KESHAV MEMORIAL INSTITUTE OF TECHNOLOGY
(AN AUTONOMOUS INSTITUTE)
Approved by AICTE, Affiliated to JNTUH, Hyderabad-500029
III- B. Tech I Semester-KR23

ARTIFICIAL INTELLIGENCE-(DEEPLEARNING)

PREDICTION OF NEXT WORD USING LSTM

PROBLEM STATEMENT:

To develop and train an LSTM-based neural network capable of predicting the next word in a given text sequence using natural language data.

LEARNING OBJECTIVES

- Understand how LSTM networks model sequential dependencies.
- Learn the process of tokenization and text preprocessing for NLP tasks.
- Implement word-level sequence modeling using LSTM.
- Evaluate the model's performance and interpret next-word predictions.

SOFTWARE REQUIREMENTS

- Python 3.x
- TensorFlow / Keras
- NumPy
- Matplotlib
- Jupyter Notebook

THEORY

A Recurrent Neural Network (RNN) learns from sequential data but struggles to capture long-term dependencies due to vanishing gradients.
To solve this, LSTM (Long Short-Term Memory) introduces a cell state and gating mechanisms:

- Forget Gate: decides what past information to discard.

- Input Gate: decides what new information to store.
- Output Gate: decides what to output from the cell.

In a Next Word Prediction task, the LSTM learns the likelihood of the next word based on preceding words.

Example:

Input Sequence: “Deep learning models are”

Predicted Next Word: “powerful”

Each word is integer-encoded and passed through an Embedding Layer that maps it to dense vectors capturing semantic meaning.
An LSTM layer processes these embeddings over time, and a Dense Softmax layer predicts the probability distribution for the next word.

PROCEDURE:

STEP 1: Import Required Libraries

```
import numpy as np

from keras.preprocessing.text import Tokenizer

from keras.preprocessing.sequence import pad_sequences

from keras.utils import to_categorical

from keras.models import Sequential

from keras.layers import Embedding, LSTM, Dense
```

These libraries handle tokenization, padding, one-hot encoding, and model creation using Keras’ Sequential API.

STEP 2: Prepare Text Data

```
data = """Deep learning allows machines to learn from data.

It helps in predicting future outcomes.

Deep learning models are powerful and flexible."""
```

A small text corpus is used. The model learns from sentence patterns and tries to predict the next word.

STEP 3: Tokenize the Text

```
tokenizer = Tokenizer()

tokenizer.fit_on_texts([data])

encoded = tokenizer.texts_to_sequences([data])[0]

vocab_size = len(tokenizer.word_index) + 1

print("Vocabulary Size:", vocab_size)
```

Output:

```
Vocabulary Size: 17
```

The text is split into unique words and assigned numeric tokens. The vocabulary size is the total count of unique words plus one (for padding).

STEP 4: Create Sequences for Training

```
sequences = []

for i in range(1, len(encoded)):

    sequence = encoded[:i+1]

    sequences.append(sequence)

print("Total Sequences:", len(sequences))
```

Output:

```
Total Sequences: 16
```

Each sequence is a progressively increasing set of words. For example: [‘Deep’, ‘learning’] → Predict next word.

STEP 5: Pad Sequences and Split Inputs/Labels

```
max_length = max([len(seq) for seq in sequences])

sequences = pad_sequences(sequences, maxlen=max_length, padding='pre')

sequences = np.array(sequences)

X, y = sequences[:, :-1], sequences[:, -1]

y = to_categorical(y, num_classes=vocab_size)
```

All sequences are padded to the same length for batch training.

Each input sequence (X) predicts one output word (y).

STEP 6: Define the LSTM Model

```
model = Sequential()

model.add(Embedding(vocab_size, 10, input_length=max_length-1))

model.add(LSTM(100))

model.add(Dense(vocab_size, activation='softmax'))

model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])

model.summary()
```

Output:

```
Model: "sequential"

_________________________________________________________________

 Layer (type)            Output Shape          Param #

=================================================================

 embedding (Embedding)     (None, 5, 10)         170

 lstm (LSTM)              (None, 100)          44400

 dense (Dense)             (None, 17)          1717

Total params: 46,287
```

Trainable params: 46,287

- **Embedding Layer:** Converts tokens into 10-dimensional dense vectors.
- **LSTM Layer:** Learns sequence relationships with 100 memory units.
- **Dense Layer:** Predicts probabilities for each word in the vocabulary.

STEP 7: Train the Model

```
model.fit(X, y, epochs=500, verbose=2)
```

Output (sample):

```
Epoch 1/500 - loss: 2.83 - accuracy: 0.20

Epoch 100/500 - loss: 1.01 - accuracy: 0.75

Epoch 500/500 - loss: 0.20 - accuracy: 1.00
```

The loss decreases while accuracy improves over epochs, showing successful learning.

STEP 8: Predict the Next Word

```
def predict_next_word(model, tokenizer, text, max_length):
    encoded = tokenizer.texts_to_sequences([text])[0]
    encoded = pad_sequences([encoded], maxlen=max_length-1,
padding='pre')
    y_pred = model.predict(encoded, verbose=0)
    predicted_word = ''
    for word, index in tokenizer.word_index.items():
        if index == np.argmax(y_pred):
            predicted_word = word
            break
    print(text + ' → ' + predicted_word)
```

Example Predictions:

```
predict_next_word(model, tokenizer, "Deep learning", max_length)

predict_next_word(model, tokenizer, "learning models", max_length)
```

Output:

```
Deep learning → allows

learning models → are
```

The model predicts the most probable next word based on learned context.

MODEL EVALUATION METRICS

Training Accuracy: Measures correct predictions per epoch.

Loss: Measures model error (should decrease over time).

Observation:

- Accuracy rises steadily, reaching near 100% with small datasets.
- Loss converges to a minimal value, indicating stable learning.

Sample Log:

```
Epoch 1/500 - loss: 2.83 - accuracy: 0.20

Epoch 250/500 - loss: 0.60 - accuracy: 0.89

Epoch 500/500 - loss: 0.20 - accuracy: 1.00
```

OBSERVATION / VIVA QUESTIONS

1. What is the function of the Embedding layer in LSTM?
2. Why do we need padding for variable-length sequences?
3. Explain the role of the softmax activation in the output layer.
4. How does the Forget Gate help LSTM learn better than a basic RNN?
5. What is the importance of tokenization in text preprocessing?
6. Why is categorical_crossentropy used as the loss function?
7. What would happen if we increase the embedding dimension?
8. Define vanishing gradients in the context of RNNs.
9. Implement the same model using simple RNN , Plot training loss vs epochs for both models and mark the convergence rate.Observe which model's loss decreases steadily and generalises better.