COMP 554 / CSDS 553 Advanced NLP

Faizad Ullah

Vector Semantics and Embeddings

荃者所以在鱼，得鱼而忘荃

Nets are for fish; Once you get the fish, you can forget the net.

言者所以在意，得意而忘言

Words are for meaning; Once you get the meaning, you can forget the words

Vector Space Models

- Set-of-Words
- Bag-of-Words
- Term Frequency - Inverse Document Frequency
- Word Embeddings

Vector Space Models

d_1 = The car is driven on the road

d_2 = The truck is driven on the highway

A bag-of-bigrams representation is much more powerful than bag-of-words.

	The	car	is	driven	on	the	road	truck	highway
d1									
d2									

Term Frequency - Inverse Document Frequency

- The first is the **Term Frequency**: the frequency of the word/term *t* in the document *d*. We can just use the raw count as the term frequency:

  - $tf_{t,d}$ = count(t,d)

Term Frequency

$$\mathrm{tf}_{t,d} = \begin{cases} 1 + \log_{10} \mathrm{count}(t,d) & \text{if } \mathrm{count}(t,d) > 0 \\ 0 & \text{otherwise} \end{cases}$$

- If we use log weighting, terms which occur:
  - 0 times in a document would have tf = 0
  - 1 times in a document tf = 1 + $\log_{10}(1)$ = 1 + 0 = 1
  - 10 times in a document tf = 1+$\log_{10}(10)$ = 2
  - 100 times tf = 1+$\log_{10}(100)$ = 3
  - 1000 times tf = 4, and so on.

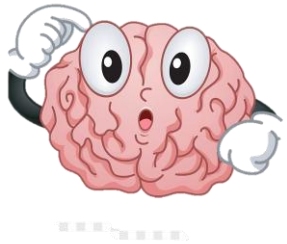
Inverse Document Frequency

$$\text{idf}_t = \log_{10}\left(\frac{N}{\text{df}_t}\right)$$

- Where N is the total number of documents in the collection, and df_t is the number of documents in which term t occurs.

Term Frequency - Inverse Document Frequency

- The second factor in tf-idf is used to give a higher weight to words that occur only in a few documents.



- Terms that are limited to a few documents are useful for discriminating those documents from the rest of the collection; terms that occur frequently across the entire collection aren't as helpful.

Term Frequency - Inverse Document Frequency

$$w_{x,y} = tf_{x,y} \times \log\left(\frac{N}{df_x}\right)$$

TF-IDF

Term x within document y

$tf_{x,y}$ = frequency of x in y
df_x = number of documents containing x
N = total number of documents

$$TF_{ij} = \frac{f_{ij}}{n_j} \tag{1}$$

Where f_{ij} is the frequency of term i in document j. n_j is the total number of words in document j.

$$IDF_i = 1 + log(\frac{N}{c_i}) \tag{2}$$

Where N is the total number of documents in the corpus. c_i is the number of documents that contain word i.

$$w_{ij} = TF_{ij} \times IDF_i \tag{3}$$

Where w_{ij} is the TF-IDF score of term i in document j.

Term Frequency - Inverse Document Frequency

D_1: "The car is driven on the road"

D_2: "The truck is driven on the highway"

Word	TF		IDF	TF*IDF	
	A	B		A	B
The	1/7	1/7	log(2/2) = 0	0	0
Car	1/7	0	log(2/1) = 0.3	0.043	0
Truck	0	1/7	log(2/1) = 0.3	0	0.043
Is	1/7	1/7	log(2/2) = 0	0	0
Driven	1/7	1/7	log(2/2) = 0	0	0
On	1/7	1/7	log(2/2) = 0	0	0
The	1/7	1/7	log(2/2) = 0	0	0
Road	1/7	0	log(2/1) = 0.3	0.043	0
Highway	0	1/7	log(2/1) = 0.3	0	0.043

Common words score is zero (not significant)

Score of "car", "truck", "road", and "highway" are non-zero (significant words)

Vector Semantics

- Vector semantics is the standard way to represent word meaning in NLP.
- The roots of the model lie in the 1950s when two big ideas converged: Osgood's 1957 idea is to use a point in three-dimensional space to represent the connotation of a word, and the proposal by linguists like Joos (1950), Harris (1954), and Firth (1957) to define the meaning of a word by its distribution in language use, meaning its neighboring words or grammatical environments.
- Their idea was that two words that occur in very similar distributions (whose neighboring words are similar) have similar meanings.

Lexical Semantics

- How should we represent the meaning of a word?

- It should tell us that:
  - some words have similar meanings (cat is similar to dog)
  - others are antonyms (cold is the opposite of hot)
  - some have positive connotations (happy)
  - while others have negative connotations (sad).

Words and Vectors

- The term-document matrix

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

Figure 6.2 The term-document matrix for four words in four Shakespeare plays. Each cell contains the number of times the (row) word occurs in the (column) document.

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

Figure 6.3 The term-document matrix for four words in four Shakespeare plays. The red boxes show that each document is represented as a column vector of length four.

Words and Vectors

- Term-document matrices were originally defined as a means of finding similar documents for the task of document information retrieval.
- Two documents that are similar will tend to have similar words, and if two documents have similar words their column vectors will tend to be similar.

Words and Vectors

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

Figure 6.3 The term-document matrix for four words in four Shakespeare plays. The red boxes show that each document is represented as a column vector of length four.

- The vectors for the comedies As You Like It [1,114,36,20] and Twelfth Night [0,80,58,15] look a lot more like each other (more fools and wit than battles) than they look like Julius Caesar [7,62,1,2] or Henry V [13,89,4,3].

Words and Vectors

- We can think of the vector for a document as a point in |V|-dimensional space; thus the documents in Fig. 6.3 are points in 4-dimensional space.
- Since 4-dimensional spaces are hard to visualize, Fig. 6.4 shows a visualization in two dimensions; we've arbitrarily chosen the dimensions corresponding to the words battle and fool.

Words and Vectors

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

Figure 6.3 The term-document matrix for four words in four Shakespeare plays. The red boxes show that each document is represented as a column vector of length four.

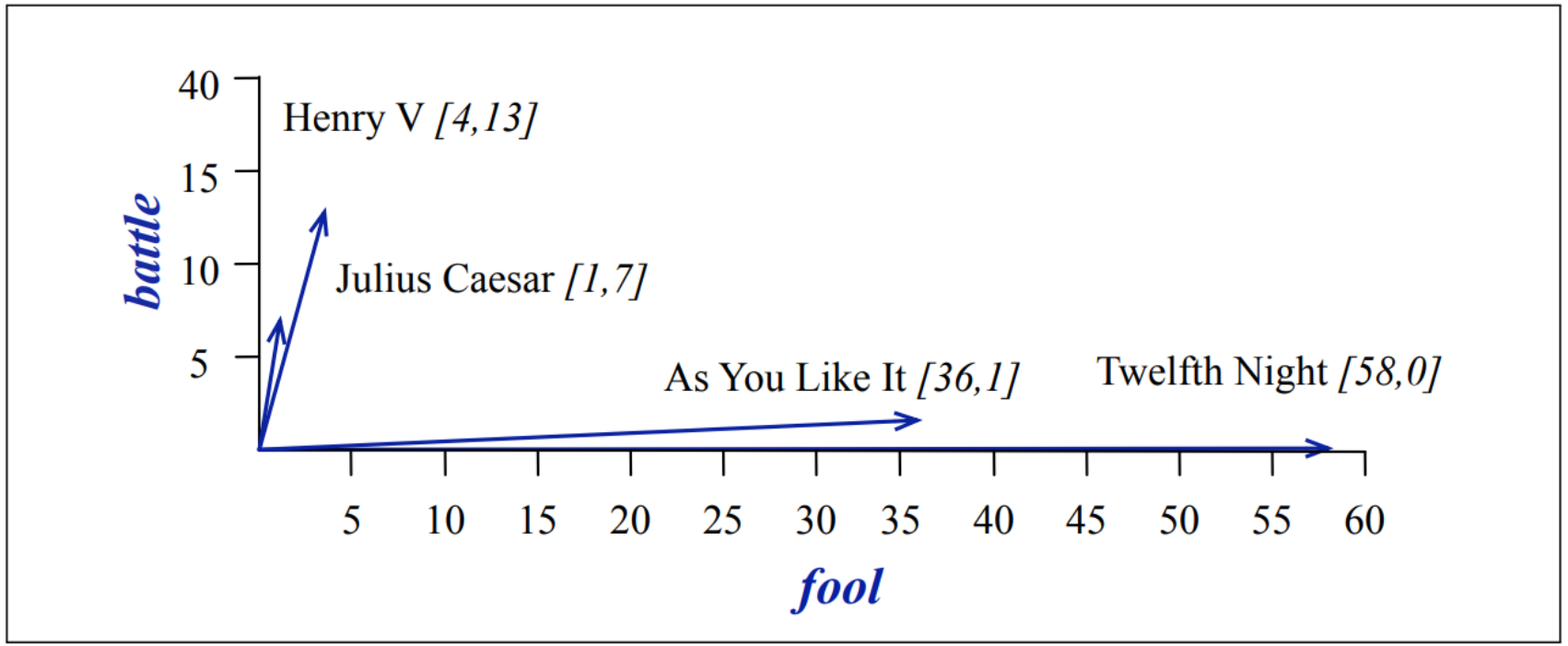


Figure 6.4 A spatial visualization of the document vectors for the four Shakespeare play documents, showing just two of the dimensions, corresponding to the words *battle* and *fool*. The comedies have high values for the *fool* dimension and low values for the *battle* dimension.

Words as vectors: document dimensions

- We've seen that documents can be represented as vectors in a vector space.
- But vector semantics can also be used to represent the meaning of words.
- We do this by associating each word with a word vector— a row vector rather than a column vector, hence with different dimensions, as shown in Fig. 6.5.

Words as vectors: document dimensions

- The four dimensions of the vector for fool, [36,58,1,4], correspond to the four Shakespeare plays.
- Word counts in the same four dimensions are used to form the vectors for the other 3 words: wit, [20,15,2,3]; battle, [1,0,7,13]; and good [114,80,62,89].

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

Figure 6.5 The term-document matrix for four words in four Shakespeare plays. The red boxes show that each word is represented as a row vector of length four.

Word Meaning

- Osgood et al. (1957) noticed that in using these 3 numbers to represent the meaning of a word, the model was representing each word as a point in a three dimensional space, a vector whose three dimensions corresponded to the word's rating on the three scales.
- This revolutionary idea that word meaning could be represented as a point in space (e.g., that part of the meaning of heartbreak can be represented as the point [2.45,5.65,3.58]) was the first expression of the vector semantics models that we introduce next.

Personality Embeddings: What are you like?

- On a scale of 0 to 100, how introverted/extraverted are you (where 0 is the most introverted, and 100 is the most extraverted)?

Word Meaning

	Valence	Arousal	Dominance
courageous	8.05	5.5	7.38
music	7.67	5.57	6.5
heartbreak	2.45	5.65	3.58
cub	6.71	3.95	4.24

valence: the pleasantness of the stimulus

arousal: the intensity of emotion provoked by the stimulus

dominance: the degree of control exerted by the stimulus

Embeddings

- The idea of vector semantics is to represent a word as a point in a multidimensional semantic space that is derived (in ways we'll see) from the distributions of embeddings word neighbors.
- Vectors for representing words are called embeddings (although the term is sometimes more strictly applied only to dense vectors like word2vec, rather than sparse tf-idf or PPMI vectors.
- The word "embedding" derives from its mathematical sense as a mapping from one space or structure to another, although the meaning has shifted;

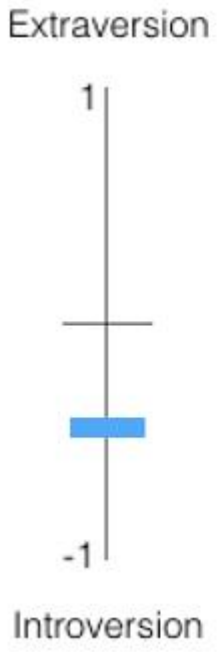
Embeddings

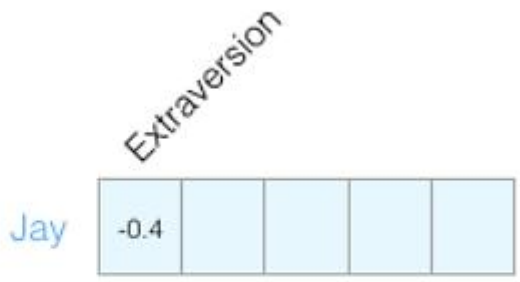
Openness to experience	79 out of 100
Agreeableness	75 out of 100
Conscientiousness	42 out of 100
Negative emotionality	50 out of 100
Extraversion	58 out of 100

Embeddings

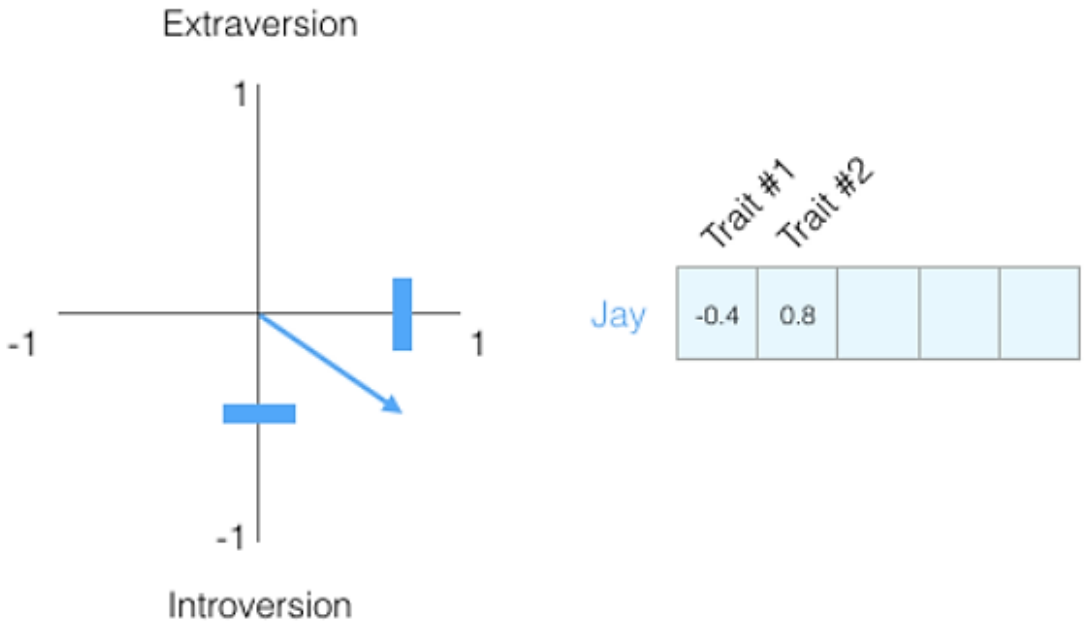

Extraversion
100
0
Introversion

Extraversion
Jay
38


Extraversion
1
-1
Introversion


Extraversion
Jay
-0.4

Embeddings



Embeddings

	pie	data	computer
cherry	442	8	2
digital	5	1683	1670
information	5	3982	3325

	Valence	Arousal	Dominance
courageous	8.05	5.5	7.38
music	7.67	5.57	6.5
heartbreak	2.45	5.65	3.58
cub	6.71	3.95	4.24

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

Figure 6.3 The term-document matrix for four words in four Shakespeare plays. The red boxes show that each document is represented as a column vector of length four.

Cosine for measuring similarity

- To measure similarity between two target words v and w, we need a metric that takes two vectors (of the same dimensionality, either both with words as dimensions, hence of length |V|, or both with documents as dimensions, of length |D|) and gives a measure of their similarity. By far the most common similarity metric is the cosine of the angle between the vectors.

$$\text{dot product}(\mathbf{v},\mathbf{w}) = \mathbf{v}\cdot\mathbf{w} = \sum_{i=1}^{N} v_i w_i = v_1 w_1 + v_2 w_2 + \ldots + v_N w_N$$

Cosine for measuring similarity

$$\text{dot product}(\mathbf{v},\mathbf{w}) = \mathbf{v}\cdot\mathbf{w} = \sum_{i=1}^{N} v_i w_i = v_1 w_1 + v_2 w_2 + \ldots + v_N w_N$$

The dot product acts as a similarity metric because it will tend to be high just when the two vectors have large values in the same dimensions.

Cosine for measuring similarity

- This raw dot product, however, has a problem as a similarity metric: it favors vector length long vectors.
- The vector length is defined as:

$$|\mathbf{v}| = \sqrt{\sum_{i=1}^{N} v_i^2}$$

Cosine for measuring similarity

$$\mathbf{a} \cdot \mathbf{b} = |\mathbf{a}||\mathbf{b}| \cos\theta$$

$$\frac{\mathbf{a} \cdot \mathbf{b}}{|\mathbf{a}||\mathbf{b}|} = \cos\theta$$

$$\text{cosine}(\mathbf{v}, \mathbf{w}) = \frac{\mathbf{v} \cdot \mathbf{w}}{|\mathbf{v}||\mathbf{w}|} = \frac{\sum_{i=1}^{N} v_i w_i}{\sqrt{\sum_{i=1}^{N} v_i^2} \sqrt{\sum_{i=1}^{N} w_i^2}}$$

Cosine for measuring similarity

	pie	data	computer
cherry	442	8	2
digital	5	1683	1670
information	5	3982	3325

$$\text{cosine}(\mathbf{v},\mathbf{w}) = \frac{\mathbf{v}\cdot\mathbf{w}}{|\mathbf{v}||\mathbf{w}|} = \frac{\sum_{i=1}^{N} v_i w_i}{\sqrt{\sum_{i=1}^{N} v_i^2}\sqrt{\sum_{i=1}^{N} w_i^2}}$$

$$\cos(\text{cherry},\text{information}) = \frac{442*5+8*3982+2*3325}{\sqrt{442^2+8^2+2^2}\sqrt{5^2+3982^2+3325^2}} = .018$$

$$\cos(\text{digital},\text{information}) = \frac{5*5+1683*3982+1670*3325}{\sqrt{5^2+1683^2+1670^2}\sqrt{5^2+3982^2+3325^2}} = .996$$

Cosine for measuring similarity

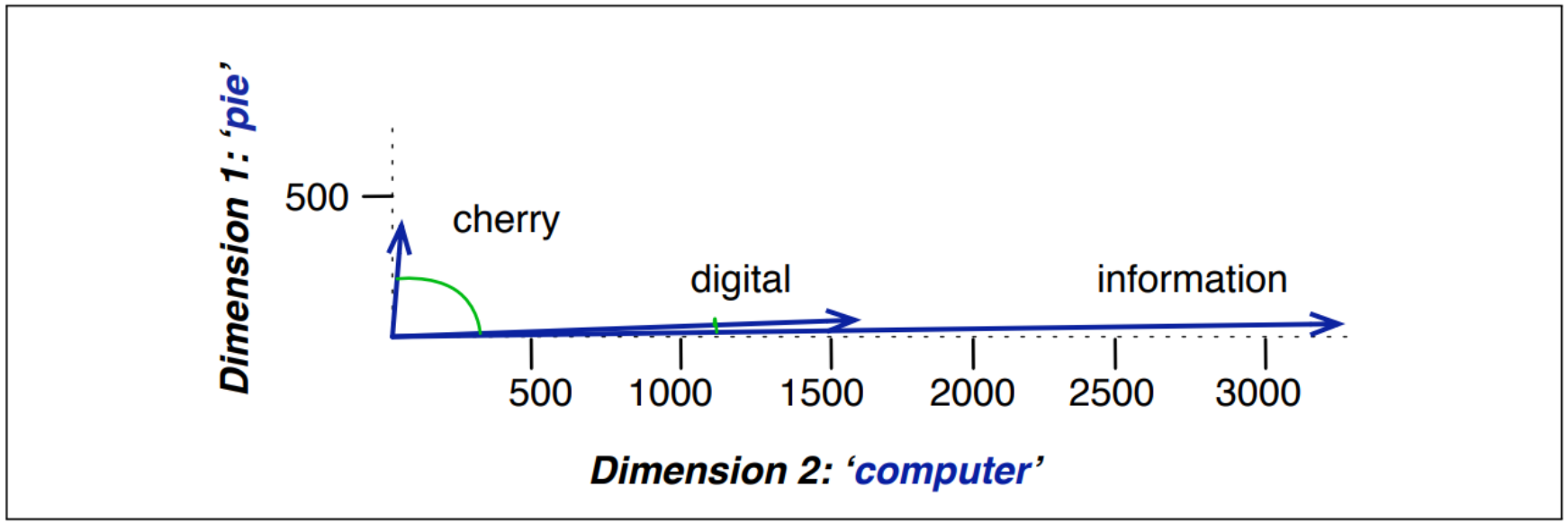
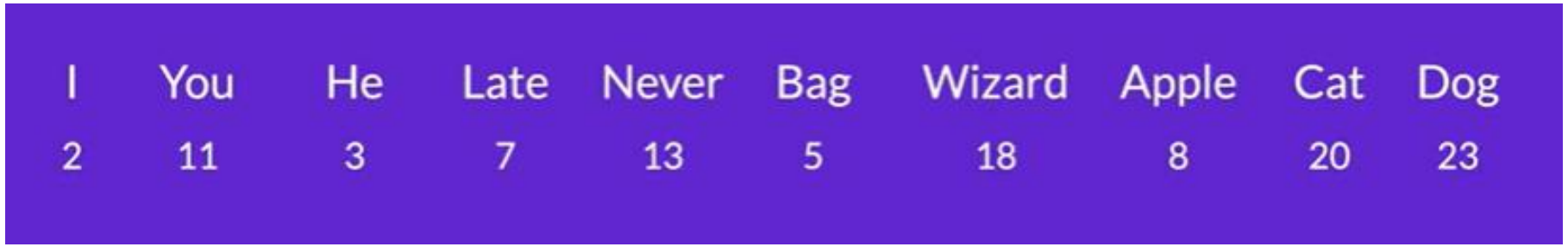
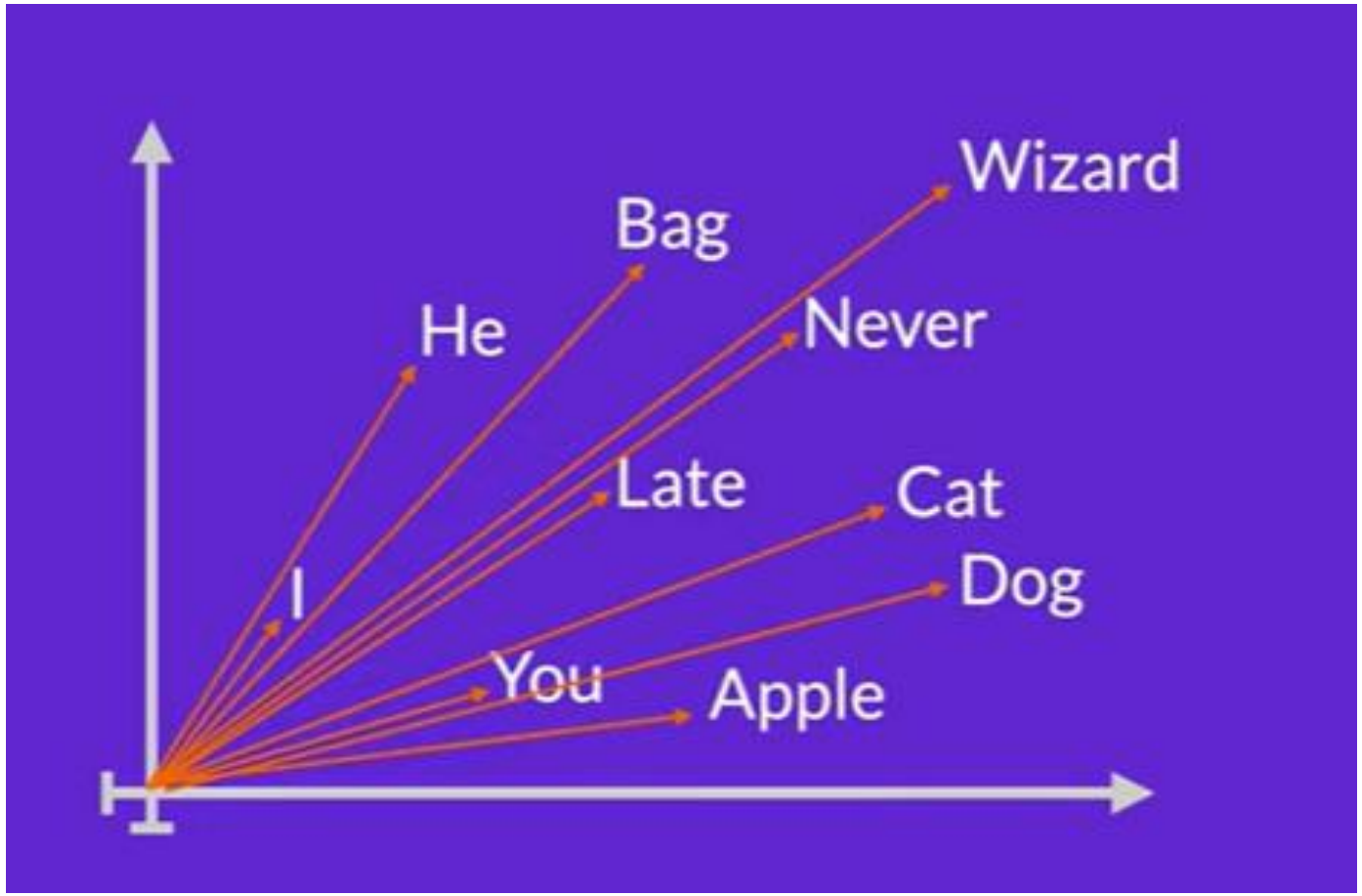


Figure 6.8 A (rough) graphical demonstration of cosine similarity, showing vectors for three words (*cherry*, *digital*, and *information*) in the two dimensional space defined by counts of the words *computer* and *pie* nearby. The figure doesn't show the cosine, but it highlights the angles; note that the angle between *digital* and *information* is smaller than the angle between *cherry* and *information*. When two vectors are more similar, the cosine is larger but the angle is smaller; the cosine has its maximum (1) when the angle between two vectors is smallest (0°); the cosine of all other angles is less than 1.

Embedding Spcae

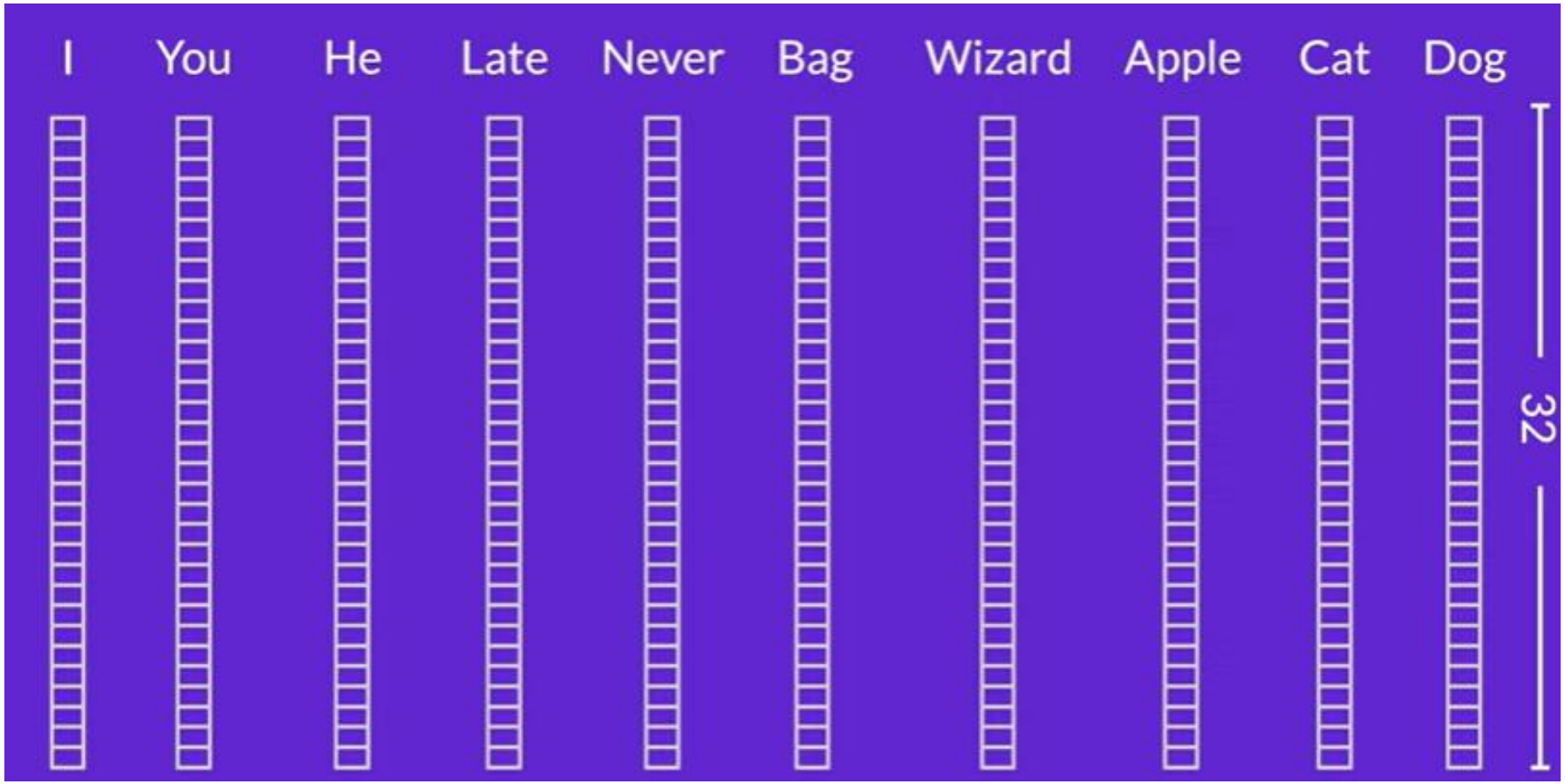


Word Embeddings – 2D



Word Embeddings – 3D

Word Embeddings – 32D



Word Embeddings

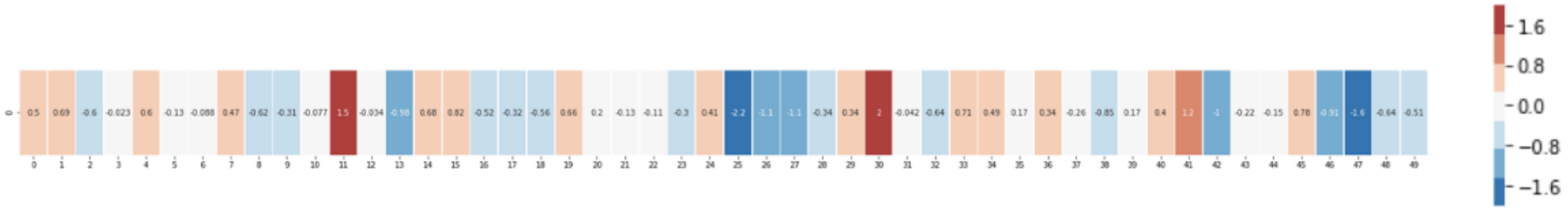
Word Embeddings

- In natural language processing, a word embedding is a representation of a word.
- Typically, the representation is a real-valued vector that encodes the meaning of the word in such a way that the words that are closer in the vector space are expected to be similar in meaning.

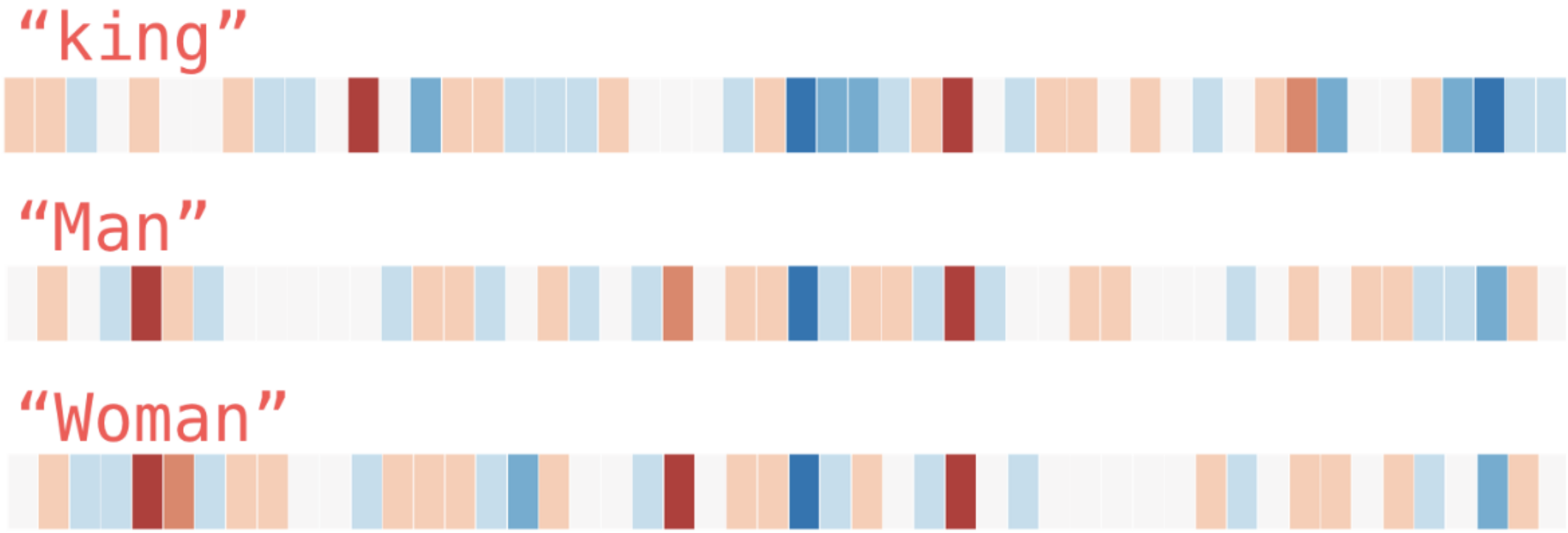
Word Embeddings

King:

```
[ 0.50451 , 0.68607 , -0.59517 , -0.022801, 0.60046 , -0.13498 , -0.08813 , 0.47377 , -0.61798 , -0.31012 ,
-0.076666, 1.493 , -0.034189, -0.98173 , 0.68229 , 0.81722 , -0.51874 , -0.31503 , -0.55809 , 0.66421 , 0.1961
, -0.13495 , -0.11476 , -0.30344 , 0.41177 , -2.223 , -1.0756 , -1.0783 , -0.34354 , 0.33505 , 1.9927 ,
-0.04234 , -0.64319 , 0.71125 , 0.49159 , 0.16754 , 0.34344 , -0.25663 , -0.8523 , 0.1661 , 0.40102 , 1.1685 ,
-1.0137 , -0.21585 , -0.15155 , 0.78321 , -0.91241 , -1.6106 , -0.64426 , -0.51042 ]
```



Word Embeddings



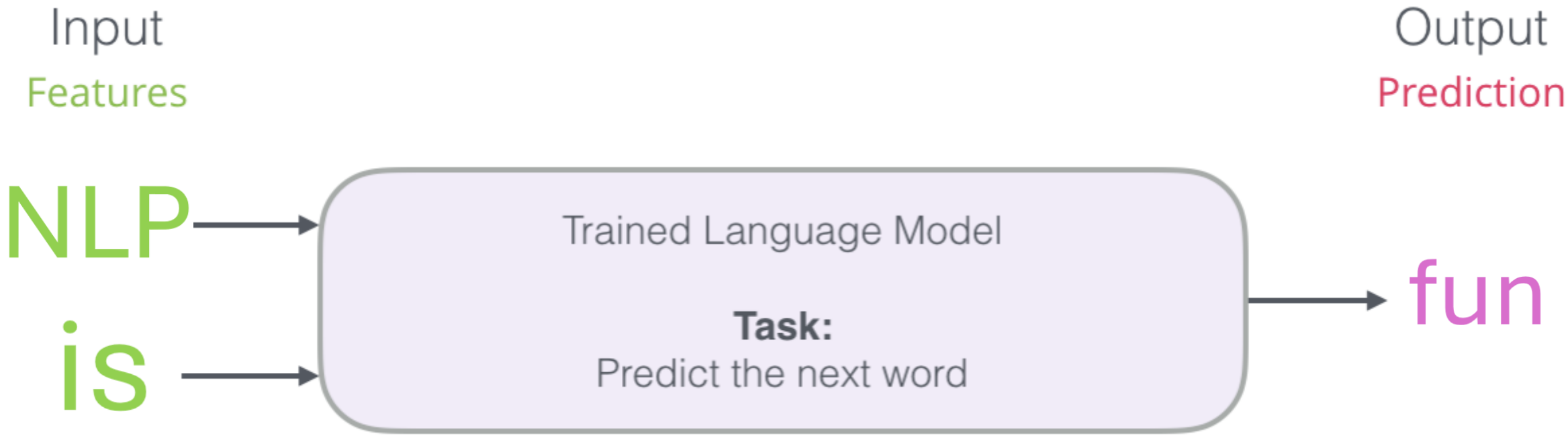
Word Embeddings

- Next-word prediction is a task that can be addressed by a language model.

- A language model can take a list of words (let’s say two words) and attempt to predict the word that follows them.

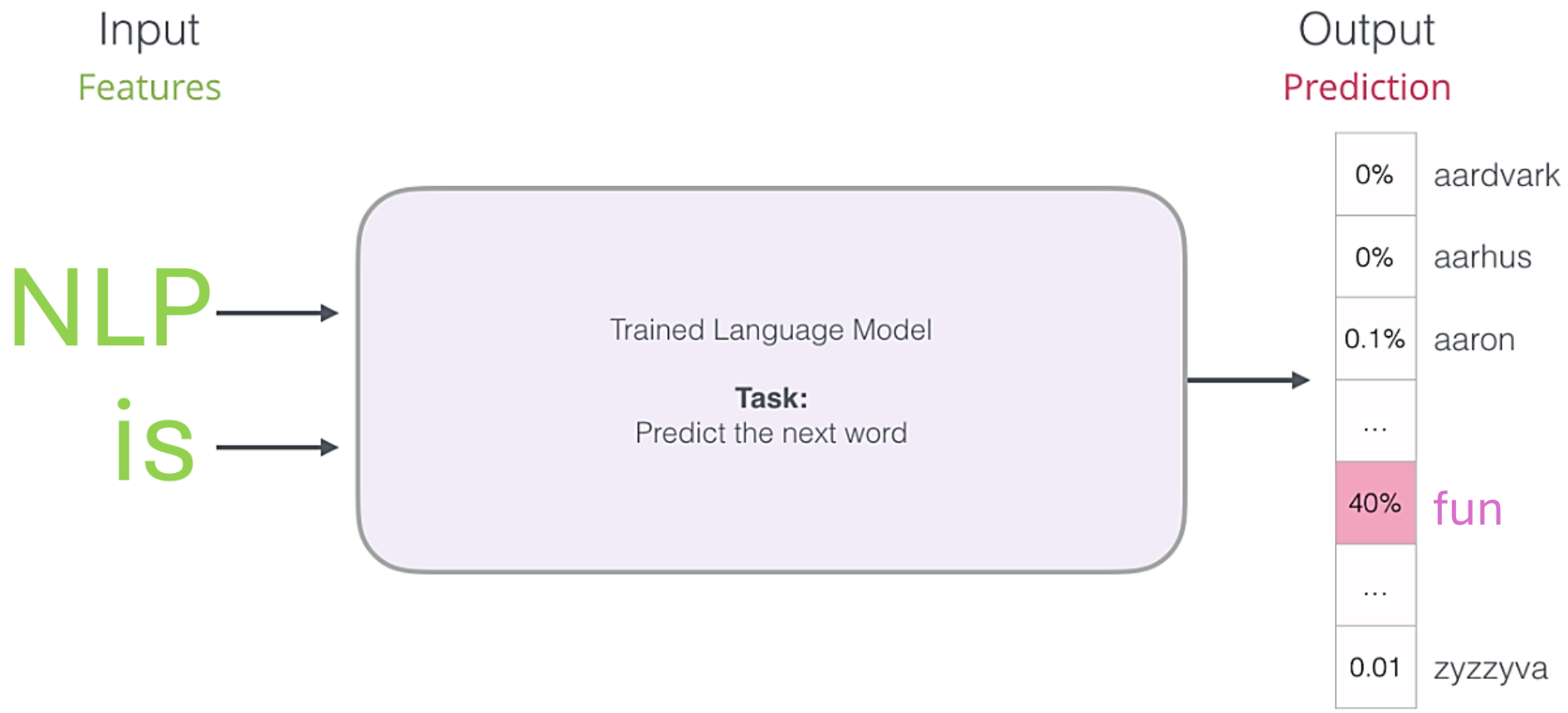
Word Embeddings

Word Embeddings



Word Embeddings

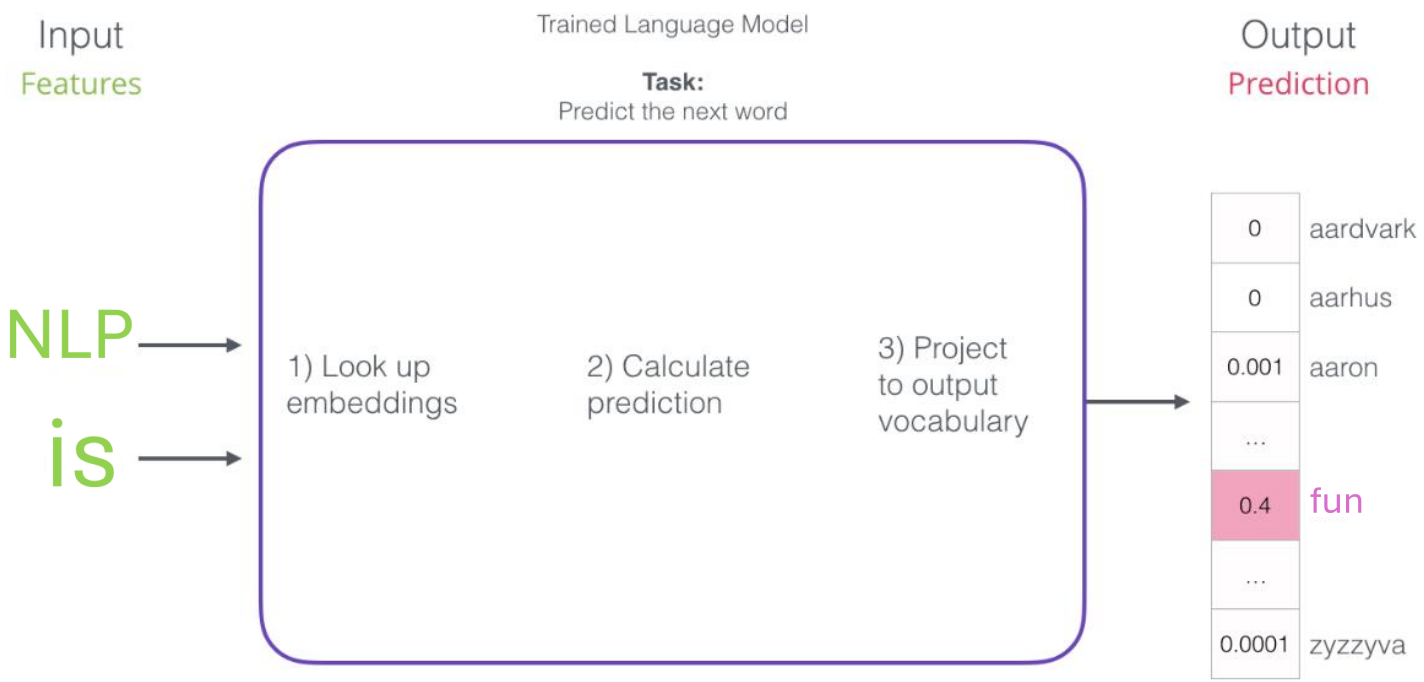
- But in practice, the model doesn’t output only one word.
- It actually outputs a probability score for all the words it knows (the model’s “vocabulary”, which can range from a few thousand to over a million words).
- The keyboard application then has to find the words with the highest scores, and present those to the user.



The output of the neural language model is a probability score for all the words the model knows.

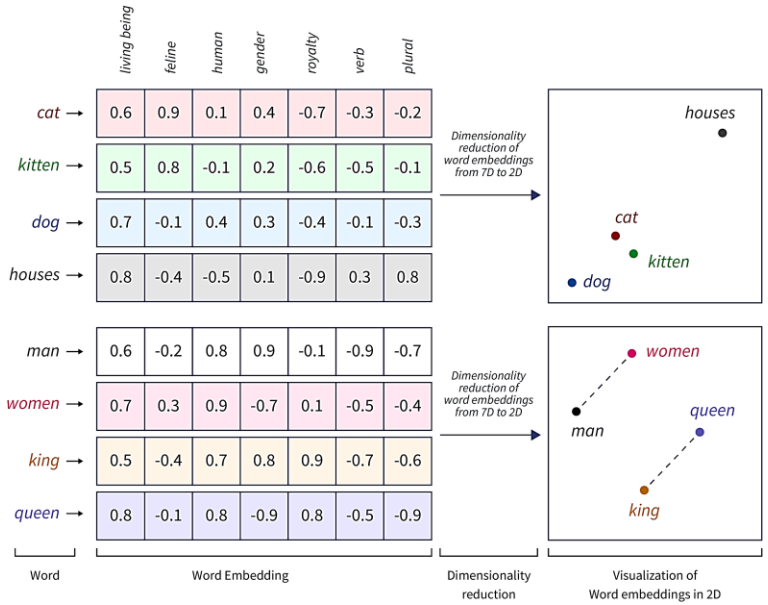
Neural Language Model

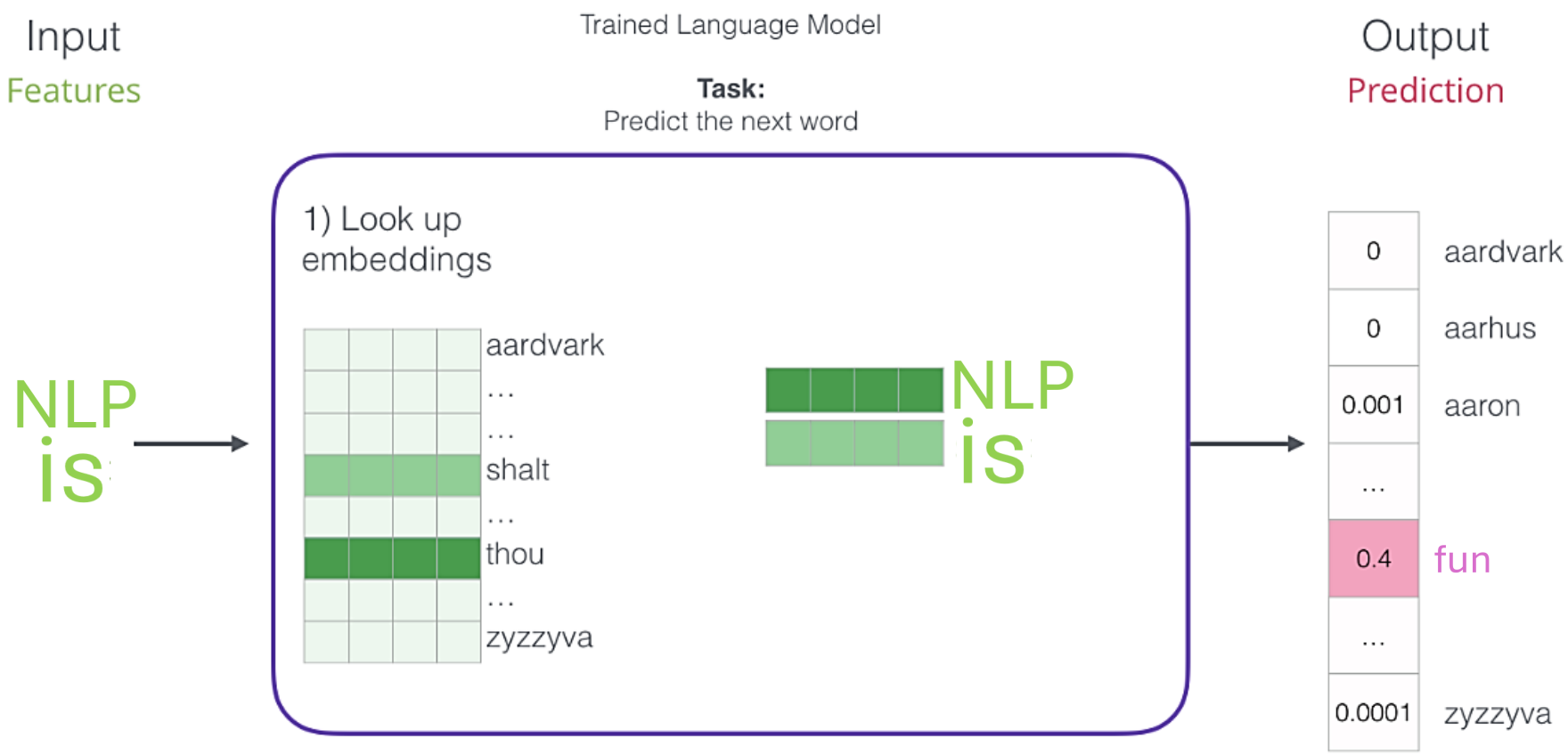
- After being trained, early neural language models (Bengio 2003) would calculate a prediction in three steps:



Neural Language Model

- After training, we need a matrix that contains an embedding for each word in our vocabulary.
- During prediction time, we just use these embeddings to calculate the prediction.




Input
Features
NLP
is
Trained Language Model
Task:
Predict the next word
1) Look up
embeddings
aardvark
…
…
shalt
…
thou
…
zyzzyva
NLP
is
Output
Prediction
0
aardvark
0
aarhus
0.001
aaron
…
0.4
fun
…
0.0001
zyzzyva

Language Model Training

- Let’s now turn to the training process to learn more about how this embedding matrix was developed.

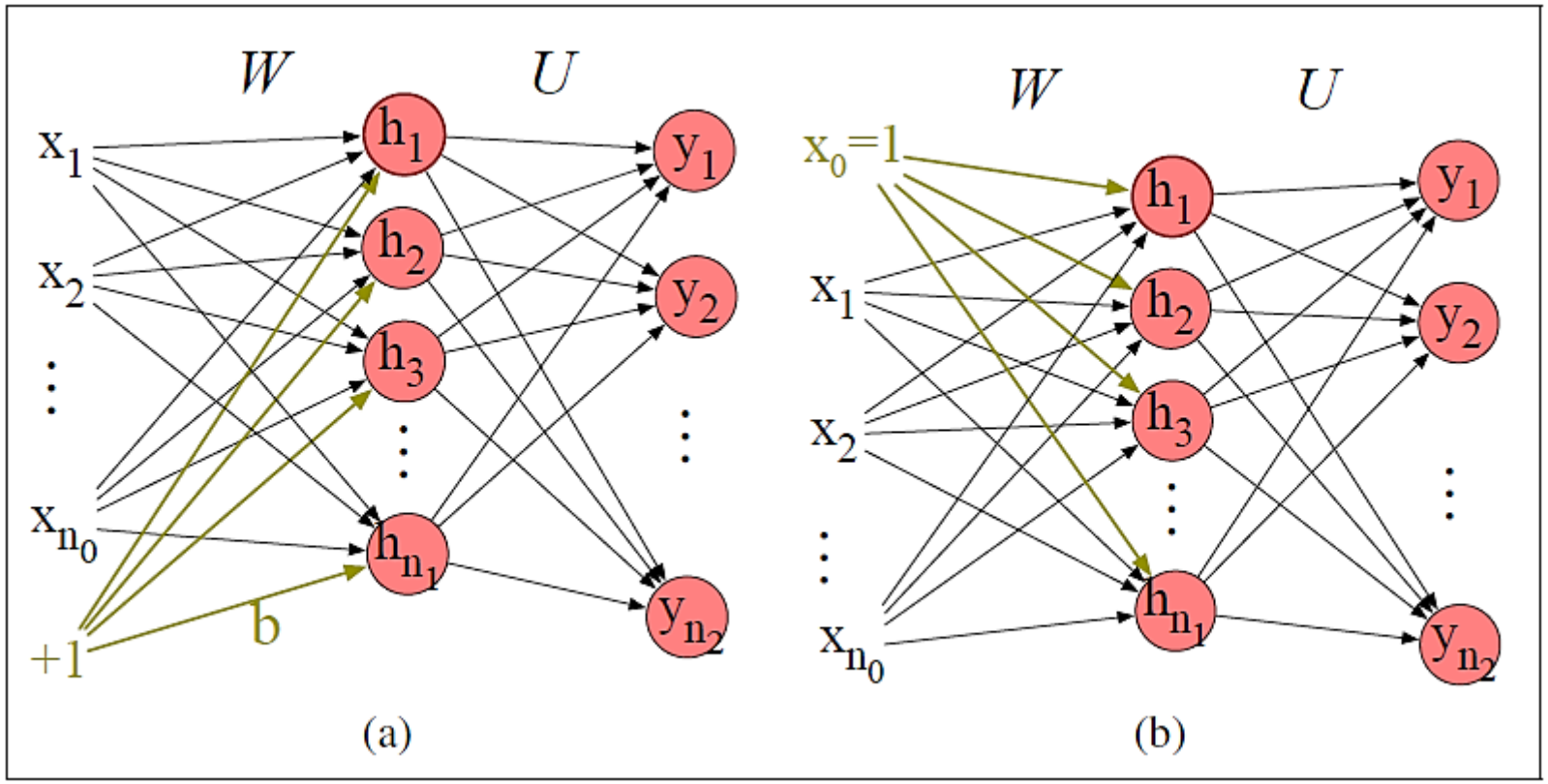
Neural Networks for Text Classification

- Let's see how to apply feedforward networks to NLP tasks!

- Let's begin with a simple 2-layer sentiment classifier.

$$
\begin{aligned}
\mathbf{x} &= [\mathbf{x}_1, \mathbf{x}_2, \ldots \mathbf{x}_N] \quad \text{(each } \mathbf{x}_i \text{ is a hand-designed feature)} \\
\mathbf{h} &= \sigma(\mathbf{W}\mathbf{x} + \mathbf{b}) \\
\mathbf{z} &= \mathbf{U}\mathbf{h} \\
\hat{\mathbf{y}} &= \text{softmax}(\mathbf{z})
\end{aligned}
$$

Neural Networks for Text Classification



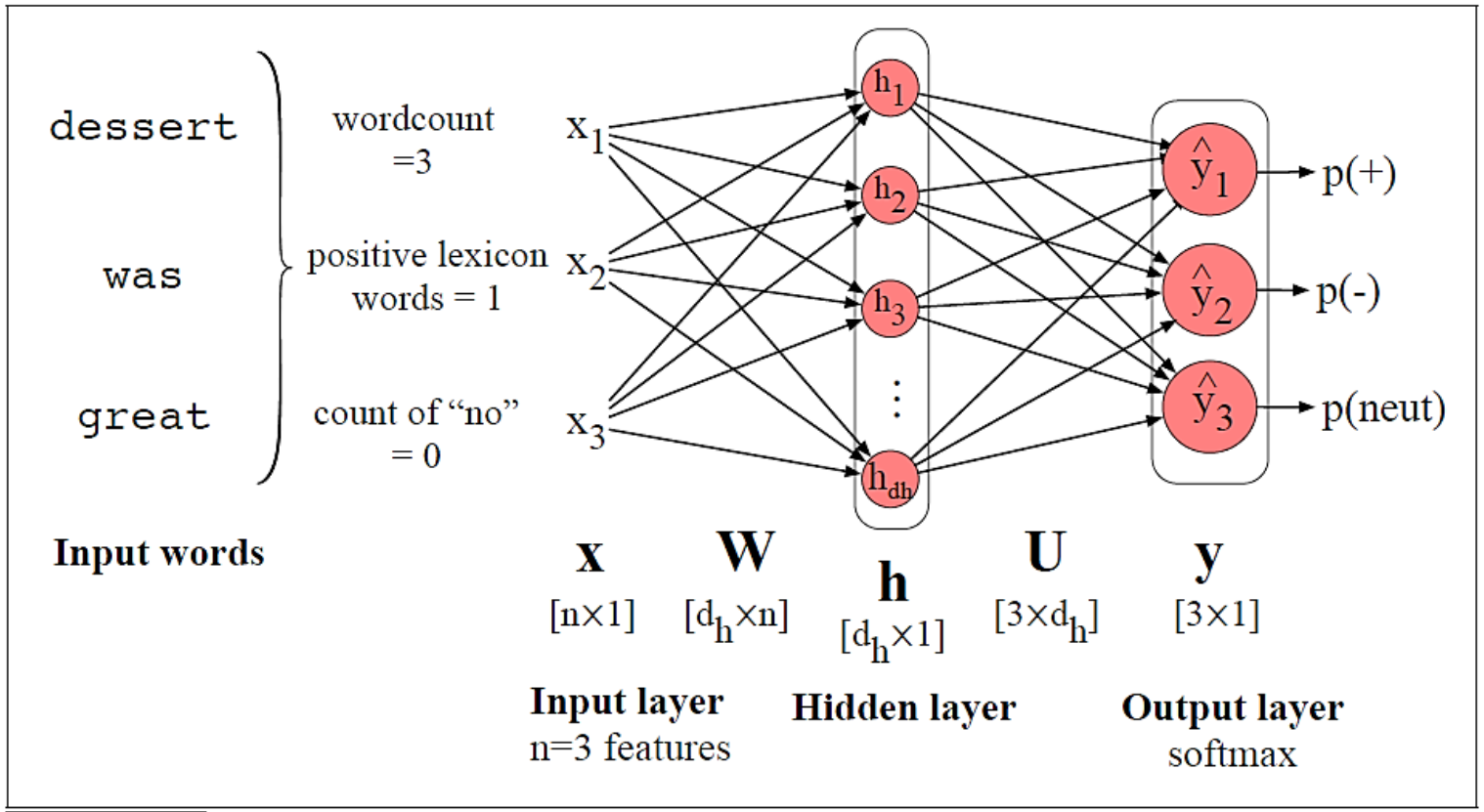


Figure 7.10 Feedforward network sentiment analysis using traditional hand-built features of the input text.

$$\begin{aligned}
\mathbf{x} &= [\mathbf{x}_1, \mathbf{x}_2, \ldots \mathbf{x}_N] \quad \text{(each } \mathbf{x}_i \text{ is a hand-designed feature)} \\
\mathbf{h} &= \sigma(\mathbf{W}\mathbf{x} + \mathbf{b}) \\
\mathbf{z} &= \mathbf{U}\mathbf{h} \\
\hat{\mathbf{y}} &= \text{softmax}(\mathbf{z})
\end{aligned}$$

Neural Networks for Text Classification

- How to represent input to neural network?
- There are various ways to represent an input for classification
- One simple baseline pooling is to apply some sort of pooling function to the embeddings of all the words in the input.
- For example, for a text with n input words/tokens w_1, ..., w_n, we can turn the n embeddings $e(w_1)$, ..., $e(w_n)$ (each of dimensionality d) into a single embedding also of dimensionality d by just summing the embeddings, or by taking their mean (summing and then dividing by n):

$$\mathbf{x}_{mean} = \frac{1}{n}\sum_{i=1}^{n}\mathbf{e}(w_i)$$

Neural Networks for Text Classification

- There are many other options, like taking the element-wise max.
- The element-wise max of a set of n vectors is a new vector whose kth element is the max of the kth elements of all the n vectors.
- Here are the equations for this classifier assuming mean pooling.

$$
\begin{aligned}
\mathbf{x} &= \text{mean}(\mathbf{e}(w_1), \mathbf{e}(w_2), \ldots, \mathbf{e}(w_n)) \\
\mathbf{h} &= \sigma(\mathbf{W}\mathbf{x} + \mathbf{b}) \\
\mathbf{z} &= \mathbf{U}\mathbf{h} \\
\hat{\mathbf{y}} &= \text{softmax}(\mathbf{z})
\end{aligned}
$$

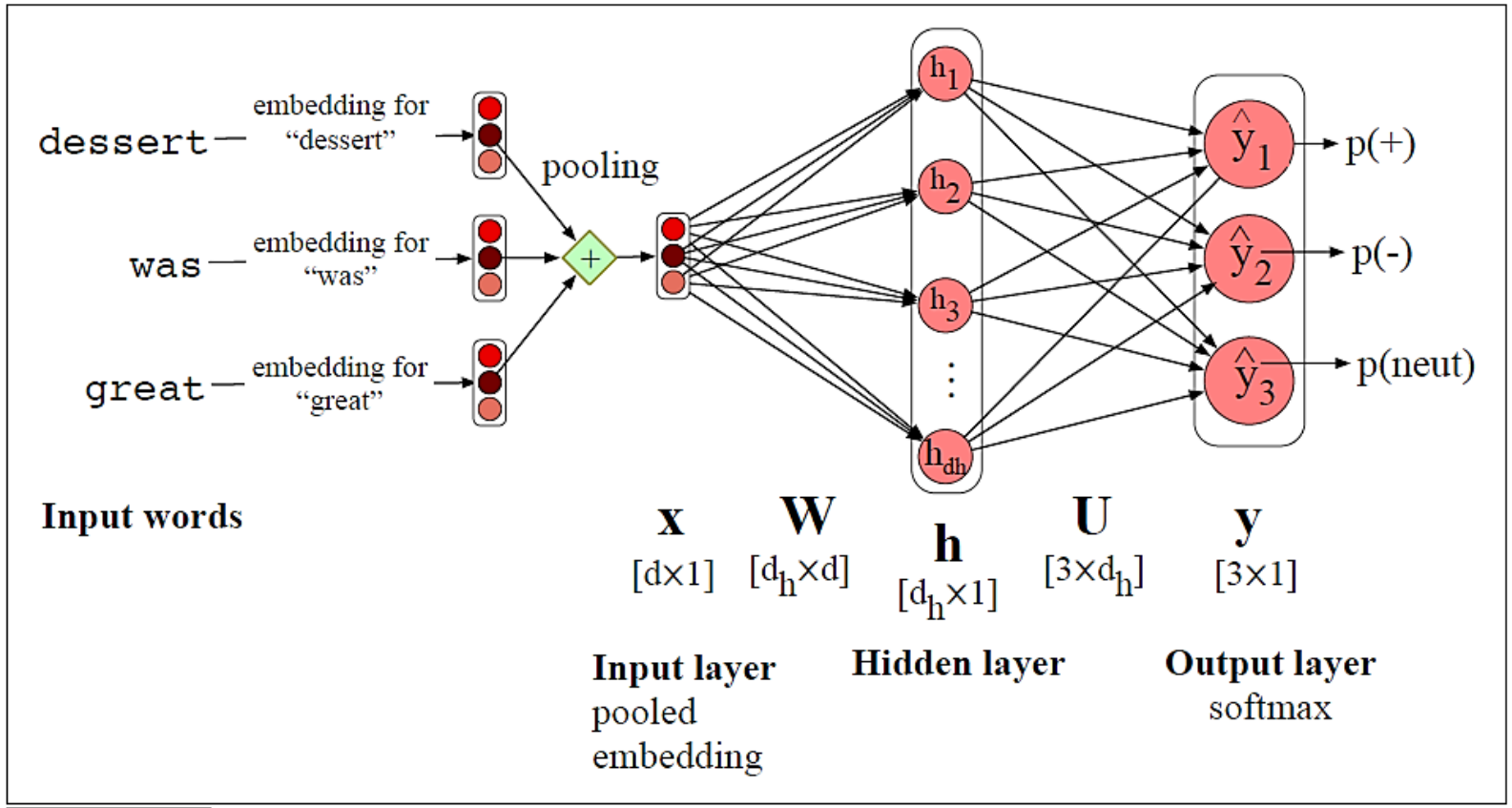


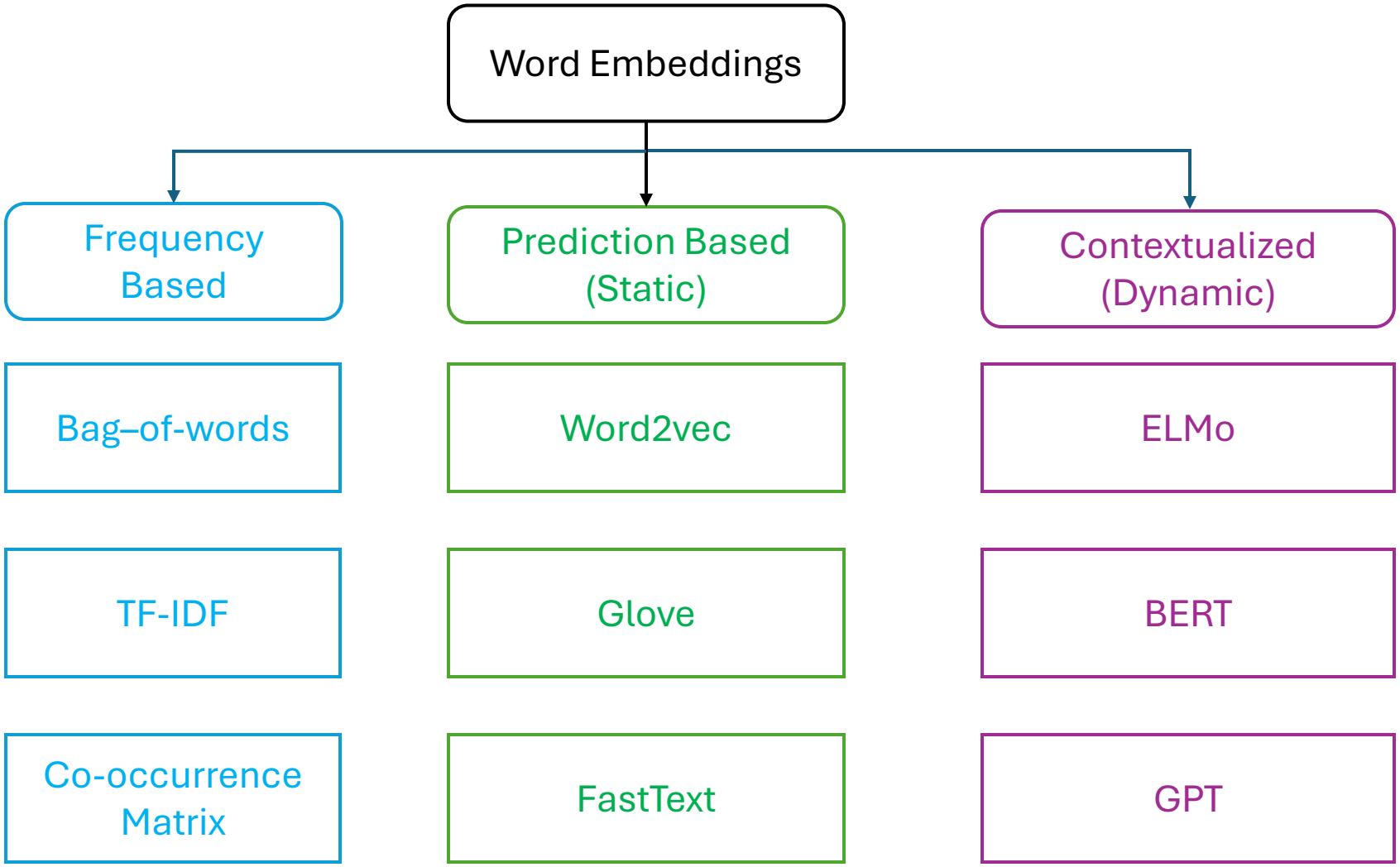
Figure 7.11 Feedforward network sentiment analysis using a pooled embedding of the input words.

$$\begin{aligned}
\mathbf{x} &= \text{mean}(\mathbf{e}(w_1), \mathbf{e}(w_2), \ldots, \mathbf{e}(w_n)) \\
\mathbf{h} &= \sigma(\mathbf{W}\mathbf{x} + \mathbf{b}) \\
\mathbf{z} &= \mathbf{U}\mathbf{h} \\
\hat{\mathbf{y}} &= \text{softmax}(\mathbf{z})
\end{aligned}$$

Pretraining

- Pretraining: The idea of using word2vec or GloVe embeddings as our input representation— and more generally the idea of relying on another algorithm to have already learned an embedding representation for our input words—is called pretraining.
- Pretraining refers to the process of training a model or a part of a model on a large corpus of data before it is fine-tuned for a specific task.
- In the context of neural networks and natural language processing (NLP), pretraining usually means training word representations (embeddings) on a large dataset, then using these pretrained representations for specific downstream tasks, like text classification or sentiment analysis.

Word Embeddings



Embedding Types

- Frequency-based embeddings
  - Count Vectors
  - TF-IDF (Term Frequency-Inverse Document Frequency)
- Prediction-based Static Embeddings
  - Word2Vec
  - FastText
  - Glove (Global Vectors for Word Representation)
- Pre-trained Language Model-based Dynamic Embeddings
  - ELMo (Embeddings from Language Models)
  - BERT (Bidirectional Encoder Representations from Transformers)
  - GPT (Generative Pre-trained Transformer):

Language Model Training

- We get a lot of text data (say, all Wikipedia articles).
- We have a window (say, of three words) that we slide against all of that text.
- The sliding window generates training samples for our model

The **Hyperion Cantos** is a series of science fiction novels by Dan Simmons. The title ... in the series, *Hyperion* and *The Fall of Hyperion*,[1][2] and later came to refer to the overall storyline, including *Endymion*, *The Rise of Endymion*, and a number of short stories.[3][4] More narrowly, inside the fictional storyline, after the first volume, the Hyperion Cantos is an epic poem written by the character Martin Silenus covering in verse form the events of the first book.[5]

Of the four novels, *Hyperion* received the Hugo and Locus Awards in 1990;[6] *The Fall of Hyperion* won the Locus and British Science Fiction Association Awards in 1991;[7] and *The Rise of Endymion* received the Locus Award in 1998.[8] All four novels were also nominated for various science fiction awards.

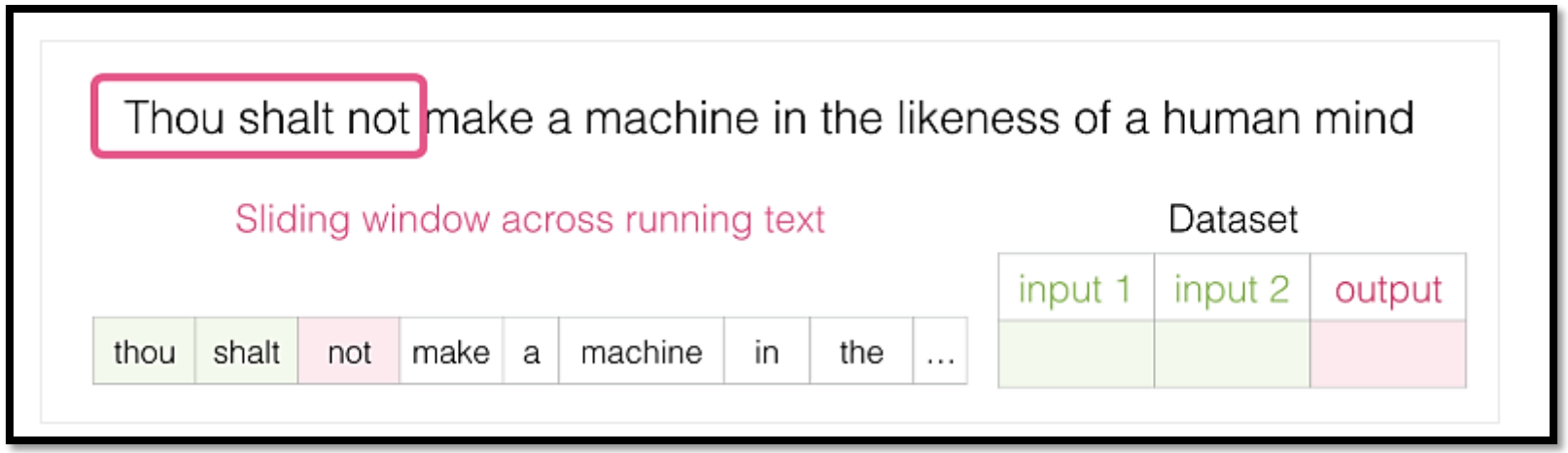
An event series is being developed by Bradley Cooper, Graham King, and Todd Phillips for Syfy based on the first novel *Hyperion*.[9]

Dune is a 1965 science fiction novel by American author Frank Herbert, originally published as two separate serials in *Analog* magazine. It tied with Roger Zelazny's *This Immortal* for the Hugo Award in 1966,[3] and it won the inaugural Nebula Award for Best Novel.[4] It is the first installment of the *Dune* saga, and in 2003 was cited as the world's best-selling science fiction novel.[5][6]

Set in the distant future amidst a feudal interstellar society in which noble houses, in control of individual planets, owe allegiance to the Padishah Emperor, *Dune* tells the story of young Paul Atreides, whose noble family accepts the stewardship of the planet Arrakis. It is an inhospitable and sparsely populated desert wasteland, but is also the only source of melange, also known as "spice", a drug that enhances mental abilities. As melange is the most important and valuable substance in the universe, control of Arrakis is a coveted—and dangerous—undertaking. The story explores the multi-layered interactions of politics, religion, ecology, technology, and human emotion, as the factions of the empire confront each other in a struggle for the control of Arrakis

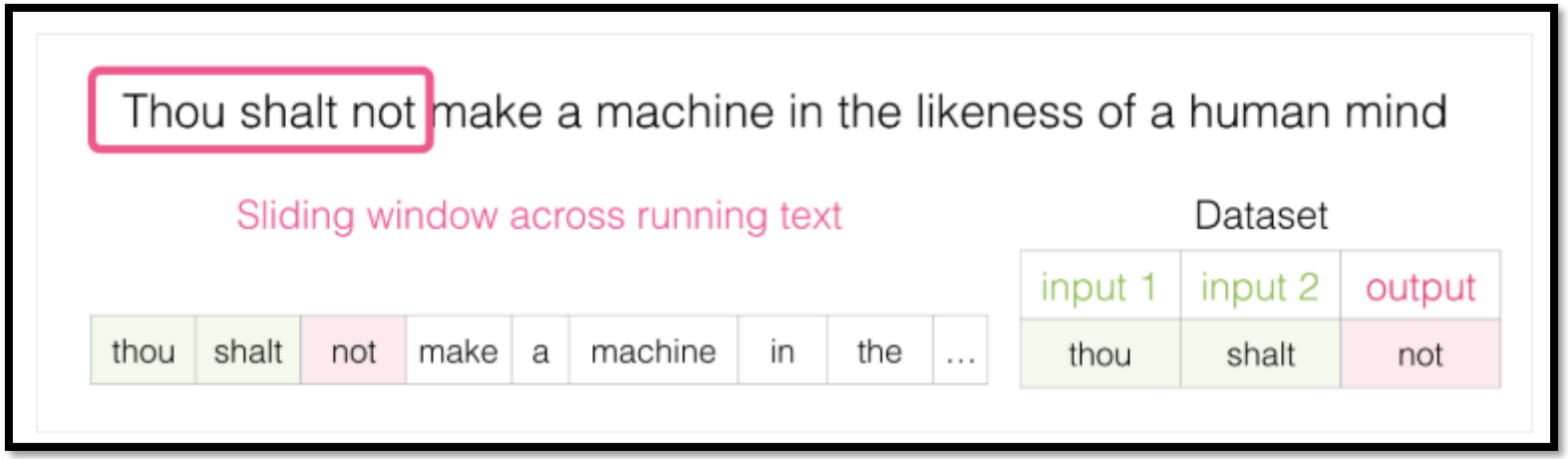
The Matrix is a 1999 science fiction action film written and directed by The Wachowskis[note 1] and starring Keanu Reeves, Laurence Fishburne, Carrie-Anne Moss, Hugo Weaving, and Joe Pantoliano. It depicts a dystopian "future in which reality as perceived by humans is actually" a simulated reality called "the Matrix," created by thought-capable machines (artificial beings)[note 2] "to subdue the human population, while their bodies' heat and electrical activity

- As this window slides against the text, we (virtually) generate a dataset that we use to train a model.
- To look exactly at how that's done, let's see how the sliding window processes this phrase:
- "Thou shalt not make a machine in the likeness of a human mind"
- When we start, the window is on the first three words of the sentence:



Language Model Training

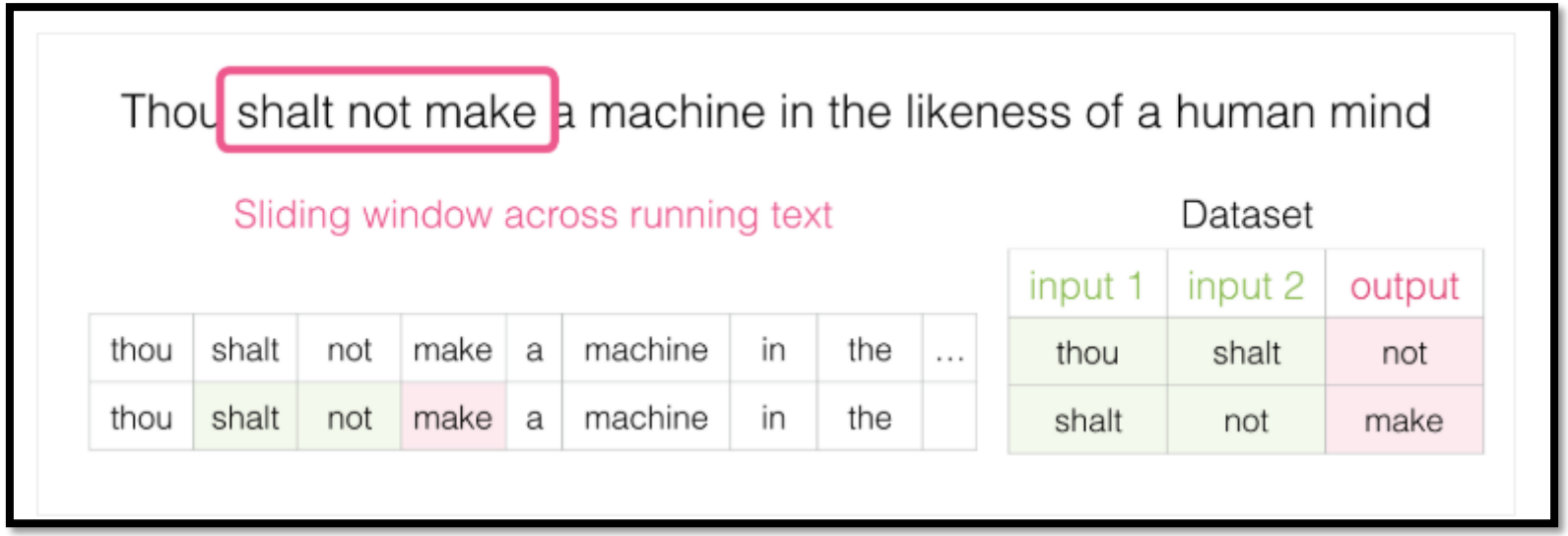
- We take the first two words to be features, and the third word to be a label:



- We now have generated the first sample in the dataset we can later use to train a language model.

Language Model Training

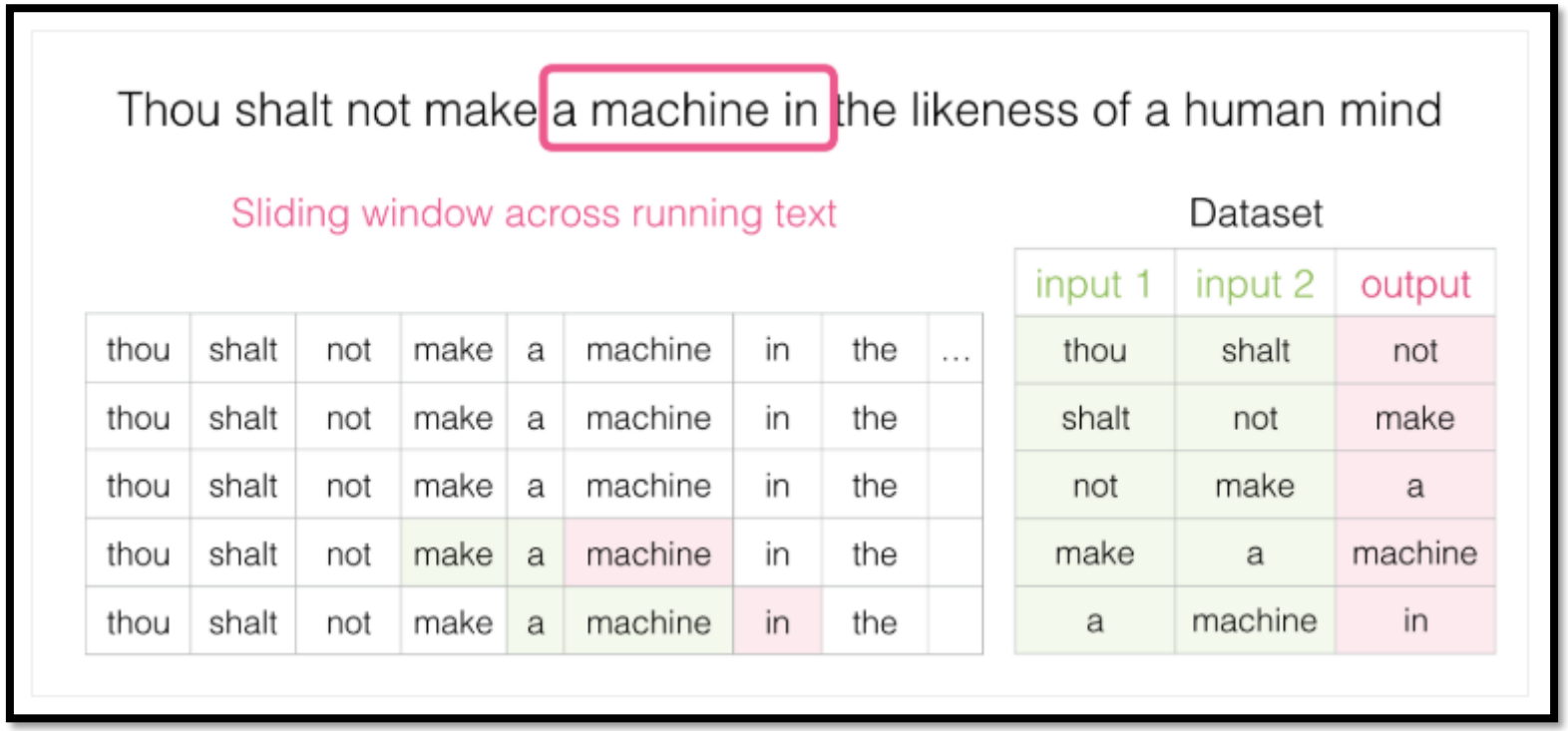
- We then slide our window to the next position and create a second sample:



- And the second example is now generated.

Language Model Training

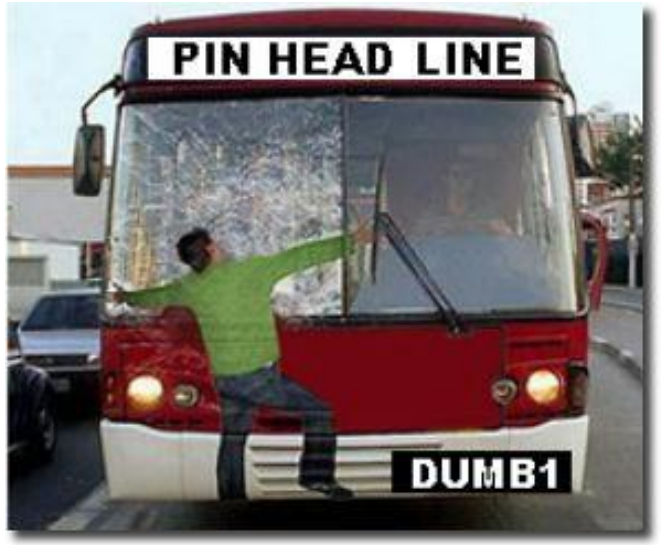
- And pretty soon we have a larger dataset of which words tend to appear after different pairs of words:



thou	shalt	not	make	a	machine	in	the	…
thou	shalt	not	make	a	machine	in	the	
thou	shalt	not	make	a	machine	in	the	
thou	shalt	not	make	a	machine	in	the	
thou	shalt	not	make	a	machine	in	the	

input 1	input 2	output
thou	shalt	not
shalt	not	make
not	make	a
make	a	machine
a	machine	in

Jay was hit by a _____



- The context I gave you here is five words before the blank word.
- I’m sure most people would guess the word bus goes into the blank.

Language Model Training

- But what if I gave you one more piece of information – a word after the blank, would that change your answer?

Jay was hit by a _____ bus

Language Model Training

- This completely changes what should go in the blank.
- the word red is now the most likely to go into the blank.
- What we learn from this is the words both before and after a specific word carry informational value.
- It turns out that accounting for both directions (words to the left and to the right of the word we’re guessing) leads to better word embeddings.
- Let’s see how we can adjust the way we’re training the model to account for this.

Language Model Training

- Instead of only looking two words before the target word, we can also look at two words after it.

Jay was hit by a _____ bus in...

by	a	red	bus	in

- If we do this, the dataset we're virtually building and training the model against would look like this:

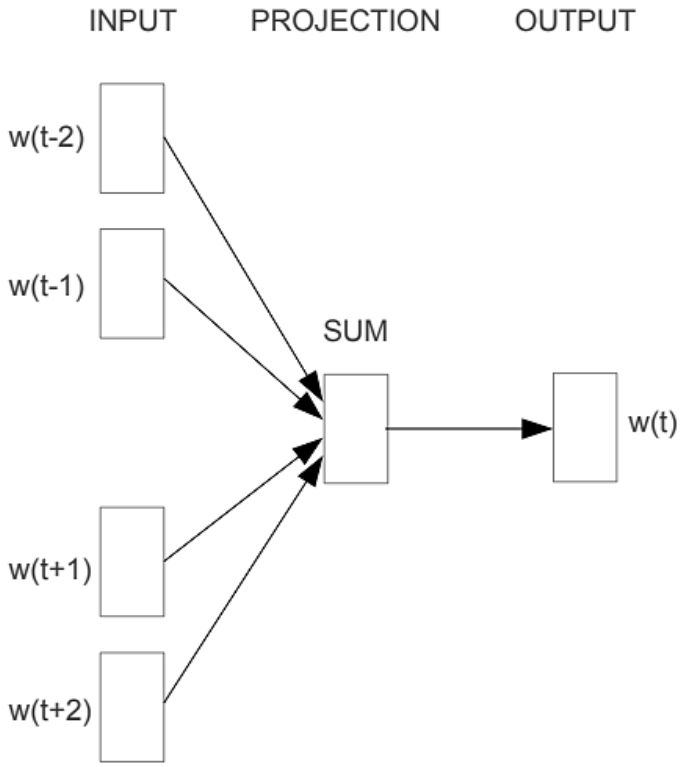
input 1	input 2	input 3	input 4	output
by	a	bus	in	red

Continuous Bag of Words

Jay was hit by a _____ bus in...

by	a	red	bus	in

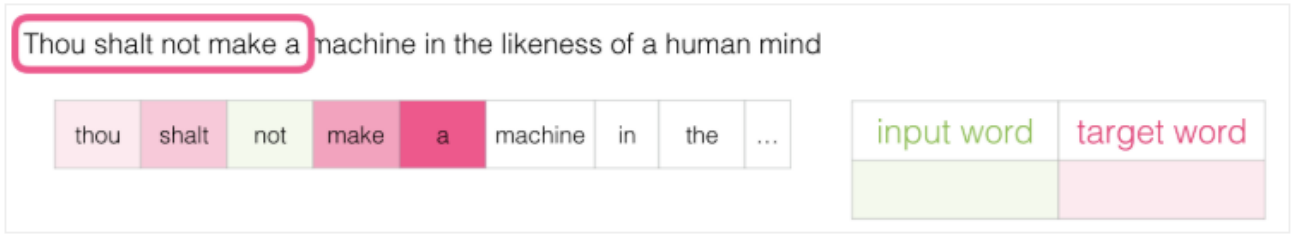
input 1	input 2	input 3	input 4	output
by	a	bus	in	red

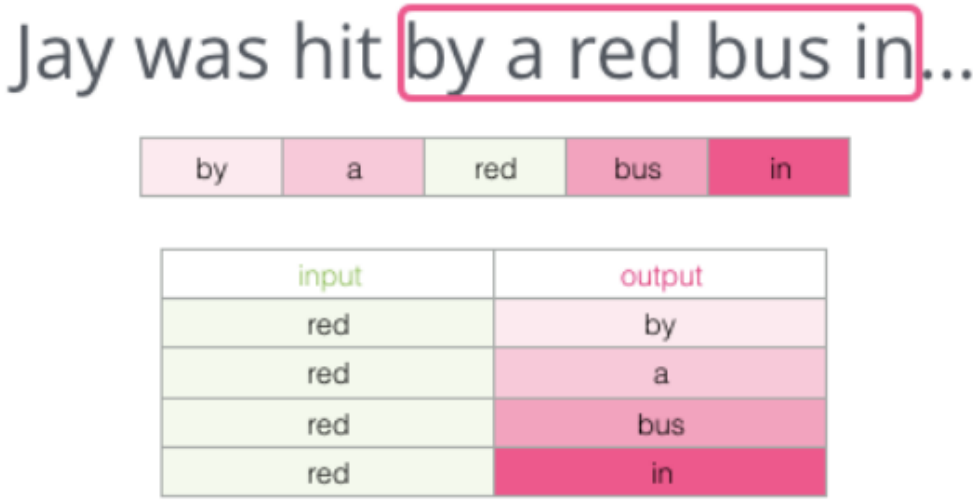


CBOW

Skipgram

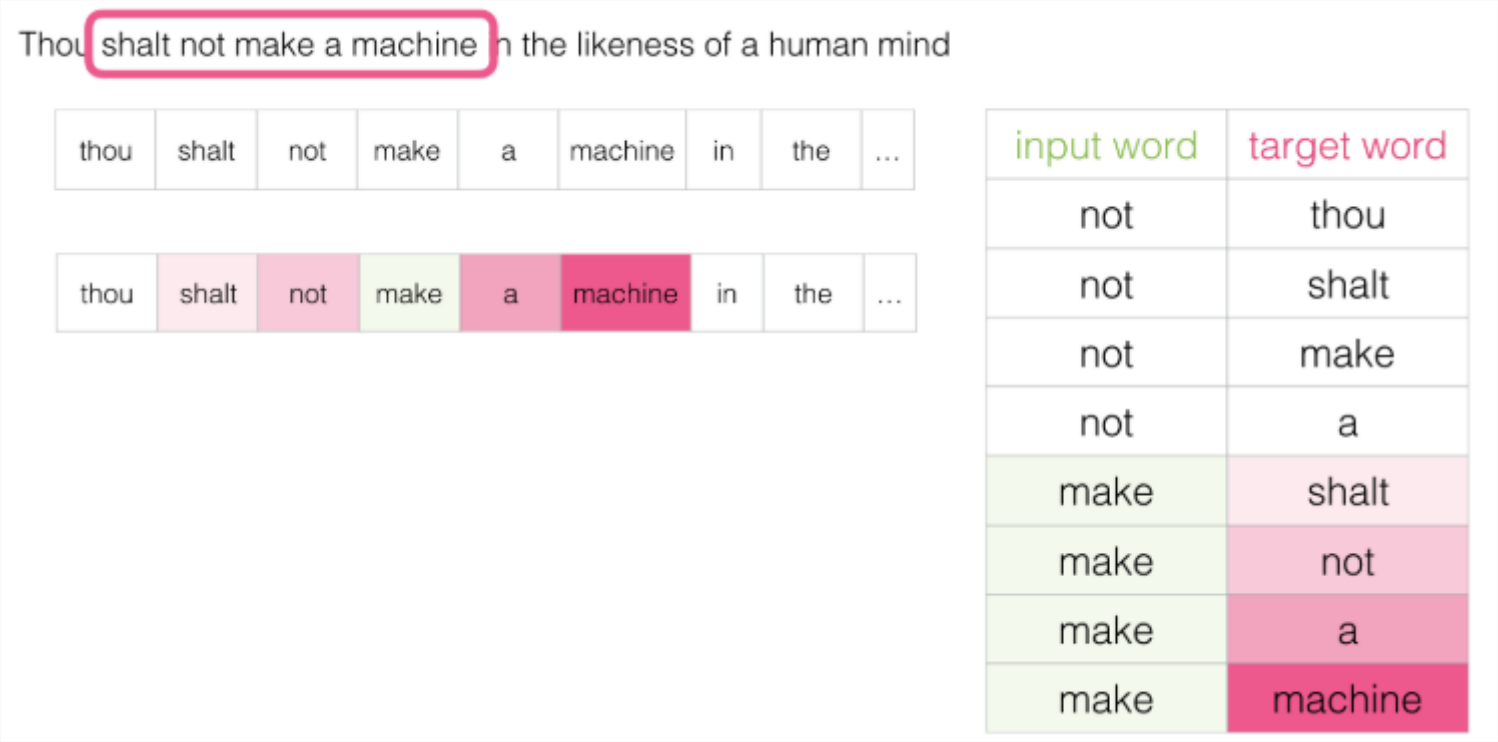
- This method is called the skipgram architecture.
- We can visualize the sliding window as doing the following:





Skipgram

- We then slide our window to the next position:

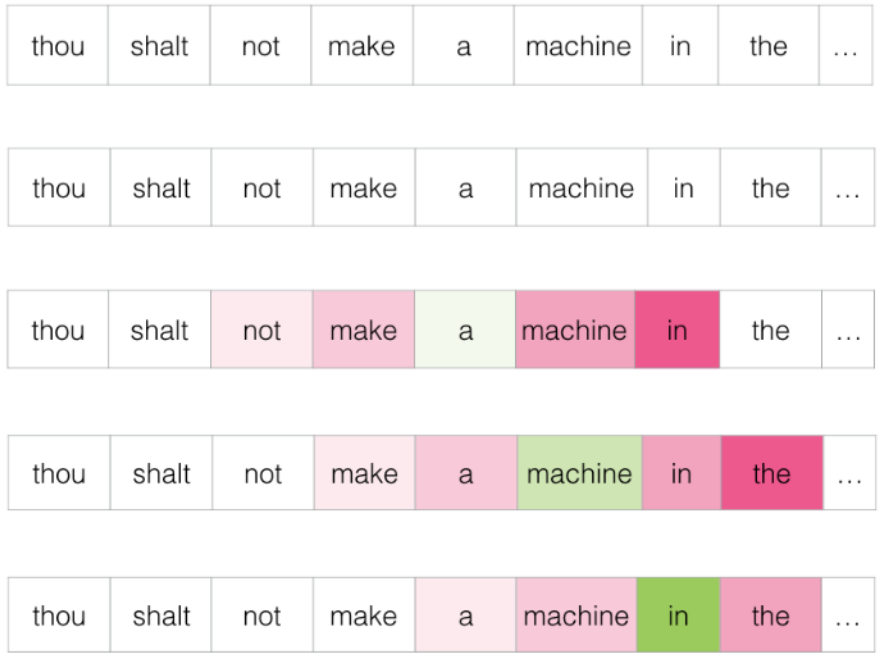


input word	target word
not	thou
not	shalt
not	make
not	a
make	shalt
make	not
make	a
make	machine

Skipgram

- A couple of positions later, we have a lot more examples:

Thou shalt not make a machine in the likeness of a human mind



input word	target word
not	thou
not	shalt
not	make
not	a
make	shalt
make	not
make	a
make	machine
a	not
a	make
a	machine
a	in
machine	make
machine	a
machine	in
machine	the
in	a
in	machine
in	the
in	likeness

Word2vec

- Now that we have our skipgram training dataset that we extracted from existing running text, let's glance at how we use it to train a basic neural language model that predicts the neighboring word.

input word	target word
not	thou
not	shalt
not	make
not	a
make	shalt
make	not
make	a
make	machine
a	not
a	make
a	machine
a	in
machine	make
machine	a
machine	in
machine	the
in	a
in	machine
in	the
in	likeness

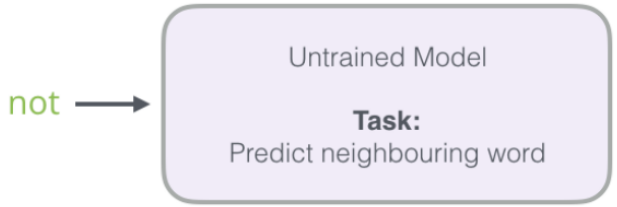
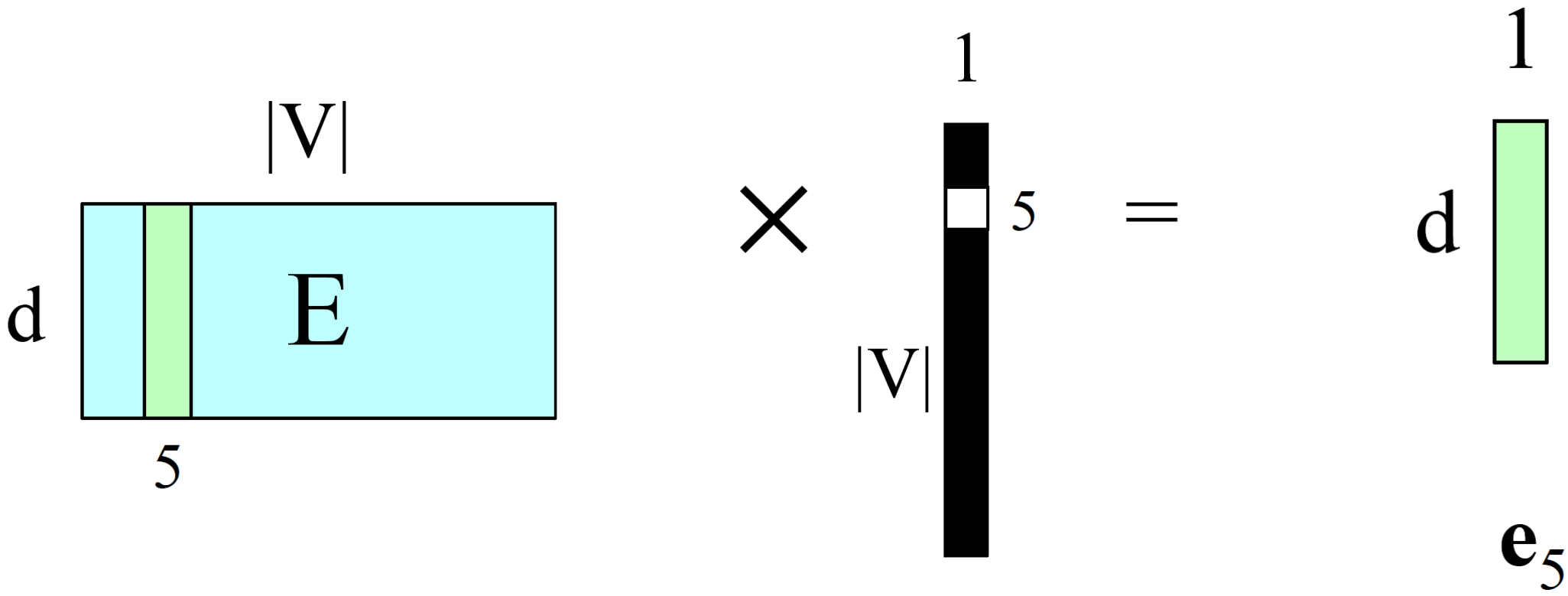


Fig. 6.1 shows a visualization of embeddings learned for sentiment analysis, showing the location of selected words projected down from 60-dimensional space into a two dimensional space. Notice the distinct regions containing positive words, negative words, and neutral function words.

Word2vec



Feedforward Neural Language Modeling

- We then multiply these one-hot vectors by the embedding matrix E.
- The embedding weight matrix E has a column for each word, each a column vector of d dimensions.
- Multiplying by a one-hot vector that has only one non-zero element $x_i = 1$ simply selects out the relevant column vector for word i, resulting in the embedding for word i, as shown in Fig. 7.16.

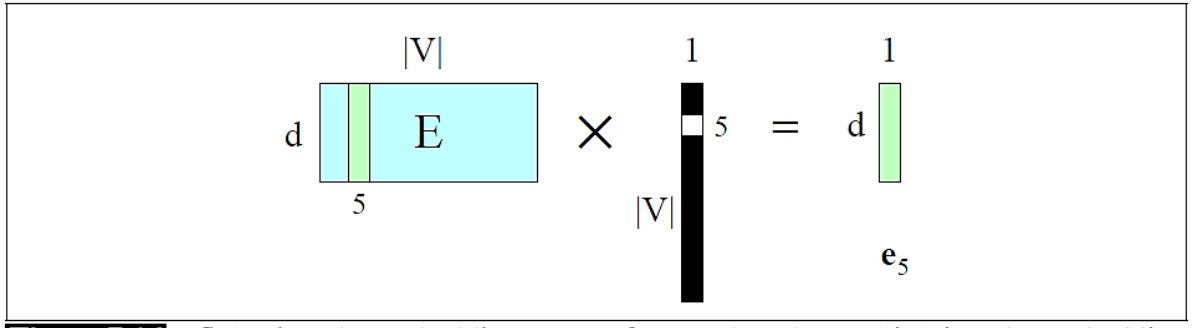
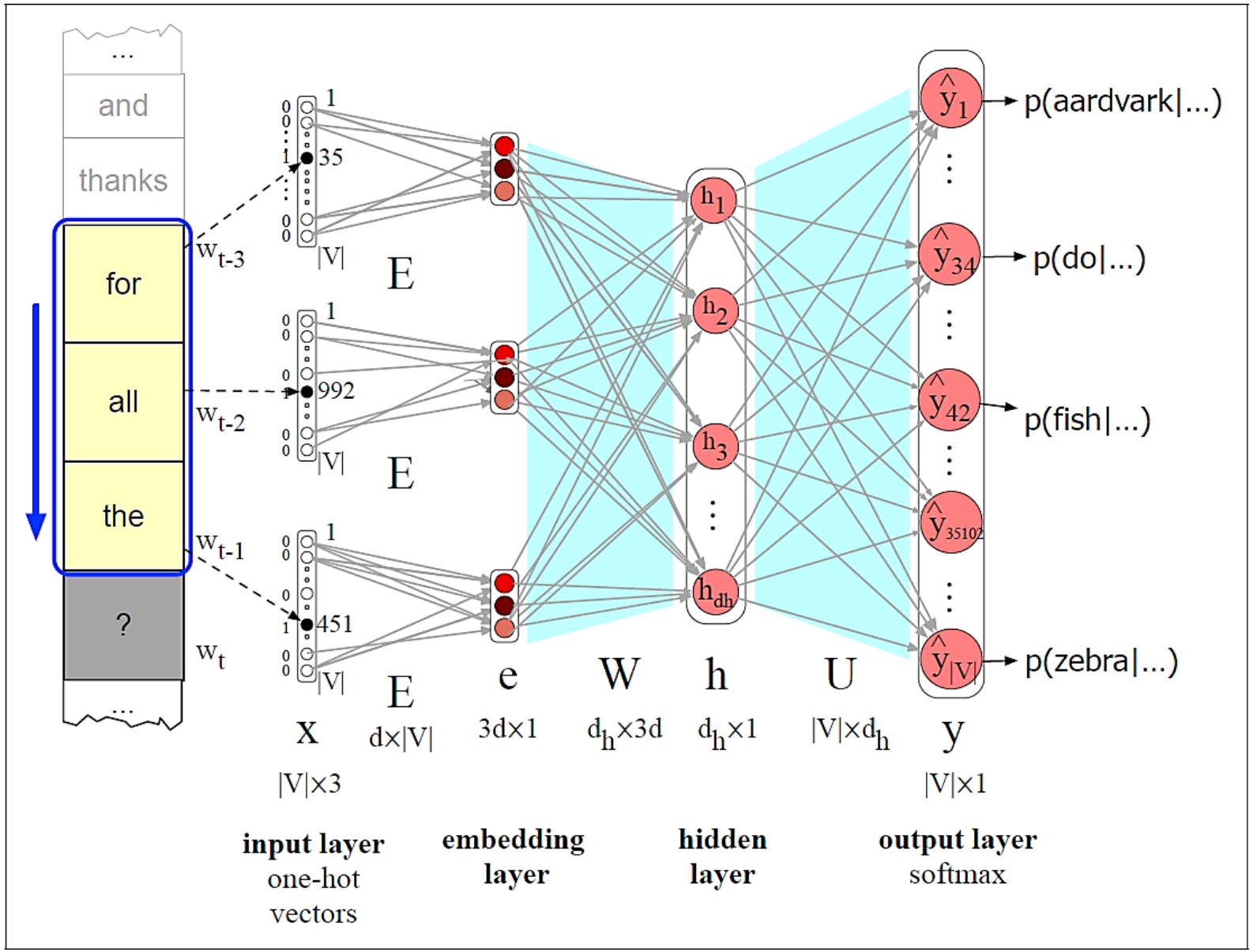


Figure 7.16 Selecting the embedding vector for word V_5 by multiplying the embedding matrix **E** with a one-hot vector with a 1 in index 5.



$$\mathbf{e} = [\mathbf{E}\mathbf{x}_{t-3}; \mathbf{E}\mathbf{x}_{t-2}; \mathbf{E}\mathbf{x}_{t-1}]$$
$$\mathbf{h} = \sigma(\mathbf{W}\mathbf{e} + \mathbf{b})$$
$$\mathbf{z} = \mathbf{U}\mathbf{h}$$
$$\hat{\mathbf{y}} = \text{softmax}(\mathbf{z})$$

Sources

- Speech and Language Processing, 3rd edition, by Jurafsky and Martin
  - https://web.stanford.edu/~jurafsky/slp3/6.pdf
  - https://web.stanford.edu/~jurafsky/slp3/7.pdf
  - https://jalammar.github.io/illustrated-word2vec/