UNIVERSITÀ DEGLI STUDI DI NAPOLI
PARTHENOPE

Natural Language Processing

Word Embeddings

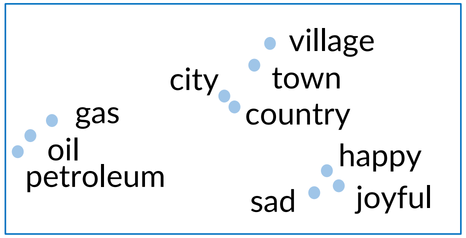
LESSON 22-23

prof. Antonino Staiano

M.Sc. In ''Machine Learning e Big Data'' - University Parthenope of Naples

Basic Applications of Word Embeddings

- Given trained word embeddings, one might use them for
  - Finding analogies between words and calculating the similarity of words
  - Combining with a classifier
  - to perform, for instance, sentiment analysis or
  - To classify customer comments or reviews from user feedback surveys



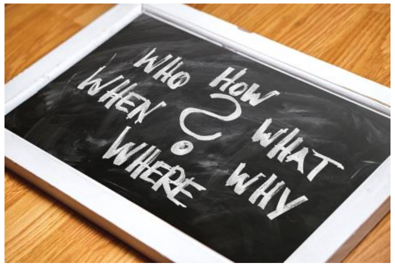
Semantic analogies and similarity

Sentiment analysis

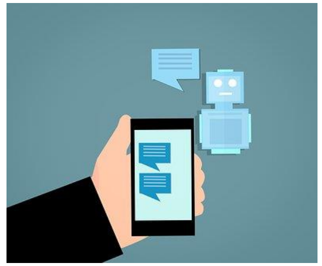
Classification of customer feedback

Advanced Applications of Word Embeddings

Machine translation



Information extraction



Question answering

What we’re going to learn

- Identify the key concepts of word representations
  - Words’ numerical representation for using mathematical models
- Generative word embeddings
  - How a model learns word embeddings from data
- Prepare text for machine learning
  - Transforming a corpus of text into a training set for a machine learning model
- Continuous bag-of-words model
  - One of the ways to create word embeddings

Basic word representation

- With word vectors, one will be able to create a numerical matrix to represent all the words in a vocabulary
  - Each row vector of the matrix corresponds to one of the words
- There are several ways to represent words a numbers
  - Integers
  - One-hot vectors
  - Word embeddings

Integers

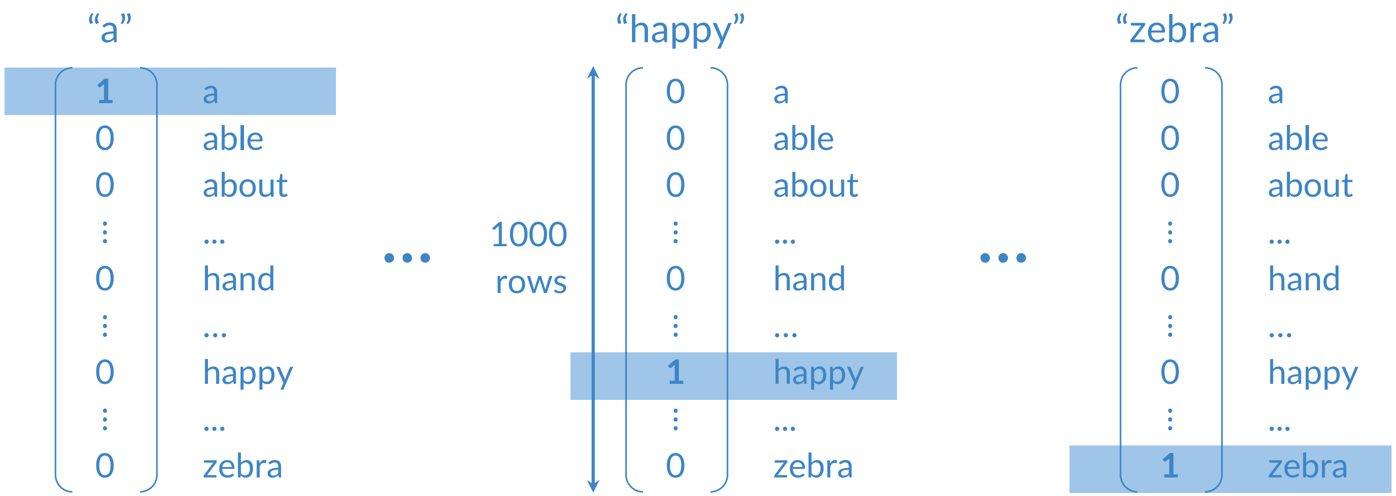
- To assign a unique integer to each word
  - \+ Simple
  - \- Ordering: little semantic sense

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

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

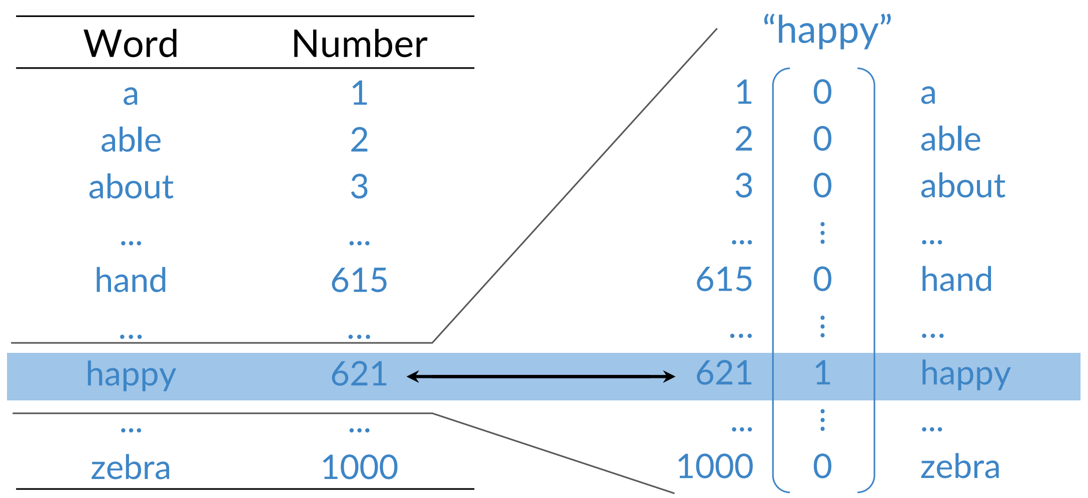
One-hot vectors

- Represent the words using a column vector where each element corresponds to a word in the vocabulary



One-hot vectors

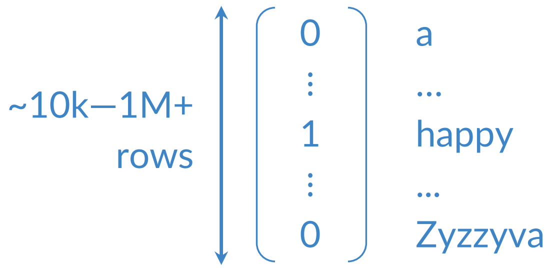
- Words can be considered categorical variables
  - Simple to go from integer to one-hot vectors and back
    - Mapping the words in the rows to their corresponding row number



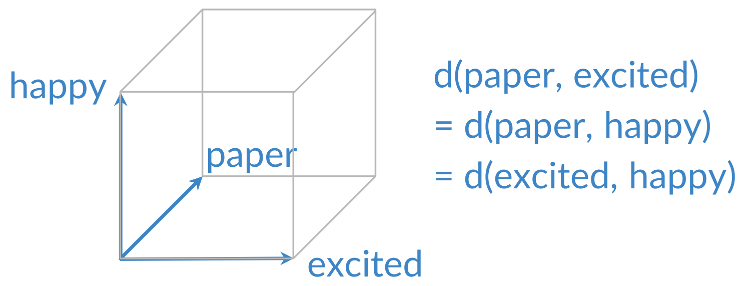
One-hot vectors

- + Simple
- + No implied ordering

- - Huge vectors

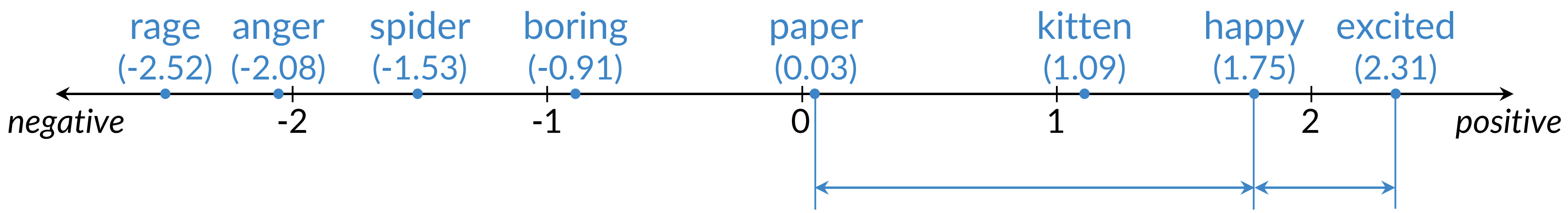


- - No embedded meaning



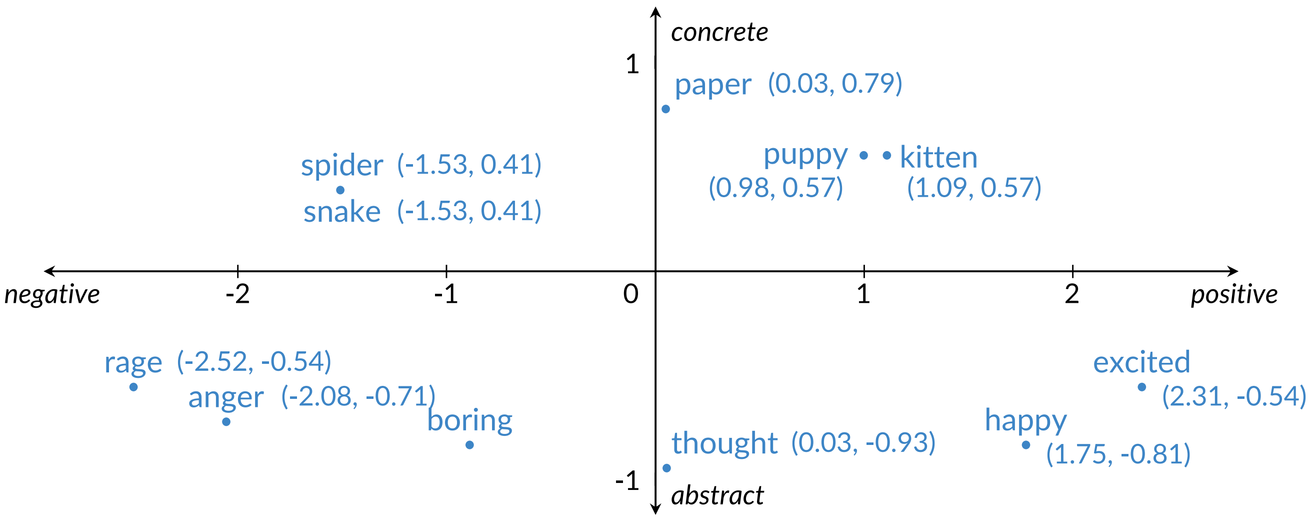
Word Embeddings

- Meaning as vectors
- Example
  - Words along one axis
    - Imagine storing their positions as numbers in a single 1-length vector
    - We can use any decimal value
      - *happy* and *excited* most similar to each other as compared to paper



Meaning as vectors

- We can extend by adding a vertical number line
  - The vocabulary of words is represented with a small vector of length 2



  - We gained some meaning while losing some precision
    - It is possible for two words to be located on the same point in this 2D plot (e.g., *snake* and *spider*)
  - The more coordinates you have, the more things you can capture
- That is an example of the word embedding

Word embedding vectors

- \+ Low dimension
  - Practical for computations
- \+ Embed meaning
  - e.g., semantic distance
    - *forest* ≈ *tree* and *forest* ≠ *ticket*
  - e.g., analogies
    - *Paris*:*France* = *Rome*:?
- Encoding the meaning of words is also the first step towards encoding the meaning of entire sentences
  - which is the foundation for more complex NLP use cases, e.g., question answering and translation

“happy”

$$\text{\textasciitilde 100—\textasciitilde 1000 rows} \quad \begin{pmatrix} 0.123 \\ \vdots \\ -4.059 \\ \vdots \\ 1.891 \end{pmatrix}$$

Terminology

- All vector representations of words, including one-hot vectors and word embedding vectors, are known as word vectors
  - More commonly, the terms word vector and word embeddings are used as well to refer to word embedding vectors

word vectors

integers

one-hot vectors

word embedding vectors

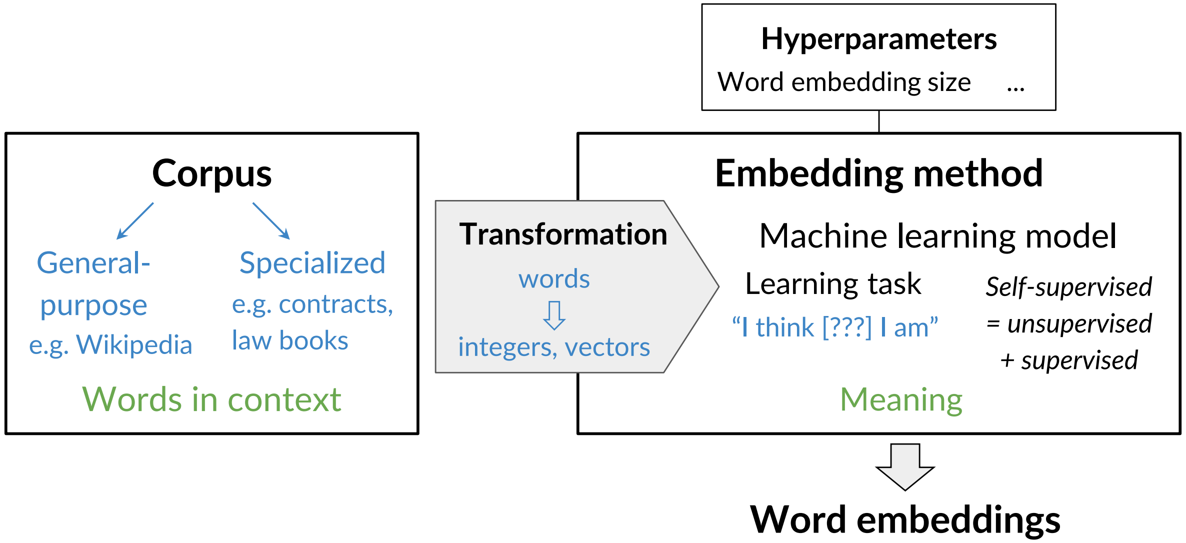
"word vectors"

word embeddings

How to create word Embeddings

Word embeddings process

- To create a word embedding we need
  - a corpus text
    - The context of a word tells you what type of words tend to occur near that specific word. The context is important as this is what will give meaning to each word embedding
  - an embedding method



Basic Word Embedding Methods

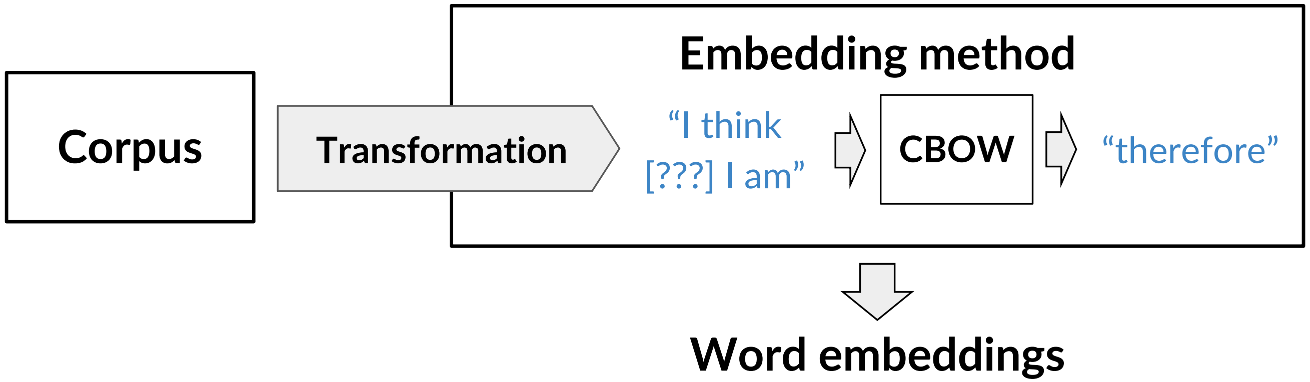
- Word2vec (Google, 2013)
  - Uses a shallow neural network to learn word embeddings
    - Continuous bag-of-words (CBOW)
    - Continuous skip-gram/Skip-gram with negative sampling
- Global vectors (GloVe) (Stanford, 2014)
  - Factorizes the log of the corpora word co-occurrence matrix
- fastText (Facebook, 2016)
  - Considers the structure of words by representing words as an n-gram of characters
  - Supports out-of-vocabulary (OOV) words
  - Word embedding vectors can be averaged together to make vector representations of phrases and sentences

Advanced Word Embedding Methods

- Use advanced deep neural network architectures to refine the representation of the words' meaning according to their contexts
  - The words have different embedding depending on their context
- Deep Learning, contextual embeddings
  - BERT (Google, 2018)
    - Bidirectional Encoder Representations from Transformers
  - ELMo (Allen Institute for Ai, 2018)
    - Embeddings from Language Models
  - GPT-2 (OpenAI, 2018)
    - Generative PreTraining models
- Available off-the-shelf pretrained models

Continuous Bag-of-Words Model

- Recap, one needs
  - Corpus
  - ML model for the learning task
  - Corpus transformation into a representation suited to the ML model
- The set of word embeddings is a byproduct of the learning task



Center word prediction: rationale

Corpus | Transformation | CBOW

- If two unique words are both frequently surrounded by similar sets of words in various sentences, then those words are semantically related

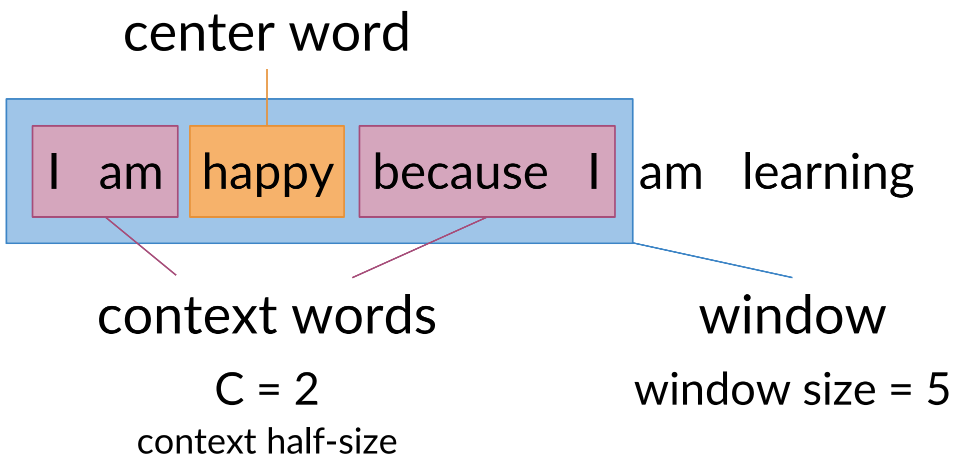
The little ___?___ is barking

dog
puppy
hound
terrier
...

- The model will end up learning the meaning of words based on their contexts

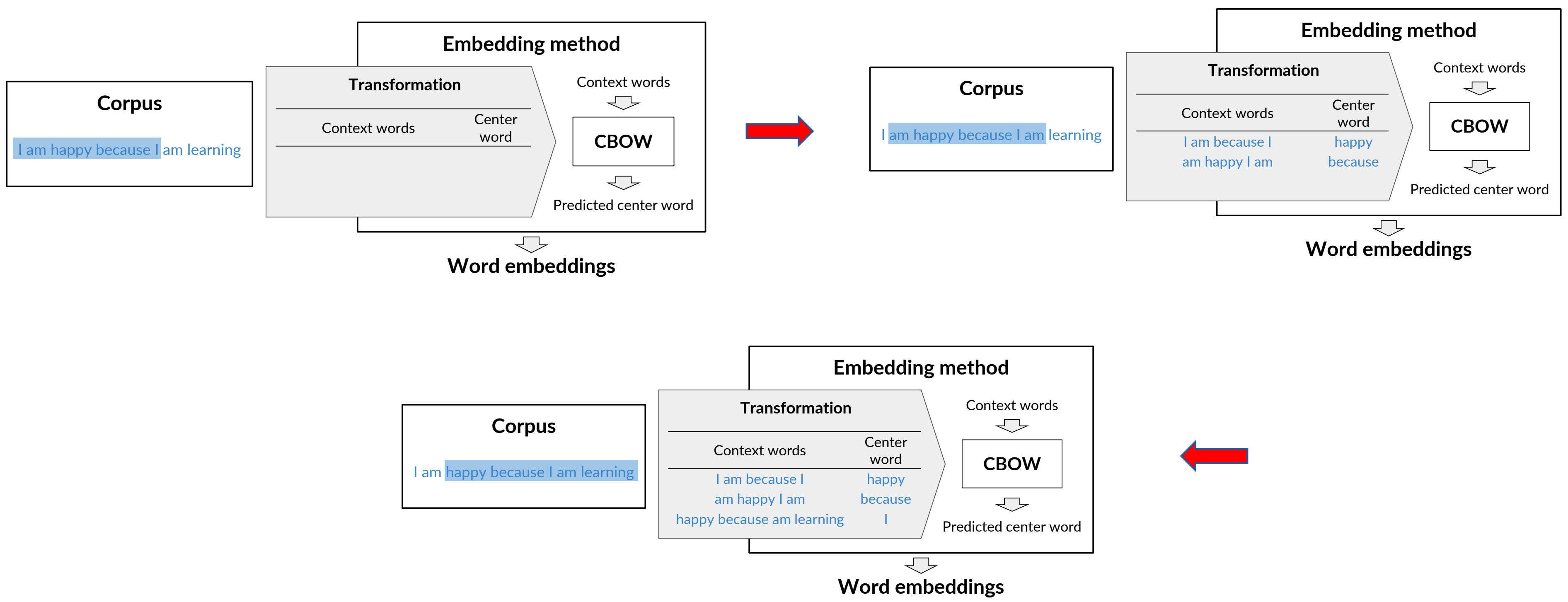
Creating a training example

- Using the corpus to create training data
  - *I am happy because I am learning*
- Given a center word, e.g., *happy*, define the context as the C words just before and after the center word
  - C (hyperparameter of CBOW) is the half size of the context, C = 2 in this example
  - The window is the count of the center word plus the context words



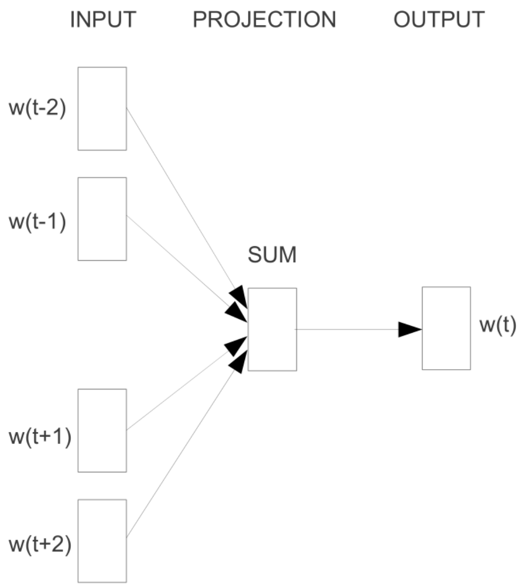
From corpus to training

- To train the models, one needs a set of examples
  - Context words and the center word to predict, each



CBOW in a nutshell

- To the model
  - context words as inputs
  - center words as outputs



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

Cleaning and tokenization

- The words of the corpus should be case insensitive
  - Uppercase or lowercase
- Handling of punctuations
  - E.g., all interrupting punctuation marks as a single special word in the vocabulary
    - One could ignore non-interrupting punctuation marks, e.g., quotation marks
    - Collapse multi-sign marks into single marks, …
- Handling of numbers
  - Drop all numbers not carrying any meaning
  - Keep the numbers if having special meaning for the use case
  - Tag as a special token if too many, e.g., many area codes
- Handling of special characters (Math, currency, … symbols)
  - Usually, dropped
- Handling special words (from tweets or reviews, e.g., Emojis, hashtags)
  - Depending on the goals of your task

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

Transforming words into vectors

- To feed the context words into the model and to predict and central word, they must be suitably represented
  - Center words into vectors
    - First, create the vocabulary V of unique words in the corpus
    - Encode each word as a one-hot vector of size |V|

Corpus I am happy because I am learning

Vocabulary am, because, happy, I, learning

One-hot vector

$$
\begin{array}{r}
\\ \text{am} \\ \text{because} \\ \text{happy} \\ \text{I} \\ \text{learning}
\end{array}
\begin{array}{c}
\text{am} \\ \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}
\end{array}
\begin{array}{c}
\text{because} \\ \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}
\end{array}
\begin{array}{c}
\text{happy} \\ \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}
\end{array}
\begin{array}{c}
\text{I} \\ \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}
\end{array}
\begin{array}{c}
\text{learning} \\ \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}
\end{array}
$$

Transforming context words into vectors

- To feed the context words into the model and to predict and central word, they must be suitably represented
  - Context words into vectors
    - Create a single vector that represents the context from all the context words

Average of individual one-hot vectors

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

Final prepared training set

- Example
  - First window

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]

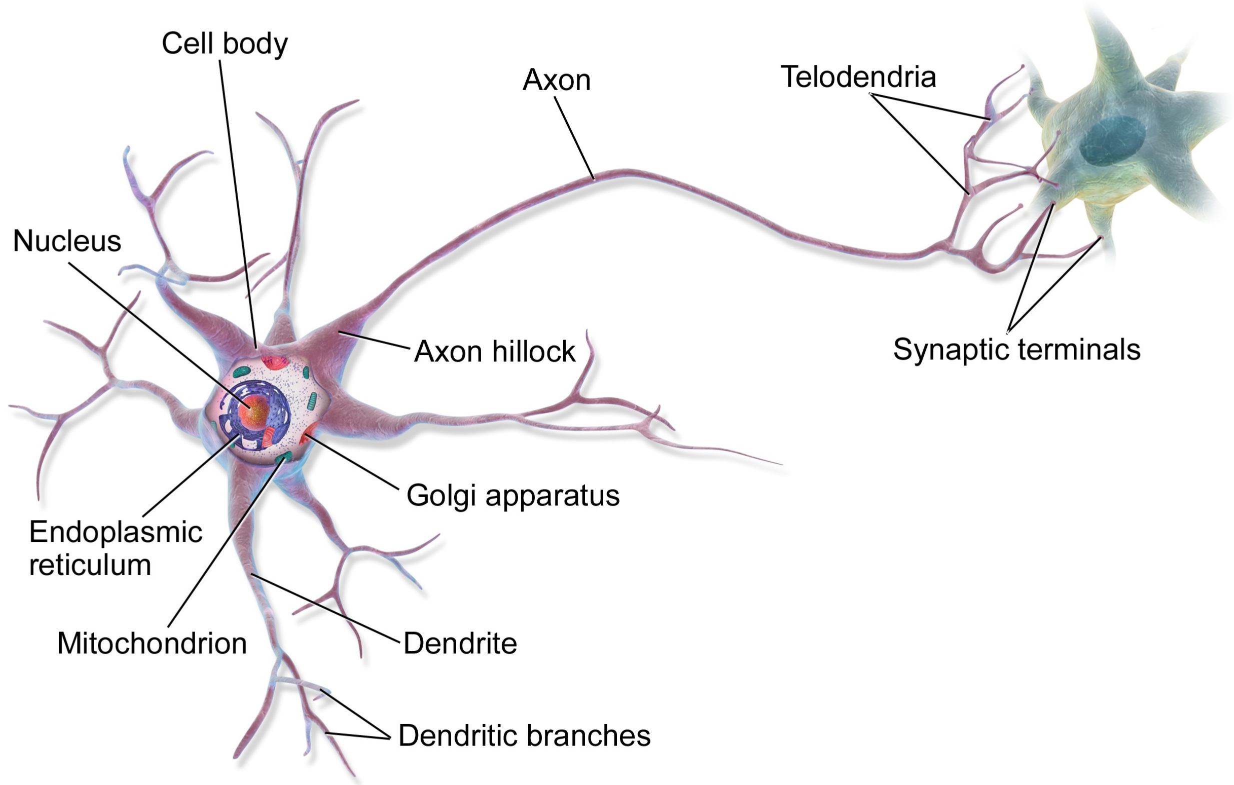
  - Note that the vectors are actually column vectors

Towards the CBOW model

The quickest recap on neural networks

Biological neuron

- A neuron is a small "computational" unit
  - 10^{10} in our brain
  - Its "power" is due to links to other neurons making a big network of 100 trillion connections!!!



Artificial neuron

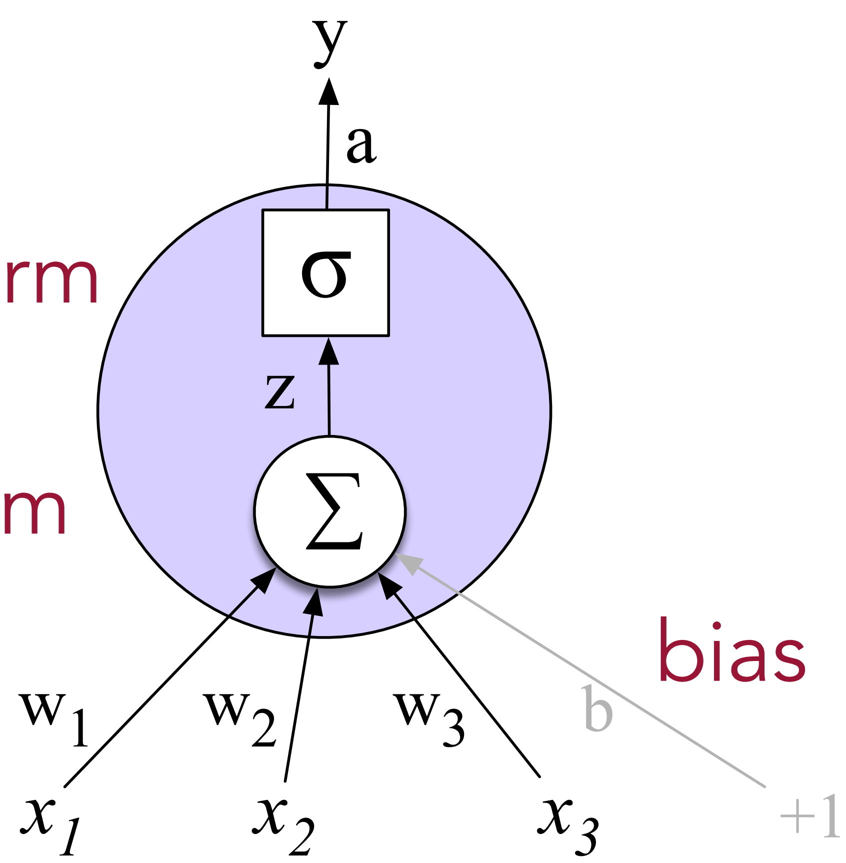
Output value

Non-linear transform

Weighted sum

Weights

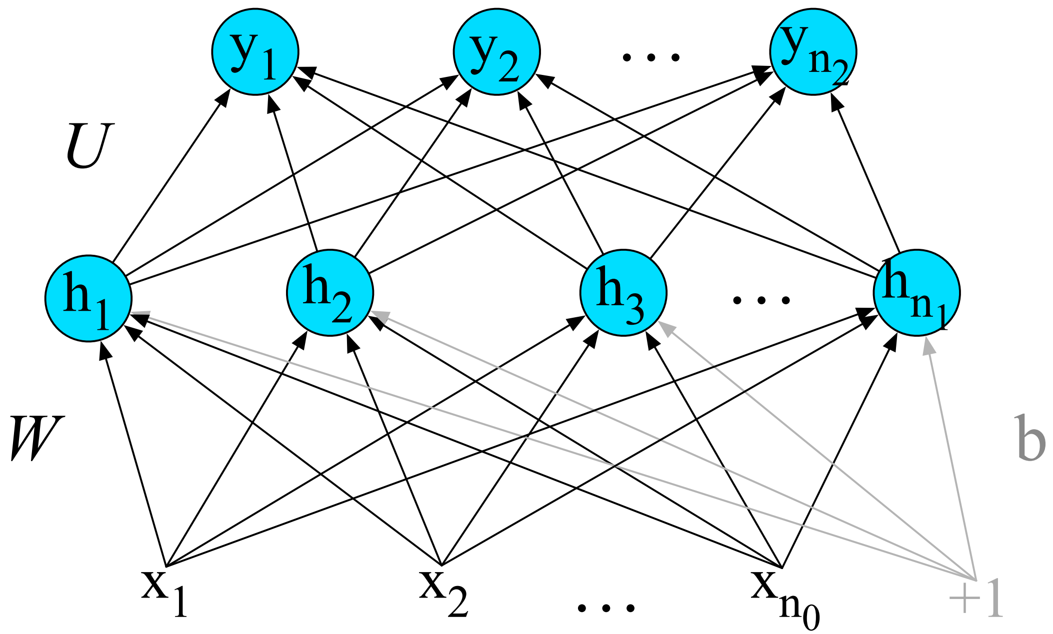
Input layer



bias

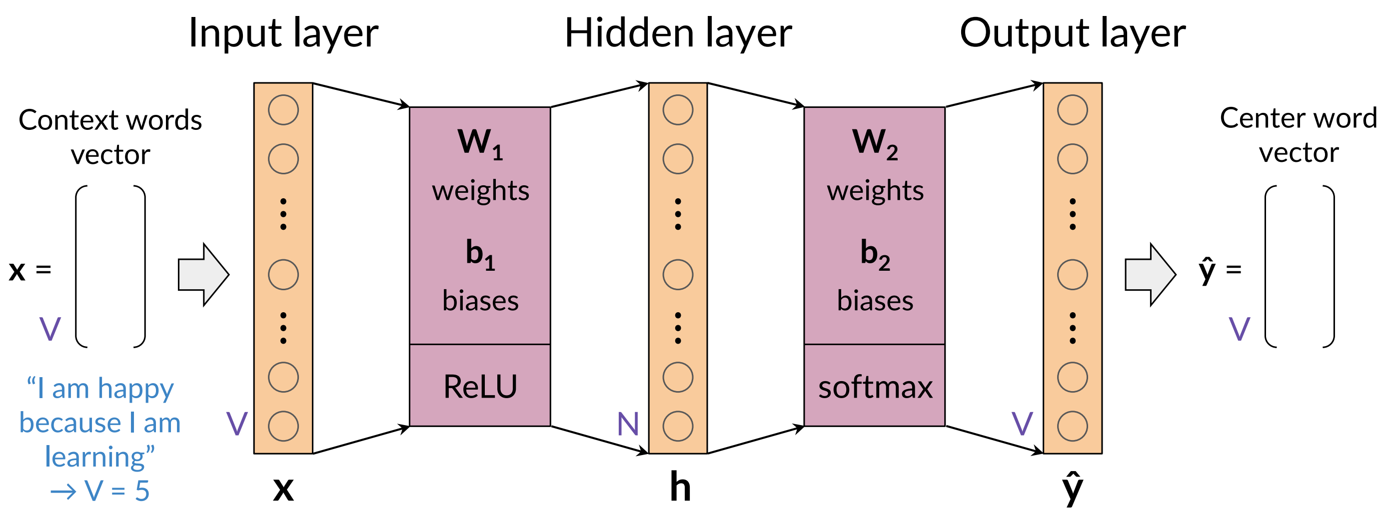
Feedforward Neural Networks

- Can also be called **multi-layer perceptrons** (or **MLPs**)

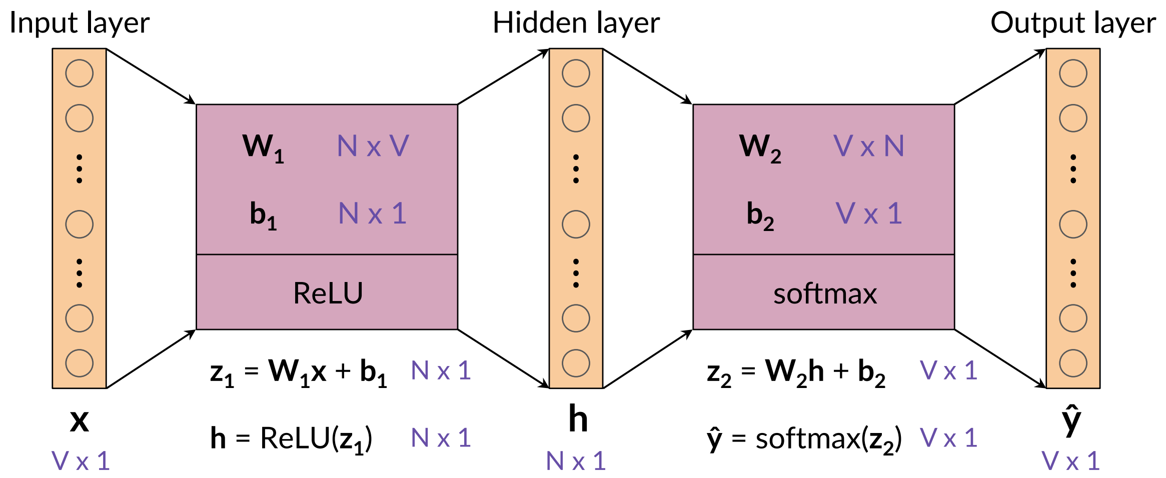


Architecture of the CBOW model

- The CBOW model is based on a shallow dense neural network
  - Hyperparameters
    - N: word embedding size … (typically 100 - 1000)
    - W_1,b_1, W_2, b_2, parameters to be learned during training
      - Word embeddings are derived from weight matrices



Dimensions (single input)



Column vectors

$$\mathbf{z_1} = \mathbf{W_1 x} + \mathbf{b_1}$$

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

Row vectors

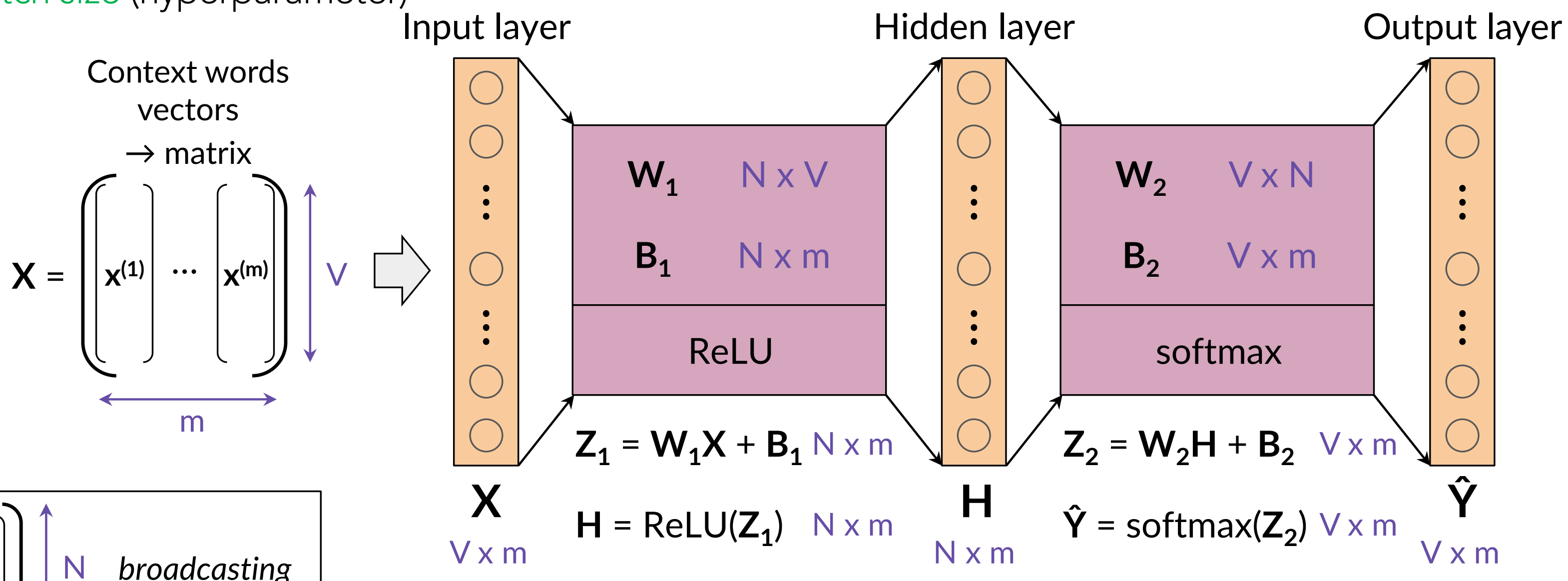
$$\mathbf{z_1} = \mathbf{x W_1}^T + \mathbf{b_1}$$

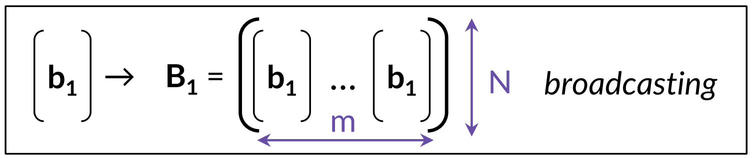
$\mathbf{b_1} = \begin{bmatrix} 1 \times N \end{bmatrix}$ $\quad \mathbf{W_1} = \begin{bmatrix} N \times V \end{bmatrix}$ $\quad \mathbf{b_1} = \begin{bmatrix} 1 \times N \end{bmatrix}$

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

Dimensions (batch input)

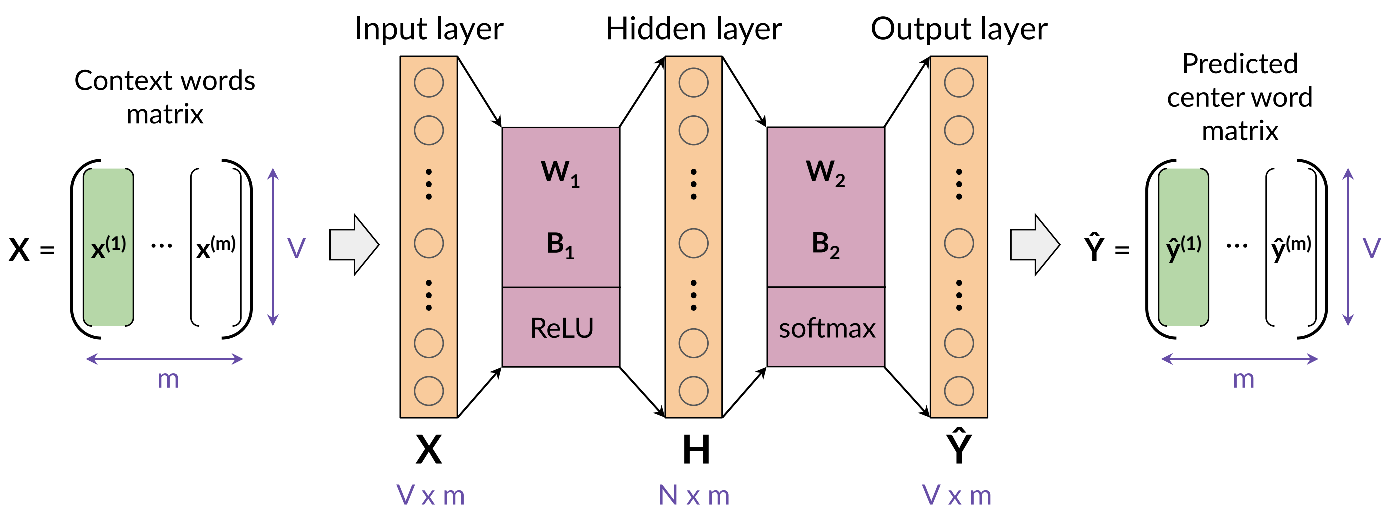
- Batch processing
  - Quicker learning
    - the model is fed with several inputs (m) and provides several outputs at the same time
    - m is called batch size (hyperparameter)





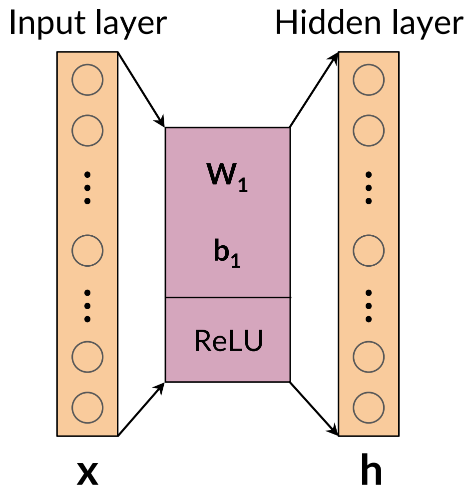
Dimensions (batch input)

- The vector from the first column of X is transformed into the vector corresponding to the first column of $\hat{Y}$
  - Similarly for the remaining m-1 vectors



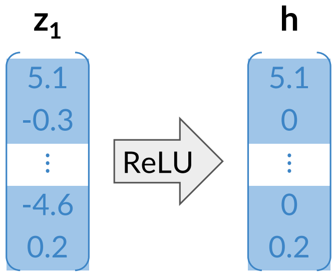
Activation Functions: Hidden layer neurons

- Rectified Linear Unit (ReLU)

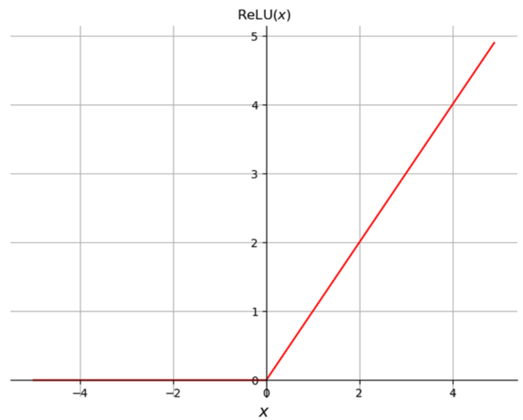


$$\mathbf{z_1} = \mathbf{W_1 x} + \mathbf{b_1}$$

$$\mathbf{h} = \mathrm{ReLU}(\mathbf{z_1})$$

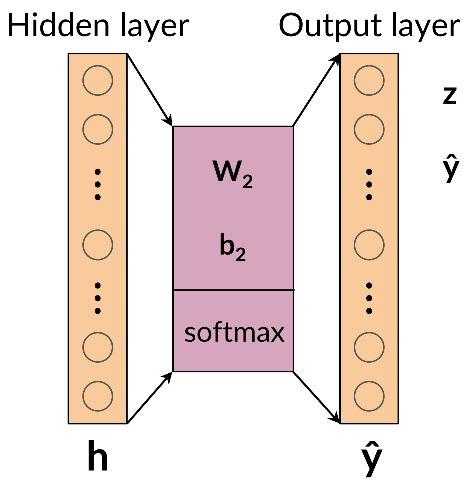


ReLU(x) = max(0, x)



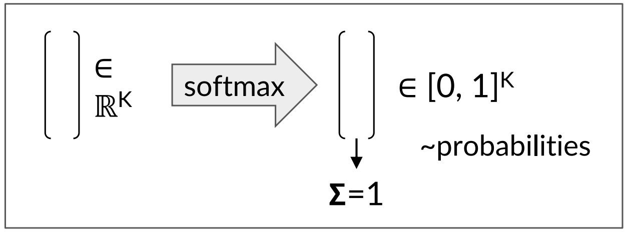
Activation Functions: Output neurons

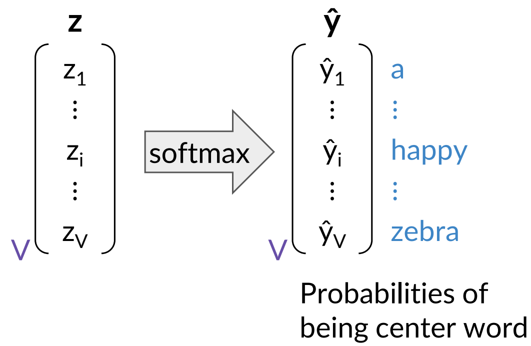
- Softmax



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

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

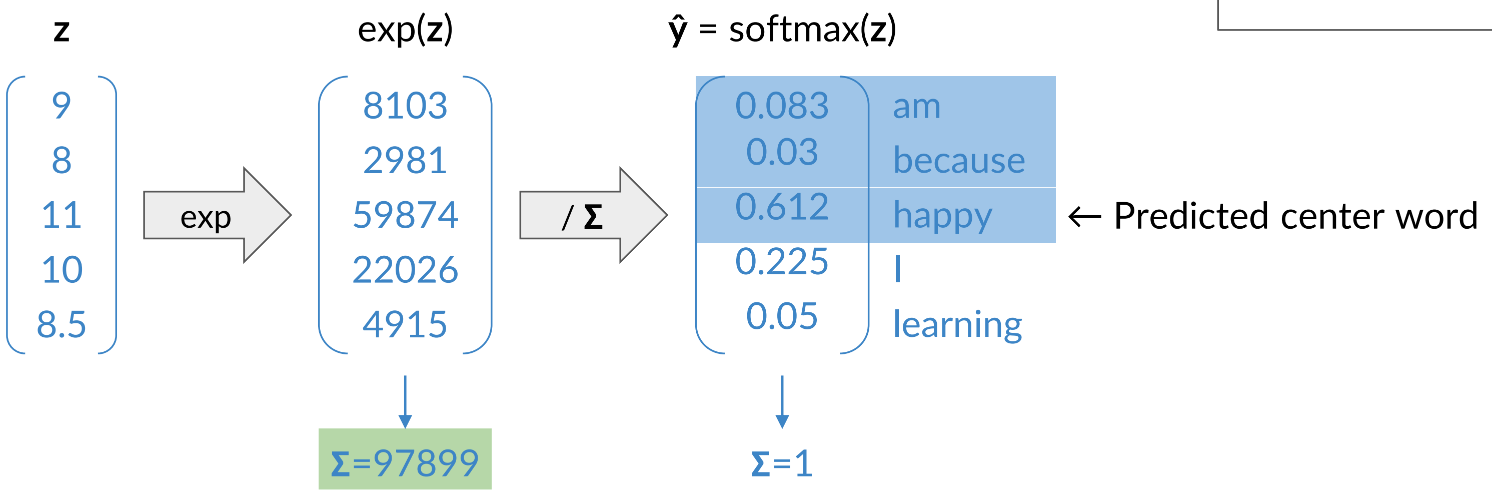




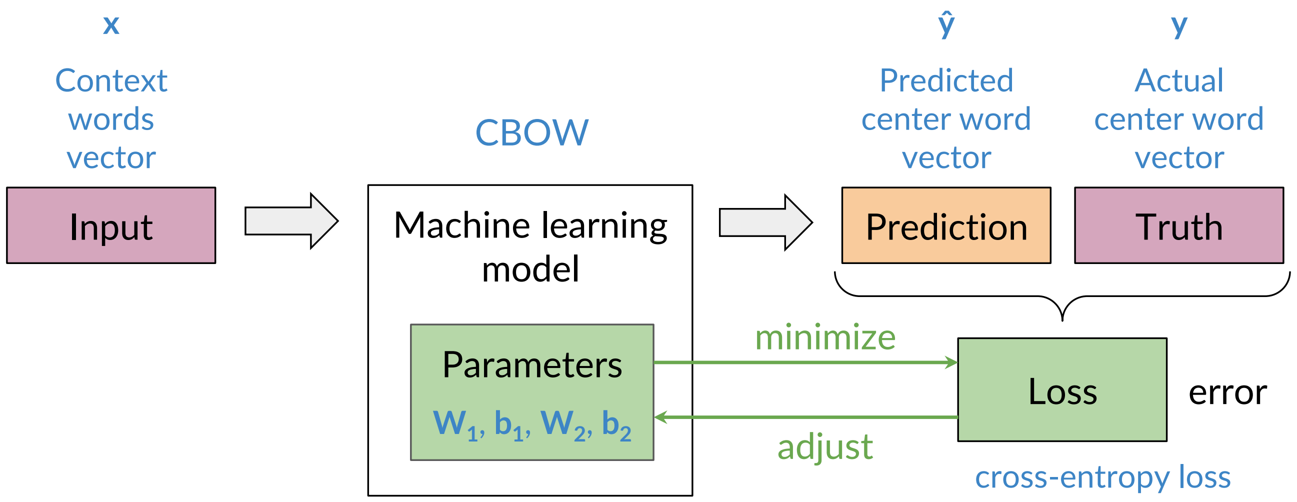
$$\hat{y}_i = \frac{e^{z_i}}{\sum_{j=1}^{V} e^{z_j}}$$

Softmax example

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



Training: Loss



Cross-entropy loss

- Used in classification tasks
  - softmax activations in the output layer

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

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

I am happy because I am learning

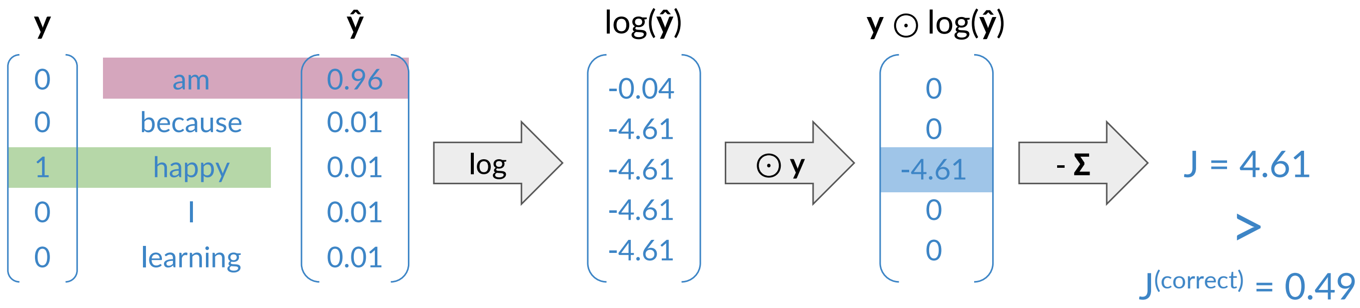
y		ŷ	log(ŷ)	y ⊙ log(ŷ)
0	am	0.083	-2.49	0
0	because	0.03	-3.49	0
1	happy	0.611	-0.49	-0.49
0	I	0.225	-1.49	0
0	learning	0.05	-2.49	0

ŷ → log → log(ŷ) → ⊙ y → y ⊙ log(ŷ) → - Σ → J = 0.49

Cross-Entropy loss

- Used in classification tasks
  - softmax activations in the output layer

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



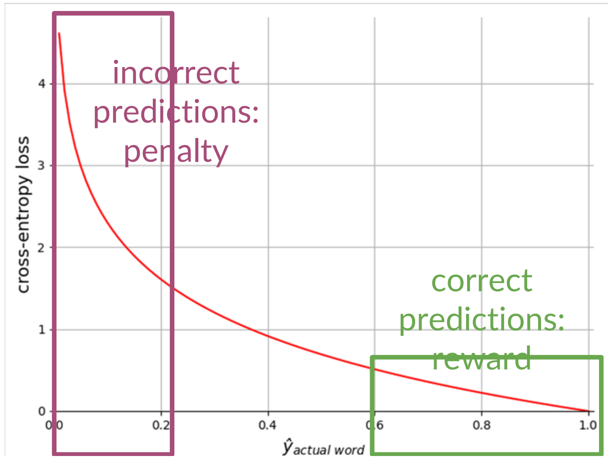
Cross-entropy loss

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

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

y		ŷ
0	am	0.96
0	because	0.01
1	happy	0.01
0	I	0.01
0	learning	0.01

$\rightarrow J = 4.61$

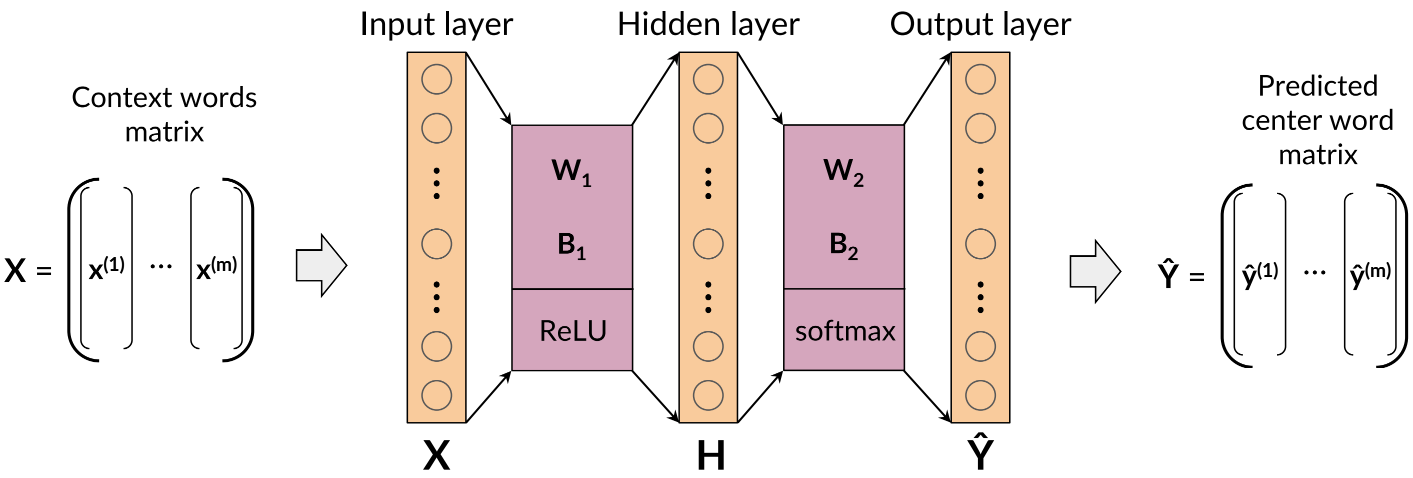


- The loss rewards correct predictions and penalizes the incorrect ones

Training: Forward propagation

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

$$H = \mathrm{ReLU}(Z_1) \qquad \hat{Y} = \mathrm{softmax}(Z_2)$$



Cost

- The cost is referred to as the loss computed on batch examples
  - Mean of cross-entropy losses of the individual examples

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

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

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

Actual center word matrix

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

Learning: Minimizing the cost

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

- Backpropagation
  - Calculate partial derivatives of cos with respect to weights and biases
  - Using the chain rule for derivatives
    - starting from the output layer and working back through the layers

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

  - Gradient descent
    - Update weights and biases

Backpropagation and gradient descent

- To perform gradient descent
  - the partial derivatives of the cost function J are calculated
    - You'll learn the details in the Machine Learning course part II
  - Perform gradient descent with partial derivatives

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

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

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

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

Gradient descent

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

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

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

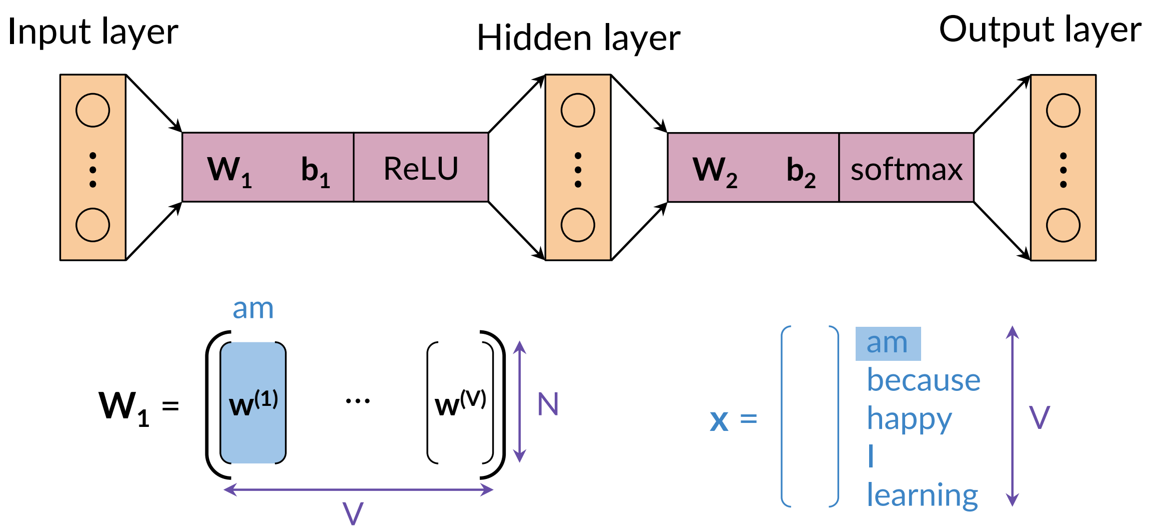
$$\mathbf{b_2} := \mathbf{b_2} - \alpha \frac{\partial J_{batch}}{\partial \mathbf{b_2}}$$

Hyperparameter:

Learning rate α

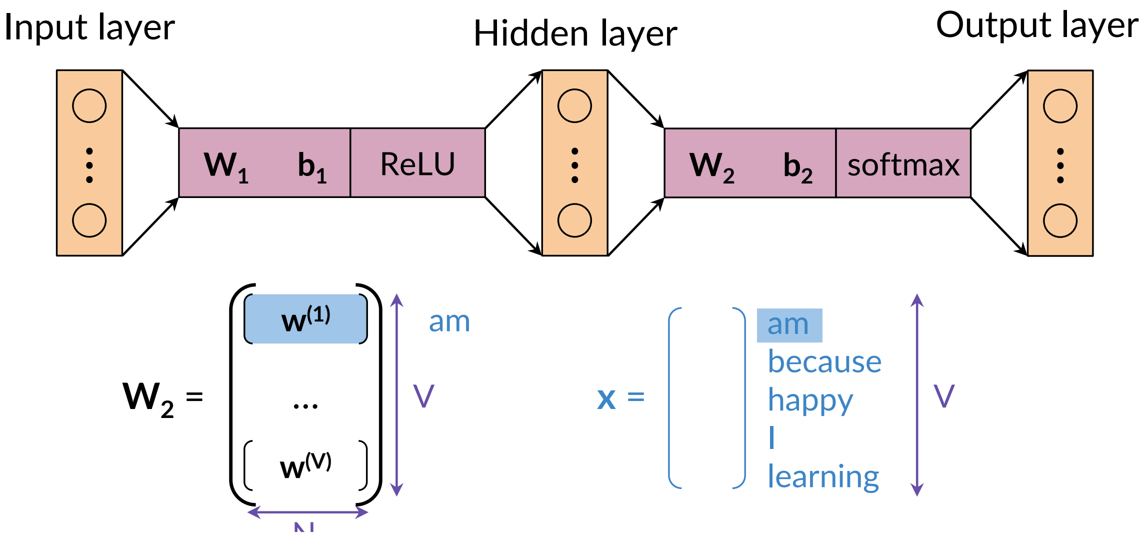
Extracting Word Embedding Vectors

- Once completed the training, one needs to get the word embeddings
  - word embeddings are not directly output by the training process, they are a by-product of the process
- Option 1
  - Column vectors of W_1



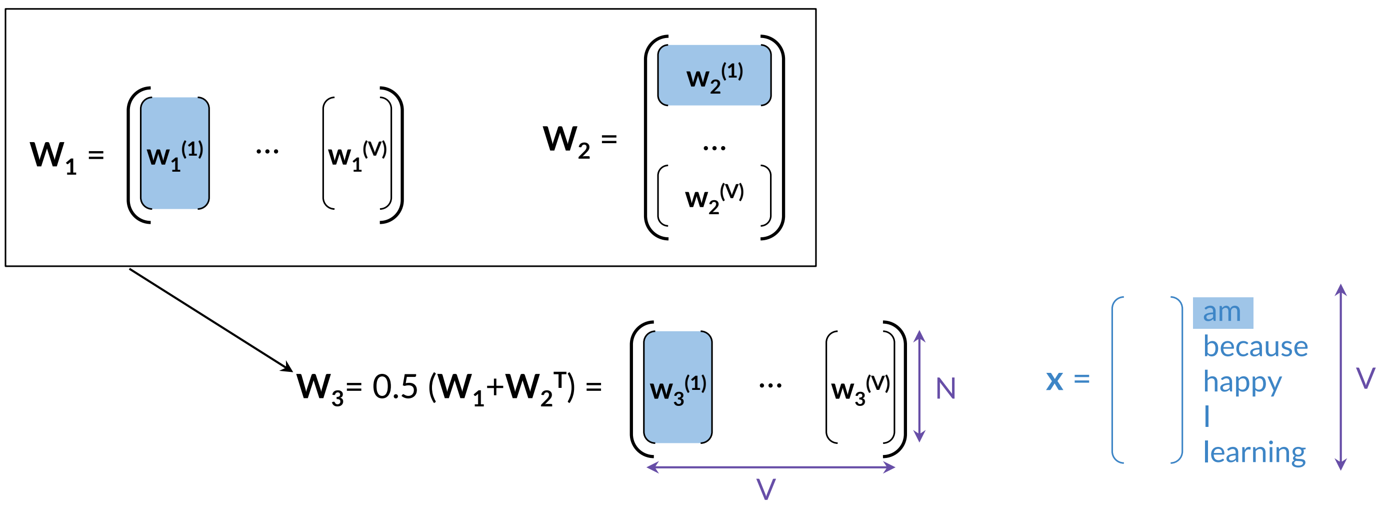
Extracting Word Embedding Vectors

- Option 2
  - Row vectors of W_2



Extracting Word Embedding Vectors

- Option 3
  - Average of the representations from option 1 and option 2



Evaluating Word Embeddings

- Two types of evaluation metrics, intrinsic and extrinsic evaluations
  - Depending on the task
- Intrinsic evaluation
  - Assesses how well the word embeddings capture the semantic (meaning) or syntactic (grammar) relationships between words
    - Analogies

Semantic analogies

"France" is to "Paris" as "Italy" is to <?>

Syntactic analogies

"seen" is to "saw" as "been" is to <?>

- Be aware of possible correct answers

Ambiguity

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

Evaluating Word Embeddings

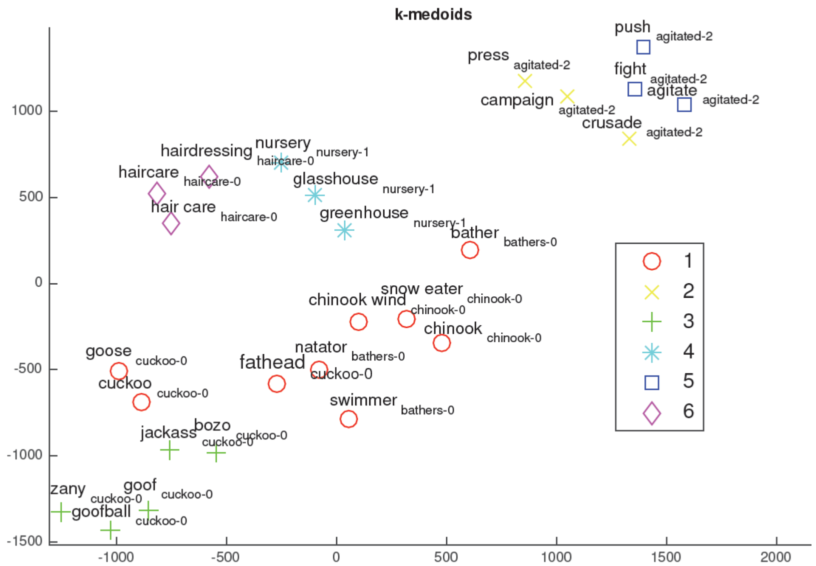
- Intrinsic evaluation
  - Test relationship between words
    - Analogies

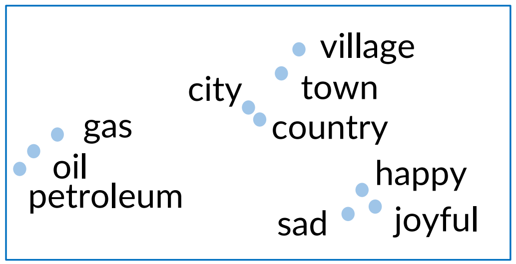
Relationship	Example 1	Example 2	Example 3
France - Paris	Italy: Rome	Japan: Tokyo	Florida: Tallahassee
big - bigger	small: larger	cold: colder	quick: quicker
Miami - Florida	Baltimore: Maryland	Dallas: Texas	Kona: Hawaii
Einstein - scientist	Messi: midfielder	Mozart: violinist	Picasso: painter
Sarkozy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

    - From the original Word2vec paper
      - Word embedding created by a continuous skip-gram model

Evaluating word embeddings

- Test relationships between words
  - Clustering
    - To group similar word embedding vectors (thesaurus)
  - Visualization
    - Human judgment





Evaluating Word Embeddings

- Extrinsic Evaluation
  - Test word embeddings on external tasks, e.g., named entity recognition, part-of-speech tagging
    - Evaluates the actual usefulness of embeddings (+)
    - Time-consuming (-)
    - More difficult to troubleshoot (-)
      - If performing poor, one does not know the specific part of the end-to-end process responsible
  - Example

Named Entity

Antonino Staiano works at UniParthenope

Person Organization

Some properties of word embeddings

- Small windows (C=+/- 2)
  - Nearest words are syntactically similar words in the same taxonomy
    - *Hogwarts* nearest neighbors are other fictional schools
      - *Sunnydale*, *Evernight*, *Blandings*
- Large windows (C = +/- 5)
  - Nearest words are topically related (not similar) words in the same semantic field
    - *Hogwarts* nearest neighbors are *Harry Potter* world:
      - *Dumbledore*, *half-blood*, *Malfoy*

A window onto historical semantics

- Train embeddings on different decades of historical text to see meanings shift

~30 million books, 1850-1990, Google Books data

William L. Hamilton, Jure Leskovec, and Dan Jurafsky. 2016. Diachronic Word Embeddings Reveal Statistical Laws of Semantic Change. Proceedings of ACL.

Embeddings reflect cultural bias

- Ask "Paris : France = Tokyo : x"
  - x = Japan
- Ask "father : doctor = mother : x"
  - x = nurse
- Ask "man : computer programmer = woman : x"
  - x = homemaker
- Algorithms that use embeddings as part of, e.g., hiring search for programmers, might lead to bias in hiring

Bolukbasi, Tolga, Kai-Wei Chang, James Y. Zou, Venkatesh Saligrama, and Adam T. Kalai. "Man is to computer programmer as woman is to homemaker? debiasing word embeddings." In *NeurIPS*, pp. 4349-4357. 2016.

The vector offset method

- Learned word vectors capture meaningful syntactic and semantic regularities
  - Observed as constant vectors offset between pairs of words sharing a particular relationship
- Example
  - Let's denote as w_i the vector for the word i and focus on the singular/plural relation
    - $w_{apple} - w_{apples} \approx w_{car} - w_{cars}$
    - $w_{family} - w_{families} \approx w_{car} - w_{cars}$

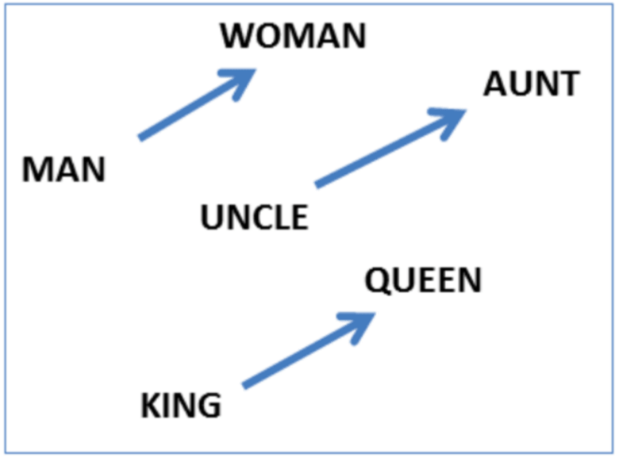
Vector offset

- Syntactic and semantic tasks as analogy questions
- It is assumed that relationships are present as vector offsets
  - That is, in the embedding space, all pairs of words sharing a particular relationship are related by the same constant offset
- To answer the analogy question
  - A is to b as c is to , where di is unknown, we find the embedding vectors w_a, w_b, w_c, and w_d (normalized to unit norm) and compute $y = w_b - w_a + w_c$
  - We search for the word whose embedding vector has the greatest cosine similarity to y

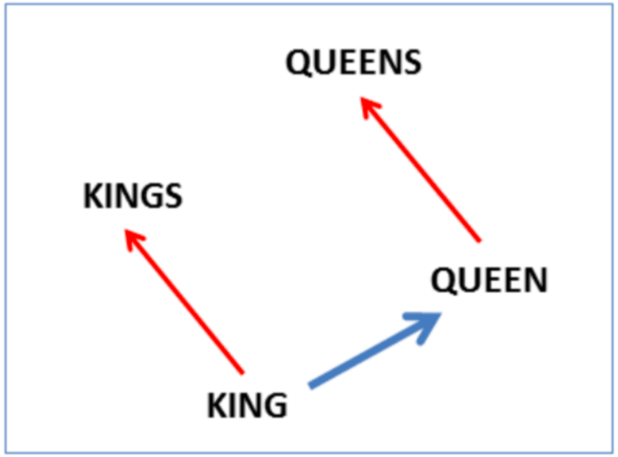
$$x^* = argmax_x \frac{w_x y}{||w_x|| ||y||}$$

Vector offsets

- Semantic: w(*king*) – w(*man*) + w (*woman*) ≈ w(*queen*)
- Syntactic: w(*kings*) – w(*king*) + w(*queen*) ≈ w(*queens*)



Gender relation



Singular/plural relation