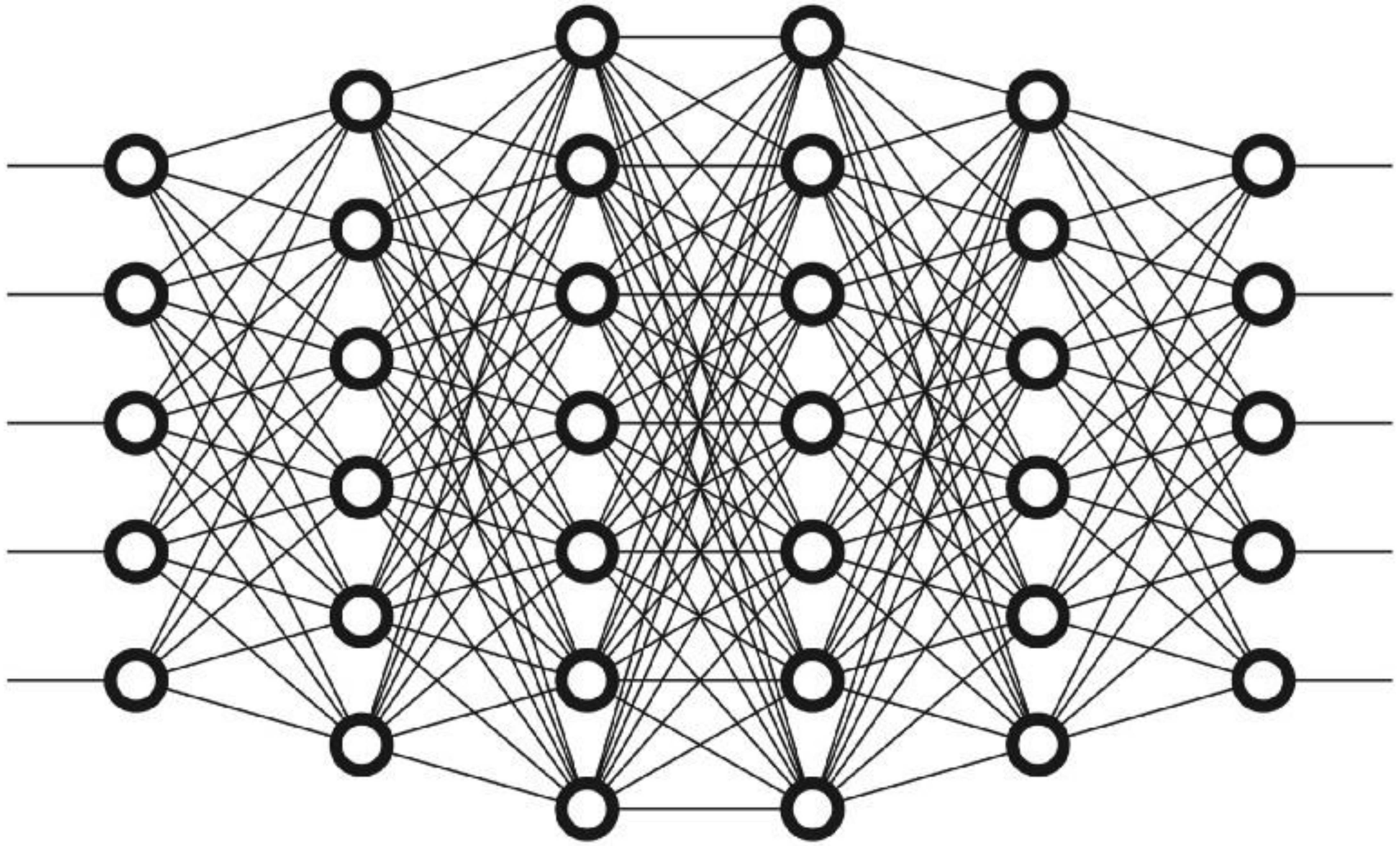
National Taiwan University of Science and Technology
Computer Science and Information Engineering

TEXT MINING TUTORIAL #7

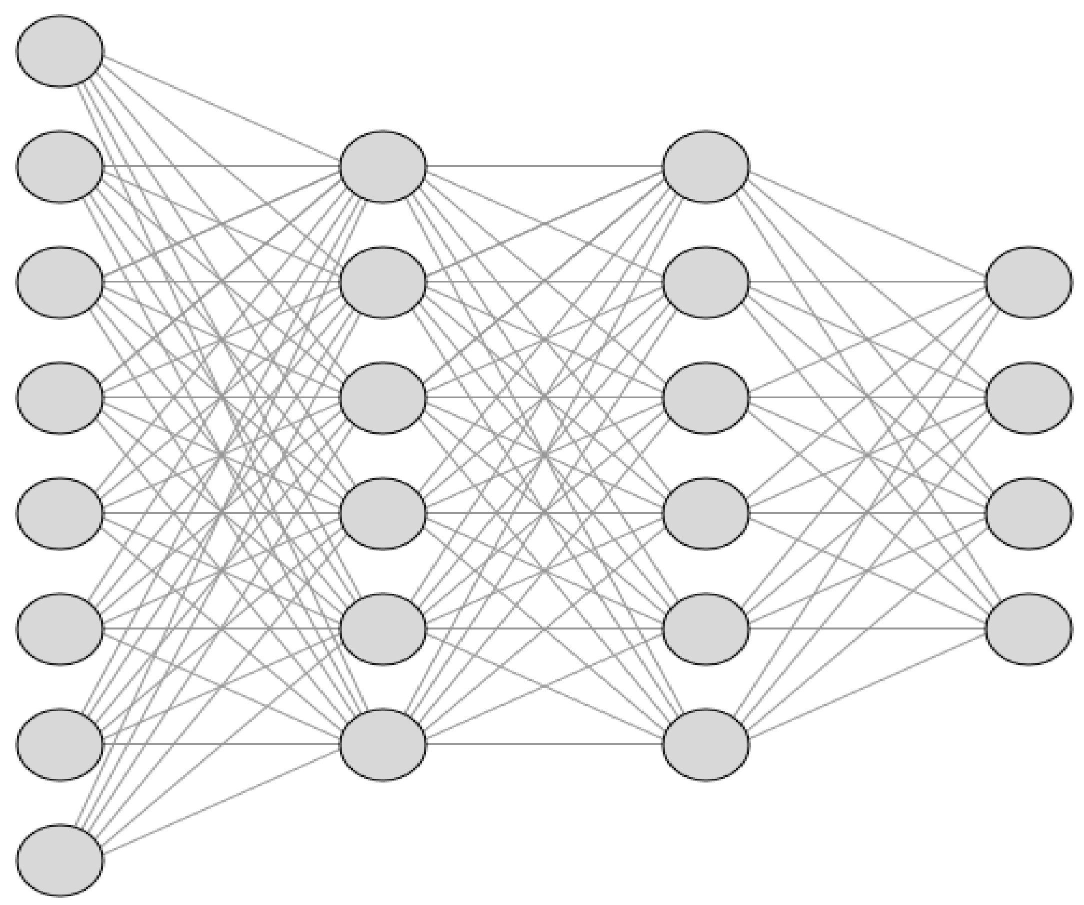


Instructor: Prof. Hsing-Kuo Pao
TA: Zolnamar Dorjsembe (Zola)

Date: Apr 9, 2024

Learning Objectives:
What's Ahead on Our Agenda for the Rest of this Semester

- Code tutorial: One-hot encoding & Word embedding
- ANN
- RNN & LSTM
- Transformers
- Large Language Models (Bert, GPT-4)
- Natural Language Generation
- Visualization, Analysis and Interpretability Basics



PRELIMINARY DISCUSSION

needto
know

Brief Feedback Survey

https://forms.gle/YgSUfuCr1WCTnTwj8

LEARN PYTHON & PYTORCH

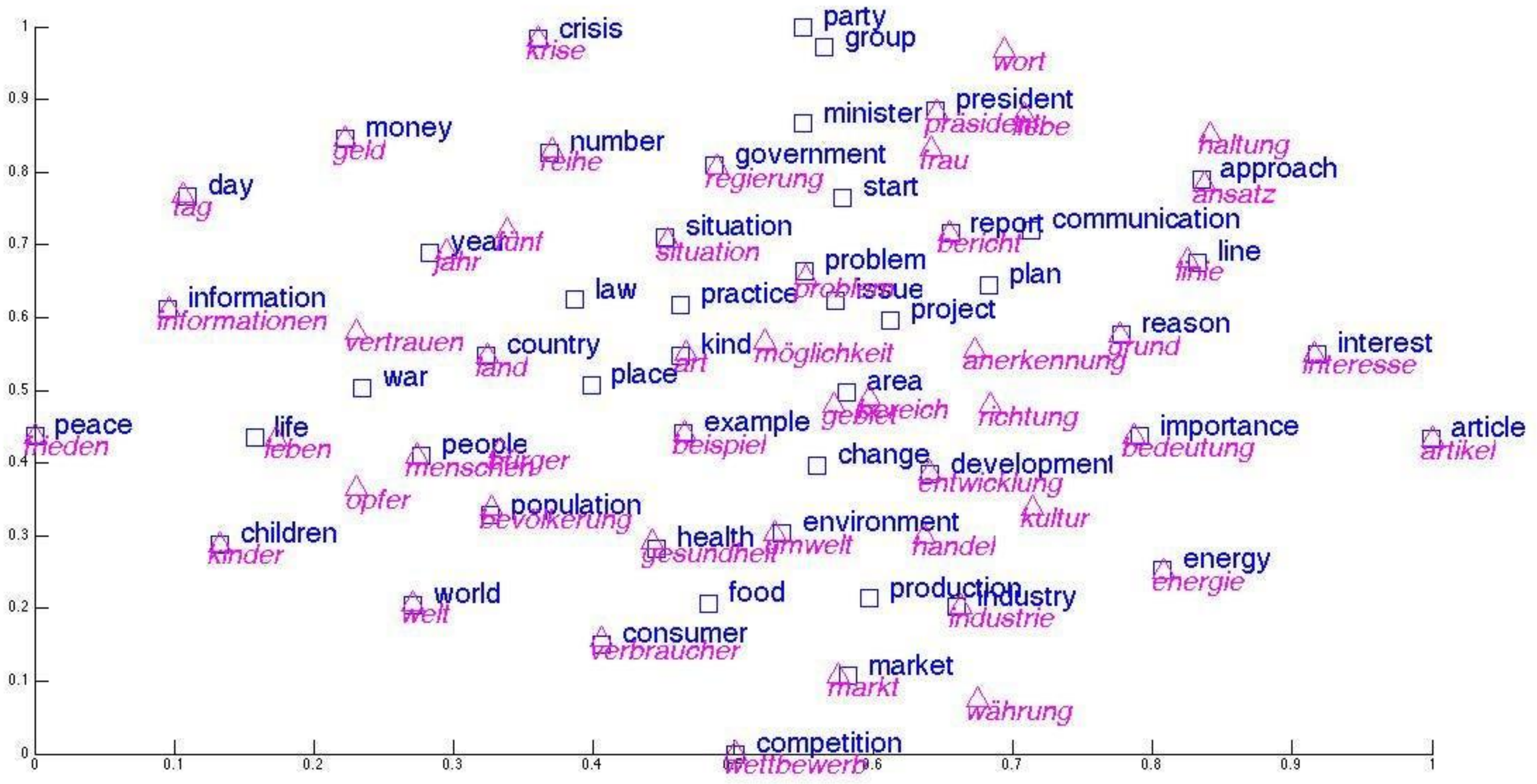
- **Learn Python in 8 hours:** https://www.youtube.com/watch?v=ld9z_gQpPho
- **Learn Pytorch in 4 hours:** https://www.youtube.com/watch?v=c36lUUr864M&t=922s
- **PyTorch for Deep Learning & Machine Learning – Full Course:** https://www.youtube.com/watch?v=V_xro1bcAuA

ONE HOT ENCODING & WORD EMBEDDING

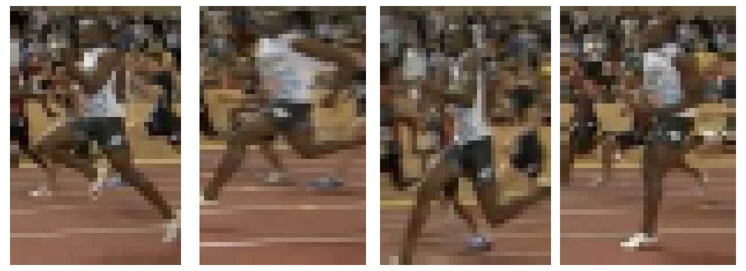
Color
Red
Green
Blue

One-hot encoding →

d1	d2	d3
1	0	0
0	1	0
0	0	1



SEQUENCE DATA

Speech recognition		→	"The quick brown fox jumped over the lazy dog."
Music generation	∅	→	
Sentiment classification	"There is nothing to like in this movie."	→	★☆☆☆☆
DNA sequence analysis	AGCCCCTGTGAGGAACTAG	→	AGCCCCTGTGAGGAACTAG
Machine translation	Voulez-vous chanter avec moi?	→	Do you want to sing with me?
Video activity recognition		→	Running
Name entity recognition	Yesterday, Harry Potter met Hermione Granger.	→	Yesterday, Harry Potter met Hermione Granger.

(Source: towardsdatascience.com)

TEXT DATA

Text is among the most common forms of **sequence data**

Text mining

30 languages

Article Talk Read Edit View history Tools

From Wikipedia, the free encyclopedia

Text mining, **text data mining** (**TDM**) or **text analytics** is the process of deriving high-quality information from text. It involves "the discovery by computer of new, previously unknown information, by automatically extracting information from different written resources."[1] Written resources may include websites, books, emails, reviews, and articles. High-quality information is typically obtained by devising patterns and trends by means such as statistical pattern learning. According to Hotho et al. (2005) we can distinguish between three different perspectives of text mining: information extraction, data mining, and a knowledge discovery in databases (KDD) process.[2] Text mining usually involves the process of structuring the input text (usually parsing, along with the addition of some derived linguistic features and the removal of others, and subsequent insertion into a database), deriving patterns within the structured data, and finally evaluation and interpretation of the output. 'High quality' in text mining usually refers to some combination of relevance, novelty, and interest. Typical text mining tasks include text categorization, text clustering, concept/entity extraction, production of granular taxonomies, sentiment analysis, document summarization, and entity relation modeling (*i.e.*, learning relations between named entities).

Text analysis involves information retrieval, lexical analysis to study word frequency distributions, pattern recognition, tagging/annotation, information extraction, data mining techniques including link and association analysis, visualization, and predictive analytics. The overarching goal is, essentially, to turn text into data for analysis, via the application of natural language processing (NLP), different types of algorithms and analytical methods. An important phase of this process is the interpretation of the gathered information.

HOW TO TRANSFORM TEXT TO NUMBERS?
What are some possible ways to accomplish this?

HOW TO TRANSFORM TEXT INTO NUMBERS: One-Hot Encoding = 1-of-N encoding

The most straightforward way to encode a word

I	ate	an	apple	and	played	the	piano
1	**2**	**3**	**4**	**5**	**6**	**7**	**8**

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**
I	1	0	0	0	0	0	0	0
ate	0	1	0	0	0	0	0	0
an	0	0	1	0	0	0	0	0
apple	0	0	0	1	0	0	0	0
and	0	0	0	0	1	0	0	0
played	0	0	0	0	0	1	0	0
the	0	0	0	0	0	0	1	0
piano	0	0	0	0	0	0	0	1

Dimension: 8x8
64 units of memory space

ONE-HOT ENCODING: CODE IMPLEMENTATION (SIMPLE)

```python
import numpy as np
text = "I like to learn new things".lower().split()
orderedSet = sorted(set(text))
print("Ordered set: ", orderedSet)

def get_vec(len_text, word):
    empty_vector = [0] * len_text
    vect = 0
    find = np.where( np.array(orderedSet) == word)[0][0]
    empty_vector[find] = 1
    return empty_vector

def get_matrix(orderedSet):
    one_hot = []
    len_text = len(orderedSet)
    for i in text:
        vec = get_vec(len_text,i)
        one_hot.append(vec)

    return np.asarray(one_hot)

print ("\nTHE RESULT OF ONE-HOT ENCODING:")
print (get_matrix(orderedSet))
```

```
Ordered set:  ['i', 'learn', 'like', 'new', 'things', 'to']

THE RESULT OF ONE-HOT ENCODING:
[[1 0 0 0 0 0]
 [0 0 1 0 0 0]
 [0 0 0 0 0 1]
 [0 1 0 0 0 0]
 [0 0 0 1 0 0]
 [0 0 0 0 1 0]]
```

- The set() function creates a set object
- numpy.where() function returns the indices of elements in an input array where the given condition is satisfied.

ONE-HOT ENCODING: CODE IMPLEMENTATION (SIMPLE)

```
import numpy as np
text = "<h1>Here is another example</h1>, <b>here</b> is<br/> <i>another example</i>.".lower().split()
orderedSet = sorted(set(text))
print("Ordered set: ", orderedSet)

def get_vec(len_text, word):
    empty_vector = [0] * len_text
    vect = 0
    find = np.where( np.array(orderedSet) == word)[0][0]
    empty_vector[find] = 1
    return empty_vector

def get_matrix(orderedSet):
    one_hot = []
    len_text = len(orderedSet)
    for i in text:
        vec = get_vec(len_text,i)
        one_hot.append(vec)

    return np.asarray(one_hot)

print ("\nTHE RESULT OF ONE-HOT ENCODING:")
print (get_matrix(orderedSet))
```

ONE-HOT ENCODING: CODE IMPLEMENTATION (SIMPLE)

```
import numpy as np
text = "<h1>Here is another example</h1>, <b>here</b> is<br/> <i>another example</i>.".lower().split()
orderedSet = sorted(set(text))
print("Ordered set: ", orderedSet)

def get_vec(len_text, word):
    empty_vector = [0] * len_text
    vect = 0
```

```
Ordered set:  ['<b>here</b>', '<h1>here', '<i>another', 'another', 'example</h1>,', 'example</i>.', 'is', 'is<br/>']

THE RESULT OF ONE-HOT ENCODING:
[[0 1 0 0 0 0 0 0]
 [0 0 0 0 0 0 1 0]
 [0 0 0 1 0 0 0 0]
 [0 0 0 0 1 0 0 0]
 [1 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 1]
 [0 0 1 0 0 0 0 0]
 [0 0 0 0 0 1 0 0]]
```

```
print ("\nTHE RESULT OF ONE-HOT ENCODING:")
print (get_matrix(orderedSet))
```

ONE-HOT ENCODING: CODE IMPLEMENTATION (SIMPLE)

```python
import numpy as np
import re

def cleanText(text):
    clean = re.compile('<.*?>')
    text = re.sub(clean, '', text)
    return re.sub('\W+',' ', text)

text = "<h1>Here is another example</h1>, <b>here</b> is<br/> <i>another example</i>."
text = cleanText(text)
print("Cleaned text: ", text)
text = text.lower().split()
print("Split result: ", text)
orderedSet = sorted(set(text))
print("Ordered set: ", orderedSet)

def get_vec(len_text, word):
    empty_vector = [0] * len_text
    vect = 0
    find = np.where( np.array(orderedSet) == word)[0][0]
    empty_vector[find] = 1
    return empty_vector

def get_matrix(orderedSet):
    one_hot = []
    len_text = len(orderedSet)
    for i in text:
        vec = get_vec(len_text,i)
        one_hot.append(vec)

    return np.asarray(one_hot)

print ("\nTHE RESULT OF ONE-HOT ENCODING:")
print (get_matrix(orderedSet))
```

```
Cleaned text:  Here is another example here is another example
Split result:  ['here', 'is', 'another', 'example', 'here', 'is', 'another', 'example']
Ordered set:  ['another', 'example', 'here', 'is']

THE RESULT OF ONE-HOT ENCODING:
[[0 0 1 0]
 [0 0 0 1]
 [1 0 0 0]
 [0 1 0 0]
 [0 0 1 0]
 [0 0 0 1]
 [1 0 0 0]
 [0 1 0 0]]
```

WHAT DO YOU THINK ARE THE MAIN ISSUES WITH USING ONE-HOT ENCODING?

HOW TO TRANSFORM TEXT INTO NUMBERS: Problems with One-Hot Encoding

Let's assume: lexicon = {monkey, ape, banana}

monkey → [1, 0, 0]
ape → [0, 1, 0]
banana → [0, 0, 1]

1. Hard to extract meanings

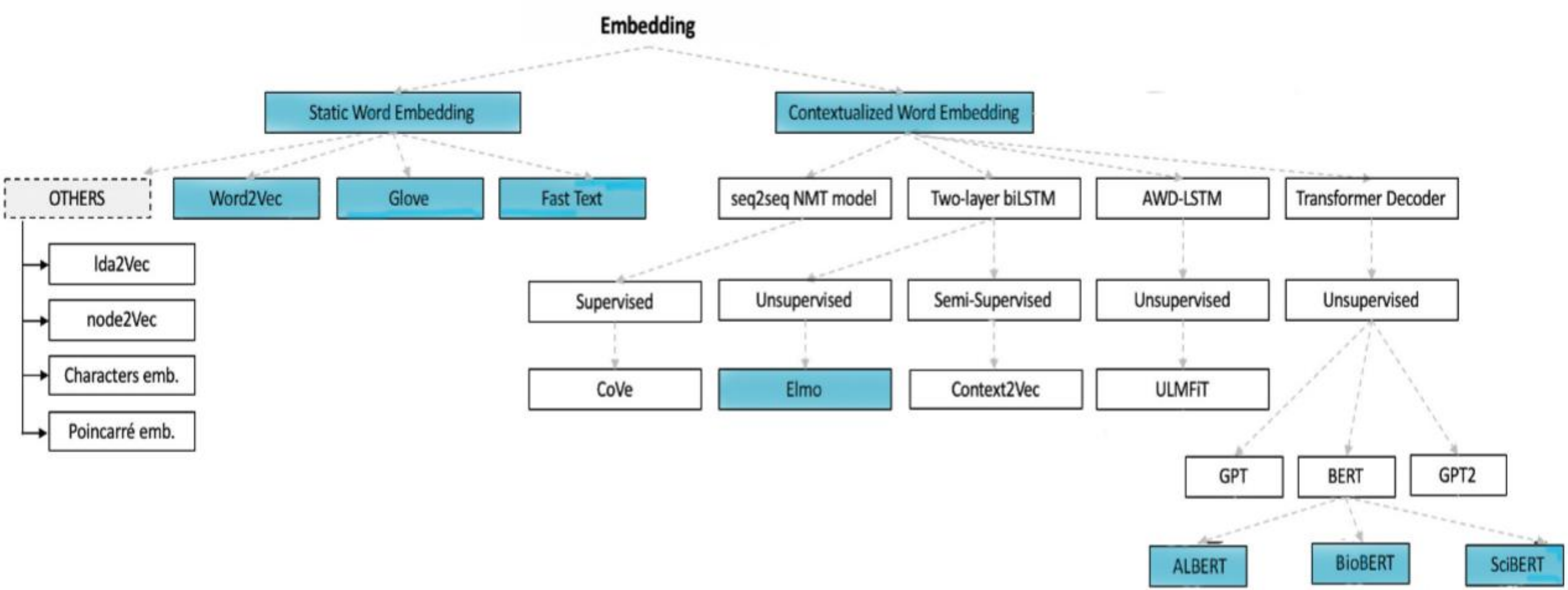
If: Entire vocabulary set has 50k words

2. High dimensions

Word #i → $[\underbrace{0, 0, 0, \ldots, 0, 0, \overbrace{1}^{i}, 0, \ldots}_{50k}]$

50k squared = 2.5 billion units of memory space

WORD EMBEDDING



(Qudar & Mago, 2020)

Word2Vec

Efficient Estimation of Word Representations in Vector Space

Tomas Mikolov
Google Inc., Mountain View, CA
tmikolov@google.com

Kai Chen
Google Inc., Mountain View, CA
kaichen@google.com

Greg Corrado
Google Inc., Mountain View, CA
gcorrado@google.com

Jeffrey Dean
Google Inc., Mountain View, CA
jeff@google.com

Abstract

We propose two novel model architectures for computing continuous vector representations of words from very large data sets. The quality of these representations is measured in a word similarity task, and the results are compared to the previously best performing techniques based on different types of neural networks. We observe large improvements in accuracy at much lower computational cost, i.e. it takes less than a day to learn high quality word vectors from a 1.6 billion words data set. Furthermore, we show that these vectors provide state-of-the-art performance on our test set for measuring syntactic and semantic word similarities.

Word2vec is a technique for natural language processing (NLP) published in 2013.

The word2vec algorithm uses a neural network model to learn word associations from a large corpus of text.

Word2Vec (cont'd)

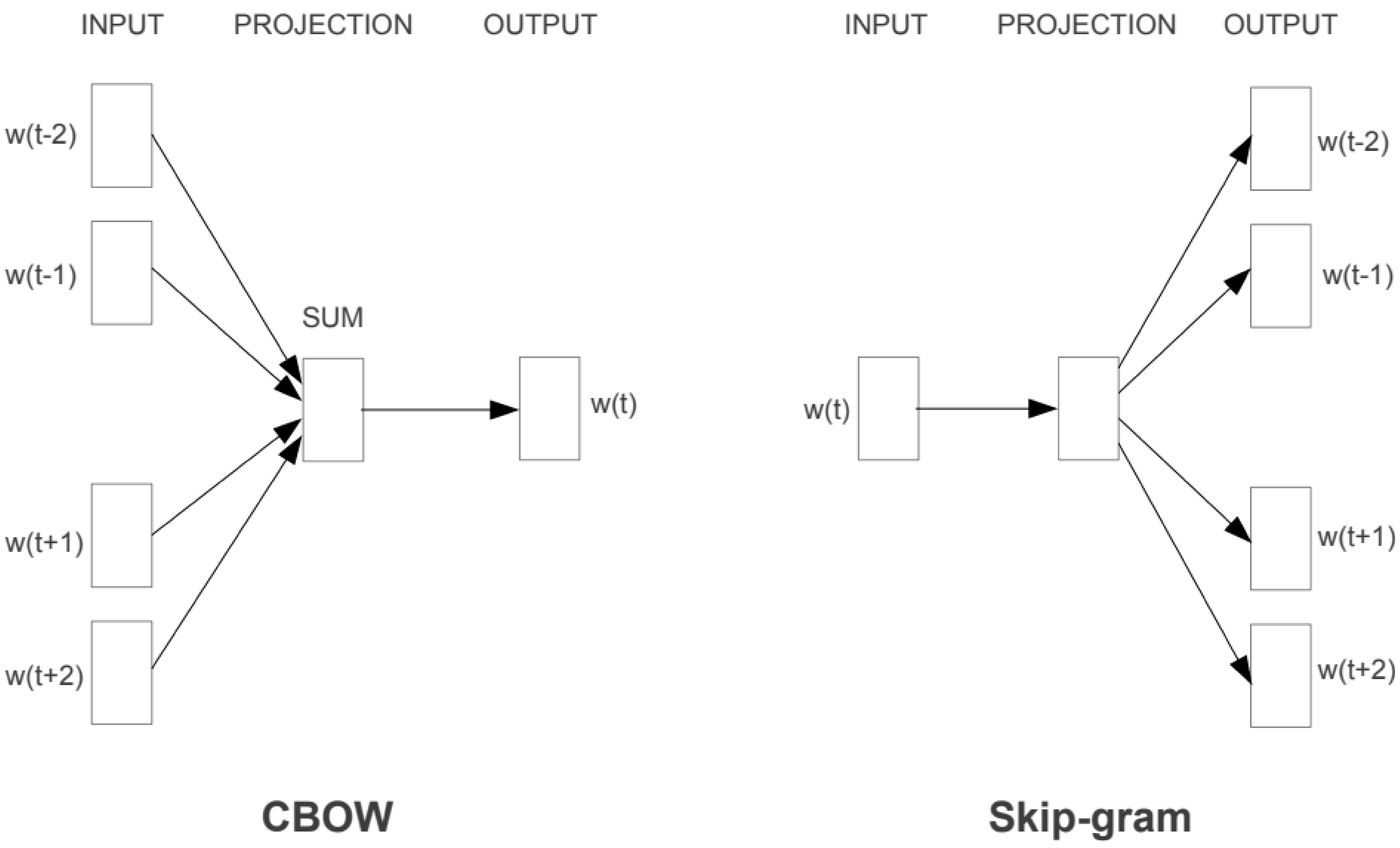
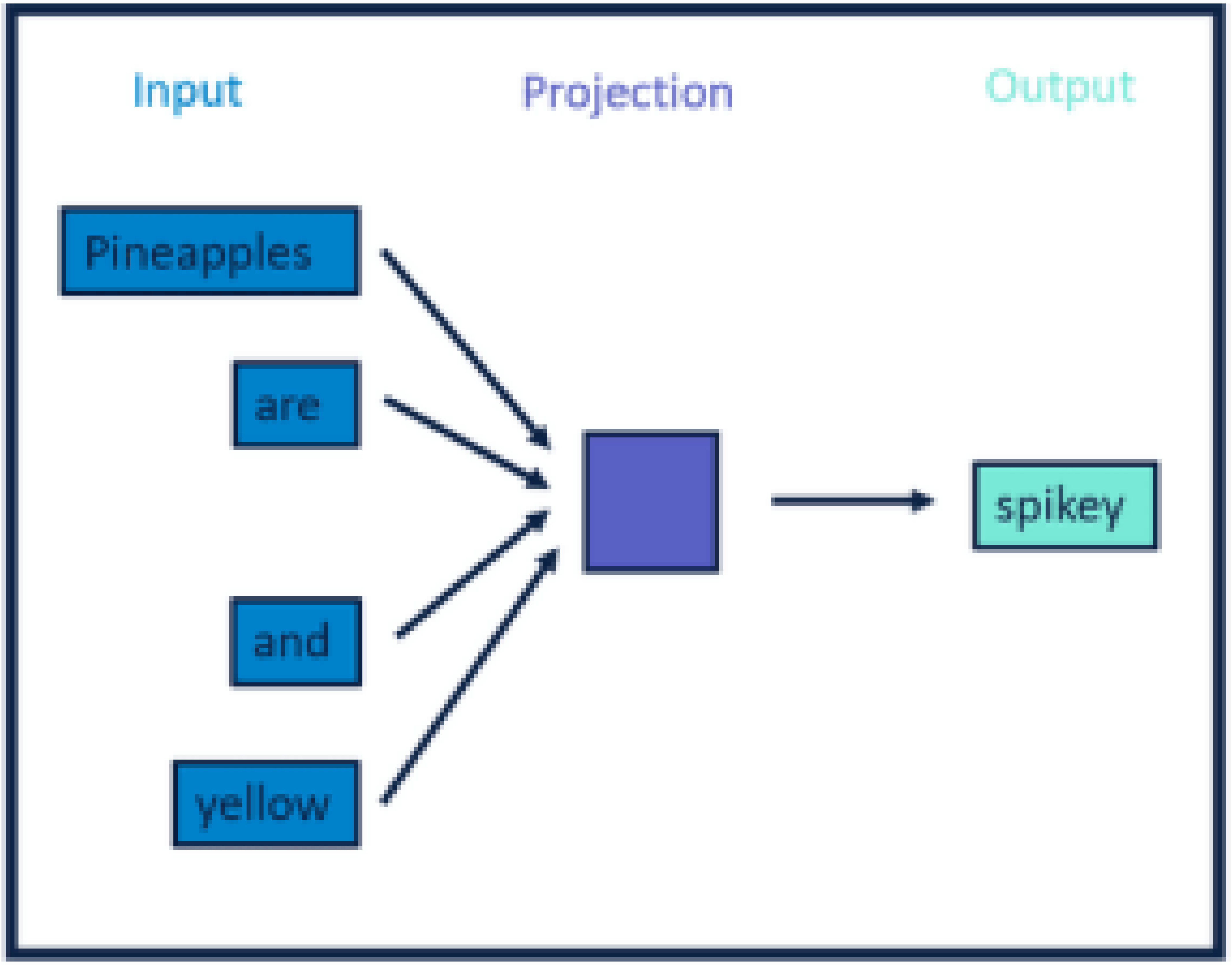
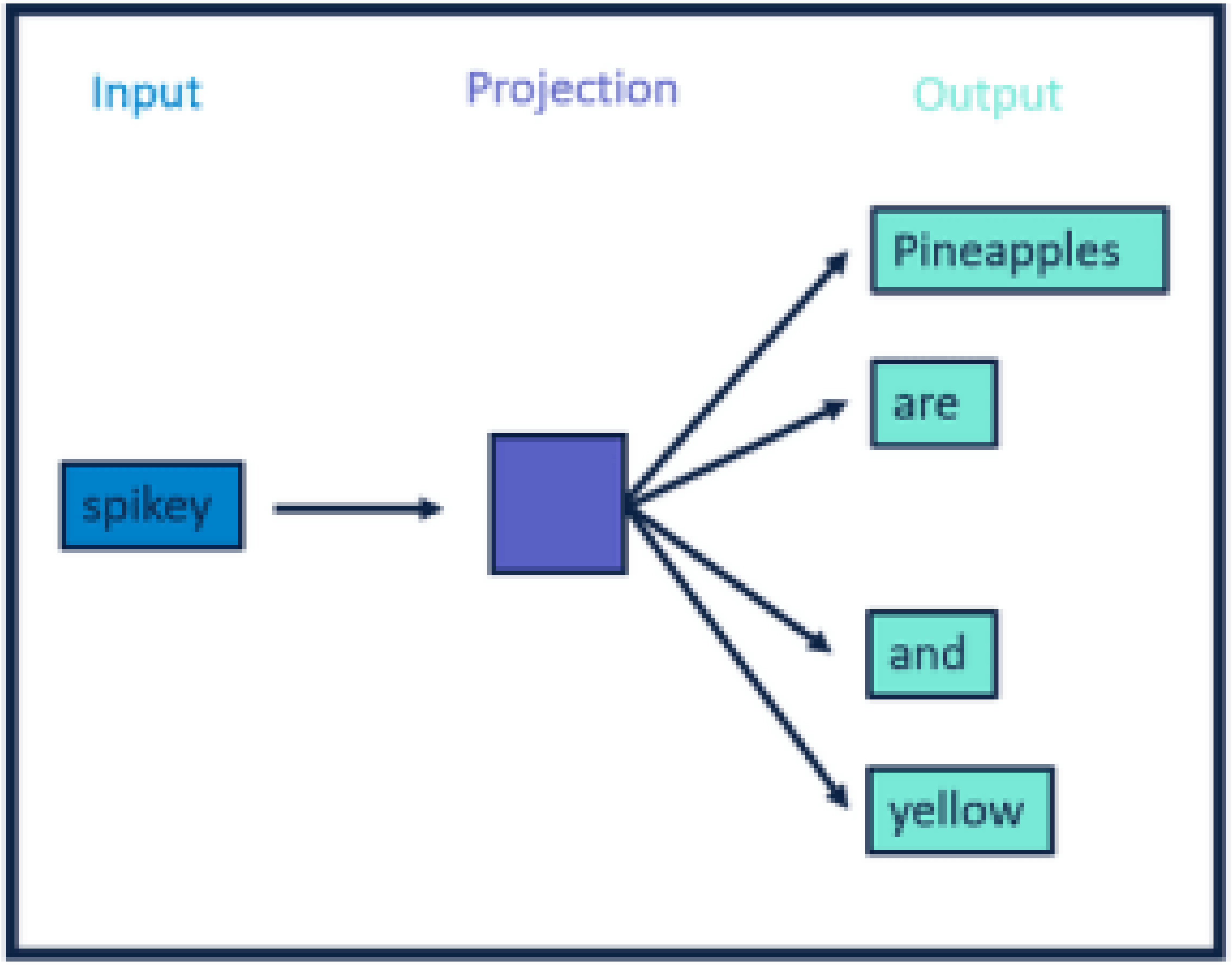


Figure 1: New model architectures. The CBOW architecture predicts the current word based on the context, and the Skip-gram predicts surrounding words given the current word.

Word2Vec (cont’d)



CBOW



Skip-gram

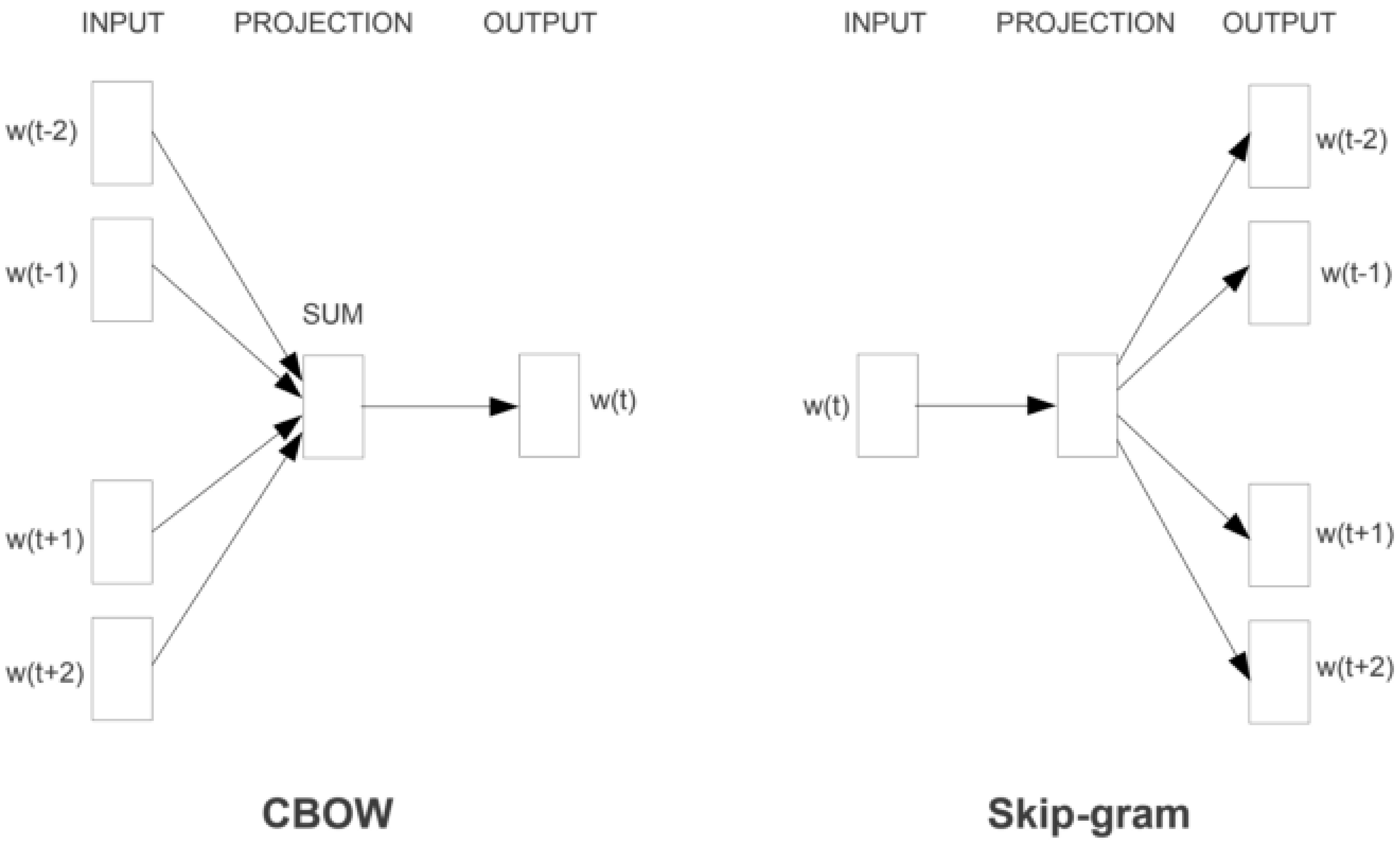
(Source: https://community.alteryx.com)

Word2Vec (cont'd): Difference between Skip gram & CBow

Skip gram:	CBow:
• In this input is center word and output is context words (neighbour words). • Works well with small datasets. • Skip-gram identifies rarer words better.	• In this context or neighbor words are input and output is the center word. • Works good with large datasets. • Better representation for frequent words than rarer.

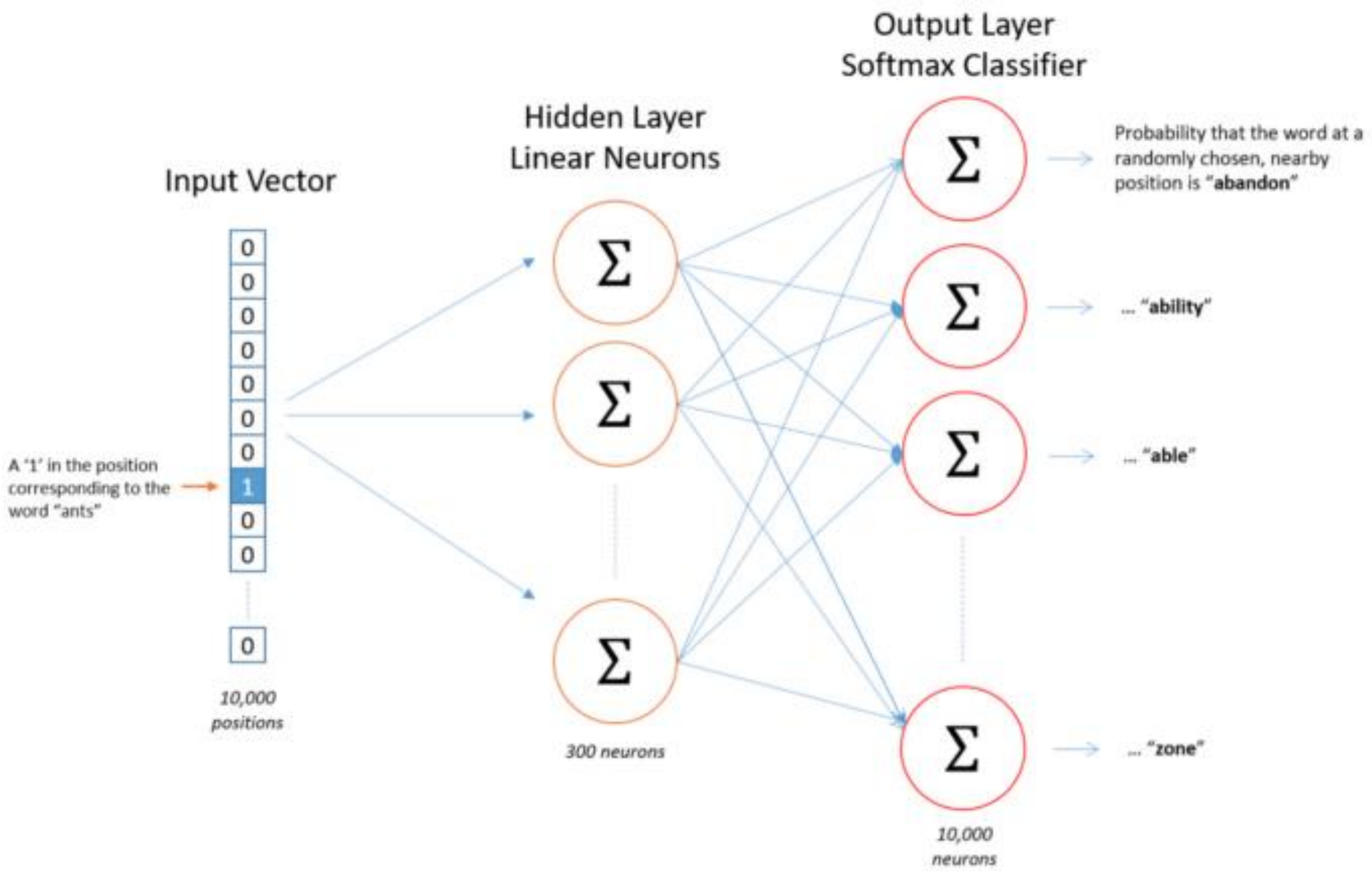
HOW TO TRANSFORM TEXT INTO NUMBERS
Word embedding model = Skip-Gram Neural Network

- Word2Vec
- Learn from big data
- Self-supervised learning



[Source: “Efficient Estimation of Word Representations in Vector Space”]

HOW TO TRANSFORM TEXT INTO NUMBERS
Word embedding model = Skip-Gram Neural Network



[Source: http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model]

HOW TO TRANSFORM TEXT INTO NUMBERS
Word embedding model = Skip-Gram Neural Network

Basic concept: One-hot vector + Linear transformation

- Each word represents as an embedding vector
- Changed by training process.

For example: "monkey" → [0, 1, 0]

$$\begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \times \begin{bmatrix} 0.2 \\ -0.8 \\ 0.6 \\ 1 \\ 0.4 \end{bmatrix} = \begin{pmatrix} 0, & 0, & 0, & 0, & 0 \\ \boxed{0.2 \quad -0.8 \quad 0.6 \quad 1 \quad 0.4} \\ 0, & 0, & 0, & 0, & 0 \end{pmatrix}$$

Dense vector

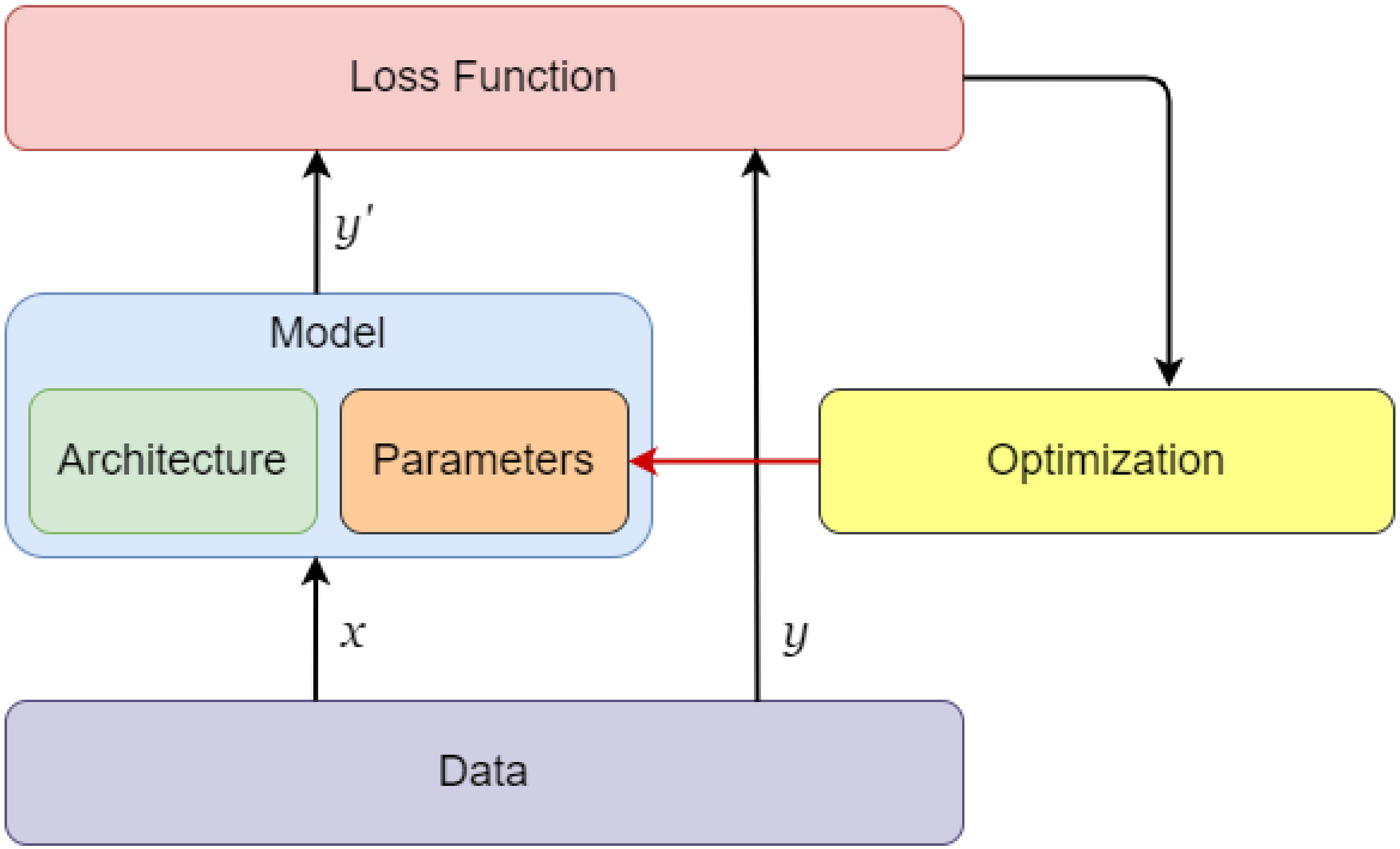
No need to do matrix multiplication

Question: what is the purpose of this weight vector?
What is it?

WHAT IS A NEURAL NETWORK?

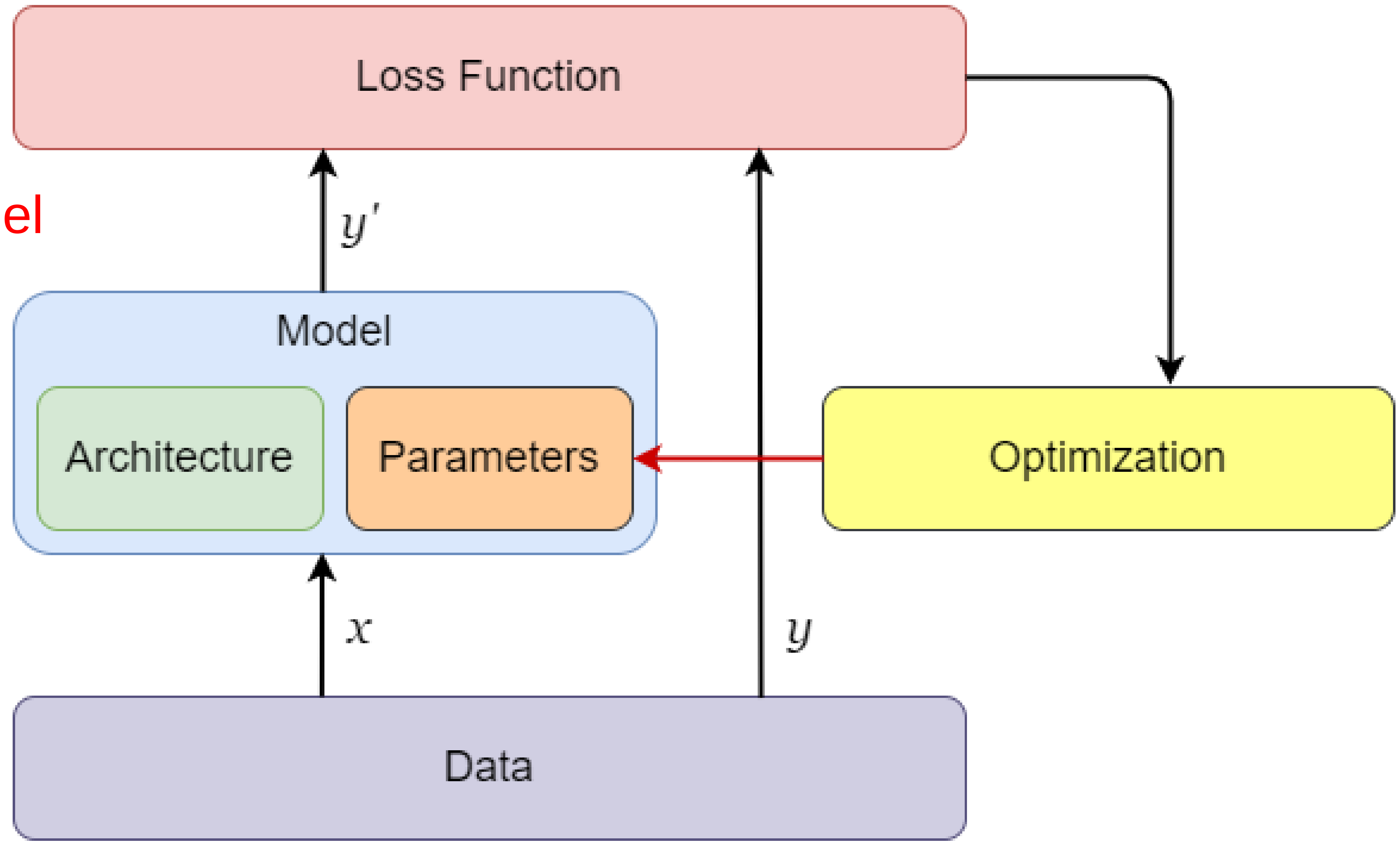
(Source: www.dreamstime.com)

NEURAL NETWORK OVERVIEW

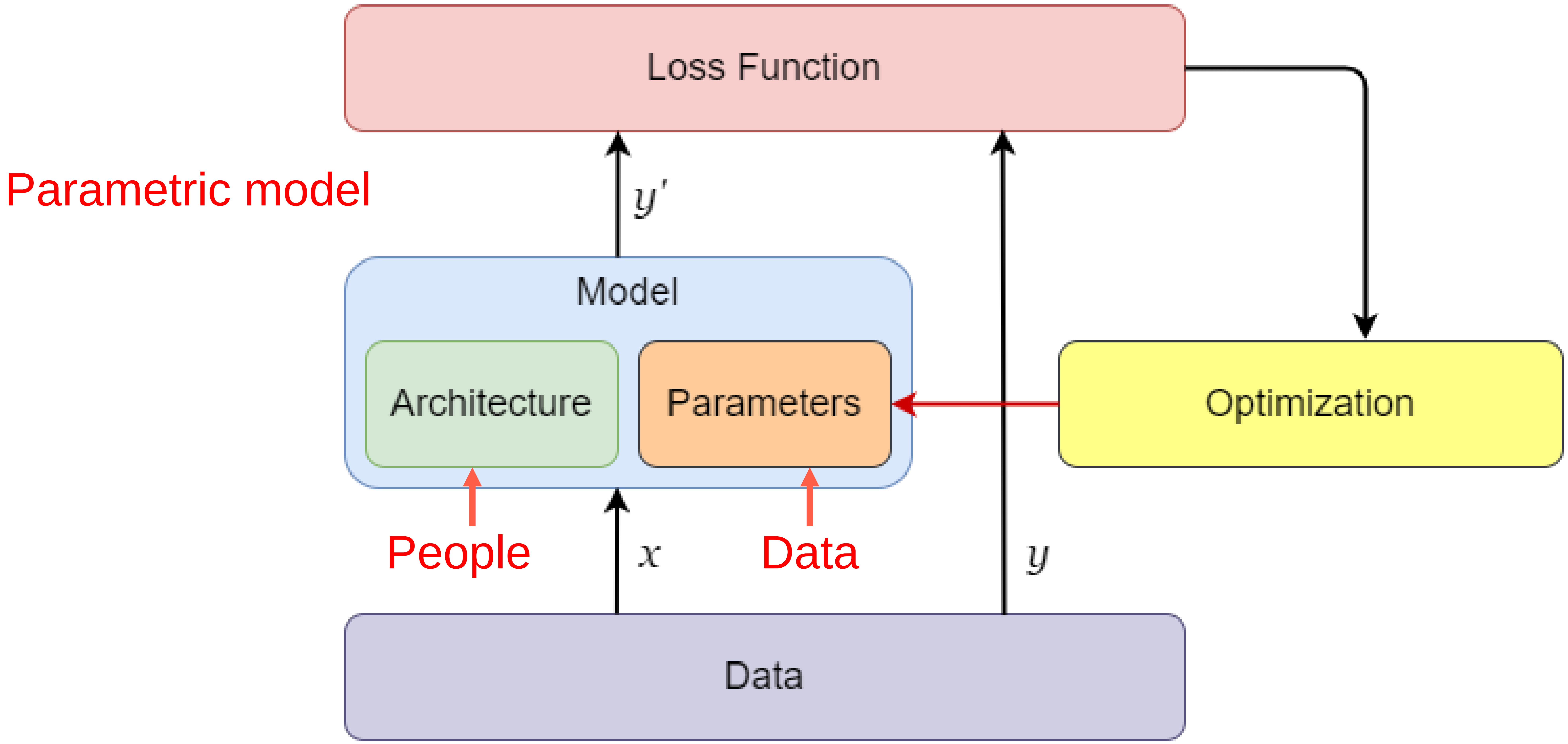


NEURAL NETWORK OVERVIEW

Parametric model



NEURAL NETWORK OVERVIEW



How to decide neural network architecture?

PARAMETRIC MODEL

Architecture + Parameters = Parametric model

- Layers
- Neurons

[1, 0.8, -1, 0.5, 0.2, …, -0.3]

- Constant
- Determined by people

- Variable
- Determined by data

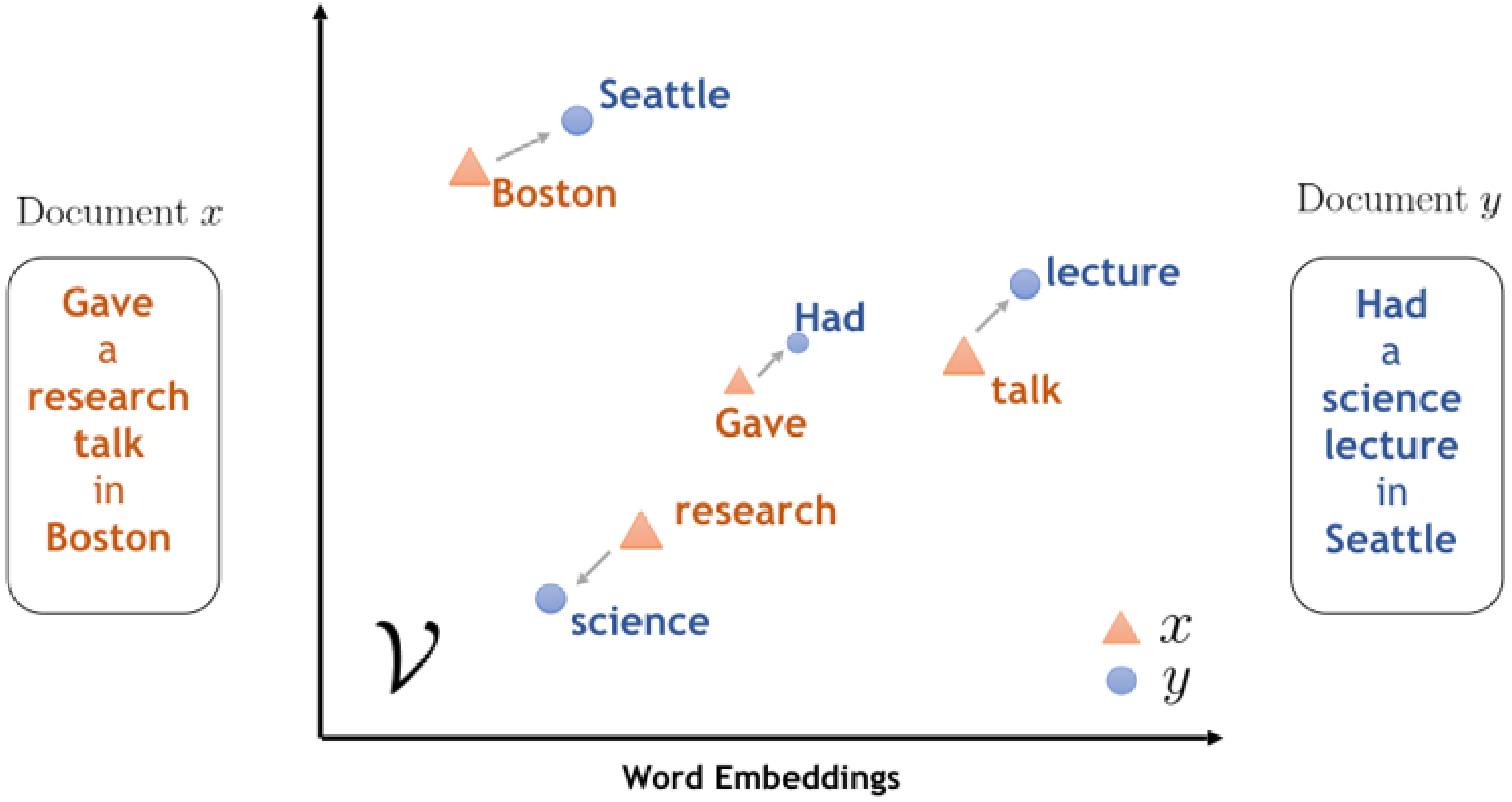
HOW TO TRANSFORM TEXT INTO NUMBERS
Word embedding model = Skip-Gram Neural Network

- Nearest neighbors

moon	score
mars	0.615
moons	0.611
lunar	0.602
sun	0.602
venus	0.583

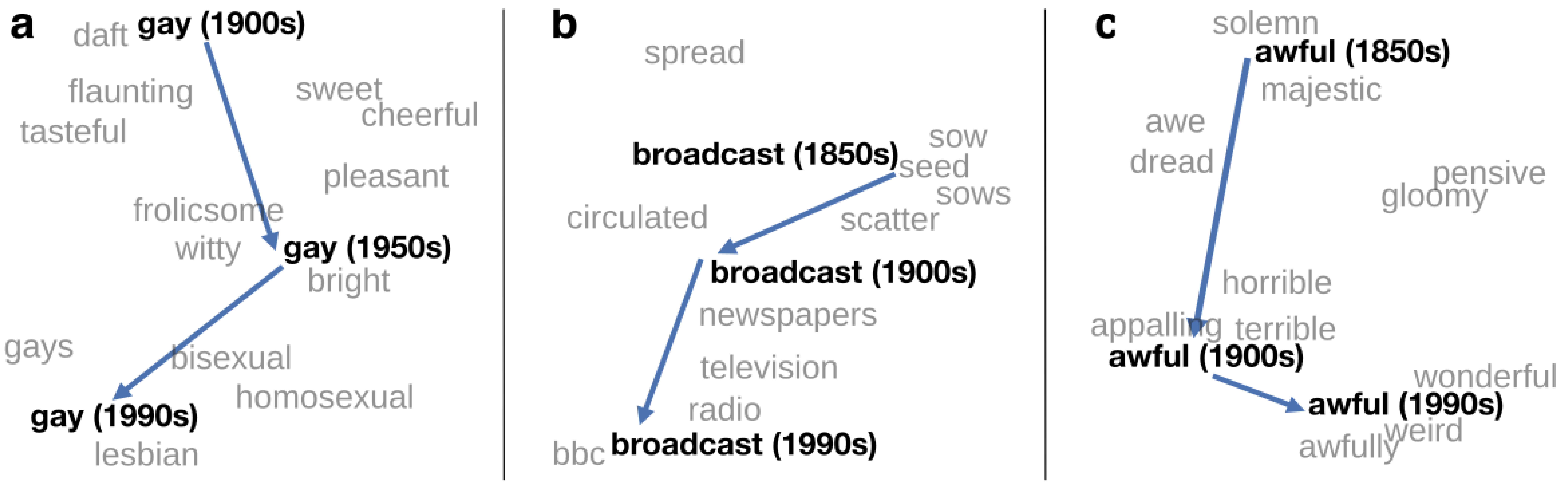
talking	score
discussing	0.663
telling	0.657
joking	0.632
thinking	0.627
talked	0.624

blue	score
red	0.704
yellow	0.677
purple	0.676
green	0.655
pink	0.612



[Source: https://www.ibm.com/blogs/research/2018/11/word-movers-embedding/]

HOW TO TRANSFORM TEXT INTO NUMBERS
Word Embeddings for Historical Text



(Source: nlp.stanford.edu/projects/histwords)

HOW TO TRANSFORM TEXT INTO NUMBERS
Word embedding model (Word2Vec)

Demo: http://vectors.nlpl.eu/explore/embeddings/en/associates/

Word2Vec implementation in Pytorch

SkipGram from scratch

```
import numpy as np
import torch
import torch.nn as nn
import torch.optim as optim
from torch.autograd import Variable
import matplotlib.pyplot as plt

dtype = torch.FloatTensor

# 3 Words Sentence (to semplify)
# All them form our text corpus
sentences = [ "i like rabbit", "i like dog", "i like cat",
              "i like animal", "dog is cute",
              "cat chases mouse", "dog is animal",
              "cat is animal", "fish is animal",
              "dog and cat", "fish and rabbit", "i like apple",
              "rabbit is cute", "music and movie", "watch movie",
              "i like book", "i hate mouse", "listen to music"]
```

Word2Vec implementation in Pytorch (cont'd)

SkipGram from scratch

```
# list all the words present in our corpus
word_sequence = " ".join(sentences).split()
print(word_sequence )
# build the vocabulary
word_list = list(set(word_sequence))
print(word_list)
word_dict = {w: i for i, w in enumerate(word_list)}
print(word_dict)
```

```
['i', 'like', 'rabbit', 'i', 'like', 'dog', 'i', 'like', 'cat', 'i', 'like', 'animal', 'dog', 'is',
'cute', 'cat', 'chases', 'mouse', 'dog', 'is', 'animal', 'cat', 'is', 'animal', 'fish', 'is', 'anima
l', 'dog', 'and', 'cat', 'fish', 'and', 'rabbit', 'i', 'like', 'apple', 'rabbit', 'is', 'cute', 'musi
c', 'and', 'movie', 'watch', 'movie', 'i', 'like', 'book', 'i', 'hate', 'mouse', 'listen', 'to', 'mus
ic']
['dog', 'mouse', 'animal', 'and', 'watch', 'hate', 'rabbit', 'i', 'music', 'chases', 'like', 'movie',
'fish', 'cute', 'cat', 'listen', 'apple', 'to', 'book', 'is']
{'dog': 0, 'mouse': 1, 'animal': 2, 'and': 3, 'watch': 4, 'hate': 5, 'rabbit': 6, 'i': 7, 'music': 8,
'chases': 9, 'like': 10, 'movie': 11, 'fish': 12, 'cute': 13, 'cat': 14, 'listen': 15, 'apple': 16,
'to': 17, 'book': 18, 'is': 19}
```

Word2Vec implementation in Pytorch (cont'd)

SkipGram from scratch

```
# Word2Vec Parameter
batch_size = 20
embedding_size = 2  # To show 2 dim embedding graph
voc_size = len(word_list)
# input word
j = 1
print("Input word : ")
print(word_sequence[j], word_dict[word_sequence[j]])
# context words
print("Context words : ")
print(word_sequence[j - 1], word_sequence[j + 1])
print([word_dict[word_sequence[j - 1]], word_dict[word_sequence[j + 1]]])
```

```
Input word :
like 10
Context words :
i rabbit
[7, 6]
```

Word2Vec implementation in Pytorch (cont'd)

SkipGram from scratch

```
# Make skip gram of one size window
skip_grams = []
for i in range(1, len(word_sequence) - 1):
    input = word_dict[word_sequence[i]]
    context = [word_dict[word_sequence[i - 1]], word_dict[word_sequence[i + 1]]]

    for w in context:
        skip_grams.append([input, w])


#lets plot some data
skip_grams[:6]
```

```
[[10, 7], [10, 6], [6, 10], [6, 7], [7, 6], [7, 10]]
```

Word2Vec implementation in Pytorch (cont'd)

SkipGram from scratch

```
np.random.seed(172)

def random_batch(data, size):
    random_inputs = []
    random_labels = []
    random_index = np.random.choice(range(len(data)), size, replace=False)

    for i in random_index:
        # one-hot encoding of words
        random_inputs.append(np.eye(voc_size)[data[i][0]])  # input
        random_labels.append(data[i][1])  # context word

    return random_inputs, random_labels

random_batch(skip_grams[:6], size=3)
```

```
([array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0.,
        0., 0., 0.]),
  array([0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0.]),
  array([0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0.])],
 [7, 6, 7])
```

Word2Vec implementation in Pytorch (cont'd)

SkipGram from scratch

```
# Model
class Word2Vec(nn.Module):
    def __init__(self):
        super(Word2Vec, self).__init__()

        # parameters between -1 and + 1
        # voc_size -> embedding_size Weight
        self.W = nn.Parameter(-2 * torch.rand(voc_size, embedding_size) + 1).type(dtype)
        # embedding_size -> voc_size Weight
        self.V = nn.Parameter(-2 * torch.rand(embedding_size, voc_size) + 1).type(dtype)

    def forward(self, X):
        hidden_layer = torch.matmul(X, self.W) # hidden_layer : [batch_size, embedding_size]
        output_layer = torch.matmul(hidden_layer, self.V) # output_layer : [batch_size, voc_size]
        #return output_layer
        return output_layer

model = Word2Vec()
# Set the model in train mode
model.train()
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
```

Word2Vec implementation in Pytorch (cont'd)

SkipGram from scratch

```
# Training
for epoch in range(5000):

    input_batch, target_batch = random_batch(skip_grams, batch_size)

    # new_tensor(data, dtype=None, device=None, requires_grad=False)
    input_batch = torch.Tensor(input_batch)
    target_batch = torch.LongTensor(target_batch)

    optimizer.zero_grad()
    output = model(input_batch)

    # output : [batch_size, voc_size], target_batch : [batch_size] (LongTensor, not one-hot)
    loss = criterion(output, target_batch)
    if (epoch + 1)%1000 == 0:
        print('Epoch:', '%04d' % (epoch + 1), 'cost =', '{:.6f}'.format(loss))

    loss.backward()
    optimizer.step()
```

Word2Vec implementation in Pytorch (cont'd)

SkipGram from scratch

```python
# Training
for epoch in range(5000):

    input_batch, target_batch = random_batch(skip_grams, batch_size)

    # new_tensor(data, dtype=None, device=None, requires_grad=False)
    input_batch = torch.Tensor(input_batch)
    target_batch = torch.LongTensor(target_batch)

    optimizer.zero_grad()
    output = model(input_batch)

    # output : [batch_size, voc_size], target_                              )
    loss = criterion(output, target_batch)
    if (epoch + 1)%1000 == 0:
        print('Epoch:', '%04d' % (epoch + 1),

    loss.backward()
    optimizer.step()
```

```
Epoch: 1000 cost = 2.759658
Epoch: 2000 cost = 2.565675
Epoch: 3000 cost = 2.525607
Epoch: 4000 cost = 2.482483
Epoch: 5000 cost = 2.337673
```

Word2Vec implementation in Pytorch (cont'd)

SkipGram from scratch

```
# Learned W
W, _= model.parameters()
print(W.detach())
```

```
tensor([[-3.9530e-01, -1.1977e+00],
        [-1.1074e-01,  5.5578e-01],
        [-1.1224e+00, -1.7714e+00],
        [-4.7694e-01,  1.1233e-01],
        [-1.4306e+00,  3.0196e+00],
        [ 1.7081e-01, -1.3468e-01],
        [ 6.7669e-01, -1.3274e+00],
        [ 1.1087e-01, -6.3640e-01],
        [-2.3755e+00, -5.2150e-01],
        [-2.2572e+00,  5.6926e-01],
        [ 7.6038e-01, -1.1725e+00],
        [ 1.3367e+00, -3.7740e-01]
```

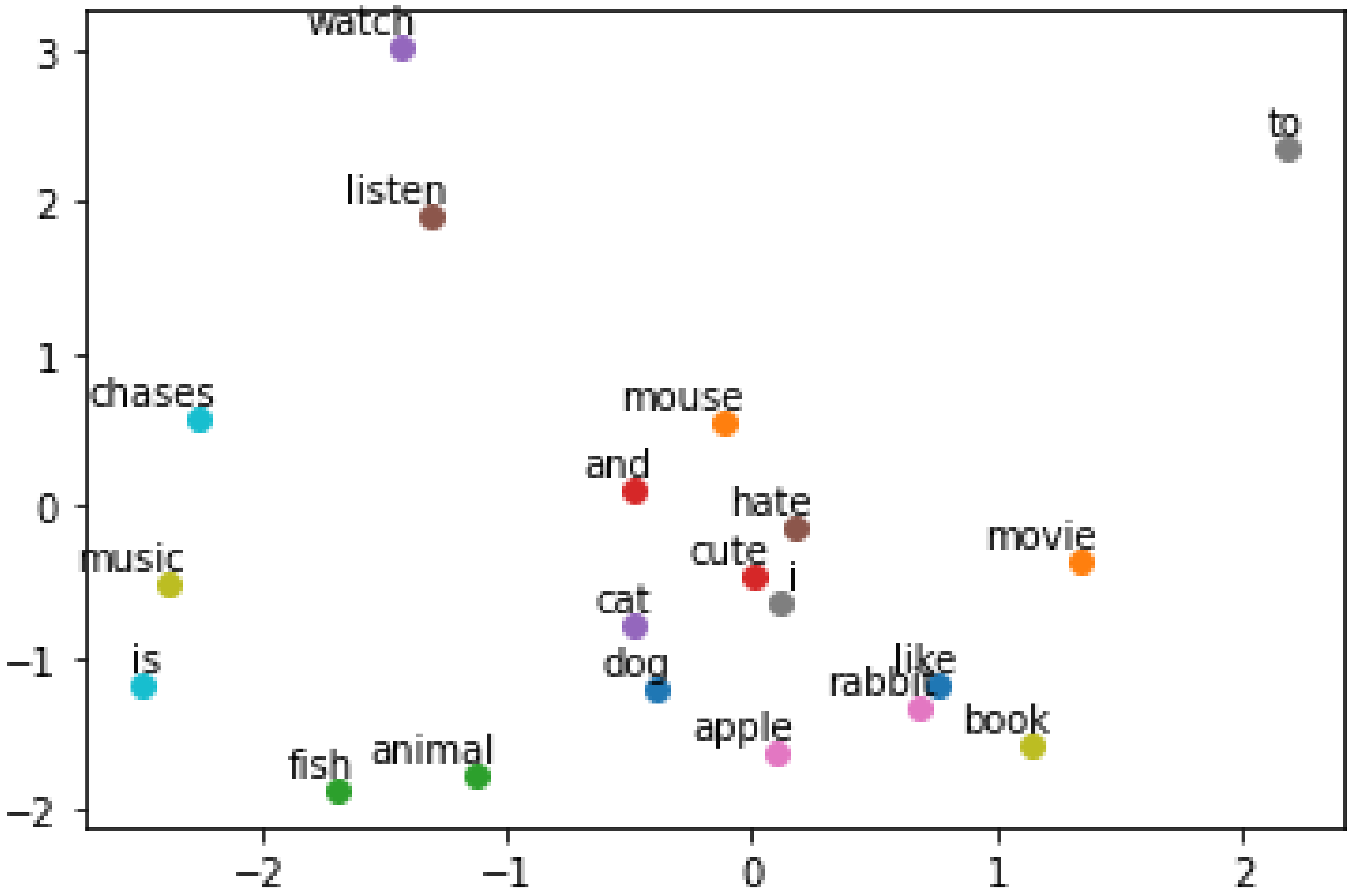
Word2Vec implementation in Pytorch (cont'd)

SkipGram from scratch

```
for i, word in enumerate(word_list):
    W, _= model.parameters()
    W = W.detach()
    x,y = float(W[i][0]), float(W[i][1])
    plt.scatter(x, y)
    plt.annotate(word, xy=(x, y), xytext=(5, 2), textcoords='offset points', ha='right', va='bottom')
plt.show()
```

Word2Vec implementation in Pytorch (cont'd)

SkipGram from scratch



Building Word2Vec model with custom text using existing libraries

Word2vec with Gensim

Required libraries:

- pip install pandas
- pip install gensim
- pip install spacy

```
import pandas as pd
import gensim
import spacy
import en_core_web_sm
from tqdm import tqdm
```

```
tqdm.pandas(desc="Progress")
```

```
nlp_en = en_core_web_sm.load()
```

Dataset: https://www.kaggle.com/datasets/snap/amazon-fine-food-reviews?select=Reviews.csv

Building Word2Vec model with custom text using existing libraries (cont’d)

1.1 Get data

```
pd_data = pd.read_csv("Reviews.csv")
```

```
pd_data.head(3)
```

ıctId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summary	Text
⁽FG0	A3SGXH7AUHU8GW	delmartian	1	1	5	1303862400	Good Quality Dog Food	I have bought several of the Vitality canned d...
RG4	A1D87F6ZCVE5NK	dll pa	0	0	1	1346976000	Not as Advertised	Product arrived labeled as Jumbo

Building Word2Vec model with custom text using existing libraries (cont'd)

1.2. Process data

```
def get_tokens(sentence):
    return [x.text for x in nlp_en(sentence)]
```

```
pd_data["tokens"] = pd_data["Text"].progress_apply(get_tokens)
```

```
Progress:  87%|████████████████████████████████████████
| 493236/568454 [1:18:19<10:36, 118.12it/s]
```

```
pd_data.to_pickle("pd_data_tokenized.pickle")
```

```
pd_data = pd.read_pickle("pd_data_tokenized.pickle")
```

Building Word2Vec model with custom text using existing libraries (cont'd)

1.3. Train word embeddings using word2vec

```
model_w2v = gensim.models.Word2Vec(pd_data["tokens"].tolist(), min_count=5, window = 9, size = 100)
```

1.4. Train word embeddings using fasttext

```
model_ft = gensim.models.FastText(pd_data["tokens"].tolist(), min_count=5, window = 9, size = 100)
```

Building Word2Vec model with custom text using existing libraries (cont'd)

1.5. Persistence

```
model_w2v.save("model_w2v.model")
model_ft.save("model_ft.model")
```

```
model_w2v = gensim.models.Word2Vec.load("model_w2v.model")
model_ft = gensim.models.FastText.load("model_ft.model")
```

Building Word2Vec model with custom text using existing libraries (cont'd)

1.6. Similarity

```
model_w2v.most_similar("salmon", topn=5)
```

```
C:\Users\peter\Anaconda3\lib\site-packages\ipykernel\__main__.py:1: DeprecationWarning: Call to depre
cated `most_similar` (Method will be removed in 4.0.0, use self.wv.most_similar() instead).
  if __name__ == '__main__':
```

```
[('fish', 0.8536328077316284),
 ('tuna', 0.7662709951400757),
 ('chicken', 0.7630202174186707),
 ('seafood', 0.7627329230308533),
 ('turkey', 0.7592297792434692)]
```

Building Word2Vec model with custom text using existing libraries (cont'd)

```
model_w2v.most_similar(positive=['cheese'], topn=5)
```

```
C:\Users\peter\Anaconda3\lib\site-packages\ipykernel\__main__.py:1: DeprecationWarning: Call to depre
cated `most_similar` (Method will be removed in 4.0.0, use self.wv.most_similar() instead).
  if __name__ == '__main__':
```

```
[('cheddar', 0.7746697068214417),
 ('mozzarella', 0.7572810649871826),
 ('parmesan', 0.7331867218017578),
 ('chedder', 0.7296013236045837),
 ('mayo', 0.7252874374389648)]
```

Building Word2Vec model with custom text using existing libraries (cont'd)

1.7. Correlation

```
model_w2v.most_similar(positive=['avocado', 'salsa'], negative=['tomato'], topn=3)
```

```
C:\Users\peter\Anaconda3\lib\site-packages\ipykernel\__main__.py:1: DeprecationWarning: Call to depre
cated `most_similar` (Method will be removed in 4.0.0, use self.wv.most_similar() instead).
  if __name__ == '__main__':
```

```
[('hummus', 0.6872091293334961),
 ('guacamole', 0.6551289558410645),
 ('burritos', 0.6080722808837891)]
```

Building Word2Vec model with custom text using existing libraries (cont'd)

```
model_w2v.most_similar(positive=['lemon', 'water'], topn=3)
```

```
C:\Users\peter\Anaconda3\lib\site-packages\ipykernel\__main__.py:1: DeprecationWarning: Call to depre
cated `most_similar` (Method will be removed in 4.0.0, use self.wv.most_similar() instead).
  if __name__ == '__main__':
```

```
[('tequila', 0.7341920137405396),
 ('lemonade', 0.7284362316131592),
 ('juice', 0.728117346763610 8)]
```

```
model_w2v.most_similar(positive=['salami', 'crust'], topn=3)
```

```
C:\Users\peter\Anaconda3\lib\site-packages\ipykernel\__main__.py:1: DeprecationWarning: Call to depre
cated `most_similar` (Method will be removed in 4.0.0, use self.wv.most_similar() instead).
  if __name__ == '__main__':
```

```
[('bread', 0.7283815145492554),
 ('pizza', 0.7018527388572693),
 ('dough', 0.6836484670639038)]
```

Building Word2Vec model with custom text using existing libraries (cont'd)

```
model_w2v.most_similar(positive=['beef', 'bun'], topn=3)
```

```
C:\Users\peter\Anaconda3\lib\site-packages\ipykernel\__main__.py:1: DeprecationWarning: Call to depre
cated `most_similar` (Method will be removed in 4.0.0, use self.wv.most_similar() instead).
  if __name__ == '__main__':
```

```
[('hamburger', 0.814429521560669),
 ('ham', 0.795830488204956),
 ('sausage', 0.7887133359909058)]
```

Building Word2Vec model with custom text using existing libraries (cont'd)

2. Visualise them

```
from sklearn.manifold import TSNE
import matplotlib.pyplot as plt
from bokeh.plotting import figure, output_file, show
from bokeh.models import ColumnDataSource, Range1d, LabelSet, Label
```

```
%%time
model_w2v = gensim.models.Word2Vec(pd_data["tokens"].tolist(), min_count=500, window = 9, size = 100)
```

```
Wall time: 2min 7s
```

Building Word2Vec model with custom text using existing libraries (cont'd)

```
tokens = []
labels = []

for x in model_w2v.wv.vocab:
    tokens.append(model_w2v[x])
    labels.append(x)
```

```
C:\Users\peter\Anaconda3\lib\site-packages\ipykernel\__main__.py:5: DeprecationWarning: Call to depre
cated `__getitem__` (Method will be removed in 4.0.0, use self.wv.__getitem__() instead).
```

```
%%time
tsne_model = TSNE(n_components=2, random_state=11)
fitted = tsne_model.fit_transform(tokens)
```

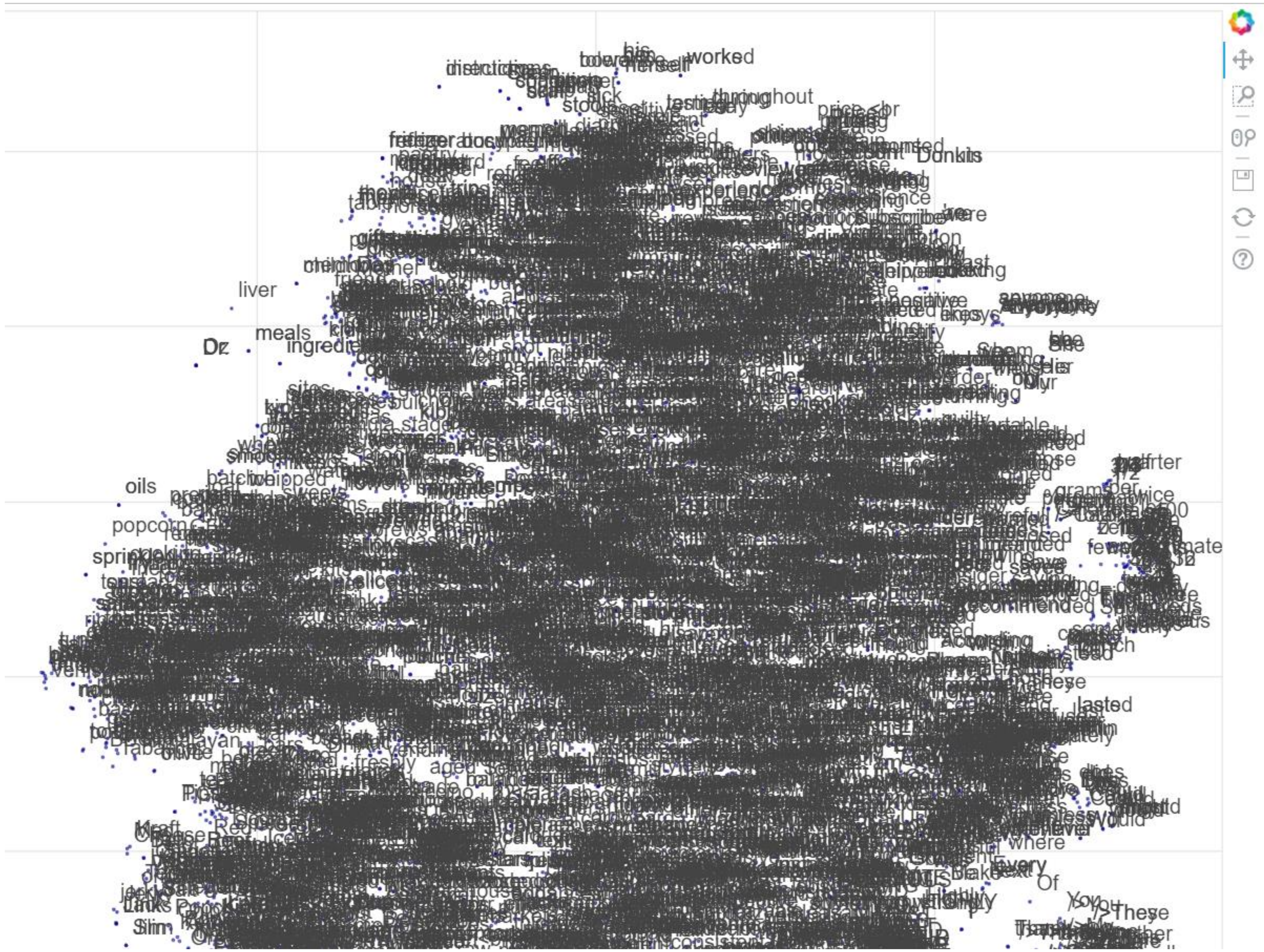
```
Wall time: 2min 47s
```

Building Word2Vec model with custom text using existing libraries (cont'd)

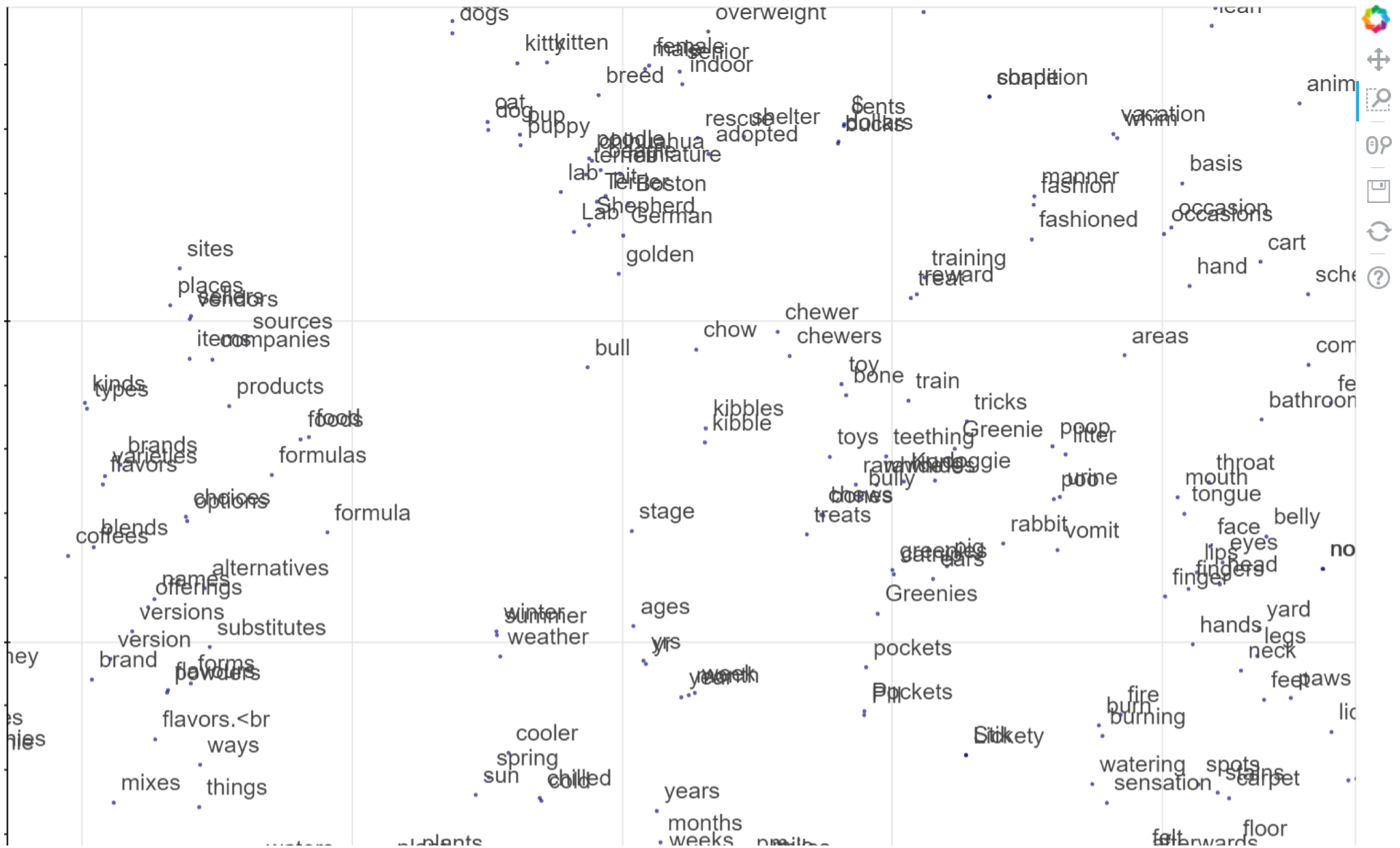
```
output_file("plot.html")

p = figure(plot_width=1000, plot_height=1000)
lst = list(model_w2v.wv.vocab)
p.circle(fitted[:, 0], fitted[:, 1], size=2, color="navy", alpha=0.5)
texts = lst
source = ColumnDataSource(data=dict(x=fitted[:, 0], y=fitted[:, 1], text=texts))
labels = LabelSet(x='x', y='y', text='text',
         x_offset=5, y_offset=5, source=source)
p.add_layout(labels)
show(p)
```

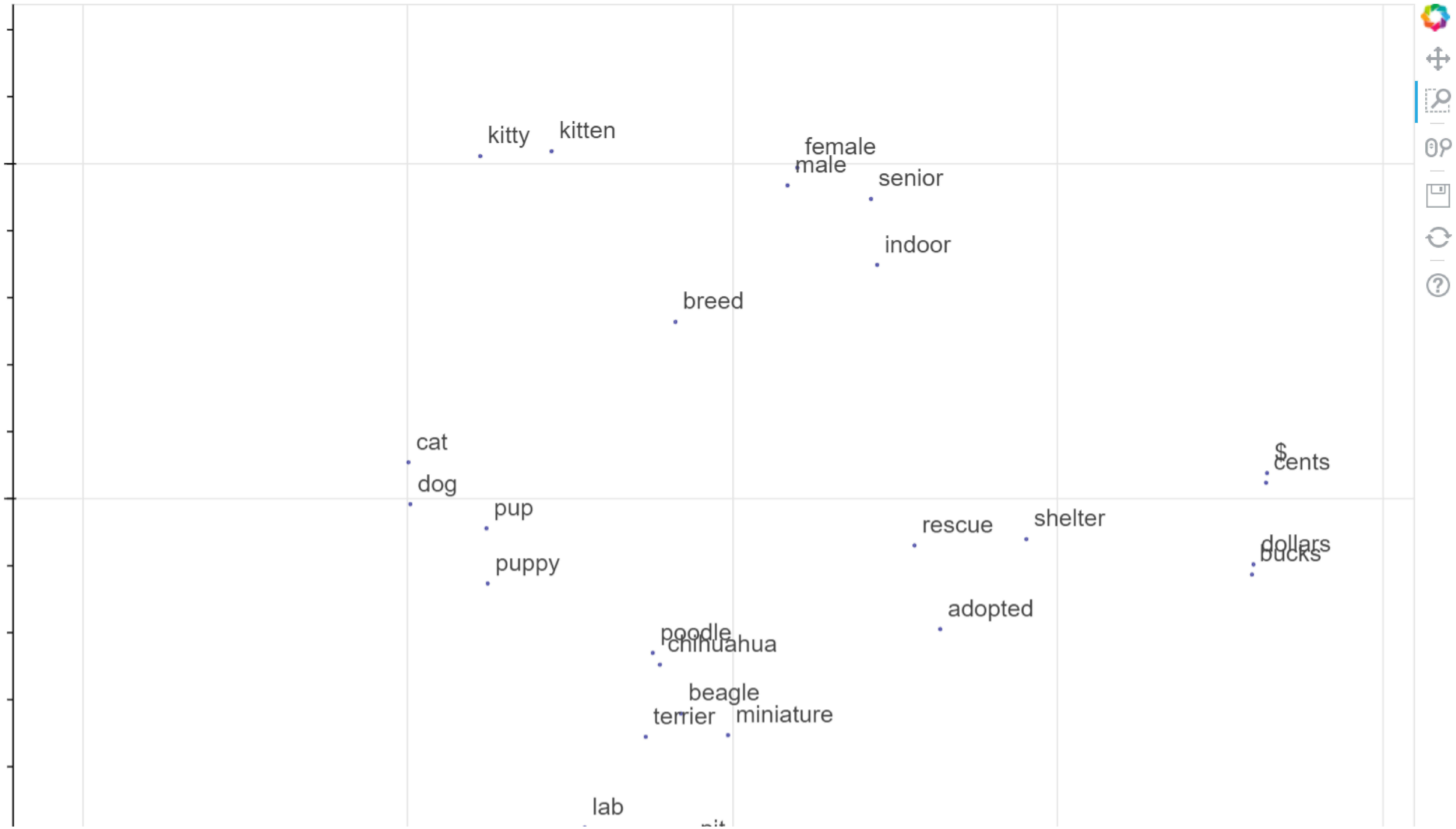
Building Word2Vec model with custom text using existing libraries (cont'd)



Building Word2Vec model with custom text using existing libraries (cont’d)



Building Word2Vec model with custom text using existing libraries (cont’d)



DISCUSSION