Introduction to Neural Machine Translation (2/3)

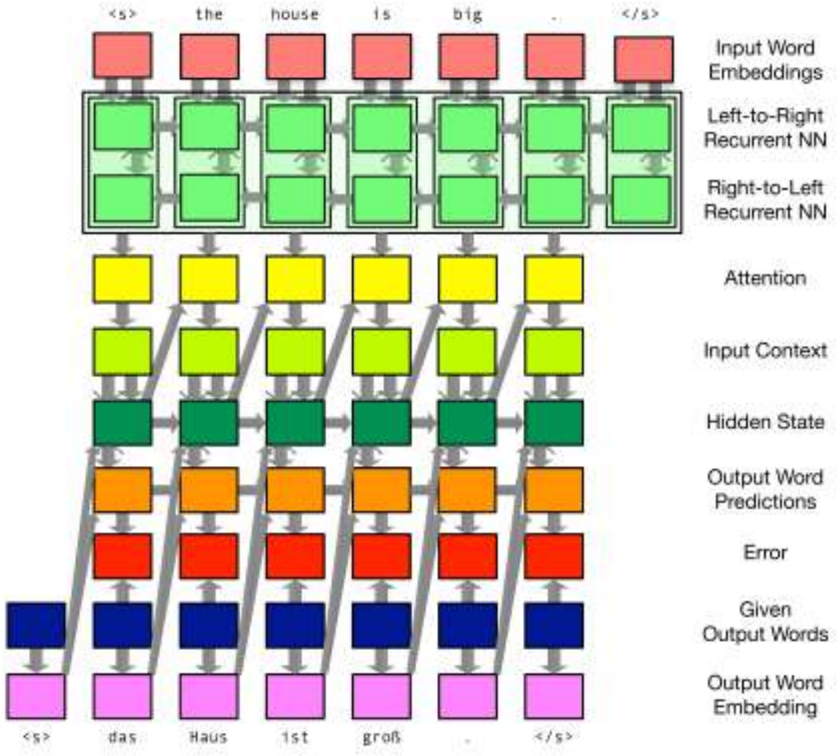
Marine Carpuat

Computer Science

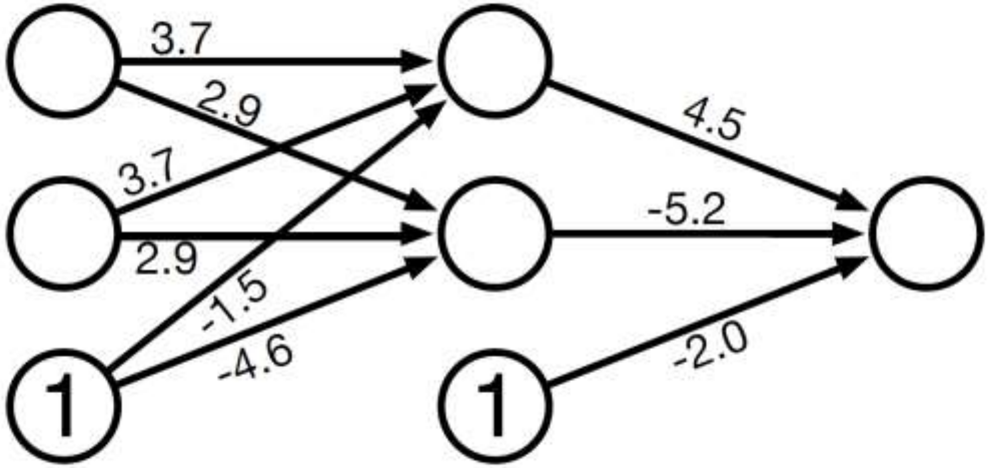
University of Maryland

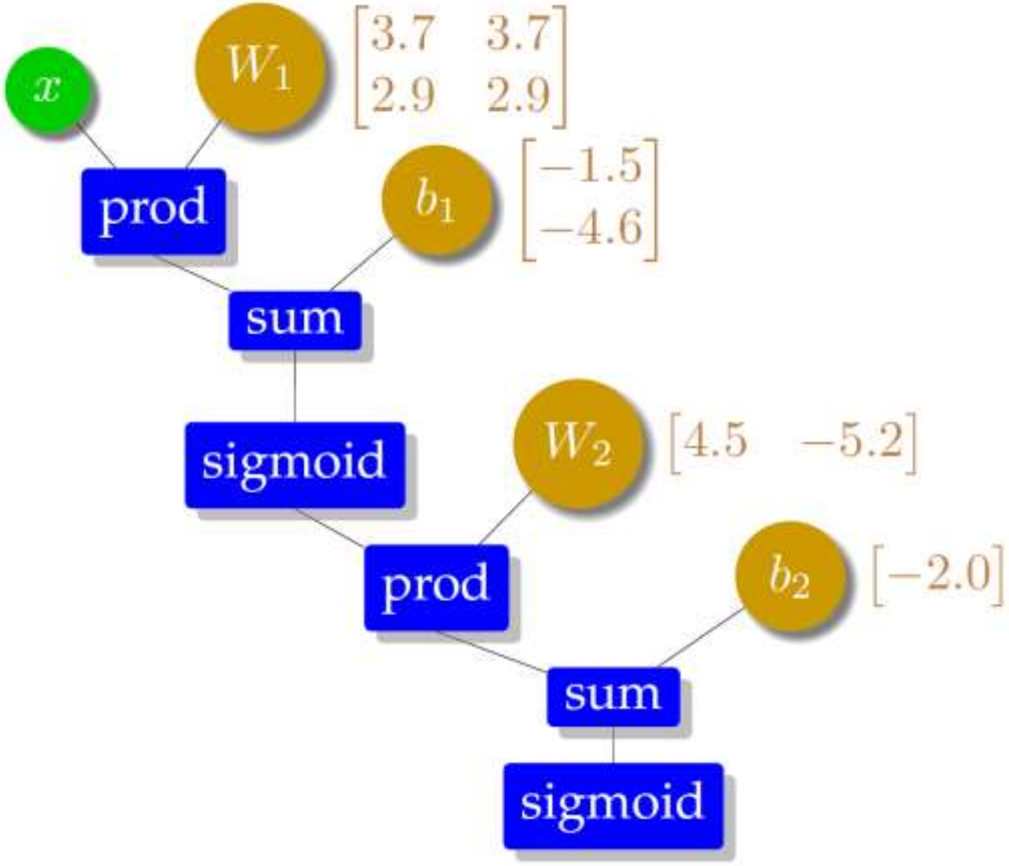
Roadmap

- Evaluating machine translation
- Introduction to neural networks
- Modeling sequences of words with neural language models
- Translating with encoder-decoder models
- Attention mechanism



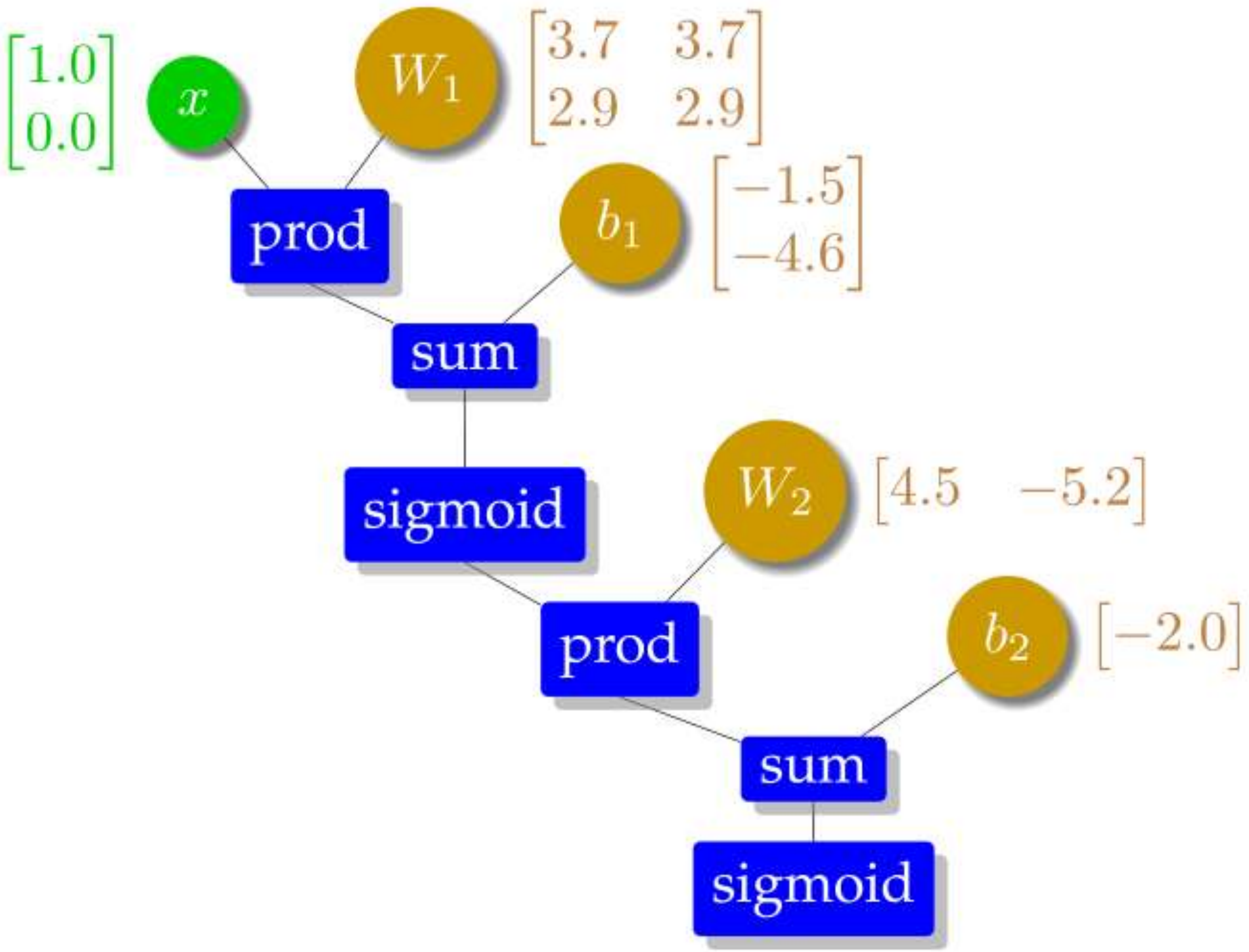
Neural Networks as Computation Graphs



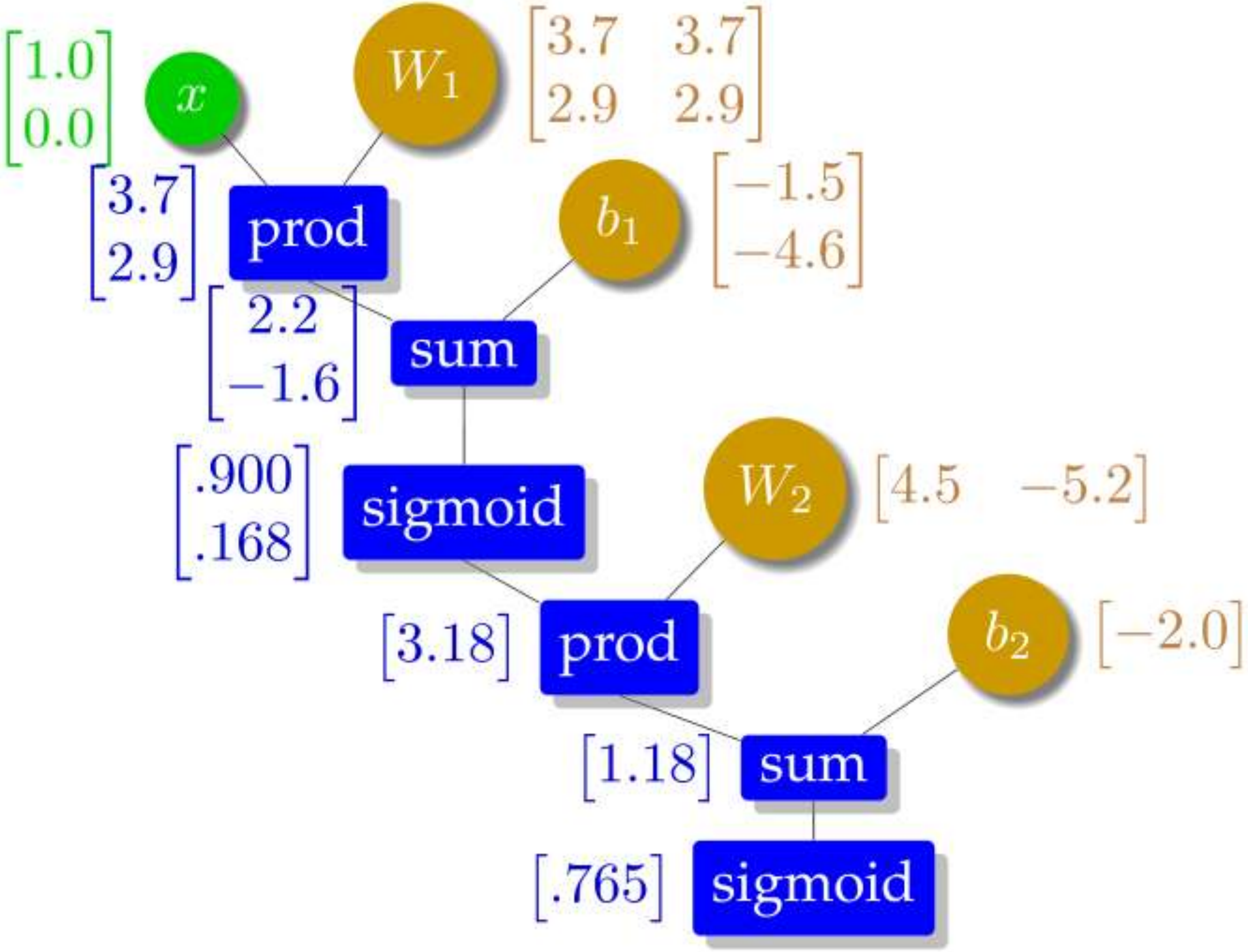


Example & figures by Philipp Koehn

Computation Graphs Make Prediction Easy: Forward Propagation



Computation Graphs Make Prediction Easy: Forward Propagation



Stochastic Gradient Descent

Start with some initial parameter values

- *w = **0***

for *I* iterations

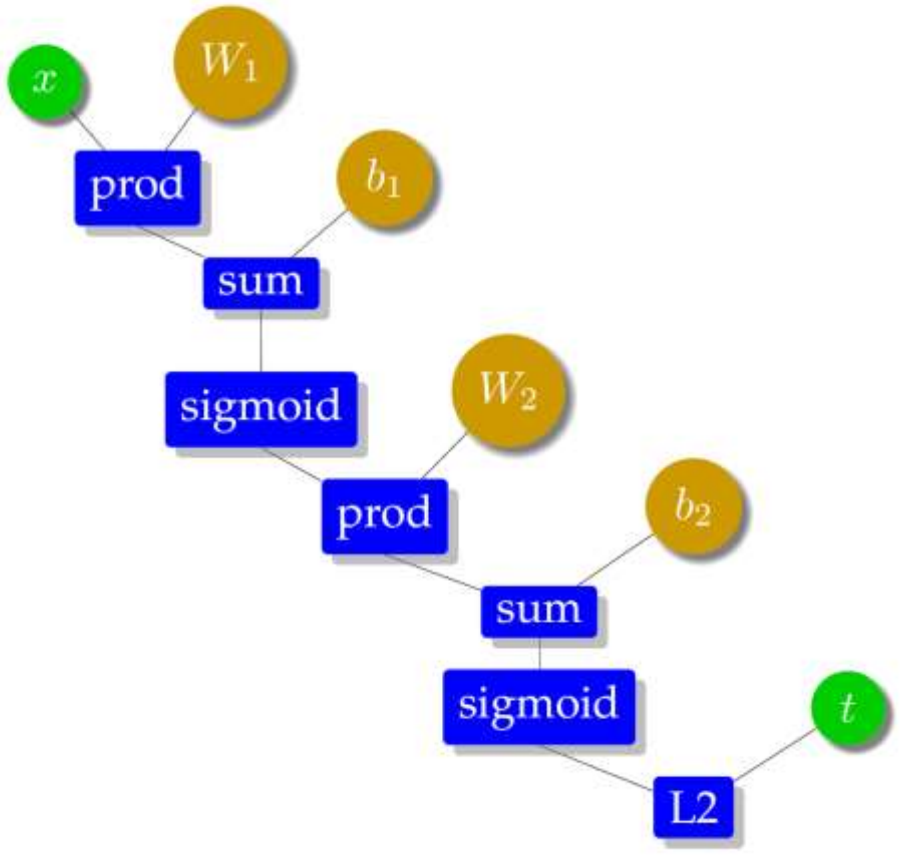
  for each labeled pair *x*, *y* in the data

$$w = w - \mu \frac{d\,\text{error}(w, x, y)}{dw}$$

Go through the training data one example at a time

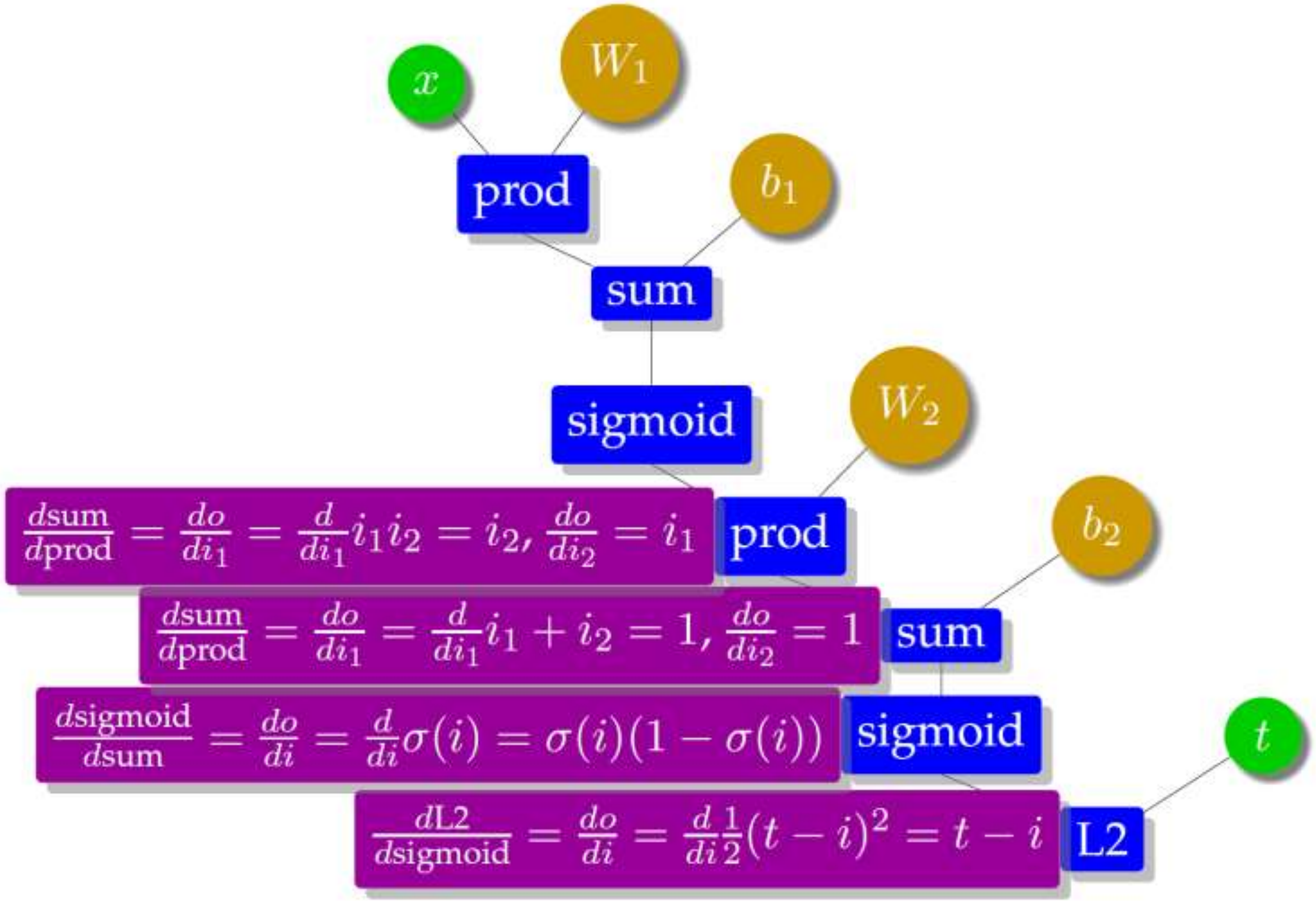
Take a step down the gradient

Computation Graphs Make Training Easy: Computing Error

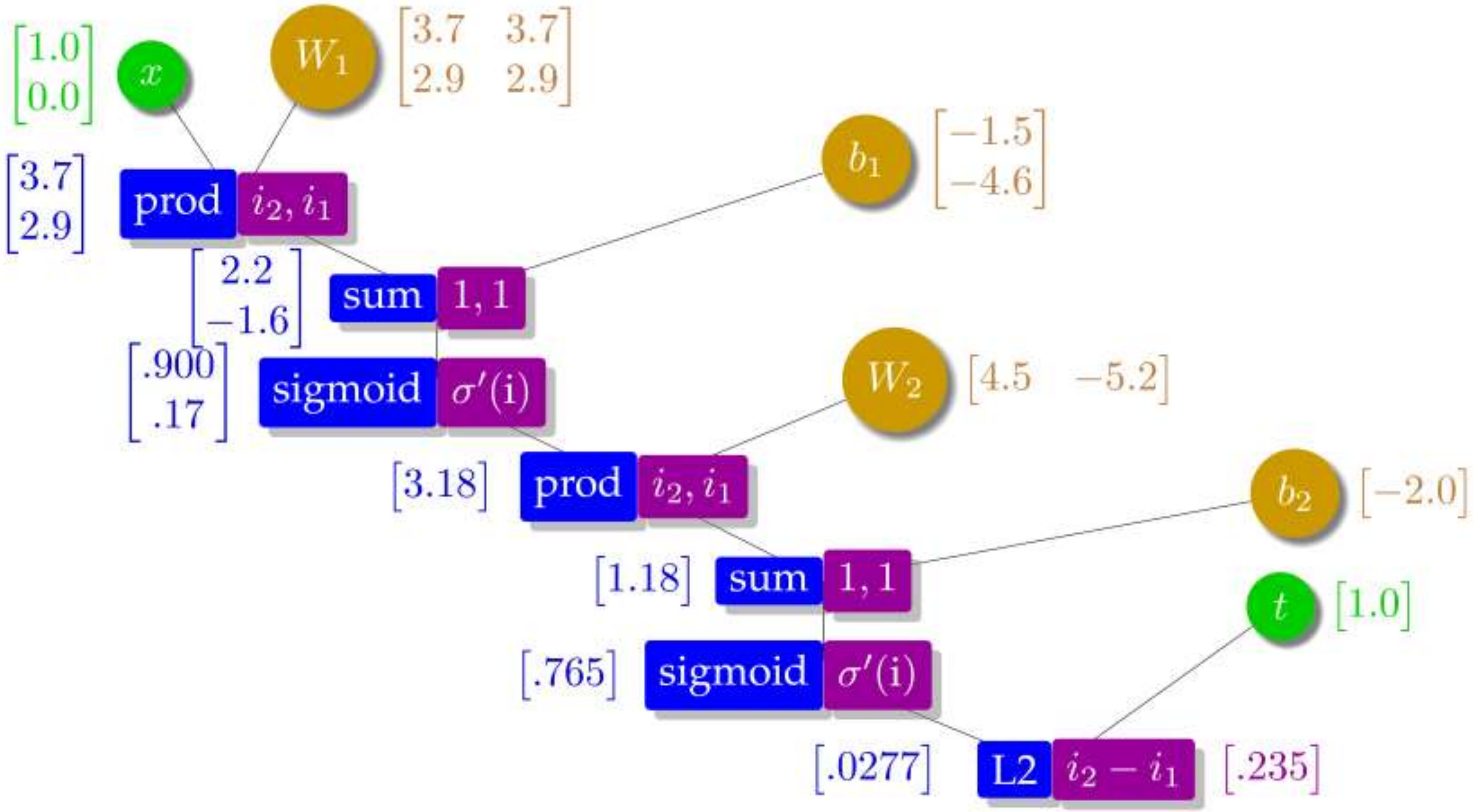


Computation Graphs Make Training Easy: Computing Gradients

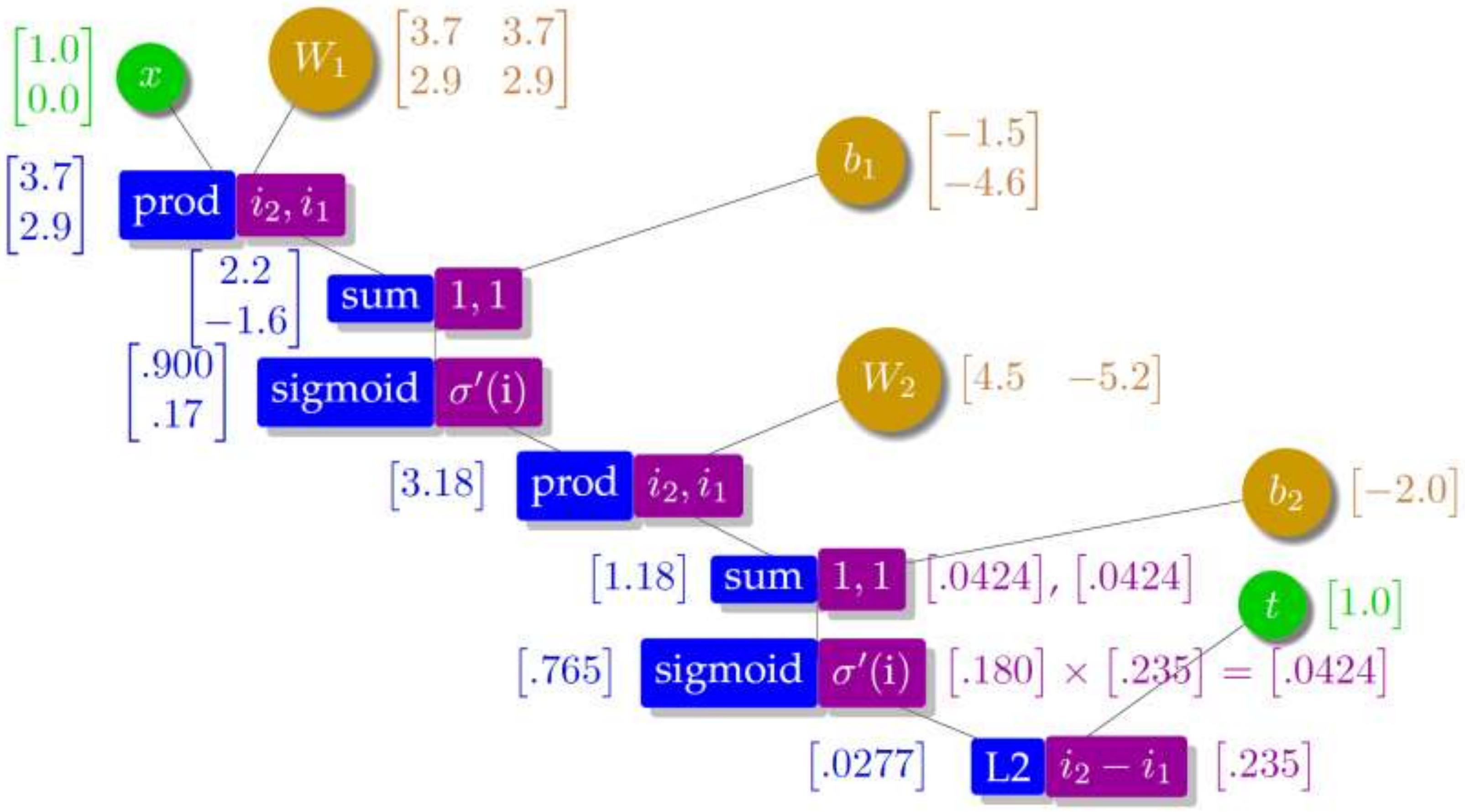
$$\frac{dE}{dA} = \frac{dE}{dB}\frac{dB}{dA}$$



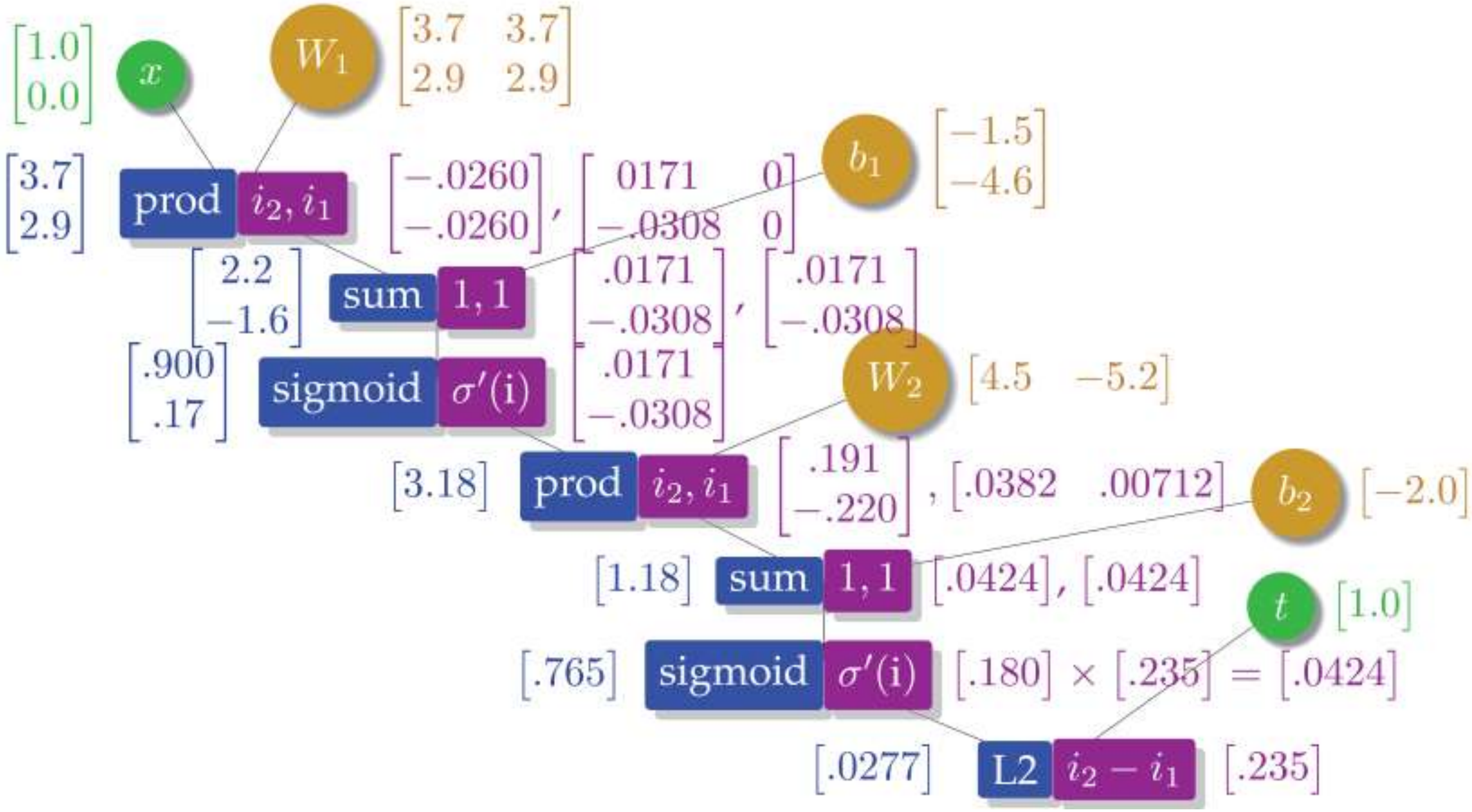
Computation Graphs Make Training Easy: Given forward pass + derivatives for each node



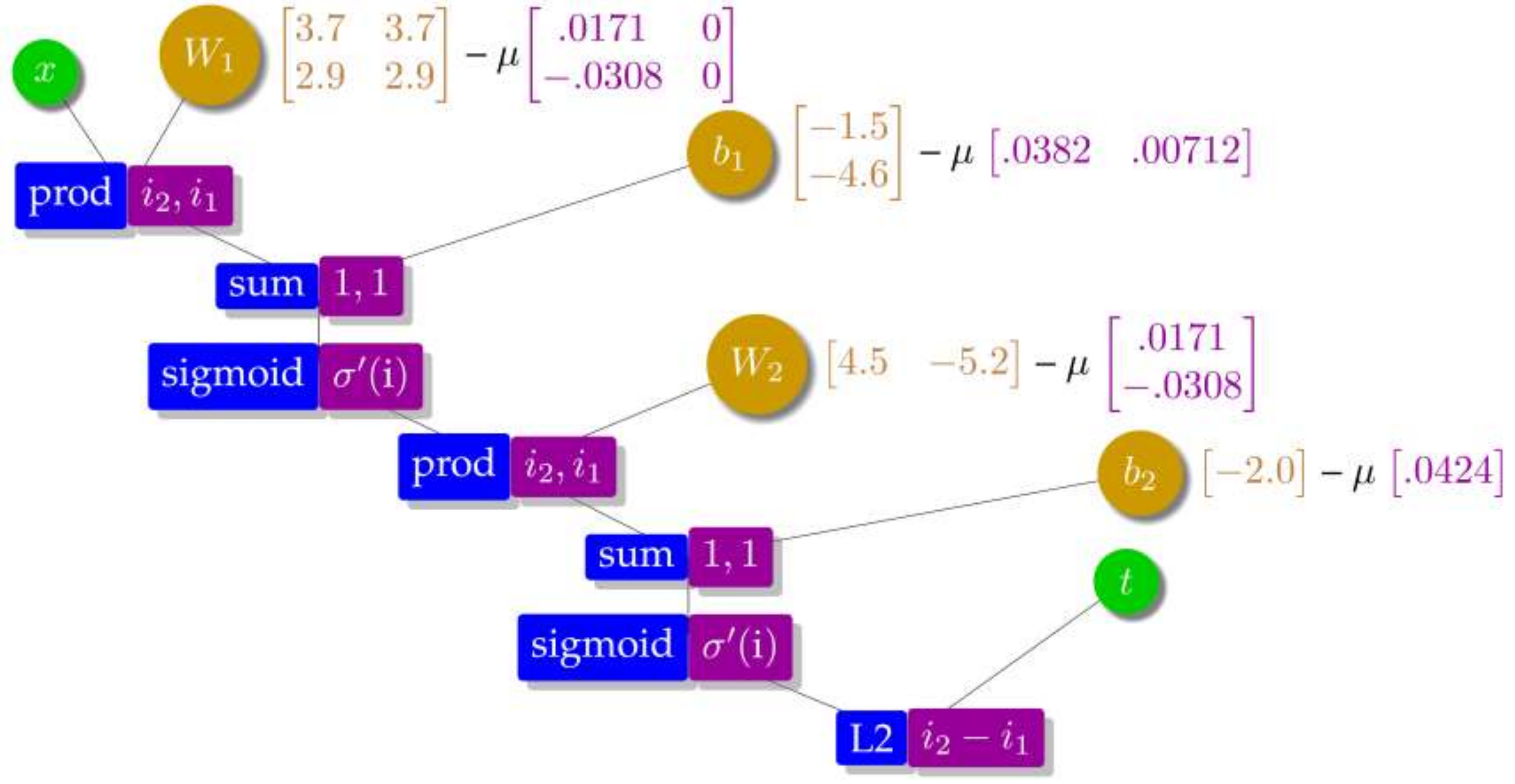
Computation Graphs Make Training Easy: Computing Gradients



Computation Graphs Make Training Easy: Computing Gradients



Computation Graphs Make Training Easy: Updating Parameters



Computation Graph: A Powerful Abstraction

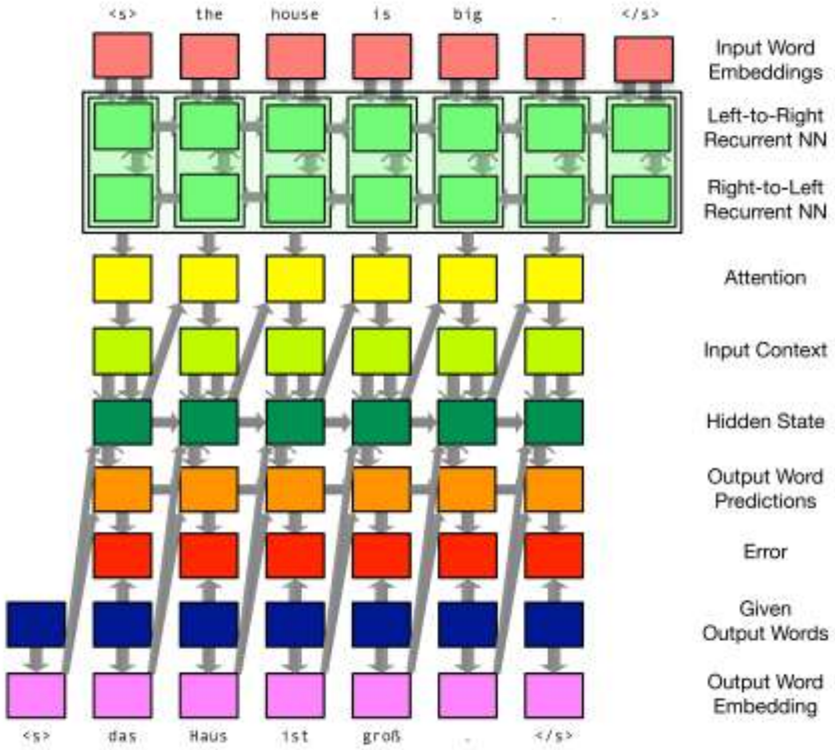
- To build a system, we only need to:
  - Define network structure
  - Define loss
  - Provide data
  - (and set a few more hyperparameters to control training)

- Given network structure
  - Prediction is done by forward pass through graph (forward propagation)
  - Training is done by backward pass through graph (back propagation)
  - Based on simple matrix vector operations

- Forms the basis of neural network libraries
  - Tensorflow, Pytorch, mxnet, etc.

Roadmap

- Evaluating machine translation
- Introduction to neural networks
- Modeling sequences of words with neural language models
- Translating with encoder-decoder models
- Attention mechanism



Language Modeling

- Goal: compute the probability of a sentence or sequence of words

  $P(E) = P(e_1,e_2,e_3,e_4,e_5 \ldots e_n)$

- Related task: probability of an upcoming word

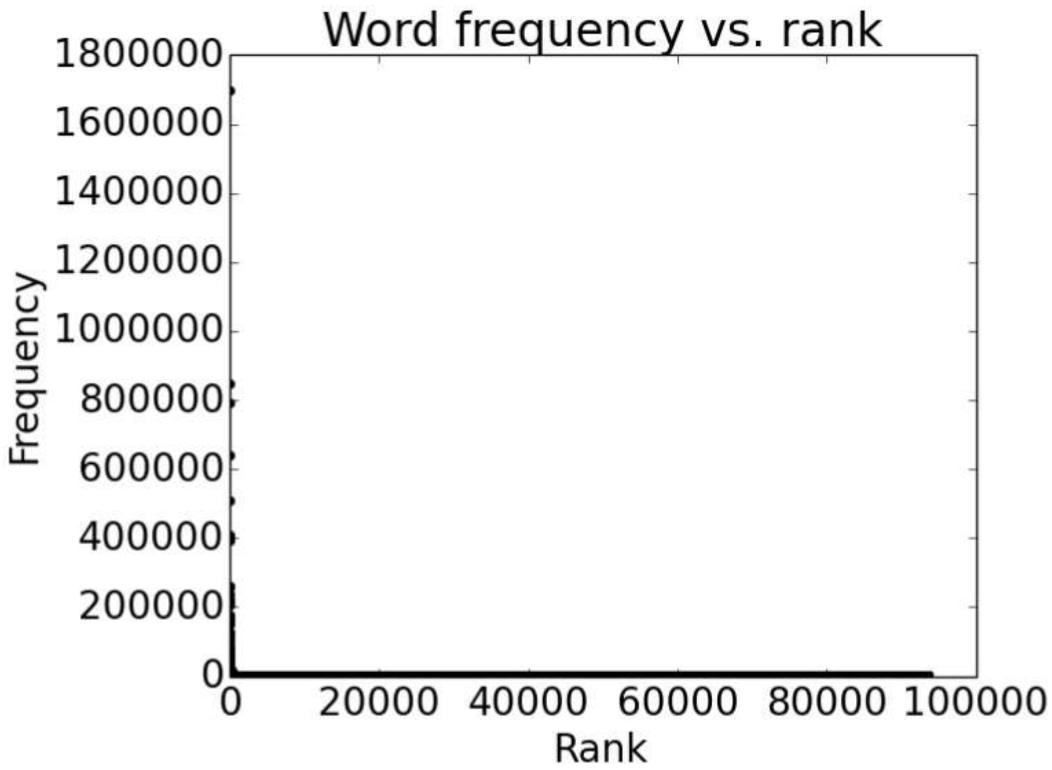
  $P(e_5 | e_1,e_2,e_3,e_4)$

- A model that computes either of these:

  $P(E)$ or $P(e_n | e_1,e_2 \ldots e_{n-1})$

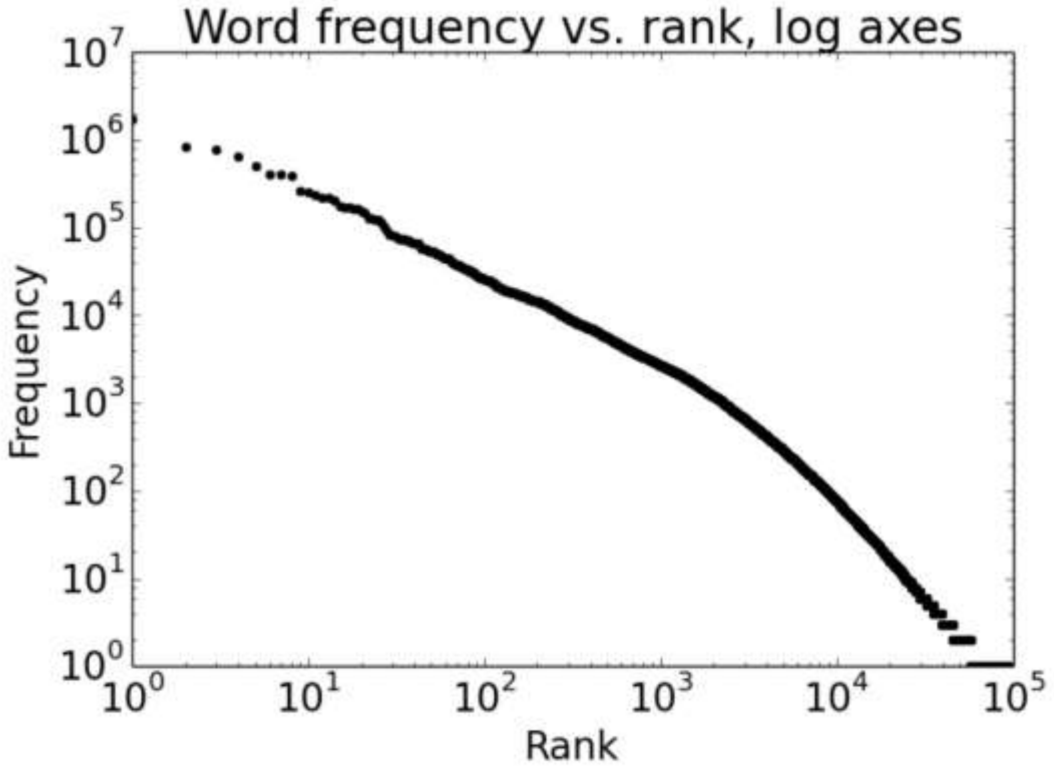
  is called a **language model**.

Zipf’s Law



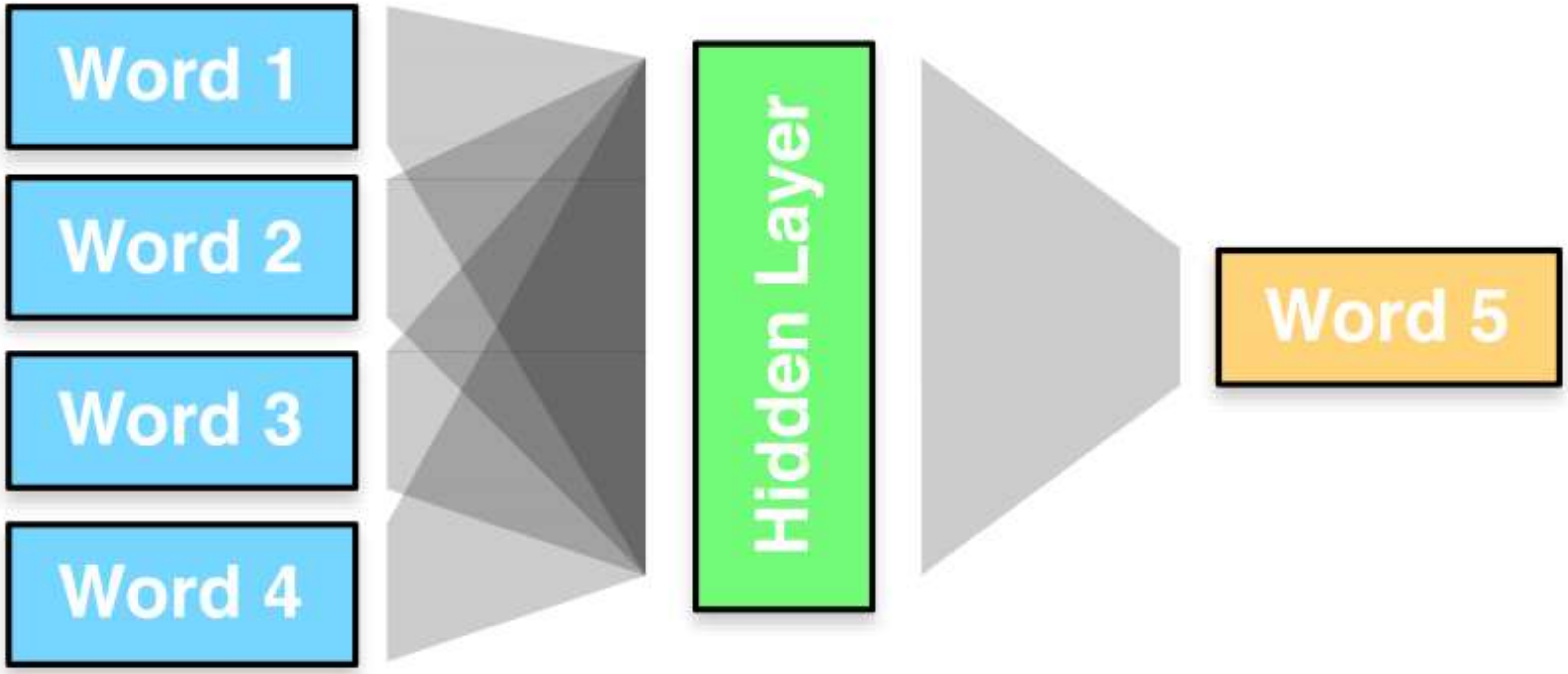
Word frequency vs. rank
Frequency
Rank
1800000
1600000
1400000
1200000
1000000
800000
600000
400000
200000
0
0
20000
40000
60000
80000
100000

Zipf’s Law



- Even in a very large corpus, there will be a lot of infrequent words
- The same holds for many other levels of linguistic structure
- NLP/MT challenge: we need to be able to make predictions for things we have rarely or never seen

Toward a Neural Language Model



Figures by Philipp Koehn (JHU)

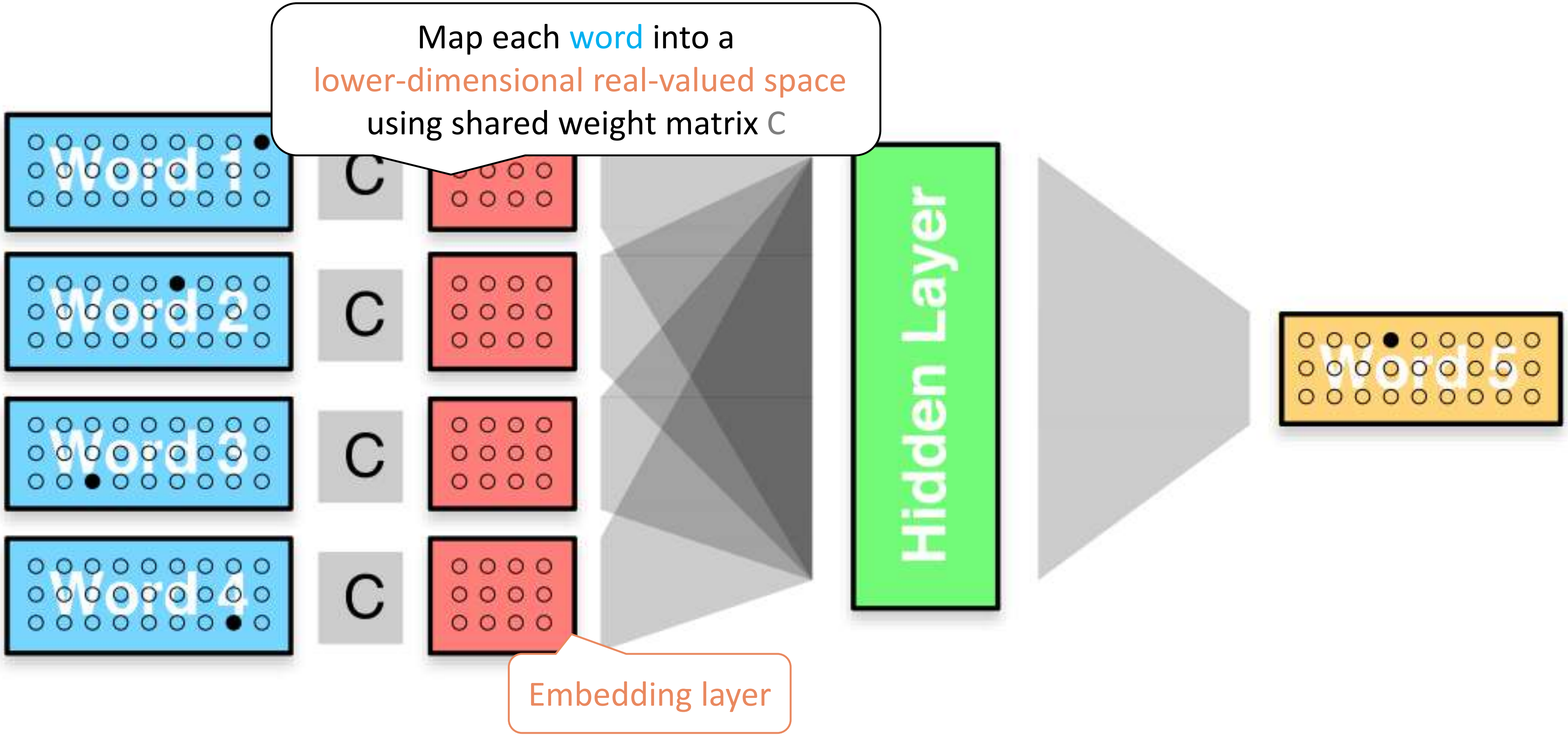
Representing Words

- "one hot vector"

```
dog = [ 0, 0, 0, 0, 1, 0, 0, 0 …]
cat = [ 0, 0, 0, 0, 0, 0, 1, 0 …]
eat = [ 0, 1, 0, 0, 0, 0, 0, 0 …]
```

- That's a large vector! practical solutions:
  - limit to most frequent words (e.g., top 20000)
  - cluster words into classes
  - break up rare words into subword units

Language Modeling with Feedforward Neural Networks



Bengio et al. 2003

An Output Layer to Predict Words

- Network will output a probability for each word in the vocabulary V
- Step 1: compute a score for each word in V

$$s = Wx + b$$

$s \in \mathbb{R}^{|V|}$

$W \in \mathbb{R}^{|V| \times N}$

$b \in \mathbb{R}^{|V|}$

- Step 2: turn scores into probabilities using softmax function

$$p = \mathrm{softmax}(s)$$

Where the probability of the j-th word in V is $p_j = \dfrac{e^{s_j}}{\sum_{\tilde{j}} e^{s_{\tilde{j}}}}$

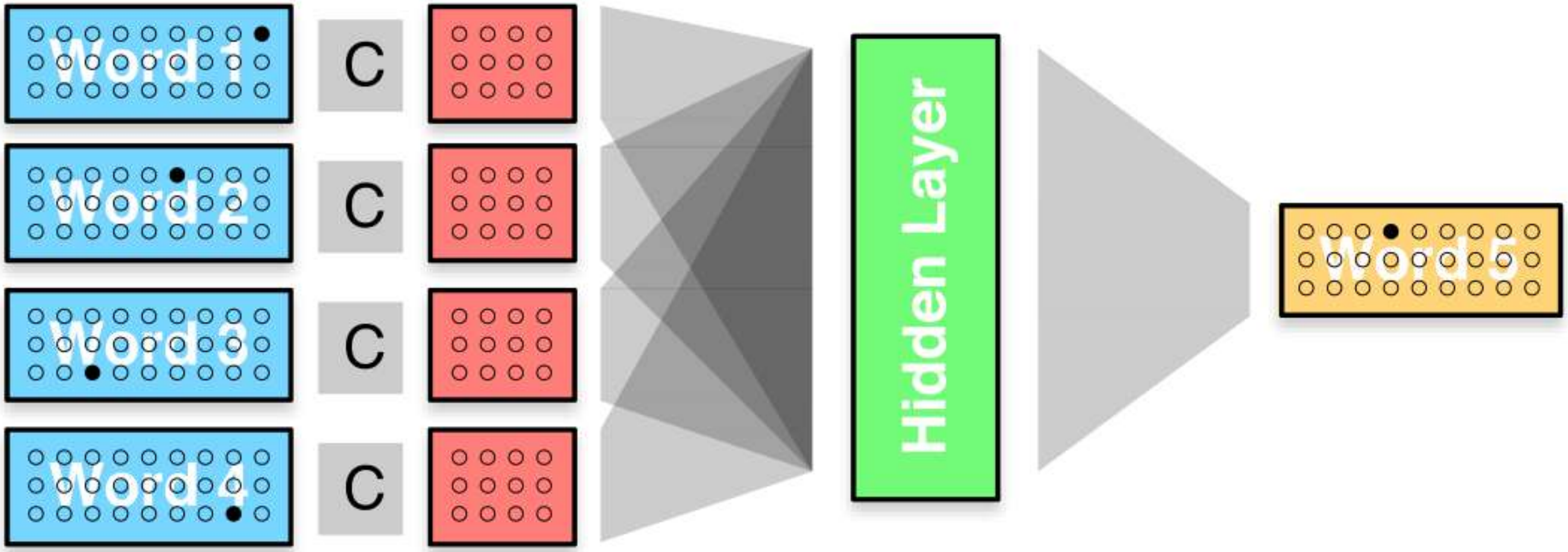
Estimating Model Parameters

- Intuition: a model is good if it gives high probability to existing word sequences

- Training examples:
  - sequences of words in the language of interest

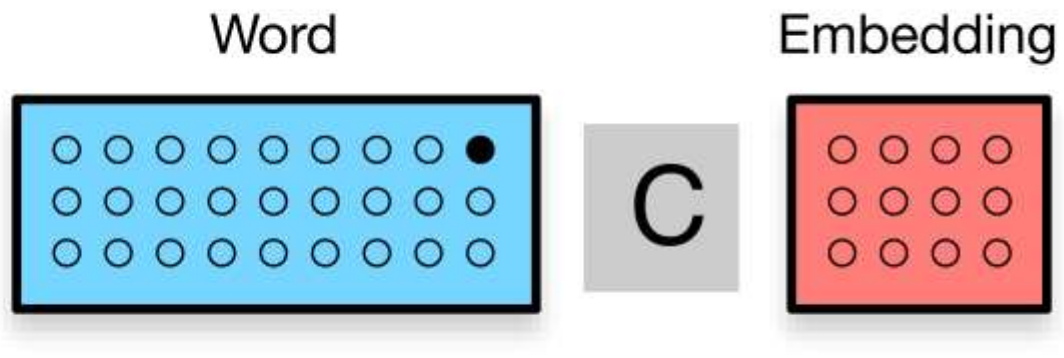
- Error/loss: negative log likelihood
  - At the corpus level $\mathrm{error}(\lambda) = -\sum_{E \text{ in corpus}} \log P_\lambda(E)$
  - At the word level $\mathrm{error}(\lambda) = -\log P_\lambda(e_t | e_1 \ldots e_{t-1})$

Language Modeling with Feedforward Neural Networks



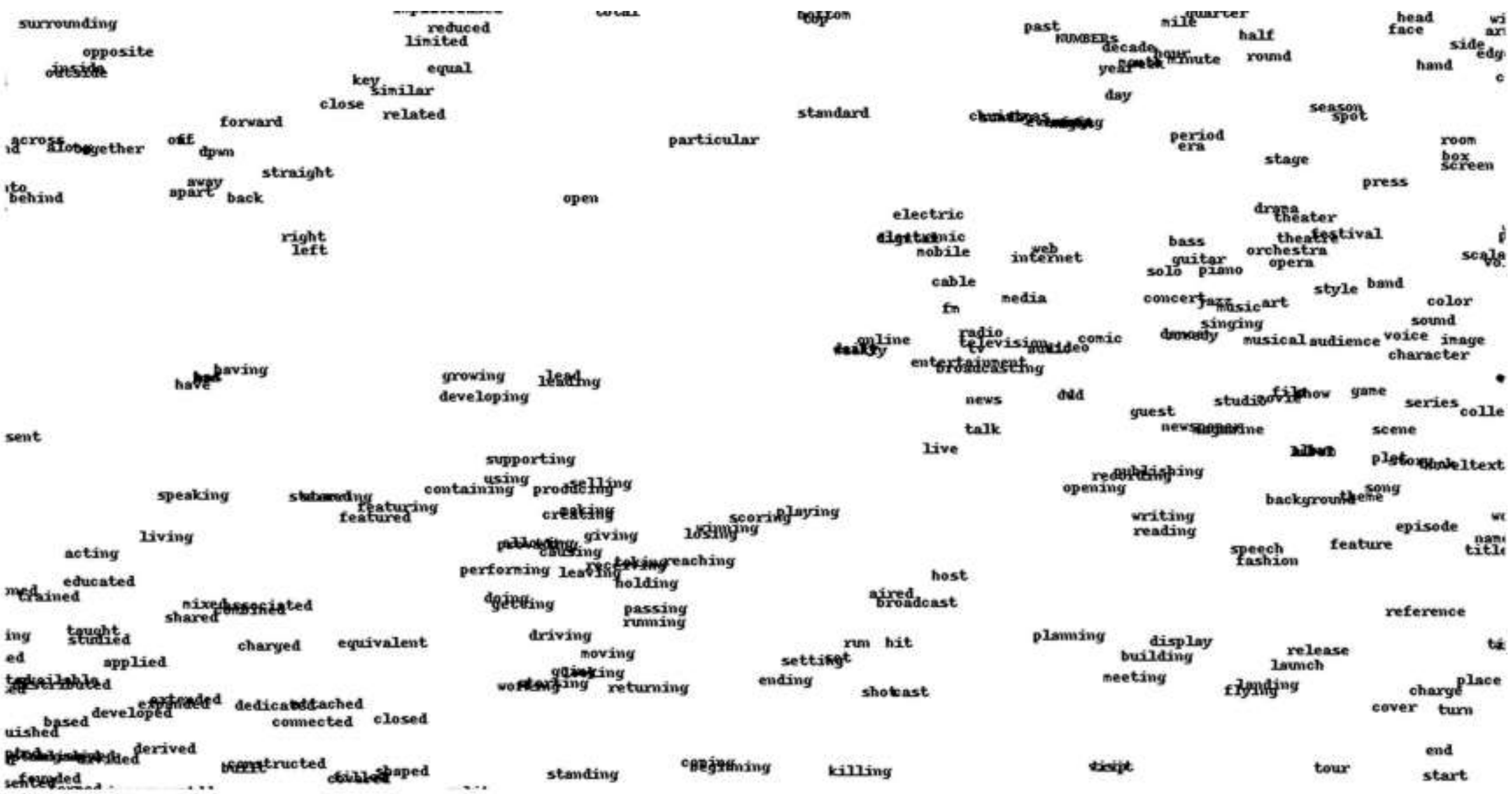
Bengio et al. 2003

Word Embeddings: a useful by-product of neural LMs

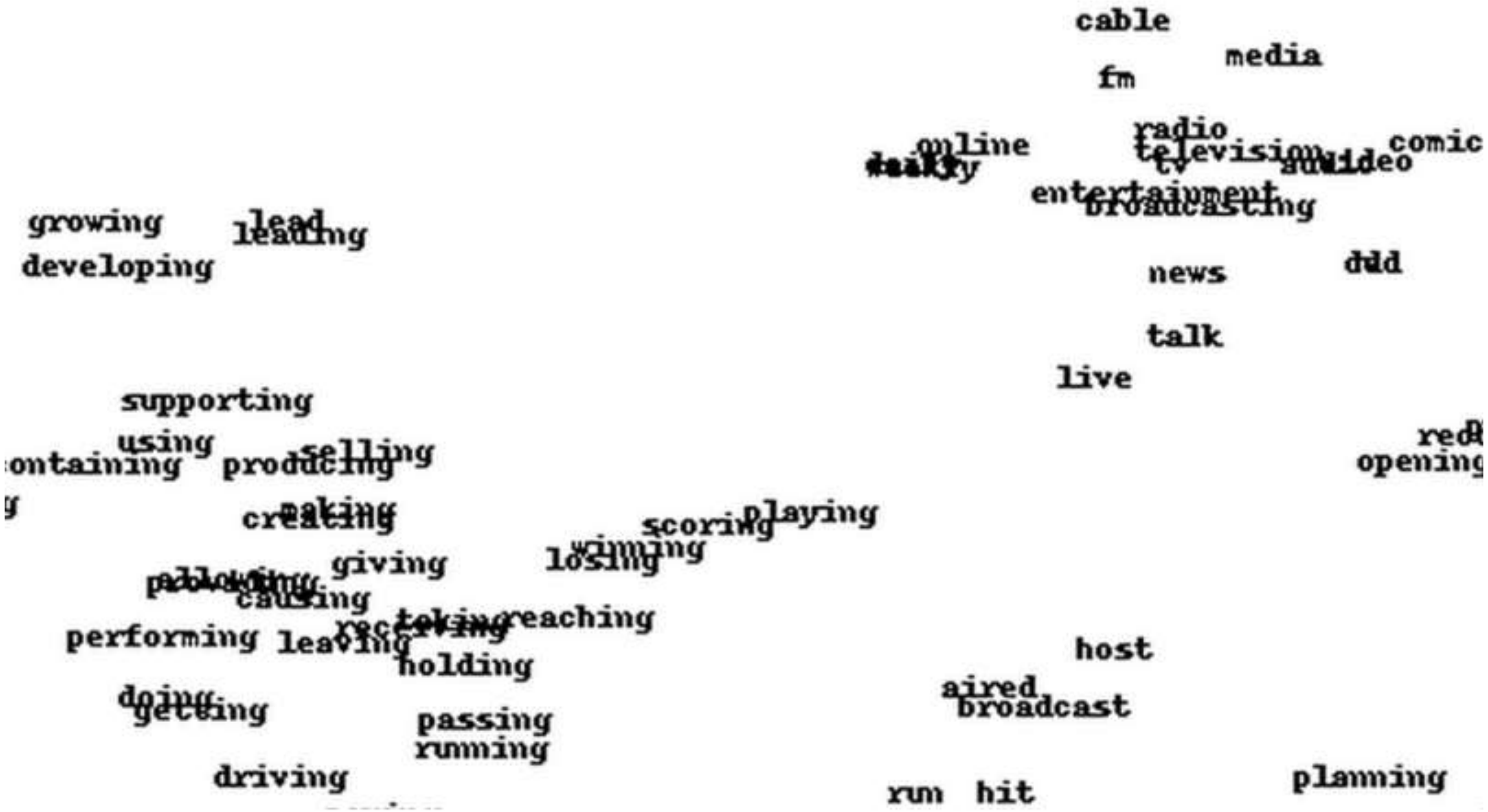


- Words that occurs in similar contexts tend to have similar embeddings
- Embeddings capture many usage regularities
- Useful features for many NLP tasks

Word Embeddings



Word Embeddings

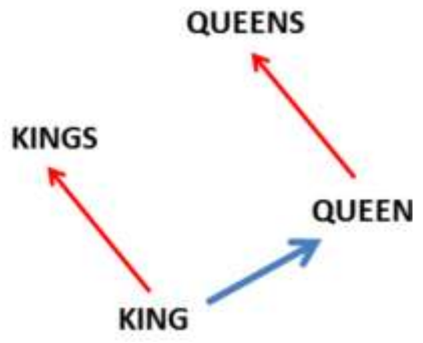


Word Embeddings Capture Useful Regularities

Morpho-Syntactic

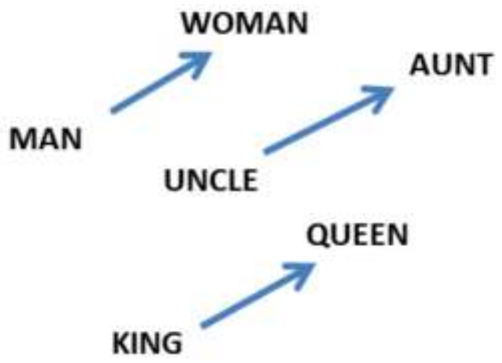
- Adjectives: base form vs. comparative
- Nouns: singular vs. plural
- Verbs: present tense vs. past tense

[Mikolov et al. 2013]

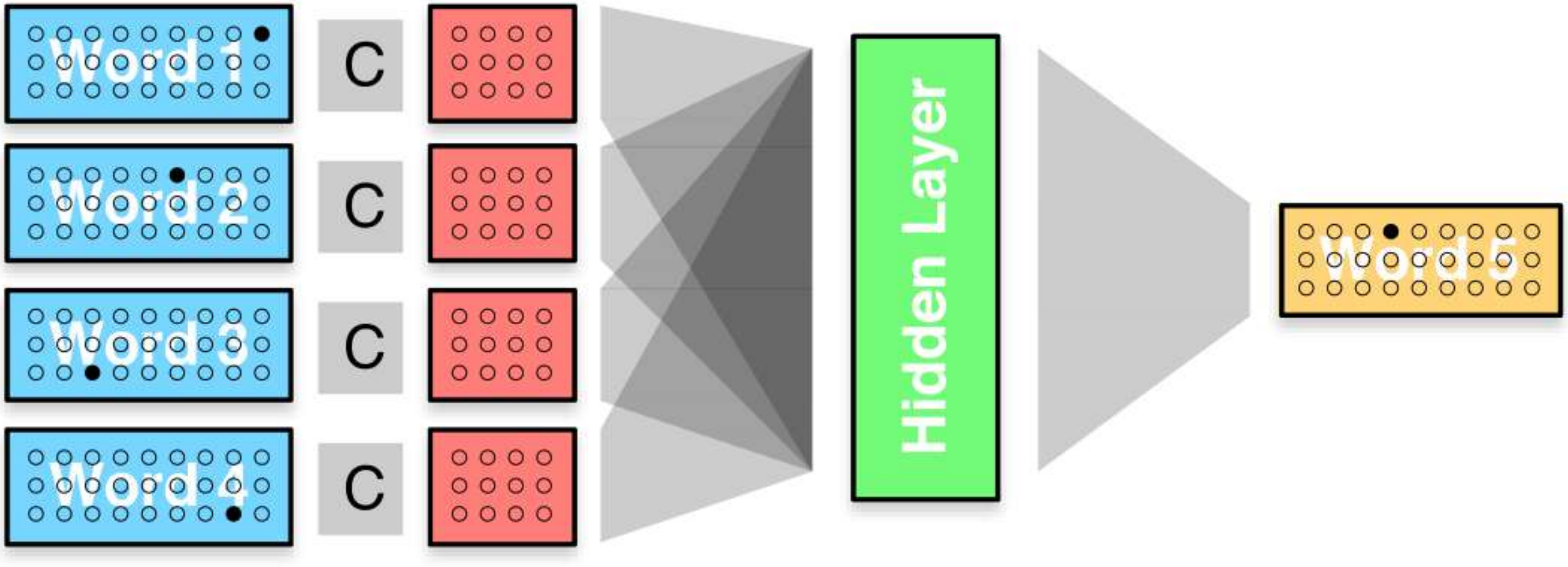


Semantic

- Word similarity/relatedness
- Semantic relations
- But tends to fail at distinguishing
  - Synonyms vs. antonyms
  - Multiple senses of a word



Language Modeling with Feedforward Neural Networks

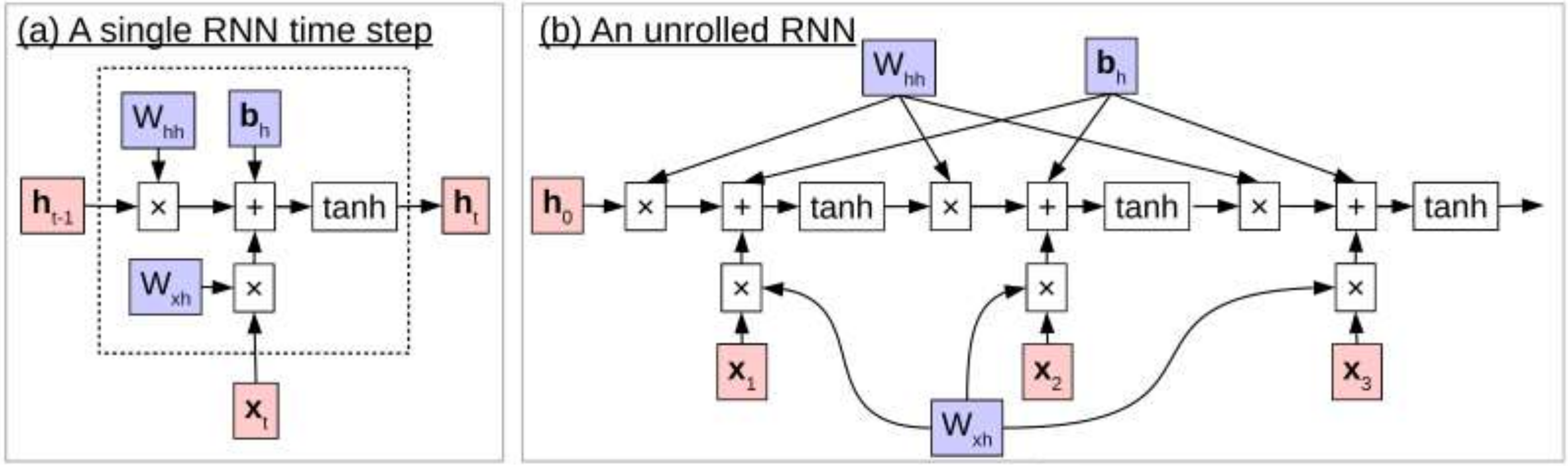


Bengio et al. 2003

Language Modeling with Recurrent Neural Networks

Figure by Philipp Koehn

Formalizing our Recurrent Language Model



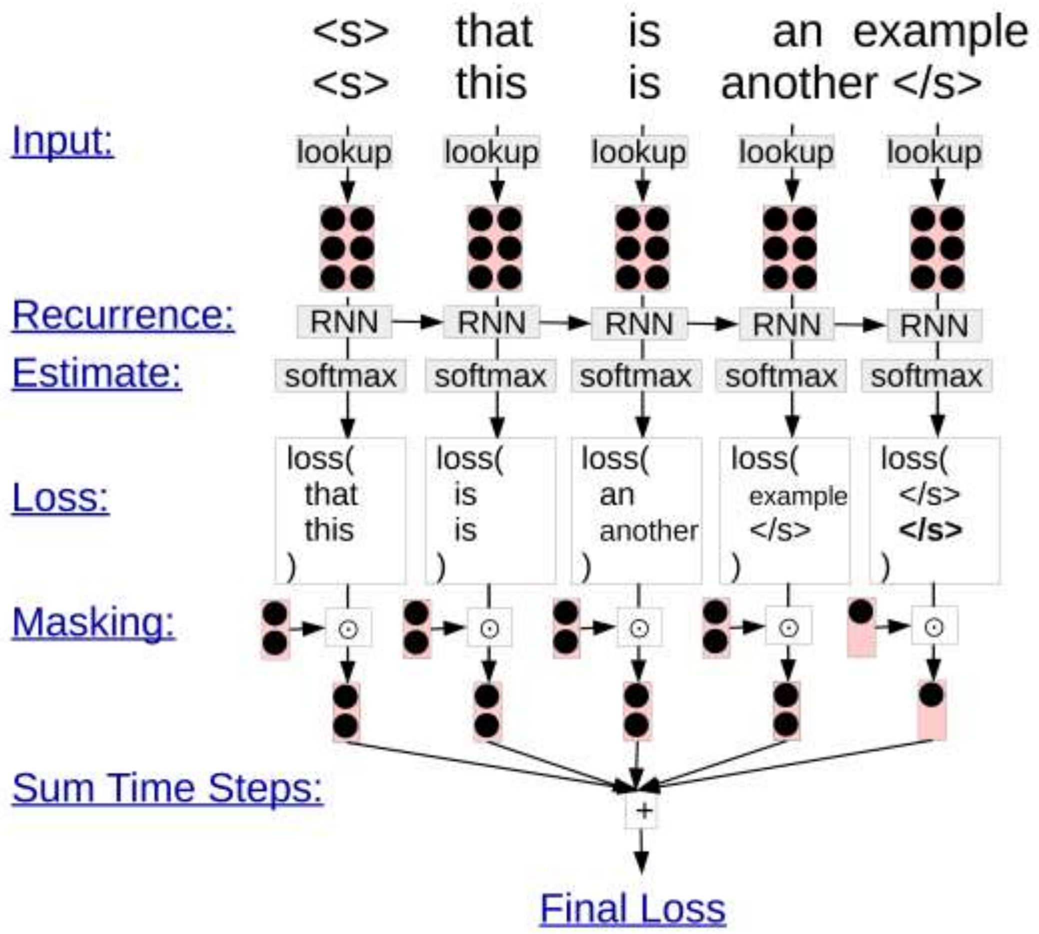
$$\boldsymbol{m}_t = M_{\cdot, e_{t-1}}$$

$$\boldsymbol{h}_t = \begin{cases} \tanh(W_{mh}\boldsymbol{m}_t + W_{hh}\boldsymbol{h}_{t-1} + \boldsymbol{b}_h) & t \geq 1, \\ \boldsymbol{0} & \text{otherwise.} \end{cases}$$

$$\boldsymbol{p}_t = \text{softmax}(W_{hs}\boldsymbol{h}_t + b_s).$$

Figure by Graham Neubig

Practical Training Issues



- Process examples as a "minibatch"
  - yields better models faster
- Vanishing/Exploding Gradients
  - can be handled with variant of RNN architecture (Long Short Term Memory Networks)

Figure by Graham Neubig

What do Recurrent Language Models Learn?

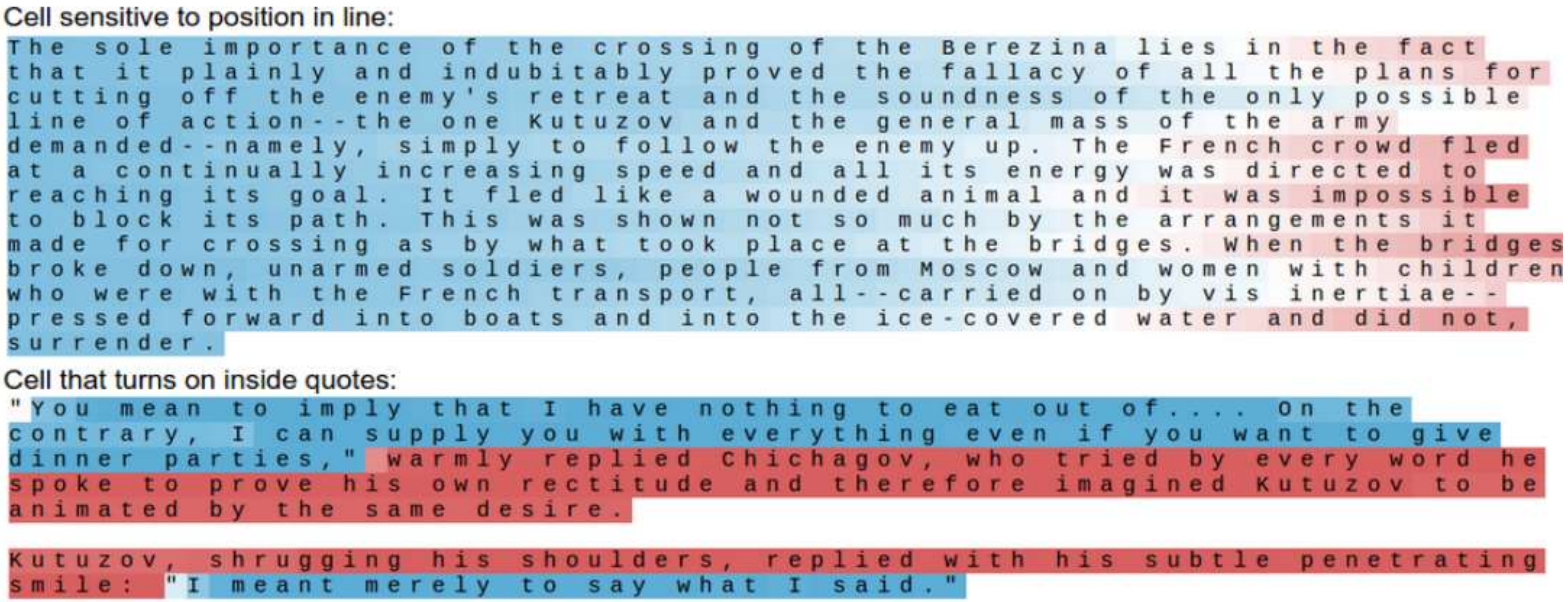
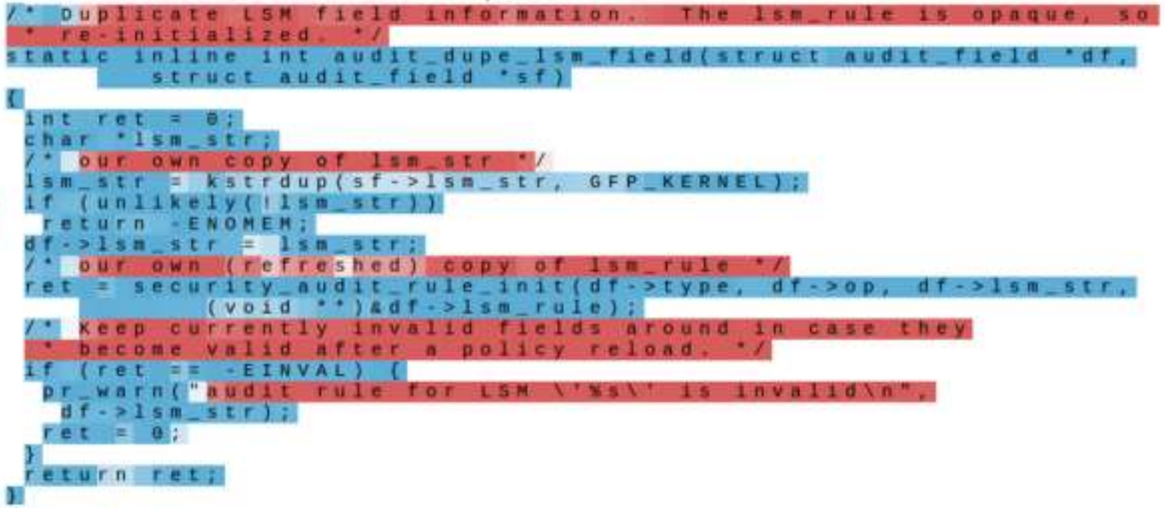


Figure from Karpathy 2015

What do Recurrent Language Models Learn?

Cell that turns on inside comments and quotes:



Cell that robustly activates inside if statements:

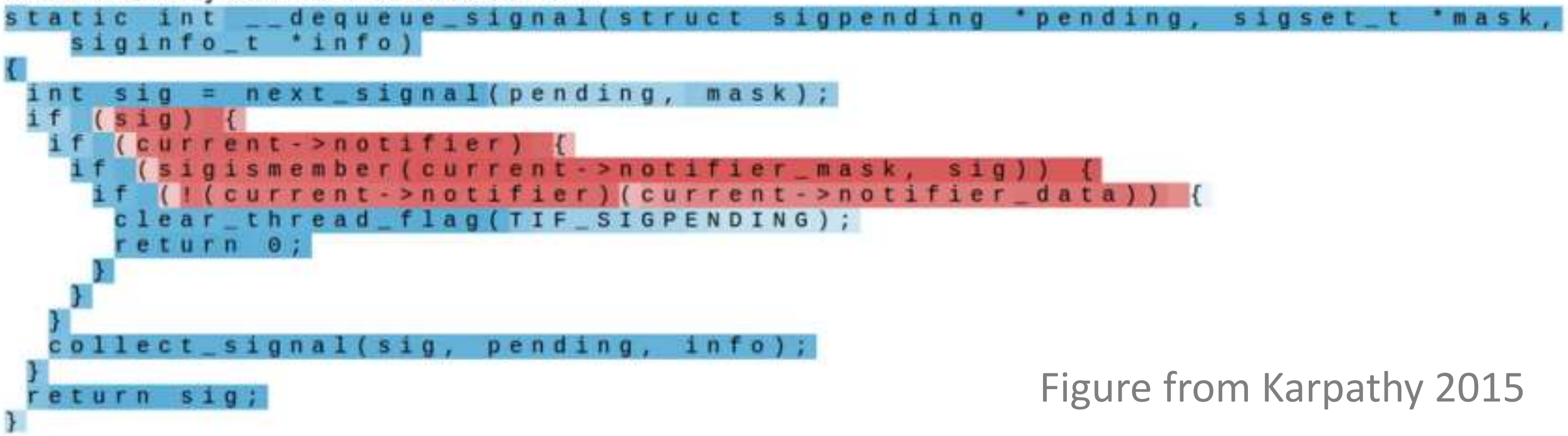


Figure from Karpathy 2015

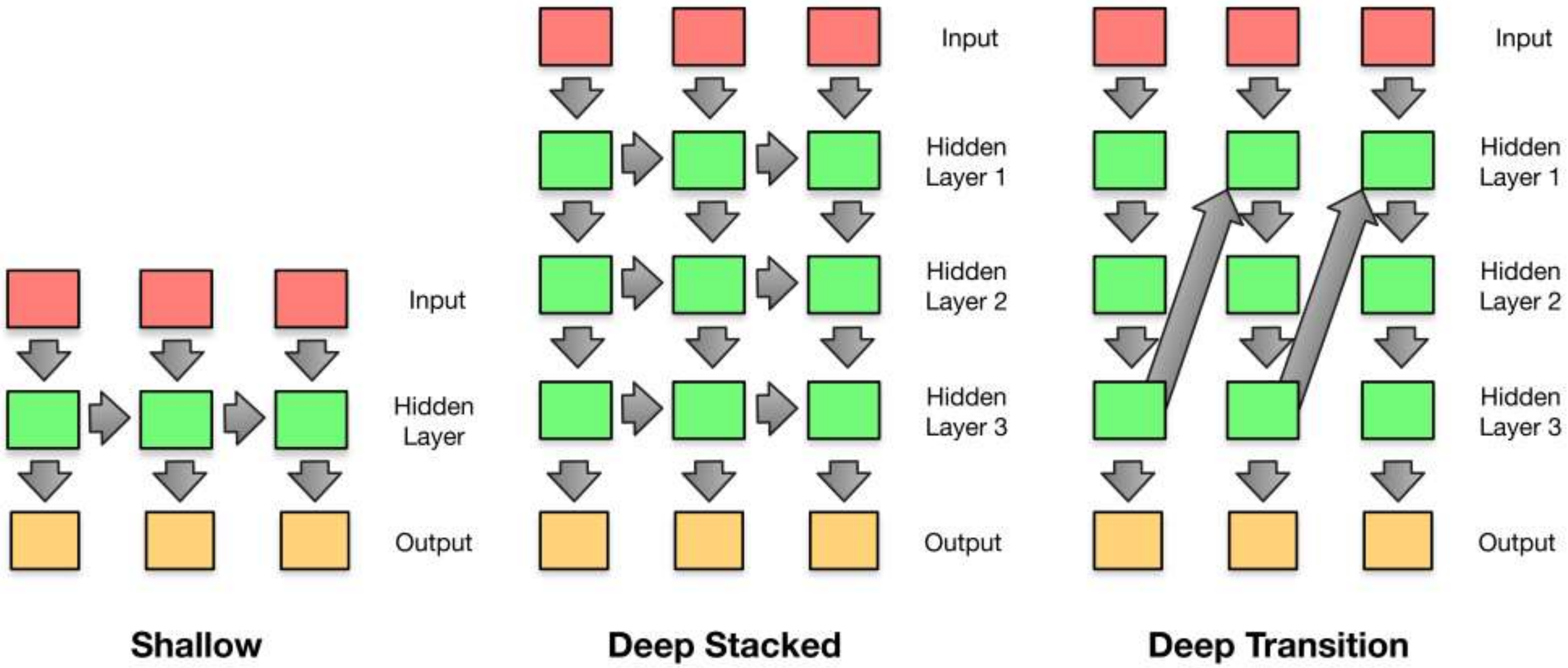
What do Recurrent Language Models Learn?

- Can capture (some) long-distance dependencies

```
After much economic progress over the years, the country has…

The country, which has made much economic progress over the years, still has…
```

Deeper Models



Recurrent Neural Language Models Summary

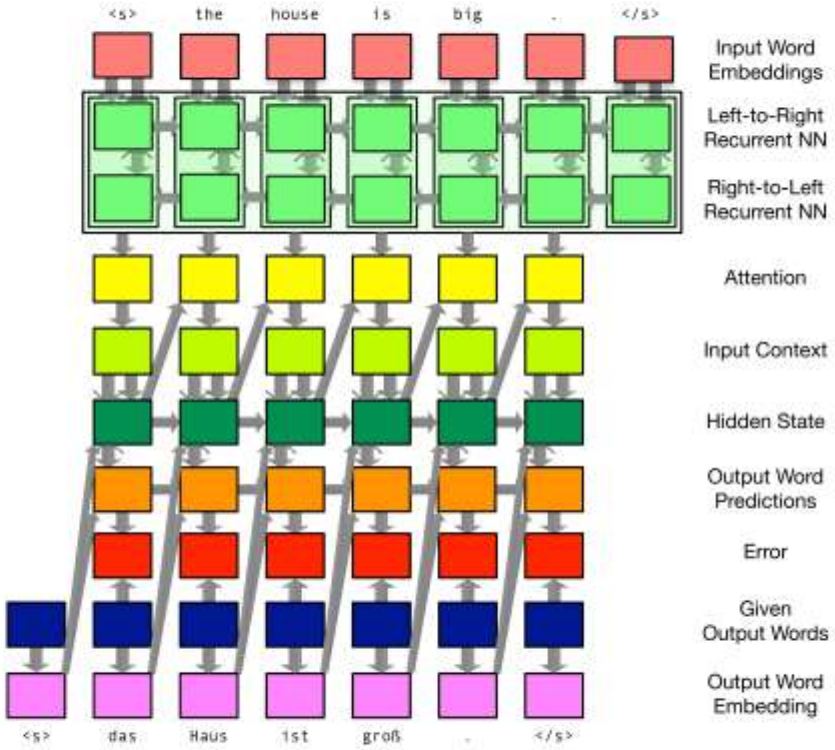
- A powerful tool for modeling language
  - Captures generalizations over words via embeddings
  - Captures some long-distance dependencies

- Many tricks required to train and predict efficiently

- Helps performance in hard extrinsic tasks
  - speech recognition (Mikolov et al. 2011)
  - machine translation (Devlin et al. 2014)

Roadmap

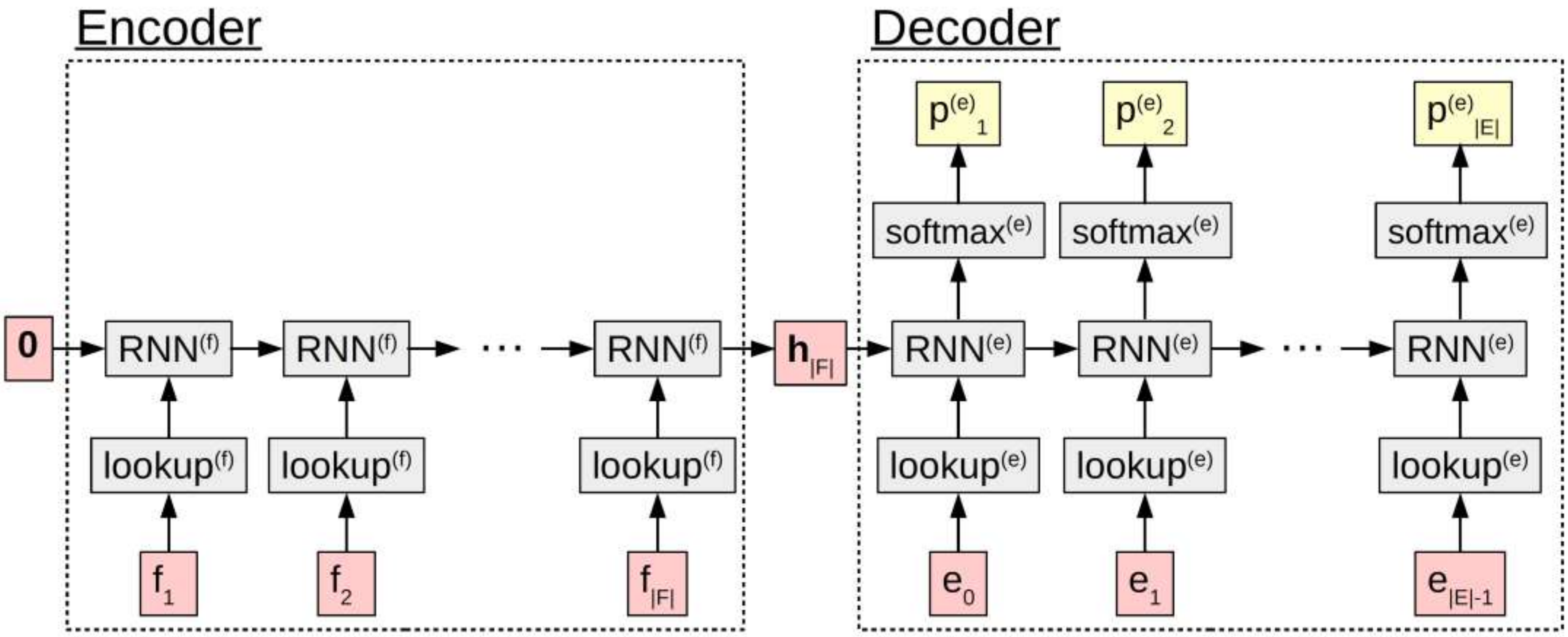
- Evaluating machine translation
- Introduction to neural networks
- Modeling sequences of words with neural language models
- Translating with encoder-decoder models
- Attention mechanism



From Language Modeling to Translation

- Language models give us P(E)
  - Where E is a sentence in a language, say English
- A translation model can be defined as P(E|F)
  - Where E is an English sentence
  - And F is a French sentence

RNN Encoder-Decoder Translation Model



Training

- Same as for RNN language modeling
- Training examples: pairs of sentences (E,F)
- Loss function
  - Negative log-likelihood of training data
  - Total loss for one example (sentence) = sum of loss at each time step (word)

Note that training loss differs from evaluation metric (BLEU)

N-gram overlap between machine translation output and reference translation

Compute precision for n-grams of size 1 to 4

Add brevity penalty (for too short translations)

$$\text{BLEU} = \min\left(1, \frac{\textit{output-length}}{\textit{reference-length}}\right) \Big(\prod_{i=1}^{4} \textit{precision}_i\Big)^{\frac{1}{4}}$$

Typically computed over the entire corpus, not single sentences

Generating Output

- We have a model P(E|F), how can we generate translations?

- 2 methods
  - **Sampling**: generate a random sentence according to P(E|F)

  - **Argmax**: generate sentence with highest probability

$$\hat{E} = \mathrm{argmax}_E P(E|F)$$

Ancestral Sampling

- Randomly generate words one by one

- Until end of sentence symbol

- Done!

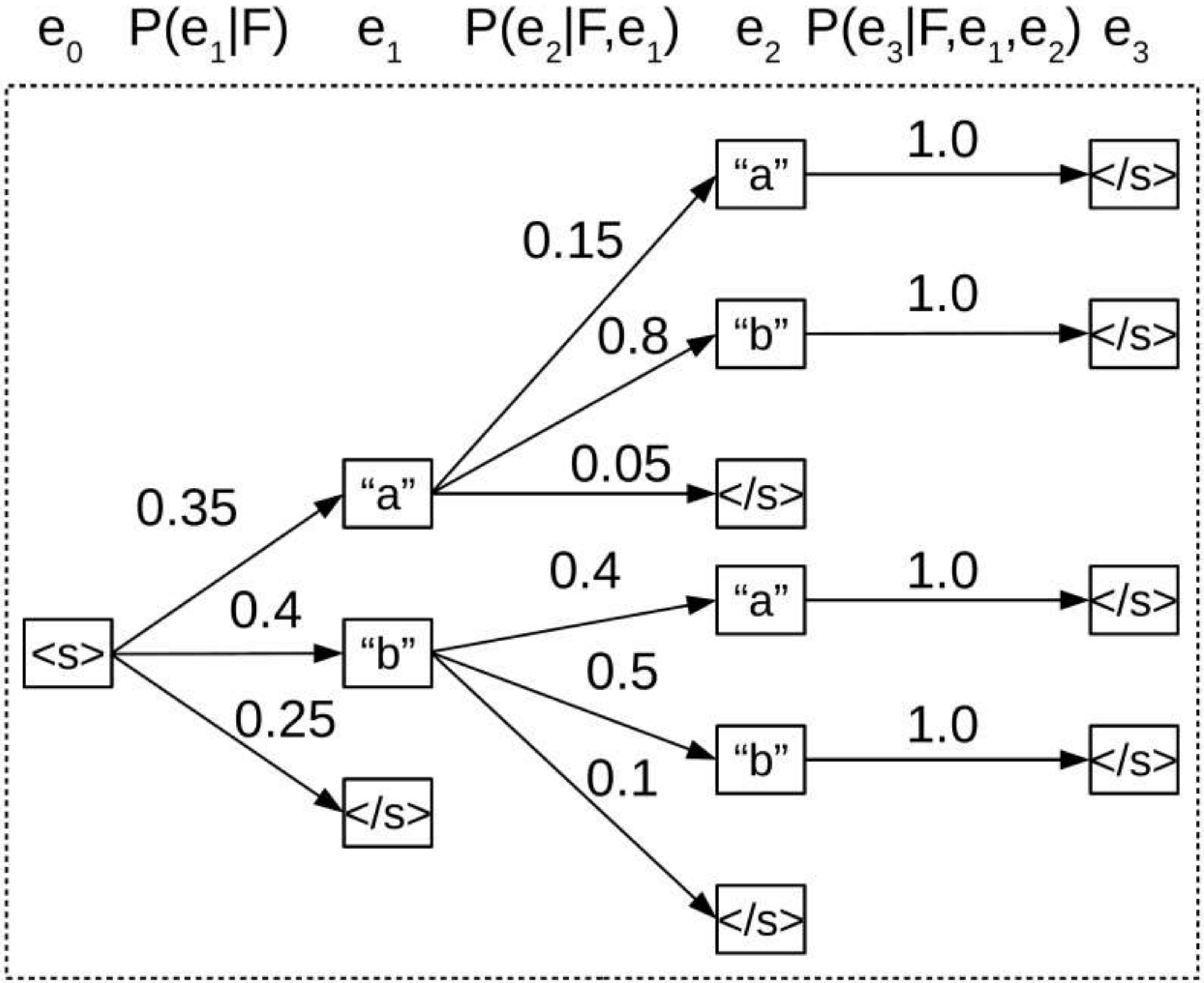
```
while y_{j-1} != "</s>":
    y_j ~ P(y_j | X, y_1, ..., y_{j-1})
```

Greedy search

- One by one, pick single highest probability word

```
while y_{j-1} != "</s>":
  y_j = argmax P(y_j | X, y_1, ..., y_{j-1})
```

- Problems
  - Often generates easy words first
  - Often prefers multiple common words to rare words



Example by Graham Neubig