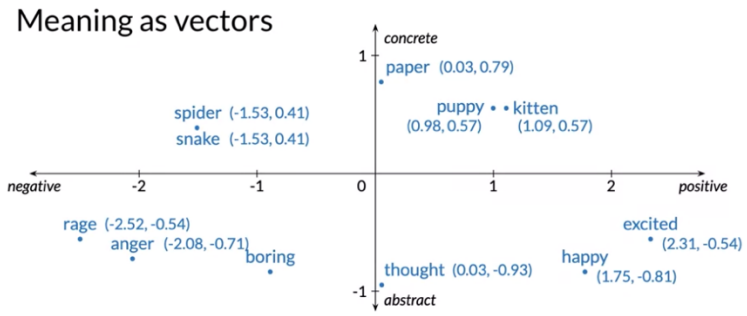
Word Embeddings

Basic Word Representations

- One hot vector instead of numerical index.
    - Simple, No implied ordering but Sparse matrix, no embedded meaning

---

Word Embeddings

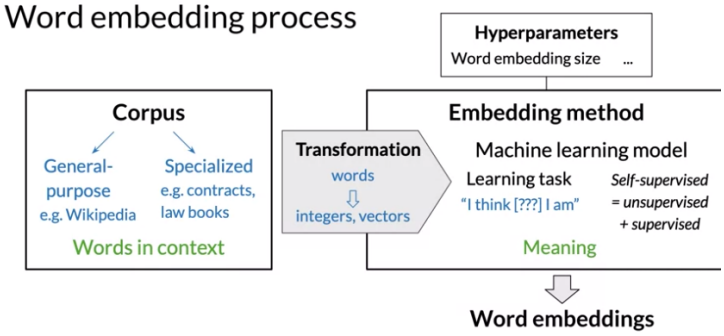


Word embedding vectors are

+ Low dimension

+ Embed meaning (semantic distance, analogies)

How to create word embeddings



Word Embedding Methods

<u>Basic word embedding methods</u>

Given word always has the same embedding

- Word2Vec (Google 2013)
    - Uses a shallow NN to learn word embeddings
    - 2 model architectures

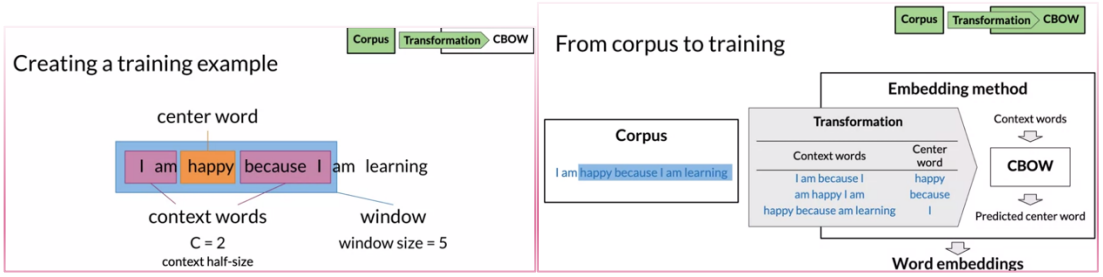
        - continuous bag of words - learn to predict a missing word given the surrounding words
        - Continuous skip-gram, also known as the skip-gram with negative sampling, which does the reverse of the continuous bag of words method. The model learns to predict the word surrounding a given input word
- Global Vectors GloVe (Stanford, 2014)
    - factorizing the logarithm of the corpuses word co-occurrence matrix, which is similar to the counter matrix
- FastText (Facebook, 2016)
    - 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

<u>Advanced word embedding methods</u>

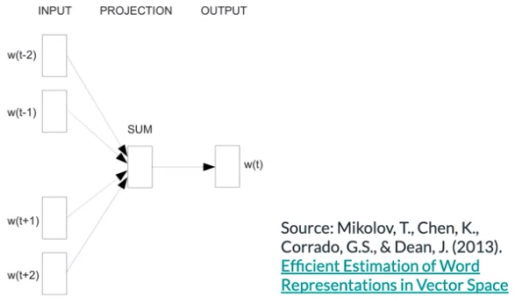
- Deep Learning, contextual embeddings
    - Words have different embeddings, depending on their context.
    - Adds support for words with similar meanings.
- BERT – Bidirectional encoder representations from transformers by Google
- ELMO for embeddings from language models by Allen Institute
- GPT-2 or generative pre-training 2 by Open AI

Continuous Bag-of-Words Model

Take context words and predict center word. Context word size left is same as size on right. That is known as **<u>half-size</u>**. Hyperparameter to the model.



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

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

```
# 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
```

```
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)
       ]
```

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

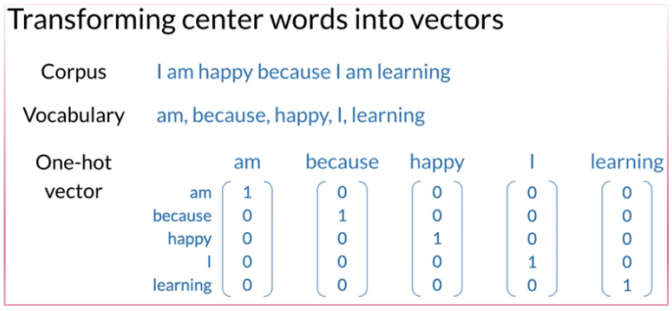
```
for x, y in get_windows(
            ['i', 'am', 'happy', 'because', 'i', 'am', 'learning'],
            2
        ):
    print(f'{x}\t{y}')
```

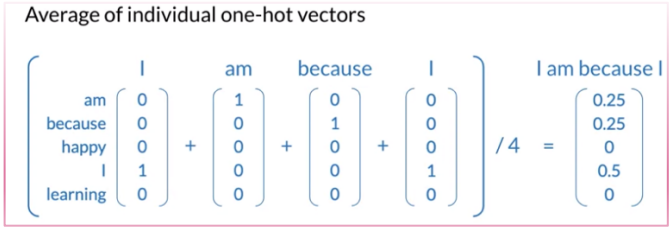
```
→ ['I', 'am', 'because', 'I']    happy
  ['am', 'happy', 'I', 'am']     because
  ['happy', 'because', 'am', 'learning']  I
```

```
# Define the 'word_to_one_hot_vector' function that will include the steps previously seen
def word_to_one_hot_vector(word, word2Ind, V):
    one_hot_vector = np.zeros(V)
    one_hot_vector[word2Ind[word]] = 1
    return one_hot_vector
```

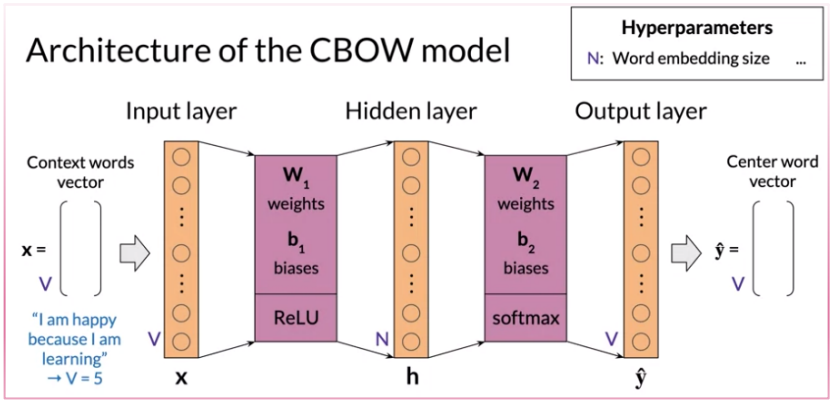
```
# Define the 'context_words_to_vector' function that will include the steps previously seen
def context_words_to_vector(context_words, word2Ind, V):
    context_words_vectors = [word_to_one_hot_vector(w, word2Ind, V) for w in context_words]
    context_words_vectors = np.mean(context_words_vectors, axis=0)
    return context_words_vectors
```

Transforming Words into vectors

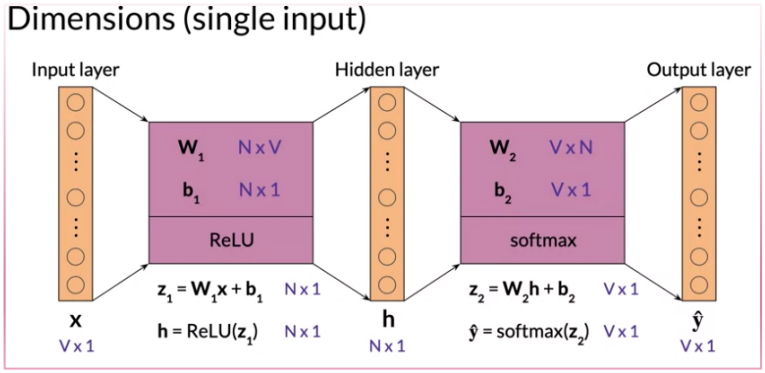


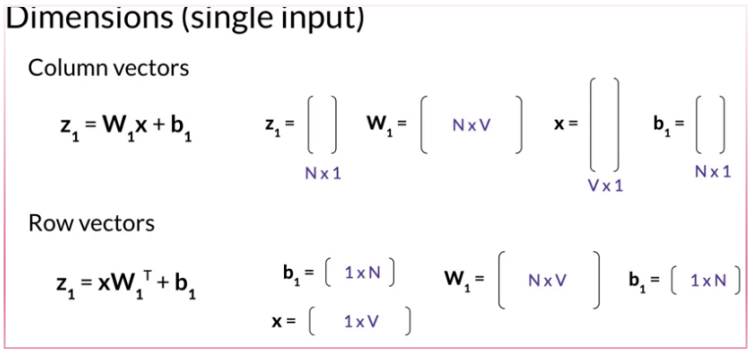


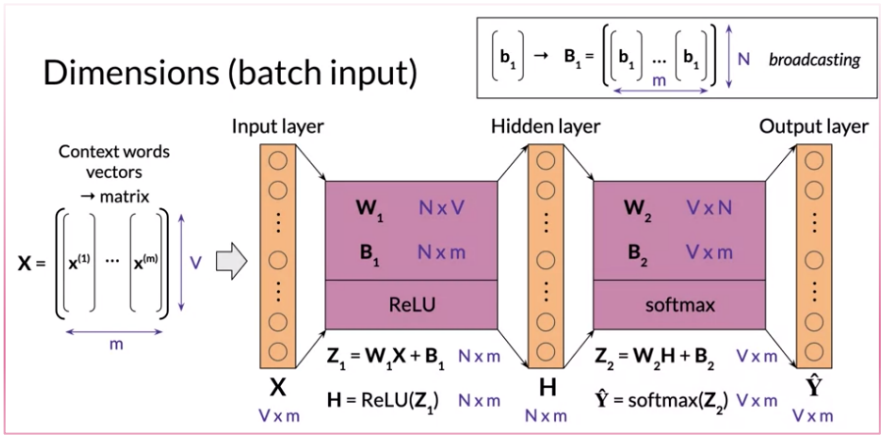
Architecture of the CBOW Model

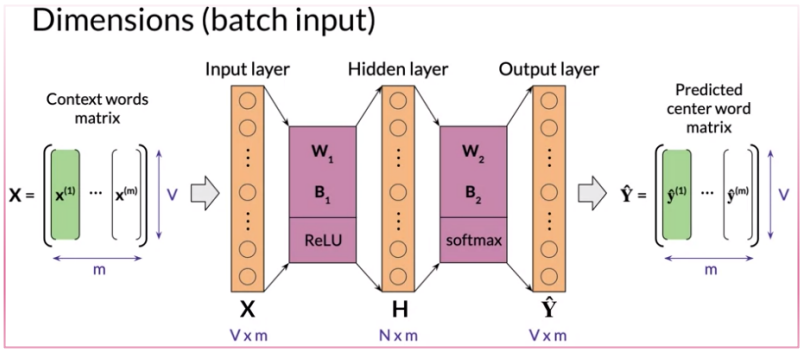


Architecture of the CBOW Model: Dimensions







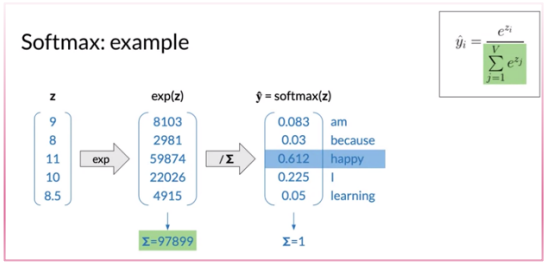


Architecture of the CBOW Model: Activation Functions

Relu(x) = max(0, x)

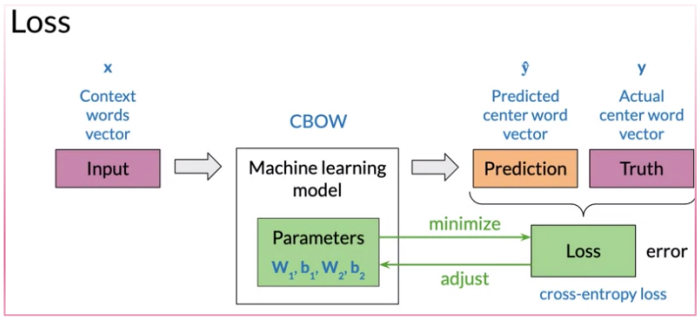
```
def relu(z):
    result = z.copy()
    result[result < 0] = 0
    return result
```

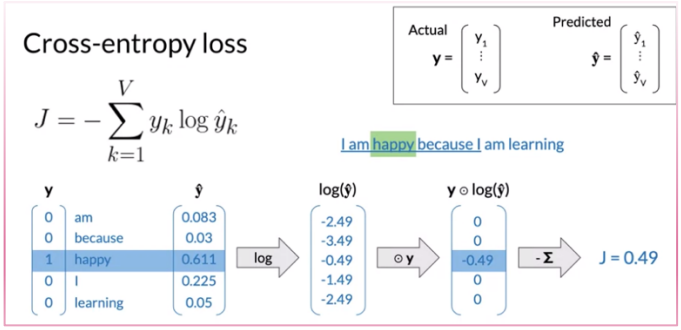
Softmax




```
def softmax(z):
    e_z = np.exp(z)
    sum_e_z = np.sum(e_z)
    return e_z / sum_e_z
```

Training a CBOW Model: Cost Function





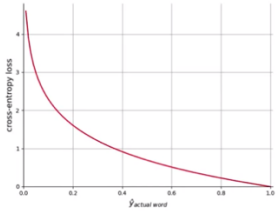
Cross-entropy loss

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

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

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

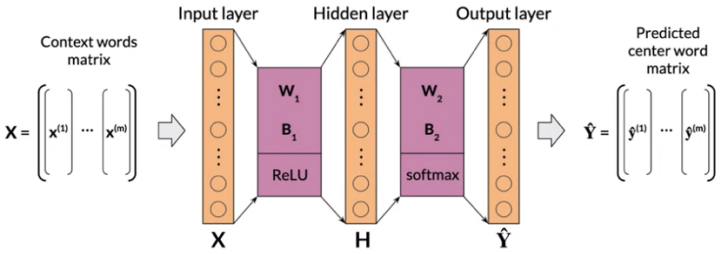


Training a CBOW Model: Forward Propagation

Forward propagation

$Z_1 = W_1X + B_1$ $\quad Z_2 = W_2H + B_2$

$H = \text{ReLU}(Z_1)$ $\quad \hat{Y} = \text{softmax}(Z_2)$



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

$\hat{Y}$=

Actual center word matrix

Y=

Training a CBOW Model: Backpropagation and Gradient Descent

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

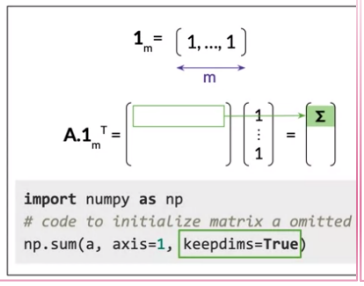
Backpropagation

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

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

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

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



Gradient descent

Hyperparameter: learning rate α

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

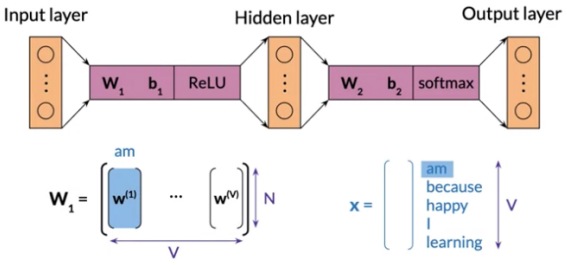
$$\mathbf{W_2} := \mathbf{W_2} - \alpha\frac{\partial J_{batch}}{\partial \mathbf{W_2}}$$

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

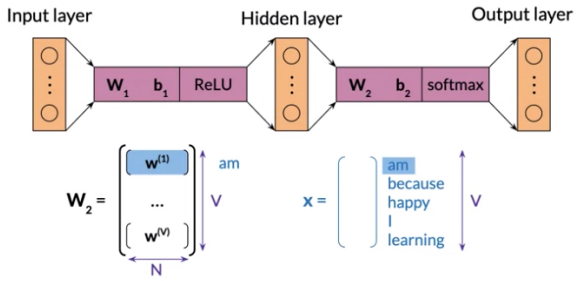
$$\mathbf{b_2} := \mathbf{b_2} - \alpha\frac{\partial J_{batch}}{\partial \mathbf{b_2}}$$

Extracting Word Embedding Vectors

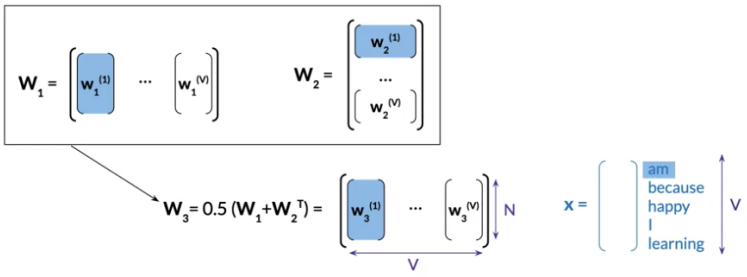
Extracting word embedding vectors: option 1



Extracting word embedding vectors: option 2



Extracting word embedding vectors: option 3



Evaluating Word Embeddings: Intrinsic Evaluation

- Test relationship between words

- Semantic refers to meaning of words
  - Analogies – France is to Paris as Italy is to _____
- Syntactic refers to grammar
  - Analogies Seen is to saw as been is to ?
- Clustering

-
- Visualization

Evaluating Word Embeddings: Extrinsic Evaluation

To evaluate word embeddings with extrinsic evaluation, you use the word embeddings to perform an external task, which is typically the real world task that you initially needed the word embeddings for. Then, use the performance metric of this task as a proxy for the quality of the word embeddings. Examples of useful word level tasks include named entity recognition or 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

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)
```