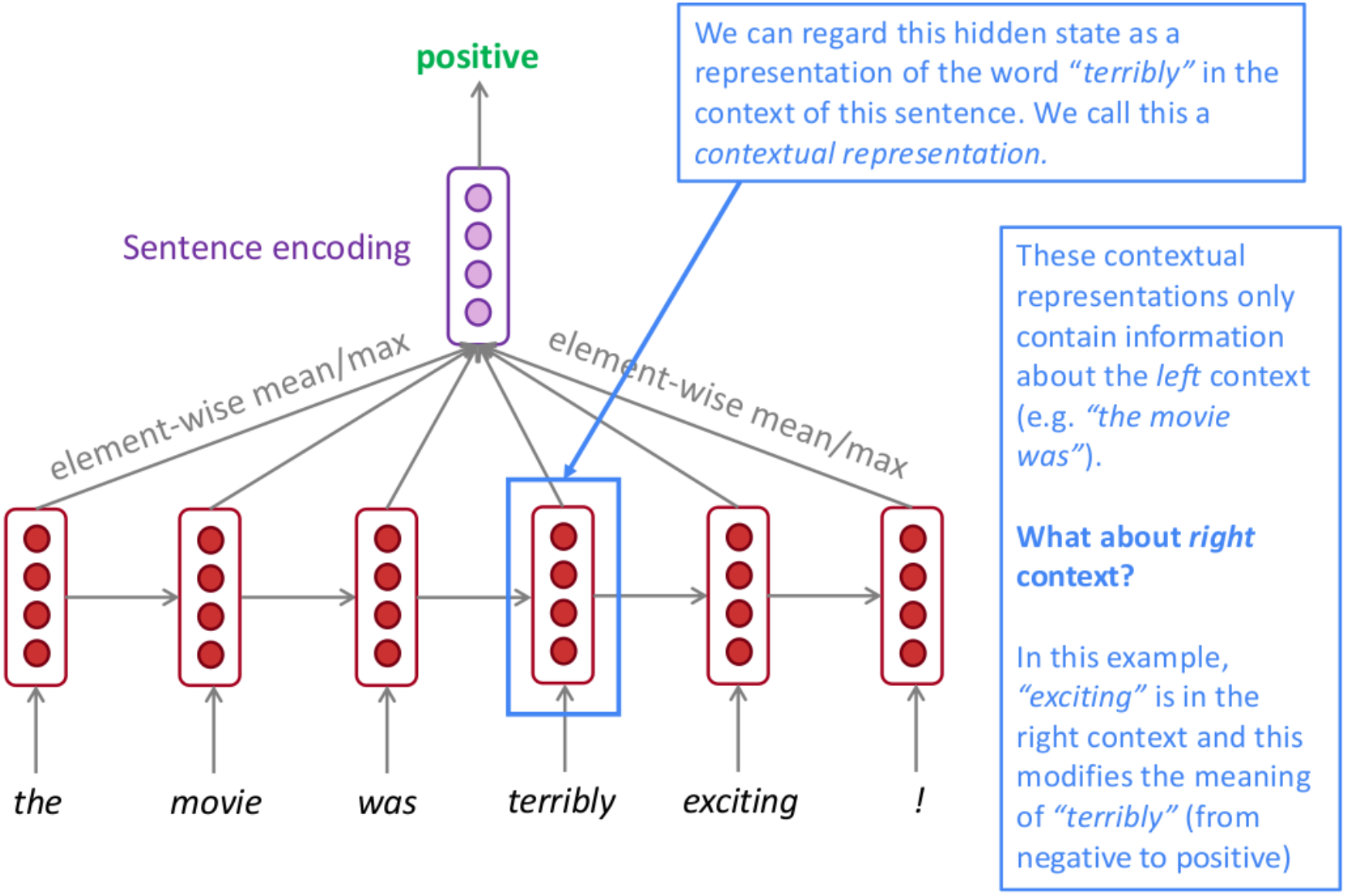
We regard this as "the hidden state" of a bidirectional RNN. This is what we pass on to the next parts of the network.

Require full sequence available=> not for LMs

But similar bidirectional LMs exists which are 2 independent LMs

Source: Abigail See. CS224N/Ling284 slides: http://web.stanford.edu/class/cs224n/

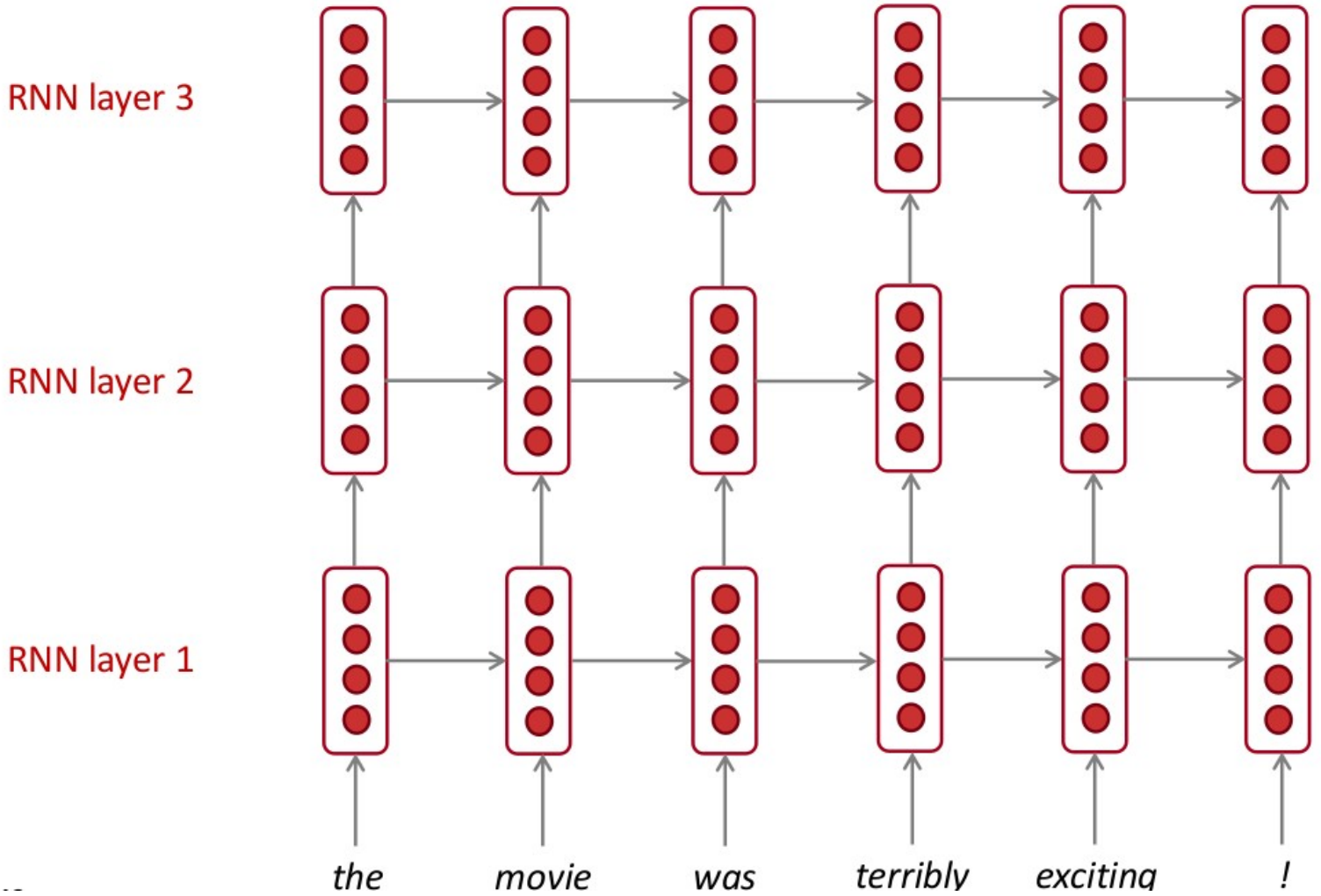
Bidirectional RNNs



Source: Abigail See. CS224N/Ling284 slides: http://web.stanford.edu/class/cs224n/

Multi-layer RNNs

The hidden states from RNN layer *i* are the inputs to RNN layer *i+1*



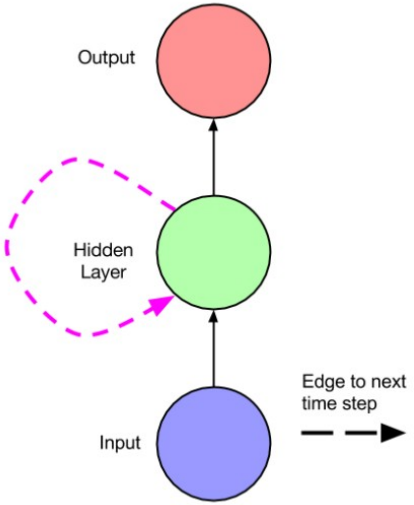
Source: Abigail See. CS224N/Ling284 slides: http://web.stanford.edu/class/cs224n/

The Problem with Vanilla RNNs (or Elman/Simple RNNs)

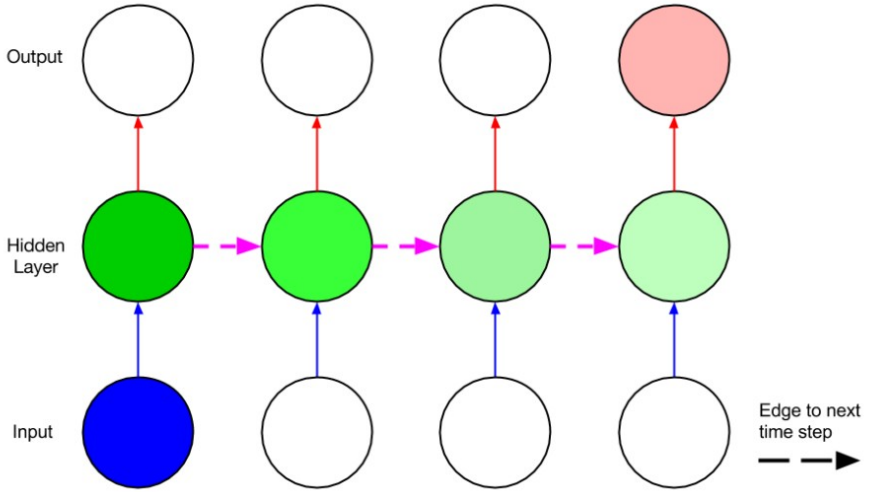
- The inability to retain information for long-range predictions:
  - at each time step we simply updated the hidden state vector regardless of whether it made sense;
  - RNN has no control over which values are retained and which are discarded in the hidden state;
    - that is entirely determined by the input;
    - no way to decide if the update is optional or not.
- Gradient stability:
  - tendency to cause gradients to spiral out of control to zero or to infinity;
  - large absolute value of the gradient or a really small (less than 1) value can make the optimization procedure unstable: (Hochreiter et al., 2001; Pascanu et al., 2013)

Source: Rao D. & McMahan (2019): Natural Language Processing with PyTorch: Build Intelligent Language Applications Using Deep Learning.

The Problem with Vanilla RNNs (or Elman/Simple RNNs)



Simple recurrent network



Unfolded network, visualizing the vanishing gradient

- **Gradients vanish (explode) exponentially** across time steps when the recurrent connection is <1 (>1)
- Problem is connected to the fact that it is always the **same connection weight**
- In the same way a **product of n real numbers can shrink to zero or explode** to infinity, so does this product of matrices
- See details in the papers below:
  - Pascanu, R., Tomas M., and Yoshua B. On the difficulty of training recurrent neural networks. ICML 2013
  - Graves A. Supervised sequence labelling with recurrent neural networks, Volume 385. Springer, 2012.

The Problem with Vanilla RNNs (or Elman/Simple RNNs)

- Vanishing/exploding gradients solutions:
  - Vanishing gradients:
    - **LSTM/GRU** cells
    - ...and other **gated** cells
  - Exploding gradients:
    - Gradient norm **clipping**

Algorithm 1 Pseudo-code for norm clipping

$\hat{\mathbf{g}} \leftarrow \frac{\partial \mathcal{E}}{\partial \theta}$
if $\|\hat{\mathbf{g}}\| \geq threshold$ **then**
  $\hat{\mathbf{g}} \leftarrow \frac{threshold}{\|\hat{\mathbf{g}}\|} \hat{\mathbf{g}}$
end if

Source: Pascanu et al. (2013): On the difficulty of training recurrent neural networks.

Exploding gradients in RNNs

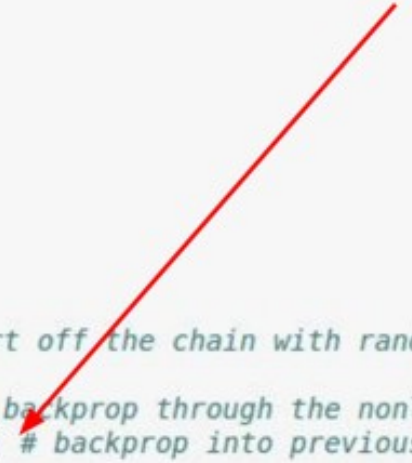
- The gradient signal going backwards in time through all the hidden states is always being multiplied by the same matrix (the recurrence matrix Whh), interspersed with non-linearity backprop
  - Here we only compute the recurrence on the hidden state (equivalently, the input x could always be zero).

```
H = 5    # dimensionality of hidden state
T = 50   # number of time steps
Whh = np.random.randn(H,H)

# forward pass of an RNN (ignoring inputs x)
hs = {}
ss = {}
hs[-1] = np.random.randn(H)
for t in xrange(T):
    ss[t] = np.dot(Whh, hs[t-1])
    hs[t] = np.maximum(0, ss[t])

# backward pass of the RNN
dhs = {}
dss = {}
dhs[T-1] = np.random.randn(H) # start off the chain with random gradient
for t in reversed(xrange(T)):
    dss[t] = (hs[t] > 0) * dhs[t] # backprop through the nonlinearity
    dhs[t-1] = np.dot(Whh.T, dss[t]) # backprop into previous hidden state
```

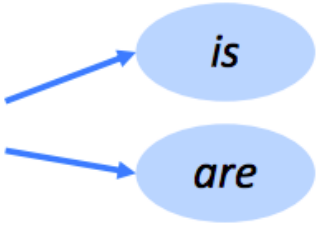
if the largest eigenvalue is > 1, gradient will explode
if the largest eigenvalue is < 1, gradient will vanish



Source: https://medium.com/@karpathy/yes-you-should-understand-backprop-e2f06eab496b

Effect of vanishing gradient on RNN language model

- **LM task:** *The writer of the books ___*



- **Correct answer**: *The writer of the books is planning a sequel*
- **Syntactic recency:** *The writer of the books is* (correct)
- **Sequential recency:** *The writer of the books are* (incorrect)
- Due to vanishing gradient, RNN-LMs are better at learning from sequential recency than syntactic recency, so they make this type of error more often than we'd like [Linzen et al 2016]

Source: Abigail See. CS224N/Ling284 slides: http://web.stanford.edu/class/cs224n/

Plan of the lecture

- **Part 1**: Language modeling.
- **Part 2**: Recurrent neural networks.
- **Part 3**: Long-Short Term Memory (LSTM).
- **Part 4**: LSTMs for sequence labelling.
- **Part 5**: LSTMs for text categorization.

Intuition behind the gating mechanism

- Suppose that you were adding two quantities, a and b, but you wanted to control how much of b gets into the sum:

$$a + \lambda b$$

- λ is a value between 0 and 1.
- λ acts as a "switch" or a "gate" in controlling the amount of b that gets into the sum.

Source: Rao D. & McMahan (2019): Natural Language Processing with PyTorch: Build Intelligent Language Applications Using Deep Learning.

A simple gate example

- Elman RNN:

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

- A gated version of Elman RNN:

$$h_t = h_{t-1} + \lambda(h_{t-1}, x_t) F(h_{t-1}, x_t)$$

  - function λ controls how much of the current input gets to update the state h_{t-1};
  - function λ is context-dependent.

- Incorporate not only conditional updates, but also forgetting of the values in the previous state h_{t-1}

$$h_t = \mu(h_{t-1}, x_t) h_{t-1} + \lambda(h_{t-1}, x_t) F(h_{t-1}, x_t)$$

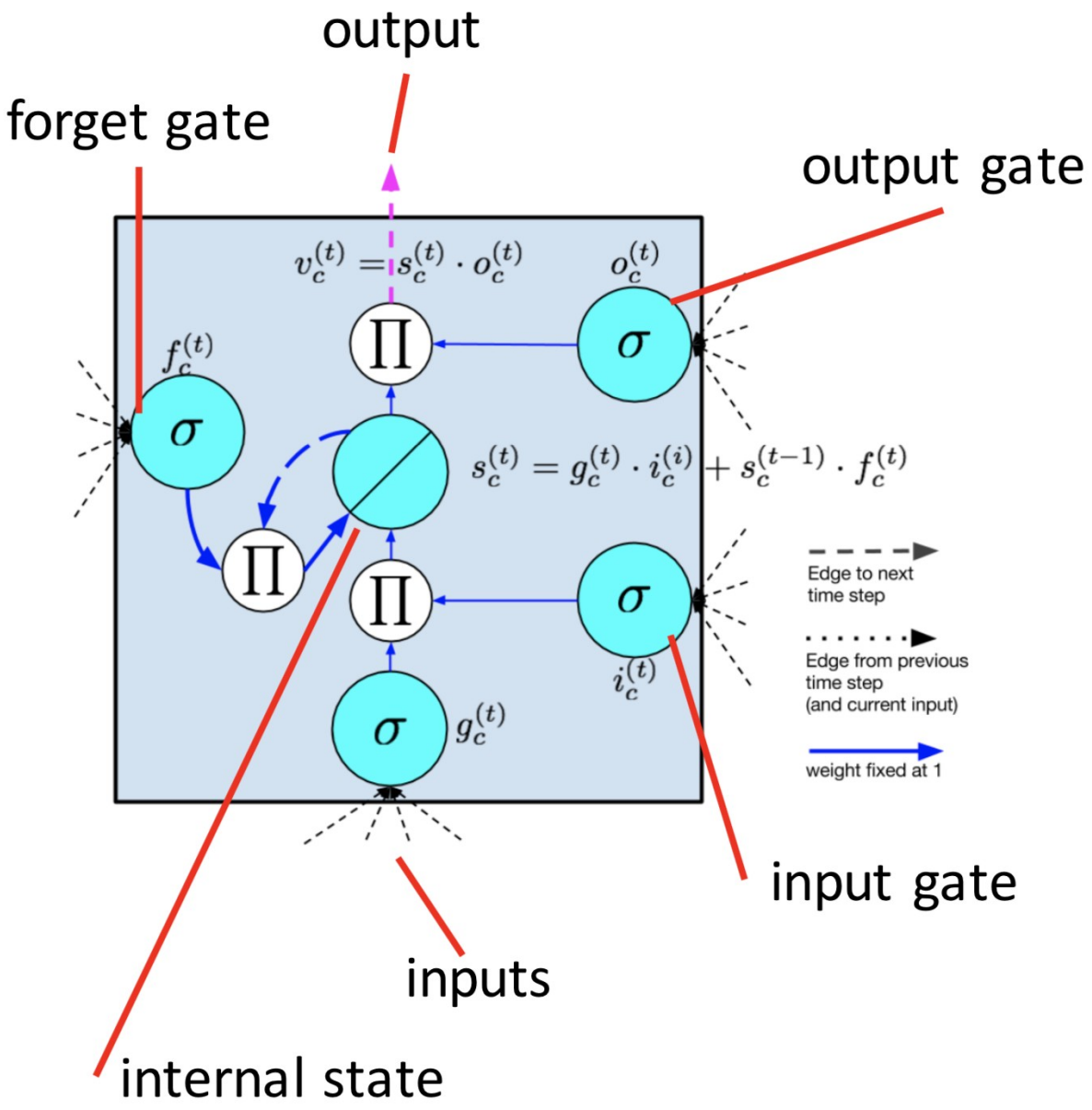
Source: Rao D. & McMahan (2019): Natural Language Processing with PyTorch: Build Intelligent Language Applications Using Deep Learning.

Long Short-Term Memory (LSTM)

- LSTM resembles a standard RNN with a hidden layer
- Nodes in the hidden layer are replaced by a memory cell
- Memory cells contain a node with a self-connected recurrent edge of fixed weight 1 (no gradient issues)

- Hochreiter S. and Schmidhuber H. (1997). Long short-term memory. Neural Computation, 9(8):1735–1780.
- Greff, K., Srivastava, R. K., Koutník, J., Steunebrink, B. R., & Schmidhuber, J. (2017). LSTM: A search space odyssey. IEEE Trans. on neural networks and learning systems, 28(10)

Memory Cell in LSTM



- **inputs**: from sequence and from other memory cells
- **input gate**: regulates whether to take input into account
- **output gate**: regulates whether to output the internal state
- **forget gate**: can flush internal state
- **recurrent link** with weight 1: “constant error carousel”.

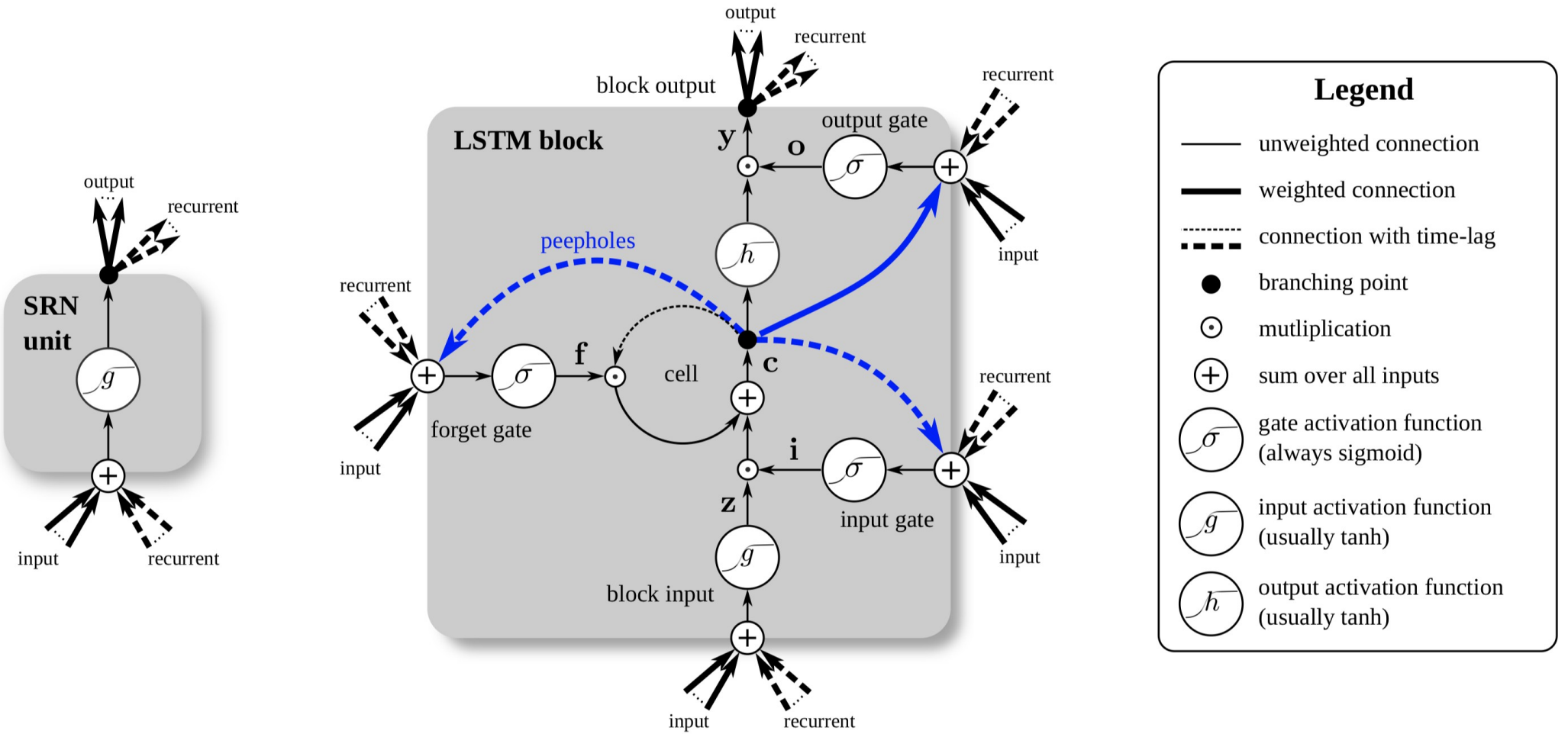
LSTM Intuitions

- "**long short-term memory**": standard NNs have
  - long-term memory in the weights
  - short-term memory in the activations
  - LSTM mixes both notions

- **Gate**: pointwise multiplication regulates how much is passed through, based on inputs
- Internal state serves as a **memory**
- Recurrent connection of weight 1: error can flow across time steps without vanishing or exploding

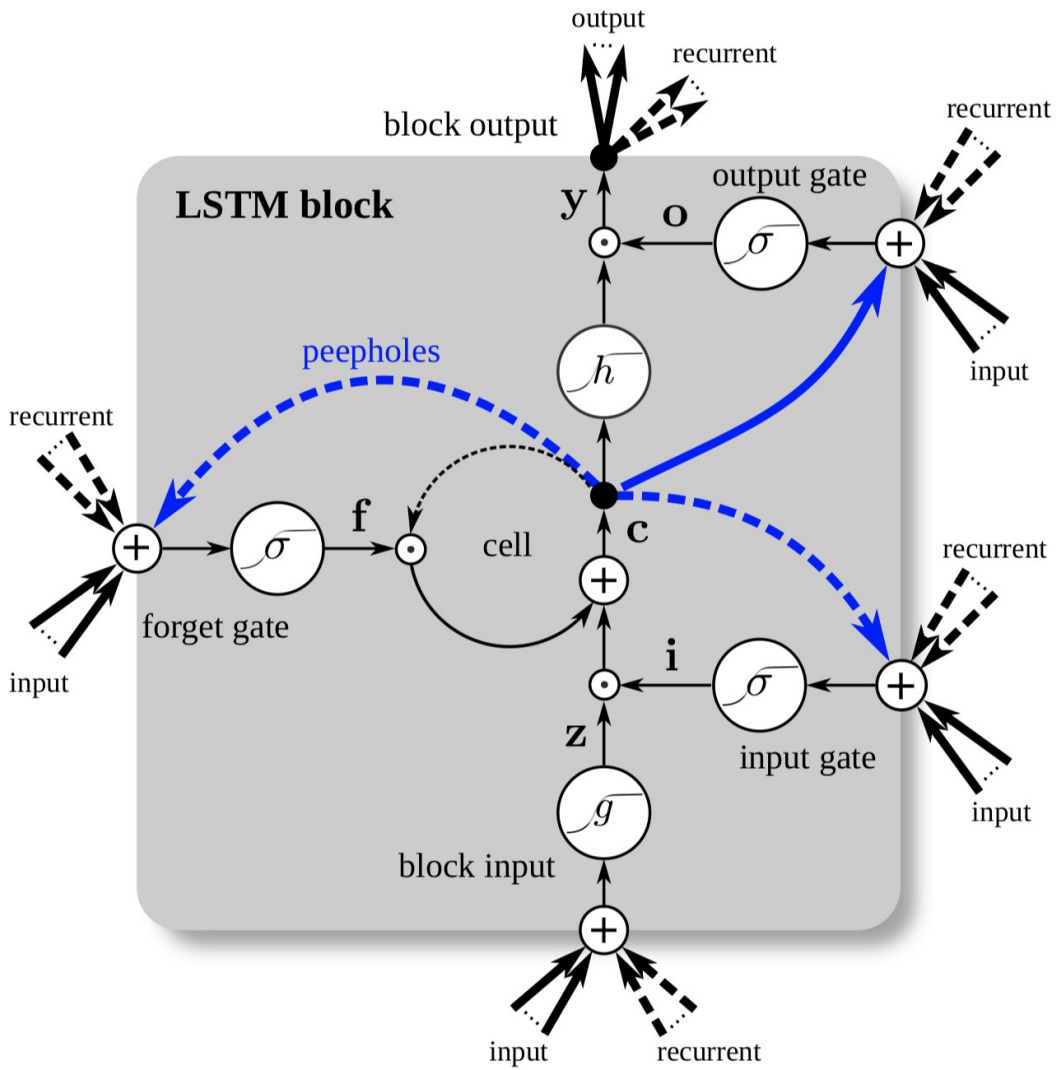
- LSTM can learn:
  - when to let the input (and error) in *e.g. set the new grammatical subject*
  - when to let the output (and error) out *e.g. predict verb that takes the subject*
  - when to reset its memory *e.g. remove old subject once its taken*

Long Short-Term Memory (LSTM)



Source: Greff, K., Srivastava, R. K., Koutník, J., Steunebrink, B. R., & Schmidhuber, J. (2017). LSTM: A search space odyssey. IEEE Trans. on neural networks and learning systems, 28(10)

Long Short-Term Memory (LSTM)



- Input weights: $\mathbf{W}_z, \mathbf{W}_i, \mathbf{W}_f, \mathbf{W}_o \in \mathbb{R}^{N \times M}$
- Recurrent weights: $\mathbf{R}_z, \mathbf{R}_i, \mathbf{R}_f, \mathbf{R}_o \in \mathbb{R}^{N \times N}$
- Peephole weights: $\mathbf{p}_i, \mathbf{p}_f, \mathbf{p}_o \in \mathbb{R}^{N}$
- Bias weights: $\mathbf{b}_z, \mathbf{b}_i, \mathbf{b}_f, \mathbf{b}_o \in \mathbb{R}^{N}$

$$\bar{\mathbf{z}}^t = \mathbf{W}_z \mathbf{x}^t + \mathbf{R}_z \mathbf{y}^{t-1} + \mathbf{b}_z$$
$$\mathbf{z}^t = g(\bar{\mathbf{z}}^t) \quad \textit{block input}$$
$$\bar{\mathbf{i}}^t = \mathbf{W}_i \mathbf{x}^t + \mathbf{R}_i \mathbf{y}^{t-1} + \mathbf{p}_i \odot \mathbf{c}^{t-1} + \mathbf{b}_i$$
$$\mathbf{i}^t = \sigma(\bar{\mathbf{i}}^t) \quad \textit{input gate}$$
$$\bar{\mathbf{f}}^t = \mathbf{W}_f \mathbf{x}^t + \mathbf{R}_f \mathbf{y}^{t-1} + \mathbf{p}_f \odot \mathbf{c}^{t-1} + \mathbf{b}_f$$
$$\mathbf{f}^t = \sigma(\bar{\mathbf{f}}^t) \quad \textit{forget gate}$$
$$\mathbf{c}^t = \mathbf{z}^t \odot \mathbf{i}^t + \mathbf{c}^{t-1} \odot \mathbf{f}^t \quad \textit{cell}$$
$$\bar{\mathbf{o}}^t = \mathbf{W}_o \mathbf{x}^t + \mathbf{R}_o \mathbf{y}^{t-1} + \mathbf{p}_o \odot \mathbf{c}^t + \mathbf{b}_o$$
$$\mathbf{o}^t = \sigma(\bar{\mathbf{o}}^t) \quad \textit{output gate}$$
$$\mathbf{y}^t = h(\mathbf{c}^t) \odot \mathbf{o}^t \quad \textit{block output}$$

Source: Greff, K., Srivastava, R. K., Koutník, J., Steunebrink, B. R., & Schmidhuber, J. (2017). LSTM: A search space odyssey. IEEE Trans. on neural networks and learning systems, 28(10)

Long Short-Term Memory (LSTM)

LEARNABLE PARAMETERS

- Input weights: $\mathbf{W}_z$, $\mathbf{W}_i$, $\mathbf{W}_f$, $\mathbf{W}_o \in \mathbb{R}^{N \times M}$
- Recurrent weights: $\mathbf{R}_z$, $\mathbf{R}_i$, $\mathbf{R}_f$, $\mathbf{R}_o \in \mathbb{R}^{N \times N}$
- Peephole weights: $\mathbf{p}_i$, $\mathbf{p}_f$, $\mathbf{p}_o \in \mathbb{R}^{N}$
- Bias weights: $\mathbf{b}_z$, $\mathbf{b}_i$, $\mathbf{b}_f$, $\mathbf{b}_o \in \mathbb{R}^{N}$

FORWARD PASS

$$\bar{\mathbf{z}}^t = \mathbf{W}_z \mathbf{x}^t + \mathbf{R}_z \mathbf{y}^{t-1} + \mathbf{b}_z$$
$$\mathbf{z}^t = g(\bar{\mathbf{z}}^t) \qquad \textit{block input}$$
$$\bar{\mathbf{i}}^t = \mathbf{W}_i \mathbf{x}^t + \mathbf{R}_i \mathbf{y}^{t-1} + \mathbf{p}_i \odot \mathbf{c}^{t-1} + \mathbf{b}_i$$
$$\mathbf{i}^t = \sigma(\bar{\mathbf{i}}^t) \qquad \textit{input gate}$$
$$\bar{\mathbf{f}}^t = \mathbf{W}_f \mathbf{x}^t + \mathbf{R}_f \mathbf{y}^{t-1} + \mathbf{p}_f \odot \mathbf{c}^{t-1} + \mathbf{b}_f$$
$$\mathbf{f}^t = \sigma(\bar{\mathbf{f}}^t) \qquad \textit{forget gate}$$
$$\mathbf{c}^t = \mathbf{z}^t \odot \mathbf{i}^t + \mathbf{c}^{t-1} \odot \mathbf{f}^t \qquad \textit{cell}$$
$$\bar{\mathbf{o}}^t = \mathbf{W}_o \mathbf{x}^t + \mathbf{R}_o \mathbf{y}^{t-1} + \mathbf{p}_o \odot \mathbf{c}^t + \mathbf{b}_o$$
$$\mathbf{o}^t = \sigma(\bar{\mathbf{o}}^t) \qquad \textit{output gate}$$
$$\mathbf{y}^t = h(\mathbf{c}^t) \odot \mathbf{o}^t \qquad \textit{block output}$$

BACKWARD PASS (BPTT)

$$\delta \mathbf{y}^t = \Delta^t + \mathbf{R}_z^T \delta \mathbf{z}^{t+1} + \mathbf{R}_i^T \delta \mathbf{i}^{t+1} + \mathbf{R}_f^T \delta \mathbf{f}^{t+1} + \mathbf{R}_o^T \delta \mathbf{o}^{t+1}$$
$$\delta \bar{\mathbf{o}}^t = \delta \mathbf{y}^t \odot h(\mathbf{c}^t) \odot \sigma'(\bar{\mathbf{o}}^t)$$
$$\delta \mathbf{c}^t = \delta \mathbf{y}^t \odot \mathbf{o}^t \odot h'(\mathbf{c}^t) + \mathbf{p}_o \odot \delta \bar{\mathbf{o}}^t + \mathbf{p}_i \odot \delta \bar{\mathbf{i}}^{t+1} + \mathbf{p}_f \odot \delta \bar{\mathbf{f}}^{t+1} + \delta \mathbf{c}^{t+1} \odot \mathbf{f}^{t+1}$$
$$\delta \bar{\mathbf{f}}^t = \delta \mathbf{c}^t \odot \mathbf{c}^{t-1} \odot \sigma'(\bar{\mathbf{f}}^t)$$
$$\delta \bar{\mathbf{i}}^t = \delta \mathbf{c}^t \odot \mathbf{z}^t \odot \sigma'(\bar{\mathbf{i}}^t)$$
$$\delta \bar{\mathbf{z}}^t = \delta \mathbf{c}^t \odot \mathbf{i}^t \odot g'(\bar{\mathbf{z}}^t)$$

How does LSTM handles the vanishing gradient problem?

- The LSTM architecture makes it easier for the RNN to preserve information over many timesteps
  - e.g. if the forget gate is set to remember everything on every timestep, then the info in the cell is preserved indefinitely
  - By contrast, it's harder for vanilla RNN to learn a recurrent weight matrix W_h that preserves info in hidden state

- LSTM doesn't *guarantee* that there is no vanishing/exploding gradient, but it does provide an easier way for the model to learn long-distance dependencies

Source: Abigail See. CS224N/Ling284 slides: http://web.stanford.edu/class/cs224n/

Examples generated (Shakespeare)

```
PANDARUS:
Alas, I think he shall be come approached and the day
When little srain would be attain'd into being never fed,
And who is but a chain and subjects of his death,
I should not sleep.

Second Senator:
They are away this miseries, produced upon my soul,
Breaking and strongly should be buried, when I perish
The earth and thoughts of many states.

DUKE VINCENTIO:
Well, your wit is in the care of side and that.

Second Lord:
They would be ruled after this chamber, and
my fair nues begun out of the fact, to be conveyed,
Whose noble souls I'll have the heart of the wars.

Clown:
Come, sir, I will make did behold your worship.

VIOLA:
I'll drink it.
```

Source: Karpathy. The Unreasonable Effectiveness of Recurrent Neural Networks
http://karpathy.github.io/2015/05/21/rnn-effectiveness/

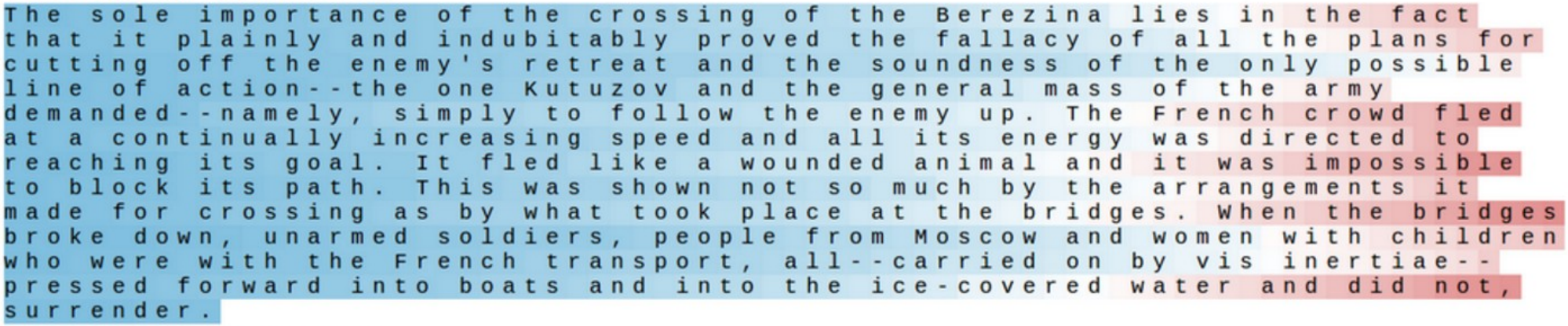
Examples generated (Linux kernel)

```c
/*
 * Increment the size file of the new incorrect UI_FILTER group information
 * of the size generatively.
 */
static int indicate_policy(void)
{
  int error;
  if (fd == MARN_EPT) {
    /*
     * The kernel blank will coeld it to userspace.
     */
    if (ss->segment < mem_total)
      unblock_graph_and_set_blocked();
    else
      ret = 1;
    goto bail;
  }
  segaddr = in_SB(in.addr);
  selector = seg / 16;
  setup_works = true;
  for (i = 0; i < blocks; i++) {
    seq = buf[i++];
    bpf = bd->bd.next + i * search;
    if (fd) {
      current = blocked;
    }
  }
  rw->name = "Getjbbregs";
  bprm_self_clearl(&iv->version);
  regs->new = blocks[(BPF_STATS << info->historidac)] | PFMR_CLOBATHINC_SECONDS << 12;
  return segtable;
}
```

Source: Karpathy. The Unreasonable Effectiveness of Recurrent Neural Networks
http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Cells activations

Cell sensitive to position in line:



Cell that turns on inside quotes:

Source: Karpathy. The Unreasonable Effectiveness of Recurrent Neural Networks
http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Cells are sometimes interpretable

Cell that robustly activates inside if statements:

```
static int __dequeue_signal(struct sigpending *pending, sigset_t *mask,
        siginfo_t *info)
{
  int sig = next_signal(pending, mask);
  if (sig) {
    if (current->notifier) {
      if (sigismember(current->notifier_mask, sig)) {
        if (!(current->notifier)(current->notifier_data)) {
          clear_thread_flag(TIF_SIGPENDING);
          return 0;
        }
      }
    }
    collect_signal(sig, pending, info);
  }
  return sig;
}
```

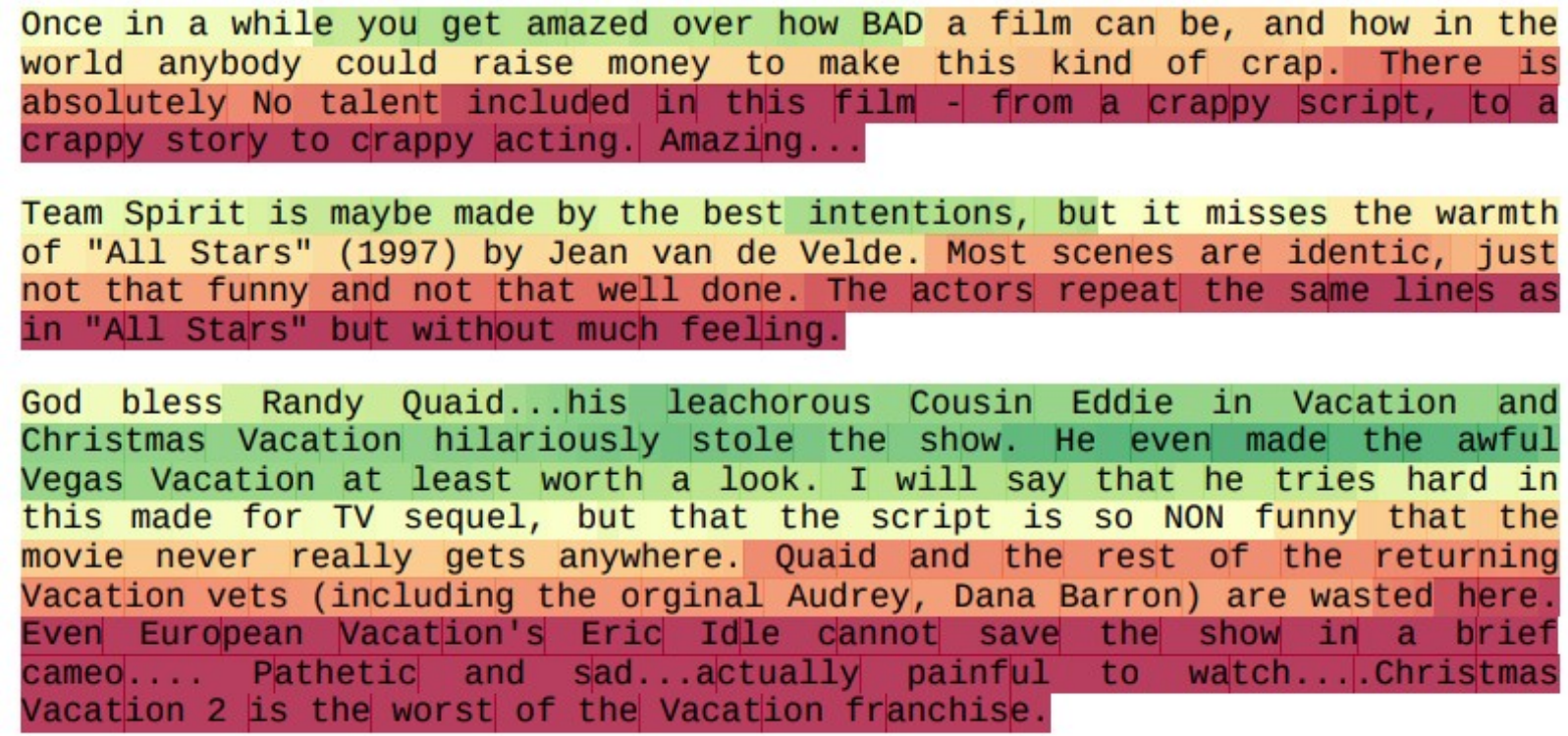
Cell that turns on inside comments and quotes:

```
/* Duplicate LSM field information.  The lsm_rule is opaque, so
 * re-initialized. */
static inline int audit_dupe_lsm_field(struct audit_field *df,
           struct audit_field *sf)
{
  int ret = 0;
  char *lsm_str;
  /* our own copy of lsm_str */
  lsm_str = kstrdup(sf->lsm_str, GFP_KERNEL);
  if (unlikely(!lsm_str))
    return -ENOMEM;
  df->lsm_str = lsm_str;
  /* our own (refreshed) copy of lsm_rule */
  ret = security_audit_rule_init(df->type, df->op, df->lsm_str,
            (void **)&df->lsm_rule);
```

Source: Karpathy. The Unreasonable Effectiveness of Recurrent Neural Networks
http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Sentiment Neuron Visualizations

- How sentiment neuron changes while reading text?



Source: [Radford et al. Learning to Generate Reviews and Discovering Sentiment, 2017]

Other variants of gated cells

- Jozefowicz et al. (2015) tried about 10k various architectures: did not find anything substantially better than LSTM and GRU.

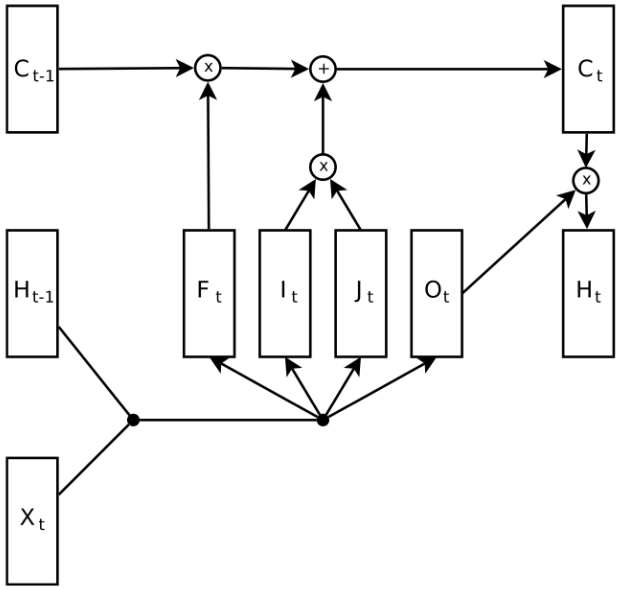


Figure 1. The LSTM architecture. The value of the cell is increased by $i_t \odot j_t$, where $\odot$ is element-wise product. The LSTM's output is typically taken to be h_t, and c_t is not exposed. The forget gate f_t allows the LSTM to easily reset the value of the cell.

$$
\begin{aligned}
i_t &= \tanh(W_{\mathrm{xi}} x_t + W_{\mathrm{hi}} h_{t-1} + b_{\mathrm{i}}) \\
j_t &= \mathrm{sigm}(W_{\mathrm{xj}} x_t + W_{\mathrm{hj}} h_{t-1} + b_{\mathrm{j}}) \\
f_t &= \mathrm{sigm}(W_{\mathrm{xf}} x_t + W_{\mathrm{hf}} h_{t-1} + b_{\mathrm{f}}) \\
o_t &= \tanh(W_{\mathrm{xo}} x_t + W_{\mathrm{ho}} h_{t-1} + b_{\mathrm{o}}) \\
c_t &= c_{t-1} \odot f_t + i_t \odot j_t \\
h_t &= \tanh(c_t) \odot o_t
\end{aligned}
$$

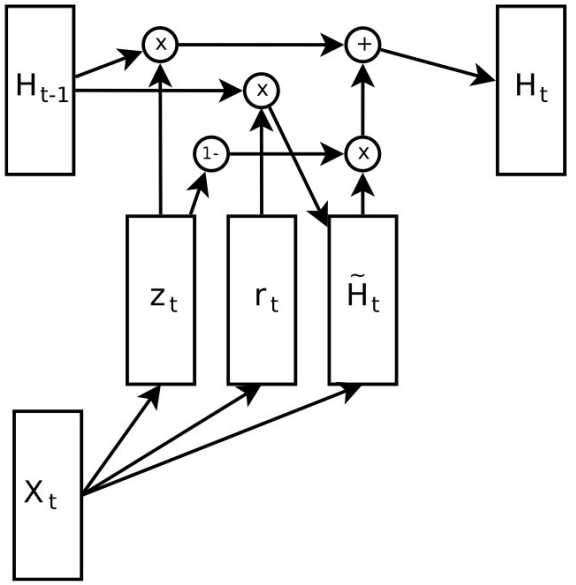


Figure 2. The Gated Recurrent Unit. Like the LSTM, it is hard to tell, at a glance, which part of the GRU is essential for its functioning.

$$
\begin{aligned}
r_t &= \mathrm{sigm}\left(W_{\mathrm{xr}} x_t + W_{\mathrm{hr}} h_{t-1} + b_{\mathrm{r}}\right) \\
z_t &= \mathrm{sigm}(W_{\mathrm{xz}} x_t + W_{\mathrm{hz}} h_{t-1} + b_{\mathrm{z}}) \\
\tilde{h}_t &= \tanh(W_{\mathrm{xh}} x_t + W_{\mathrm{hh}} (r_t \odot h_{t-1}) + b_{\mathrm{h}}) \\
h_t &= z_t \odot h_{t-1} + (1 - z_t) \odot \tilde{h}_t
\end{aligned}
$$

Source: http://proceedings.mlr.press/v37/jozefowicz15.pdf

Plan of the lecture

- **Part 1**: Language modeling.
- **Part 2**: Recurrent neural networks.
- **Part 3**: Long-Short Term Memory (LSTM).
- **Part 4**: LSTMs for sequence labelling.
- **Part 5**: LSTMs for text categorization.

Sequence tagging

- We want to know properties of words for further processing, e.g. word classes, names, etc.
- It is possible to learn a method that assigns these properties from **labeled training text**.
- In Machine Learning, this is a classification task. If the sequence of events is taken into account, this is called **sequence tagging**

Examples for tagged text:

- Part-of-Speech:
  I/**PRO** saw/**V** the/**DET** man/**N** with/**P** the/**DET** saw/**N** ./**P**
- Name tagging:
  Valerie/**B-PERS** and/**O** Rose/**B-PERS** travel/**O**
  to/**O** New/**B-LOC** York/**I-LOC** ./**O**

No independence assumption on data samples

- Standard ML setups: Assumption on the independence of training resp. test examples
  - Can shuffle and sample training examples
  - Can classify test examples in parallel
- Sequence Learning
  - Previous train/test examples are an informative context
  - Previous classifications/outputs are an informative context
- Examples for sequential data:
  - Frames from video
  - Snippets from audio
  - Text: streams of words or characters
  - DNA

The part-of-speech (POS) tagging: solving morphological ambiguity

Words often have more than one POS: back

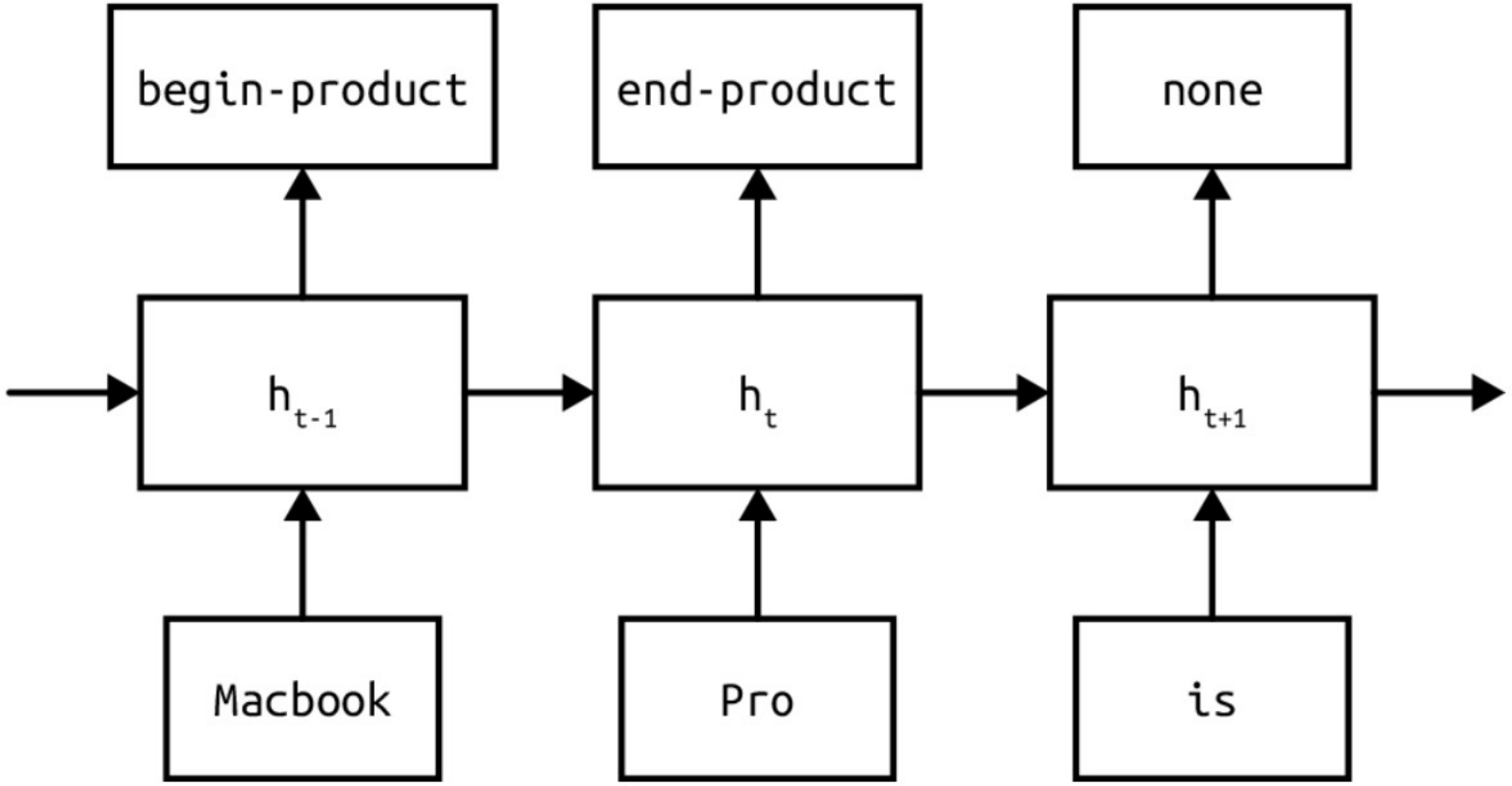
- The *back* door = JJ
- On my *back* = NN
- Win the voters *back* = RB
- Promised to *back* the bill = VB

$$L_{\max} = (l^1_{\max}, l^2_{\max}, \ldots l^T_{\max}) = \underset{L}{\operatorname{argmax}} P(L \mid W)$$

The POS tagging problem is to determine the POS tag label sequence *L* for a particular sequence of words *W*:

Named Entity Recognition (NER)

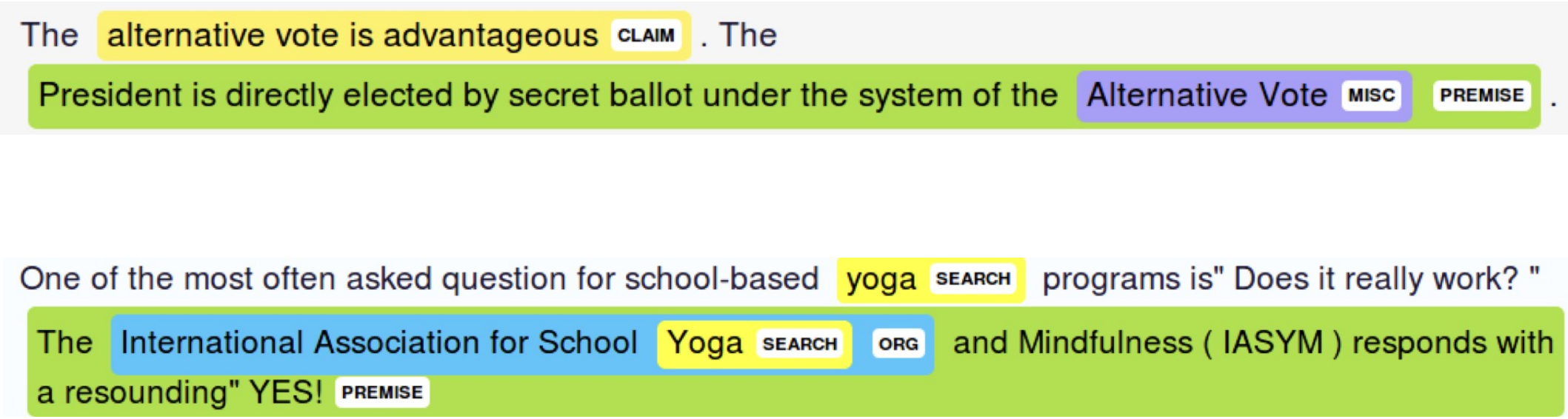
- $[\text{Jim}]_{\text{Person}}$ bought 300 shares of $[\text{Acme Corp.}]_{\text{Organization}}$ in $[2006]_{\text{Time}}$.



Source: Rao D. & McMahan (2019): Natural Language Processing with PyTorch: Build Intelligent Language Applications Using Deep Learning.

Argument Mining

- Premise-Claim model example annotations:



POS tagging and other sequence labelling problems

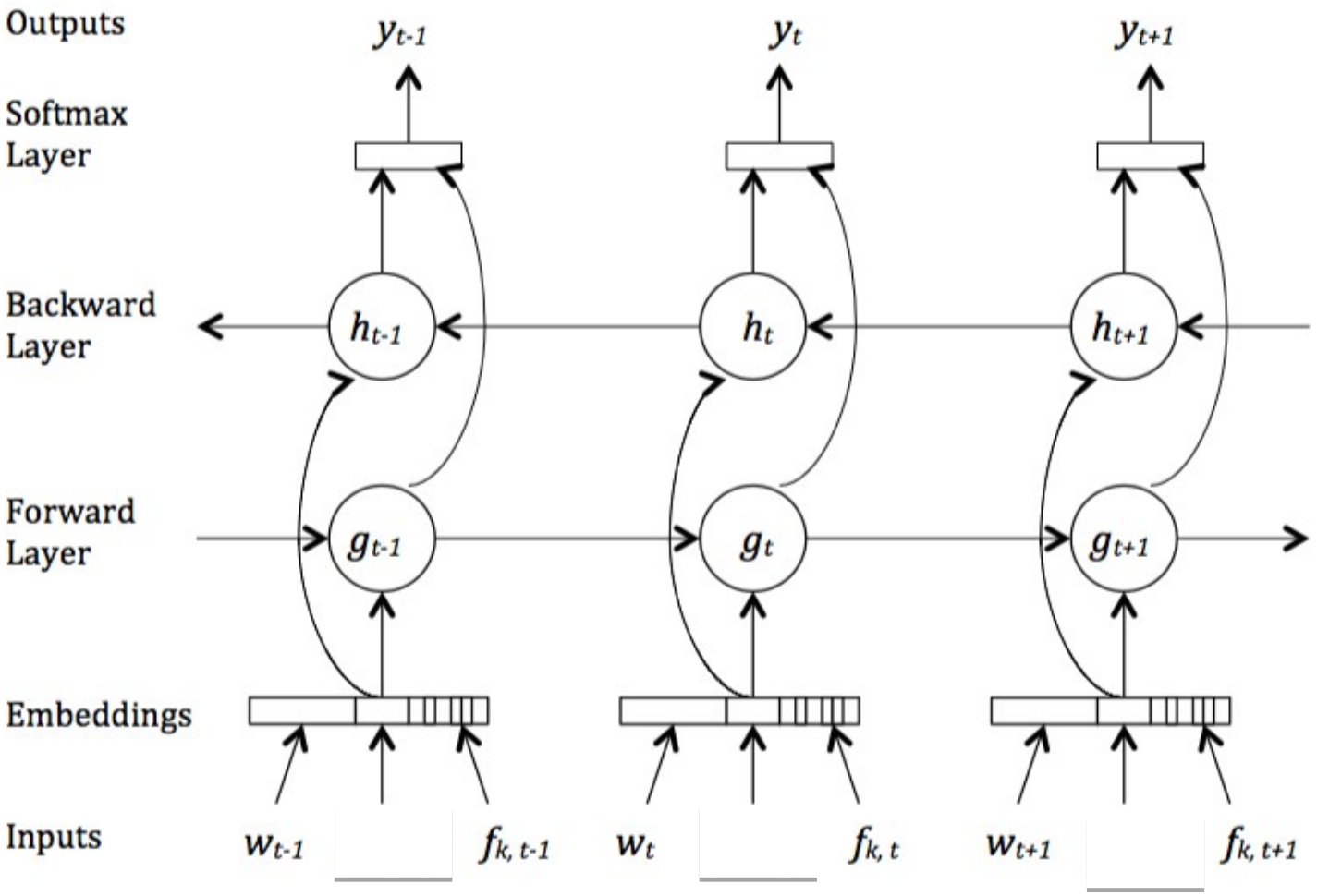
Commonly used approaches in the past:

- Hidden Markov Models (HMM)
- Maximum Entropy Markov Model (MEMM)
- Conditional Random Fields (CRF)

Currently used approaches:

- Bidirectional LSTMs, incl. CRF layer
- Transformer-based models (BERT, ...)

Bi-LSTM for sequence tagging

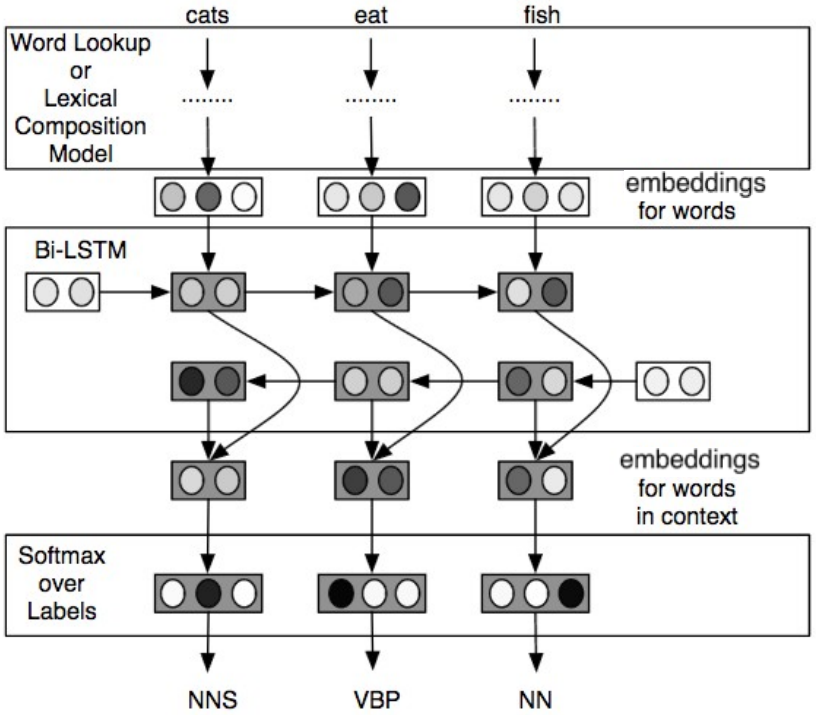


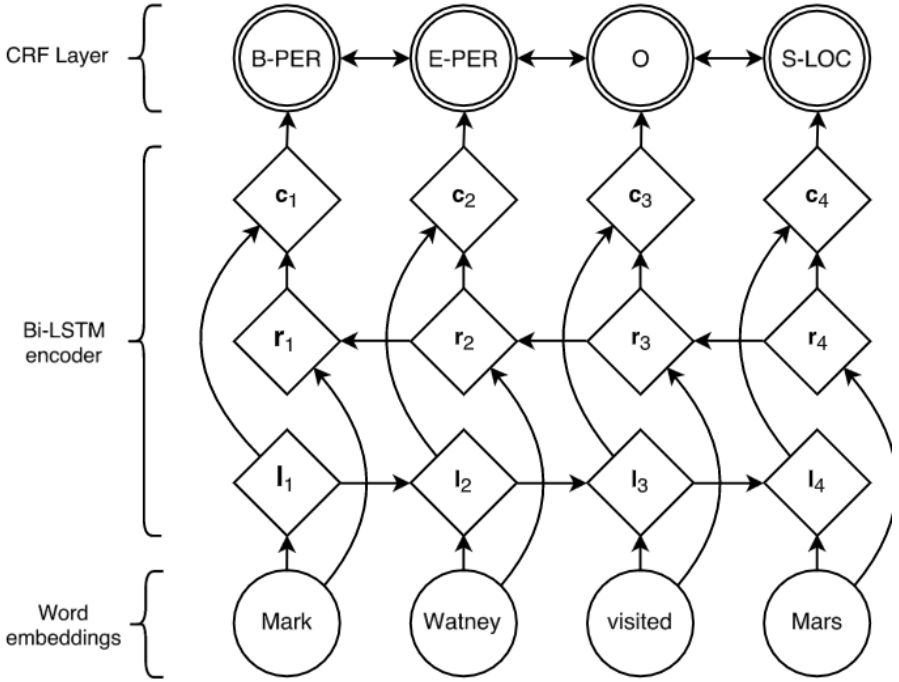
- **Input**: Word embeddings, additional word features
- Combine two directions: usually concatenation
- **Output**: 1-hot-encoding over labels (softmax)

- State size: there are many 'parallel' LSTM cells in each layer
- LSTM layers can be stacked for deeper networks

Source: Fig. from: Zayats, V., Ostendorf, M., Hajishirzi, H. (2016): Disfluency Detection using a Bidirectional LSTM. Proceedings of Interspeech 2016

Bi-LSTM for POS Tagging - Variants

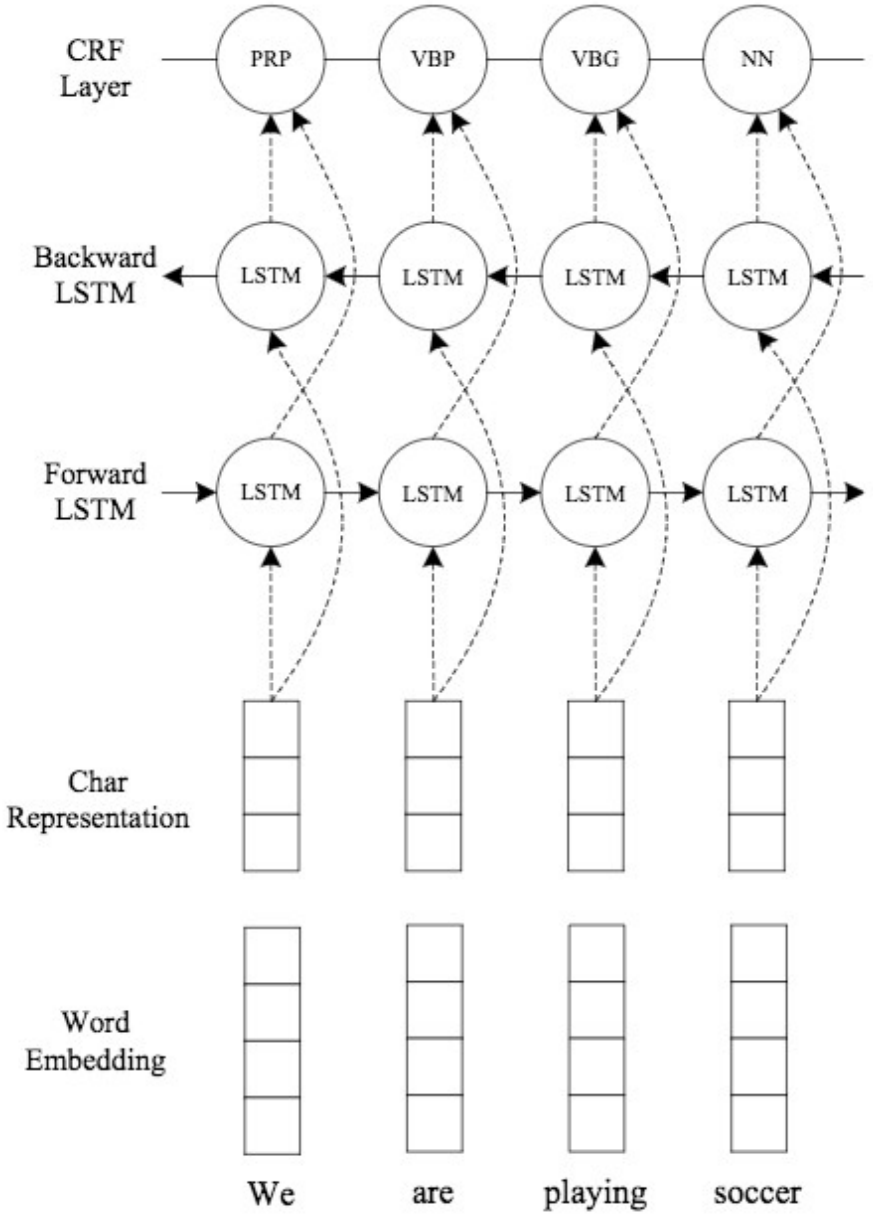




- Compose words from character embeddings to addess unseen words
- Use combined outputs as features in CRF layer, better making use of neighboring labels

- Ling, W., Dyer, C., Black, A.W., Trancoso, I., Fermandez, R., Amir, S., Marujo, L. and Luis, T. (2015): Finding Function in Form: Compositional Character Models for Open Vocabulary Word Representation, Proceedings of EMNLP
- Lample, G., Ballesteros, M., Subramanian, S., Kawakami, K. and Dyer, C. (2016): Neural Architectures for Named Entity Recognition. Proceedings of NAACL.

2016 state-of-the-art in POS tagging and NER



=

Layer	Hyper-parameter	POS	NER
CNN	window size	3	3
	number of filters	30	30
LSTM	state size	200	200
	initial state	0.0	0.0
	peepholes	no	no
Dropout	dropout rate	0.5	0.5
	batch size	10	10
	initial learning rate	0.01	0.015
	decay rate	0.05	0.05
	gradient clipping	5.0	5.0

Table 1: Hyper-parameters for all experiments.

One of the first papers that has state-of-the-art performance with end-to-end approach on standard text processing:

Ma, X. and Hovy, E. (2016): End-to-end Sequence Labeling via Bi-directional LSTM-CNNs-CRF. Proceedings of ACL 2016, pp. 1064-1074, Berlin, Germany

Plan of the lecture

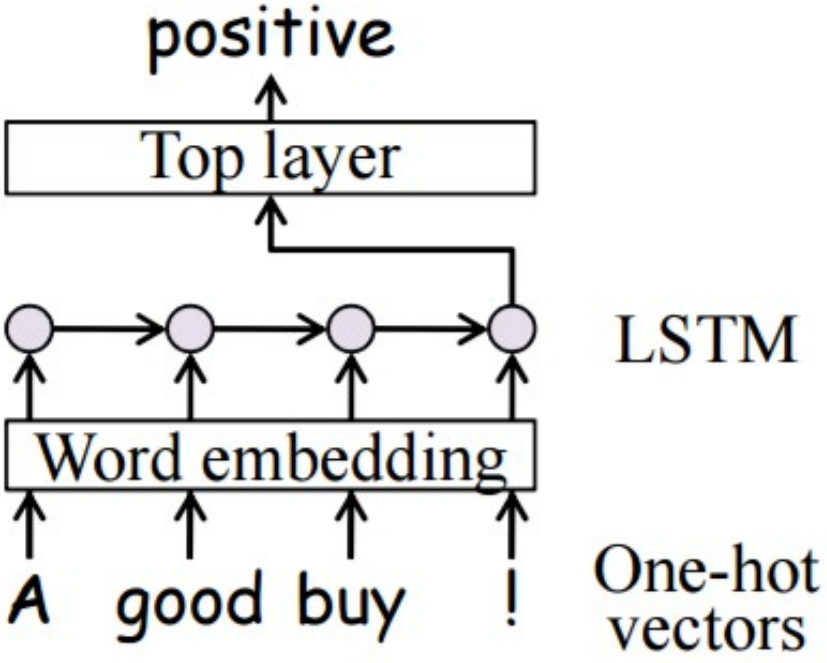
- **Part 1**: Language modeling.
- **Part 2**: Recurrent neural networks.
- **Part 3**: Long-Short Term Memory (LSTM).
- **Part 4**: LSTMs for sequence labelling.
- **Part 5**: LSTMs for text categorization.

Sentiment Analysis: Error Rates of various IMDB Models

- Binary Multinomial Naive Bayes:
  - **15.7** on 1-grams
  - **11.6** on 2-3grams ← Assignment 1
- Logistic Regression:
  - **11.5** on 1-grams
  - **9.3** on 1-3grams ← Assignment 2
- NB scaler + linear classifier
  - **8.8** [Wang and Manning. Baselines and Bigrams: Simple, Good Sentiment and Topic Classification, 2012]
  - **8.1** [Mesnil et al. Ensemble of Generative and Discriminative Techniques for Sentiment Analysis of Movie Reviews, 2015]
- FFNN on Glove average
  - **10.6** [Iyyer et al. Deep Unordered Composition Rivals Syntactic Methods for Text Classification, 2015] – worse then Logistic Regression?! ← Assignment 2
- Best models all use **LSTMs with unsupervised pretrain** (and several other tricks)
  - **7.3 [Dai and Le. Semi-supervised Sequence Learning, 2015]**
  - **7.1 [Radford et al. Learning to Generate Reviews and Discovering Sentiment, 2017]**
  - 6.3 [Dieng et al. TopicRNN: A Recurrent Neural Network with Long-Range Semantic Dependency, 2016]
  - **5.9 [Miyato et al. Adversarial Training Methods for Semi-supervised Text Classification, 2017]**
  - 5.9 [Johnson and Zhang. Supervised and Semi-Supervised Text Categorization using LSTM for Region Embeddings, 2016]
  - 4.6 [Howard and Ruder. Universal Language Model Fine-tuning for Text Classification, 2018]

LSTM classifier: a naive approach

- The hidden state (=output) at last time step can represent the whole input sequence
  - seq2seq
- Add FFNN classifier on top
- Dai and Le. (2015) in “Semi-supervised Sequence Learning” tried and it didn’t work that well:
  - **13.5%** error rate (worse then NB)
  - Very unstable training
  - Too little information about outputs?
    - Only 1 bit for each (long) review
    - Complex model like LSTM can correlate it with lots of different input patterns
  - Vanishing gradient is still a problem for (long) reviews?

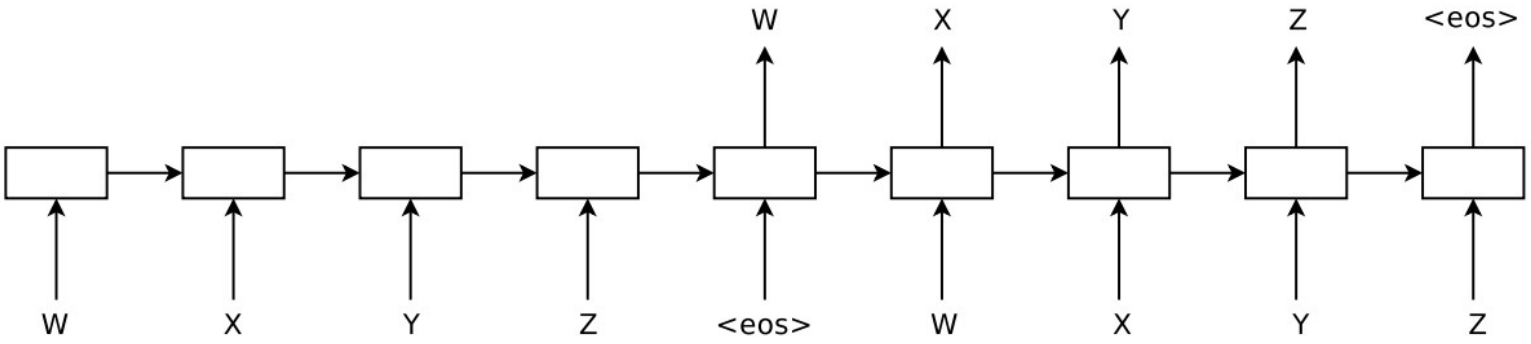


Source: Johnson and Zhang (2016): Supervised and Semi-Supervised Text Categorization using LSTM for Region Embeddings.

LSTM Unsupervised Pretraining

- Give/require more information about outputs
  - **Pretraining**: train another model on some (distantly) related task, for which we have / can generate a large train set (preferably)
    - Language Model [unsupervised!]
    - Sequence Autoencoder [unsupervised!]

    => sensible initial weights
  - Fine-tuning: train classifier on the target task, initializing embeddings and LSTM weights non-randomly, FFNN weights – randomly
    - can fine-tune or fix non-randomly initialized weights



Source: Dai and Le. (2015): Semi-supervised Sequence Learning

IMDB Results

- Embeddings initialized with word2vec are better than random
- Embeddings and LSTM weights initialized with LM/SA weights is much better!
- Paragraph Vectors results is invalid,
  - Train and test sets were not shuffled: It is **11.3**%

[Mesnil et al. Ensemble of Generative and Discriminative Techniques for Sentiment Analysis of Movie Reviews. 2015]

This was improved to 7.33% with properly selected hyperparameters

Table 2: Performance of models on the IMDB sentiment classification task.

Model	Test error rate
LSTM with tuning and dropout	13.50%
LSTM initialized with word2vec embeddings	10.00%
LM-LSTM (see Section 2)	7.64%
SA-LSTM (see Figure 1)	7.24%
SA-LSTM with linear gain (see Section 3)	9.17%
SA-LSTM with joint training (see Section 3)	14.70%
Full+Unlabeled+BoW [22]	11.11%
WRRBM + BoW (bnc) [22]	10.77%
NBSVM-bi (Naïve Bayes SVM with bigrams) [36]	8.78%
seq2-bow*n*-CNN (ConvNet with dynamic pooling) [12]	7.67%
Paragraph Vectors [19]	7.42%

Source: Dai and Le. (2015): Semi-supervised Sequence Learning.

Results

Model	Test error rate
LSTM with tuning and dropout	20.3%
LM-LSTM	21.9%
LSTM with linear gain	22.2%
SA-LSTM	19.3%
LSTM with word vectors from word2vec Google News	20.5%
SA-LSTM with unlabeled data from IMDB	18.6%
SA-LSTM with unlabeled data from Amazon reviews	16.7%
MV-RNN [29]	21.0%
NBSVM-bi [36]	20.6%
CNN-rand [13]	23.5%
CNN-non-static (ConvNet with word vectors from word2vec Google News) [13]	18.5%

Larger unlabeled corpus for pretraining helps!
IMDB: 50K
Amazon reviews: 8M

sometimes LM is better than SA pretrain

Model	Test error rate
LSTM	18.0%
LM-LSTM	15.3%
LSTM with linear gain	71.6%
SA-LSTM	15.6%
Hybrid Class RBM [18]	23.8%
RBM-MLP [5]	20.5%
SVM + Bag-of-words [3]	17.1%
Naïve Bayes [3]	19.0%

Character-level topic categorization:
=> long sequences, awful results without pretrain!
Stacking LSTMs help

Model	Test error rate
LSTM	13.64%
LM-LSTM	1.50%
LSTM with linear gain	1.32%
SA-LSTM	2.34%
SA-LSTM with linear gain	1.23%
SA-LSTM with 3 layers and linear gain	1.19%
SA-LSTM (word-level)	1.40%
Bag-of-words	3.57%
Small ConvNet	1.98%
Large ConvNet	1.73%

Source: Dai and Le (2015): Semi-supervised Sequence Learning.

Unsupervised Sentiment Neuron

- After training byte-level mLSTM for LM they found “Sentiment neuron”
  - Amazon Product Reviews: 82M reviews over 18 years, 38GB of unlabeled texts
  - 1 month of training of 4GPUs, 1 epoch / 1M steps
  - Adam, initial lr: 5e-4 decayed linearly to 0, batch: 128 subsequences of length 256 bytes
- **7.70**% - logistic regression on single neuron (it is simply 1 scalar threshold)!
  - 7.12% on all 4096 units

Table 2. IMDB sentiment classification

METHOD	ERROR
FULLUNLABELEDBOW (MAAS ET AL., 2011)	11.11%
NB-SVM TRIGRAM (MESNIL ET AL., 2014)	8.13%
SENTIMENT UNIT (OURS)	7.70%
SA-LSTM (DAI & LE, 2015)	7.24%
BYTE MLSTM (OURS)	7.12%
TOPICRNN (DIENG ET AL., 2016)	6.24%
VIRTUAL ADV (MIYATO ET AL., 2016)	5.91%

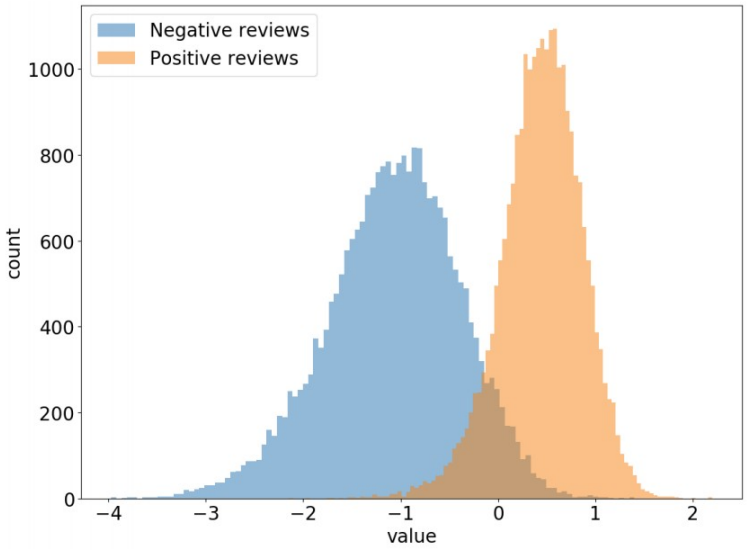
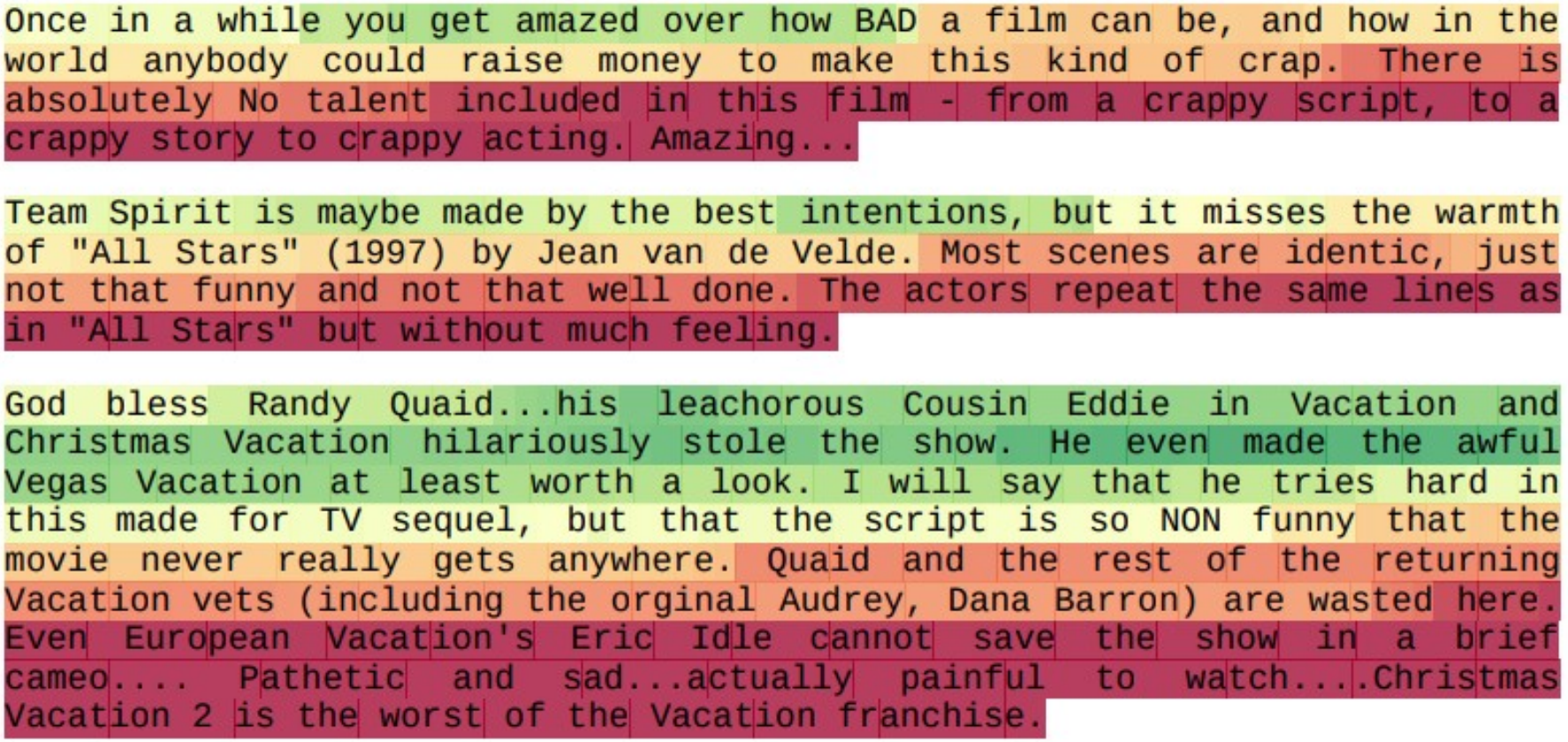


Figure 3. Histogram of cell activation values for the sentiment unit on IMDB reviews.

- Source: Radford et al. (2017): Learning to Generate Reviews and Discovering Sentiment

Sentiment Neuron Visualizations

- How sentiment neuron changes while reading text?



Source: Radford et al. Learning to Generate Reviews and Discovering Sentiment (2017)

Conditional text generation

- LM can be used to generate new reviews
  - fix sentiment neuron to generate desired sentiment

Sentiment fixed to positive	Sentiment fixed to negative
Just what I was looking for. Nice fitted pants, exactly matched seam to color contrast with other pants I own. Highly recommended and also very happy!	The package received was blank and has no barcode. A waste of time and money.
This product does what it is supposed to. I always keep three of these in my kitchen just in case ever I need a replacement cord.	Great little item. Hard to put on the crib without some kind of embellishment. My guess is just like the screw kind of attachment I had.
Best hammock ever! Stays in place and holds it's shape. Comfy (I love the deep neon pictures on it), and looks so cute.	They didn't fit either. Straight high sticks at the end. On par with other buds I have. Lesson learned to avoid.
Dixie is getting her Doolittle newsletter we'll see another new one coming out next year. Great stuff. And, here's the contents - information that we hardly know about or forget.	great product but no seller. couldn't ascertain a cause. Broken product. I am a prolific consumer of this company all the time.
I love this weapons look . Like I said beautiful !!! I recommend it to all. Would suggest this to many roleplayers , And I stronge to get them for every one I know. A must watch for any man who love Chess!	Like the cover, Fits good. . However, an annoying rear piece like garbage should be out of this one. I bought this hoping it would help with a huge pull down my back & the black just doesn't stay. Scrap off everytime I use it.... Very disappointed.

Table 5. Random samples from the model generated when the value of sentiment hidden state is fixed to either -1 or 1 for all steps. The sentiment unit has a strong influence on the model's generative process.

Source: Radford et al. Learning to Generate Reviews and Discovering Sentiment (2017)

Adversarial training

- Want predict the same class for nearby points
  - add (to the loss) a penalty for low predicted probability of the **correct class** in the small neighborhood of a labeled example

$$-\log p(y \mid \boldsymbol{x} + \boldsymbol{r}_{\text{adv}}; \boldsymbol{\theta}) \text{ where } \boldsymbol{r}_{\text{adv}} = \arg\min_{\boldsymbol{r}, \|\boldsymbol{r}\| \leq \epsilon} \log p(y \mid \boldsymbol{x} + \boldsymbol{r}; \hat{\boldsymbol{\theta}})$$

- r_{adv} can be small random perturbation, but the worst possible perturbation given epsilon works much better!
- Need to calculate r_{adv} at each timestep (effectively)!
  - use gradient ascent (linear approximation)

$$\boldsymbol{r}_{\text{adv}} = -\epsilon \boldsymbol{g} / \|\boldsymbol{g}\|_2 \text{ where } \boldsymbol{g} = \nabla_{\boldsymbol{s}} \log p(y \mid \boldsymbol{s}; \hat{\boldsymbol{\theta}})$$

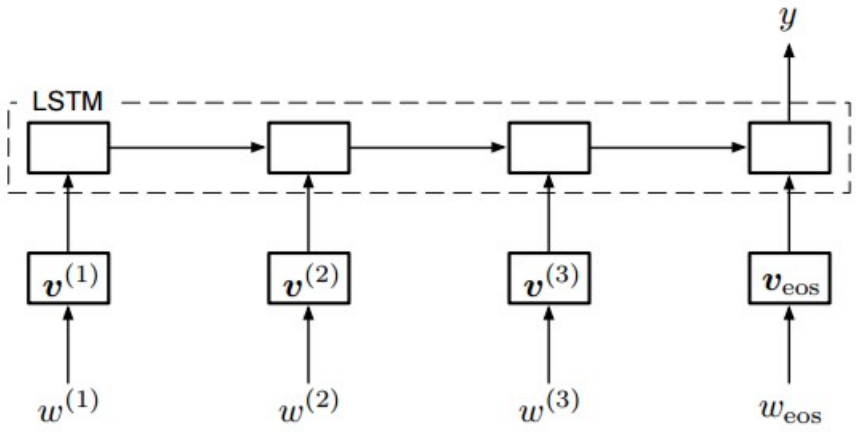
$$L_{\text{adv}}(\boldsymbol{\theta}) = -\frac{1}{N} \sum_{n=1}^{N} \log p(y_n \mid \boldsymbol{s}_n + \boldsymbol{r}_{\text{adv},n}; \boldsymbol{\theta})$$

Source: Miyato, Adversarial Training Methods for Semi-Supervised Text Classification, 2017

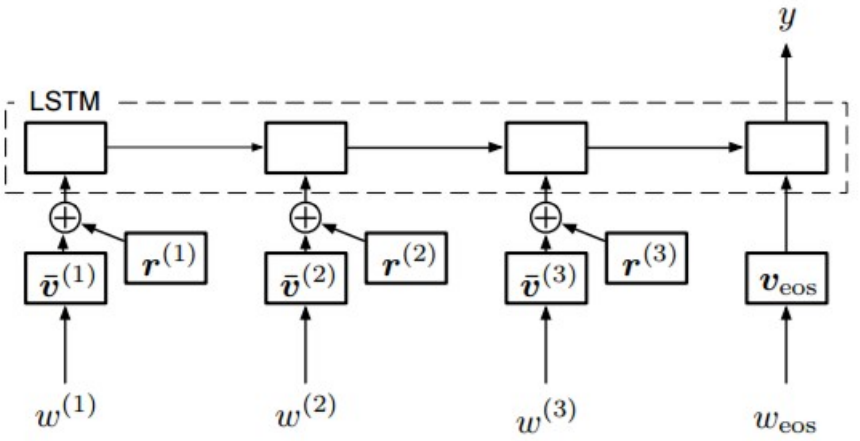
Adversarial training for LSTM

- For texts: add adversarial perturbation to (standardized) embeddings

$$\bar{\boldsymbol{v}}_k = \frac{\boldsymbol{v}_k - \mathrm{E}(\boldsymbol{v})}{\sqrt{\mathrm{Var}(\boldsymbol{v})}} \text{ where } \mathrm{E}(\boldsymbol{v}) = \sum_{j=1}^{K} f_j \boldsymbol{v}_j, \mathrm{Var}(\boldsymbol{v}) = \sum_{j=1}^{K} f_j \left(\boldsymbol{v}_j - \mathrm{E}(\boldsymbol{v})\right)^2$$



(a) LSTM-based text classification model.



(b) The model with perturbed embeddings.

Source: Miyato, Adversarial Training Methods for Semi-Supervised Text Classification, 2017

Results

- SOTA on IMDB (and several other datasets)

Table 2: Test performance on the IMDB sentiment classification task. * indicates using pretrained embeddings of CNN and bidirectional LSTM.

Method	Test error rate
Baseline (without embedding normalization)	7.33%
Baseline	7.39%
Random perturbation with labeled examples	7.20%
Random perturbation with labeled and unlabeled examples	6.78%
Adversarial	6.21%
Virtual Adversarial	**5.91**%
Adversarial + Virtual Adversarial	6.09%
Virtual Adversarial (on bidirectional LSTM)	**5.91**%
Adversarial + Virtual Adversarial (on bidirectional LSTM)	6.02%
Full+Unlabeled+BoW (Maas et al., 2011)	11.11%
Transductive SVM (Johnson & Zhang, 2015b)	9.99%
NBSVM-bigrams (Wang & Manning, 2012)	8.78%
Paragraph Vectors (Le & Mikolov, 2014)	7.42%
SA-LSTM (Dai & Le, 2015)	7.24%
One-hot bi-LSTM* (Johnson & Zhang, 2016b)	5.94%

Source: Miyato, Adversarial Training Methods for Semi-Supervised Text Classification, 2017