Lecture 10

Neural Networks & Word Embeddings

COMP 474/6741, Winter 2021

René Witte
Department of Computer Science
and Software Engineering
Concordia University

Outline

Summary of Chatbot Approaches

Approach	Advantages	Disadvantages
Grammar	Easy to get started Training easy to reuse Modular Easily controlled/restrained	Limited “domain” Capability limited by human effort Difficult to debug Rigid, brittle rules
Grounding	Answers logical questions well Easily controlled/restrained	Sounds artificial, mechanical Difficulty with ambiguity Difficulty with common sense Limited by structured data Requires large scale information extraction Requires human curation
Retrieval	Simple Easy to “train” Can mimic human dialog	Difficult to scale Incoherent personality Ignorant of context Can’t answer factual questions
Generative	New, creative ways of talking Less human effort Domain limited only by data Context aware	Difficult to “steer” Difficult to train Requires more data (dialog) Requires more processing to train

Generative Models

René Witte

Examples

- Generate answers to analogy questions like:

  "Man is to Woman what King is to ___?"

  "Japan is to Sushi what Germany is to ___?"

Today

- Introduction to Neural Networks
- Building word vectors (word embeddings)
- Math with word vectors

→ Worksheet #9: Task 1

Outline

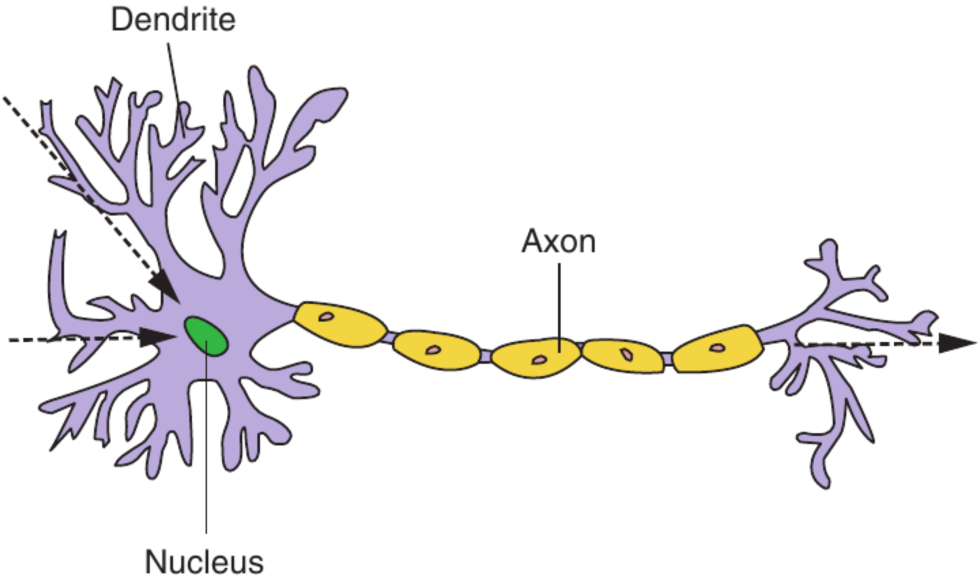
1 Introduction

2 **Neural Networks 101**
- Perceptron
- Backpropagation
- Keras & TensorFlow

3 Word Embeddings

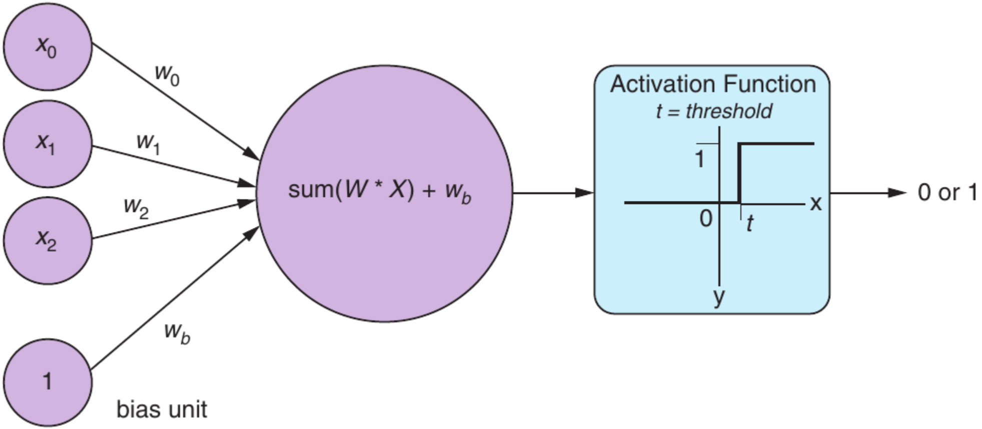
4 Notes and Further Reading

Say hello to one of your neurons



René Witte
Concordia University

Basic Perceptron (Franz Rosenblatt, 1957)



Perceptron Details

Mathematical Perceptron

Input vector:

$$\vec{x} = [x_0, x_1, ..., x_n]$$

Weights vector:

$$\vec{w} = [w_0, w_1, ..., w_n]$$

Dot product:

$$\vec{x} \cdot \vec{w} = \sum_{0}^{n} w_i \cdot x_i$$

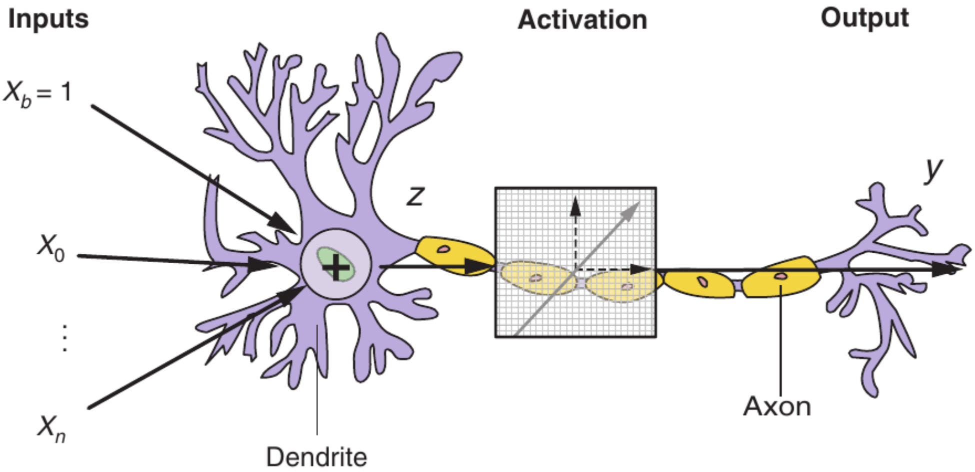
Activation function:

$$f(\vec{x}) = \begin{cases} 1, & \text{if } \vec{x} \cdot \vec{w} \geq \text{threshold} \\ 0, & \text{otherwise} \end{cases}$$

The ’bias’ unit & weight

- Bias: additional input that is always “1”
- Why? Consider the case that all $x_i = 0$, but we need to output 1
- Notation differs in the literature, but idea is always the same

Perceptron vs. Biological Neuron



→ Worksheet #9: Task 2

Perceptron Learning

Learning the weights

Perceptron uses *supervised learning*:

- look at each training sample
- output correct?
  - Yes: don't change any weights
  - No: update the weights that were activated

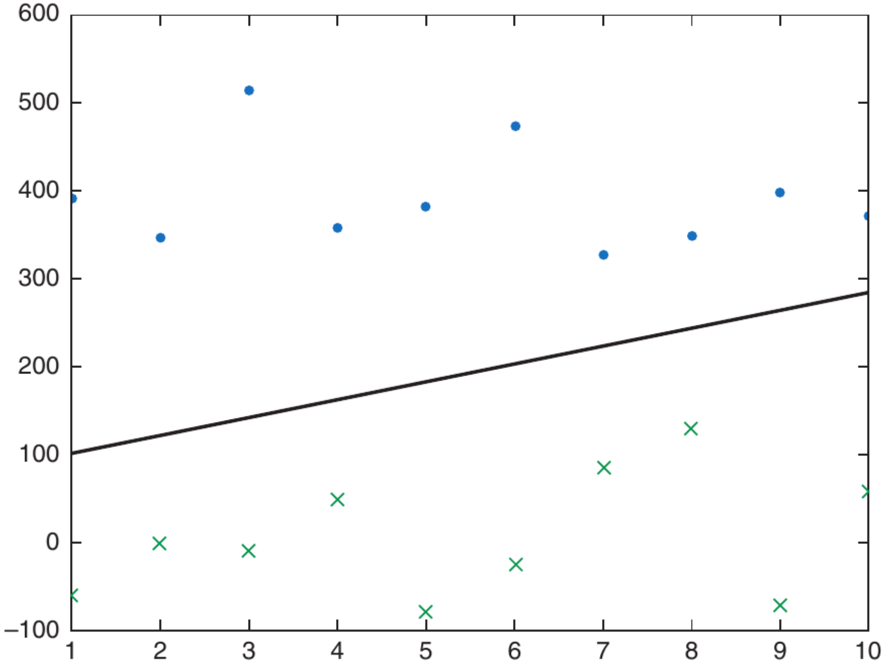
Updating the weights

Based on how much they contributed to the error:

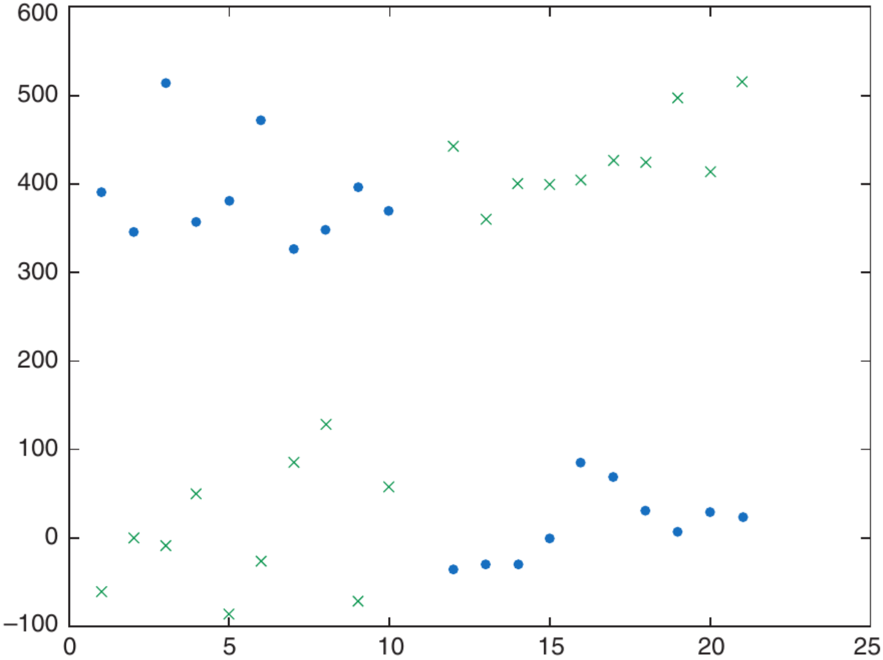
- $w_i' = w_i + \eta \cdot (label - predicted) \cdot x_i$
  (label: training example, predicted: calculated output)
- η is called the learning rate (e.g., $\eta = 0.2$)
- Going through all training examples once is called an epoch

$\rightarrow$ Worksheet #9: Task 3

Linearly Separable Data



Nonlinearly Separable Data

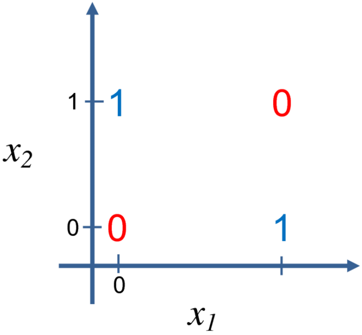


René Witte

Concordia University

What can a *single* Perceptron learn?

- A single Perceptron can learn linearly separable data
- Two dimensions: line, three dimensions: plane, etc.
- It can not learn data that is not linearly separable
- Example: the XOR function



This was pointed out in a famous book by Minksy & Papert in 1969*

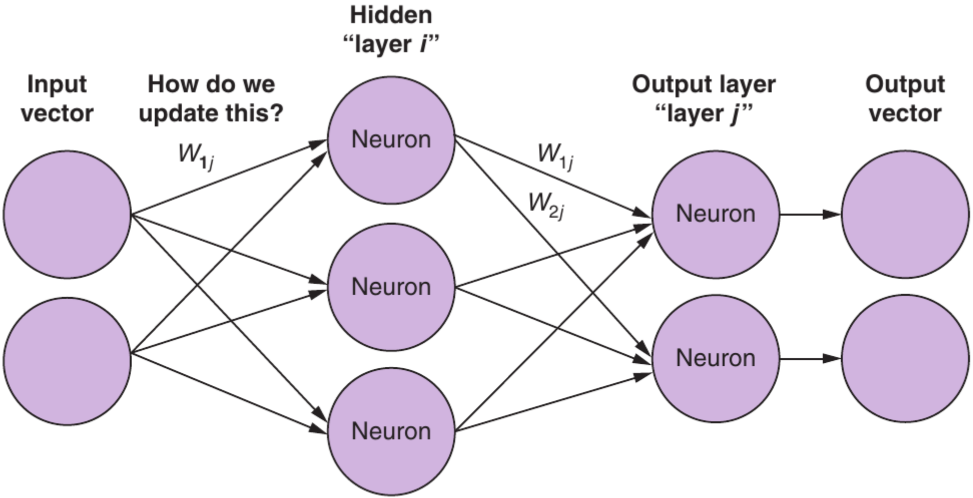
So what, it's useless?

Not quite...so far, we only used a single neuron.

- We can use a network of neurons to also learn non-linearly separable data!

*[Marvin Minsky and Seymour Papert: *Perceptrons: an introduction to computational geometry*, MIT Press, 1969]

Multi-layer neural networks with hidden weights



Training multi-layer neural networks

Backpropagation

- First proposed in 1969, but not used until 1980s because of high computational demands
- Form of supervised learning like Perceptron training
- Basic idea like before: show input, compute output, determine error, and adjust weights to reduce error
- learning is done in two phases
  - first, apply input and propagate forward until output layer is reached
  - then, compute error and propagate backwards, adjusting weights until input layer is reached

Forward step

René Witte

Weighted input

Neurons in backpropagation networks compute the net weighted input like the Perceptron:

$$X = \sum_{i=1}^{n} x_i w_i - \theta$$

Activation function

But here we use a sigmoid activation function

$$Y^{\text{sigmoid}} = \frac{1}{1 + e^{-X}}$$

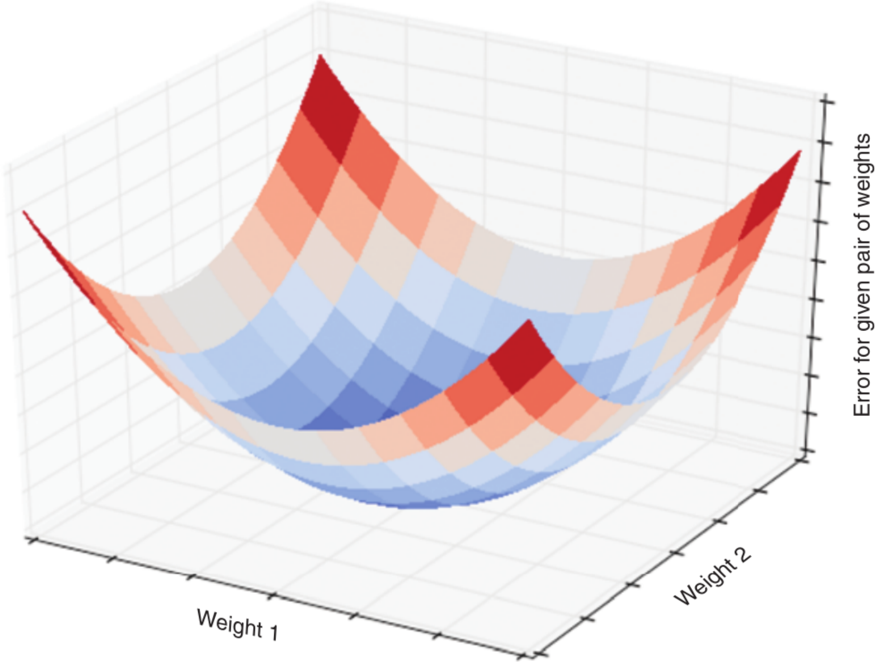
Updating weights

René Witte

Backpropagation rule

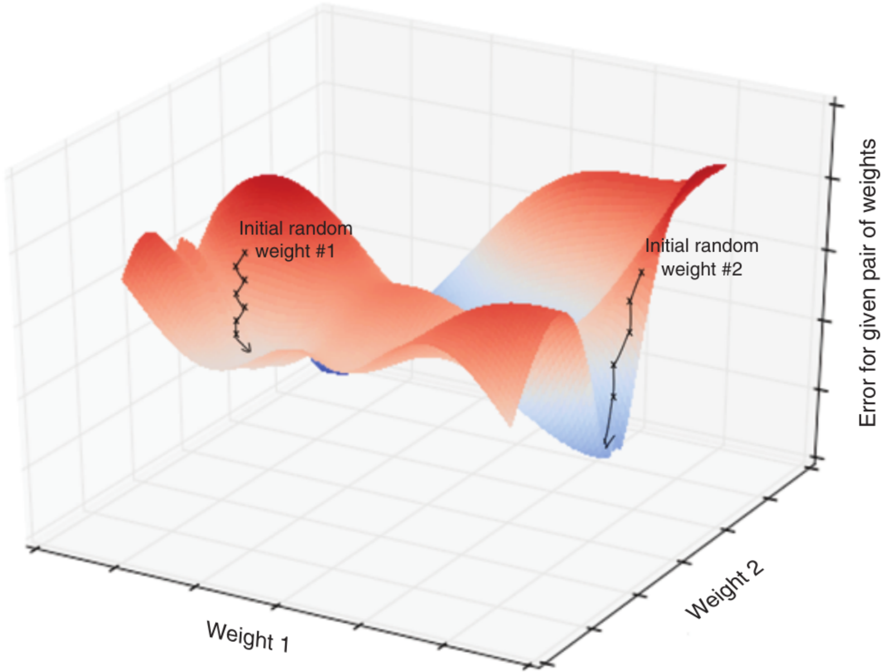
- compute the gradient of the loss function with respect to the weights of the network for a single input–output example
- iterating backwards from output layer to input layer, updating weights
- intuitively: minimize cost function representing the error of the network
- algorithm performs gradient descent to try minimizing the error

Convex Error Curve



René Witte
Concordia University

Nonconvex Error Curve



René Witte

Keras

Simple. Flexible. Powerful.

Get started | Guides | API docs

```python
from tensorflow import keras
from tensorflow.keras import layers

# Instantiate a trained vision model
vision_model = keras.applications.ResNet50()

# This is our video.encoding branch using the trained vision_model
video_input = keras.Input(shape=(100, None, None, 3))
encoded_frame_sequence = layers.TimeDistributed(vision_model)(video_input)
encoded_video = layers.LSTM(256)(encoded_frame_sequence)

# This is our text-processing branch for the question input
question_input = keras.Input(shape=(100,), dtype='int32')
embedded_question = layers.Embedding(10000, 256)(question_input)
encoded_question = layers.LSTM(256)(embedded_question)

# And this is our video question answering model:
merged = keras.layers.concatenate([encoded_video, encoded_question])
output = keras.layers.Dense(1000, activation='softmax')(merged)
video_qa_model = keras.Model(inputs=[video_input, question_input],
                             outputs=output)
```

Deep learning for humans.

Keras is an API designed for human beings, not machines. Keras follows best practices for reducing cognitive load: it offers consistent & simple APIs, it minimizes the number of user actions required for common use cases, and it provides clear & actionable error messages. It also has extensive documentation and developer guides.

Missed TensorFlow Dev Summit? Check out the video playlist. Watch recordings

An end-to-end open source machine learning platform

TensorFlow | For JavaScript | For Mobile & IoT | For Production

The core open source library to help you develop and train ML models. Get started quickly by running Colab notebooks directly in your browser.

Get started with TensorFlow

FROM RESEARCH TO PRODUCTION

An open source machine learning framework that accelerates the path from research prototyping to production deployment.

Install

Introducing PyTorch Profiler - the new and improved performance tool

KEY FEATURES & CAPABILITIES

See all Features

Production Ready

Distributed Training

Robust Ecosystem

Cloud Support

Example Neural Network in Keras


```python
from numpy import loadtxt
from keras.models import Sequential
from keras.layers import Dense
# load the dataset
dataset = loadtxt('pima-indians-diabetes.data.csv', delimiter=',')
# split into input (X) and output (y) variables
X = dataset[:,0:8]
y = dataset[:,8]
# define the keras model
model = Sequential()
model.add(Dense(12, input_dim=8, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
# compile the keras model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
# fit the keras model on the dataset
model.fit(X, y, epochs=150, batch_size=10)
```

Example

- Using the *Pima Indians Diabetes* dataset (predicting the onset of diabetes based on diagnostic measures, like *2-Hour serum insulin (mu U/ml)* and *Diastolic blood pressure (mm Hg)*

See https://machinelearningmastery.com/tutorial-first-neural-network-python-keras/

Output

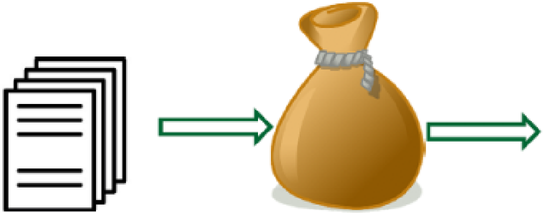

```
Using TensorFlow backend.
Epoch 1/150
768/768 [==============================] - 0s 257us/step - loss: 4.7881 - accuracy: 0.6107
Epoch 2/150
768/768 [==============================] - 0s 87us/step - loss: 0.8344 - accuracy: 0.5964
Epoch 3/150
768/768 [==============================] - 0s 93us/step - loss: 0.7119 - accuracy: 0.6510
Epoch 4/150
768/768 [==============================] - 0s 87us/step - loss: 0.6776 - accuracy: 0.6484
Epoch 5/150
768/768 [==============================] - 0s 87us/step - loss: 0.6315 - accuracy: 0.6888
Epoch 6/150
768/768 [==============================] - 0s 84us/step - loss: 0.6358 - accuracy: 0.6602
Epoch 7/150
768/768 [==============================] - 0s 89us/step - loss: 0.6254 - accuracy: 0.6810
Epoch 8/150
768/768 [==============================] - 0s 85us/step - loss: 0.6086 - accuracy: 0.6615
Epoch 9/150
768/768 [==============================] - 0s 80us/step - loss: 0.6121 - accuracy: 0.6745
Epoch 10/150
768/768 [==============================] - 0s 80us/step - loss: 0.6072 - accuracy: 0.6745

...

Epoch 150/150
768/768 [==============================] - 0s 86us/step - loss: 0.5269 - accuracy: 0.7096
```

Outline

Bag-of-Words (BOW) Model

Task

Turn words into numbers.



Word	Freq.
Mary	2
apples	1
did	2
eat	1
John	1
kill	1
like	1
not	1
to	1

Problems with the Bag-of-Words Model

Mary did kill John.
Mary did not like to eat apples.

John did not kill Mary.
Mary did like to eat apples.

Mary did not like to kill John.
Mary did eat apples.

...

Word	Freq.
Mary	2
apples	1
did	2
eat	1
John	1
kill	1
like	1
not	1
to	1

Word order is ignored

Meaning of the text is lost.

One-Hot Vectors

René Witte

Concordia University

Vector dimensionality $=$ Vocabulary size

With n-dimensional vectors of $\{0, 1\}$, we can represent each word in our vocabulary that has 1 (one) for the word, else 0 (zero).

Example

We can encode the sentence The big dog as a series of three-dimensional vectors:

```
the big dog
 1   0   0
 0   1   0
 0   0   1
```

(a "1" means on, or hot; a "0" means off, or absent.)

Note

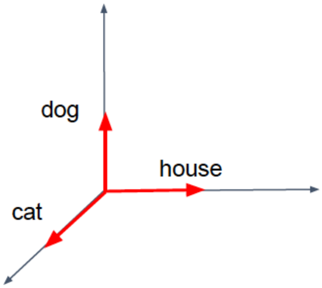
- Unlike in the BOW model, we do not lose information
- Not practical for long documents

The ‘Curse of Dimensionality’

René Witte

Concordia University

‘cat’ = [0, 0, 1]
‘dog’ = [0, 1, 0]
‘house’ = [1, 0, 0]



https://en.wikipedia.org/wiki/Curse_of_dimensionality

→ Worksheet #9: Task 4

Towards better 'word vectors'

Word Vector Requirements

- Dense vectors (smaller dimensions, fewer 0's)
- Capture semantics of words
  - E.g., *Animal*-ness, *Place*-ness, *Action*-ness...
  - The (cosine) distance between "cat" and "dog" should be smaller than between "cat" and "house"
  - Synonyms (e.g., "inflammable" and "flammable") should have nearly identical word vectors

Answer analogy questions

We could then use these vectors for semantic word math, e.g., to answer analogy questions like:

> *"Who is to physics what Louis Pasteur is to germs?"*

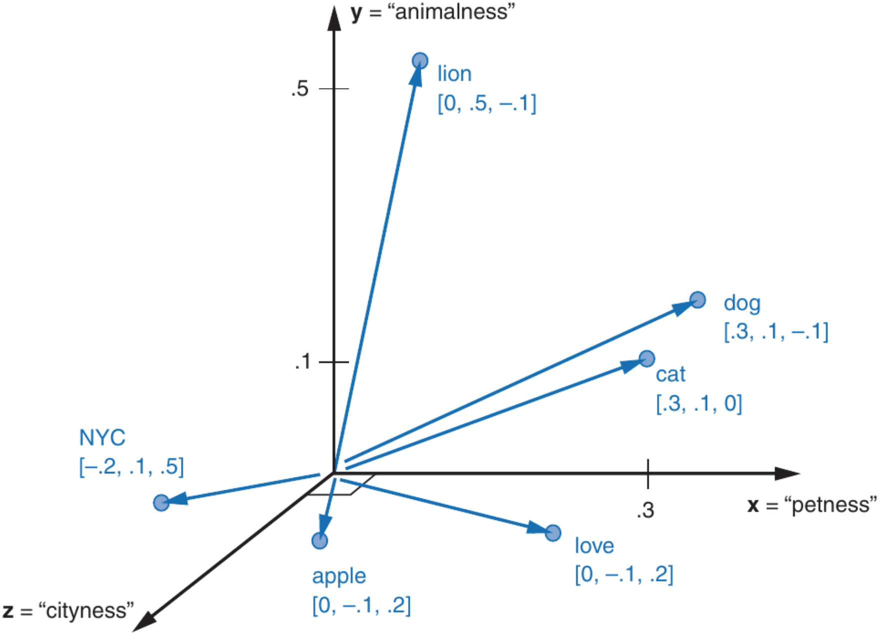
By calculating $\vec{w}$('Louis Pasteur') $-\vec{w}$('germs') $+\vec{w}$('physics')

→ Worksheet #9: Task 5

Hand-crafting Word Vectors (6 words, 3 dimensions)

```
word_vector['cat'] =    .3*topic['petness'] +
                        .1*topic['animalness'] +
                         0*topic['cityness']

word_vector['dog'] =    .3*topic['petness'] +
                        .1*topic['animalness'] -
                        .1*topic['cityness']

word_vector['apple'] =   0*topic['petness'] -
                        .1*topic['animalness'] +
                        .2*topic['cityness']

word_vector['lion'] =    0*topic['petness'] +
                        .5*topic['animalness'] -
                        .1*topic['cityness']

word_vector['NYC'] =   -.2*topic['petness'] +
                        .1*topic['animalness'] +
                        .5*topic['cityness']

word_vector['love'] =   .2*topic['petness'] -
                        .1*topic['animalness'] +
                        .1*topic['cityness']
```

3D vectors for six words about pets and NYC



René Witte

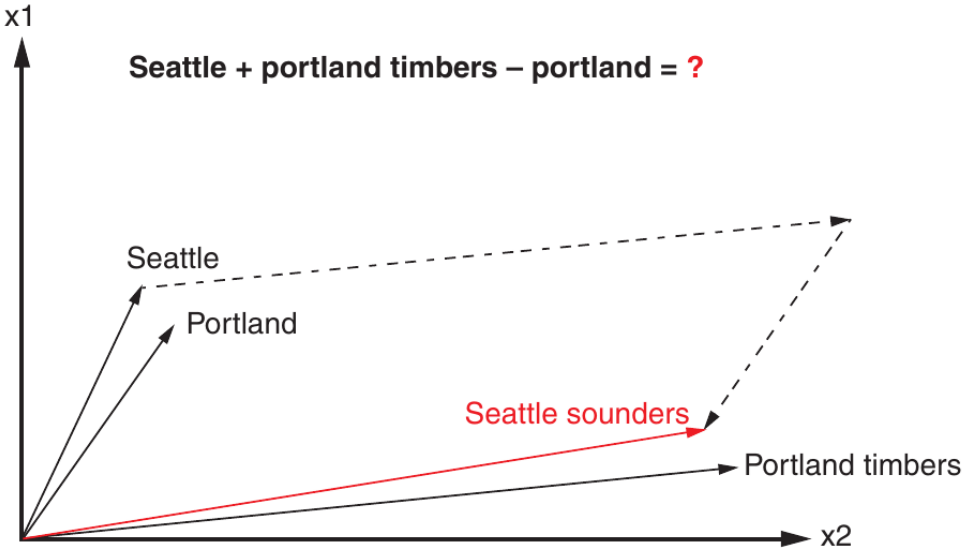
Concordia University

Automatic computation of word vectors

Word2vec

- In 2012, Thomas Mikolov (intern at Microsoft) trained a neural network to predict word occurrences near each target word
- Released in 2013 (then working at Google) as Word2vec
- Word vectors (a.k.a. word embeddings) typically have 100-500 dimensions and are trained on large corpora (e.g., Google's 100 billion words news feed)
- Unsupervised learning (using a so-called autoencoder)

Geometry of Word2vec math



René Witte
Concordia University

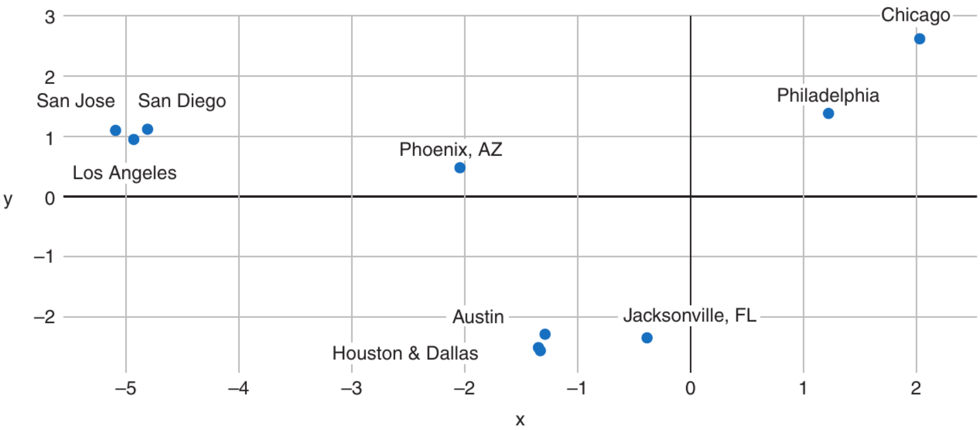
Computing the answer to the soccer team question

$$\begin{bmatrix} 0.0168 \\ 0.007 \\ 0.247 \\ \cdots \end{bmatrix} + \begin{bmatrix} 0.093 \\ -0.028 \\ -0.214 \\ \cdots \end{bmatrix} - \begin{bmatrix} 0.104 \\ 0.0883 \\ -0.318 \\ \cdots \end{bmatrix} = \begin{bmatrix} 0.006 \\ -0.109 \\ 0.352 \\ \cdots \end{bmatrix}$$

Finding word vectors near the result

- Result vector (with 100s of dimensions) is not going to match any other word vector exactly
- Find closest results (e.g., using cosine similarity) for the answer

Word vectors for ten US cities projected onto a 2D map



Training a Word2vec model

Approaches

Skip-gram: predict the context of words (output words) from an input word

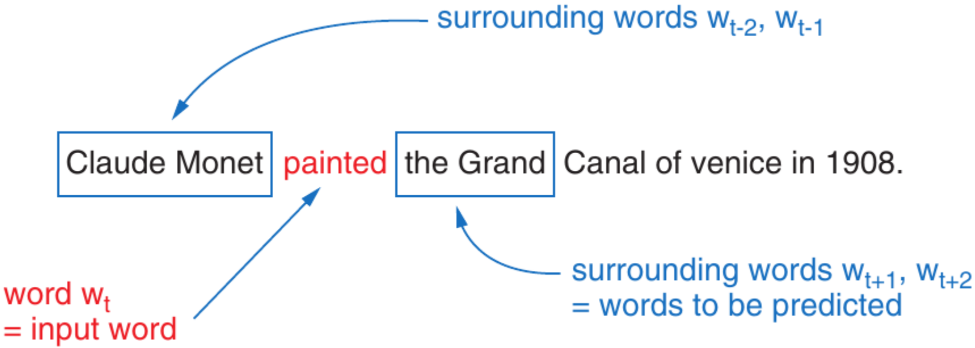
CBOW: *(continuous-bag-of-words)* predicts output word from nearby (input) words

Using a pre-trained model

You can download pre-trained word embeddings for many domains:

- Google's Word2vec model trained on Google News articles
- spaCy comes with word vector models (shown later)
- Facebook's fastText model (for 294 languages)
- Various models trained on medical documents, Harry Potter, LOTR, ...

Training input and output example for the skip-gram approach



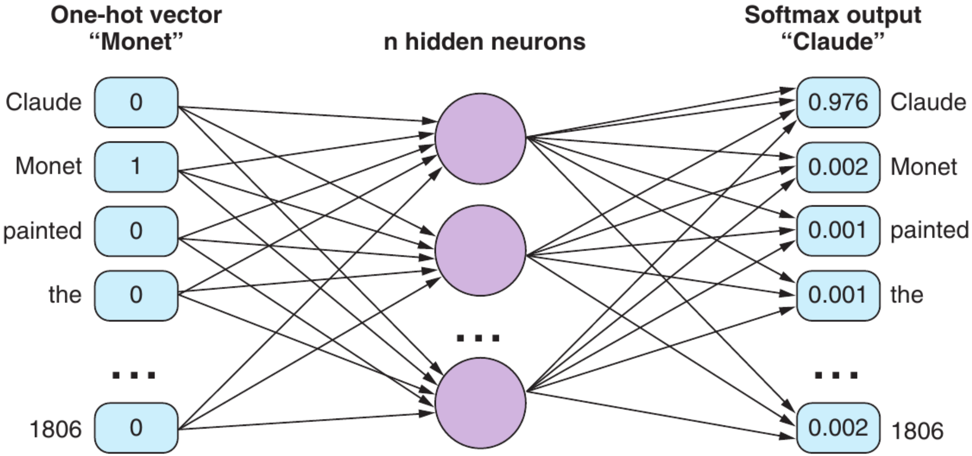
Skip-gram

- Skip-gram is an n-gram with gaps
- Goal: predict surrounding window of words based on input word

Training: Ten 5-grams from the sentence about Monet

Input word w_t	Expected output w_{t-2}	Expected output w_{t-1}	Expected output w_{t+1}	Expected output w_{t+2}
Claude			Monet	painted
Monet		Claude	painted	the
painted	Claude	Monet	the	Grand
the	Monet	painted	Grand	Canal
Grand	painted	the	Canal	of
Canal	the	Grand	of	Venice
of	Grand	Canal	Venice	in
Venice	Canal	of	in	1908
in	of	Venice	1908	
1908	Venice	in		

Neural Network example for the skip-gram training (1/2)



Softmax

Softmax function

The softmax function σ takes as input a vector of K real numbers, and normalizes it into a probability distribution consisting of K probabilities proportional to the exponentials of the input numbers:

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$$

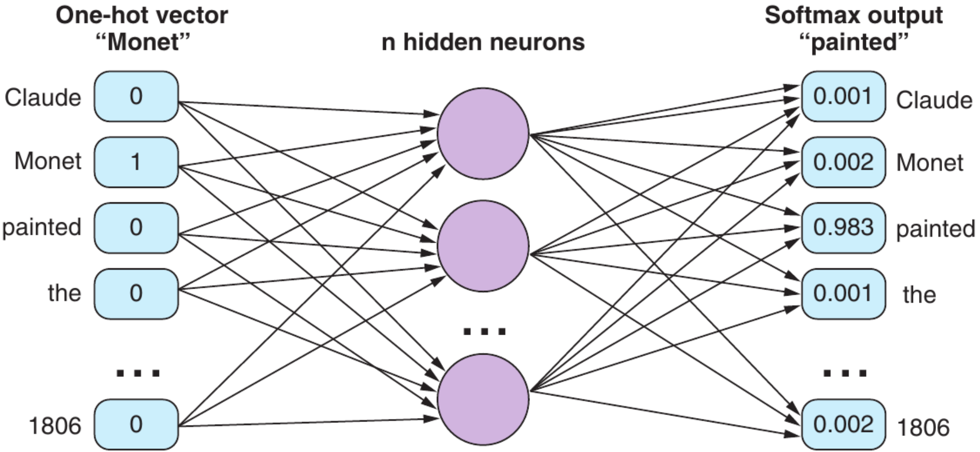
($e = \textit{Euler's}$ number ≈ 2.71828)

Softmax properties

- "normalizes" vector to a [0..1] interval, where all values add up to 1
- often used as activation function in the output layer of a neural network

$\rightarrow$ Worksheet #9: Task 6

Neural Network example for the skip-gram training (2/2)



René Witte

Concordia University

Conversion of one-hot vector to word vector

René Witte

Concordia University

One-hot vector in vocabulary of six words × **Three neuron weight matrix** = **The dot product calculation**

$$
\begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}
\times
\begin{bmatrix} .03 & .92 & .66 \\ .06 & .32 & .61 \\ .14 & .62 & .43 \\ .24 & .99 & .62 \\ .12 & .02 & .44 \\ .32 & .23 & .55 \end{bmatrix}
=
\begin{bmatrix} (0{*}.03) + (1{*}.06) + (0{*}.14) + (0{*}.24) + (0{*}.12) + (0.{*}.32) \\ (0{*}.92) + (1{*}.32) + (0{*}.62) + (0{*}.99) + (0{*}.02) + (0.{*}.23) \\ (0{*}.66) + (1{*}.61) + (0{*}.43) + (0{*}.62) + (0{*}.44) + (0.{*}.55) \end{bmatrix}
=
\begin{bmatrix} .06 \\ .32 \\ .61 \end{bmatrix}
$$

Resulting 3-D word vector

In other words...

Hidden weights are our word vectors

- We're not actually using the neural network we trained
- We're just using the weights as our word embeddings
- (that's a common trick in using neural networks)

Why does this work?

- Two different words that have a similar meaning will have similar context words appearing around them
- So the output vector for these different words have to be similar
- So the neural network has to learn weights for the hidden layer that map these (different) input words to similar output vectors
- So we will get similar word vectors for words that have a different surface form, but similar (or related) semantics

Note

This does not solve the disambiguation problem: there will be one word vector for *"bank"*, including both "river bank" and "financial bank" contexts.

Now what?

René Witte

Now we can do math with word vectors:

$$king - man + woman = queen$$
$$Paris - France + Germany = Berlin$$
$$fish + music = bass$$
$$road - ocean + car = sailboat$$
$$desert - sand + suburbia = driveways$$
$$dorm - students = bachelor\ pad$$
$$barn - cows = garage$$
$$yeti - snow + economics = homo\ economicus$$

See https://graceavery.com/word2vec-fish-music-bass/ for more fun examples

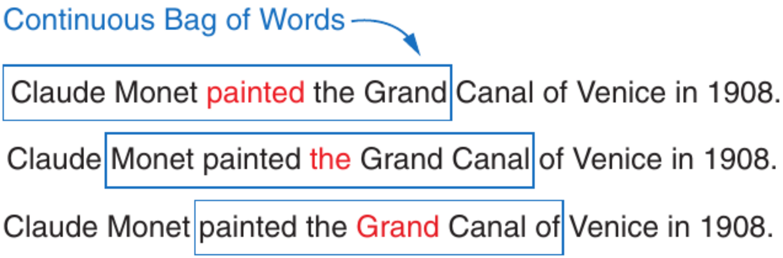
Continuous Bag Of Words (CBOW)

René Witte

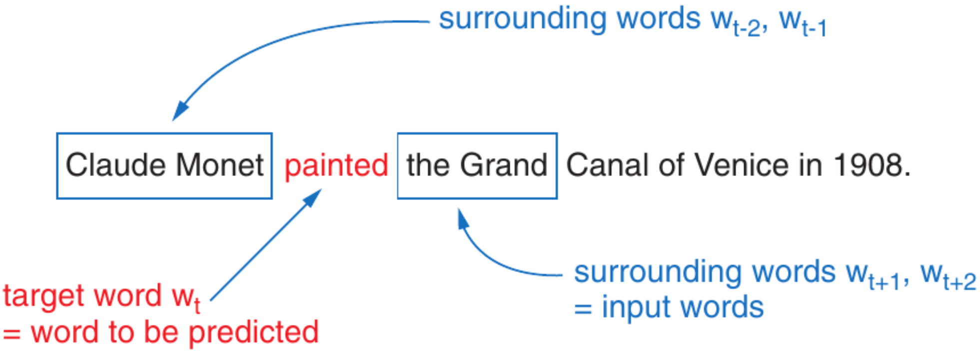
Concordia University

Idea

- Slide a rolling window across a sentence to select the surrounding words for the target word
- All words within the sliding window are considered to be the content of the CBOW



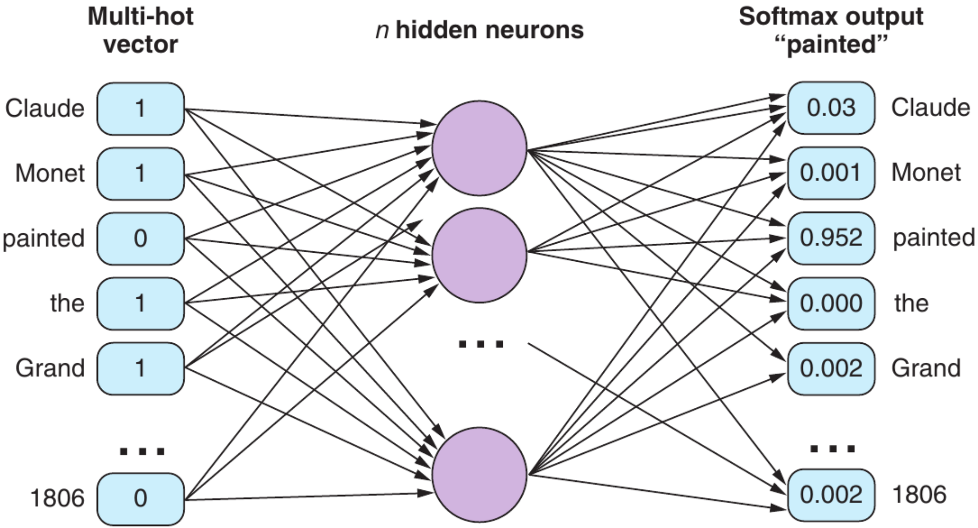
Training input and output example for the CBOW approach



Ten CBOW 5-grams from sentence about Monet

Input word w_{t-2}	Input word w_{t-1}	Input word w_{t+1}	Input word w_{t+2}	Expected output w_t
		Monet	painted	Claude
	Claude	painted	the	Monet
Claude	Monet	the	Grand	painted
Monet	painted	Grand	Canal	the
painted	the	Canal	of	Grand
the	Grand	of	Venice	Canal
Grand	Canal	Venice	in	of
Canal	of	in	1908	Venice
of	Venice	1908		in
Venice	in			1908

CBOW Word2vec network



Which one to use?

René Witte

Pros & Cons

- Skip-gram approach works well with small corpora and rare terms (more training data due to the network structure)
- CBOW shows higher accuracies for frequent words and is faster to train

René Witte

Various Improvements

Frequent Bigrams: Pre-process the corpus and add frequent bigrams as terms (e.g., "New York", "Elvis Presley")

Subsampling: Sample words according to their frequencies (no stop word removal for words like "a", "the") – similar to idf in tf-idf

Negative sampling: To speed up training, don't update all weights, but pick some negative samples to decide which weights to update

Using Word Vectors with spaCy

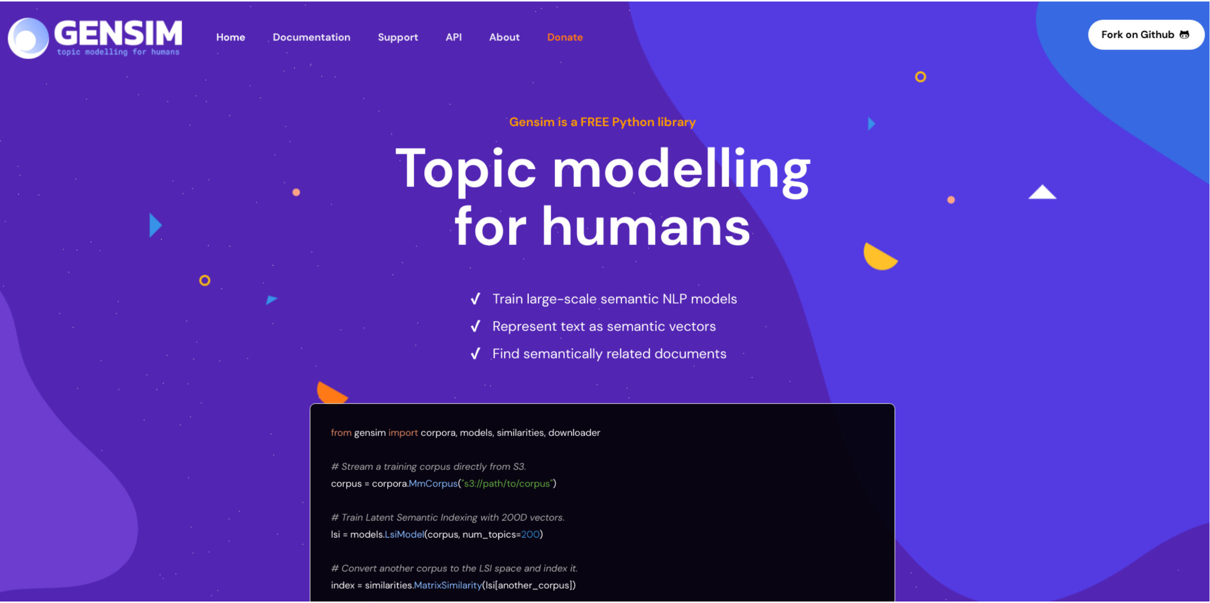
René Witte


```python
import spacy

nlp = spacy.load("en_core_web_lg")  # make sure to use larger model!
tokens = nlp("dog cat banana")

for token1 in tokens:
    for token2 in tokens:
        print(token1.text, token2.text, token1.similarity(token2))
```

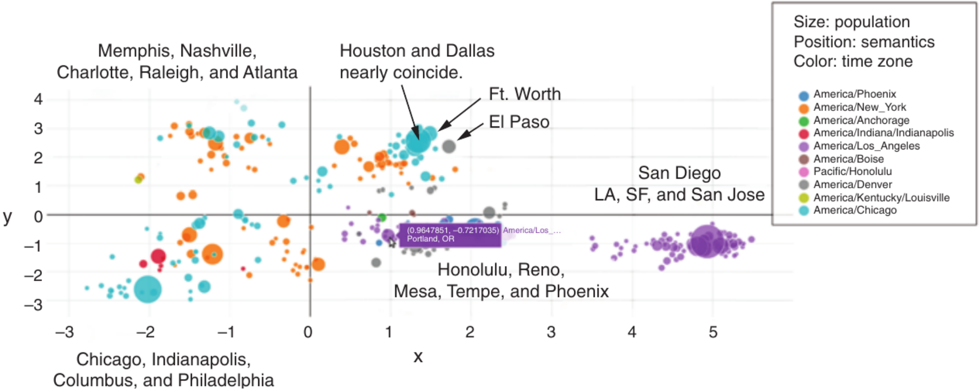
Output

```
dog dog 1.0
dog cat 0.80168545
dog banana 0.24327646
cat dog 0.80168545
cat cat 1.0
cat banana 0.2815437
banana dog 0.24327646
banana cat 0.2815437
banana banana 1.0
```

Training your own Word2vec model using gensim



Google News Word2vec 300-D vectors projected onto a 2D map using PCA



Word vectors can be biased

Example

Your word vectors represent what is in your corpus:

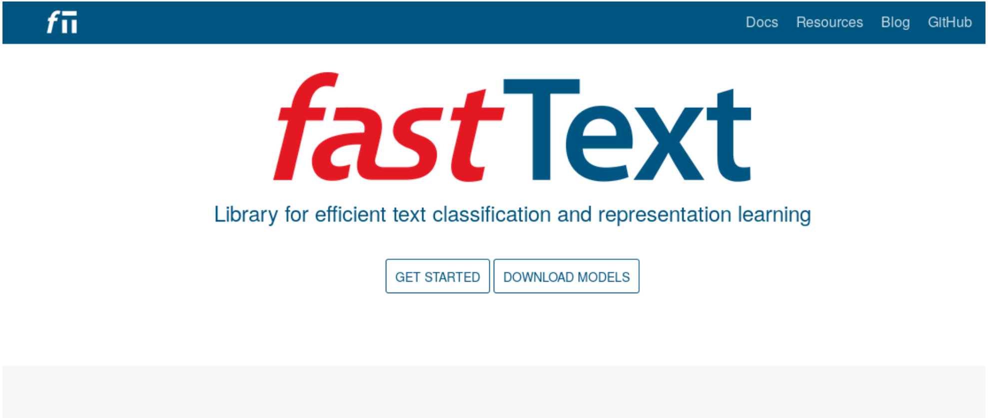
```
>>> word_model.distance('man', 'nurse')
0.7453
>>> word_model.distance('woman', 'nurse')
0.5586
```

So an AI using these word vectors will now have a gender bias!

October 11, 2018

Amazon Scraps Secret AI Recruiting Engine that Showed Biases Against Women

AI Research scientists at Amazon uncovered biases against women on their recruiting machine



fasttext.cc

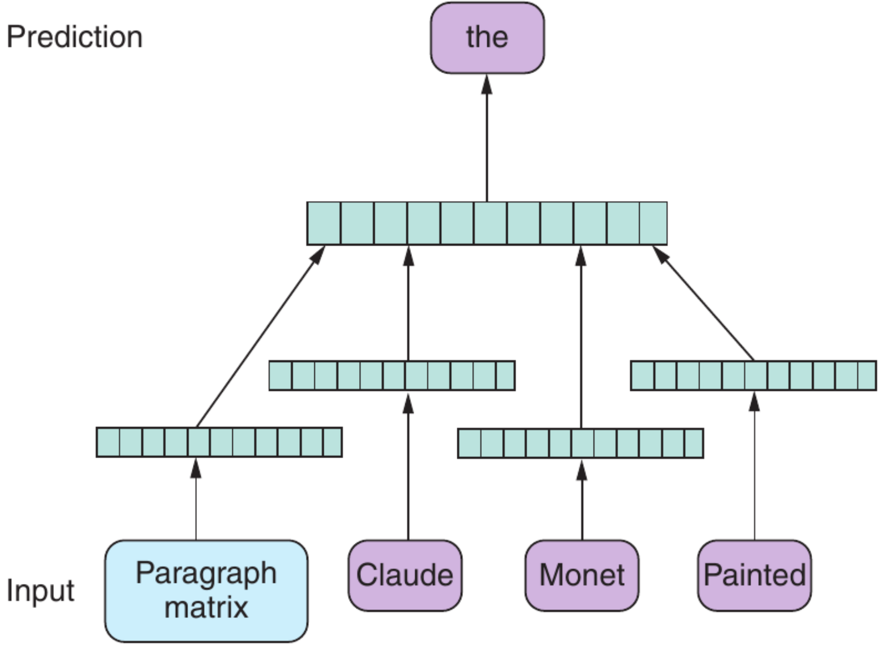
Idea: train on character n-grams, not on word n-grams:

- E.g., for *"whisper"*, we can generate the following 2-grams and 3-grams wh, whi, hi, his, is, isp, sp, spe, pe, per, er
- We can now deal with unseen words, misspelled words, partial words, etc.
- Open source project by Facebook research; pre-trained models for 294 languages from Abkhazian to Zulu

René Witte

Université Concordia University

Doc2vec Training



Outline

1. Introduction
2. Neural Networks 101
3. Word Embeddings
4. **Notes and Further Reading**

Reading Material

Required

- [LHH19, Chapters 5, 6] (Neural Networks, Word Vectors)

References

[LHH19] Hobson Lane, Cole Howard, and Hannes Max Hapke.
Natural Language Processing in Action.
Manning Publications Co., 2019.
https://concordiauniversity.on.worldcat.org/oclc/1102387045.