

# Introduction to Neural Machine Translation (2/3)

Marine Carpuat

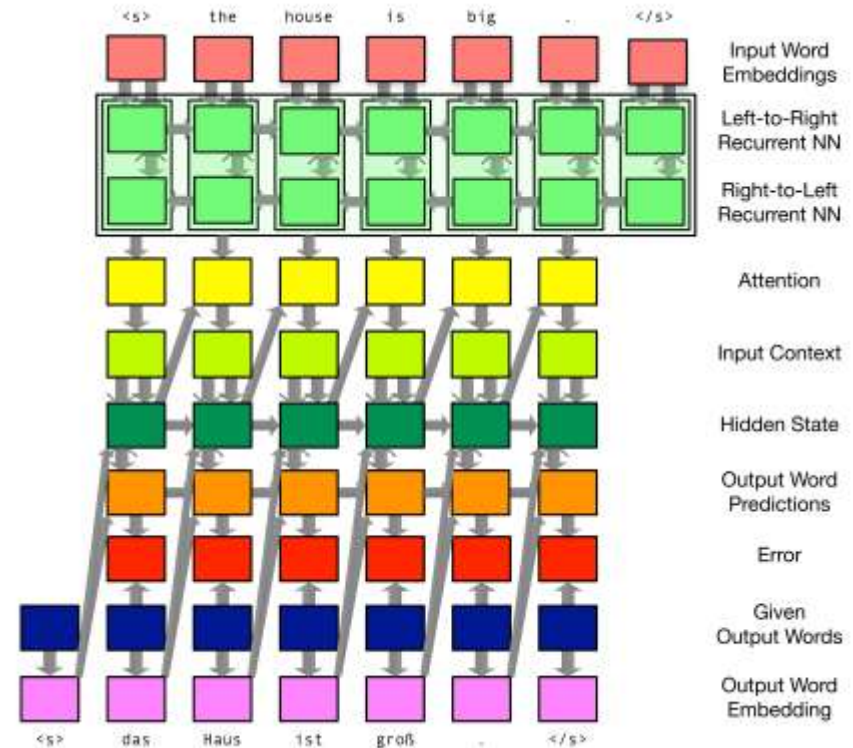
Computer Science

University of Maryland

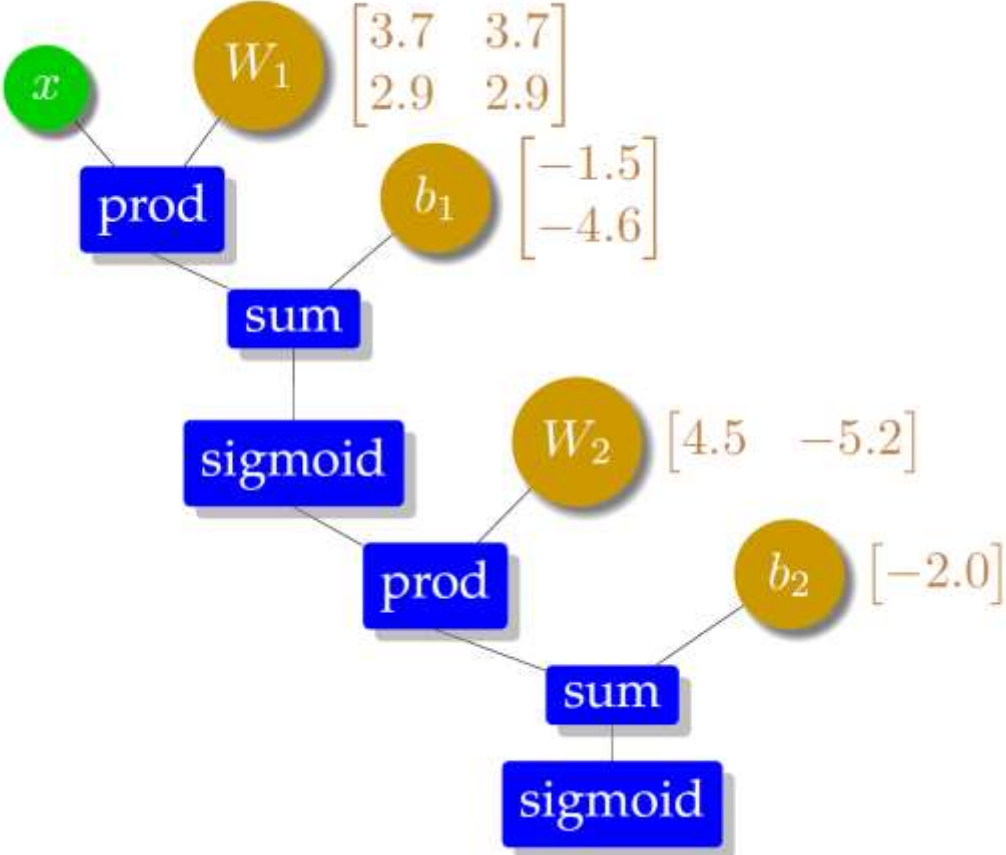
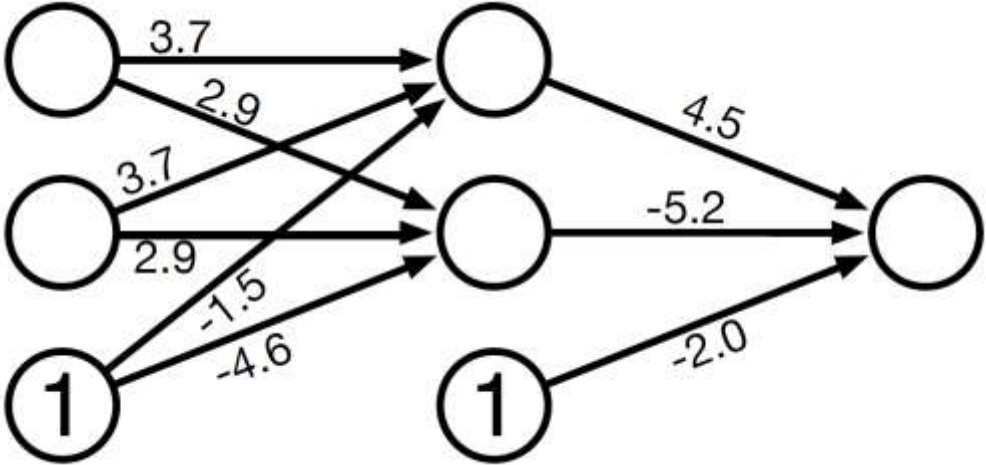


# Roadmap

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

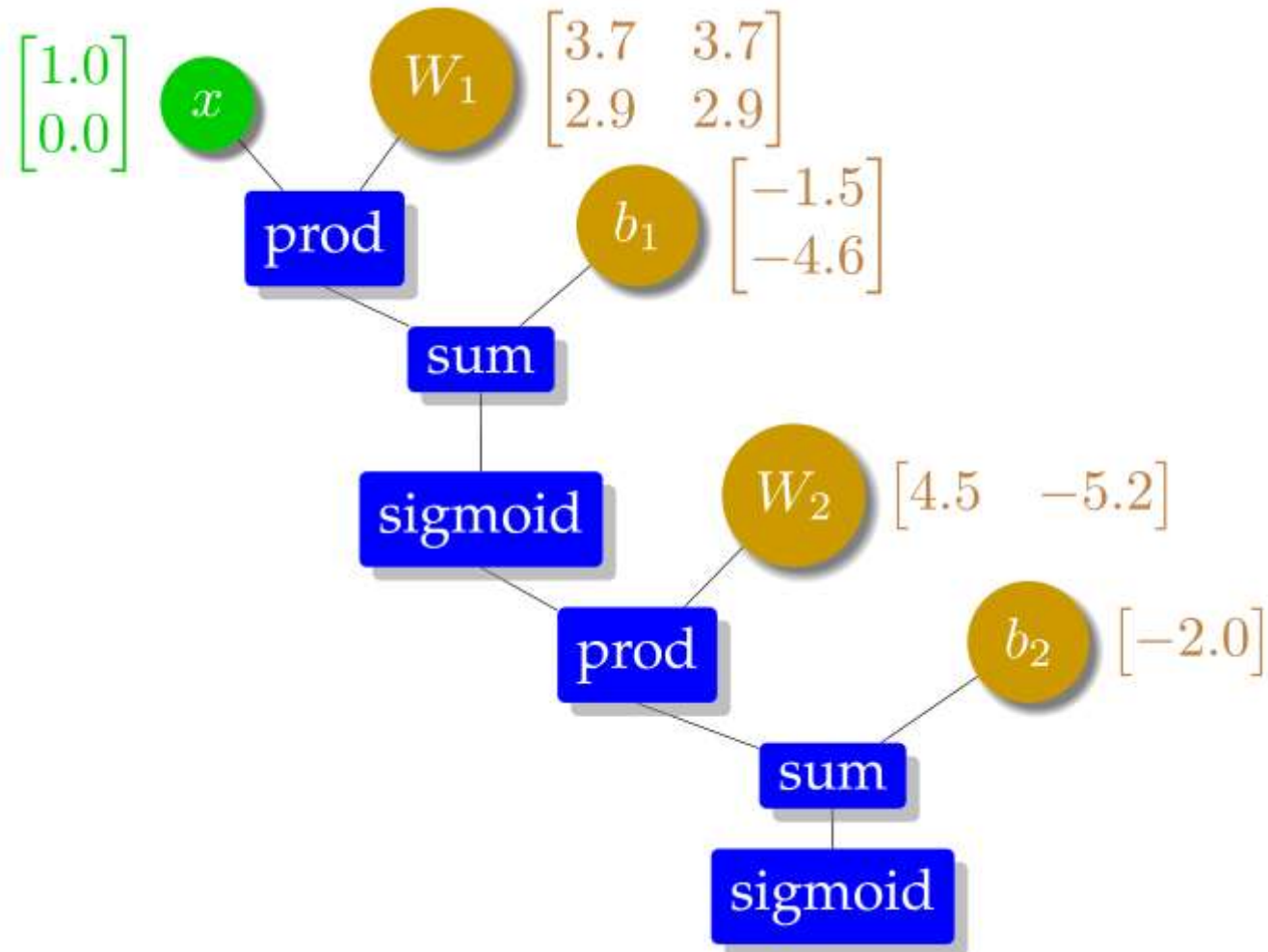


# Neural Networks as Computation Graphs

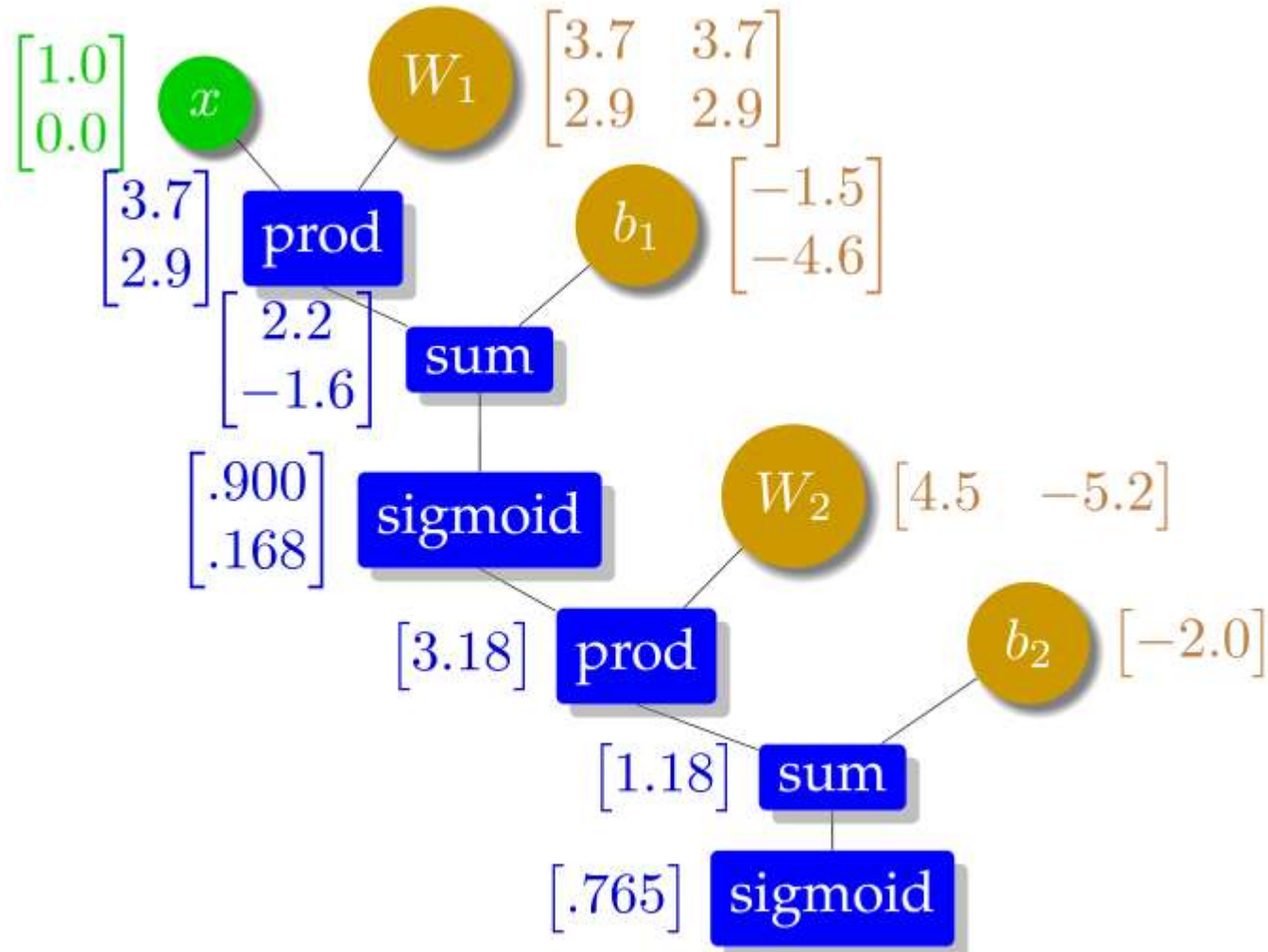


Example & figures by Philipp Koehn

# Computation Graphs Make Prediction Easy: Forward Propagation



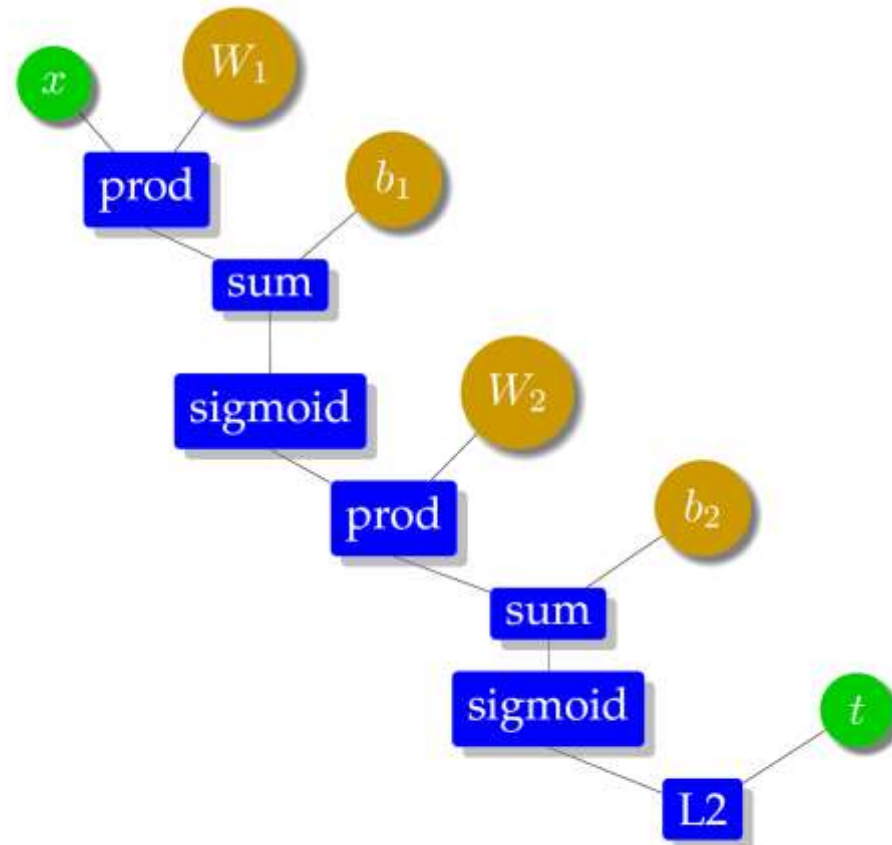
# Computation Graphs Make Prediction Easy: Forward Propagation



# Stochastic Gradient Descent

- Start with some initial parameter values
- $w = 0$
- for / iterations
  - Go through the training data one example at a time
  - for each labeled pair  $x, y$  in the data
$$w = w - \mu \frac{d\text{error}(w, x, y)}{dw}$$
    - Take a step down the gradient

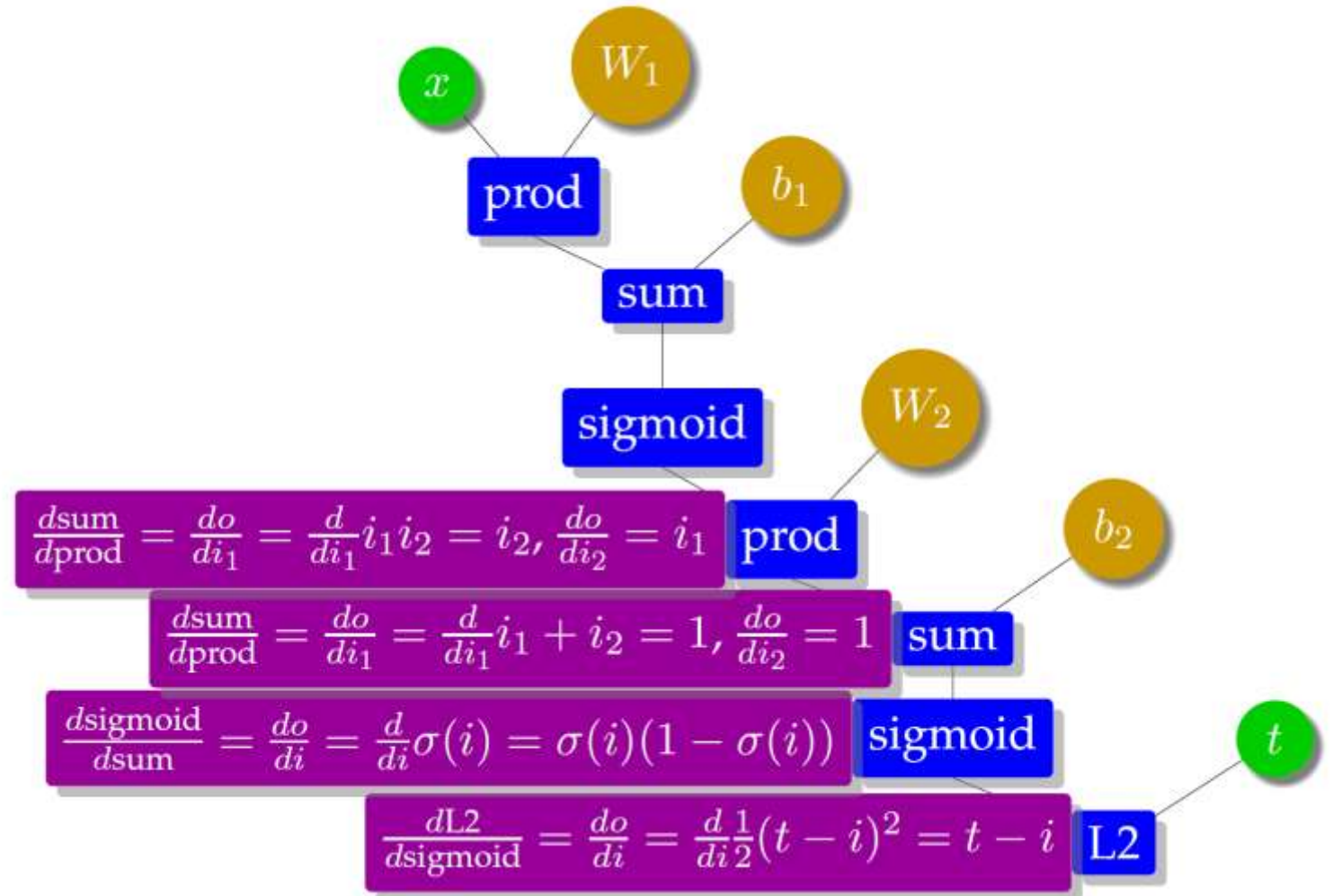
# Computation Graphs Make Training Easy: Computing Error



# Computation Graphs Make Training Easy: Computing Gradients

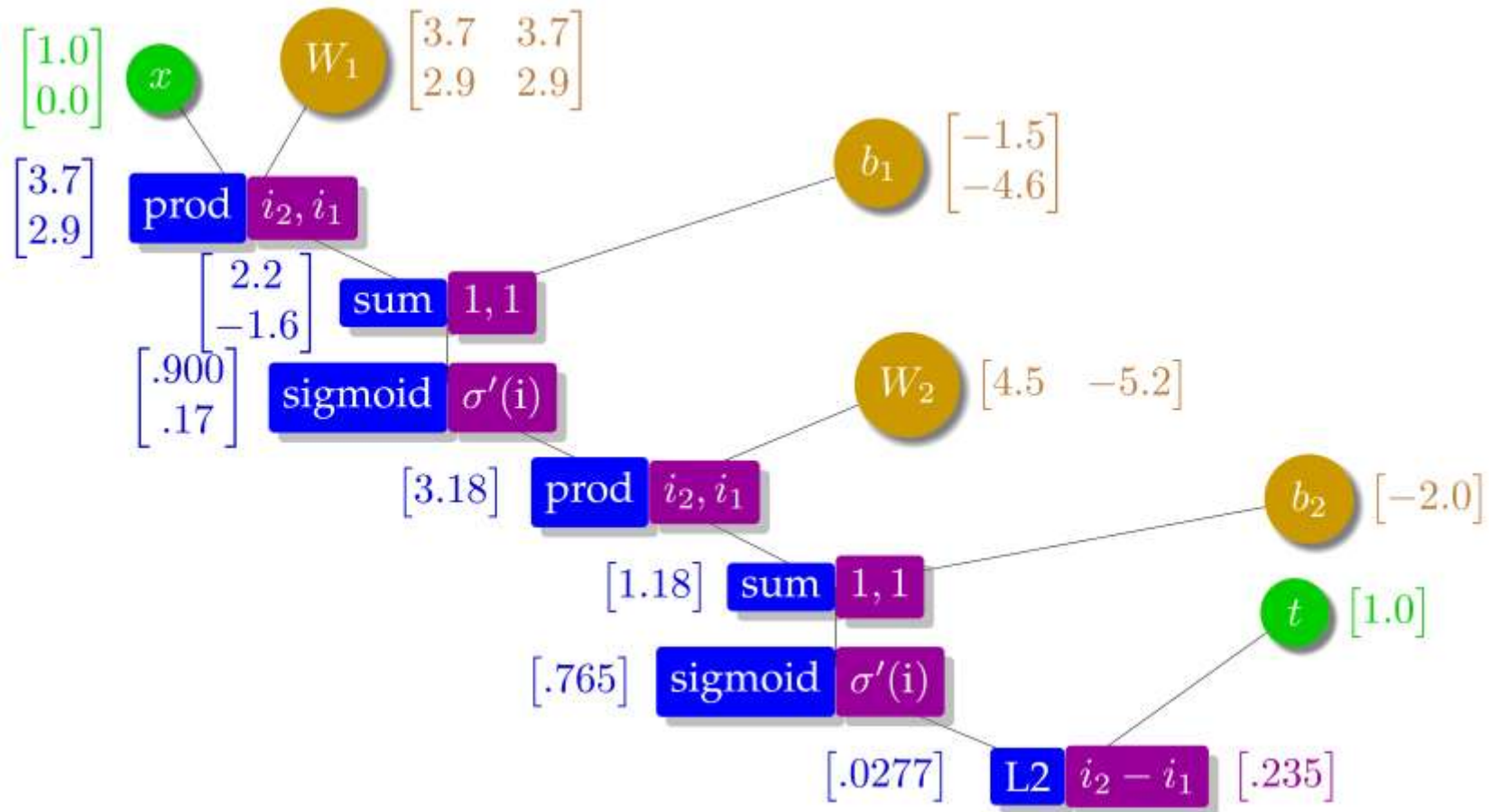


$$\frac{dE}{dA} = \frac{dE}{dB} \frac{dB}{dA}$$

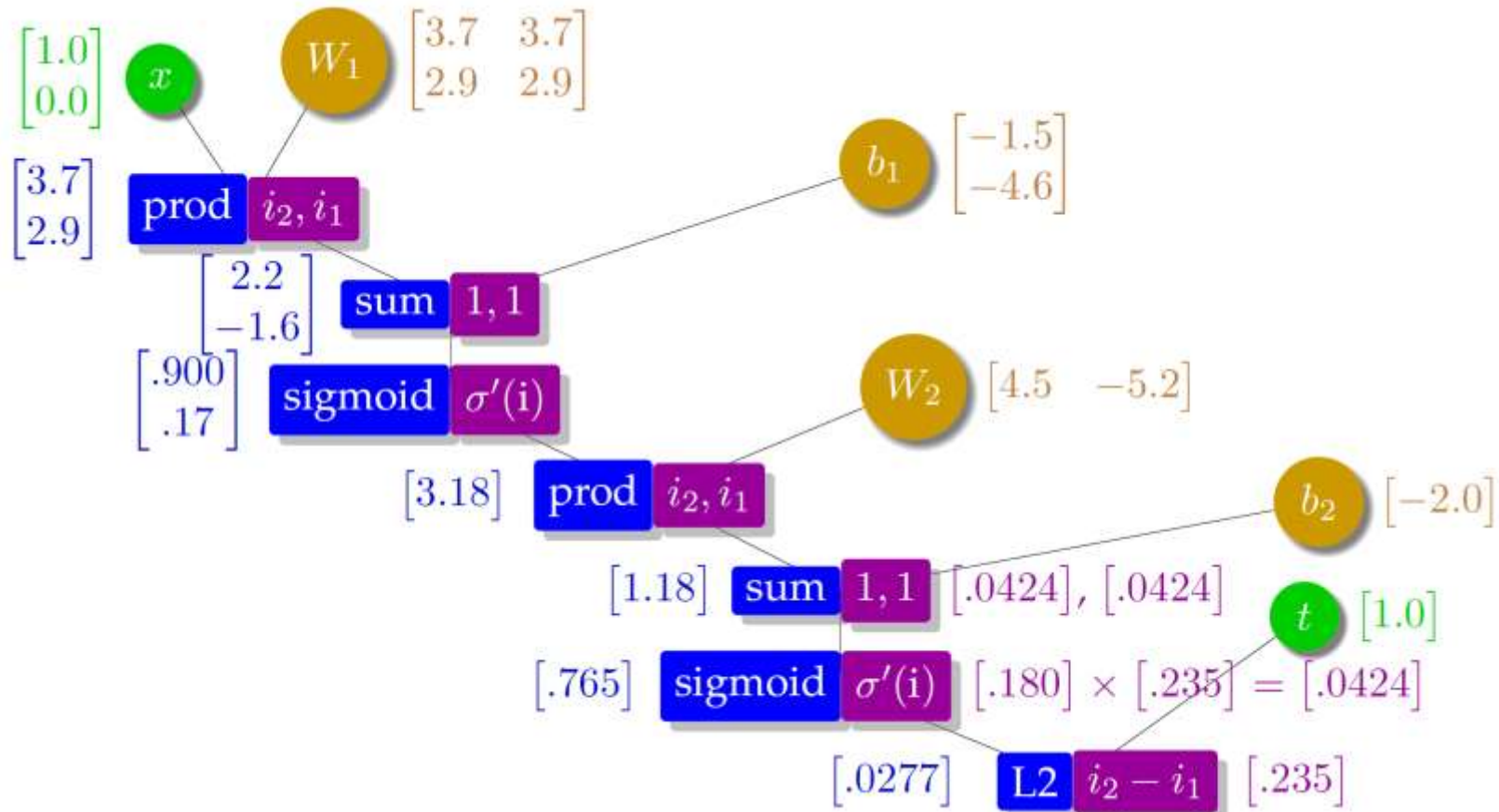




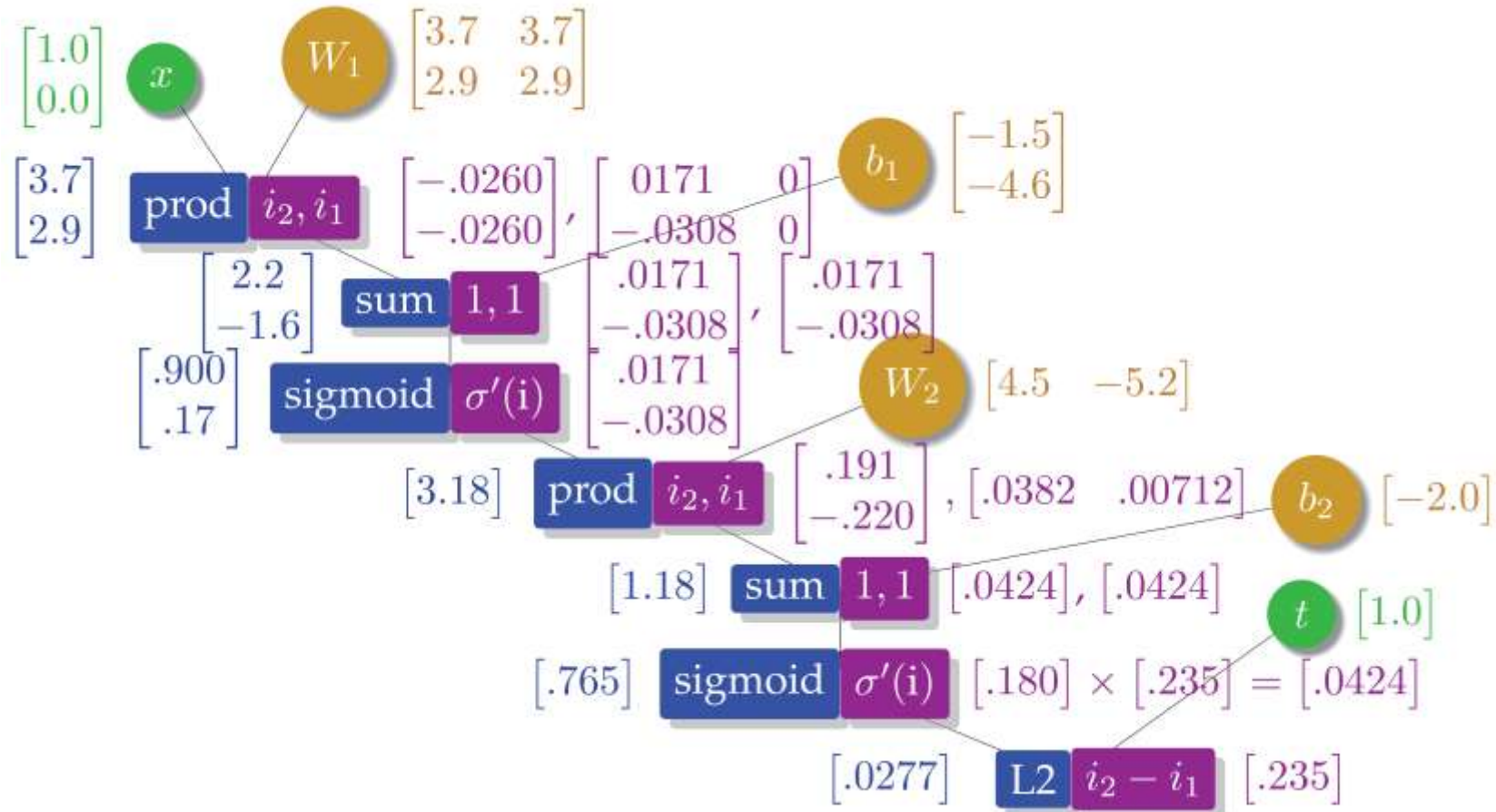
Computation Graphs Make Training Easy:  
 Given forward pass + derivatives for each node



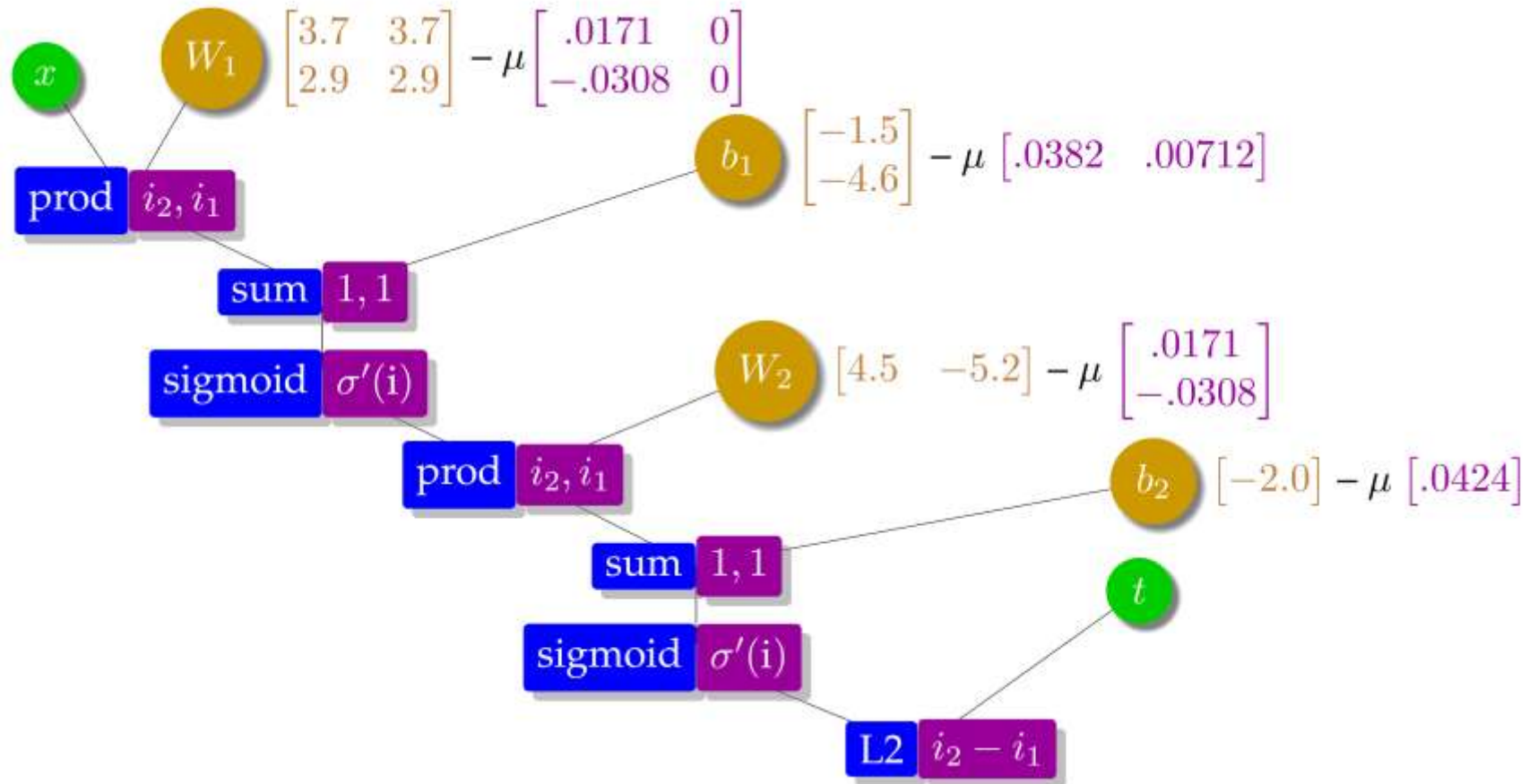
# Computation Graphs Make Training Easy: Computing Gradients



# Computation Graphs Make Training Easy: Computing Gradients



# Computation Graphs Make Training Easy: Updating Parameters

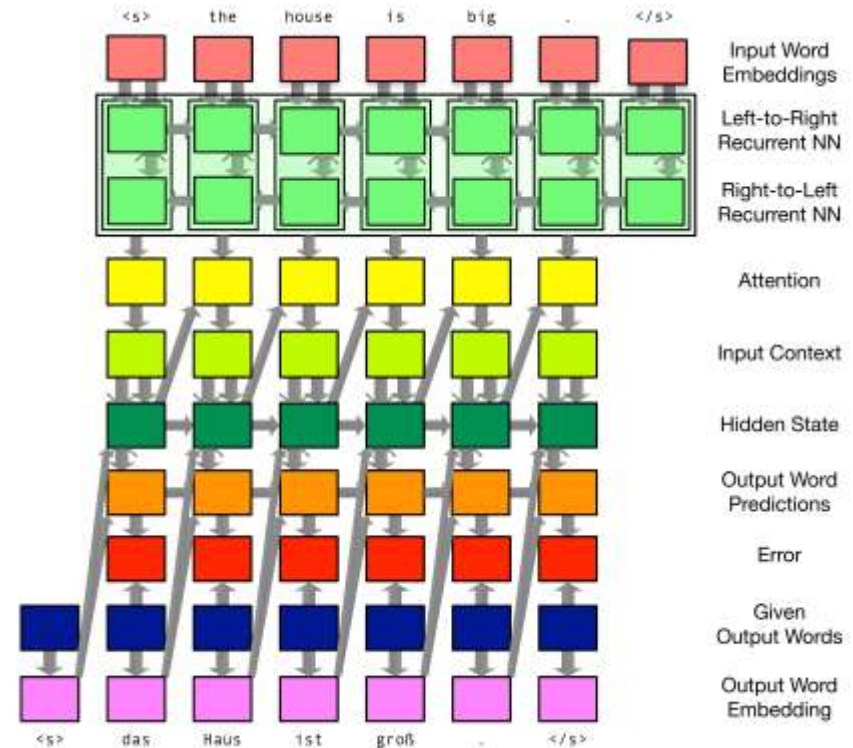


# Computation Graph: A Powerful Abstraction

- 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

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

- Related task: probability of an upcoming word

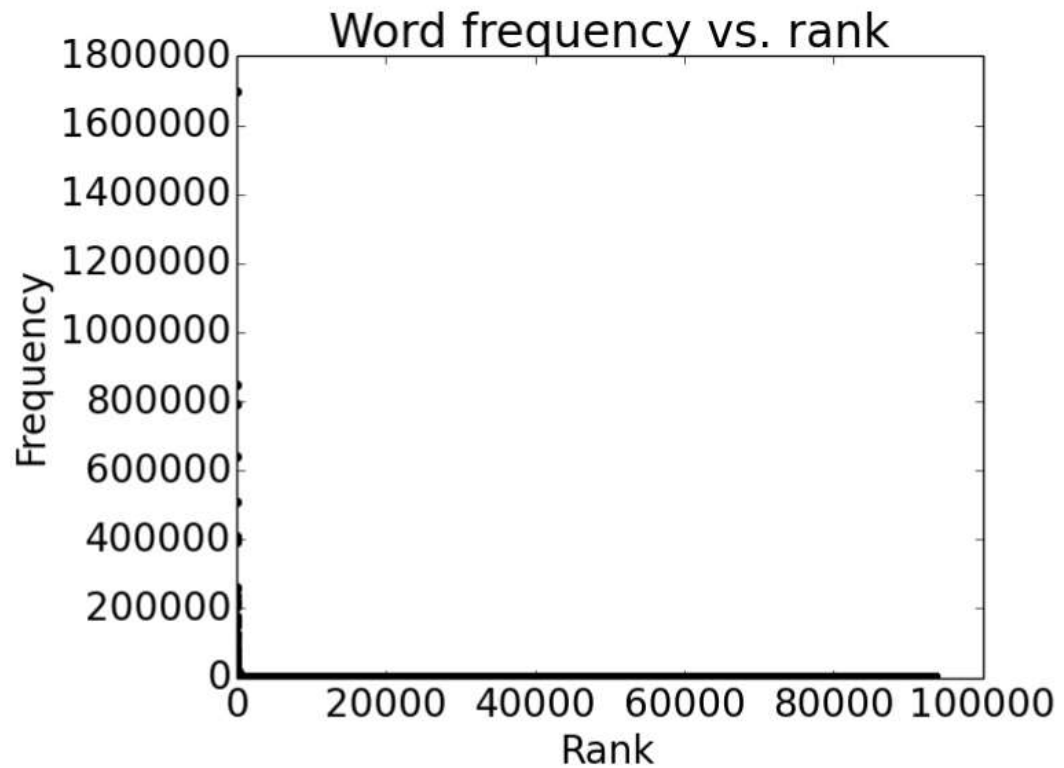
$$P(e_5 | e_1, e_2, e_3, e_4)$$

- A model that computes either of these:

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

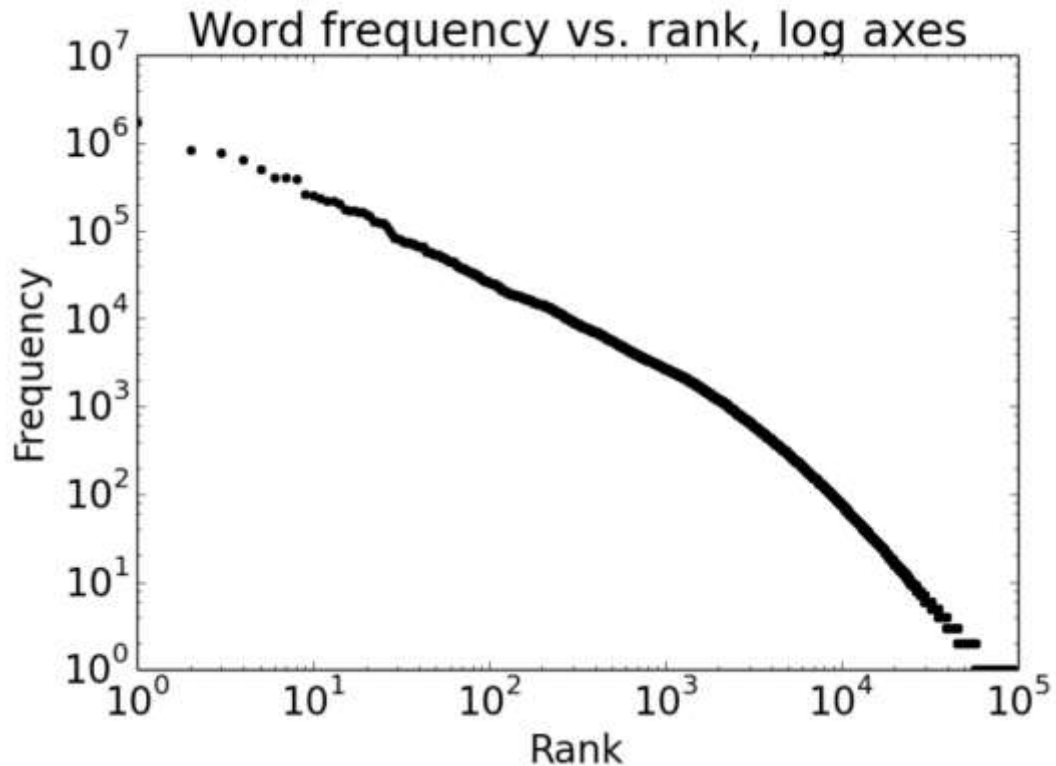
is called a **language model**.

# Zipf's Law



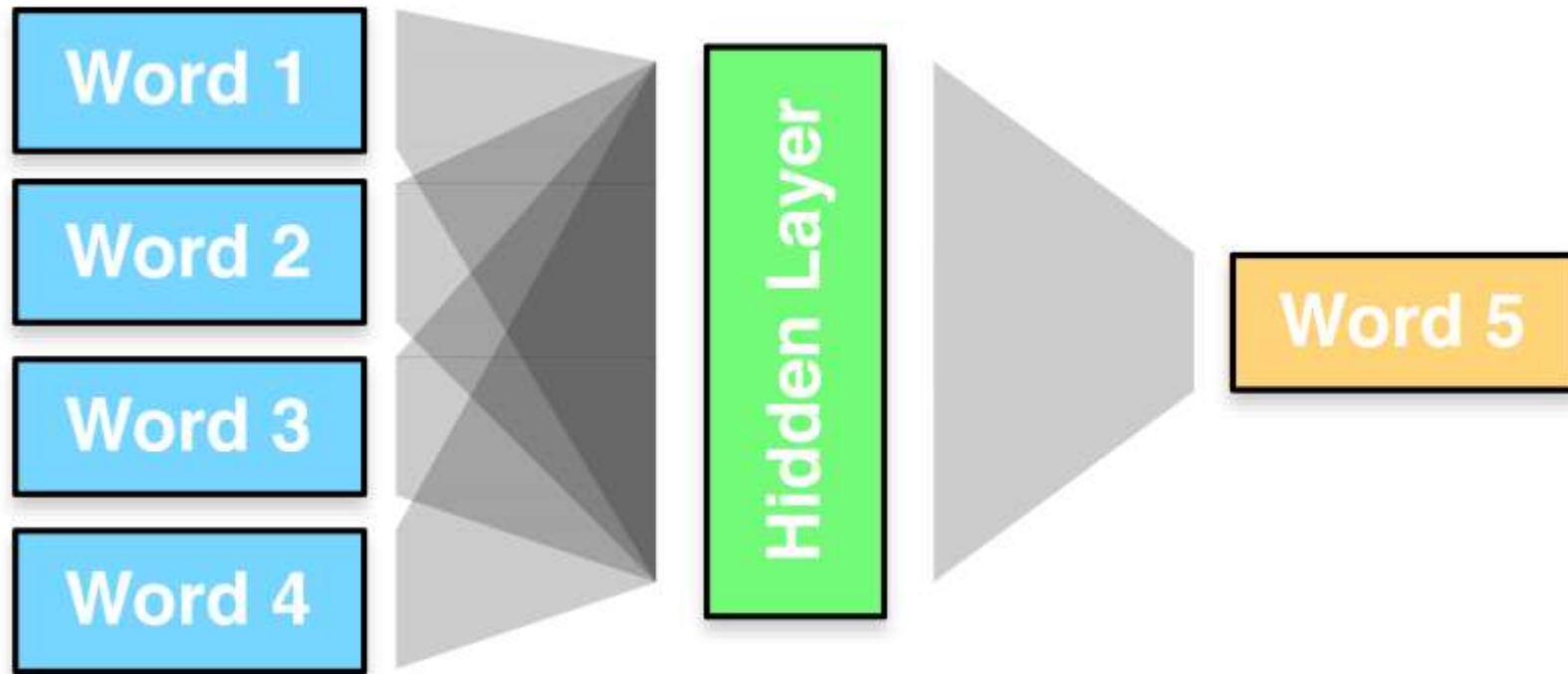


# Zipf's Law



- 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



# Representing Words

- “one hot vector”

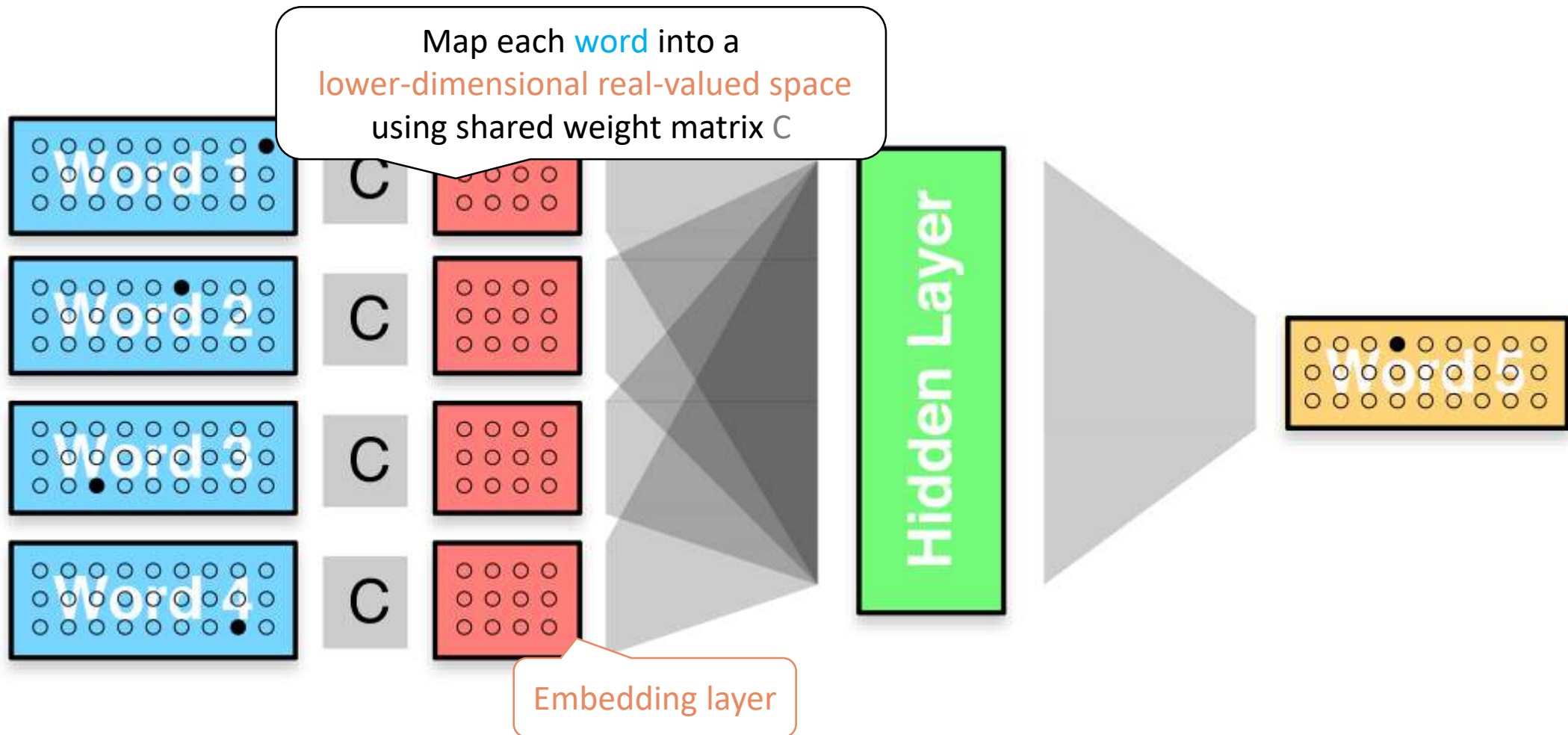
dog = [ 0, 0, 0, 0, 1, 0, 0, 0 ...]

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



# 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

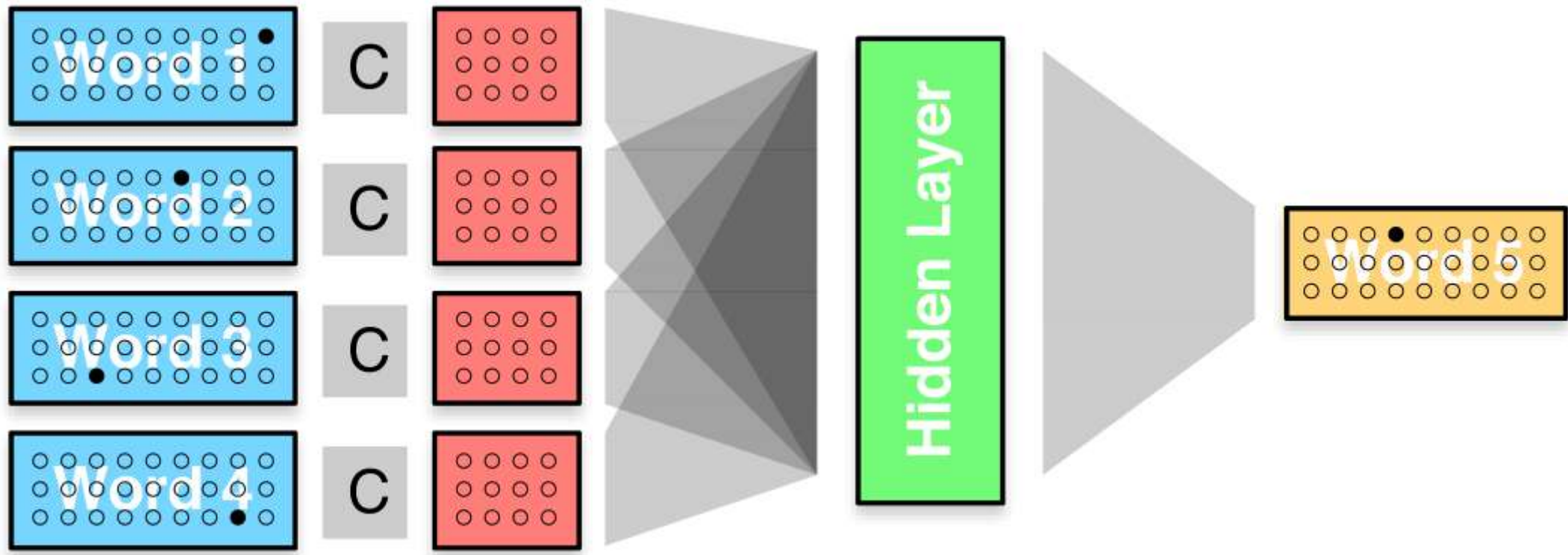
$$p = \text{softmax}(s)$$

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

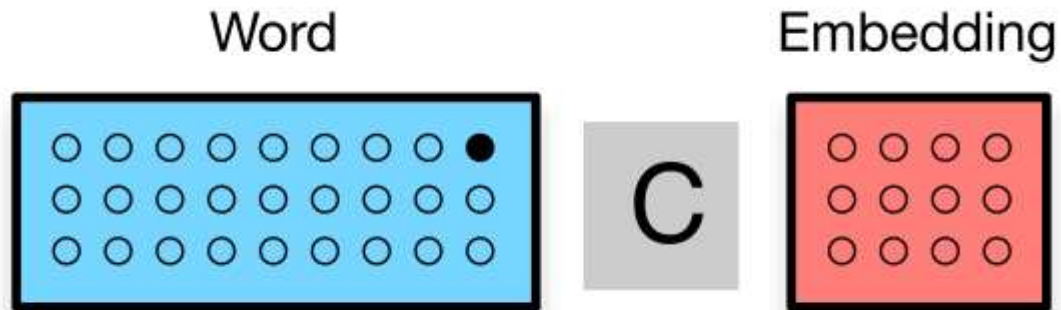
# Estimating Model Parameters

- 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  $\text{error}(\lambda) = -\sum_{E \text{ in corpus}} \log P_{\lambda}(E)$
  - At the word level  $\text{error}(\lambda) = -\log P_{\lambda}(e_t | e_1 \dots e_{t-1})$

# Language Modeling with Feedforward Neural Networks



# Word Embeddings: a useful by-product of neural LMs

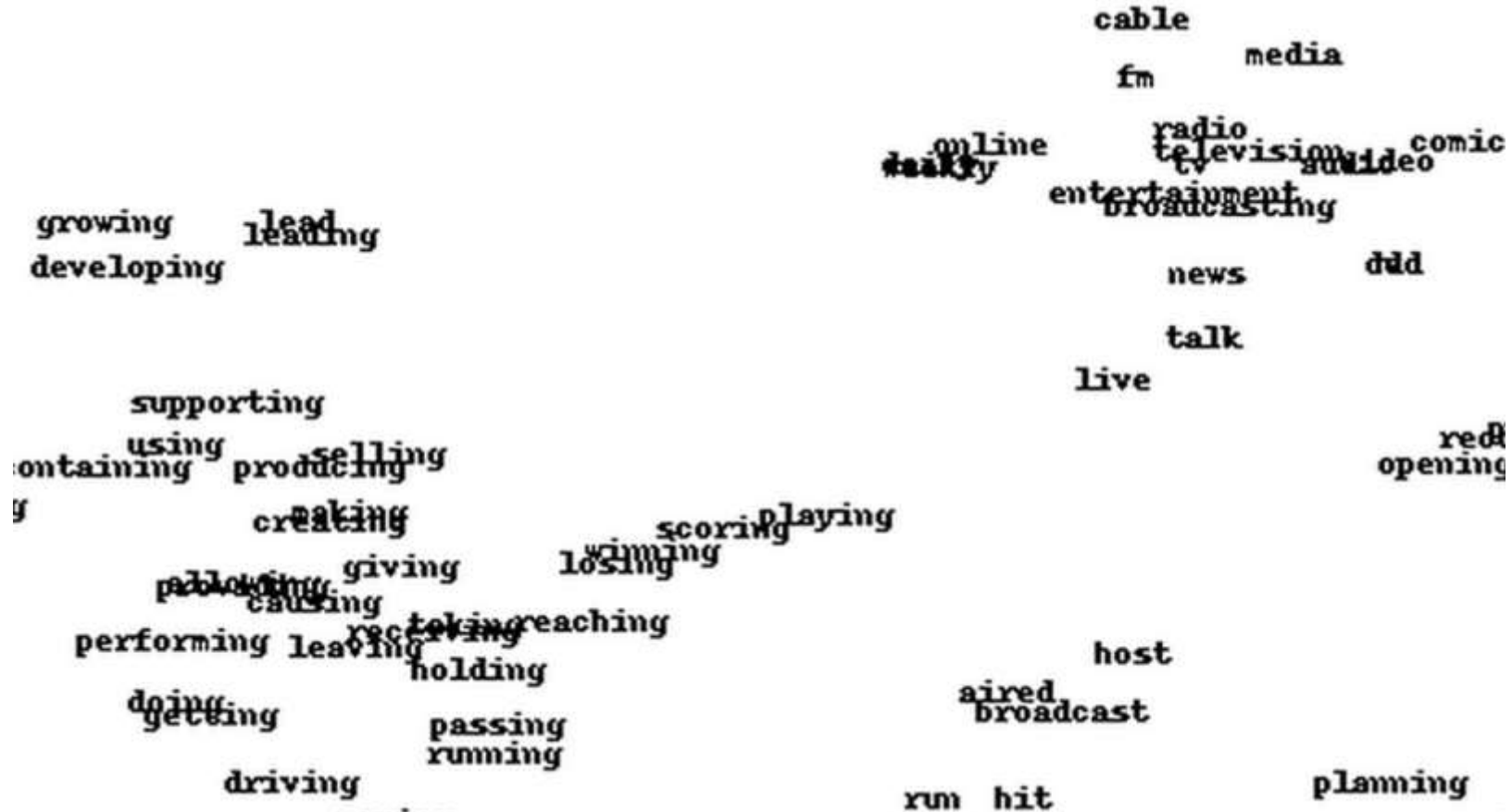


- Words that occurs in similar contexts tend to have similar embeddings
- Embeddings capture many usage regularities
- Useful features for many NLP tasks





# Word Embeddings

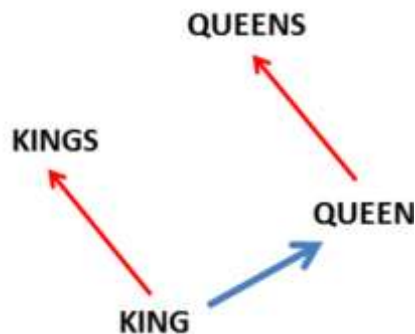


# Word Embeddings Capture Useful Regularities

## Morpho-Syntactic

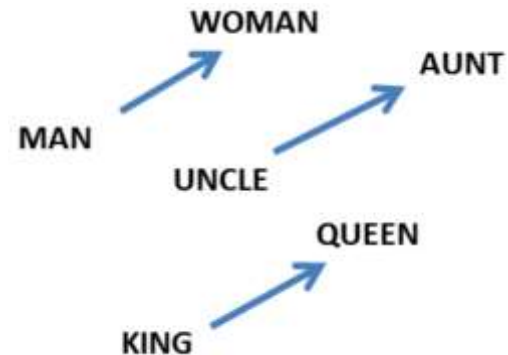
- Adjectives: base form vs. comparative
- Nouns: singular vs. plural
- Verbs: present tense vs. past tense

[Mikolov et al. 2013]

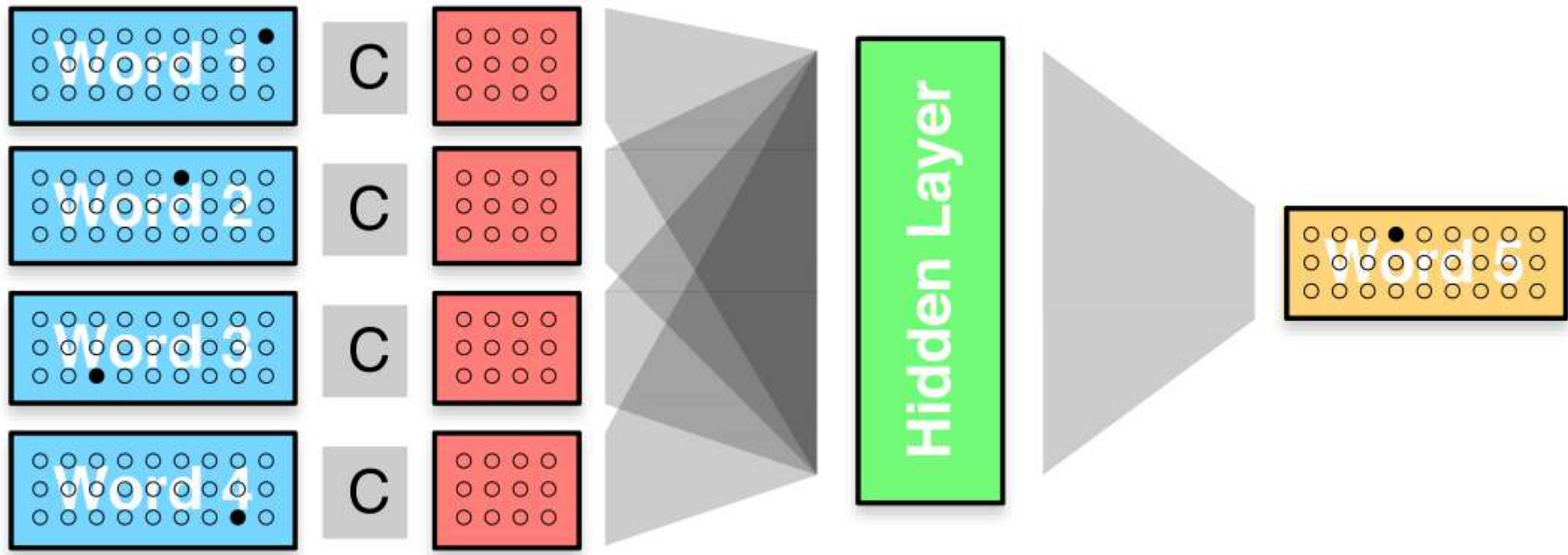


## Semantic

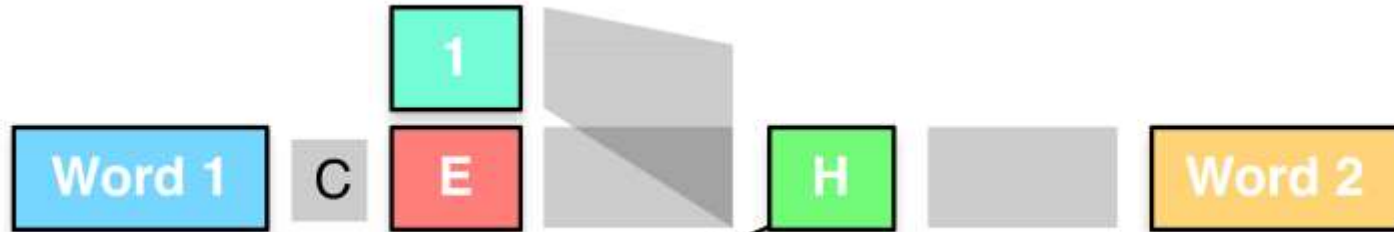
- Word similarity/relatedness
- Semantic relations
- But tends to fail at distinguishing
  - Synonyms vs. antonyms
  - Multiple senses of a word



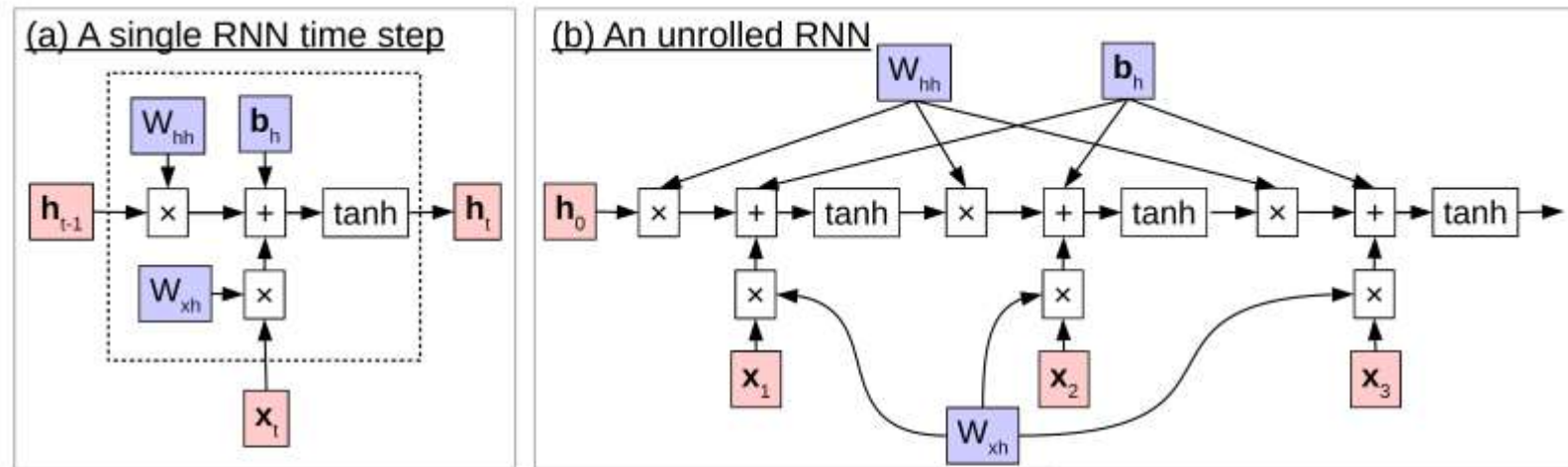
# Language Modeling with Feedforward Neural Networks



# Language Modeling with Recurrent Neural Networks



# Formalizing our Recurrent Language Model

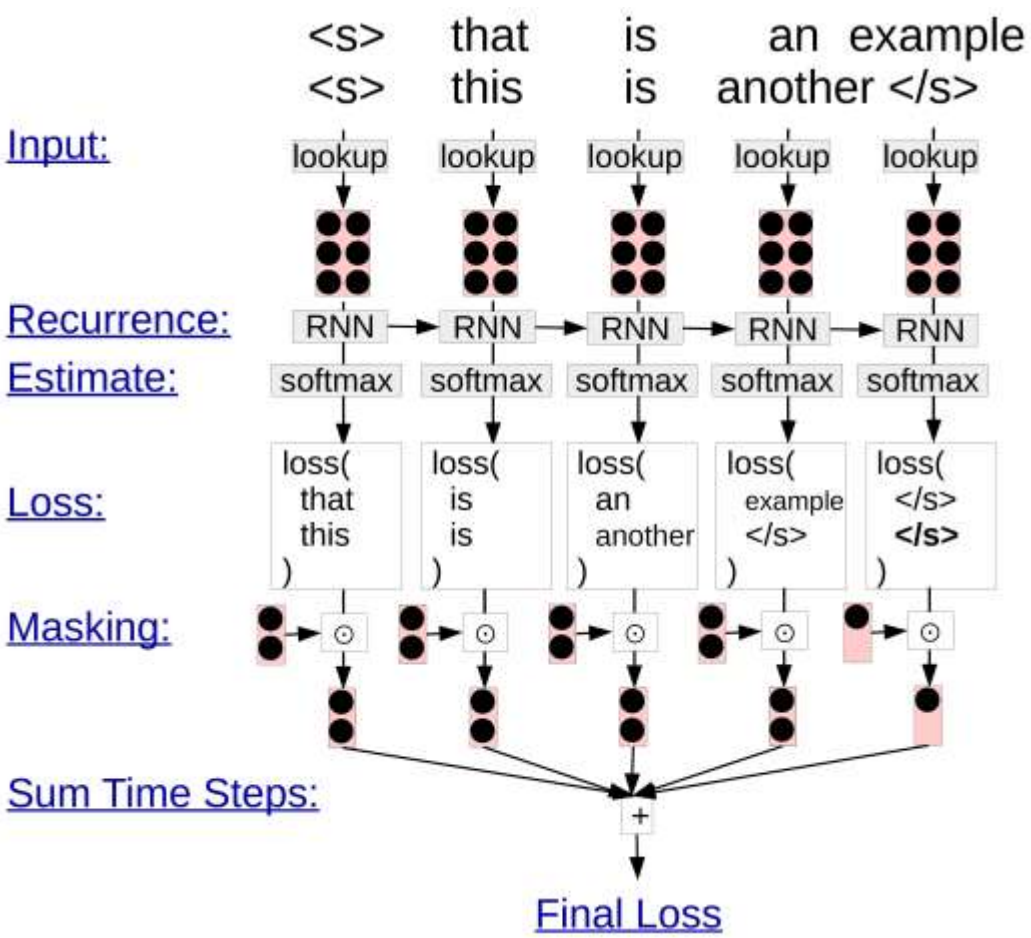


$$\mathbf{m}_t = M_{\cdot, e_{t-1}}$$

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

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

# 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?

Cell sensitive to position in line:

The sole importance of the crossing of the Berezina lies in the fact that it plainly and indubitably proved the fallacy of all the plans for cutting off the enemy's retreat and the soundness of the only possible line of action--the one Kutuzov and the general mass of the army demanded--namely, simply to follow the enemy up. The French crowd fled at a continually increasing speed and all its energy was directed to reaching its goal. It fled like a wounded animal and it was impossible to block its path. This was shown not so much by the arrangements it made for crossing as by what took place at the bridges. When the bridges broke down, unarmed soldiers, people from Moscow and women with children who were with the French transport, all--carried on by vis inertiae--pressed forward into boats and into the ice-covered water and did not, surrender.

Cell that turns on inside quotes:

"You mean to imply that I have nothing to eat out of... On the contrary, I can supply you with everything even if you want to give dinner parties," warmly replied Chichagov, who tried by every word he spoke to prove his own rectitude and therefore imagined Kutuzov to be animated by the same desire.

Kutuzov, shrugging his shoulders, replied with his subtle penetrating smile: "I meant merely to say what I said."



# What do Recurrent Language Models Learn?

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);
    /* Keep currently invalid fields around in case they
     * become valid after a policy reload. */
    if (ret == -EINVAL) {
        pr_warn("audit rule for LSM '%s' is invalid\n",
                df->lsm_str);
        ret = 0;
    }
    return ret;
}
```

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;
}
```

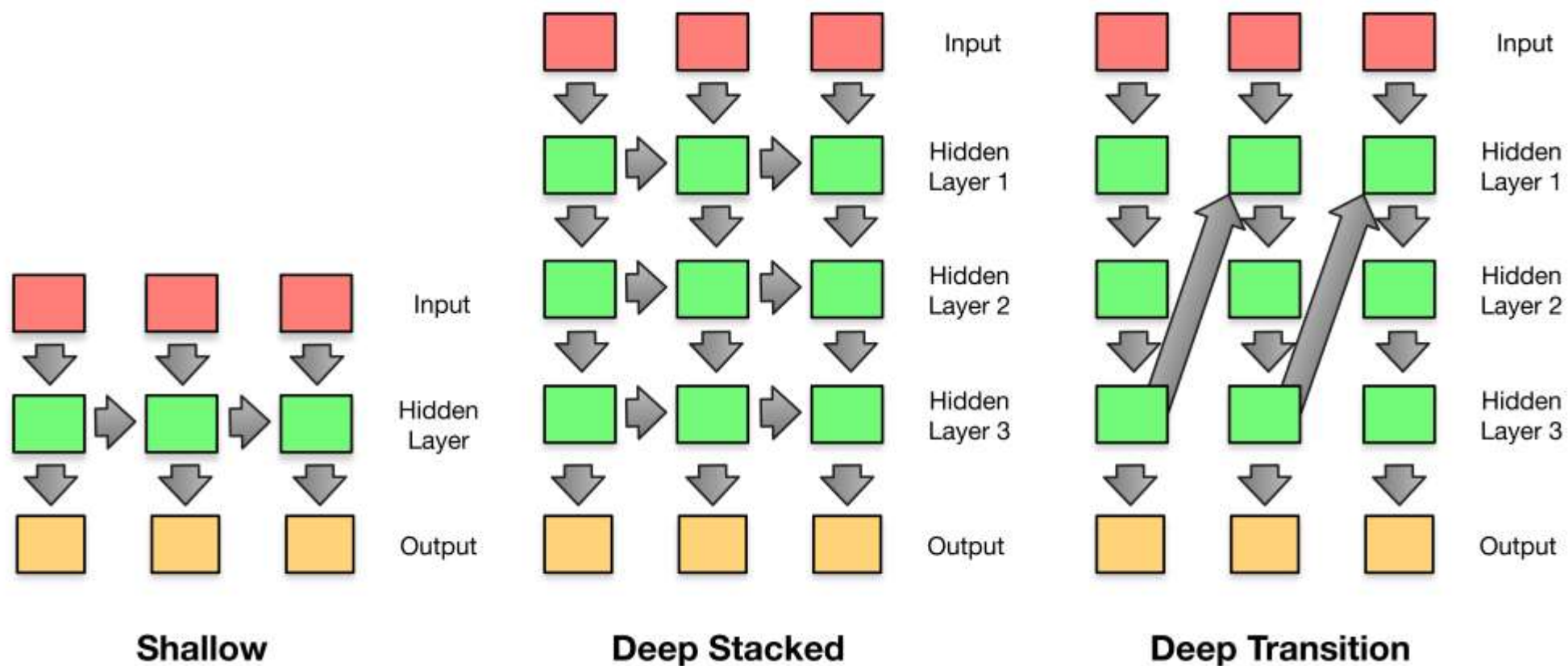
# What do Recurrent Language Models Learn?

- 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

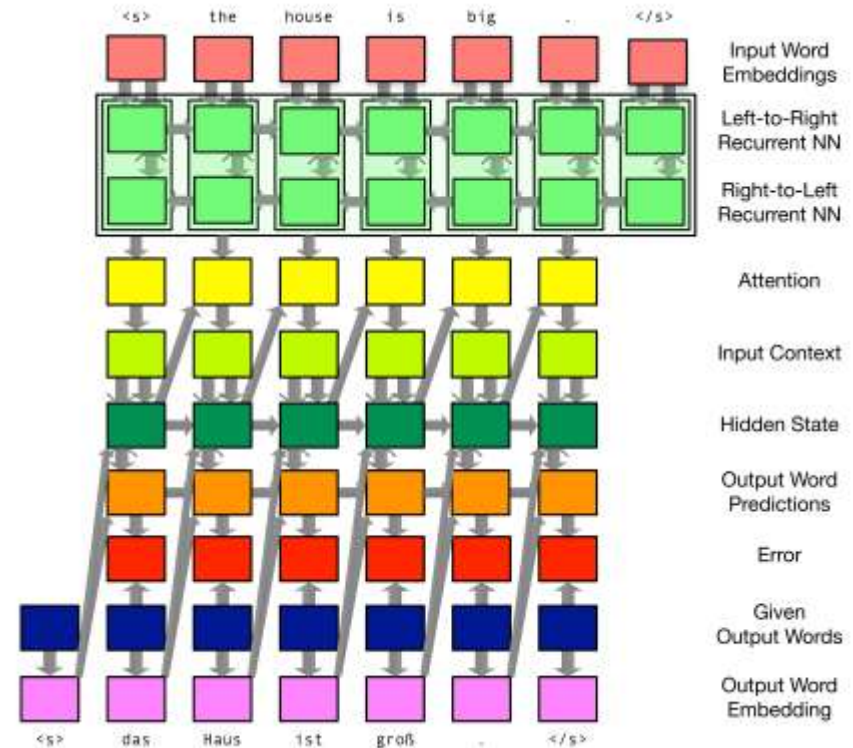


# Recurrent Neural Language Models Summary

- 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

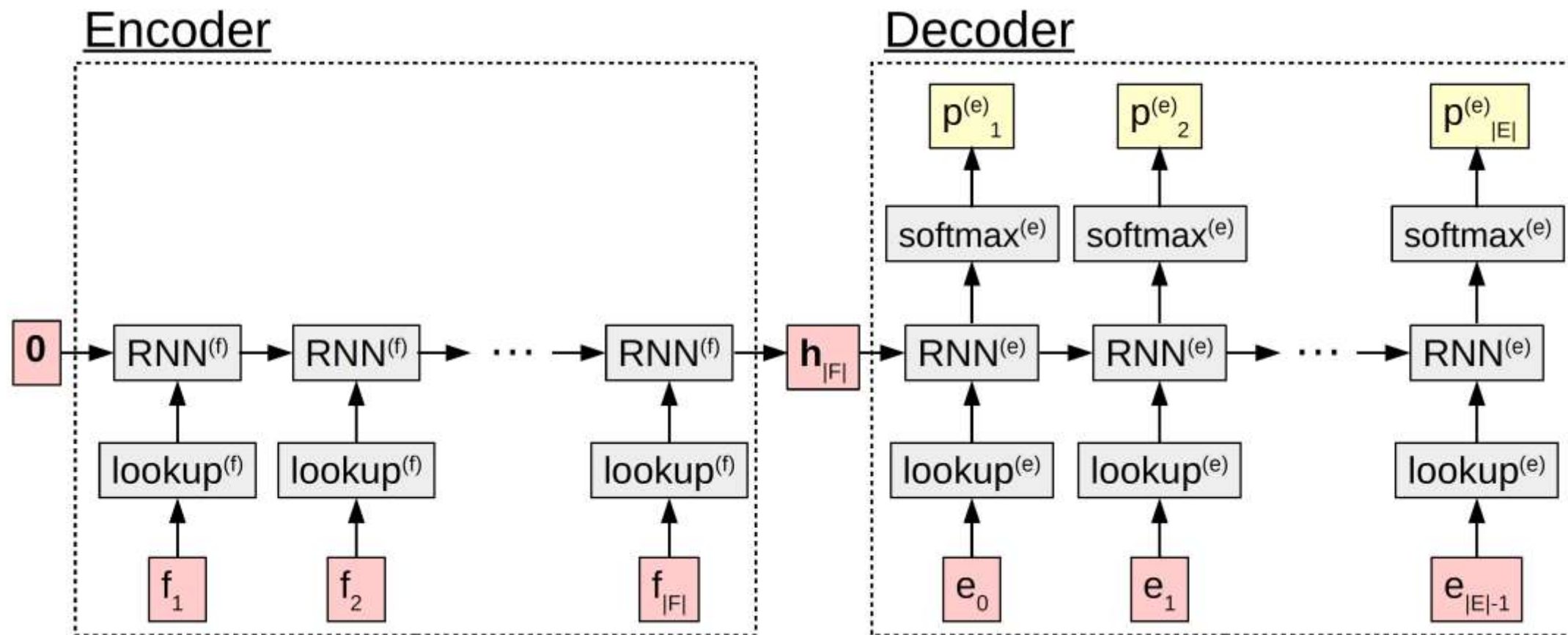
- Evaluating machine translation
- Introduction to neural networks
- Modeling sequences of words with neural language models
- Translating with encoder-decoder models
- Attention mechanism



# 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)

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

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

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

# Ancestral Sampling

- Randomly generate words one by one
- Until end of sentence symbol
- Done!

```
while  $y_{j-1} \neq \text{"</s>"}$ :  
   $y_j \sim P(y_j \mid X, y_1, \dots, y_{j-1})$ 
```

# Greedy search

- One by one, pick single highest probability word
- Problems
  - Often generates easy words first
  - Often prefers multiple common words to rare words

```
while  $y_{j-1} \neq \text{"</s>"}$ :  
   $y_j = \operatorname{argmax} P(y_j \mid X, y_1, \dots, y_{j-1})$ 
```

