

# Copyright Notice

These slides are distributed under the Creative Commons License.

[DeepLearning.AI](#) makes these slides available for educational purposes. You may not use or distribute these slides for commercial purposes. You may make copies of these slides and use or distribute them for educational purposes as long as you cite [DeepLearning.AI](#) as the source of the slides.

For the rest of the details of the license, see <https://creativecommons.org/licenses/by-sa/2.0/legalcode>

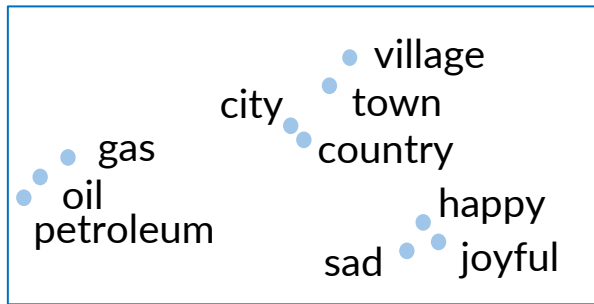


deeplearning.ai

# Overview

---

# Some basic applications of word embeddings



Semantic analogies  
and similarity



Sentiment analysis

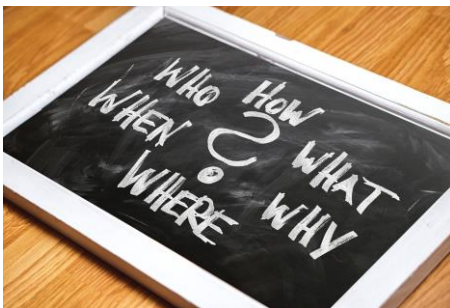


Classification of  
customer feedback

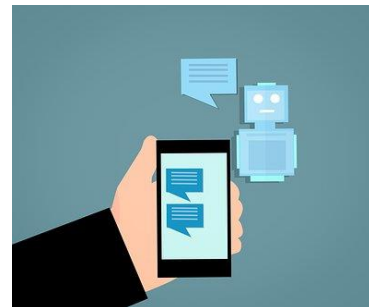
# Advanced applications of word embeddings



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   |

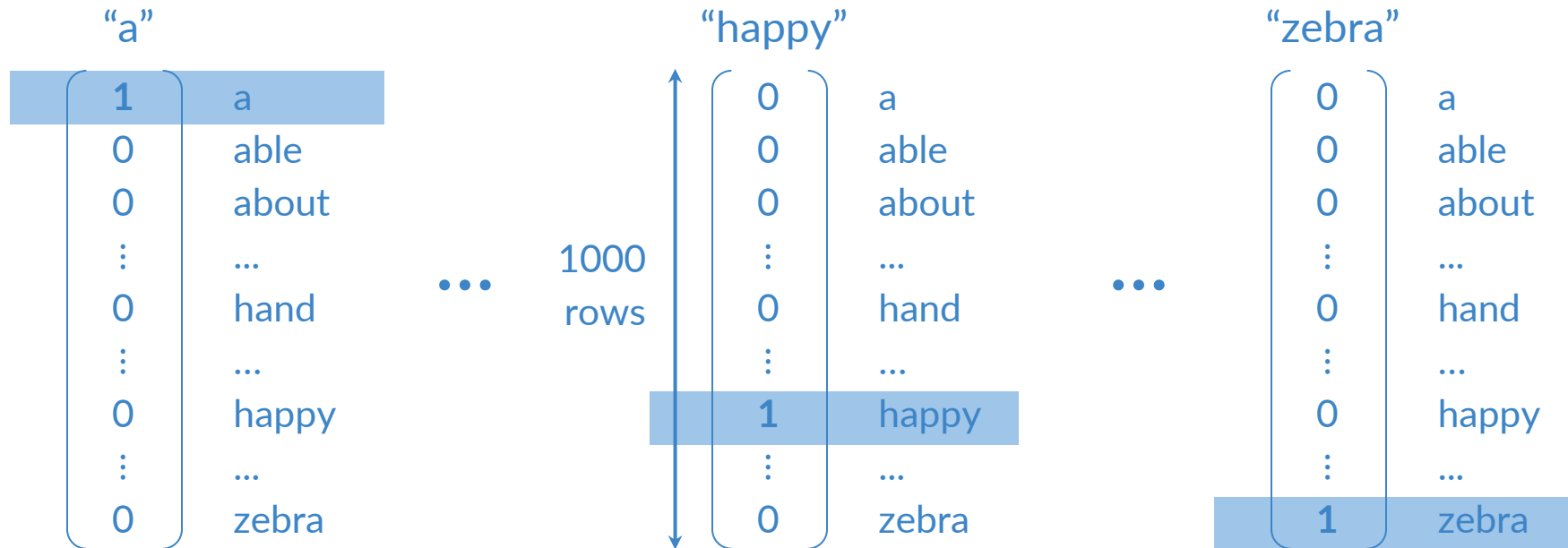


# Integers

- + Simple
- Ordering: little semantic sense

hand 615 < happy 621 < zebra 1000  
?! ?!

# One-hot vectors

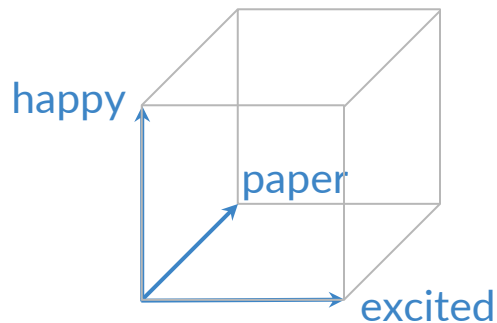
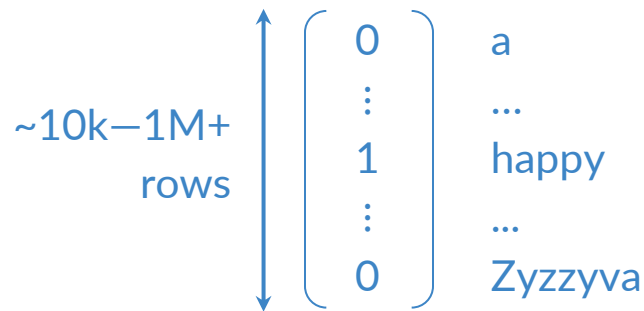


# One-hot vectors

| Word  | Number |   | "happy" |   |       |
|-------|--------|---|---------|---|-------|
| a     | 1      |   | 1       | 0 | a     |
| able  | 2      |   | 2       | 0 | able  |
| about | 3      |   | 3       | 0 | about |
| ...   | ...    |   | ...     | ⋮ | ...   |
| hand  | 615    |   | 615     | 0 | hand  |
| ...   | ...    |   | ...     | ⋮ | ...   |
| happy | 621    | ↔ | 621     | 1 | happy |
| ...   | ...    |   | ...     | ⋮ | ...   |
| zebra | 1000   |   | 1000    | 0 | zebra |

# One-hot vectors

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



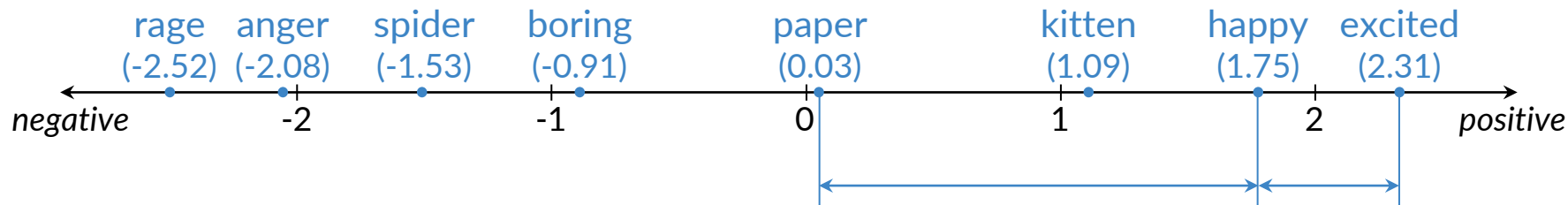
$$\begin{aligned} d(\text{paper}, \text{excited}) \\ &= d(\text{paper}, \text{happy}) \\ &= d(\text{excited}, \text{happy}) \end{aligned}$$



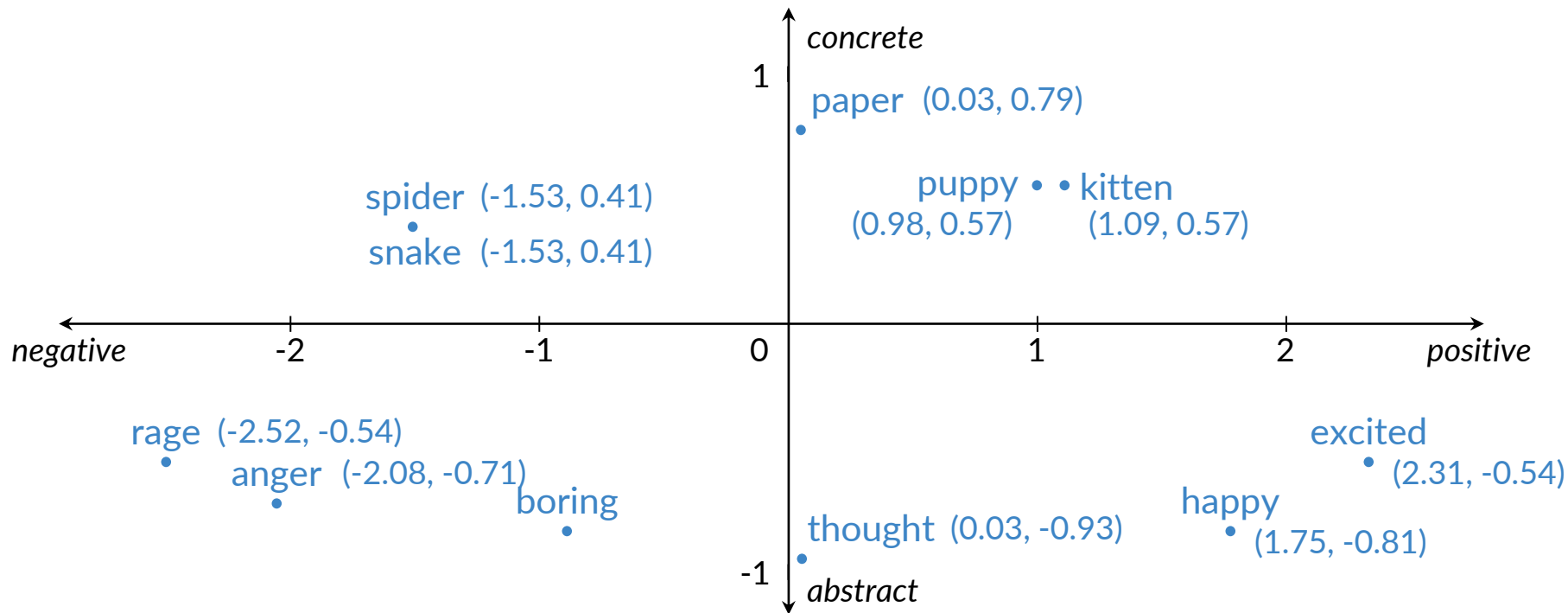
deeplearning.ai

# Word Embeddings

# Meaning as vectors



# Meaning as vectors



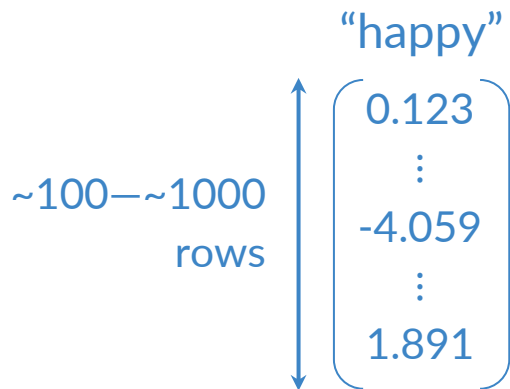
# Word embedding vectors

- + Low dimension
- + Embed meaning
  - o e.g. semantic distance

forest  $\approx$  tree      forest  $\not\approx$  ticket

- o e.g. analogies

Paris:France :: Rome:?





# Terminology

integers

**word vectors**

one-hot vectors

word embedding vectors

“word vectors”

word embeddings

# Summary

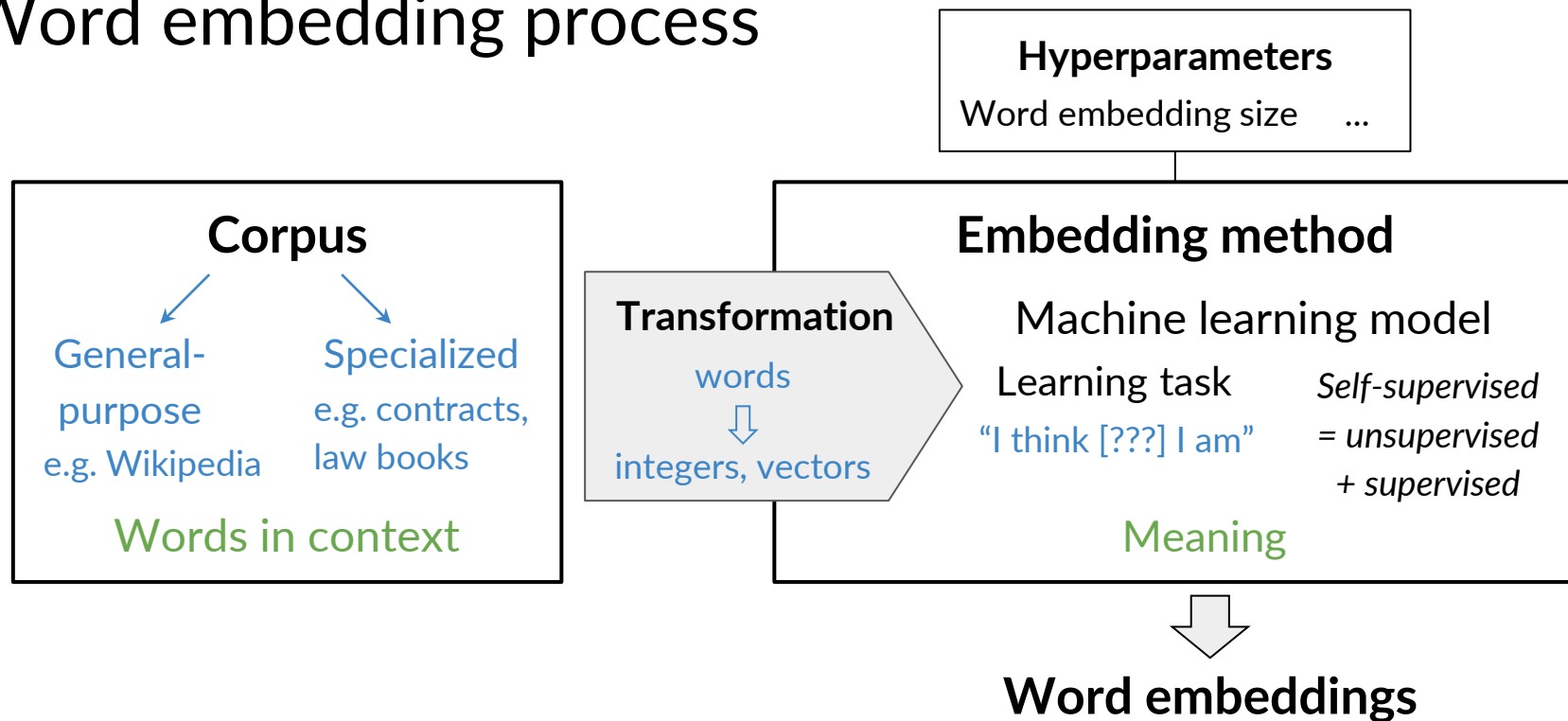
- Words as integers
- Words as vectors
  - One-hot vectors
  - Word embedding vectors
- Benefits of word embeddings for NLP



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

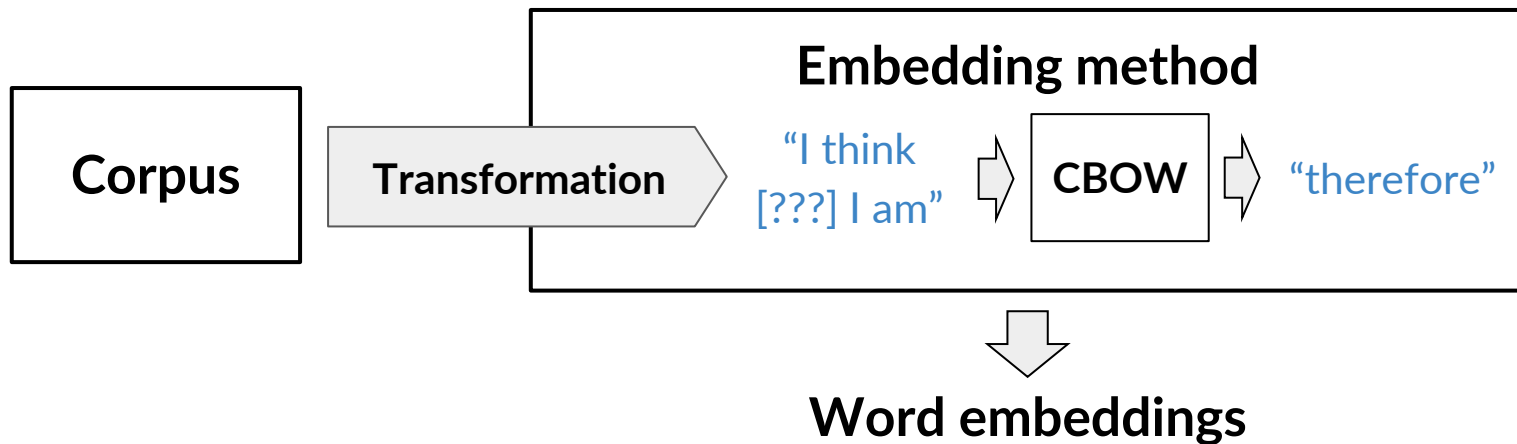


deeplearning.ai

# Continuous Bag-of-Words Model



# Continuous bag-of-words word embedding process



Corpus

Transformation

CBOW

# Center word prediction: rationale

The little ? is barking



*dog*  
*puppy*  
*hound*  
*terrier*

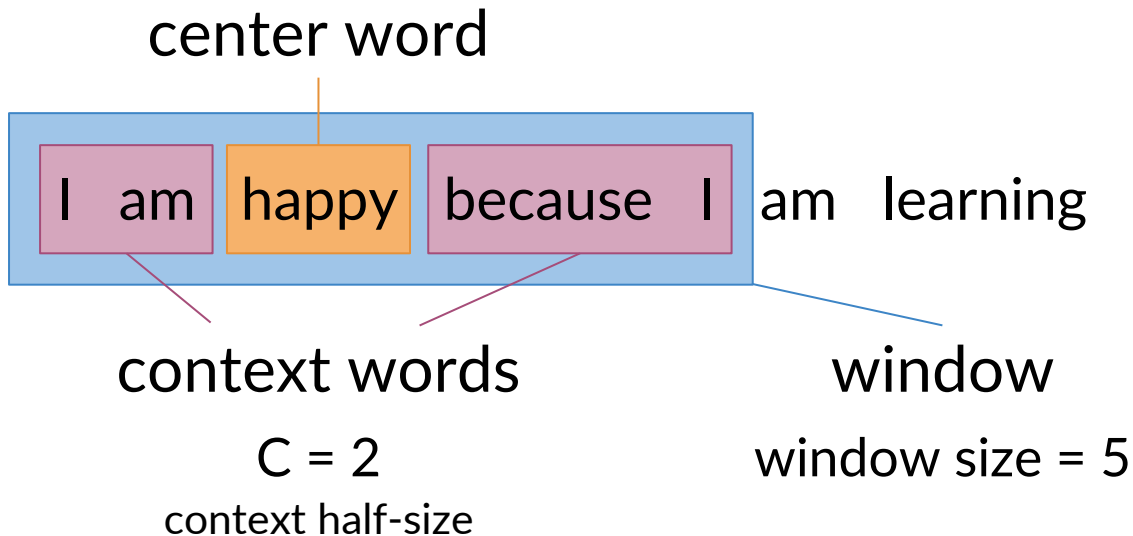
...

Corpus

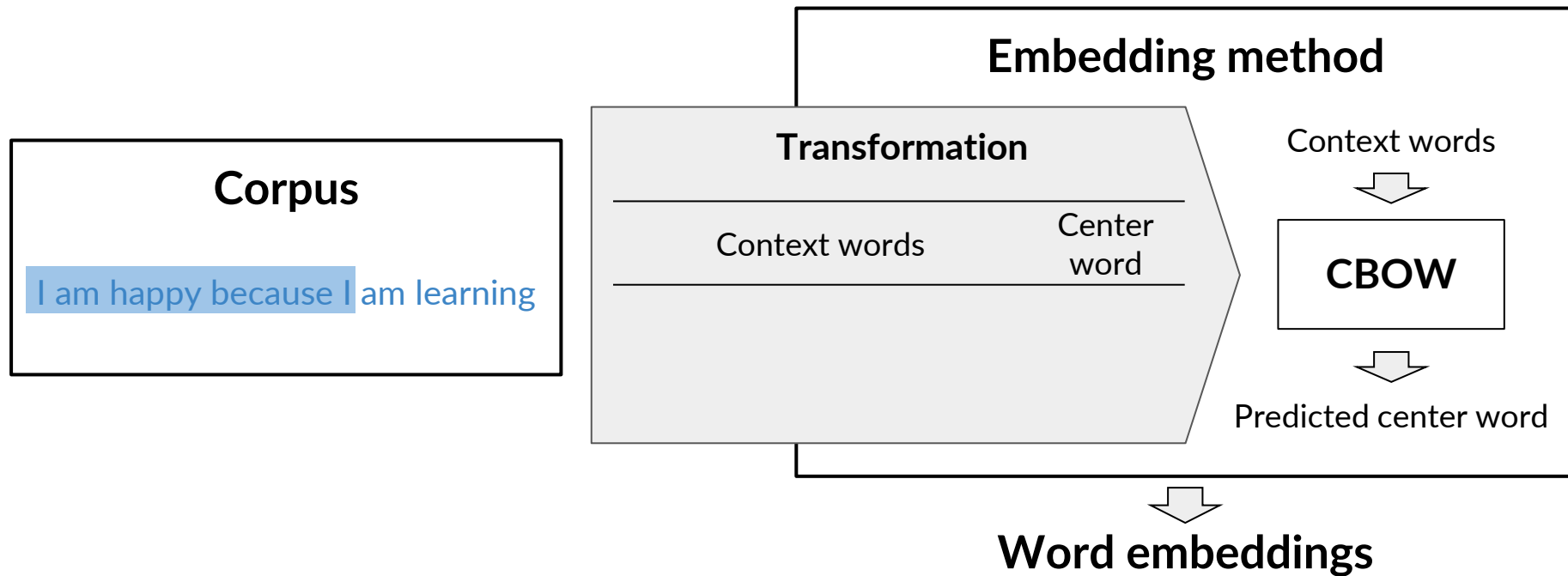
Transformation

CBOW

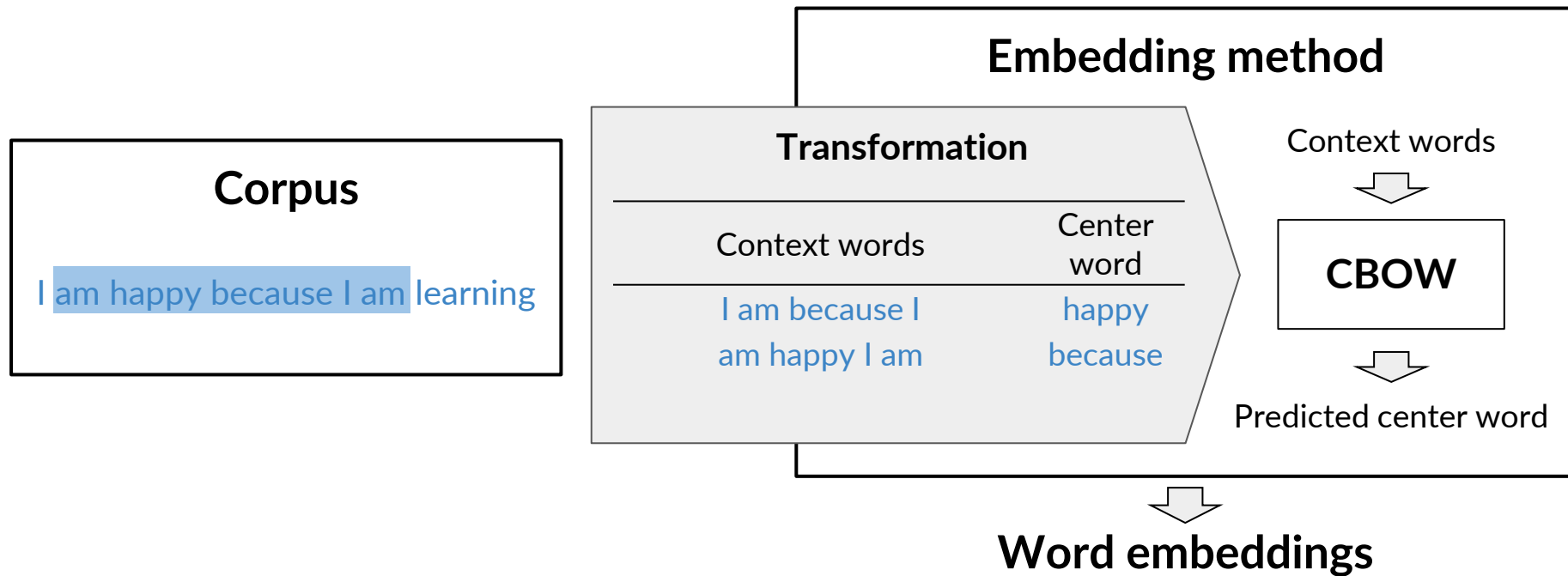
# Creating a training example



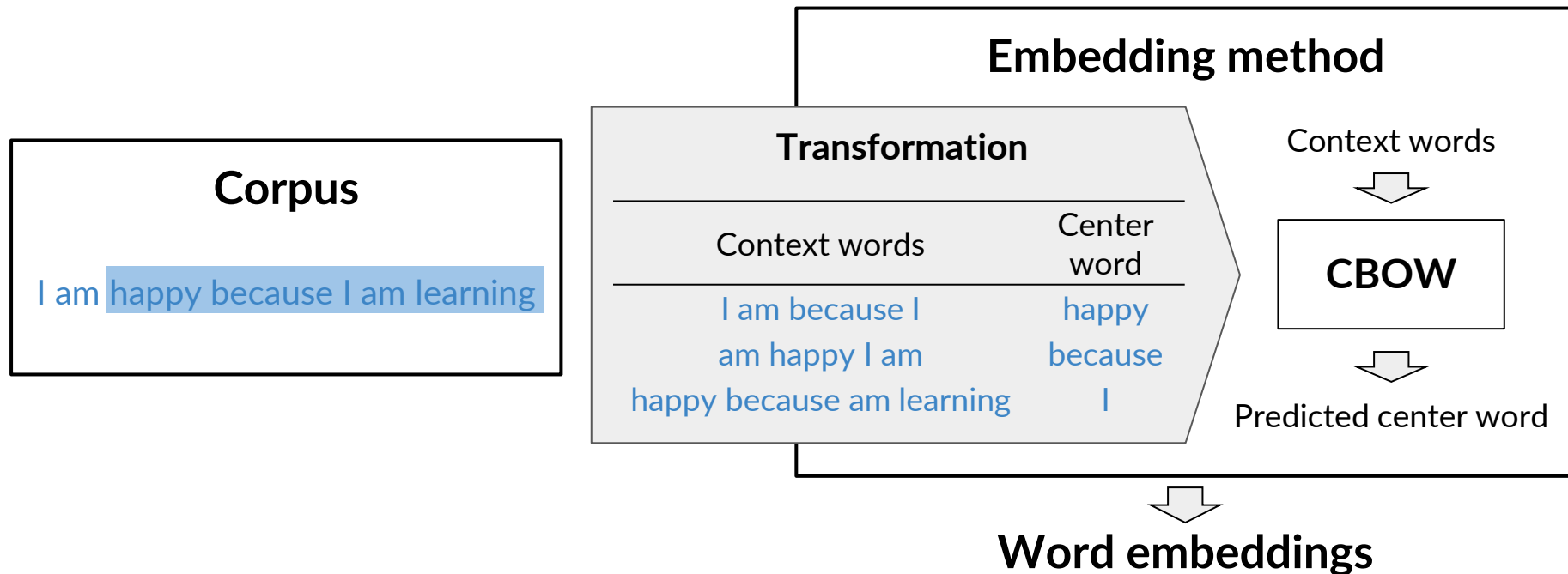
# From corpus to training



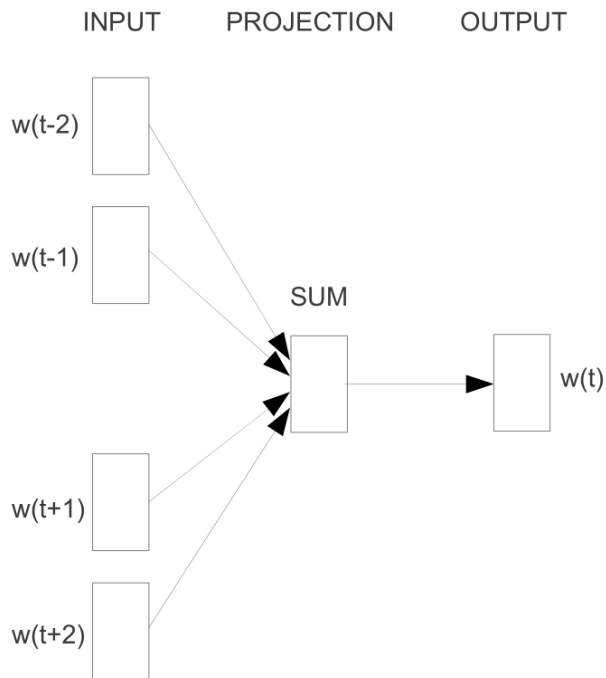
# From corpus to training



# 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

Who  "word embeddings" in 2020? I do!!!

emoji

punctuation

number

# 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
```

|   |    |       |         |   |    |          |
|---|----|-------|---------|---|----|----------|
| I | am | happy | because | I | am | learning |
| 0 | 1  | 2     | 3       | 4 | 5  | 6        |

# 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<br>vector | am                                | because                           | happy                             | I                                 | learning                          |
|-------------------|-----------------------------------|-----------------------------------|-----------------------------------|-----------------------------------|-----------------------------------|
| am                | $\begin{pmatrix} 1 \end{pmatrix}$ | $\begin{pmatrix} 0 \end{pmatrix}$ | $\begin{pmatrix} 0 \end{pmatrix}$ | $\begin{pmatrix} 0 \end{pmatrix}$ | $\begin{pmatrix} 0 \end{pmatrix}$ |
| because           | $\begin{pmatrix} 0 \end{pmatrix}$ | $\begin{pmatrix} 1 \end{pmatrix}$ | $\begin{pmatrix} 0 \end{pmatrix}$ | $\begin{pmatrix} 0 \end{pmatrix}$ | $\begin{pmatrix} 0 \end{pmatrix}$ |
| happy             | $\begin{pmatrix} 0 \end{pmatrix}$ | $\begin{pmatrix} 0 \end{pmatrix}$ | $\begin{pmatrix} 1 \end{pmatrix}$ | $\begin{pmatrix} 0 \end{pmatrix}$ | $\begin{pmatrix} 0 \end{pmatrix}$ |
| I                 | $\begin{pmatrix} 0 \end{pmatrix}$ | $\begin{pmatrix} 0 \end{pmatrix}$ | $\begin{pmatrix} 0 \end{pmatrix}$ | $\begin{pmatrix} 1 \end{pmatrix}$ | $\begin{pmatrix} 0 \end{pmatrix}$ |
| learning          | $\begin{pmatrix} 0 \end{pmatrix}$ | $\begin{pmatrix} 0 \end{pmatrix}$ | $\begin{pmatrix} 0 \end{pmatrix}$ | $\begin{pmatrix} 0 \end{pmatrix}$ | $\begin{pmatrix} 1 \end{pmatrix}$ |



# Transforming context words into vectors

Average of individual one-hot vectors

$$\left( \begin{array}{c} \text{I} \\ \text{am} \\ \text{because} \\ \text{happy} \\ \text{I} \\ \text{learning} \end{array} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} + \begin{array}{c} \text{am} \\ \\ \\ \\ \\ \end{array} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} + \begin{array}{c} \text{because} \\ \\ \\ \\ \\ \end{array} \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} + \begin{array}{c} \text{I} \\ \\ \\ \\ \\ \end{array} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \right) / 4 = \begin{array}{c} \text{I am because I} \\ \\ \\ \\ \end{array} \begin{bmatrix} 0.25 \\ 0.25 \\ 0 \\ 0.5 \\ 0 \end{bmatrix}$$

# Final prepared training set

| Context words         | Context words vector      | Center word  | Center word vector |
|-----------------------|---------------------------|--------------|--------------------|
| <i>I am because I</i> | $[0.25; 0.25; 0; 0.5; 0]$ | <i>happy</i> | $[0; 0; 1; 0; 0]$  |



deeplearning.ai

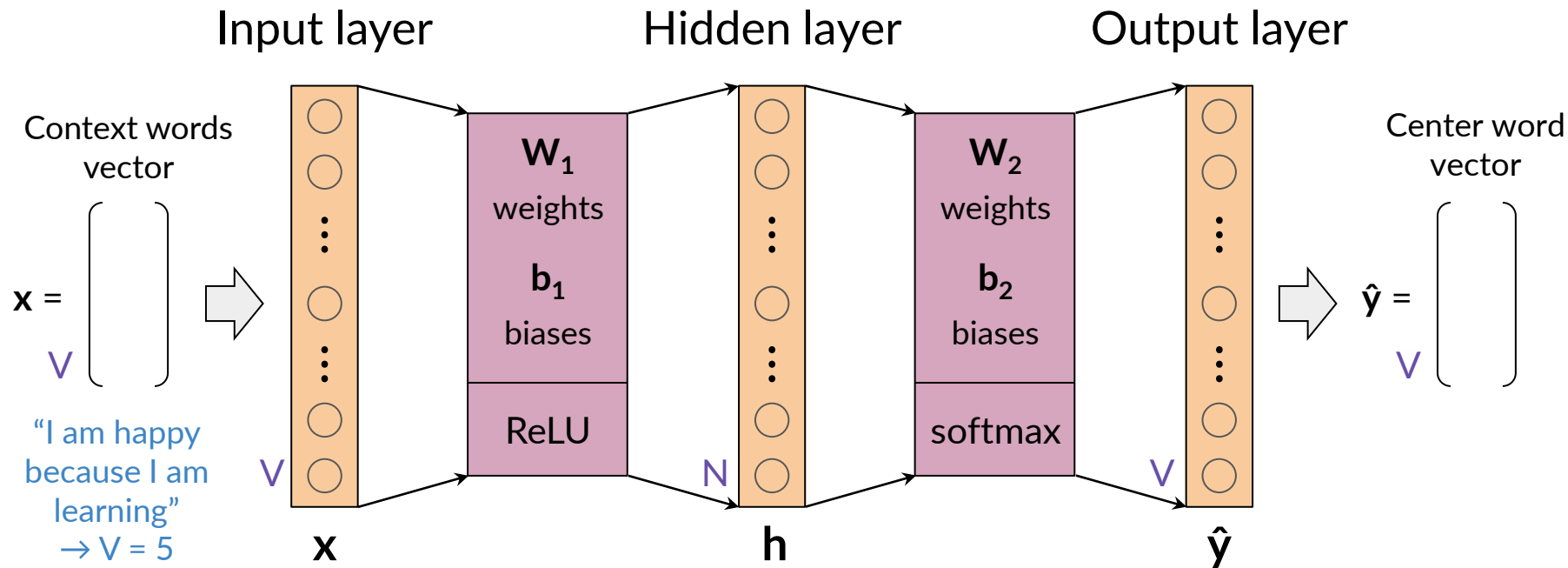
# Architecture of the CBOW Model

---

# Architecture of the CBOW model

## Hyperparameters

$N$ : Word embedding size ...





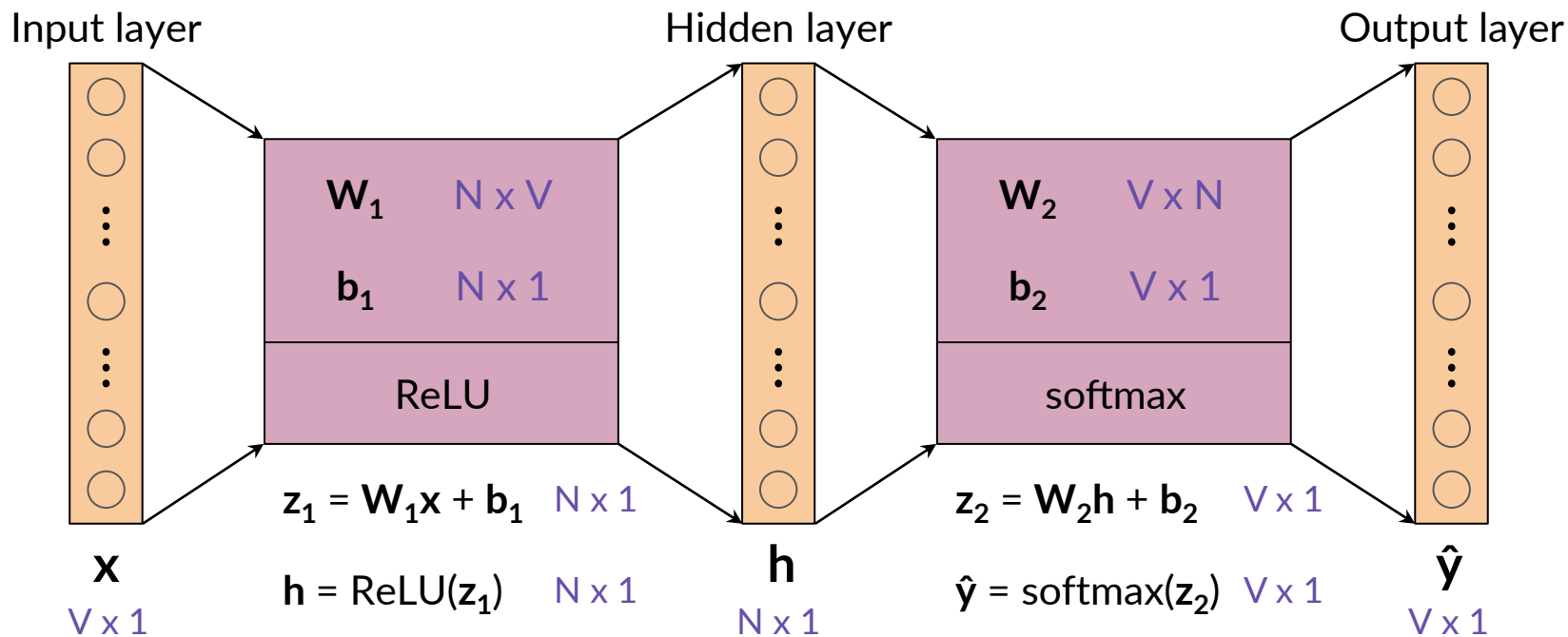
deeplearning.ai

# Architecture of the CBOW Model:

---

Dimensions

# Dimensions (single input)



# Dimensions (single input)

Column vectors

$$\mathbf{z}_1 = \mathbf{W}_1 \mathbf{x} + \mathbf{b}_1$$
$$\mathbf{z}_1 = \begin{pmatrix} \phantom{0} \\ \phantom{0} \end{pmatrix}_{N \times 1} \quad \mathbf{W}_1 = \begin{pmatrix} \phantom{0} & \phantom{0} \\ \phantom{0} & \phantom{0} \end{pmatrix}_{N \times V} \quad \mathbf{x} = \begin{pmatrix} \phantom{0} \\ \phantom{0} \end{pmatrix}_{V \times 1} \quad \mathbf{b}_1 = \begin{pmatrix} \phantom{0} \\ \phantom{0} \end{pmatrix}_{N \times 1}$$

Row vectors

$$\mathbf{z}_1 = \mathbf{x} \mathbf{W}_1^T + \mathbf{b}_1$$
$$\mathbf{b}_1 = \begin{pmatrix} \phantom{0} & \phantom{0} \end{pmatrix}_{1 \times N} \quad \mathbf{W}_1 = \begin{pmatrix} \phantom{0} & \phantom{0} \\ \phantom{0} & \phantom{0} \end{pmatrix}_{N \times V} \quad \mathbf{b}_1 = \begin{pmatrix} \phantom{0} & \phantom{0} \end{pmatrix}_{1 \times N}$$
$$\mathbf{x} = \begin{pmatrix} \phantom{0} & \phantom{0} \end{pmatrix}_{1 \times V}$$



deeplearning.ai

# Architecture of the CBOW Model:

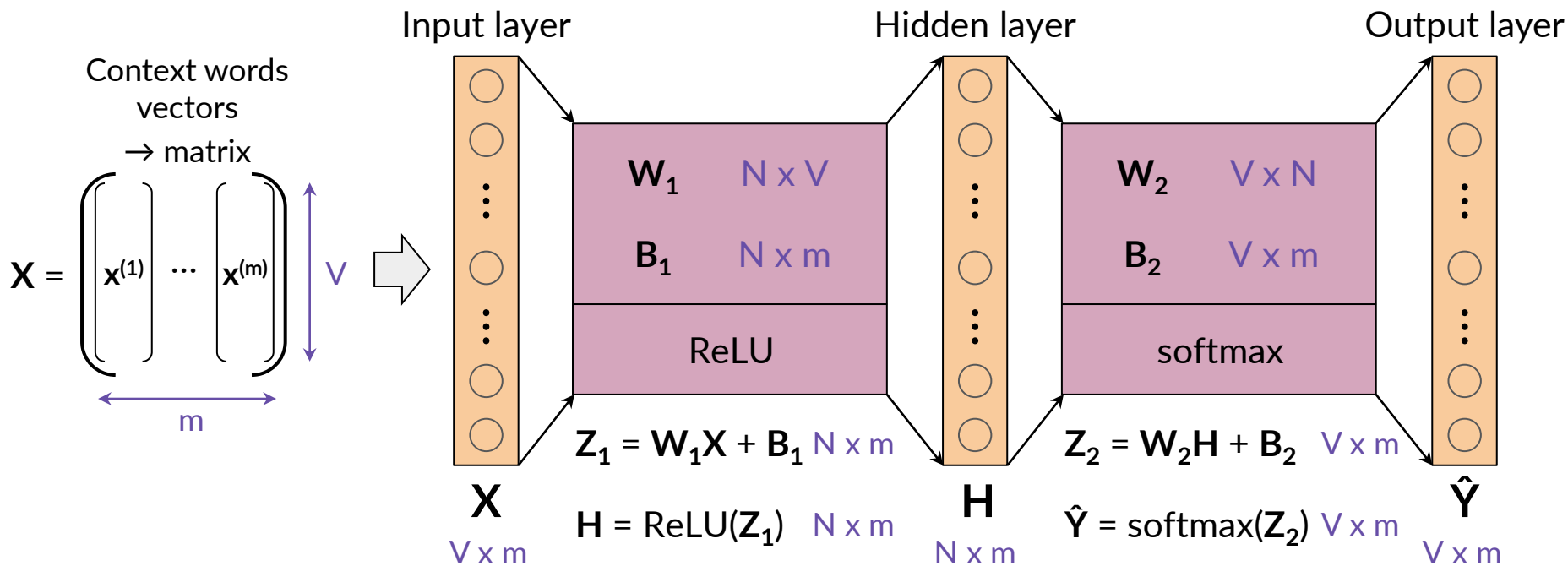
---

Dimensions 2

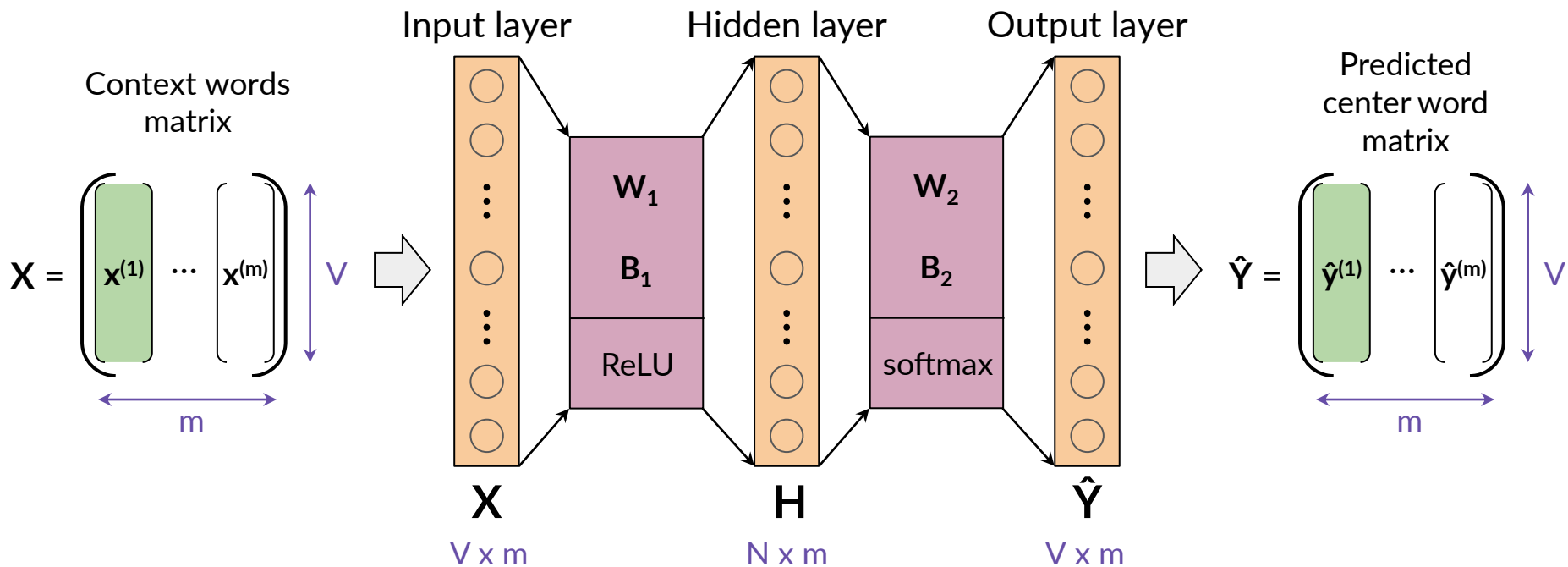


# Dimensions (batch input)

$$\begin{bmatrix} \mathbf{b}_1 \end{bmatrix} \rightarrow \mathbf{B}_1 = \begin{pmatrix} \begin{bmatrix} \mathbf{b}_1 \end{bmatrix} & \dots & \begin{bmatrix} \mathbf{b}_1 \end{bmatrix} \end{pmatrix} \begin{matrix} \xleftrightarrow{m} \\ \updownarrow N \text{ broadcasting} \end{matrix}$$



# Dimensions (batch input)





deeplearning.ai

# Architecture of the CBOW Model

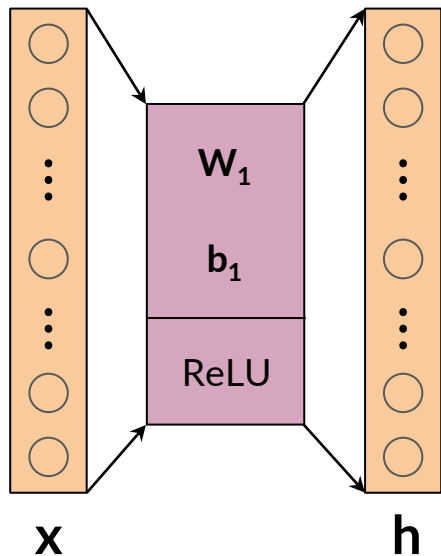
---

Activation Functions

# Rectified Linear Unit (ReLU)

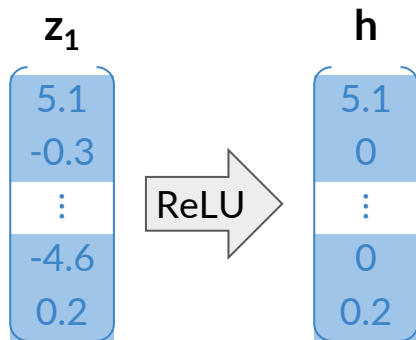
Input layer

Hidden layer

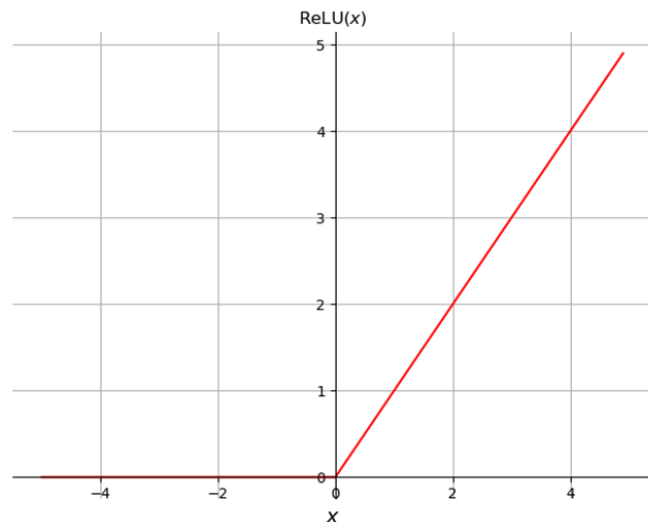


$$z_1 = W_1 x + b_1$$

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



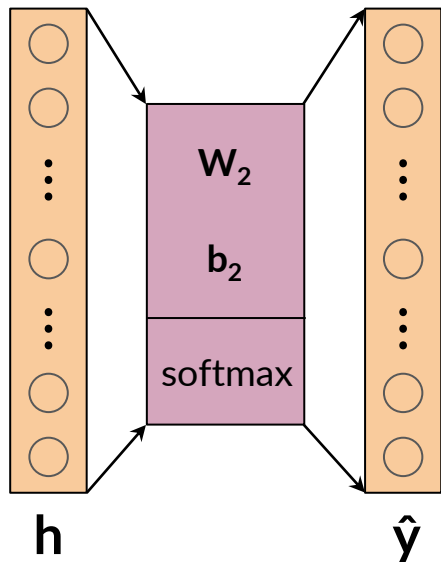
$$\text{ReLU}(x) = \max(0, x)$$



# Softmax

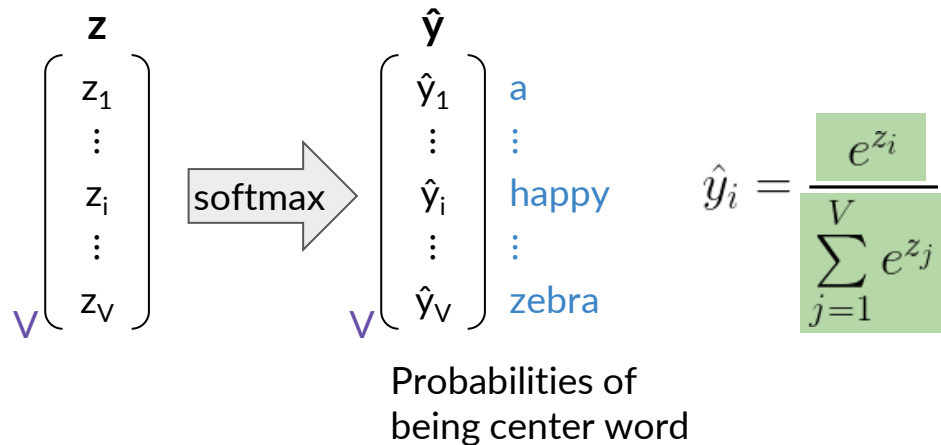
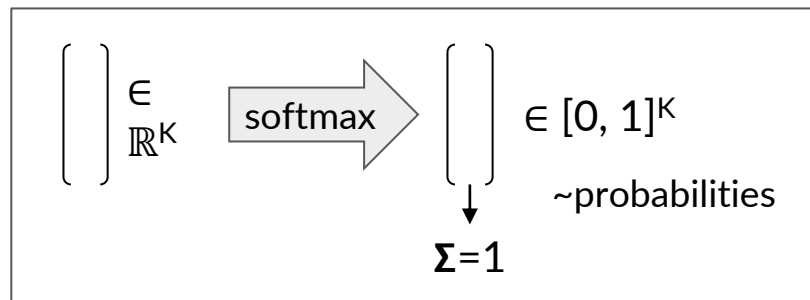
Hidden layer

Output layer

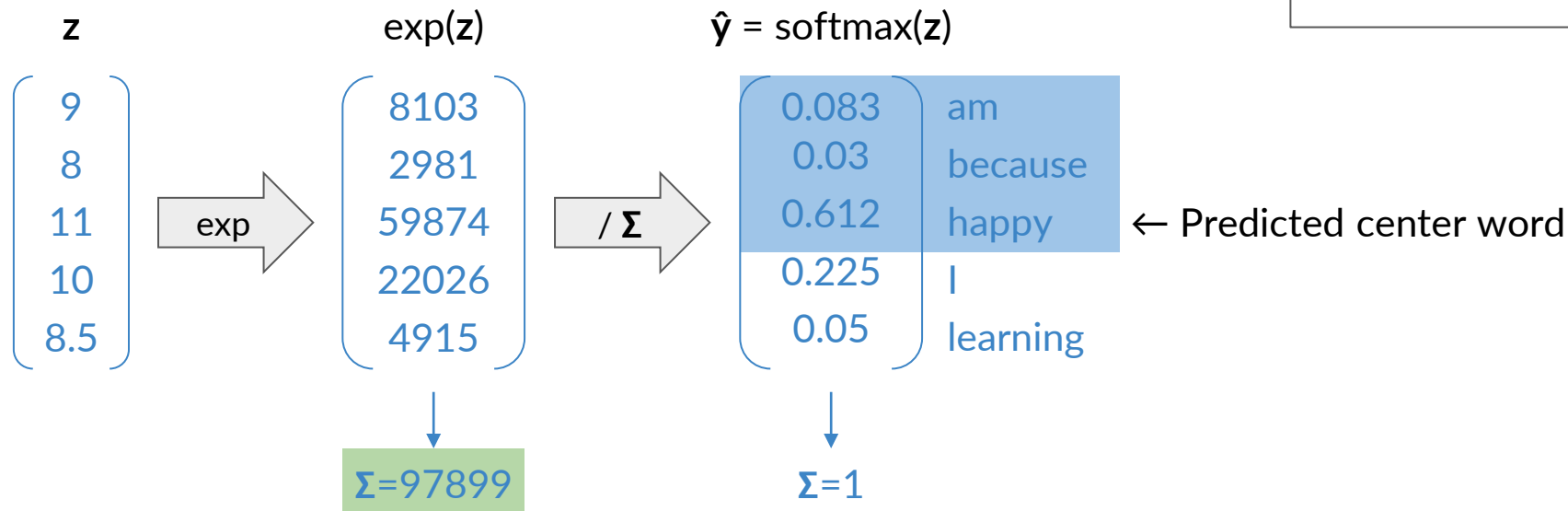


$$\mathbf{z} = \mathbf{W}_2 \mathbf{h} + \mathbf{b}_2$$

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



# Softmax: example





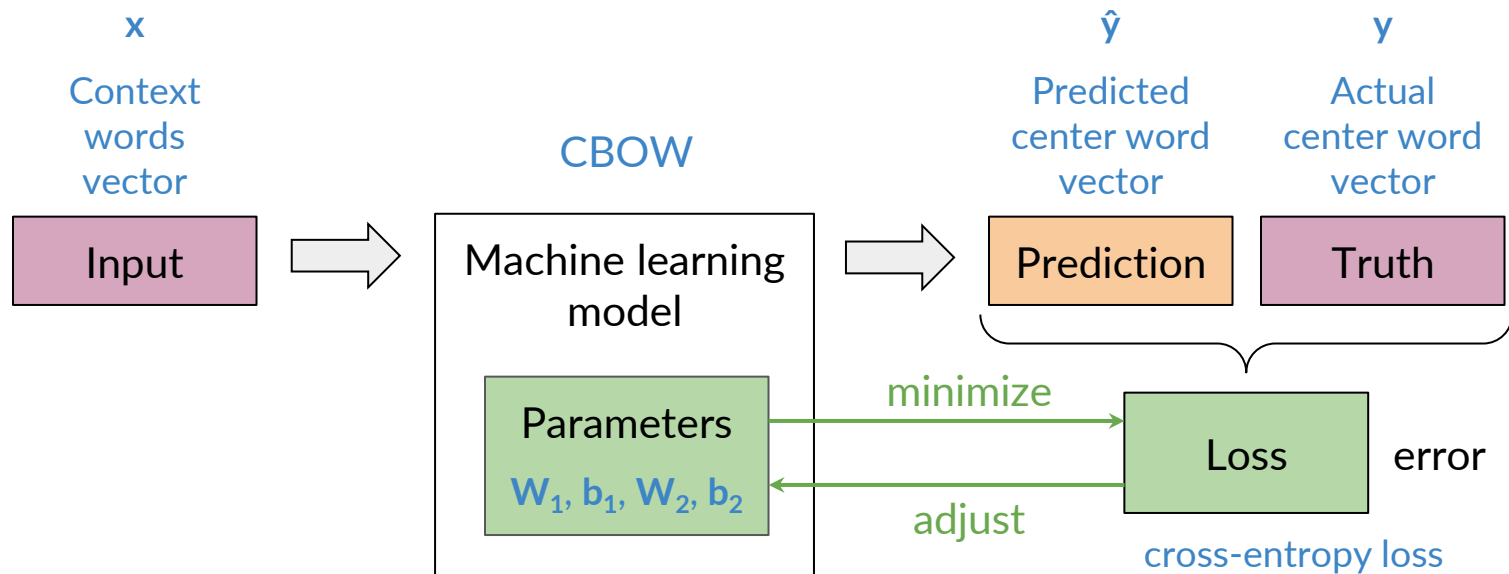
deeplearning.ai

# Training a CBOW Model

---

Cost Function

# Loss



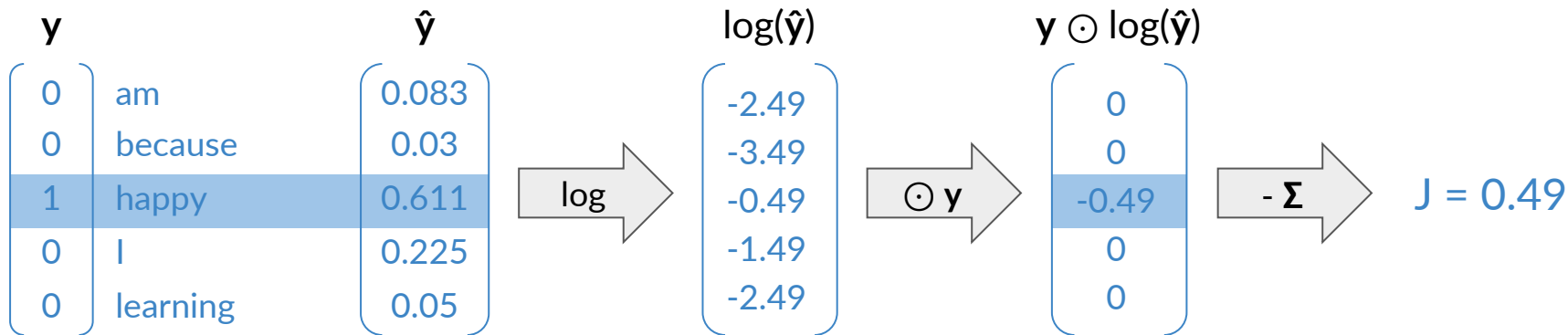


# Cross-entropy loss

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

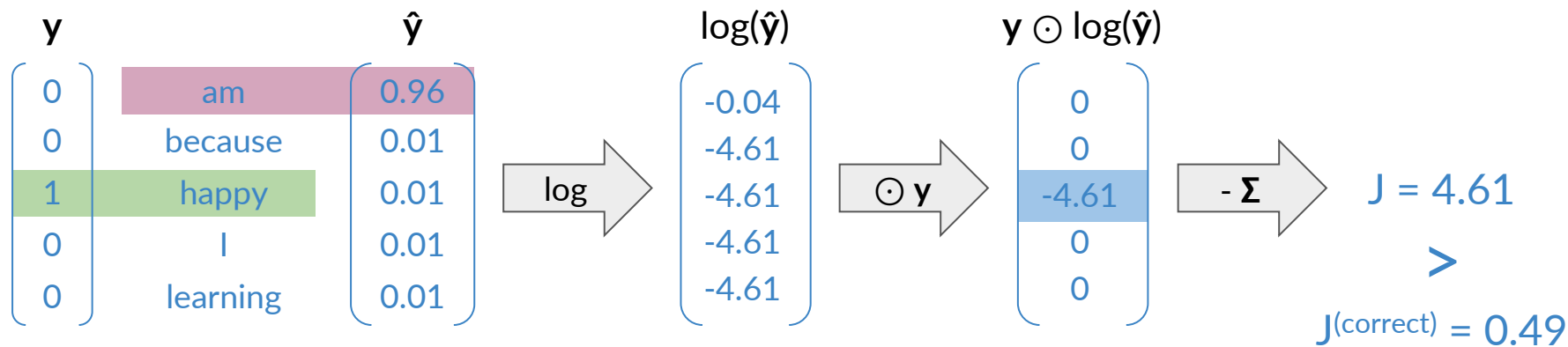
|   |   |
|---|---|
| Actual  | Predicted   |
| $\mathbf{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_V \end{bmatrix}$ | $\hat{\mathbf{y}} = \begin{bmatrix} \hat{y}_1 \\ \vdots \\ \hat{y}_V \end{bmatrix}$ |

I am happy because I am learning



# Cross-entropy loss

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

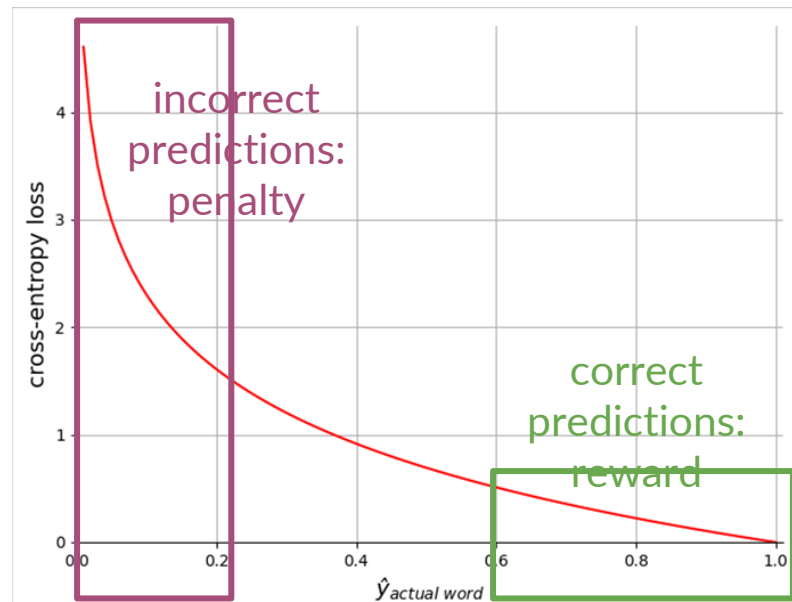


# Cross-entropy loss

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

| $y$ |          | $\hat{y}$ |                        |
|-----|----------|-----------|------------------------|
| 0   | am       | 0.96      |                        |
| 0   | because  | 0.01      |                        |
| 1   | happy    | 0.01      | $\rightarrow J = 4.61$ |
| 0   | I        | 0.01      |                        |
| 0   | learning | 0.01      |                        |

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





deeplearning.ai

# Training a CBOW Model

---

Forward Propagation

# Training process

- Forward propagation
- Cost
- Backpropagation and gradient descent

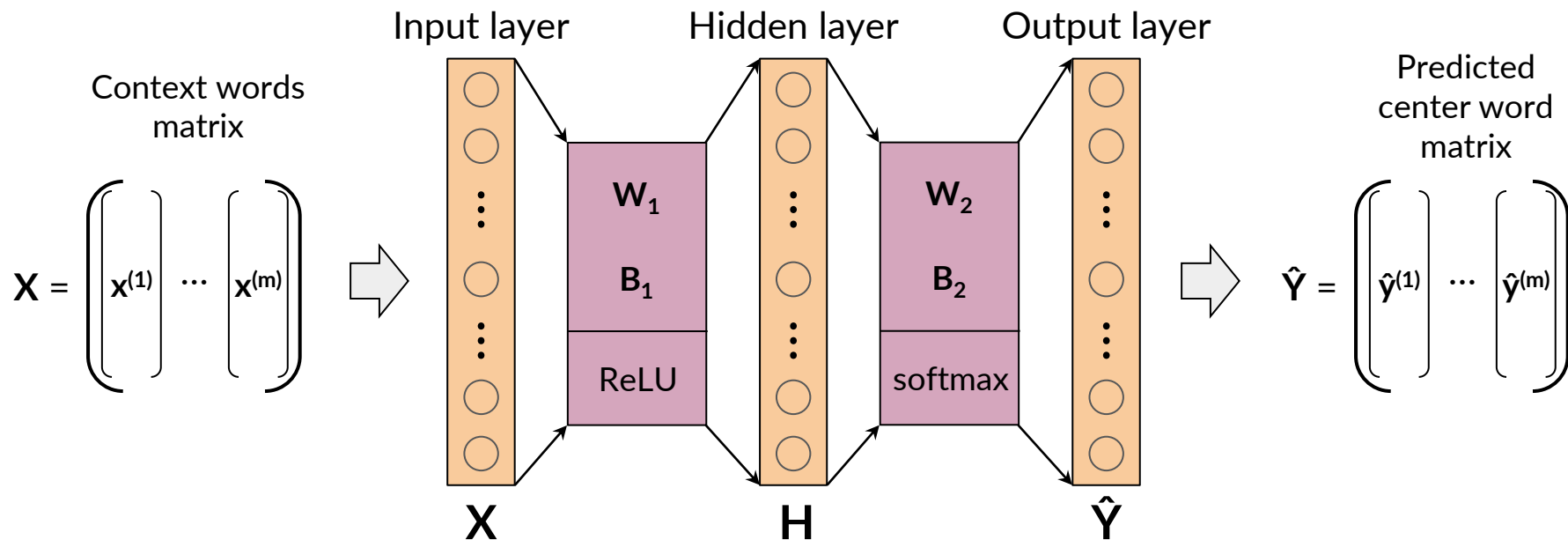
# Forward propagation

$$\mathbf{Z}_1 = \mathbf{W}_1\mathbf{X} + \mathbf{B}_1$$

$$\mathbf{Z}_2 = \mathbf{W}_2\mathbf{H} + \mathbf{B}_2$$

$$\mathbf{H} = \text{ReLU}(\mathbf{Z}_1)$$

$$\hat{\mathbf{Y}} = \text{softmax}(\mathbf{Z}_2)$$



# 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

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

Actual center  
word matrix

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



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

$$\frac{\partial J_{batch}}{\partial \mathbf{W}_1} = \frac{1}{m} (\mathbf{W}_2^\top (\hat{\mathbf{Y}} - \mathbf{Y}) \cdot \text{step}(\mathbf{Z}_1)) \mathbf{X}^\top$$

$$\frac{\partial J_{batch}}{\partial \mathbf{W}_2} = \frac{1}{m} (\hat{\mathbf{Y}} - \mathbf{Y}) \mathbf{H}^\top$$

$$\frac{\partial J_{batch}}{\partial \mathbf{b}_1} = \frac{1}{m} (\mathbf{W}_2^\top (\hat{\mathbf{Y}} - \mathbf{Y}) \cdot \text{step}(\mathbf{Z}_1)) \mathbf{1}_m^\top$$

$$\frac{\partial J_{batch}}{\partial \mathbf{b}_2} = \frac{1}{m} (\hat{\mathbf{Y}} - \mathbf{Y}) \mathbf{1}_m^\top$$

$\mathbf{1}_m = \begin{bmatrix} 1, \dots, 1 \end{bmatrix}$   
↔  
m

$\mathbf{A} \cdot \mathbf{1}_m^\top = \begin{bmatrix} \phantom{\vdots} \end{bmatrix} \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} = \begin{bmatrix} \Sigma \end{bmatrix}$

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

# Gradient descent

Hyperparameter: learning rate  $\alpha$

$$\mathbf{W}_1 := \mathbf{W}_1 - \alpha \frac{\partial J_{batch}}{\partial \mathbf{W}_1}$$

$$\mathbf{W}_2 := \mathbf{W}_2 - \alpha \frac{\partial J_{batch}}{\partial \mathbf{W}_2}$$

$$\mathbf{b}_1 := \mathbf{b}_1 - \alpha \frac{\partial J_{batch}}{\partial \mathbf{b}_1}$$

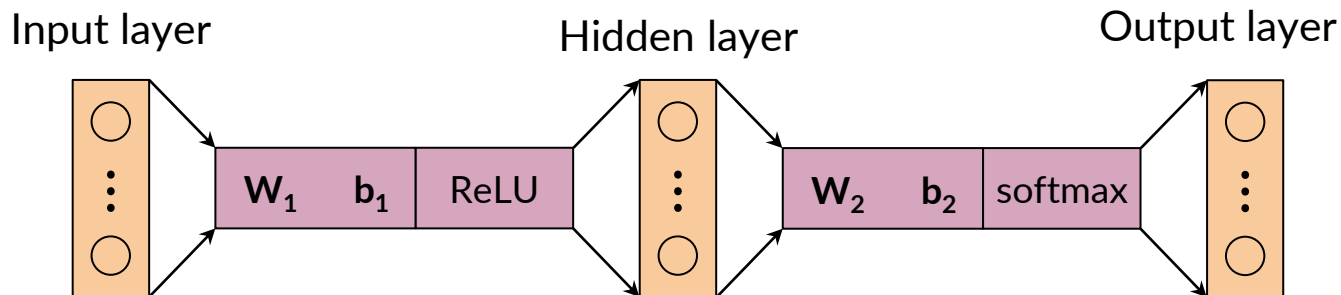
$$\mathbf{b}_2 := \mathbf{b}_2 - \alpha \frac{\partial J_{batch}}{\partial \mathbf{b}_2}$$



deeplearning.ai

# Extracting Word Embedding Vectors

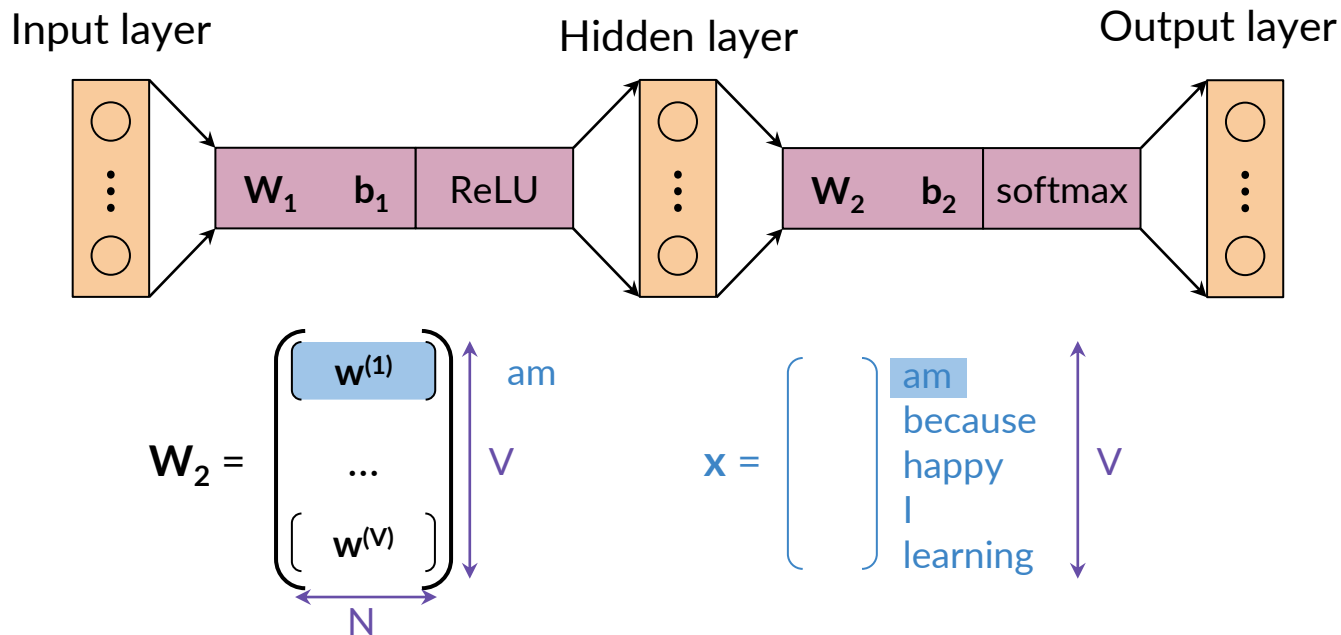
# Extracting word embedding vectors: option 1



$$W_1 = \begin{pmatrix} \text{am} \\ \vdots \\ w^{(1)} \\ \vdots \\ w^{(V)} \end{pmatrix} \quad \text{...} \quad \begin{pmatrix} \vdots \\ w^{(V)} \end{pmatrix} \quad \begin{matrix} \updownarrow N \\ \leftarrow V \end{matrix}$$

$$x = \begin{pmatrix} \text{am} \\ \text{because} \\ \text{happy} \\ \vdots \\ \text{learning} \end{pmatrix} \quad \begin{matrix} \updownarrow V \end{matrix}$$

# Extracting word embedding vectors: option 2



# Extracting word embedding vectors: option 3

$$\mathbf{W}_1 = \begin{bmatrix} \boxed{w_1^{(1)}} & \dots & w_1^{(V)} \end{bmatrix} \quad \mathbf{W}_2 = \begin{bmatrix} \boxed{w_2^{(1)}} \\ \dots \\ w_2^{(V)} \end{bmatrix}$$

$\mathbf{W}_3 = 0.5 (\mathbf{W}_1 + \mathbf{W}_2^T) = \begin{bmatrix} \boxed{w_3^{(1)}} & \dots & w_3^{(V)} \end{bmatrix}$

Dimensions:  $V$  (horizontal),  $N$  (vertical)

$$\mathbf{x} = \begin{bmatrix} \boxed{\text{am}} \\ \text{because} \\ \text{happy} \\ \text{I} \\ \text{learning} \end{bmatrix}$$

Dimensions:  $V$  (vertical)





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

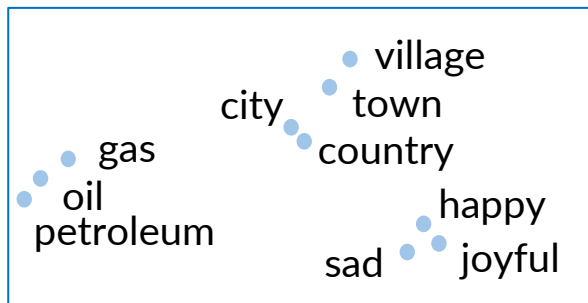
| Relationship         | Example 1           | Example 2         | Example 3            |
|----------------------|---------------------|-------------------|----------------------|
| France - Paris       | Italy: Rome         | Japan: Tokyo      | Florida: Tallahassee |
| big - bigger         | small: 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
- 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

Named entity

Andrew works at deeplearning.ai

*person*

*organization*

# 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)
```