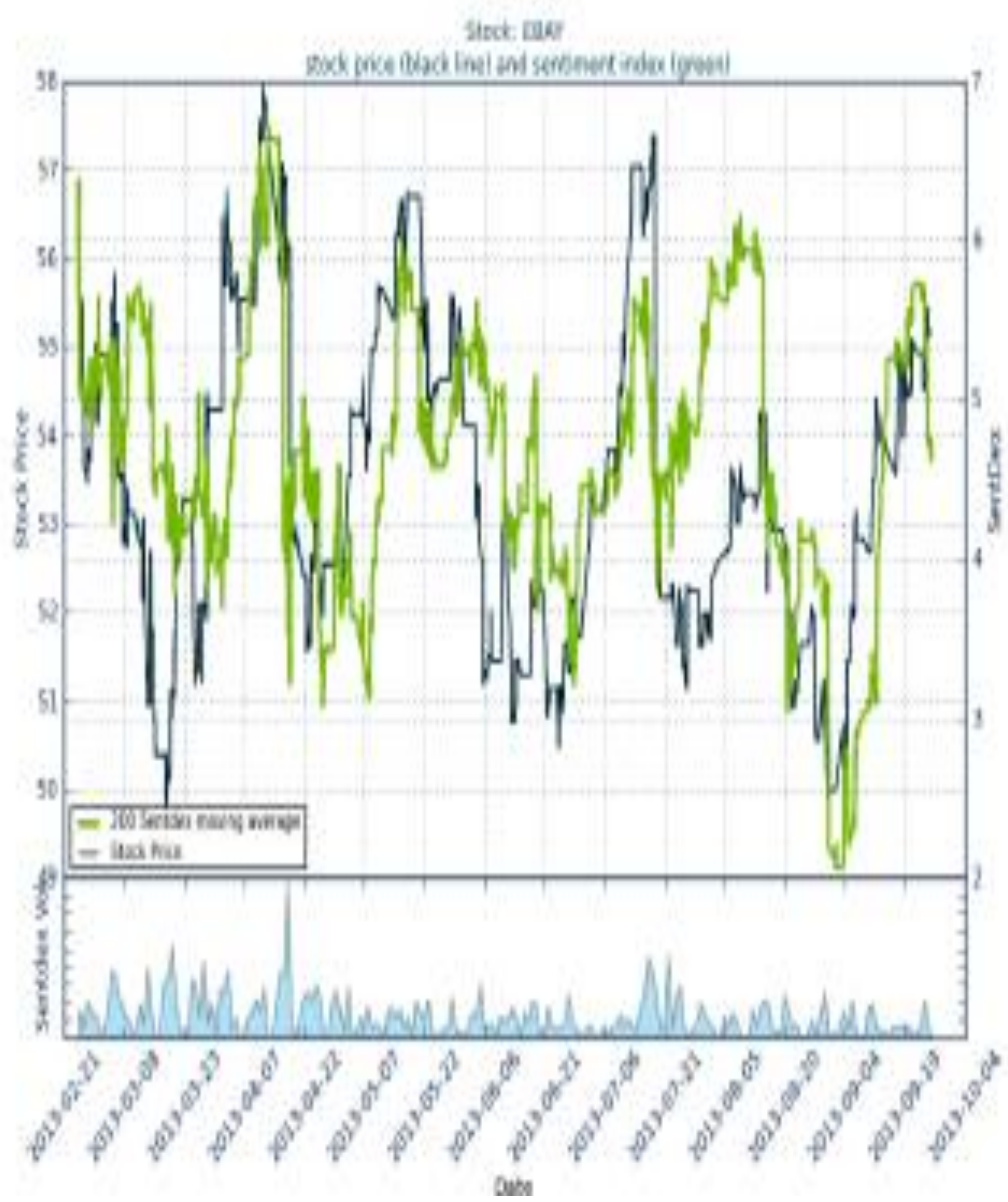
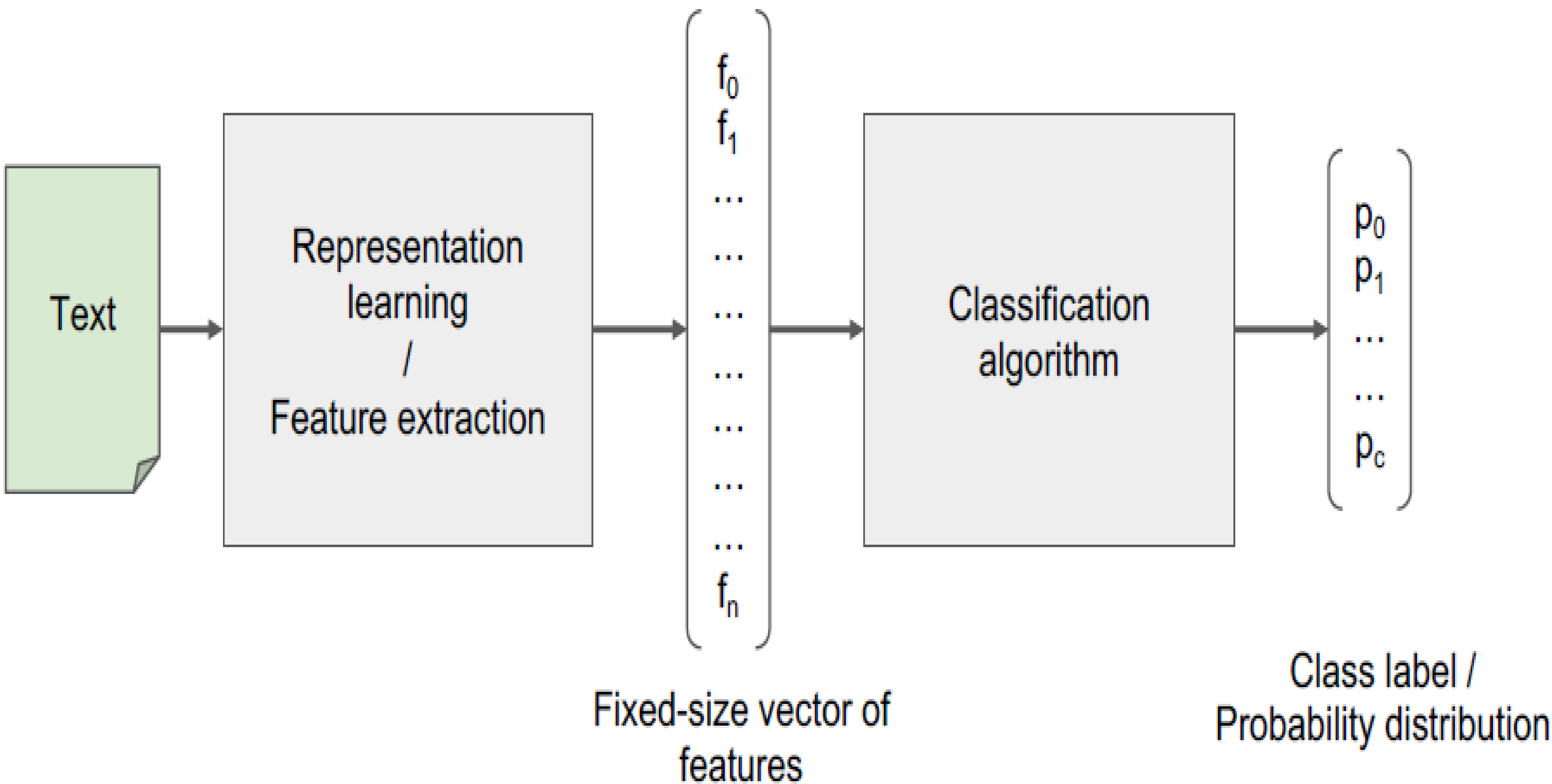
Lect 7 Text Classification

Why do we need to classify texts?

- As a self-sufficient task:
  - Spam filtering
  - Sentiment analysis



Text classification in general



Text label kinds

1. Discrete labels:
    a. Label is known:
        i. Binary classification: spam filtering/sentiment analysis
        ii. Multi-class classification: categorization of goods
        iii. Multi-label classification: #hashtag prediction
    b. Label is unknown:
        i. Text clasterization: user intent search
2. Continuous labels: predict a salary by CV, predict a price by a product description

Classifier – Naïve Bayes (1)

- Given document d and a fixed set of classes $C = \{c_1, c_2\}$, x is the words of d, compute the class of d :

$$C_{MAP} = \underset{c \in C}{\operatorname{argmax}}\, P(c|d)$$

$$= \underset{c \in C}{\operatorname{argmax}} \boxed{\frac{P(d|c)P(c)}{P(d)}}$$

Bayes rule

$$= \underset{c \in C}{\operatorname{argmax}}\, P(d|c)P(c)$$

$$= \underset{c \in C}{\operatorname{argmax}}\, P(x_1, x_2, \ldots, x_n|c)P(c)$$

$$= \underset{c \in C}{\operatorname{argmax}}\; P(c) \prod_{x \in X} P(x|c)$$

Classifier – Naïve Bayes (2)

$$\hat{P}(c) = \frac{N_c}{N}$$

$$\hat{P}(w \mid c) = \frac{count(w,c)+1}{count(c)+|V|}$$

	Doc	Words	Class
Training	1	Chinese Beijing Chinese	c
	2	Chinese Chinese Shanghai	c
	3	Chinese Macao	c
	4	Tokyo Japan Chinese	j
Test	5	Chinese Chinese Chinese Tokyo Japan	?

Priors:

$P(c) = \frac{3}{4}$

$P(j) = \frac{1}{4}$

Conditional Probabilities:

P(Chinese|*c*) = (5+1) / (8+6) = 6/14 = 3/7

P(Tokyo|*c*) = (0+1) / (8+6) = 1/14

P(Japan|*c*) = (0+1) / (8+6) = 1/14

P(Chinese|*j*) = (1+1) / (3+6) = 2/9

P(Tokyo|*j*) = (1+1) / (3+6) = 2/9

P(Japan|*j*) = (1+1) / (3+6) = 2/9

Choosing a class:

$P(c|d5) \propto 3/4 * (3/7)^3 * 1/14 * 1/14 \approx 0.0003$

$P(j|d5) \propto 1/4 * (2/9)^3 * 2/9 * 2/9 \approx 0.0001$

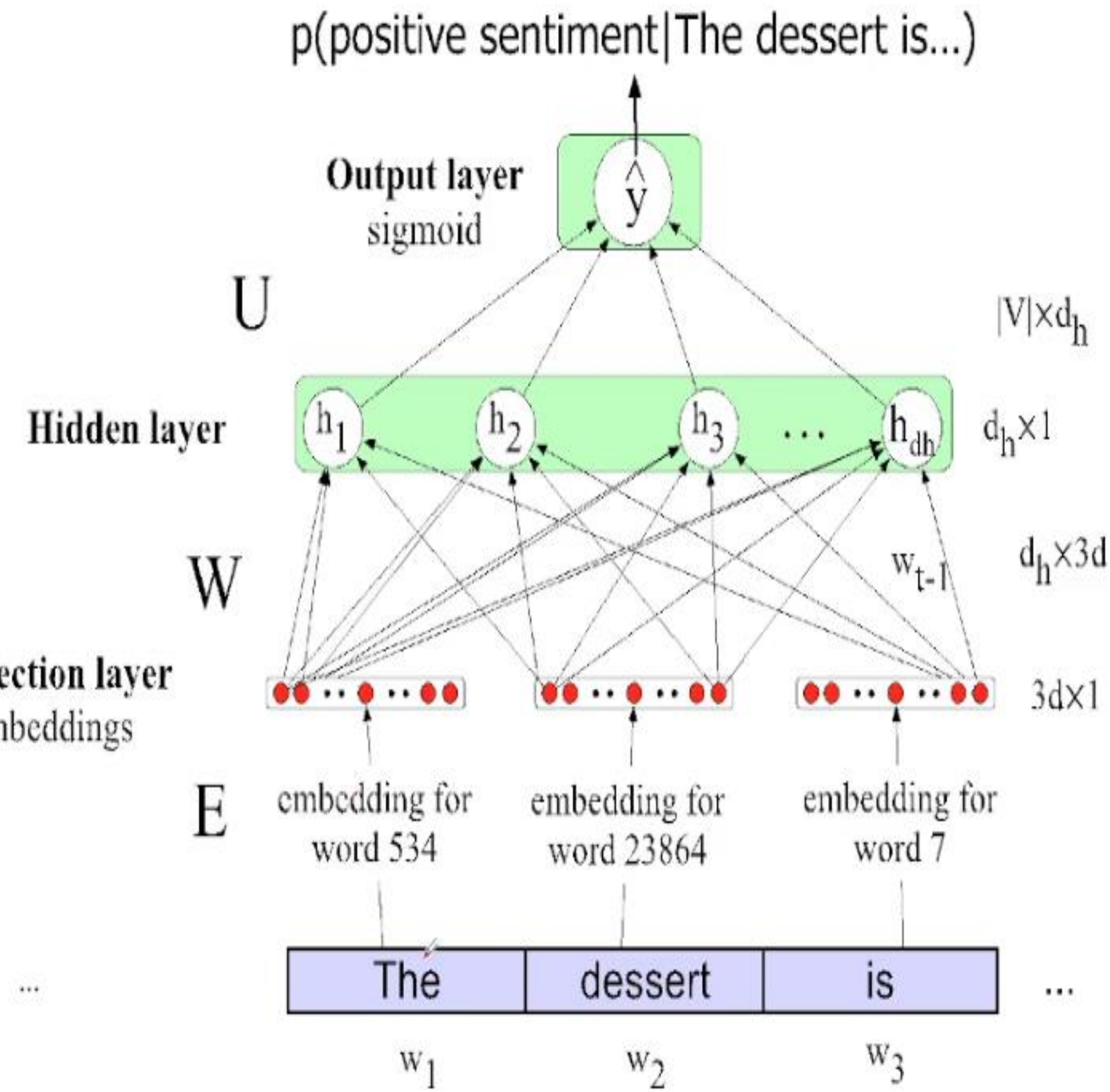
from Dan Jurafsky and Christopher Manning, Stanford University

Use cases for feedforward networks

Let's consider 2 (simplified) sample tasks:

1. Text classification
2. Language modeling

Neural Net Classification with embeddings as input features!



Simple feedforward Neural Language Models

Task: predict next word w_t

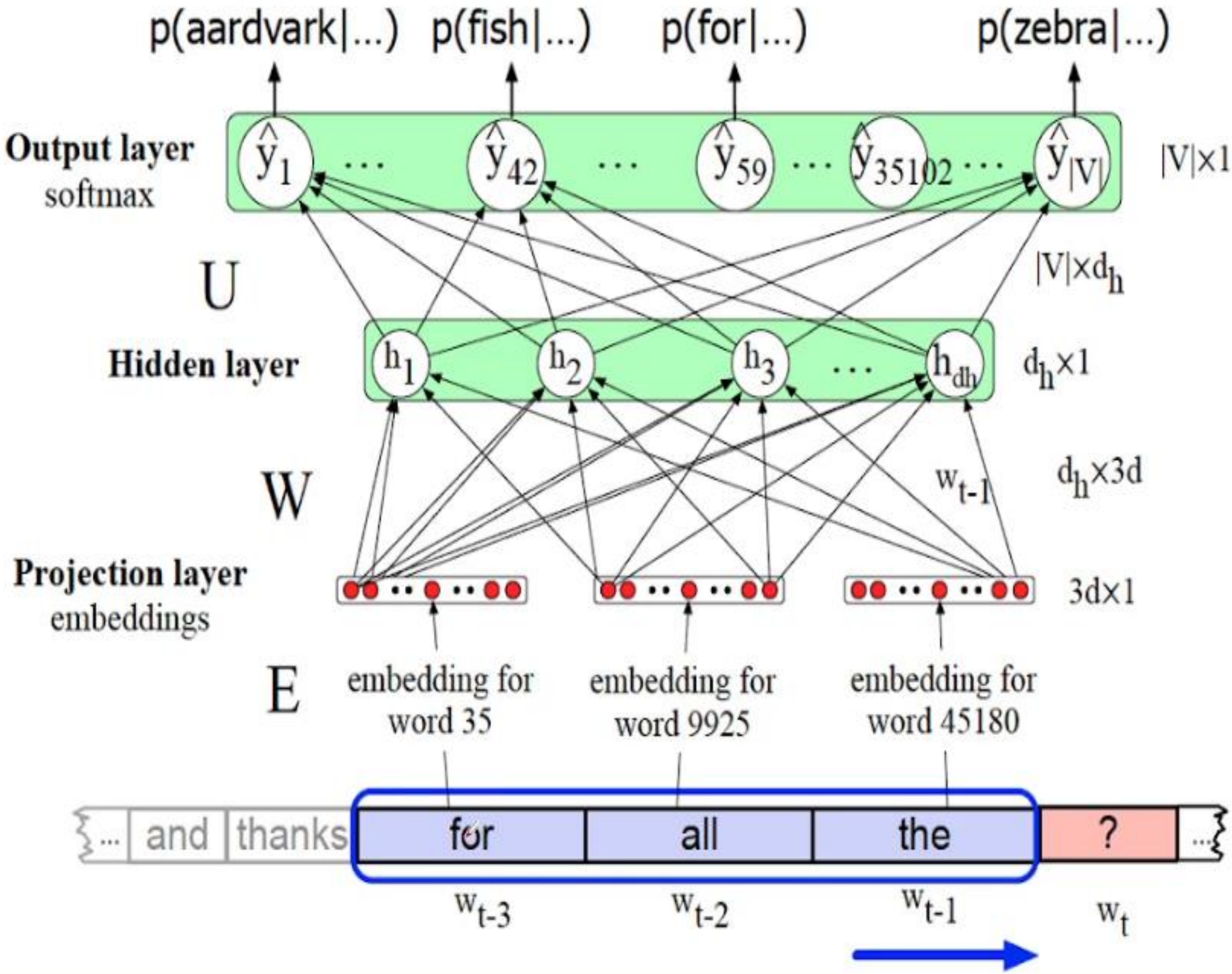
given prior words $w_{t-1}, w_{t-2}, w_{t-3}, \ldots$

Problem: Now we're dealing with sequences of arbitrary length.

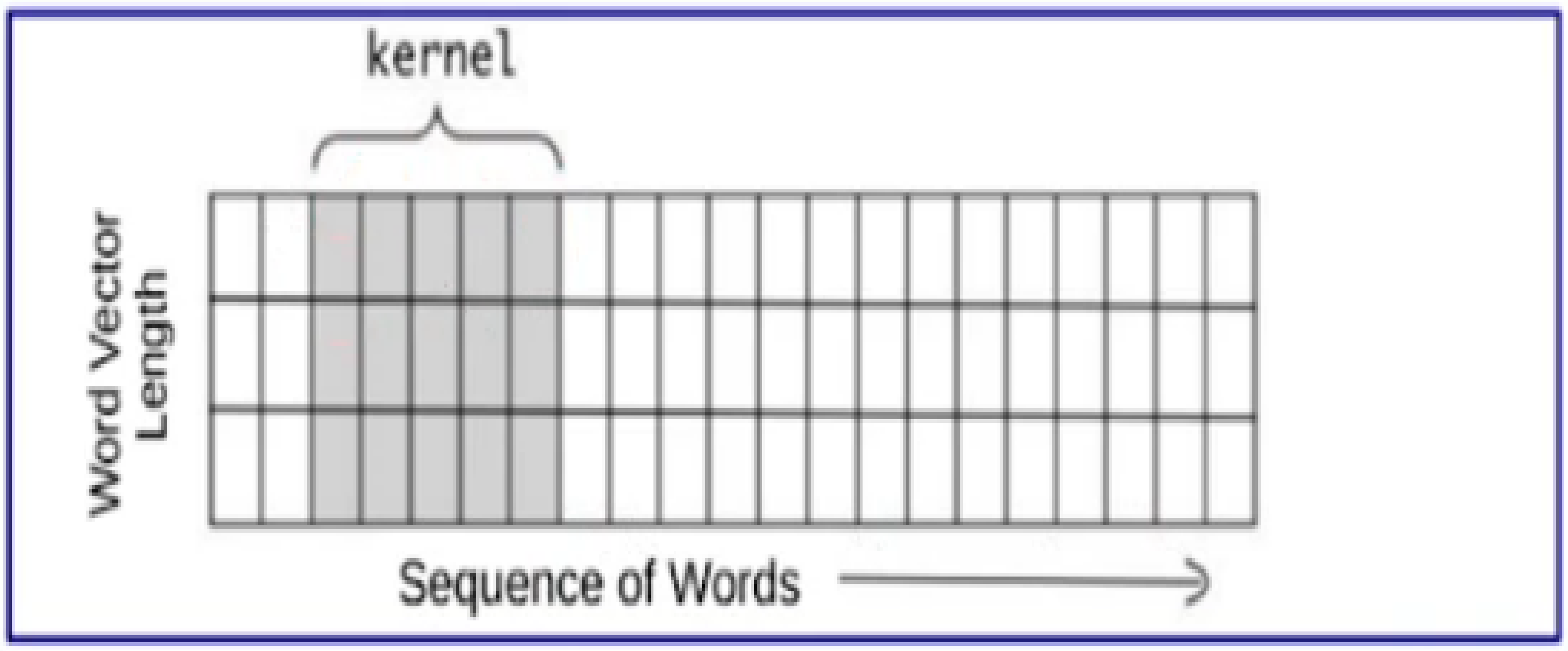
Solution: Sliding windows (of fixed length)

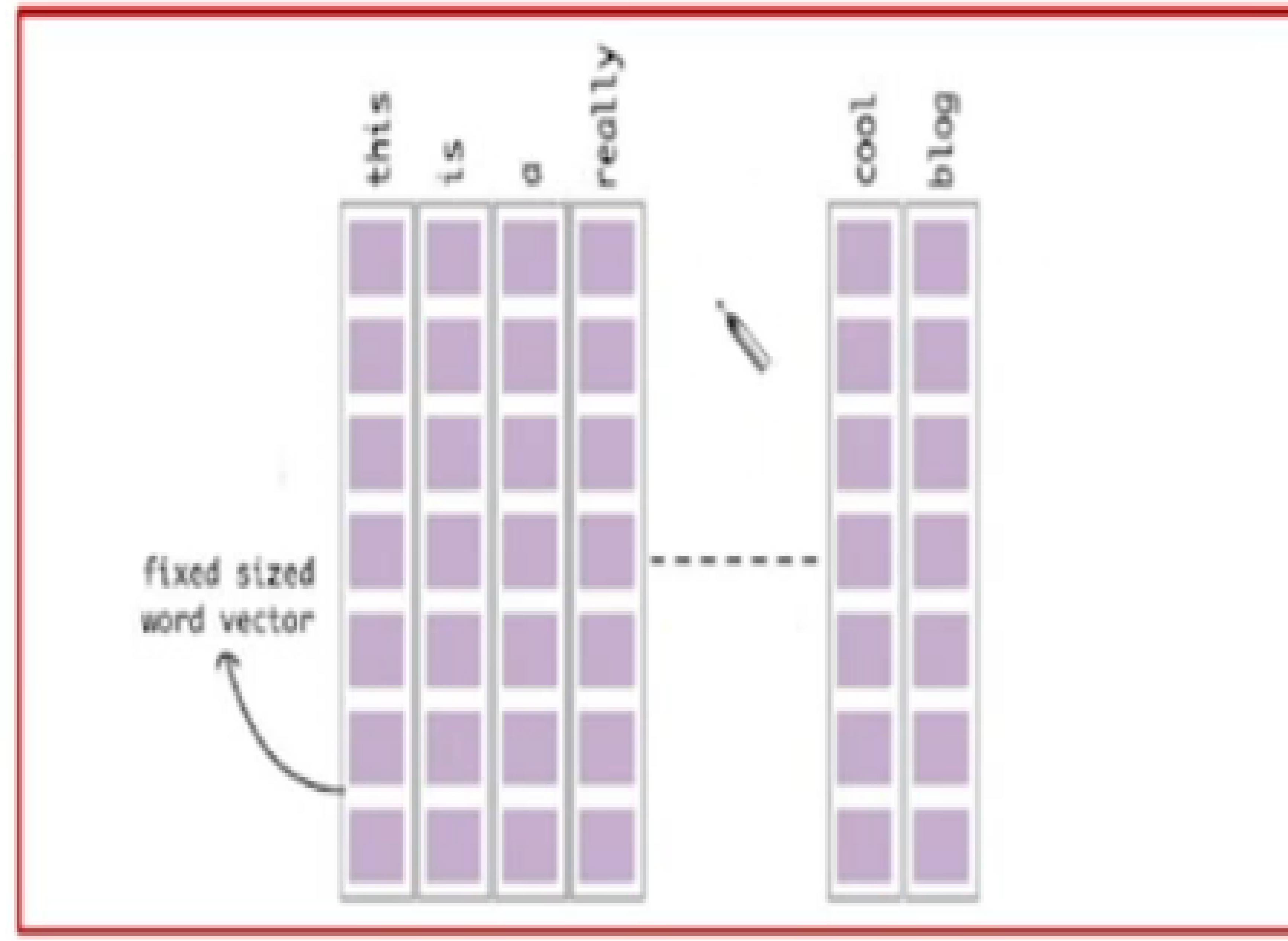
$$P(w_t|w_1^{t-1}) \approx P(w_t|w_{t-N+1}^{t-1})$$

Neural Language Model



1 Dimensional CNN | Conv1D (e.g. Text Data)

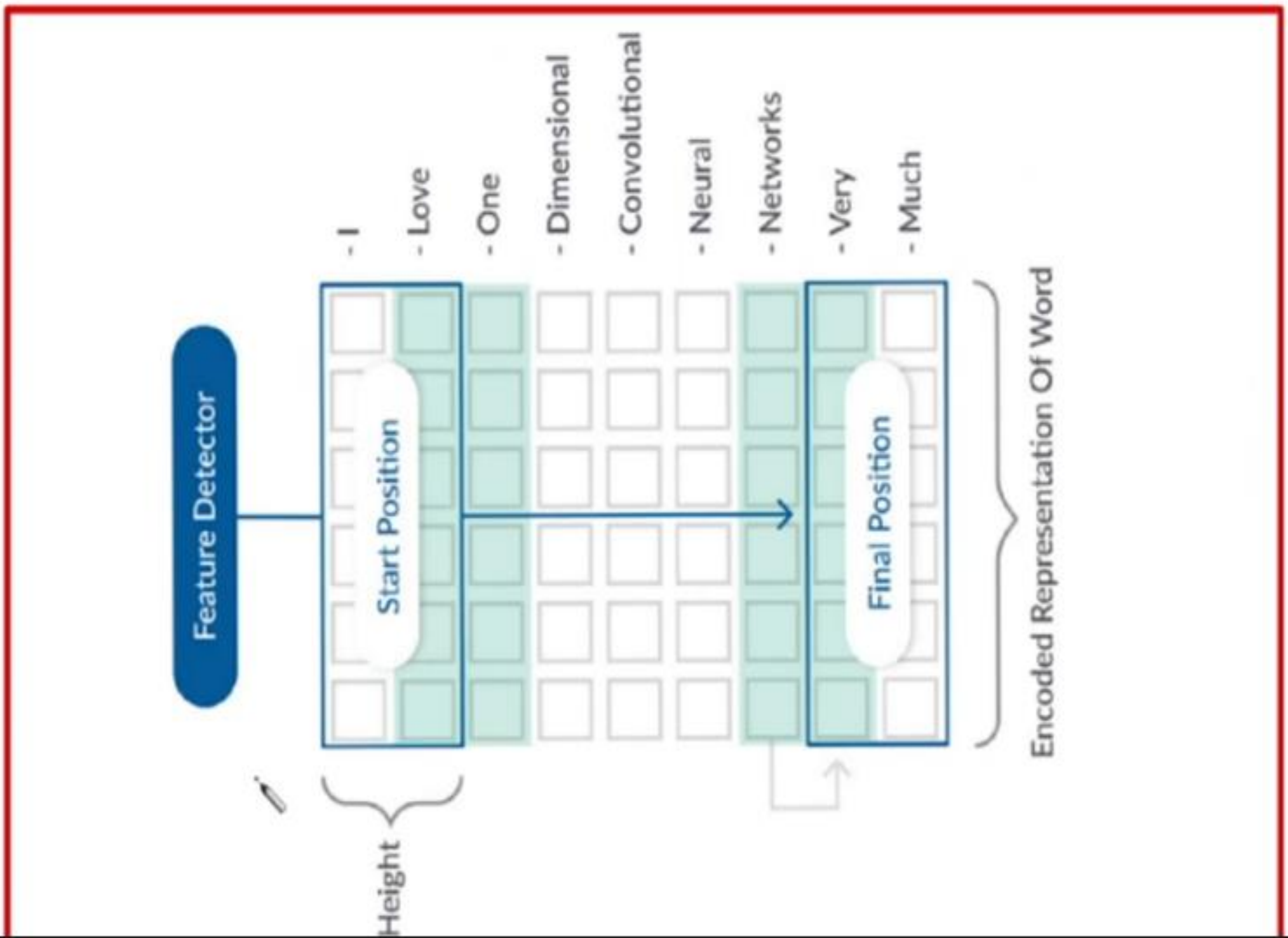




- 1D CNNs are also applied on:
  - Sensory data, (e.g. accelerometer data)
  - Audio Data
  - Text data

  Since we can also represent the Sensor Readings, Sound and Texts as a time series data.

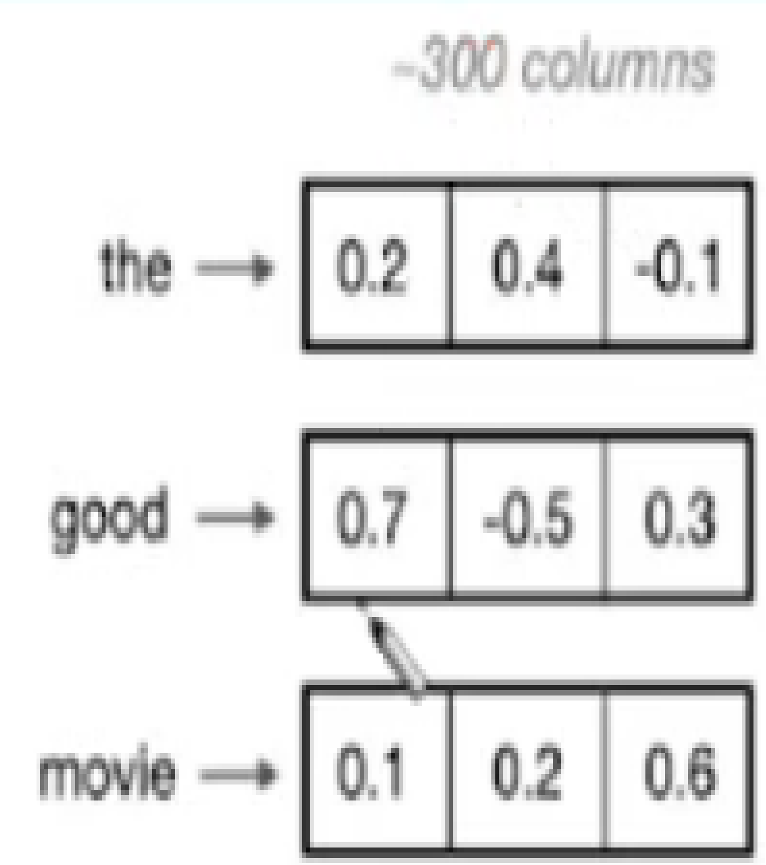
1 Dimensional CNN | Conv1D (e.g. Text Data)



CNNs for Text Classification

[3] Word Embedding

- Word embeddings are vectors of a specified length, typically on the order of 100s
- Each vector of 100 or so values, represents one word.
- The values in each column represent the features of a word, rather than any specific word.
- These embeddings are formed in an unsupervised manner by training a neural network (a Word2Vec model) on an input word and a few surrounding words in a sentence.



word2vec embeddings

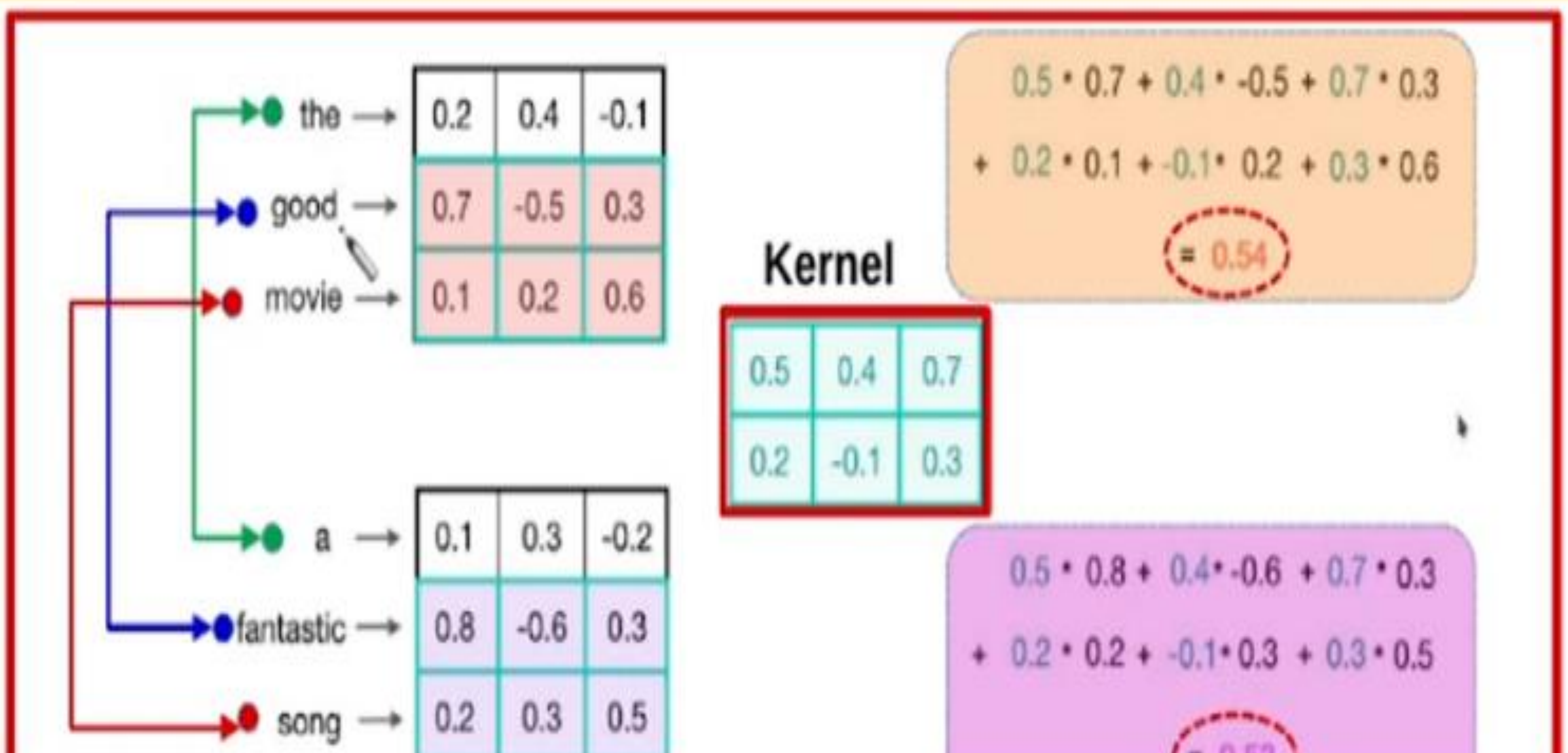
CNNs for Text Classification

[4] Convolutional Kernel

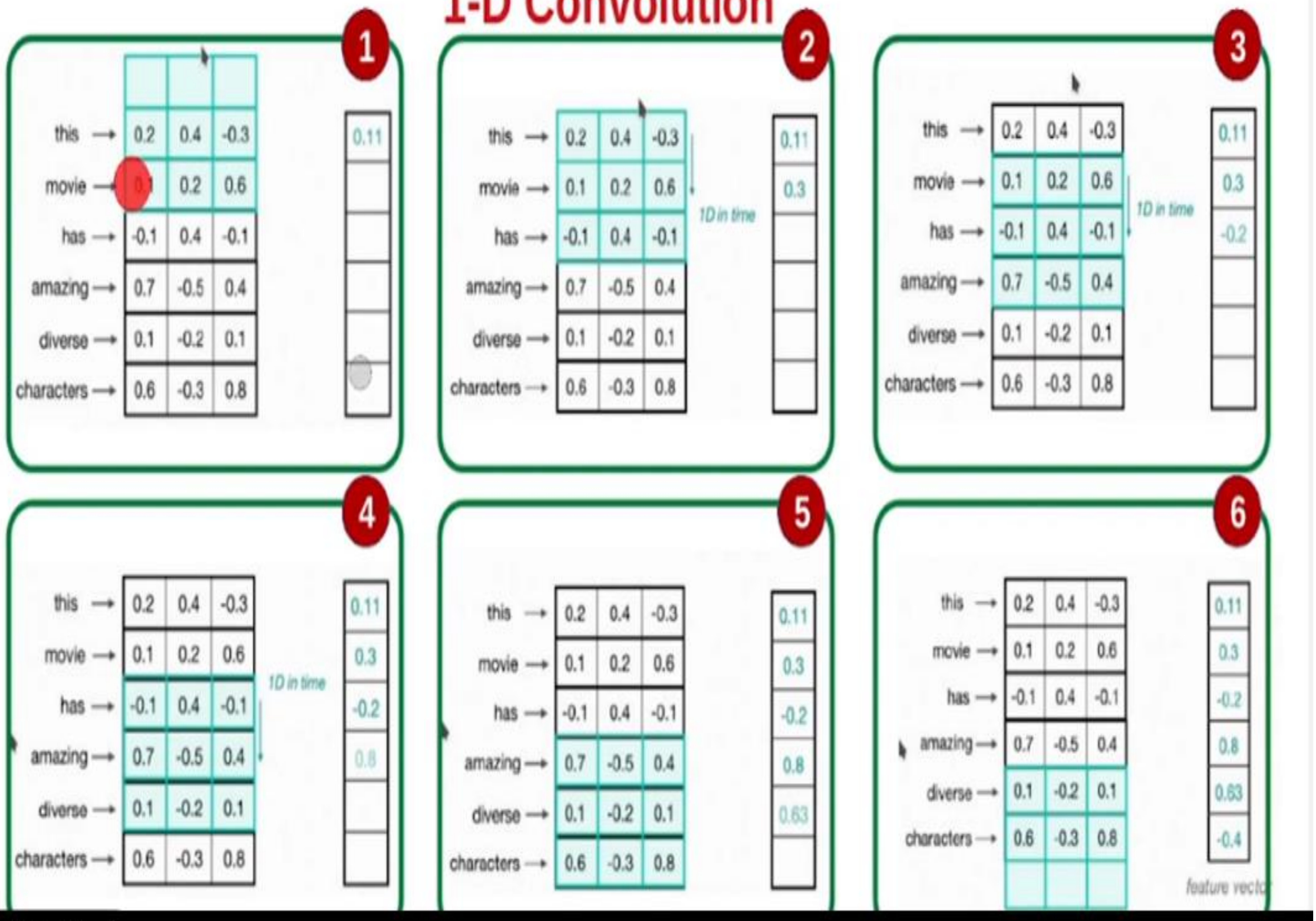
- <u>**Convolution in TEXT**</u>
- Convolutional kernel will still be a sliding window, only its job is to look at embeddings for multiple words, rather than small areas of pixels in an image.
- To look at sequences of word embeddings, we want a window to look at multiple word embeddings in a sequence.
- The kernels will no longer be square, instead, they will be a wide rectangle with dimensions like 3x300 or 5x300 (assuming an embedding length of 300).
- The **height of the kernel** will be the number of embeddings it will see at once, similar to representing an n-gram in a word model.
- The width of the kernel should span the length of an entire word embedding.

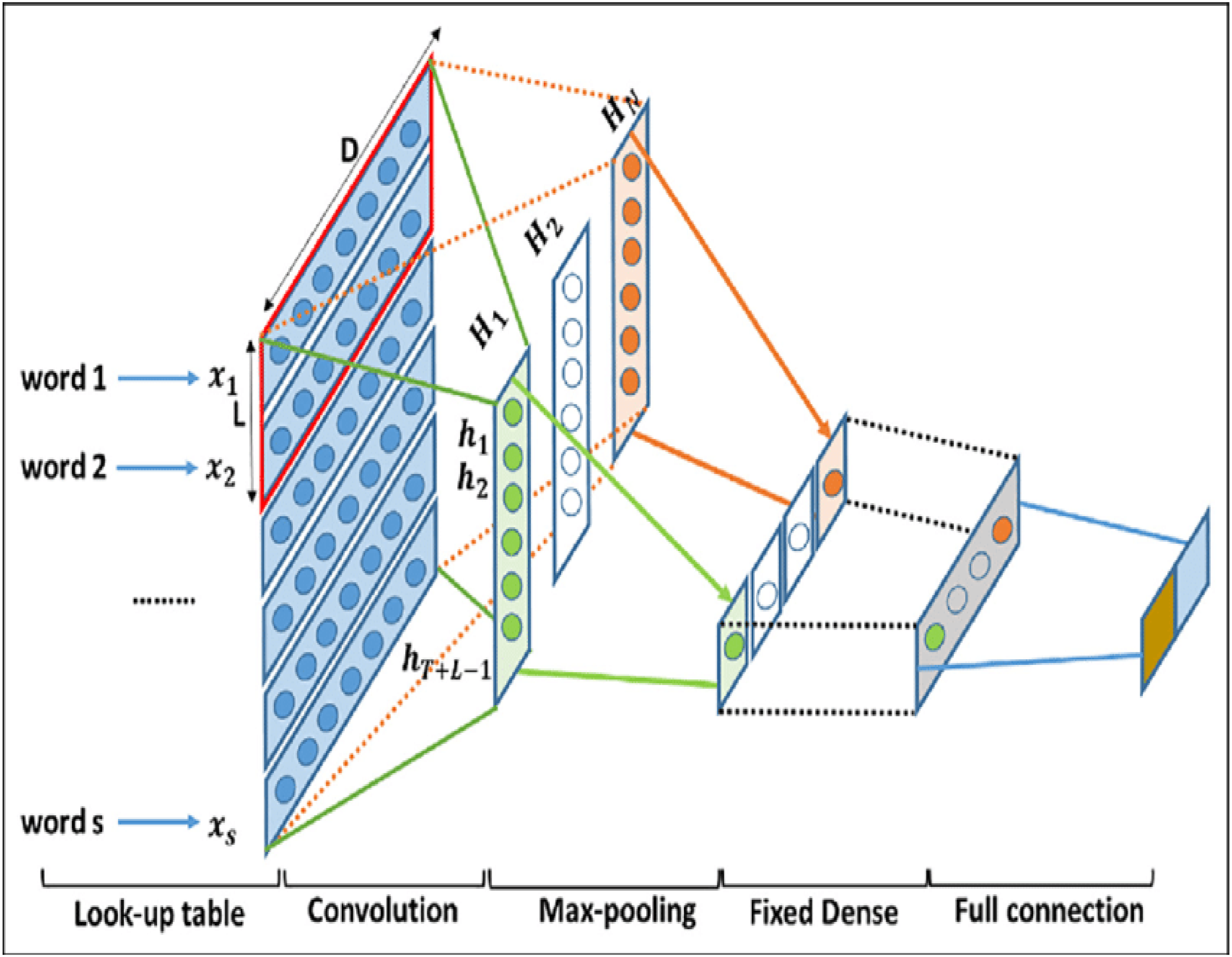
Recognizing General Patterns

- Recall that <u>similar words will have similar embeddings</u> and a convolution operation is just a linear operation on these vectors.
- when a <u>convolutional kernel</u> is applied to different sets of <u>similar words</u>, it will produce a <u>similar output value</u>!
- the convolutional output value for the input 2-grams **"good movie"** and **"fantastic song"** are about the same because the word embeddings for those pairs of words are also very similar.



CNNs for Text Classification
1-D Convolution





D
word 1
x_1
L
word 2
x_2
word s
x_s
H_1
H_2
H_N
h_1
h_2
h_{T+L-1}
Look-up table
Convolution
Max-pooling
Fixed Dense
Full connection

Example

We build a CNN model that converts words into vectors, selects important features using pooling and combines them in fully connected layers. Dropout prevents overfitting and the final layer outputs a probability for classification.

models.Sequential(): Creates a linear stack of layers where each layer passes output to the next.

layers.Embedding(input_dim=10000, output_dim=100, input_length=500): Converts word indices into 100-dimensional vectors, helping the model learn word meanings. Handles a vocabulary of 10,000 words and sequences of 500 words.

layers.Conv1D(filters=128, kernel_size=5, activation='relu'): Applies 128 sliding filters that look at 5 words at a time to detect patterns.

layers.GlobalMaxPooling1D(): Reduces data by taking the maximum value from each filter's output, keeping only the most important features.

layers.Dense(64, activation='relu'): A fully connected layer with 64 neurons that learns complex patterns.

layers.Dropout(0.5): Randomly disables 50% of neurons during training to prevent overfitting.

layers.Dense(1, activation='sigmoid'): Final output layer that predicts a probability (0–1) for binary classification.

Importing Libraries .1

We will import the required libraries such as tensorflow, numpy required for building CNN model ،
creating layers, handling numerical operations and padding text sequences.

•**tensorflow.keras:** Used for importing layers like Embedding, Conv1D and Sequential for model building.

•**imdb**: Loads the IMDB dataset.

•**pad_sequences**: Pads text sequences to a fixed length.

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, Conv1D, GlobalMaxPooling1D, Dense, Dropout
from tensorflow.keras.datasets import imdb
from tensorflow.keras.preprocessing import sequence
```

Loading Data .2

We will load and preprocess the IMDB dataset.

•**imdb.load_data(num_words=10000)**: Loads the IMDB dataset, keeping only the 10,000 most frequent words.

•**pad_sequences(sequences, maxlen=500)**: Pads or cuts reviews so each is exactly 500 words long.

```
vocab_size = 10000
max_length = 500
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=vocab_size)
x_train = sequence.pad_sequences(x_train, maxlen=max_length)
x_test = sequence.pad_sequences(x_test, maxlen=max_length)
```

Building CNN model .3

We build a CNN model that converts words into vectors, selects important features using pooling and combines them in fully connected layers. Dropout prevents overfitting and the final layer outputs a probability for classification.

•**models.Sequential()**: Creates a linear stack of layers where each layer passes output to the next.

•**layers.Embedding(input_dim=10000, output_dim=100, input_length=500)**: Converts word indices into 100-dimensional vectors, helping the model learn word meanings. Handles a vocabulary of 10,000 words and sequences of 500 words.

•**layers.Conv1D(filters=128, kernel_size=5, activation='relu')**: Applies 128 sliding filters that look at 5 words at a time to detect patterns.

•**layers.GlobalMaxPooling1D()**: Reduces data by taking the maximum value from each filter’s output, keeping only the most important features.

•**layers.Dense(64, activation='relu')**: A fully connected layer with 64 neurons that learns complex patterns.

•**layers.Dropout(0.5)**: Randomly disables 50% of neurons during training to prevent overfitting.

•**layers.Dense(1, activation='sigmoid')**: Final output layer that predicts a probability (0–1) for binary classification.

```
model = Sequential([
    Embedding(vocab_size, 100, input_length=max_length),
    Conv1D(filters=128, kernel_size=5, activation='relu'),
    GlobalMaxPooling1D(),
    Dense(64, activation='relu'),
    Dropout(0.5),
    Dense(1, activation='sigmoid')
])
```

.4Compiling and Training the Model

We will compile the model and train it using the IMDB dataset.

Here we will use Adam optimizer with binary cross-entropy as loss function.

- **model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy']):** Defines the optimizer (Adam), loss function (binary cross-entropy) and accuracy metric for evaluating performance.
- **model.fit(x_train, y_train, epochs=5, batch_size=128, validation_split=0.2):** Trains the model for 5 epochs using batches of 128 samples, with 20% of the training data reserved for validation.

```
model.compile(optimizer='adam',
loss='binary_crossentropy',
metrics=['accuracy'])
model.fit(x_train, y_train, batch_size=32,
epochs=5, validation_split=0.2)
```

5. Evaluating the Model

We will evaluate the trained model on the test dataset.

model.evaluate(x_test, y_test): Evaluates model performance by returning loss and accuracy.

print(f"Test Accuracy: {test_accuracy:.4f}"): Prints the accuracy percentage on the test data.

```
test_loss, test_accuracy = model.evaluate(x_test, y_test)

print(f"Test Accuracy: {test_accuracy:.4f}")
```

```
Epoch 1/5
625/625 ━━━━━━━━━━━━━━━━━━━━ 111s 173ms/step - accuracy: 0.6519 - loss: 0.5842 - val_accuracy: 0.8842 - val_loss: 0.2808
Epoch 2/5
625/625 ━━━━━━━━━━━━━━━━━━━━ 140s 171ms/step - accuracy: 0.9219 - loss: 0.2098 - val_accuracy: 0.9020 - val_loss: 0.2517
Epoch 3/5
625/625 ━━━━━━━━━━━━━━━━━━━━ 143s 172ms/step - accuracy: 0.9773 - loss: 0.0777 - val_accuracy: 0.9006 - val_loss: 0.2752
Epoch 4/5
625/625 ━━━━━━━━━━━━━━━━━━━━ 142s 171ms/step - accuracy: 0.9945 - loss: 0.0248 - val_accuracy: 0.8924 - val_loss: 0.3894
Epoch 5/5
625/625 ━━━━━━━━━━━━━━━━━━━━ 139s 167ms/step - accuracy: 0.9986 - loss: 0.0081 - val_accuracy: 0.8988 - val_loss: 0.4441
782/782 ━━━━━━━━━━━━━━━━━━━━ 35s 45ms/step - accuracy: 0.8860 - loss: 0.4716
Test Accuracy: 0.8884
```