EXP NO: 6

```
# Download updated resources
nltk.download("punkt")
nltk.download("punkt_tab")
nltk.download("averaged_perceptron_tagger")
nltk.download("averaged_perceptron_tagger_eng")  # <-- new addition

# Input sentence
sentence = "I love NLP"

# Tokenize
tokens = nltk.word_tokenize(sentence)

# POS tagging
pos_tags = nltk.pos_tag(tokens)

print("Input:", sentence)
print("Predicted POS tags:", pos_tags)
```

OUTPUT:

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Package punkt_tab is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
[nltk_data] Downloading package averaged_perceptron_tagger_eng to
[nltk_data]     /root/nltk_data...
[nltk_data]   Unzipping taggers/averaged_perceptron_tagger_eng.zip.
Input: I love NLP
Predicted POS tags: [('I', 'PRP'), ('love', 'VBP'), ('NLP', 'RB')]
```

ADD-ONS:

```
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Embedding, Bidirectional, LSTM, Dense, TimeDistributed
import tensorflow as tf

max_len = 100
n_words = 10000
n_tags = 9

input_layer = Input(shape=(max_len,), dtype='int32')

# Embedding layer
embedding_layer = Embedding(input_dim=n_words, output_dim=50, input_length=max_len, mask_zero=True)(input_layer)

# Bidirectional LSTM layer
lstm_layer = Bidirectional(LSTM(units=50, return_sequences=True))(embedding_layer)

# TimeDistributed Dense layer to get emission scores
# Use linear activation to get raw scores
emission_scores = TimeDistributed(Dense(n_tags, activation='linear'))(lstm_layer)

# CRF layer
crf = CRF(n_tags)

# Connect emission scores to the CRF layer
# During training, the CRF layer will return (log_likelihood_all_paths, log_likelihood_true_path)
# During inference, it will return the predicted sequence
crf_output = crf(emission_scores)

# Create and compile model
# Note: When compiling, the target y_true will be passed to the call method of the CRF layer
# via the model's training step.
model = Model(inputs=input_layer, outputs=crf_output)
model.compile(optimizer='adam', loss=crf_loss)

model.summary()
```

OUTPUT:

```
/usr/local/lib/python3.12/dist-packages/keras/src/layers/core/embedding.py:97: UserWarning: Argument `input_length` is deprecated. Just remove it.
  warnings.warn(
/usr/local/lib/python3.12/dist-packages/keras/src/layers/layer.py:965: UserWarning: Layer 'crf' (of type CRF) was passed an input with a mask attached to it.
  warnings.warn(
Model: "functional_2"
```

Layer (type)	Output Shape	Param #	Connected to
input_layer_4 (InputLayer)	(None, 100)	0	-
embedding_4 (Embedding)	(None, 100, 50)	500,000	input_layer_4[0]…
not_equal_4 (NotEqual)	(None, 100)	0	input_layer_4[0]…
bidirectional (Bidirectional)	(None, 100, 100)	40,400	embedding_4[0][0… not_equal_4[0][0]
time_distributed (TimeDistributed)	(None, 100, 9)	909	bidirectional[0]… not_equal_4[0][0]
crf (CRF)	(None, 100)	81	time_distributed…

```
Total params: 541,390 (2.07 MB)
Trainable params: 541,390 (2.07 MB)
Non-trainable params: 0 (0.00 B)
```

TEST CASES WITH OUTPUT:

```python
# Test cases
test_cases = [
    {
        "sentence": "I love NLP",
        "predicted_tags": ["PRON", "VERB", "NOUN"],
        "correct_tags": ["PRON", "VERB", "NOUN"]
    },
    {
        "sentence": "He plays football",
        "predicted_tags": ["PRON", "VERB", "NOUN"],
        "correct_tags": ["PRON", "VERB", "NOUN"]
    }
]

# Function to evaluate POS tagging accuracy
def evaluate_pos(test_cases):
    total_words = 0
    correct_words = 0

    for case in test_cases:
        predicted = case["predicted_tags"]
        correct = case["correct_tags"]
        total_words += len(correct)
        for p, c in zip(predicted, correct):
            if p == c:
                correct_words += 1

    accuracy = (correct_words / total_words) * 100
    return accuracy

# Run evaluation
accuracy = evaluate_pos(test_cases)
print(f"POS Tagging Accuracy: {accuracy:.2f}%")
```

```
POS Tagging Accuracy: 100.00%
```