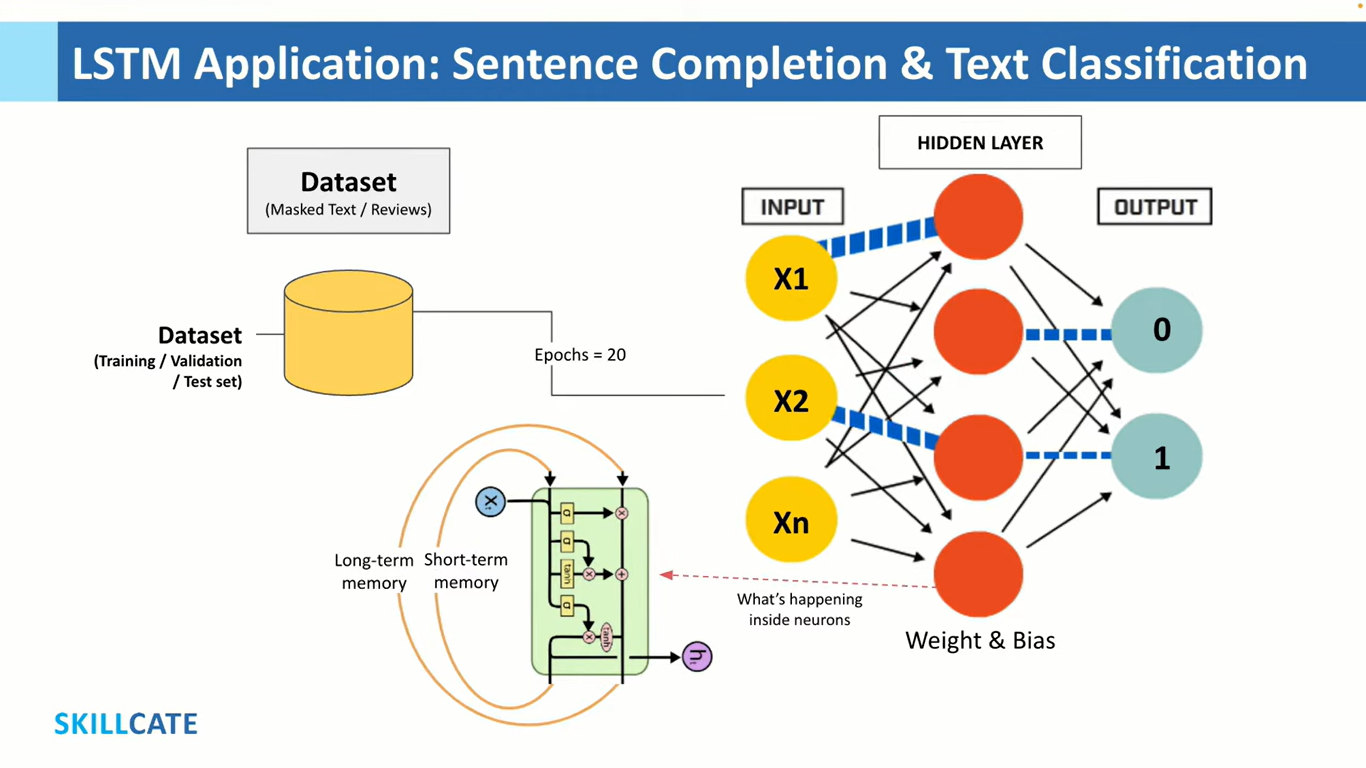
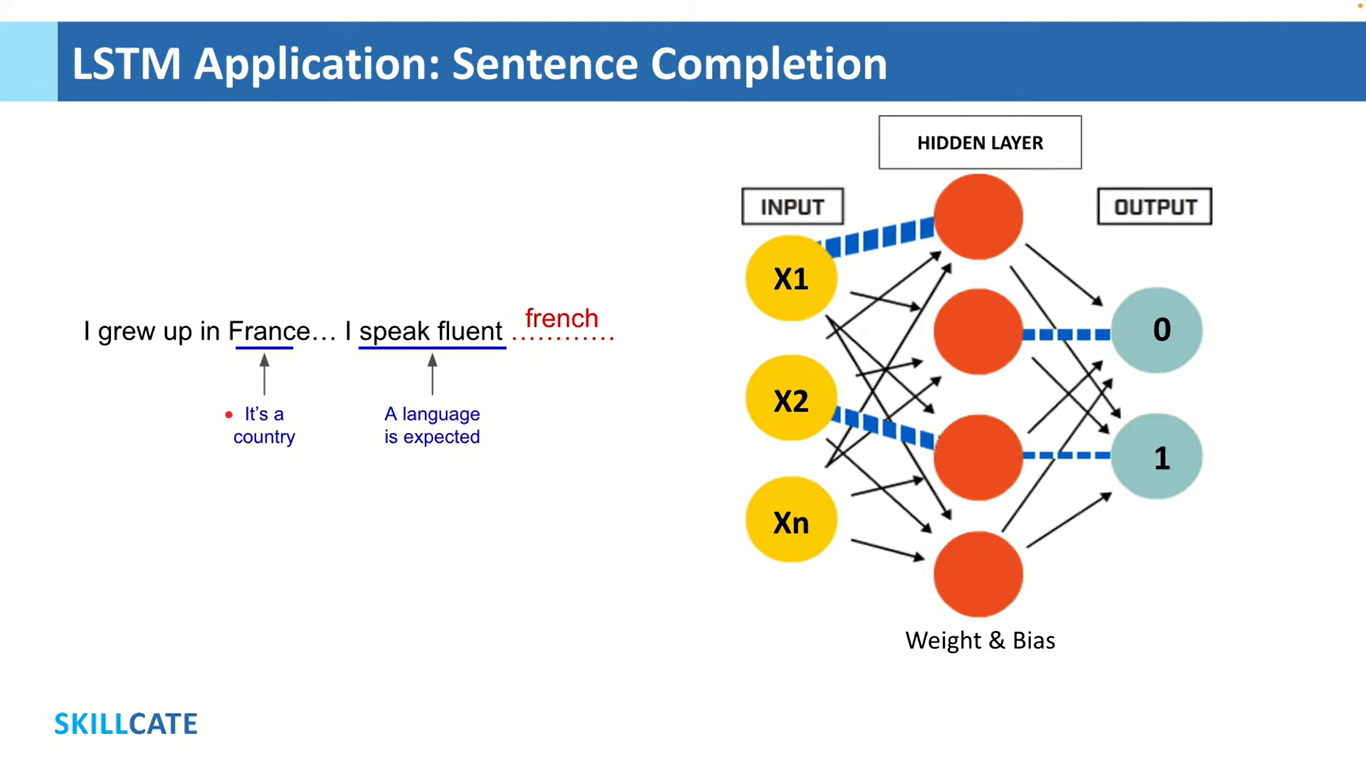
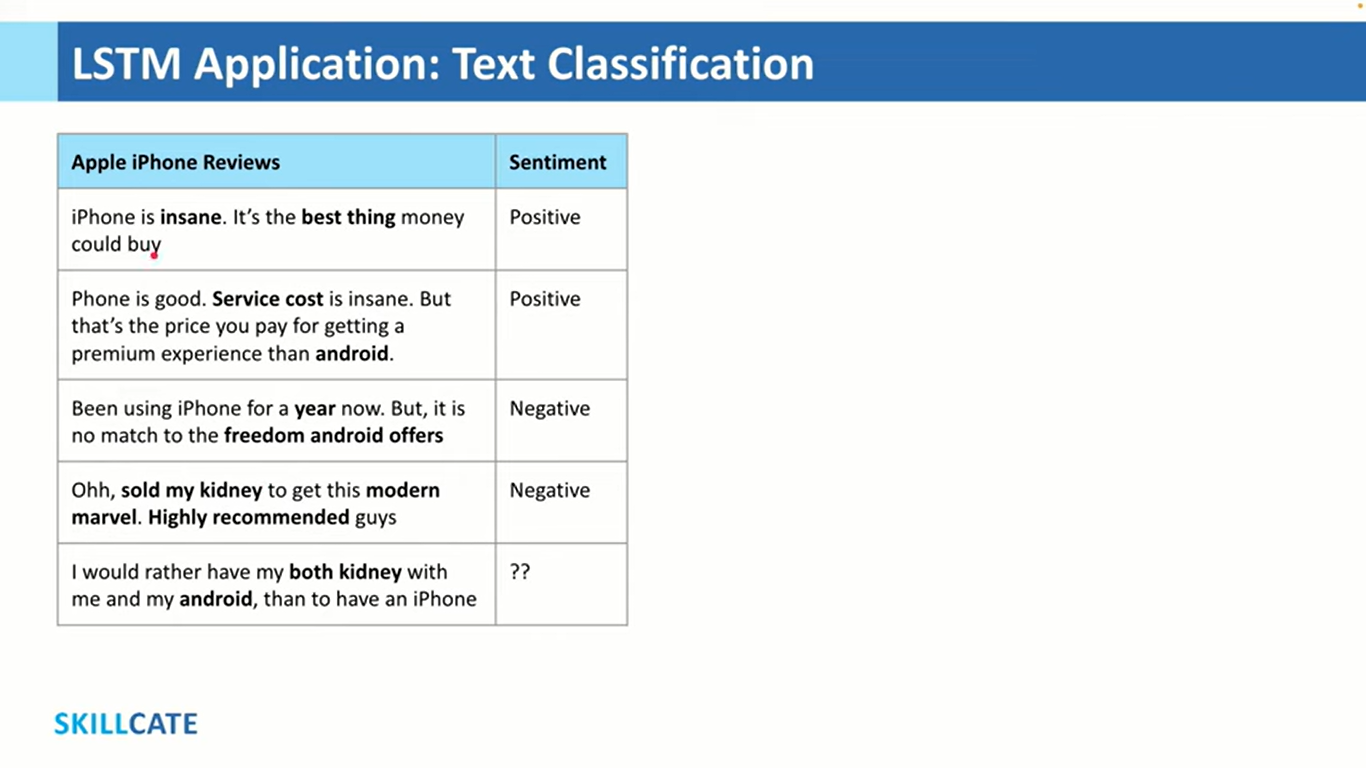


 **Cₜ** is like your brain’s memory — things you store and recall later.

 **hₜ** is like your mouth — what you say out loud at the moment.







**PART 1: SimpleRNN – Every Line Explained**

**STEP 1: Import Required Libraries**

import numpy as np

import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Embedding, SimpleRNN, Dense

**What this does:**

* numpy: Used to store and process sequences as numerical arrays.
* tensorflow: The main machine learning library used.
* Sequential: A Keras model type that stacks layers one after another.
* Embedding: Converts words (as numbers) into vector representations.
* SimpleRNN: Basic recurrent layer that processes sequences.
* Dense: Fully connected layer for output.

**STEP 2: Prepare the Data**

word\_to\_index = {

'i':1, 'love':2, 'hate':3, 'ai':4,

'bugs':5, 'is':6, 'amazing':7,

'code':8, 'hard':9

}

**Explanation:**

* We define a dictionary mapping words to unique integers (IDs).
* This is how text is **tokenized** into numerical form.

X = np.array([

[1, 2, 4], # i love ai

[1, 3, 5], # i hate bugs

[4, 6, 7], # ai is amazing

[8, 6, 9] # code is hard

])

**Explanation:**

* Each sub-list is a sentence represented by its token IDs.
* These are **fixed-length sequences** (3 words per sentence).

y = np.array([1, 0, 1, 0])

**Explanation:**

* These are the labels:
  + 1 = Positive sentiment
  + 0 = Negative sentiment

**STEP 3: Define the Model**

model = Sequential([

Embedding(input\_dim=10, output\_dim=4, input\_length=3),

SimpleRNN(4),

Dense(1, activation='sigmoid')

])

**Line-by-Line:**

* Sequential([]): A linear stack of layers.
* Embedding(input\_dim=10, output\_dim=4, input\_length=3):
  + input\_dim=10: We have 10 words in vocabulary.
  + output\_dim=4: Each word is mapped to a 4D vector.
  + input\_length=3: Each sentence has 3 words.
* SimpleRNN(4):
  + RNN layer with 4 hidden units.
  + Processes sequences one word at a time, keeps memory.
* Dense(1, activation='sigmoid'):
  + Fully connected output layer with sigmoid:
    - Outputs probability from 0 to 1
    - Used for binary classification

**🛠️ STEP 4: Compile the Model**

model.compile(

optimizer='adam',

loss='binary\_crossentropy',

metrics=['accuracy']

)

**🔍 Explanation:**

* optimizer='adam': Adaptive learning rate optimizer; widely used.
* loss='binary\_crossentropy': Ideal for 0/1 classification.
* metrics=['accuracy']: We want to track how often the model is correct.

**📈 STEP 5: Train the Model**

model.fit(X, y, epochs=50, verbose=0)

**🔍 Explanation:**

* X, y: Our input and output data.
* epochs=50: Model will loop 50 times over all data.
* verbose=0: Suppresses training output for cleaner display.

**🔮 STEP 6: Make Predictions**

print("Predictions:", model.predict(X).round())

**🔍 Explanation:**

* model.predict(X): Returns predicted probabilities (e.g., 0.91).
* .round(): Rounds them to 0 or 1.

**Expected Output:**

lua

Predictions: [[1.], [0.], [1.], [0.]]

**🔐 PART 2: LSTM – Same Example, More Power**

from tensorflow.keras.layers import LSTM

model = Sequential([

Embedding(10, 4, input\_length=3),

LSTM(4),

Dense(1, activation='sigmoid')

])

**🔍 What changed:**

* We imported and used LSTM(4) instead of SimpleRNN(4).
* LSTM has:
  + Internal **cell state** (C\_t) for long-term memory
  + **Forget**, **Input**, and **Output** gates for better control
* Useful when sequences are longer or more complex.

**Remaining Steps:**

model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])

model.fit(X, y, epochs=50, verbose=0)

print("LSTM Predictions:", model.predict(X).round())

✅ These are exactly the same as in the RNN section — because Keras allows you to swap layers easily.

**RNN**

Word 1 → Word 2 → Word 3 → [SimpleRNN] → Prediction

**LSTM:**

Word 1 → Word 2 → Word 3 → [LSTM: uses memory] → Prediction

