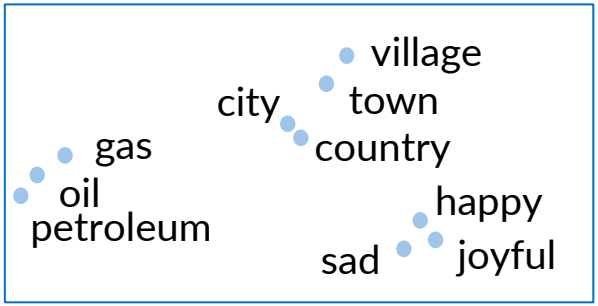
Copyright Notice

deeplearning.ai

Overview

Some basic applications of word embeddings

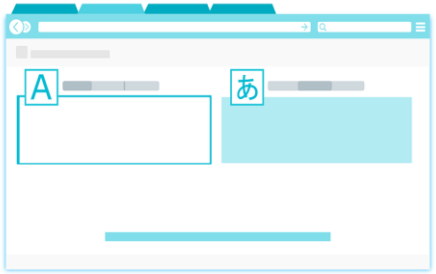


Semantic analogies and similarity

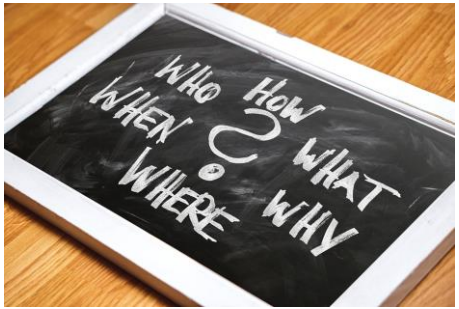
Sentiment analysis

Classification of customer feedback

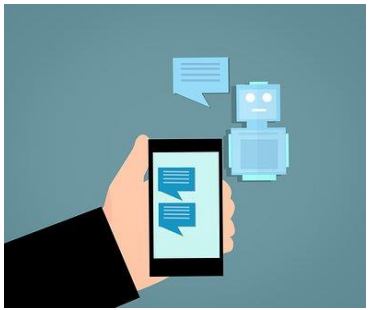
Advanced applications of word embeddings



Machine translation



Information extraction



Question answering

Learning objectives

Prerequisite: neural networks

- Identify the key concepts of word representations
- Generate word embeddings
- Prepare text for machine learning
- Implement the continuous bag-of-words model

deeplearning.ai

Basic Word Representations

Outline

- Integers
- One-hot vectors
- Word embeddings

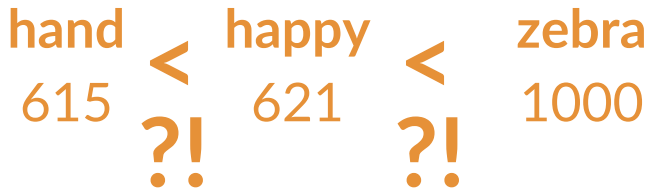
Integers

Word	Number
a	1
able	2
about	3
...	...
hand	615
...	...
happy	621
...	...
zebra	1000

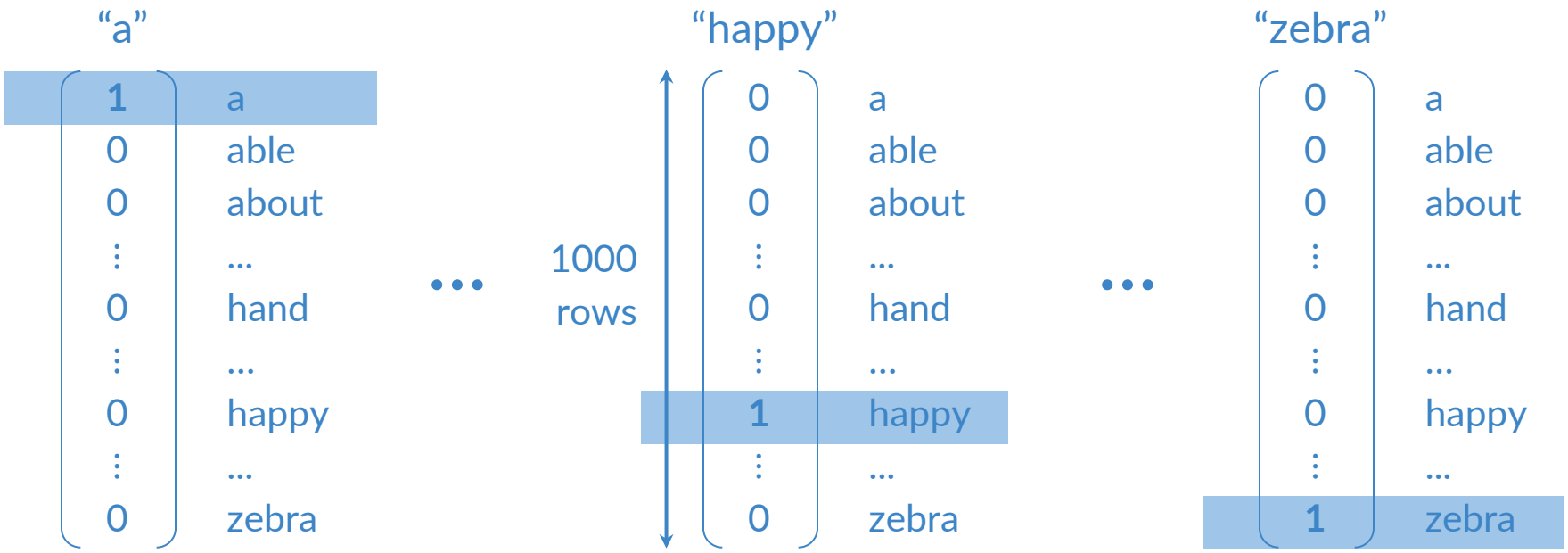
Integers

+ Simple

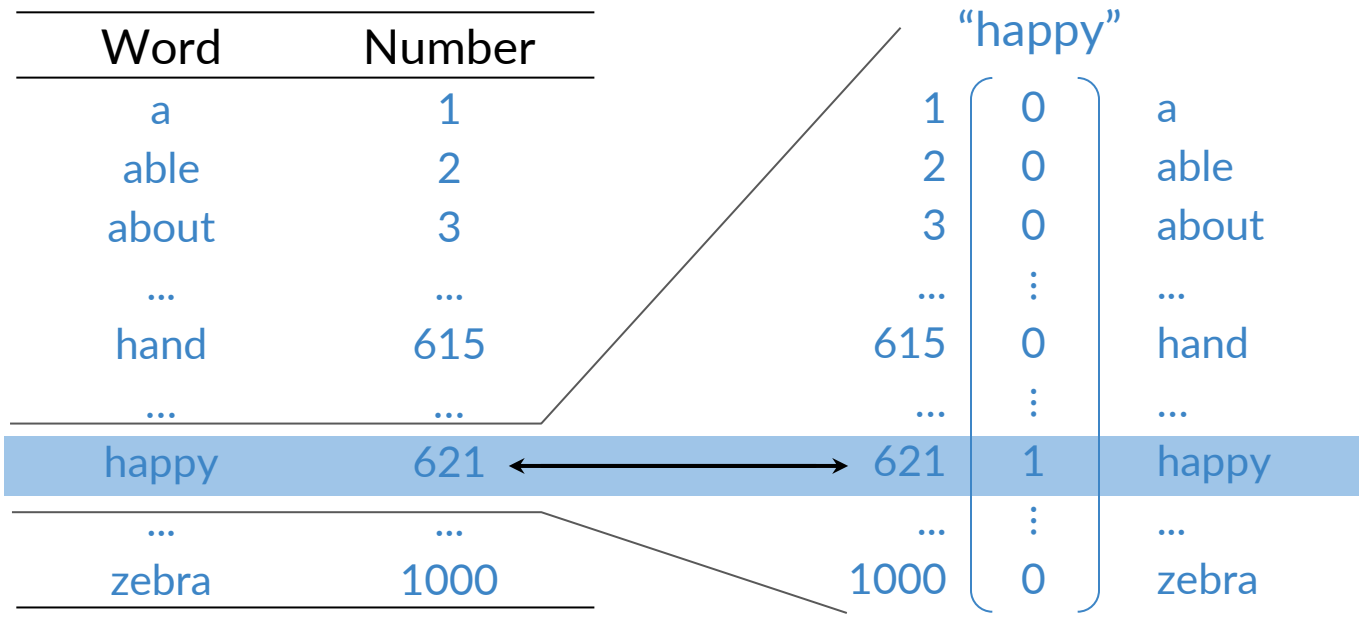
- Ordering: little semantic sense



One-hot vectors

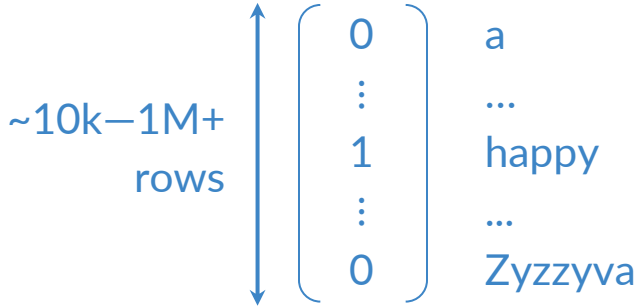


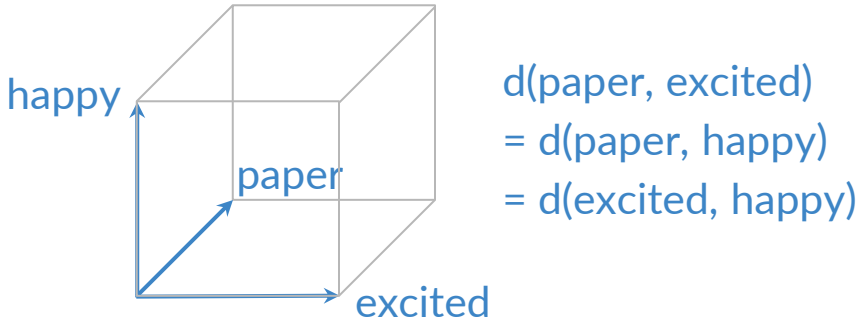
One-hot vectors



One-hot vectors

+ Simple
+ No implied ordering
- Huge vectors
- No embedded meaning

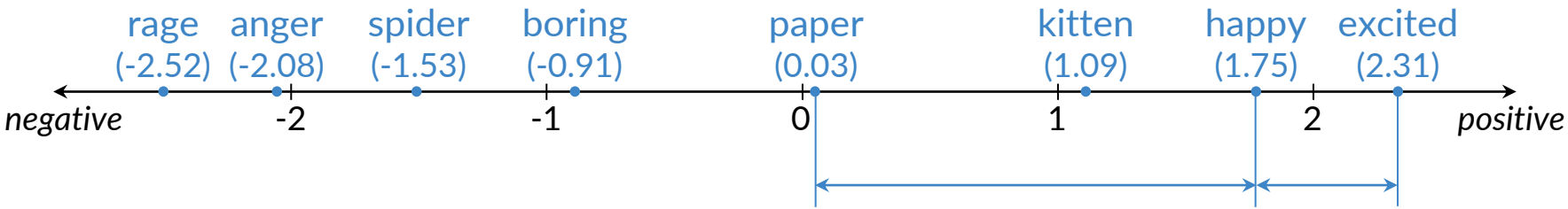




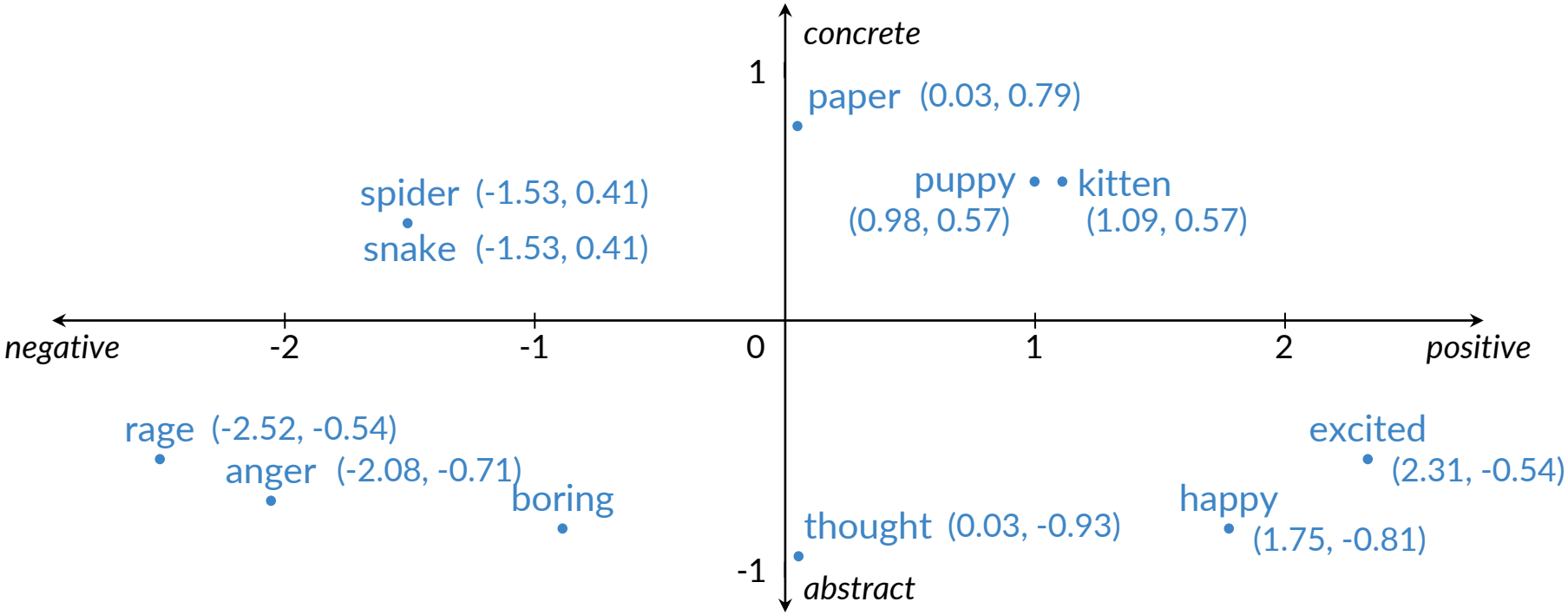
deeplearning.ai

Word Embeddings

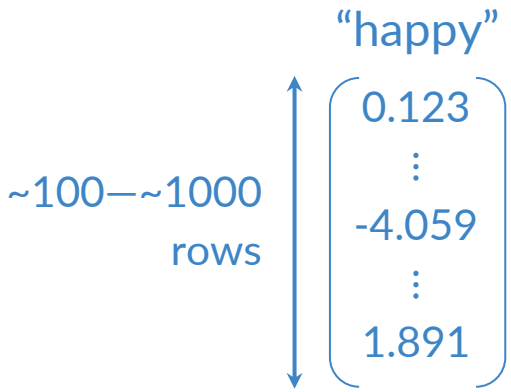
Meaning as vectors



Meaning as vectors



Word embedding vectors



+ Low dimension

+ Embed meaning
  - e.g. semantic distance

    forest ≈ tree forest ≉ ticket

  - e.g. analogies

    Paris:France :: Rome:?

Terminology

integers

word vectors

one-hot vectors

word embedding vectors

"word vectors"

word embeddings

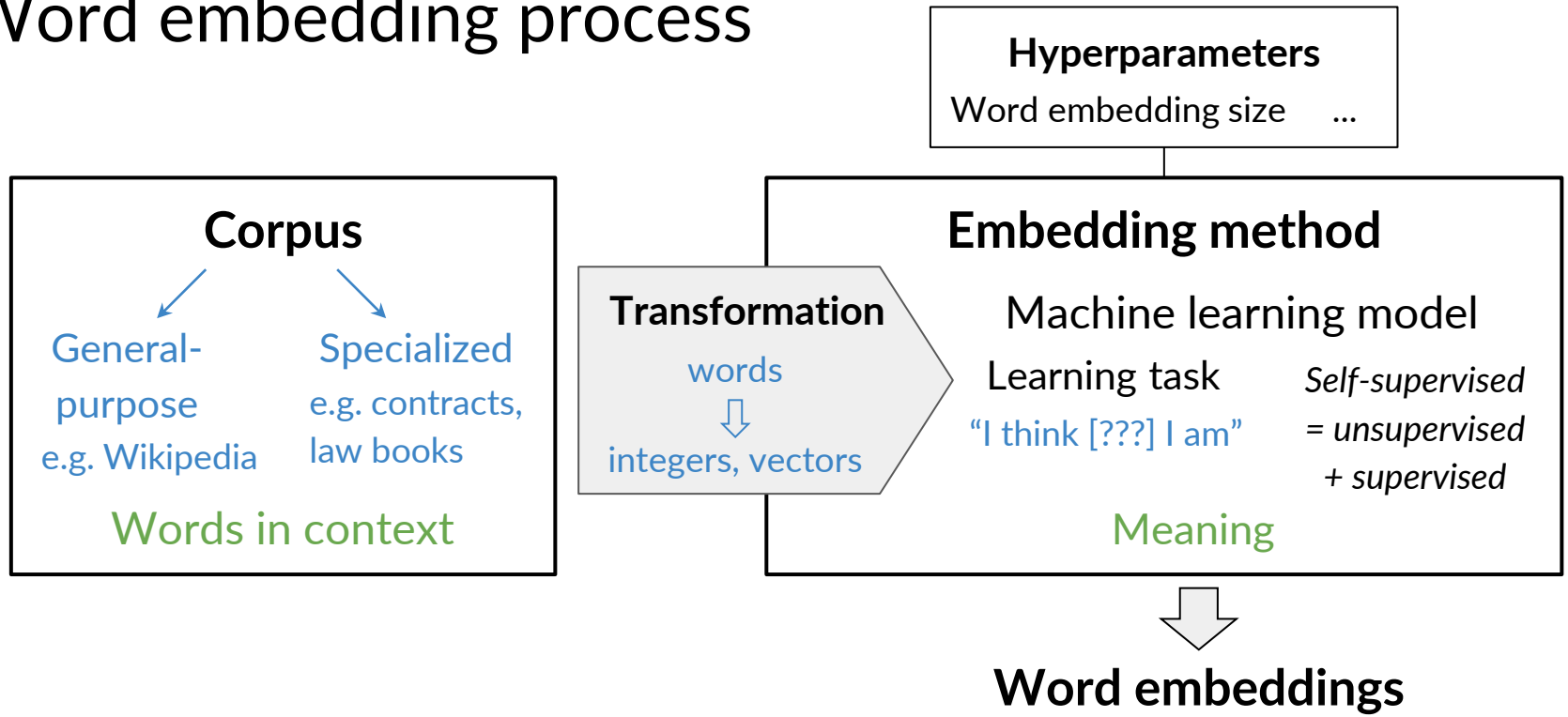
Summary

- Words as integers
- Words as vectors
  - One-hot vectors
  - Word embedding vectors
- Benefits of word embeddings for NLP

deeplearning.ai

How to Create Word Embeddings

Word embedding process



deeplearning.ai

Word Embedding Methods

Basic word embedding methods

- word2vec (Google, 2013)
  - Continuous bag-of-words (CBOW)
  - Continuous skip-gram / Skip-gram with negative sampling (SGNS)

- Global Vectors (GloVe) (Stanford, 2014)

- fastText (Facebook, 2016)
  - Supports out-of-vocabulary (OOV) words

Advanced word embedding methods

Deep learning, contextual embeddings

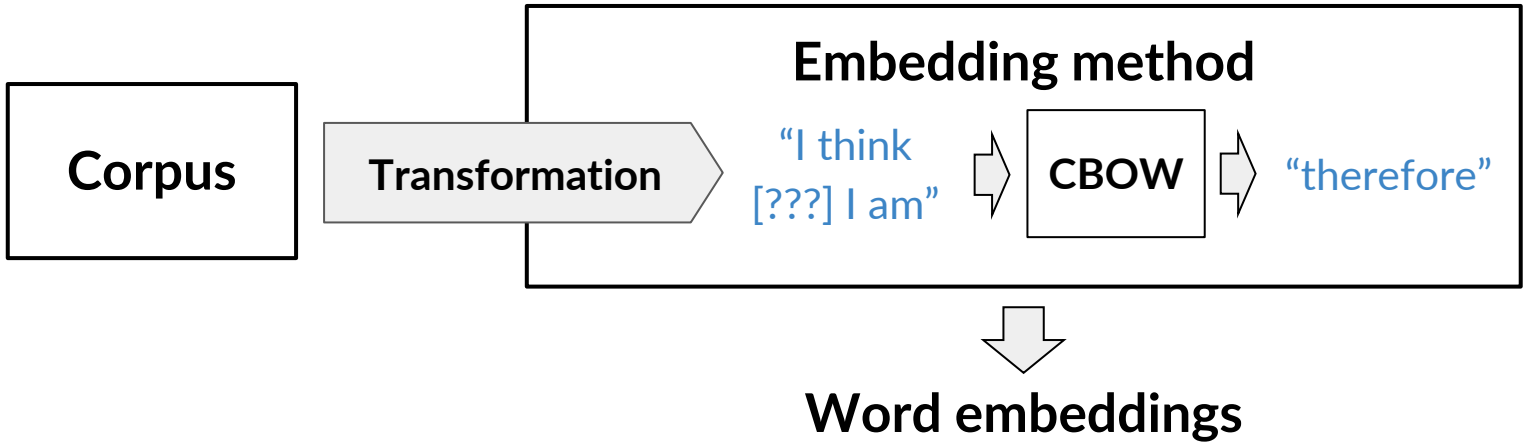
- BERT (Google, 2018)
- ELMo (Allen Institute for AI, 2018)
- GPT-2 (OpenAI, 2018)

} Tunable pre-trained models available

deeplearning.ai

Continuous Bag-of-Words Model

Continuous bag-of-words word embedding process



Center word prediction: rationale

The little ____?____ is barking

dog

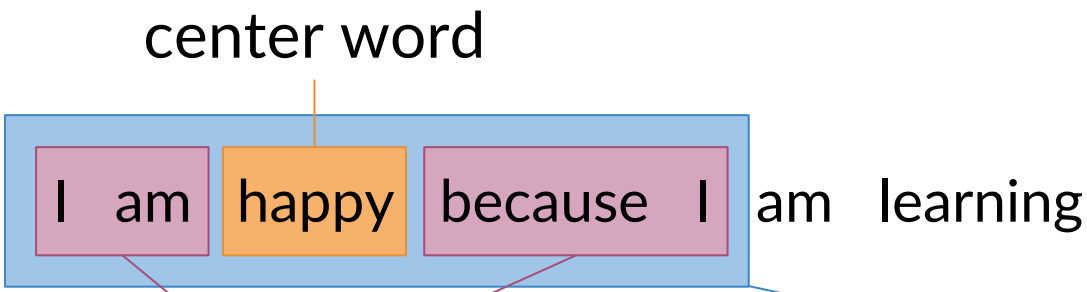
puppy

hound

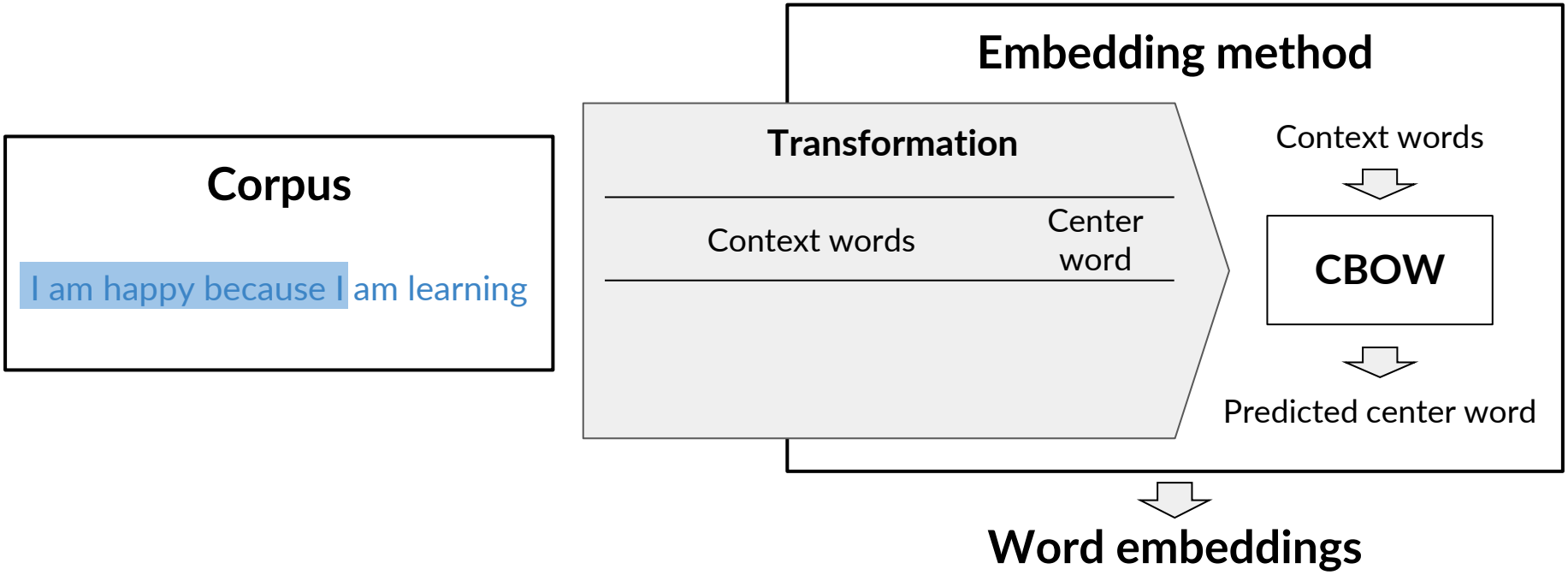
terrier

...

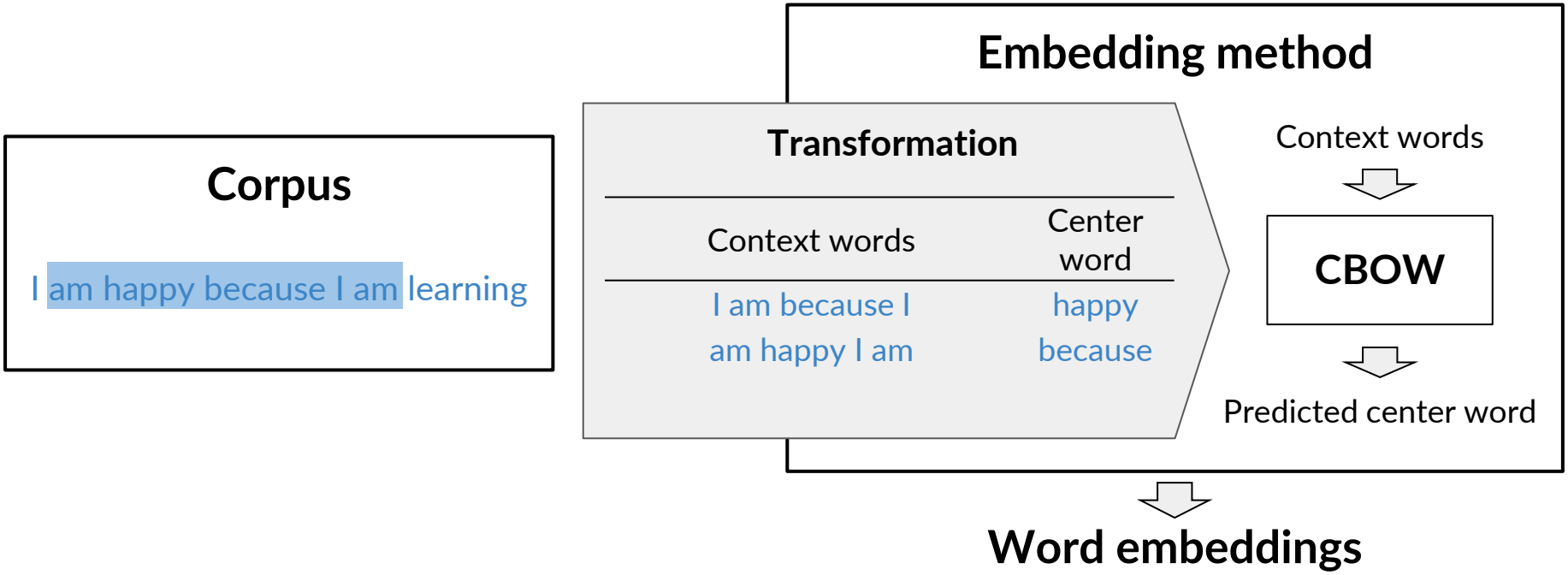
Creating a training example



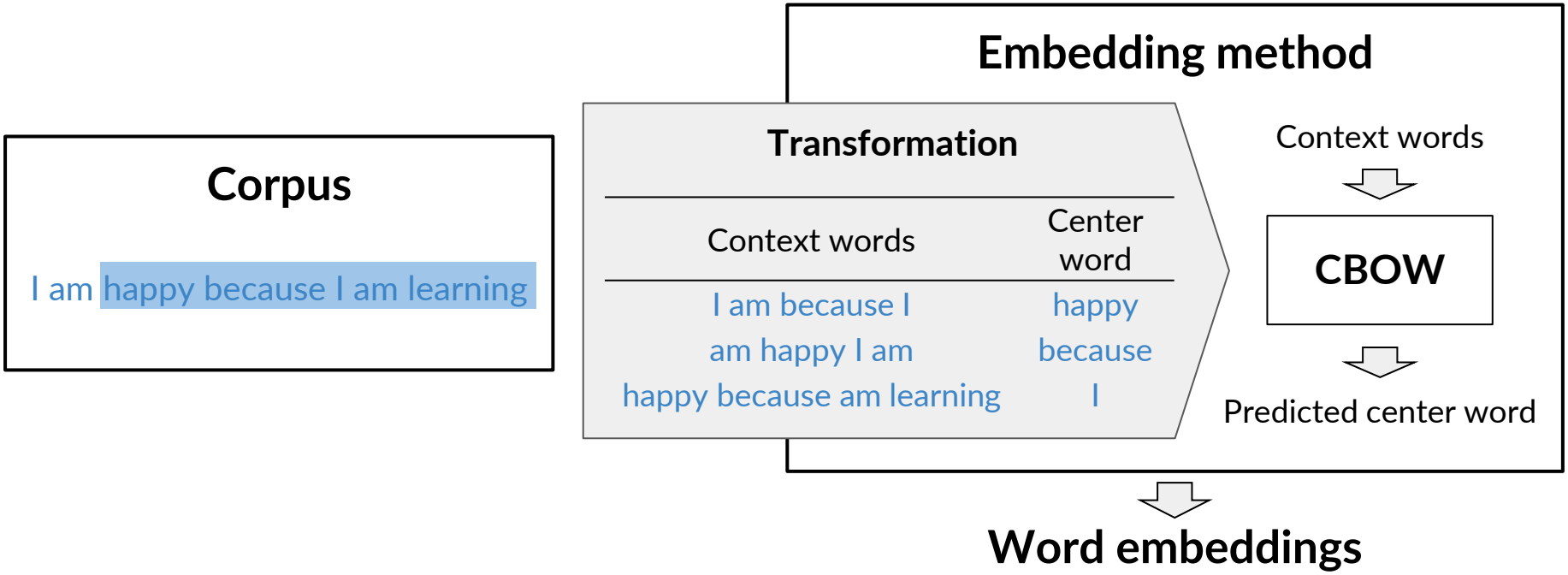
From corpus to training



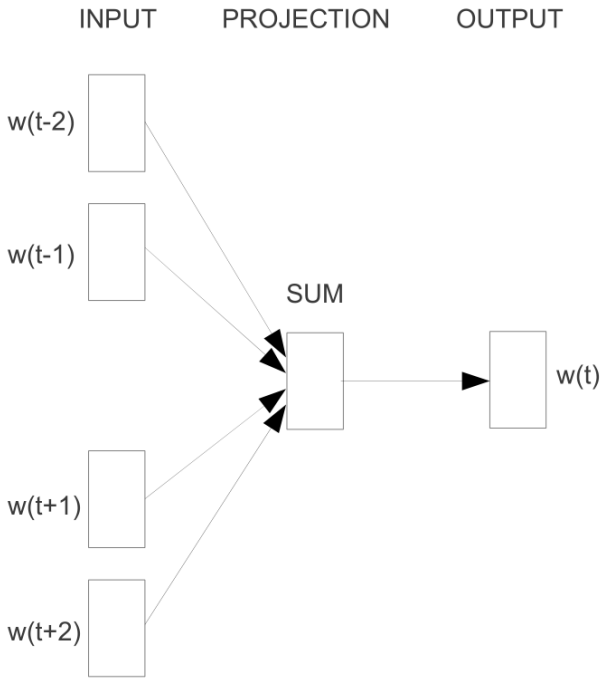
From corpus to training



From corpus to training



CBOW in a nutshell



Source: Mikolov, T., Chen, K., Corrado, G.S., & Dean, J. (2013). Efficient Estimation of Word Representations in Vector Space

deeplearning.ai

Cleaning and Tokenization

Cleaning and tokenization matters

- Letter case “The” == “the” == “THE” → *lowercase / upper case*

Cleaning and tokenization matters

- Letter case “The” == “the” == “THE” → *lowercase / upper case*
- Punctuation , ! . ? → . “ ‘ « » ’ ” → ∅ … !! ??? → .

Cleaning and tokenization matters

- Letter case — “The” == “the” == “THE” → *lowercase / upper case*
- Punctuation — , ! . ? → . “ ‘ « » ’ ” → ∅ … !! ??? → .
- Numbers — 1 2 3 5 8 → ∅ 3.14159 90210 → *as is/<NUMBER>*

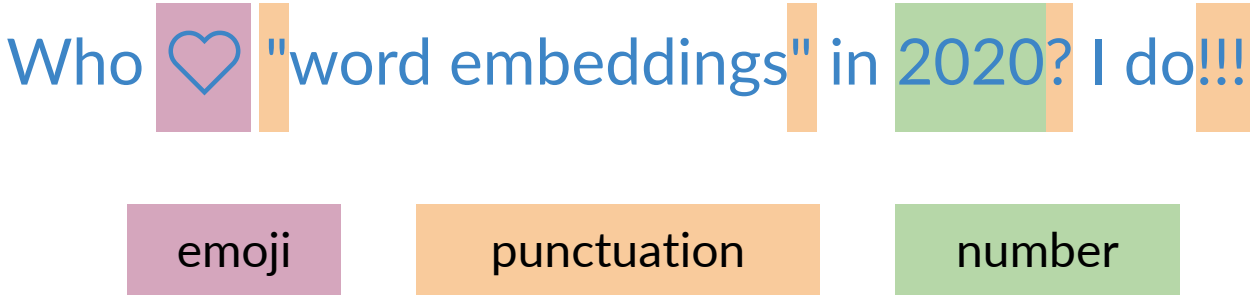
Cleaning and tokenization matters

- Letter case — “The” == “the” == “THE” → *lowercase / upper case*
- Punctuation — , ! . ? → . “ ‘ « » ’ ” → ∅ … !! ??? → .
- Numbers — 1 2 3 5 8 → ∅ 3.14159 90210 → *as is/<NUMBER>*
- Special characters — ∇ $ € § ¶ ** → ∅

Cleaning and tokenization matters

- Letter case — “The” == “the” == “THE” → *lowercase / upper case*
- Punctuation — , ! . ? → . “ ‘ « » ’ ” → ∅ … !! ??? → .
- Numbers — 1 2 3 5 8 → ∅ 3.14159 90210 → *as is/<NUMBER>*
- Special characters — ∇ $ € § ¶ ** → ∅
- Special words — 😊 #nlp → :happy: #nlp

Example in Python: corpus



Example in Python: libraries

```
# pip install nltk
# pip install emoji

import nltk
from nltk.tokenize import word_tokenize
import emoji

nltk.download('punkt')  # download pre-trained Punkt tokenizer for English
```

Example in Python: code

```
corpus = 'Who ♡ "word embeddings" in 2020? I do!!!'

data = re.sub(r'[,!?;-]+', '.', corpus)
```

→ Who ♡ "word embeddings" in 2020. I do.

Example in Python: code

```
corpus = 'Who ♡ "word embeddings" in 2020? I do!!!'

data = re.sub(r'[,!?;-]+', '.', corpus)
data = nltk.word_tokenize(data)  # tokenize string to words
```

→ **['Who', '♡', '``', 'word', 'embeddings', "''", 'in', '2020', '.', 'I', 'do', '.']**

Example in Python: code

```
corpus = 'Who ♡ "word embeddings" in 2020? I do!!!'

data = re.sub(r'[,!?;-]+', '.', corpus)
data = nltk.word_tokenize(data)  # tokenize string to words
data = [ ch.lower() for ch in data
         if ch.isalpha()
         or ch == '.'
         or emoji.get_emoji_regexp().search(ch)
       ]
```

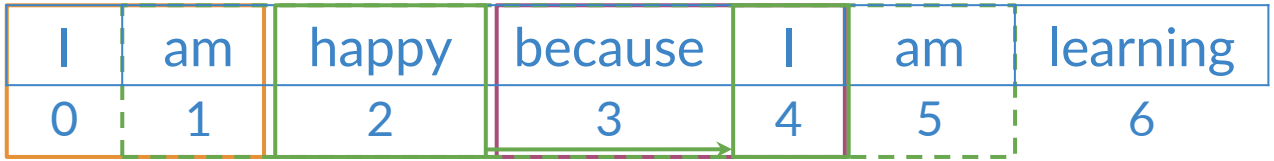
```
→ ['who', '♡', 'word', 'embeddings', 'in', '.', 'i', 'do', '.']
```

deeplearning.ai

Sliding Window of Words in Python

Sliding window of words in Python

```
def get_windows(words, C):
    i = C
    while i < len(words) - C:
        center_word = words[i]
        context_words = words[(i - C):i] + words[(i+1):(i+C+1)]
        yield context_words, center_word
        i += 1
```



Sliding window of words in Python

```
def get_windows(words, C):
    ...
        yield context_words, center_word
```

```
for x, y in get_windows(
            ['i', 'am', 'happy', 'because', 'i', 'am', 'learning'],
            2
        ):
    print(f'{x}\t{y}')
```

Sliding window of words in Python

```
for x, y in get_windows(
            ['i', 'am', 'happy', 'because', 'i', 'am', 'learning'],
            2
        ):
    print(f'{x}\t{y}')
```

```
→ ['I', 'am', 'because', 'I']        happy
  ['am', 'happy', 'I', 'am']                    because
  ['happy', 'because', 'am', 'learning']        I
```

deeplearning.ai

Transforming Words into Vectors

Transforming center words into vectors

Corpus: I am happy because I am learning

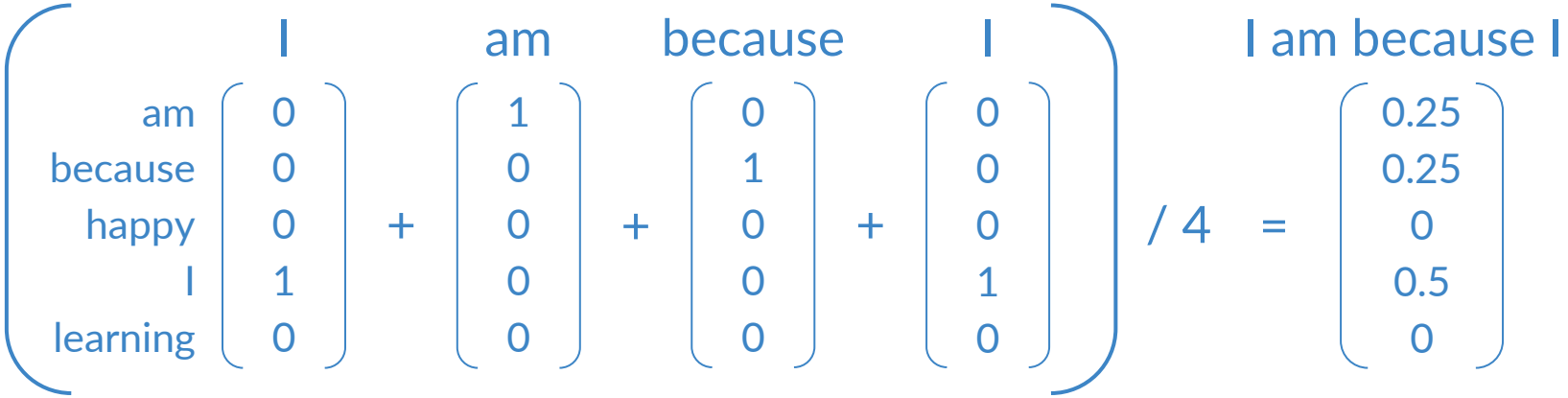
Vocabulary: am, because, happy, I, learning

One-hot vector

	am	because	happy	I	learning
am	1	0	0	0	0
because	0	1	0	0	0
happy	0	0	1	0	0
I	0	0	0	1	0
learning	0	0	0	0	1

Transforming context words into vectors

Average of individual one-hot vectors



Final prepared training set

Context words	Context words vector	*Center word*	Center word vector
I am because I	[0.25; 0.25; 0; 0.5; 0]	*happy*	[0; 0; 1; 0; 0]

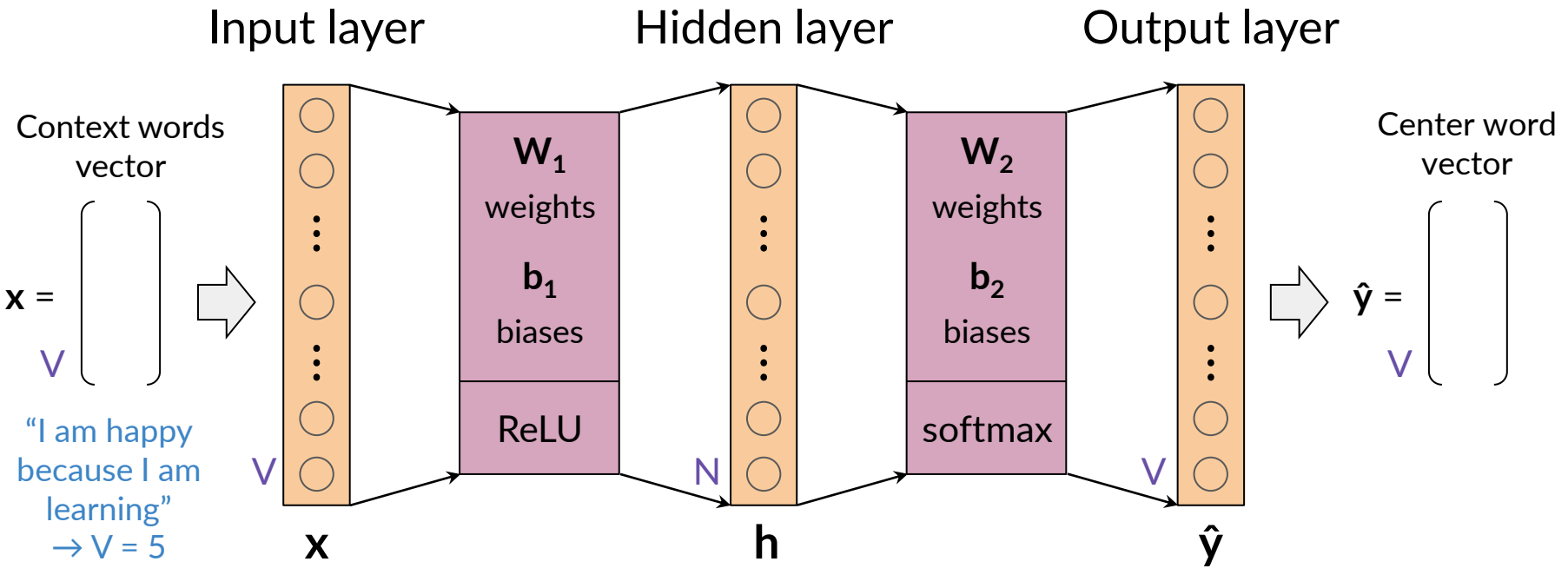
deeplearning.ai

Architecture of the CBOW Model

Architecture of the CBOW model

Hyperparameters

N: Word embedding size ...

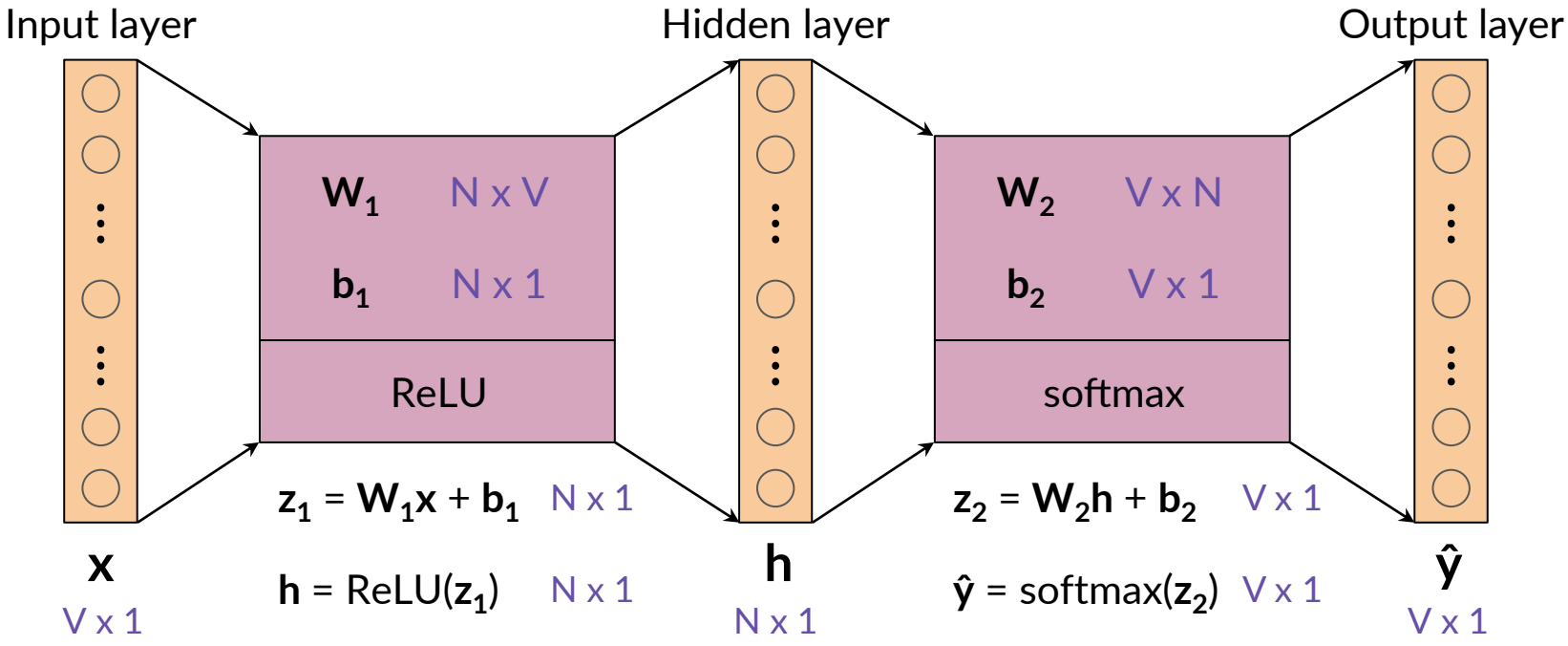


deeplearning.ai

Architecture of the CBOW Model:

Dimensions

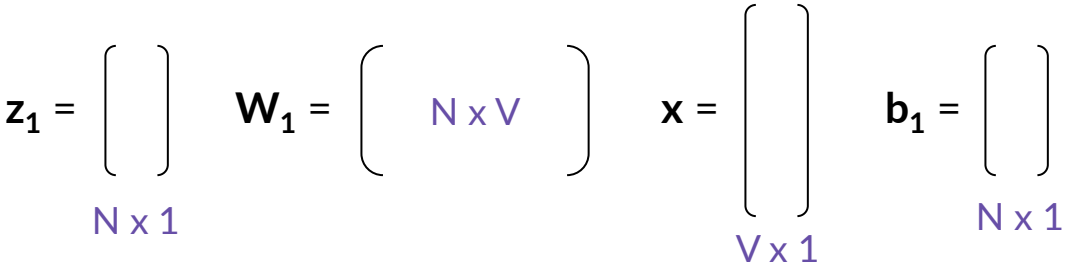
Dimensions (single input)



Dimensions (single input)

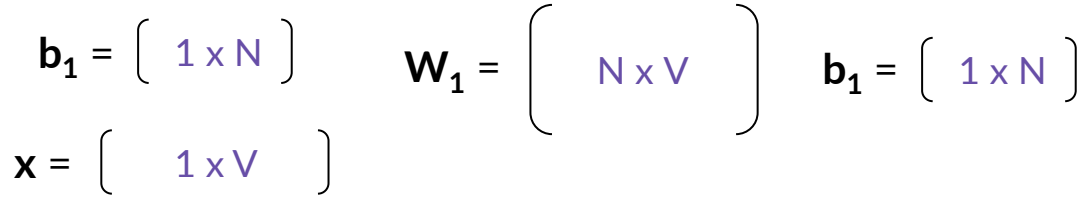
Column vectors

$$z_1 = W_1 x + b_1$$



Row vectors

$$z_1 = x W_1^{\mathsf{T}} + b_1$$



deeplearning.ai

Architecture of the CBOW Model:

Dimensions 2

Dimensions (batch input)

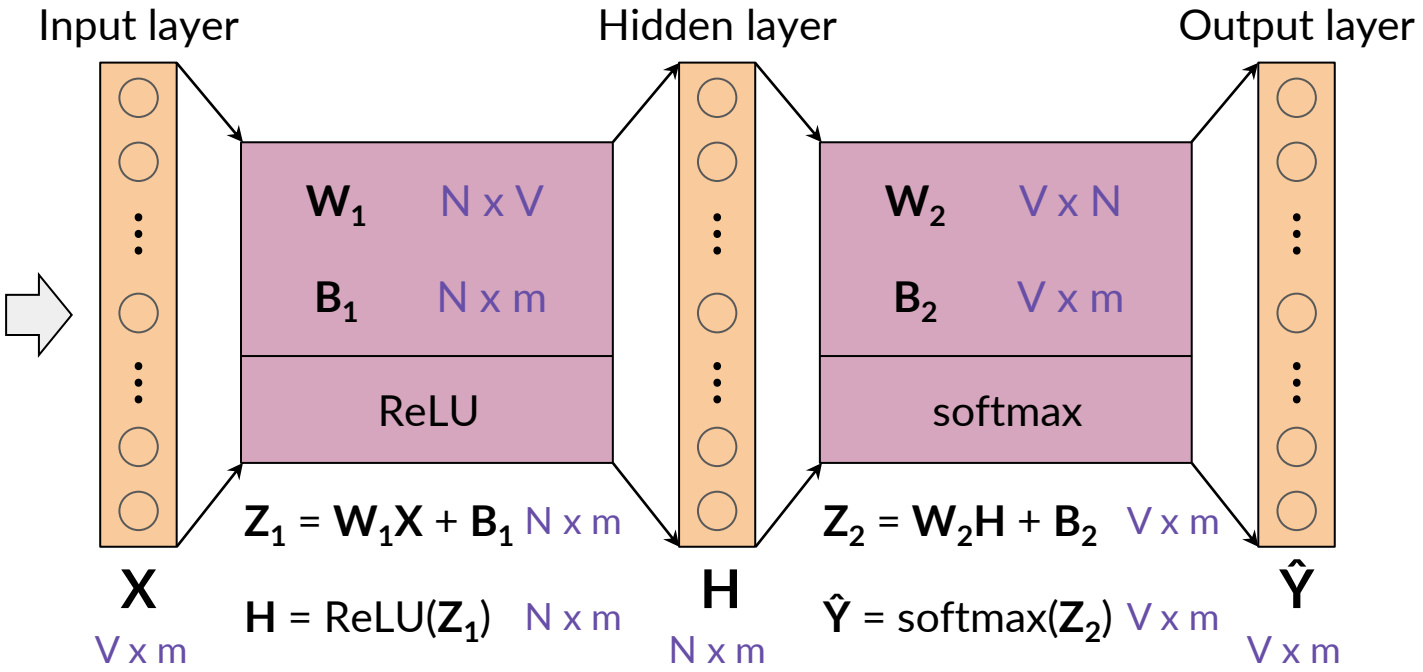
$$\begin{pmatrix} b_1 \end{pmatrix} \rightarrow B_1 = \begin{pmatrix} \begin{pmatrix} b_1 \end{pmatrix} & \cdots & \begin{pmatrix} b_1 \end{pmatrix} \end{pmatrix} \quad N \quad \textit{broadcasting}$$

m

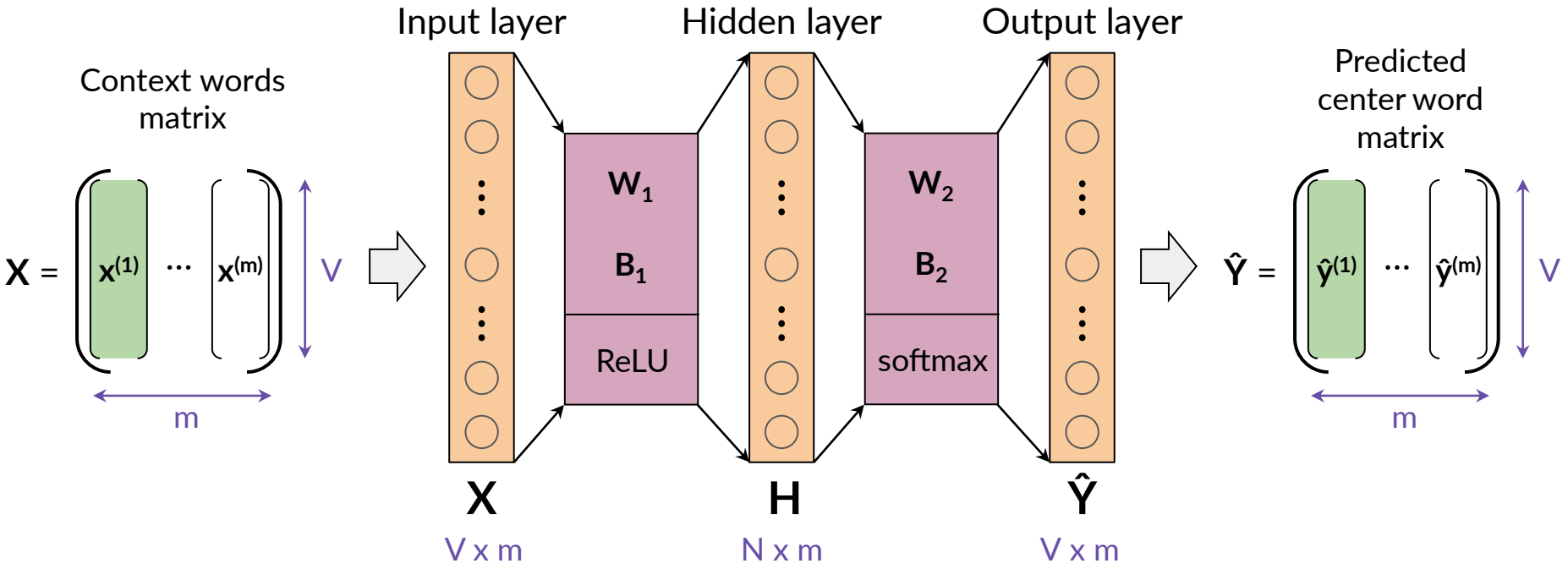
Context words vectors → matrix

$$X = \begin{pmatrix} x^{(1)} & \cdots & x^{(m)} \end{pmatrix} \quad V$$

m



Dimensions (batch input)

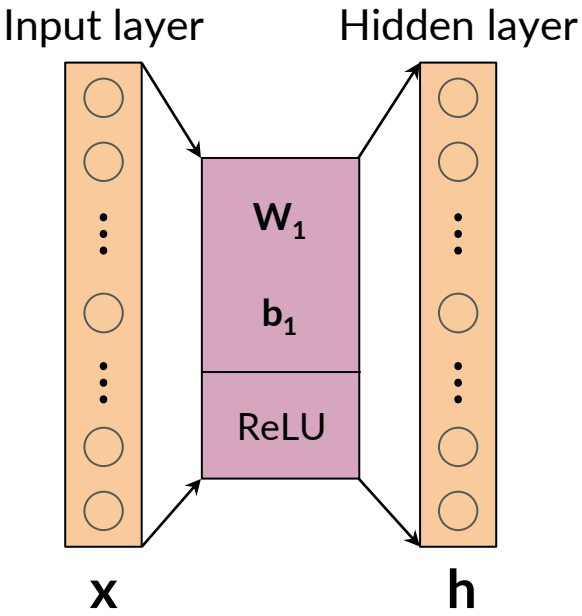


deeplearning.ai

Architecture of the CBOW Model

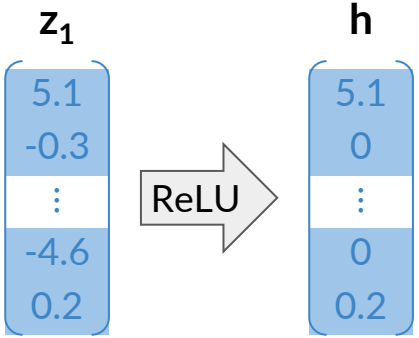
Activation Functions

Rectified Linear Unit (ReLU)

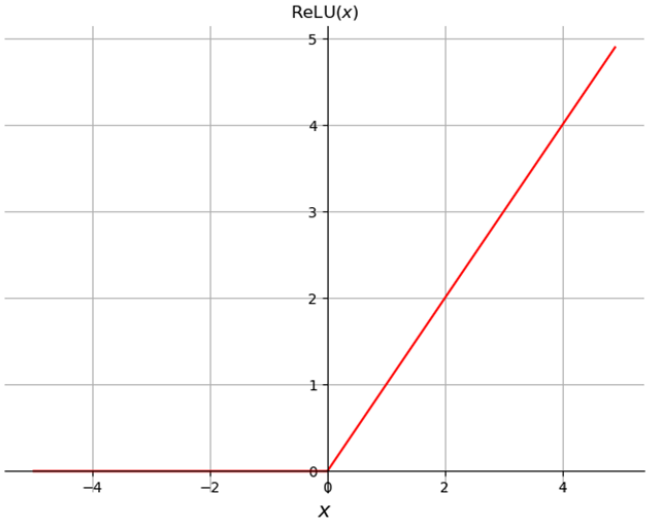


$$z_1 = W_1 x + b_1$$

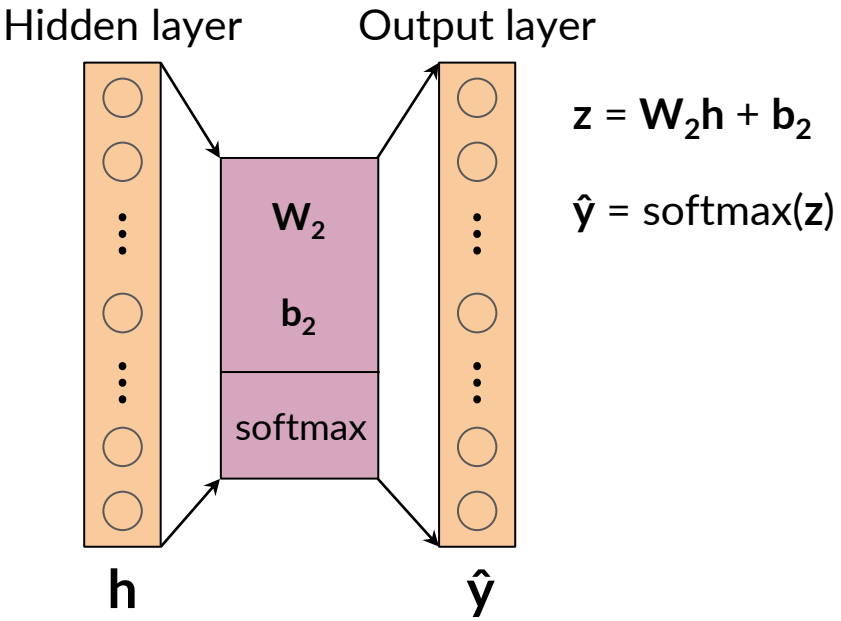
$$h = \text{ReLU}(z_1)$$



ReLU(x) = max(0, x)

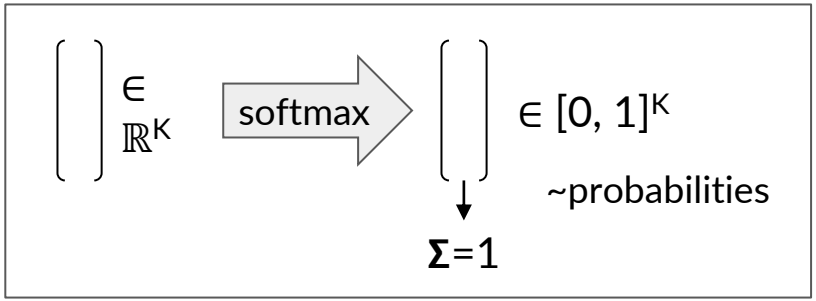


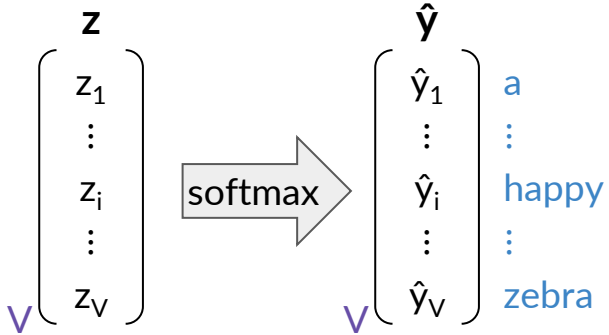
Softmax



$\mathbf{z} = \mathbf{W_2 h} + \mathbf{b_2}$

$\hat{\mathbf{y}} = \text{softmax}(\mathbf{z})$



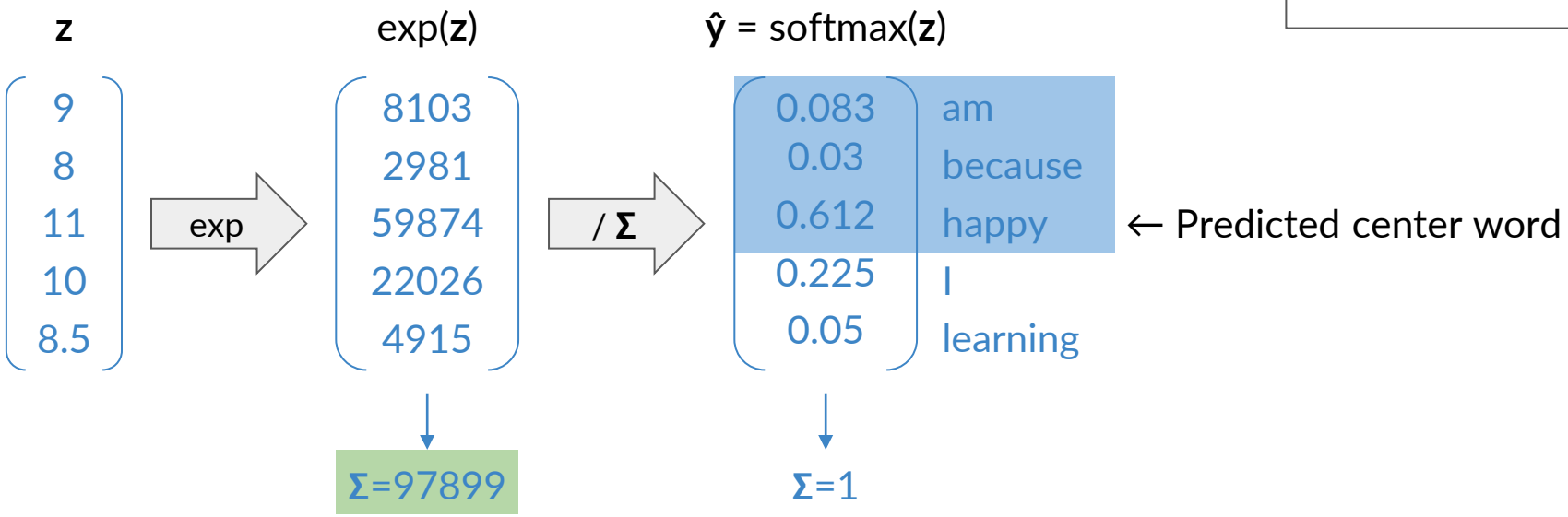


$$\hat{y}_i = \frac{e^{z_i}}{\sum_{j=1}^{V} e^{z_j}}$$

Probabilities of being center word

Softmax: example

$$\hat{y}_i = \frac{e^{z_i}}{\sum_{j=1}^{V} e^{z_j}}$$

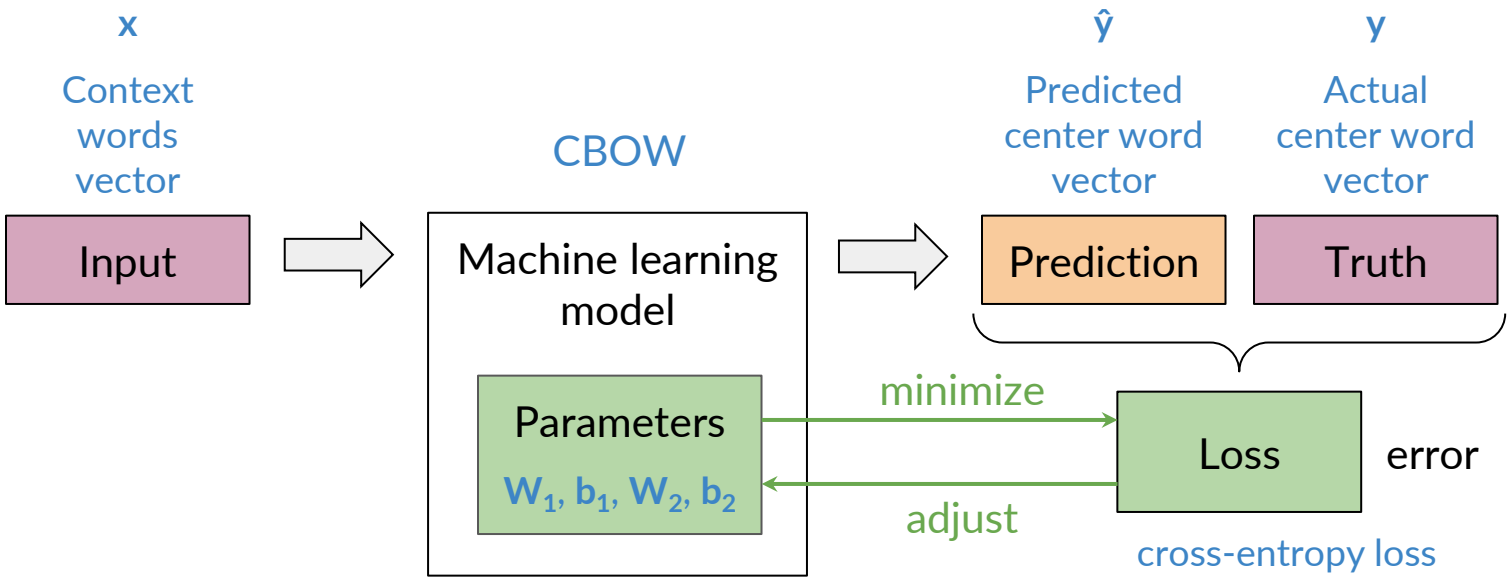


deeplearning.ai

Training a CBOW Model

Cost Function

Loss

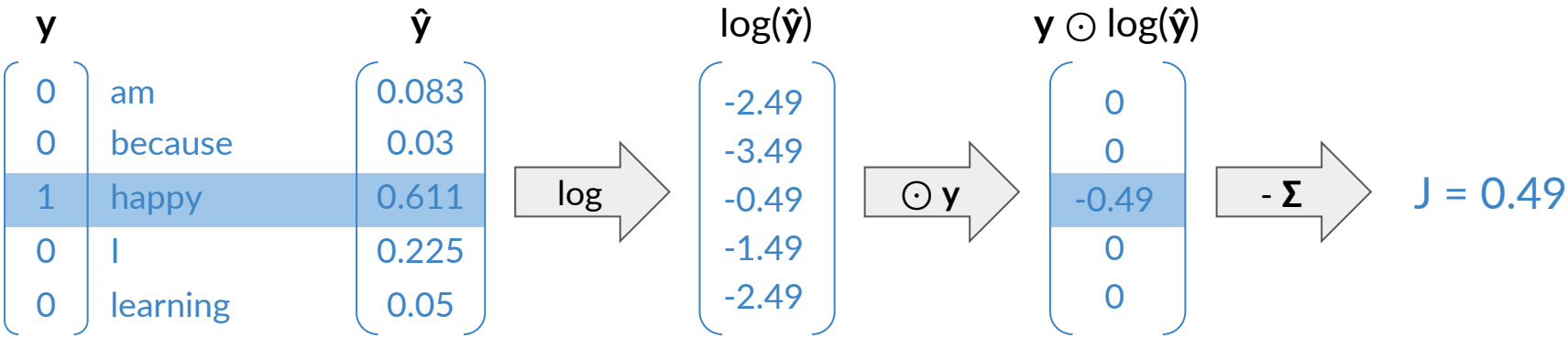

x
Context words vector
Input
CBOW
Machine learning model
Parameters
W_1, b_1, W_2, b_2
ŷ
Predicted center word vector
Prediction
y
Actual center word vector
Truth
minimize
adjust
Loss
error
cross-entropy loss

Cross-entropy loss

$$J = -\sum_{k=1}^{V} y_k \log \hat{y}_k$$

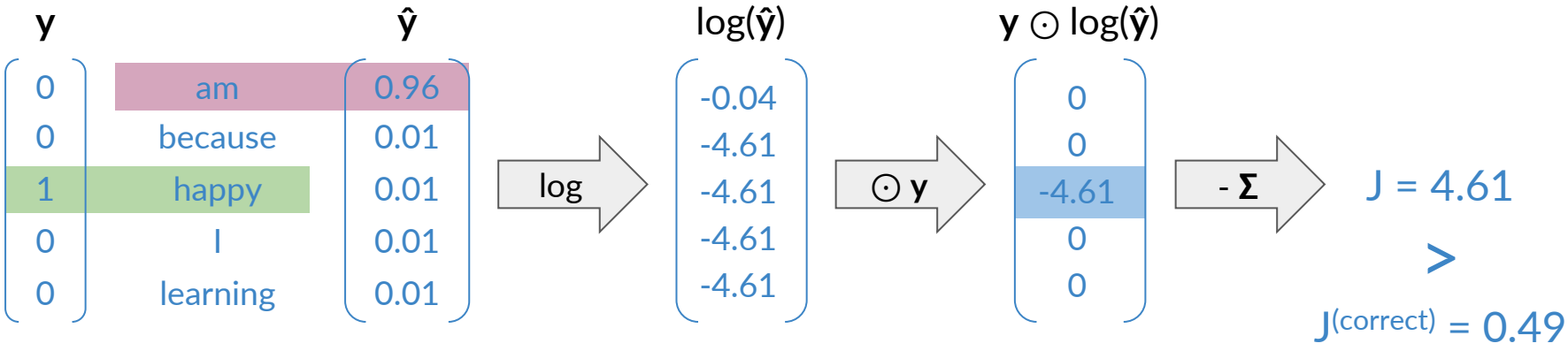
Actual $\mathbf{y} = \begin{pmatrix} y_1 \\ \vdots \\ y_V \end{pmatrix}$ Predicted $\hat{\mathbf{y}} = \begin{pmatrix} \hat{y}_1 \\ \vdots \\ \hat{y}_V \end{pmatrix}$

I am happy because I am learning



Cross-entropy loss

$$J = -\sum_{k=1}^{V} y_k \log \hat{y}_k$$



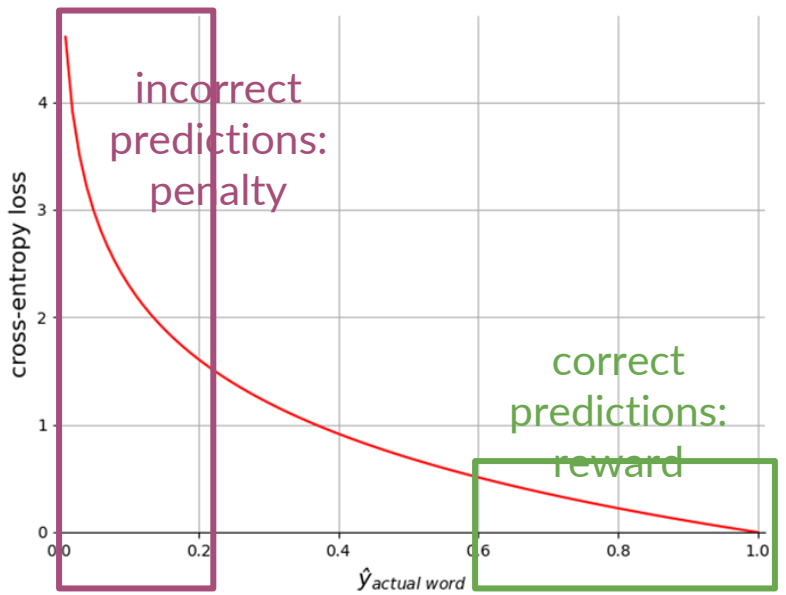
Cross-entropy loss

$$J = -\sum_{k=1}^{V} y_k \log \hat{y}_k$$

$J = -\log \hat{y}_{\text{actual word}}$

y		$\hat{y}$
0	am	0.96
0	because	0.01
1	happy	0.01
0	I	0.01
0	learning	0.01

→ J = 4.61



deeplearning.ai

Training a CBOW Model

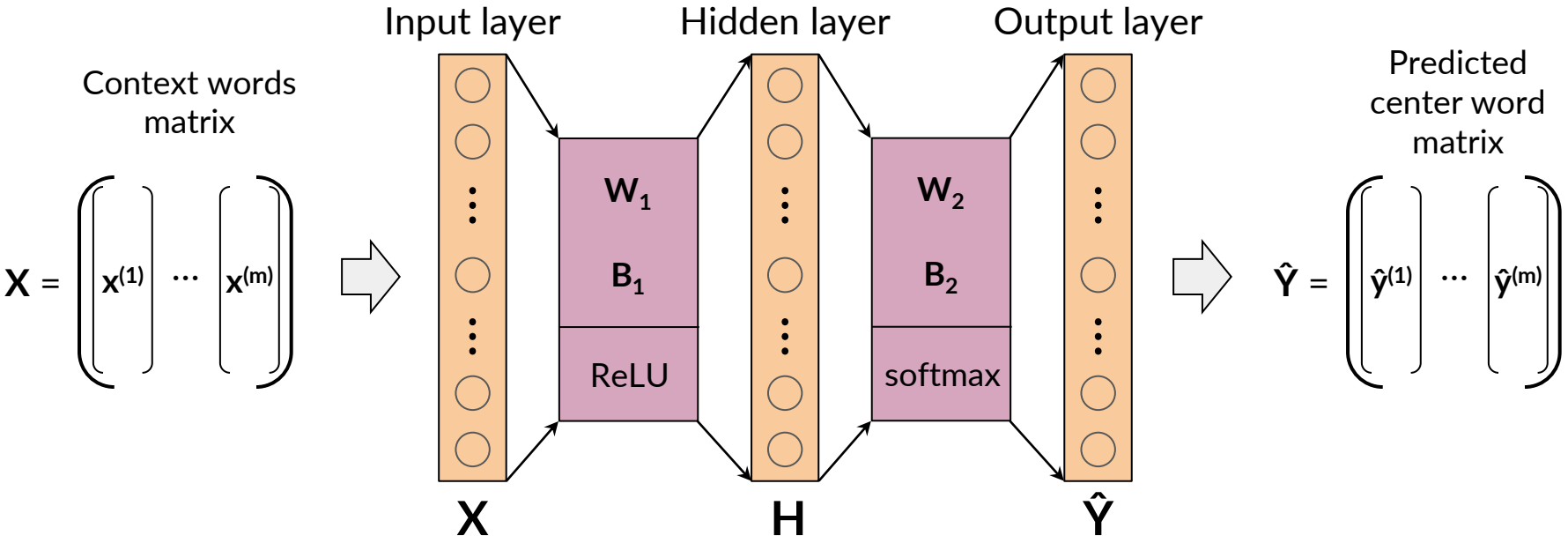
Forward Propagation

Training process

- Forward propagation
- Cost
- Backpropagation and gradient descent

Forward propagation

$$Z_1 = W_1X + B_1 \qquad Z_2 = W_2H + B_2$$
$$H = \text{ReLU}(Z_1) \qquad \hat{Y} = \text{softmax}(Z_2)$$



Cost

$$J = -\sum_{k=1}^{V} y_k \log \hat{y}_k$$

Cost: mean of losses

$$J_{batch} = -\frac{1}{m}\sum_{i=1}^{m}\sum_{j=1}^{V} y_j^{(i)} \log \hat{y}_j^{(i)}$$

$$J_{batch} = -\frac{1}{m}\sum_{i=1}^{m} J^{(i)}$$

Predicted center word matrix

$$\hat{\mathbf{Y}} = \begin{bmatrix} \hat{\mathbf{y}}^{(1)} & \cdots & \hat{\mathbf{y}}^{(m)} \end{bmatrix}$$

Actual center word matrix

$$\mathbf{Y} = \begin{bmatrix} \mathbf{y}^{(1)} & \cdots & \mathbf{y}^{(m)} \end{bmatrix}$$

deeplearning.ai

Training a CBOW Model

Backpropagation and Gradient Descent

Minimizing the cost

$$J_{batch} = f(\mathbf{W_1}, \mathbf{W_2}, \mathbf{b_1}, \mathbf{b_2})$$

- Backpropagation: calculate partial derivatives of cost with respect to weights and biases

$$\frac{\partial J_{batch}}{\partial \mathbf{W_1}}, \frac{\partial J_{batch}}{\partial \mathbf{W_2}}, \frac{\partial J_{batch}}{\partial \mathbf{b_1}}, \frac{\partial J_{batch}}{\partial \mathbf{b_2}}$$

Minimizing the cost

- Backpropagation: calculate partial derivatives of cost with respect to weights and biases

$$\frac{\partial J_{batch}}{\partial \mathbf{W_1}}, \frac{\partial J_{batch}}{\partial \mathbf{W_2}}, \frac{\partial J_{batch}}{\partial \mathbf{b_1}}, \frac{\partial J_{batch}}{\partial \mathbf{b_2}}$$

- Gradient descent: update weights and biases

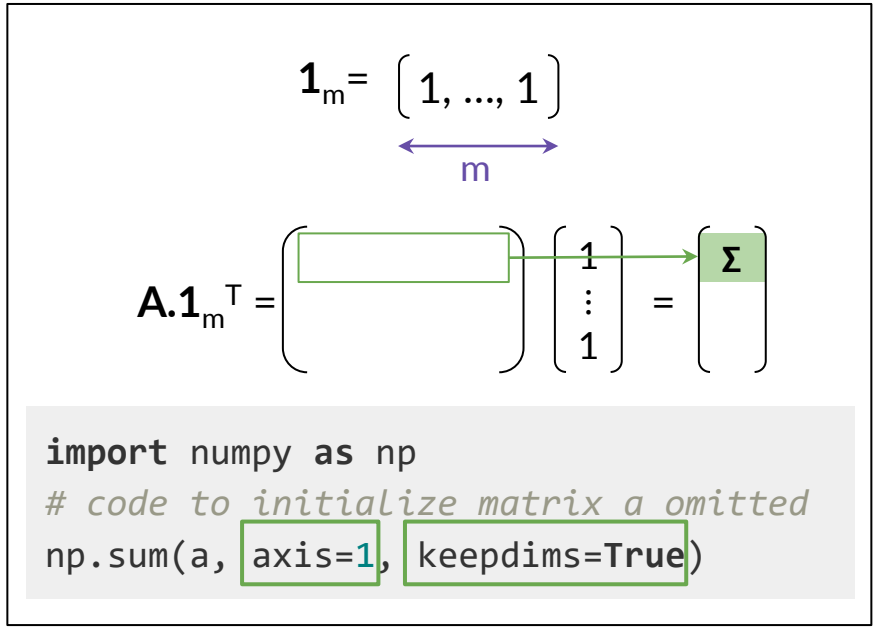
Backpropagation

$$\frac{\partial J_{batch}}{\partial \mathbf{W_1}}$$

$$\frac{\partial J_{batch}}{\partial \mathbf{W_2}}$$

$$\frac{\partial J_{batch}}{\partial \mathbf{b_1}}$$

$$\frac{\partial J_{batch}}{\partial \mathbf{b_2}}$$




```
import numpy as np
# code to initialize matrix a omitted
np.sum(a, axis=1, keepdims=True)
```

Gradient descent

Hyperparameter: learning rate α

$$\mathbf{W_1} := \mathbf{W_1} - \alpha \frac{\partial J_{batch}}{\partial \mathbf{W_1}}$$

$$\mathbf{W_2} := \mathbf{W_2} - \alpha \frac{\partial J_{batch}}{\partial \mathbf{W_2}}$$

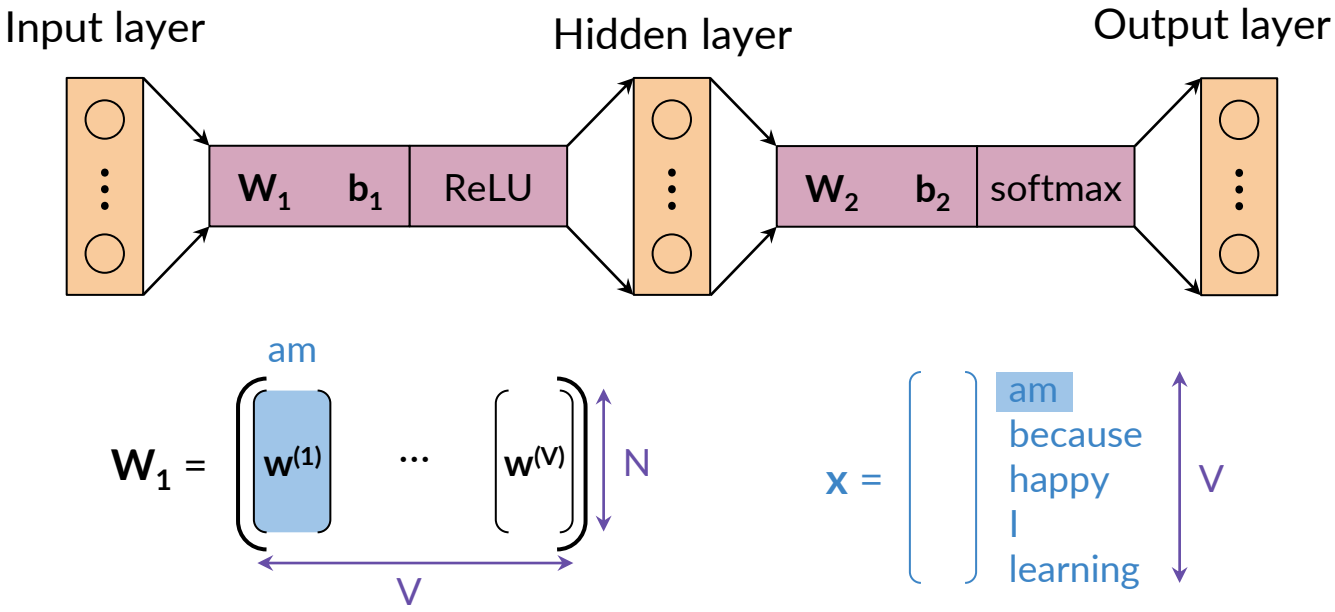
$$\mathbf{b_1} := \mathbf{b_1} - \alpha \frac{\partial J_{batch}}{\partial \mathbf{b_1}}$$

$$\mathbf{b_2} := \mathbf{b_2} - \alpha \frac{\partial J_{batch}}{\partial \mathbf{b_2}}$$

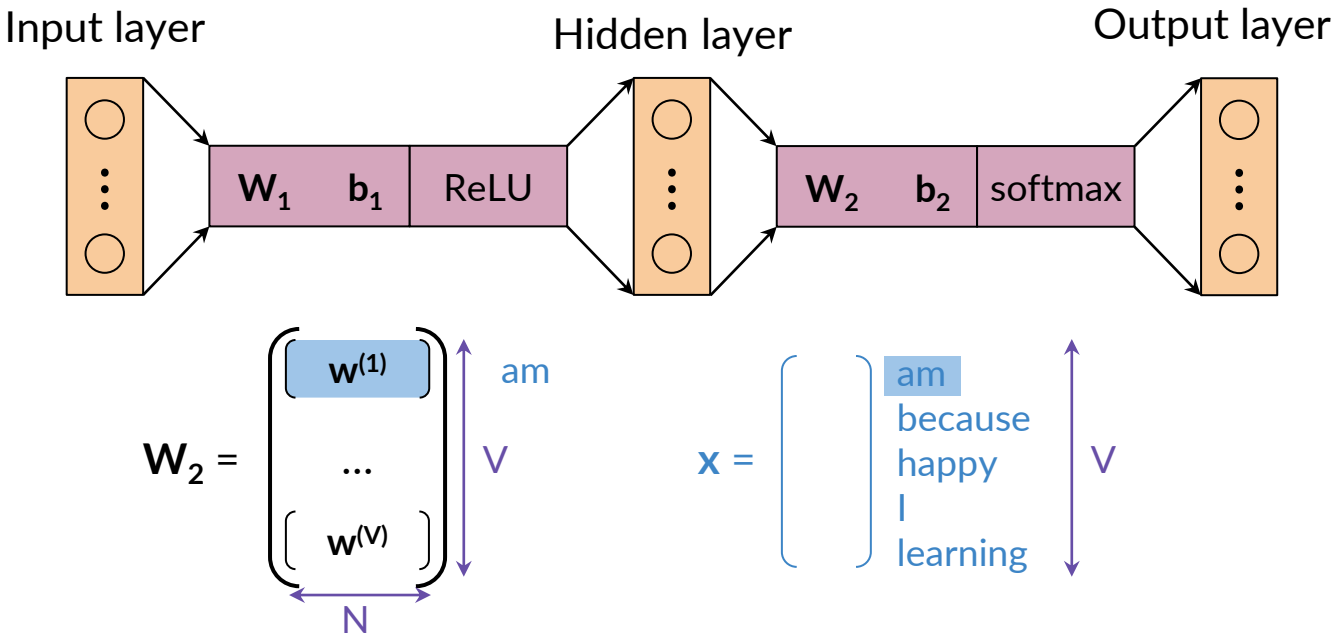
deeplearning.ai

Extracting Word Embedding Vectors

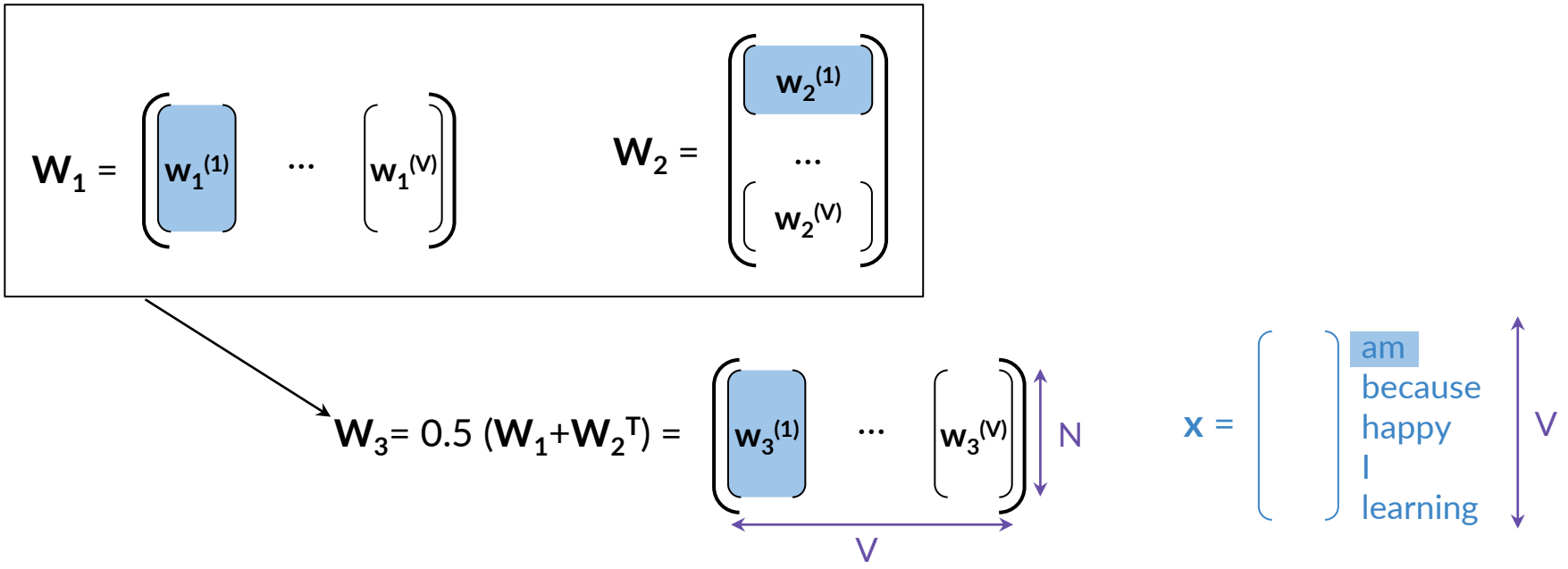
Extracting word embedding vectors: option 1



Extracting word embedding vectors: option 2



Extracting word embedding vectors: option 3



deeplearning.ai

Evaluating Word Embeddings

Intrinsic Evaluation

Intrinsic evaluation

Test relationships between words

- Analogies

  Semantic analogies

  “France” is to “Paris” as “Italy” is to <?>

  Syntactic analogies

  “seen” is to “saw” as “been” is to <?>

  ⚡ Ambiguity

  “wolf” is to “pack” as “bee” is to <?> → swarm? colony?

Intrinsic evaluation

Test relationships between words

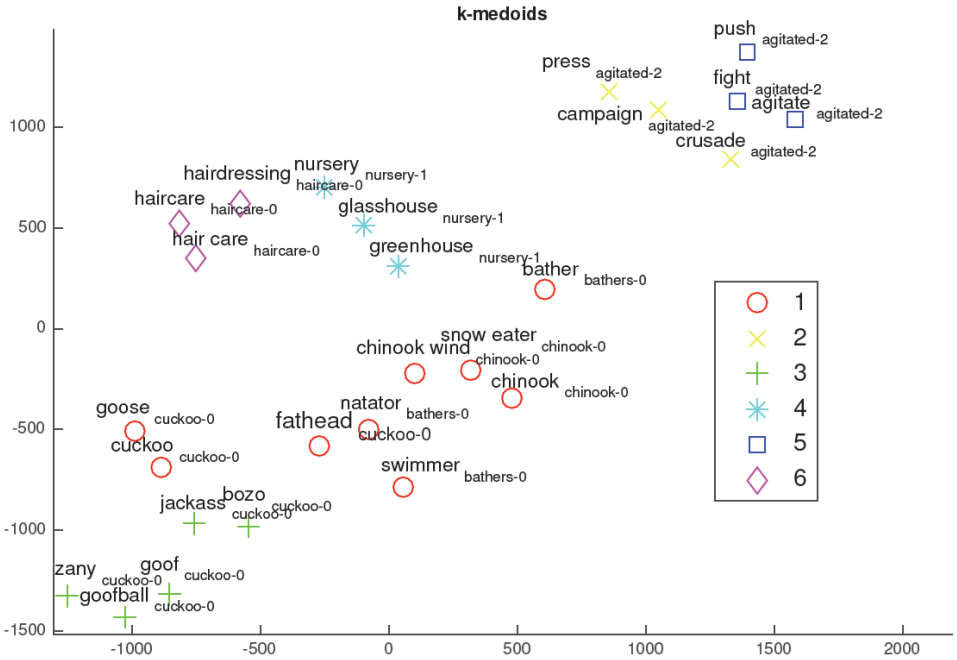
- Analogies

Relationship	Example 1	Example 2	Example 3
France - Paris	Italy: Rome	Japan: Tokyo	Florida: Tallahassee
big - bigger	small: larger	cold: colder	quick: quicker
Miami - Florida	Baltimore: Maryland	Dallas: Texas	Kona: Hawaii
Einstein - scientist	Messi: midfielder	Mozart: violinist	Picasso: painter
Sarkozy - France	Berlusconi: Italy	Merkel: Germany	Koizumi: Japan
copper - Cu	zinc: Zn	gold: Au	uranium: plutonium
Berlusconi - Silvio	Sarkozy: Nicolas	Putin: Medvedev	Obama: Barack
Microsoft - Windows	Google: Android	IBM: Linux	Apple: iPhone
Microsoft - Ballmer	Google: Yahoo	IBM: McNealy	Apple: Jobs
Japan - sushi	Germany: bratwurst	France: tapas	USA: pizza

Intrinsic evaluation

Test relationships between words

- Analogies
- Clustering

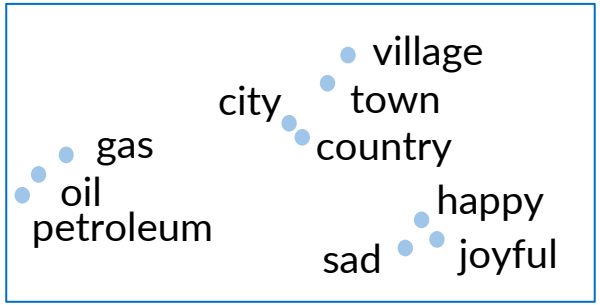


Source: Michael Zhai, Johnny Tan, and Jinho D. Choi. 2016. Intrinsic and extrinsic evaluations of word embeddings

Intrinsic evaluation

Test relationships between words

- Analogies
- Clustering
- Visualization

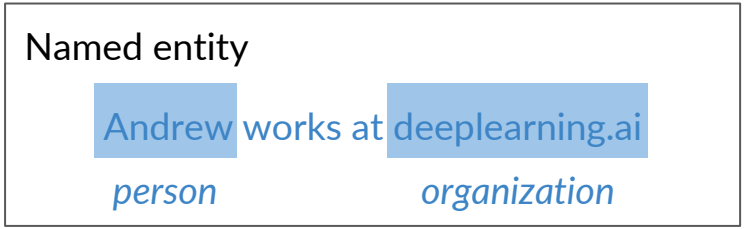


deeplearning.ai

Evaluating Word Embeddings

Extrinsic Evaluation

Extrinsic evaluation



Test word embeddings on external task

e.g. named entity recognition, parts-of-speech tagging

Extrinsic evaluation

Test word embeddings on external task
e.g. named entity recognition, parts-of-speech tagging

\+ Evaluates actual usefulness of embeddings

\- Time-consuming

\- More difficult to troubleshoot

deeplearning.ai

Conclusion

Recap and assignment

- Data preparation
- Word representations
- Continuous bag-of-words model
- Evaluation

Going further

- Advanced language modelling and word embeddings

- NLP and machine learning libraries

Keras

```
# from keras.layers.embeddings import Embedding
embed_layer = Embedding(10000, 400)
```

PyTorch

```
# import torch.nn as nn
embed_layer = nn.Embedding(10000, 400)
```