Statistical Machine Translation LING-462/COSC-482 Week 8: Neural language models

Achim Ruopp
achim.ruopp@Georgetown.edu

Agenda

- Language in ten minutes: Mackenzie Gong - Vietnamese
- N-gram language model recap
- Neural language models

- Break -

- Text representation, word embedding and sentiment analysis in Keras

Language Model p(f)

- Given the target language corpus

  <s> la maison </s>

  <s> la lune </s>

  <s> une maison </s>

Unigrams maximum likelihood probabilities

w	p(w)
<s>	0.25
la	0.167
maison	0.167
lune	0.084
une	0.084
</s>	0.25

Bigram counts

w1/w2	<s>	la	maison	lune	une	</s>	Total
<s>		2			1		3
la			1	1			2
maison						2	2
lune						1	1
une			1				1
</s>							0

Bigram maximum likelihood probabilities

p(w2|w1)	<s>	la	maison	lune	une	</s>	Total
<s>		0.667			0.333		**1.000**
la			0.500	0.500			**1.000**
maison						1.000	**1.000**
lune						1.000	**1.000**
une			1.000				**1.000**
</s>							**0.000**

Bigram maximum likelihood probabilities

- What happens if translation model suggests “une lune” to translate “a moon”?
- Bigram model estimates probability zero!
- Language model smoothing/ interpolation/backoff needed

Bigram add-one counts

w1/w2	<s>	la	maison	lune	une	</s>	Total
<s>	1	3	1	1	2	1	9
la	1	1	2	2	1	1	8
maison	1	1	1	1	1	3	8
lune	1	1	1	1	1	2	7
une	1	1	2	1	1	1	7
</s>	1	1	1	1	1	1	6

Bigram add-one probabilities

p(w2|w1)	<s>	la	maison	lune	une	</s>	Total
<s>	0.111	0.333	0.111	0.111	0.222	0.111	**1.000**
la	0.125	0.125	0.250	0.250	0.125	0.125	**1.000**
maison	0.125	0.125	0.125	0.125	0.125	0.375	**1.000**
lune	0.143	0.143	0.143	0.143	0.143	0.286	**1.000**
une	0.143	0.143	0.286	0.143	0.143	0.143	**1.000**
</s>	0.167	0.167	0.167	0.167	0.167	0.167	**1.000**

Measuring Language Model Quality

- Measured on a held out development set
  - Also called “validation set” in deep learning
- A good language model assigns text *W* of length *n* a high probability
- Or: Cross-entropy

$$H(W) = \frac{1}{n}\log p(W_1^n)$$

- Or: Perplexity

$$perplexity(W) = 2^{H(W)}$$

N-Gram Backoff Language Model

- Previously, we approximated

$$p(W) = p(w_1, w_2, ..., w_n)$$

- ... by applying the chain rule

$$p(W) = \sum_i p(w_i|w_1, ..., w_{i-1})$$

- ... and limiting the history (Markov order)

$$p(w_i|w_1, ..., w_{i-1}) \simeq p(w_i|w_{i-4}, w_{i-3}, w_{i-2}, w_{i-1})$$

- Each $p(w_i|w_{i-4}, w_{i-3}, w_{i-2}, w_{i-1})$ may not have enough statistics to estimate

$\rightarrow$ we back off to $p(w_i|w_{i-3}, w_{i-2}, w_{i-1})$, $p(w_i|w_{i-2}, w_{i-1})$, etc., all the way to $p(w_i)$

  – exact details of backing off get complicated — "interpolated Kneser-Ney"

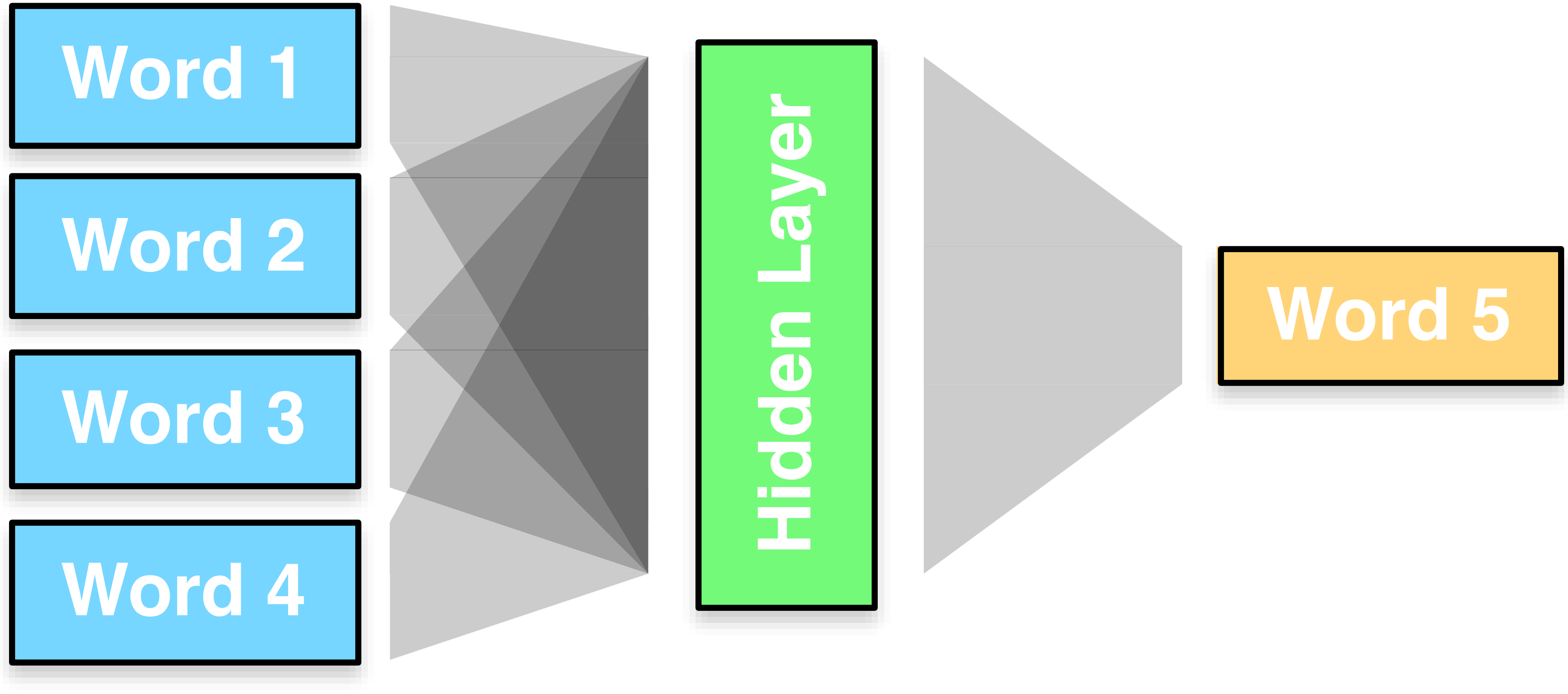
Refinements

- A whole family of back-off schemes
- Skip-n gram models that may back off to $p(w_i|w_{i-2})$
- Class-based models $p(C(w_i)|C(w_{i-4}), C(w_{i-3}), C(w_{i-2}), C(w_{i-1}))$

$\Rightarrow$ We are wrestling here with

- using as much relevant evidence as possible
- pooling evidence between words

First Sketch



Representing Words

- Words are represented with a one-hot vector, e.g.,
  - dog = (0,0,0,0,1,0,0,0,0,....)
  - cat = (0,0,0,0,0,0,0,1,0,....)
  - eat = (0,1,0,0,0,0,0,0,0,....)

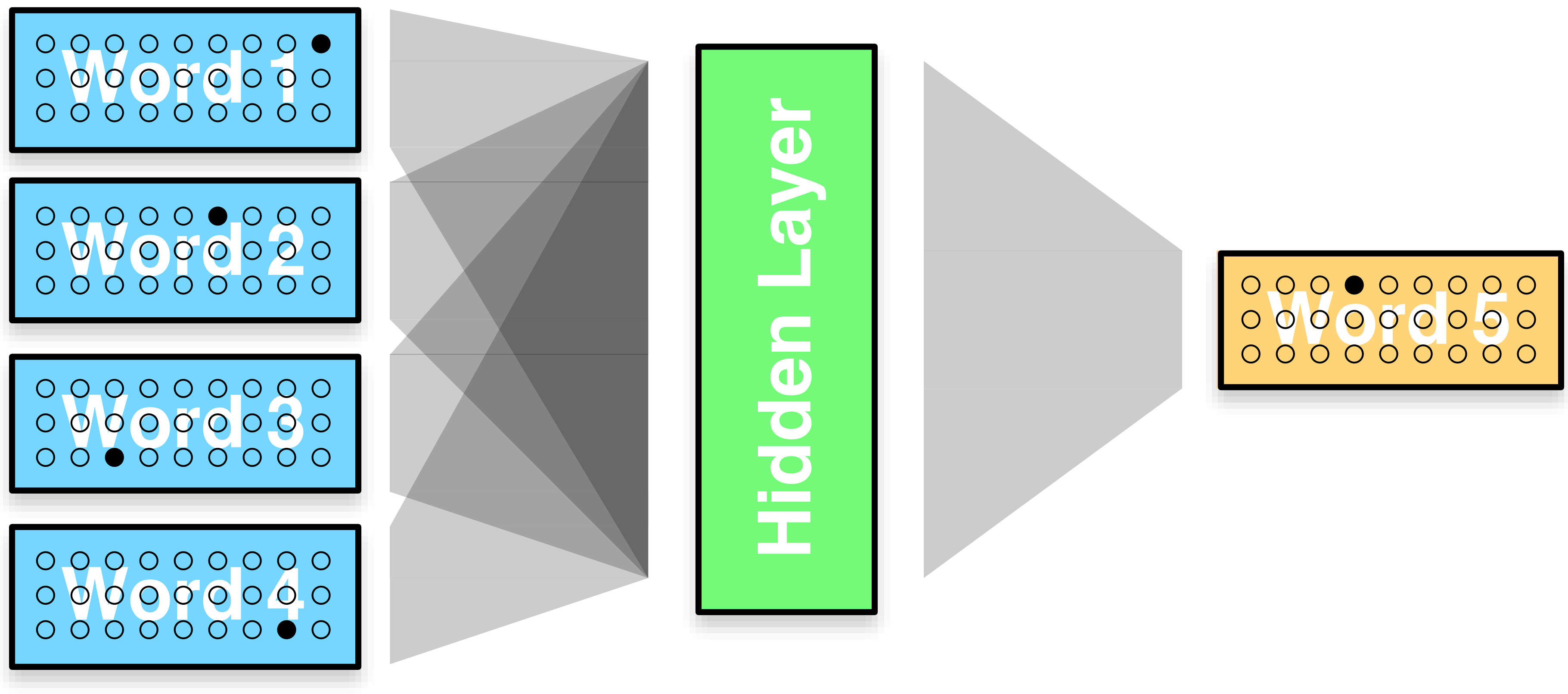
- That's a large vector!

- Remedies
  - limit to, say, 20,000 most frequent words, rest are OTHER
  - place words in $\sqrt{n}$ classes, so each word is represented by
    - 1 class label
    - 1 word in class label

Word Classes for Two-Hot Representations

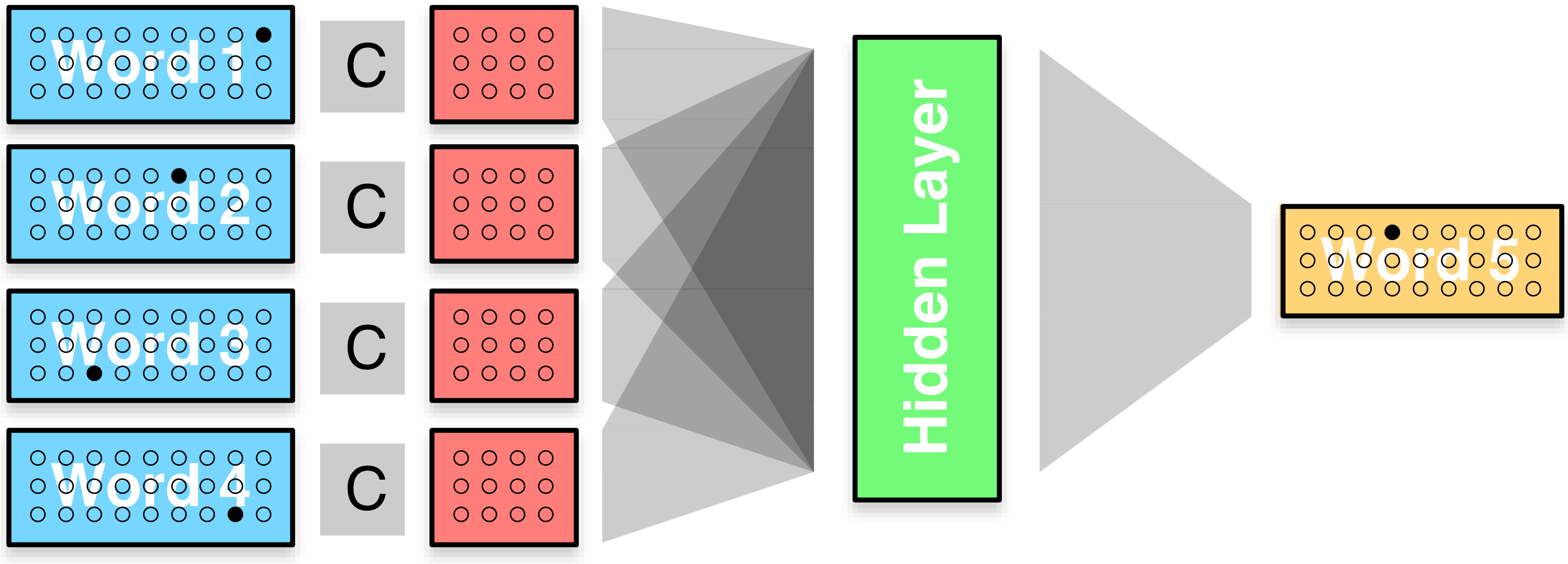
- WordNet classes
- Brown clusters
- Frequency binning
  - sort words by frequency
  - place them in order into classes
  - each class has same token count
  - $\rightarrow$ very frequent words have their own class
  - $\rightarrow$ rare words share class with many other words
- Anything goes: assign words randomly to classes

Second Sketch



word embeddings

Add a Hidden Layer

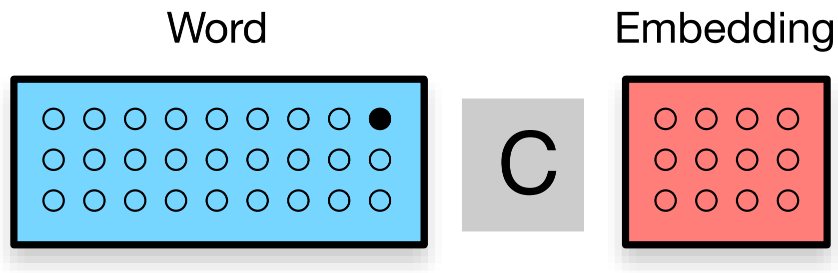


- Map each word first into a lower-dimensional real-valued space
- Shared weight matrix C

Details (Bengio et al., 2003)

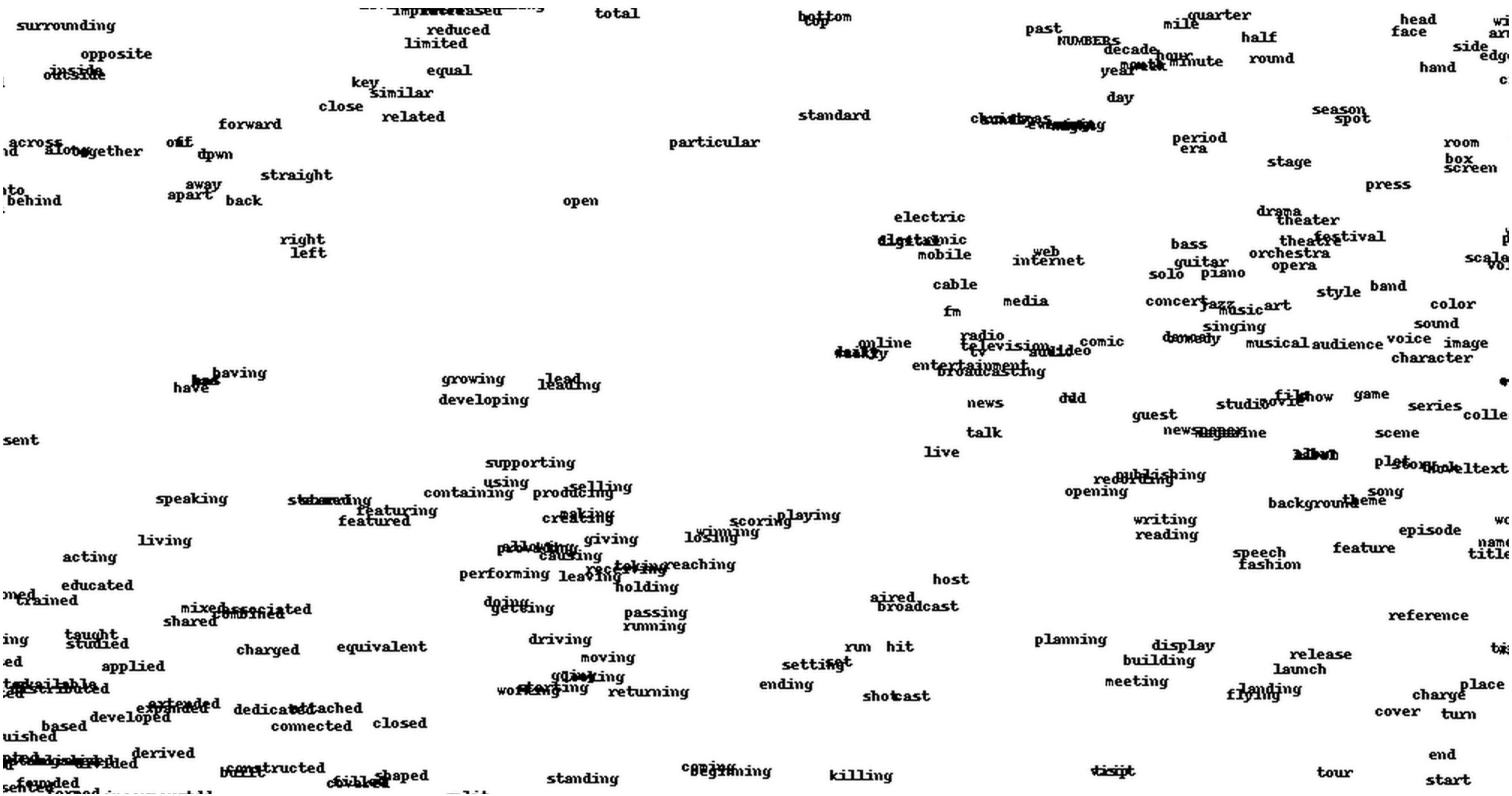
- Add direct connections from embedding layer to output layer
- Activation functions
  - input→embedding: none
  - embedding→hidden: tanh
  - hidden→output: softmax
- Training
  - loop through the entire corpus
  - update between predicted probabilities and 1-hot vector for output word

Word Embeddings

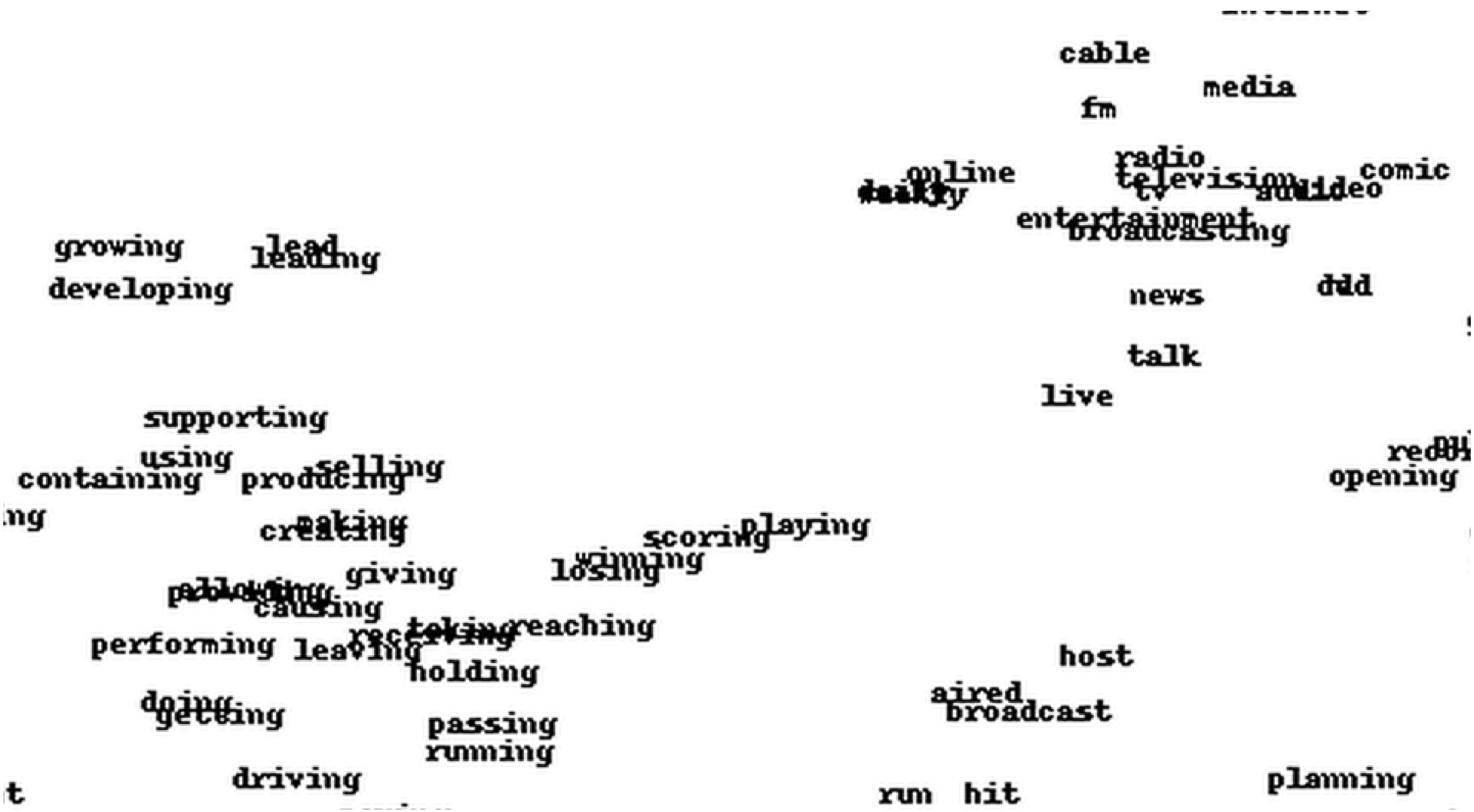


- By-product: embedding of word into continuous space
- Similar contexts $\rightarrow$ similar embedding
- Recall: distributional semantics

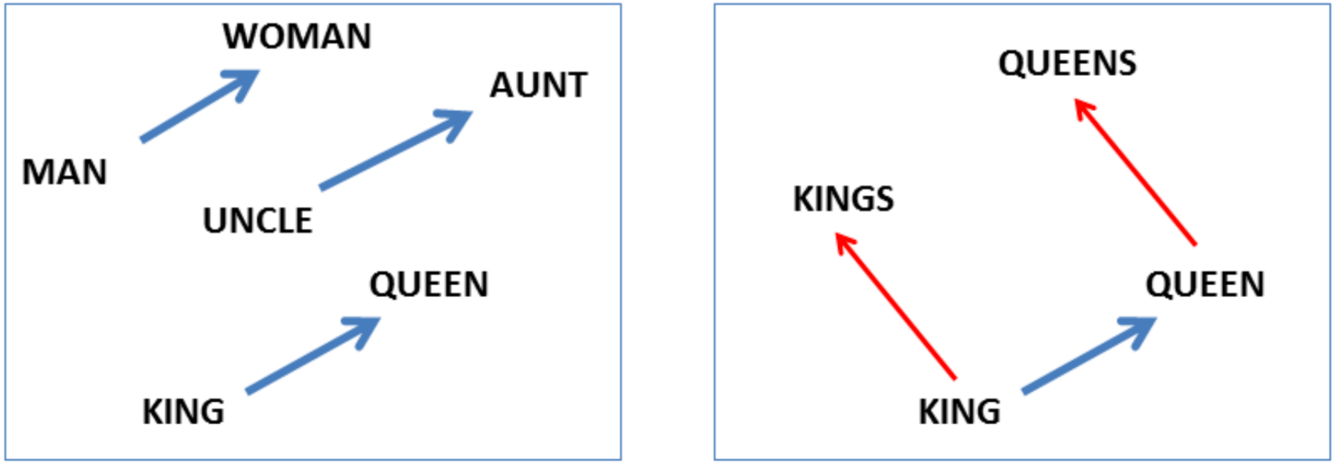
Word Embeddings



Word Embeddings



Are Word Embeddings Magic?

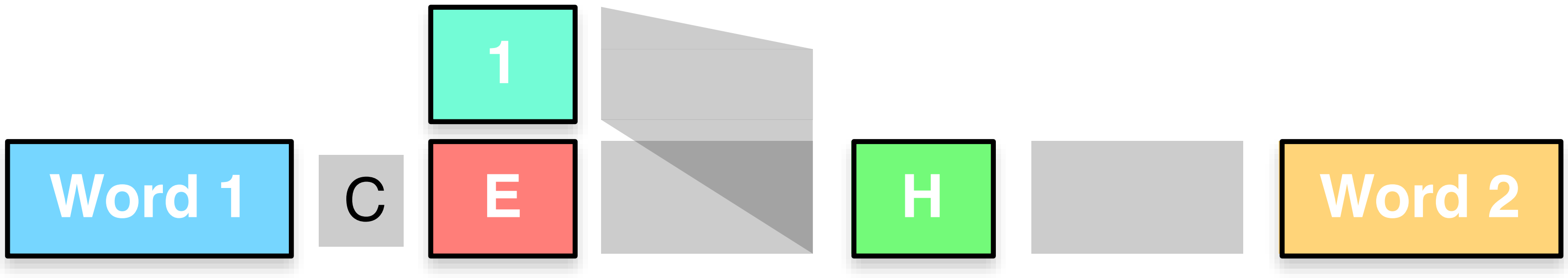


- Morphosyntactic regularities (Mikolov et al., 2013)
  - adjectives base form vs. comparative, e.g., good, better
  - nouns singular vs. plural, e.g., year, years
  - verbs present tense vs. past tense, e.g., see, saw

- Semantic regularities
  - clothing is to shirt as dish is to bowl
  - evaluated on human judgment data of semantic similarities

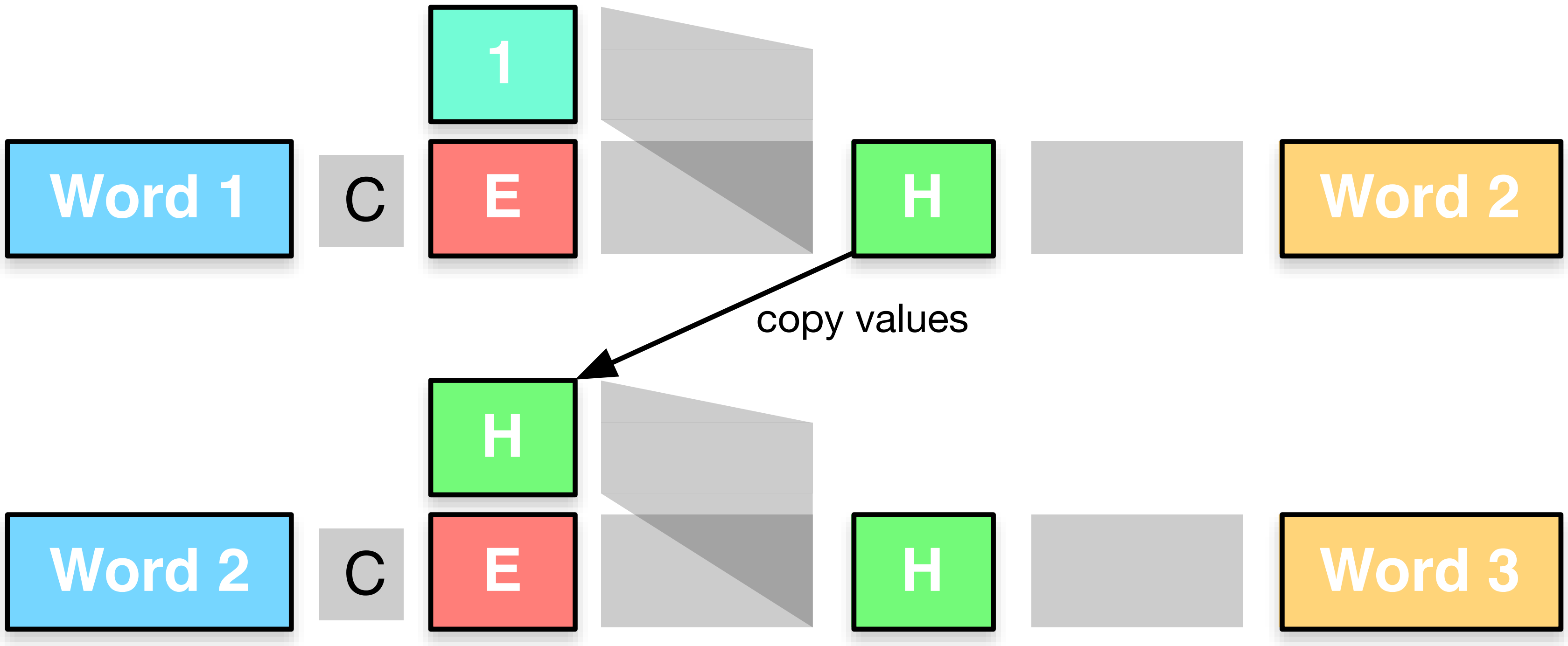
recurrent neural networks

Recurrent Neural Networks

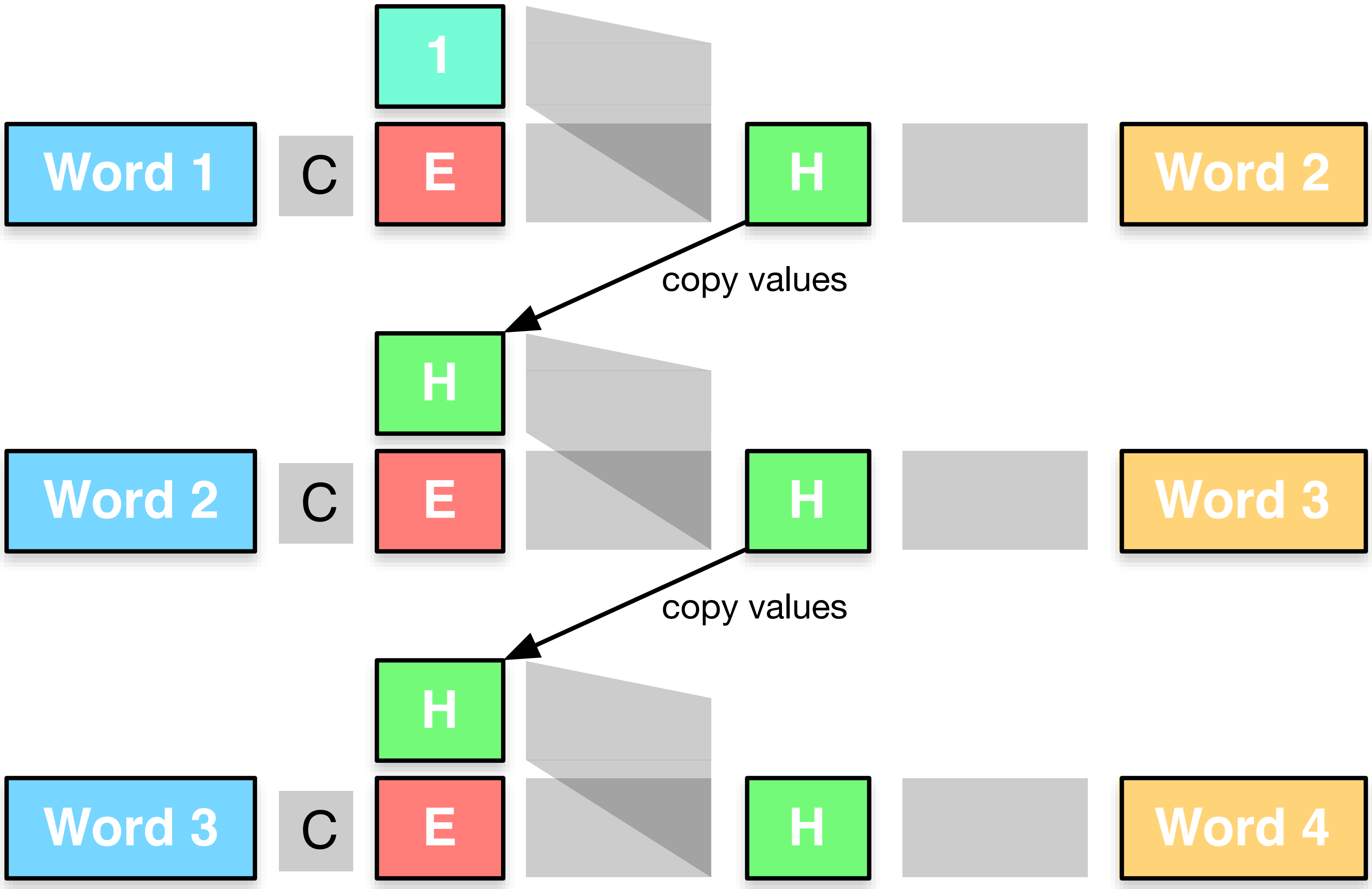


- Start: predict second word from first
- Mystery layer with nodes all with value 1

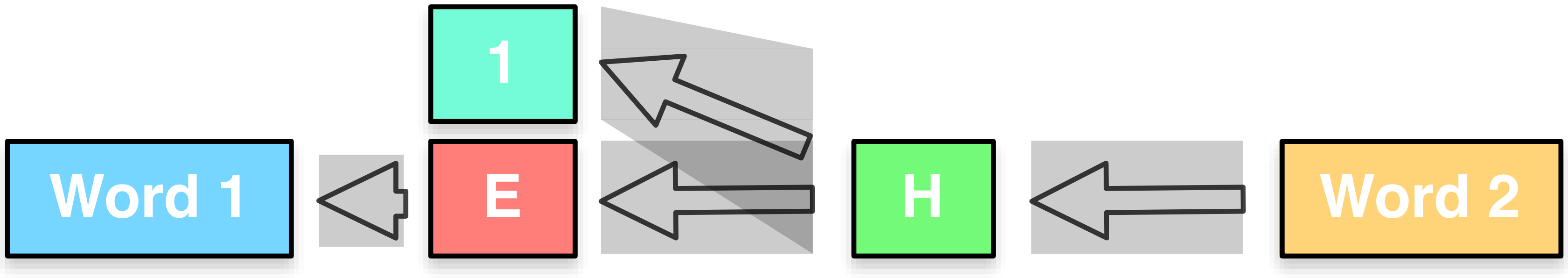
Recurrent Neural Networks



Recurrent Neural Networks

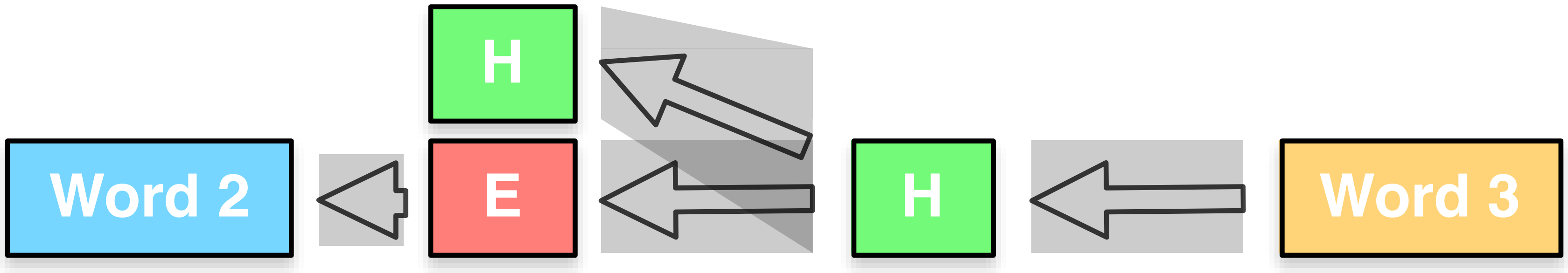


Training



- Process first training example
- Update weights with back-propagation

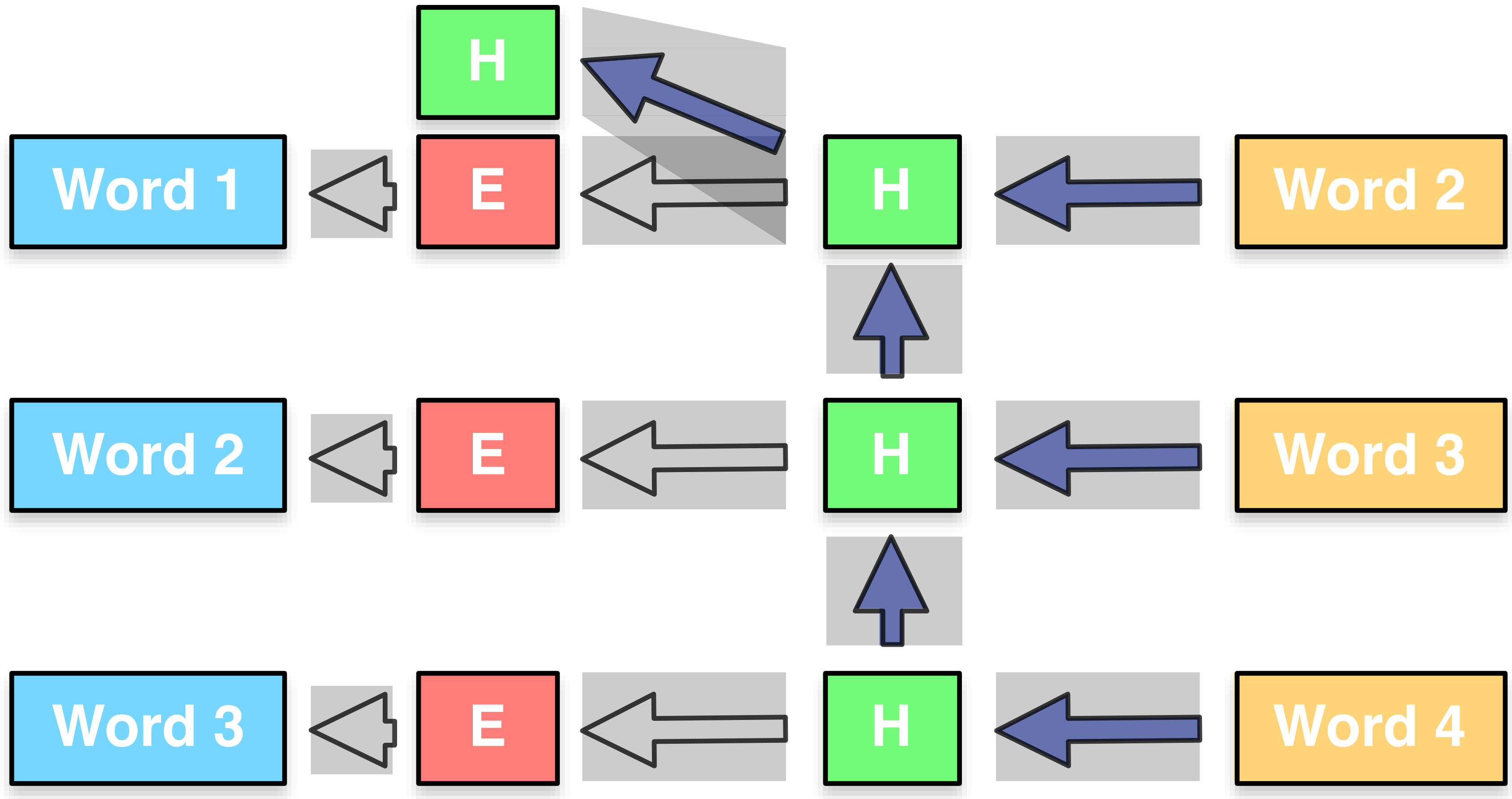
Training



- Process second training example
- Update weights with back-propagation
- And so on...

- But: no feedback to previous history

Back-Propagation Through Time



- After processing a few training examples, update through the unfolded recurrent neural network

Back-Propagation Through Time

- Carry out back-propagation though time (BPTT) after each training example
  - 5 time steps seems to be sufficient
  - network learns to store information for more than 5 time steps

- Or: update in mini-batches
  - process 10-20 training examples
  - update backwards through all examples
  - removes need for multiple steps for each training example

long short term memory

Vanishing Gradients

- Error is propagated to previous steps
- Updates consider
  - prediction at that time step
  - impact on future time steps
- Vanishing gradient: propagated error disappears

Recent vs. Early History

- Hidden layer plays double duty
  - memory of the network
  - continuous space representation used to predict output words

- Sometimes only recent context important

  After much economic progress over the years, the **country** $\rightarrow$ *has*

- Sometimes much earlier context important

The **country** *which has made much economic progress over the years still* $\rightarrow$ *has*

Long Short Term Memory (LSTM)

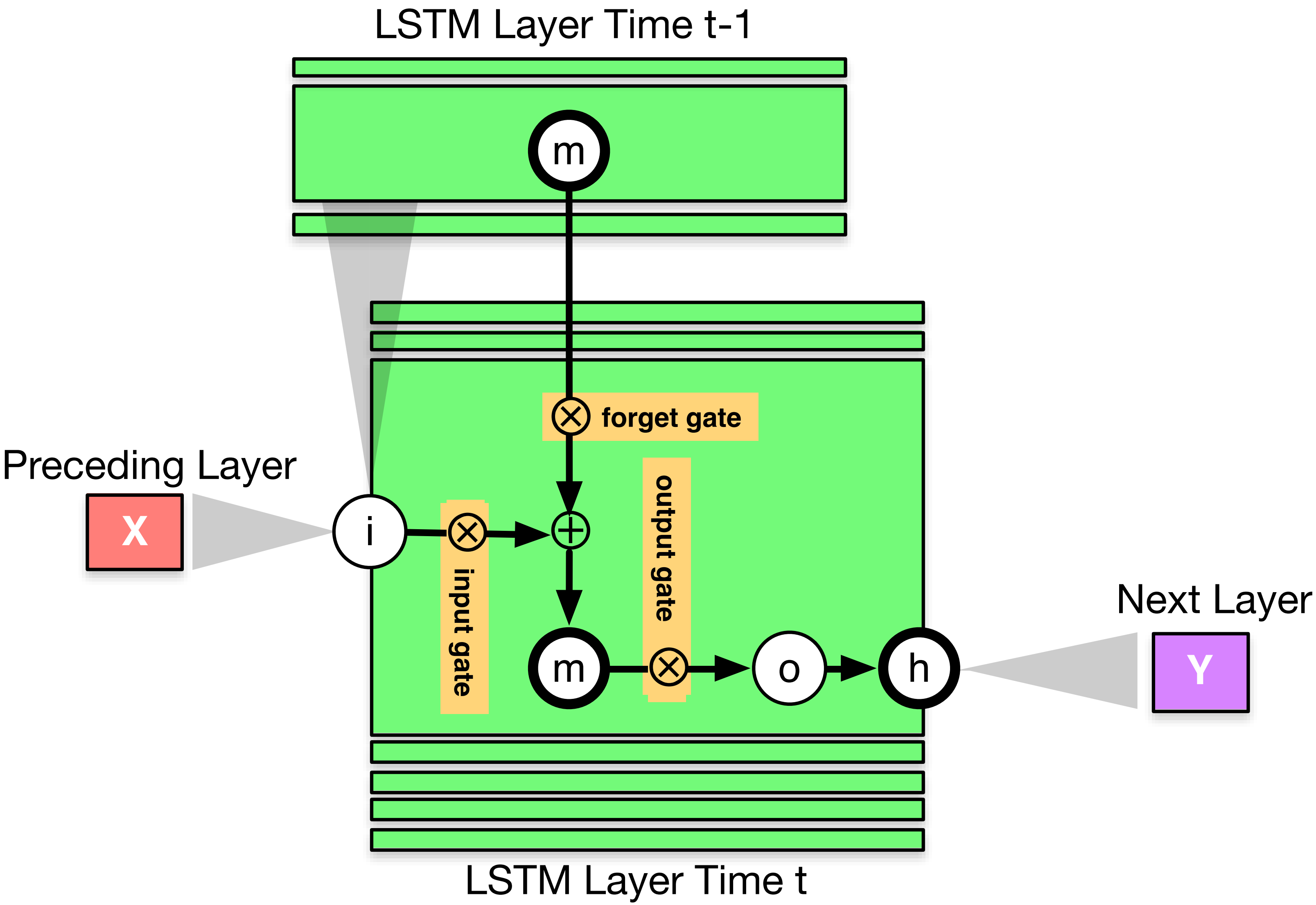
- Design quite elaborate, although not very complicated to use

- Basic building block: **LSTM cell**
  - similar to a node in a hidden layer
  - but: has a explicit memory state

- Output and memory state change depends on gates
  - **input gate**: how much new input changes memory state
  - **forget gate**: how much of prior memory state is retained
  - **output gate**: how strongly memory state is passed on to next layer.

- Gates can be not just be open (1) and closed (0), but slightly ajar (e.g., 0.2)

LSTM Cell


LSTM Layer Time t-1
m
forget gate
Preceding Layer
X
i
input gate
output gate
m
o
h
Next Layer
Y
LSTM Layer Time t

LSTM Cell (Math)

- Memory and output values at time step t

$$\text{memory}^t = \text{gate}_\text{input} \times \text{input}^t + \text{gate}_\text{forget} \times \text{memory}^{t-1}$$
$$\text{output}^t = \text{gate}_\text{output} \times \text{memory}^t$$

- Hidden node value h^t passed on to next layer applies activation function f

$$h^t = f(\text{output}^t)$$

- Input computed as input to recurrent neural network node
  - given node values for prior layer $\vec{x}^t = (x^t_1, ..., x^t_X)$
  - given values for hidden layer from previous time step $\vec{h}^{t-1} = (h^{t-1}_1, ..., h^{t-1}_H)$
  - input value is combination of matrix multiplication with weights w^x and w^h and activation function g

$$\text{input}^t = g\left(\sum_{i=1}^{X} w^x_i x^t_i + \sum_{i=1}^{H} w^h_i h^{t-1}_i\right)$$

Values for Gates

- Gates are very important
- How do we compute their value?

  $\rightarrow$ with a neural network layer!

- For each gate $a \in (\text{input}, \text{forget}, \text{output})$
  - weight matrix W^{xa} to consider node values in previous layer $\vec{x}^t$
  - weight matrix W^{ha} to consider hidden layer $\vec{h}^{t-1}$ at previous time step
  - weight matrix W^{ma} to consider memory at previous time step $\vec{\text{memory}}^{t-1}$
  - activation function h

$$\text{gate}_a = h\left(\sum_{i=1}^{X} w_i^{xa} x_i^t + \sum_{i=1}^{H} w_i^{ha} h_i^{t-1} + \sum_{i=1}^{H} w_i^{ma} \text{memory}_i^{t-1}\right)$$

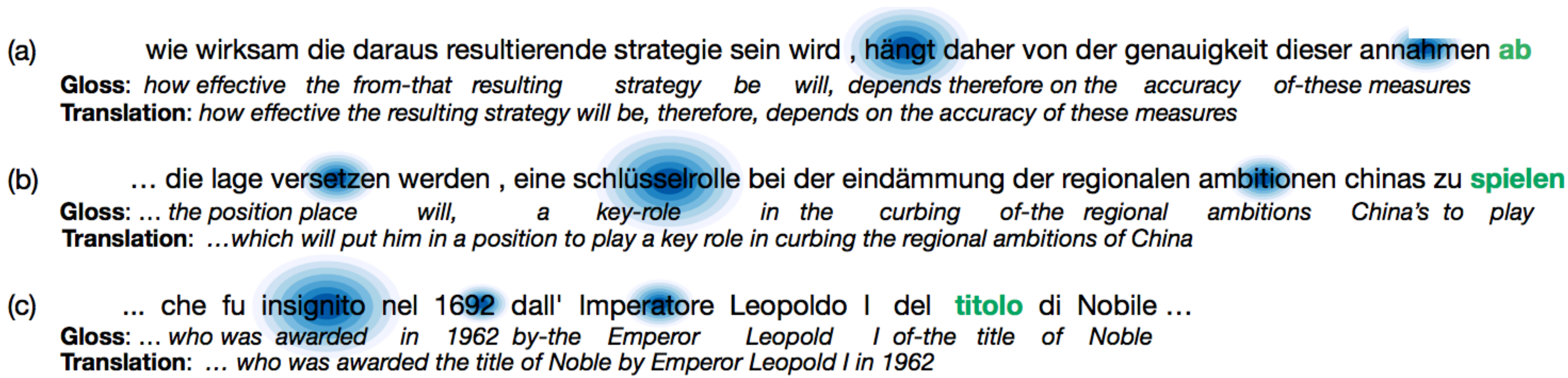
Training

- LSTM are trained the same way as recurrent neural networks
- Back-propagation through time
- This looks all very complex, but:
  - all the operations are still based on
    * matrix multiplications
    * differentiable activation functions

$\rightarrow$ we can compute gradients for objective function with respect to all parameters

$\rightarrow$ we can compute update functions

What is the Point?

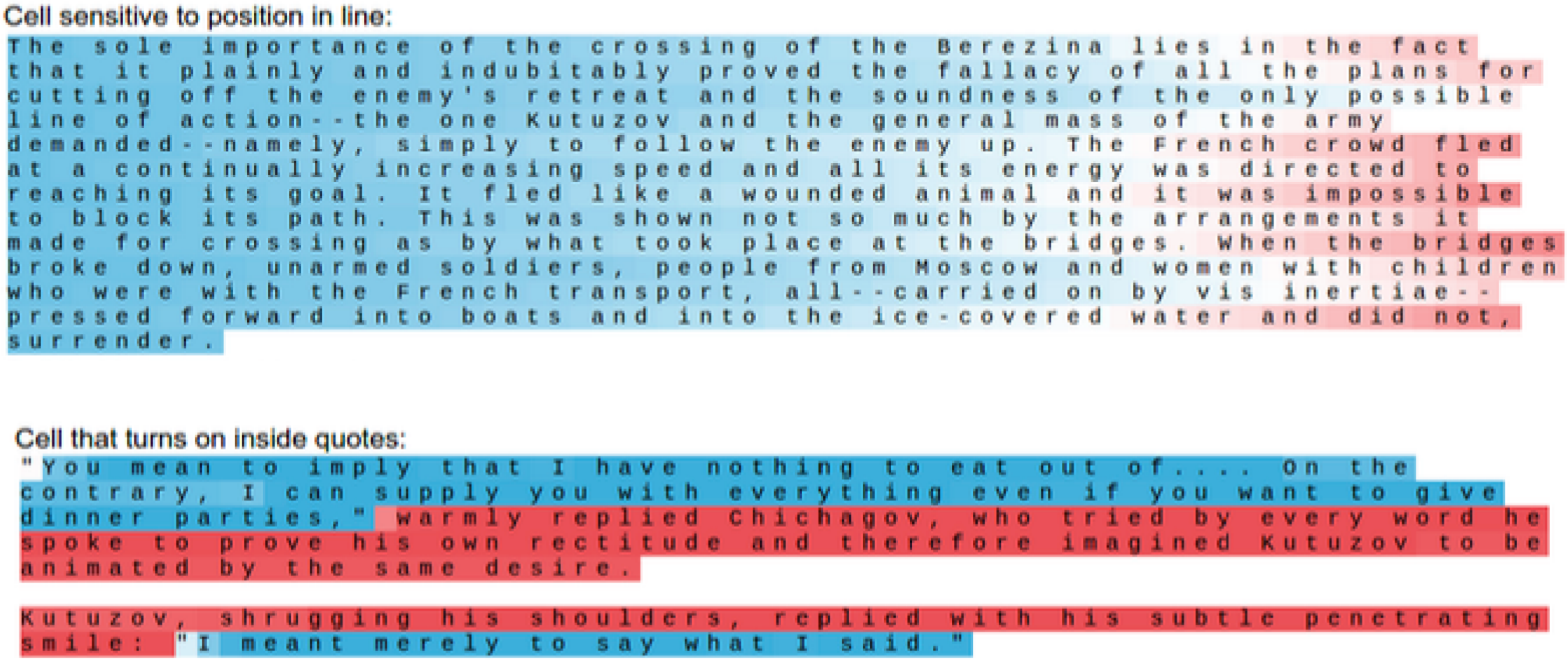


(from Tran, Bisazza, Monz, 2016)

- Each node has memory memory_i independent from current output h_i
- Memory may be carried through unchanged ($\text{gate}^i_{\text{input}} = 0$, $\text{gate}^i_{\text{memory}} = 1$)

⇒ can remember important features over long time span
(capture long distance dependencies)

Visualizing Individual Cells



Karpathy et al. (2015): "Visualizing and Understanding Recurrent Networks"

Visualizing Individual Cells

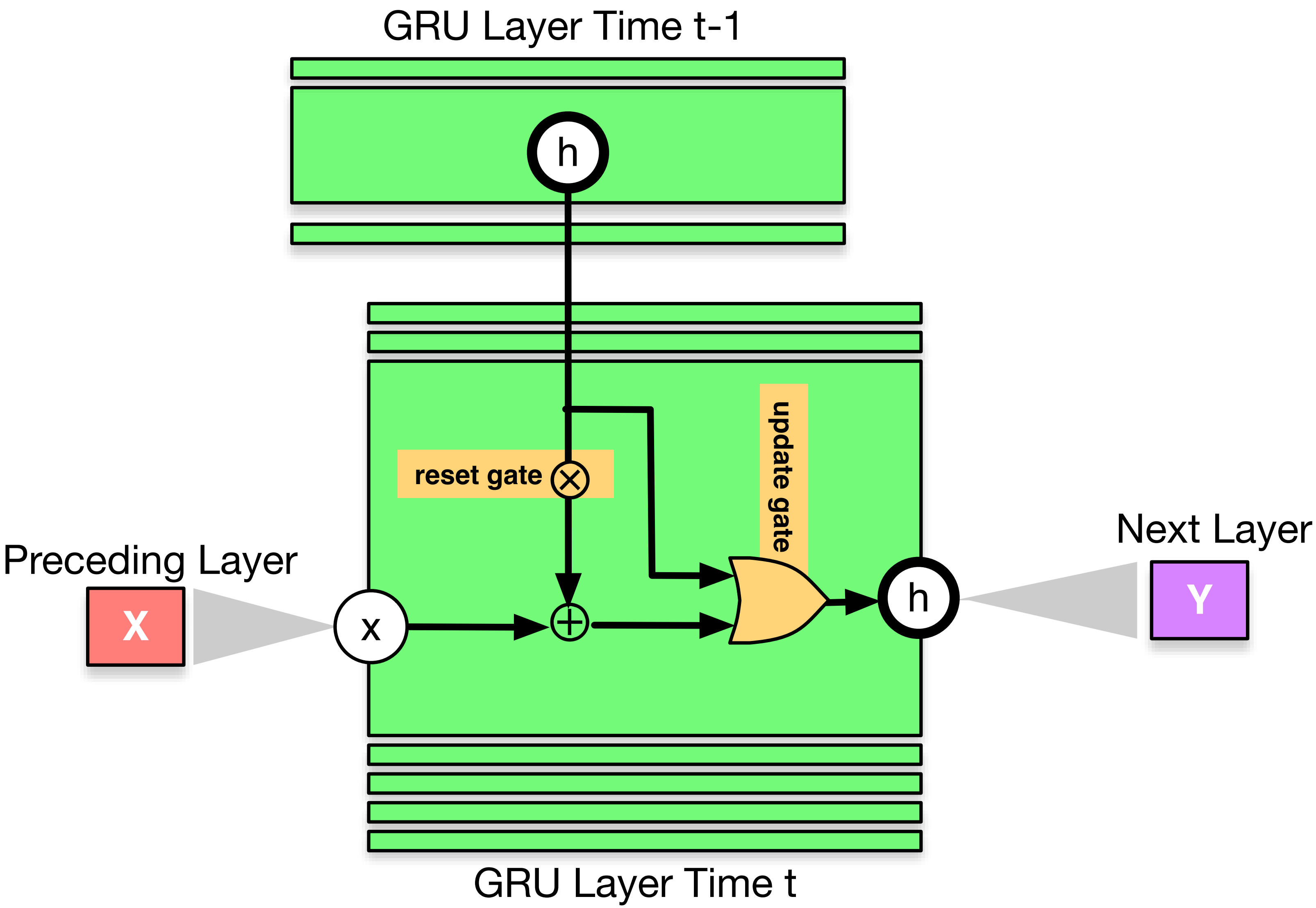
Cell that robustly activates inside if statements:

```
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;
}
```

A large portion of cells are not easily interpretable. Here is a typical example:

```
/* Unpack a filter field's string representation from user-space
 * buffer. */
char *audit_unpack_string(void **bufp, size_t *remain, size_t len)
{
 char *str;
 if (!*bufp || (len == 0) || (len > *remain))
  return ERR_PTR(-EINVAL);
 /* Of the currently implemented string fields, PATH_MAX
  * defines the longest valid length.
  */
```

Gated Recurrent Unit (GRU)



Gated Recurrent Unit (Math)

- Two Gates

$$\text{update}_t = g(W_{\text{update}}\ \text{input}_t + U_{\text{update}}\ \text{state}_{t-1} + \text{bias}_{\text{update}})$$
$$\text{reset}_t = g(W_{\text{reset}}\ \ \text{input}_t + U_{\text{reset}}\ \ \text{state}_{t-1} + \text{bias}_{\text{reset}})$$

- Combination of input and previous state
(similar to traditional recurrent neural network)

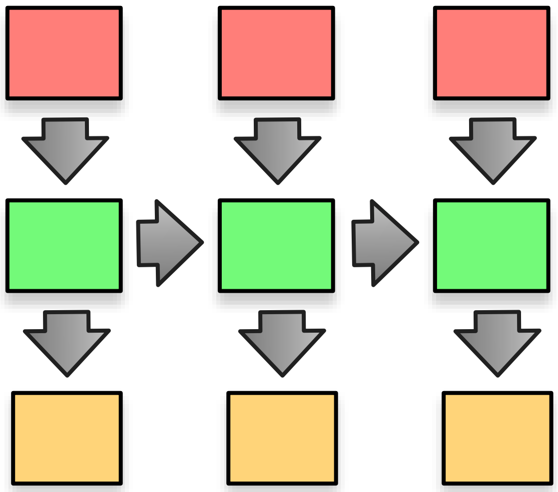
$$\text{combination}_t = f(W\ \text{input}_t + U(\text{reset}_t \circ \text{state}_{t-1}))$$

- Interpolation with previous state

$$\begin{aligned}\text{state}_t =& (1 - \text{update}_t) \circ \text{state}_{t-1} + \\ & \text{update}_t \qquad \circ \text{combination}_t) + \text{bias}\end{aligned}$$

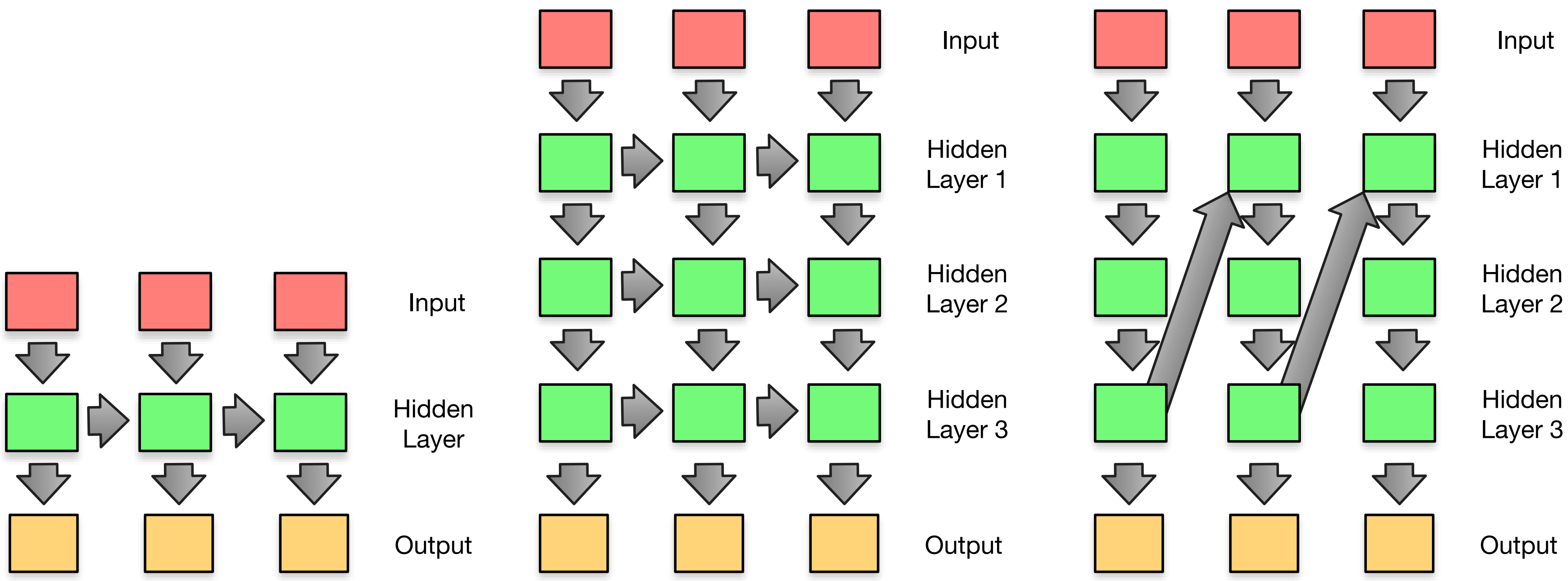
deeper models

Deep Learning?



- Not much **deep** learning so far
- Between prediction from input to output: only 1 hidden layer
- How about more hidden layers?

Deep Models



Shallow **Deep Stacked** **Deep Transition**

TEXT REPRESENTATION AND WORD EMBEDDING IN KERAS

Text representation, word embedding and sentiment analysis in Keras

- Examples from François Chollet’s
  - Article: Deep Learning for Text
  - Book: Deep Learning with Python, 2018, Manning
- Encoding text for use in Keras neural networks
- Word embedding in Keras
- Example: sentiment analysis in IMDB movie reviews using a Dense (hidden) layer
- Example: sentiment analysis in IMDB movie reviews using recurrent neural networks with SimpleRNN and LSTM