Word Embeddings

Wei Xu

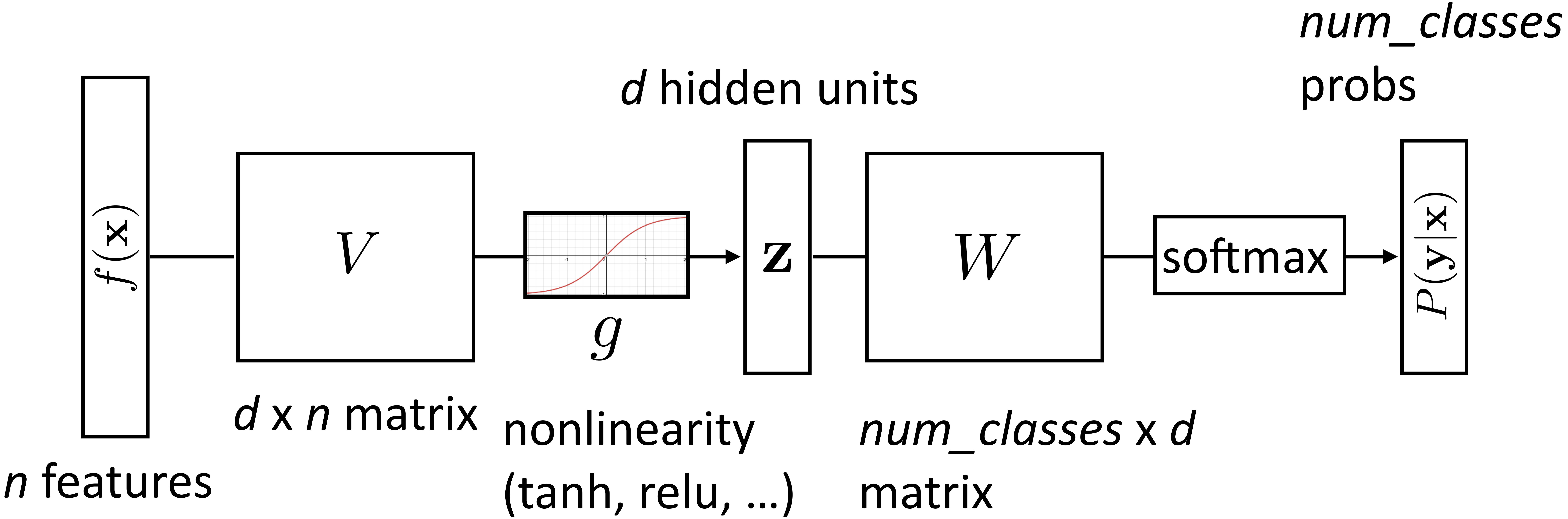
(many slides from Greg Durrett)

Administrivia

- Homework 2 due next Tuesday
- Reading: Eisenstein 3.3.4, 14.5, 14.6, J+M 6

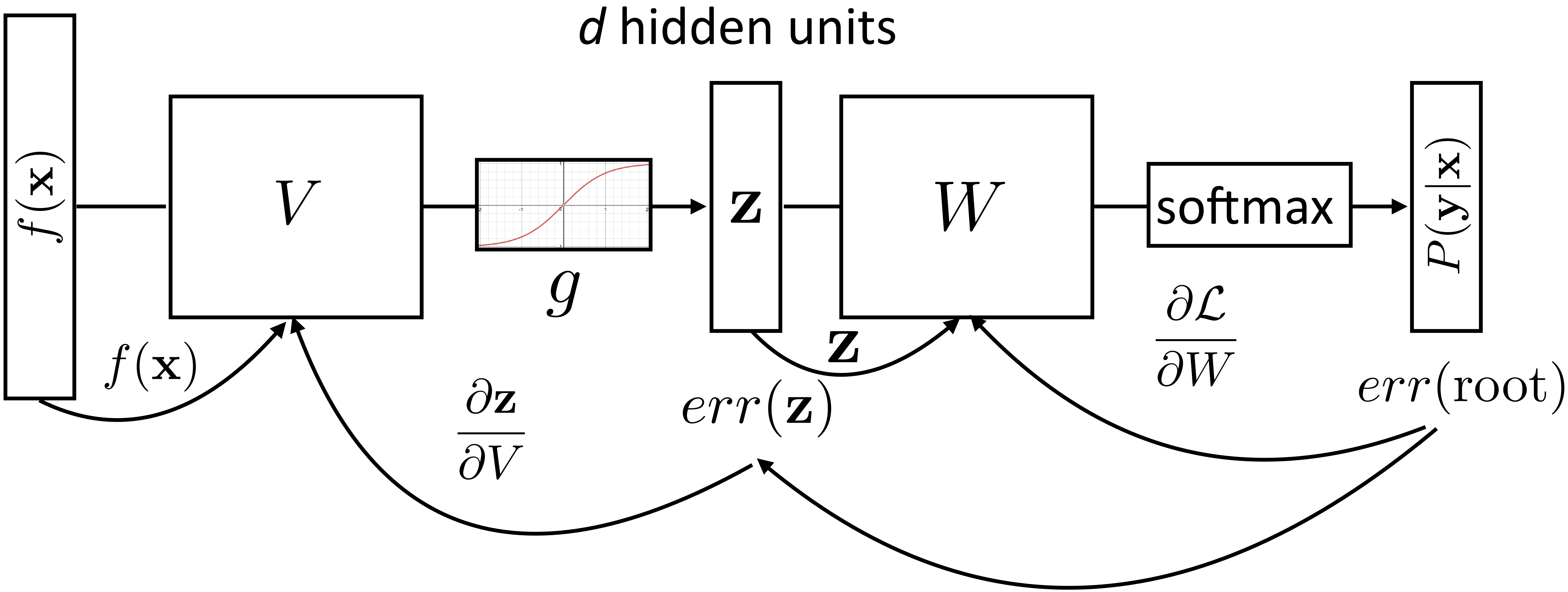
Recall: Feedforward NNs

$$P(\mathbf{y}|\mathbf{x}) = \mathrm{softmax}(Wg(Vf(\mathbf{x})))$$



Recall: Backpropagation

$$P(\mathbf{y}|\mathbf{x}) = \text{softmax}(Wg(Vf(\mathbf{x})))$$



This Lecture

- Training
- Word representations
- word2vec/GloVe
- Evaluating word embeddings

Training Tips

Batching

- Batching data gives speedups due to more efficient matrix operations

- Need to make the computation graph process a batch at the same time

```
# input is [batch_size, num_feats]
# gold_label is [batch_size, num_classes]
def make_update(input, gold_label)
    ...
    probs = ffnn.forward(input) # [batch_size, num_classes]
    loss = torch.sum(torch.neg(torch.log(probs)).dot(gold_label))
    ...
```

- Batch sizes from 1-100 often work well

Training Basics

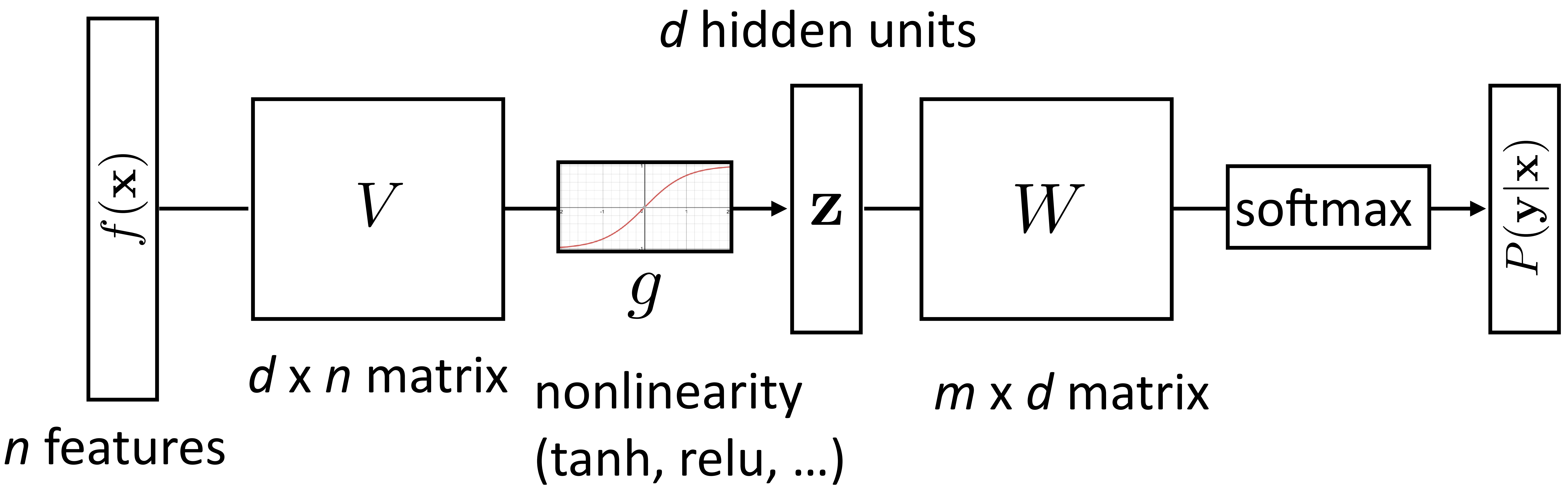
- Basic formula: compute gradients on batch, use first-order optimization method (SGD, Adagrad, etc.)

- How to initialize? How to regularize? What optimizer to use?

- This lecture: some practical tricks. Take deep learning or optimization courses to understand this further

How does initialization affect learning?

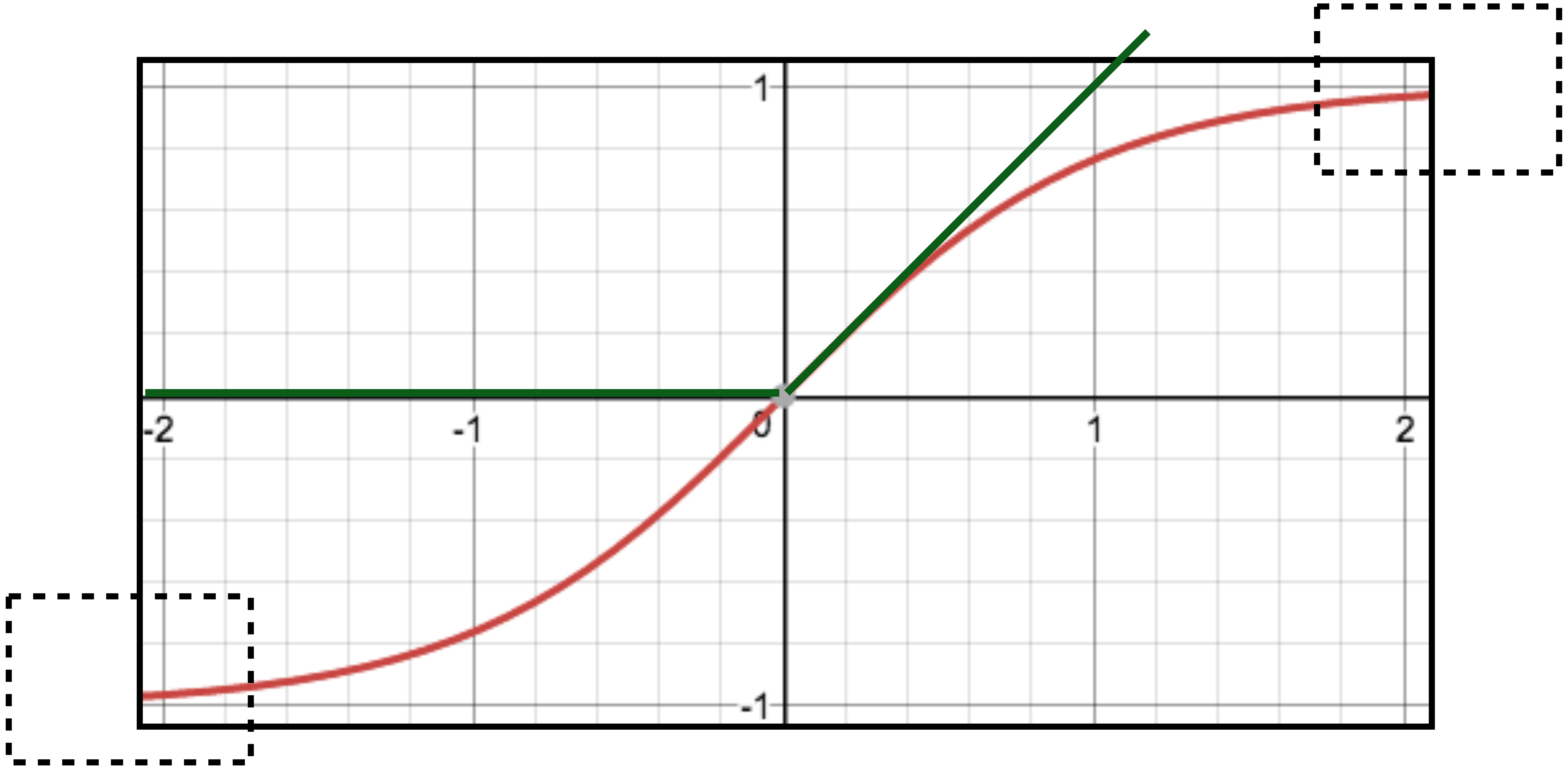
$$P(\mathbf{y}|\mathbf{x}) = \text{softmax}(Wg(Vf(\mathbf{x})))$$



- How do we initialize V and W? What consequences does this have?
- *Nonconvex* problem, so initialization matters!

How does initialization affect learning?

- Nonlinear model…how does this affect things?



- Tanh: If cell activations are too large in absolute value, gradients are small
- ReLU: larger dynamic range (all positive numbers), but can produce big values, and can break down if everything is too negative (“dead” ReLU)

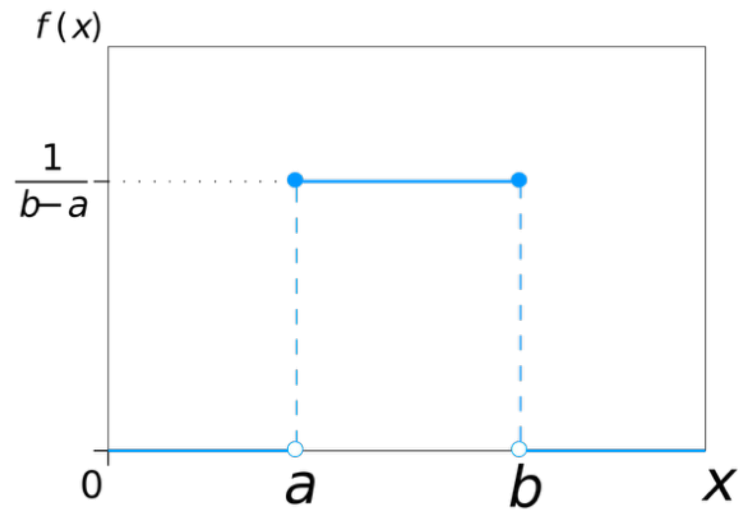
Krizhevsky et al. (2012)

Initialization

1) Can’t use zeroes for parameters to produce hidden layers: all values in that hidden layer are always 0 and have gradients of 0, never change

2) Initialize too large and cells are saturated

- Can do random uniform / normal initialization with appropriate scale
- Xavier initializer: $U\left[-\sqrt{\frac{6}{\text{fan-in} + \text{fan-out}}}, +\sqrt{\frac{6}{\text{fan-in} + \text{fan-out}}}\right]$
  - Want variance of inputs and gradients for each layer to be the same



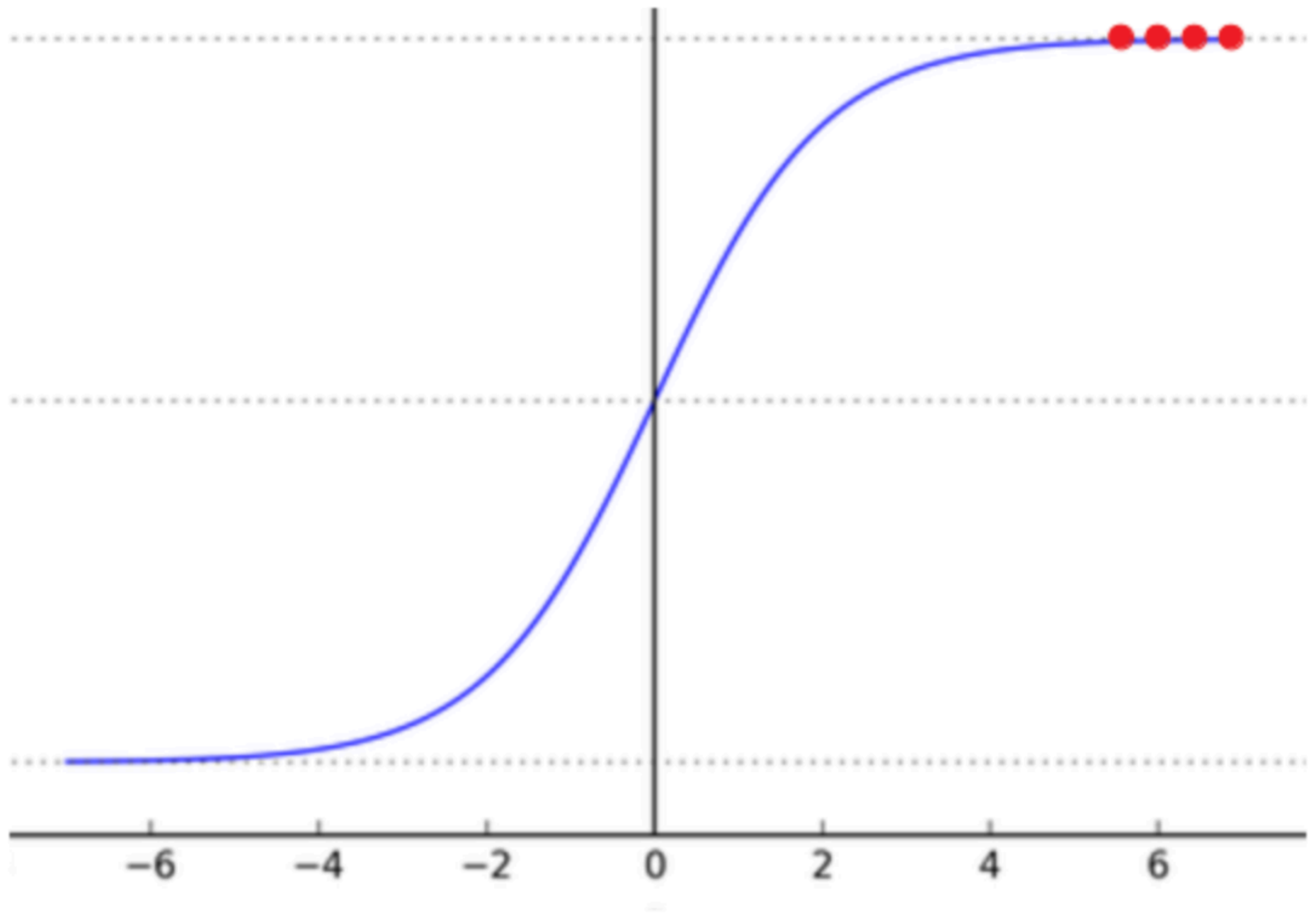
Mean & Standard Deviation

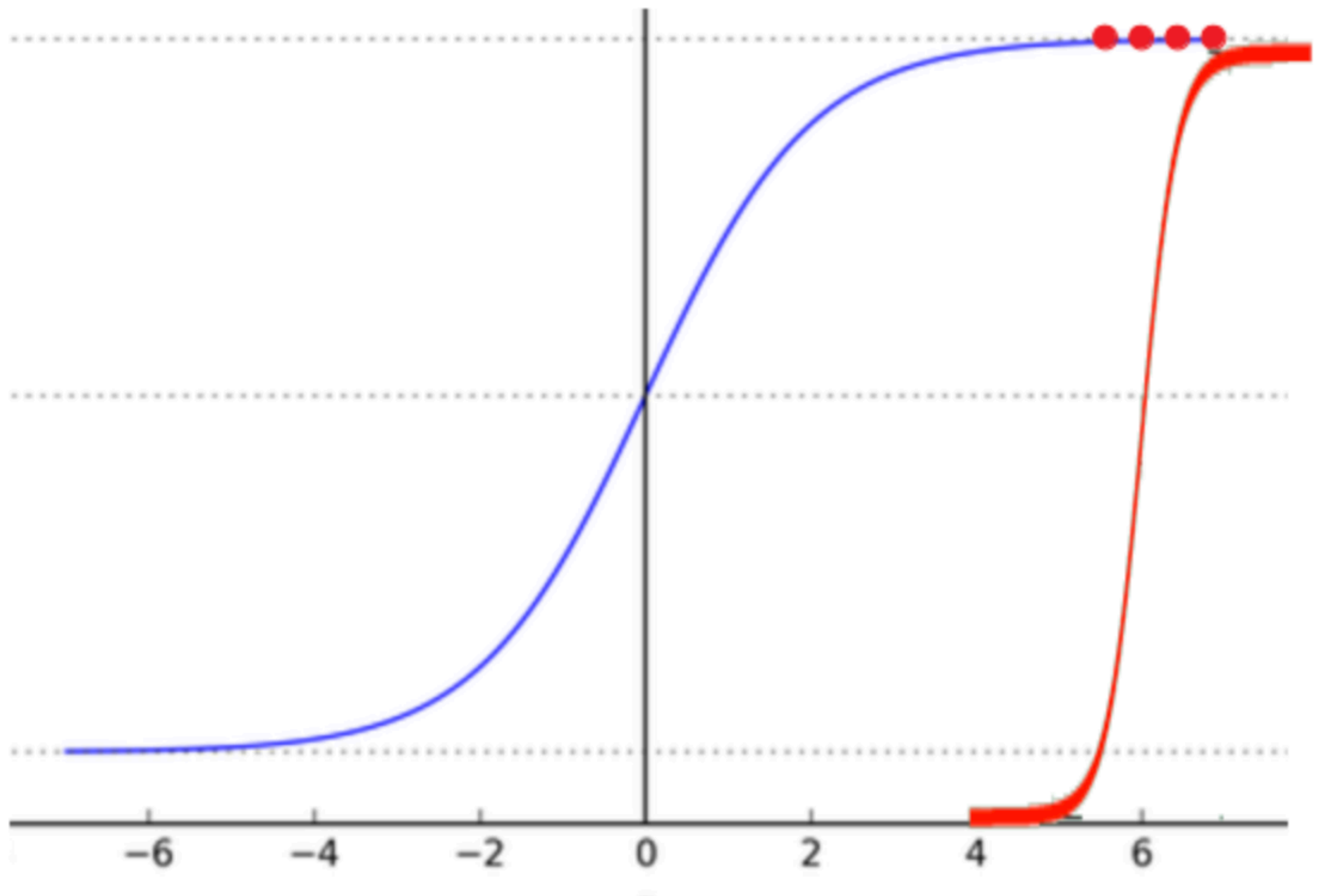
$$\mu = \frac{a+b}{2} \text{ and } \sigma = \frac{b-a}{\sqrt{12}}$$

https://mmuratarat.github.io/2019-02-25/xavier-glorot-he-weight-init
https://arxiv.org/pdf/1502.01852v1.pdf

Batch Normalization

- Batch normalization (Ioffe and Szegedy, 2015): periodically shift+rescale each layer to have mean 0 and variance 1 over a batch (useful if net is deep)

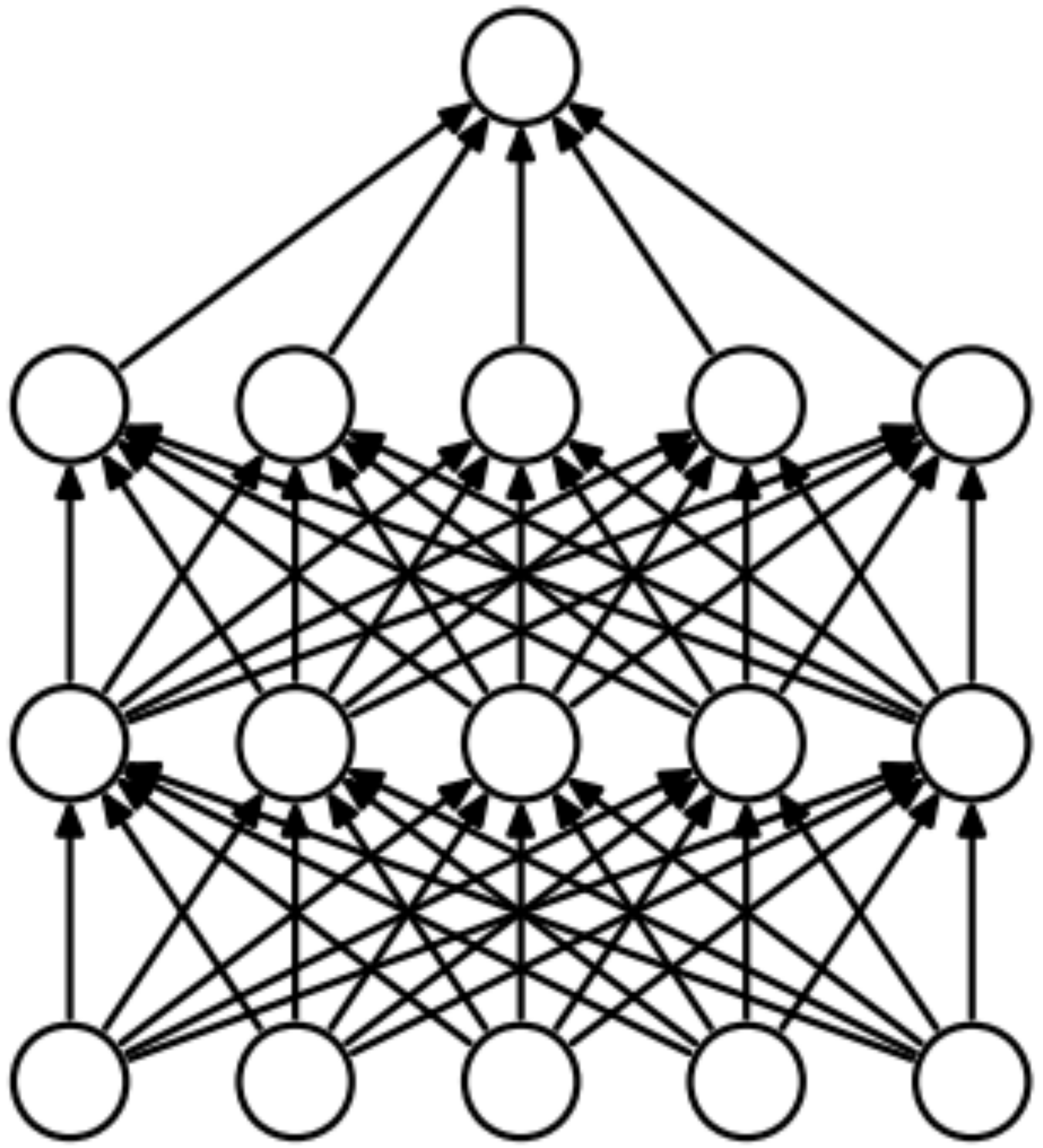




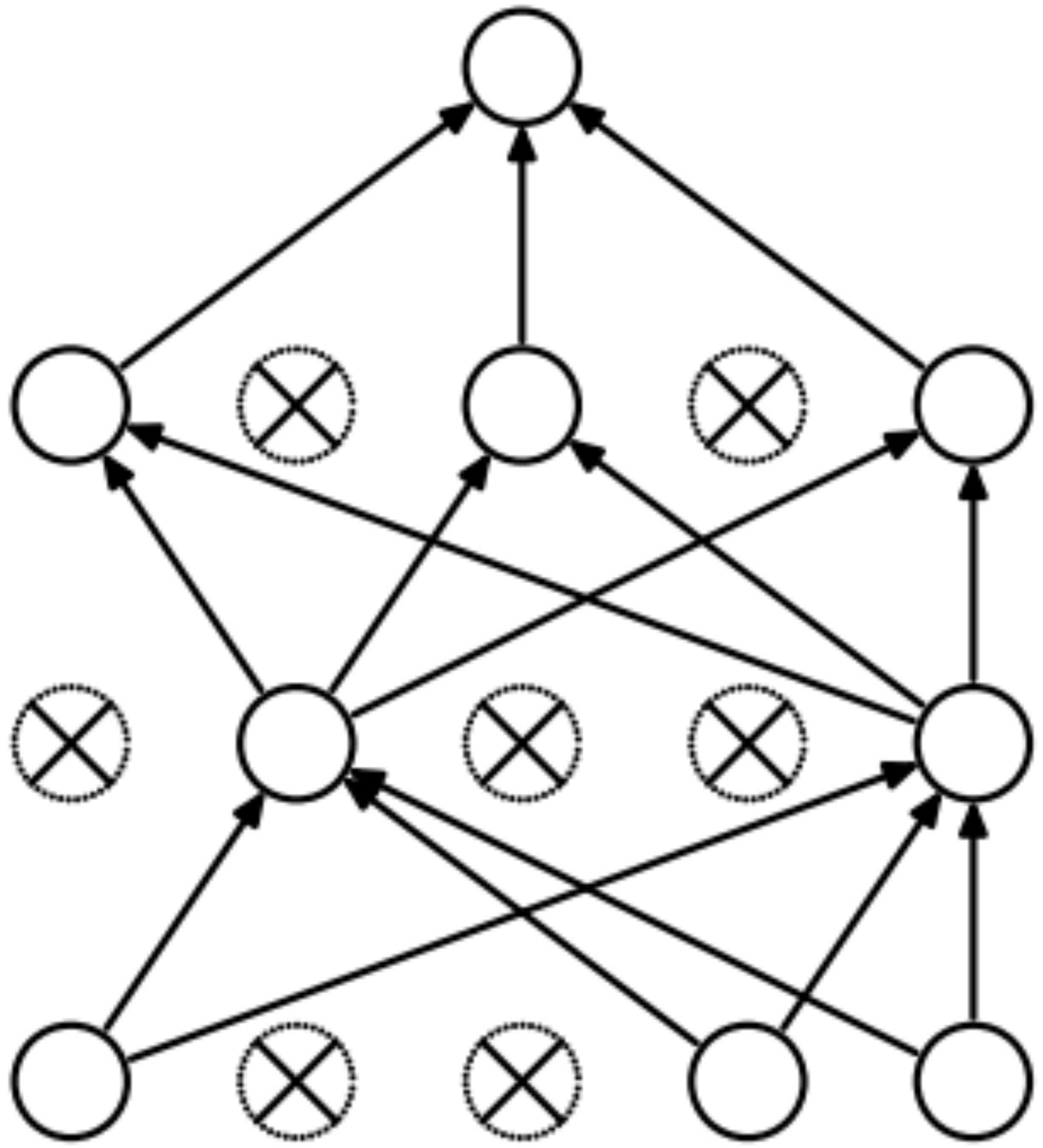
https://medium.com/@shiyan/xavier-initialization-and-batch-normalization-my-understanding-b5b91268c25c

Dropout

- Probabilistically zero out parts of the network during training to prevent overfitting, use whole network at test time
- Form of stochastic regularization
- Similar to benefits of ensembling: network needs to be robust to missing signals, so it has redundancy



(a) Standard Neural Net



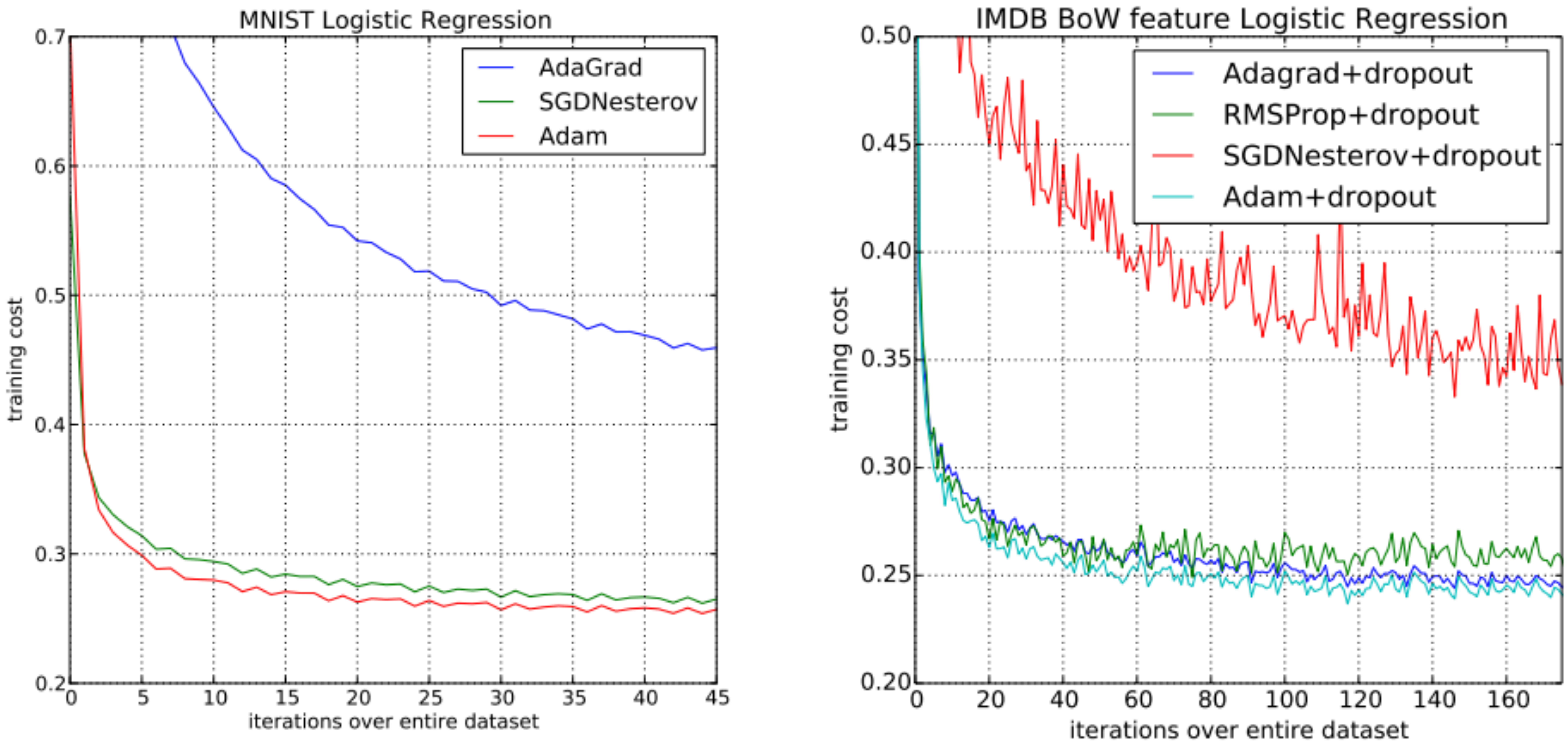
(b) After applying dropout.

- One line in Pytorch/Tensorflow

Srivastava et al. (2014)

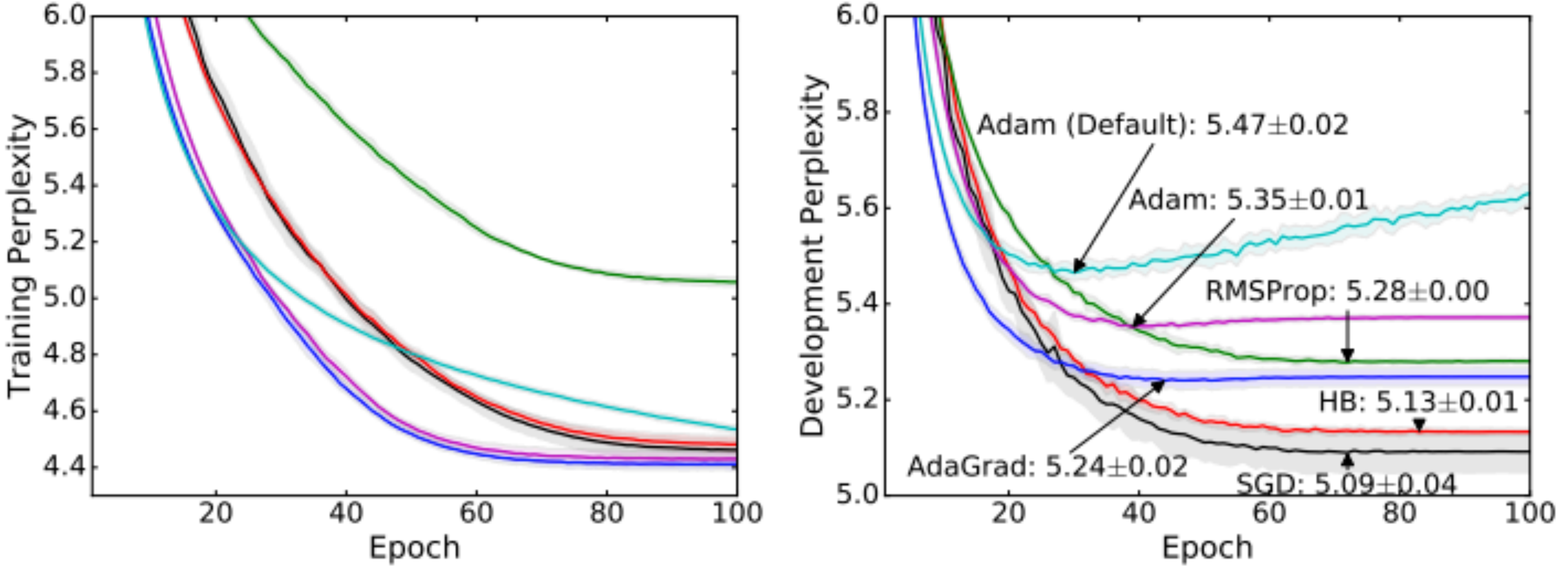
Optimizer

- Adam (Kingma and Ba, ICLR 2015) is very widely used
- Adaptive step size like Adagrad, incorporates momentum



Optimizer

- Wilson et al. NIPS 2017: adaptive methods can actually perform badly at test time (Adam is in pink, SGD in black)
- Check dev set periodically, decrease learning rate if not making progress

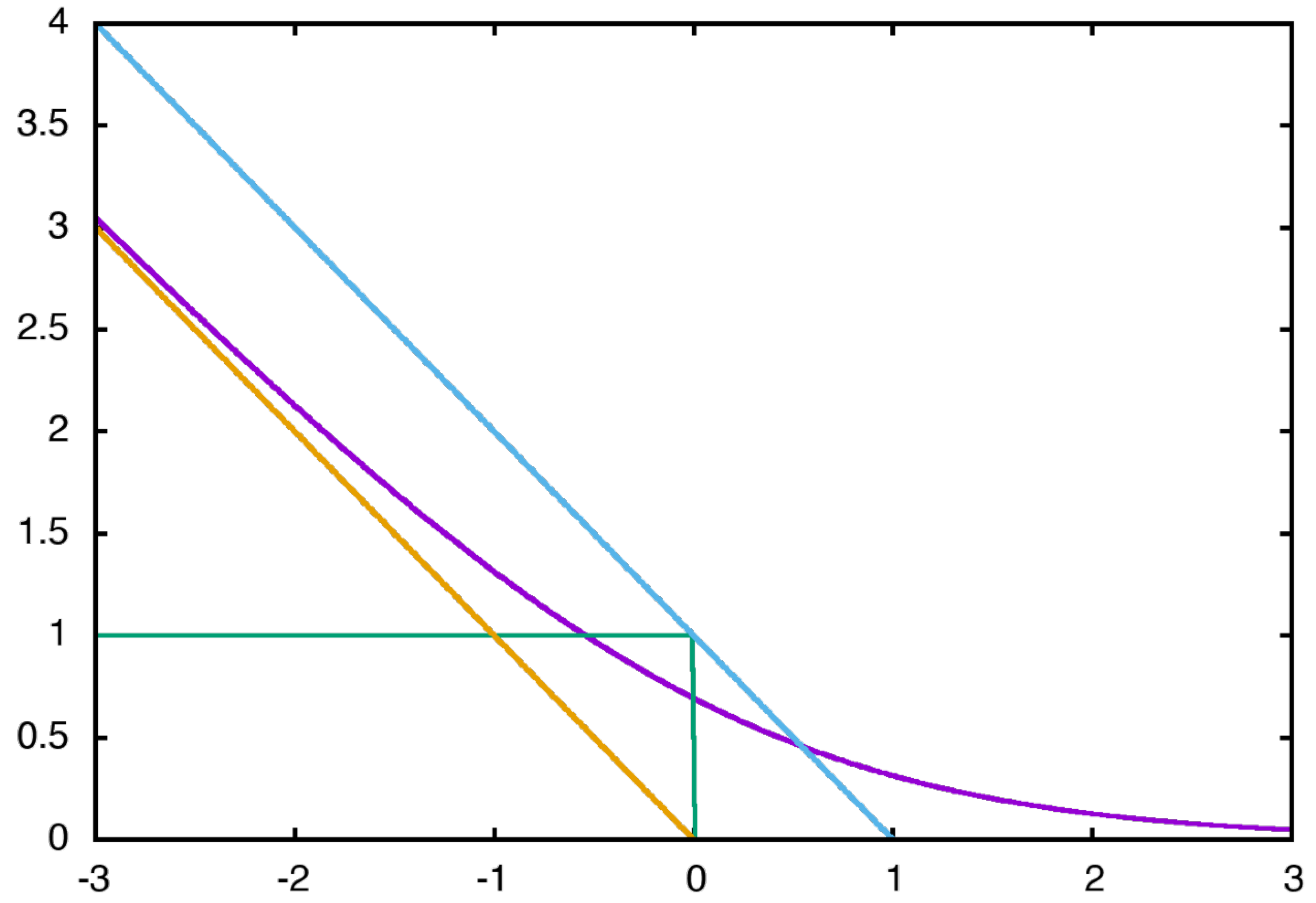


(e) Generative Parsing (Training Set) (f) Generative Parsing (Development Set)

Four Elements of NNs

- Model: feedforward, RNNs, CNNs can be defined in a uniform framework

- Objective: many loss functions look similar, just changes the last layer of the neural network



- Inference: define the network, your library of choice takes care of it (mostly…)

- Training: lots of choices for optimization/hyperparameters

Word Representations

Word Representations

- Neural networks work very well at continuous data, but words are discrete
- Continuous model <-> expects continuous semantics from input
- “You shall know a word by the company it keeps” Firth (1957)

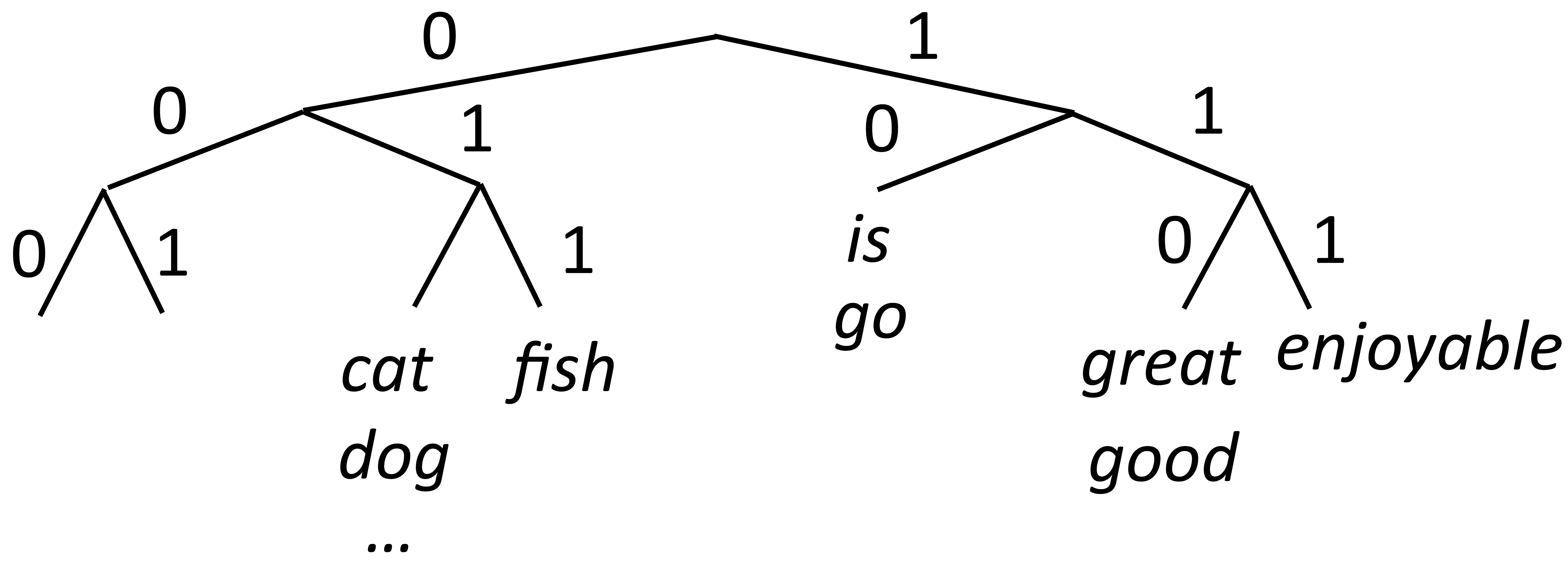
A bottle of ***tesgüino*** is on the table
Everybody likes ***tesgüino***
Tesgüino makes you drunk
We make ***tesgüino*** out of corn.

John R. Firth

slide credit: Dan Klein, Dan Jurafsky

Discrete Word Representations

- Brown clusters: hierarchical agglomerative *hard* clustering (each word has one cluster, not some posterior distribution like in mixture models)



- Maximize $P(w_i|w_{i-1}) = P(c_i|c_{i-1})P(w_i|c_i)$
- Useful features for tasks like NER, not suitable for NNs

Brown et al. (1992)

Discrete Word Representations

- Brown clusters: hierarchical agglomerative *hard* clustering
- We give a very brief sketch of the algorithm here:

- k: a hyper-parameter, sort words by frequency
- Take the top k most frequent words, put each of them in its own cluster $c_1, c_2, \; c_3, \; \dots c_k$
- For $i = \left(k+1\right) \dots |V|$
  - Create a new cluster c_{k+1} *(we have $k+1$ clusters)*
  - Choose two clusters from $k+1$ clusters based on quality(C) and merge *(back to k clusters)*

$$Quality(C) = \sum_{i}^{n} \log e(w_i \mid C(w_i)) q(C(w_i) \mid C(w_{i-1})) = \sum_{c=1}^{k} \sum_{c'=1}^{k} p(c,c') \log \frac{p(c,c')}{p(c)p(c')} + G$$

mutual information between adjacent clusters

entropy of the word distribution

- Carry out $k-1$ final merges *(full hierarchy)*
- Running time $O\left(\left|V\right| k^2 + n\right)$, n=#words in corpus

Learn more: Percy Liang's phd thesis - Semi-Supervised Learning for Natural Language

Discrete Word Representations

- Brown clusters: hierarchical agglomerative *hard* clustering
- Example Clusters from Miller et al. 2004

mailman	10000011010111
salesman	100000110110000
bookkeeper	1000001101100010
troubleshooter	10000011011000110
bouncer	10000011011000111
technician	1000001101100100
janitor	1000001101100101
saleswoman	1000001101100110
...	
Nike	1011011100100101011100
Maytag	10110111001001010111010
Generali	10110111001001010111011
Gap	1011011100100101011110
Harley-Davidson	10110111001001010111110
Enfield	101101110010010101111110
genus	101101110010010101111111
Microsoft	10110111001001011000
Ventritex	101101110010010110010
Tractebel	1011011100100101100110
Synopsys	1011011100100101100111
WordPerfect	1011011100100101101000
....	
John	101110010000000000
Consuelo	101110010000000001
Jeffrey	101110010000000010
Kenneth	10111001000000001100
Phillip	101110010000000011010
WILLIAM	101110010000000011011
Timothy	10111001000000001110

word cluster features (bit string prefix)

NER in Twitter

2m 2ma 2mar 2mara 2maro 2marrow 2mor 2mora 2moro 2morow 2morr 2morro 2morrow 2moz 2mr 2mro 2mrrw 2mrw 2mw tmmrw tmo tmoro tmorrow tmoz tmr tmro tmrow tmrrow tmrrw tmrw tmrww tmw tomaro tomarow tomarro tomarrow tomm tommarow tommarrow tommoro tommorow tommorrow tommorw tommrow tomo tomolo tomoro tomorow tomorro tomorrw tomoz tomrw tomz

Word2vec

Both

System	Fin10Dev	Rit11	Fro14	Avg
CoNLL	27.3	27.1	29.5	28.0
+ Brown	38.4	39.4	42.5	40.1
+ Vector	40.8	40.4	42.9	41.4
+ Reps	42.4	42.2	46.2	43.6
Fin10	36.7	29.0	30.4	32.0
+ Brown	59.9	53.9	56.3	56.7
+ Vector	61.5	56.4	58.4	58.8
+ Reps	64.0	58.5	60.2	60.9
CoNLL+Fin10	44.7	39.9	44.2	42.9
+ Brown	54.9	52.9	58.5	55.4
+ Vector	58.9	55.2	59.9	58.0
+ Reps	58.9	56.4	61.8	59.0
+ Weights	64.4	59.6	63.3	62.4

Table 5: Impact of our components on Twitter NER performance, as measured by F1, under 3 data scenarios.

Ritter et al. (2011)

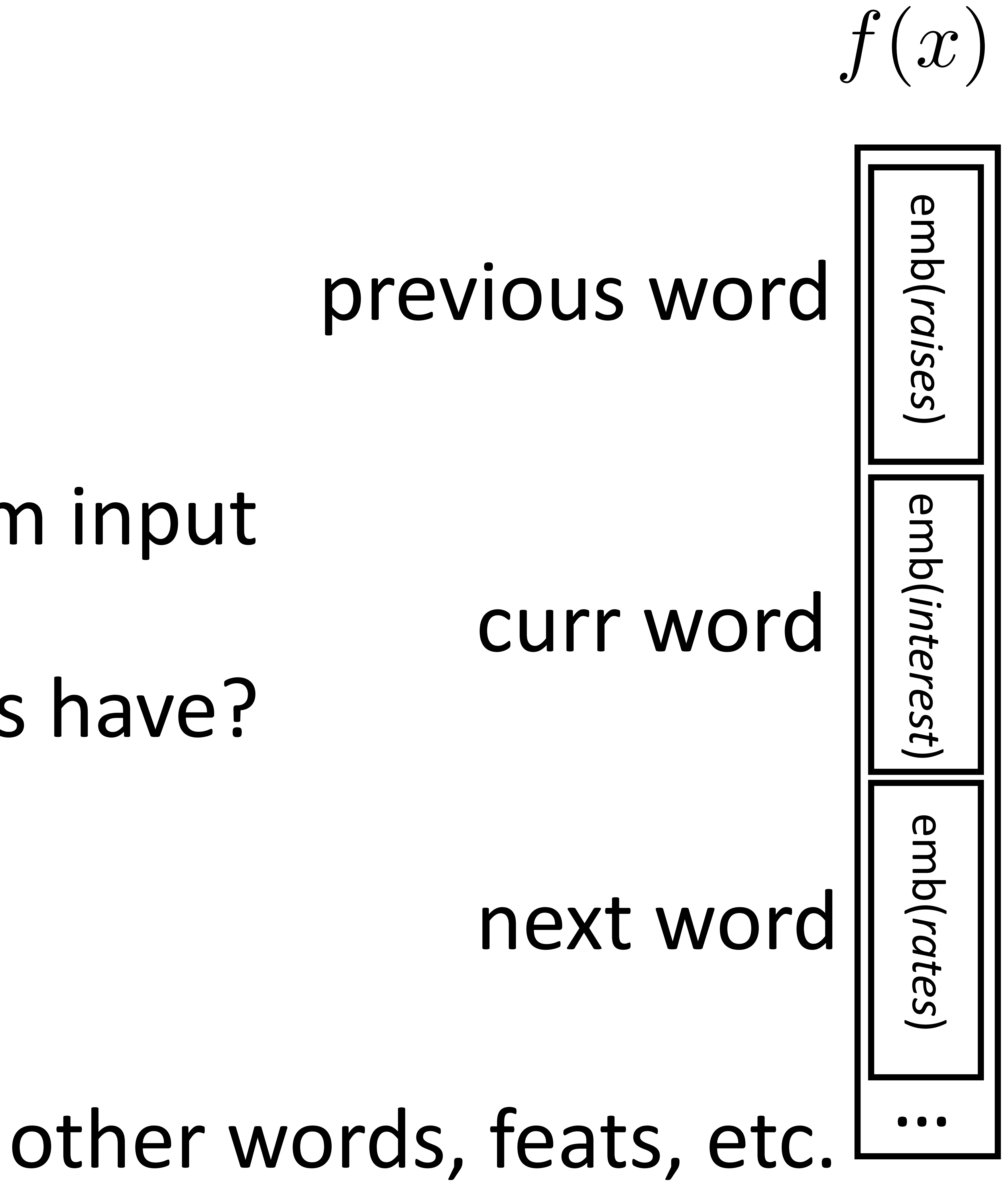
Cherry & Guo (2015)

Word Embeddings

- Part-of-speech tagging with FFNNs

??

*Fed raises **interest** rates in order to …*



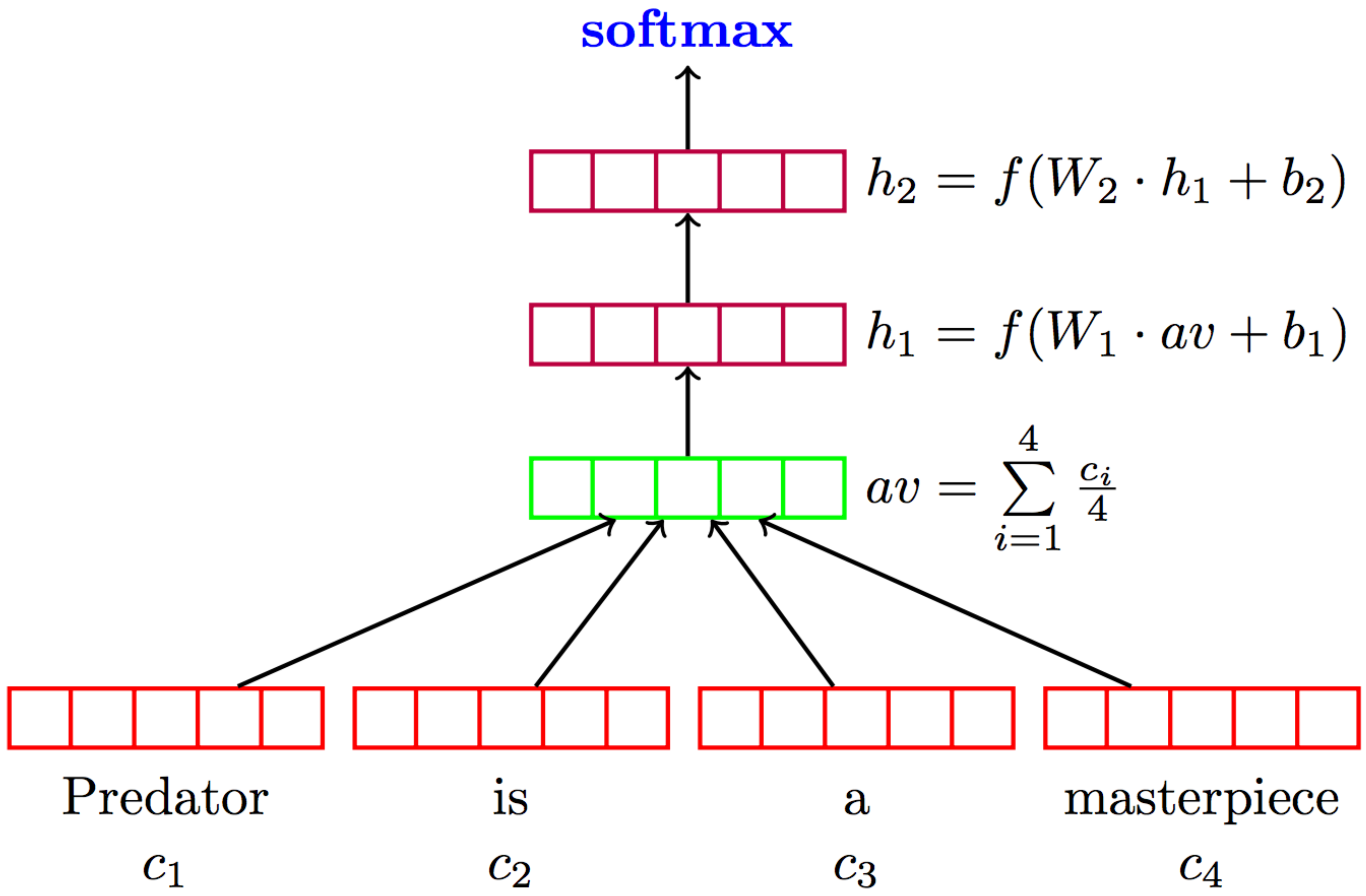
- Word embeddings for each word form input

- What properties should these vectors have?

Botha et al. (2017)

Sentiment Analysis

- Deep Averaging Networks: feedforward neural network on average of word embeddings from input



Iyyer et al. (2015)

Word Embeddings

- Want a vector space where similar words have similar embeddings

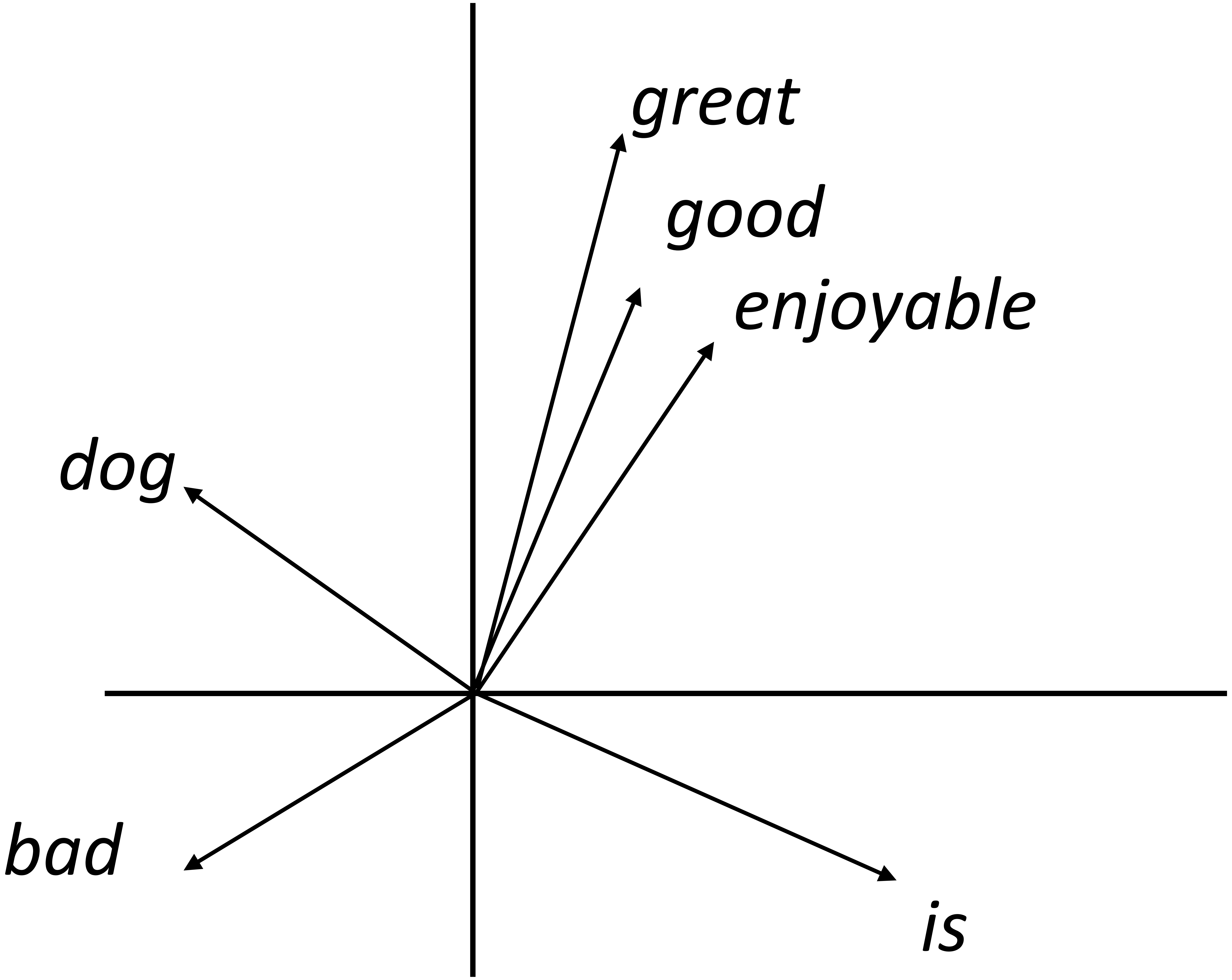
the movie was great

≈

the movie was good

- Goal: come up with a way to produce these embeddings

- For each word, want "medium" dimensional vector (50-300 dims) representing it.



Word Representations

- Count-based: tf*idf, PPMI, …
- Class-based: Brown Clusters, …
- Distributed prediction-based embeddings: Word2vec, GloVe, FastText, …
- Distributed contextual embeddings: ELMo, BERT, GPT, …
- + many more variants: multi-sense embeddings, syntactic embeddings, …

word2vec/GloVe

Neural Probabilistic Language Model

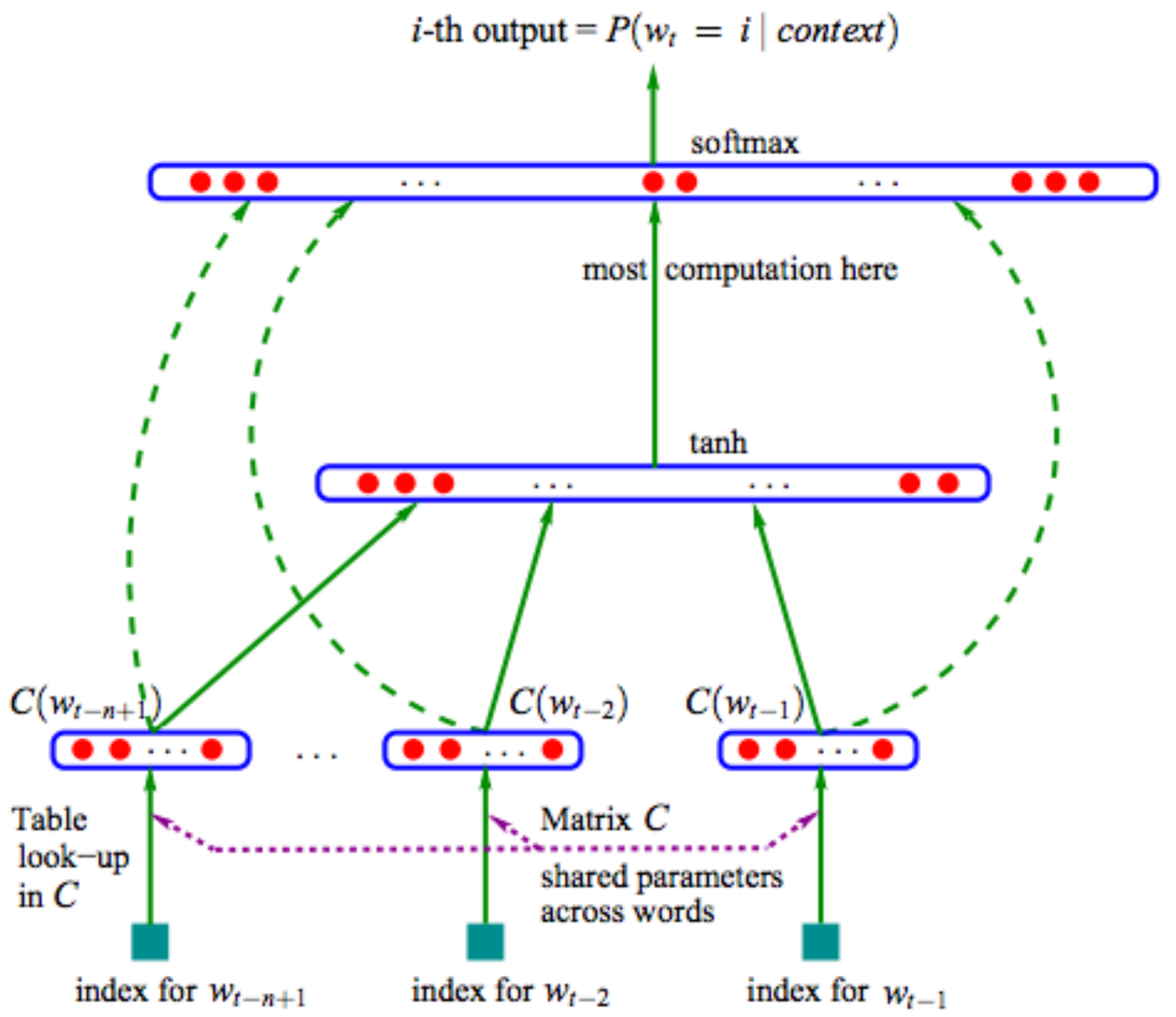
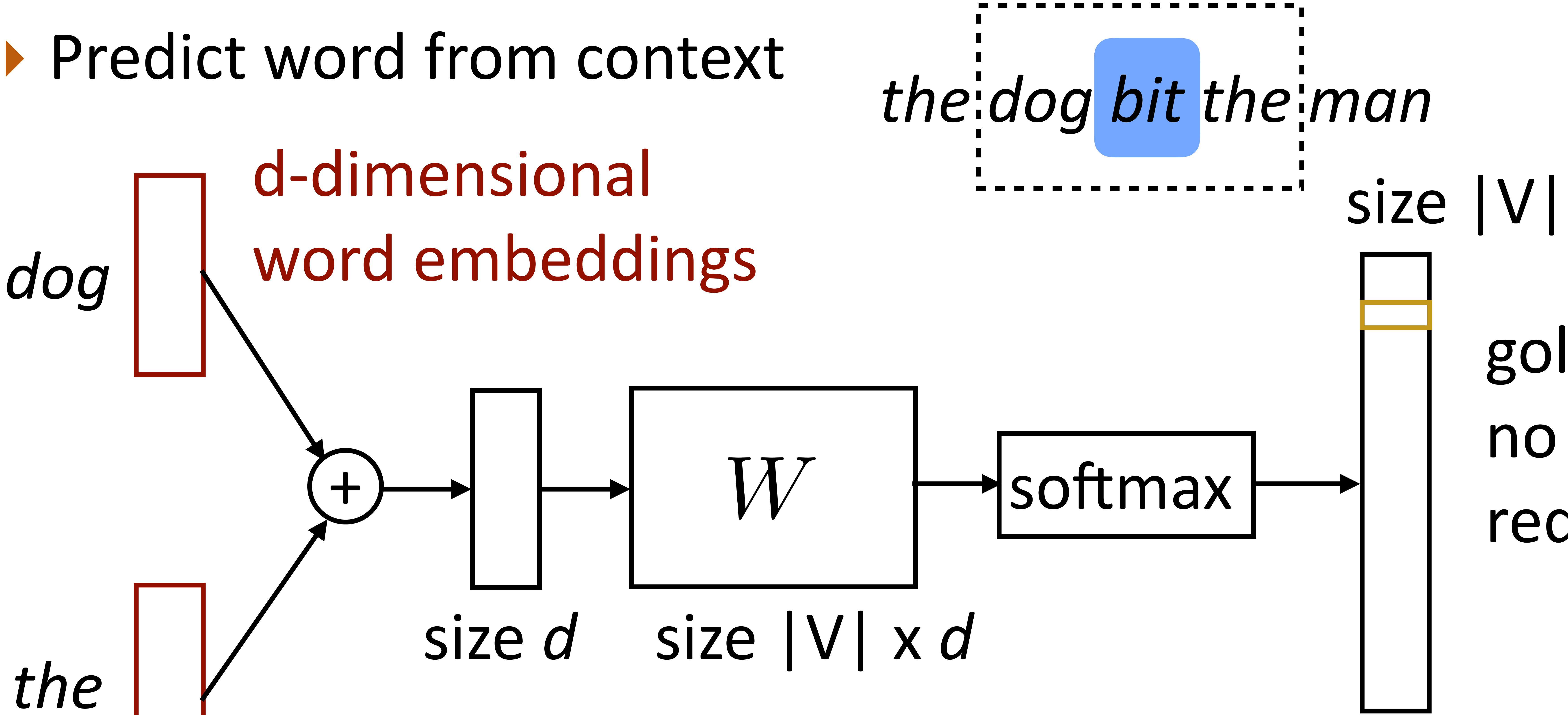


Figure 1: Neural architecture: $f(i, w_{t-1}, \cdots, w_{t-n+1}) = g(i, C(w_{t-1}), \cdots, C(w_{t-n+1}))$ where g is the neural network and $C(i)$ is the i-th word feature vector.

Bengio et al. (2003)

word2vec: Continuous Bag-of-Words

- Predict word from context



gold label = *bit*, no manual labeling required!

$$P(w|w_{-1}, w_{+1}) = \text{softmax}\left(W(c(w_{-1}) + c(w_{+1}))\right)$$

- Parameters: *d* x |V| (one *d*-length context vector per voc word), |V| x *d* output parameters (W)

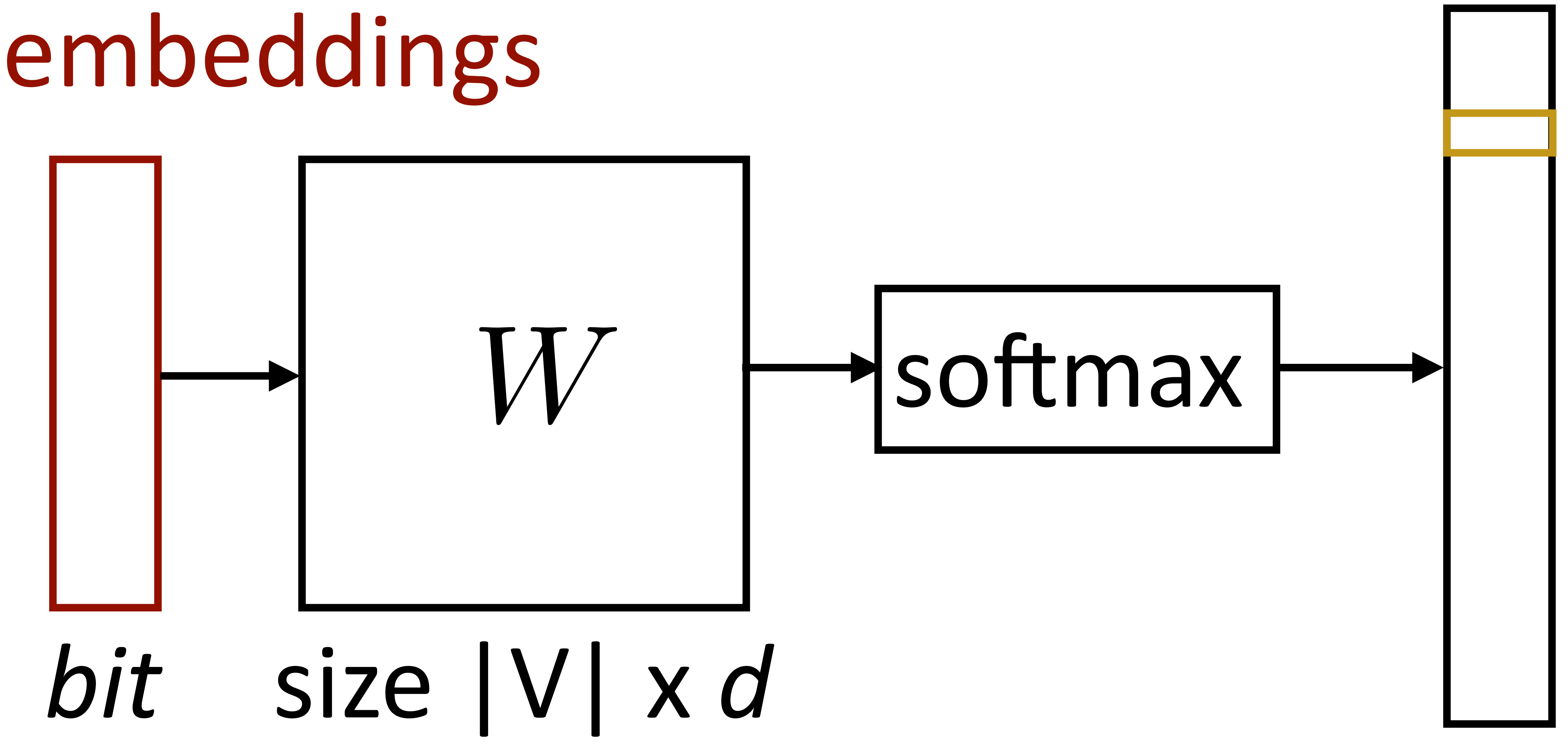
Mikolov et al. (2013)

word2vec: Skip-Gram

- Predict one word of context from word

the dog bit the man

d-dimensional
word embeddings



gold label = *dog*

$$P(w'|w) = \text{softmax}(We(w))$$

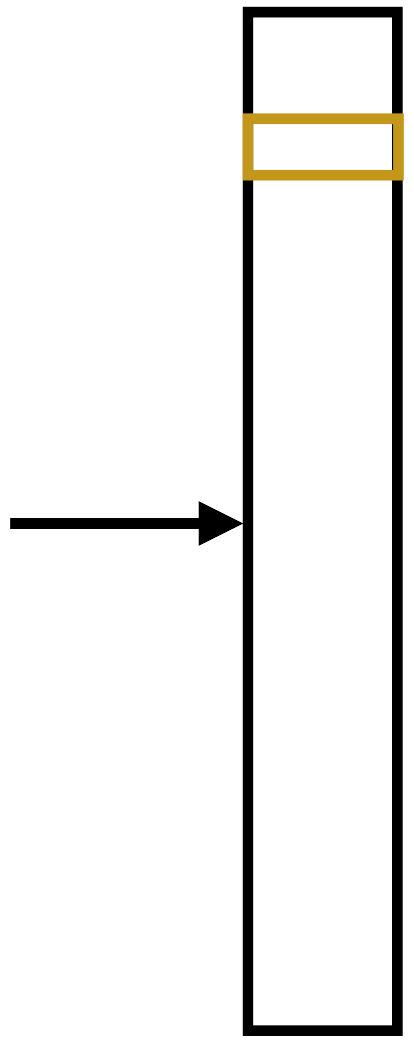
- Another training example: *bit -> the*
- Parameters: *d* x |V| vectors, |V| x *d* output parameters (W) (also usable as vectors!)

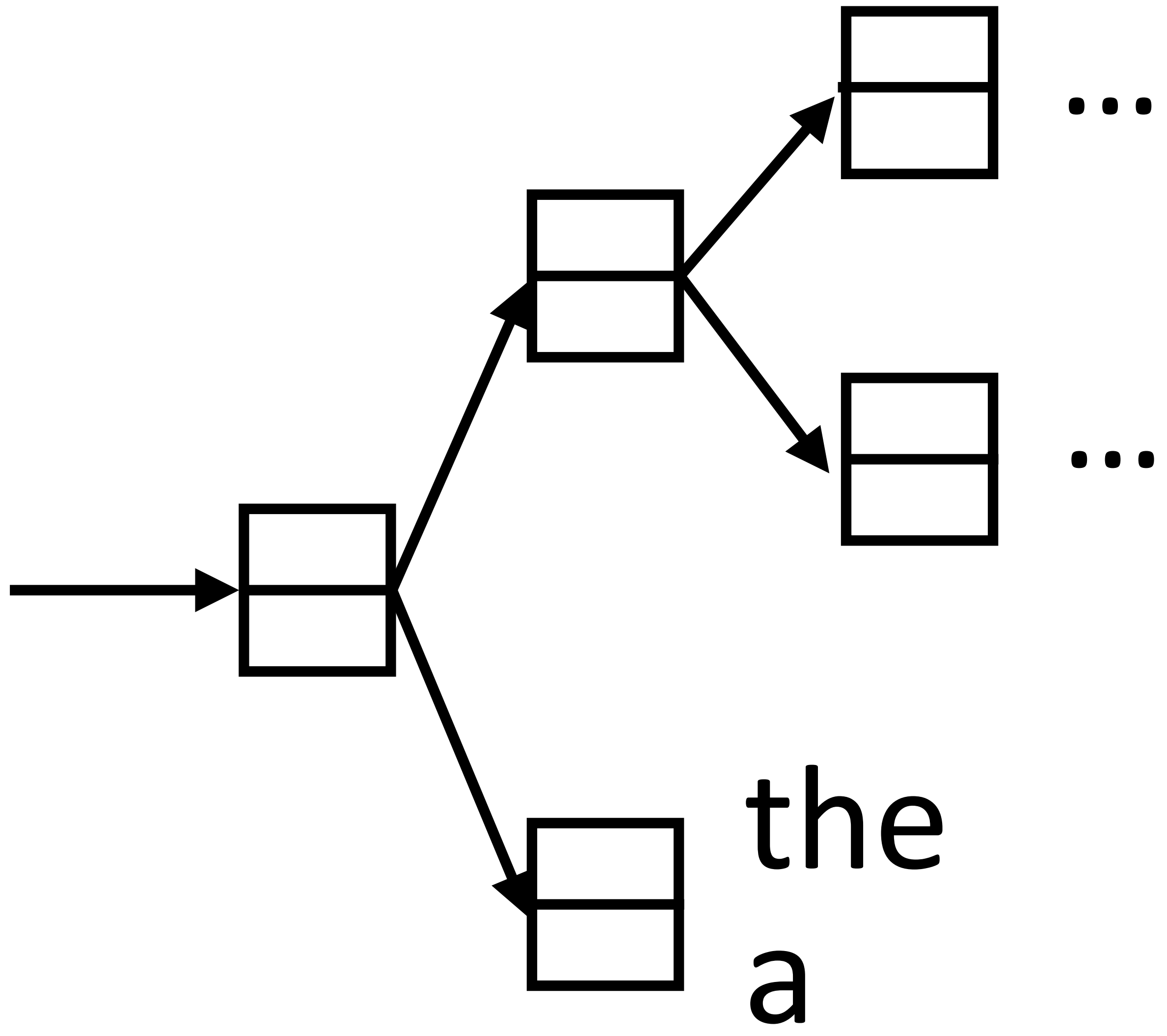
Mikolov et al. (2013)

Hierarchical Softmax

$$P(w|w_{-1}, w_{+1}) = \text{softmax}\left(W(c(w_{-1}) + c(w_{+1}))\right) \qquad P(w'|w) = \text{softmax}(We(w))$$

- Matmul + softmax over |V| is very slow to compute for CBOW and SG





- Huffman encode vocabulary, use binary classifiers to decide which branch to take
- log(|V|) binary decisions

- Standard softmax:
  O(|V|) dot products of size *d*
  - per training instance per context word

- Hierarchical softmax:
  O(log(|V|)) dot products of size *d*,
  |V| x *d* parameters

http://building-babylon.net/2017/08/01/hierarchical-softmax/

Mikolov et al. (2013)

Skip-Gram with Negative Sampling

- Take (word, context) pairs and classify them as “real” or not. Create random negative examples by sampling from unigram distribution

(*bit*, *the*) => +1

(*bit*, *cat*) => -1

(*bit*, *a*) => -1

(*bit*, *fish*) => -1

the $\boxed{\textit{dog}}$ $\boxed{\textit{bit}}$ *the man*

$$P(y=1|w,c) = \frac{e^{w \cdot c}}{e^{w \cdot c} + 1}$$

words in similar contexts select for similar *c* vectors

- *d* x |V| vectors, *d* x |V| context vectors (same # of params as before)

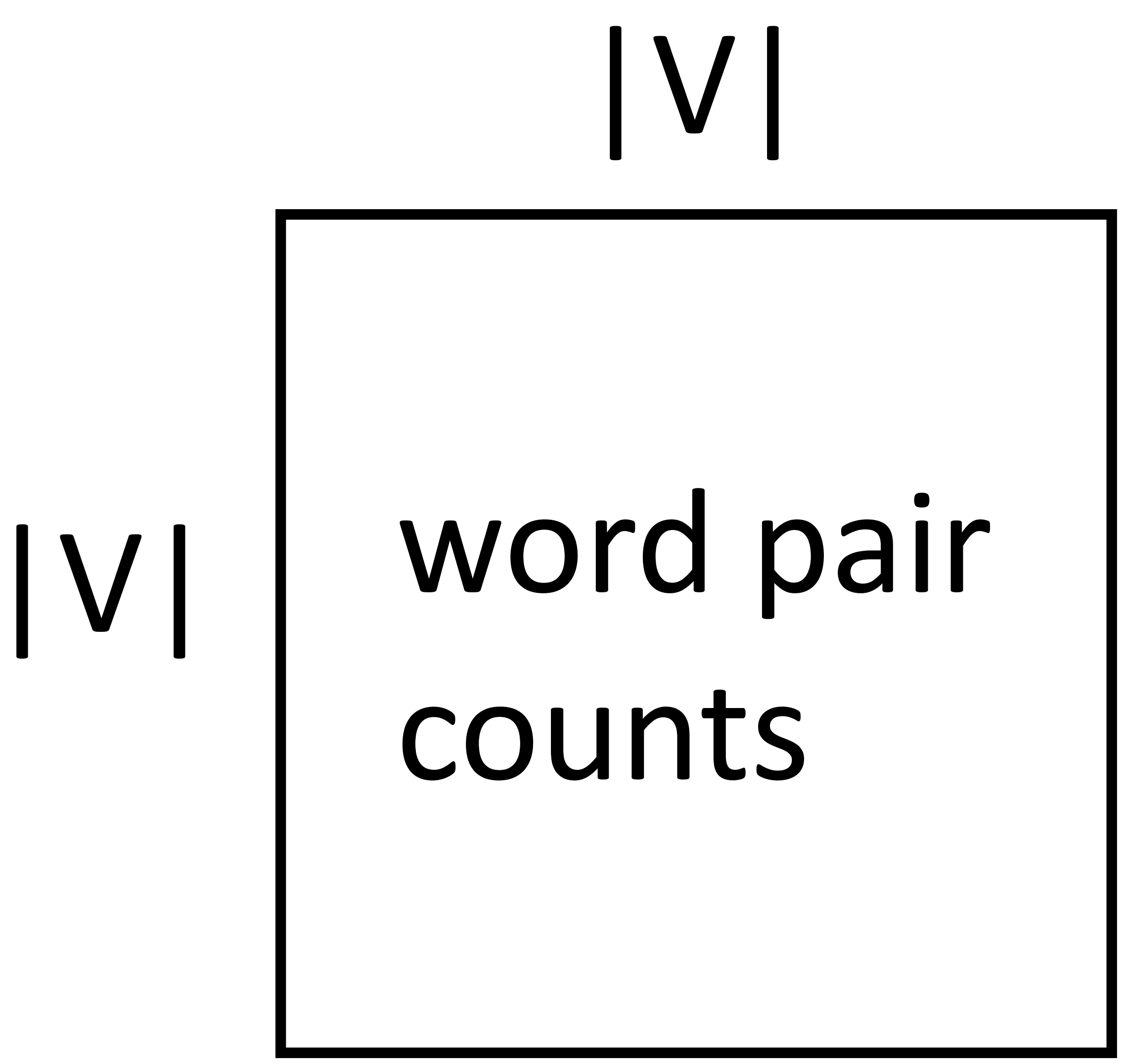
- Objective = $\log P(y=1|w,c) - \sum_{i=1}^{k} \log P(y=0|w_i,c)$

sampled

Mikolov et al. (2013)

Connections with Matrix Factorization

- Skip-gram model looks at word-word co-occurrences and produces two types of vectors

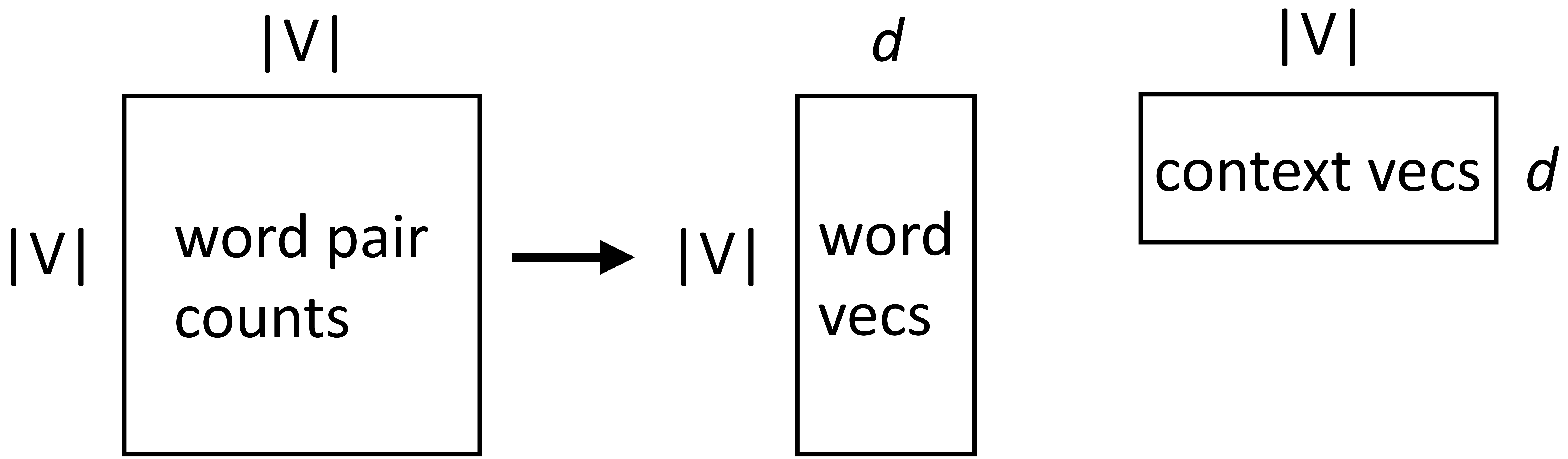


	knife	dog	sword	love	like
knife	0	1	6	5	5
dog	1	0	5	5	5
sword	6	5	0	5	5
love	5	5	5	0	5
like	5	5	5	5	2

Two words are “similar” in meaning if their context vectors are similar. Similarity == relatedness

Connections with Matrix Factorization

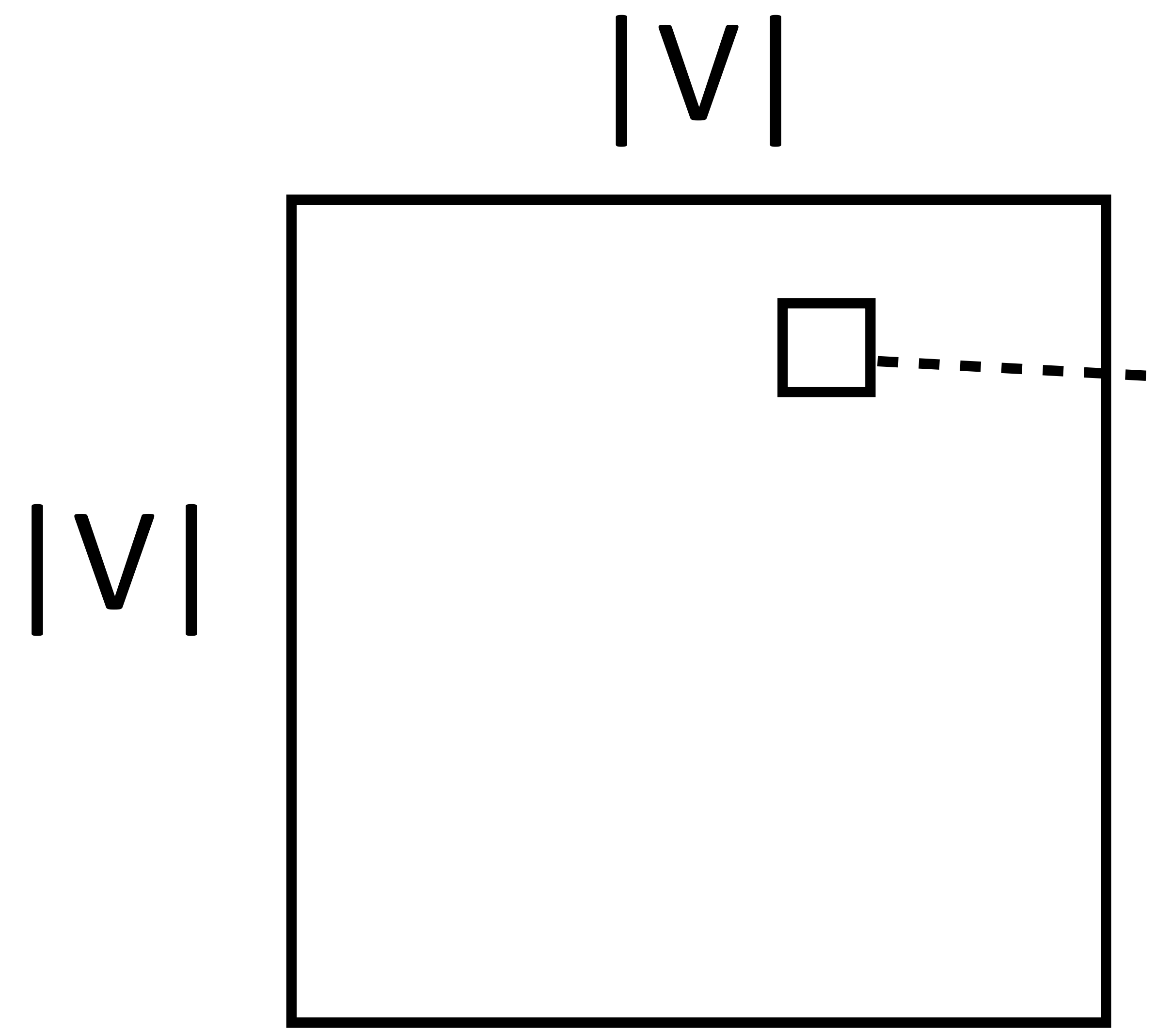
- Skip-gram model looks at word-word co-occurrences and produces two types of vectors



- Looks almost like a matrix factorization…can we interpret it this way?

Levy et al. (2014)

Skip-Gram as Matrix Factorization



num negative samples

$$M_{ij} = \mathrm{PMI}(w_i, c_j) - \log k$$

$$\mathrm{PMI}(w_i, c_j) = \frac{P(w_i, c_j)}{P(w_i)P(c_j)} = \frac{\frac{\mathrm{count}(w_i, c_j)}{D}}{\frac{\mathrm{count}(w_i)}{D}\,\frac{count(c_j)}{D}}$$

Skip-gram objective *exactly* corresponds to factoring this matrix:

- *If* we sample negative examples from the uniform distribution over words
- …and it's a *weighted* factorization problem (weighted by word freq)

Levy et al. (2014)

Co-occurence Matrix

- Typical problems in word-word co-occurrences:
  - Raw frequency is not the best measure of association between words.
  - Frequent words are often more important than rare words that only appear once or twice;
  - But, frequent words (e.g., *the*) that appear in all documents are also not very useful signal.
- Solutions — weighing terms in word-word/word-doc co-occurrence matrix
  - Tf*idf
  - PPMI (Positive PMI)

Co-occurence Matrix

- Tf*idf
  - Tf: term frequency

$$tf = \log_{10}(\text{count}(t,\ d) + 1)$$

  - Idf: inverse document frequency

$$idf_i = \log_{10}\left(\frac{N}{df_i}\right)$$

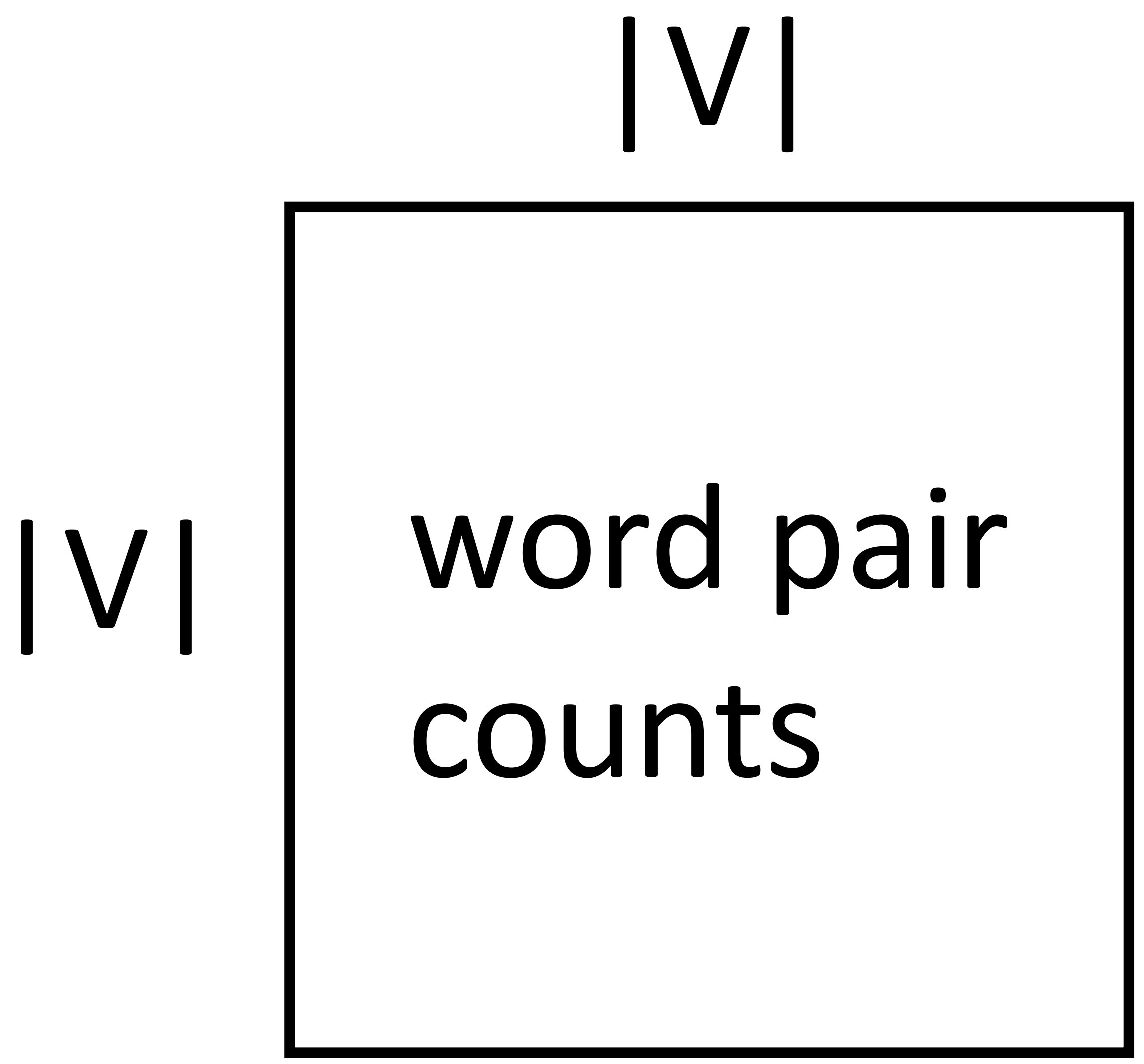
Total number of docs in collection

number of docs that have word i

word-doc co-occurrences

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	17
solider	2	80	62	89
fool	36	58	1	4
clown	20	15	2	3

GloVe (**Glo**bal **Ve**ctors)



- Also operates on counts matrix, weighted regression on the log co-occurrence matrix

- Loss $= \sum_{i,j} f(\mathrm{count}(w_i, c_j)) \left(w_i^\top c_j + a_i + b_j - \log \mathrm{count}(w_i, c_j)\right)^2$

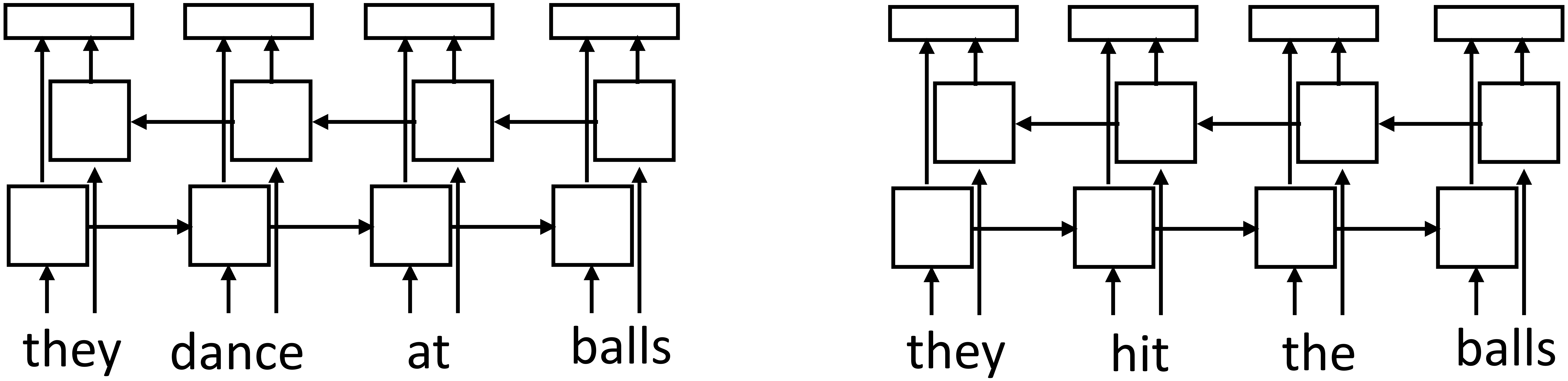
- Constant in the dataset size (just need counts), quadratic in voc size

- By far the most common non-contextual word vectors used today (10000+ citations)

Pennington et al. (2014)

Preview: Context-dependent Embeddings

- How to handle different word senses? One vector for *balls*

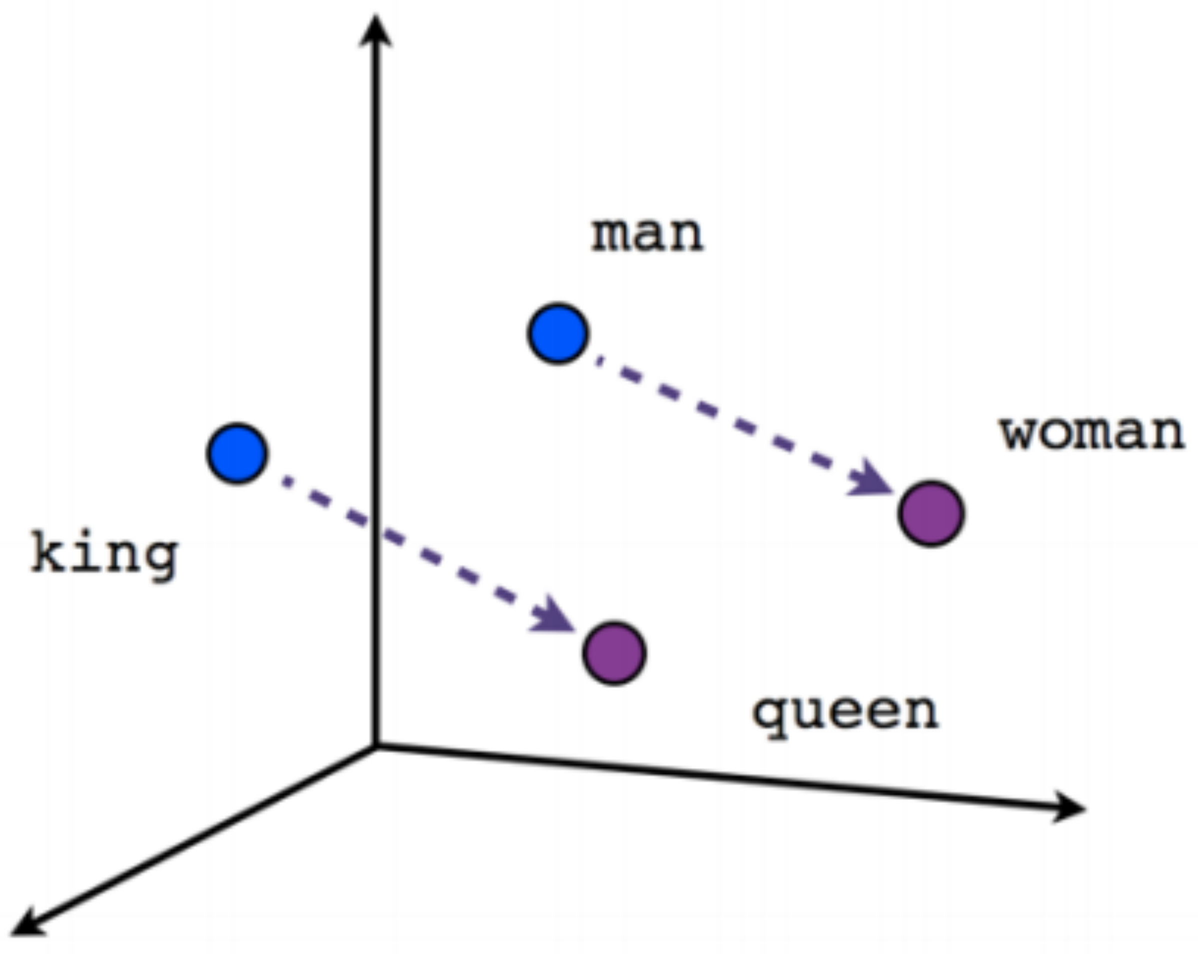


- Train a neural language model to predict the next word given previous words in the sentence, use its internal representations as word vectors
- *Context-sensitive* word embeddings: depend on rest of the sentence
- *Huge* improvements across nearly all NLP tasks over word2vec & GloVe

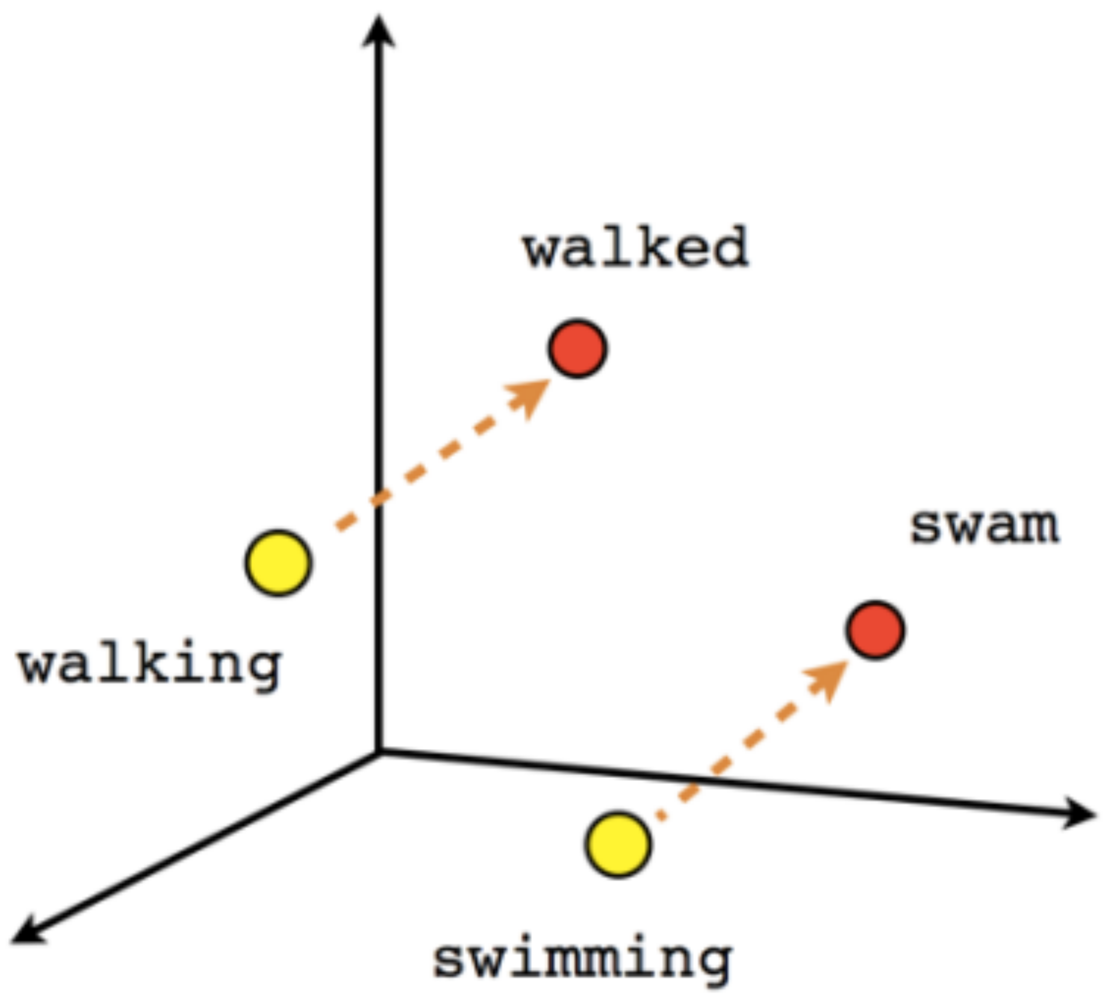
Peters et al. (2018)

Evaluation

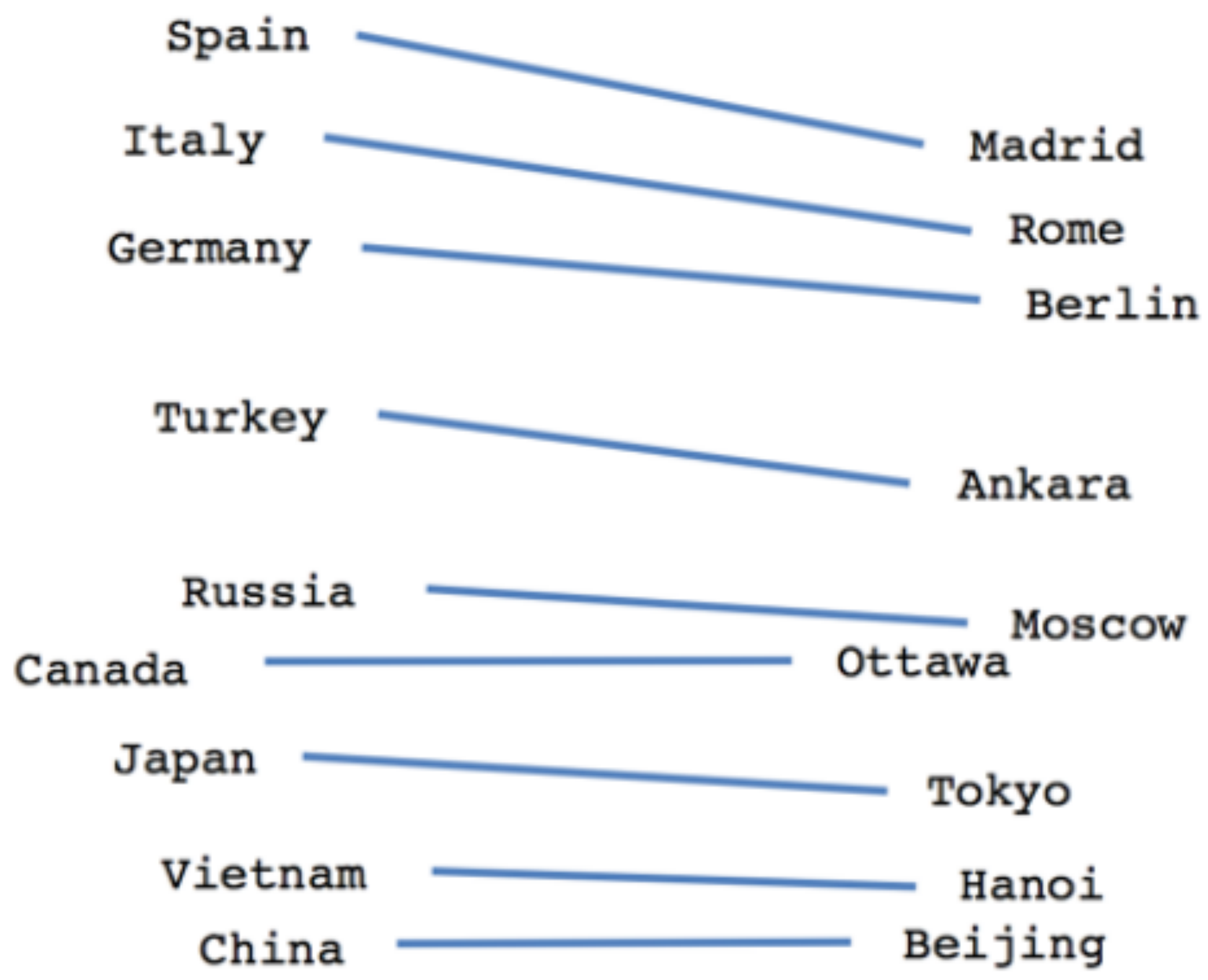
Visualization



Male-Female



Verb tense



Country-Capital

Source: tensorflow.org

Visualization

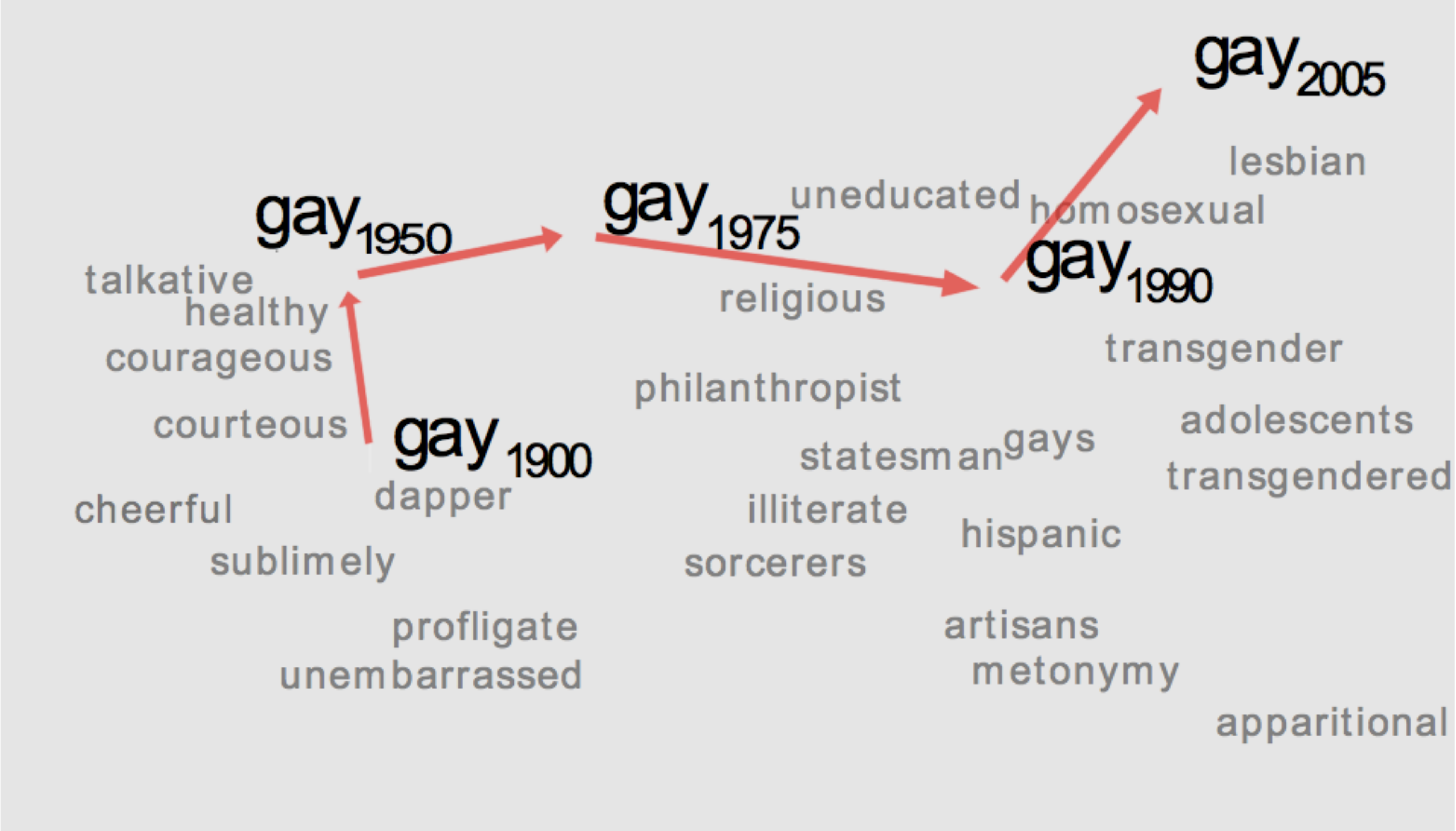


Figure 1: A 2-dimensional projection of the latent semantic space captured by our algorithm. Notice the semantic trajectory of the word **gay** transitioning meaning in the space.

Kulkarni et al. (2015)

Evaluating Word Embeddings

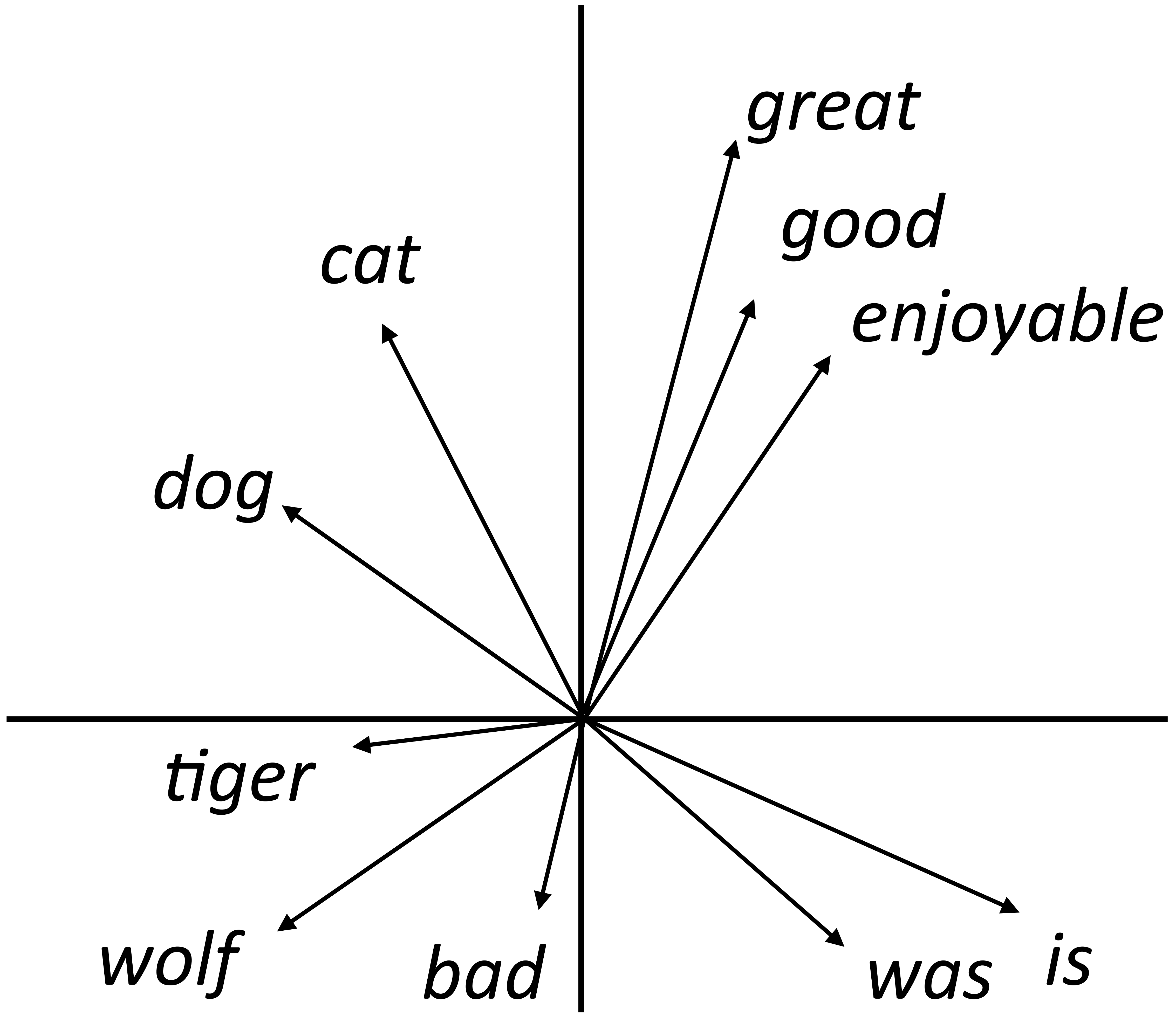
- What properties of language should word embeddings capture?

- Similarity: similar words are close to each other

- Analogy:

  good is to best as smart is to ???

  Paris is to France as Tokyo is to ???



Word Similarity

- Cosine Similarity:

$$\text{cosine}(\vec{v}, \vec{w}) = \frac{\vec{v} \cdot \vec{w}}{|\vec{v}|\,|\vec{w}|} = \frac{\sum_{i=1}^{N} v_i w_i}{\sqrt{\sum_{i=1}^{N} v_i^2}\sqrt{\sum_{i=1}^{N} w_i^2}}$$

Levy et al. (2015)

Word Similarity

Method	WordSim Similarity	WordSim Relatedness	Bruni et al. MEN	Radinsky et al. M. Turk	Luong et al. Rare Words	Hill et al. SimLex
PPMI	.755	**.697**	.745	.686	.462	.393
SVD	**.793**	.691	**.778**	.666	**.514**	.432
SGNS	**.793**	.685	.774	**.693**	.470	**.438**
GloVe	.725	.604	.729	.632	.403	.398

Word2vec

- SVD = singular value decomposition on PMI matrix
- GloVe does not appear to be the best when experiments are carefully controlled, but it depends on hyperparameters + these distinctions don't matter in practice

Levy et al. (2015)

Hypernymy Detection

- Hypernyms: detective *is a* person, dog *is a* animal
- Do word vectors encode these relationships?

Dataset	TM14	Kotlerman 2010	HypeNet	WordNet	**Avg (10 datasets)**
Random	52.0	30.8	24.5	55.2	23.2
Word2Vec + C	52.1	**39.5**	20.7	**63.0**	25.3
GE + C	53.9	36.0	21.6	58.2	26.1
GE + KL	52.0	39.4	23.7	54.4	25.9
DIVE + C$\cdot\Delta$S	**57.2**	36.6	**32.0**	60.9	**32.7**

- word2vec (SGNS) works barely better than random guessing here

Table 1: Comparison with other unsupervised embedding methods. The scores are AP@all (%) for the first 10 datasets and Spearman ρ (%) for HyperLex. Avg (10 datasets) shows the micro-average AP of all datasets except HyperLex. Word2Vec+C scores word pairs using cosine similarity on skip-grams. GE+C and GE+KL compute cosine similarity and negative KL divergence on Gaussian embedding, respectively.

Chang et al. (2017)

Analogies

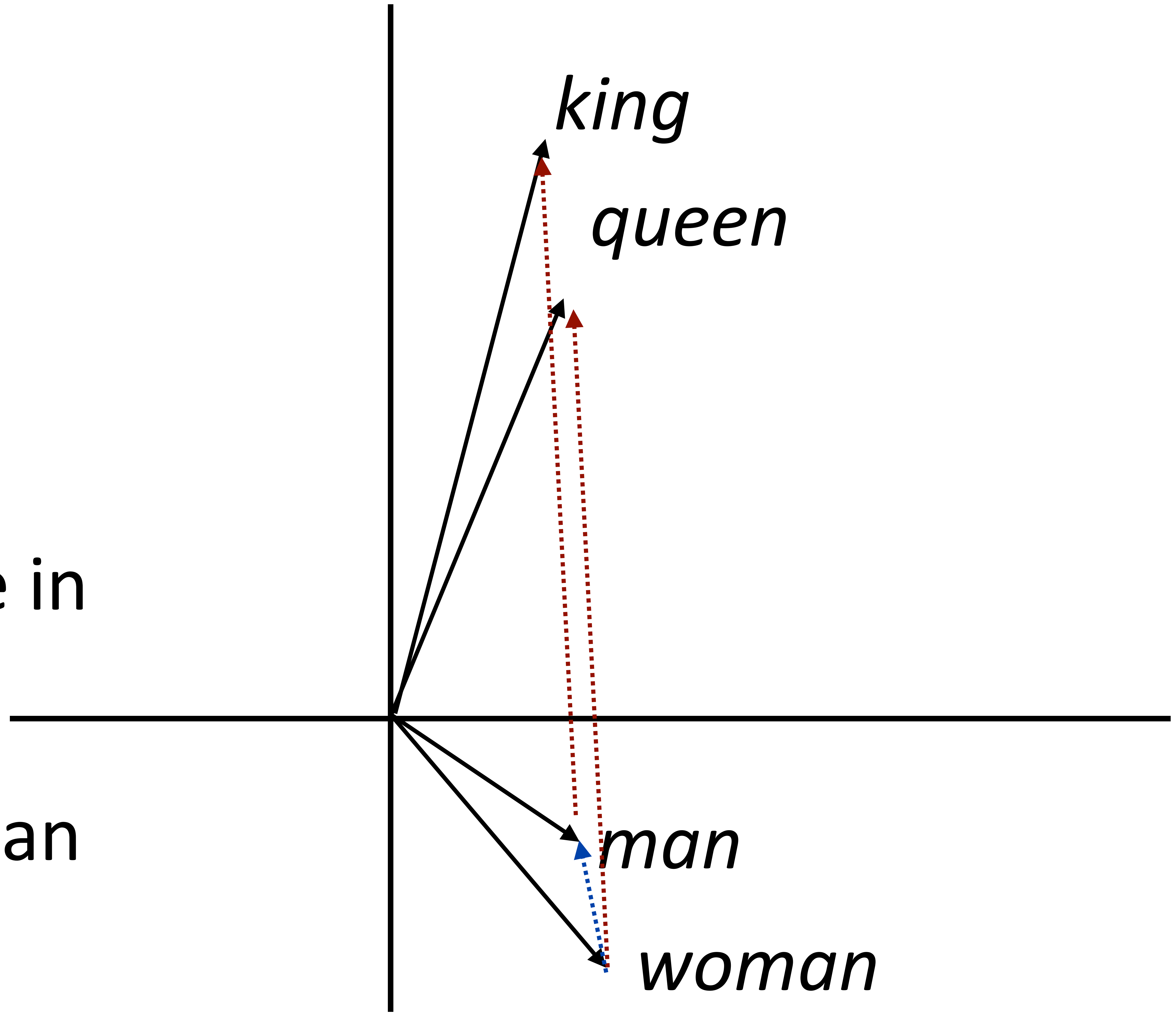
(king - man) + woman = queen

king + (woman - man) = queen

- Why would this be?

- woman - man captures the difference in the contexts that these occur in

- Dominant change: more "he" with man and "she" with woman — similar to difference between king and queen



Analogies

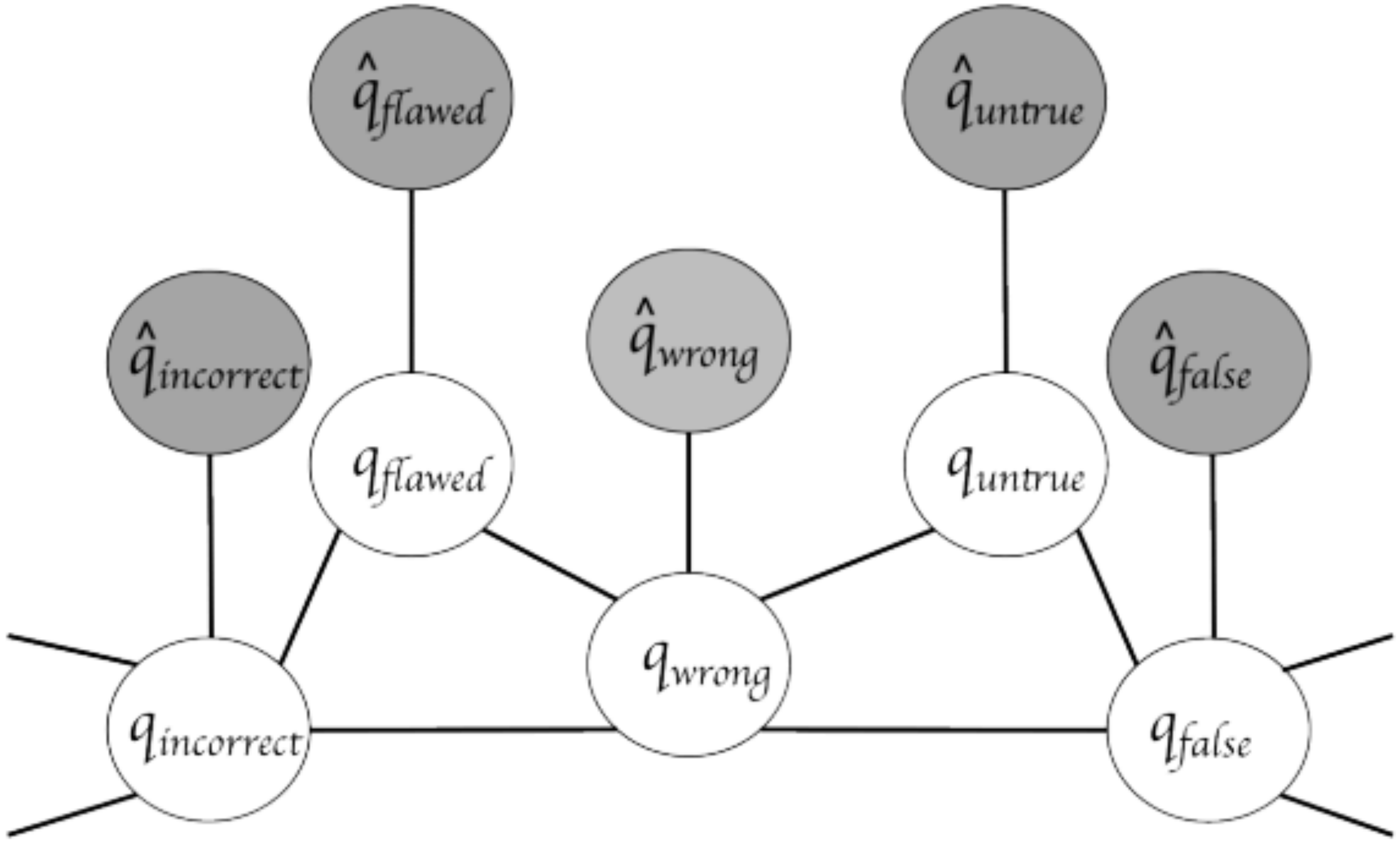
Method	Google Add / Mul	MSR Add / Mul
PPMI	.553 / .679	.306 / .535
SVD	.554 / .591	.408 / .468
SGNS	.676 / **.688**	.618 / **.645**
GloVe	.569 / .596	.533 / .580

- These methods can perform well on analogies on two different datasets using two different methods

Maximizing for *b*: Add = $\cos(b, a_2 - a_1 + b_1)$ Mul = $\dfrac{\cos(b_2, a_2)\cos(b_2, b_1)}{\cos(b_2, a_1) + \epsilon}$

Levy et al. (2015)

Using Semantic Knowledge



Original vector for *false*

Adapted vector for *false*

- Structure derived from a resource like WordNet
- Doesn't help most problems

Faruqui et al. (2015)

Using Word Embeddings

- Approach 1: learn embeddings as parameters from your data
  - Often works pretty well
- Approach 2: initialize using GloVe/word2vec/ELMo, keep fixed
  - Faster because no need to update these parameters
- Approach 3: initialize using GloVe, fine-tune
  - Works best for some tasks, not used for ELMo, often used for BERT

Takeaways

- Lots to tune with neural networks
  - Training: optimizer, initializer, regularization (dropout), …
  - Hyperparameters: dimensionality of word embeddings, layers, …
- Word vectors: learning word -> context mappings has given way to matrix factorization approaches (constant in dataset size)
- Lots of pretrained embeddings work well in practice, they capture some desirable properties
- Even better: context-sensitive word embeddings (ELMo/BERT)
- Next time: RNNs and CNNs