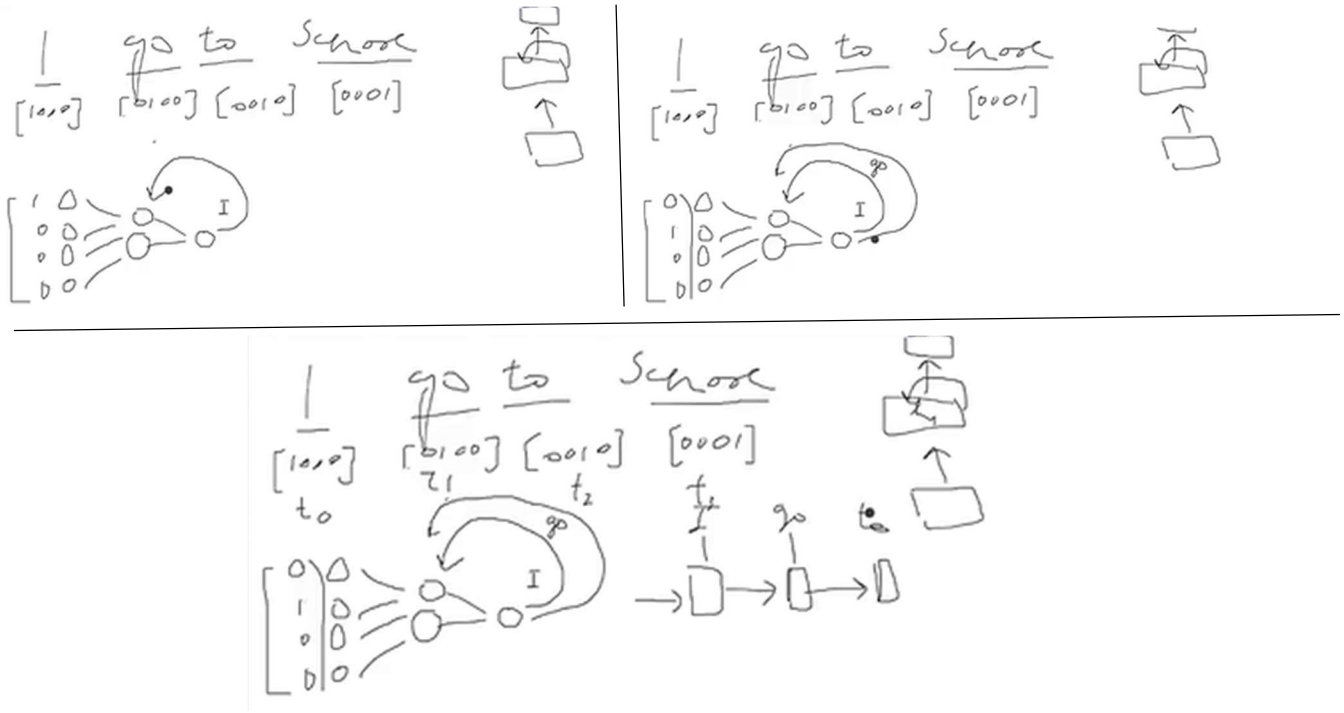
Recurrent Neural Network RNN

Recurrent Neural Networks (RNNs) are a type of artificial neural network designed to process sequences of data.

Example:

I go to school – total tokens = 4 #Tokens connect with different neurons and give output

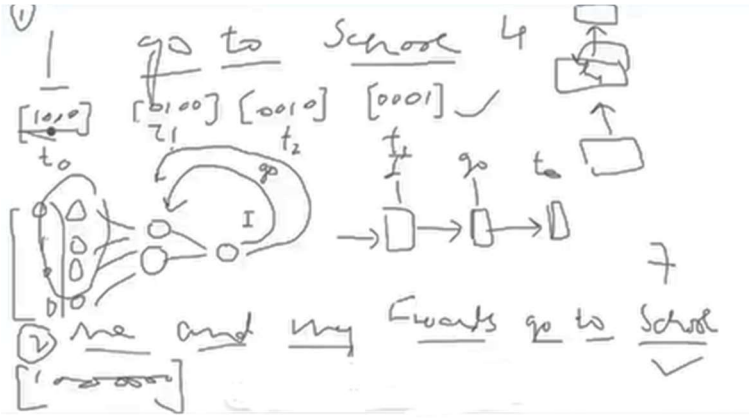
I [10000] + go [0100] + to [0010] + school [0001] #here 1 shows position of word / text.



Example:

1. I go to school – total tokens = 4
2. me and my friends go to school – total token = 7

Padding apply to same the size of vector for both. So vector size need to be updated.



Note:

- RNN cannot memorize long sequences
- It has long short term memory

Types of Recurrent Neural Networks

There are four types of Recurrent Neural Networks:

- **One to One:** Standard neural networks where each input corresponds to a single output (e.g., image classification).
- **One to Many:** Single input generates a sequence of outputs (e.g., image captioning).
- **Many to One:** Sequence of inputs leads to a single output (e.g., sentiment analysis).
- **Many to Many:** Sequence of inputs is mapped to a sequence of outputs (e.g., machine translation).

Problem with RNN:

- Long Term Dependency – vanishing gradient problem
- Unstable training – exploding gradient problem

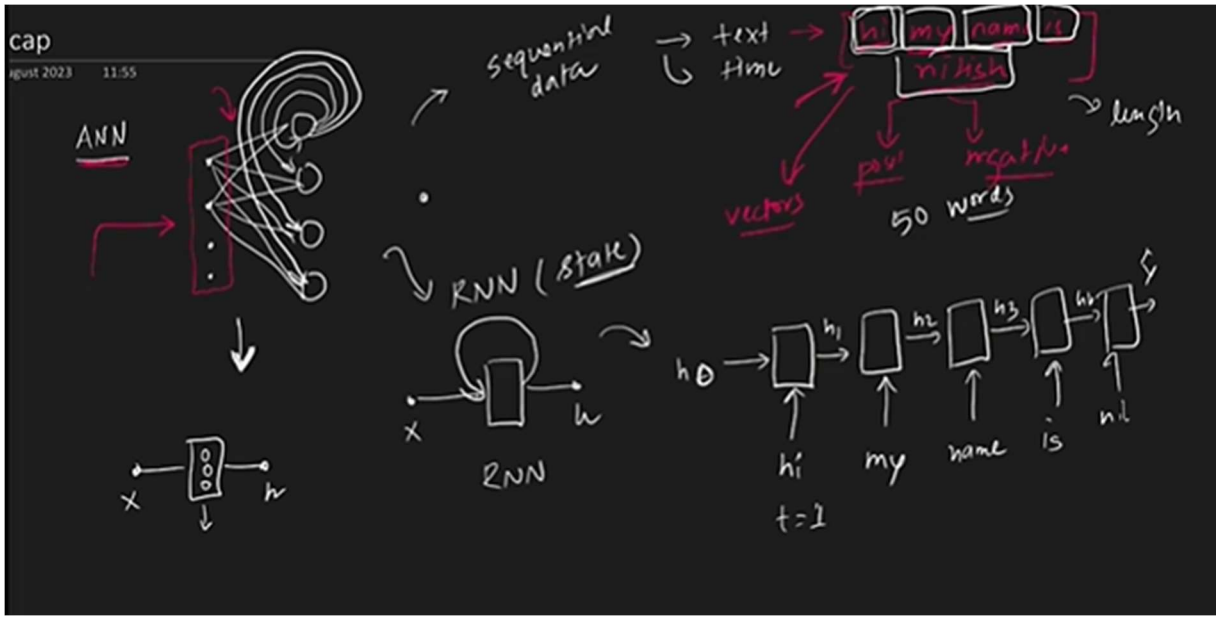
These are the two problems with RNN due to which we go for LSTM

Vanishing Gradient: When gradients become very small during backpropagation, making it difficult for the network to update its weights, especially in deeper layers.

Exploding Gradient: When gradients become excessively large, causing unstable updates and making the model's training diverge.

RNN cannot able to link new words with old words because it forgets previous data while learning new data.

This limitation is addressed by more advanced architectures like LSTMs (Long Short-Term Memory) and GRUs (Gated Recurrent Units), which are designed to better retain and utilize past information over long sequences.



Long Short Term Memory (LSTM)

LSTM (Long Short-Term Memory) is a type of Recurrent Neural Network (RNN) architecture designed to overcome the limitations of traditional RNNs, particularly the issue of vanishing gradients. LSTMs are capable of learning long-term dependencies in sequential data because they can maintain information over extended time periods.

An LSTM cell contains three key gates:

- **Forget Gate:** Decides what information to discard from the cell state.
- **Input Gate:** Decides which new information to store in the cell state.
- **Output Gate:** Controls the output based on the cell state and hidden state.

These gates help the LSTM to selectively remember or forget information, making it effective for tasks like time series prediction, natural language processing, and more, where maintaining context over long sequences is crucial.

Ways to represent input data

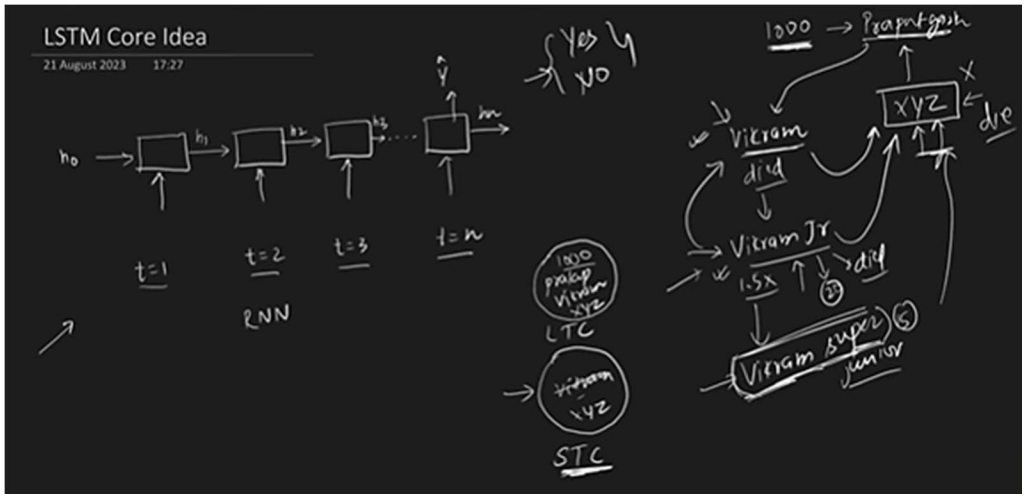
Different ways to represent input data, especially when dealing with categorical data like words in a sequence are as follows:

One-Hot Vector: A representation of categorical data where each class is encoded as a binary vector with one "1" and the rest "0"s. This can be used as input to an LSTM, but it's not very efficient for large vocabularies.

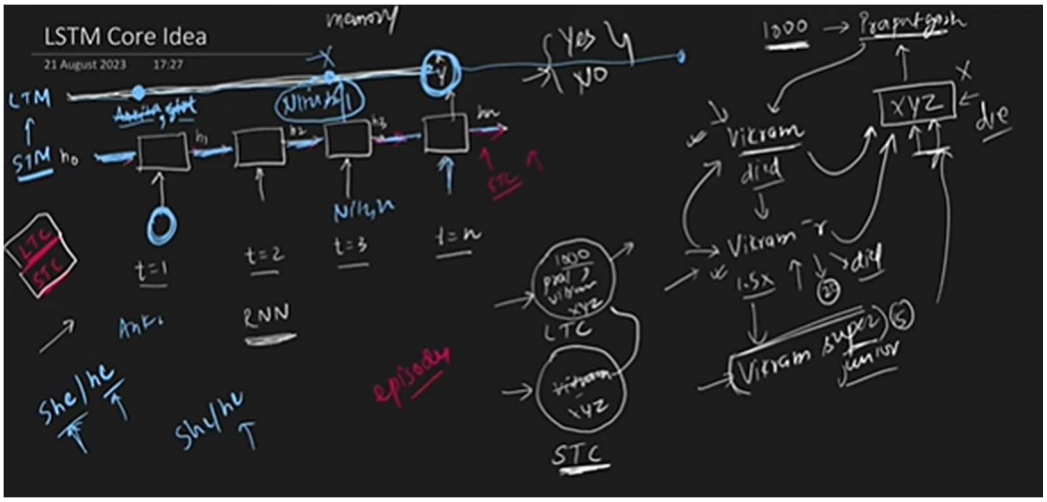
Embedding: A dense, learned representation of categorical data, where words (or categories) are mapped to continuous vector spaces, capturing semantic relationships. This is commonly used with LSTMs for tasks like language modeling.

Note: Embeddings are preferred over one-hot vectors in LSTM models for handling large vocabularies because they provide more compact and meaningful representations.

LSTM Core Idea

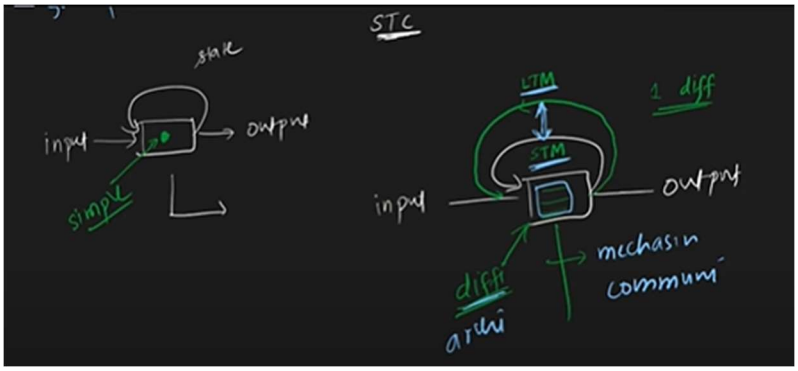


LSTM's Key Feature: It uses a memory cell to store information, allowing it to "remember" for longer periods compared to standard RNNs.



Internal model become short term memory and external model become short term memory.

LSTM vs. RNN



LSTM vs. RNN Differences

Core Concept:

RNN (Recurrent Neural Network): Designed for sequential data, RNNs pass information from one step to the next using hidden states. They struggle with long-term dependencies due to the vanishing gradient problem.

LSTM (Long Short-Term Memory): A special type of RNN that uses memory cells and gates to preserve information over long sequences, overcoming the vanishing gradient issue.

Communication Differences:

RNN: At each time step, the network processes the input and passes the hidden state to the next time step. The hidden state is the only carrier of information between steps, making it difficult to retain relevant information from distant steps.

LSTM: In addition to the hidden state, LSTMs introduce a cell state (memory) that can pass information unaltered across multiple time steps. The gates (forget, input, output) control how much information is added or removed from the cell state, making it easier to handle long-term dependencies.

Architecture Differences:

RNN:

- Input: Receives the input and previous hidden state.
- Hidden State: Updates based on the current input and hidden state.
- Output: The hidden state serves as both the internal memory and the output.
- Simple design, but information tends to fade over time (vanishing gradient).

LSTM:

- Input: Receives the input, previous hidden state, and cell state.
- Cell State: A conveyor belt-like memory that can carry information over long periods.
- Forget Gate: Decides what to discard from the cell state.
- Input Gate: Decides what new information to add to the cell state.
- Output Gate: Decides what to output and passes part of the memory forward.
- More complex, but able to remember both short and long-term information.

Handling Long-Term Dependencies:

RNN: Poor at handling long-term dependencies due to the vanishing gradient problem.

LSTM: Designed to handle long-term dependencies efficiently by regulating the flow of information through gates and a memory cell.

Training Challenges:

RNN: Prone to the vanishing/exploding gradient problem, making it difficult to train on long sequences.

LSTM: Alleviates the vanishing gradient problem through its gating mechanism, making it more stable during training on long sequences.

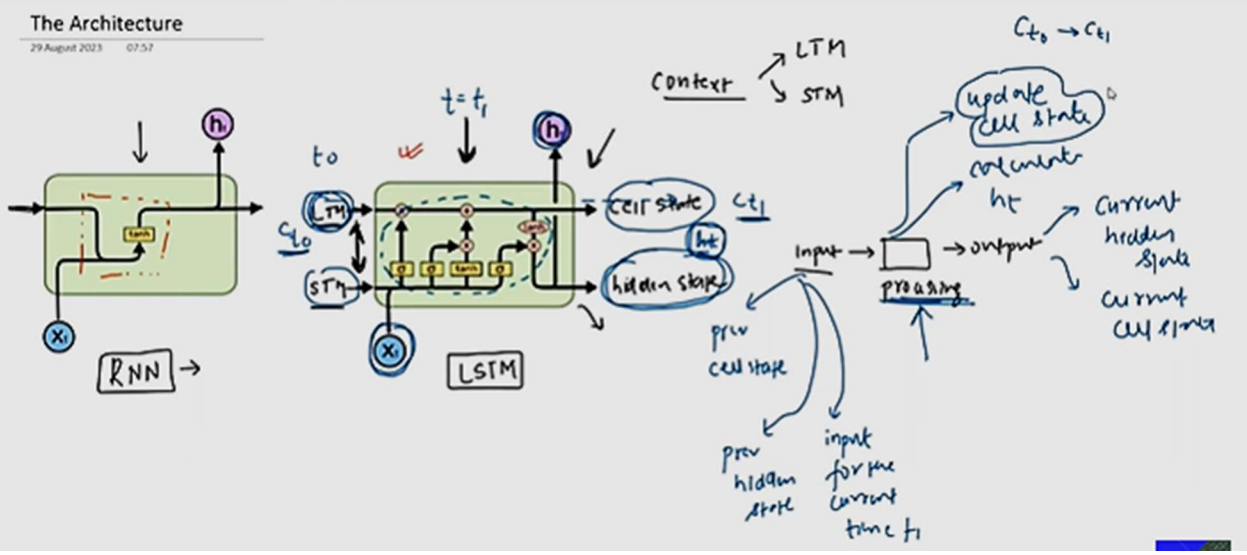
Use Cases:

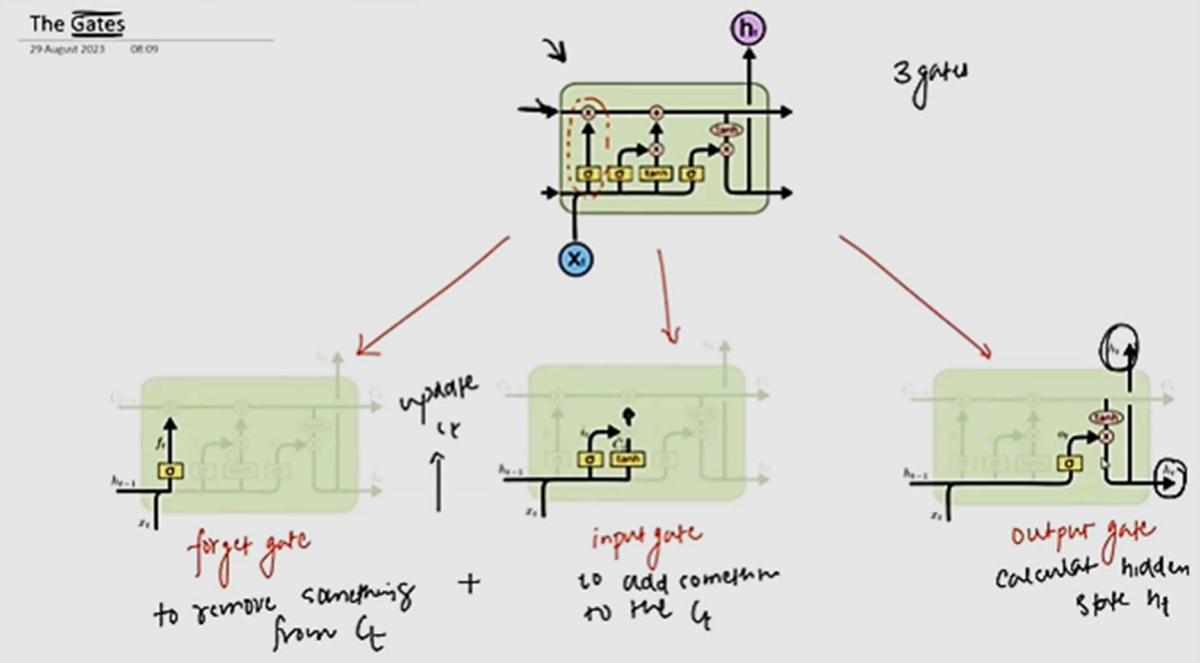
RNN: Works well for short sequences (e.g., character-level text generation).

LSTM: Ideal for tasks involving long sequences or where important information might be far back in the sequence, such as in language translation, speech recognition, and time series prediction.

Summary of Differences:

Feature	RNN	LSTM
Memory Mechanism	Hidden state only	Hidden state + Cell state
Long-Term Dependency	Struggles (due to vanishing gradients)	Handles well (due to gating)
Architecture Simplicity	Simple	More complex (uses gates)
Gates	None	Forget, Input, Output
Handling Gradients	Prone to vanishing/exploding	Mitigates vanishing gradient issue
Common Use Cases	Short sequences	Long sequences, speech, language modeling



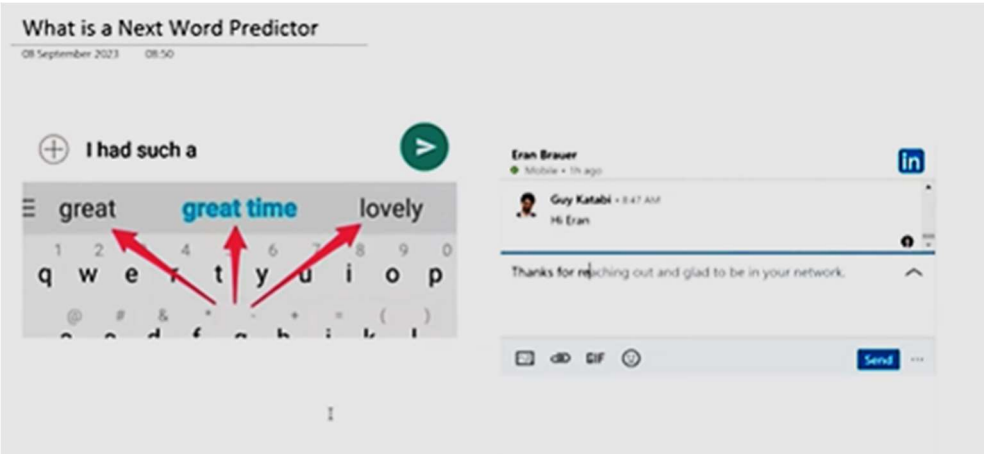


Next Word Predictor

A **Next Word** Predictor is a type of model or application that predicts the next word in a sequence of words based on the preceding words. It is widely used in applications like text auto-completion, keyboards, and voice assistants.

Applications:

- Text Auto-Completion: Predicting the next word as the user types (e.g., in smartphone keyboards or search engines).
- Smart Reply: Suggesting responses in messaging apps.
- Language Translation: Helping predict the next word in translation tasks.
- Virtual Assistants: Predicting what a user might say next to provide relevant suggestions.



Sentiment analysis:

Sentiment analysis using RNNs in deep learning involves classifying text data into different sentiment categories, such as positive, negative, or neutral.

Example: Text check on an online model (https://text2data.com/Demo)

Have stawhoever wrote the screenplay for this movie obviously never consulted any books about Lucille Ball, especially her autobiography. I've never seen so many mistakes in a biopic, ranging from her early years in Celoron and Jamestown to her later years with Desi. I could write a whole list of factual errors, but it would go on for pages. In all, I believe that Lucille Ball is one of those inimitable people who simply cannot be portrayed by anyone other than themselves. If I were lucie Arnaz and Desi, Jr., I would be irate at how many mistakes were made in this film. The filmmakers tried hard, but the movie seems awfully sloppy to me.yed here off and on over several years. great location - access to trains, underground and buses plus good walking distance to Westminster. I However, as a single traveller end up paying for small, narrow room. Room was adequate, bathroom with shower stall very cramped. A bit disappointed this time, but since I really like the location I will go back. Did not have much interaction with staff.


```
# Import SentimentIntensityAnalyzer and create an sid object

from nltk.sentiment.vader import SentimentIntensityAnalyzer

sid = SentimentIntensityAnalyzer()
```

```
# Write a review as one continuous string (multiple sentences are ok)
review = 'The film is a worthwhile watch. On a scale from zero to five, I give this film a five. The film literally brought
```

```
# Obtain the sid scores for your review
sid.polarity_scores(review)
```

```
{'neg': 0.062, 'neu': 0.779, 'pos': 0.159, 'compound': 0.4588}
```

CHALLENGE: Write a function that takes in a review and returns a score of "Positive", "Negative" or "Neutral"

```
def review_rating(string):
    scores = sid.polarity_scores(string)
    if scores['compound'] == 0:
        return 'Neutral'
    elif scores['compound'] > 0:
        return 'Positive'
    else:
        return 'Negative'
```

```
# Test the function on your review above:
review_rating(review)
```

```
'Positive'
```