6CS012 - Artificial Intelligence and Machine Learning. Text Pre - processing and Representations.

Prepared By: Siman Giri {Module Leader - 6CS012}

April 11, 2025

Tutorial - 8.

1 Instructions

This sheet contains hands on exercises for various operations performed during Text Pre - processing, followed by Text Representations.

- Please Complete all the exercise with pen and on paper.

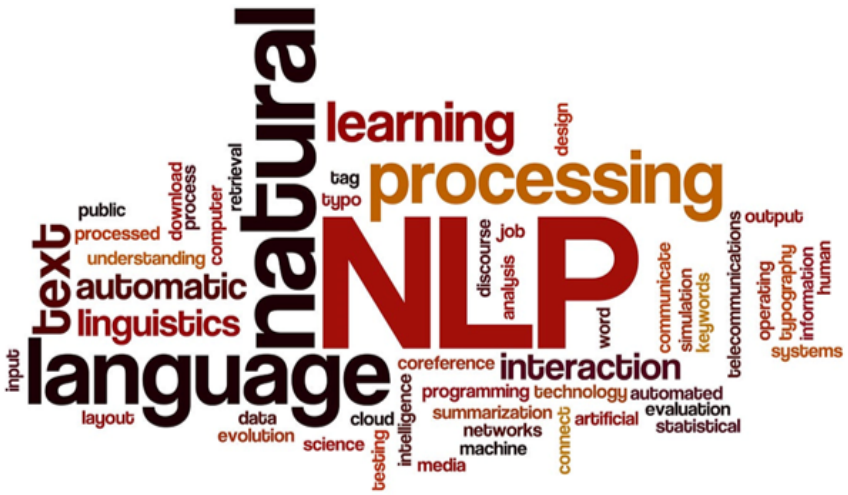

Figure 1: Text and Natural Language Processing.

2 Exercise - 1.

Manual Computation of TF - IDF Vectorization.

Objective:

In this exercise, you will gain a practical understanding of the Term Frequency - Inverse Document Frequency (TF - IDF) vectorization technique by manually computing the TF - IDF representations of a small document corpus. This will help reinforce your understanding of how text data is transformed into numerical vectors for use in machine learning models.

Corpus:

Consider the following corpus of three simple documents:

- **Document 1 - D1:** " The Cat chased the Mouse."
- **Document 2 - D2:** " The dog Barked at the Cat."
- **Document 3 - D3:** " The mouse ran away from the Cat."

Instructions:

Step - 1 - Tokenization and Pre-processing:

1. Convert all text to lowecase.
2. Remove punctuations.
3. Tokenize each document into individual words (tokens).
4. Remove stop - words if desired (optional for this exercise).

Write down the list of tokens for each document.

Step - 2 - Vocabulary Construction:

1. Identify the unique set of words (vocabulary) present across all documents.
2. Sort them alphabetically.

Step - 3 - Term Frequency Calculation:

For each document and each word in the vocabulary, compute the Term Frequency (TF) using the following formula:

$$\mathbf{TF_{t,d}} = \frac{\mathbf{f_{t,d}}}{\sum_{\mathbf{k}} \mathbf{f_{k,d}}}$$

- Where:
  - $f_{t,d} \rightarrow$ frequency of terms term t in document d.
  - The denominator is the total number of terms in document d.

Step - 3.1 - Normalized TF:

Normalized the TF using following formula:

$$\mathbf{w_{tf}(t,d)} = \begin{cases} 1 + \log(tf_{t,d}) & \text{if } tf_{t,d} > 0 \\ 0 & \text{otherwise} \end{cases}$$

Create a TF(normalized) matrix with documents as rows and vocabulary terms as columns.

Step - 4 - Inverse Document Frequency (IDF) Calculation:

Compute the **Inverse Document Frequency (IDF)** for each term in the vocabulary:

$$\mathbf{IDF(t)} = \mathbf{log_{10}}(\frac{\mathbf{N}}{\mathbf{1 + df_t}})$$

- Where:
  - $\mathbf{N} \rightarrow$ is the total number of documents in the corpus.
  - $\mathbf{df_t} \rightarrow$ is the number of documents in which the term t appears.
  - Add 1 to the denominator to prevent division by zero (smoothing)

List the IDF values for each term.

Step - 5 - Compute TF - IDF Vectors.

Use the formula:

$$\mathbf{TF - IDF(t,d)} = \mathbf{TF_{t,d}} \times \mathbf{IDF(t)}$$

Compute the TF - IDF score for each term in each document and present the Matrix.

Discussion Questions:

1. What is the intuition behind using TF - IDF instead of raw term frequency?
2. Which terms receive higher importance in the TF - IDF representations?
3. How would the representation change if you added more documents to the corpus?

Deliverables:

- Tokenize version of each document.
- Complete vocabulary list - sorted.
- Term Frequency TF table.
- Final TF - IDF vectors for each document.
- Answer to the Discussion Question.

3 "word2vec"

A Comprehensive Guide on Continuous Bag of Words Model.

Objective:

This section provides a step -by- step walkthrough of the CBOW model for training word embeddings. We will start by constructing the look - up table using a simple corpus we used in **Exercise - 1**, then proceed through the architecture design of the model, followed by detailed steps on how context words are projected into the hidden layer, averaged, and then passed to the output layer.

3.1 Exercise - 2 - Data Generation for CBOW.

Step 1 - Vocabulary and Indexing:

We will be using the corpus from **Exercise - 1**:

Corpus:

Consider the following corpus of three simple documents:

- **Document 1 - D1:** " The Cat chased the Mouse."
- **Document 2 - D2:** " The dog Barked at the Cat."
- **Document 3 - D3:** " The mouse ran away from the Cat."

After Stop - words removal:

- **D1:** `["cat", "chased", "mouse"]`
- **D2:** `["dog", "barked", "cat"]`
- **D3:** ["**mouse**", "**ran**", "**cat**"]

Vocabulary Construction

Sorted unique vocabulary across all documents:

```
"barked", "cat", "chased", "dog", "mouse", "ran"
```

Indexing:

- **To - Do:**
  - Assign a unique index to each word.
  - Also assign a one - hot encoded vector for each word.

Word	Index	One-Hot Vector
barked	0	$[\mathbf{1}, \mathbf{0}, \mathbf{0}, \mathbf{0}, \mathbf{0}, \mathbf{0}]$
cat	1	$[\mathbf{0}, \mathbf{1}, \mathbf{0}, \mathbf{0}, \mathbf{0}, \mathbf{0}]$
chased	2	$[\mathbf{0}, \mathbf{0}, \mathbf{1}, \mathbf{0}, \mathbf{0}, \mathbf{0}]$
dog	3	$[\mathbf{0}, \mathbf{0}, \mathbf{0}, \mathbf{1}, \mathbf{0}, \mathbf{0}]$
mouse	4	$[\mathbf{0}, \mathbf{0}, \mathbf{0}, \mathbf{0}, \mathbf{1}, \mathbf{0}]$
ran	5	$[\mathbf{0}, \mathbf{0}, \mathbf{0}, \mathbf{0}, \mathbf{0}, \mathbf{1}]$

Table 1: Word index mapping with one-hot vectors

Step 2 - Create a Look up table with Context Target pair:

For D1: `["cat", "chased", "mouse"]`

Context (Words)	Context (Index)	Target Word	Target (Index)
[chased]	[2]	cat	1
[cat, mouse]	[1,4]	chased	2
[chased]	[2]	mouse	4

Table 2: Look Up - Table - Word context pairs.

- **To - Do:**
  - Create a look Up table also for DOC 2 and DOC 3.
  - Answer the **Discussion Question**.

Discussion Question:

If you notice, word like "cat" appear in multiple documents, and its context can differ in each case. For example:

- **In Doc 1** : "cat" might appear as `["cat", "chased", "mouse"]` context - `["chased", "mouse"]`.
- **In Doc 2** : "cat" might appear as `["dog", "barked", "cat"]` context - `["dog", "barked"]`.
- **In Doc 3** : "cat" might appear as `["mouse", "ran", "cat"]` context - `["mouse", "ran"]`.

1. How does CBOW handles this?
2. Can you connect the idea to Distributional Hypothesis?

3.2 `"word2vec"` - CBOW Architecture Design and Training.

Objective:

In this section we will learn how the CBOW model works by computing context - based word predictions. In this exercise, we will compute the hidden layer and output layer projections step -by- step using a simple dataset and random weight initialization.

Step 1 - Define a CBOW Architecture:

- **To - Do:**
  - Define the **input layer, hidden layer and output layer.**

Understanding the Architecture:

The CBOW architecture consists of three layers defined as:

input layer:

The context words (one - hot encoded) will be fed as input. In our case, with a context window of 1, (one words in the context), there will be two one - hot vectors.In our example:

- For context `["cat", "mouse"]` and target `["chased"]`.
  - **Context words and one hot vector representations:**
    - `"cat"`$\rightarrow$ index - 1 : one hot $[\mathbf{0}, \mathbf{1}, \mathbf{0}, \mathbf{0}, \mathbf{0}, \mathbf{0}]$.
    - `"mouse"`$\rightarrow$ index - 4 : one hot $[\mathbf{0}, \mathbf{0}, \mathbf{0}, \mathbf{0}, \mathbf{1}, \mathbf{0}]$
  - **Target word and its one hot vector representations:**
    - `"chased"` $\rightarrow$ index - 2: one hot $[\mathbf{0}, \mathbf{0}, \mathbf{1}, \mathbf{0}, \mathbf{0}, \mathbf{0}, \mathbf{0}]$
- **Number of neurons in the input Layer:** Number of neurons depends on the length of one hot encoded representations of our tokens in vocabulary.

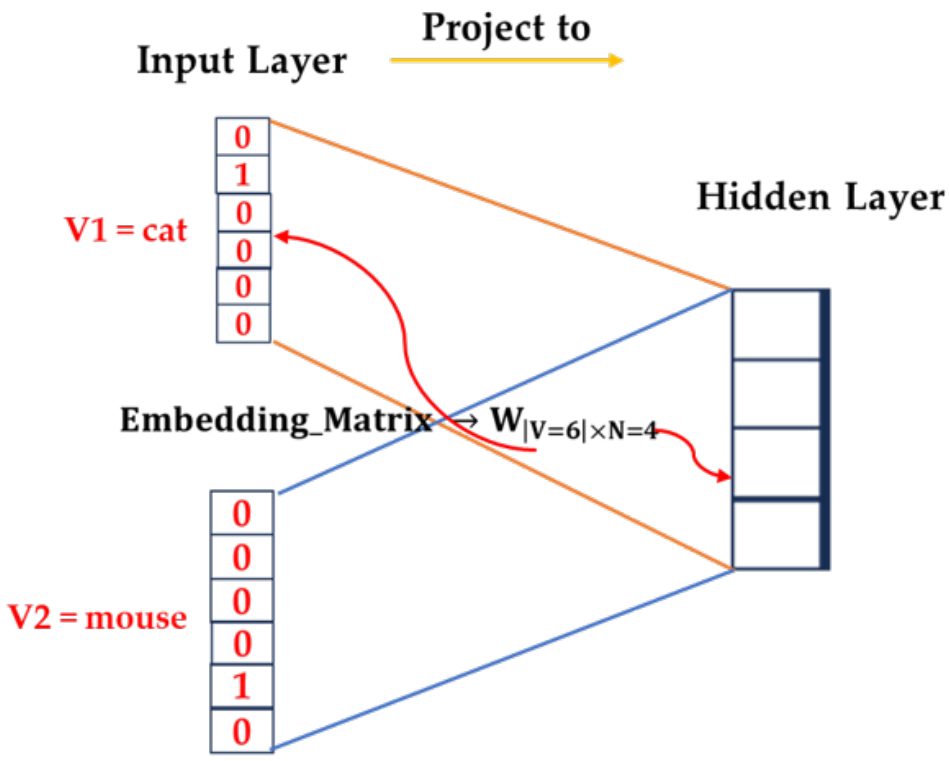

Figure 2: Understanding Input Layer.

Hidden Layer:

The hidden layer in CBOW has size N (embedding dimension), which is typically much smaller than the input layer size (vocabulary size V). This layer computes the average (or sum) of the embeddings of the context words, forming a dense representation used to predict the target word.
To further breakdown:

- **Hidden Layer Size (N):**
  - The hidden layer size (also called the embedding dimension, often denoted as d or N or H) is a hyperparameter. While you can choose any value for N, it is typically much smaller than the input layer size (which is the size of the vocabulary, V). For example, N = 300 is common for word embeddings, while V could be 10,000 or more.
  - There is no strict requirement that N must be less than the input layer size, but in practice, it almost always is because the goal is to learn a dense, low-dimensional representation (embedding) of words.
- **Role of the Hidden Layer:**
  - In CBOW, the hidden layer is where the average of the context word embeddings (projections) is computed.
  - The input layer is one-hot encoded vectors of the context words. These are multiplied by the input embedding matrix i.e.

$$\mathbf{W}_{|\mathbf{V}|\times\mathbf{N}} \rightarrow \texttt{Embedding Matrix.}$$

to get their embeddings.

1. $|\mathbf{V}| \rightarrow$ Length of input one hot encoded vector.
2. $\mathbf{N} \rightarrow$ Number of neurons in Hidden Layer.

– The embeddings of the context words are averaged (or summed) to produce the hidden layer representation.

- **No Activation Layer:**
  – The hidden layer in CBOW is a linear layer (no non-linear activation like ReLU or sigmoid is applied). This is a key difference from traditional neural networks.

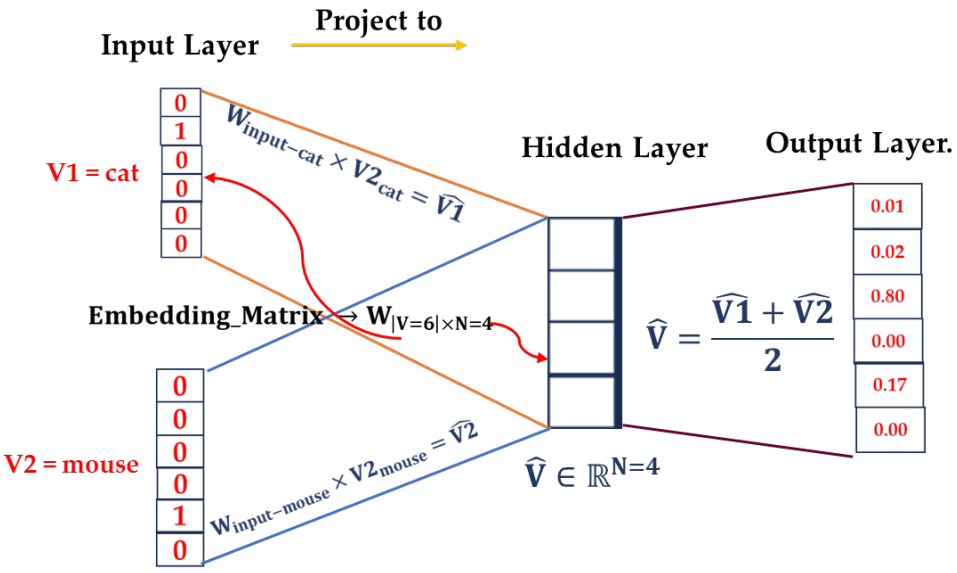

Figure 3: Understanding Hidden Layer

Step 2 - Understanding Weight Matrices Dimension and Random Initializations:

- **To - Do:**
  – Define the dimensions of the weight matrices:
    1. Embedding Weight Matrices $\mathbf{W_{input}}$
    2. Hidden Weight Matrices $\mathbf{W_{hidden}}$

Dimensions of Weight Matrices:

- **Weight Matrix for Context to Hidden Layer -** `Embedding Weight Matrix` $\rightarrow \mathbf{W_{input}}$**:**
  – The context words are input into the network, adn each word in the vocabulary is represented by a one - hot vector of size $\mathbf{V} \rightarrow$ `vocabulary size`.
  – The hidden layer has $\mathbf{N}$ units - which we can be set as desired. For our architecture we picked 4.

- The Dimension of Embedding Matrix is:

$$\mathbf{W_{input}} \rightarrow |\mathbf{V}| \times \mathbf{N}$$

For our Example $\mathbf{W_{input}} \rightarrow \mathbf{6} \times \mathbf{4}$

- **Random Initializations:**

$$W_{\text{input}} = \begin{bmatrix} 0.1, & 0.2, & -0.1, & 0.0, \rightarrow & \texttt{\# index 0} \\ 0.4, & 0.5, & -0.1, & 0.2, \rightarrow & \texttt{\# index1 = "cat"} \\ -0.2, & 0.3, & 0.2, & -0.1, \rightarrow & \texttt{\# index 2} \\ 0.3, & -0.1, & 0.1, & 0.0, \rightarrow & \texttt{\# index 3} \\ 0.5, & -0.4, & 0.1, & 0.1, \rightarrow & \texttt{\# index 4 = "mouse"} \\ -0.3, & 0.0, & -0.2, & 0.3, \rightarrow & \texttt{\# index 5} \end{bmatrix}$$

- **Weight Matrix for Hidden Layer to Output Layer -** `Weight Hidden Layer` $\rightarrow \mathbf{W_{hidden}}$**:**
  - The hidden layer output is projected onto the output layer.
  - The output layer represents the vocabulary, so it has V units (one for each word in the vocabulary).
  - The Dimension of Hidden Weight Matrix is:

$$\mathbf{W_{hidden}} \rightarrow \mathbf{N} \times \mathbf{V}$$

For our Example: $\mathbf{W_{hidden}} \rightarrow \mathbf{4} \times \mathbf{6}$.

  - **Random Initializations:**

$$W_{\text{hidden}} = \begin{bmatrix} 0.2, & -0.1, & 0.3, & 0.1, & 0.4, & 0.0, \\ 0.1, & 0.3, & -0.2, & -0.1, & 0.2, & 0.5, \\ -0.4, & 0.5, & 0.1, & -0.2, & -0.1, & 0.3, \\ 0.1, & -0.2, & -0.4, & 0.3, & -0.3, & 0.1, \end{bmatrix}$$

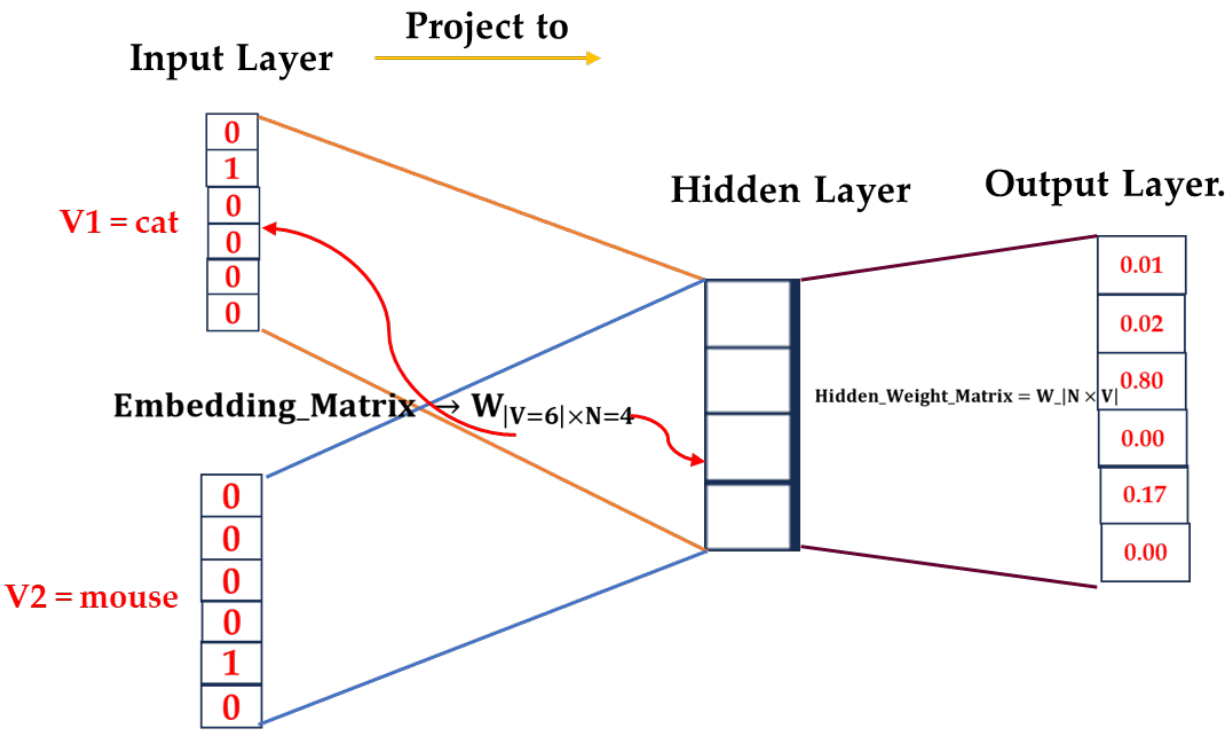

Figure 4: Random Initialization of Weight Matrix.

Step 3 - Computations at Hidden Layer:

Project Context Words to Hidden Layer:

- **To - Do:**
  - For each context word, multiply its one-hot vector with the input weight matrix $\mathbf{W_{input}}$.This operation effectively selects the row from $\mathbf{W_{input}}$ that corresponds to the word's index in the vocabulary.

For the Context Pair - `[cat, mouse]`:

Projection for "cat" (Index 1)

Using one-hot vector $[0, 1, 0, 0, 0, 0]^\top$:

$$\hat{\mathbf{V}}\mathbf{1} = \begin{bmatrix} 0.1, & 0.2, & -0.1, & 0.0, \\ 0.4, & 0.5, & -0.1, & 0.2, \\ -0.2, & 0.3, & 0.2, & -0.1, \\ 0.3, & -0.1, & 0.1, & 0.0, \\ 0.5, & -0.4, & 0.1, & 0.1, \\ -0.3, & 0.0, & -0.2, & 0.3, \end{bmatrix} \times [\mathbf{0}, \mathbf{1}, \mathbf{0}, \mathbf{0}, \mathbf{0}, \mathbf{0}]^\top = [0.4, 0.5, -0.1, 0.2] \quad (1)$$

Computation at (1) selects the second row of the matrix.

Projection for "mouse" (Index 4)

Using one-hot vector $[0, 0, 0, 0, 1, 0]^\top$:

$$\hat{\mathbf{V}}\mathbf{2} = \begin{bmatrix} 0.1, & 0.2, & -0.1, & 0.0, \\ 0.4, & 0.5, & -0.1, & 0.2, \\ -0.2, & 0.3, & 0.2, & -0.1, \\ 0.3, & -0.1, & 0.1, & 0.0, \\ 0.5, & -0.4, & 0.1, & 0.1, \\ -0.3, & 0.0, & -0.2, & 0.3, \end{bmatrix} \times [\mathbf{0}, \mathbf{0}, \mathbf{0}, \mathbf{0}, \mathbf{1}, \mathbf{0}]^\top = [0.5, -0.4, 0.1, 0.1] \quad (2)$$

Compute Average Context Vector:

- **To - Do:**
  - For Context `["cat", "mouse"]`

$$\texttt{Average Context Vector}\hat{\mathbf{V}} = \frac{[\mathbf{0.4}, \mathbf{0.5}, \mathbf{-0.1}, \mathbf{0.2}] + [\mathbf{0.5}, \mathbf{-0.4}, \mathbf{0.1}, \mathbf{0.1}]}{\mathbf{2}} = [\mathbf{0.45}, \mathbf{0.05}, \mathbf{0.0}, \mathbf{0.15}]$$

Project to Output Layer:

- **To - Do**
  - Multiply average vector $\hat{\mathbf{V}}$ with Hidden weight Matrix $\mathbf{W_{hidden}}$

$$\hat{\mathbf{V}} = \begin{bmatrix} 0.2, & -0.1, & 0.3, & 0.1, & 0.4, & 0.0, \\ 0.1, & 0.3, & -0.2, & -0.1, & 0.2, & 0.5, \\ -0.4, & 0.5, & 0.1, & -0.2, & -0.1, & 0.3, \\ 0.1, & -0.2, & -0.4, & 0.3, & -0.3, & 0.1, \end{bmatrix} \times [\mathbf{0.45}, \mathbf{0.05}, \mathbf{0.0}, \mathbf{0.15}]^\top$$

$$= [\mathbf{0.11}, -\mathbf{0.06}, \mathbf{0.65}, \mathbf{0.085}, \mathbf{0.145}, \mathbf{0.04}] \quad (3)$$

Step 4 - The Output Layer:

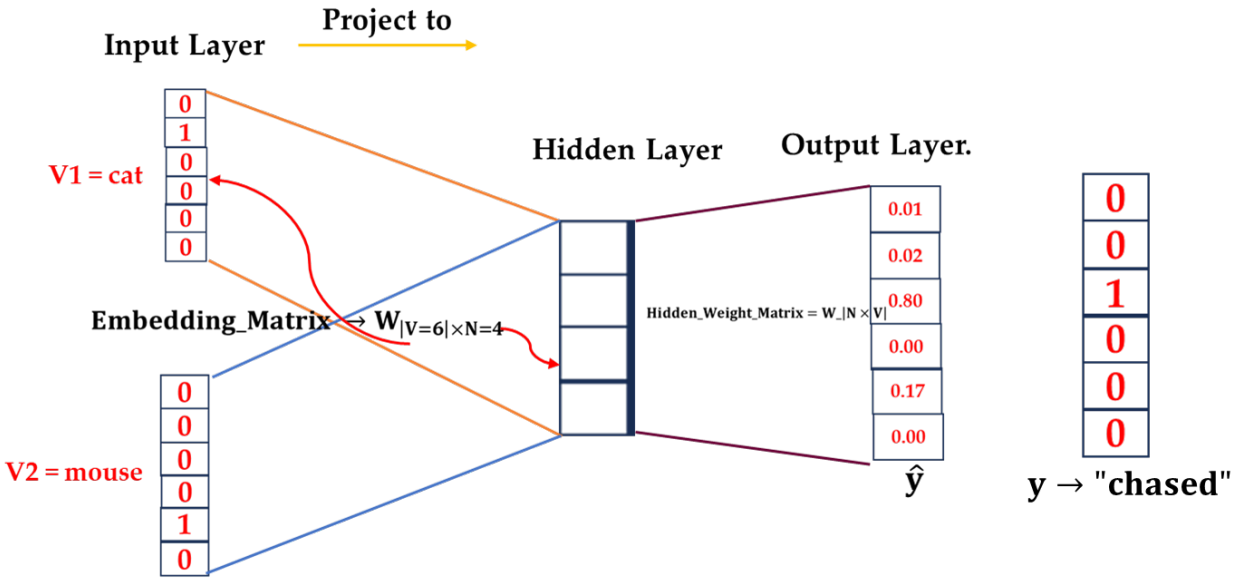

Figure 5: The Complete Architecture.

Following are the key properties of Output Layer:

Purpose:

The output layer is responsible for generating a probability distribution over the entire vocabulary, predicting the most likely target word given the averaged context embeddings (from the hidden layer).

Size:

The output layer has a size equal to the vocabulary size (V), where each neuron corresponds to a word in the vocabulary.

For example, if the vocabulary has $\mathbf{V = 10,000}$ words, the output layer will have $\mathbf{10,000}$ neurons.

- **How many Neurons we must put in our architecture?**

Operation:

The hidden layer vector (size N) is multiplied by the output embedding matrix (of size $N \times V$), producing logits (unnormalized scores) for each word in the vocabulary. These logits are then passed through a softmax activation function to convert them into probabilities:

$$\mathbf{P}(\text{word}_\mathbf{i} \mid \text{context}) = \frac{\mathbf{e^{z_i}}}{\sum_{\mathbf{j=1}}^{\mathbf{V}} \mathbf{e^{z_j}}}$$

where z_i is the score for the i-th word.

Output Embedding Matrix:

This matrix (often called the "output weights") is different from the input embedding matrix. Each column in this matrix represents an alternative embedding for a word (used only during training). After training, we typically discard the output matrix and keep only the **input embeddings** (since they encode the learned word representations).

4 Training and After Training:

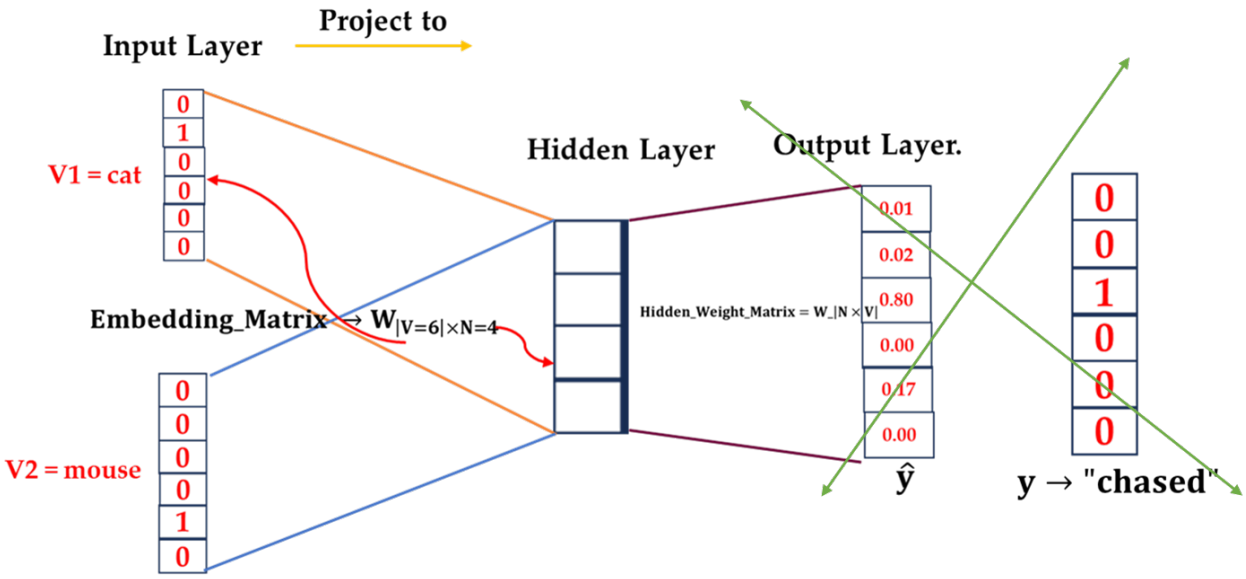

Figure 6: After Training.

The Word2Vec model is typically trained for 3 to 50 epochs using optimization techniques such as Stochastic Gradient Descent (SGD) and a suitable loss function, often binary cross-entropy in the case of negative sampling (Mikolov et al., 2013). Upon completion of training, the output layer is discarded, and the input weight matrix $\mathbf{W}_{\mathbf{input}}$is retained as the dense, low-dimensional embedding representation of the vocabulary. This matrix captures the semantic and syntactic relationships between words based on their contextual co-occurrences across the training corpus.

Good Luck.