CMR
INSTITUTE OF
TECHNOLOGY

USN | | | | | | | | | | |

Internal Assessment Test 1– Nov. 2025

Sub:	Social Media Analytics							Sub Code:	MMCA311F
Date:	7/11/2025	Duration:	90 min's	Max Marks:	50	Sem:	III	Branch:	MCA

Note : Answer FIVE FULL Questions, choosing ONE full question from each Module

		MARKS	OBE	
			CO	RBT
	PART I			
1	Differentiate between Structured and Unstructured data?	[10]	CO1	L2
	OR			
2	Describe an overview of face book, YouTube, LinkedIn and instagram social media platforms	[10]	CO1	L2
	PART II			
3	Explain the key metrics engagement, reach, sentiment score and impressions	[10]	CO1	L2
	OR			
4	Explain in detail how you access twitter data using twitter API.	[10]	CO1	L2
	PART III			
5	What are the different pre-processing techniques available for text? Explain with an example	[10]	CO2	L2
	OR			
6	Write python code to apply naive bayes algorithm for sentiment classification	[10]	CO2	L4
	PART IV			
7	Explain the advantages of using LSTM over RNN for sentiment classification?	[10]	CO2	L2
	OR			
8	Describe the fundamentals of Sentiment Analysis- Positive, Negative, Neutral Sentiments.	[10]	CO2	L2
	PARTV			
9	What is word embeddings? Explain how to generate word embeddings using BERT?	[10]	CO2	L2
	OR			
10	Discuss word2vec with code snippets in detail.	[10]	CO2	L2

1. Differentiate between Structured and Unstructured data.

	Structured Data	Semi Structured Data	Unstructured Data
Technology	Relational database table	XML / RDF	Character and binary data
Transaction Management	Matured transaction, various concurrency techniques	Transaction management adapted from RDBMS not matured	No transaction management, no concurrency
Version Management	Versioning over tuples, rows, tables etc.	Versioning over tuples or graphs is possible	Versioned as a whole
Flexibility	Schema dependent rigorous schema	Flexible, tolerant schema	Very flexible, absence of schema
Scalability	Scaling DB schema is difficult	Schema scaling is simple	Very scalable
Robustness	Very robust	New technology not widely spread	-
Query Performance	Structured query allows complex joins	Queries over anonymous nodes are possible	Only textual queries possible

2. Describe an overview of face book, YouTube, LinkedIn and instagram social media platforms.

Twitter

• **Description:** Twitter is a popular social media service that enables users to find the latest world events and interact with other users through various types of messaging content, known as **tweets**. Users can access Twitter via its website interface, mobile applications, or a short message service (SMS).

• **API Features:** Twitter provides various API endpoints for completing diverse tasks.

◦ The **Search API** can be used to retrieve historical tweets.

◦ The **Account Activity API** allows access to account activities.

◦ The **Direct Message API** is available for sending direct messages.

◦ The **Ads API** helps in creating advertisements.

• **Popularity:** Twitter is a very popular social media networking service that can enhance application engagement. At the end of 2018, Twitter reported more than **335 million monthly active users**.

• **Price:** Twitter provides its APIs for **free**.

• **Ease of Use:** The Twitter APIs are remarkably **easy to use**. Twitter offers comprehensive documentation to assist developers in flawlessly integrating its APIs for specific use cases.

Facebook

• **Description:** Facebook is a prominent social networking platform where users communicate using messages, photos, comments, videos, news, and other interactive content.

• **API Features:** Facebook offers various APIs and SDKs (Software Development Kits) that enable developers to access its data and extend their application capabilities.

◦ The **Facebook Graph API** is an HTTP-based API that serves as the main method for accessing the platform's data.

• **Popularity:** At the end of 2018, Facebook boasted more than **2.2 billion monthly active users**, making it the **most popular social media platform in the world**.

• **Price:** The Facebook APIs are provided for **free**.

Instagram

• **Description:** Instagram is a Facebook-owned social networking platform dedicated to users sharing photos and videos.

• **API Features:** Facebook provides numerous APIs to enable developers to create tools that enhance the user experience on the Instagram platform.

◦ These APIs allow users to share stories and daily highlights from an application to Instagram.

◦ The **Instagram Graph API** specifically grants developers access to data from businesses operating Instagram accounts. With this API, users can conveniently

manage and publish media objects, discover other businesses, track mentions, analyze valuable metrics, moderate comments, and search hashtags.

• **Popularity:** By the end of 2018, Instagram had over **1 billion monthly active users**.

• **Price:** The Instagram APIs are offered for **free**.

• **Ease of Use:** The Instagram APIs are **easy to use**, with Facebook providing detailed documentation.

YouTube

• **Description:** YouTube is a popular, Google-owned platform renowned for sharing videos, music, and other visual content.

• **API Features:** The YouTube API allows developers to embed YouTube functionalities directly into their websites and applications.

◦ Developers can use the API to enable users to play YouTube videos within their applications, find YouTube content, manage playlists, upload videos, and perform other tasks.

◦ The API also supports analyzing video performance, scheduling live streaming broadcasts, and adding the YouTube subscribe button.

• **Popularity:** As of May 2018, YouTube attracted more than **1.8 billion monthly users**, making it the **second most popular social media platform in the world**.

• **Price:** The YouTube API is offered for **free**.

• **Ease of Use:** Google has provided **easy-to-follow developer documentation** for the YouTube API.

3. Explain the key metrics engagement, reach, sentiment score and Impressions.

Impressions

Impressions refer to the number of times your content has been shown on someone’s screen. If we have a Facebook Ad that pops up on screens 500 times. That’s 500 impressions.

Impressions aren't really that meaningful a metric for content marketing ROI, especially considering how most platforms treat them.

Reach

While impressions refer to the number of times an ad is displayed, reach refers to the number of users who have seen a piece of content.

Say you have an ad that is displayed on a screen 500 times to 300 users. Your reach would be 300 users.

The number of impressions an ad has will always be equal to or greater than an ad's reach. Platforms like Facebook typically show an ad multiple times to at least some of the ad's target audience.

Engagement

When it comes to marketing KPIs, *engagement* trumps impressions and reach in terms of how meaningful a piece of content is. Engagement is the action a user takes with content, and it could be anything from:

- A click-through to a website or ad
- A social media or website article share
- A comment on a blog or social post
- A click of a "Shop Now" button on an Instagram photo
- A click on a hyperlink in an article that guides a user's customer journey

Sentiment Scoring:

Sentiment scoring is a metric used to quantify the sentiment or emotion expressed in qualitative data like customer feedback or social media interactions. It depicts the level of emotion analysis as positive, negative or

neutral. Its calculated using sentiment analysis resulting in a numerical representation that can range from 1 (negative) to 1 (positive). This score helps organizations understand to analyze the customer interactions are perceived as positive, neutral, or negative which helps in decision-making and strategy development.

4. Explain in detail how you access twitter data using Twitter API.

- Twitter might be described as a real-time, highly social micro blogging service that allows users to post short status updates, called tweets, that appear on timelines.
- Tweets may include one or more entities in their (currently) 280 characters of content and reference one or more places that map to locations in the real world.
- An understanding of users, tweets, and timelines is particularly essential to effective use of Twitter's API
- Tweets are the essence of Twitter, and while they are notionally thought of as short strings of text content associated with a user's status update, there's really quite a bit more metadata
- In addition to the textual content of a tweet itself, tweets come bundled with two additional pieces of metadata that are of particular note: entities and places.
- Tweet entities are essentially the user mentions, hash tags, URLs, and media that may be associated with a tweet, and places are locations in the real world that may be attached to a tweet.
- Note that a place may be the actual location in which a tweet was authored, but it might also be a reference to the place described in a tweet.

Accessing Twitter Data:

- i)we can use Twitter API
- ii)we can use some of the tools like Flume
- We can customize #tag for data. Provides data for a week.

- It provides username,timestamp,content of the tweet,URL,retweets.
- Twitter has taken great care to craft an elegantly simple RESTful API that is intuitive and easy to use.
- A particularly beautiful Python package that wraps the Twitter API and mimics the public API semantics almost one-to-one is twitter.
- Like most other Python packages, you can install it with pip by typing
  pip install twitter in a terminal.
- One popular alternative is tweepy.
- Before we make any API requests to Twitter, we need to create an application at https://dev.twitter.com/apps.
- Creating an application is the standard way for devel-
  opers to gain API access and for Twitter to monitor and interact with third-party platform developers as needed.
- To use twitter API we must apply for a Twitter developer account and be approved in order to create new apps.
- Creating an app will also create a set of authentication tokens that will let us programmatically access the Twitter platform.

```
import twitter
# Go to http://dev.twitter.com/apps/new to create an app and get values
# for these credentials, which you'll need to provide in place of these
# empty string values that are defined as placeholders.
# See https://developer.twitter.com/en/docs/basics/authentication/overview/oauth
# for more information on Twitter's OAuth implementation.
CONSUMER_KEY = ''
CONSUMER_SECRET = ''
OAUTH_TOKEN = ''
OAUTH_TOKEN_SECRET = ''
auth = twitter.oauth.OAuth(OAUTH_TOKEN, OAUTH_TOKEN_SECRET,
CONSUMER_KEY, CONSUMER_SECRET)
twitter_api = twitter.Twitter(auth=auth)
# Nothing to see by displaying twitter_api except that it's now a
# defined variable
print(twitter_api)
```

Ex 2: Retrieving Topics

Example 1-2. Retrieving trends

```
# The Yahoo! Where On Earth ID for the entire world is 1.
# See http://bit.ly/2BGWJBU and
# http://bit.ly/2MsvwCQ
WORLD_WOE_ID = 1
US_WOE_ID = 23424977
# Prefix ID with the underscore for query string parameterization.
# Without the underscore, the twitter package appends the ID value
# to the URL itself as a special case keyword argument.
world_trends = twitter_api.trends.place(_id=WORLD_WOE_ID)
us_trends = twitter_api.trends.place(_id=US_WOE_ID)
print(world_trends)
print()
print(us_trends)
```

5. What are the different pre-processing techniques available for text? Explain with an example

Text preprocessing is the process of transforming raw text into a structured and analyzable form for NLP tasks.

Tokenization:

- Tokenization is the process of splitting text into smaller units called tokens, such as words, subwords, or sentences.

  Ex: Text: "NLP makes machines understand language."

  Tokens: ["NLP", "makes", "machines", "understand", "language", "."]

  Tokenization will
- Enables word-level analysis
- Simplifies downstream NLP tasks
- Helps in vectorization and embedding creation

  Types of Tokenization
- **Word Tokenization:** Splits text into words

- **Sentence Tokenization:** Splits text into sentences
- **Subword Tokenization:** Splits words into smaller meaningful units (used in BERT, GPT)

```
import nltk

nltk.download('punkt')

from nltk.tokenize import word_tokenize, sent_tokenize

text = "NLP makes machines understand language. It's fascinating!"

print("Word Tokens:", word_tokenize(text))

print("Sentence Tokens:", sent_tokenize(text))
```

Stopword Removal:

- In **NLP(Natural Language Processing)**, stop words are the words that are filtered out before or after processing text data, such as "is", "and", "a" etc. These words do not add meaning to the text and can be removed to improve the efficiency.
- The **Natural Language Toolkit (NLTK)** is the python library that provides the easy to use interface and the tools for text processing such as tokenization and stop word removal.

Stemming:

- Stemming is the process of reducing words to their root form by chopping off prefixes or suffixes — often without considering linguistic correctness.

Example:

- “playing” → “play”
- “flies” → “fly”

Lemmatization:

- Lemmatization reduces words to their lemma (dictionary form) using vocabulary and morphological analysis. Unlike stemming, it returns meaningful root words, which is often part of image processing that combine vision and language data.

Example:

- “running” → “run”
- “better” → “good”

Stemming:

- Reduces dimensionality of the dataset

- Speeds up processing in large text corpora
- Useful when exact meaning is less critical (e.g., search engines)

Popular Stemming Algorithms

- Porter Stemmer (most common in NLP)
- Snowball Stemmer (improved version of Porter)
- Lancaster Stemmer (more aggressive)

`

Why Use Lemmatization?

- Preserves semantic meaning
- More accurate than stemming
- Necessary for linguistic tasks like part-of-speech tagging

Ex:

from nltk.stem import WordNetLemmatizer

Initialize the lemmatizer

lemmatizer = WordNetLemmatizer()

Lemmatize words

print(lemmatizer.lemmatize("running", pos="v")) # Output: run

print(lemmatizer.lemmatize("better", pos="a")) # Output: good

print(lemmatizer.lemmatize("geese", pos="n")) # Output: goose

6. Write python code to apply naive bayes algorithm for sentiment classification.

```
import pandas as pd
from sklearn.model_selection import train_test_split
# Load the dataset
data = pd.read_csv('google_play_store_apps_reviews_training.csv')
# Preprocess the data
def preprocess_data(data):
  data = data.drop('package_name', axis=1)
  data['review'] = data['review'].str.strip().str.lower()
  return data
data = preprocess_data(data)
x = data['review']
```

```
y = data['polarity']
x, x_test, y, y_test = train_test_split(x, y, stratify=y,
test_size=0.25, random_state=42)
from sklearn.feature_extraction.text import CountVectorizer
vec = CountVectorizer(stop_words='english')
x = vec.fit_transform(x).toarray()
x_test = vec.transform(x_test).toarray()
from sklearn.naive_bayes import MultinomialNB

model = MultinomialNB()
model.fit(x, y)
accuracy = model.score(x_test, y_test)
print(f'Model Accuracy: {accuracy * 100:.2f}%')
prediction = model.predict(vec.transform(['useless app!']))
print(f'Prediction: {prediction[0]}')
```

7. Explain the advantages of using LSTM over RNN for sentiment classification?

Long Short-Term Memory (LSTM) networks offer several advantages over traditional Recurrent Neural Networks (RNNs) for sentiment classification tasks. RNNs are designed to process sequential data and maintain a hidden state that captures information from previous time steps. However, they suffer from the **vanishing and exploding gradient problem**, which makes it difficult to learn long-term dependencies in text sequences. As a result, RNNs tend to forget important words that appeared earlier in a sentence, which can lead to incorrect sentiment predictions. For example, in the sentence *"The movie was not good,"* a simple RNN may focus on the word "good" and predict a positive sentiment, forgetting that the word "not" changes the meaning to negative.

LSTMs overcome this limitation by introducing a **memory cell** and three control mechanisms called **gates** — the forget gate, input gate, and output gate — which regulate the flow of information through time. This gated architecture allows LSTMs to selectively retain or discard information, effectively capturing both short-term and long-term dependencies in text. Consequently, they can remember contextual clues like negations ("not good"), contrasting clauses ("although the plot was slow, the ending was great"), and subtle shifts in tone or emotion. This

ability to preserve contextual meaning across longer sequences leads to **more accurate sentiment classification**, especially in complex sentences where word relationships are spread far apart.

Moreover, LSTMs maintain more stable training compared to RNNs since the gating mechanism prevents gradient vanishing. They can process longer and variable-length texts while retaining relevant emotional cues. In practice, models using LSTMs consistently outperform traditional RNNs in sentiment analysis because they better understand word order, handle negations effectively, and capture the overall sentiment flow of a sentence. In summary, while RNNs are suitable for simple sequential data, LSTMs provide a significant improvement for sentiment classification by retaining long-term dependencies, understanding context, and delivering higher accuracy.

LSTM Improves Sentiment Classification

(a) Retains Long-Term Dependencies

LSTM can remember important words from earlier in the sentence.

Example: "Although the plot was slow, the ending was great."
LSTM understands that "ending was great" dominates the overall sentiment.

(b) Handles Negations and Context Shifts

Example: "The movie was not bad."
A basic RNN might see "bad" → negative.
LSTM keeps "not" in memory → learns overall meaning = positive.

(c) Better Gradient Flow

LSTM uses **cell state** and **gates** to control what information flows forward or fades out — preventing gradient vanishing.

(d) Captures Sequential Patterns

Sentiment often depends on **word order**.

"Hardly enjoyable" ≠ "Enjoyable hardly ever."
LSTM recognizes such patterns better than vanilla RNNs.

(e) Robust to Longer Texts

Reviews or tweets vary in length.
LSTMs handle both short and long sentences effectively due to their dynamic memory management.

8. Describe the fundamentals of Sentiment Analysis-Positive, Negative, Neutral Sentiments.

Sentiment analysis, or opinion mining, is the process of analyzing large volumes of text to determine whether it expresses a positive sentiment, a negative sentiment or a neutral sentiment.

Sentiment analysis involves several stages — from cleaning the text data to classifying it into positive, negative, or neutral categories.
The main steps are:

1**Text Preprocessing**

2 **Feature Extraction**

3 **Sentiment Classification**

1. Text Preprocessing

Before analyzing sentiments, the raw text data must be cleaned and standardized.
This step improves the quality and accuracy of the model.

a. Cleaning the text

- **Remove punctuation:** Characters like , . ! ? do not affect sentiment directly.
  Example: *"I love this movie!" → "I love this movie"*

- **Remove stop words:** Common words (like *is, the, and, of*) that don't contribute to sentiment.

- **Remove special characters/numbers:** To focus only on meaningful text.

b. Tokenization

- Splitting text into smaller units called **tokens** (words or phrases).
  Example:
  "I love this movie" → ["I", "love", "this", "movie"]

c. Normalization

- **Lowercasing:** Converts all words to lowercase → ensures uniformity.
  Example: *"Love"* and *"love"* are treated the same.

- **Stemming:** Reduces words to their **root form** by chopping endings.
  Example: *"loved", "loving" → "love"*

- **Lemmatization:** Converts words to their **dictionary base form** (more accurate).
  Example: *"better" → "good", "running" → "run"*

2. Feature Extraction

After preprocessing, the text is converted into numerical format so that machine learning models can process it.

Techniques:

1. **Bag of Words (BoW):**

- Represents text as a set of word counts (frequency of each word).

- Ignores grammar and order.
  Example:
  "I love movies" → {I:1, love:1, movies:1}

2. **TF-IDF (Term Frequency – Inverse Document Frequency):**

   - Weights words based on how important they are to a document.

   - Common words (like "the") get less weight; rare but important words get more.

3. **Word Embeddings (Semantic Representations):**

   - Captures **meaning and context** of words.

   - Each word is represented as a dense vector.

   - Examples:

     - **Word2Vec** and **GloVe**: Capture semantic similarity.

     - **BERT**: Context-aware embeddings; considers the position and meaning of words in sentences.

3. Sentiment Classification

Once the text is represented numerically, models are trained to **classify** the sentiment as **positive**, **negative**, or **neutral**.

Common Models:

1. **Traditional Machine Learning Models:**
    - **Naïve Bayes:** Uses probability based on word frequencies.
    - **Logistic Regression:** Assigns weights to features and predicts the sentiment.
    - **SVM (Support Vector Machine):** Finds boundaries separating different sentiment classes.
2. **Deep Learning Models:**
    - **LSTM (Long Short-Term Memory):** Captures sequence and context in text (useful for long sentences).
    - **BERT (Bidirectional Encoder Representations from Transformers):**
        - Understands both left and right context.
        - Achieves state-of-the-art performance in sentiment analysis tasks.

9. What is word embeddings? Explain how to generate word embeddings using BERT?

Word embeddings are numerical vector representations of words in a continuous, multi-dimensional space. These vectors capture the semantic and syntactic relationships between words, enabling machines to process and understand text more effectively. Unlike traditional methods like one-hot encoding, which are sparse and fail to capture word relationships, word embeddings are dense and encode meaningful patterns based on the context in which words appear.

Need of word embeddings

- Similar words have similar vectors
- Dimensions are low
- Dense Representation
- Arithmetic operations can be performed

Word Embeddings using BERT:

BERT (Bidirectional Encoder Representations from Transformers) is a deep learning model developed by Google in 2018 for Natural Language Processing (NLP). It introduced a revolutionary approach to language representation by learning contextual embeddings in a bidirectional way. Unlike previous models such as Word2Vec or GloVe, which generate a single fixed vector for each word, BERT's embeddings vary depending on the context in which a word appears.

Traditional language models are unidirectional — they read text either left-to-right or right-to-left. BERT, however, reads text in both directions simultaneously using Transformer encoders. This bidirectional nature allows it to understand the full context of a word based on all its surrounding words.

BERT is based on the Transformer architecture, which uses the mechanism of self-attention to capture dependencies between words in a sentence, regardless of their distance. The Transformer model consists of an encoder and a decoder; BERT uses only the encoder stack, making it suitable for understanding tasks such as classification, question answering, and sentence similarity.

BERT uses a subword tokenization algorithm called WordPiece. This helps handle rare or unknown words by breaking them into smaller subword units. For example, the word 'dancing' might be split as ['dan', '##cing'], where '##' indicates that the subword is part of a larger word. This ensures that even unfamiliar words can be represented meaningfully based on their components.

WordPiece starts with a small base vocabulary and merges frequently occurring character pairs to form subwords. The frequency of pairs determines their merging

priority. This reduces the vocabulary size while maintaining coverage of all words in a corpus.

Pre-training Objectives of BERT

BERT is pre-trained using two unsupervised tasks: Masked Language Modeling (MLM) and Next Sentence Prediction (NSP).

Masked Language Modeling (MLM)

The MLM objective helps BERT learn deep bidirectional context representations. During pre-training, 15% of the input tokens are randomly masked. The model then predicts these masked tokens based on both the left and right context. This enables BERT to understand how words relate to one another across the entire sentence.

Example:

[CLS] The quick brown [MASK] jumps over the lazy dog [SEP]

Here, BERT predicts the masked token as 'fox'.

Unlike traditional left-to-right models, BERT's bidirectional approach allows it to use both preceding and following words to predict the masked term.

Next Sentence Prediction (NSP)

The NSP task teaches BERT about the relationships between sentences — useful for tasks like question answering and text coherence. During training, BERT is given pairs of sentences (A and B) and asked to predict whether B follows A in the original text.

- 50% of the time, B is the actual next sentence (labeled 'IsNext').
- 50% of the time, B is a random sentence (labeled 'NotNext').

This task allows BERT to learn inter-sentence relationships and logical flow, which is crucial for higher-level NLP applications.

BERT uses special tokens to structure its input properly:

• [CLS] (Classification Token): Added at the start of every input sequence. The final hidden state corresponding to this token is used for classification tasks, such as sentiment analysis.
• [SEP] (Separator Token): Used to separate two sentences or indicate the end of a single sentence. Essential for distinguishing between input segments in NSP and QA tasks.

Input format for a pair of sentences:
[CLS] Sentence A [SEP] Sentence B [SEP]

BERT's input representation is a combination of three types of embeddings:
• Token Embeddings: Represent each word or subword.
• Segment Embeddings: Indicate whether a token belongs to sentence A or B.
• Position Embeddings: Capture the position of each token in the sequence to retain order information.

After pre-training, BERT can be fine-tuned for specific NLP tasks by adding a small output layer. The same pre-trained parameters are slightly adjusted (fine-tuned) based on the labeled dataset of the target task.

10. Discuss word2vec with code snippets in detail.

•provides Feature Representation

•Uses a neural network model to learn word associations from a large corpus of text.

•A 2 layer neural network to generate word embeddings given in a text corpus.

•Once trained such a model can detect synonymous words or suggest additional words for a particular sentence

Word2vec is needed to provide

•Preserves relationship between words.

•Deals with addition of new words in the vocabulary.

•Better results in lots of deep learning applications

CBOW Working:

•We generate one hot word vectors corresponding to the context

•These vectors are embedded using n dimensions say 300

$$\left(v_{c-m} = \gamma x^{(c-m)}, v_{c-m+1} = \gamma x^{(c-m+1)}, \ldots, v_{c+m} = \gamma x^{(c+m)}\right)$$

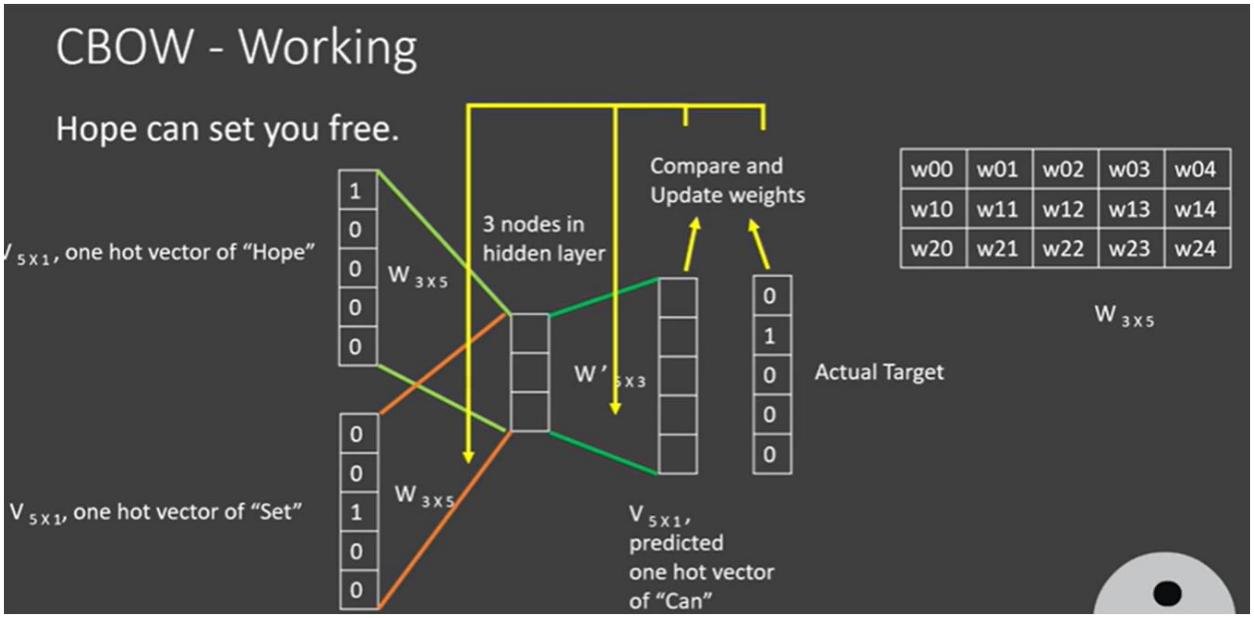
•The context vectors are averaged before using in prediction.

$$\hat{v} = \frac{v_{c-m} + v_{c-m+1} + \ldots + v_{c+m}}{2m}$$

•Generate a score vector z= U^V

•Turn the scores into probabilities using softmax(z)

•Match the the probabilities generated with true probabilities



Skip Gram

•It is required to predict if candidate word c is a neighbor of a given target word t

•The target word t and a neighboring context word c are treated as positive examples.

•Now other words in the lexicon are sampled randomly to obtain negative examples.

•Then logistic regression is used to train a classifier to distinguish the two types of cases.

•The learned weights are used as embeddings.