COMPUTER SCIENCE
UNIVERSITY OF MARYLAND

Language Models (2)

CMSC 470

Marine Carpuat

Slides credit: Jurasky & Martin

Roadmap

- Language Models
  - Our first example of modeling sequences

- n-gram language models
- How to estimate them?
- How to evaluate them?
- Neural models

Pros and cons of n-gram models

- Really easy to build, can train on billions and billions of words
- Smoothing helps generalize to new data
- Only work well for word prediction if the test corpus looks like the training corpus
- Only capture short distance context

Evaluation:
How good is our model?

- Does our language model prefer good sentences to bad ones?
  - Assign higher probability to “real” or “frequently observed” sentences
    - Than “ungrammatical” or “rarely observed” sentences?

- Extrinsic vs intrinsic evaluation

Intrinsic evaluation: intuition

- The Shannon Game:
  - How well can we predict the next word?

I always order pizza with cheese and ____

The 33rd President of the US was ____

I saw a ____

mushrooms 0.1

pepperoni 0.1

anchovies 0.01

….

fried rice 0.0001

….

and 1e-100

  - Unigrams are terrible at this game. (Why?)
- A better model of a text assigns a higher probability to the word that actually occurs

Intrinsic evaluation metric: perplexity

The best language model is one that best predicts an unseen test set

- Gives the highest P(sentence)

Perplexity is the inverse probability of the test set, normalized by the number of words:

$$PP(W) = P(w_1 w_2 \dots w_N)^{-\frac{1}{N}}$$

$$= \sqrt[N]{\frac{1}{P(w_1 w_2 \dots w_N)}}$$

Chain rule:

$$\mathrm{PP}(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i|w_1 \dots w_{i-1})}}$$

For bigrams:

$$\mathrm{PP}(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i|w_{i-1})}}$$

Minimizing perplexity is the same as maximizing probability

Perplexity as branching factor

- Let's suppose a sentence consisting of random digits
- What is the perplexity of this sentence according to a model that assign P=1/10 to each digit?

$$
\begin{aligned}
\mathrm{PP}(W) &= P(w_1 w_2 \ldots w_N)^{-\frac{1}{N}} \\
&= \left(\frac{1}{10}^{N}\right)^{-\frac{1}{N}} \\
&= \frac{1}{10}^{-1} \\
&= 10
\end{aligned}
$$

Lower perplexity = better model

- Training 38 million words, test 1.5 million words, WSJ

N-gram Order	Unigram	Bigram	Trigram
Perplexity	962	170	109

The perils of overfitting

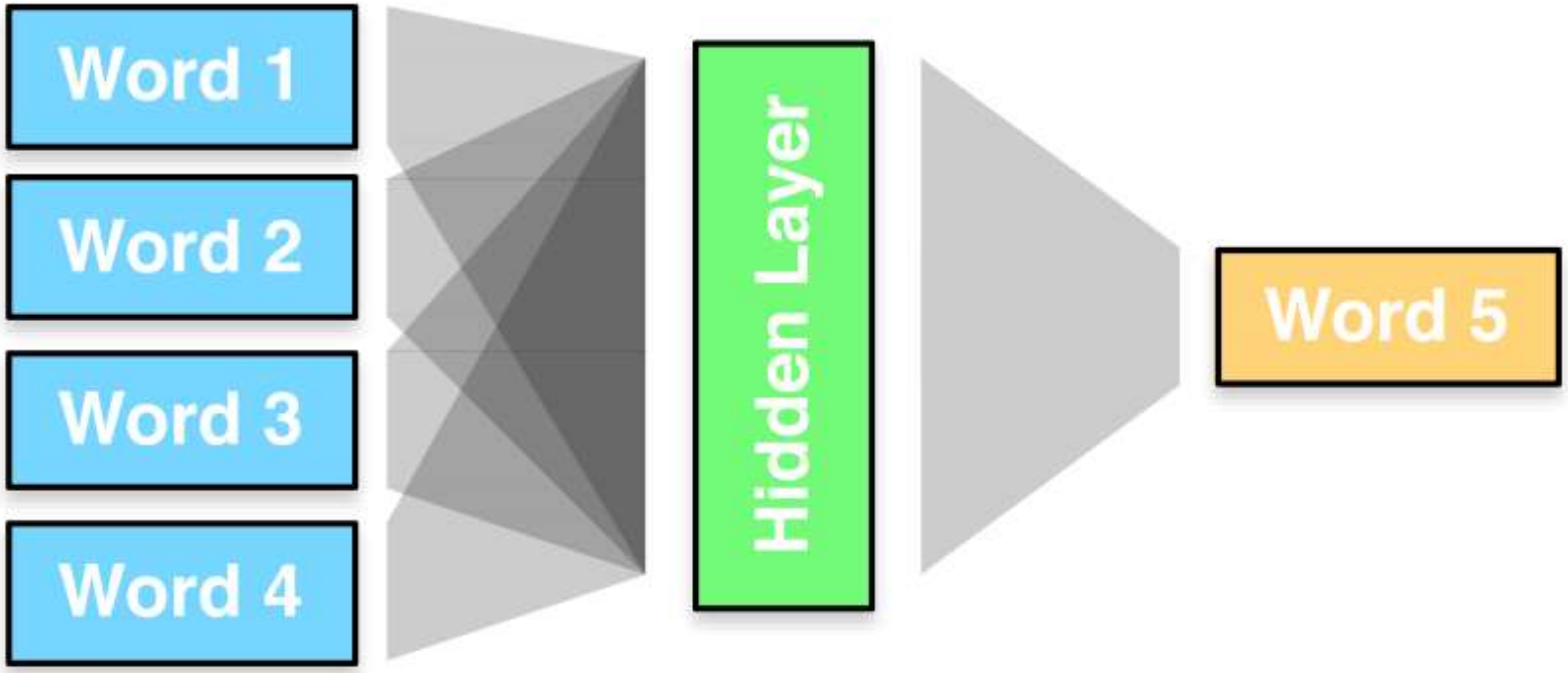
- N-grams only work well for word prediction if the test corpus looks like the training corpus
- In real life, it often doesn’t!
- We need to train robust models that generalize
    - Smoothing is important
    - Choose n carefully

Roadmap

- Language Models
  - Our first example of modeling sequences

- n-gram language models
- How to estimate them?
- How to evaluate them?
- Neural models

Toward a Neural Language Model



Figures by Philipp Koehn (JHU)

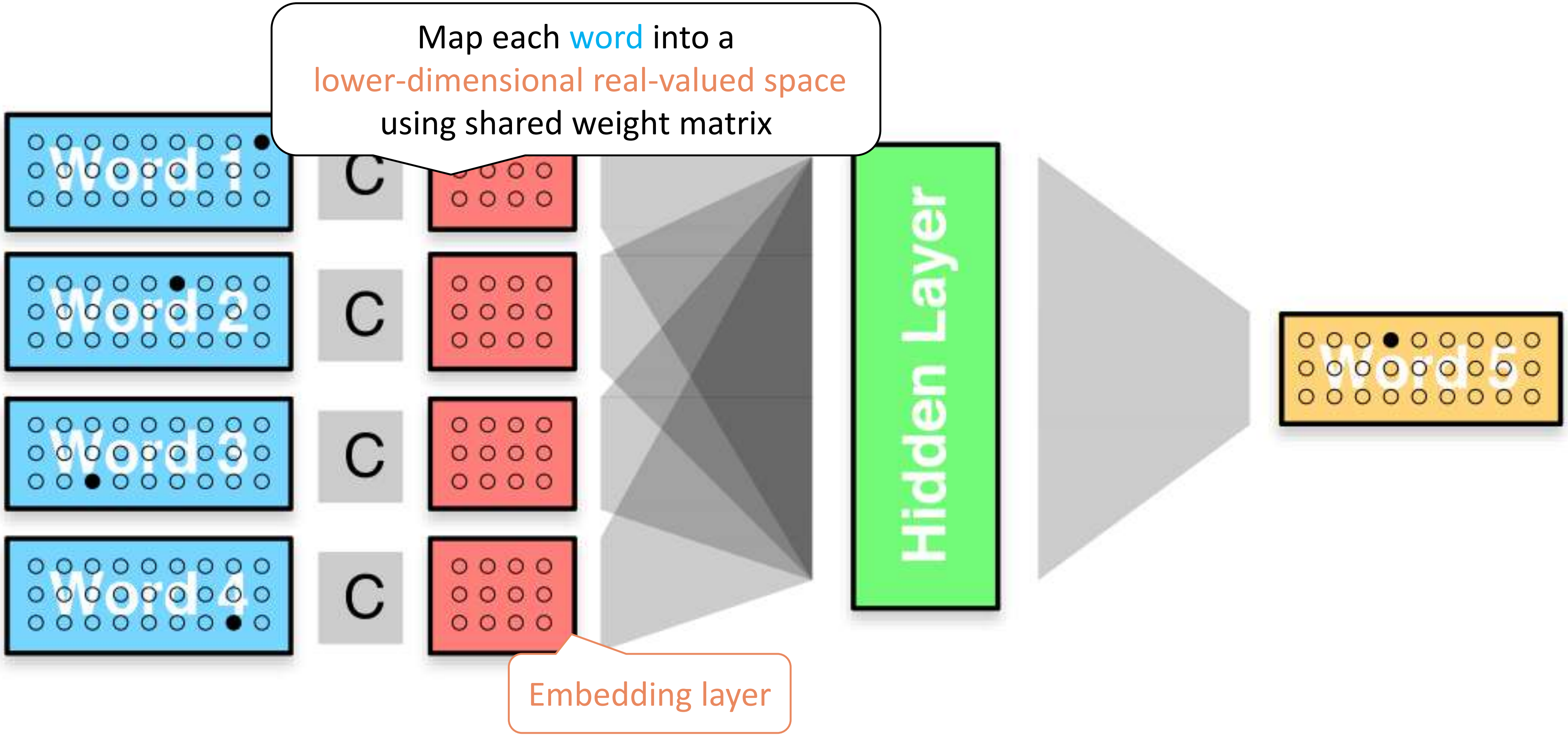
Representing Words

- “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 …]
```

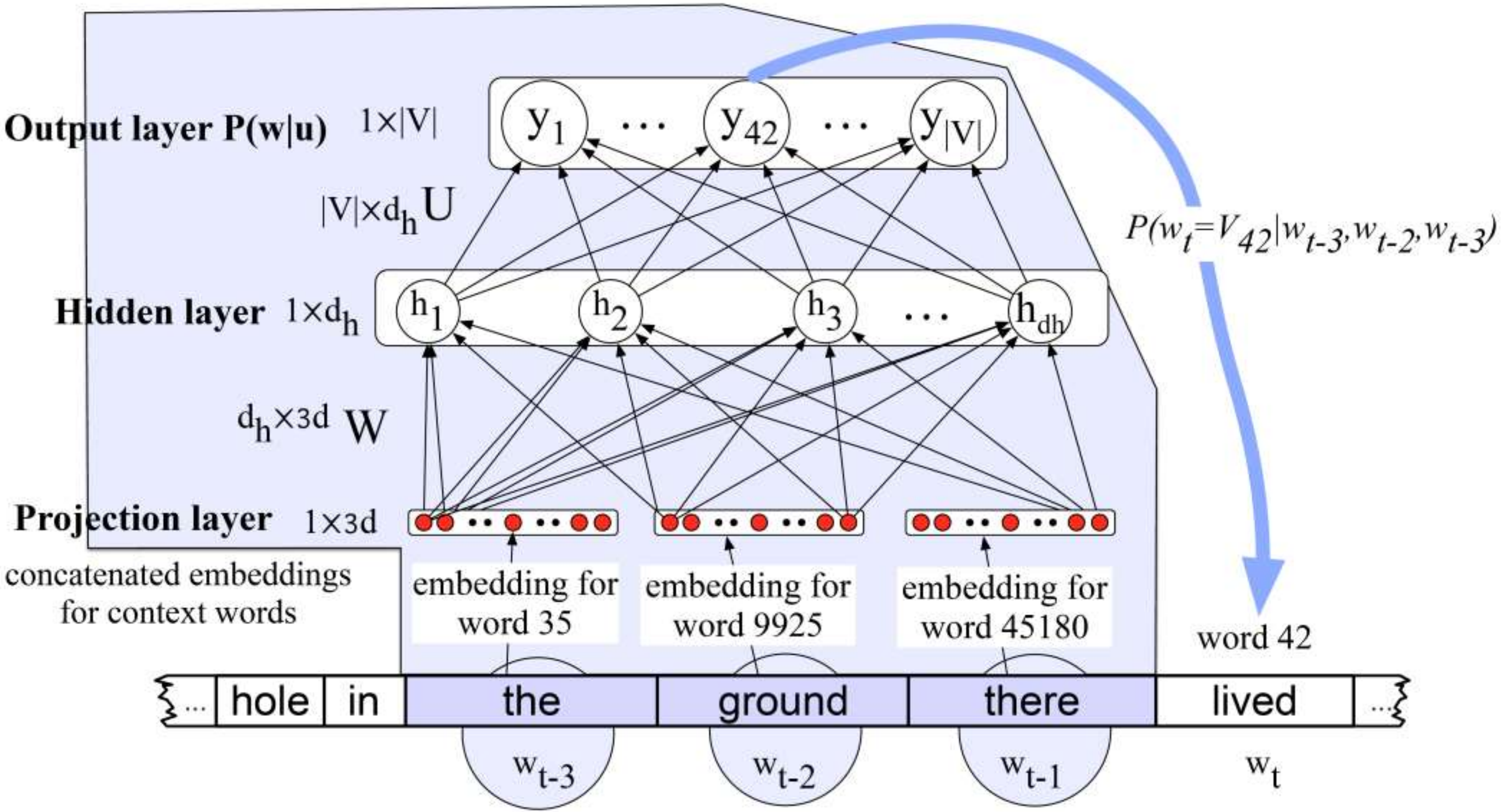
- That’s a large vector! practical solutions:
  - limit to most frequent words (e.g., top 20000)
  - cluster words into classes
  - break up rare words into subword units

Language Modeling with Feedforward Neural Networks

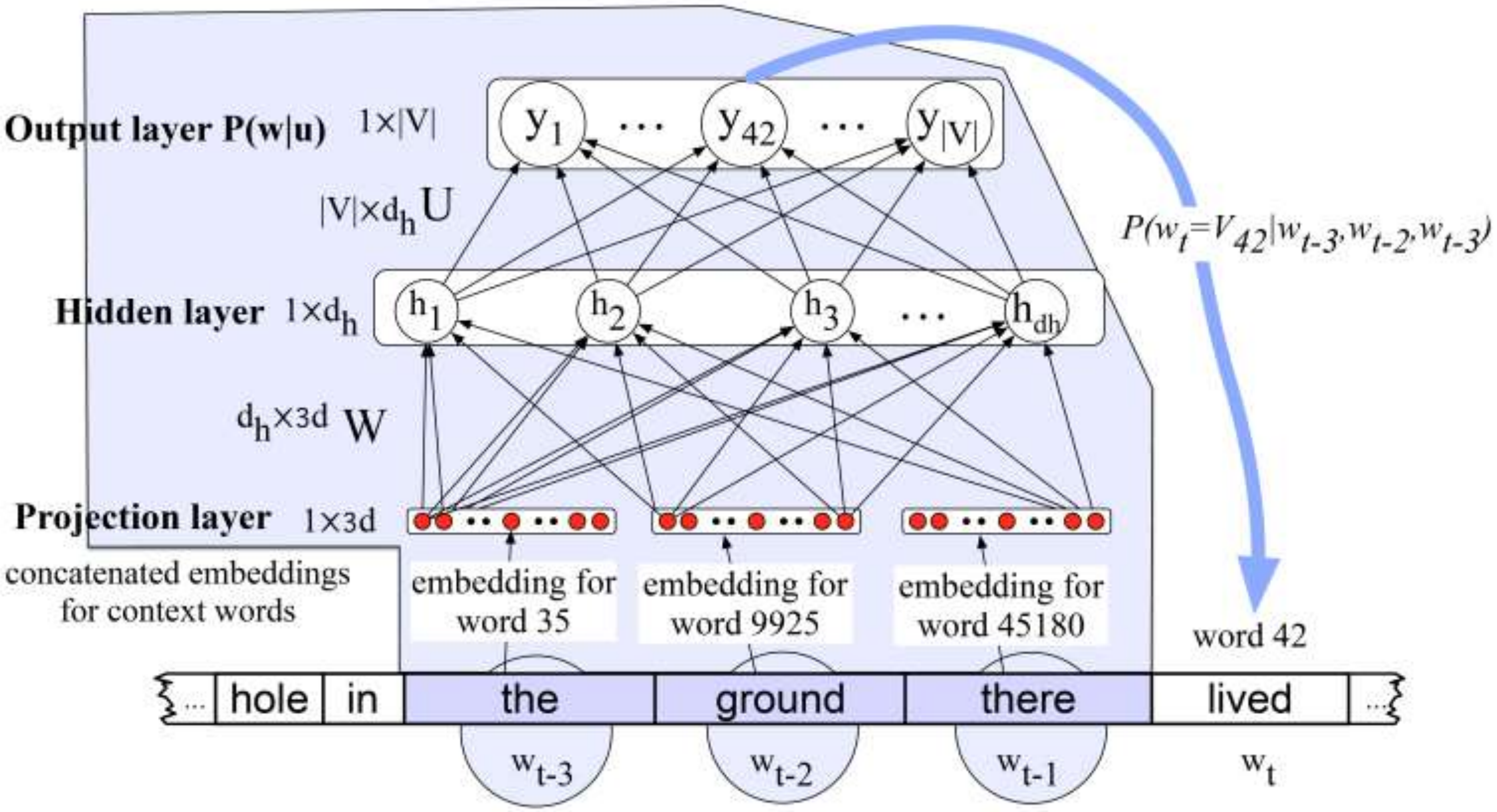


Bengio et al. 2003

Example: Prediction with a Feedforward LM



Example: Prediction with a Feedforward LM



$$
\begin{aligned}
e &= (Ex_1, Ex_2, \ldots, Ex) \\
h &= \sigma(We + b) \\
z &= Uh \\
y &= \mathrm{softmax}(z)
\end{aligned}
$$

Note: bias omitted in figure

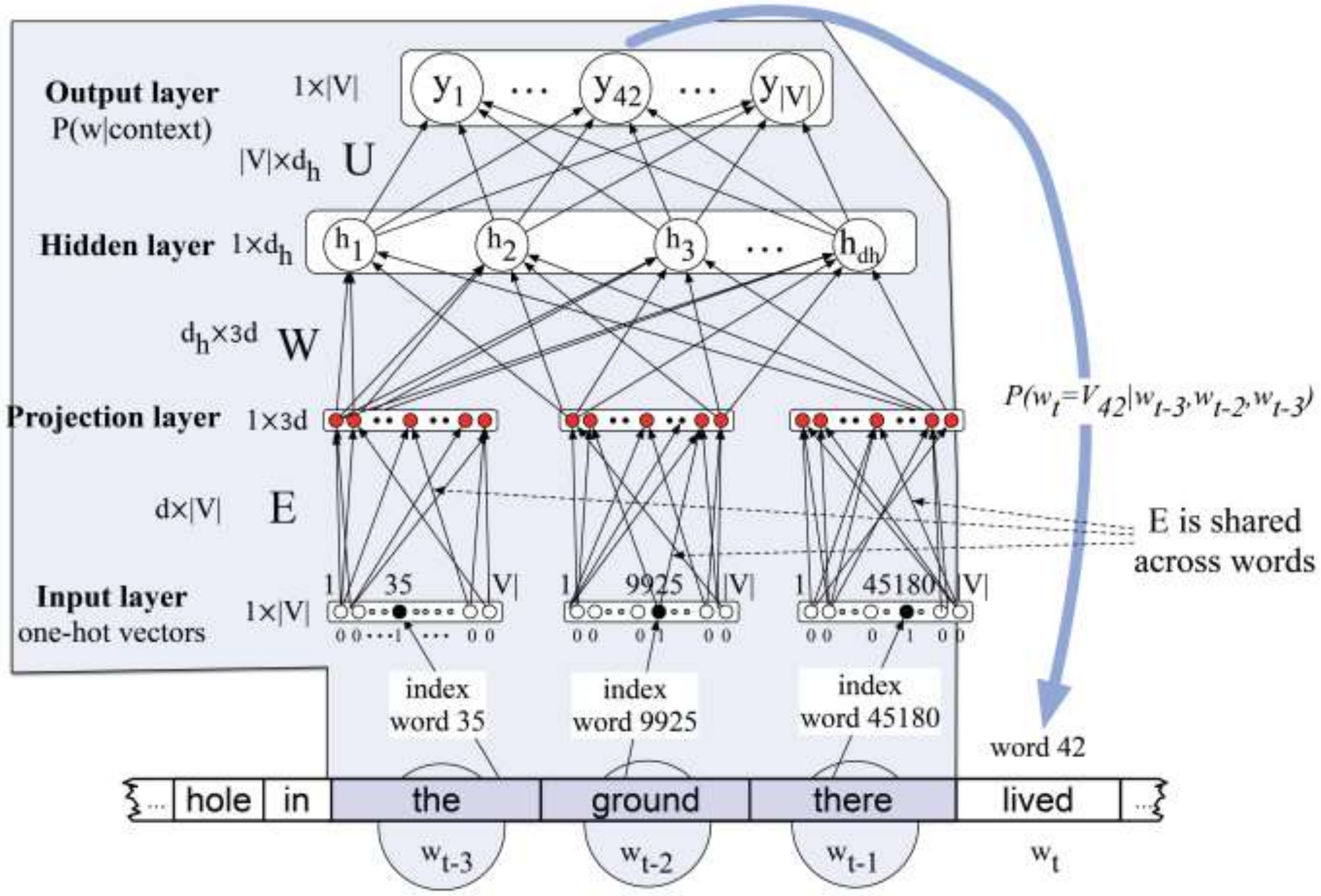
Estimating Model Parameters

- 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 $\text{error}(\lambda) = -\log P_\lambda(e_t | e_1 \dots e_{t-1})$

Example: Parameter Estimation



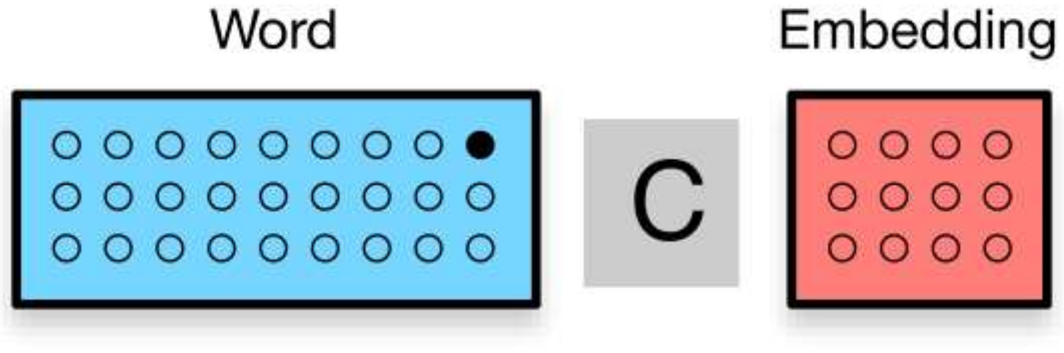
Loss function at each position t

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

Parameter update rule

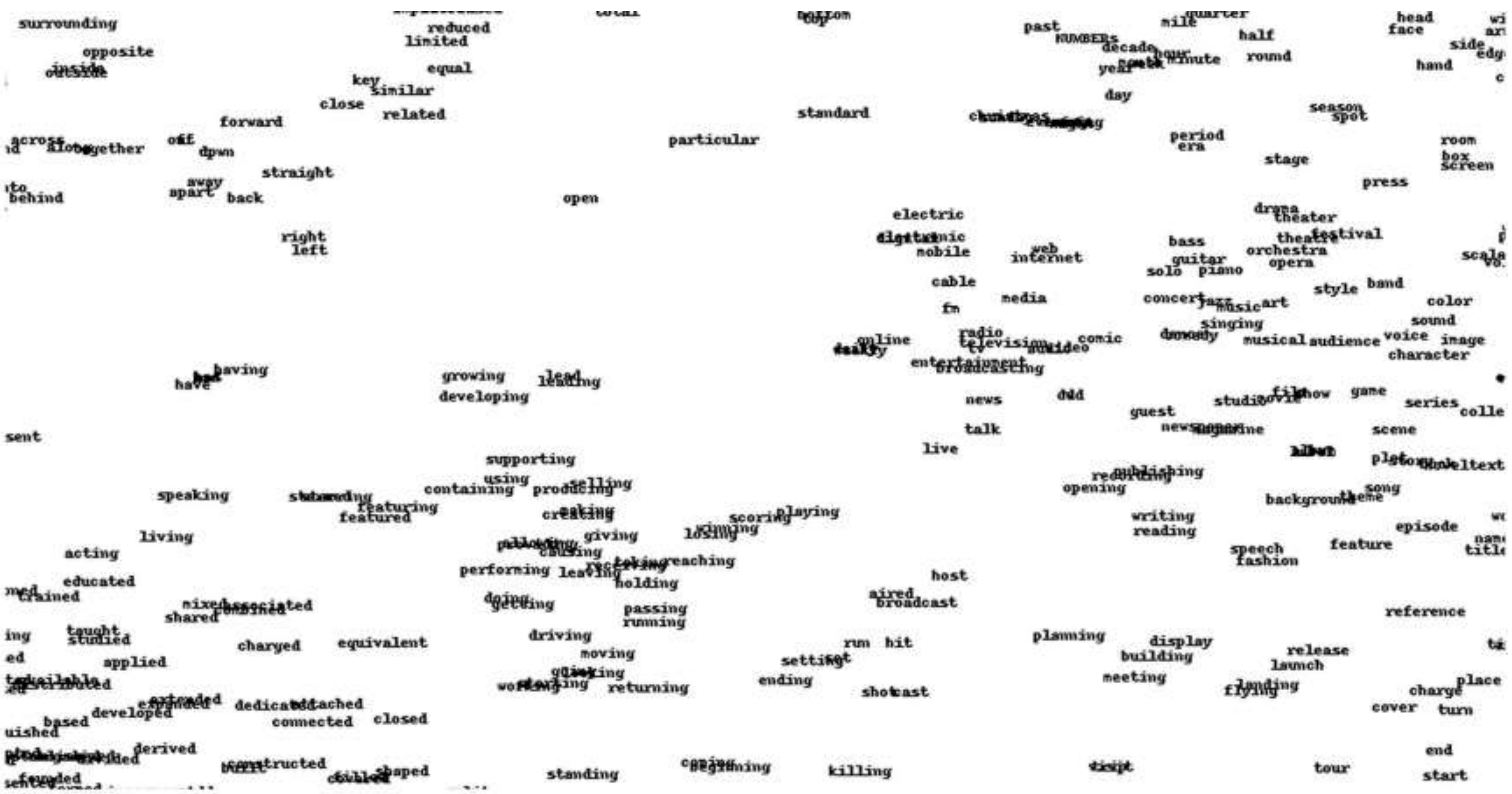
$$\theta_{t+1} = \theta_t - \eta \frac{\partial - \log p(w_t | w_{t-1}, \dots, w_{t-n+1})}{\partial \theta}$$

Word Embeddings: a useful by-product of neural LMs

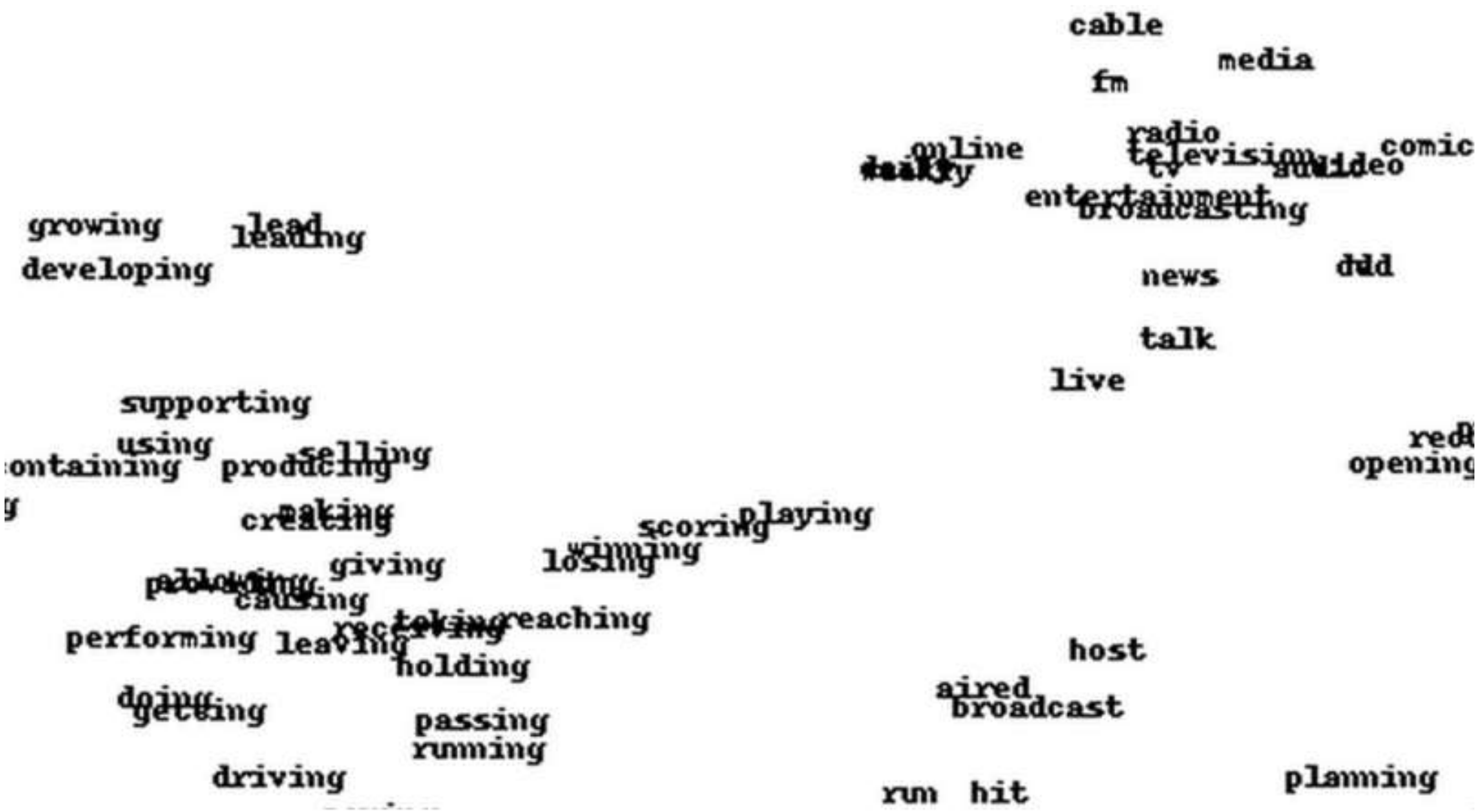


- Words that occurs in similar contexts tend to have similar embeddings
- Embeddings capture many usage regularities
- Useful features for many NLP tasks

Word Embeddings



Word Embeddings

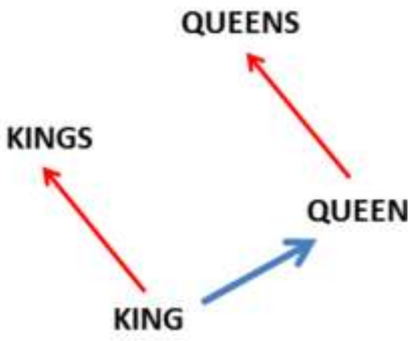


Word Embeddings Capture Useful Regularities

Morpho-Syntactic

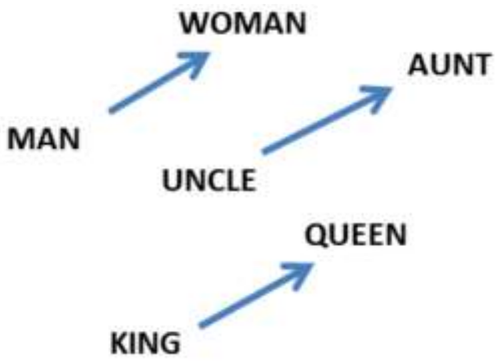
- Adjectives: base form vs. comparative
- Nouns: singular vs. plural
- Verbs: present tense vs. past tense

[Mikolov et al. 2013]

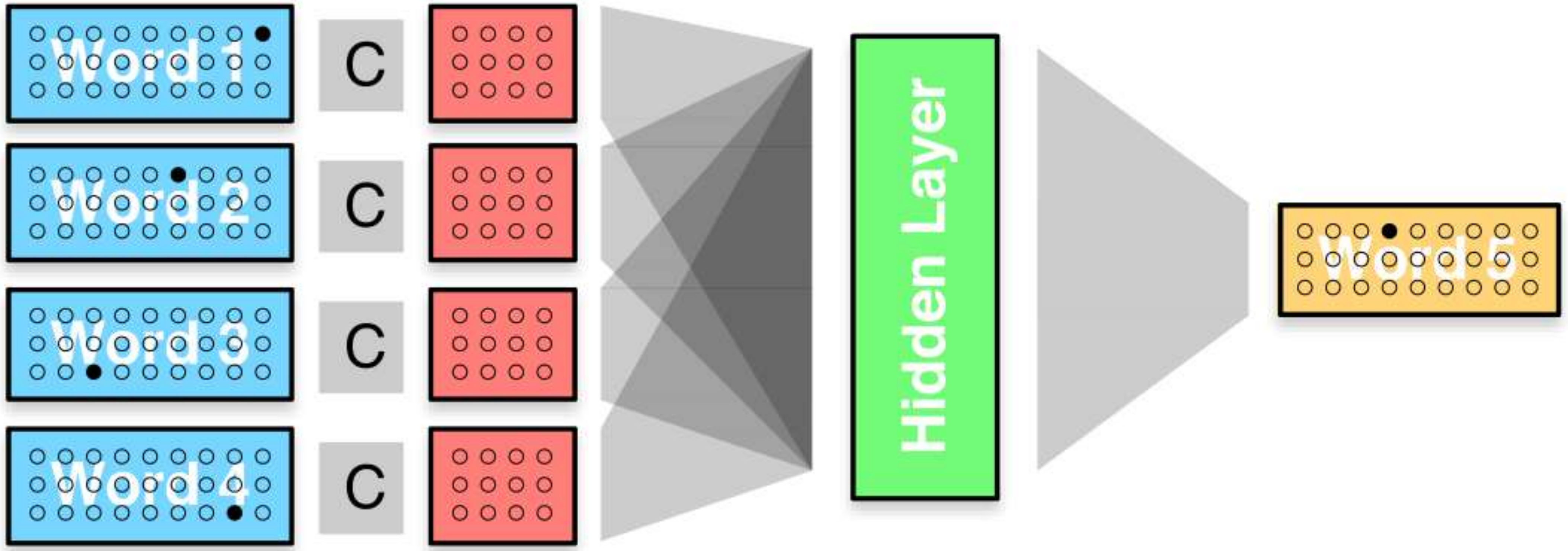


Semantic

- 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

Count-based n-gram models vs. feedforward neural networks

- Pros of feedforward neural LM
  - Word embeddings capture generalizations across word typesq

- Cons of feedforward neural LM
  - Closed vocabulary
  - Training/testing is more computationally expensive

- Weaknesses of both types of model
  - Only work well for word prediction if the test corpus looks like the training corpus
  - Only capture short distance context

Roadmap

- Language Models
  - Our first example of modeling sequences

- n-gram language models
- How to estimate them?
- How to evaluate them?
- Neural models
  - Feedfworward neural networks
  - Recurrent neural networks