BASICS OF NLP

Sachin Shivakalimath
sachin.smath@gmail.com
9449764697

Tokenization

- Tokenization is the process of breaking down text into smaller units, or "tokens," which can then be processed by machine learning or natural language processing (NLP) models.
- There are various levels of tokenization, depending on how granular you want the analysis to be, including character-level, word-level, sentence-level, and paragraph-level tokenization.
  - Character-Level Tokenization
  - Word-Level Tokenization
  - Sentence-Level Tokenization
  - Paragraph-Level Tokenization
  - Subword Tokenization
  - Morphological Tokenization
  - N-Gram Tokenization
  - Syllable-Level Tokenization
  - Entity-Level Tokenization
  - Regex-Based Tokenization

Character-Level Tokenization

Definition: Each individual character in the text becomes a token.

Example:
For the word "Hello", character-level tokens would be ['H', 'e', 'l', 'l', 'o’].

Use Cases: Useful for languages with complex character structures, such as Chinese or Arabic, where characters carry semantic meaning. It’s also helpful in text generation, spelling correction, and applications where minor textual variations (e.g., typos) need to be considered.

Pros: Captures details at the finest level, making it adaptable to handling unknown words or spelling variations.

Cons: Results in a large number of tokens, which can increase computational complexity and lead to issues like long sequence lengths in deep learning models.

Word-Level Tokenization

Definition: The text is split into tokens based on words. Typically, words are separated by whitespace or punctuation.

Example:
For the sentence "The quick brown fox," word-level tokens would be ['The', 'quick', 'brown', 'fox’].

Use Cases: Commonly used in NLP tasks like sentiment analysis, machine translation, and text classification. Most word embeddings (like Word2Vec and GloVe) are designed for word-level tokens.

Pros: Captures semantic meaning more effectively than character-level tokenization, making it efficient and interpretable.

Cons: Cannot handle out-of-vocabulary (OOV) words, misspellings, or variations effectively. Models trained on word-level tokens may struggle with unseen words in new text.

Sentence-Level Tokenization

Definition: The text is split into individual sentences, which are treated as tokens.

Example:
For the text "Hello world! How are you?", sentence-level tokens would be ['Hello world!', 'How are you?’].

Use Cases: Useful for tasks requiring context within a sentence, such as summarization, sentence classification, or text segmentation. Sentence-level tokenization can help retain contextual information that spans multiple words.

Pros: Preserves sentence structure, which is beneficial for tasks requiring an understanding of complete thoughts or ideas.

Cons: Sentence boundaries can be challenging to identify in certain languages or informal writing styles. Additionally, it may overlook finer details within each sentence.

Paragraph-Level Tokenization

Definition: The text is divided into paragraphs, often separated by newlines or indentation.

Example:
For a text with two paragraphs, "Hello world! This is the first paragraph.\nAnd here’s the second paragraph," the tokens would be:
['Hello world! This is the first paragraph.', 'And here’s the second paragraph.’]

Use Cases: Used in applications like document summarization, topic modeling, and legal or medical text analysis, where entire paragraphs may hold unique, contextually complete thoughts.

Pros: Retains the highest level of context, making it suitable for analyses that rely on larger portions of text or where paragraph structure has meaning.

Cons: This approach is less granular, so it may overlook important details within sentences or words. Additionally, it’s less suitable for tasks that require detailed text analysis at a finer level.

N-Gram Tokenization

- **Definition**: Breaks text into continuous sequences of n words or characters, known as n-grams. Common values are bigrams (2 words), trigrams (3 words), etc.

- **Example**:
- For the sentence "I love NLP," bigram tokenization produces ['I love', 'love NLP'].

- **Use Cases**: Useful in text classification, sentiment analysis, and language modeling, as it captures context within neighboring tokens.

- **Pros**: Captures local context and word associations, making it effective for analyzing phrases or short sequences.

- **Cons**: Large n-grams can lead to very large feature spaces, which increases computational cost.

Regex-Based Tokenization

•**Definition**: Uses regular expressions to define custom patterns for tokenization, often for domain-specific applications.

•**Example**:
•Splitting text based on dates or URLs, like turning "See more at https://example.com on 12/12/2023" into ['https://example.com', '12/12/2023'].

•**Use Cases**: Useful in text preprocessing for cleaning or splitting based on specific patterns.

•**Pros**: Highly customizable for specific tokenization needs, especially in preprocessing.

•**Cons**: Requires expertise in regular expressions and can be error-prone if patterns are not defined carefully.

Vectorization

Vectorization

- Vectorization is the process of converting textual data into numerical vectors so that machine learning models can process and learn from it. Different vectorization methods are used depending on the complexity, context requirements, and applications.
  1. Count Vectorization (Term Frequency)
  2. Term Frequency-Inverse Document Frequency (TF-IDF)
  3. Word Embeddings (e.g., Word2Vec, GloVe)
  4. FastText
  5. Contextualized Embeddings (ELMo, BERT)
  6. Transformer-Based Models and Beyond (GPT, T5)
  7. Sentence Embeddings (Sentence-BERT, Universal Sentence Encoder)

Method	Level	Contextual?	Handles Polysemy	Common Use Cases
Common Use Cases	Word-level	No	No	Simple text classification
	Word-level	No	No	Text classification, relevance ranking
Word2Vec / GloVe	Word embeddings	No	No	Sentiment analysis, word similarity
FastText	Subword embeddings	No	Partial	Text classification, languages with complex morphology
ELMo	Contextual word embeddings	Yes	Yes	Named entity recognition, part-of-speech tagging
BERT	Contextual word embeddings	Yes	Yes	QA systems, language inference, translation
Sentence-BERT / USE	Sentence embeddings	Yes	Yes	Semantic similarity, document clustering

Count Vectorization (Term Frequency)

- **Count Vectorization** converts text to vectors by counting the occurrences of each word in a document. Each unique word becomes a feature, and its value is the number of times it appears in the document.
- **Example**: For two documents, "I love NLP" and "NLP is great," the count vector might look like:

I	love	NLP	is	great
1	1	1	0	0
0	0	1	1	1

- **Limitations**: High dimensionality and inability to capture semantic relationships. It only tells you if a word appears and how often, not its importance.

TF-IDF

- **TF-IDF** improves on raw counts by weighing words according to their importance. It combines two metrics:
- **Term Frequency (TF)**: How frequently a word appears in a document.
- **Inverse Document Frequency (IDF)**: The inverse frequency of the word across all documents, reducing the weight of commonly occurring words (e.g., "the," "and").
- **Advantages**:
  - Amplifies rare words, panelizes frequent words
  - Highlights important words and reduces the influence of commonly used words.
- **Limitations**: Still treats each word independently, losing the context of word sequences.

Formula:

$$\text{TF-IDF} = \text{TF} \times \text{IDF}$$

where:

$$\text{IDF}(t) = \log\left(\frac{\text{Total Number of Documents}}{\text{Number of Documents Containing } t}\right)$$

Word2Vec / GloVe

- Word embeddings capture semantic relationships by mapping words into a continuous vector space. **Word2Vec** and **GloVe** are two popular word embedding techniques that use context to understand word relationships.
- **Word2Vec**: Uses neural networks to learn word representations based on neighboring words. It has two approaches:
  - **CBOW (Continuous Bag of Words)**: Predicts a word given its surrounding words.
  - **Skip-gram**: Predicts the surrounding words given a specific word.
- Word2Vec learns word relationships like **king - man + woman ≈ queen**.
- **GloVe (Global Vectors for Word Representation)**: Learns embeddings by factorizing word co-occurrence matrices, capturing both global and local contexts.
- **Advantages**: Captures semantic and syntactic relationships, e.g., similar words are closer in the vector space.
- **Limitations**: Fixed embeddings, so it can't handle polysemy (words with multiple meanings).

FastText

- **FastText**, developed by Facebook, is an extension of Word2Vec that breaks words into character n-grams and then generates word embeddings by aggregating these n-grams. This allows **subword information** to be included, which is particularly useful for morphologically rich languages and for handling **out-of-vocabulary words**.
- **Example**: The word "playing" might be broken down into subword tokens like "pla," "lay," "ayi," etc., and then represented as the sum of these subword embeddings.
- **Advantages**: Improves on Word2Vec by handling rare words and morphologically complex languages better.

Contextualized Embeddings (ELMo, BERT)

- Contextualized embeddings consider the surrounding words dynamically, providing different embeddings for the same word based on its context. These embeddings are generated by large language models (LLMs).
- **ELMo (Embeddings from Language Models)**:
  - ELMo embeddings are generated using a two-layer bi-directional LSTM trained on a language modeling task.
  - Each word's embedding is generated based on the entire sentence, allowing different embeddings for the same word depending on its context.
  - **Example**: The word "bank" will have different embeddings in "river bank" and "money bank."
- **BERT (Bidirectional Encoder Representations from Transformers)**:
  - BERT is based on the **Transformer architecture**, using self-attention to capture context from both left and right sides of each word.
  - BERT embeddings are **contextualized** at a deeper level, as it's trained on a **masked language model** objective, where some words are randomly masked and predicted from the context.
  - **Advantage**: BERT embeddings capture deeper context and handle polysemy better than ELMo and Word2Vec.
  - **Limitation**: BERT embeddings are computationally expensive and require fine-tuning for specific tasks.

Transformer-Based Models and Beyond (GPT, T5)

- Advanced transformer-based models like **GPT** and **T5** generate embeddings for both words and entire sentences by understanding context bidirectionally (BERT-style) or autoregressively (GPT-style). These models create embeddings not only for individual words but also for sentences and even documents.
- **GPT (Generative Pre-trained Transformer)**:
  - GPT is an autoregressive model, generating text by predicting the next word based on prior context.
  - **Use case**: Mainly used for generating coherent and contextually relevant text.
- **T5 (Text-To-Text Transfer Transformer)**:
  - T5 is designed as a text-to-text framework, meaning that it reformulates NLP tasks as text generation.
  - This allows T5 to learn generalized embeddings across tasks like summarization, translation, and question answering.

Sentence Embeddings (Sentence-BERT, Universal Sentence Encoder)

- **Sentence embeddings** provide fixed-size vector representations for entire sentences rather than individual words. These embeddings are useful for sentence-level tasks such as semantic similarity and text classification.
- **Sentence-BERT**: A variant of BERT fine-tuned on sentence pairs to create embeddings that capture semantic similarity.
- **Universal Sentence Encoder**: Developed by Google, it generates sentence embeddings that are easy to use in various applications, especially in semantic search and clustering.

Vector Database

VectorDB

A vector database is a specialized type of database designed to handle high-dimensional data in the form of vectors. Vectors are mathematical representations of data points in a multidimensional space, commonly used in machine learning, natural language processing, and computer vision. In these applications, vector databases are essential for efficiently storing, retrieving, and searching for vectors, enabling tasks like similarity search and recommendation.

Key Concepts in Vector Databases

- **Vectors**:
  - Vectors represent features or embeddings derived from raw data like text, images, audio, or any structured or unstructured data. For example, words in a document can be converted into vectors using models like Word2Vec, BERT, or GPT. Similarly, images can be represented as vectors by passing them through a convolutional neural network (CNN).
  - Each vector is a list of numerical values representing the data in a high-dimensional space.
- **Similarity Search**:
  - One of the main applications of vector databases is similarity search, where the goal is to find vectors (data points) that are similar to a given query vector.
  - Similarity is often measured using metrics like cosine similarity, Euclidean distance, or Manhattan distance. Vector databases are optimized to perform these calculations quickly on large datasets.
- **Indexing**:
  - Vector databases use specialized indexing methods, such as Approximate Nearest Neighbor (ANN) techniques, to efficiently search through vast amounts of high-dimensional data.
  - Popular indexing algorithms include FAISS (Facebook AI Similarity Search), HNSW (Hierarchical Navigable Small World), and ANNOY (Approximate Nearest Neighbors Oh Yeah).
- **Scalability and Speed**:
  - Vector databases are built to handle large volumes of data with low latency. This makes them suitable for real-time applications, such as recommendation systems, chatbots, image search, and voice recognition.
  - Traditional relational databases are not efficient for high-dimensional vector data due to their inability to perform rapid vector similarity searches.

Why Use a Vector Database?

- **Efficient Similarity Search**:
  - Vector databases are optimized to retrieve the most similar data points (nearest neighbors) for a given vector, making them ideal for recommendation systems, search engines, and personalized content delivery.
- **Real-time Recommendations**:
  - E-commerce platforms, social media sites, and streaming services use vector databases to recommend items based on user behavior, preferences, and past interactions.
- **Handling Unstructured Data**:
  - Unstructured data like text, images, and audio can be transformed into vector representations, which vector databases can store and retrieve. This makes vector databases suitable for tasks involving natural language processing, image recognition, and more.
- **Flexibility with Machine Learning Models**:
  - With pre-trained models available for tasks like NLP and computer vision, developers can quickly generate embeddings for their data, store them in a vector database, and start performing similarity searches without additional model training.

Popular Vector Databases

- **Pinecone**:
  - A fully managed vector database that integrates with machine learning models and provides API-based access for similarity search.
- **Weaviate**:
  - An open-source, schema-based vector database with built-in support for various machine learning models, including text and image embeddings.
- **Milvus**:
  - Another open-source vector database, optimized for scalability, and frequently used in large-scale AI and data analytics projects.
- **FAISS (Facebook AI Similarity Search)**:
  - Not strictly a database, FAISS is a library for efficient similarity search and clustering of dense vectors, often integrated into vector databases.

Transfer Learning

Transfer learning

- Transfer learning is a machine learning technique where a model developed for a specific task is reused as the starting point for a model on a second, related task.
- Instead of training a model from scratch on a large dataset, transfer learning allows the use of a pre-trained model as a foundation, which is then fine-tuned for the specific requirements of a new task.

Benefits of transfer Learning

Reduced Training Time: By leveraging pre-trained models, you can significantly reduce the time required to train a new model from scratch. This is particularly beneficial for complex models like deep neural networks.

Improved Performance: Pre-trained models often achieve better performance than models trained from scratch, especially when dealing with limited data. The knowledge captured in the pre-trained model can help the new model generalize better to unseen data.

Lower Data Requirements: Transfer learning can be effective even with small datasets. The pre-trained model provides a strong foundation, allowing the new model to learn from a smaller amount of data.

Increased Accessibility: Transfer learning makes advanced machine learning techniques more accessible to researchers and developers with limited resources. By using pre-trained models, you can build complex models without the need for extensive data or computational power.

Enhanced Generalization: Pre-trained models often have a broader understanding of the underlying features and patterns in the data. This can help the new model generalize better to unseen data, leading to improved performance on new tasks.

Representation Learning

Representation Learning

- Representation learning, also known as feature learning, is a type of machine learning technique where the system automatically discovers the representations needed for a task.
- Instead of relying on manually engineered features, representation learning algorithms learn to transform raw data into the representations that make it easier to perform the desired task, such as classification, regression, clustering, etc.
  - **Basic Concepts**
    - **Raw Data**: The initial form of data that hasn't been transformed or processed, such as pixels in an image or words in a sentence.
    - **Features**: A feature is an individual measurable property or characteristic of a phenomenon being observed. In representation learning, the features are learned automatically from the data.

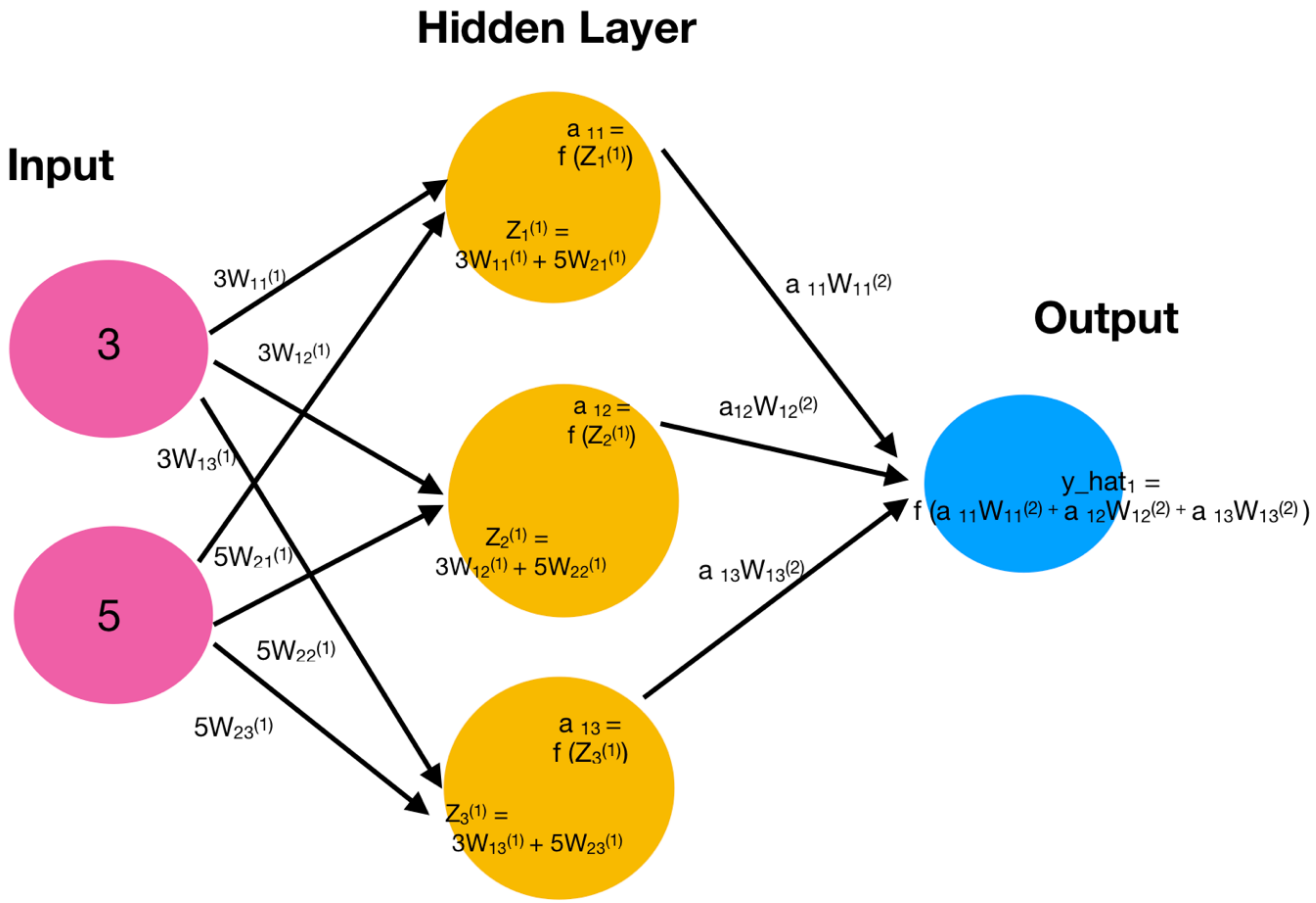
Latent Space

- Representation learning is a field within machine learning that focuses on learning efficient, meaningful, and often lower-dimensional representations of data, which can then be used for various tasks such as classification, clustering, or generation. These representations often reside in what we refer to as a "latent space."

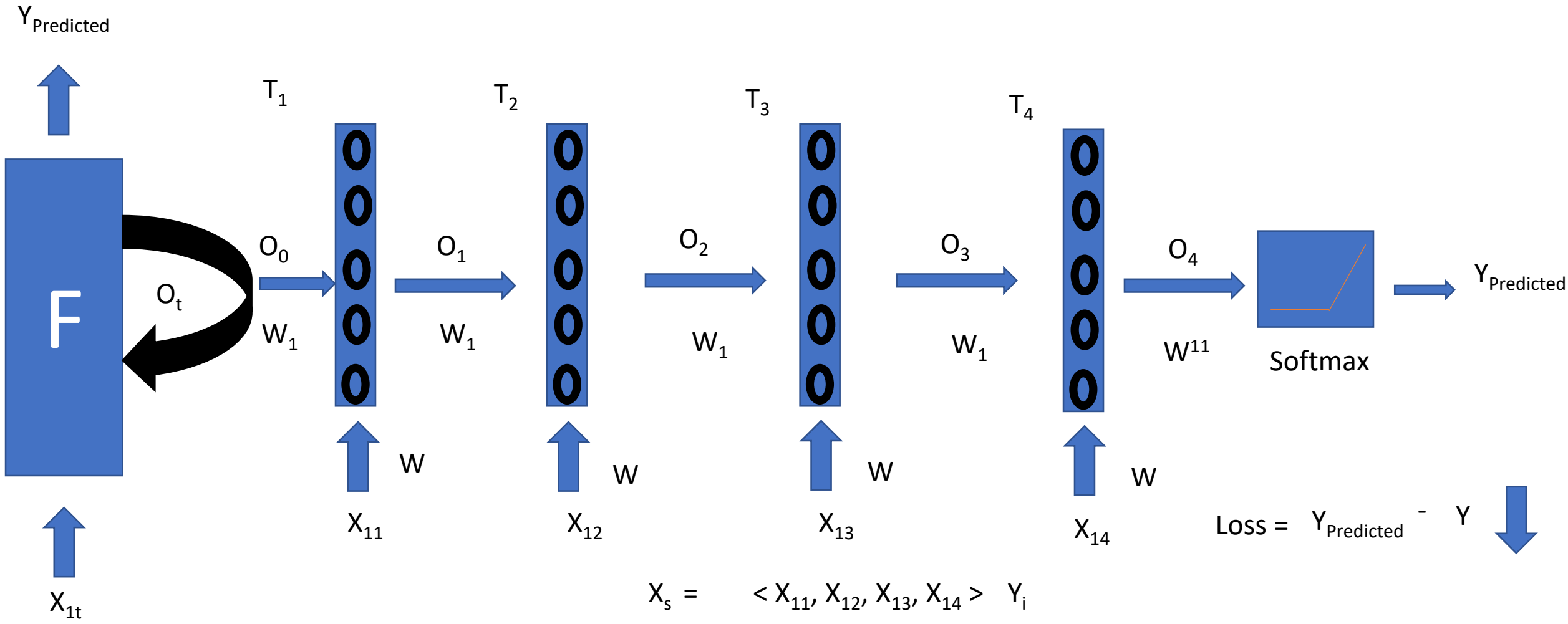
  - **Feature Space vs. Latent Space**
    - **Feature Space**: The feature space refers to the original space where the raw data is represented by its features or attributes. For instance, in image data, each pixel is a feature, so an image is represented by a vector of pixel values. This space can be very high-dimensional, especially for complex data like images, text, or audio.
    - **Latent Space**: Latent space, on the other hand, is the transformed version of this feature space, typically lower-dimensional, where the model encodes the essential information needed to describe the data. It captures the underlying patterns or structures in the data, often in a way that is more efficient or more interpretable than the original feature space.

RNN

NN with Hidden Layer



RNN architecture – Forward Network



Problem with this Simple RNN

- Suffers vanishing gradient and exploding gradient problem.
- Vanishing gradient problems when the weight updating is negligible
  - Sigmoid 0 to 1
  - Hyperbolic tangent(tanh) -1 to 1
  - ReLu 0 to infinity
  - Softmax 0 to 1
- Exploding gradient problem when weight updating is so large
  - ELU negative infinity to positive infinity
  - SELU negative infinity to positive infinity

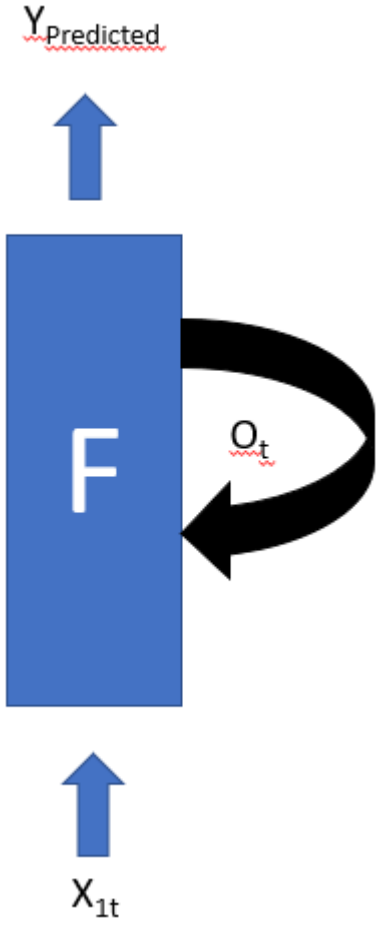
LSTM example

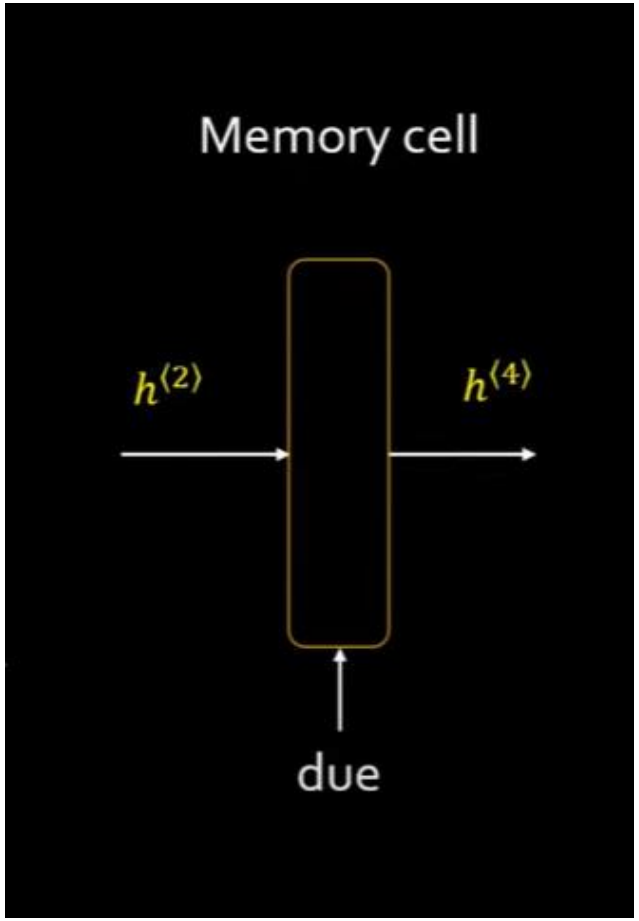
- Today, due to my current job and family condition, I need to take a loan
- Last year, due to my current job and family condition, I had to take a loan

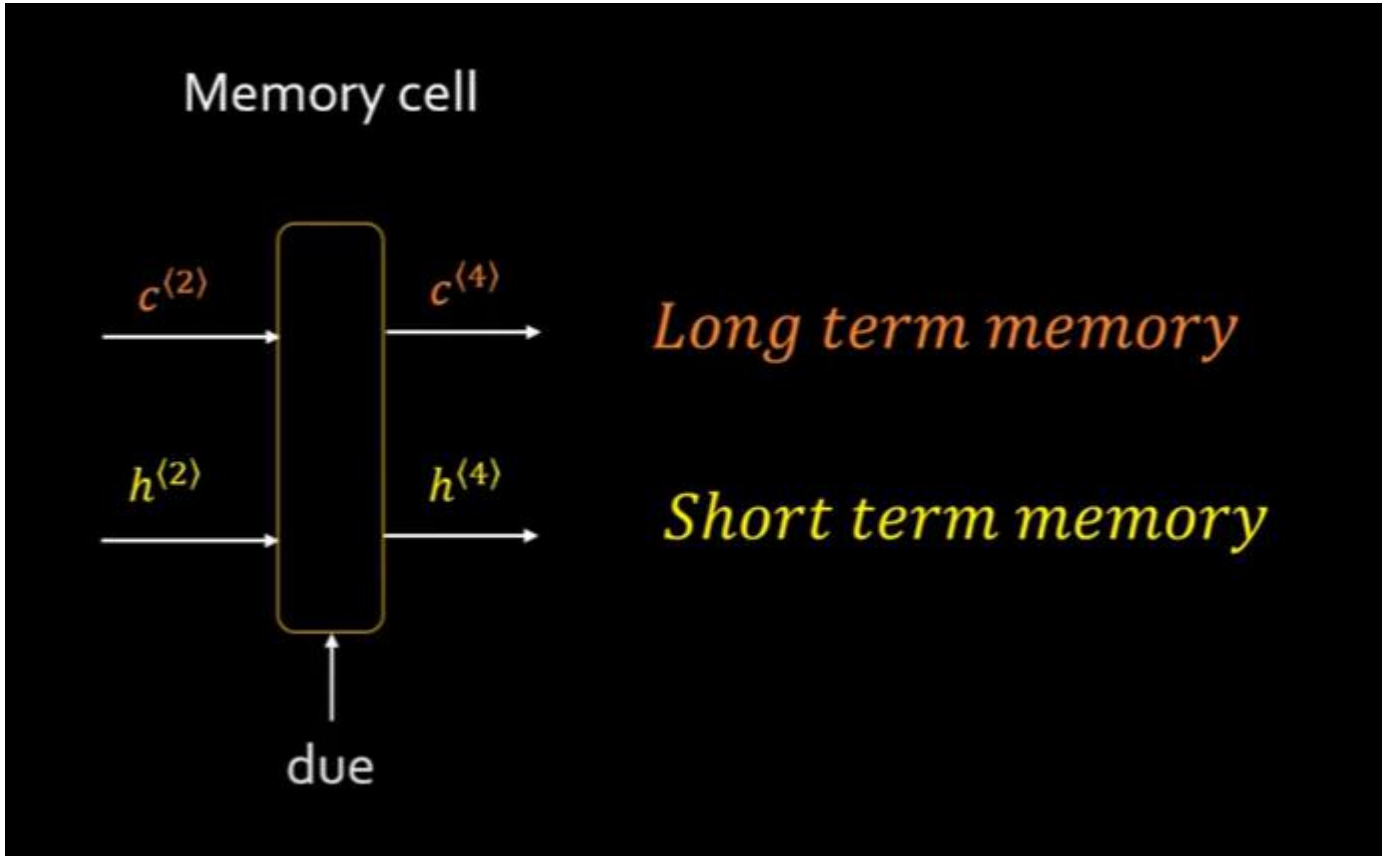
RNN - LSTM

- To solve vanishing and exploding gradients in RNN we need LSTM
  - LSTM is designed to solve vanishing gradient problem with the help of bellow mentioned gates
    - Memory Cell
      - Forget gate
      - Input Gate
      - Output Gate

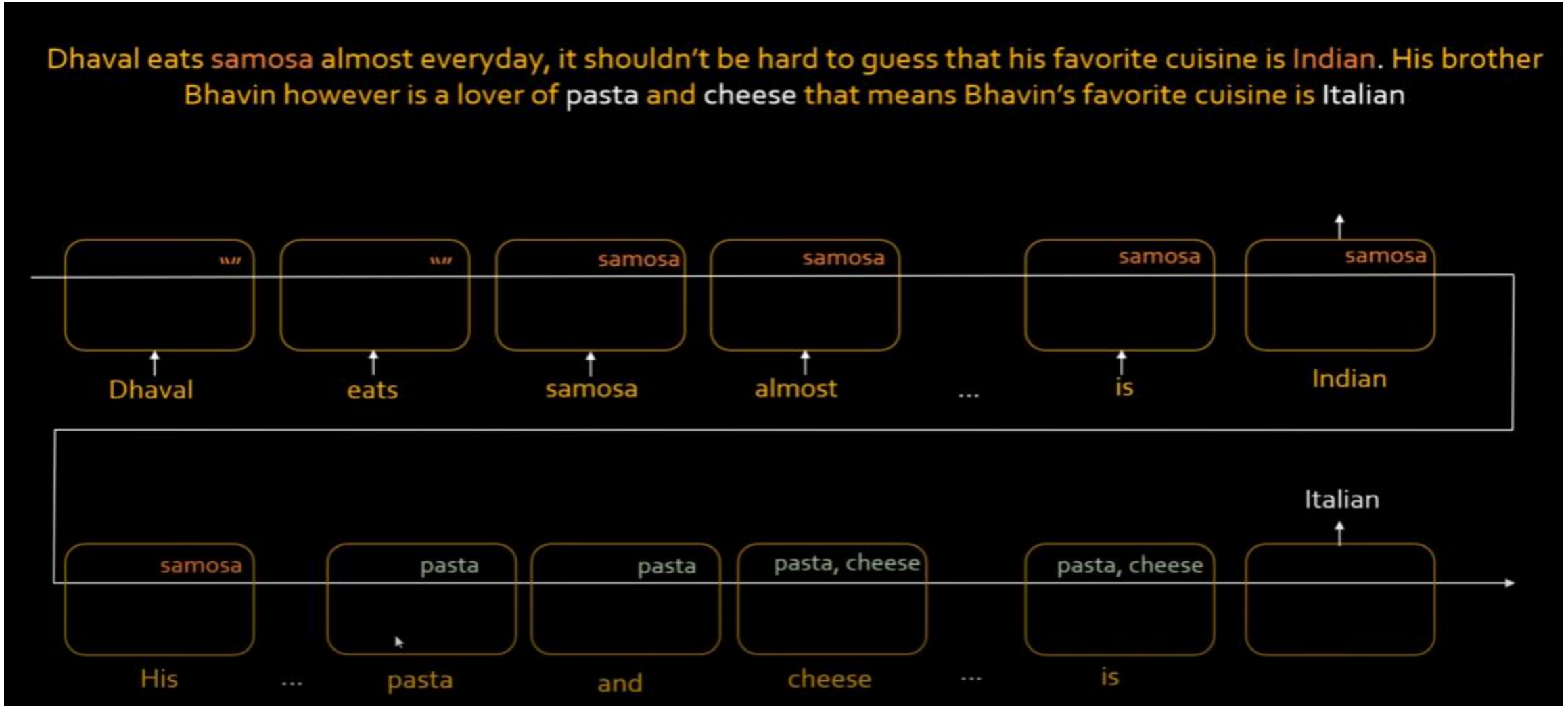
Memory Cell


Y Predicted
F
O t
X 1t

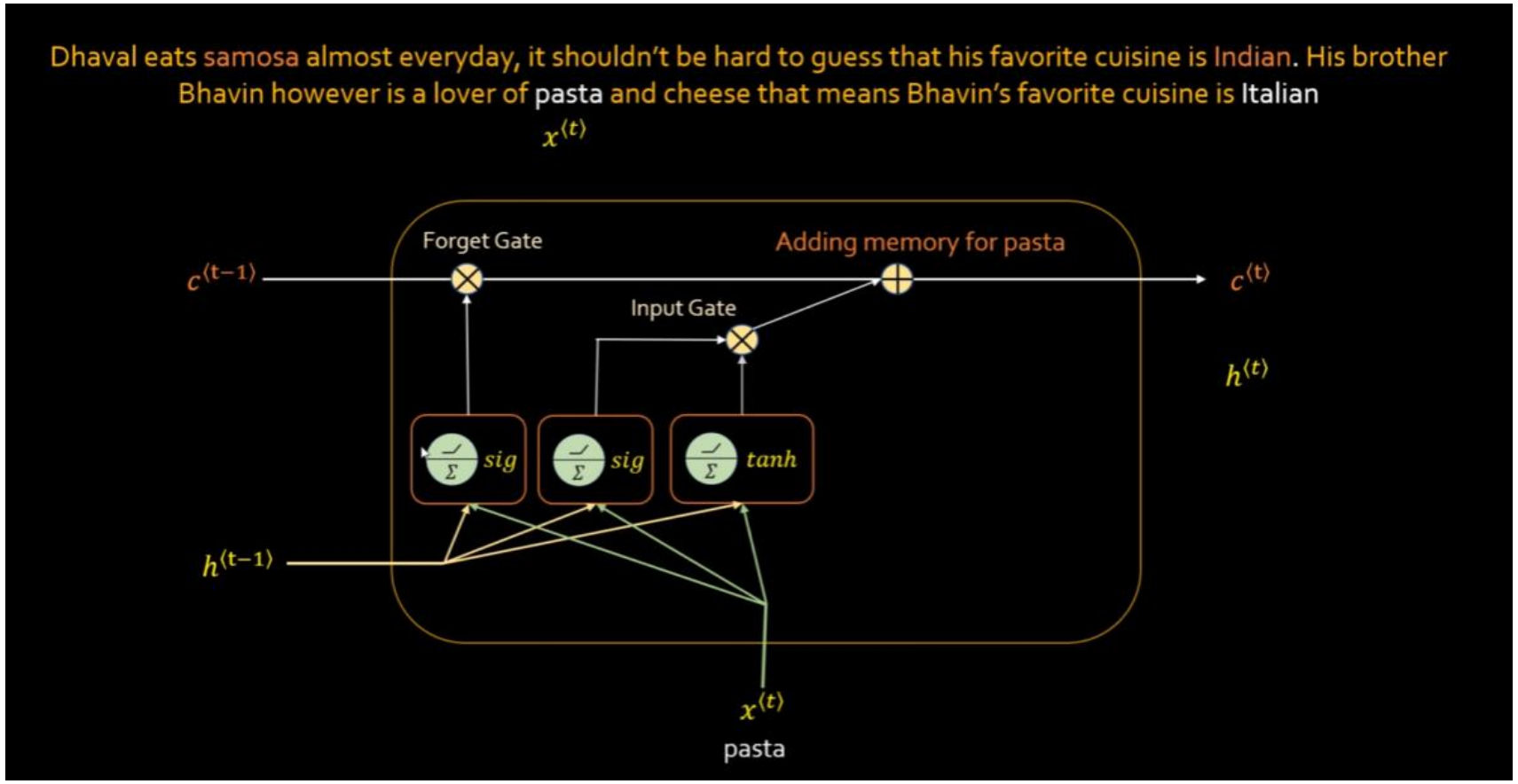

Memory cell
h⟨2⟩
h⟨4⟩
due


Memory cell
c⟨2⟩
c⟨4⟩
Long term memory
h⟨2⟩
h⟨4⟩
Short term memory
due

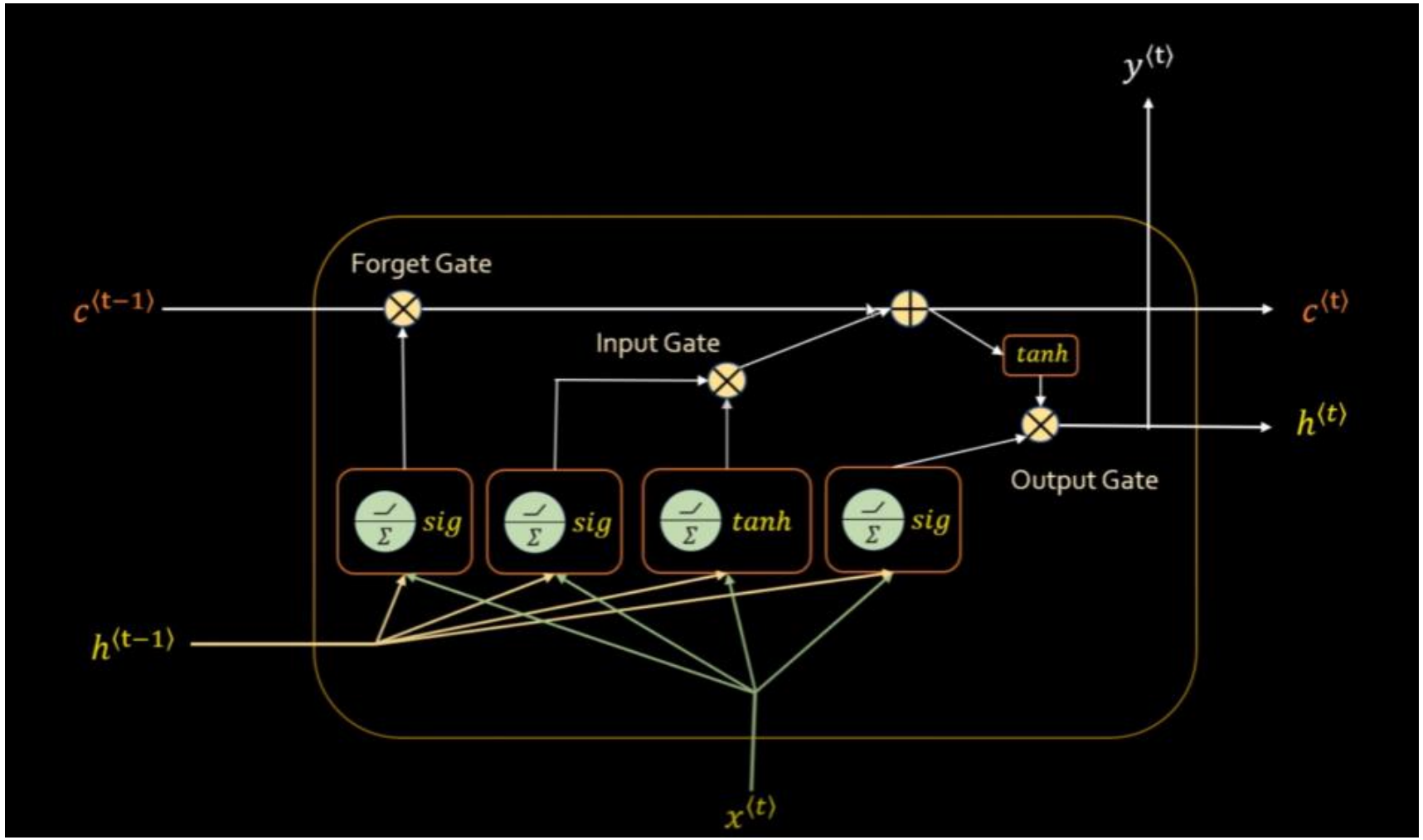
Forget gate


Dhaval eats samosa almost everyday, it shouldn't be hard to guess that his favorite cuisine is Indian. His brother Bhavin however is a lover of pasta and cheese that means Bhavin's favorite cuisine is Italian
Dhaval
eats
samosa
almost
...
is
Indian
His
...
pasta
and
cheese
...
is

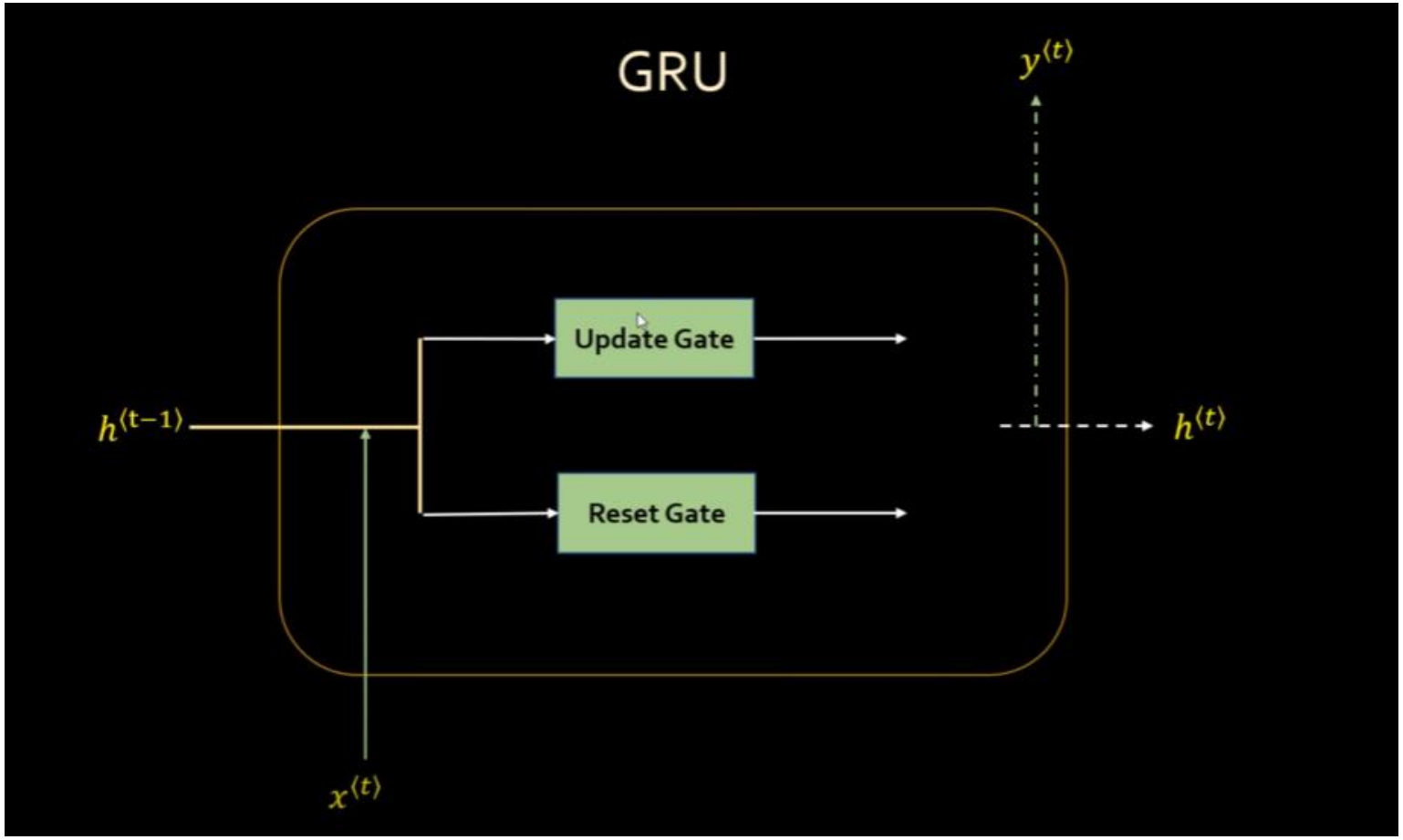
Input Gate


Dhaval eats samosa almost everyday, it shouldn't be hard to guess that his favorite cuisine is Indian. His brother Bhavin however is a lover of pasta and cheese that means Bhavin's favorite cuisine is Italian
$x^{\langle t \rangle}$
Forget Gate
Adding memory for pasta
$c^{\langle t-1 \rangle}$
$c^{\langle t \rangle}$
Input Gate
$h^{\langle t \rangle}$
σ sig
σ sig
tanh
$h^{\langle t-1 \rangle}$
$x^{\langle t \rangle}$
pasta

Output Gate


$y^{\langle t \rangle}$
Forget Gate
$c^{\langle t-1 \rangle}$
$c^{\langle t \rangle}$
Input Gate
tanh
$h^{\langle t \rangle}$
Output Gate
sig
sig
tanh
sig
$h^{\langle t-1 \rangle}$
$x^{\langle t \rangle}$

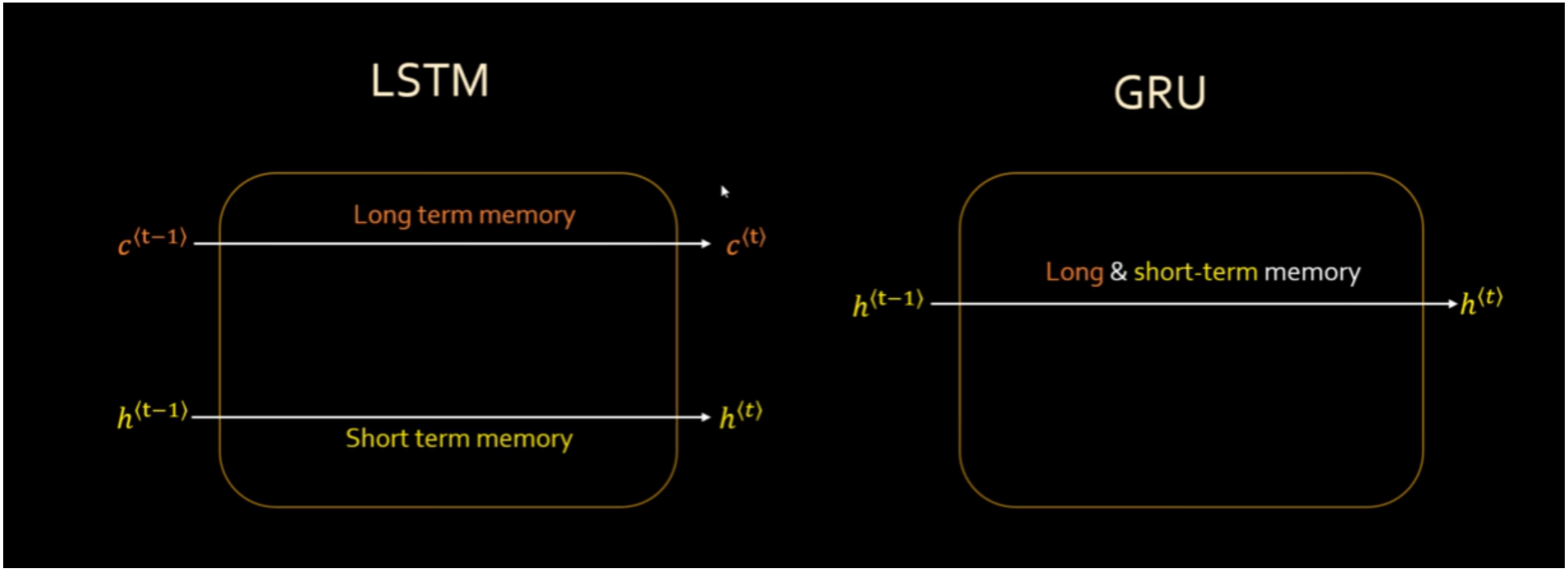
GRU Gates



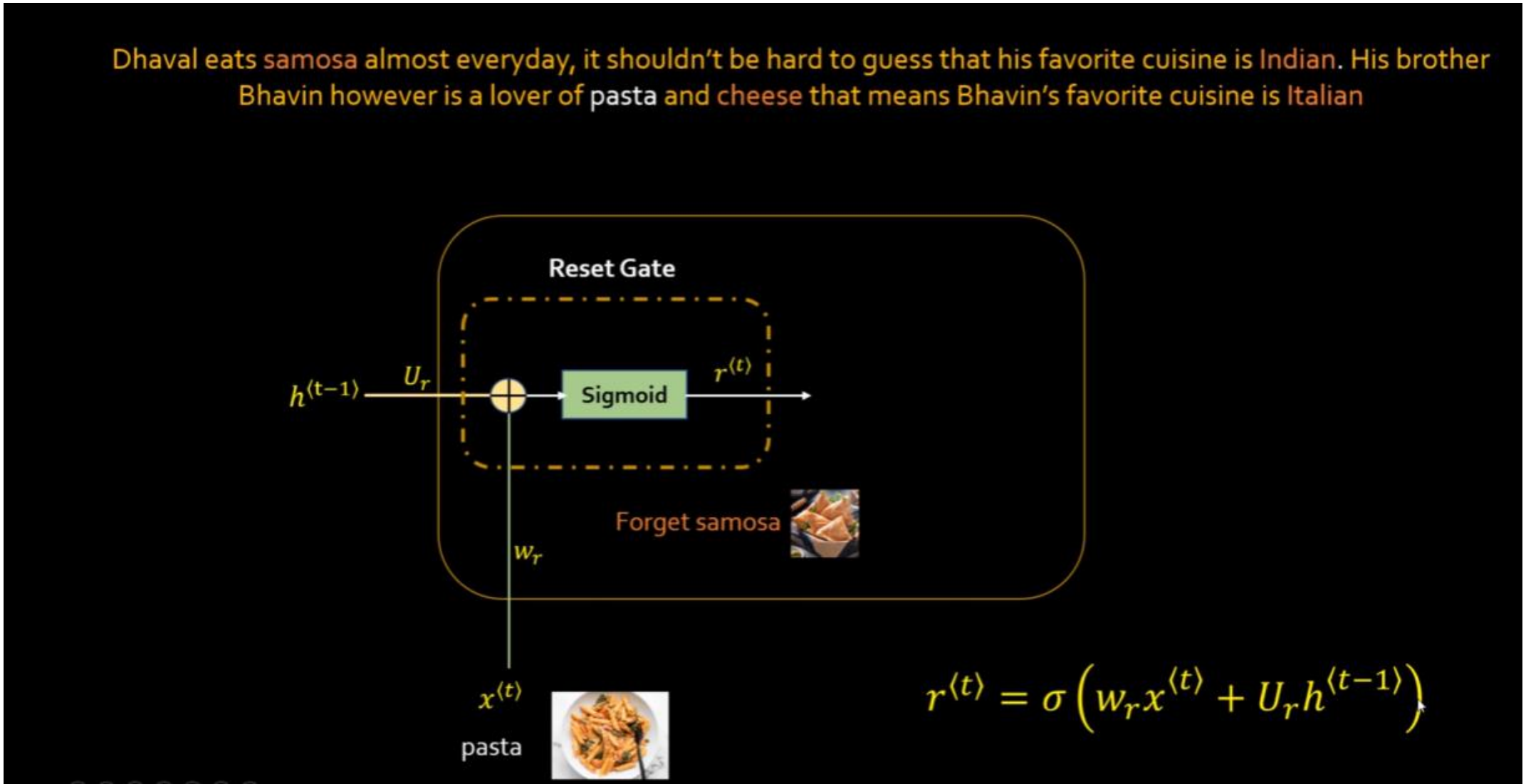
Update Gate — How much past memory to retain

Reset Gate — How much past memory to forget

GRUs – Gated Recurrent Units

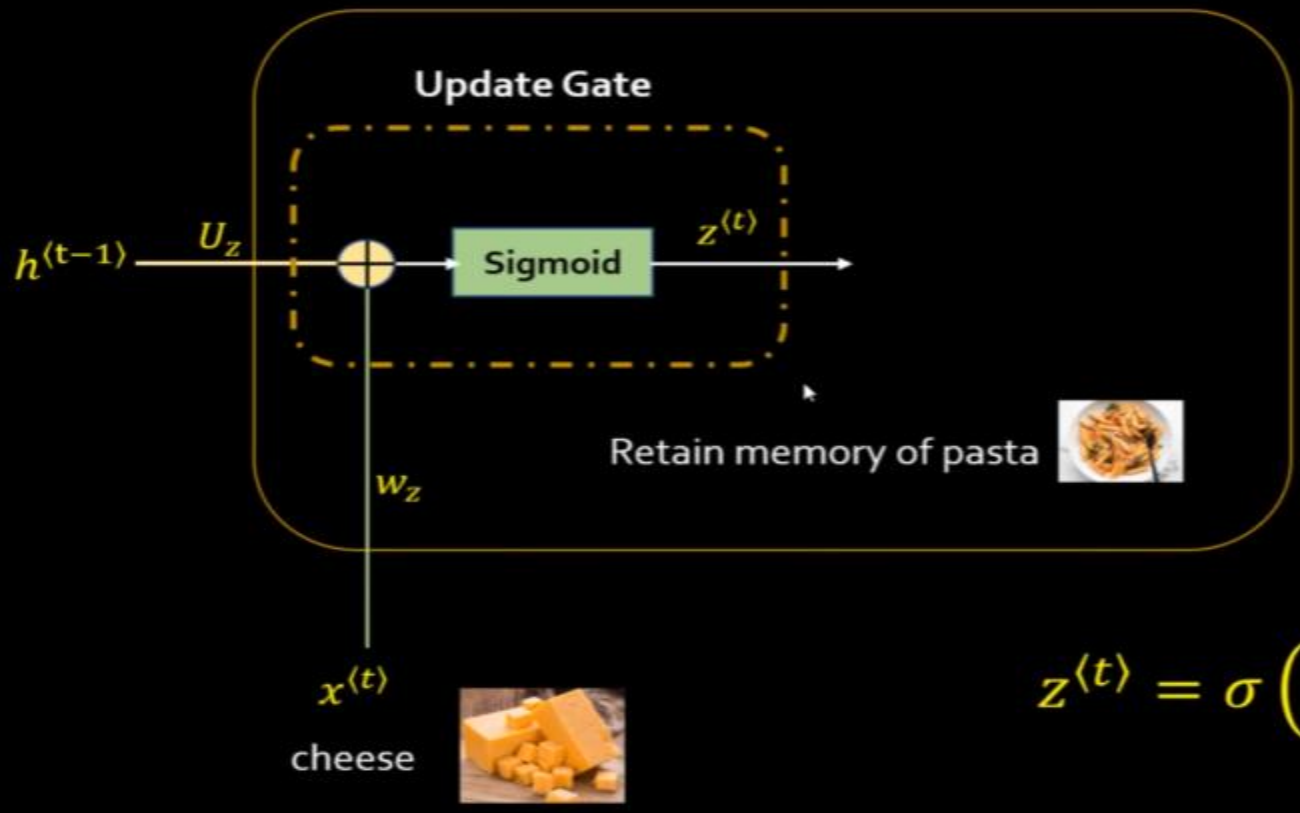


Reset Gate


Dhaval eats samosa almost everyday, it shouldn't be hard to guess that his favorite cuisine is Indian. His brother Bhavin however is a lover of pasta and cheese that means Bhavin's favorite cuisine is Italian
Reset Gate
$h^{\langle t-1 \rangle}$
U_r
Sigmoid
$r^{\langle t \rangle}$
Forget samosa
W_r
$x^{\langle t \rangle}$
pasta
$r^{\langle t \rangle} = \sigma\left(W_r x^{\langle t \rangle} + U_r h^{\langle t-1 \rangle}\right)$

Update Gate

Dhaval eats samosa almost everyday, it shouldn't be hard to guess that his favorite cuisine is Indian. His brother Bhavin however is a lover of pasta and cheese that means Bhavin's favorite cuisine is Italian



$$z^{\langle t \rangle} = \sigma\left(w_z x^{\langle t \rangle} + U_z h^{\langle t-1 \rangle}\right)$$