CS188 Discussion W10

Mingyu Derek Ma

Email: ma@cs.ucla.edu

Reminder

- Class project deadline extended to Mar 18 Friday 11:59pm
- Additional test trials on Gradescope [Details]
- Peer evaluation quiz for everyone due Mar 18 11:59pm
- Project report specifications update [Link]
  - Include model checkpoint link in your report
  - Include Gradescope trial number for the number you reported
- HW2 grades released
- Final exam: next Monday Mar 14 3pm

Common issues with source project

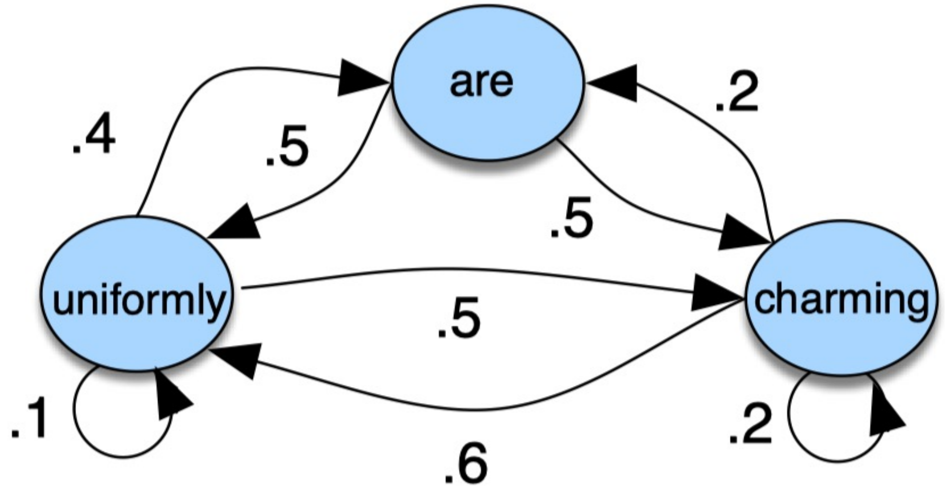
- Use `from_pretrained` instead of `from_config` to load model
- Print `labels` and `preds` to make sure your data loading is correct
- Attempt to make Sem-Eval work first to troubleshoot Com2Sense
- Look at your TensorBoard curves, your training loss has to decrease!
- Do NOT set the `logging_steps` to too small!
- Use `iters_to_eval` to specify the checkpoint iteration to run testing

Today

Two of the most voted review topics:

- Hidden Markov Models and the Viterbi algorithm
- Word vectors

Hidden Markov Model



- (Not hidden) Markov chain
  - Example: bigram LM as a Markov chain
  - States are words in the vocabulary
  - To predict next word, you only need to look at the current word
- Hidden Markov Model
  - Hidden events: such as part-of-speech tags
  - Observed events: such as words in a sentences
- Generative model

Assumption of HMM

- Assumption 1: Markov Assumption $P(q_i|q_1,...,q_{i-1}) = P(q_i|q_{i-1})$
  - When predicting the future, the past doesn't matter, only the present
  - The probability of a particular state depends only on the previous state
  - Same intuition as bigram language model
- Assumption 2: Output Independence $P(o_i|q_1,\ldots q_i,\ldots,q_T,o_1,\ldots,o_i,\ldots,o_T) = P(o_i|q_i)$
  - Probability of an output observation
    - Depends only on the state that produced the observation
    - Not on any other states or any other observations
  - Word are independent of each other given the tag sequence

HMM Components

Parameters need to be learned

- States (unique hidden events)
- Observations (observed events)
- Initial probability distribution
  - Probability that the Markov chain will start in a certain state
- Transition probability matrix
  - Probability moving from a **state** to another **state**
  - Answer questions like “which is the most likely tag after a VB tag?”
- Observation likelihoods / emission probabilities
  - Probability of an **observation** being generated from a **state**
  - Answer questions like “if we are going to generate a VB, how likely is it to be ‘eat’?”

Prepare HMM parameters

- Assume there are only 2 states (NN, VB) and 2 words (eat, food)
- Corpus

```
eat_NN food_NN food_VB
eat_VB food_NN
food_NN eat_NN eat_VB
```

- Initial state probabilities
  - P(NN | start) = 2/3
  - P(VB | start) = 1/3

Prepare HMM parameters

- Assume there are only 2 states (NN, VB) and 2 words (eat, food)
- Corpus

```
eat_NN food_NN food_VB
eat_VB food_NN
food_NN eat_NN eat_VB
```

- Transition probability P(state | state)

from/to	to NN	to VB
from NN	2 -> 2/4	2 -> 2/4
from VB	1 -> 1/1	0 -> 0

Prepare HMM parameters

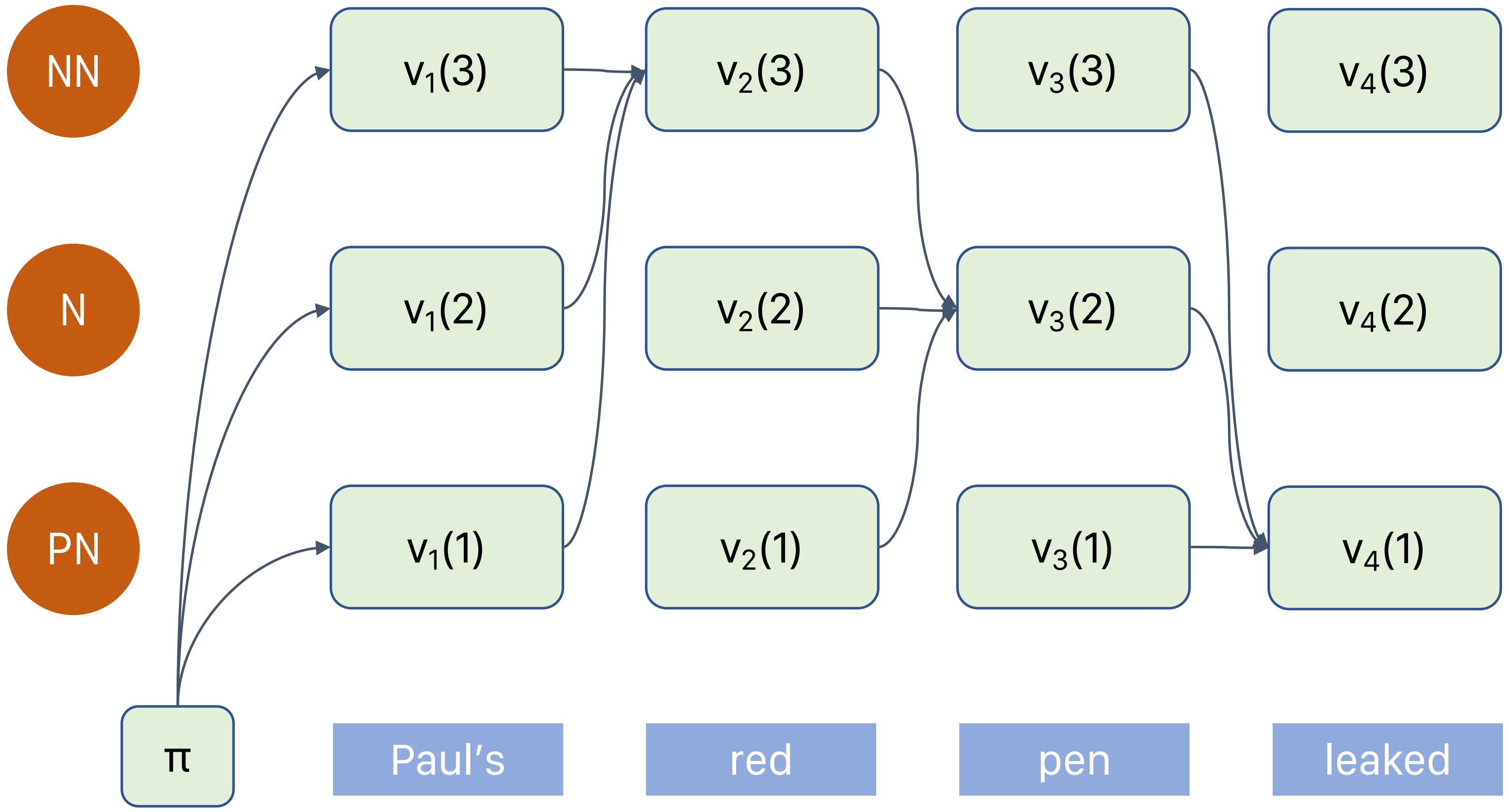
- Assume there are only 2 states (NN, VB) and 2 words (eat, food)
- Corpus

```
eat_NN food_NN food_VB
eat_VB food_NN
food_NN eat_NN eat_VB
```

- Emission probability P(word | state)

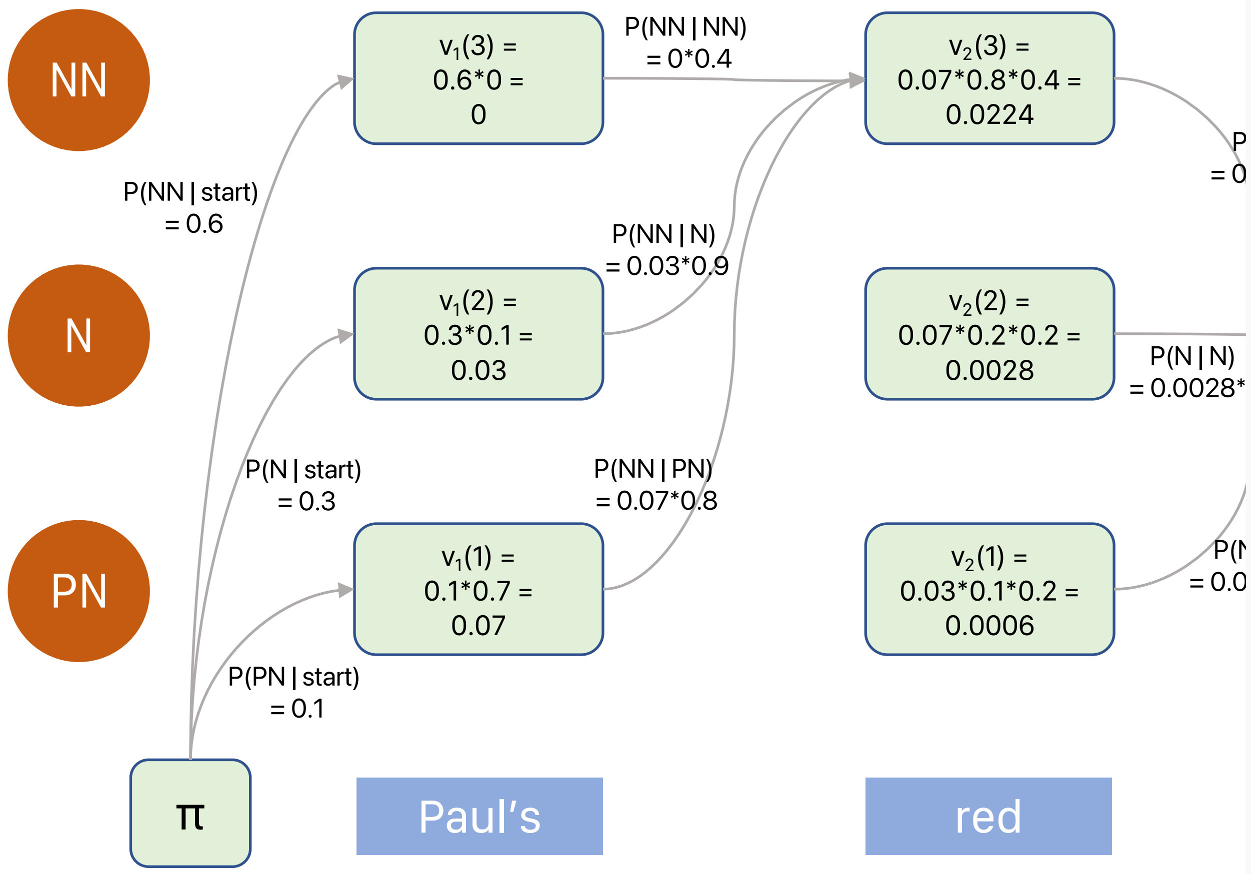
	eat	food
NN	2 -> 2/5	3 -> 3/5
VB	2 -> 2/3	1 -> 1/3

Decoding: set up lattice



Note: Some connections are omitted for simplicity

Viterbi algorithm forward



Note: Some connections are o

Initial probabilities:

	PN	N	NN
π	0.1	0.3	0.6

Transition probabilities:

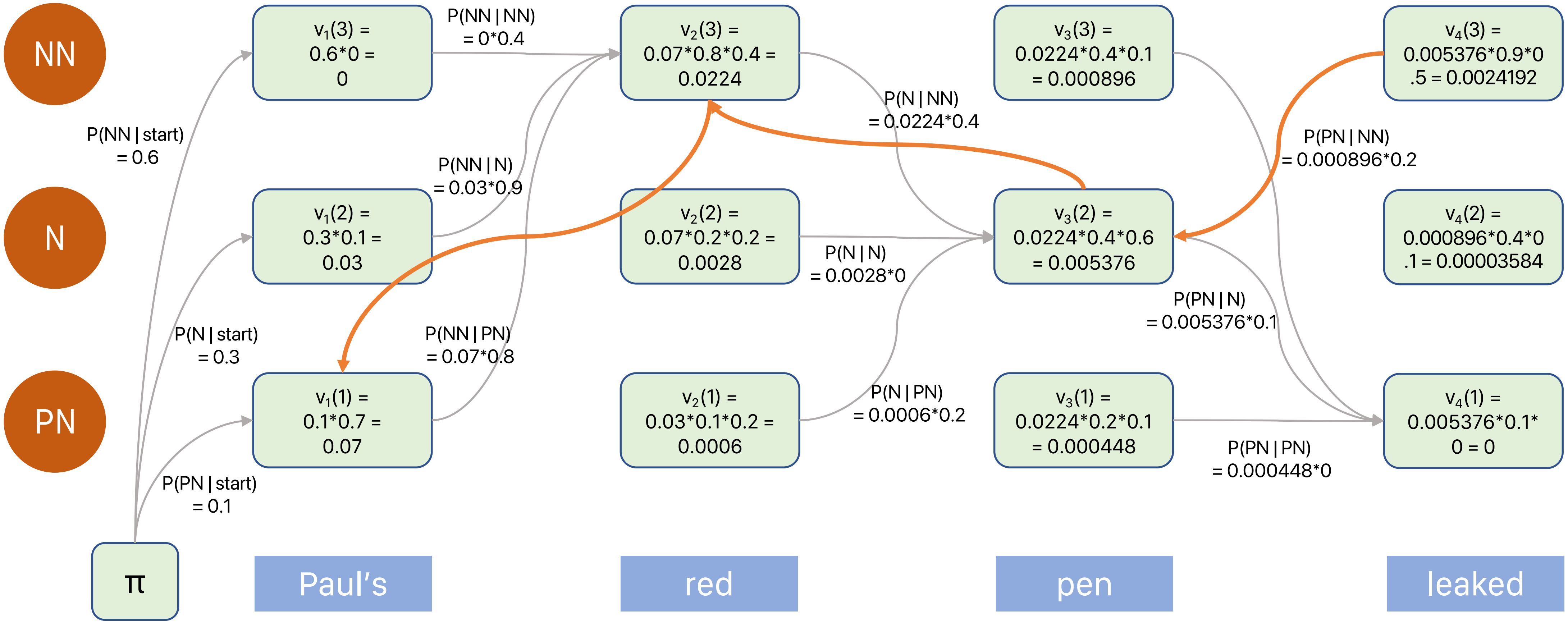
from/to	to PN	to N	to NN
from PN	0	0.2	0.8
from N	0.1	0	0.9
from NN	0.2	0.4	0.4

For example: $P(N|PN) = 0.2$ and $P(NN|PN) = 0.8$.

Emission probabilities:

	Paul's	red	pen	leaked
PN	0.7	0.2	0.1	0
N	0.1	0.2	0.6	0.1
NN	0	0.4	0.1	0.5

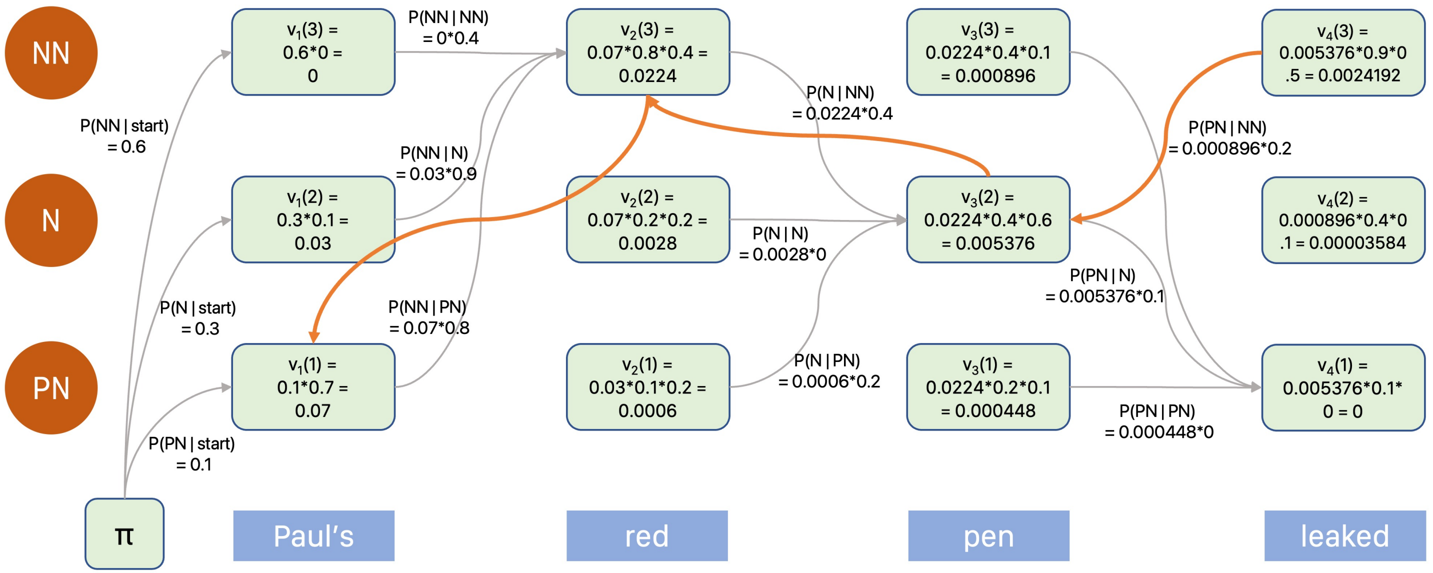
Viterbi algorithm backward



Note: Some connections are omitted for simplicity

Time complexity of the Viterbi algorithm

- Given
  - Number of states: Y
  - Sequence length: T
- YT+Y+YY(T−1)
  - YT: trellis has YT (state, observed event) pair, each we need to multiple an emission probability
  - Y: start to t=1 states
  - YY(T−1): transition between states, for each transition arc we need to multiple a transition probability



Note: Some connections are omitted for simplicity

Word vectors

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

- One-hot word vectors
- Word vectors in the term-document matrix
  - Word occurs in the documents
  - Similar words have similar vectors because they tend to occur in similar documents
  - Can use tf-idf or PPMI to weight this matrix
- Word vectors in the term-term matrix

	aardvark	...	computer	data	result	pie	sugar	...
cherry	0	...	2	8	9	442	25	...
strawberry	0	...	0	0	1	60	19	...
digital	0	...	1670	1683	85	5	4	...
information	0	...	3325	3982	378	5	13	...

- Dense word embeddings
  - Vectors are shorter
  - Values are real-valued numbers (not like the other three which are sparse and mostly zero)

Word vectors

Sparse word vectors

Dense word vectors

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

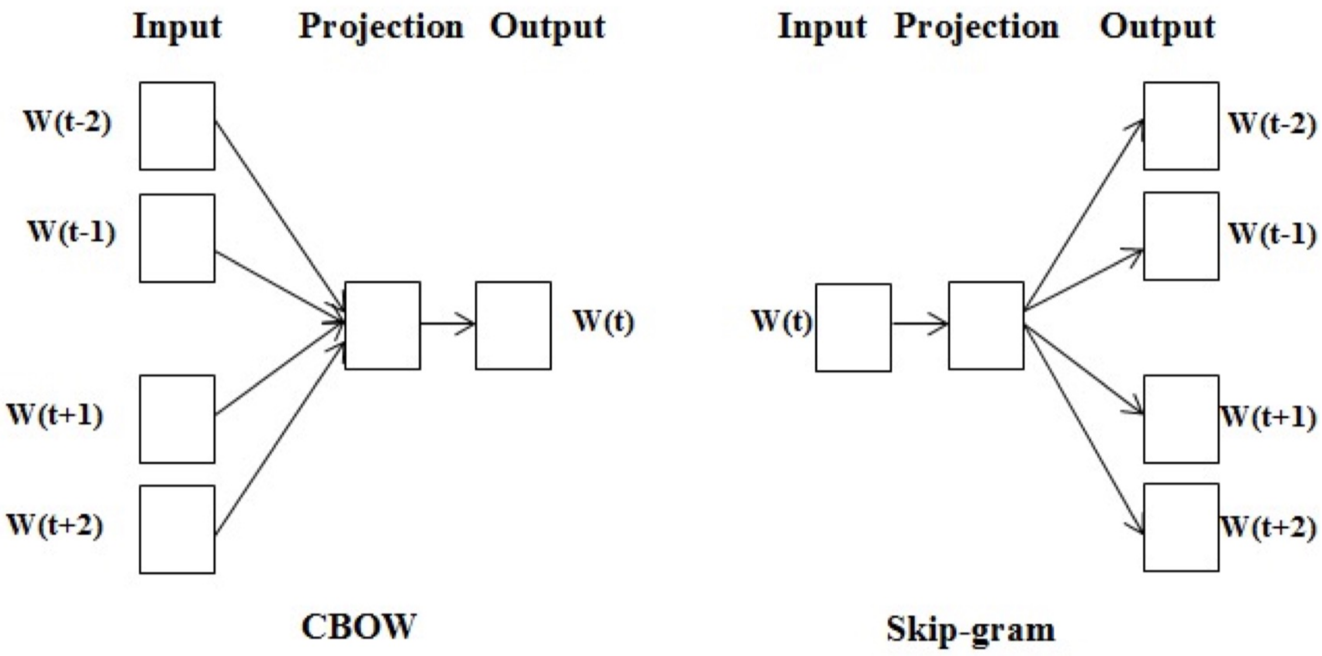
- One-hot word vectors
- Word vectors in the term-document matrix
    - Word occurs in the documents
    - Similar words have similar vectors because they tend to occur in similar documents
    - Can use tf-idf or PPMI to weight this matrix
- Word vectors in the term-term matrix

	aardvark	...	computer	data	result	pie	sugar	...
cherry	0	...	2	8	9	442	25	...
strawberry	0	...	0	0	1	60	19	...
digital	0	...	1670	1683	85	5	4	...
information	0	...	3325	3982	378	5	13	...

- Dense word embeddings
    - Vectors are shorter
    - Values are real-valued numbers (not like the other three which are sparse and mostly zero)

Word2Vec

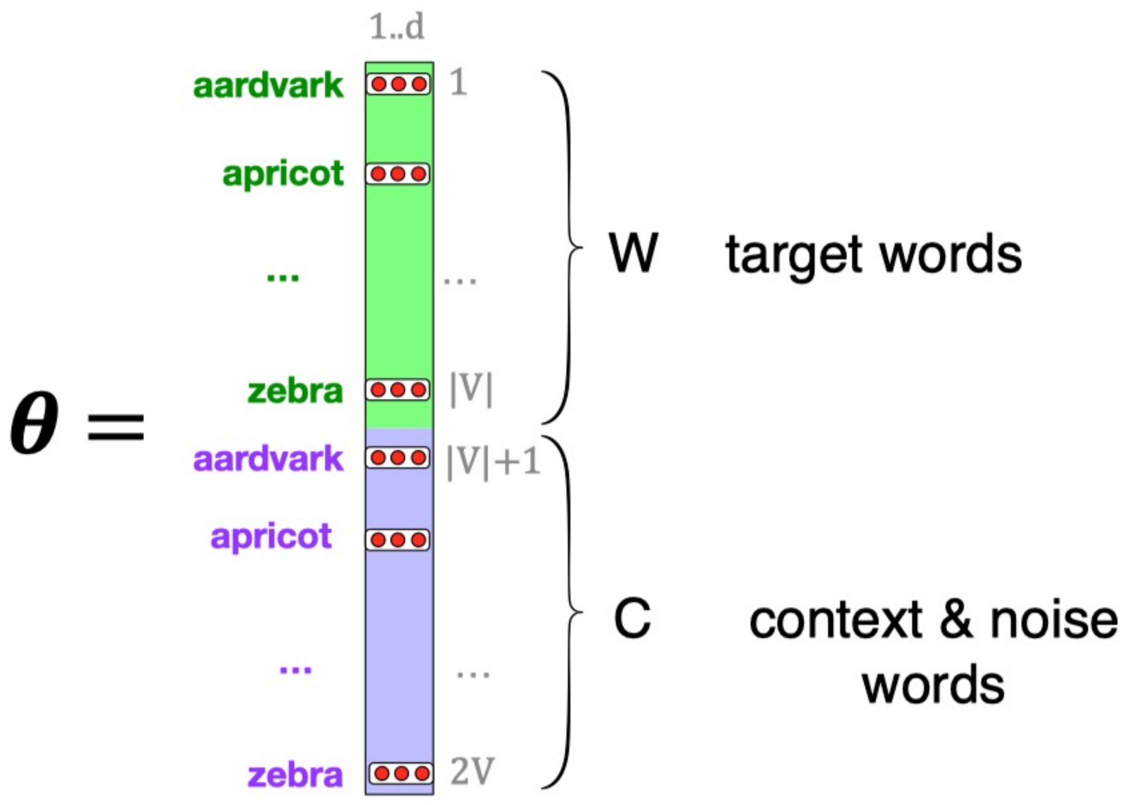
Skip-gram v.s Continuous bag-of-words



Lecture Note 03, Page 32

- Objective: should a word likely to show up in a context
- Word2vec trains a logistic regression classifier (not FFNN, nor RNN etc) to distinguish two cases
  - Positive: target word in context
  - Negative: random sampled word and context pairs
- The learned weights are the embeddings

Word2Vec: Skip-gram



Skip-gram model embeddings
Textbook J&M Figure 6.13

- The intuition of the skip-gram model is based on embedding similarity -> **dot product**
- Turn dot product to a probability [0, 1] using **sigmoid function**
- We only need embeddings of each target word and context word in the vocabulary, each has |V|d parameters
  - Target / input embedding
  - Context / output embedding

Skip-gram example

- Say we have a piece of training data

```
... lemon,  a [tablespoon of apricot jam,      a] pinch ...
               c1          c2    w     c3      c4
```

- Target word: `apricot`, 4 context words
- Create training examples

positive examples +

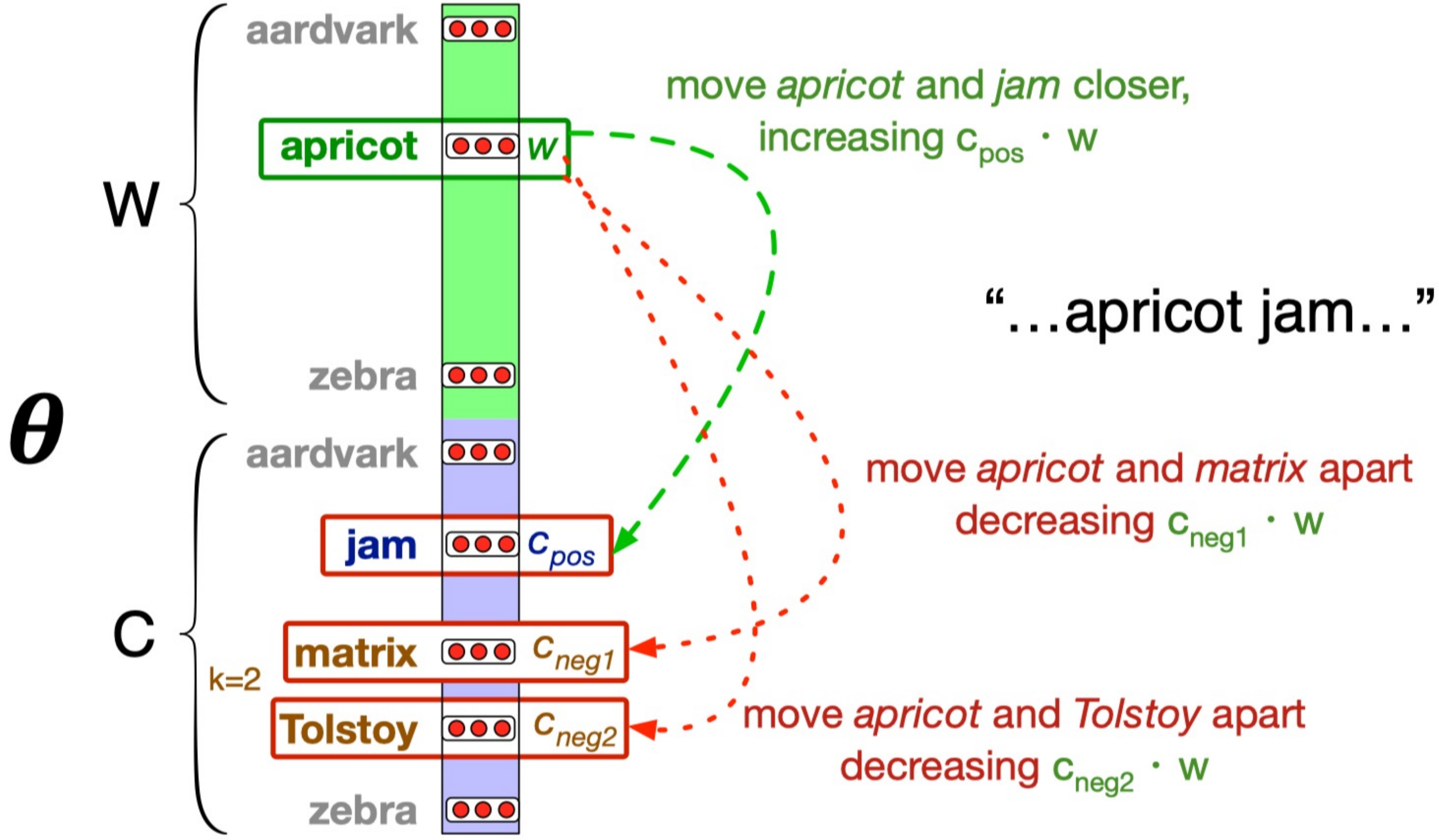
w	c_{pos}
apricot	tablespoon
apricot	of
apricot	jam
apricot	a

negative examples -

w	c_{neg}	w	c_{neg}
apricot	aardvark	apricot	seven
apricot	my	apricot	forever
apricot	where	apricot	dear
apricot	coaxial	apricot	if

Skip-gram example

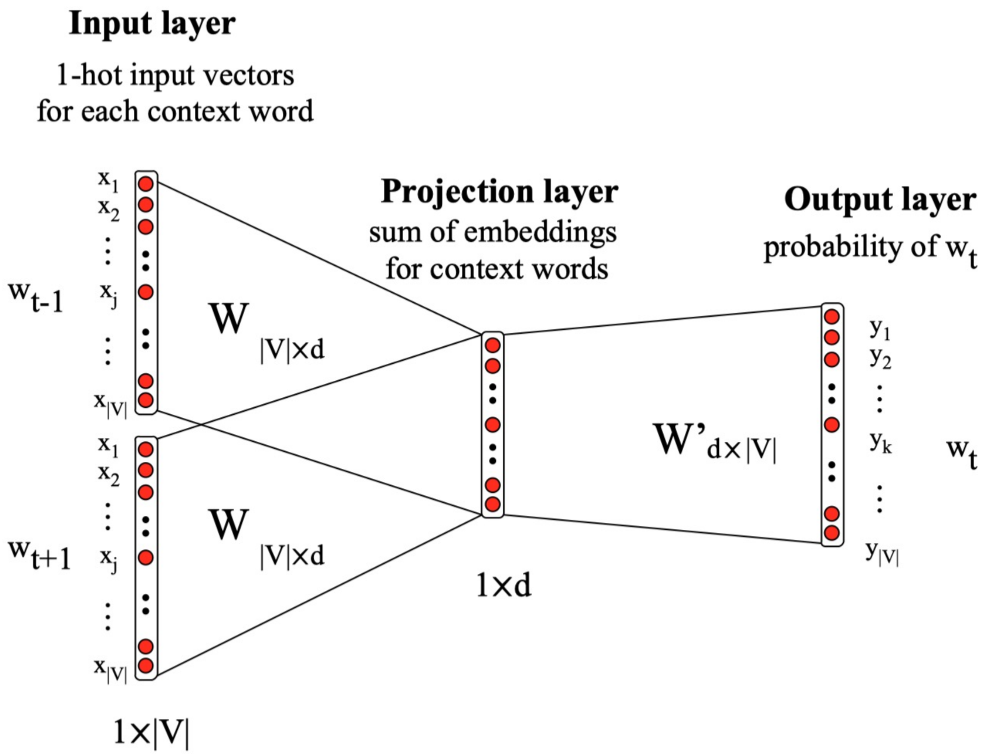
- Find target word embedding and context word embeddings
- Update these embeddings to
  - Increase the dot products with positive samples
  - Decrease the dot products with negative samples
- Using gradient descent



$$L_{CE} = -\left[\log \sigma(c_{pos} \cdot w) + \sum_{i=1}^{k} \log \sigma(-c_{neg_i} \cdot w)\right]$$

Word2Vec: Continuous Bag of Words

CBOW (Continuous Bag of Words)



- Input and output embedding matrix
- Element-wise averaging for embeddings of context words
- Nothing to do with RNN

Lecture Note 03, Page 40