KiSTi 한국과학기술정보연구원 Korea Institute of Science and Technology Information

2022

# Advanced Topic in Research Data-centric Deep Learning

## Lec 12: RNN 4 NLP

2022.3.28.

이 홍 석 (hsyi@kisti.re.kr)

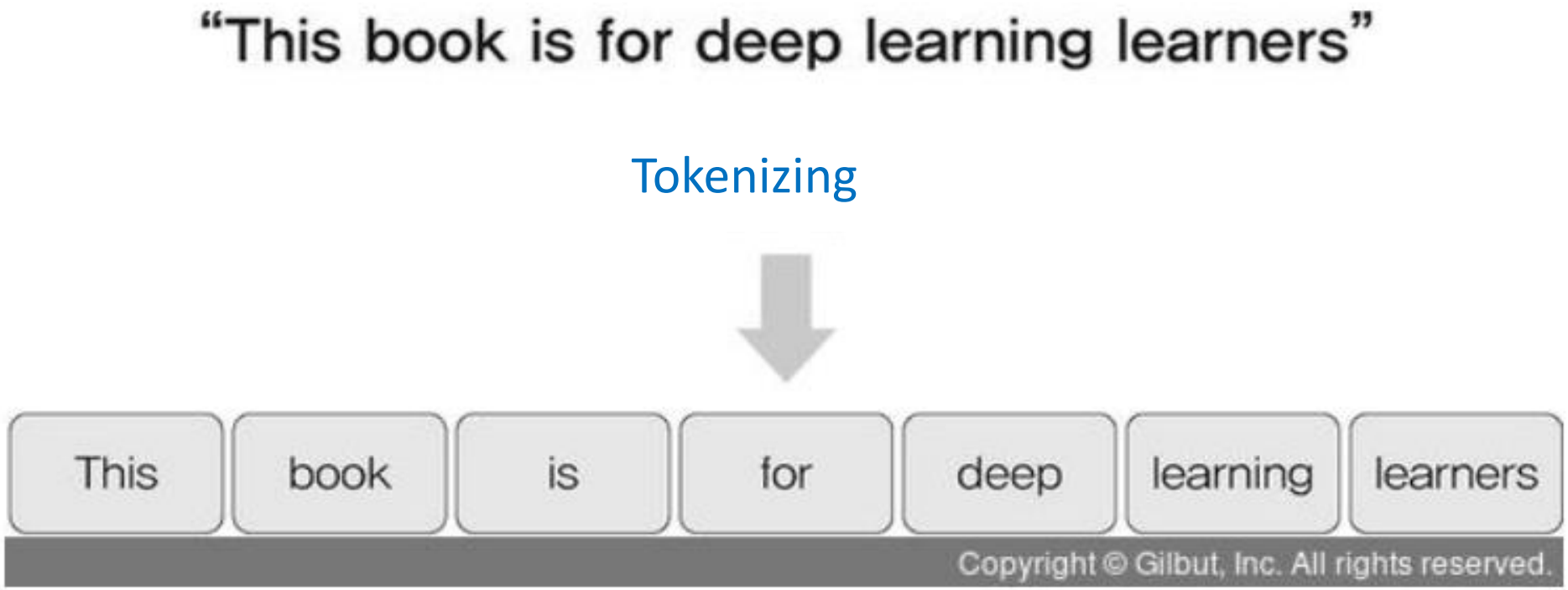
# Text Tockenizer

# Understanding the principles

- Token : Language elements that we can't share anymore
- Tokenizer
  - ✓work to input text data into the neural network.
  - ✓The preprocessing process that converts it into an appropriate form through encoding
- One-hot encoding
  - ✓In the case of text data, an embedding layer is basically used.

# Tokenizing words

- Word tokenization divides sentences based on spacing as follows.



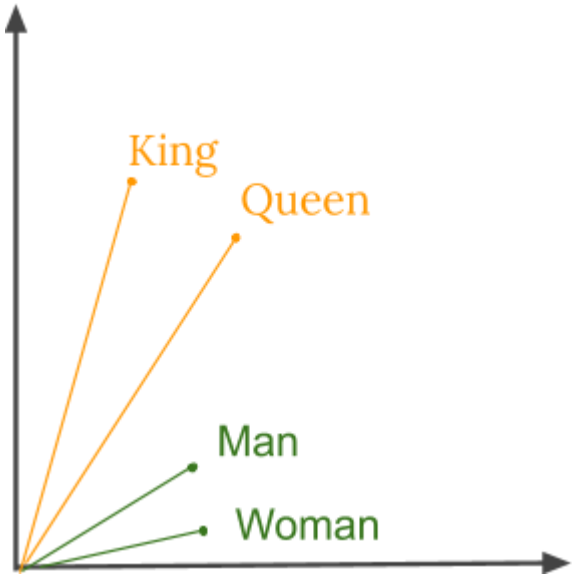
# One-Hot Encoding

| 1 | 0 | 0 | 0 | ... | 0 | 0 |
|---|---|---|---|---|---|---|
| Index: | 0 | 1 | 2 | 3 | ... | 99998 | 99999 |

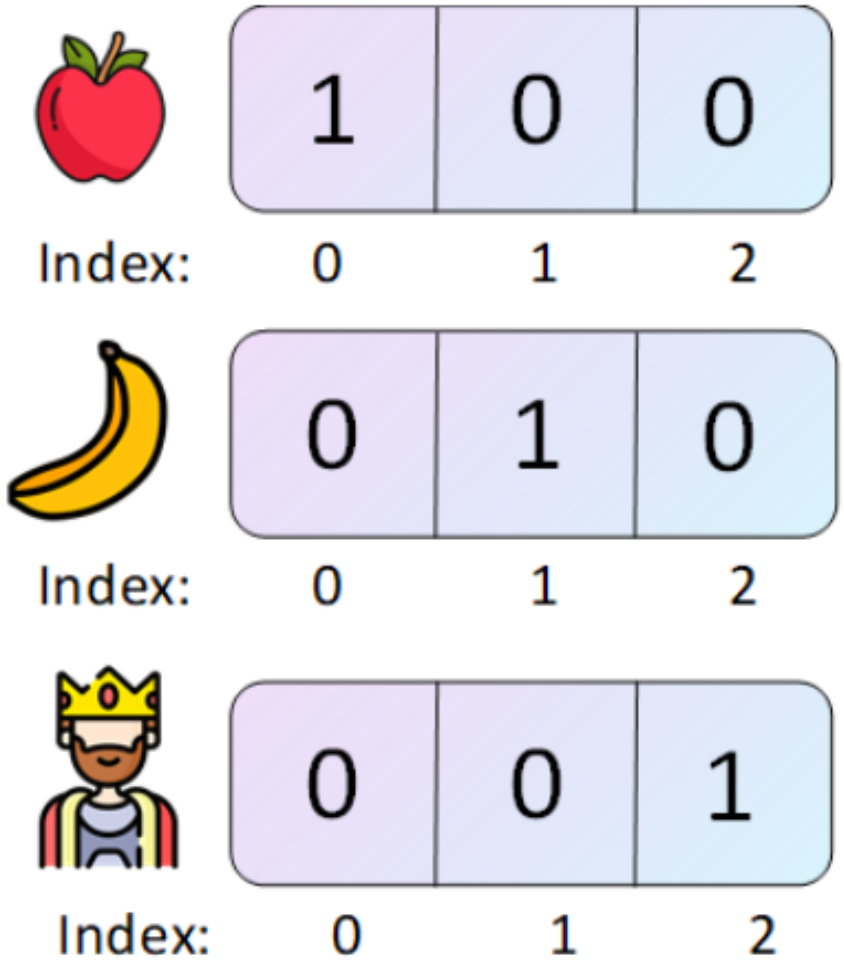
| 0 | 1 | 0 | 0 | ... | 0 | 0 |
|---|---|---|---|---|---|---|
| Index: | 0 | 1 | 2 | 3 | ... | 99998 | 99999 |

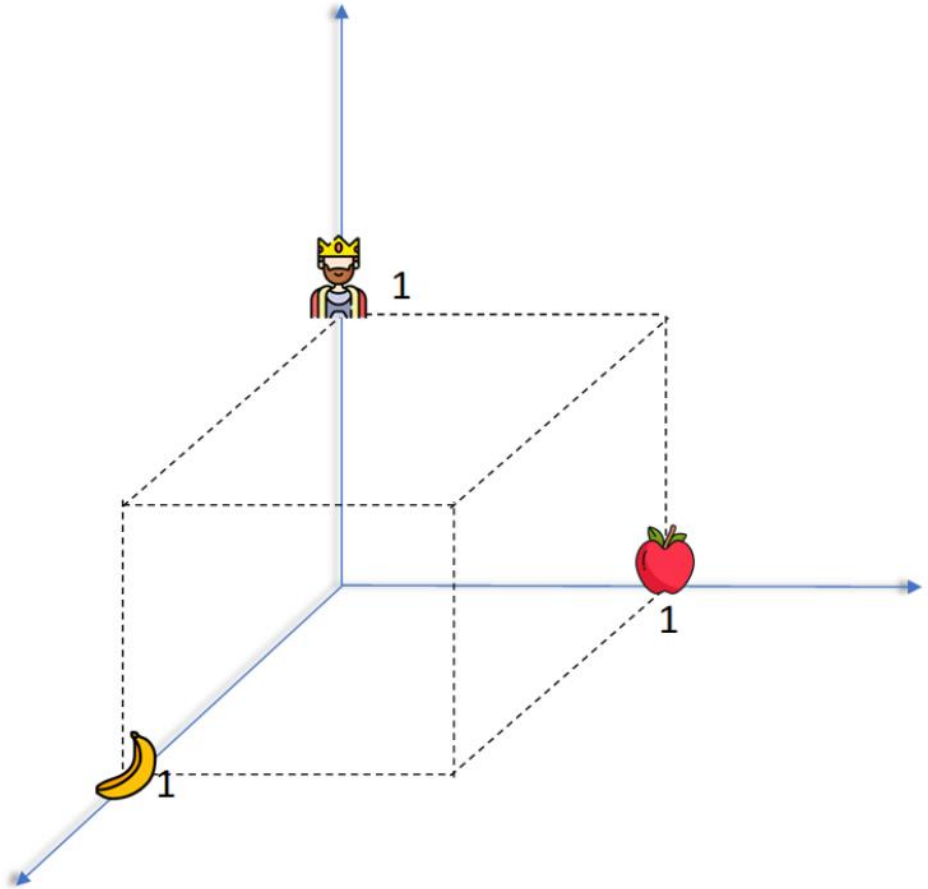
# What is Word Embedding?

- **Word embeddings are basically a form of word representation that bridges the human understanding of language to that of a machine.**
  - ✓ They have learned representations of text in an n-dimensional space where words that have the same meaning have a similar representation.
  - ✓ Meaning that two similar words are represented by almost similar vectors that are very closely placed in a vector space.

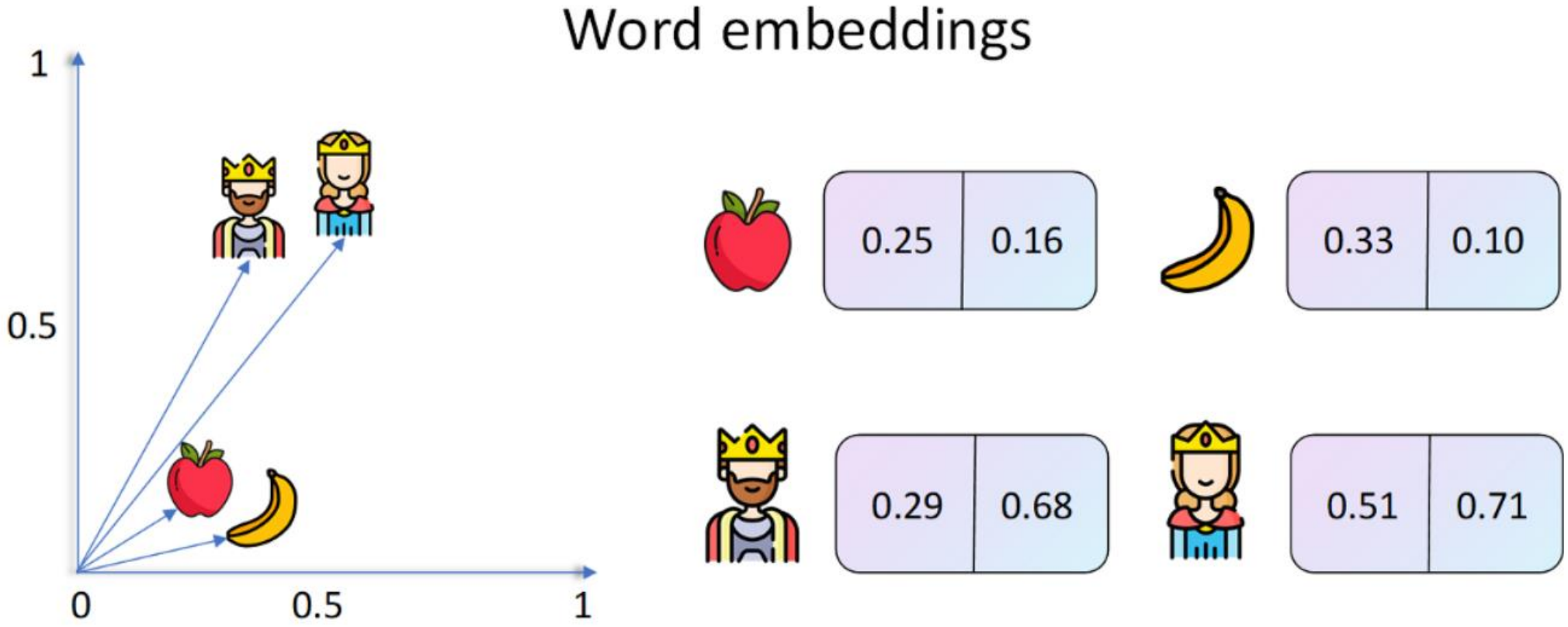


# Word Embedding



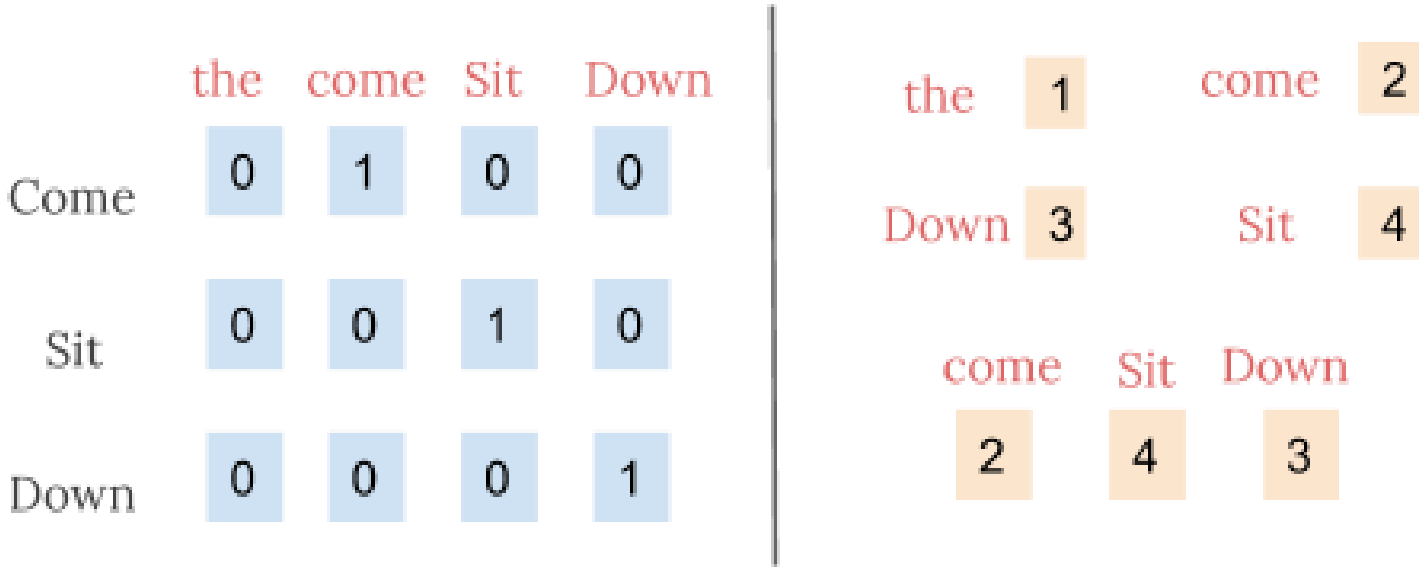



❖ 2 dimensional word embedding representation of our example words



# Why Word Embeddings are used?

- we can use two more techniques
  - ✓one-hot encoding
  - ✓we can use unique numbers to represent words in a vocabulary.
- we assume we have a small vocabulary containing just four words, using the two techniques we represent the sentence ‘Come sit down’.

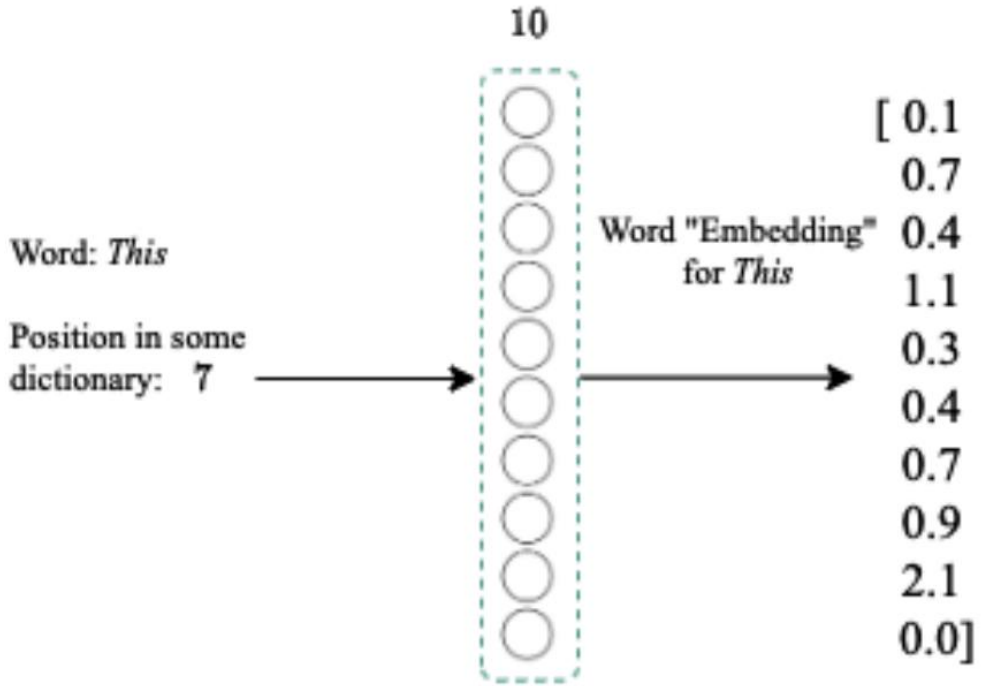


"This is a small vector"

# This

index : 7,
vector size : 10

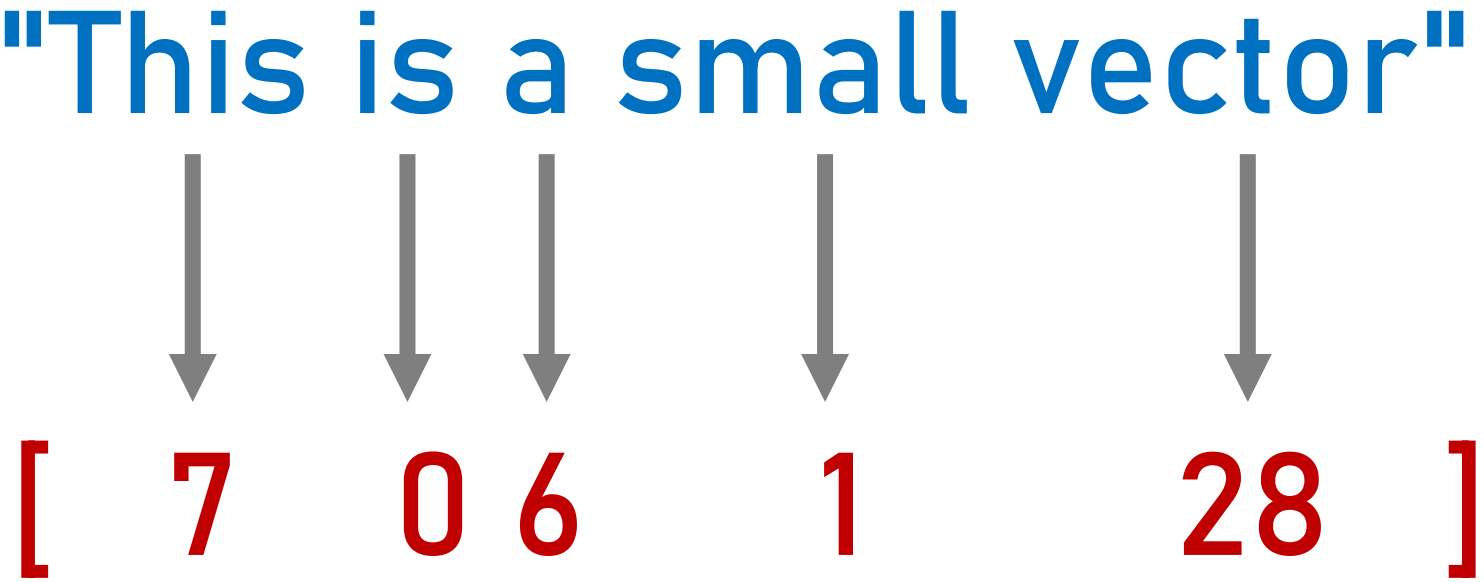
embeddng_dim = 10



# Sequence of vectors

Usually the vocabulary size is thousands : **vocab_size = 30**

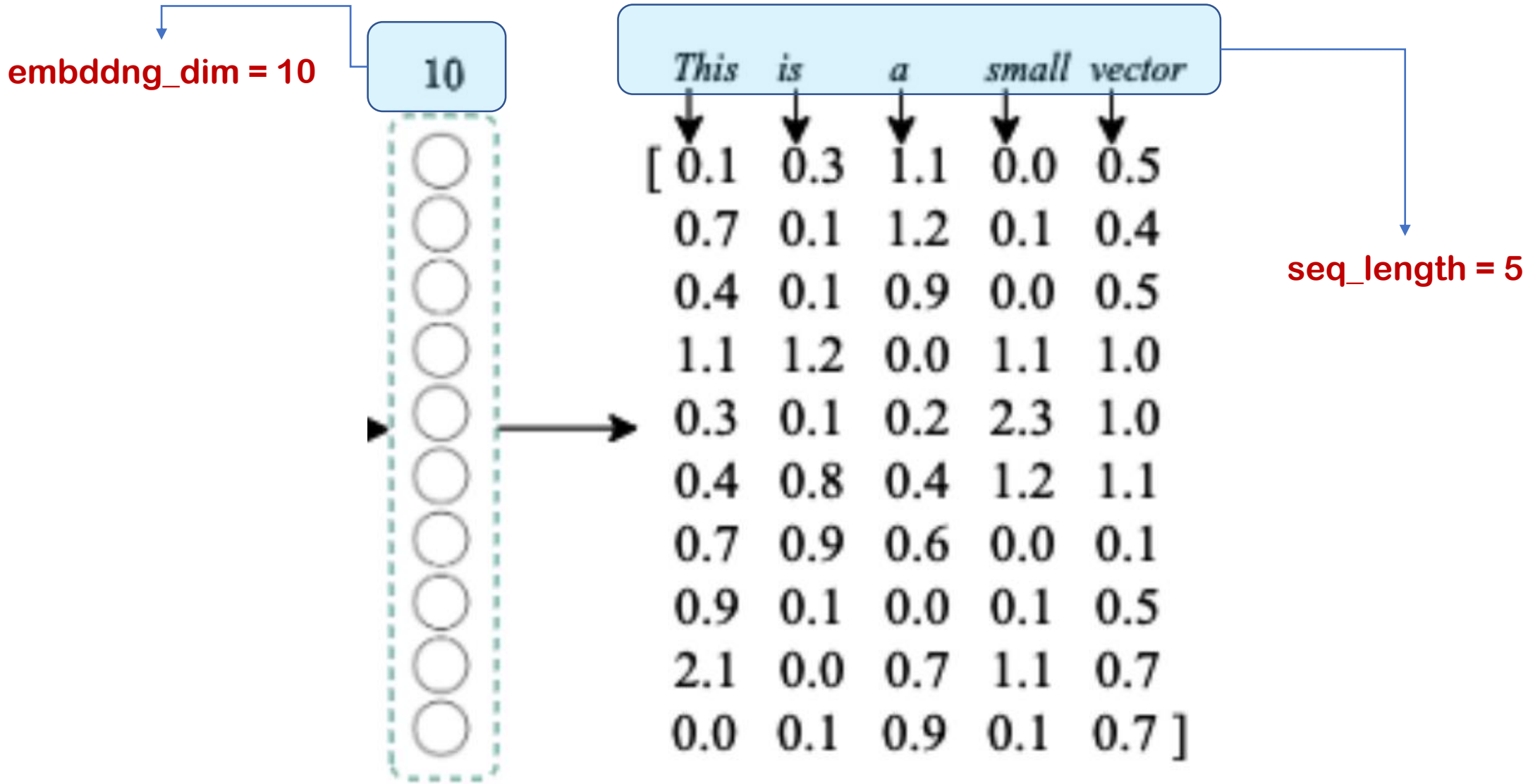
Usually, sentences consist of more than five words. **seq_length = 5**




The embedding dimension is about 50 dimensions for small models.

There are also 500 to 1,000 dimensions of a good model.

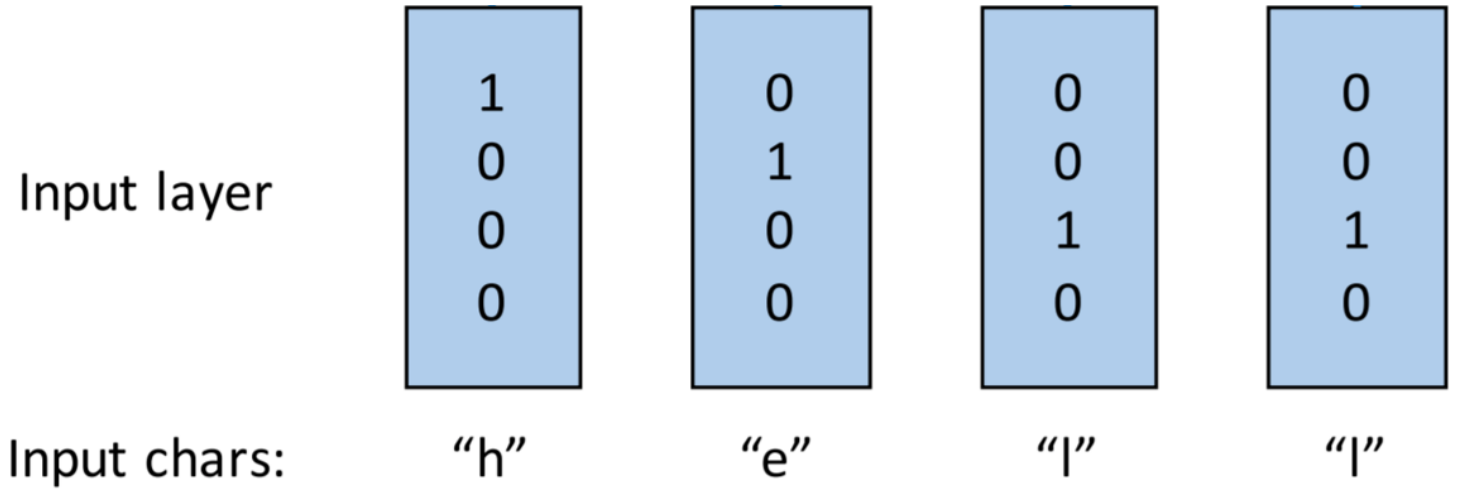
# Sequence of word embeddings



# Character-level language model

# Example training sequence : “hello”

- “The Unreasonable Effectiveness of Recurrent Neural Networks,”
  - ✓ Andrej Karpathy In 2015
- Character-level language model
  - ✓ Suppose we only had a vocabulary of four letters “hell”: [h,e,l,o]
  - ✓ We will encode each character into a vector using 1-of-k encoding



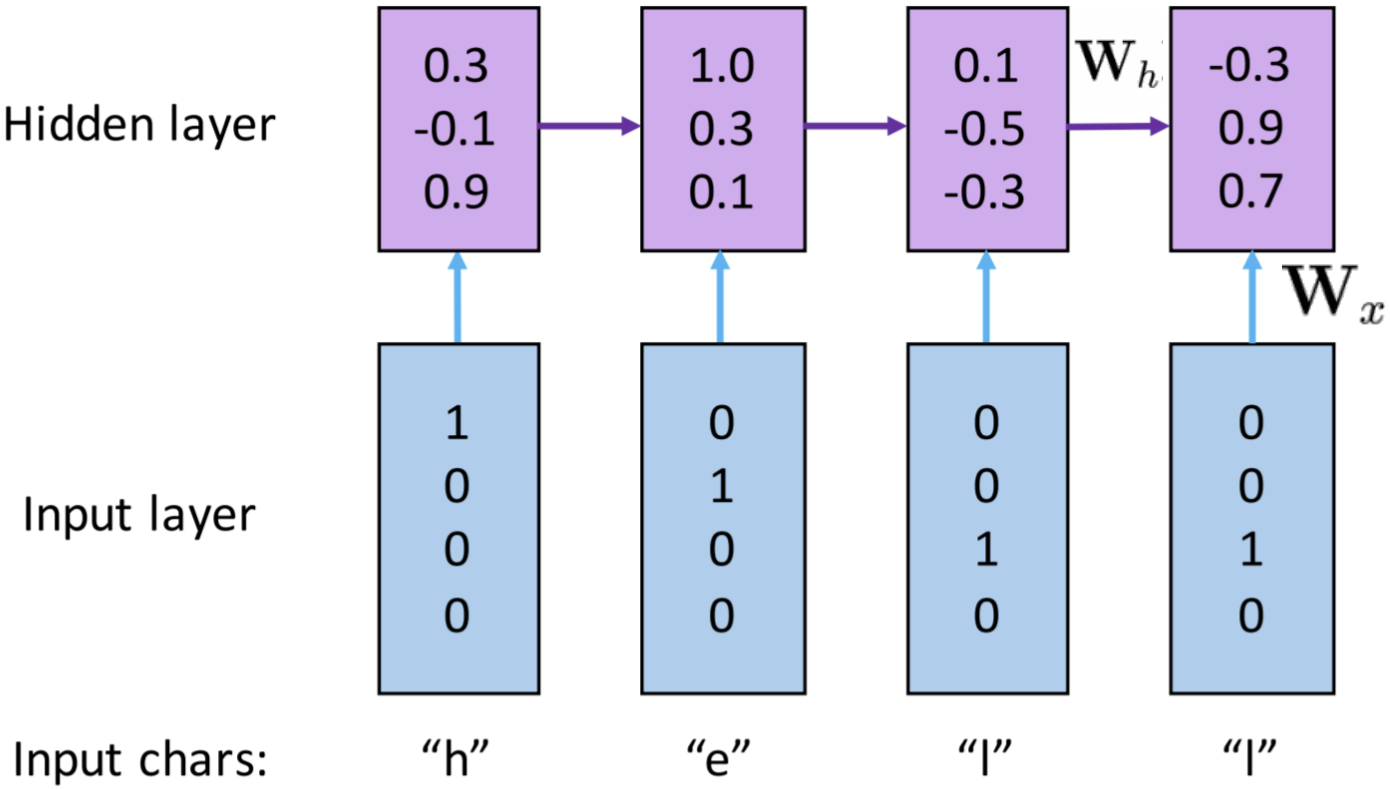
# Character-level language model

$$\mathbf{h}_t = \tanh(\mathbf{W}_h \mathbf{h}_{t-1} + \mathbf{W}_x \mathbf{x}_t)$$

**Example: Character-level Language Model**

Vocabulary: [h,e,l,o]

Example training sequence: **"hello"**

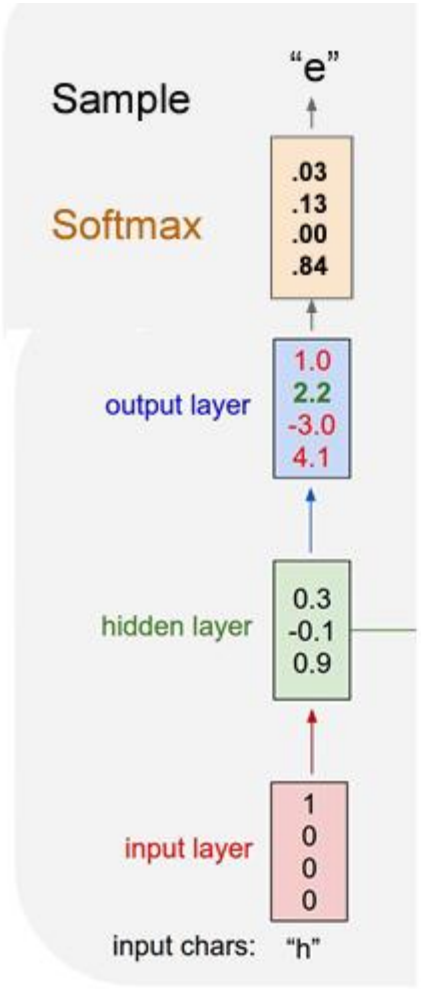


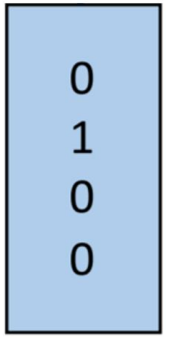
# Teacher forcing : Character-level language model

**Example:**
**Character-level**
**Language Model**
**Sampling**

Vocabulary:
[h,e,l,o]

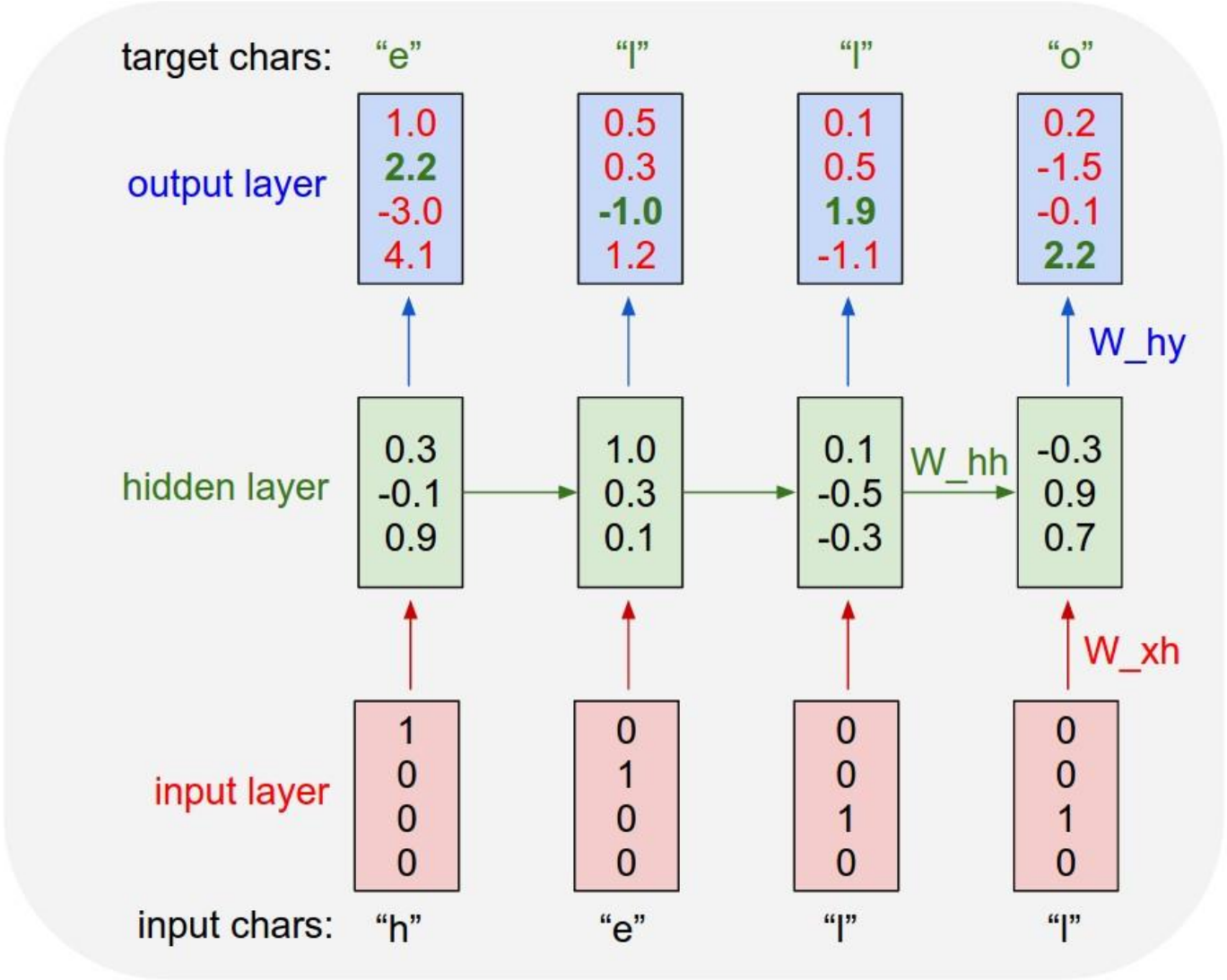
At test-time sample characters one at a time, feed back to model



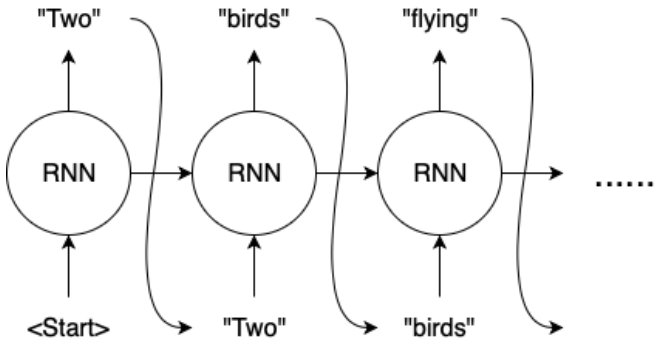

"e"

# Char-RNN

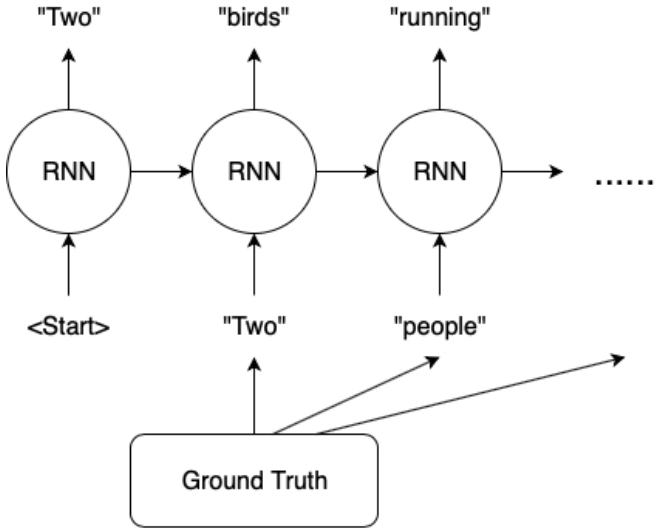



We want the green numbers to be high and red numbers to be low.

# With Teacher Forcing Learing

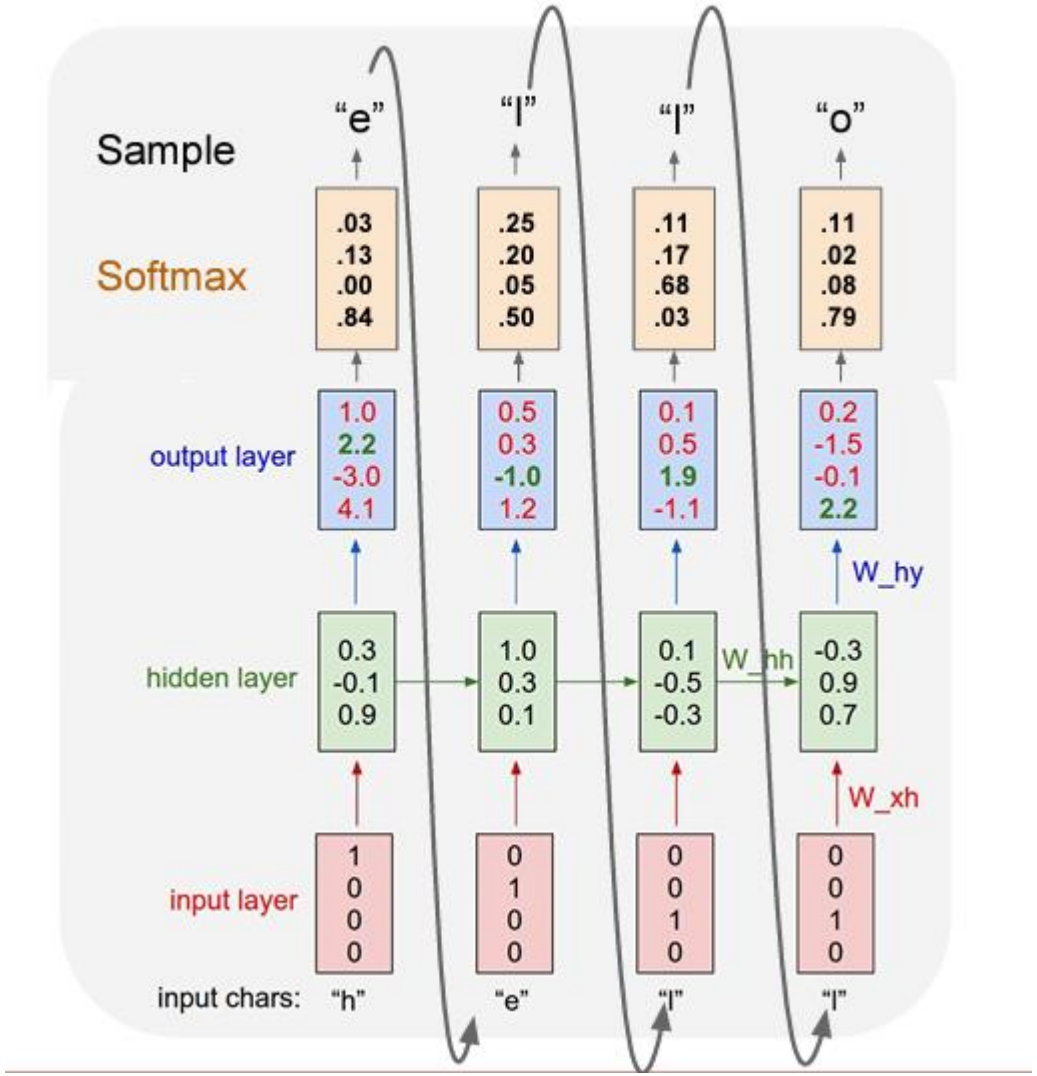



Without Teacher Forcing




With Teacher Forcing

# Teacher forcing : Character-level Language Model



# A Visual Guide to Recurrent Layers in Keras

source: https://amitness.com/2020/04/recurrent-layers-keras/

## ❖ Let's take a simple example of encoding

For simplicity, let's assume we used some word embedding to convert each word into 2 numbers.

I am groot

Credits: Marvel Studios

| Word | E1 | E2 |
|---|---|---|
| I | 0.5 | 0.4 |
| am | 0.3 | 0.1 |
| groot | 0.7 | 0.5 |

We could either use one-hot encoding, pretrained word vectors, or learn word embeddings from scratch

https://amitness.com/2020/04/recurrent-layers-keras/

# SimpleRNN (1)

## ❖ SimpleRNN with a Dense layer

✓ to build an architecture for something like sentiment analysis or text classification.

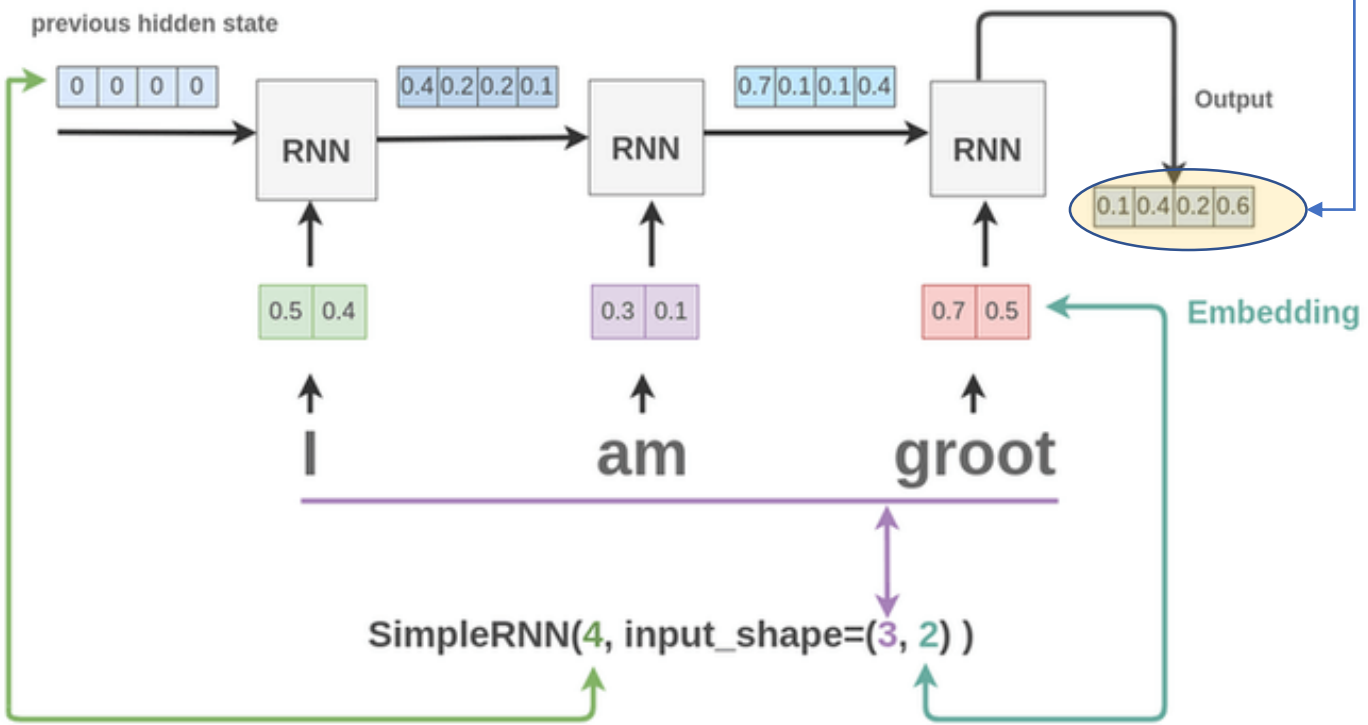
```
import tensorflow as tf
from tensorflow.keras.layers import SimpleRNN   #Dense, LSTM
# from tensorflow.keras.models import Sequential

x = tf.random.normal((1, 3, 2))

layer = SimpleRNN(4, input_shape=(3, 2))
output = layer(x)

print(output.shape)
print(x)
```

```
(1, 4)
tf.Tensor(
[[[ 0.6887584   1.3883604 ]
  [ 0.01564607 -1.4314882 ]
  [-0.05214449 -0.65099174]]], shape=(1, 3, 2), dtype=float32)
```

# SimpleRNN : Many-to-One

- we treat each word as a time-step and the embedding as features.

```
model.add(layers.SimpleRNN(4,input_shape=(3,2)))
```



# SimpleRNN (3) : return_sequences=True

```
# multiple output
layer = SimpleRNN(4, input_shape=(3, 2), return_sequences=True )
output = layer(x)
print(output.shape)
print(output)
```

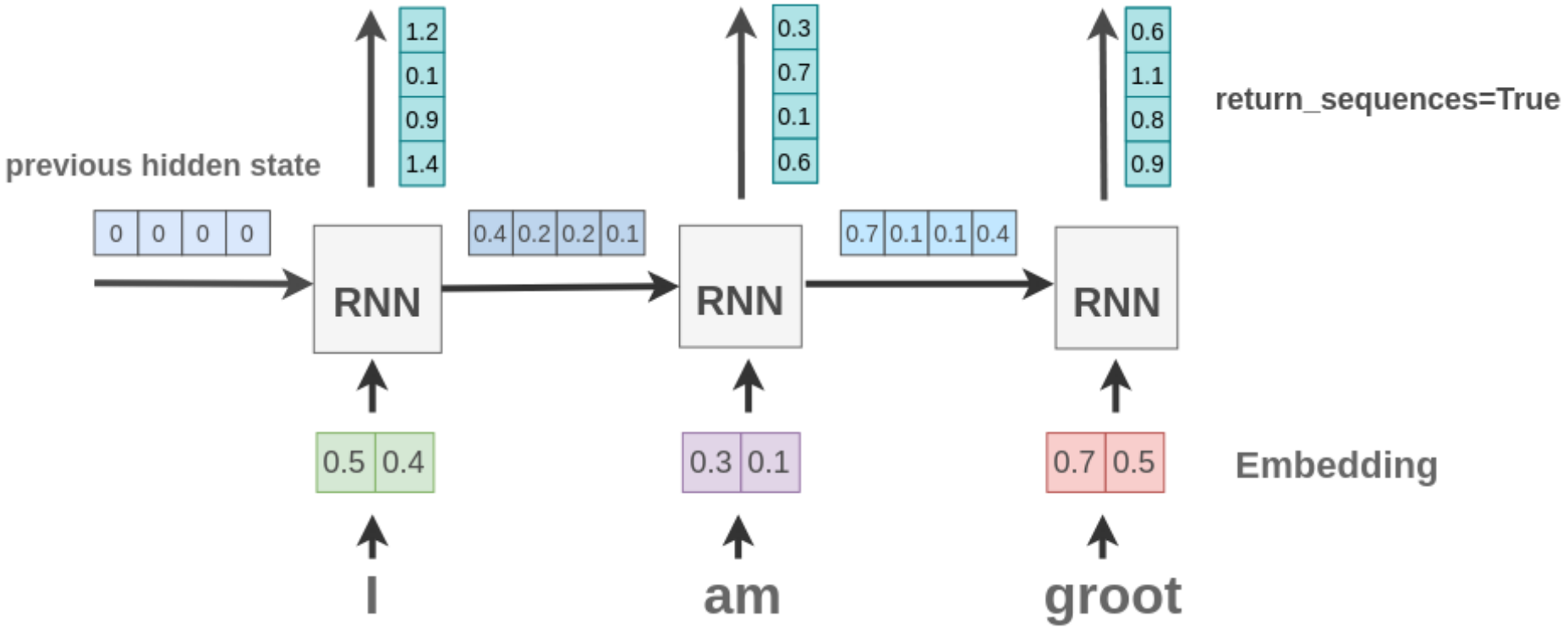
```
(1, 3, 4)
tf.Tensor(
[[[-0.6854385   0.08265962  0.30888444 -0.30752325]
  [ 0.4584542  -0.1935767  -0.91095936 -0.2416075 ]
  [ 0.7241105  -0.49960855 -0.5059616   0.7261468 ]]], shape=(1, 3, 4), dtype=float32)
```

# RNN with return_sequences : Many-to-Many

## ❖ return_sequences = True

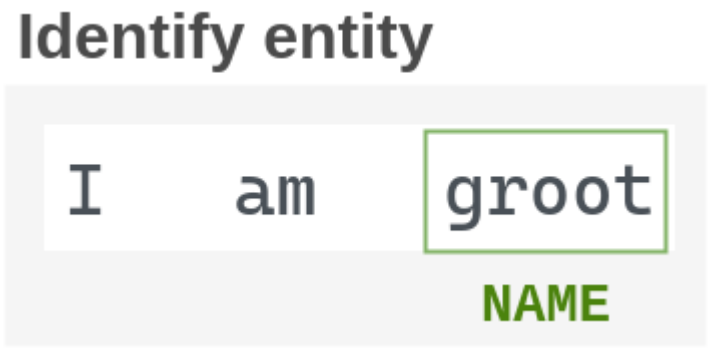
✓ True : the output from each unfolded RNN cell is returned instead of only the last cell.

```
model.add(SimpleRNN(4, input_shape=(3, 2), return_sequences=True))
```



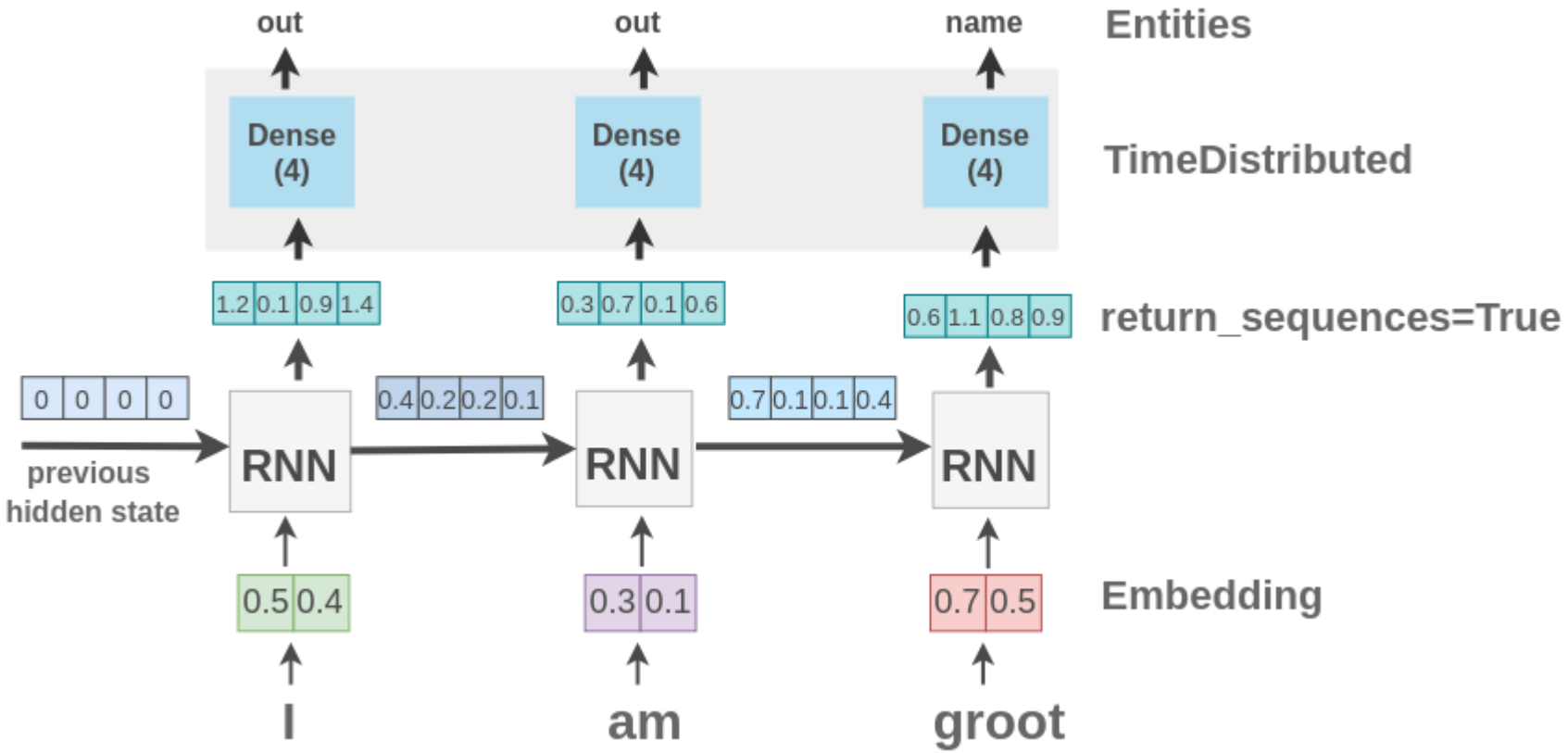
# RNN with TimeDistributed : Many-to-Many

- Suppose we want to recognize entities in a text.
  - ✓For example, in our text “I am Groot”, we want to identify “Groot” as a name.



# (3) RNN: TimeDistributed Layer

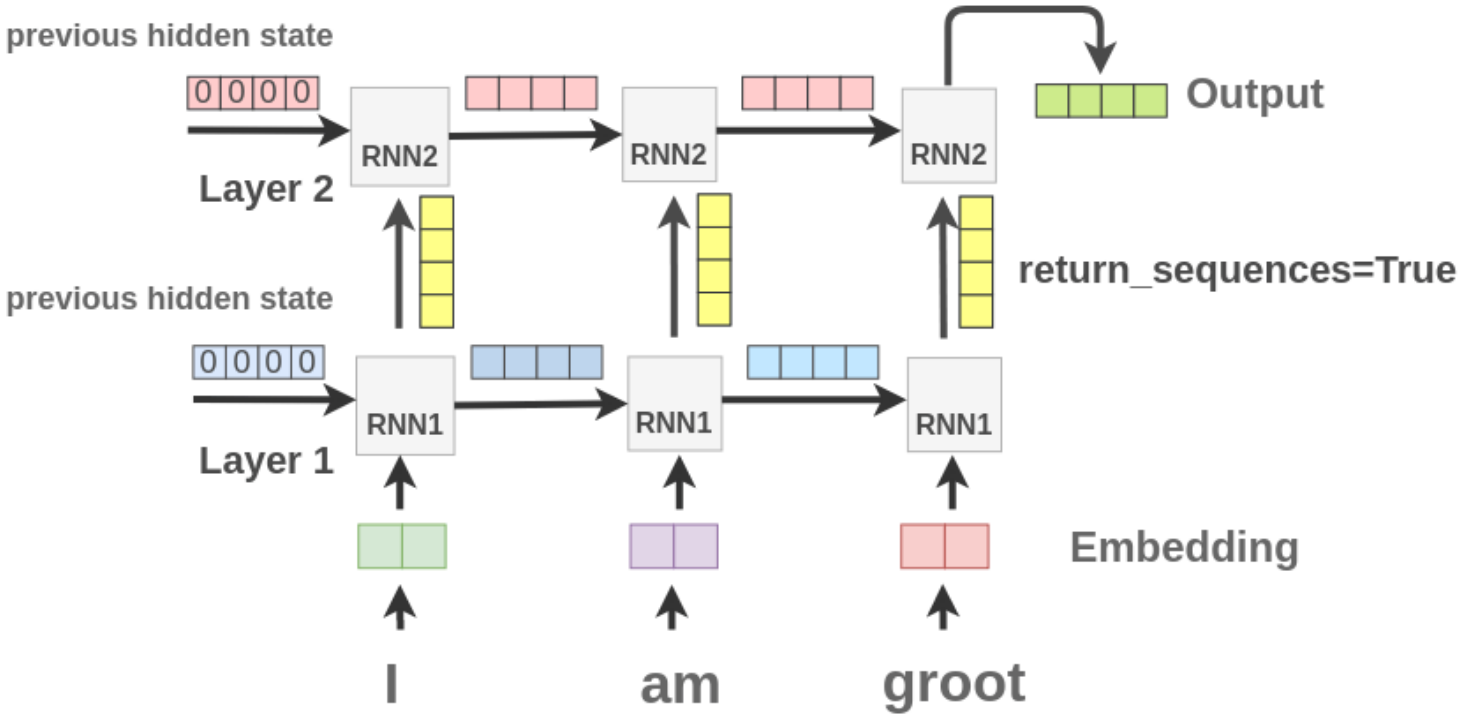
```
model.add(SimpleRNN(4, input_shape=(3, 2), return_sequences=True))
model.add(TimeDistributed(Dense(4, activation='softmax')))
```



# RNN Stacking Layer : Deep but Many-to-One

- We can also stack multiple recurrent layers one after another in Keras

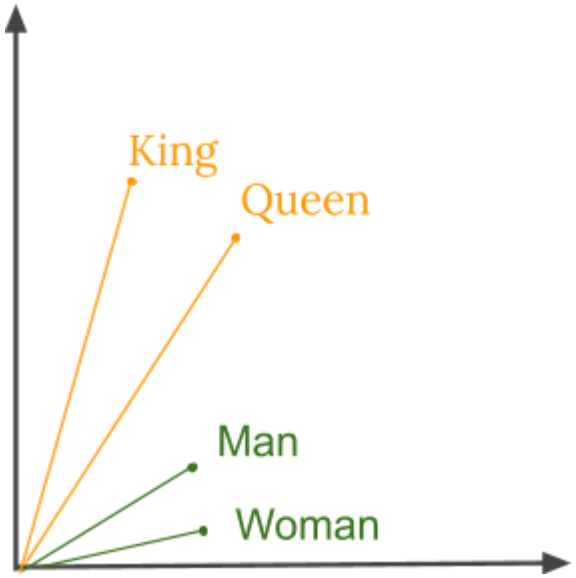
```
model.add(SimpleRNN(4, input_shape=(3, 2), return_sequences=True))
model.add(SimpleRNN(4))
```



# Simple RNN and Text

# What is Word Embedding?

- **Word embeddings are basically a form of word representation that bridges the human understanding of language to that of a machine.**
  - ✓ They have learned representations of text in an n-dimensional space where words that have the same meaning have a similar representation.
  - ✓ Meaning that two similar words are represented by almost similar vectors that are very closely placed in a vector space.



# Why Word Embeddings are used?

- we can use two more techniques
  - ✓one-hot encoding
  - ✓we can use unique numbers to represent words in a vocabulary.
- we assume we have a small vocabulary containing just four words, using the two techniques we represent the sentence ‘Come sit down’.

| | the | come | Sit | Down |
|---|---|---|---|---|
| Come | 0 | 1 | 0 | 0 |
| Sit | 0 | 0 | 1 | 0 |
| Down | 0 | 0 | 0 | 1 |

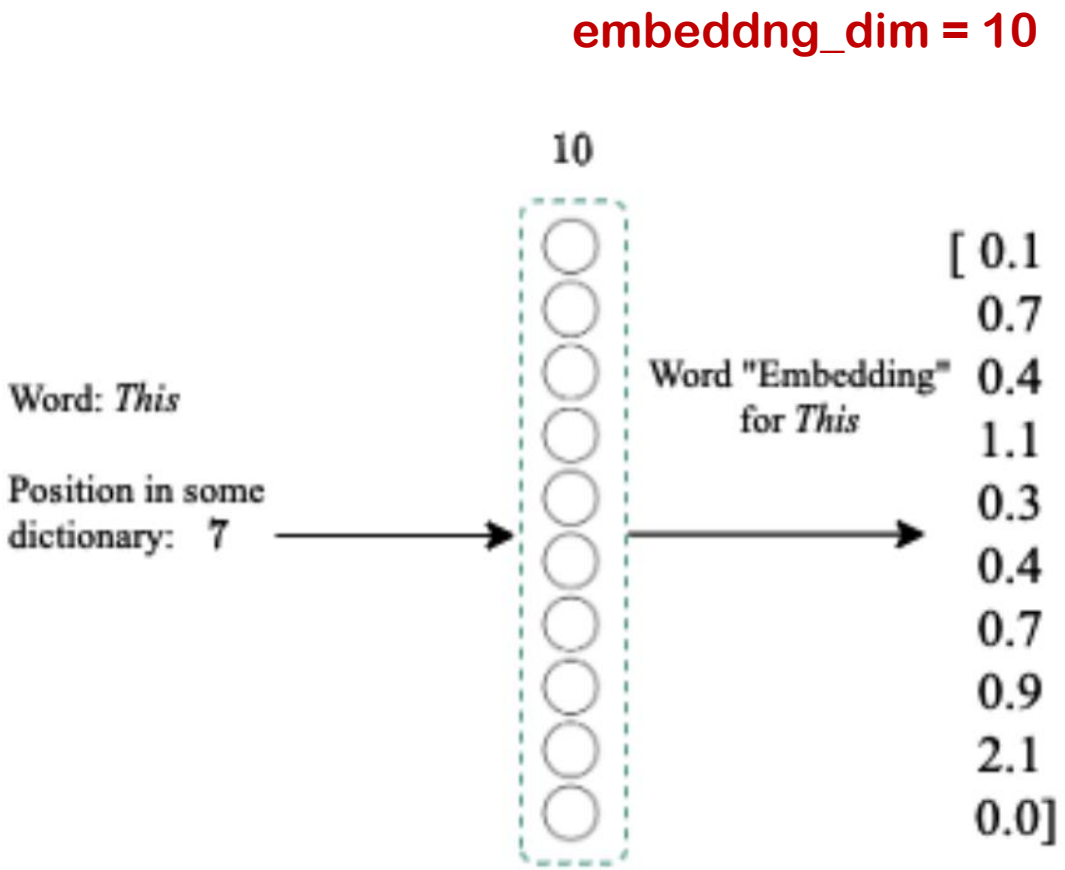
| | | | |
|---|---|---|---|
| the | 1 | come | 2 |
| Down | 3 | Sit | 4 |

| come | Sit | Down |
|---|---|---|
| 2 | 4 | 3 |

"This is a small vector"

# This

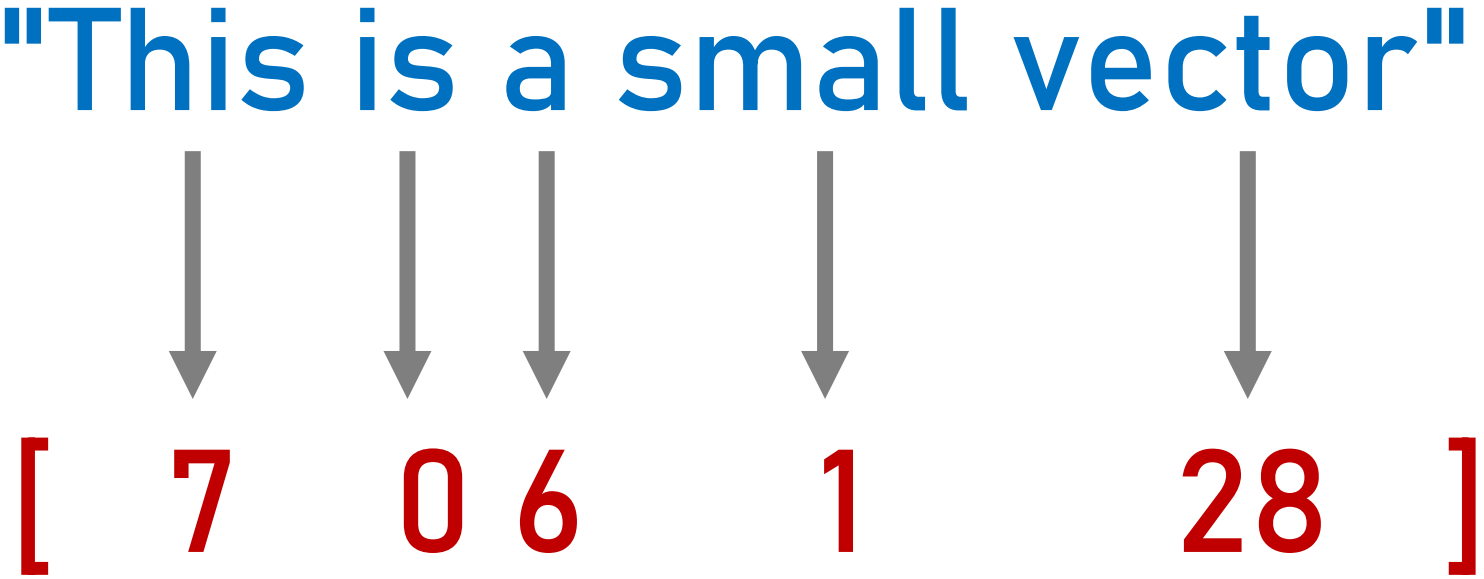
index : 7,
vector size : 10



# Sequence of vectors

Usually the vocabulary size is thousands : **vocab_size = 30**

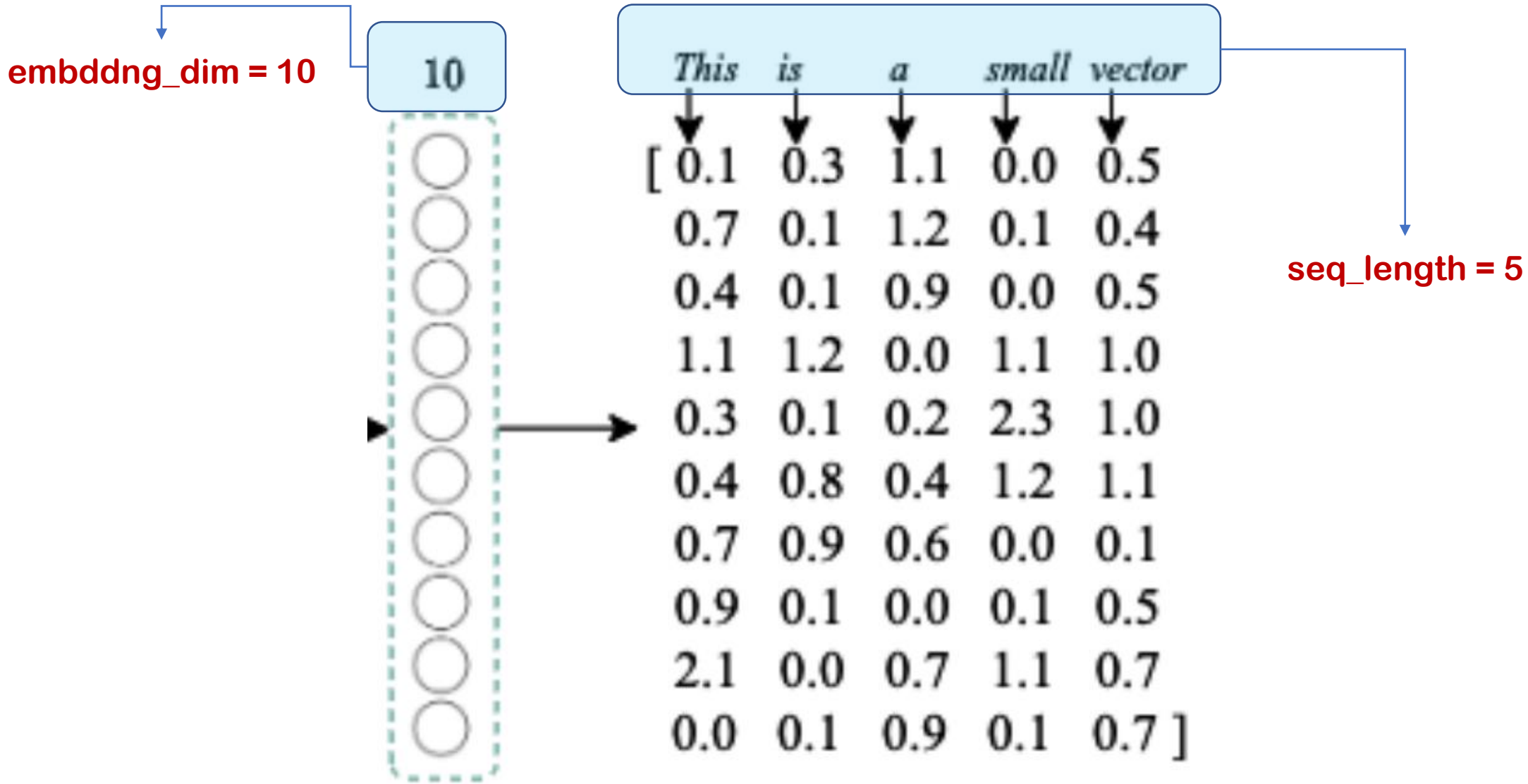
Usually, sentences consist of more than five words. **seq_length = 5**



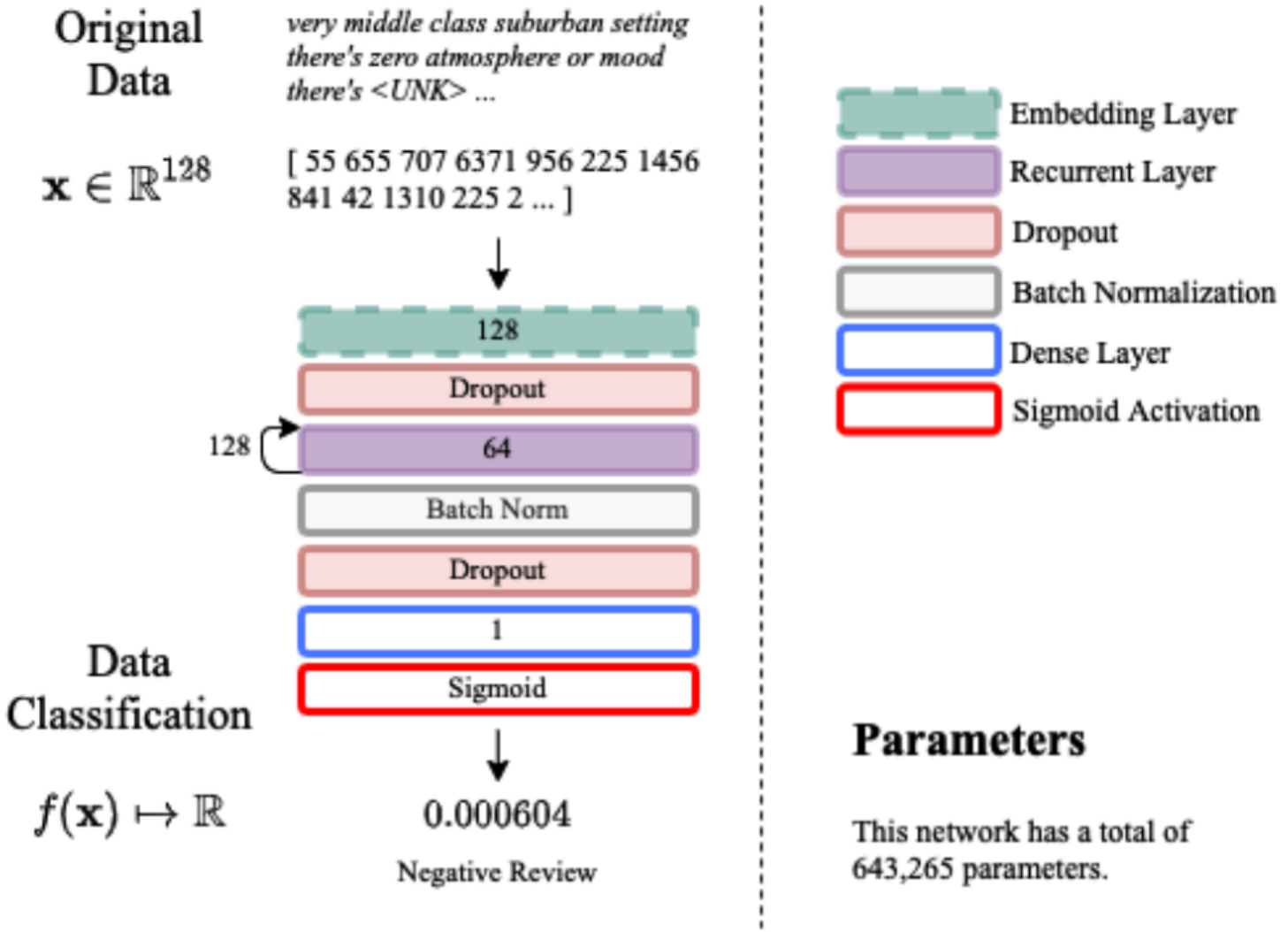

The embedding dimension is about 50 dimensions for small models.

There are also 500 to 1,000 dimensions of a good model.

# Sequence of word embeddings



# An RNN architecture for the IMDB dataset

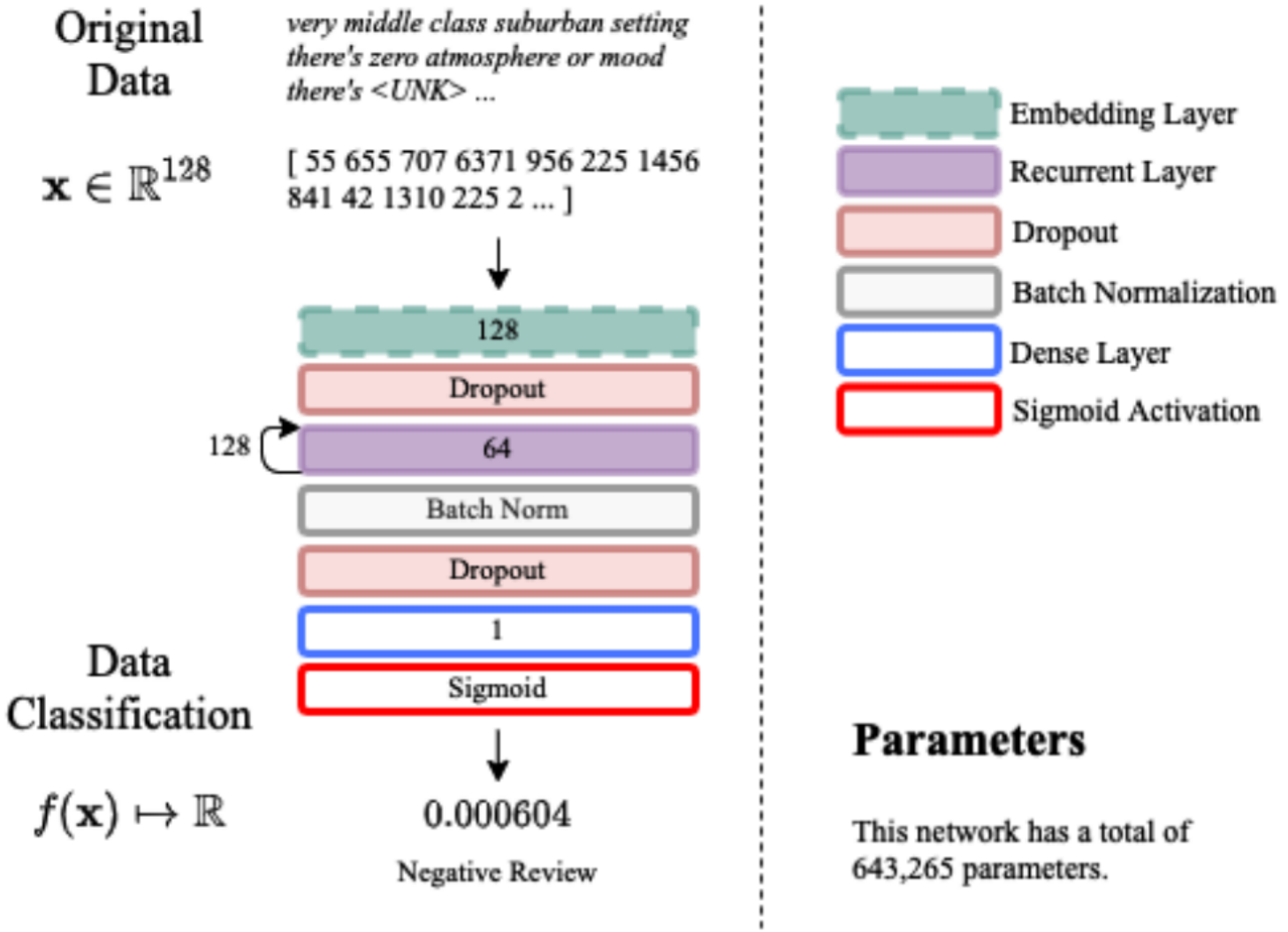



## Parameters

This network has a total of 643,265 parameters.

# IMDB Sentimental Analysis

# An RNN architecture for the IMDB dataset




## Parameters

This network has a total of 643,265 parameters.

# Typical architecture of a classification network

| Hyperparameter | Binary Classification | Multiclass classification |
|---|---|---|
| Input layer shape | Same as number of features (e.g. 5 for age, sex, height, weight, smoking status in heart disease prediction) | Same as binary classification |
| Hidden layer(s) | Problem specific, minimum = 1, maximum = unlimited | Same as binary classification |
| Neurons per hidden layer | Problem specific, generally 10 to 100 | Same as binary classification |
| Output layer shape | 1 (one class or the other) | 1 per class (e.g. 3 for food, person or dog photo) |
| Hidden activation | Usually ReLU (rectified linear unit) | Same as binary classification |
| Output activation | Sigmoid | Softmax |
| Loss function | Cross entropy (`tf.keras.losses.BinaryCrossentropy` in TensorFlow) | Cross entropy (`tf.keras.losses.CategoricalCrossentropy` in TensorFlow) |
| Optimizer | SGD (stochastic gradient descent), Adam | Same as binary classification |

***Table 1:*** *Typical architecture of a classification network.* ***Source:*** *Adapted from page 295 of Hands-On Machine Learning with Scikit-Learn, Keras & TensorFlow Book by Aurélien Géron*

# Sentiment Analysis : IMDb reviews dataset

- Internet Movie Database is an online database of information related to films, television series, home videos, video games, and streaming content online
  - ✓ IMDb began as a fan-operated movie database on the Usenet group in 1990. It is now owned and operated by **IMDb.com, Inc.**, a subsidiary of Amazon.
  - ✓ As of February 2022, the database contained some 8.7 million titles and 11.4 M person records.

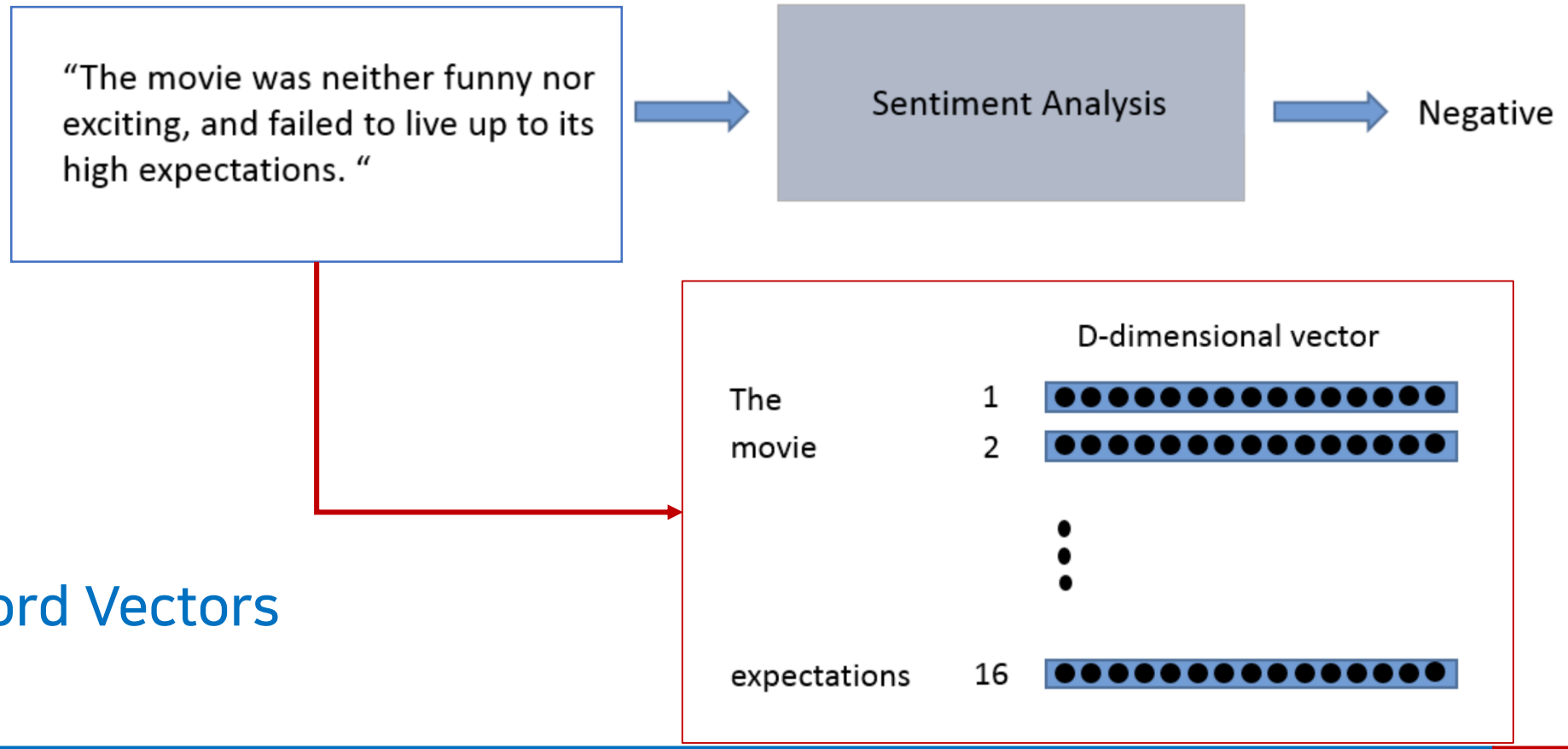
- If MNIST is the “hello world” of computer vision, the IMDb reviews dataset is the “hello world” of natural language processing
  - ✓ it consists of 50,000 movie reviews in English (25,000 for training, 25,000 for testing)

## ❖ Sentiment Analysis : IMDB (Internet Movie Database)

✓Determining the emotional tone behind a piece of text


| | review | sentiment |
|---|---|---|
| 0 | One of the other reviewers has mentioned that ... | positive |
| 1 | A wonderful little production. The filming tec... | positive |
| 2 | I thought this was a wonderful way to spend ti... | positive |
| 3 | Basically there's a family where a little boy ... | negative |
| 4 | Petter Mattei's "Love in the Time of Money" is... | positive |
| 5 | Probably my all-time favorite movie, a story o... | positive |
| 6 | I sure would like to see a resurrection of a u... | positive |
| 7 | This show was an amazing, fresh & innovative i... | negative |
| 8 | Encouraged by the positive comments about this... | negative |

- When you think of NLP tasks, however, a data pipeline like this may come to mind.




- Word Vectors

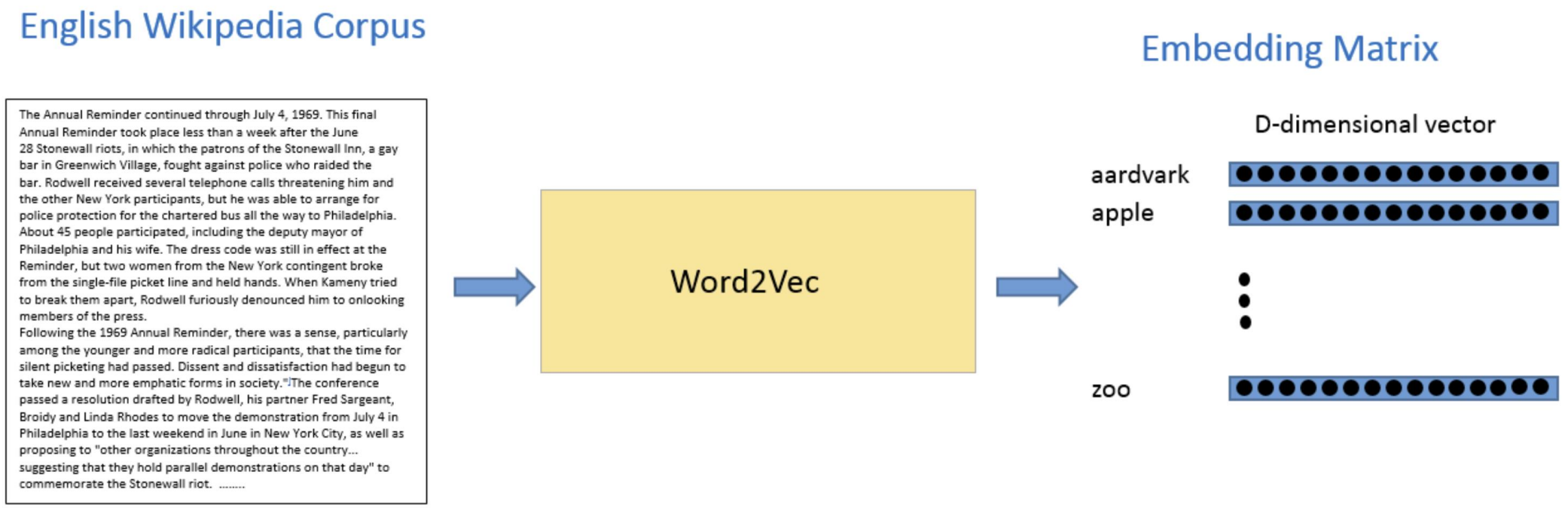
# Word embedding : Word2Vec

- The vector representation of a word is also known as a word embedding.
- Word2Vec :
  - ✓ the model creates word vectors by looking at the context with which words appear in sentences.

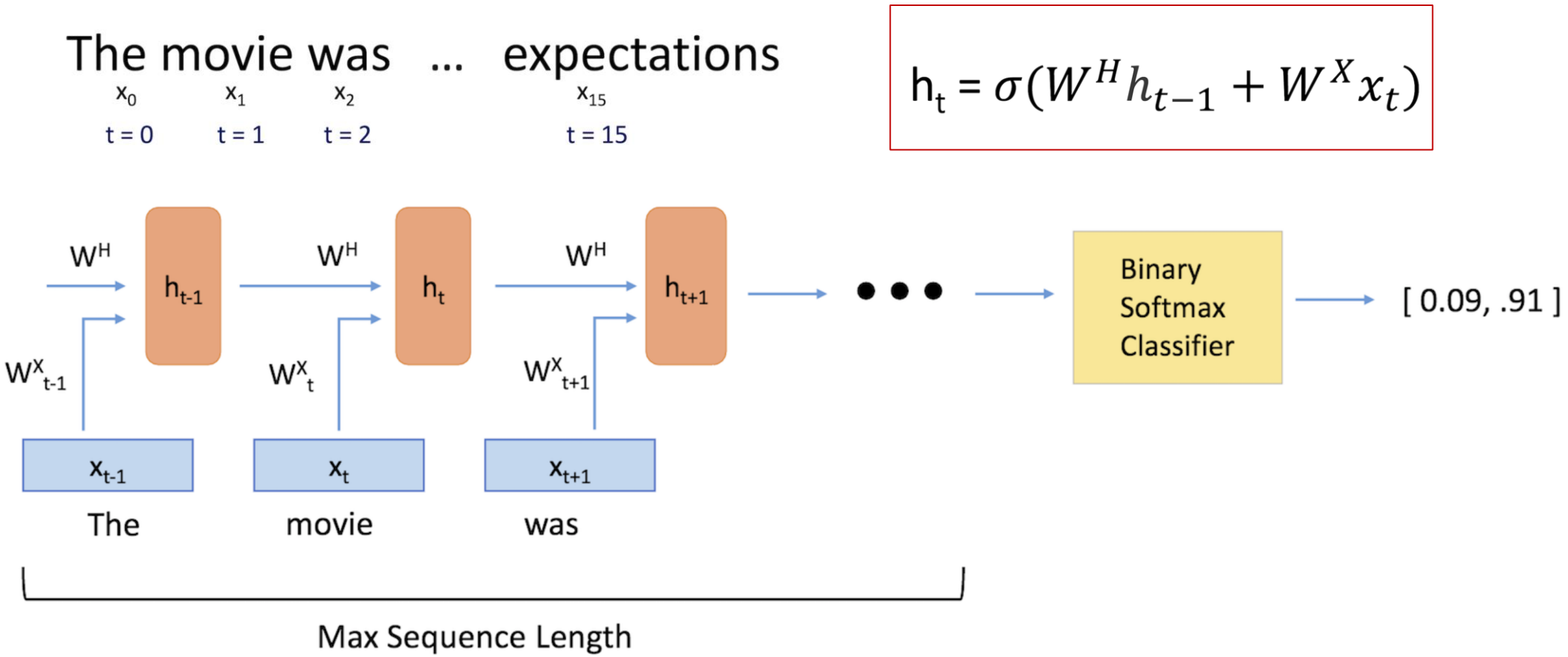
I love taking long walks on the beach.
My friends told me that they love popcorn.
⋮
The relatives adore the baby's cute face.
I adore his sense of humor.

# Embedding matrix

- The output of a Word2Vec model is called an embedding matrix
  - ✓This embedding matrix will contain vectors for every distinct word in the training corpus.



- In RNNs, each word in an input sequence will be associated with a specific time step.



# Let's Code:
# IMDB Dataset

# Computing Device settings

```
import tensorflow as tf


gpus = tf.config.experimental.list_physical_devices('GPU')
for gpu in gpus:
    tf.config.experimental.set_memory_growth(gpu, True)
```

```
print("Version: ", tf.__version__)
print("GPU is", "available" if tf.config.list_physical_devices('GPU') else "NOT AVAILABLE")
```

```
Version:  2.7.0
GPU is available
```

```
import numpy as np
import matplotlib.pyplot as plt
```

# Download the IMDB dataset

```
from tensorflow.keras.datasets import imdb
```

```
vocal_size = 10000
(X_train, y_train), (X_test, y_test) =imdb.load_data(num_words=vocal_size)
```

```
print("Training entries: {}, labels: {}".format(len(X_train), len(y_train)))
print("Test entries: {}, labels: {}".format(len(X_test), len(y_test)))
```

```
Training entries: 25000, labels: 25000
Test entries: 25000, labels: 25000
```

# The preprocessed IMDB dataset

- Internet Movie Database
  - ✓with a simple binary target for each review indicating whether it is negative (0) or positive (1).
- We will use the preprocessed dataset

```
(X_train, y_train), (X_test, y_test) = keras.datasets.imdb.load_data()
```

# Where are the movie reviews?

X_train consists of a list of reviews, each of which is represented as a numpy array of integers.

where each integer represents a word

```
print(X_train[0][:10])
print(y_train[0])
```

```
[1, 14, 22, 16, 43, 530, 973, 1622, 1385, 65]
1
```

IMDB:
All punctuation was removed, and then words were converted to lowercase, split by spaces, and finally indexed by frequency (so low integers correspond to frequent words)

# Special tokens

- The integers 0, 1, and 2 are special:
  - ✓ they represent the padding token, the *start-of-sequence* (SSS) token, and unknown words, respectively.

```
>>> word_index = keras.datasets.imdb.get_word_index()
>>> id_to_word = {id_ + 3: word for word, id_ in word_index.items()}
>>> for id_, token in enumerate(("<pad>", "<sos>", "<unk>")):
... id_to_word[id_] = token
...
>>> " ".join([id_to_word[id_] for id_ in X_train[0][:10]])
'<sos> this film was just brilliant casting location scenery story'
```

```
[1, 14, 22, 16, 43, 530, 973, 1622, 1385, 65]
```

# frequent words

```
imdb_get_word_index = {}
for key, value in imdb.get_word_index().items():
    imdb_get_word_index[value] = key
```

```
for i in range(1, 12):
    print('{}-th word which is used the most frequently = {}'.format(i, imdb_get_word_index[i]))
```

```
1-th word which is used the most frequently = the
2-th word which is used the most frequently = and
3-th word which is used the most frequently = a
4-th word which is used the most frequently = of
5-th word which is used the most frequently = to
6-th word which is used the most frequently = is
7-th word which is used the most frequently = br
8-th word which is used the most frequently = in
9-th word which is used the most frequently = it
10-th word which is used the most frequently = i
11-th word which is used the most frequently = this
```

Low integers correspond to frequent words

# Convert the integers back to words : Decode

```
reverse_word_index = dict([(value, key) for (key, value) in word_index.items()])
```

```
' '.join([reverse_word_index.get(i, '?') for i in X_train[0][:10]])
```

```
]: '<START> this film was just brilliant casting location scenery story'
```

If you want to visualize a review, you can decode it like this

When encoding words, it filters out a lot of characters, including most punctuation, line breaks, and tabs

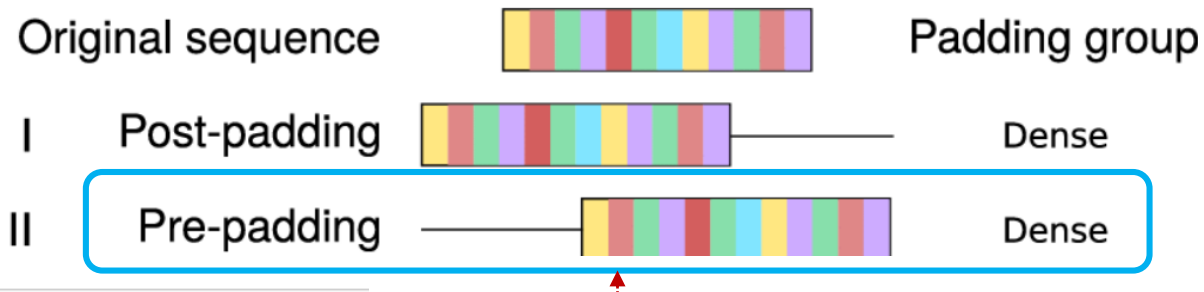
Most importantly, it uses spaces to identify word boundaries.

Even in English, spaces are not always the best way to tokenize text:
think of “San Francisco” or “#ILoveDeepLearning.”

# Prepare the data

```
seq_len = 256
print('Before pad_sequences: ', len(X_train[0]))
```

```
Before pad_sequences:  218
```




```
from tensorflow.keras.preprocessing.sequence import pad_sequences
```

```
X_train = pad_sequences(X_train, maxlen=seq_len, padding = 'pre')
X_test = pad_sequences(X_test, maxlen=seq_len, padding = 'pre')
print('After X_train pad_sequences: ', len(X_train[0]))
print('After X_test pad_sequences: ', len(X_test[0]))
```

```
After X_train pad_sequences:  256
After X_test pad_sequences:  256
```

```
y_train = np.asarray(y_train).astype("float32")
y_test = np.asarray(y_test).astype("float32")
```

Label is an integer, but there is no error if you change it to float32.

# Build the model

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, SimpleRNN, Dense
from tensorflow.keras.layers import Dropout, BatchNormalization, Bidirectional, LSTM
```

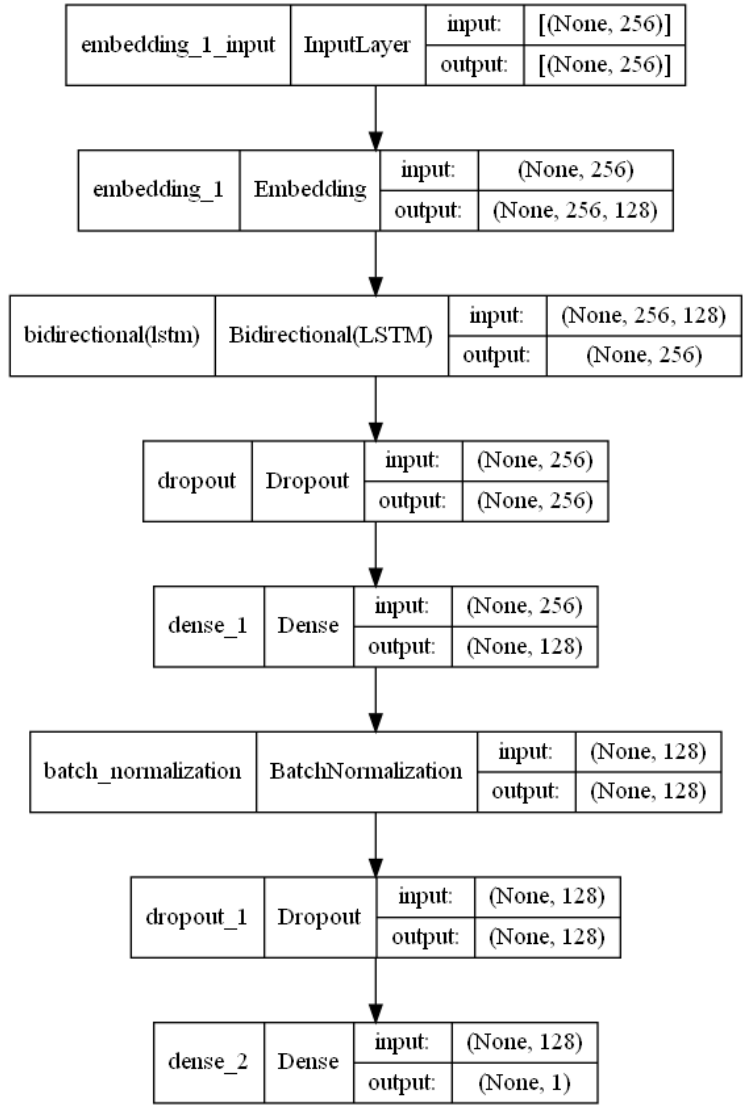
```
model_1 = Sequential()
model_1.add(Embedding(vocal_size, embedding_dim, input_length = seq_len))
model_1.add(SimpleRNN(embedding_dim))
model_1.add(Dense(1, activation='sigmoid'))
```

embedding_dim= 128 seq_len= 256

The first layer is an Embedding layer. the resulting dimensions are: (batch, sequence, embedding, seq_len).

# model 2

```
model_2 = Sequential()

model_2 .add(Embedding(vocal_size, embedding_dim, input_length = seq_len))
model_2 .add(Bidirectional(LSTM(embedding_dim)))
model_2 .add(Dropout(0.2))
model_2 .add(Dense(embedding_dim, activation = 'relu'))
model_2 .add(BatchNormalization())
model_2 .add(Dropout(0.2))
model_2 .add(Dense(1, activation='sigmoid'))
```



# model compile and tiny train dataset

```
# Spliting data further for validation
num_train = 4000
num_val   = 1000
partial_X_train = X_train[:num_train]
partial_y_train = y_train[:num_train]
X_val = X_train[:num_val]
y_val = y_train[:num_val]
```

```
print(partial_X_train.shape, partial_y_train.shape)
print(X_val.shape, y_val.shape)
```

```
(4000, 256) (4000,)
(1000, 256) (1000,)
```

Let's reduce the size of the training data and use it

Calculating using the entire data takes a very long time

Colab: Elapsed 2933.763 seconds.

# Train the model

```
import time
start = time.perf_counter()

history = model.fit(partial_X_train, partial_y_train, epochs=30,  batch_size=32,
                    validation_data=(X_val, y_val), verbose=1)

elapsed = time.perf_counter() - start
print('==========================₩n Elapsed %.3f seconds.' % elapsed)
```

```
Epoch 30/30
125/125 [==============================] - 9s 74ms/step - loss: 0.0051 - acc: 0.9987
==========================
 Elapsed 274.534 seconds.
```

This is the case calculated from my laptop GPU

# Evaluate the model

```
X_test.shape, y_test.shape
```

```
]:  ((25000, 256), (25000,))
```

```
test_loss, test_acurracy = model.evaluate(X_test, y_test)
print("Test accuracy: {}".format(test_acurracy))
print("Test loss: {}".format(test_loss))
```

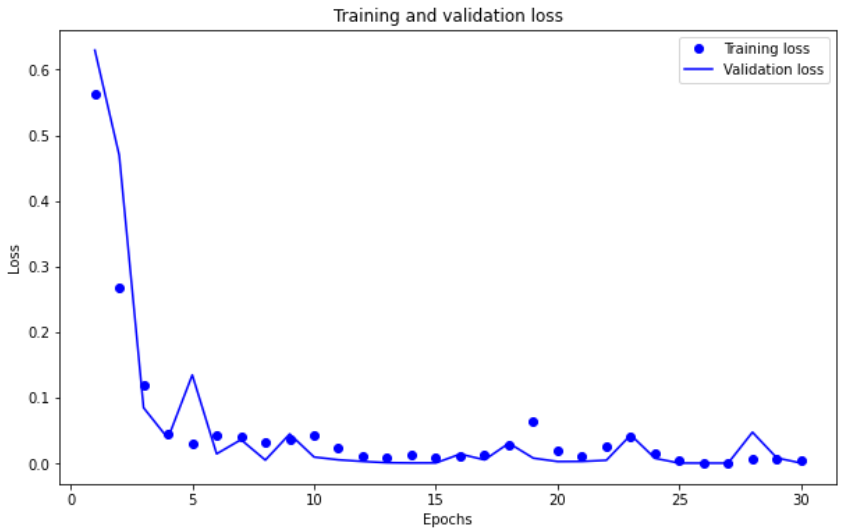
```
782/782 [==============================] - 23s 29ms/step - loss: 1.4613 - acc: 0.7792
Test accuracy: 0.779160022735957
Test loss: 1.4612730741500854
```

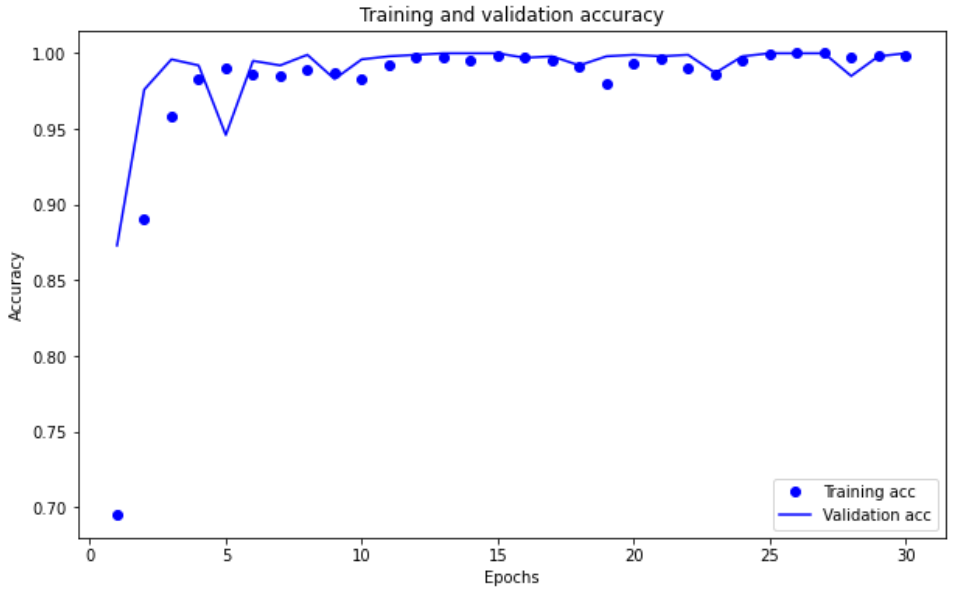
The test model runs only once, so it's very fast

# Decode

```
for i in range(2):
    INDEX_FROM=3    # word index offset
    word_to_id = imdb.get_word_index()
    word_to_id = {k:(v+INDEX_FROM) for k,v in word_to_id.items()}
    word_to_id["<PAD>"] = 0
    word_to_id["<START>"] = 1
    word_to_id["<UNK>"] = 2
    word_to_id["<UNUSED>"] = 3

    id_to_word = {value:key for key,value in word_to_id.items()}
    print('================================================')
    print(f'Sample = {i} | Length = {len(X_test[i])}')
    print('================================================')
    print(' '.join(id_to_word[id] for id in X_test[i] ))
```

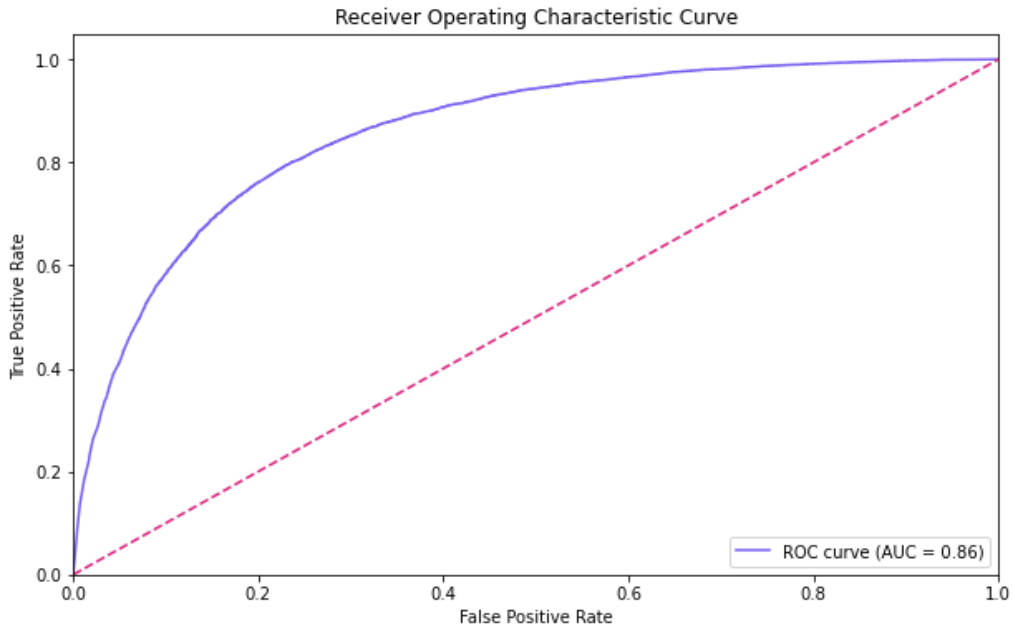
# Create a graph of accuracy and loss over time



# Model Evaluation : ROC Curve

```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import balanced_accuracy_score
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt
import numpy as np

y_hat = model.predict(X_test)

# Compute ROC curve and ROC area for each class
fpr, tpr, thresholds = roc_curve(y_test, y_hat)
roc_auc = auc(fpr, tpr)
```



# Threshold value and Score

```
optimal_idx = np.argmax(tpr - fpr)
optimal_threshold = thresholds[optimal_idx]
print("Threshold value is:", optimal_threshold)
y_pred = np.where(y_hat>=optimal_threshold, 1, 0)
print('Balanced Accuracy Score:',balanced_accuracy_score(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
```

```
Threshold value is: 0.87541753
Balanced Accuracy Score: 0.7824
[[9625 2875]
 [2565 9935]]
```

# APPENDIX

# Load Original IMDB reviews : With Preprocess

# Load Original IMDB reviews

- let's load the original IMDb reviews, as text, using TensorFlow Datasets

```
import tensorflow_datasets as tfds

datasets, info = tfds.load("imdb_reviews", as_supervised=True, with_info=True)
```

```
Downloading and preparing dataset imdb_reviews/plain_text/1.0.0 (download: 80.23 MiB, generated:
Dl Completed...: 100% 1/1 [00:10<00:00, 10.20s/ url]
Dl Size...: 100% 80/80 [00:10<00:00, 16.18 MiB/s]

Shuffling and writing examples to /root/tensorflow_datasets/imdb_reviews/plain_text/1.0.0.incompl
100% 24999/25000 [00:00<00:00, 81873.44 examples/s]
Shuffling and writing examples to /root/tensorflow_datasets/imdb_reviews/plain_text/1.0.0.incompl
100% 24999/25000 [00:00<00:00, 94731.73 examples/s]
Shuffling and writing examples to /root/tensorflow_datasets/imdb_reviews/plain_text/1.0.0.incompl
100% 49999/50000 [00:00<00:00, 177952.35 examples/s]
WARNING:absl:Dataset is using deprecated text encoder API which will be removed soon. Please use
Dataset imdb_reviews downloaded and prepared to /root/tensorflow_datasets/imdb_reviews/plain_text
```

# Load Original IMDB reviews

```
datasets.keys()
```

```
dict_keys(['test', 'train', 'unsupervised'])
```

```
train_size = info.splits["train"].num_examples
test_size = info.splits["test"].num_examples
```

```
train_size, test_size
```

```
(25000, 25000)
```

# Load Original IMDB reviews

```
for X_batch, y_batch in datasets["train"].batch(2).take(1):
    for review, label in zip(X_batch.numpy(), y_batch.numpy()):
        print("Review:", review.decode("utf-8")[:200], "...")
        print("Label:", label, "= Positive" if label else "= Negative")
        print()
```

Review: This was an absolutely terrible movie. Don't be lured in by Christopher Walken or Michael Ironside. Both are great actors, but this must simply be their worst role in history. Even their great acting ... Label: 0 = Negative

Review: I have been known to fall asleep during films, but this is usually due to a combination of things including, really tired, being warm and comfortable on the sette and having just eaten a lot. However ... Label: 0 = Negative

# let's write the preprocessing function

```
def preprocess(X_batch, y_batch):
    X_batch = tf.strings.substr(X_batch, 0, 300)
    X_batch = tf.strings.regex_replace(X_batch, rb"<br\s*/?>", b" ")
    X_batch = tf.strings.regex_replace(X_batch, b"[^a-zA-Z']", b" ")
    X_batch = tf.strings.split(X_batch) # splits the reviews by the spaces
    return X_batch.to_tensor(default_value=b"<pad>"), y_batch
```

truncating the reviews, keeping only the first 300 characters of each

padding all reviews with the padding token,
so that they all have the same length

Then it uses *regular expressions* to replace <br /> tags with spaces, and to replace any characters other than letters and quotes with spaces.

"Well, I can't<br />" → "Well I can't".

# preprocess(X_batch, y_batch)

```
preprocess(X_batch, y_batch)
```

```
(<tf.Tensor: shape=(2, 53), dtype=string, numpy= array([[b'This', b'was', b'an',
b'absolutely', b'terrible', b'movie', b"Don't", b'be', b'lured', b'in', b'by',
b'Christopher', b'Walken', b'or', b'Michael', b'Ironside', b'Both', b'are',
b'great', b'actors', b'but', b'this', b'must', b'simply', b'be', b'their',
b'worst', b'role', b'in', b'history', b'Even', b'their', b'great', b'acting',
b'could', b'not', b'redeem', b'this', b"movie's", b'ridiculous', b'storyline',
b'This', b'movie', b'is', b'an', b'early', b'nineties', b'US', b'propaganda',
b'pi', b'<pad>', b'<pad>', b'<pad>'], [b'I', b'have', b'been', b'known', b'to',
b'fall', b'asleep', b'during', b'films', b'but', b'this', b'is', b'usually',
b'due', b'to', b'a', b'combination', b'of', b'things', b'including', b'really',
b'tired', b'being', b'warm', b'and', b'comfortable', b'on', b'the', b'sette',
b'and', b'having', b'just', b'eaten', b'a', b'lot', b'However', b'on', b'this',
b'occasion', b'I', b'fell', b'asleep', b'because', b'the', b'film', b'was',
b'rubbish', b'The', b'plot', b'development', b'was', b'constant', b'Cons']],
dtype=object)>, <tf.Tensor: shape=(2,), dtype=int64, numpy=array([0, 0])>)
```

# We need to construct the vocabulary

- count the number of occurrences of each word:

```
from collections import Counter
vocabulary = Counter()
for X_batch, y_batch in datasets["train"].batch(32).map(preprocess):
    for review in X_batch:
        vocabulary.update(list(review.numpy()))
```

- Let's look at the three most common words:

```
>>> vocabulary.most_common()[:3]
[(b'<pad>', 215797), (b'the', 61137), (b'a', 38564)]
```

```
len(vocabulary)
```

```
53893
```

# keeping only the 10,000 most common words

- We probably don’t need our model to know all the words in the dictionary to get good performance, let’s truncate the vocabulary,

```
vocab_size = 10000
truncated_vocabulary = [
    word for word, count in vocabulary.most_common()[:vocab_size]]
```

# Replace each word with its ID

- we need to add a preprocessing step to replace each word with its ID (i.e., its index in the vocabulary).

```
vocab_size = 10000
truncated_vocabulary = [
    word for word, count in vocabulary.most_common()[:vocab_size]]
```

```
word_to_id = {word: index for index, word in enumerate(truncated_vocabulary)}
for word in b"This movie was faaaaaantastic".split():
    print(word_to_id.get(word) or vocab_size)
```

```
22
12
11
10000
```

# create a lookup table

- using 1,000 out-of-vocabulary (oov) buckets

```
words = tf.constant(truncated_vocabulary)
word_ids = tf.range(len(truncated_vocabulary), dtype=tf.int64)
vocab_init = tf.lookup.KeyValueTensorInitializer(words, word_ids)
num_oov_buckets = 1000
table = tf.lookup.StaticVocabularyTable(vocab_init, num_oov_buckets)
```

# Use this table to look up the IDs of a few words:

it was mapped to one of the oov buckets, with an ID greater than or equal to 10,000.

```
table.lookup(tf.constant([b"This movie was faaaaaantastic".split()]))
```

```
<tf.Tensor: shape=(1, 4), dtype=int64, numpy=array([[   22,    12,    11, 10053]])>
```

Note that the words “this,” “movie,” and “was” were found in the table, so their IDs are lower than 10,000

# encode words

- Now we are ready to create the final training set.
  - ✓We batch the reviews, then convert them to short sequences of words using the preprocess()
  - ✓then encode these words using a simple encode_words()  that uses the table
  - ✓and finally prefetch the next batch:

```
def encode_words(X_batch, y_batch):
    return table.lookup(X_batch), y_batch

train_set = datasets["train"].repeat().batch(32).map(preprocess)
train_set = train_set.map(encode_words).prefetch(1)
```

# X_batch, y_batch shape

```
for X_batch, y_batch in train_set.take(1):
    print(X_batch)
    print(y_batch)
```

```
tf.Tensor(
[[  22   11   28 ...    0    0    0]
 [   6   21   70 ...    0    0    0]
 [4099 6881    1 ...    0    0    0]
 ...
 [  22   12  118 ...  331 1047    0]
 [1757 4101  451 ...    0    0    0]
 [3365 4392    6 ...    0    0    0]], shape=(32, 60), dtype=int64)
tf.Tensor([0 0 0 1 1 1 0 0 0 0 0 1 1 0 1 0 1 1 1 0 1 1 1 1 1 0 0 0 1 0 0 0], shape=(32,), dtype=int64)
```

```
X_batch.shape, y_batch.shape
```

```
(TensorShape([32, 60]), TensorShape([32]))
```

# create the model and train it

Embedding: convert word IDs into embeddings

```
embed_size = 128
model = keras.models.Sequential([
    keras.layers.Embedding(vocab_size + num_oov_buckets, embed_size,
                           mask_zero=True, # not shown in the book
                           input_shape=[None]),
    keras.layers.GRU(128, return_sequences=True),
    keras.layers.GRU(128),
    keras.layers.Dense(1, activation="sigmoid")
])
```

the inputs of the model will be 2D tensors of shape [*batch size*, *time steps*],
the output of the Embedding layer will be a 3D tensor of shape [*batch size*, *time steps*, *embedding size*].

# compile and fit

```
model.compile(loss="binary_crossentropy", optimizer="adam",
              metrics=["accuracy"])
history = model.fit(train_set, steps_per_epoch=train_size // 32, epochs=5)
```

```
Epoch 1/5
781/781 [==============================] - 21s 18ms/step - loss: 0.5305 - accuracy: 0.7281
Epoch 2/5
781/781 [==============================] - 14s 18ms/step - loss: 0.3459 - accuracy: 0.8554
Epoch 3/5
781/781 [==============================] - 13s 17ms/step - loss: 0.1913 - accuracy: 0.9319
Epoch 4/5
781/781 [==============================] - 14s 17ms/step - loss: 0.1341 - accuracy: 0.9536
Epoch 5/5
781/781 [==============================] - 13s 17ms/step - loss: 0.1010 - accuracy: 0.9624
```

KISTI SINCE 1962

# 2022 TRUST KISTI

Korea Institute of Science and Technology Information