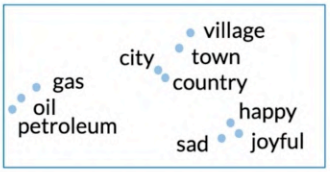
Some basic applications of word embeddings



Semantic analogies and similarity

Sentiment analysis

Classification of customer feedback

Advanced applications of word embeddings

Machine translation

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

Integers

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

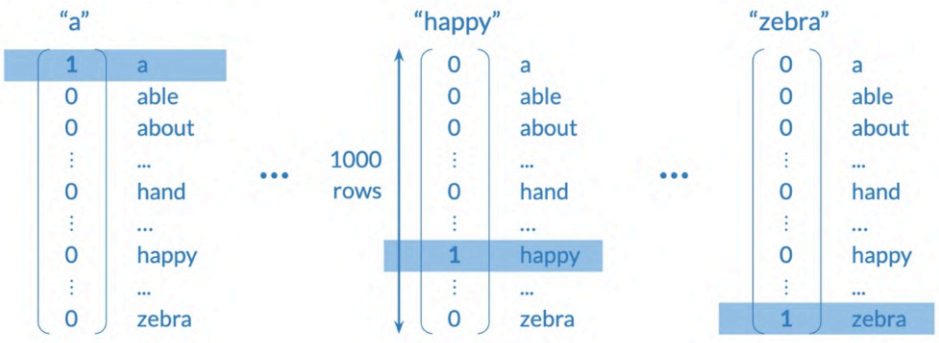
Integers

+ Simple

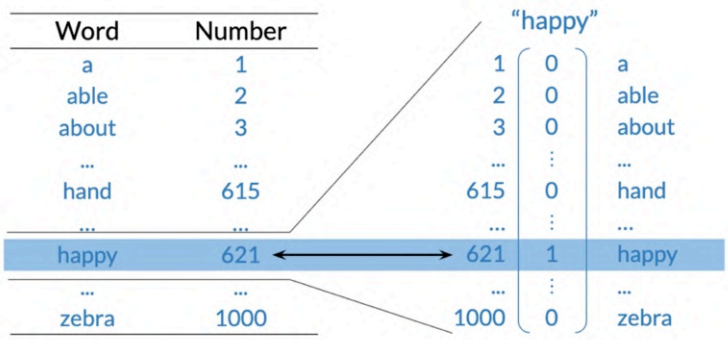
- Ordering: little semantic sense

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

One-hot vectors

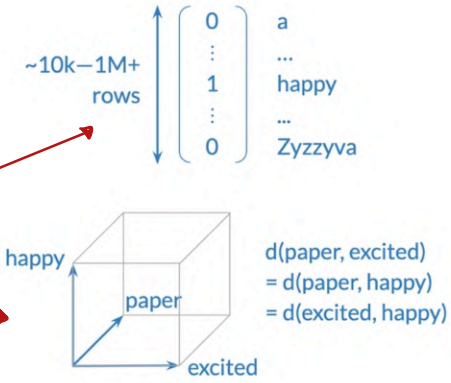


One-hot vectors

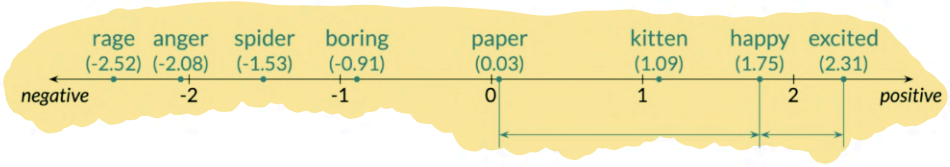


One-hot vectors

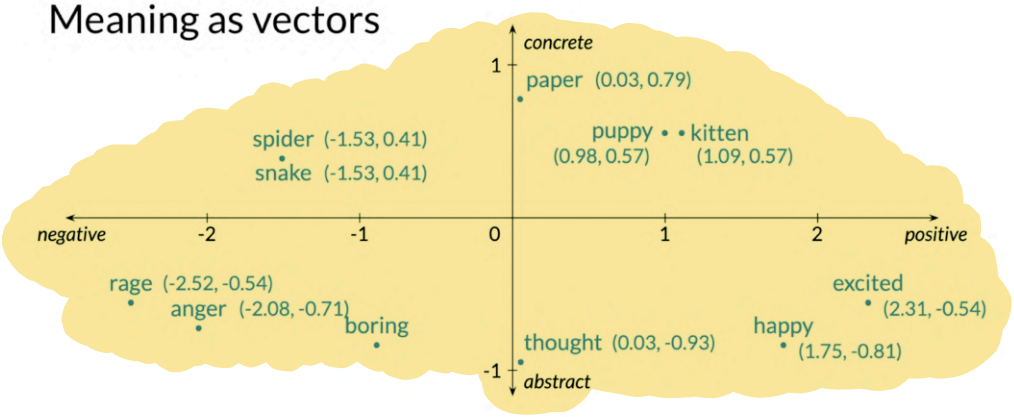
+ Simple
+ No implied ordering
- Huge vectors
- No embedded meaning



Meaning as vectors



Meaning as vectors

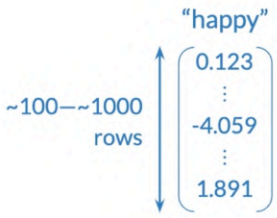


Word embedding vectors

- \+ 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

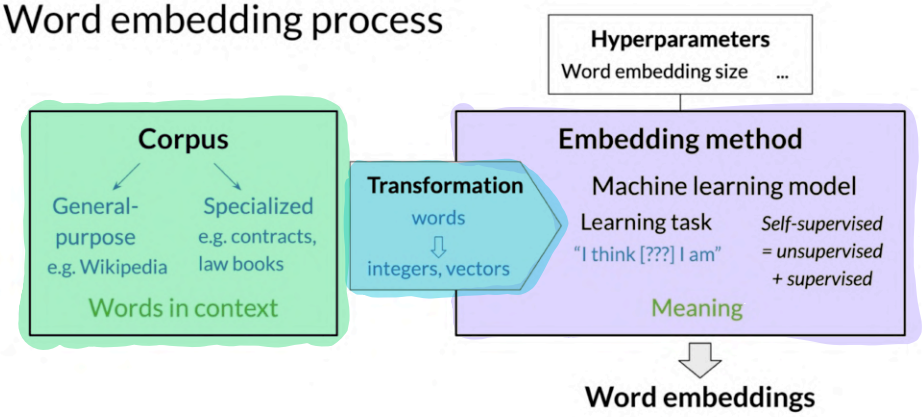
"word vectors"

word embeddings

Summary

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

Word embedding process



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

The model learns to predict the word surrounding a given input word.

Global vectors or glove by Stanford, which involves factorizing the logarithm of the corpuses word co-occurrence matrix, which is similar to the counter matrix you've used before.

fastText by Facebook, which is based on the skip-gram model and takes into account the structure of words by representing words as an n-gram of characters. This enables the model to support previously unseen words, known as outer vocabulary words, by inferring their embedding from the sequence of characters they are made of and the corresponding sequences that it was initially trained on. For example, it would create similar embeddings for kitty and kitten, even if it had never seen the word kitty before. As kitty and kitten are made of similar sequences of characters. Another benefit of fastText, is that word embedding vectors can be averaged together to make vector representations of phrases and sentences.

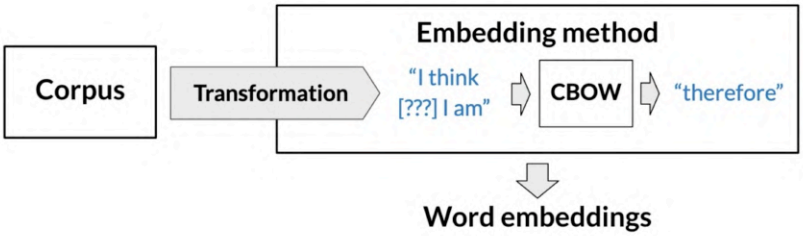
Advanced word embedding methods

Deep learning, contextual embeddings

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

} Tunable pre-trained models available

Continuous bag-of-words word embedding process



Center word prediction: rationale

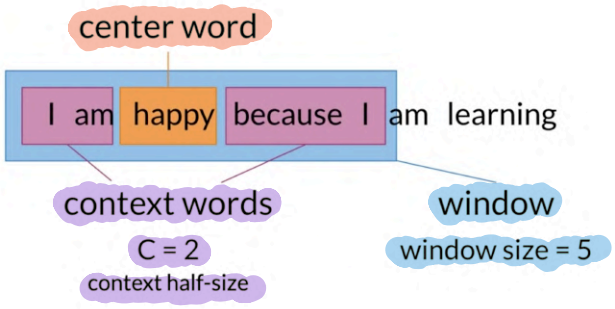
As a reminder, to create word embeddings, you need the corpus and the machine learning model that performs a learning task. One byproduct of the learning task is a set of word embeddings. You also need a way to transform the corpus into a representation that is suited to the machine learning model. In the case of the continuous bag-of-words model, the objective of the task is to predict a missing word based on the surrounding words. The rationale is that if two unique words are both frequently surrounded by a similar sets of words when used in various sentences, then those two words tend to be related in their meaning. Another way to say this is that they are related semantically. For example, in the sentence, the little something is barking. With a large enough corpus, the model will learn to predict that the missing word is related to dogs, such as the word dog itself or puppy, hound, terrier, and so on. So the model will end up learning the meaning of words based on their context.

The little ? is barking

dog
puppy
hound
terrier
...

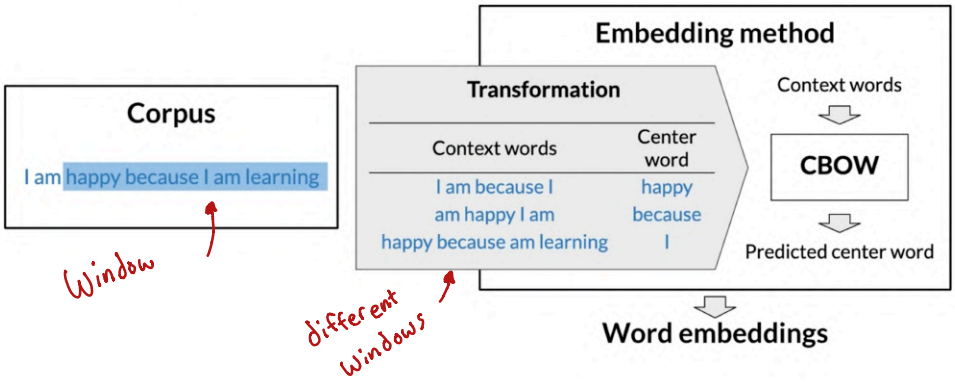
Creating a training example

Corpus → Transformation → CBOW

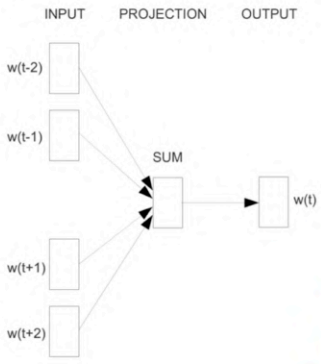


From corpus to training

Corpus → Transformation → CBOW



CBOW in a nutshell

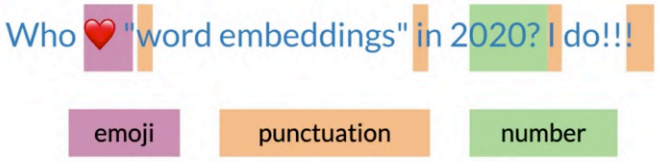


Source: Mikolov, T., Chen, K., Corrado, G.S., & Dean, J. (2013). Efficient Estimation of Word Representations in Vector Space

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', '.']
```

Sliding window of words in Python

The context have size stored in the variable C, which is the number of words to be taken on each side of the center word

number of context words

```
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):
    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

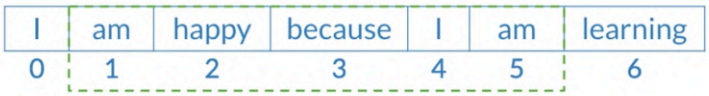
Sliding window of words in Python

```
def get_windows(words, C):
    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):
    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
```

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

	I	am	because	I	I am because I
am	0	1	0	0	0.25
because	0	0	1	0	0.25
happy	0	0	0	0	0
I	1	0	0	1	0.5
learning	0	0	0	0	0

(I + am + because + I) / 4 = I am because I

Final prepared training set

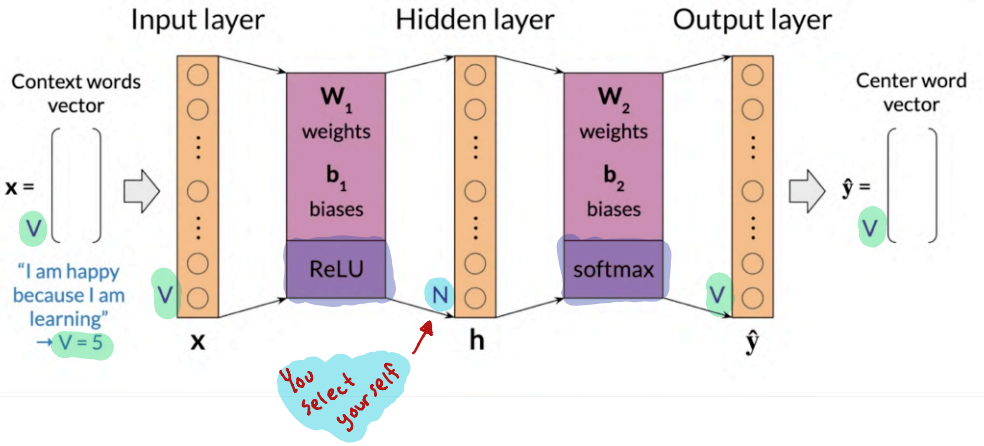
Context words	Context words vector	*Center word*	Center word vector
I am because I	[0.25; 0.25; 0; 0.5; 0]	*happy*	[0; 0; 1; 0; 0]
am happy I am	[0.5; 0; 0.25; 0.25; 0]	*because*	[0; 1; 0; 0; 0]
happy because am learning	[0.25; 0.25; 0.25; 0; 0.25]	*I*	[0; 0; 0; 1; 0]

Sliding windows

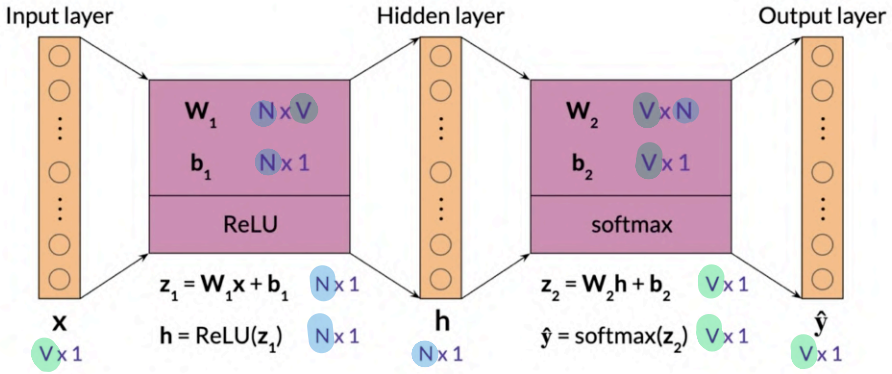
Notice, Context word vector has 0 where center word would be

Architecture of the CBOW model

Hyperparameters
N: Word embedding size ...



Dimensions (single input)



Dimensions (single input)

Column vectors

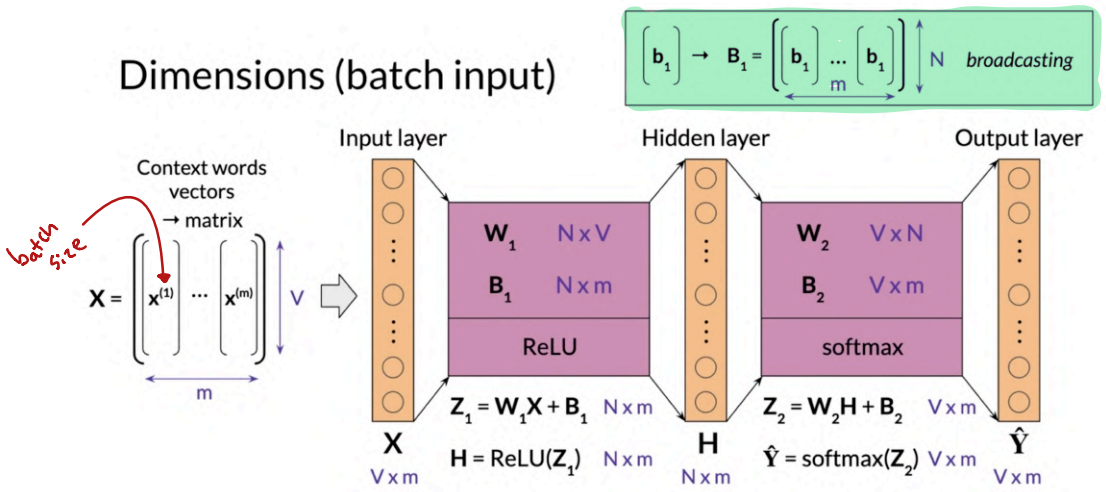
$$z_1 = W_1 x + b_1 \qquad z_1 = \begin{bmatrix} \\ \\ \end{bmatrix}_{N \times 1} \quad W_1 = \begin{bmatrix} N \times V \end{bmatrix} \quad x = \begin{bmatrix} \\ \\ \\ \end{bmatrix}_{V \times 1} \quad b_1 = \begin{bmatrix} \\ \\ \end{bmatrix}_{N \times 1}$$

Row vectors

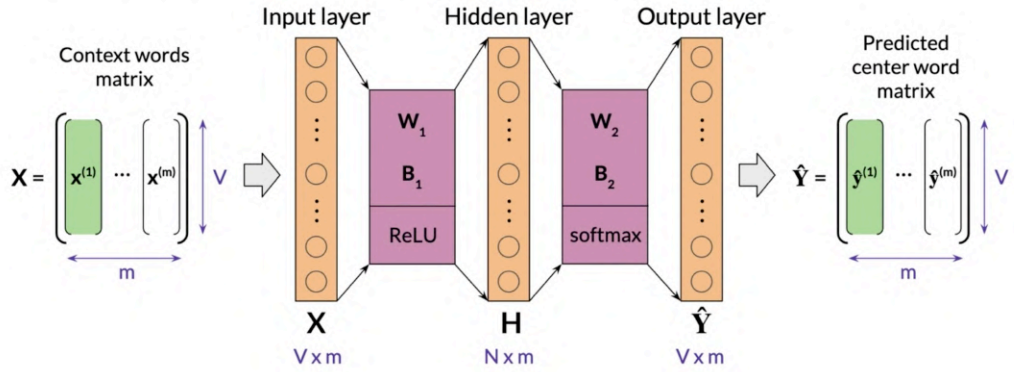
$$z_1 = x W_1^{\top} + b_1 \qquad b_1 = \begin{bmatrix} 1 \times N \end{bmatrix} \quad W_1 = \begin{bmatrix} N \times V \end{bmatrix} \quad b_1 = \begin{bmatrix} 1 \times N \end{bmatrix}$$

$$x = \begin{bmatrix} 1 \times V \end{bmatrix}$$

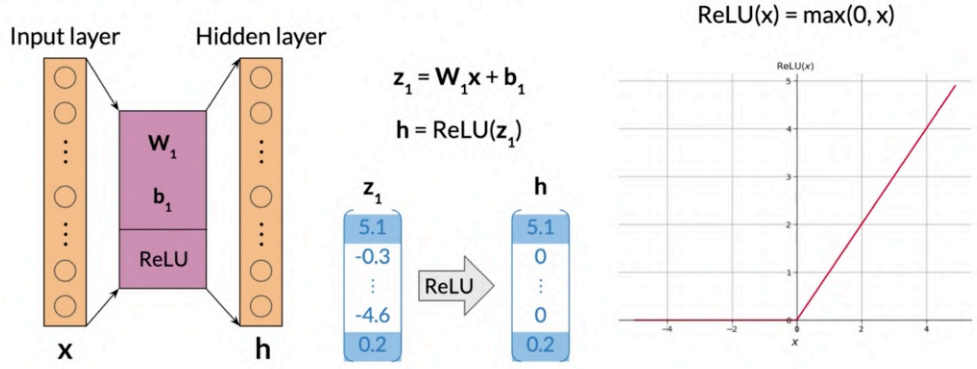
Dimensions (batch input)



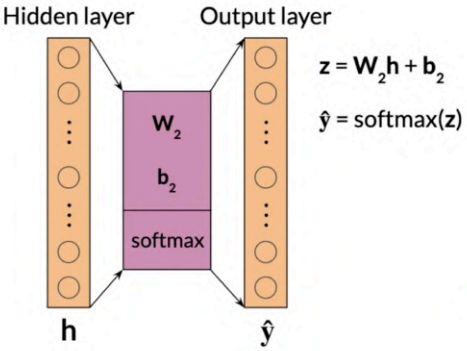
Dimensions (batch input)



Rectified Linear Unit (ReLU)

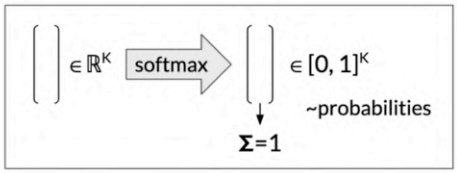


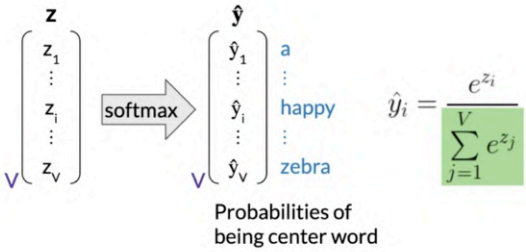
Softmax



$z = W_2 h + b_2$

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



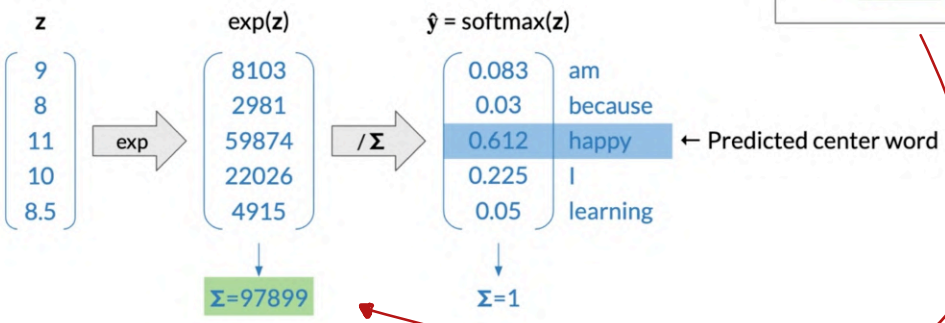


Probabilities of being center word

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

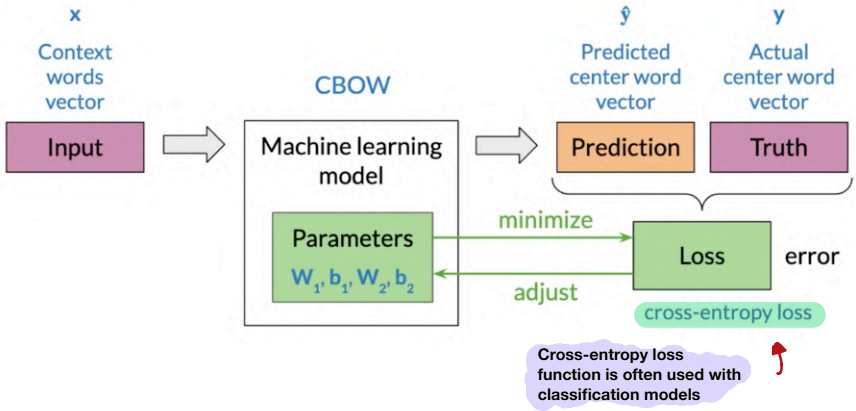
Softmax: example

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



← Predicted center word

Loss



Cross-entropy loss function is often used with classification models

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

y		ŷ	log(ŷ)	y ⊙ log(ŷ)
0	am	0.083	-2.49	0
0	because	0.03	-3.49	0
1	happy	0.611	-0.49	-0.49
0	I	0.225	-1.49	0
0	learning	0.05	-2.49	0

log → ⊙ y → -Σ → J = 0.49

Cross-entropy loss

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

What if prediction is wrong?

y		ŷ	log(ŷ)	y ⊙ log(ŷ)
0	am	0.96	-0.04	0
0	because	0.01	-4.61	0
1	happy	0.01	-4.61	-4.61
0	I	0.01	-4.61	0
0	learning	0.01	-4.61	0

log → ⊙ y → -Σ → J = 4.61 > $J^{(correct)}$ = 0.49

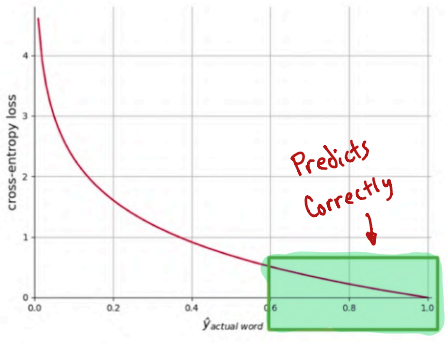
Cross-entropy loss function is often used with classification models

Cross-entropy loss

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

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

y		ŷ	
0	am	0.96	
0	because	0.01	
1	happy	0.01	→ J = 4.61
0	I	0.01	
0	learning	0.01	



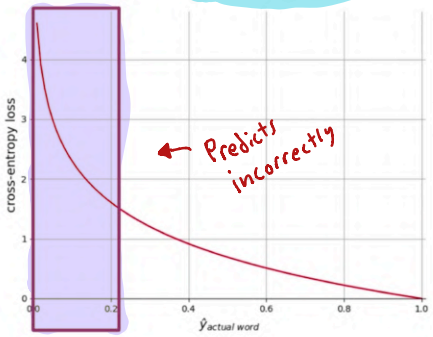
Cross-entropy loss

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

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

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



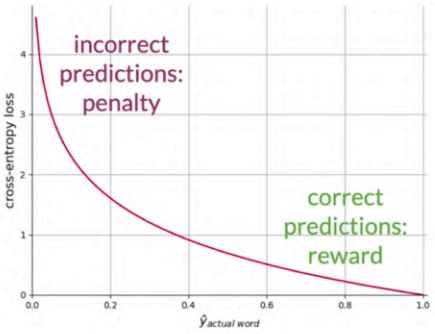
Cross-entropy loss

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

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

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

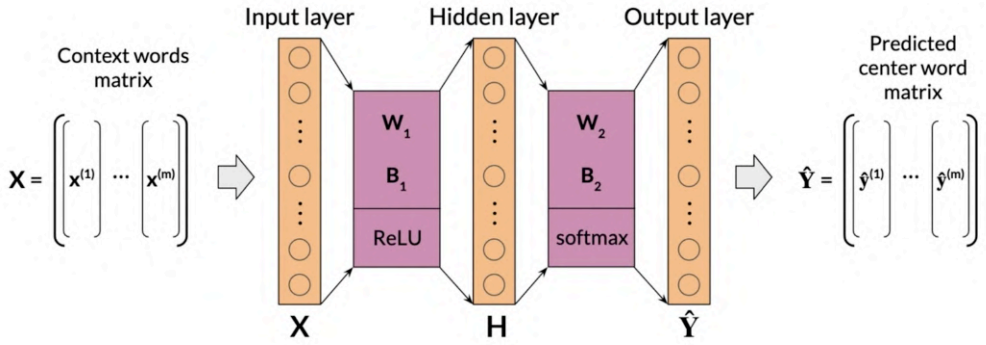


Training process

- Forward propagation
- Cost
- Backpropagation and gradient descent

Forward propagation

$$Z_1 = W_1 X + B_1 \qquad Z_2 = W_2 H + B_2$$
$$H = \text{ReLU}(Z_1) \qquad \hat{Y} = \text{softmax}(Z_2)$$



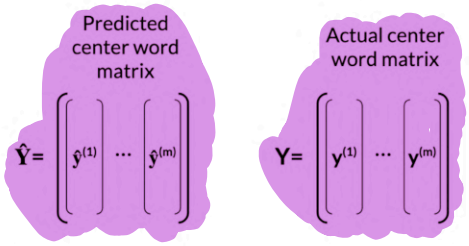
Cost

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

Cost: mean of losses

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

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



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

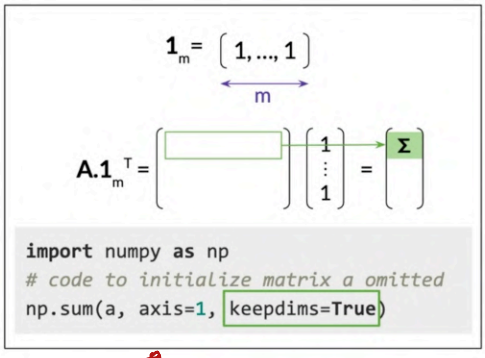
Backpropagation

$$\frac{\partial J_{batch}}{\partial \mathbf{W_1}} = \frac{1}{m}\text{ReLU}\left(\mathbf{W_2}^\top(\hat{\mathbf{Y}} - \mathbf{Y})\right)\mathbf{X}^\top$$

$$\frac{\partial J_{batch}}{\partial \mathbf{W_2}} = \frac{1}{m}(\hat{\mathbf{Y}} - \mathbf{Y})\mathbf{H}^\top$$

$$\frac{\partial J_{batch}}{\partial \mathbf{b_1}} = \frac{1}{m}\text{ReLU}\left(\mathbf{W_2}^\top(\hat{\mathbf{Y}} - \mathbf{Y})\right)\mathbf{1}_m^\top$$

$$\frac{\partial J_{batch}}{\partial \mathbf{b_2}} = \frac{1}{m}(\hat{\mathbf{Y}} - \mathbf{Y})\mathbf{1}_m^\top$$



Sum of columns

Set the keepdims parameter to True, so that the results can be broadcasts into a matrix of whatever size is necessary later in the calculations.

Gradient descent

Hyperparameter: learning rate α

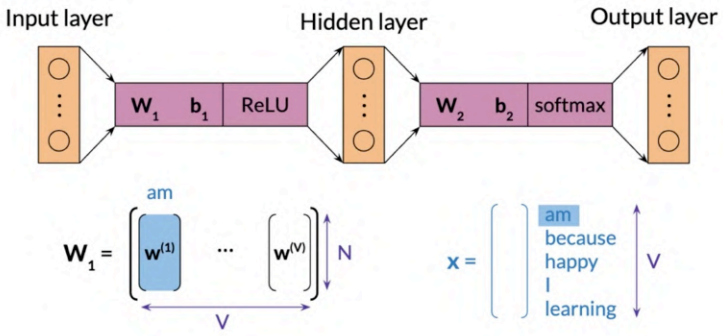
$$\mathbf{W_1} := \mathbf{W_1} - \alpha\frac{\partial J_{batch}}{\partial \mathbf{W_1}}$$

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

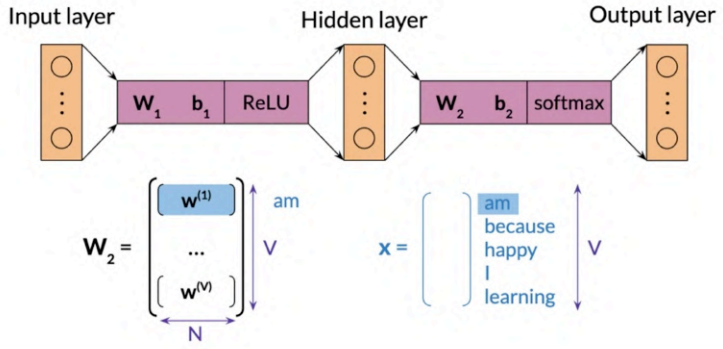
$$\mathbf{b_1} := \mathbf{b_1} - \alpha\frac{\partial J_{batch}}{\partial \mathbf{b_1}}$$

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

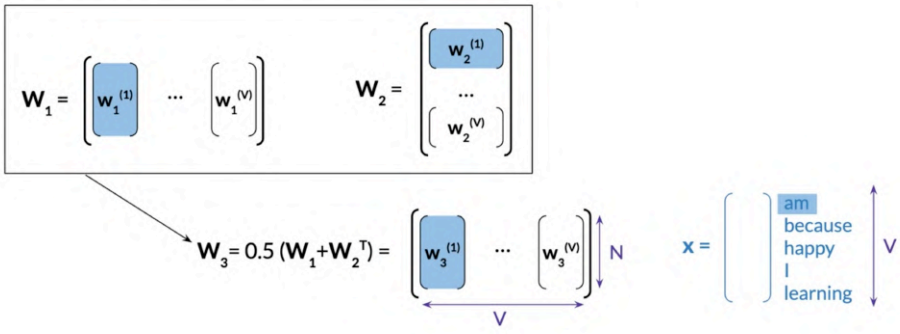
Extracting word embedding vectors: option 1



Extracting word embedding vectors: option 2



Extracting word embedding vectors: option 3



Intrinsic evaluation

Test relationships between words

- Analogies

You should be aware of for this approach, is that there could be several possible correct answers

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

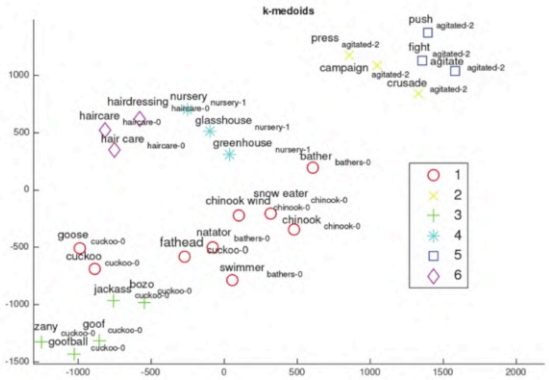
Relationship	Example 1	Example 2	Example 3
France - Paris	Italy: Rome	Japan: Tokyo	Florida: Tallahassee
big - bigger	small: larger	cold: colder	quick: quicker
Miami - Florida	Baltimore: Maryland	Dallas: Texas	Kona: Hawaii
Einstein - scientist	Messi: midfielder	Mozart: violinist	Picasso: painter
Sarkozy - 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

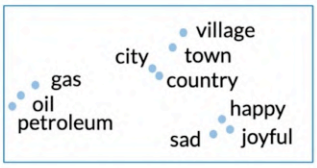
Source: Michael Zhai, Johnny Tan, and Jinho D. Choi. 2016. Intrinsic and extrinsic evaluations of word embeddings



Intrinsic evaluation

Test relationships between words

- Analogies
- Clustering
- Visualization



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

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)
```