Lecture 7: Tricks + Word Embeddings

Alan Ritter

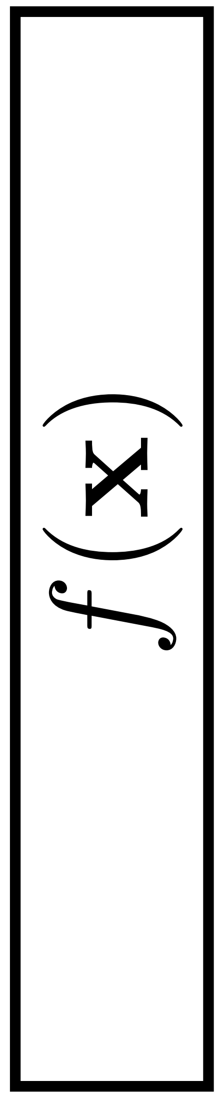
(many slides from Greg Durrett)

Recall: Feedforward NNs

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

Recall: Feedforward NNs

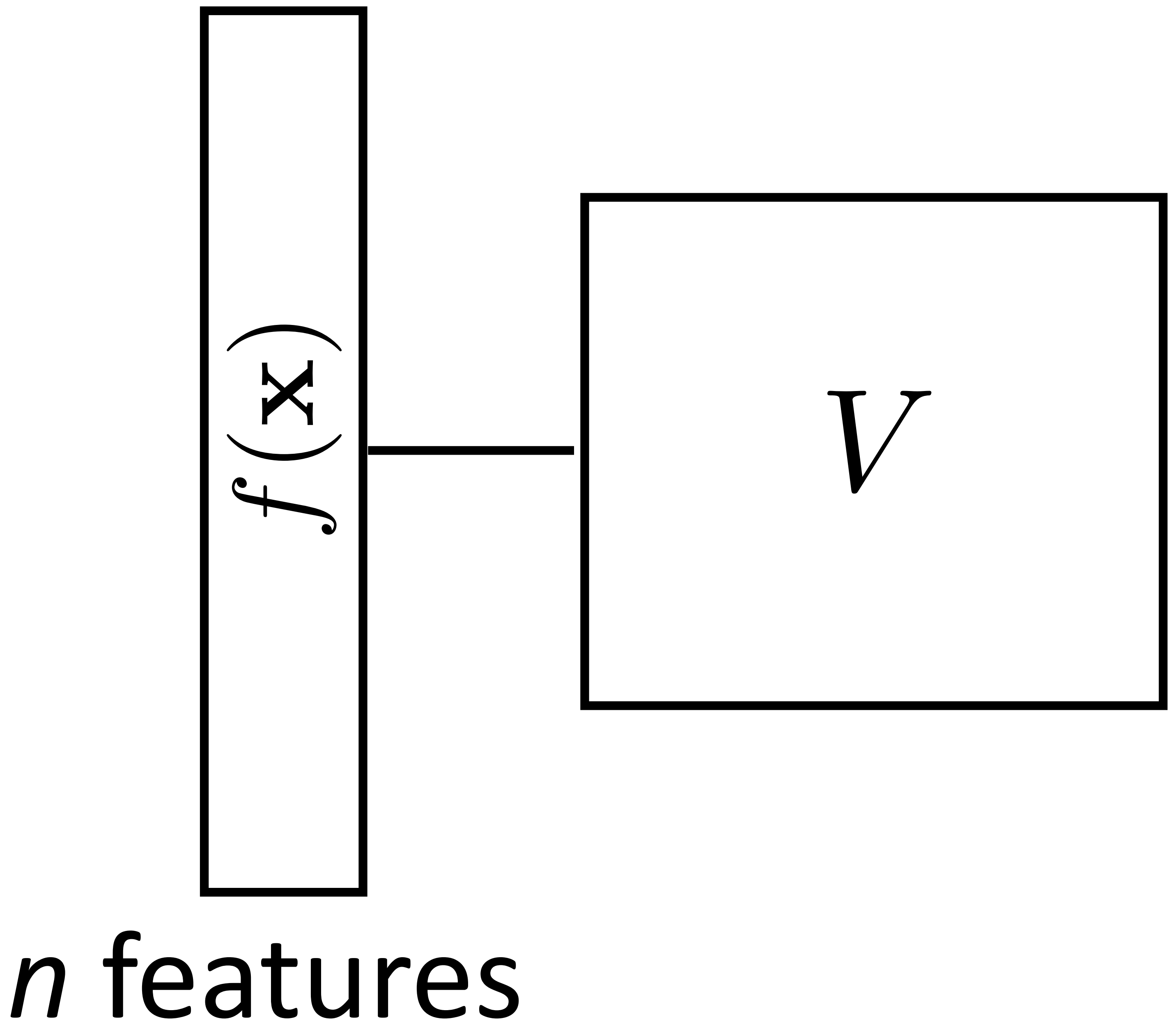
$$P(\mathbf{y}|\mathbf{x}) = \mathrm{softmax}(W g(V f(\mathbf{x})))$$



n features

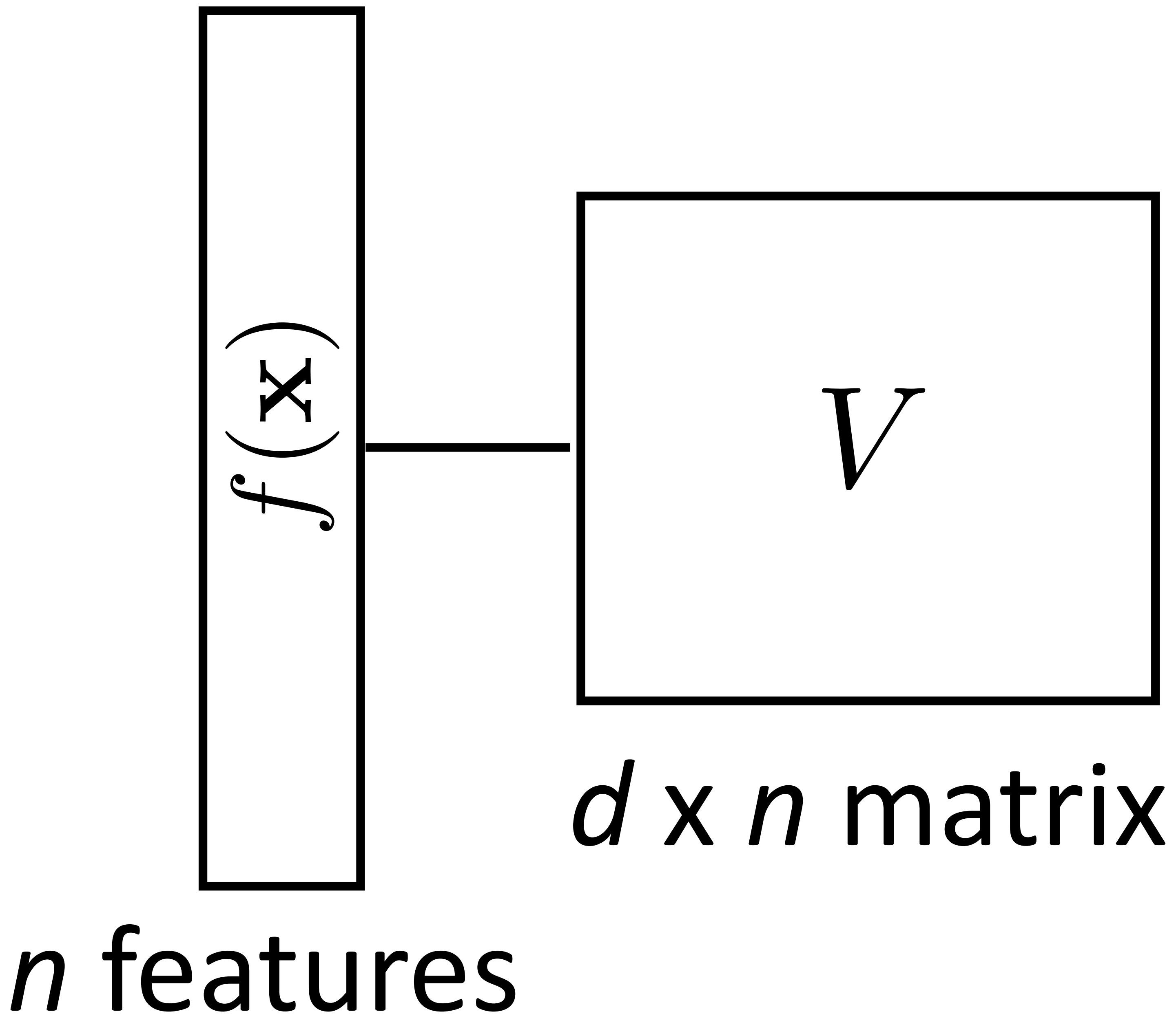
Recall: Feedforward NNs

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



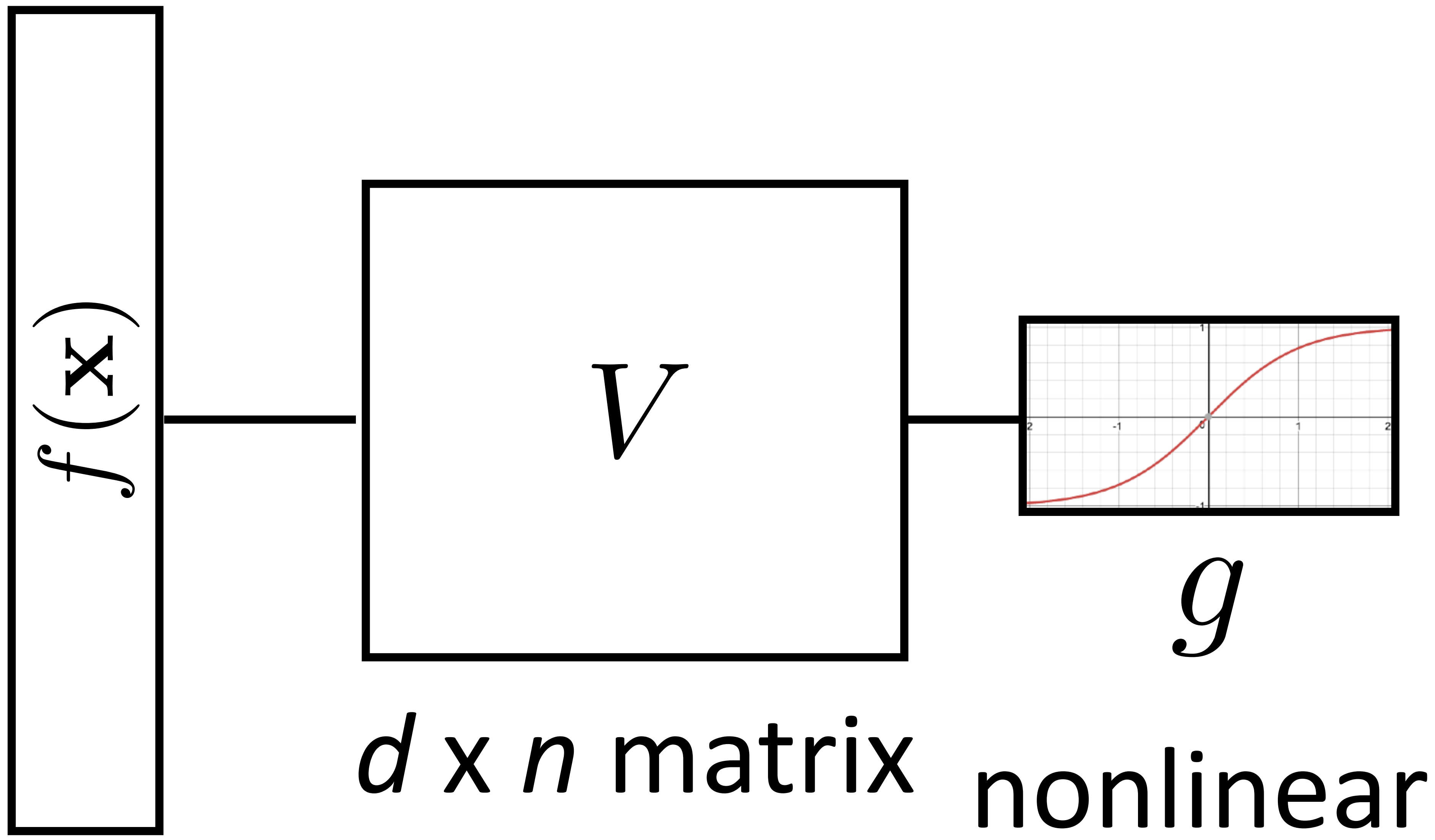
Recall: Feedforward NNs

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



Recall: Feedforward NNs

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



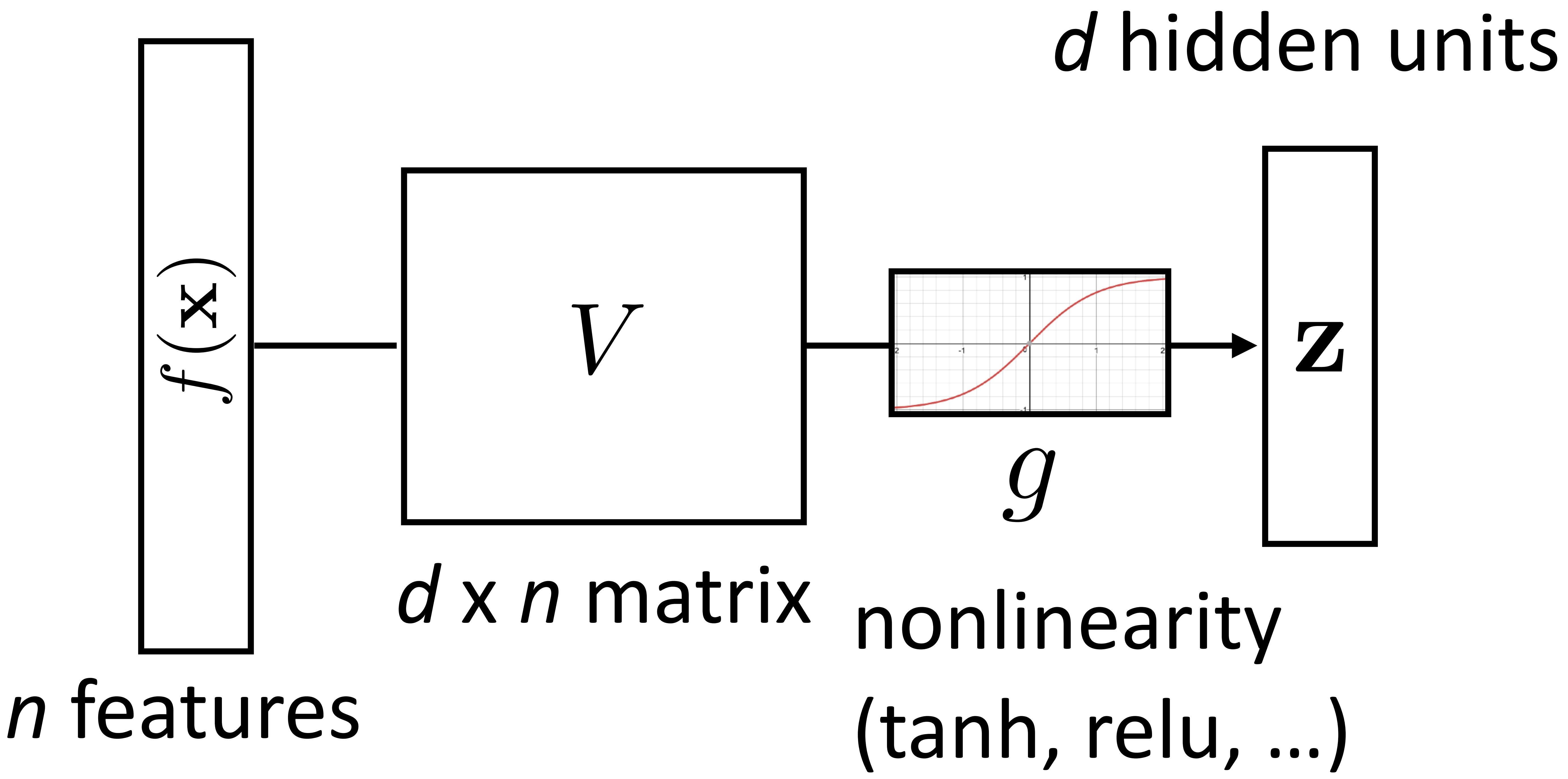
n features

d x *n* matrix

nonlinearity
(tanh, relu, …)

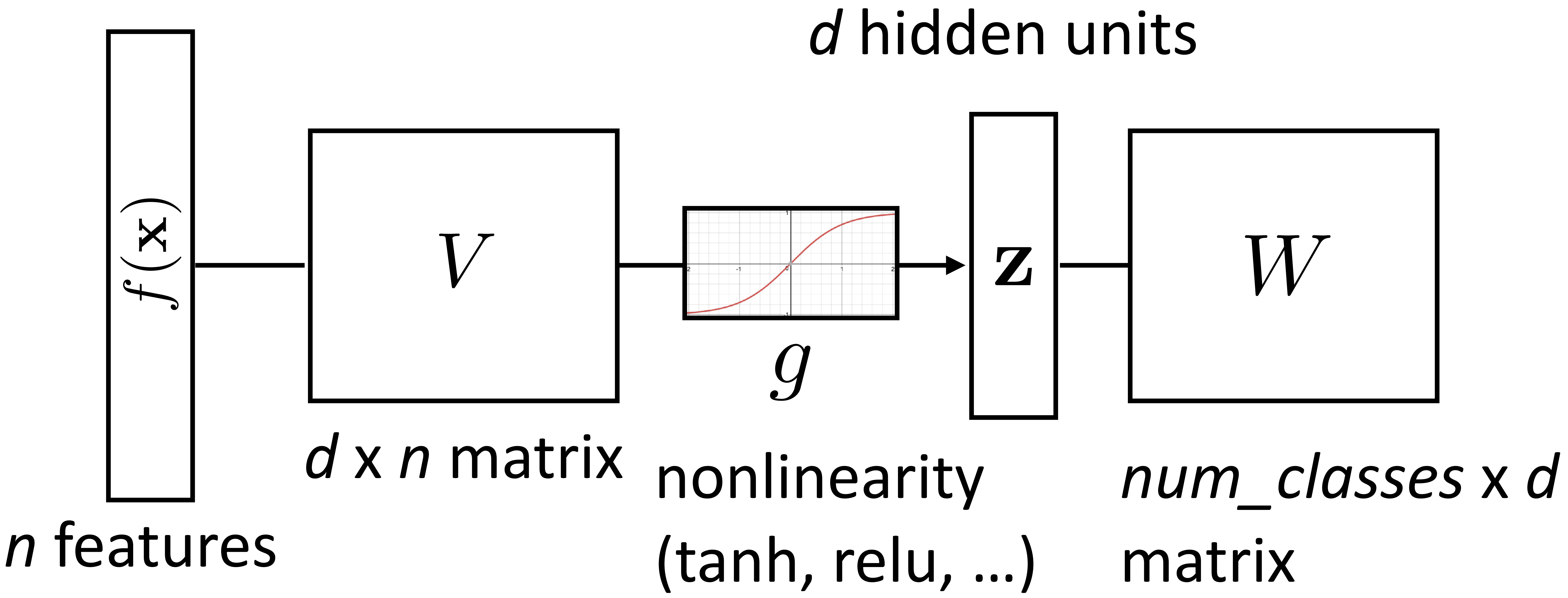
Recall: Feedforward NNs

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



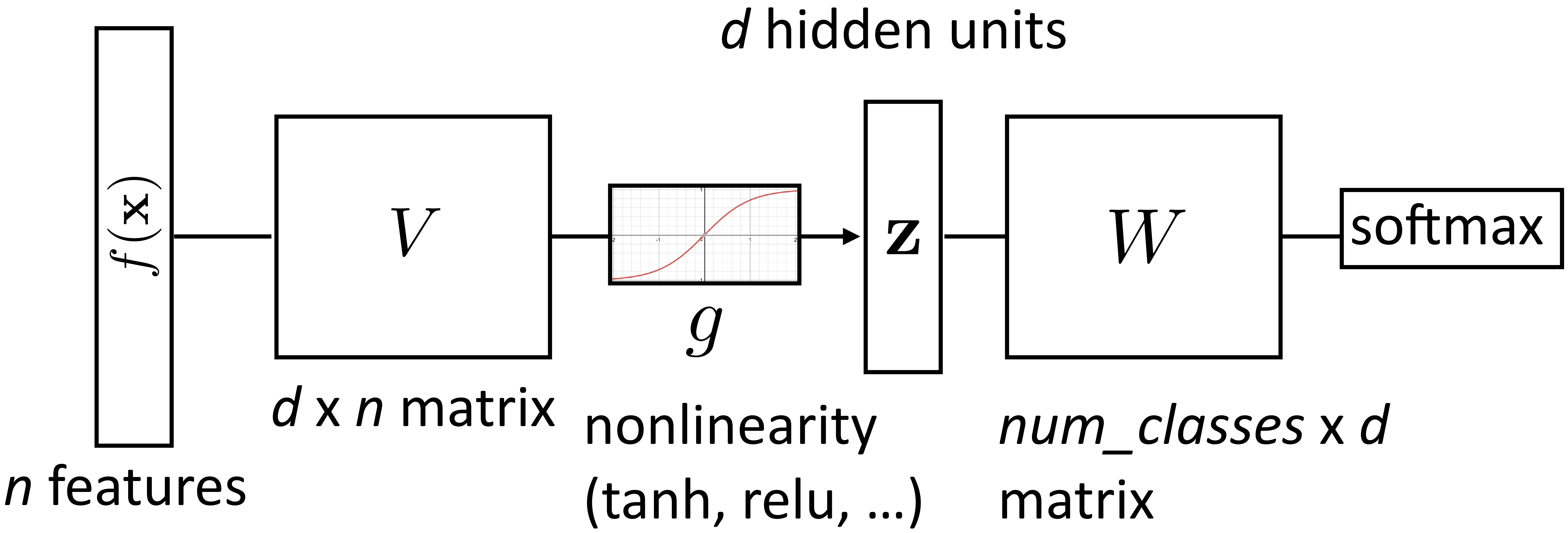
Recall: Feedforward NNs

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



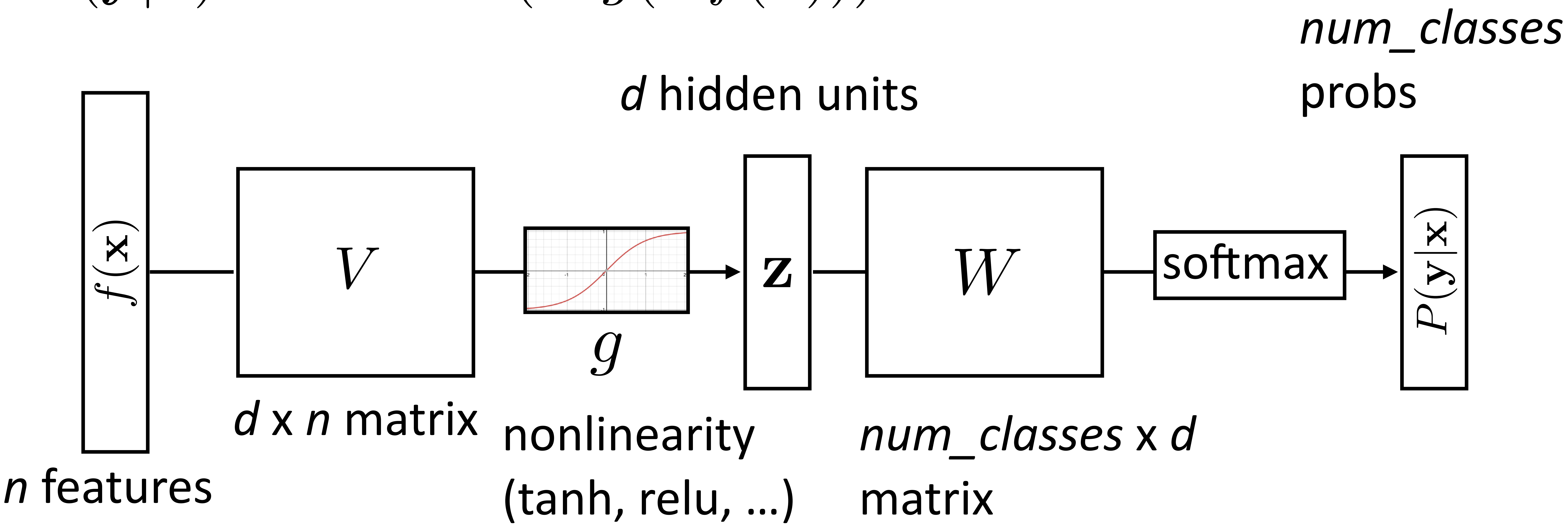
Recall: Feedforward NNs

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



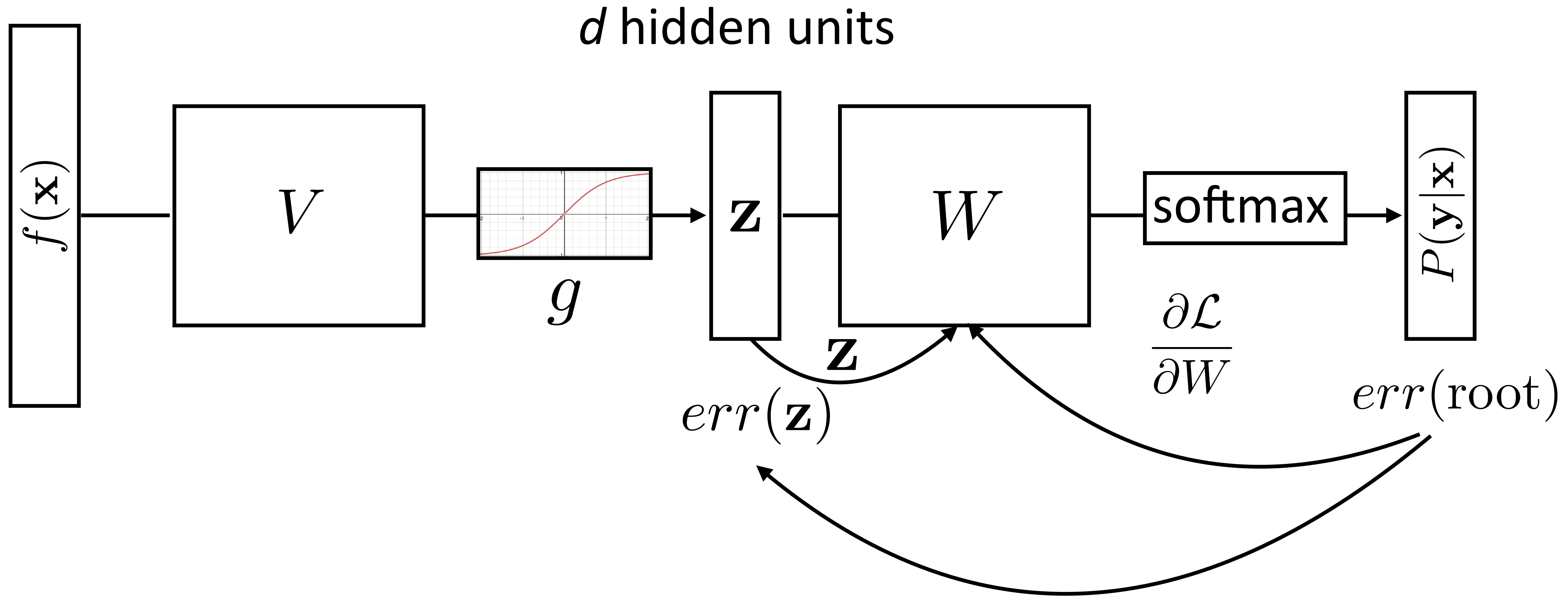
Recall: Feedforward NNs

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



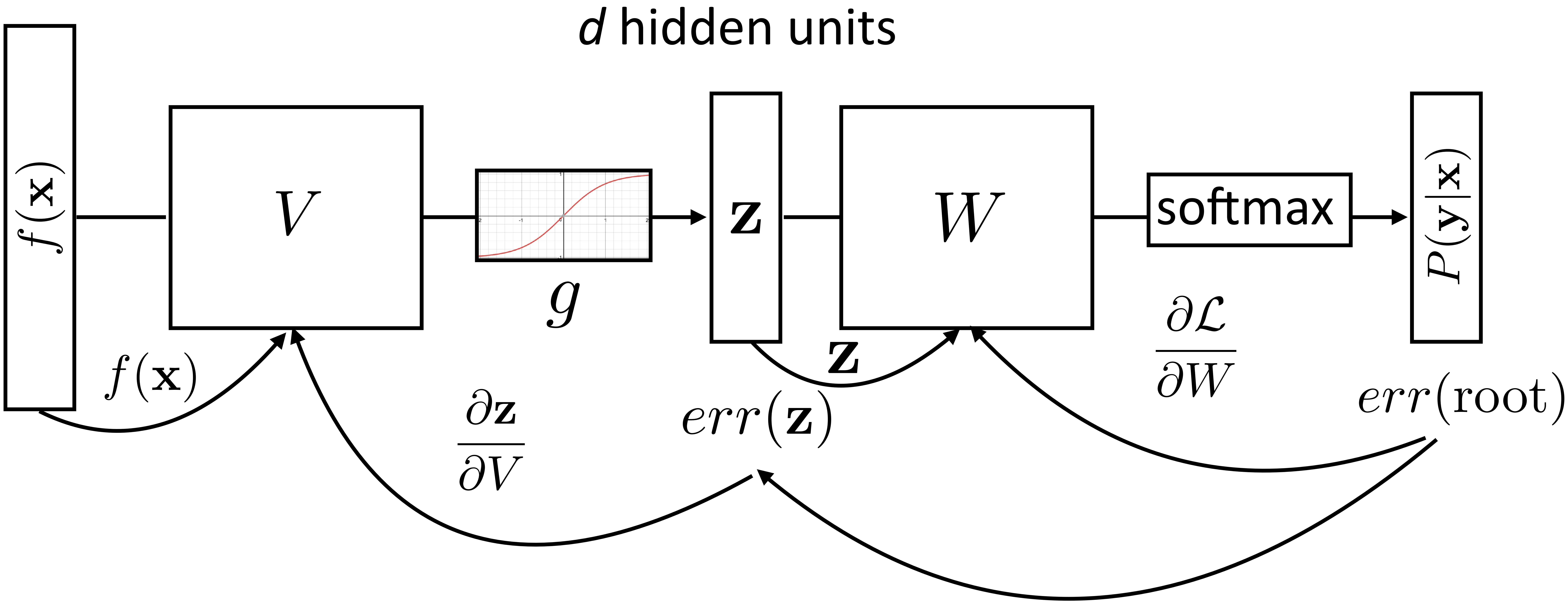
Recall: Backpropagation

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



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

Training Basics

Training Basics

- Basic formula: compute gradients on batch, use first-order opt. method

Training Basics

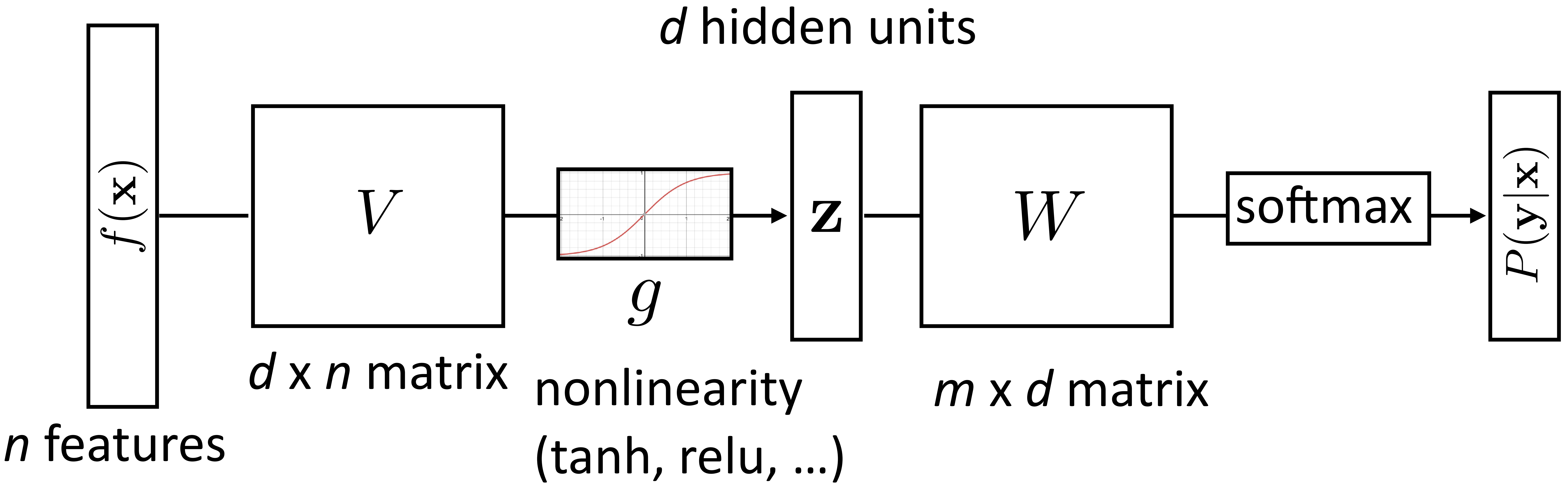
- Basic formula: compute gradients on batch, use first-order opt. method
- How to initialize? How to regularize? What optimizer to use?

Training Basics

- Basic formula: compute gradients on batch, use first-order opt. method
- 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

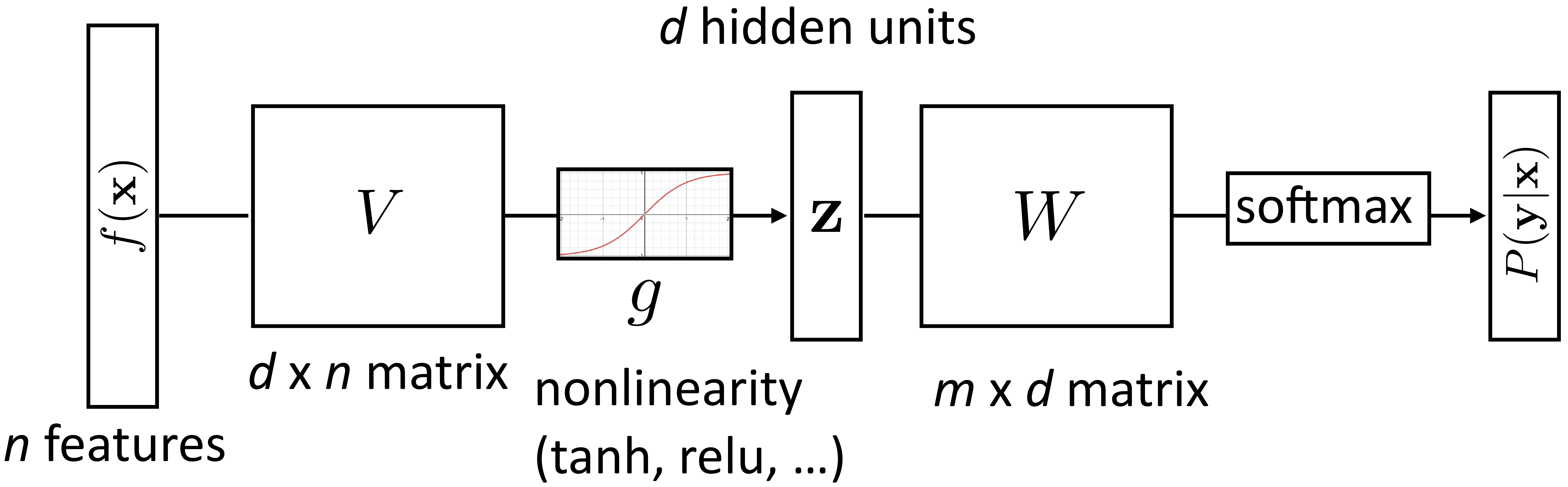
How does initialization affect learning?

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



How does initialization affect learning?

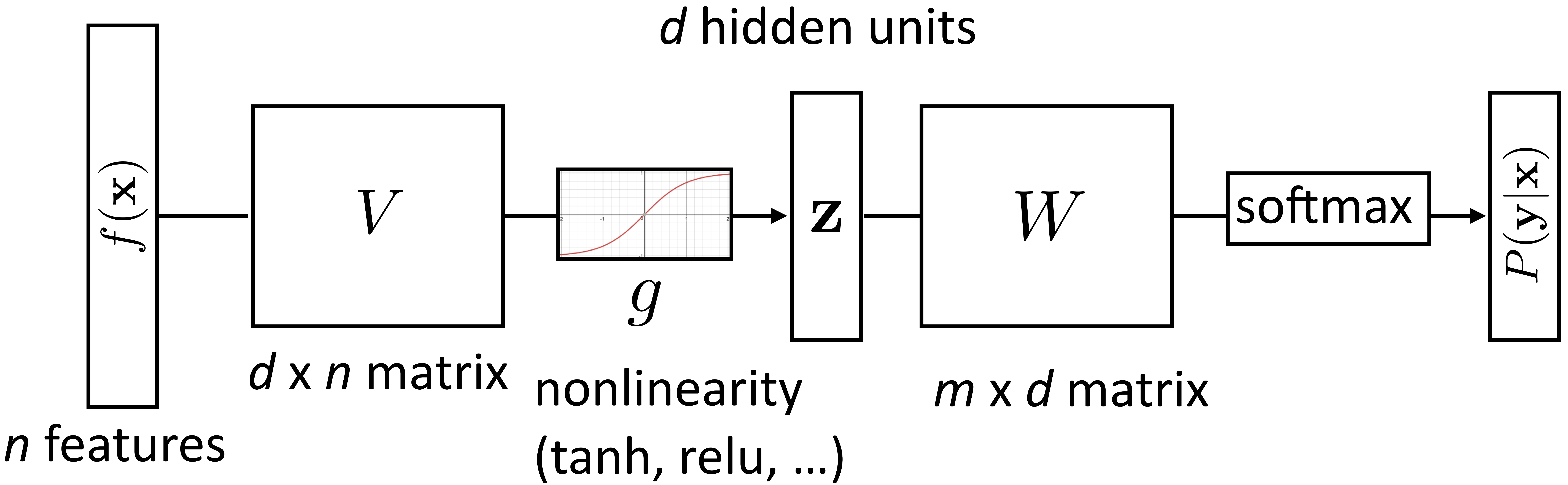
$$P(\mathbf{y}|\mathbf{x}) = \mathrm{softmax}(Wg(Vf(\mathbf{x})))$$



- How do we initialize V and W? What consequences does this have?

How does initialization affect learning?

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



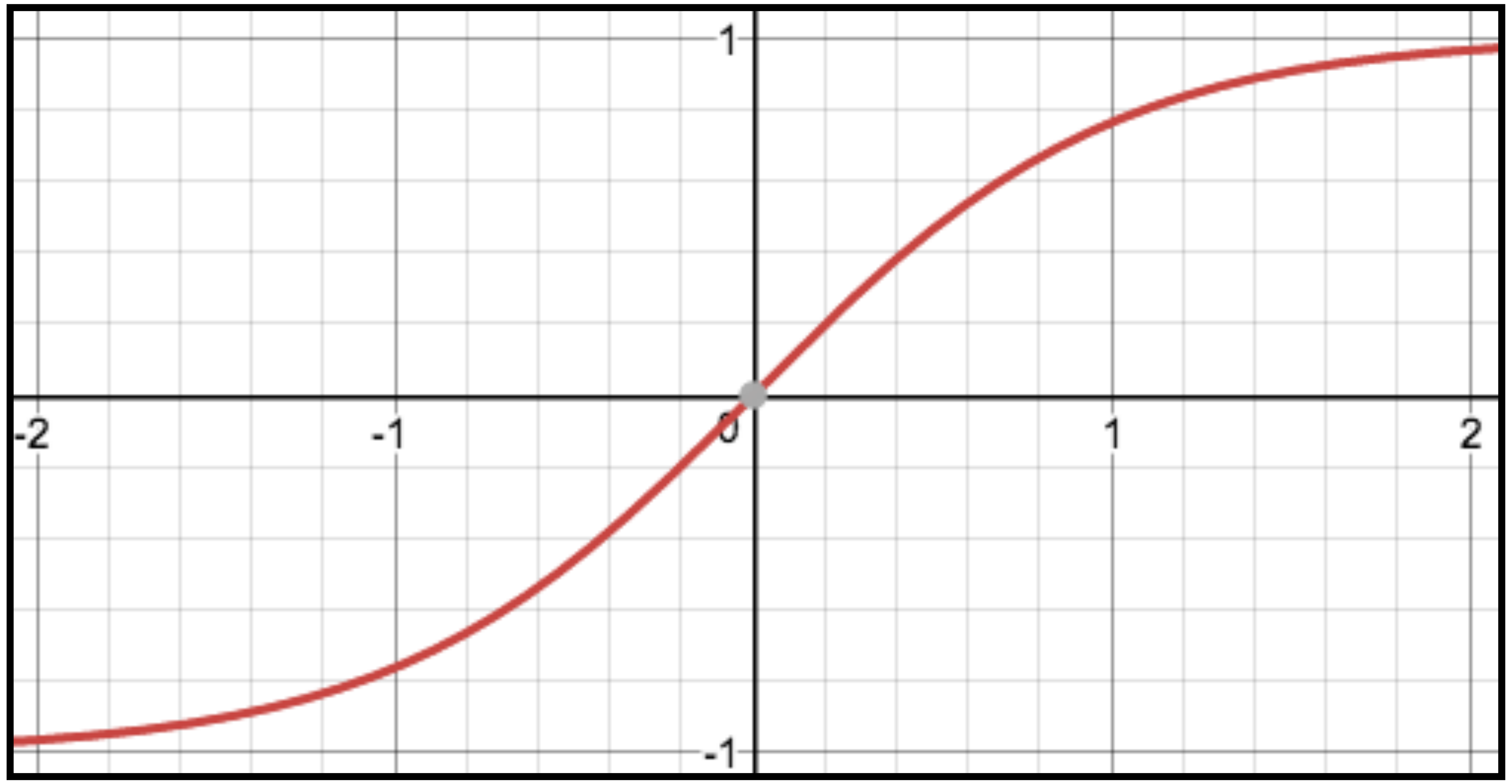
- How do we initialize V and W? What consequences does this have?
- *Nonconvex* problem, so initialization matters!

How does initialization affect learning?

- Nonlinear model…how does this affect things?

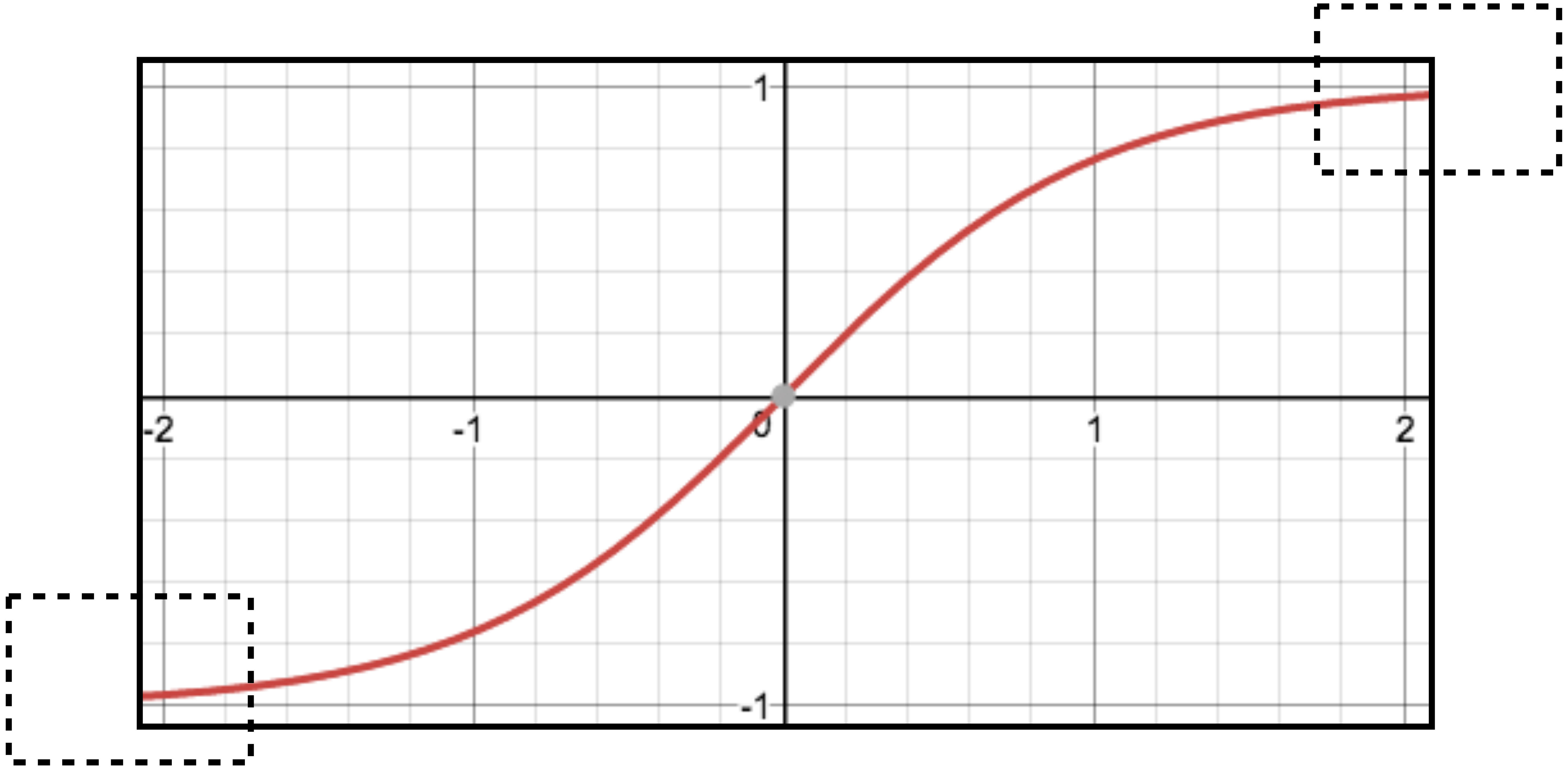
How does initialization affect learning?

- Nonlinear model…how does this affect things?



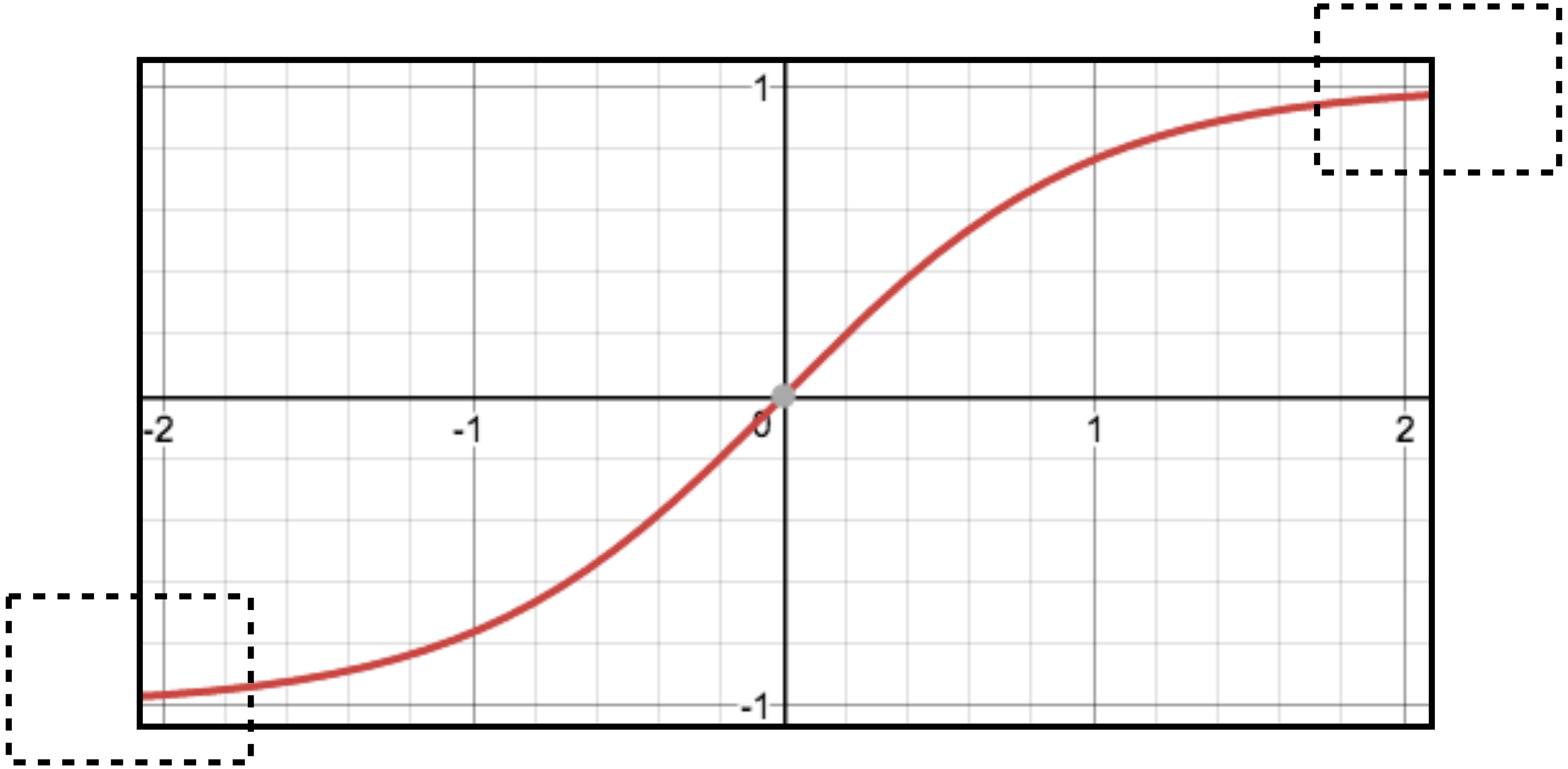
How does initialization affect learning?

- Nonlinear model…how does this affect things?



How does initialization affect learning?

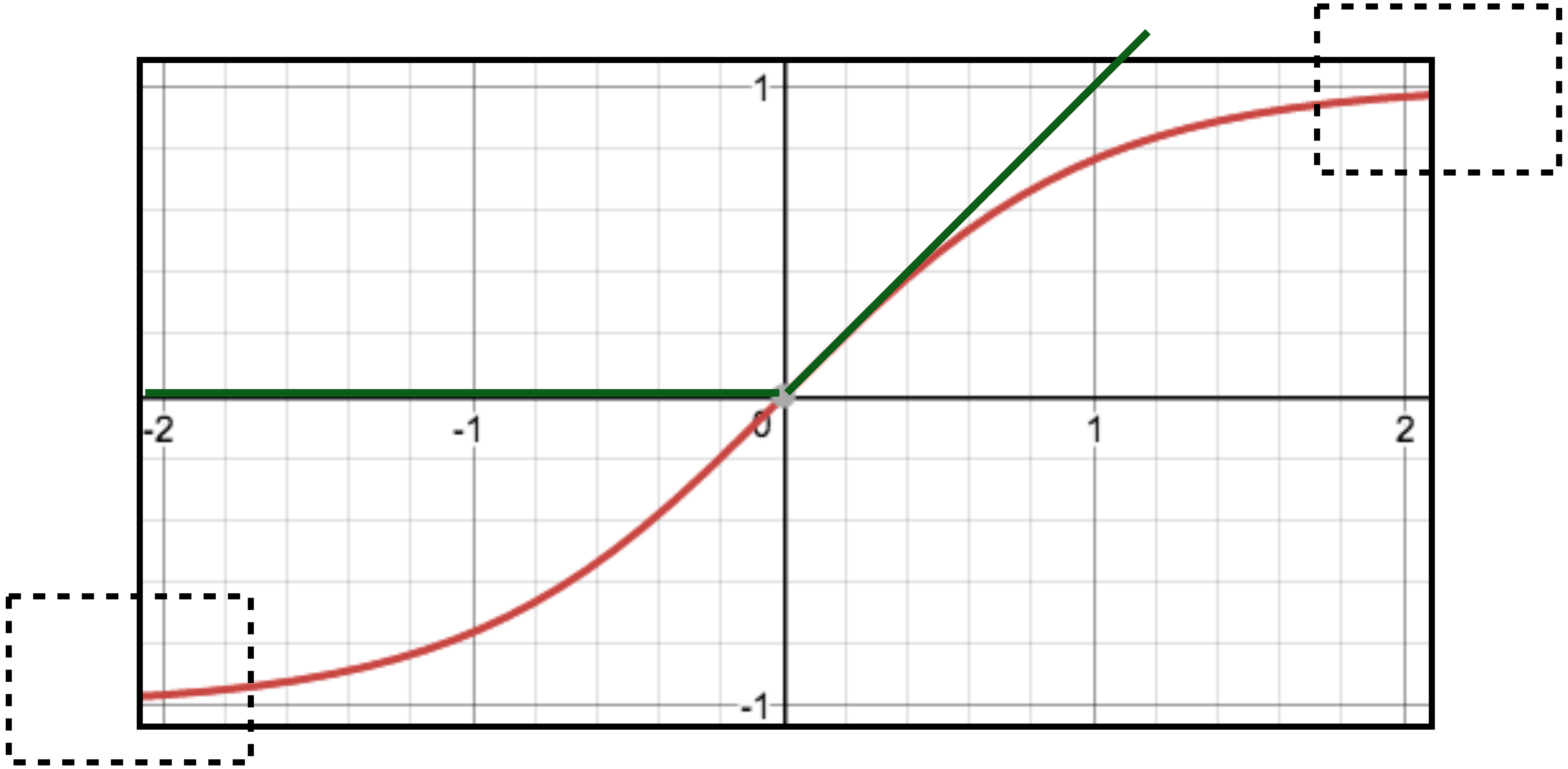
- Nonlinear model…how does this affect things?



- If cell activations are too large in absolute value, gradients are small

How does initialization affect learning?

- Nonlinear model…how does this affect things?



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

Initialization

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

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

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

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: $U\left[-\sqrt{\frac{6}{\text{fan-in}+\text{fan-out}}},+\sqrt{\frac{6}{\text{fan-in}+\text{fan-out}}}\right]$

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

Initialization

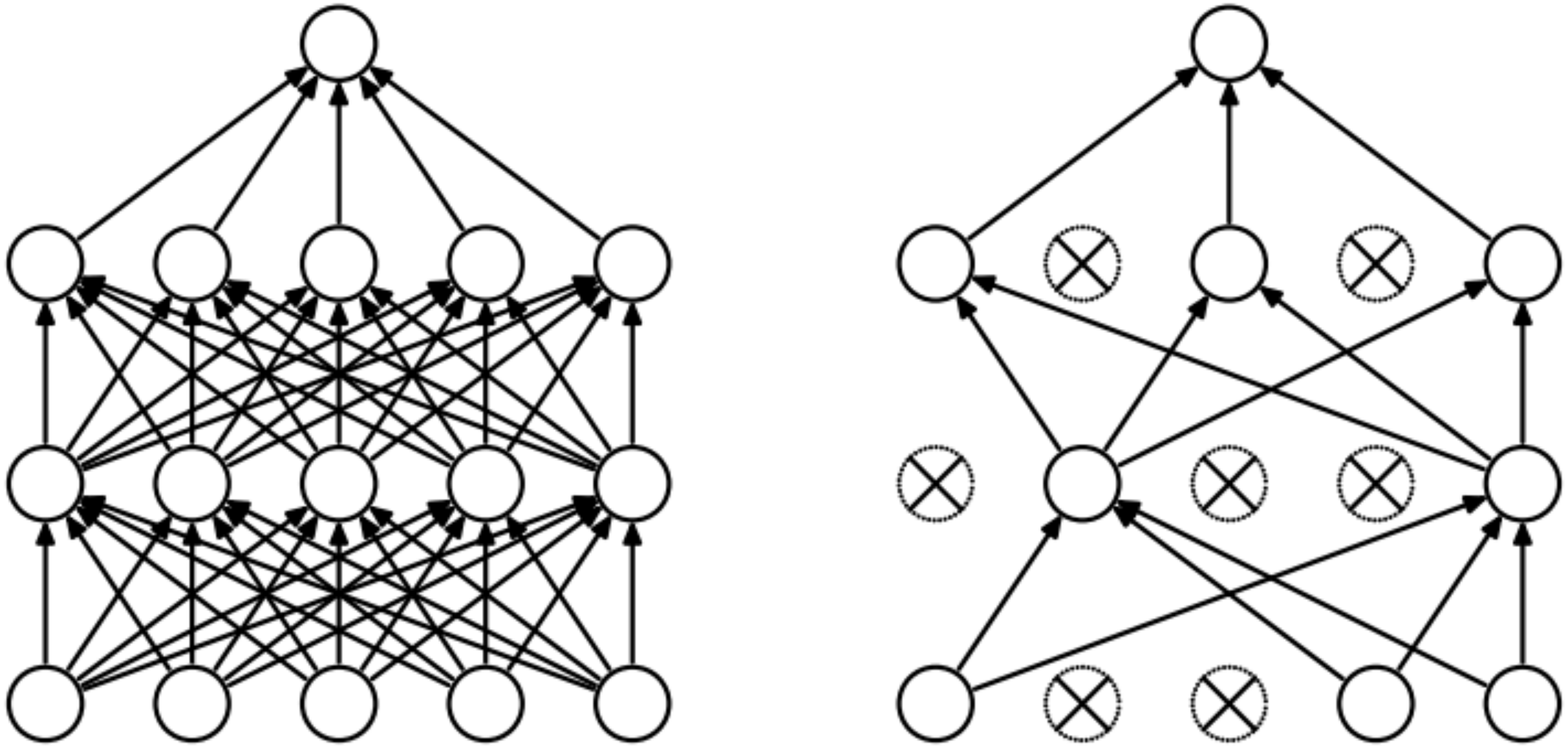
1) Can't use zeroes for parameters to produce hidden layers: all values in that hidden layer are always 0 and have gradients of 0, never change

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
- 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)

Dropout

- Probabilistically zero out parts of the network during training to prevent overfitting, use whole network at test time



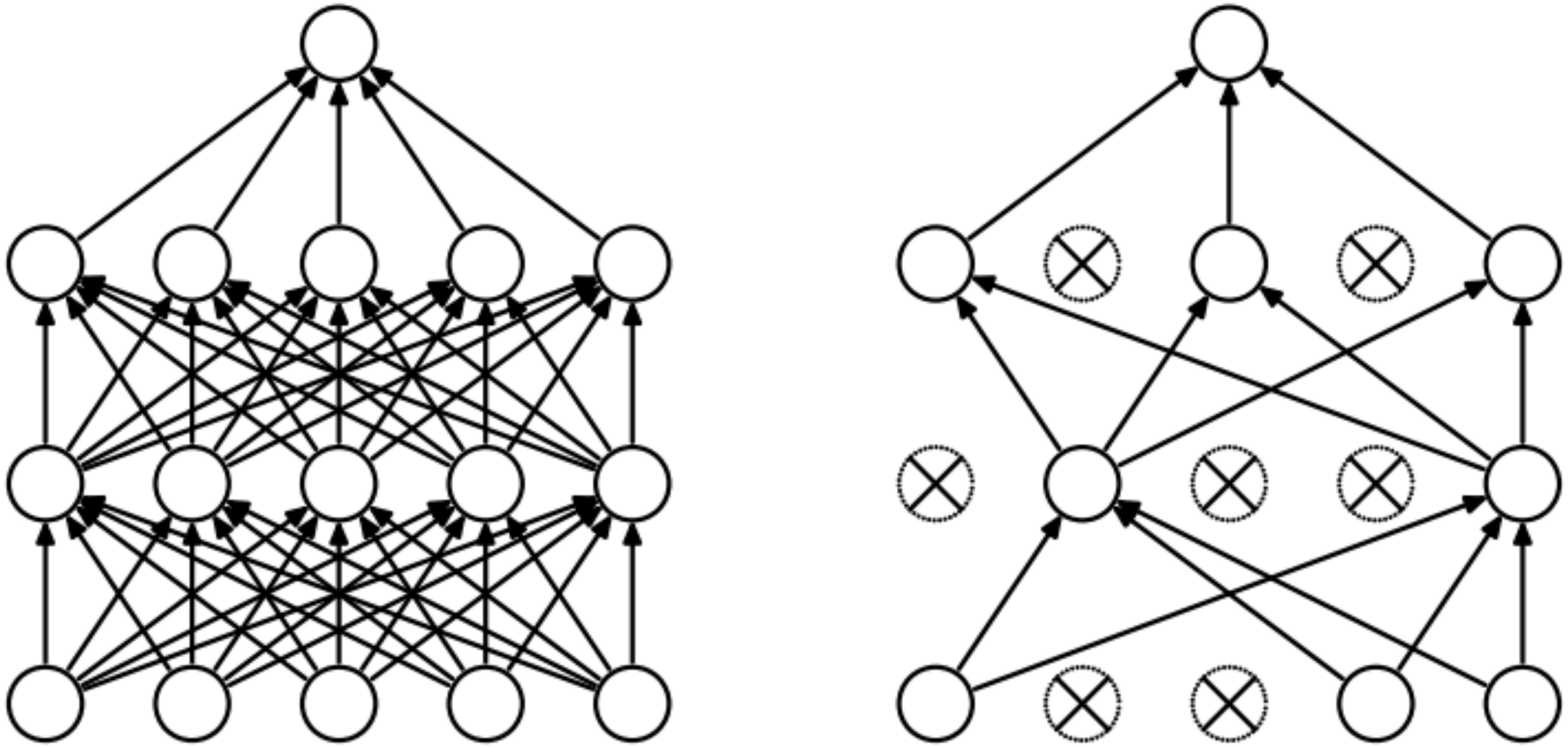
(a) Standard Neural Net (b) After applying dropout.

Srivastava et al. (2014)

Dropout

- Probabilistically zero out parts of the network during training to prevent overfitting, use whole network at test time

- Form of stochastic regularization



(a) Standard Neural Net (b) After applying dropout.

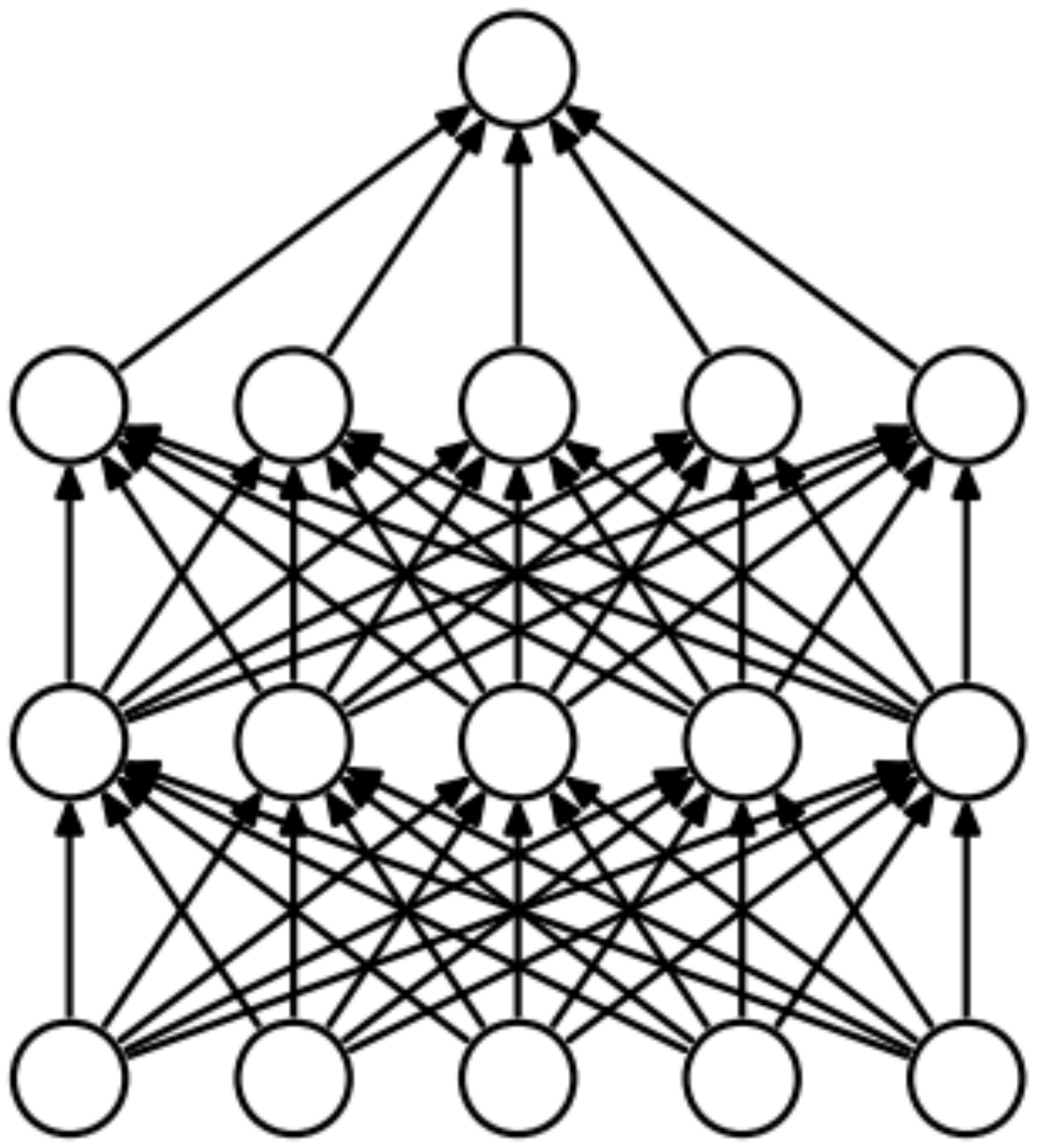
Srivastava et al. (2014)

Dropout

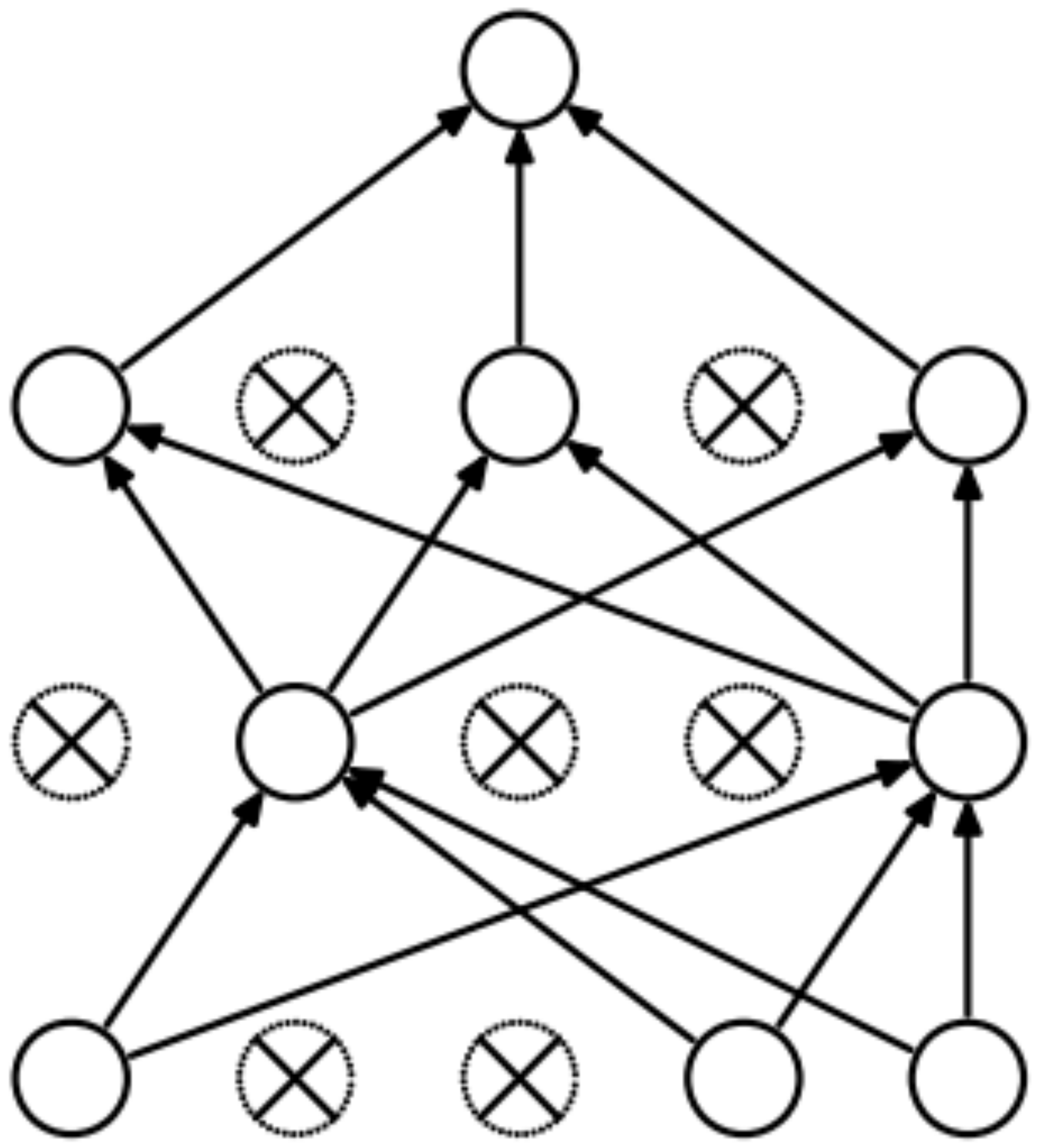
- Probabilistically zero out parts of the network during training to prevent overfitting, use whole network at test time

- 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.

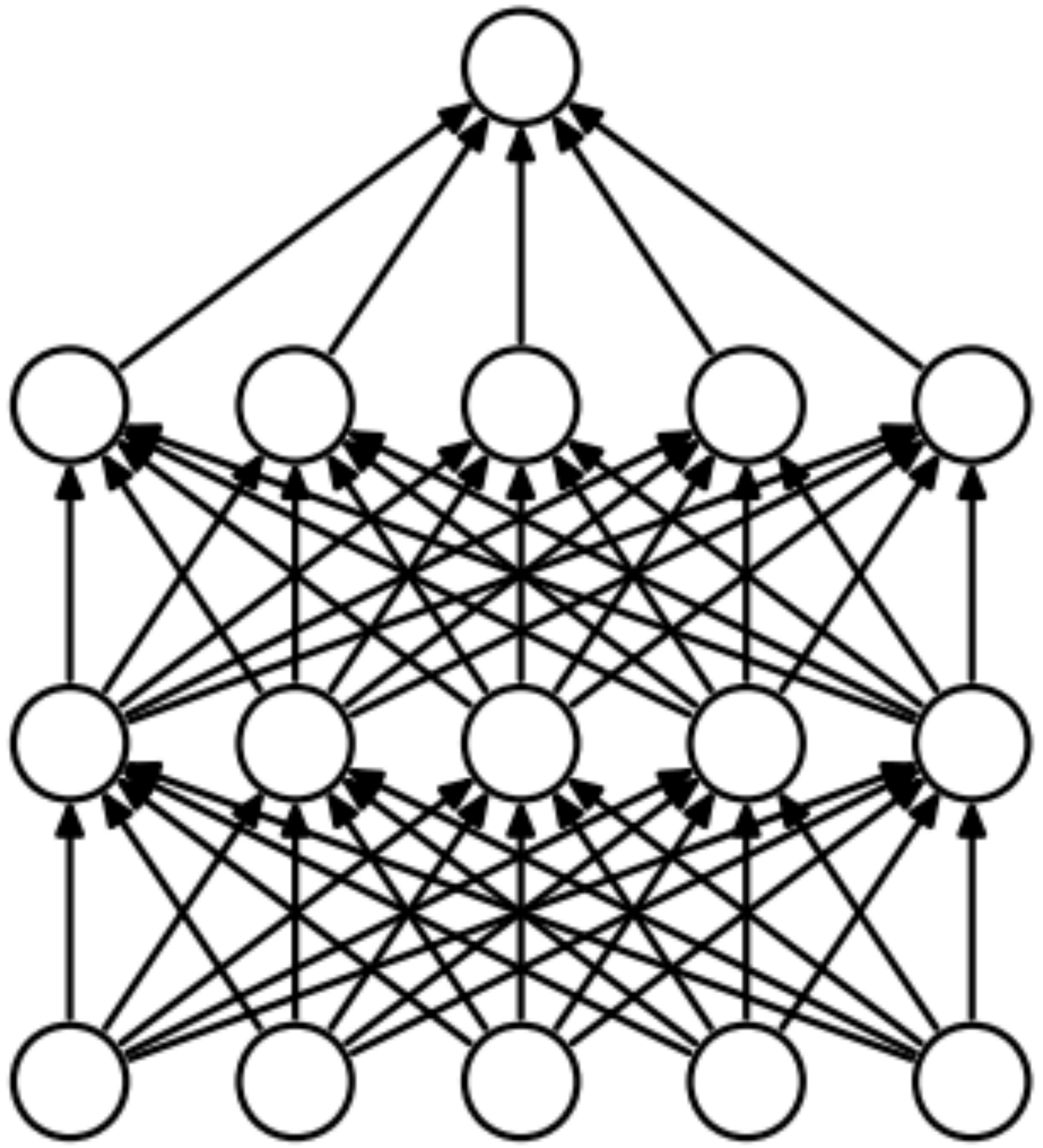
Srivastava et al. (2014)

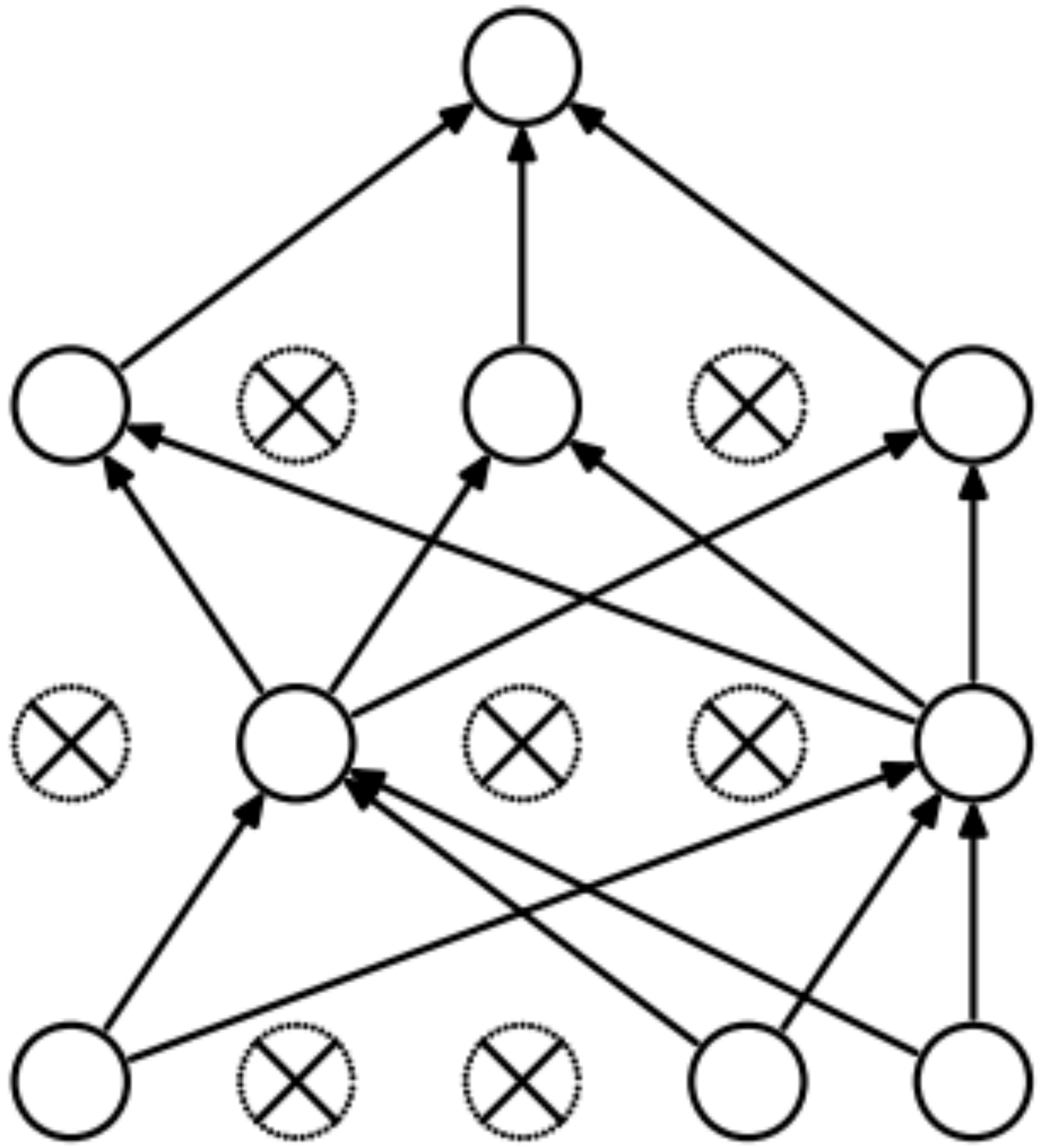
Dropout

- 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

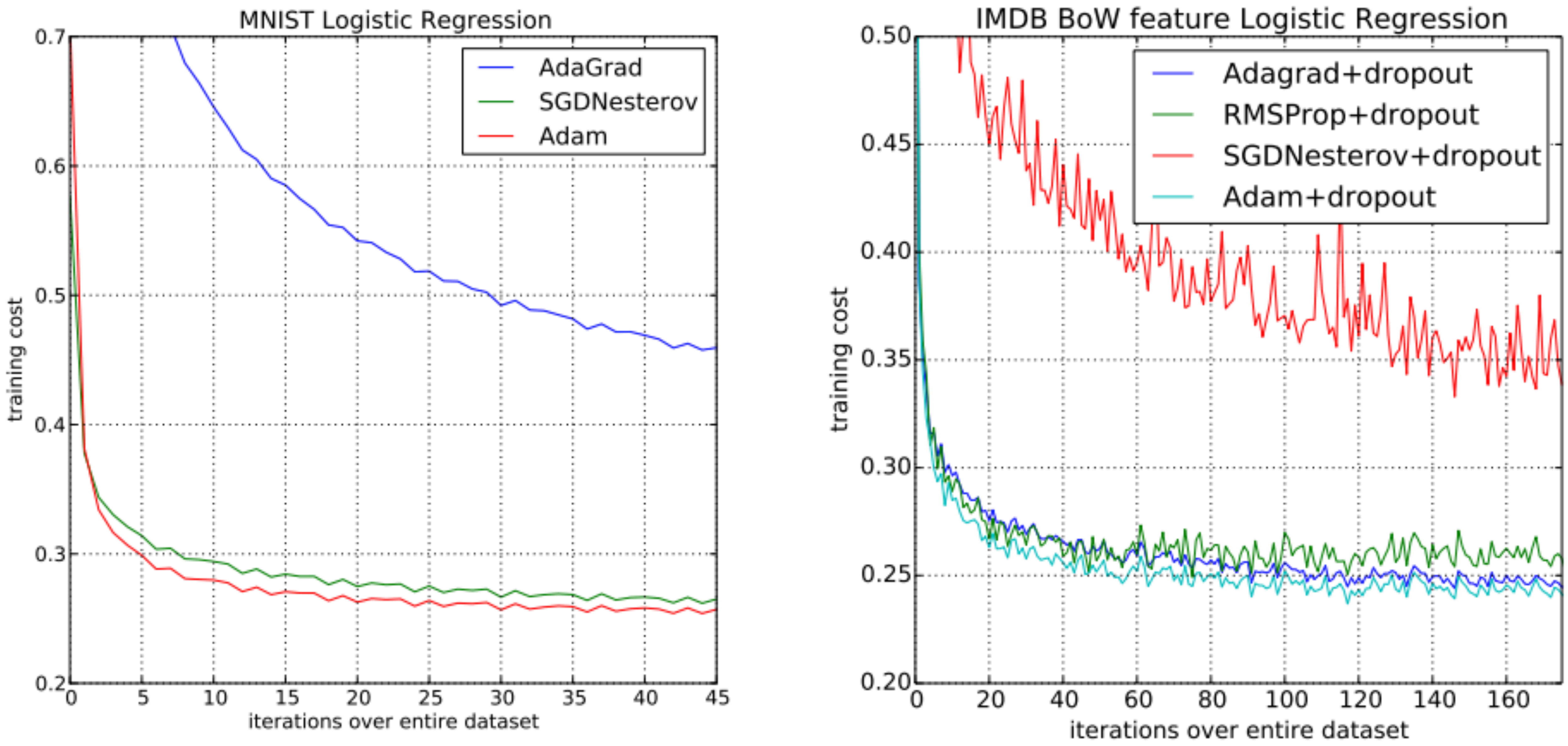
Srivastava et al. (2014)

Optimizer

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

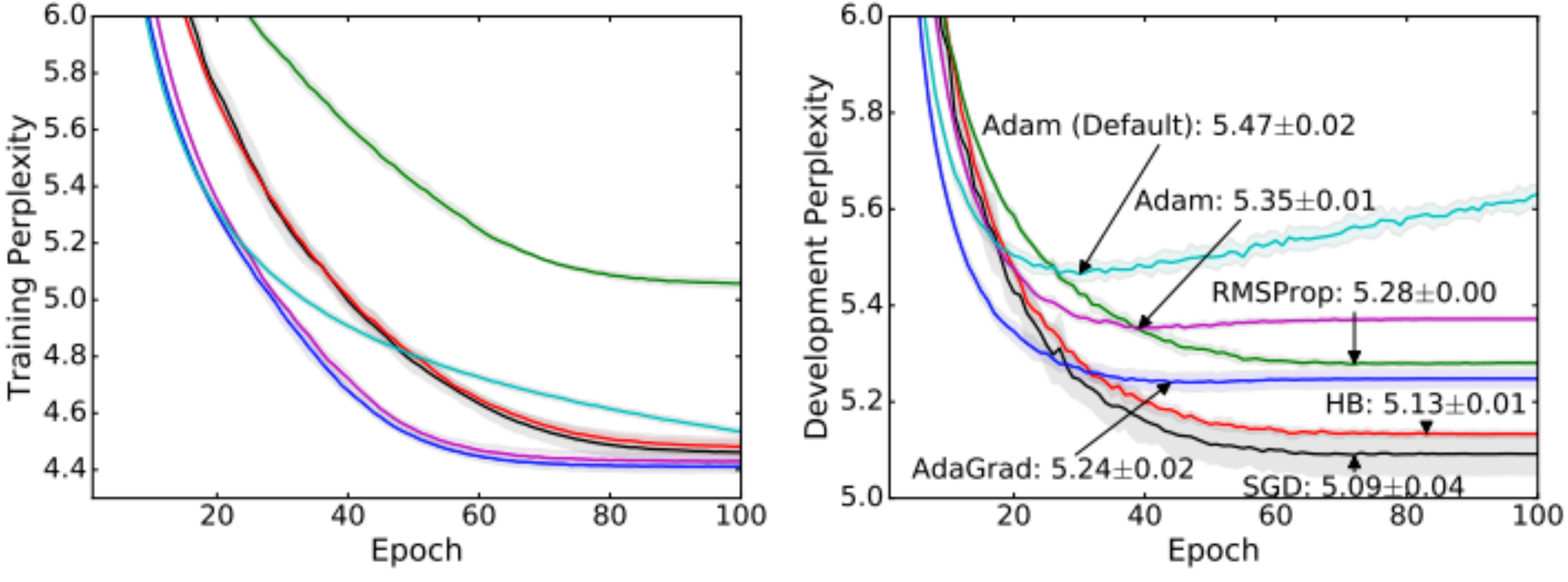
Optimizer

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



Optimizer

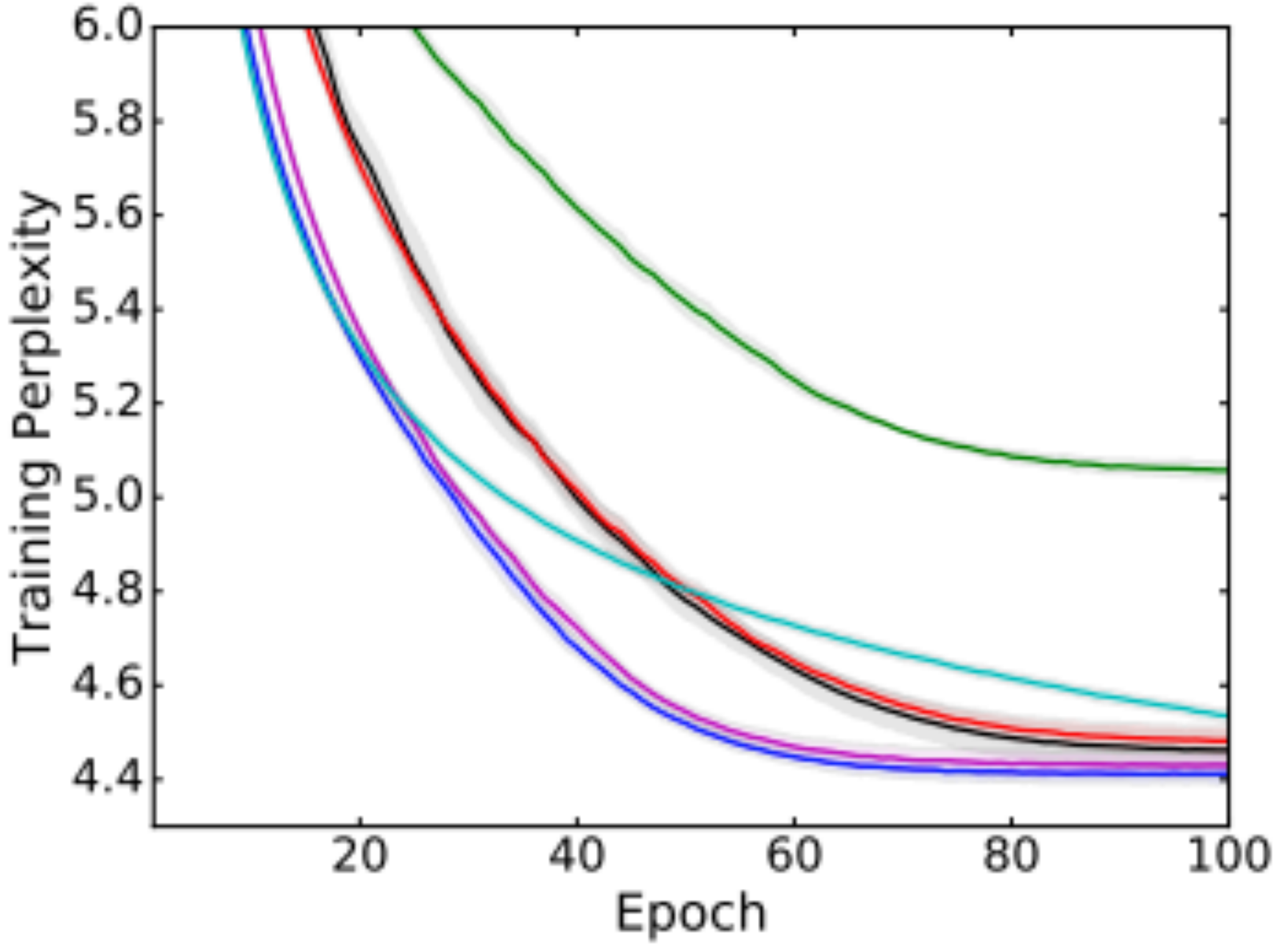
- Wilson et al. NIPS 2017: adaptive methods can actually perform badly at test time (Adam is in pink, SGD in black)



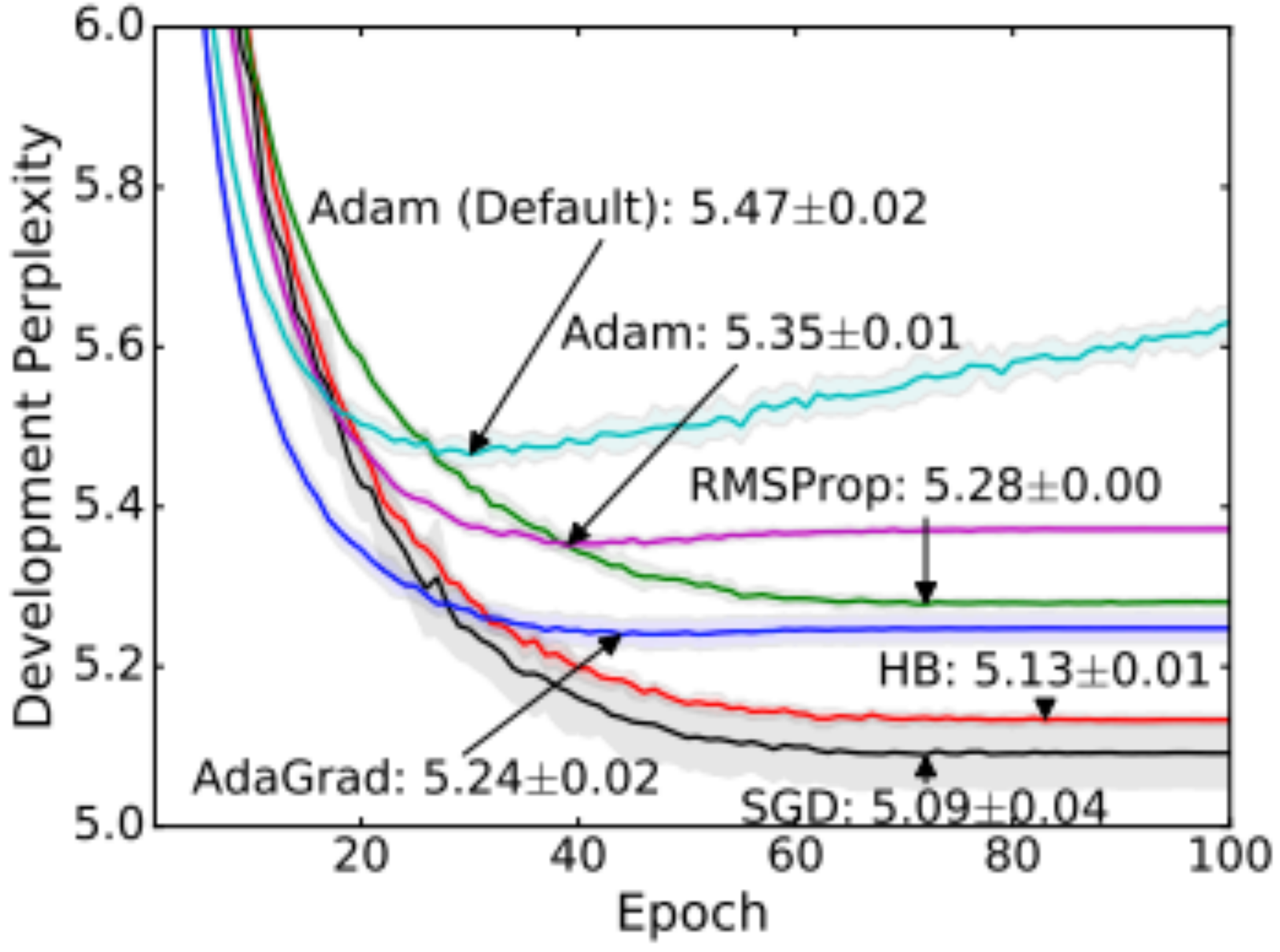
(e) Generative Parsing (Training Set) (f) Generative Parsing (Development Set)

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)

Elements of Machine Learning

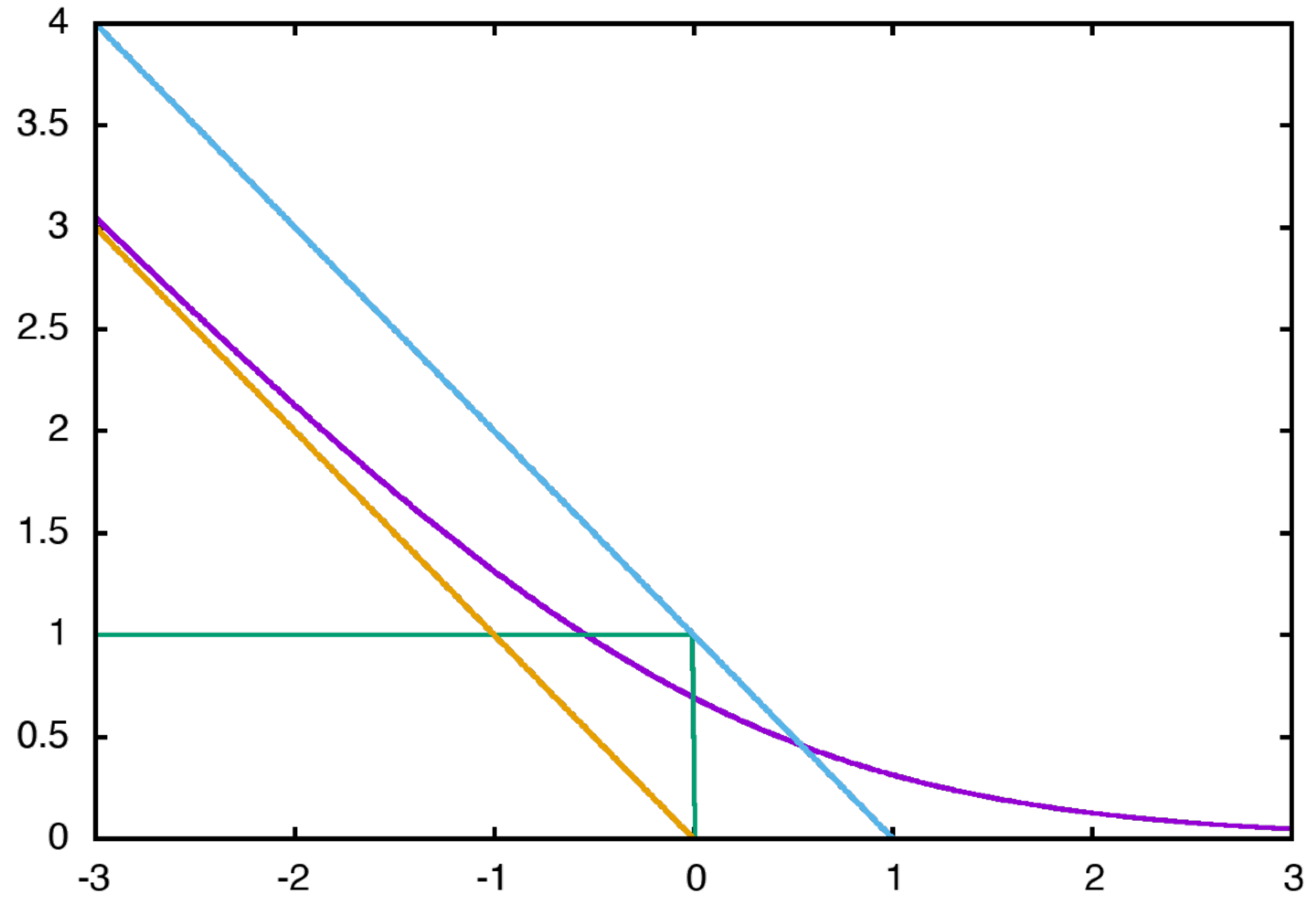
- Four elements of a machine learning method:

Elements of Machine Learning

- Four elements of a machine learning method:
- Model: feedforward, RNNs, CNNs can be defined in a uniform framework

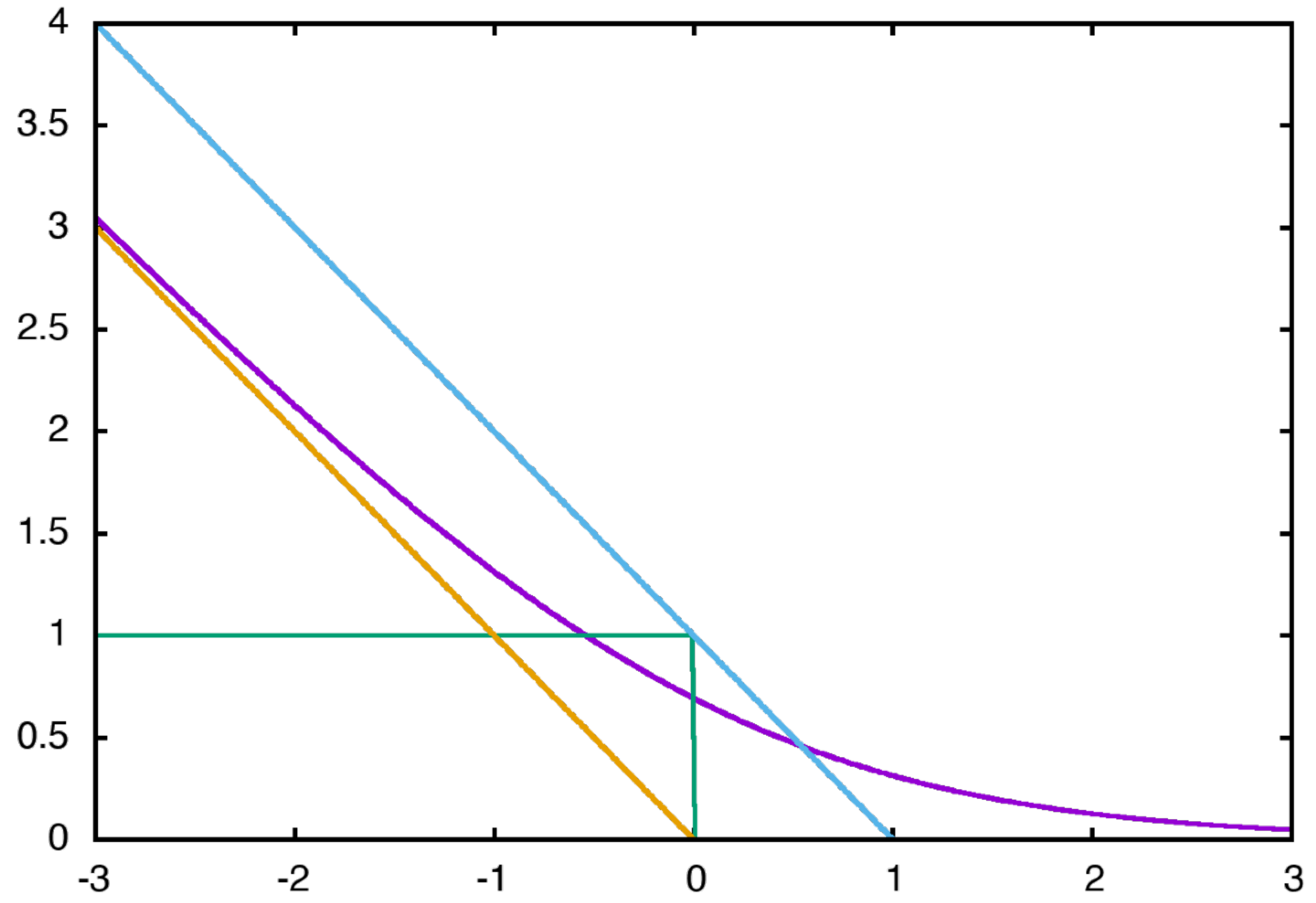
Elements of Machine Learning

- Four elements of a machine learning method:
- 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



Elements of Machine Learning

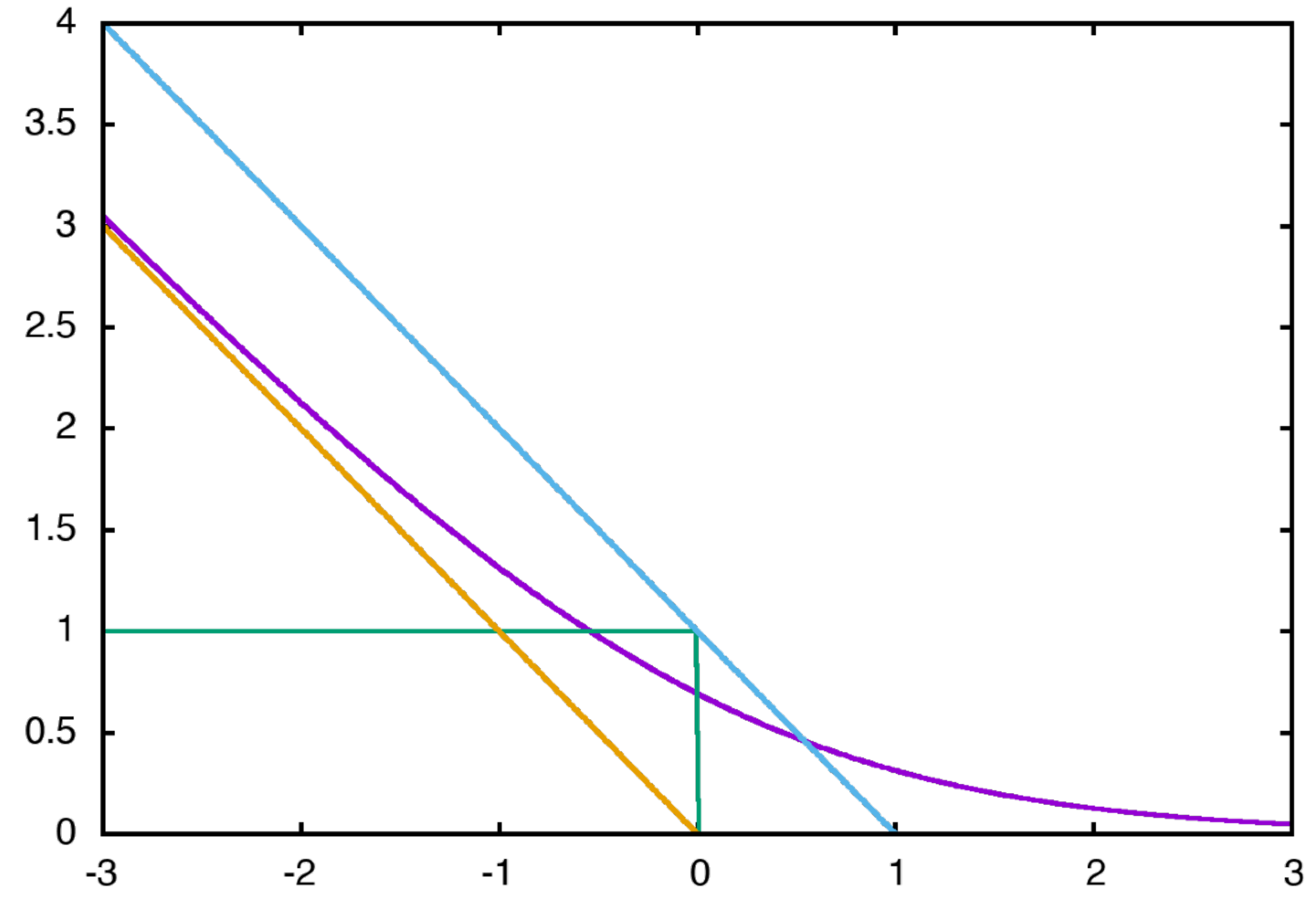
- Four elements of a machine learning method:
- Model: feedforward, RNNs, CNNs can be defined in a uniform framework
- Objective: many loss functions look similar, just changes the last layer of the neural network



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

Elements of Machine Learning

- Four elements of a machine learning method:
- 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

slide credit: Dan Klein

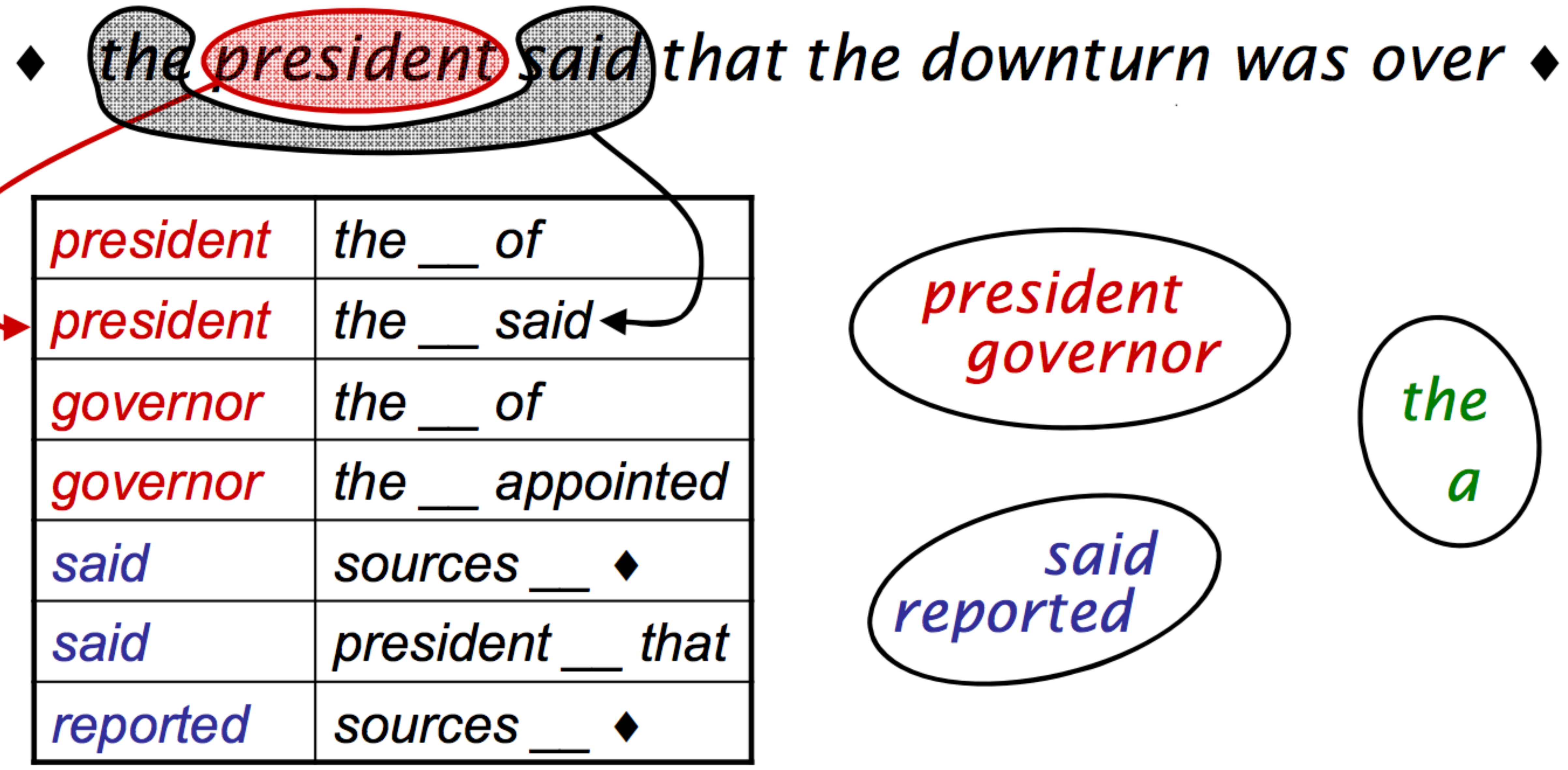
Word Representations

- Neural networks work very well at continuous data, but words are discrete
- Continuous model <-> expects continuous semantics from input

slide credit: Dan Klein

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)



[Finch and Chater 92, Shuetze 93, many others]

slide credit: Dan Klein

Discrete Word Representations

Brown et al. (1992)

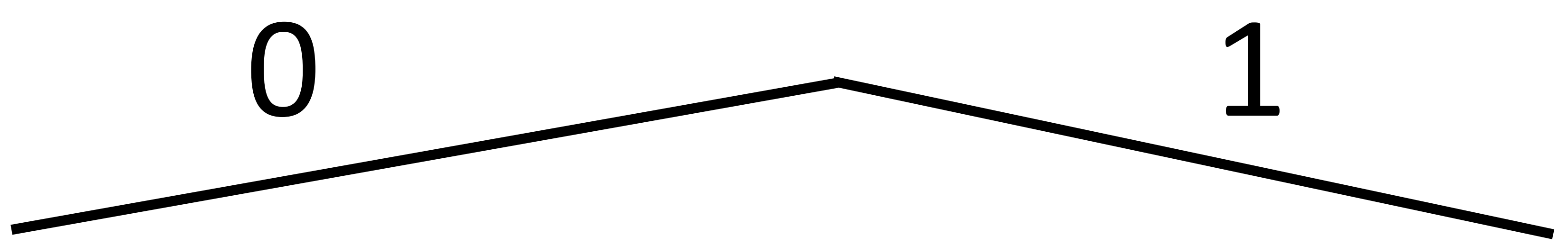
Discrete Word Representations

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

Brown et al. (1992)

Discrete Word Representations

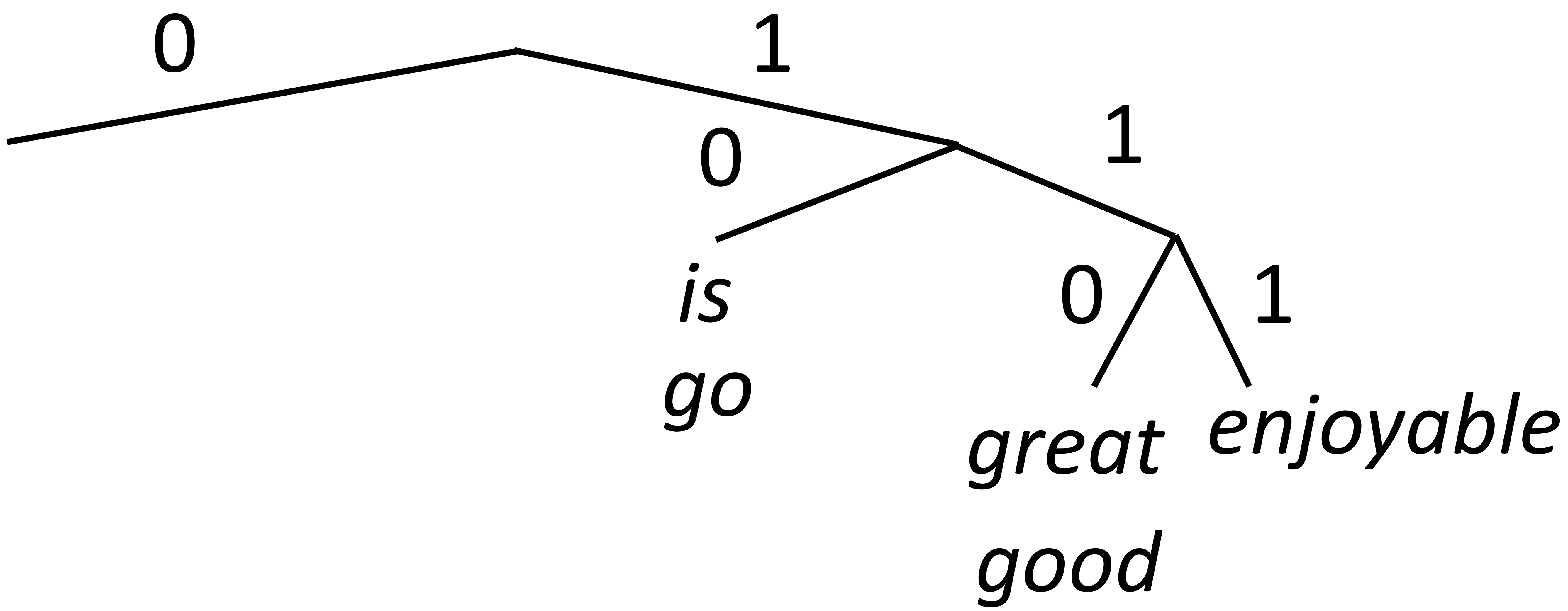
- Brown clusters: hierarchical agglomerative *hard* clustering (each word has one cluster, not some posterior distribution like in mixture models)



Brown et al. (1992)

Discrete Word Representations

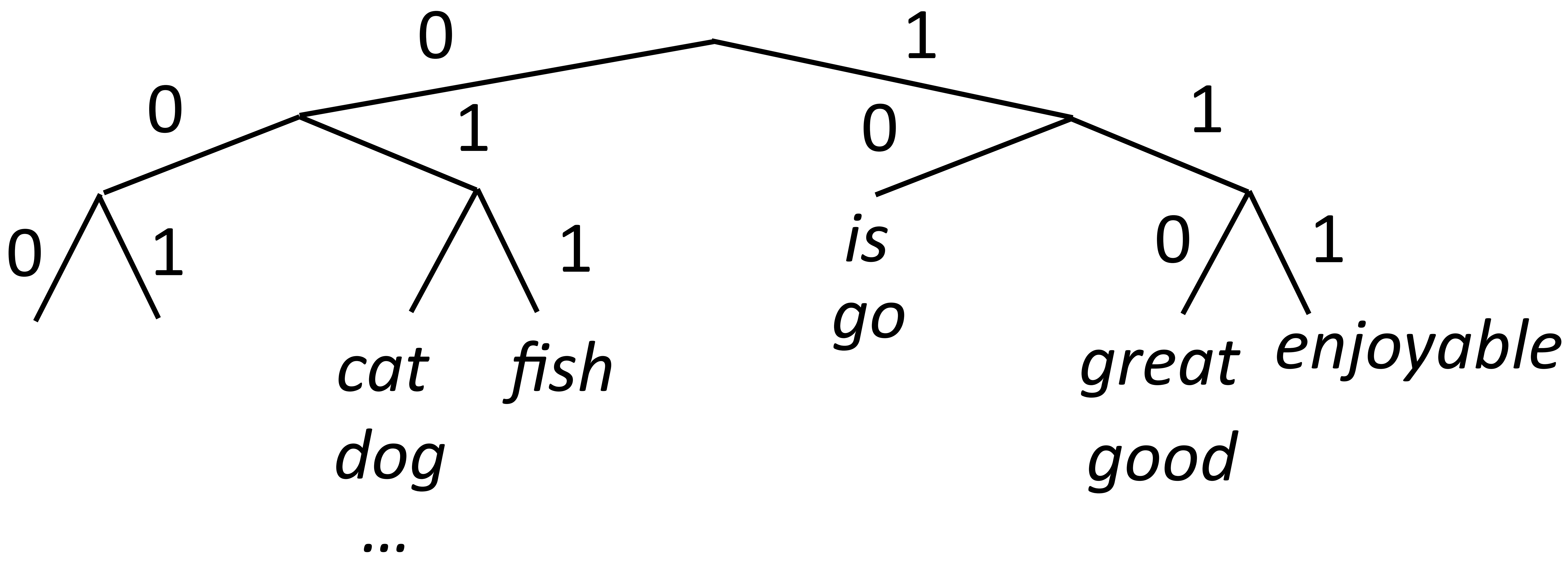
- Brown clusters: hierarchical agglomerative *hard* clustering (each word has one cluster, not some posterior distribution like in mixture models)



Brown et al. (1992)

Discrete Word Representations

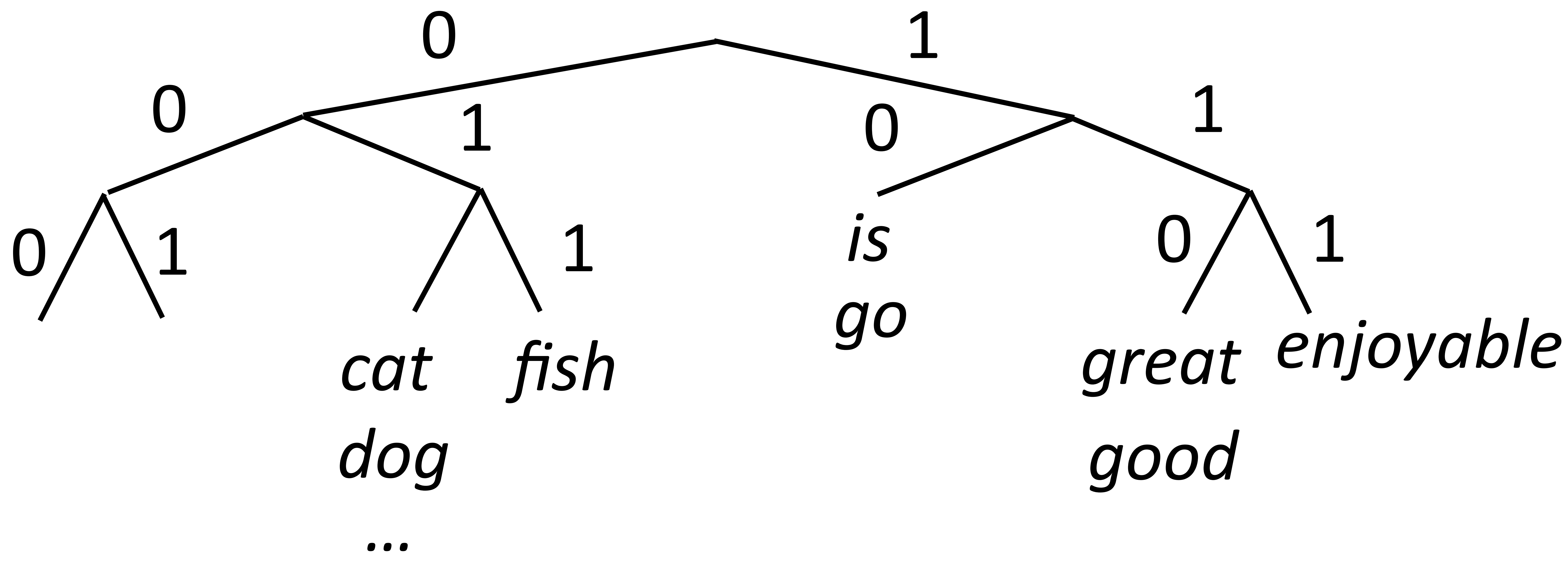
- Brown clusters: hierarchical agglomerative *hard* clustering (each word has one cluster, not some posterior distribution like in mixture models)



Brown et al. (1992)

Discrete Word Representations

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

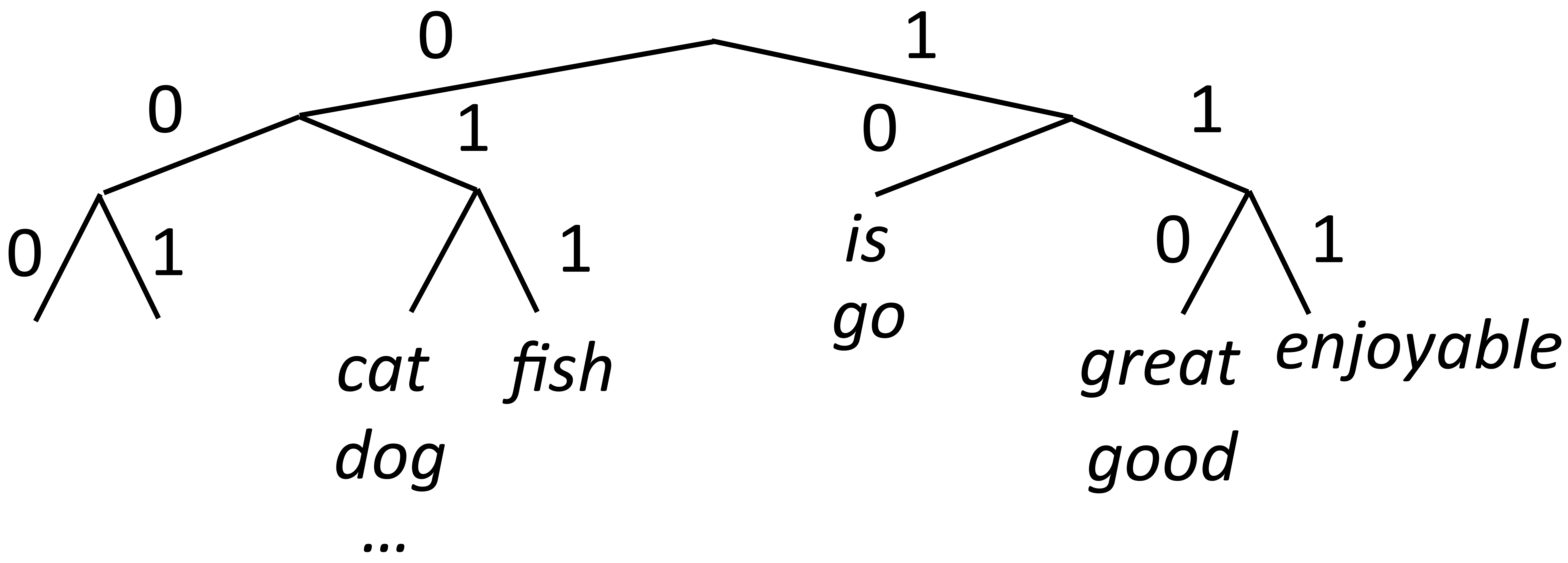


- Maximize $P(w_i|w_{i-1}) = P(c_i|c_{i-1})P(w_i|c_i)$

Brown et al. (1992)

Discrete Word Representations

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

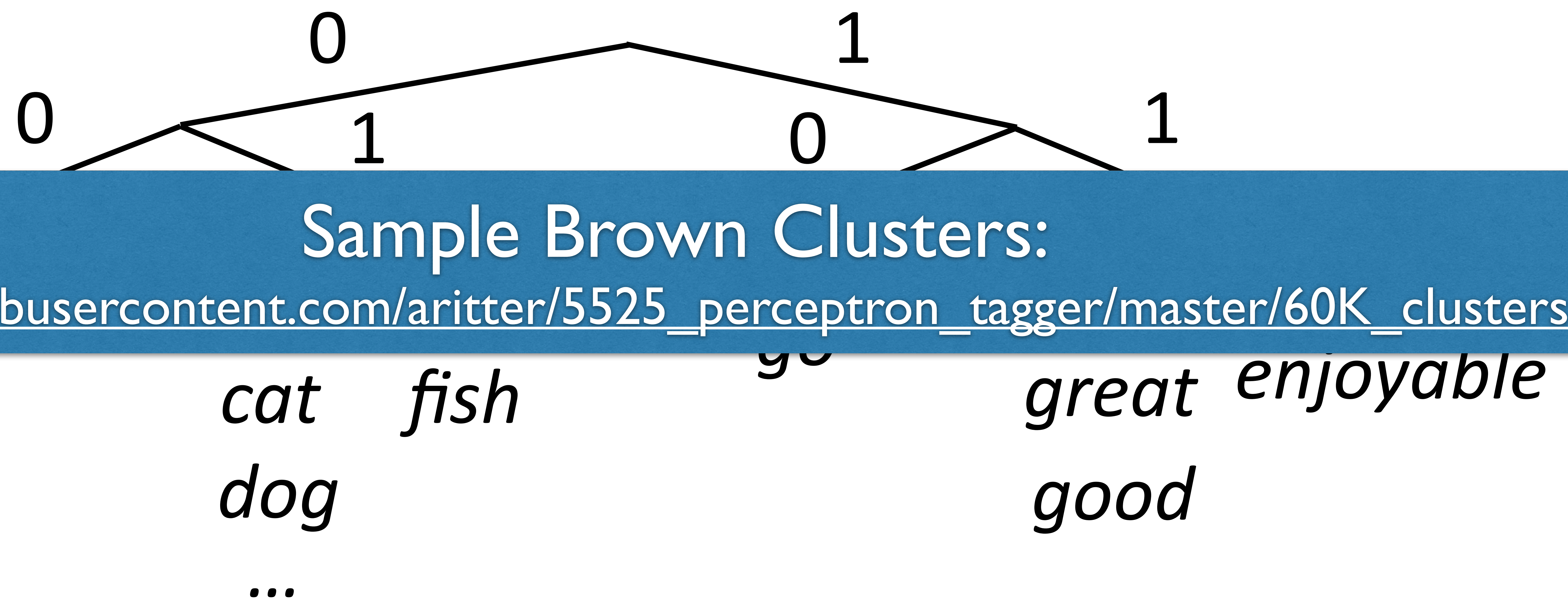


- Maximize $P(w_i|w_{i-1}) = P(c_i|c_{i-1})P(w_i|c_i)$
- Useful features for tasks like NER, not suitable for NNs

Brown et al. (1992)

Discrete Word Representations

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



Sample Brown Clusters:

- https://raw.githubusercontent.com/aritter/5525_perceptron_tagger/master/60K_clusters.bits.txt

- 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)

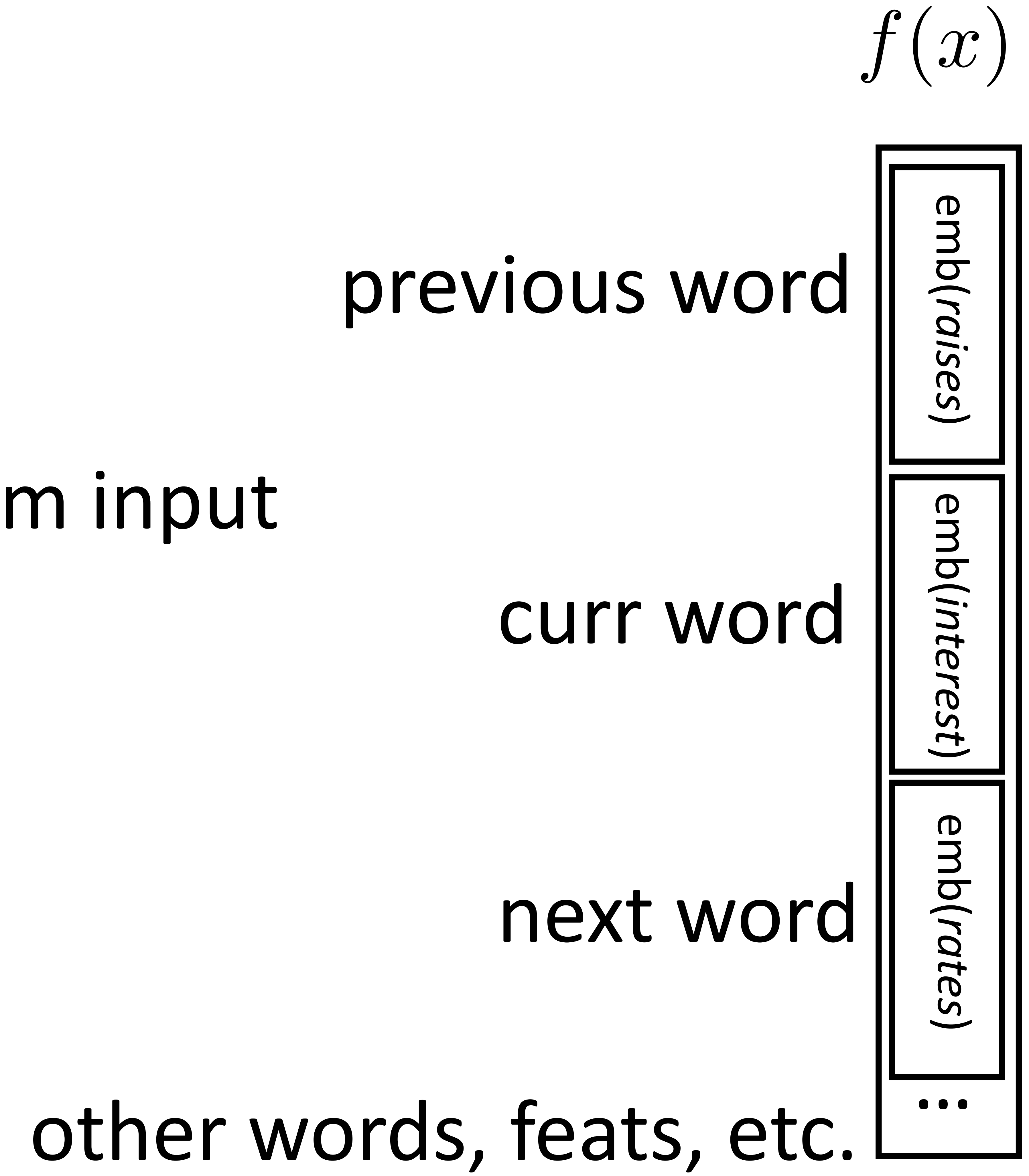
Word Embeddings

- Part-of-speech tagging with FFNNs

??

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

- Word embeddings for each word form input



Botha et al. (2017)

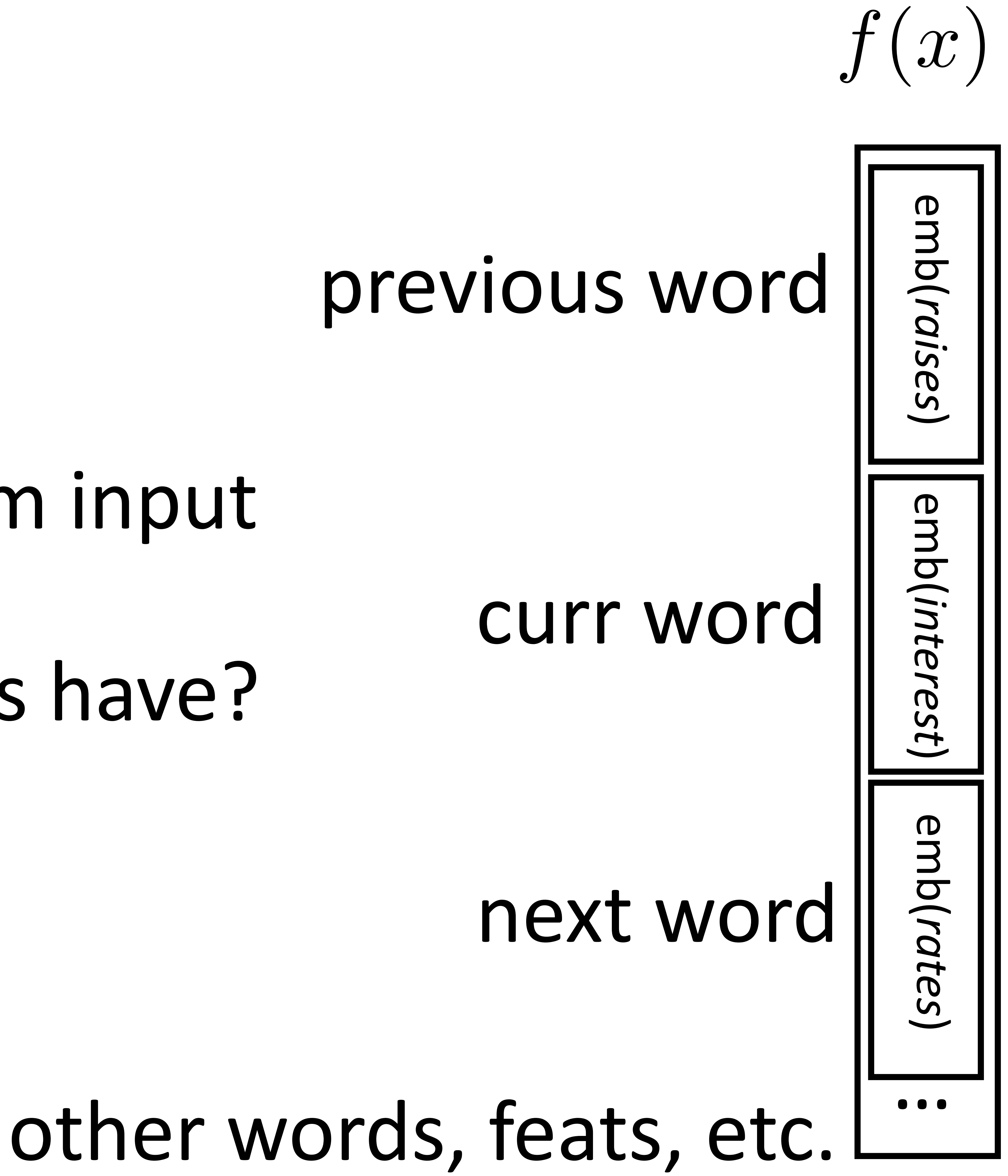
Word Embeddings

- Part-of-speech tagging with FFNNs

??

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

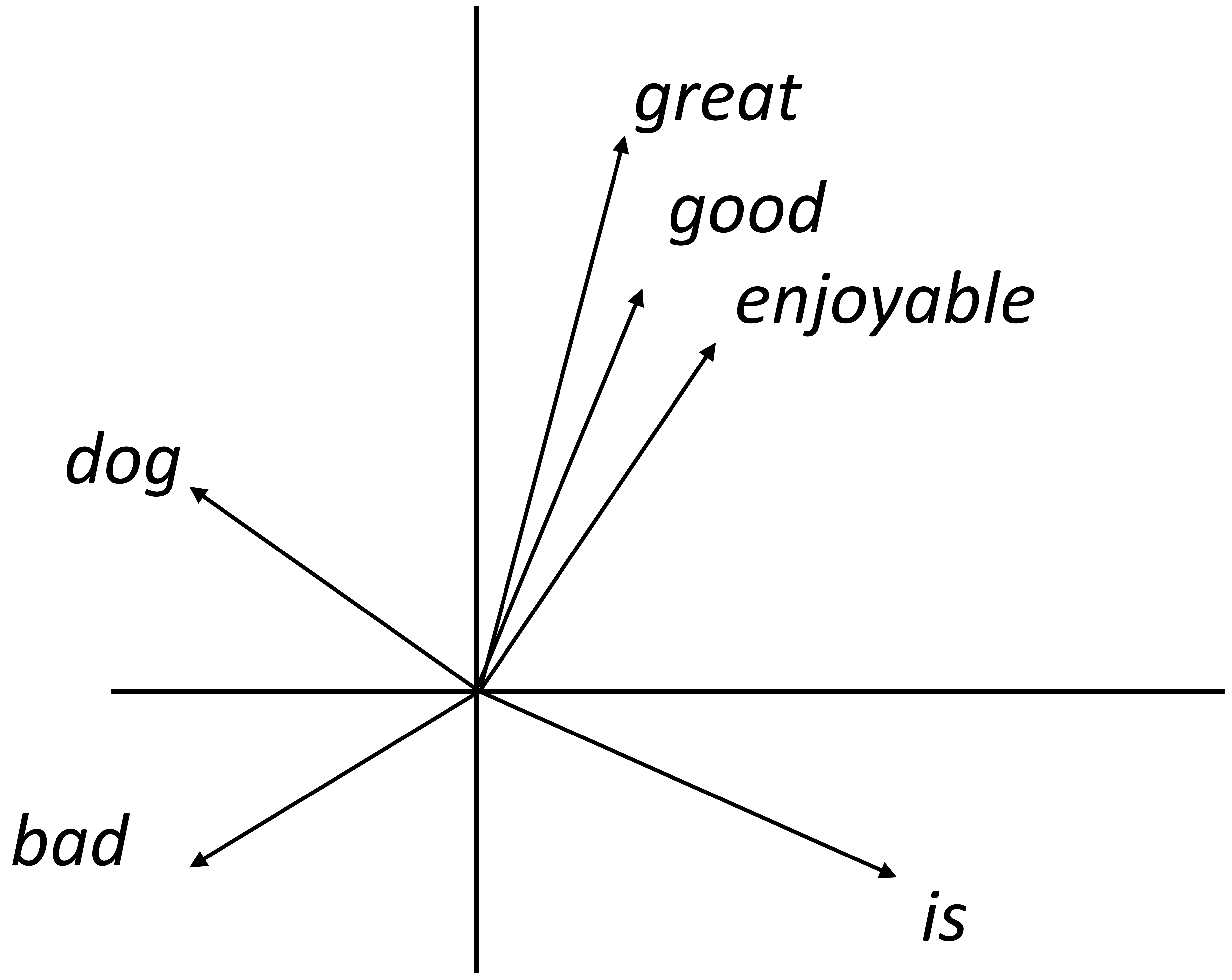
- Word embeddings for each word form input
- What properties should these vectors have?



Botha et al. (2017)

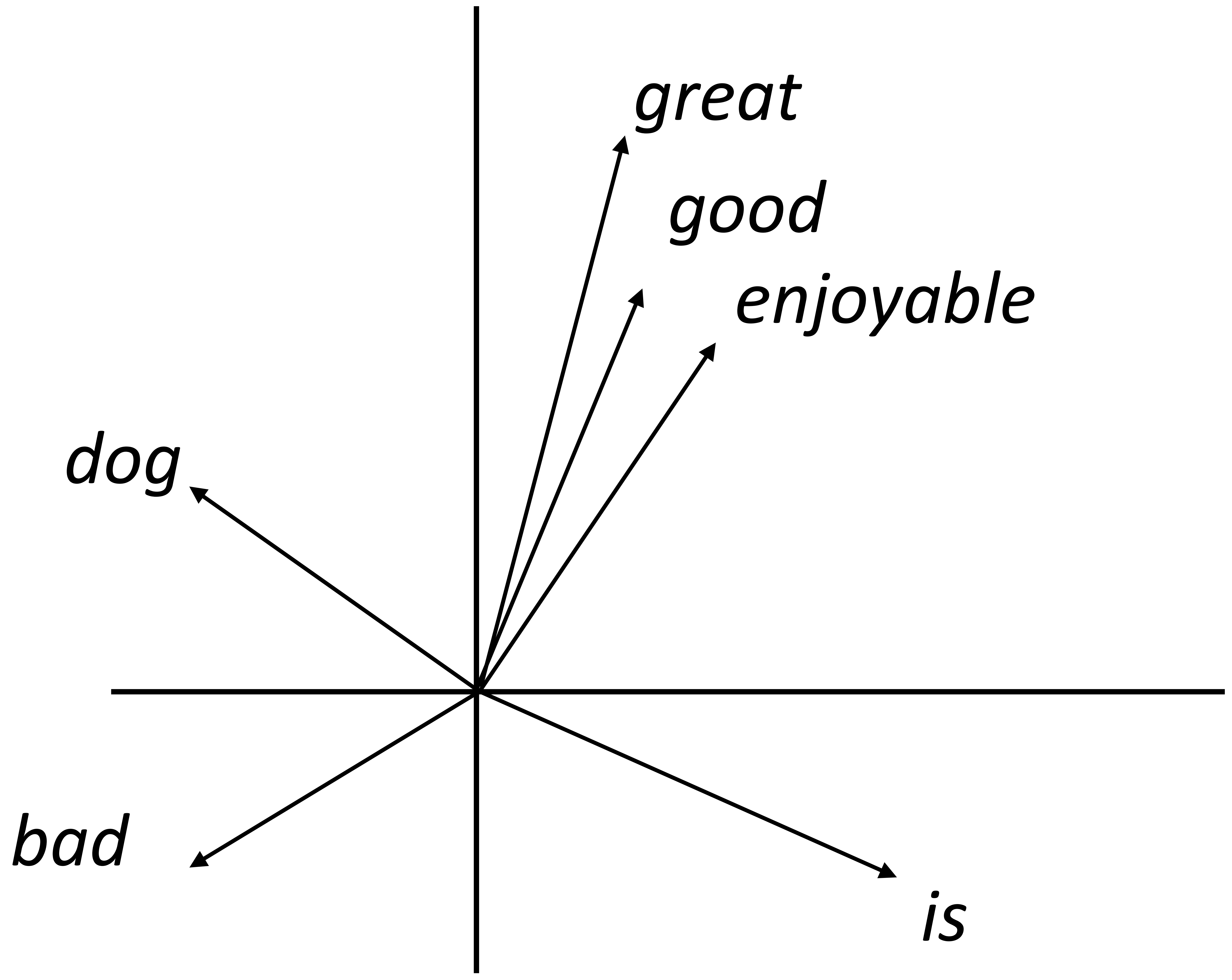
Word Embeddings

Word Embeddings



Word Embeddings

- Want a vector space where similar words have similar embeddings



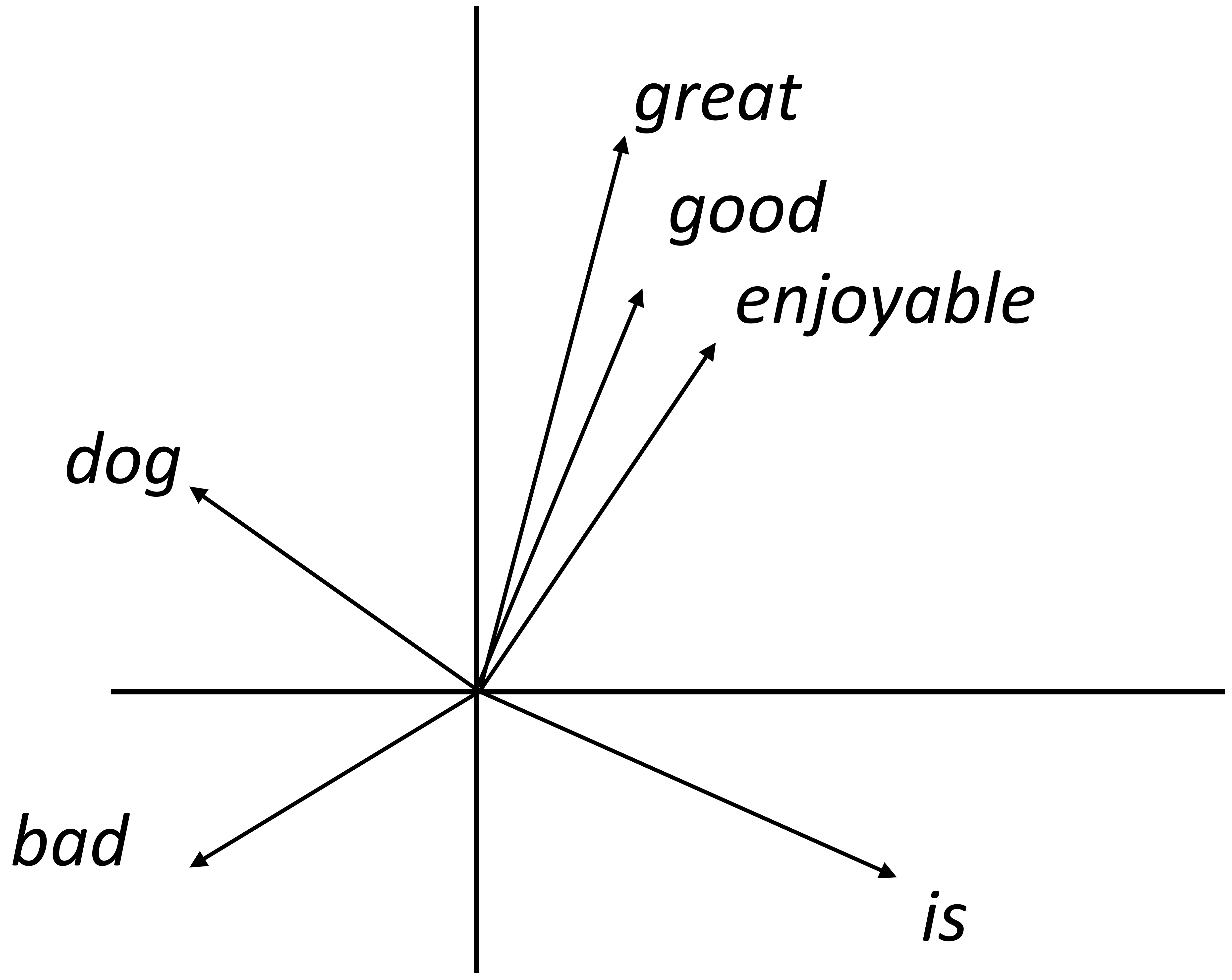
Word Embeddings

- Want a vector space where similar words have similar embeddings

the movie was great

≈

the movie was good



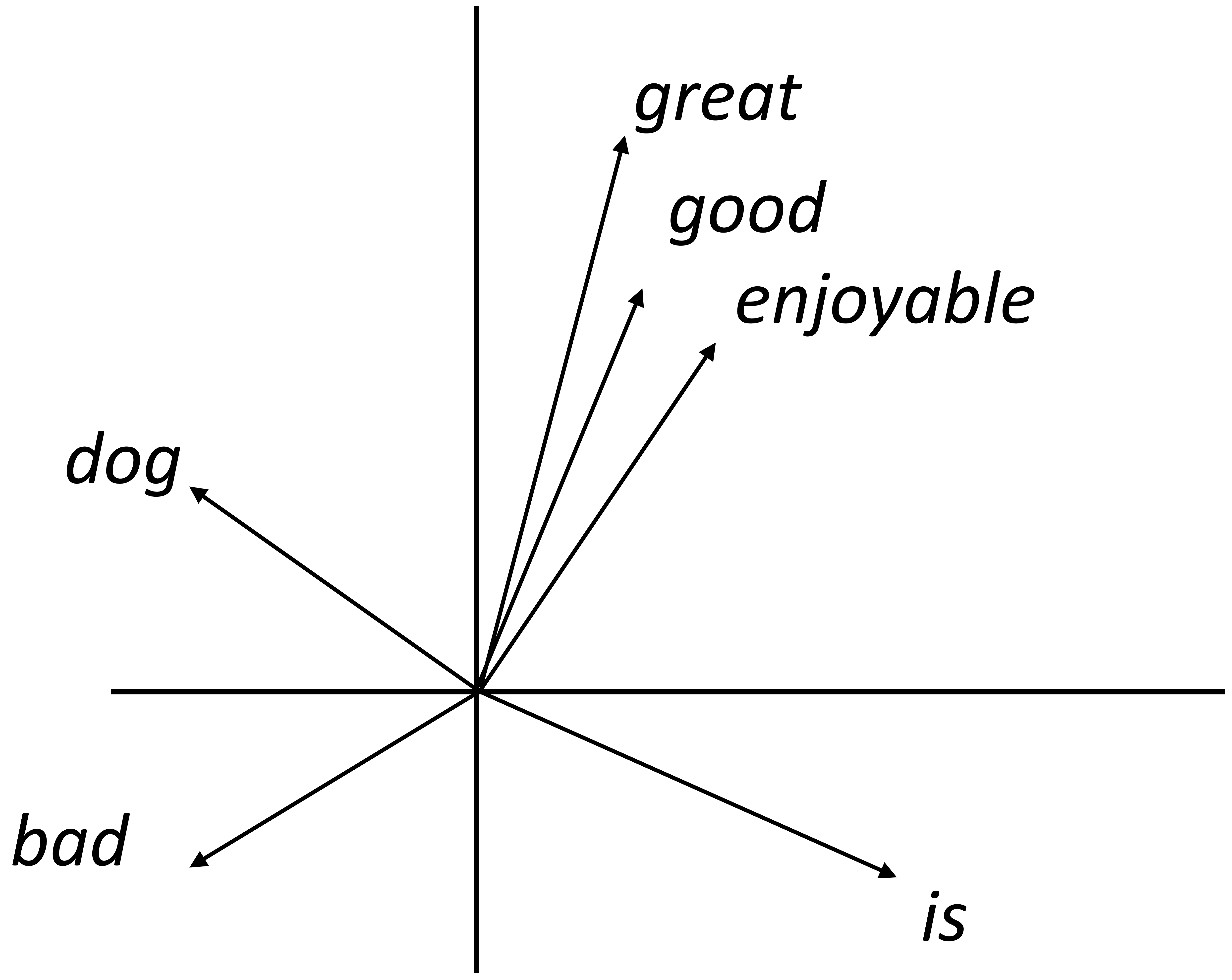
Word Embeddings

- 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

word2vec/GloVe

Continuous Bag-of-Words

- Predict word from context

the dog bit the man

Mikolov et al. (2013)

Continuous Bag-of-Words

- Predict word from context

the dog bit the man

dog

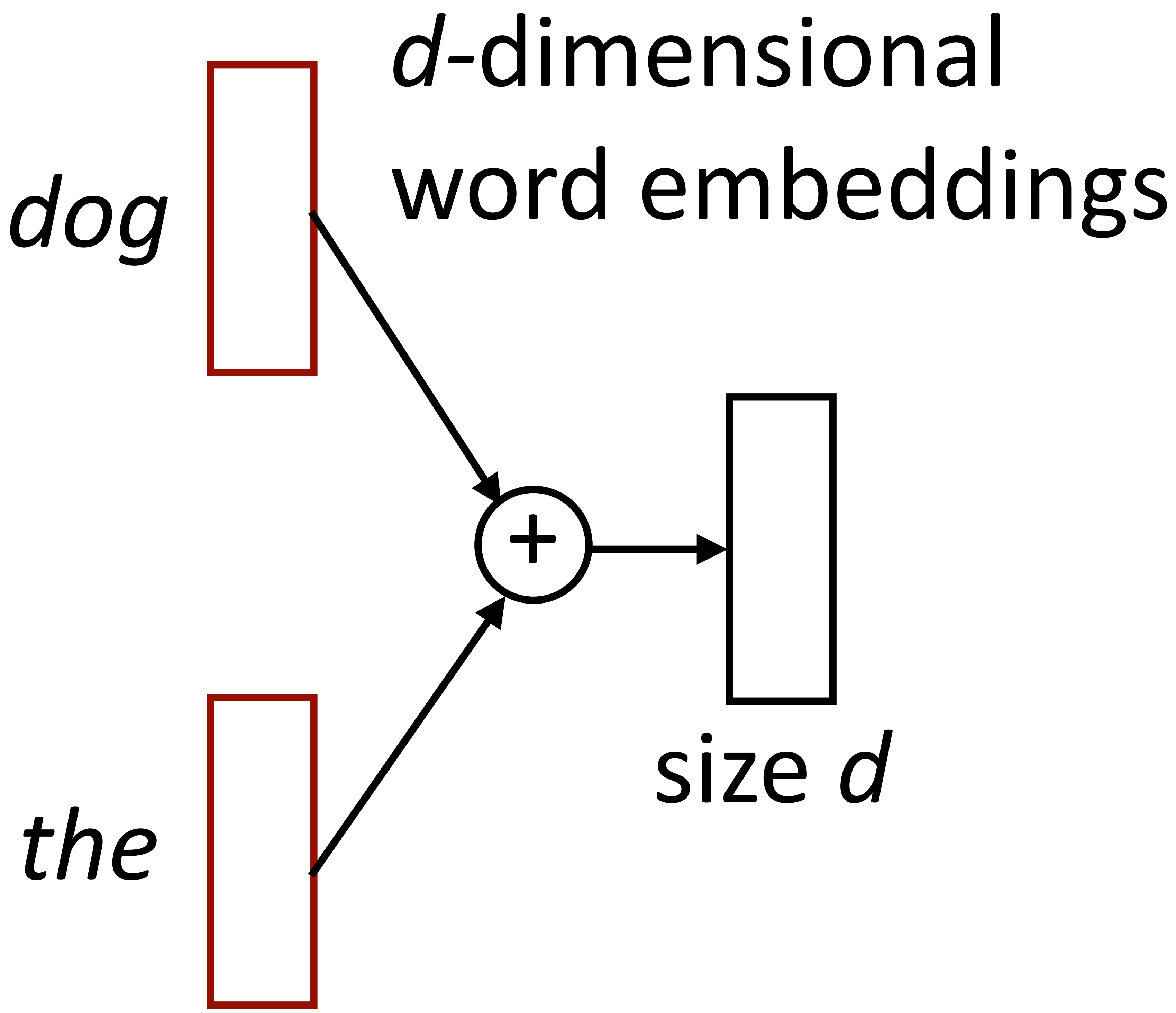
d-dimensional word embeddings

the

Mikolov et al. (2013)

Continuous Bag-of-Words

- Predict word from context

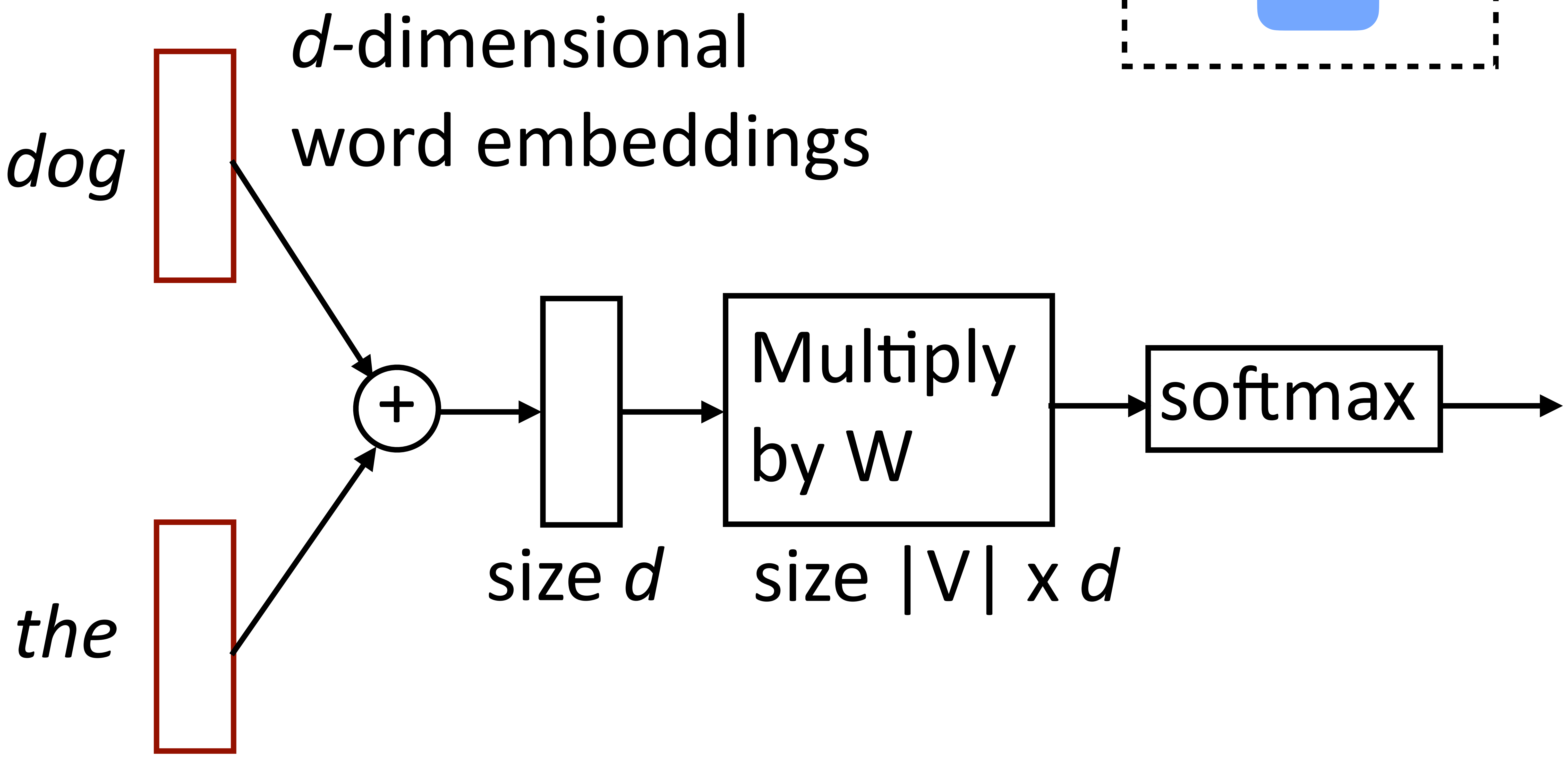


Mikolov et al. (2013)

Continuous Bag-of-Words

- Predict word from context

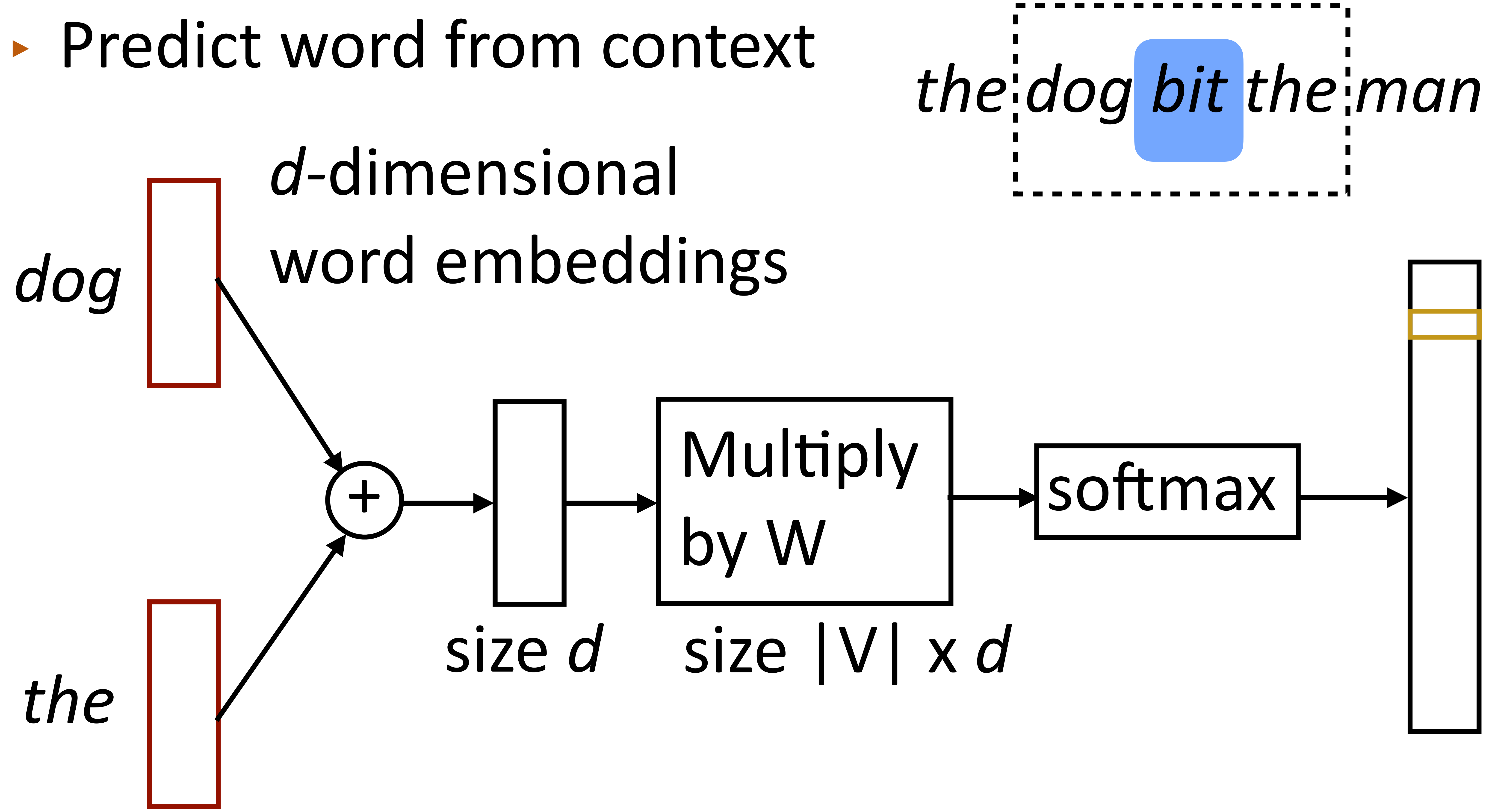
the dog bit the man



Mikolov et al. (2013)

Continuous Bag-of-Words

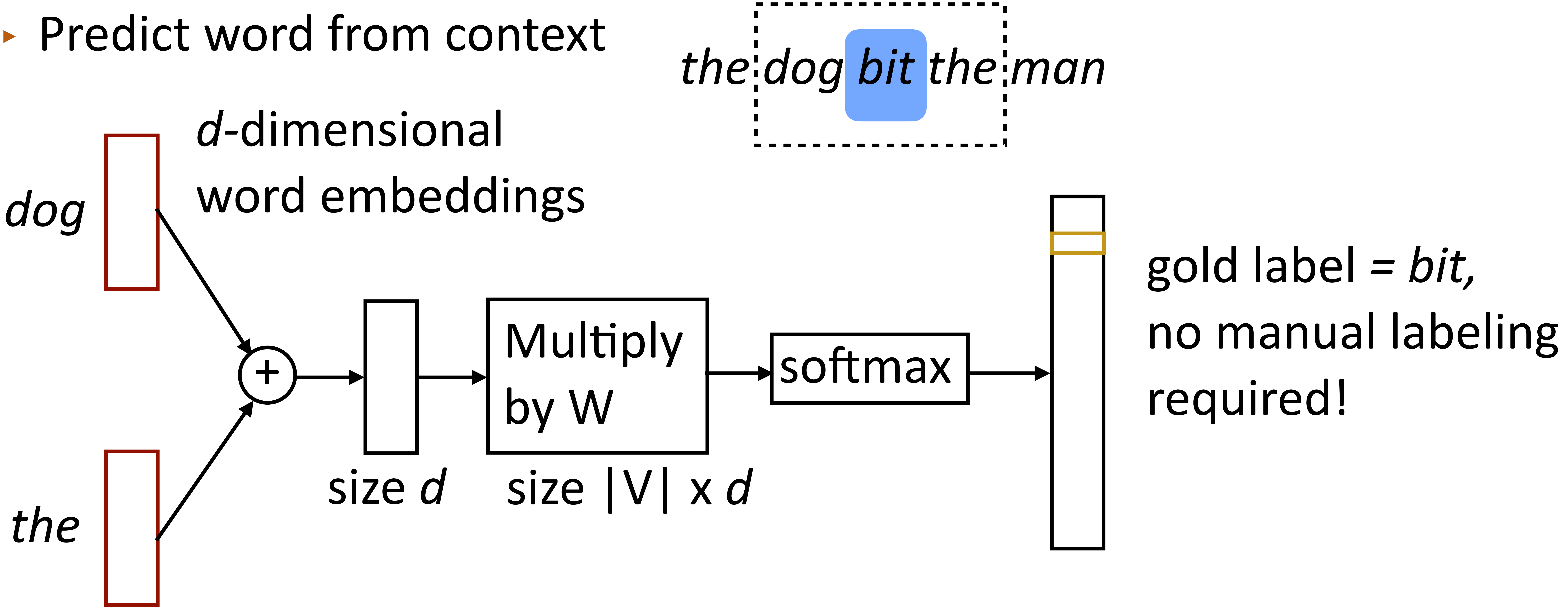
- Predict word from context



Mikolov et al. (2013)

Continuous Bag-of-Words

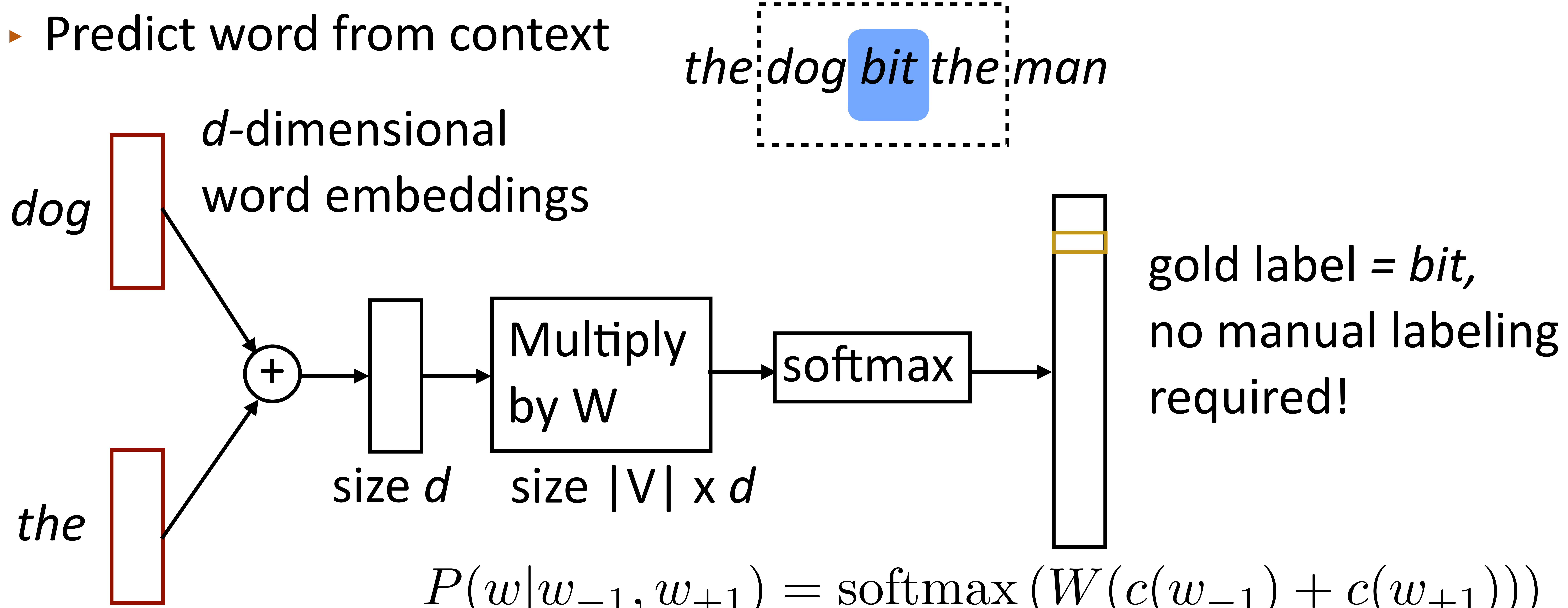
- Predict word from context



Mikolov et al. (2013)

Continuous Bag-of-Words

- Predict word from context



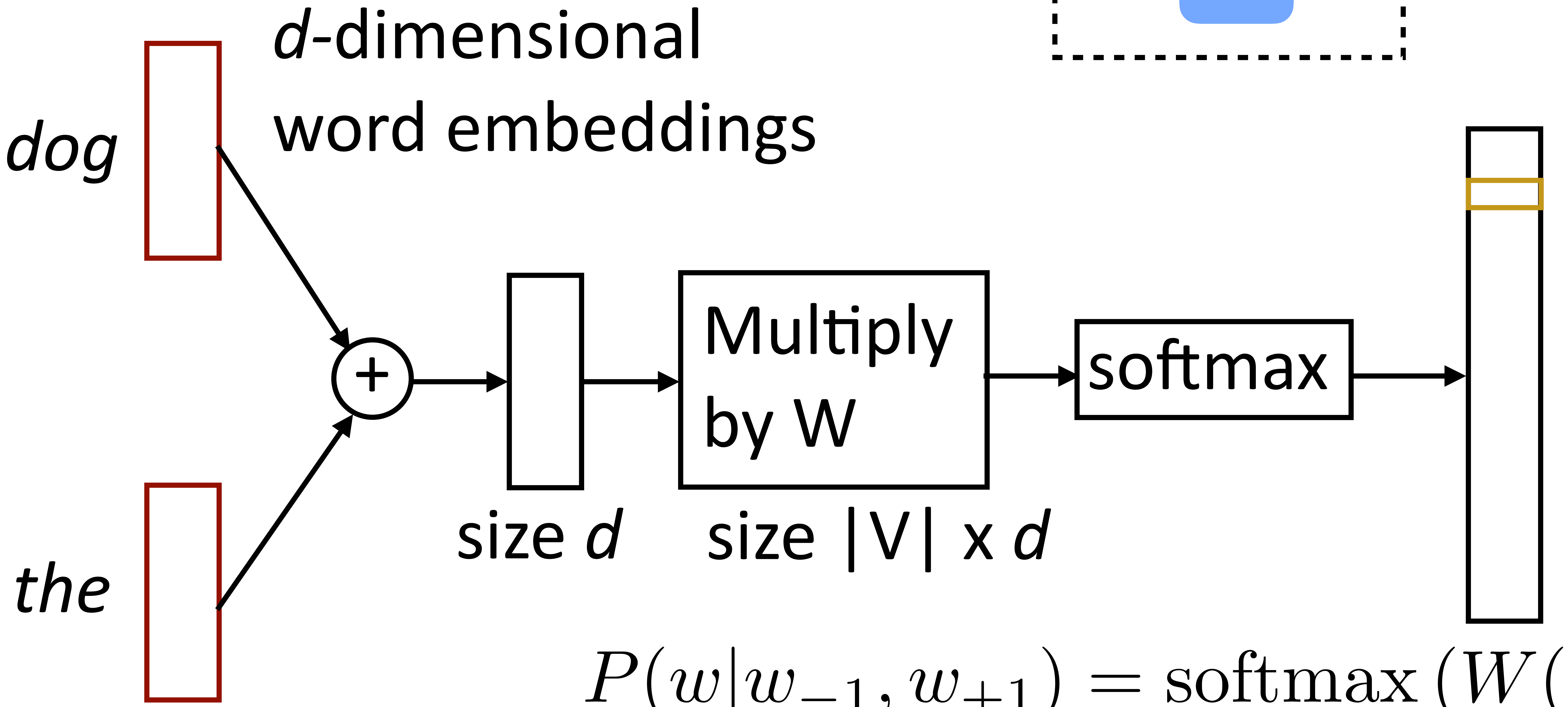
$$P(w|w_{-1}, w_{+1}) = \mathrm{softmax}\left(W(c(w_{-1}) + c(w_{+1}))\right)$$

Mikolov et al. (2013)

Continuous Bag-of-Words

- Predict word from context

the dog **bit** *the man*



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

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

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

Mikolov et al. (2013)

Skip-Gram

- Predict one word of context from word

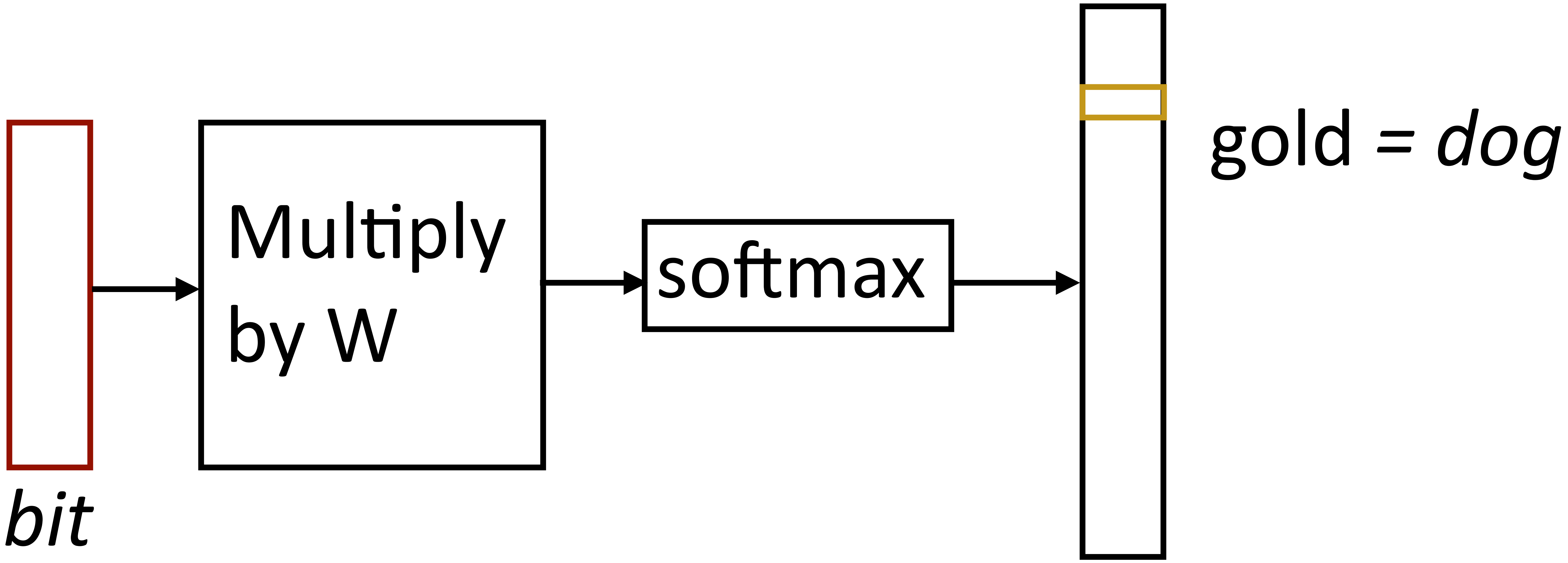
the dog bit the man

Mikolov et al. (2013)

Skip-Gram

- Predict one word of context from word

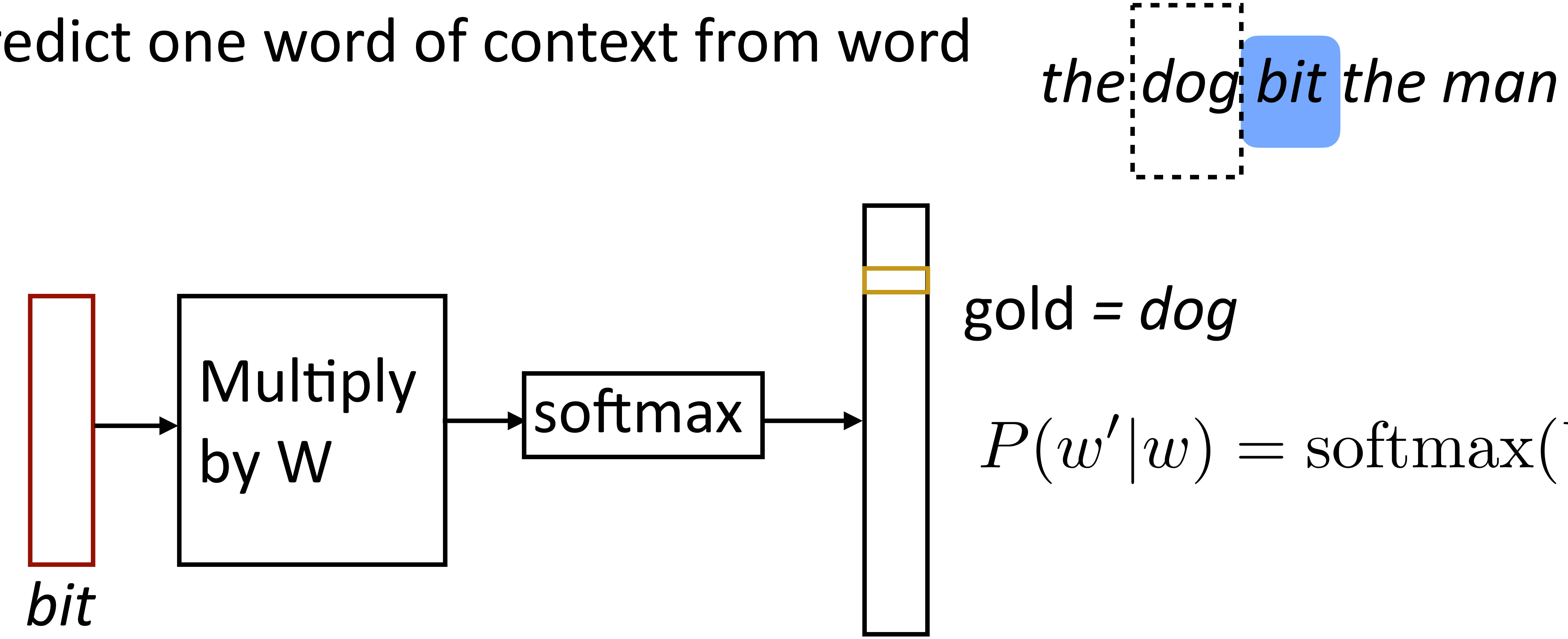
the dog bit the man



Mikolov et al. (2013)

Skip-Gram

- Predict one word of context from word



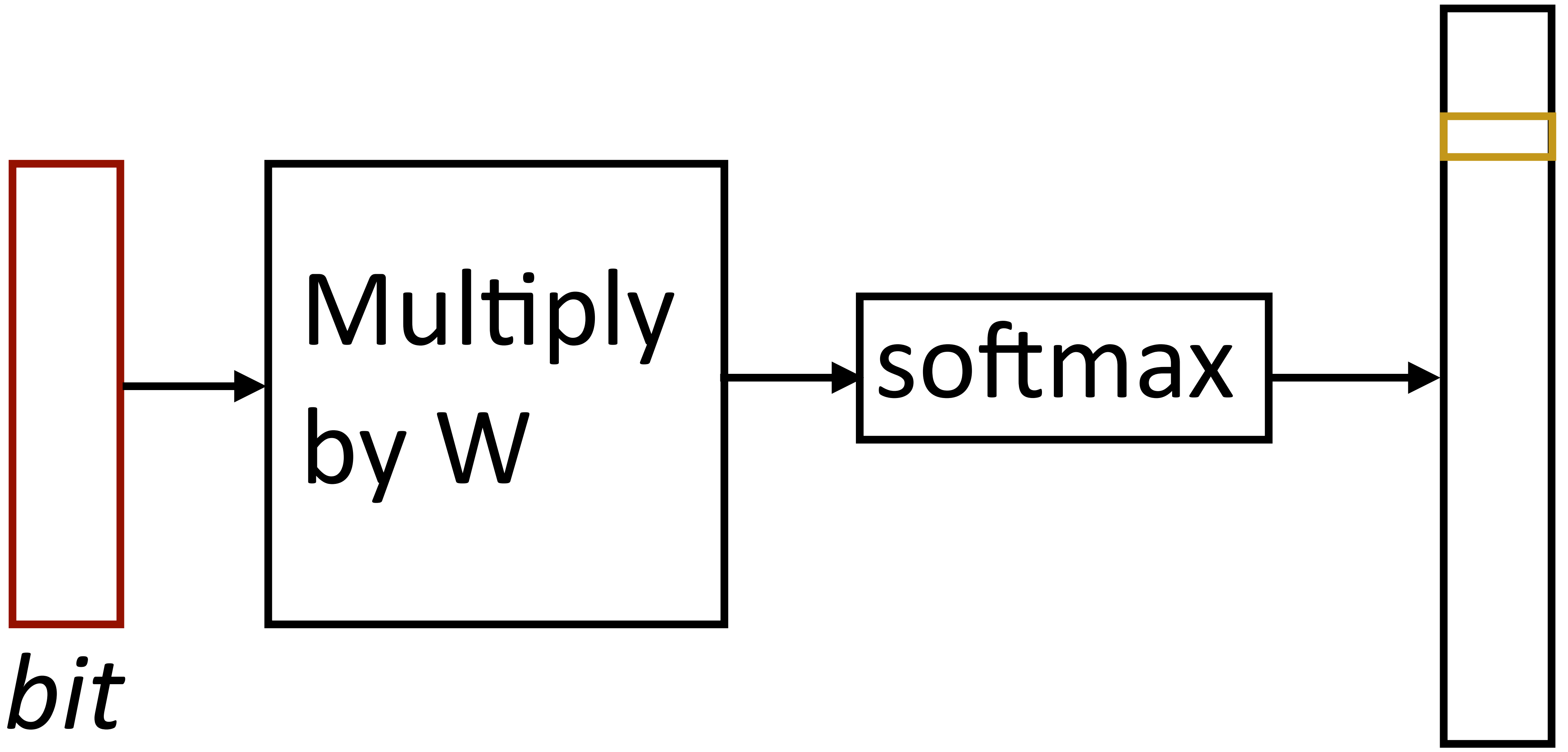
$$P(w'|w) = \mathrm{softmax}(We(w))$$

Mikolov et al. (2013)

Skip-Gram

- Predict one word of context from word

the dog bit the man



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

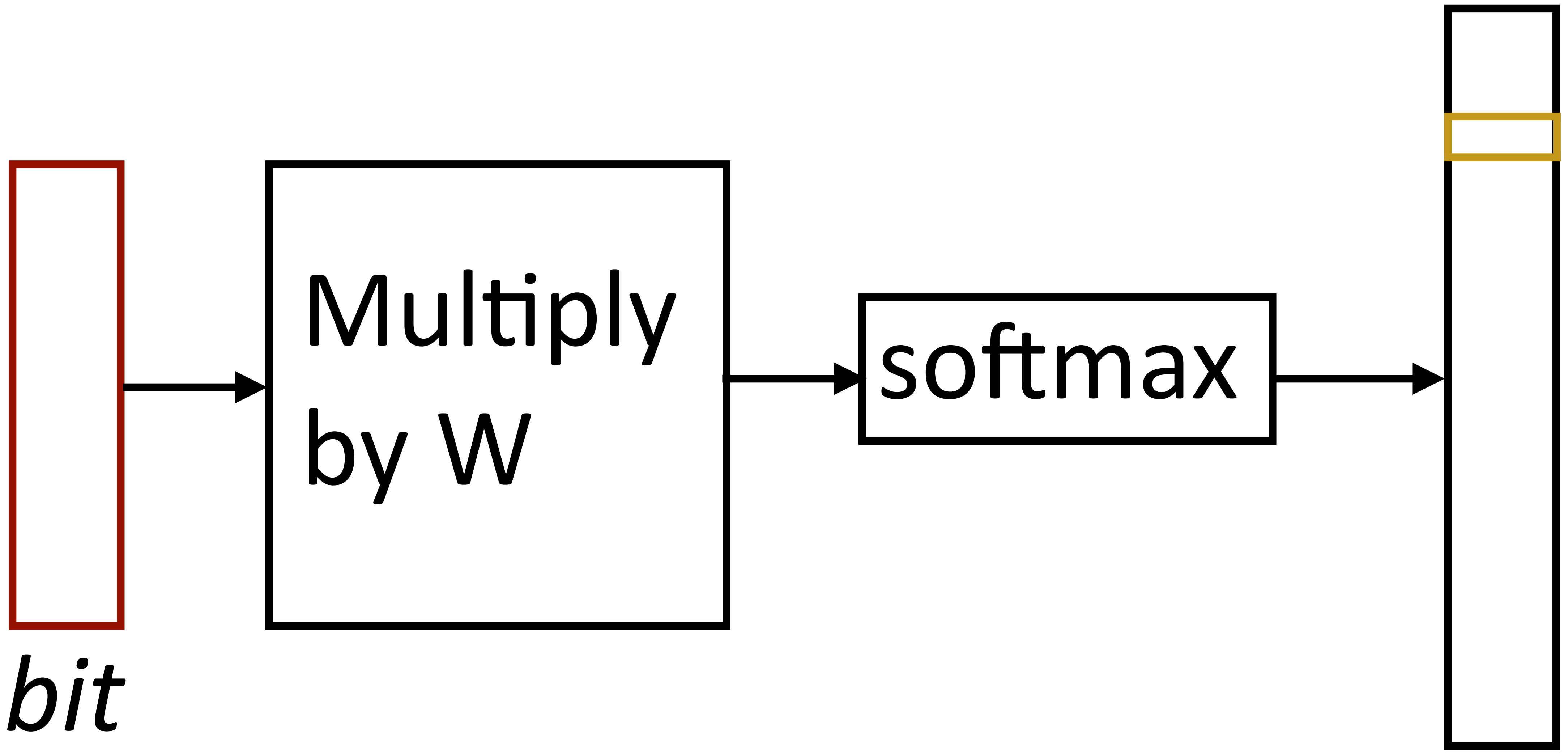
- Another training example: *bit -> the*

Mikolov et al. (2013)

Skip-Gram

- Predict one word of context from word

the dog bit the man



gold = *dog*

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

- 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

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

Mikolov et al. (2013)

Hierarchical Softmax

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

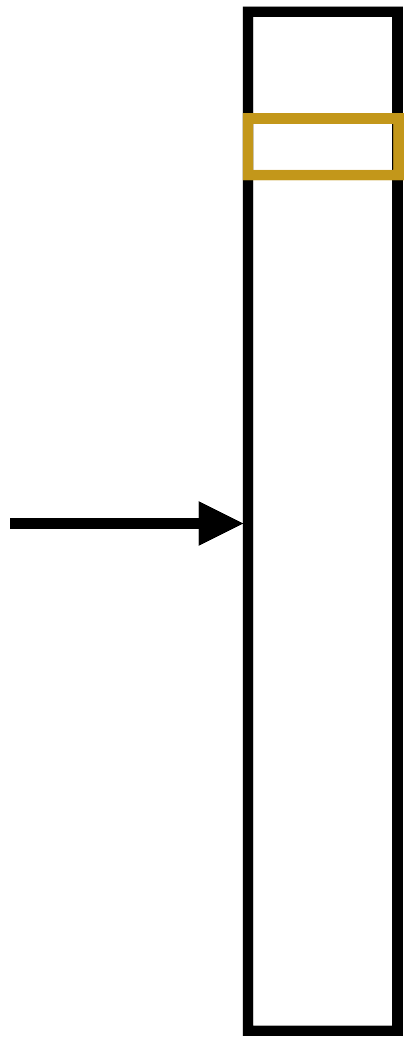
- Matmul + softmax over |V| is very slow to compute for CBOW and SG

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



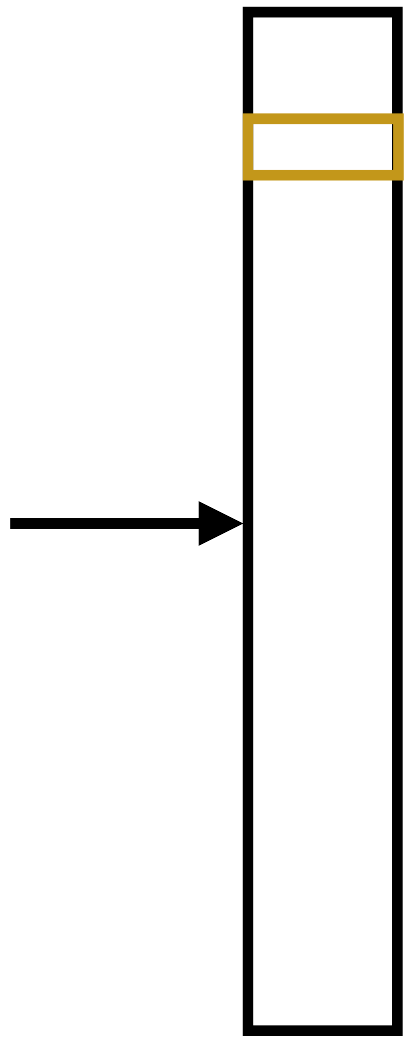
- Standard softmax:
  [|V| x *d*]

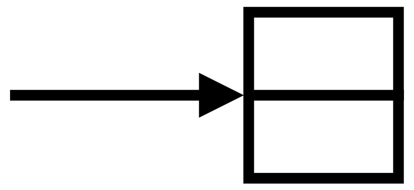
Mikolov et al. (2013)

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





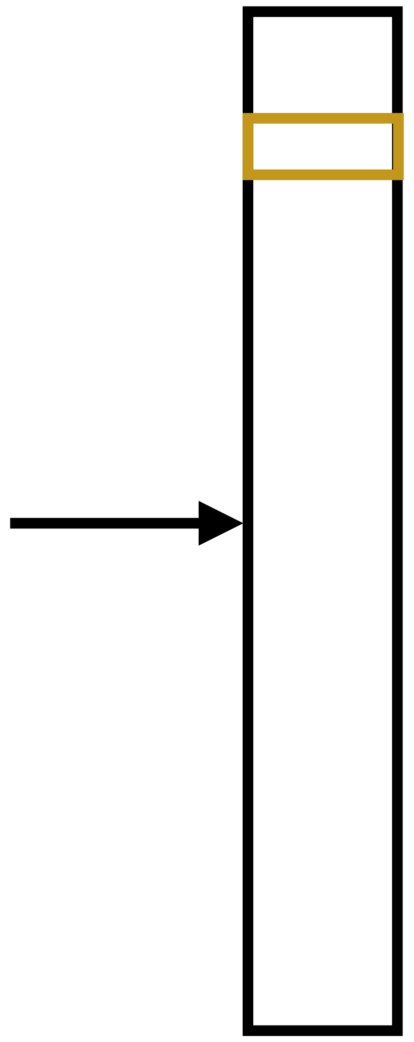
- Standard softmax:
  [|V| x *d*]

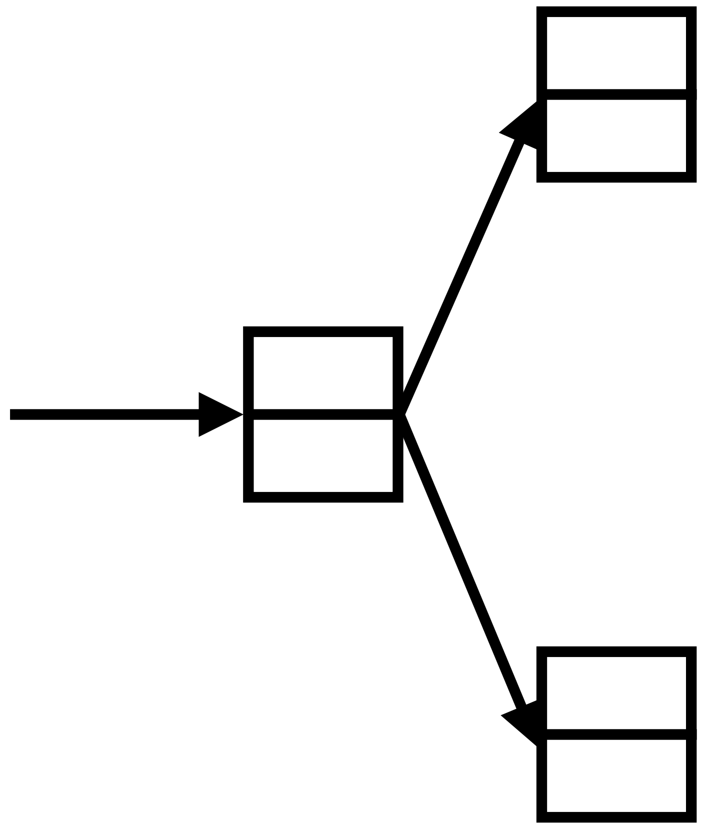
Mikolov et al. (2013)

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





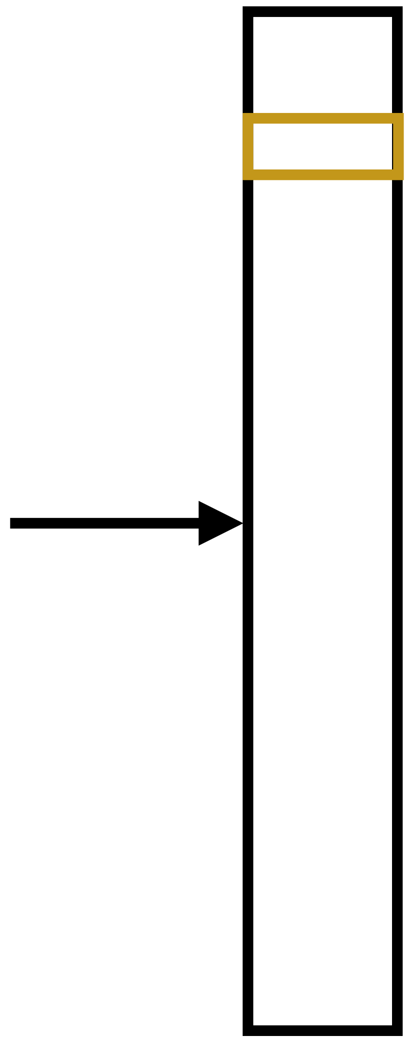
- Standard softmax:
  [|V| x *d*]

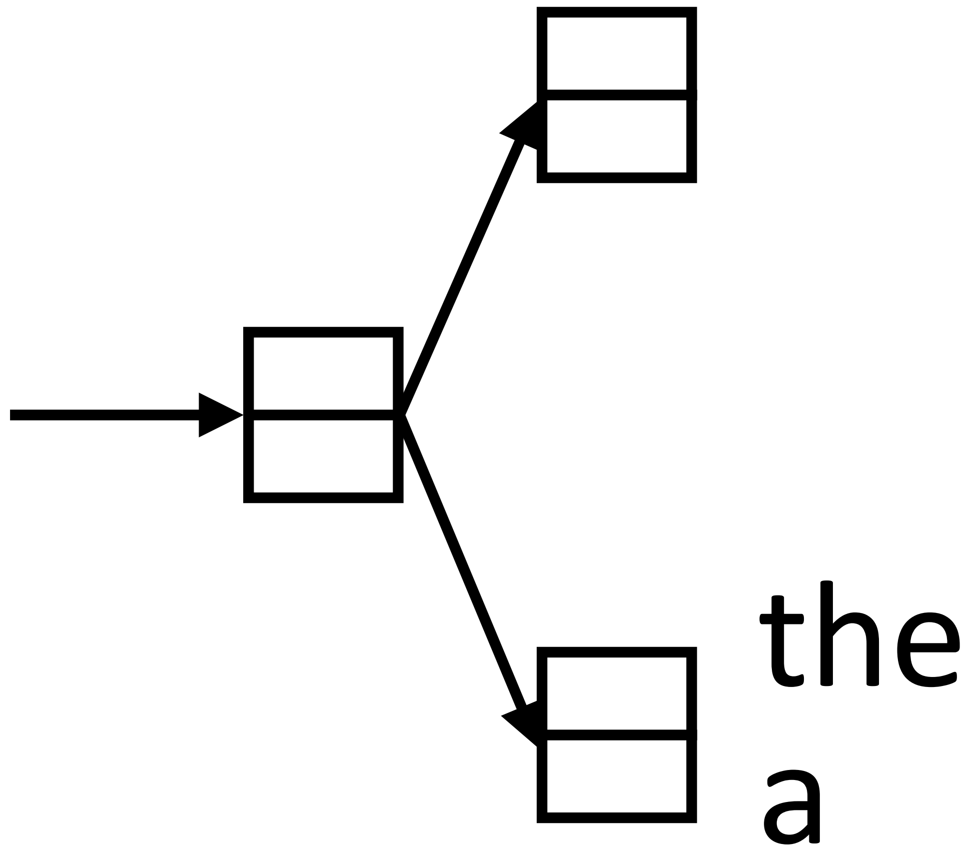
Mikolov et al. (2013)

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





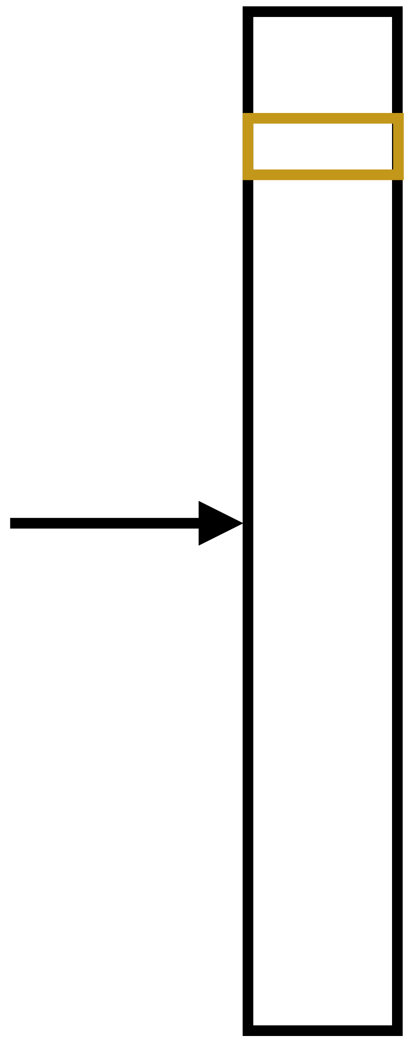
- Standard softmax:
  [|V| x *d*]

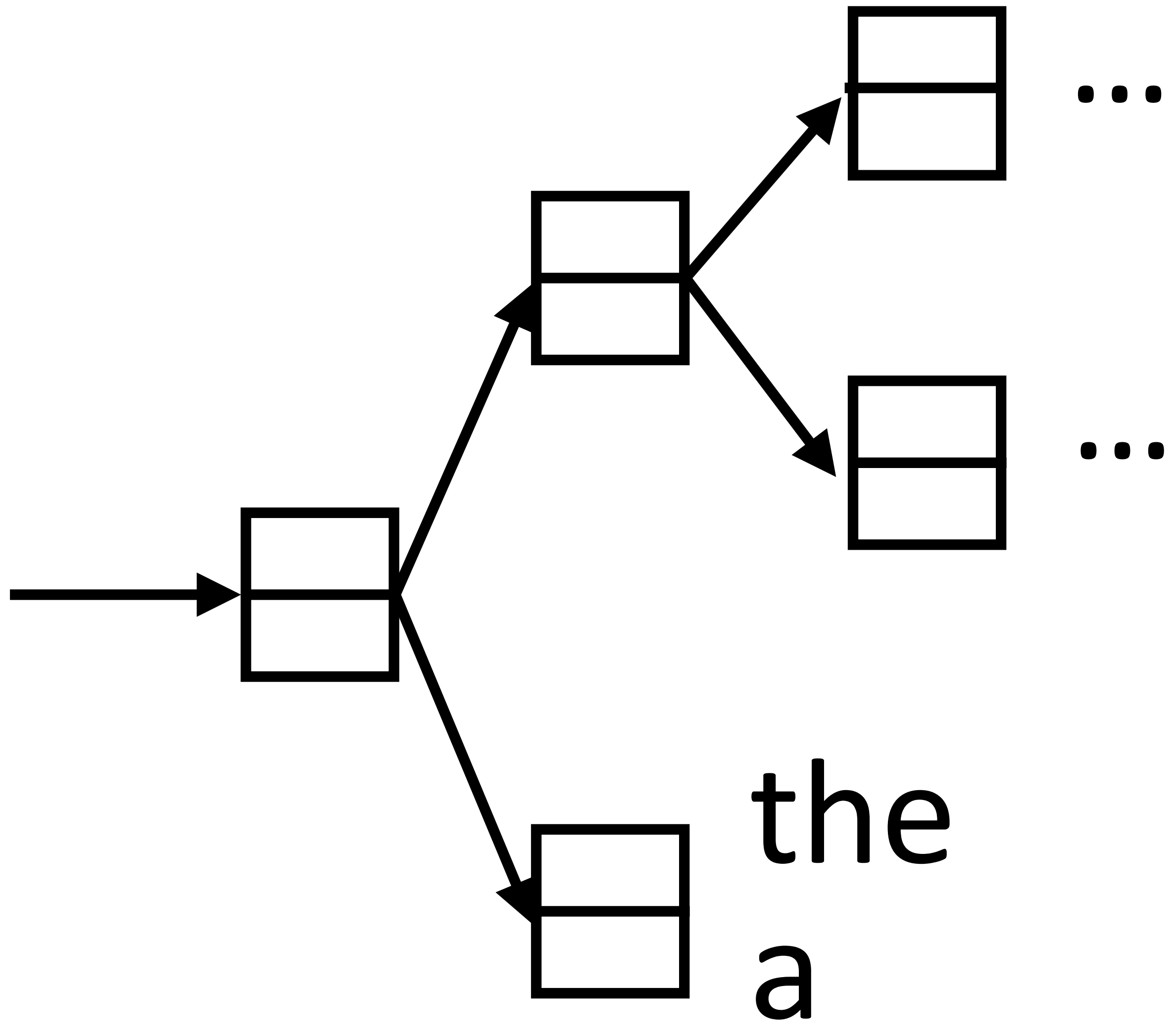
Mikolov et al. (2013)

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





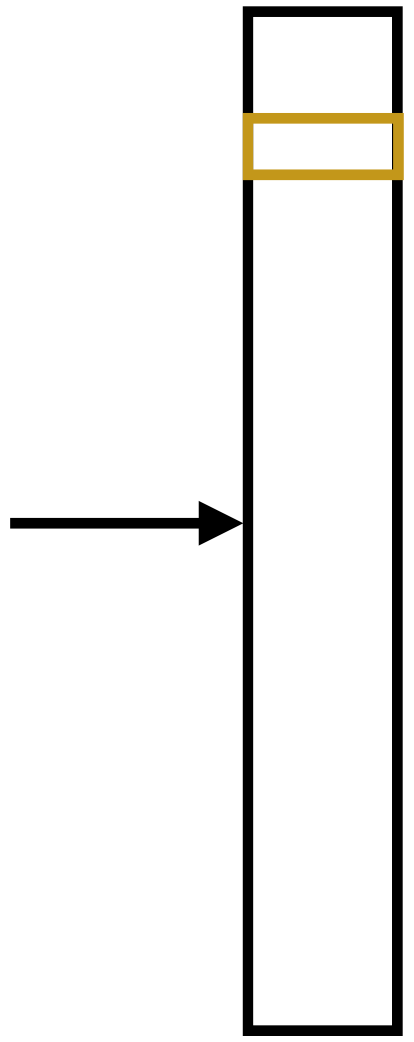
- Standard softmax:
  [|V| x *d*]

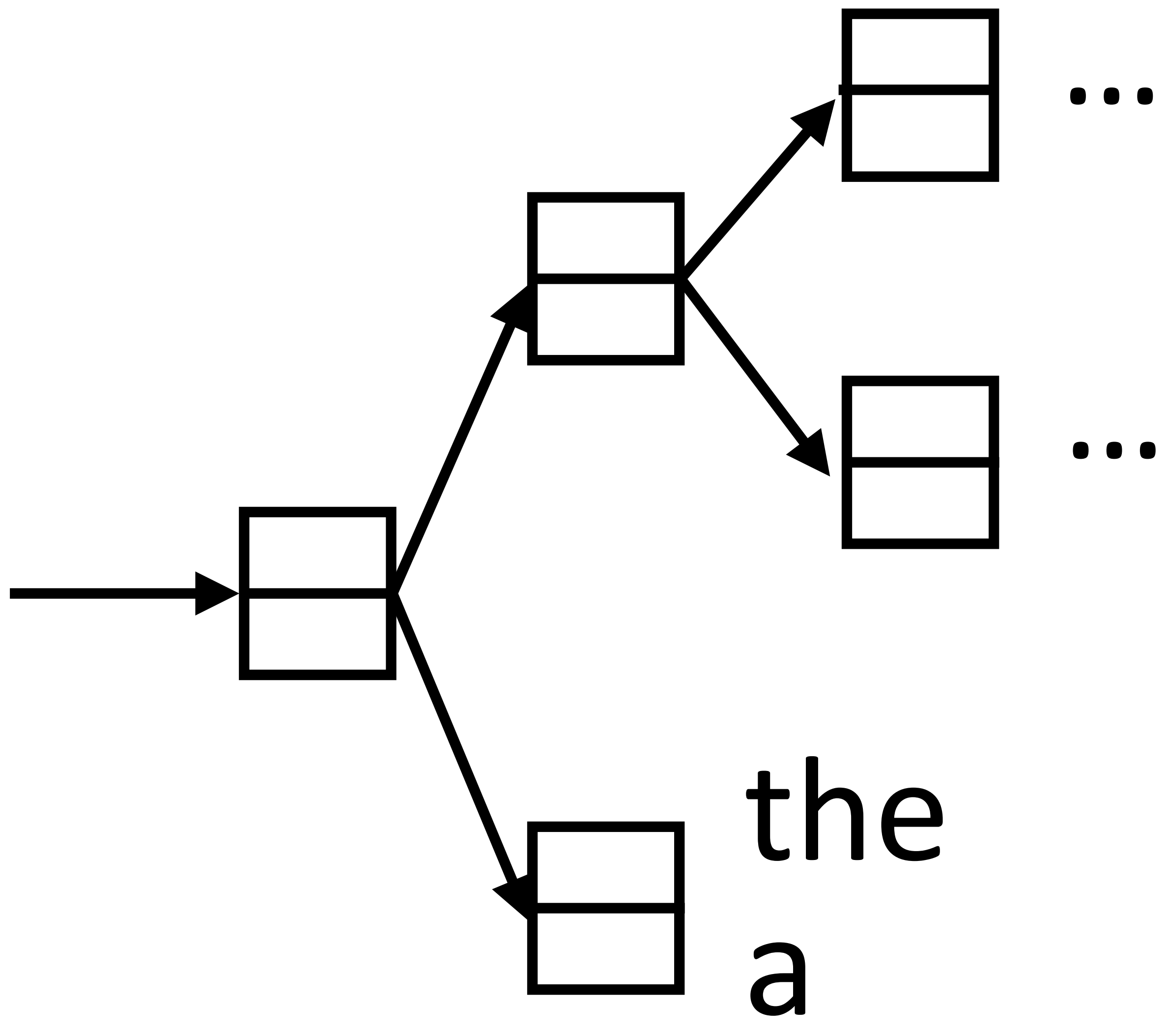
Mikolov et al. (2013)

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

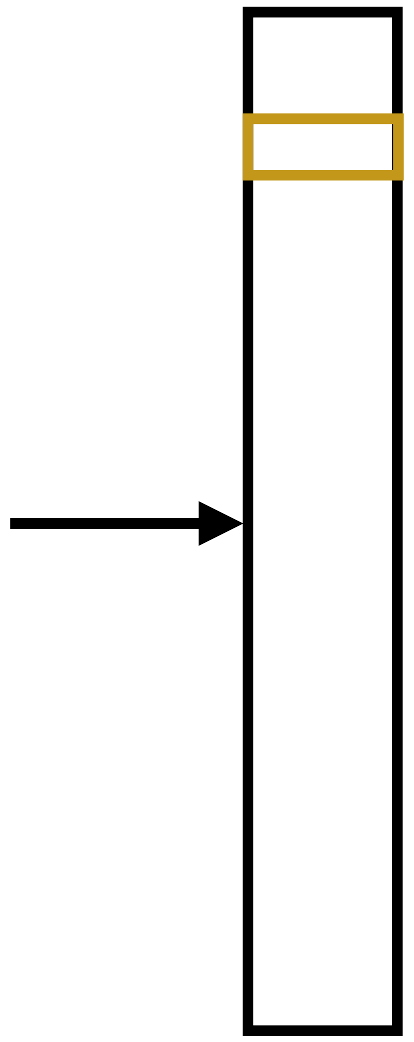
- Standard softmax: [|V| x *d*]

Mikolov et al. (2013)

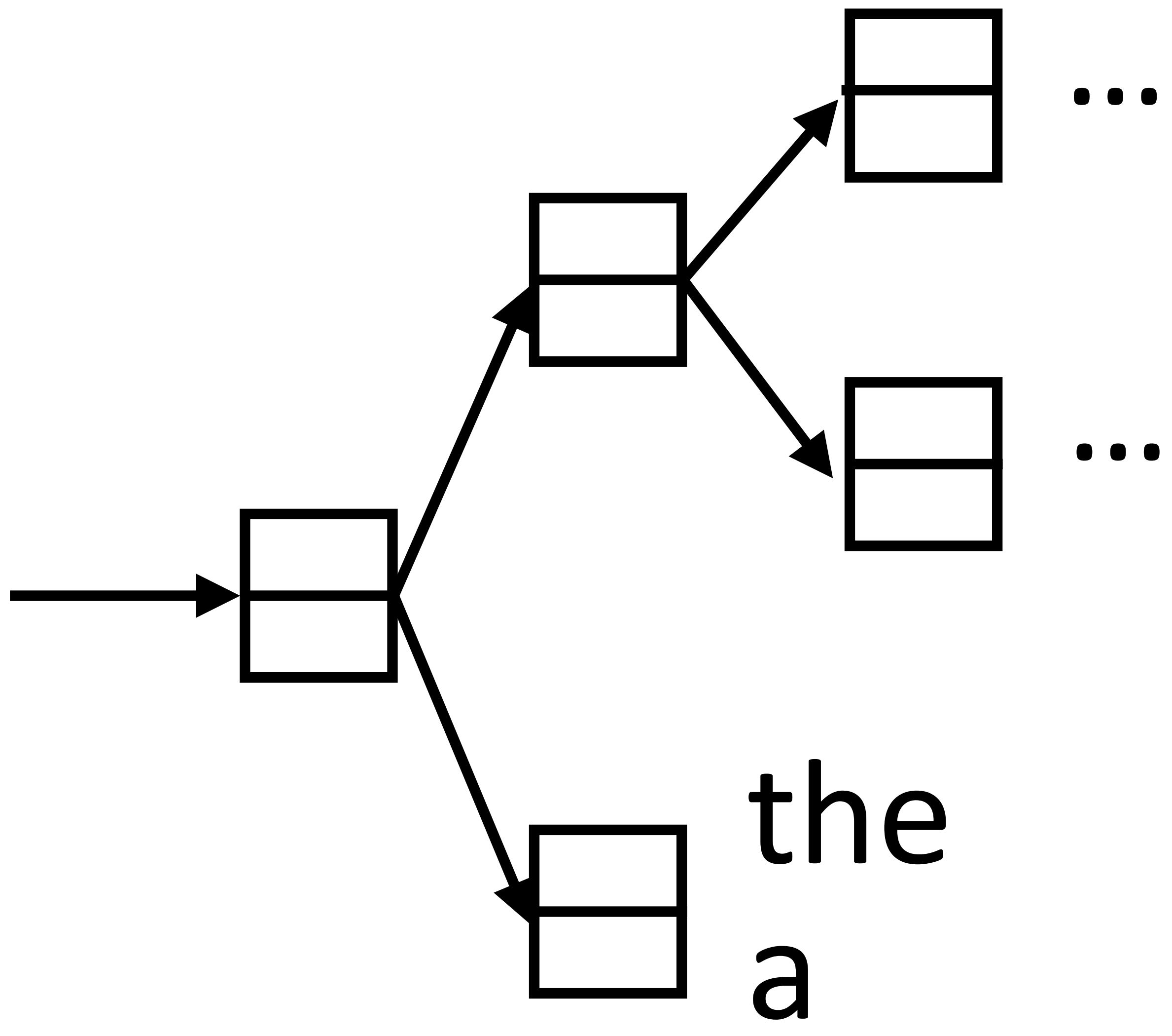
Hierarchical Softmax

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

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



- Standard softmax: [|V| x *d*]



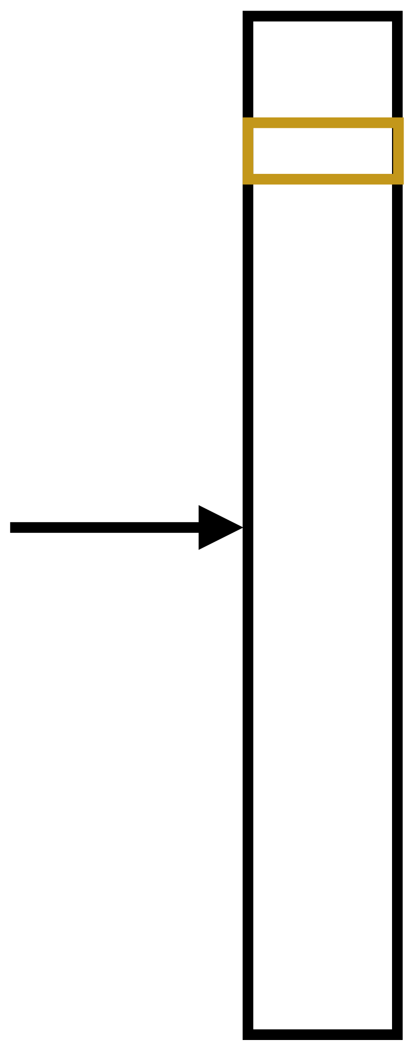
- Huffman encode vocabulary, use binary classifiers to decide which branch to take
- log(|V|) binary decisions

Mikolov et al. (2013)

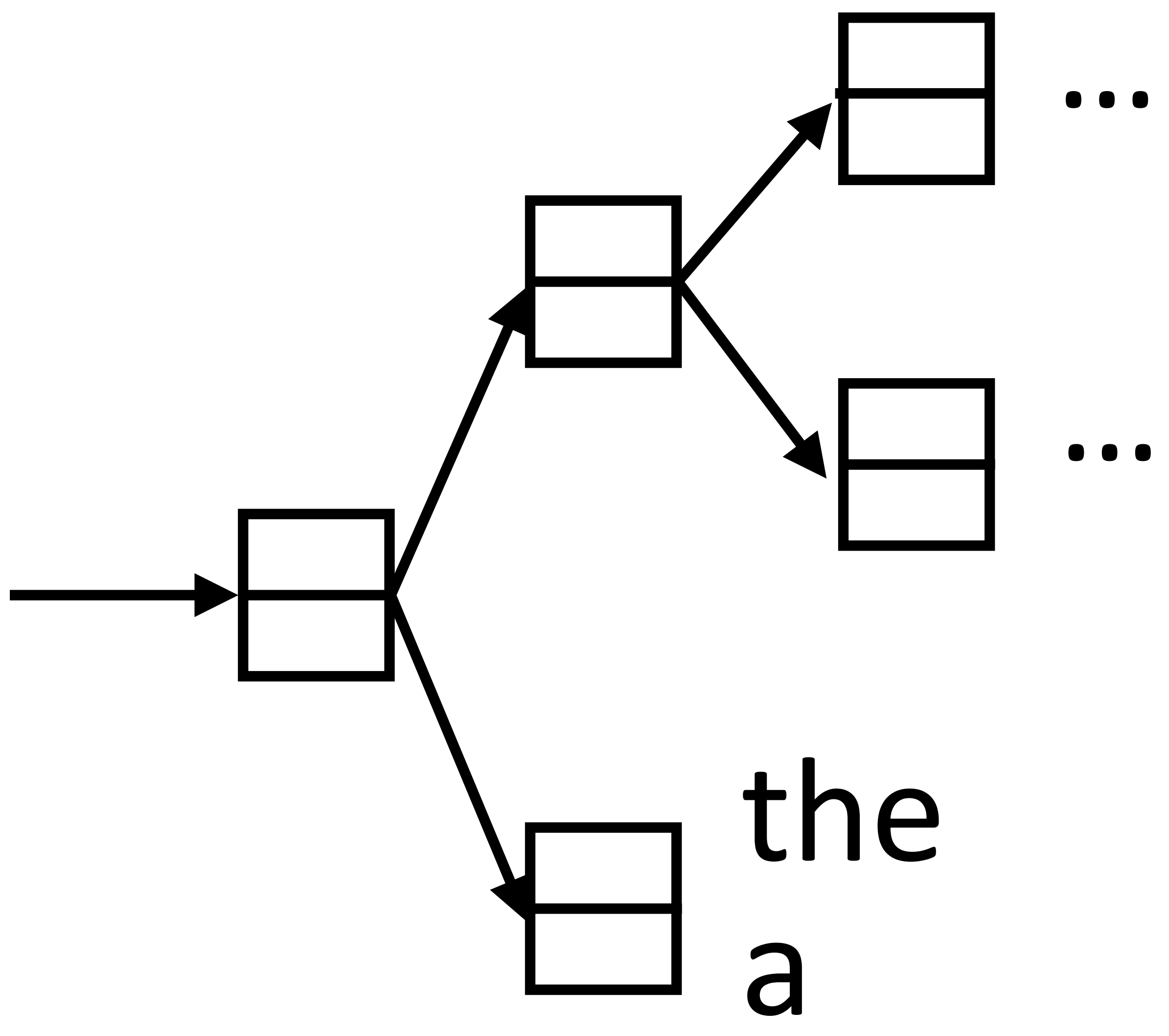
Hierarchical Softmax

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

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



- Standard softmax: [|V| x *d*]



- Hierarchical softmax:

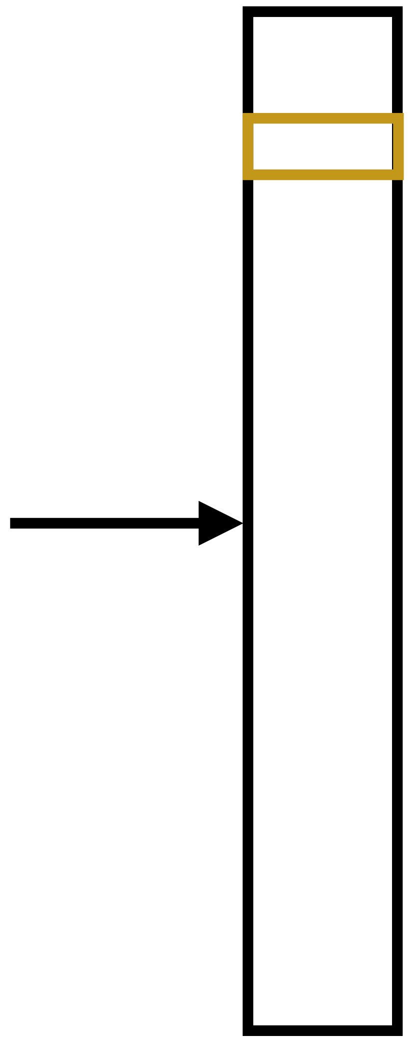
- Huffman encode vocabulary, use binary classifiers to decide which branch to take
- log(|V|) binary decisions

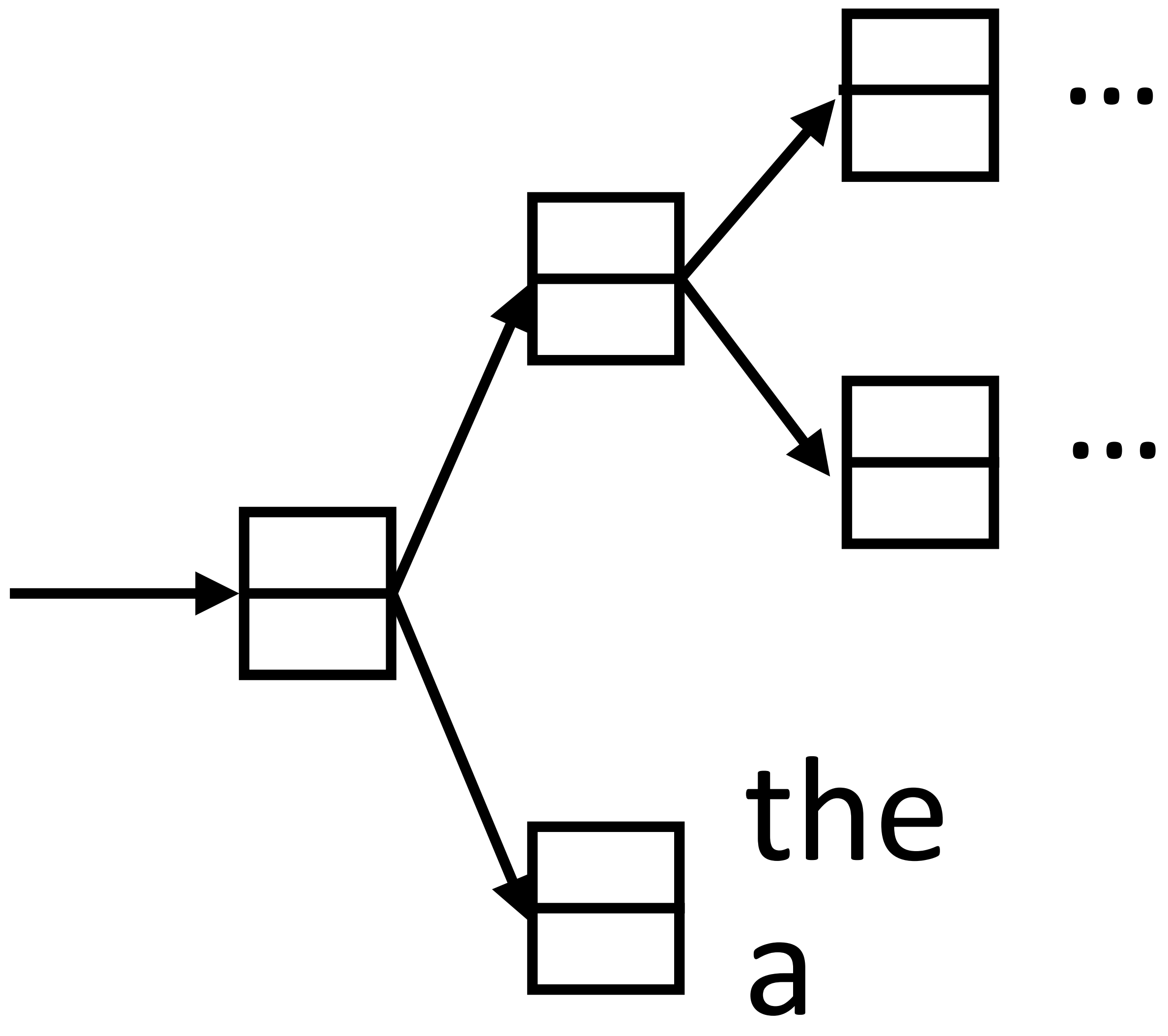
Mikolov et al. (2013)

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: [|V| x *d*]

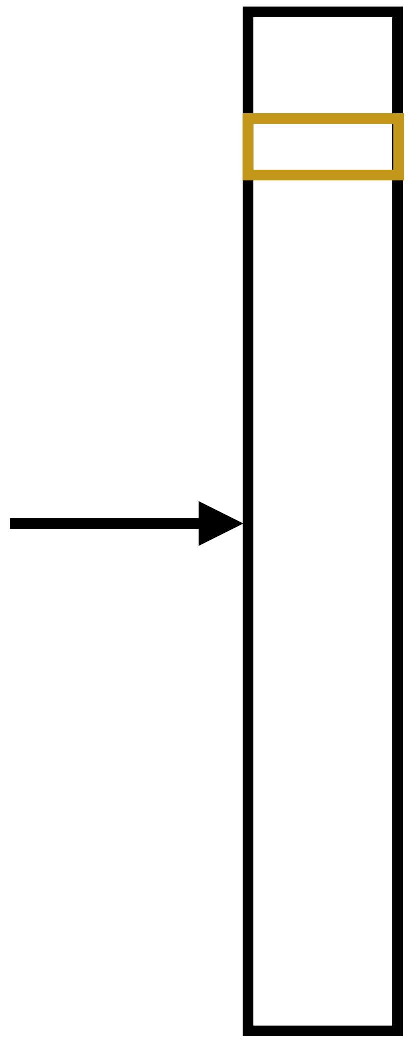
- Hierarchical softmax: log(|V|) dot products of size *d*,

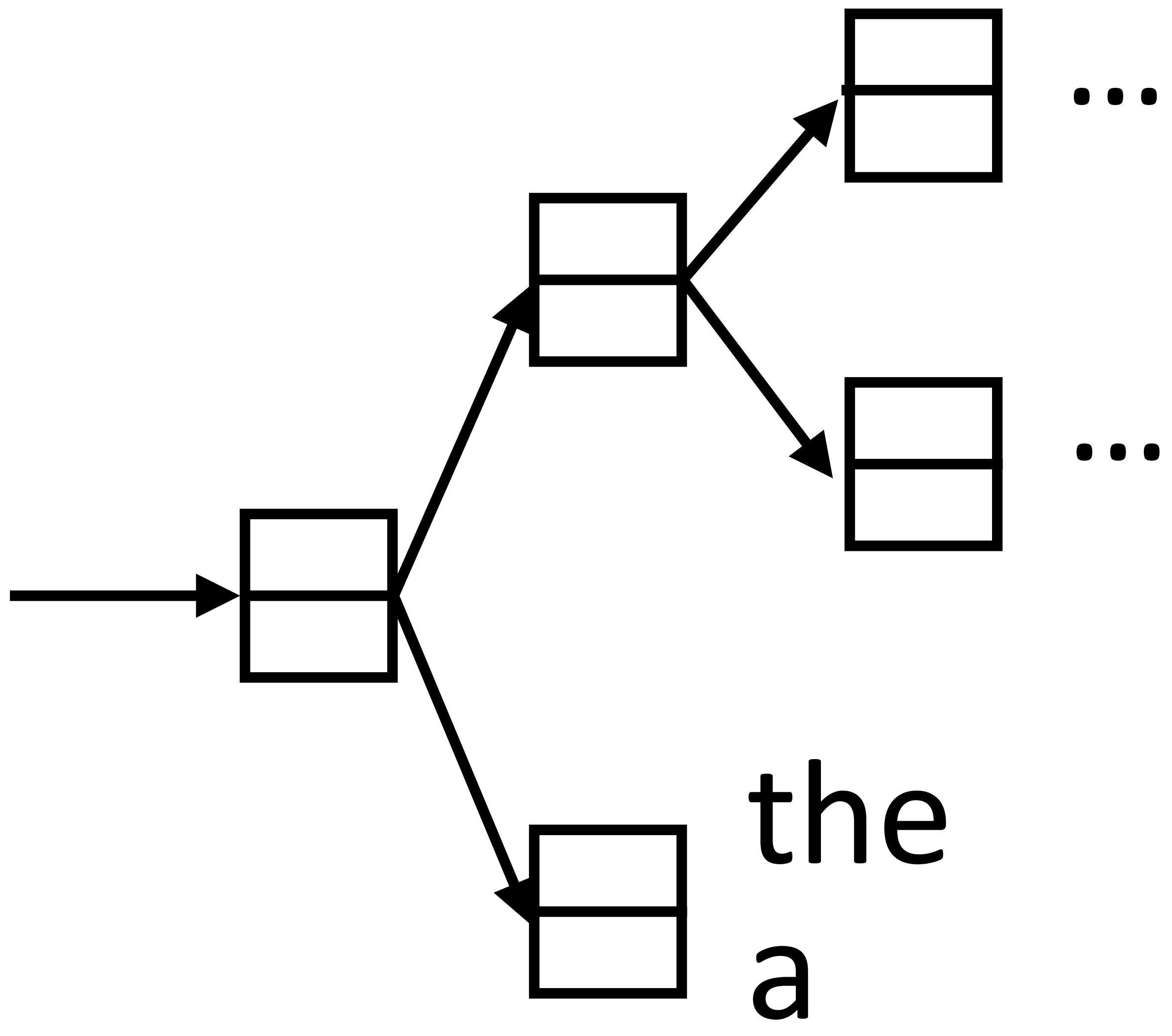
Mikolov et al. (2013)

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: [|V| x *d*]

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

Mikolov et al. (2013)

Skip-Gram with Negative Sampling

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

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

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

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

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

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

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

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

words in similar contexts select for similar *c* vectors

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

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

words in similar contexts select for similar *c* vectors

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

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

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

words in similar contexts select for similar *c* vectors

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

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

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

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

words in similar contexts select for similar *c* vectors

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

sampled

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

Mikolov et al. (2013)

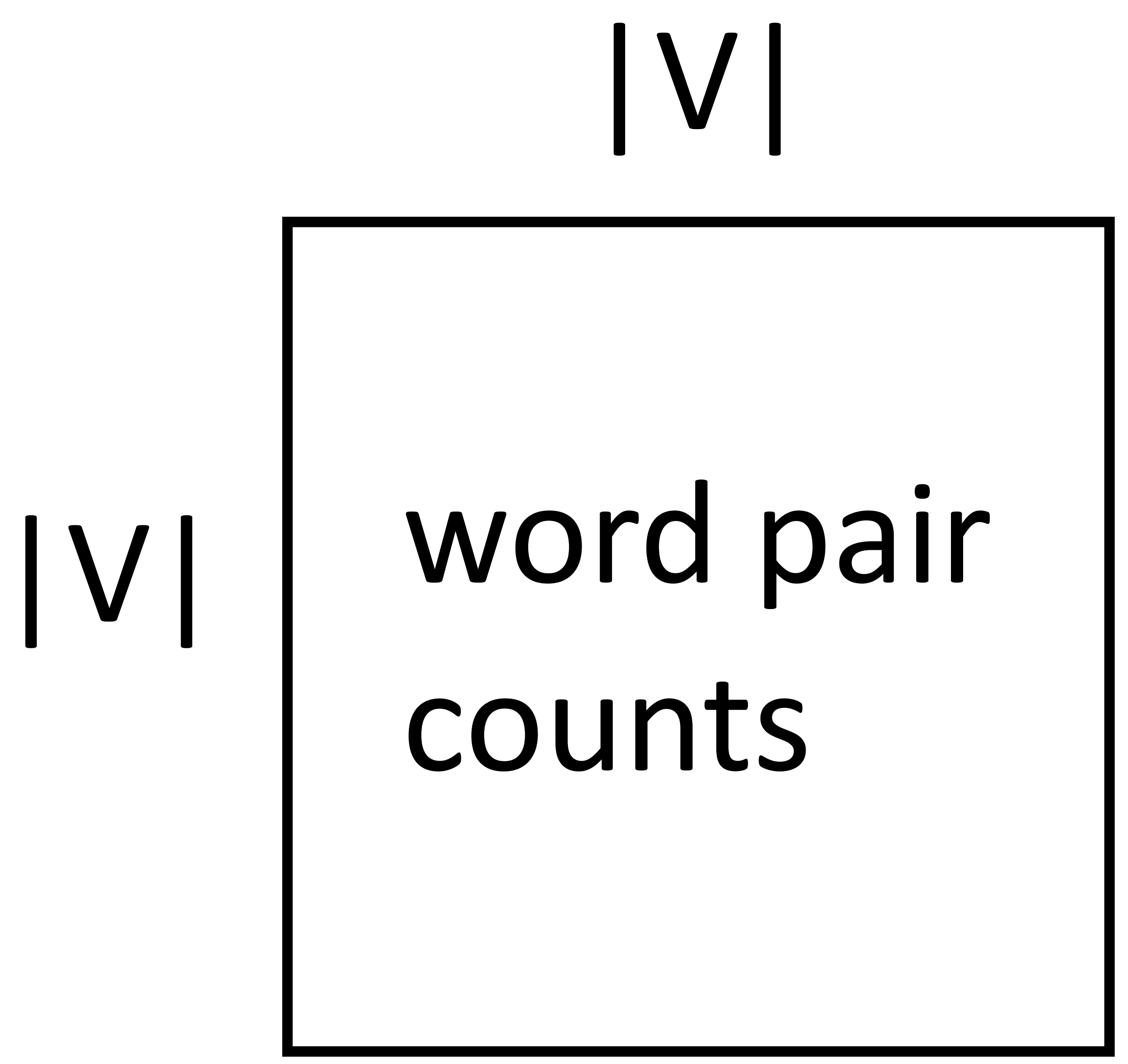
Connections with Matrix Factorization

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

Levy et al. (2014)

Connections with Matrix Factorization

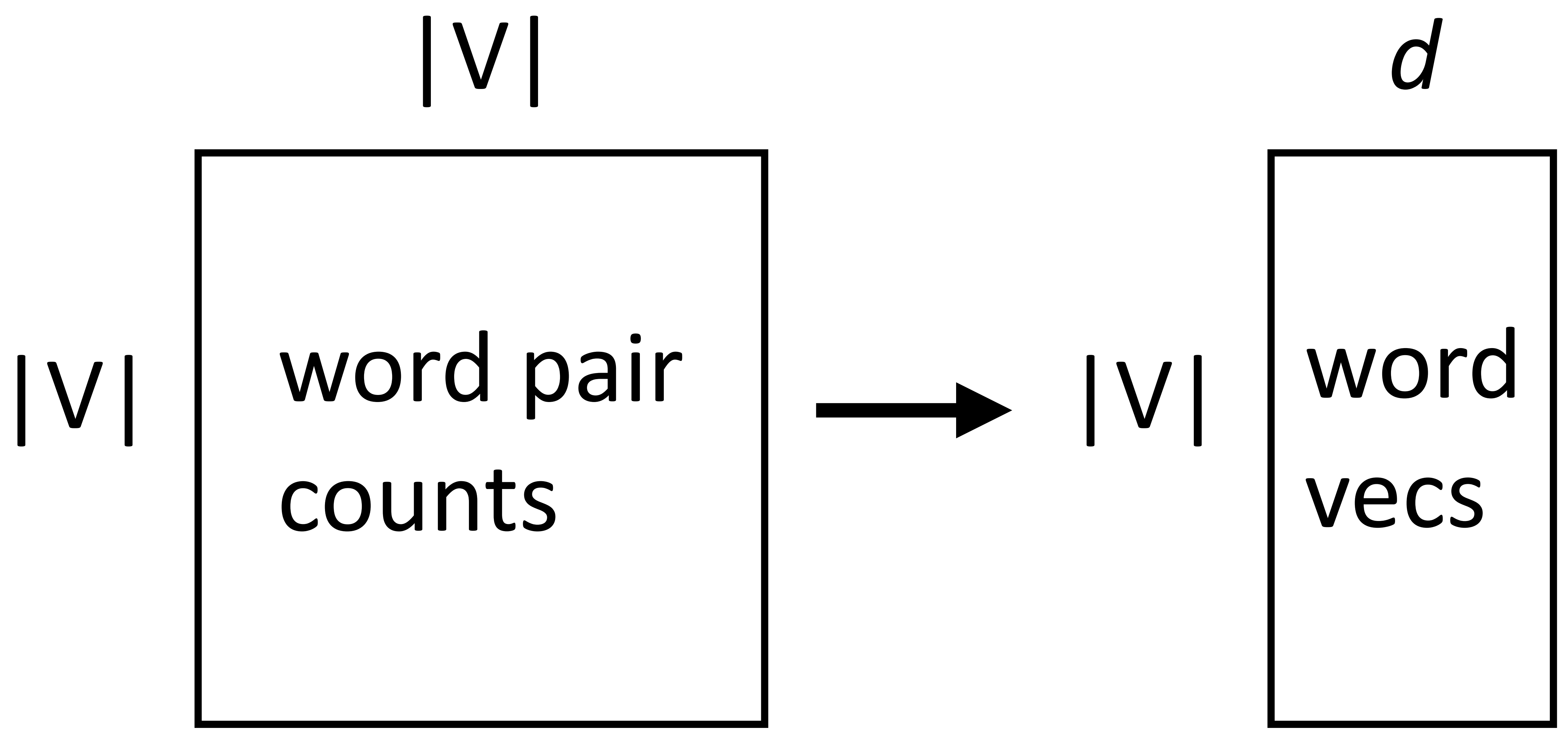
- Skip-gram model looks at word-word co-occurrences and produces two types of vectors



Levy et al. (2014)

Connections with Matrix Factorization

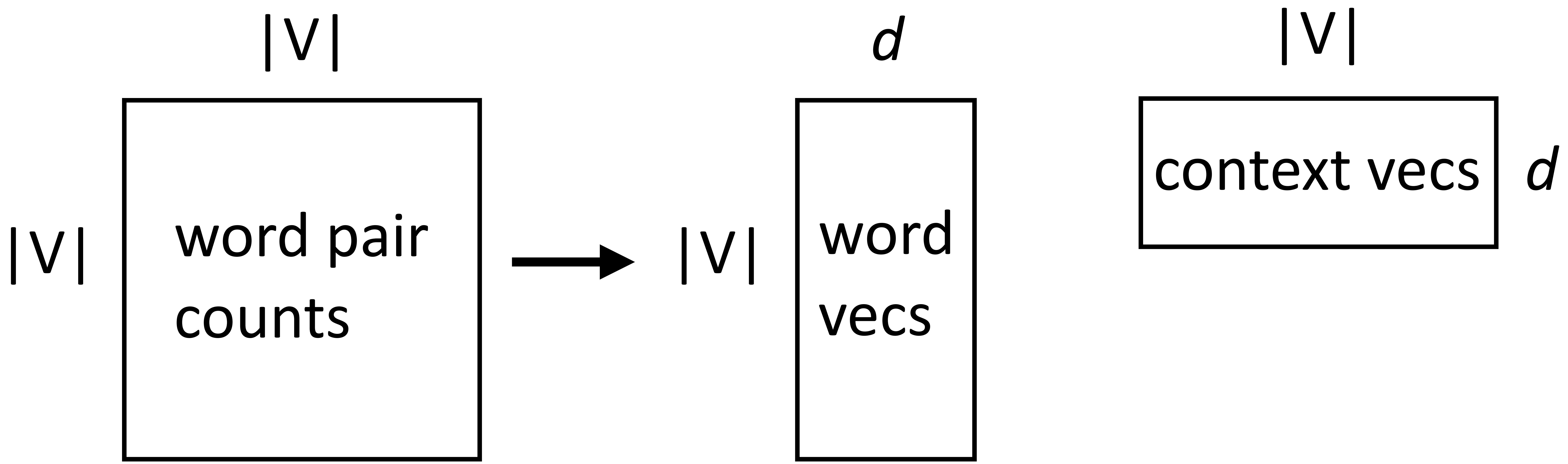
- Skip-gram model looks at word-word co-occurrences and produces two types of vectors



Levy et al. (2014)

Connections with Matrix Factorization

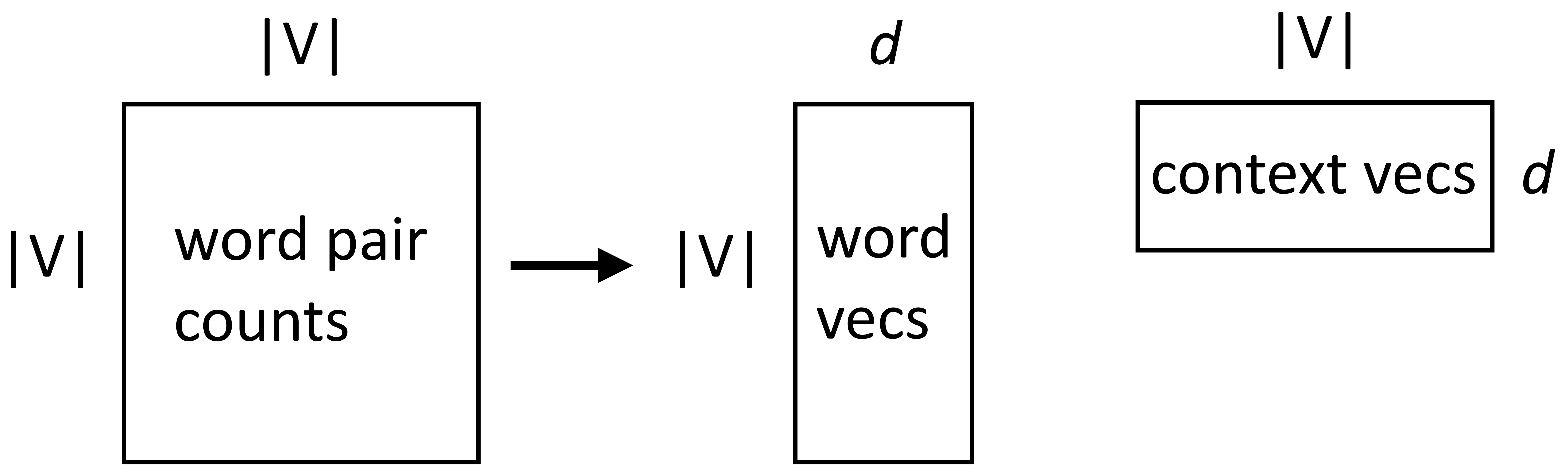
- Skip-gram model looks at word-word co-occurrences and produces two types of vectors



Levy et al. (2014)

Connections with Matrix Factorization

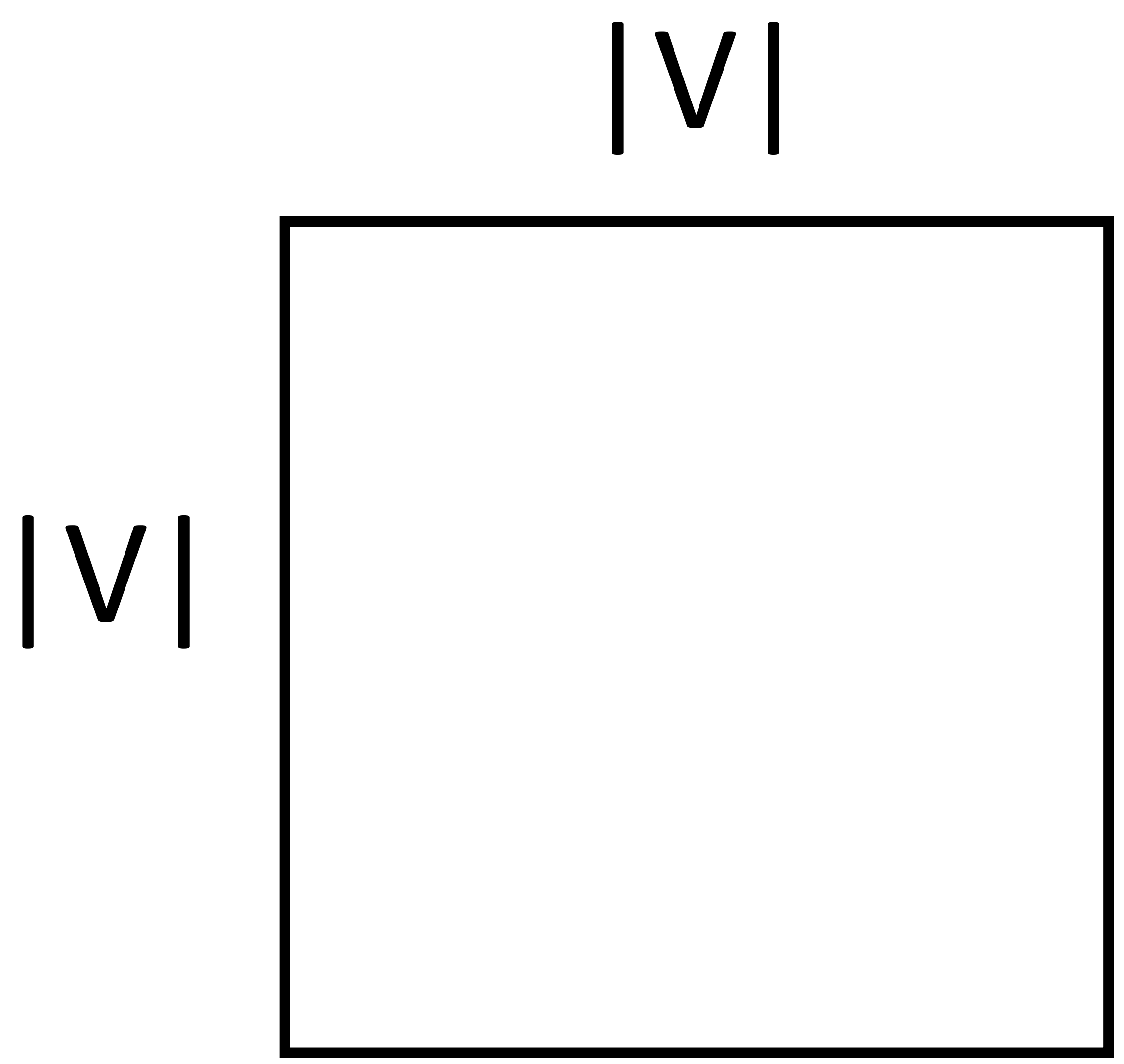
- Skip-gram model looks at word-word co-occurrences and produces two types of vectors



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

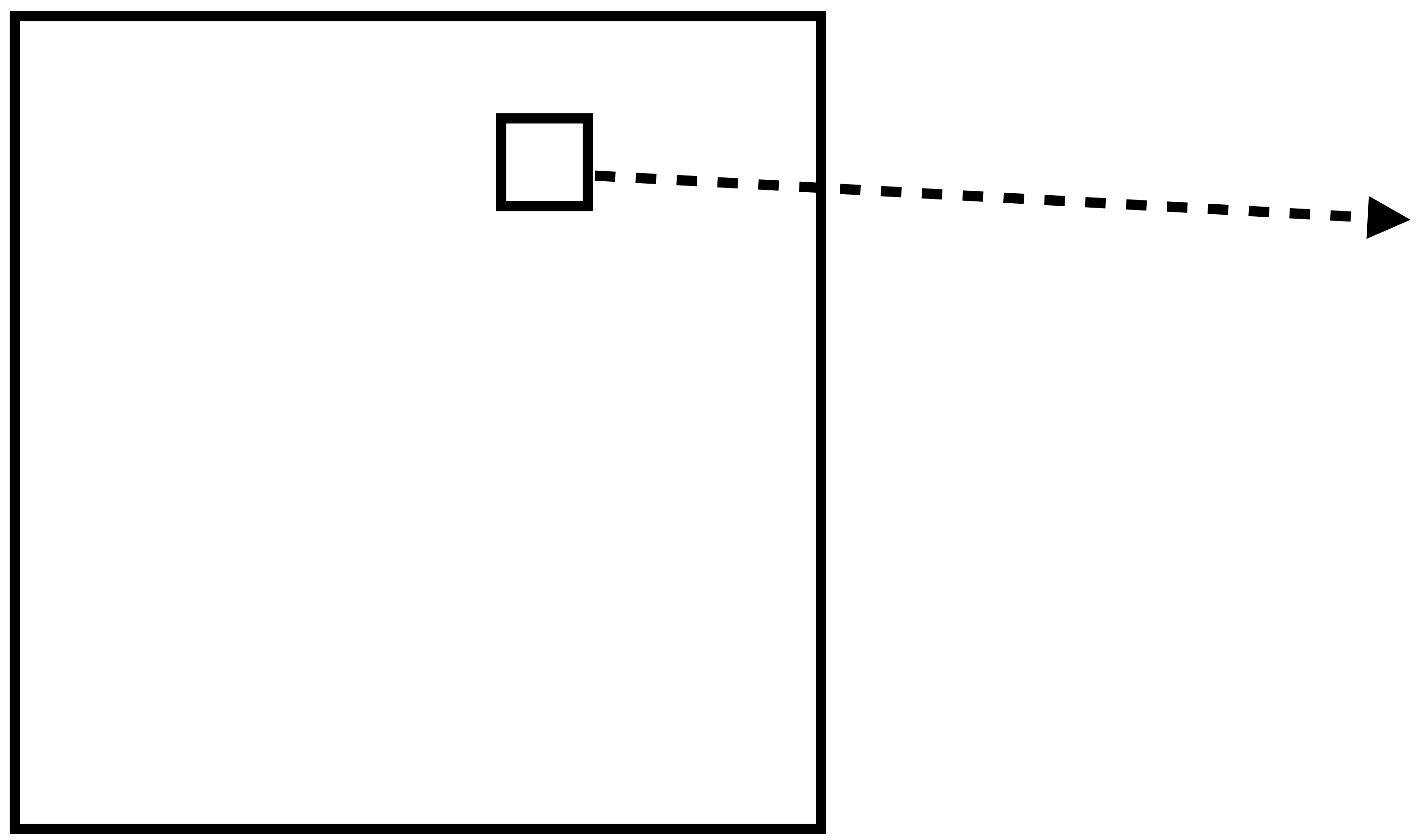
Levy et al. (2014)

Skip-Gram as Matrix Factorization



Levy et al. (2014)

Skip-Gram as Matrix Factorization

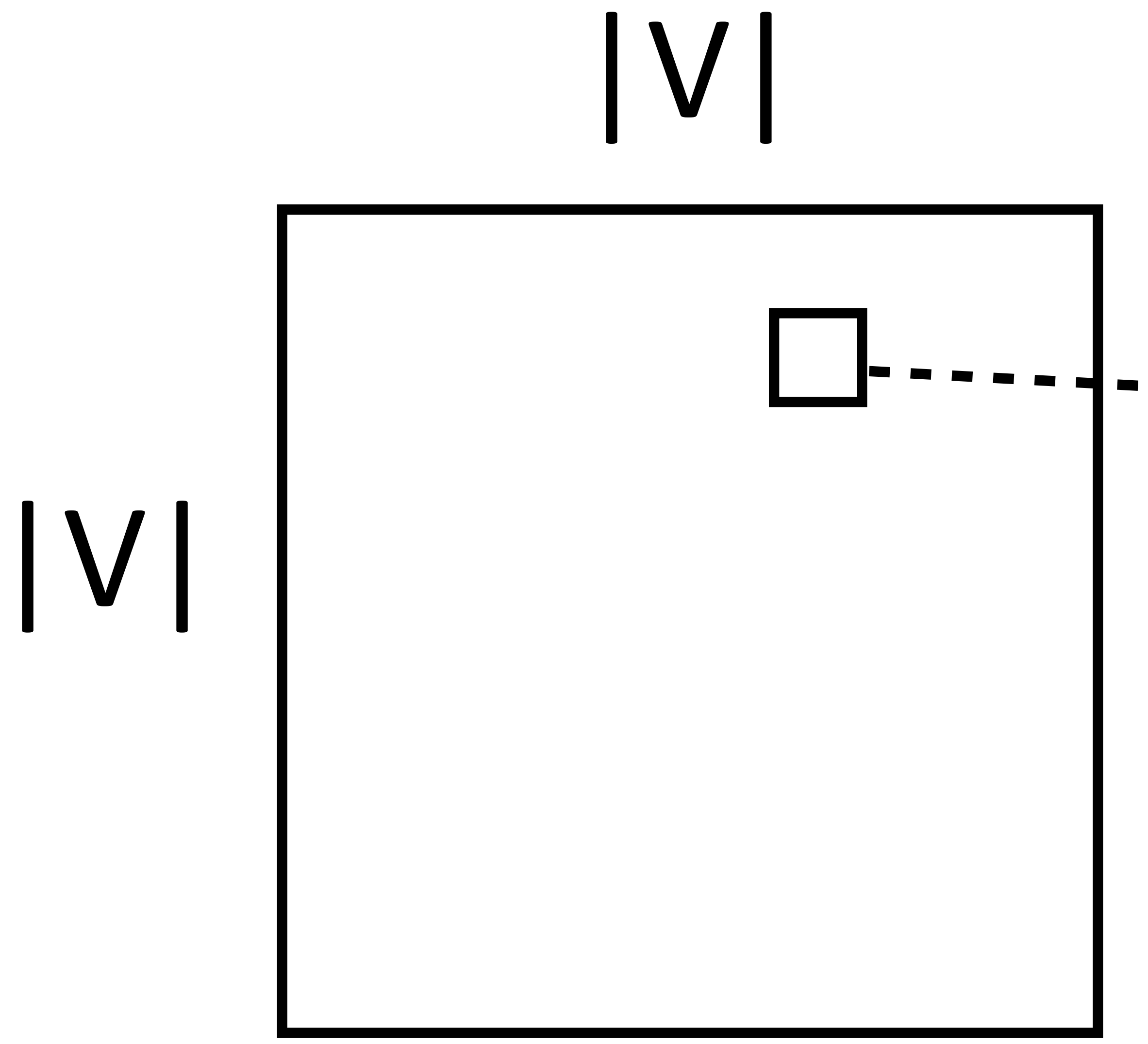


num negative samples

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

Levy et al. (2014)

Skip-Gram as Matrix Factorization



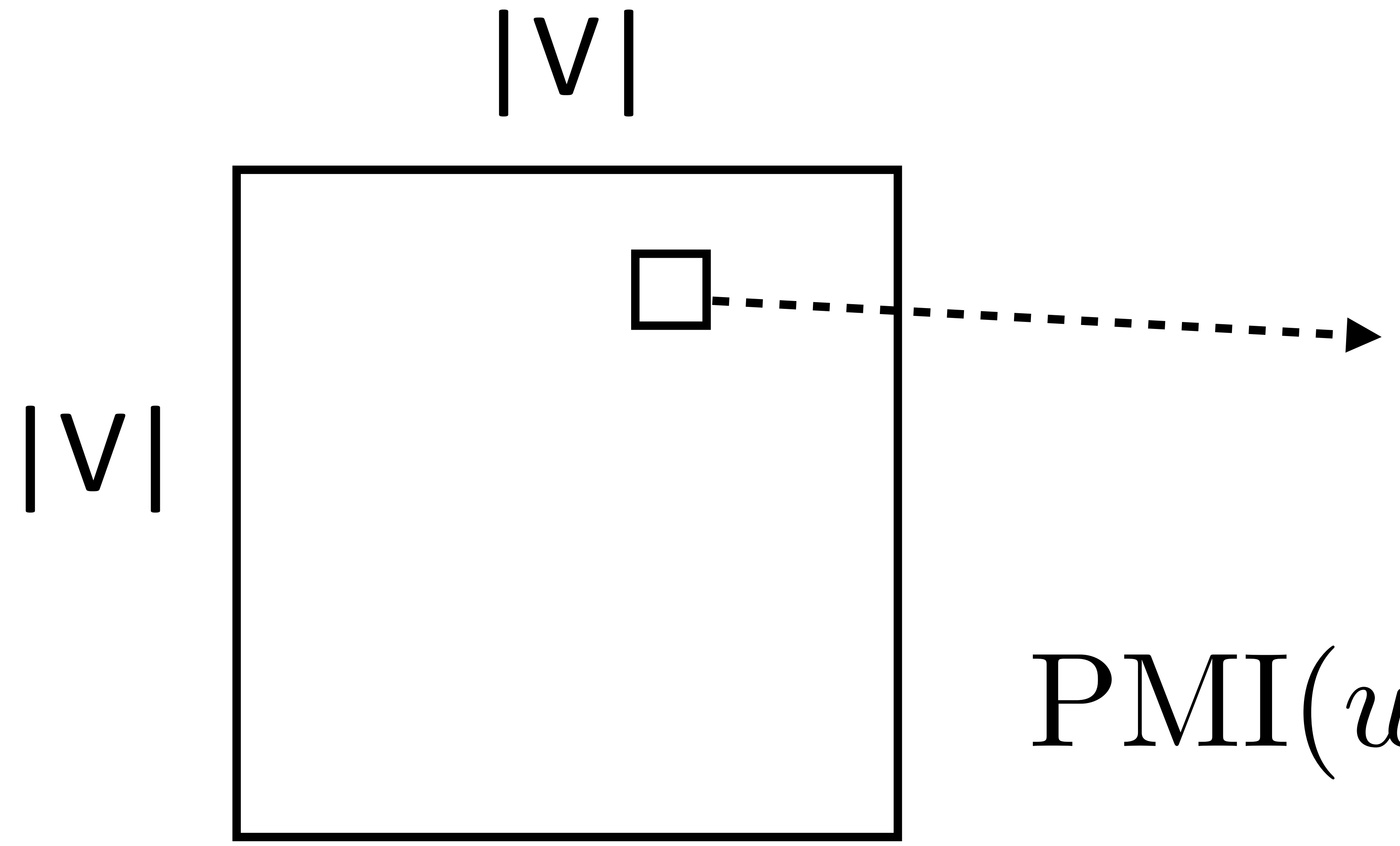
num negative samples

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

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

Levy et al. (2014)

Skip-Gram as Matrix Factorization



num negative samples

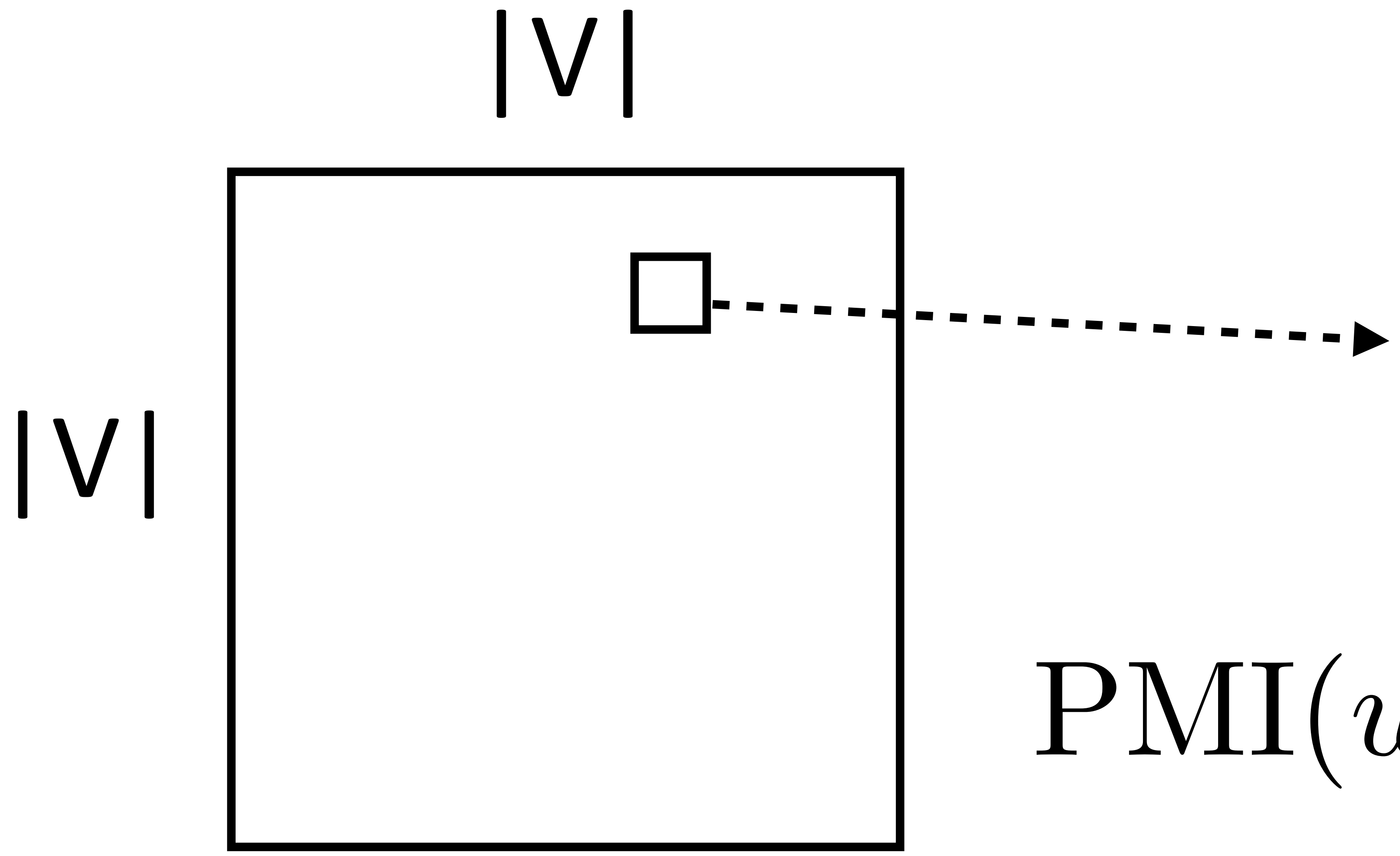
$$M_{ij} = \mathrm{PMI}(w_i, c_j) - \log k$$

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

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

Levy et al. (2014)

Skip-Gram as Matrix Factorization



num negative samples

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

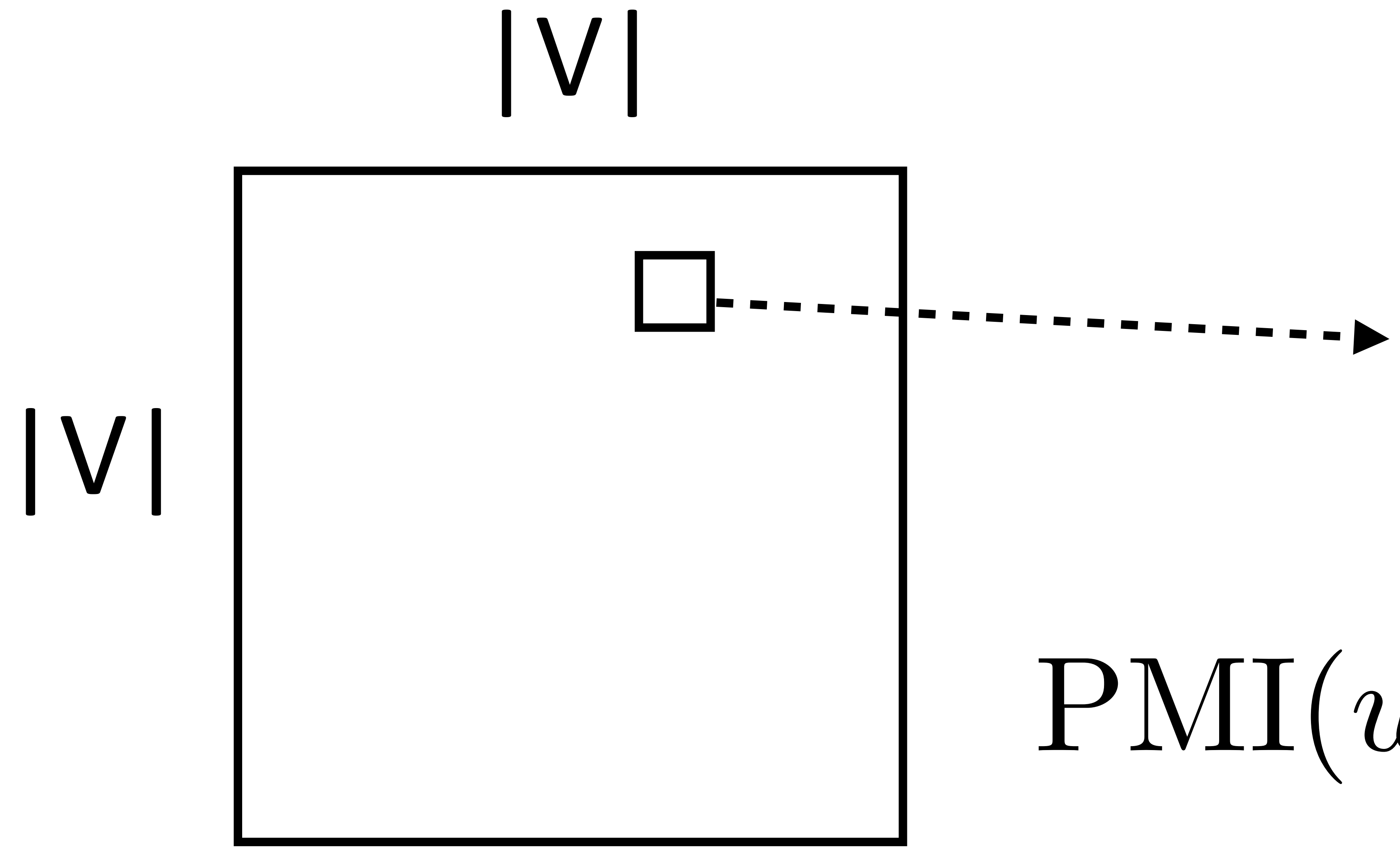
$$\mathrm{PMI}(w_i, c_j) = \frac{P(w_i, c_j)}{P(w_i)P(c_j)} = \frac{\frac{\mathrm{count}(w_i, c_j)}{D}}{\frac{\mathrm{count}(w_i)}{D}\,\frac{count(c_j)}{D}}$$

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

- *If* we sample negative examples from the uniform distribution over words

Levy et al. (2014)

Skip-Gram as Matrix Factorization



num negative samples

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

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

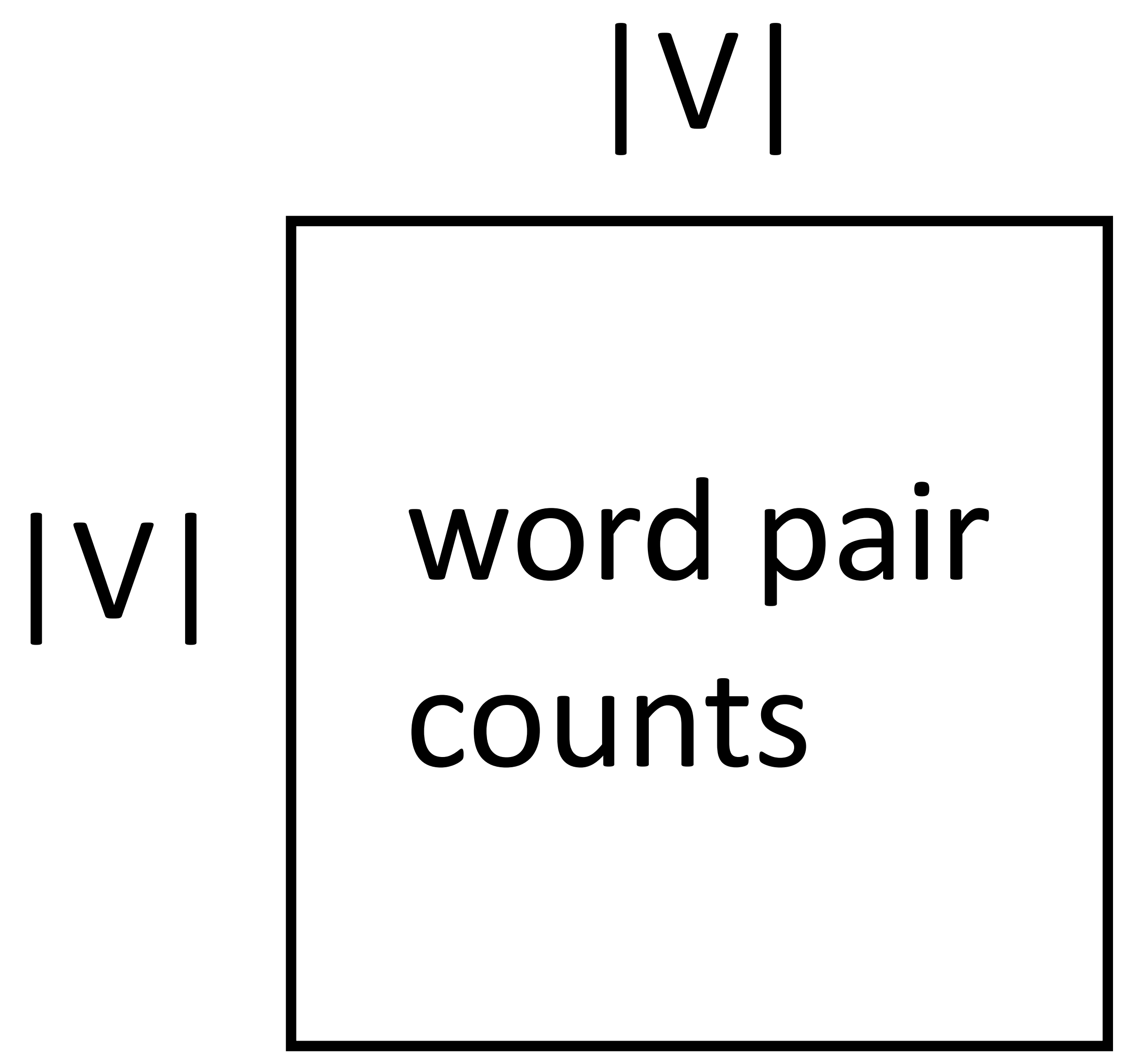
Skip-gram objective *exactly* corresponds to factoring this matrix:

- *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)

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

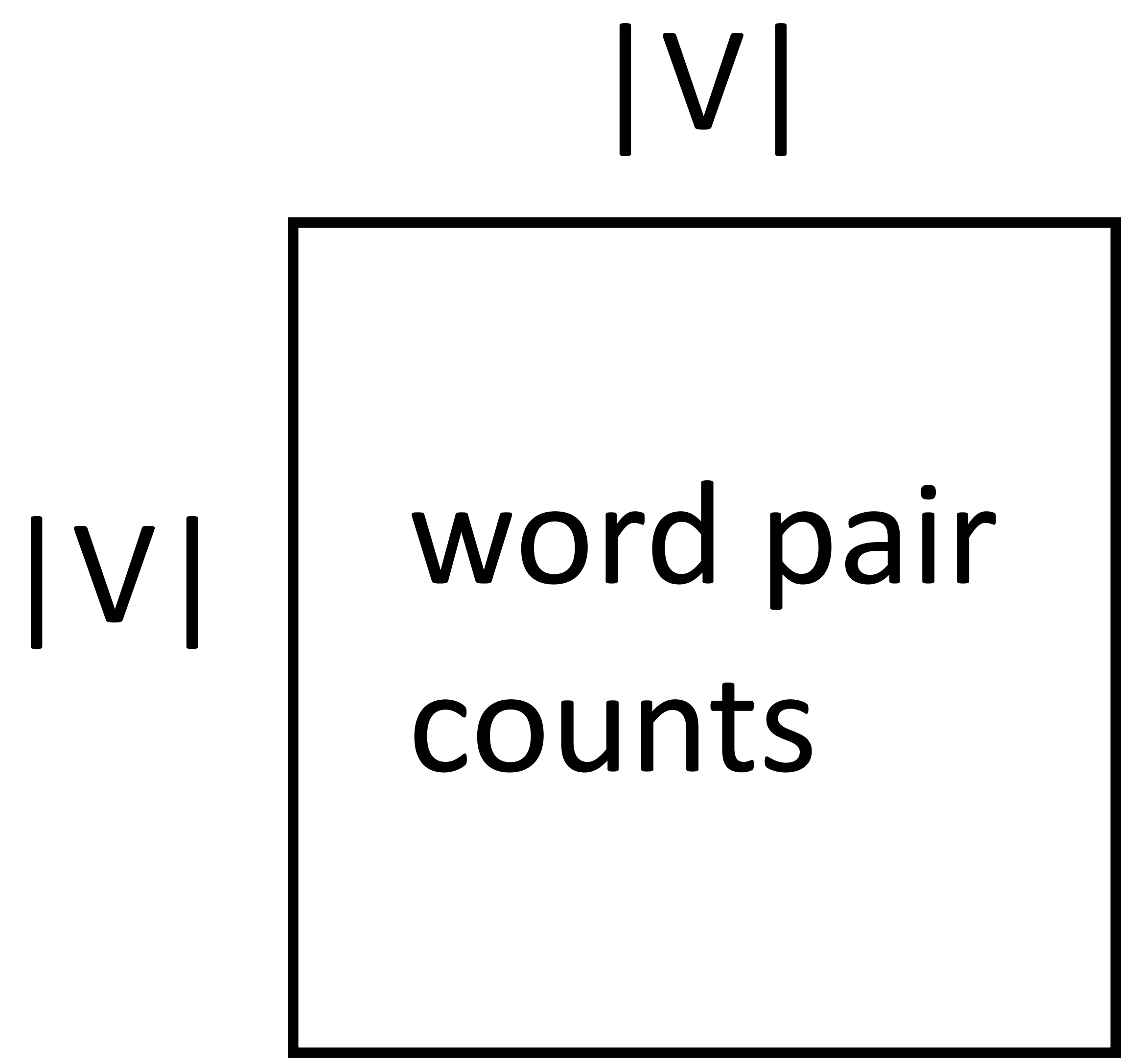
- Also operates on counts matrix, weighted regression on the log co-occurrence matrix



Pennington et al. (2014)

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

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

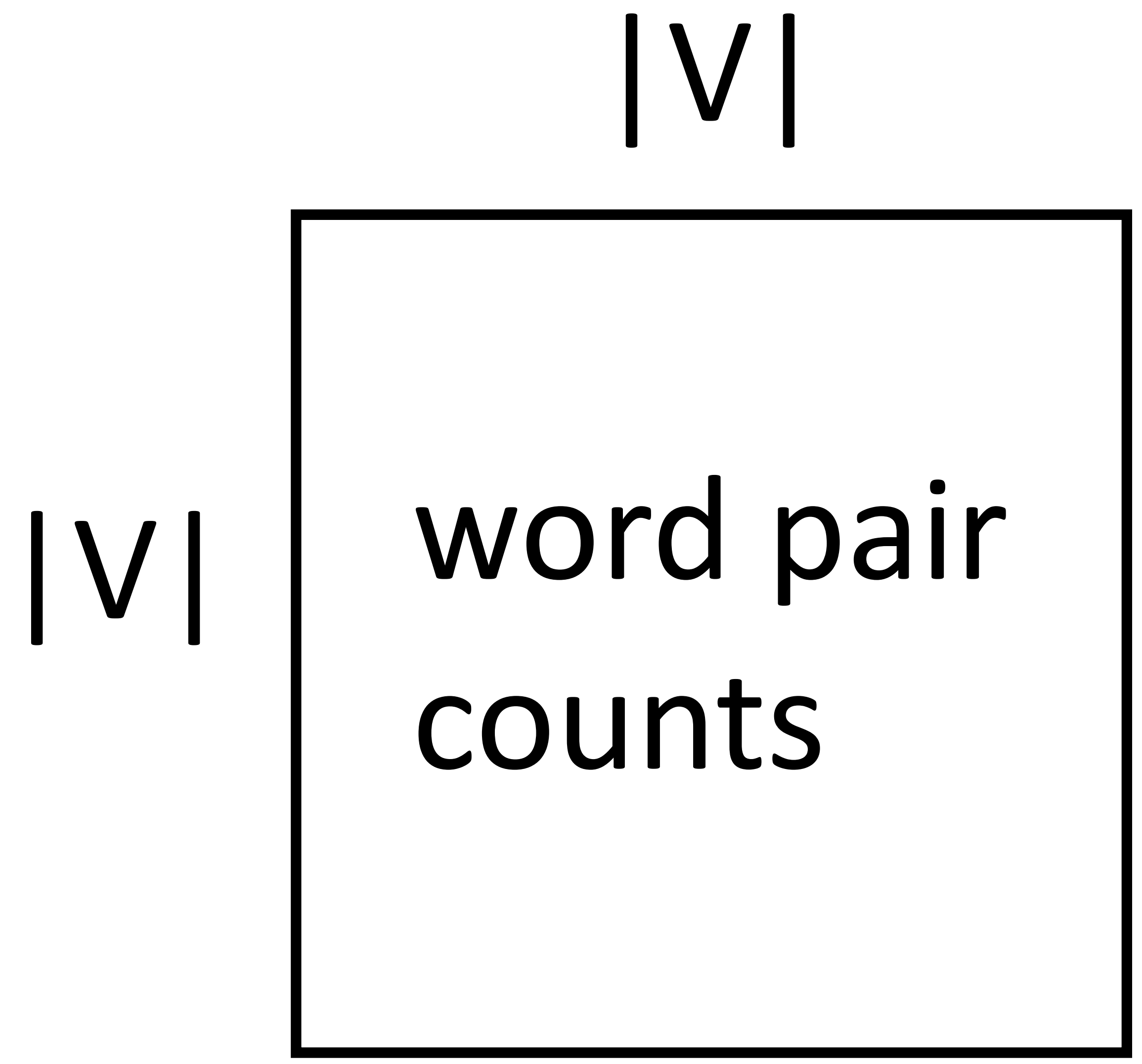


- Loss $= \sum_{i,j} f(\mathrm{count}(w_i, c_j)) \left(w_i^\top c_j + a_i + b_j - \log \mathrm{count}(w_i, c_j)\right)^2$

Pennington et al. (2014)

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

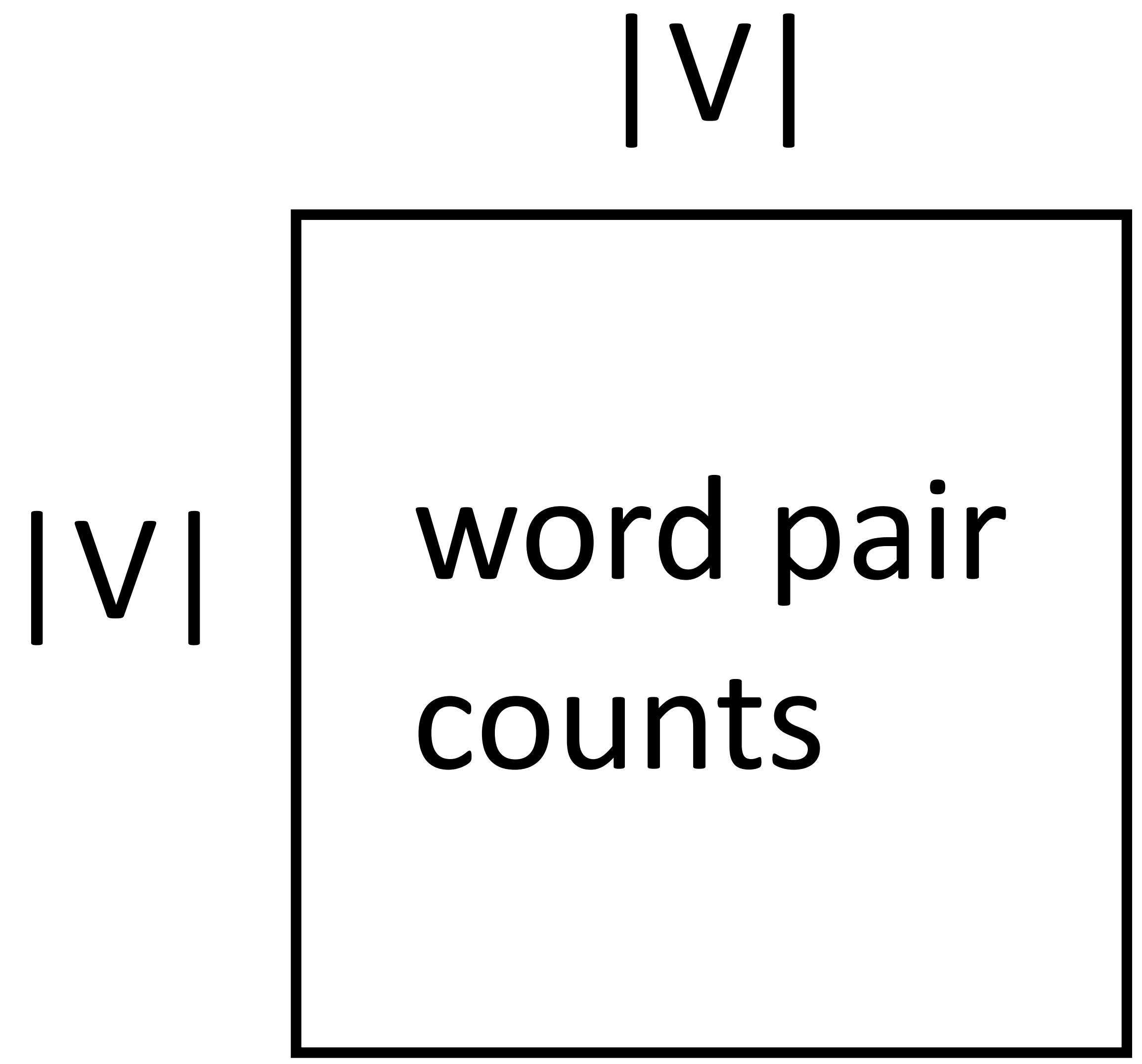
- Also operates on counts matrix, weighted regression on the log co-occurrence matrix



- 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

Pennington et al. (2014)

GloVe (**Global Vectors**)



- 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 (uncontextualized) word vectors used today (20,000+ citations)

Pennington et al. (2014)

Preview: Context-dependent Embeddings

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

they dance at balls

they hit the balls

Peters et al. (2018)

Preview: Context-dependent Embeddings

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

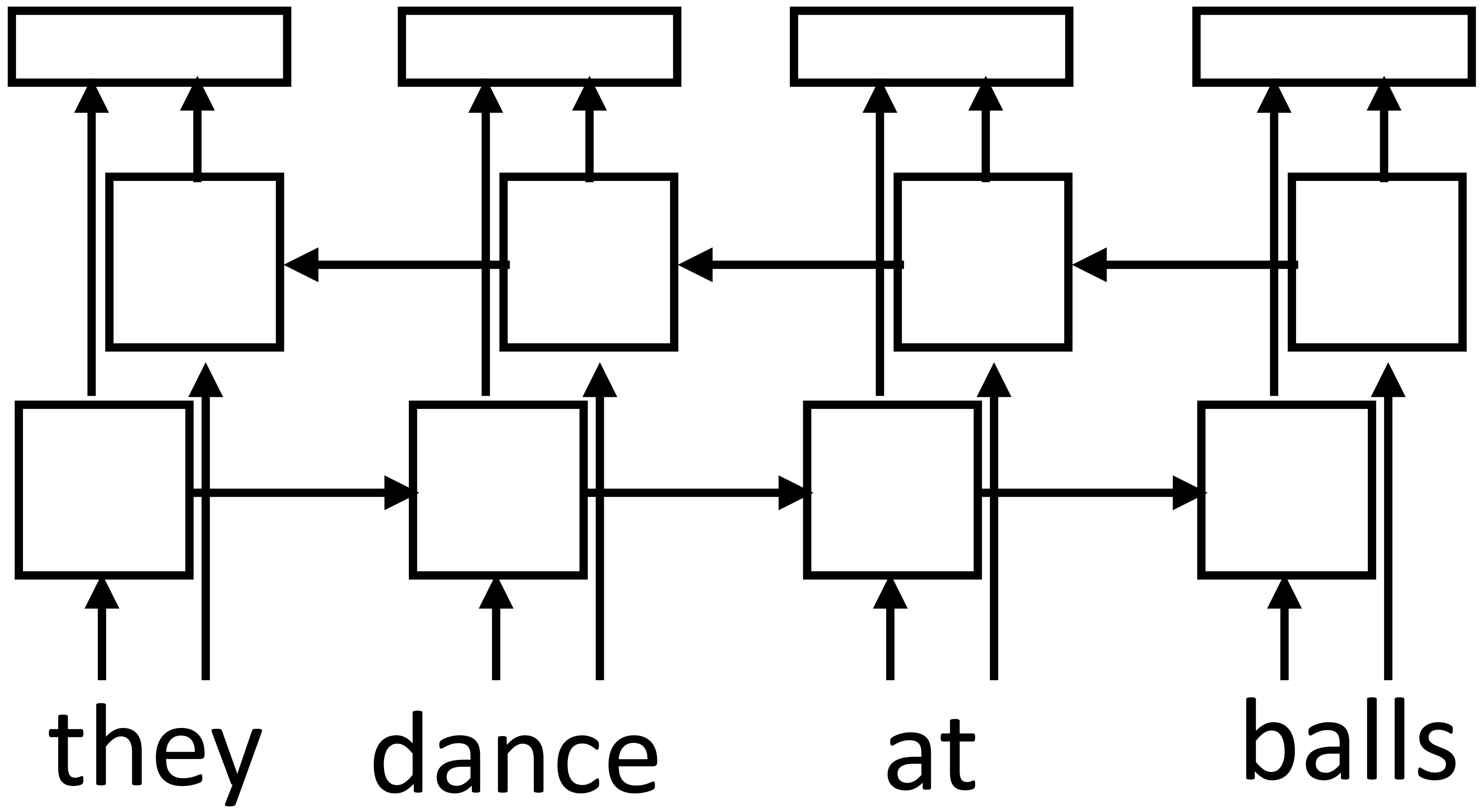
they dance at balls they hit the balls

- Train a neural language model to predict the next word given previous words in the sentence, use its internal representations as word vectors

Peters et al. (2018)

Preview: Context-dependent Embeddings

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



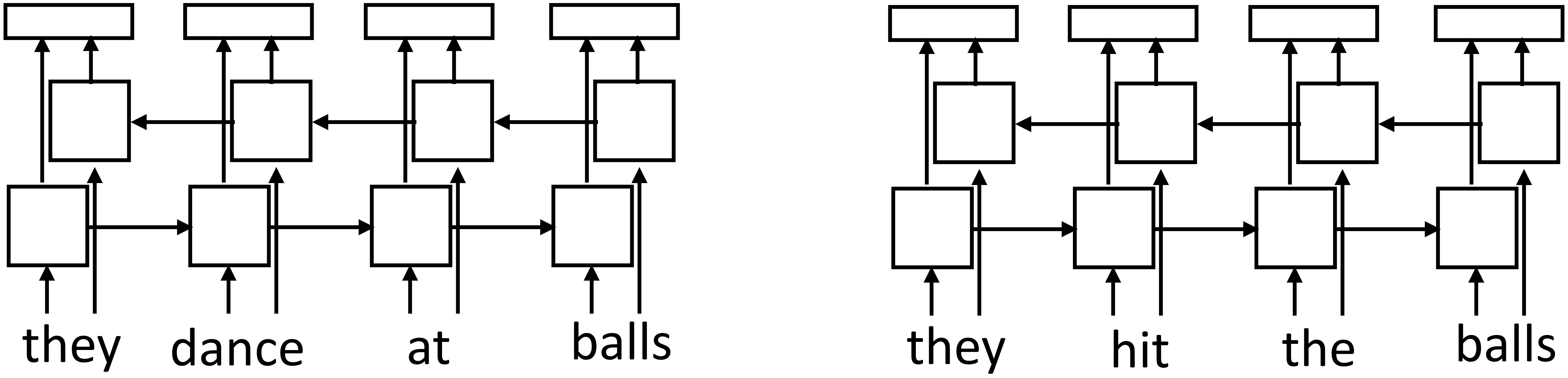
they hit the balls

- Train a neural language model to predict the next word given previous words in the sentence, use its internal representations as word vectors

Peters et al. (2018)

Preview: Context-dependent Embeddings

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

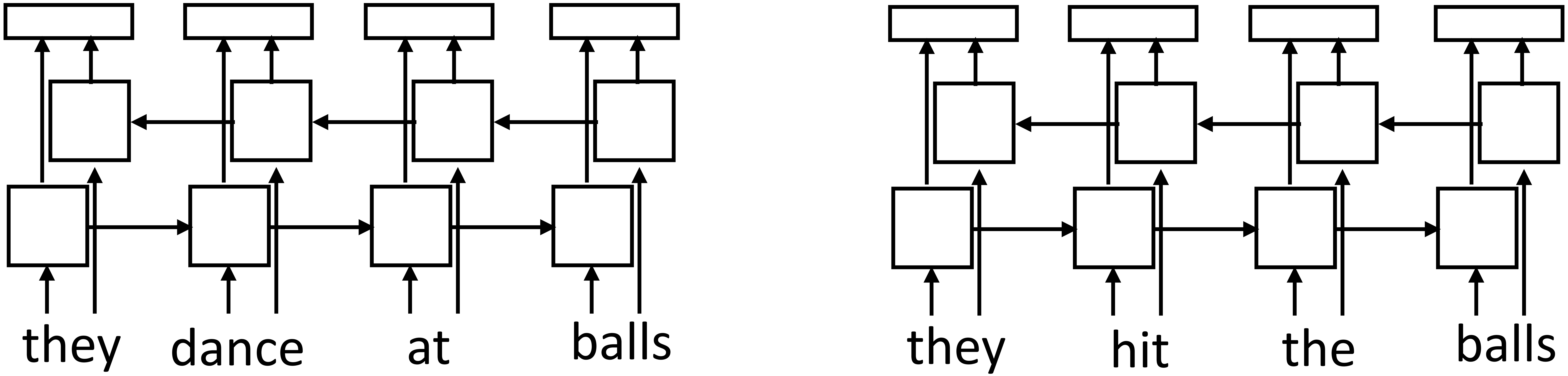


- Train a neural language model to predict the next word given previous words in the sentence, use its internal representations as word vectors

Peters et al. (2018)

Preview: Context-dependent Embeddings

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

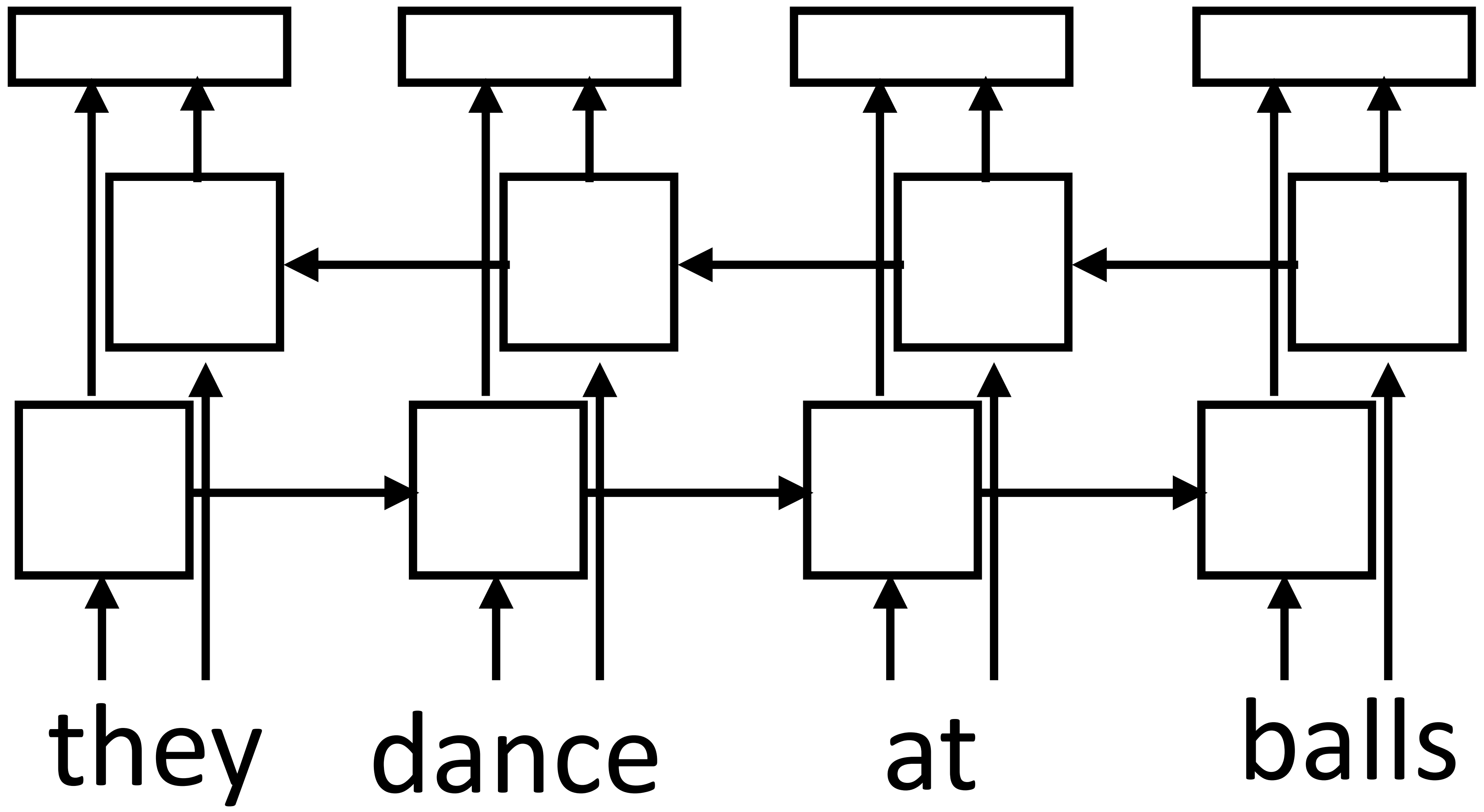


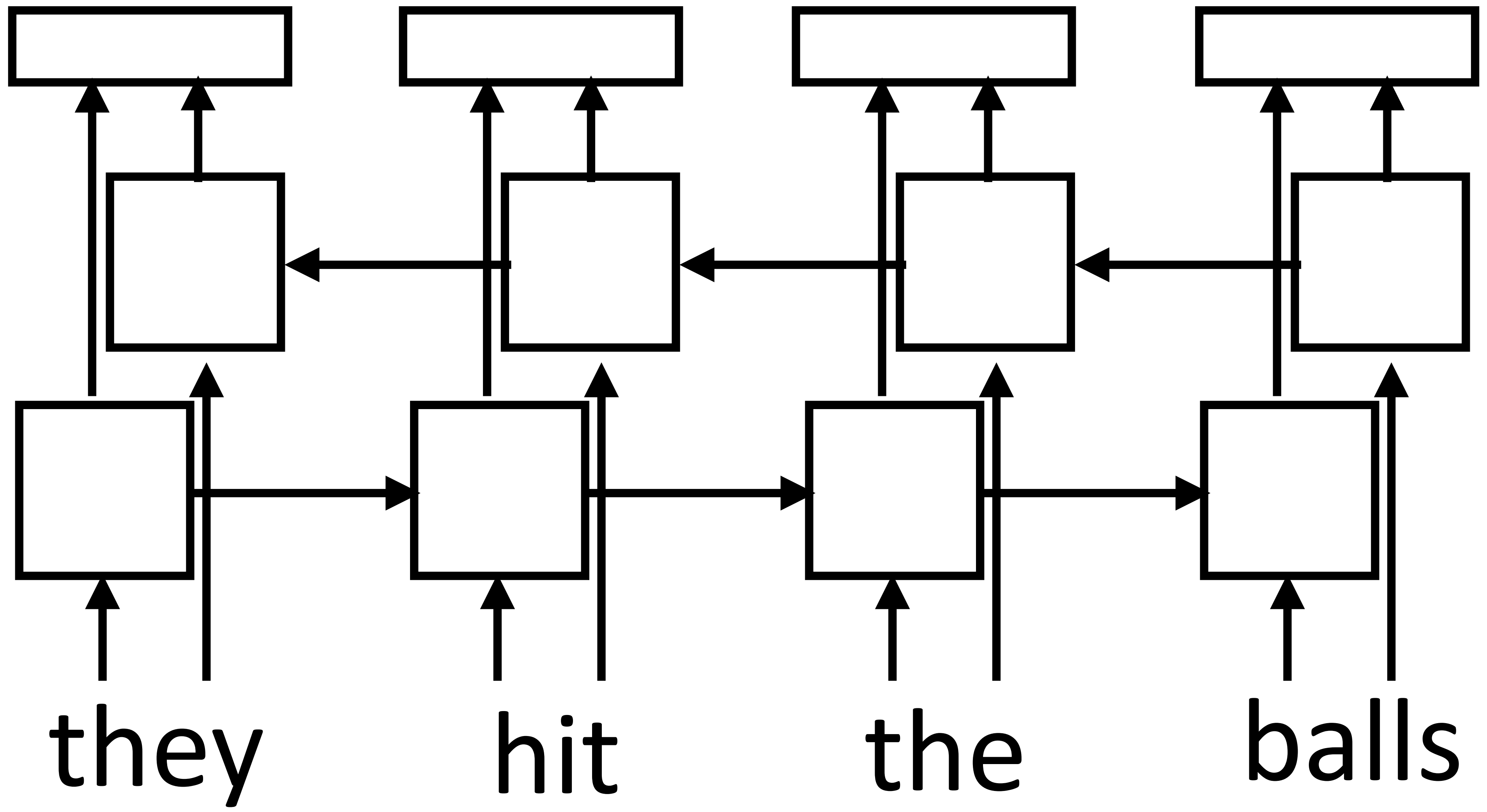
- Train a neural language model to predict the next word given previous words in the sentence, use its internal representations as word vectors
- *Context-sensitive* word embeddings: depend on rest of the sentence

Peters et al. (2018)

Preview: Context-dependent Embeddings

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





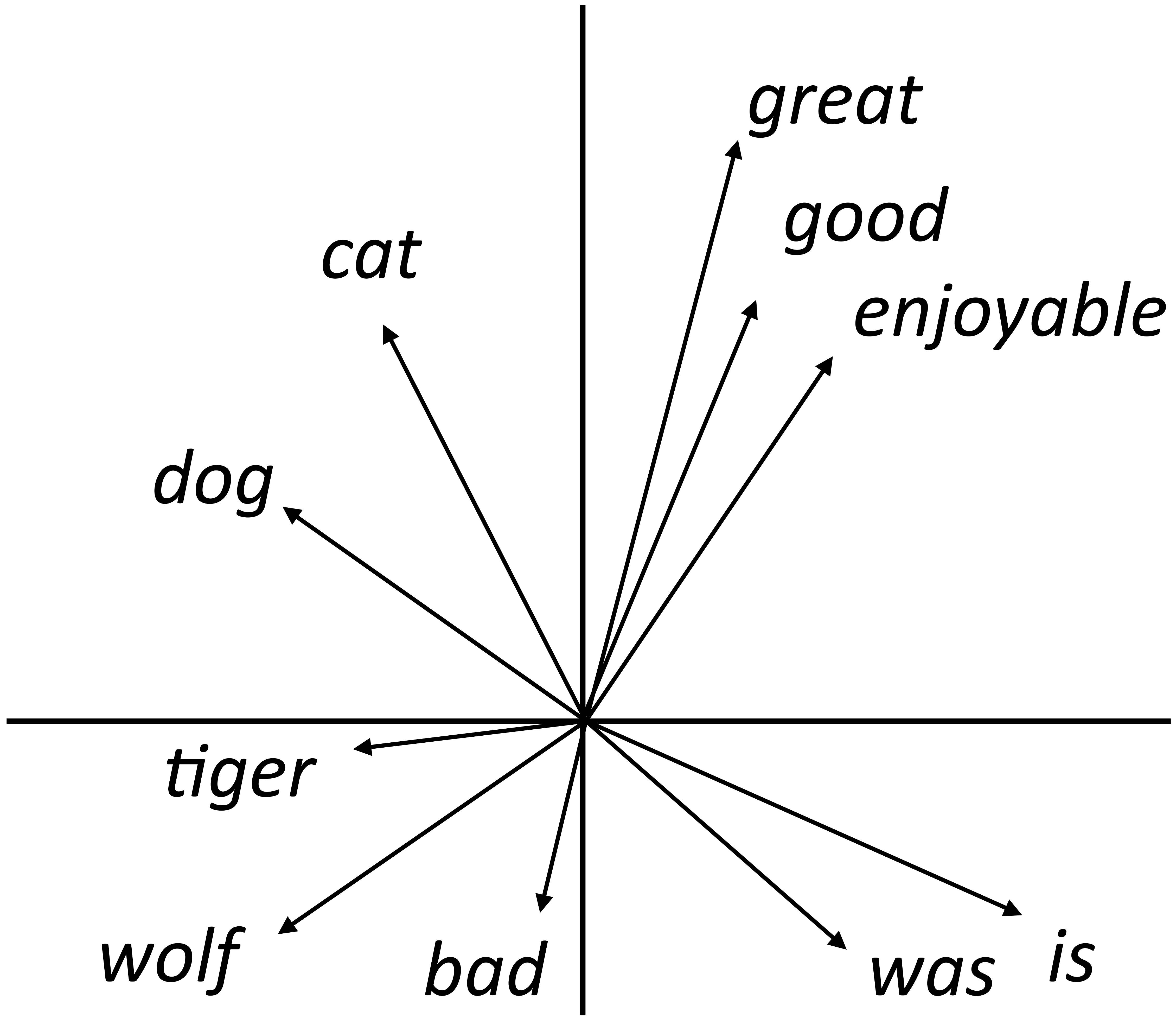
- Train a neural language model to predict the next word given previous words in the sentence, use its internal representations as word vectors
- *Context-sensitive* word embeddings: depend on rest of the sentence
- *Huge* improvements across nearly all NLP tasks over GloVe

Peters et al. (2018)

Evaluation

Evaluating Word Embeddings

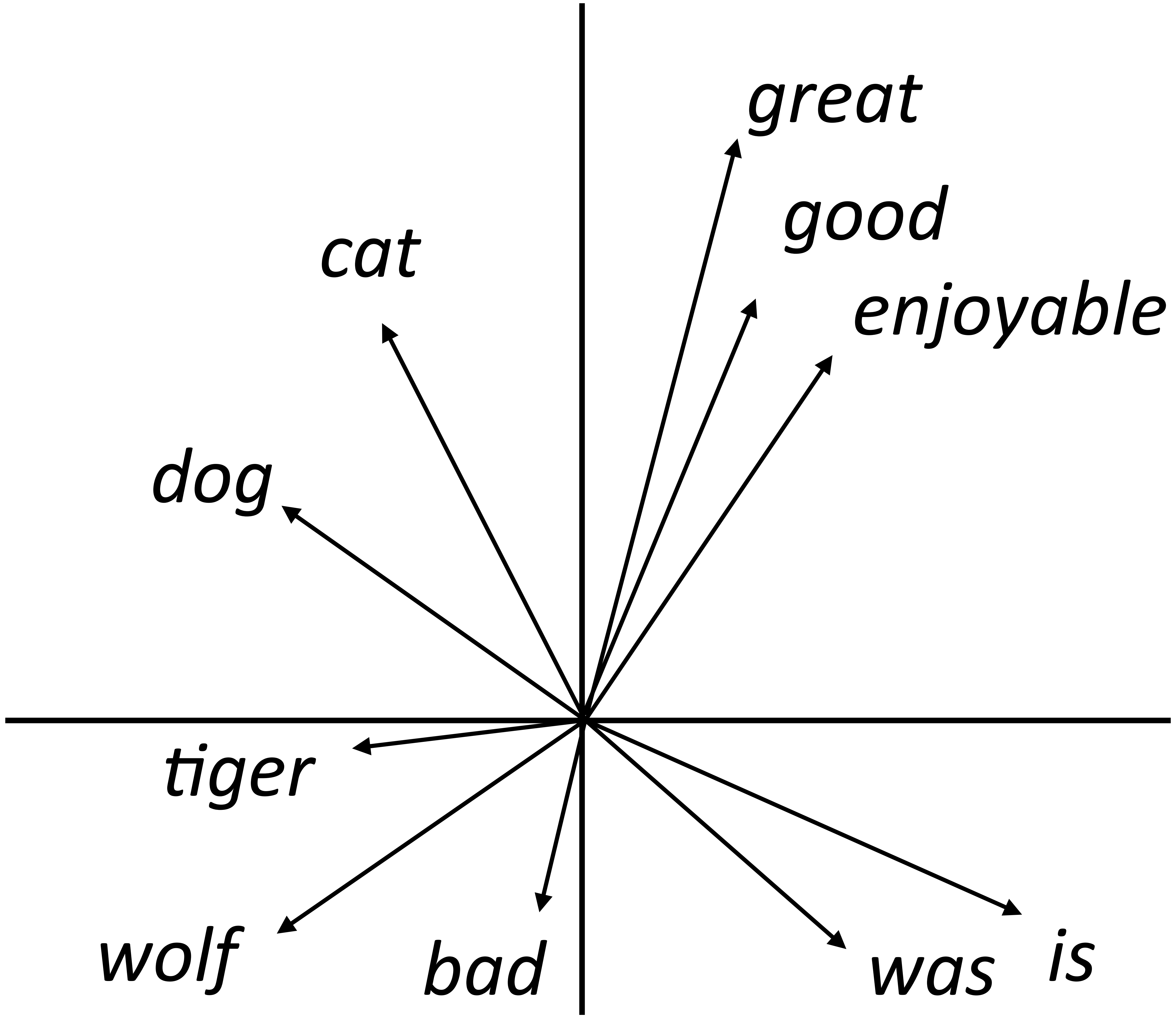
- What properties of language should word embeddings capture?



Evaluating Word Embeddings

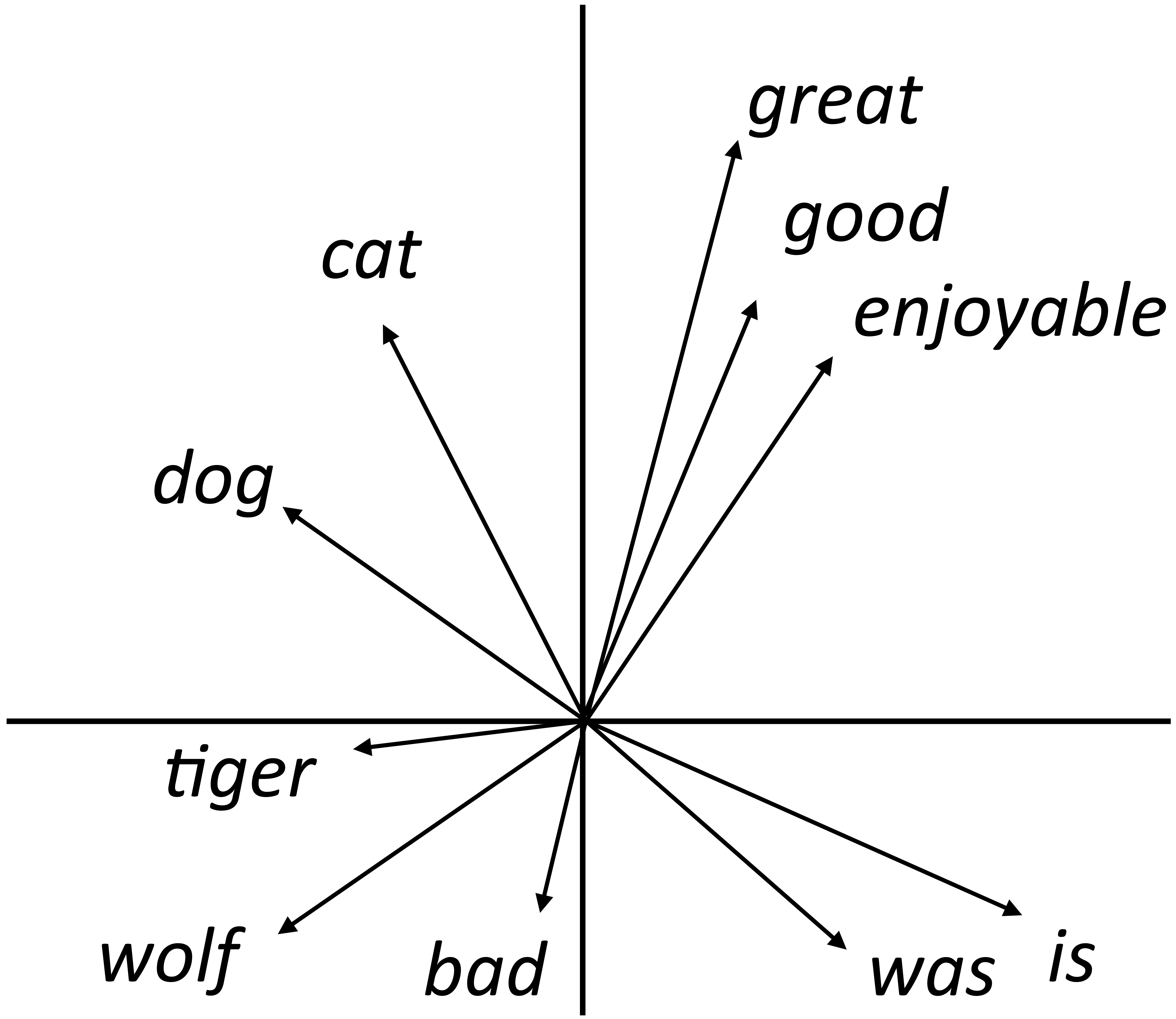
- What properties of language should word embeddings capture?

- Similarity: similar words are close to each other



Evaluating Word Embeddings

- What properties of language should word embeddings capture?
- Similarity: similar words are close to each other
- Analogy:



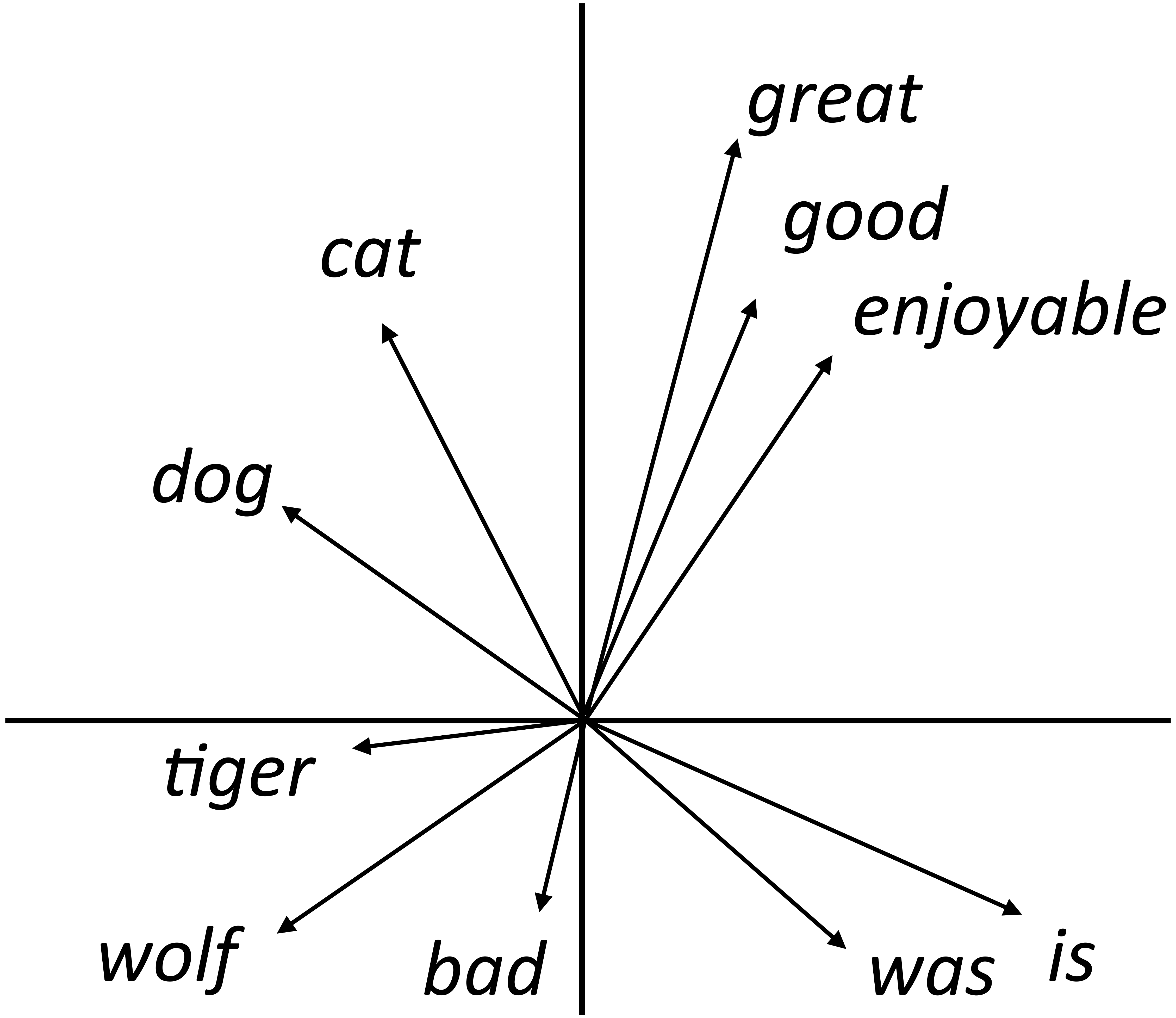
Evaluating Word Embeddings

- 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 ???

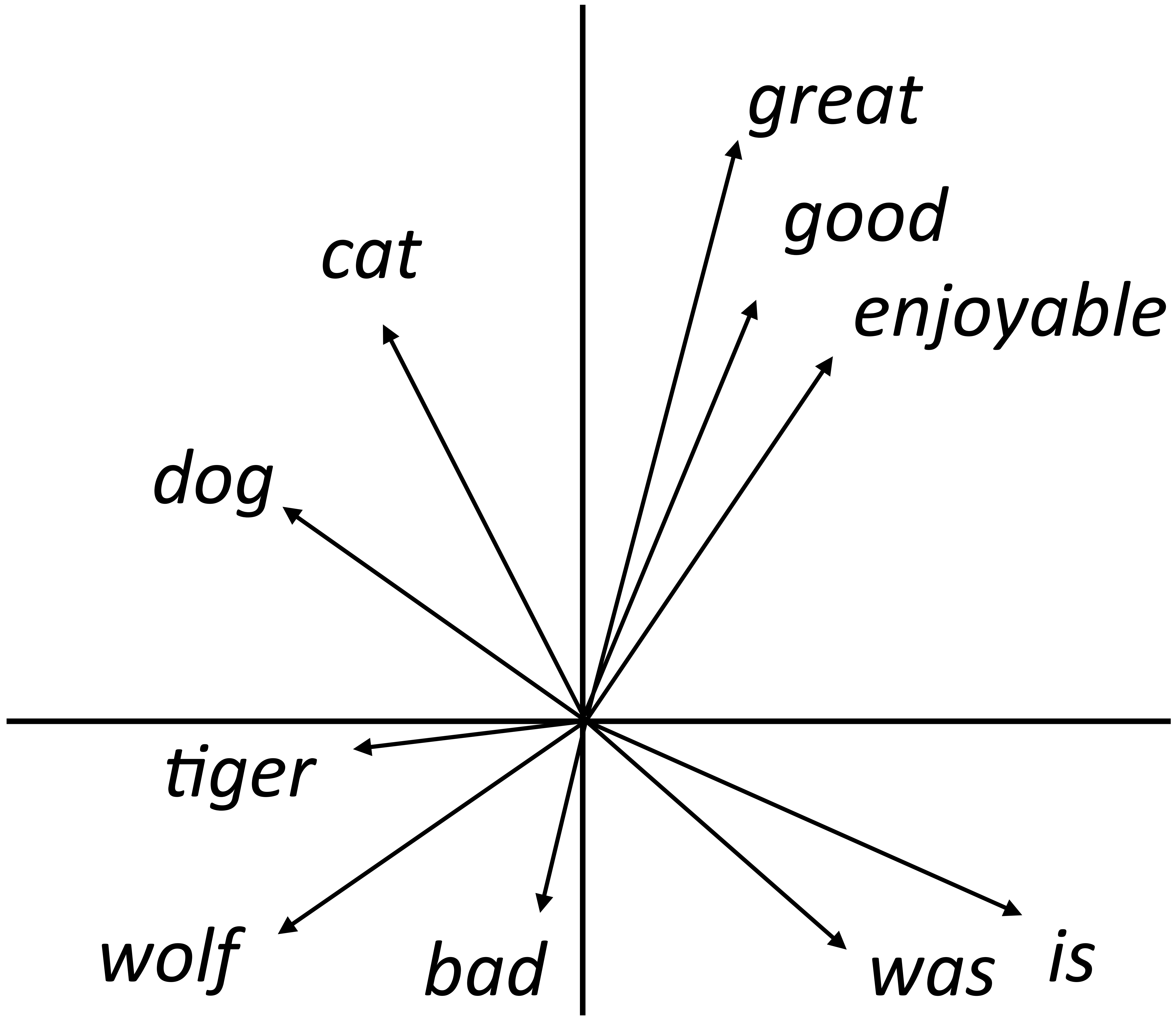


Evaluating Word Embeddings

- 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 ???



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

- SVD = singular value decomposition on PMI matrix

Levy et al. (2015)

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

- 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

Chang et al. (2017)

Hypernymy Detection

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

Chang et al. (2017)

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·ΔS	**57.2**	36.6	**32.0**	60.9	**32.7**

Chang et al. (2017)

Hypernymy Detection

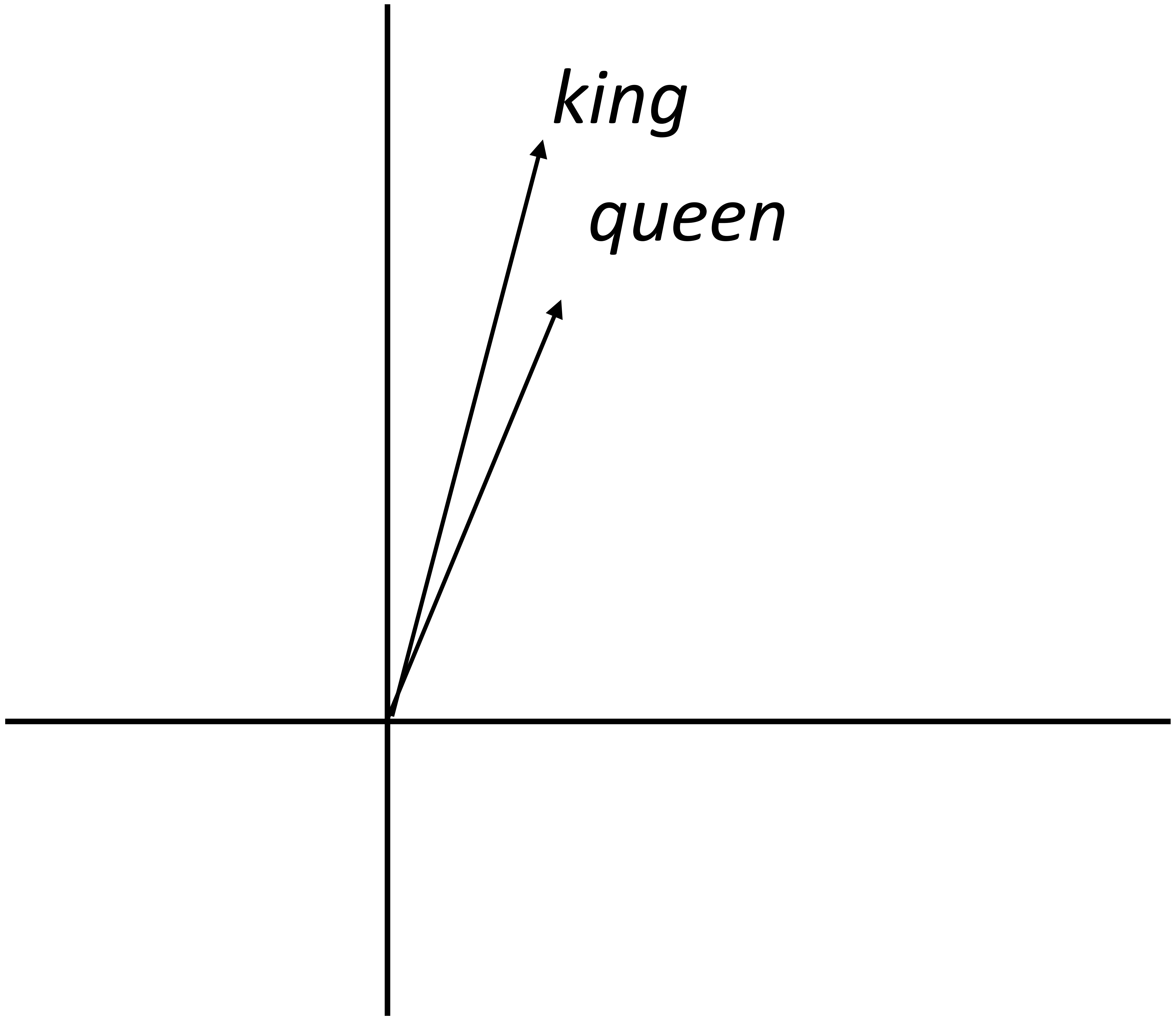
- Hypernyms: detective *is a* person, dog *is a* animal
- Do word vectors encode these relationships?

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·ΔS	**57.2**	36.6	**32.0**	60.9	**32.7**

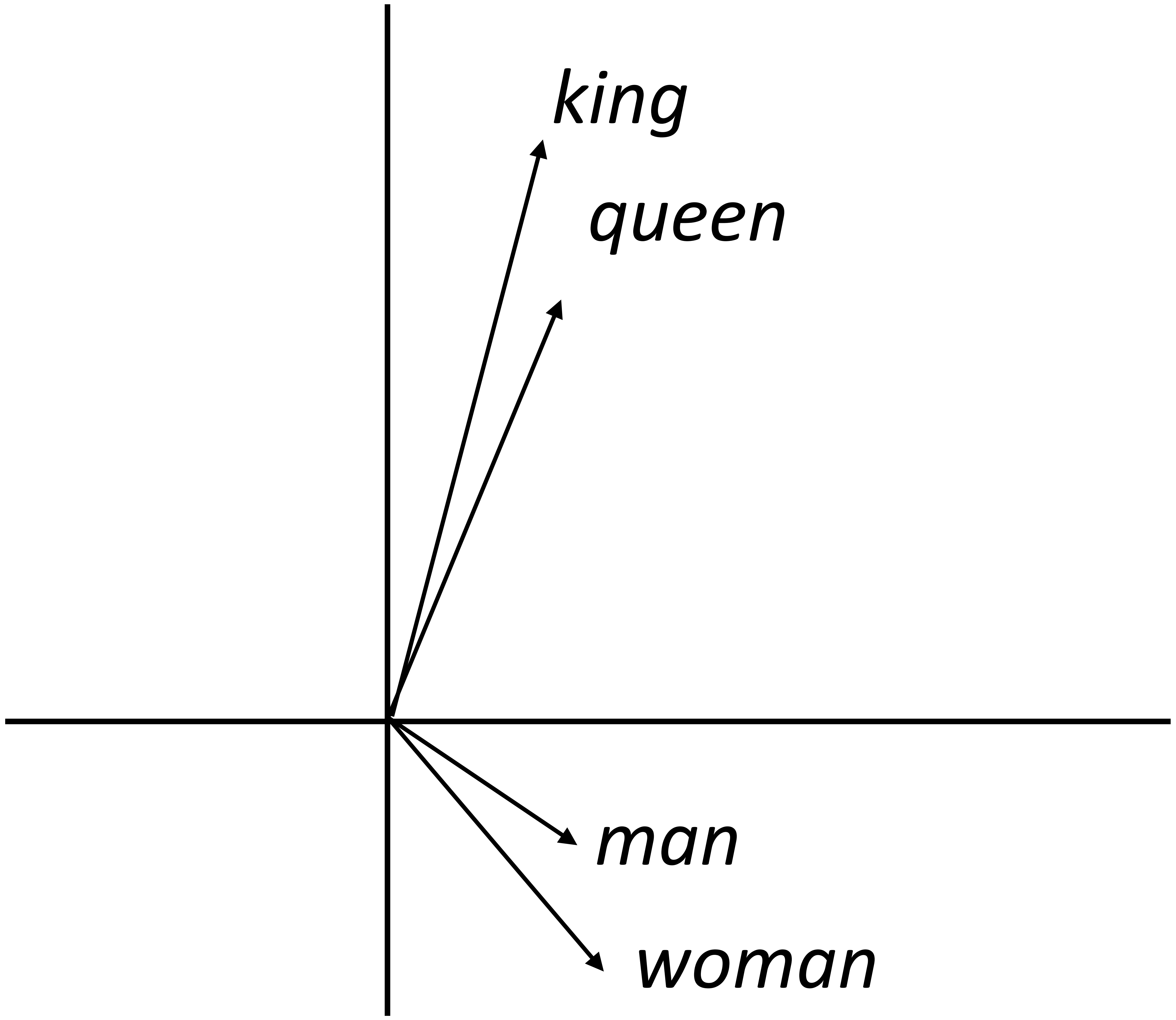
- word2vec (SGNS) works barely better than random guessing here

Chang et al. (2017)

Analogies

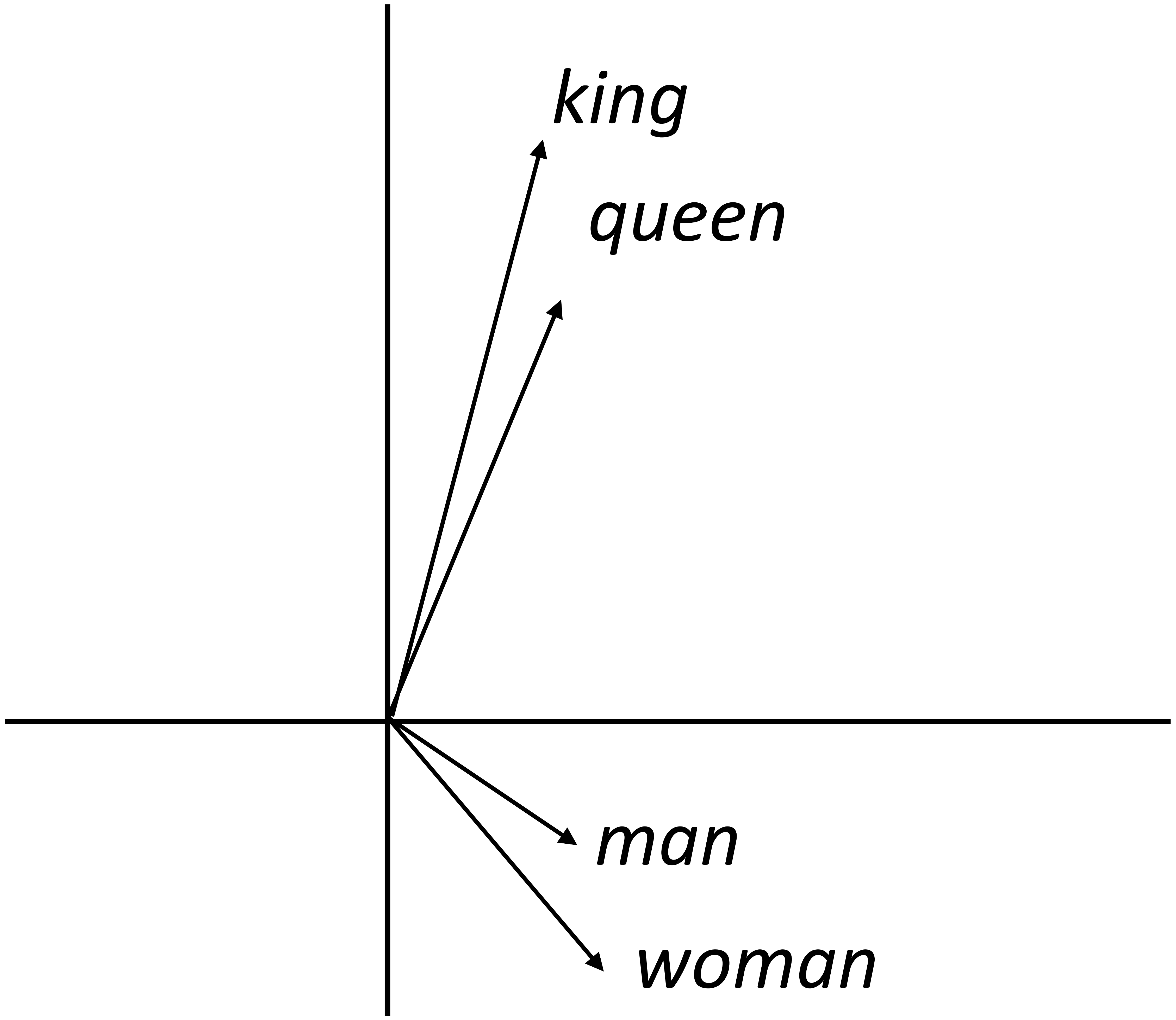


Analogies



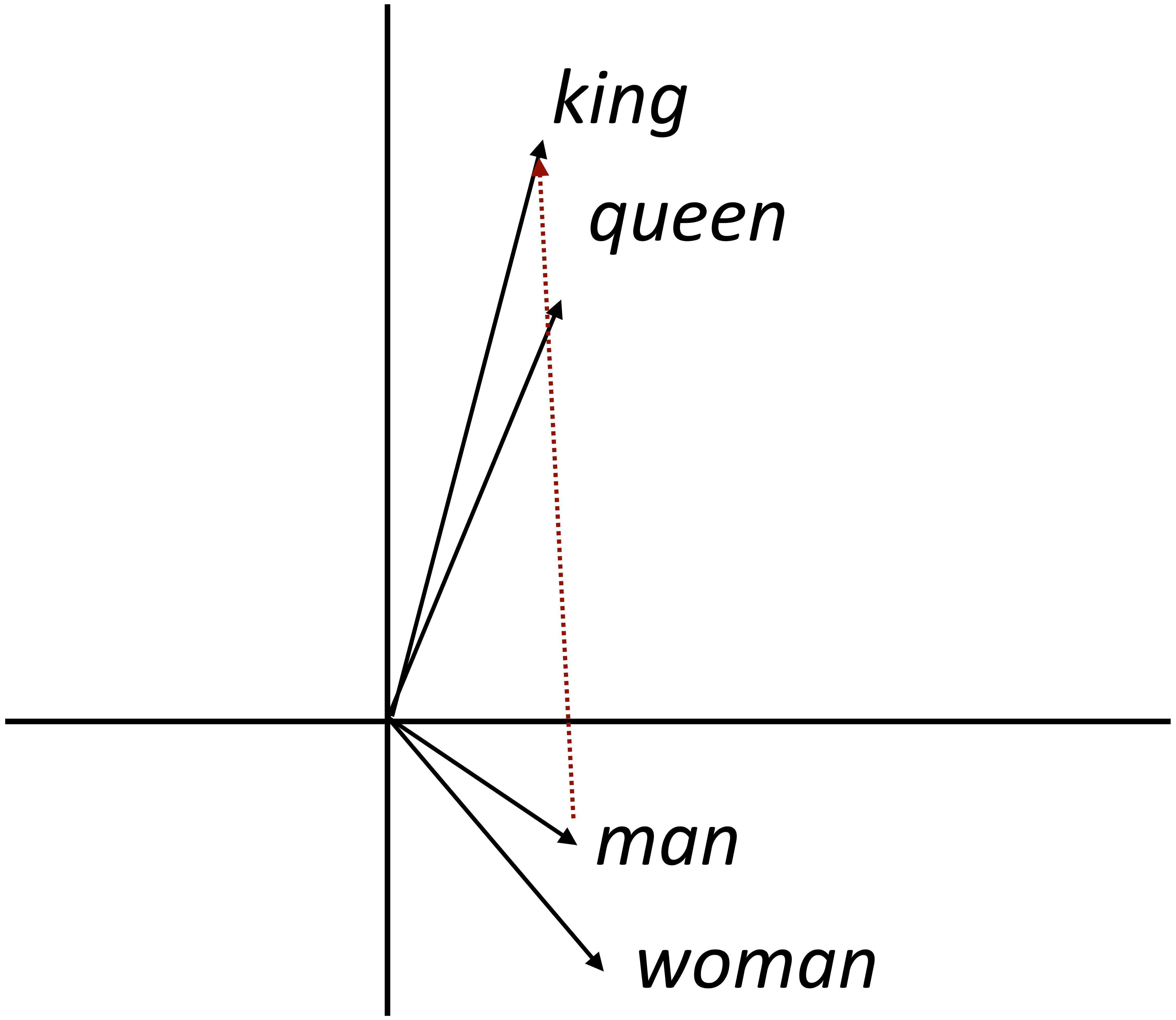
Analogies

(king - man) + woman = queen



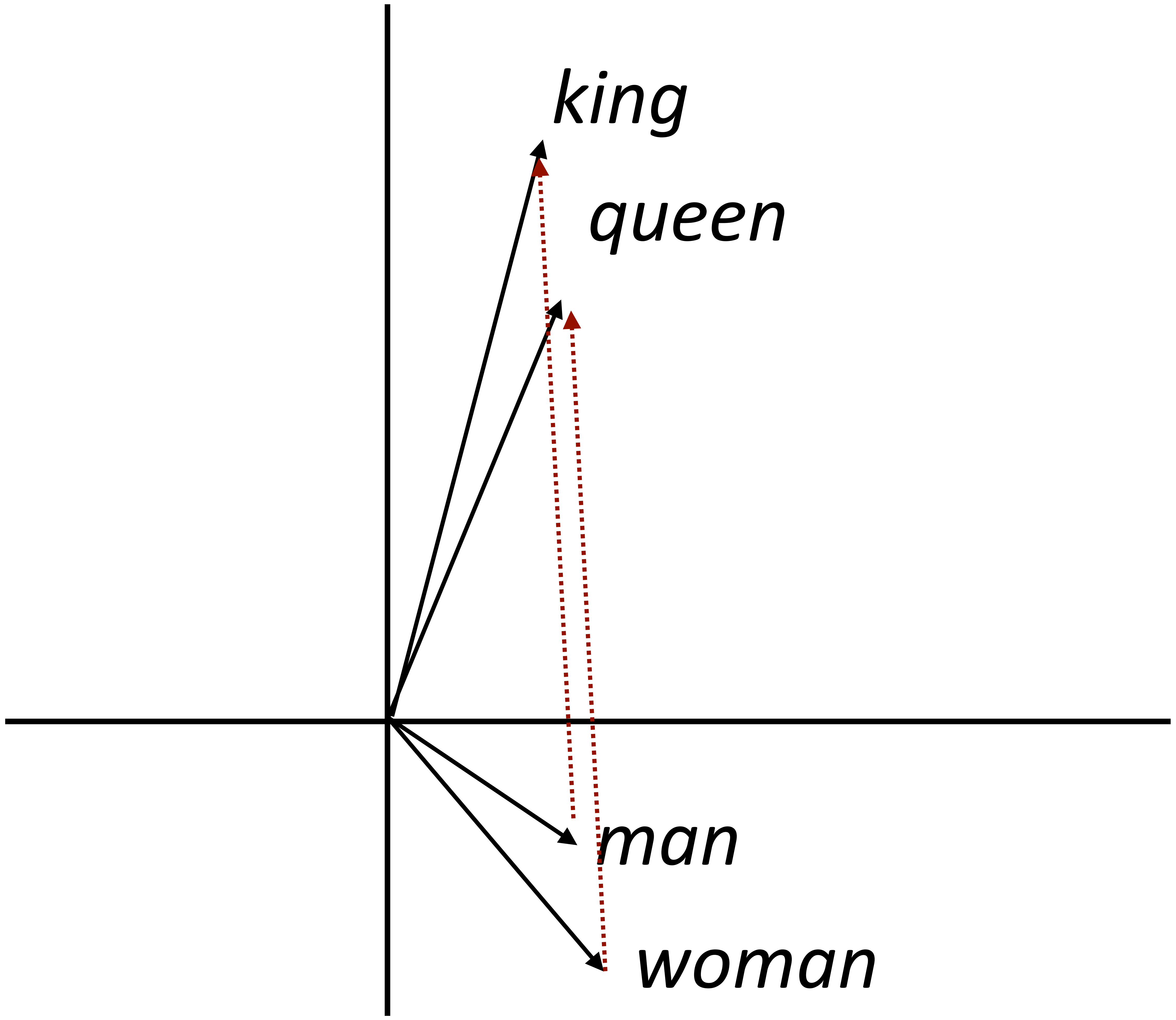
Analogies

(king - man) + woman = queen



Analogies

(king - man) + woman = queen

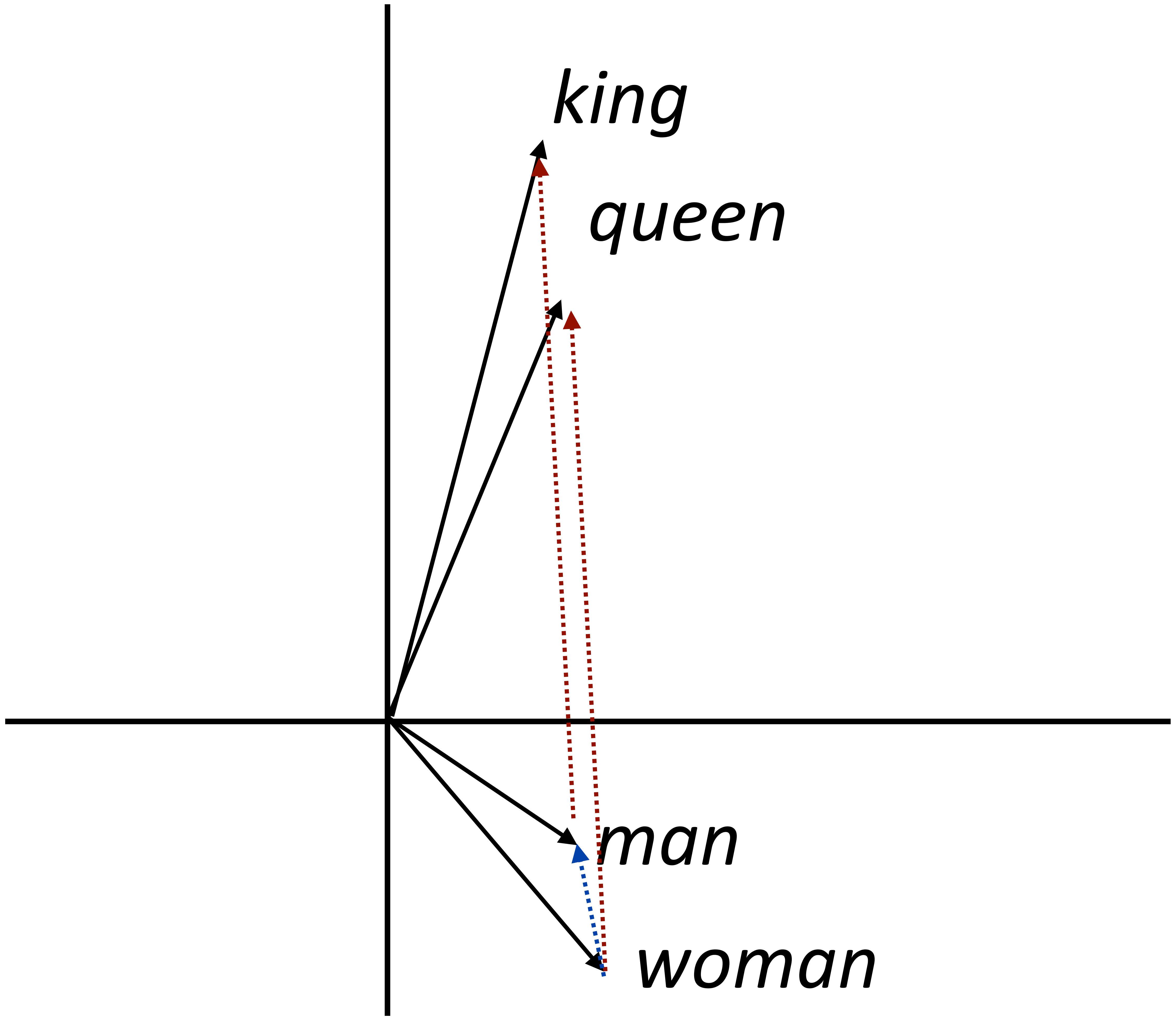


Analogies

(king - man) + woman = queen

king + (woman - man) = queen

- Why would this be?

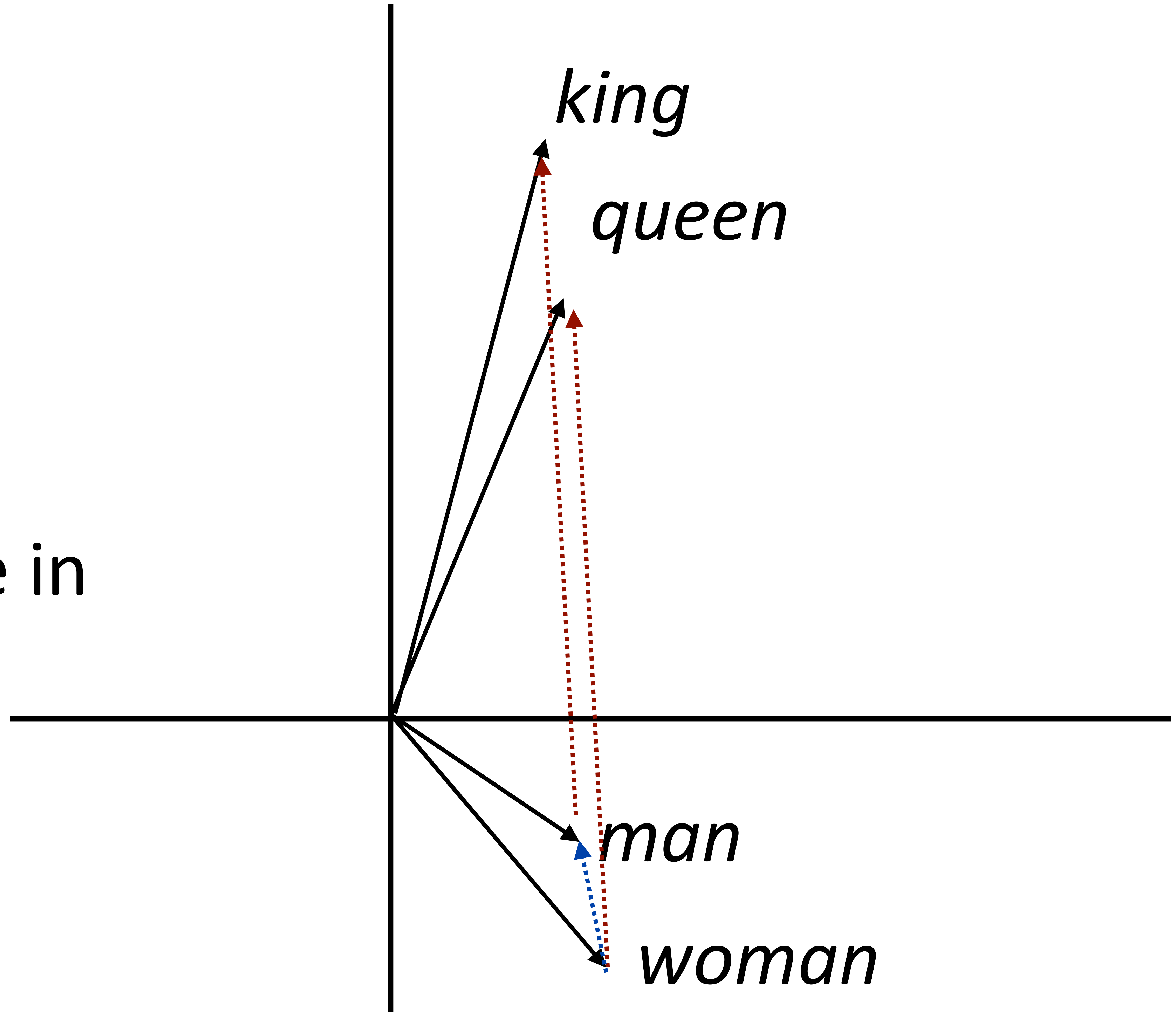


Analogies

(king - man) + woman = queen

king + (woman - man) = queen

- Why would this be?
- woman - man captures the difference in the contexts that these occur in



Analogies

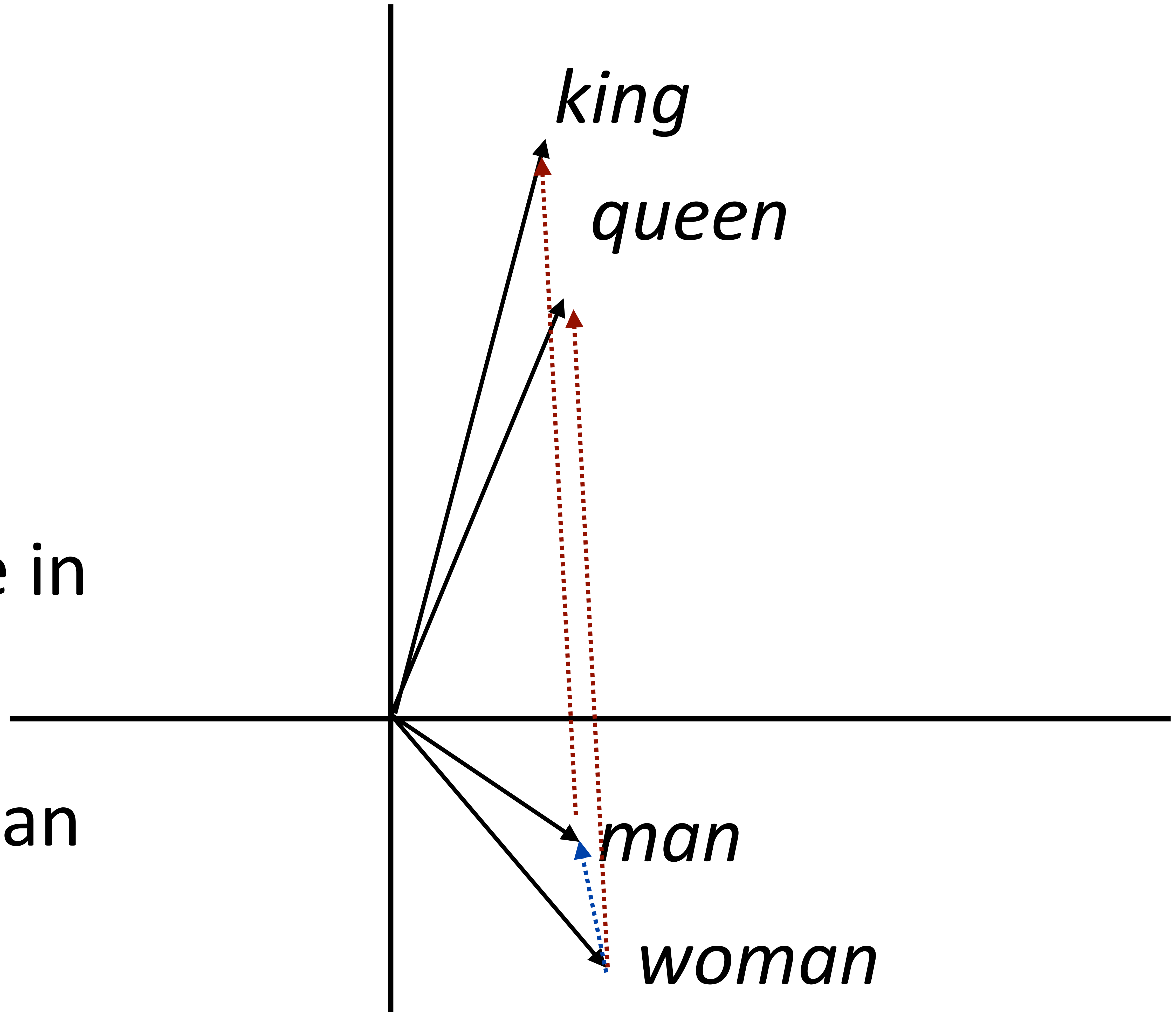
(king - man) + woman = queen

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

Levy et al. (2015)

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

Levy et al. (2015)

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 Word Embeddings

Using Word Embeddings

- Approach 1: learn embeddings as parameters from your data
  - Often works pretty well

Using Word Embeddings

- Approach 1: learn embeddings as parameters from your data
  - Often works pretty well
- Approach 2: initialize using GloVe/ELMo, keep fixed
  - Faster because no need to update these parameters

Using Word Embeddings

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

Compositional Semantics

Compositional Semantics

- What if we want embedding representations for whole sentences?

Compositional Semantics

- What if we want embedding representations for whole sentences?

- Skip-*thought* vectors (Kiros et al., 2015), similar to skip-gram generalized to a sentence level (more later)

Compositional Semantics

- What if we want embedding representations for whole sentences?
- Skip-*thought* vectors (Kiros et al., 2015), similar to skip-gram generalized to a sentence level (more later)
- Is there a way we can compose vectors to make sentence representations? Summing?

Compositional Semantics

- What if we want embedding representations for whole sentences?
- Skip-*thought* vectors (Kiros et al., 2015), similar to skip-gram generalized to a sentence level (more later)
- Is there a way we can compose vectors to make sentence representations? Summing?
- Will return to this in a few weeks

Takeaways

Takeaways

- Lots to tune with neural networks

Takeaways

- Lots to tune with neural networks
  - Training: optimizer, initializer, regularization (dropout), …

Takeaways

- Lots to tune with neural networks
  - Training: optimizer, initializer, regularization (dropout), …
  - Hyperparameters: dimensionality of word embeddings, layers, …

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)

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

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)

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)
- Next time: RNNs and CNNs