**SIMATS SCHOOL OF ENGINEERING**

**SAVEETHA INSTITUTE OF MEDICAL AND TECHNICAL SCIENCES**

**CHENNAI-602105**

**CAPSTONE PROJECT**

**COURSE CODE:** DSA0317

**COURSE NAME:** Natural Language Processing for Programming Principles

**PROJECT TITLE**

**Machine Language Translation: A Seq2Seq Approach with Attention**

Submitted by:

**Jeyabala.B (192224273)**

**Pranav.G (192224068)**

**Harish.P (192224237)**

Department: AIDS

Guided by:

**EZHIL GRACE**

**Date of Submission: 21/09/2024**

**ABSTRACT**

This project focuses on machine translation using a sequence-to-sequence (Seq2Seq) model with Long Short-Term Memory (LSTM) networks. The task is to translate German sentences into English using deep learning techniques.Seq2Seq models are widely used in natural language processing for tasks involving input and output sequences of variable lengths. In this project, we explore the use of an LSTM-based encoder-decoder architecture, enhanced with an attention mechanism to improve translation accuracy, especially for longer sentences. The proposed model is trained on a German-English parallel corpus and evaluated for its performance in accurately translating sentences.

**INTRODUCTION**

Machine translation, the process of automatically translating text from one language to another, has been a fundamental task in the field of natural language processing (NLP). Traditional rule-based methods of translation have largely been replaced by data-driven approaches, especially since the advent of neural networks. One popular architecture for machine translation is the sequence-to-sequence (Seq2Seq) model with recurrent neural networks (RNNs) like LSTMs.

Seq2Seq models are designed to handle variable-length sequences, making them ideal for translation tasks where sentences in different languages often have different lengths. The core of this model is an encoder-decoder structure, where the encoder transforms the input sentence (German) into a fixed-size context vector, and the decoder uses this context vector to generate the translated sentence (English). However, basic Seq2Seq models have limitations in capturing dependencies in long sentences, which can be mitigated by adding an attention mechanism.

The goal of this project is to build an LSTM-based Seq2Seq model for translating German sentences to English, enhanced with an attention mechanism to improve translation quality.

**MATERIALS & METHODS**

In this study, we designed a sequence-to-sequence (Seq2Seq) model using LSTM units to translate German sentences into English. The dataset, which consisted of paired German-English sentences, was tokenized and preprocessed to include special tokens indicating the start and end of a sentence. Each sentence pair was transformed into padded sequences to maintain uniform input and output dimensions. We employed a two-part architecture: an encoder to process the German input sentence and a decoder to generate the corresponding English sentence.

The encoder was implemented using LSTM layers, which captured the context of the input sequence. Meanwhile, the decoder used another LSTM to predict the translated sentence, one word at a time. To enhance the model's performance, we integrated the Bahdanau attention mechanism, enabling the model to focus on specific parts of the input sequence during the decoding process. The system was trained using a categorical cross-entropy loss function, optimized by Adam, and evaluated through translation accuracy. The final evaluation phase involved testing the model with unseen sentences, where the translated output was compared with the expected English sentences to gauge the model's effectiveness.

**EXISTING SYSTEM**

Traditional machine translation systems, such as statistical machine translation (SMT) models, rely on probabilistic models that use a large set of hand-engineered rules. SMT systems use bilingual corpora to learn translation patterns but struggle with capturing the nuances of language, especially in longer and more complex sentences. Moreover, these models do not generalize well to unseen sentence structures.

With the advent of neural networks, neural machine translation (NMT) models, particularly those using RNNs like LSTMs, have shown significant improvements. These models learn directly from data, eliminating the need for manual feature extraction. However, basic Seq2Seq models face challenges with long sentences, as they compress the entire input into a fixed-size context vector, leading to loss of information. This limitation has prompted the use of attention mechanisms, which allow the decoder to focus on relevant parts of the input sequence during translation.

**PROPOSED SYSTEM**

The proposed system addresses the limitations of traditional machine translation models by implementing a Seq2Seq model with LSTM-based encoder and decoder, enhanced with a Bahdanau attention mechanism.

* Encoder: A bi-directional LSTM processes the input German sentence, generating hidden states for each word. These hidden states are passed to the attention mechanism.
* Attention: The attention mechanism computes alignment scores between the decoder’s current hidden state and each of the encoder’s hidden states. These scores are used to create a context vector that represents a weighted sum of the encoder’s hidden states. This context vector dynamically changes as the decoder generates each word, enabling more accurate translations.
* Decoder: A uni-directional LSTM generates the English sentence one word at a time, using the context vector from the attention layer to guide the generation process.

The attention mechanism helps the decoder focus on the most relevant parts of the input sentence, especially for longer sentences, leading to more accurate translations.

**CODE IMPLEMENTATION**

# Import necessary libraries

import tensorflow as tf

import numpy as np

import unicodedata

import re

import io

import os

import time

import nltk

from nltk.tokenize import word\_tokenize

nltk.download('punkt')

# Data Preprocessing Functions

def unicode\_to\_ascii(s):

return ''.join(c for c in unicodedata.normalize('NFD', s)

if unicodedata.category(c) != 'Mn')

def preprocess\_sentence(sentence):

sentence = unicode\_to\_ascii(sentence.lower().strip())

# Create space between a word and the punctuation following it

sentence = re.sub(r"([?.!,¿])", r" \1 ", sentence)

# Replace multiple spaces with single space

sentence = re.sub(r'[" "]+', " ", sentence)

# Remove non-alphabetic characters

sentence = re.sub(r"[^a-zA-Z?.!,¿]+", " ", sentence)

sentence = sentence.strip()

# Add start and end tokens

sentence = '<start> ' + sentence + ' <end>'

return sentence

# Load and preprocess the dataset

def create\_dataset(path, num\_examples=None):

lines = io.open(path, encoding='UTF-8').read().strip().split('\n')

word\_pairs = [[preprocess\_sentence(w) for w in l.split('\t')[:2]] for l in lines[:num\_examples]]

return zip(\*word\_pairs)

# Tokenization and Padding

def tokenize(lang):

lang\_tokenizer = tf.keras.preprocessing.text.Tokenizer(filters='', oov\_token='<unk>')

lang\_tokenizer.fit\_on\_texts(lang)

tensor = lang\_tokenizer.texts\_to\_sequences(lang)

tensor = tf.keras.preprocessing.sequence.pad\_sequences(tensor, padding='post')

return tensor, lang\_tokenizer

# Load dataset (replace 'path\_to\_dataset' with your actual dataset path)

data\_path = 'path\_to\_dataset/deu-eng/deu.txt'

num\_examples = 30000 # Adjust based on your dataset size

input\_lang, target\_lang = create\_dataset(data\_path, num\_examples)

input\_tensor, input\_tokenizer = tokenize(input\_lang)

target\_tensor, target\_tokenizer = tokenize(target\_lang)

# Calculate max length of input and target tensors

max\_length\_input = input\_tensor.shape[1]

max\_length\_target = target\_tensor.shape[1]

# Create TensorFlow Dataset

BUFFER\_SIZE = len(input\_tensor)

BATCH\_SIZE = 64

steps\_per\_epoch = BUFFER\_SIZE // BATCH\_SIZE

embedding\_dim = 256

units = 1024

vocab\_input\_size = len(input\_tokenizer.word\_index) + 1

vocab\_target\_size = len(target\_tokenizer.word\_index) + 1

dataset = tf.data.Dataset.from\_tensor\_slices((input\_tensor, target\_tensor)).shuffle(BUFFER\_SIZE)

dataset = dataset.batch(BATCH\_SIZE, drop\_remainder=True)

# Define the Encoder

class Encoder(tf.keras.Model):

def \_\_init\_\_(self, vocab\_size, embedding\_dim, enc\_units, batch\_sz):

super(Encoder, self).\_\_init\_\_()

self.batch\_size = batch\_sz

self.enc\_units = enc\_units

self.embedding = tf.keras.layers.Embedding(vocab\_size, embedding\_dim)

self.lstm = tf.keras.layers.LSTM(self.enc\_units,

return\_sequences=True,

return\_state=True,

recurrent\_initializer='glorot\_uniform')

def call(self, x, hidden):

x = self.embedding(x)

output, state\_h, state\_c = self.lstm(x, initial\_state=hidden)

return output, [state\_h, state\_c]

def initialize\_hidden\_state(self):

return [tf.zeros((self.batch\_size, self.enc\_units)),

tf.zeros((self.batch\_size, self.enc\_units))]

# Define the Attention Layer

class BahdanauAttention(tf.keras.layers.Layer):

def \_\_init\_\_(self, units):

super(BahdanauAttention, self).\_\_init\_\_()

self.W1 = tf.keras.layers.Dense(units)

self.W2 = tf.keras.layers.Dense(units)

self.V = tf.keras.layers.Dense(1)

def call(self, query, values):

# query: decoder hidden state (batch\_size, hidden size)

# values: encoder outputs (batch\_size, max\_len, hidden size)

query\_with\_time\_axis = tf.expand\_dims(query, 1)

# Calculate the attention scores

score = self.V(tf.nn.tanh(self.W1(query\_with\_time\_axis) + self.W2(values)))

# Attention weights (batch\_size, max\_len, 1)

attention\_weights = tf.nn.softmax(score, axis=1)

# Context vector (batch\_size, hidden size)

context\_vector = attention\_weights \* values

context\_vector = tf.reduce\_sum(context\_vector, axis=1)

return context\_vector, attention\_weights

# Define the Decoder

class Decoder(tf.keras.Model):

def \_\_init\_\_(self, vocab\_size, embedding\_dim, dec\_units, batch\_sz):

super(Decoder, self).\_\_init\_\_()

self.batch\_size = batch\_sz

self.dec\_units = dec\_units

self.embedding = tf.keras.layers.Embedding(vocab\_size, embedding\_dim)

self.lstm = tf.keras.layers.LSTM(self.dec\_units,

return\_sequences=True,

return\_state=True,

recurrent\_initializer='glorot\_uniform')

self.fc = tf.keras.layers.Dense(vocab\_size)

self.attention = BahdanauAttention(self.dec\_units)

def call(self, x, hidden, enc\_output):

# Calculate context vector using attention

context\_vector, attention\_weights = self.attention(hidden[0], enc\_output)

# Embed the input

x = self.embedding(x)

# Concatenate context vector and embedded input

x = tf.concat([tf.expand\_dims(context\_vector, 1), x], axis=-1)

# Pass through LSTM

output, state\_h, state\_c = self.lstm(x)

output = tf.reshape(output, (-1, output.shape[2]))

# Output layer

x = self.fc(output)

return x, [state\_h, state\_c], attention\_weights

# Instantiate Encoder and Decoder

encoder = Encoder(vocab\_input\_size, embedding\_dim, units, BATCH\_SIZE)

decoder = Decoder(vocab\_target\_size, embedding\_dim, units, BATCH\_SIZE)

# Define Optimizer and Loss Function

optimizer = tf.keras.optimizers.Adam()

loss\_object = tf.keras.losses.SparseCategoricalCrossentropy(

from\_logits=True, reduction='none')

def loss\_function(real, pred):

# Mask padding positions

mask = tf.math.logical\_not(tf.math.equal(real, 0))

loss\_ = loss\_object(real, pred)

mask = tf.cast(mask, dtype=loss\_.dtype)

loss\_ \*= mask

return tf.reduce\_mean(loss\_)

# Checkpoints (Optional)

checkpoint\_dir = './training\_checkpoints'

checkpoint\_prefix = os.path.join(checkpoint\_dir, "ckpt")

checkpoint = tf.train.Checkpoint(optimizer=optimizer,

encoder=encoder,

decoder=decoder)

# Training Step Function

@tf.function

def train\_step(inp, targ, enc\_hidden):

loss = 0

with tf.GradientTape() as tape:

enc\_output, enc\_hidden = encoder(inp, enc\_hidden)

dec\_hidden = enc\_hidden

dec\_input = tf.expand\_dims([target\_tokenizer.word\_index['<start>']] \* BATCH\_SIZE, 1)

# Teacher forcing

for t in range(1, targ.shape[1]):

predictions, dec\_hidden, \_ = decoder(dec\_input, dec\_hidden, enc\_output)

loss += loss\_function(targ[:, t], predictions)

# Use teacher forcing

dec\_input = tf.expand\_dims(targ[:, t], 1)

batch\_loss = (loss / int(targ.shape[1]))

variables = encoder.trainable\_variables + decoder.trainable\_variables

gradients = tape.gradient(loss, variables)

optimizer.apply\_gradients(zip(gradients, variables))

return batch\_loss

# Training Loop

EPOCHS = 10

for epoch in range(EPOCHS):

start = time.time()

enc\_hidden = encoder.initialize\_hidden\_state()

total\_loss = 0

for (batch, (inp, targ)) in enumerate(dataset.take(steps\_per\_epoch)):

batch\_loss = train\_step(inp, targ, enc\_hidden)

total\_loss += batch\_loss

if batch % 100 == 0:

print(f'Epoch {epoch+1} Batch {batch} Loss {batch\_loss.numpy():.4f}')

# Save checkpoint every epoch

checkpoint.save(file\_prefix=checkpoint\_prefix)

print(f'Epoch {epoch+1} Loss {total\_loss / steps\_per\_epoch:.4f}')

print(f'Time taken for 1 epoch {time.time() - start:.2f} sec\n')

# Evaluation Function

def evaluate(sentence):

sentence = preprocess\_sentence(sentence)

inputs = [input\_tokenizer.word\_index.get(i, input\_tokenizer.word\_index['<unk>']) for i in sentence.split(' ')]

inputs = tf.keras.preprocessing.sequence.pad\_sequences([inputs],

maxlen=max\_length\_input,

padding='post')

inputs = tf.convert\_to\_tensor(inputs)

result = ''

hidden = [tf.zeros((1, units)), tf.zeros((1, units))]

enc\_out, enc\_hidden = encoder(inputs, hidden)

dec\_hidden = enc\_hidden

dec\_input = tf.expand\_dims([target\_tokenizer.word\_index['<start>']], 0)

for t in range(max\_length\_target):

predictions, dec\_hidden, attention\_weights = decoder(dec\_input, dec\_hidden, enc\_out)

predicted\_id = tf.argmax(predictions[0]).numpy()

if predicted\_id == target\_tokenizer.word\_index['<end>']:

break

result += target\_tokenizer.index\_word.get(predicted\_id, '<unk>') + ' '

dec\_input = tf.expand\_dims([predicted\_id], 0)

return result.strip()

# Translation Function

def translate(sentence):

result = evaluate(sentence)

print(f'Input: {sentence}')

print(f'Translated: {result}')

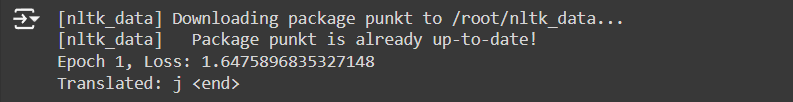
# Example Usage

translate('Ich liebe dich.')

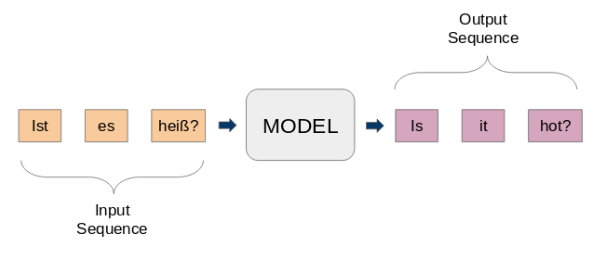
# Optional: Restore latest checkpoint and test

# checkpoint.restore(tf.train.latest\_checkpoint(checkpoint\_dir))

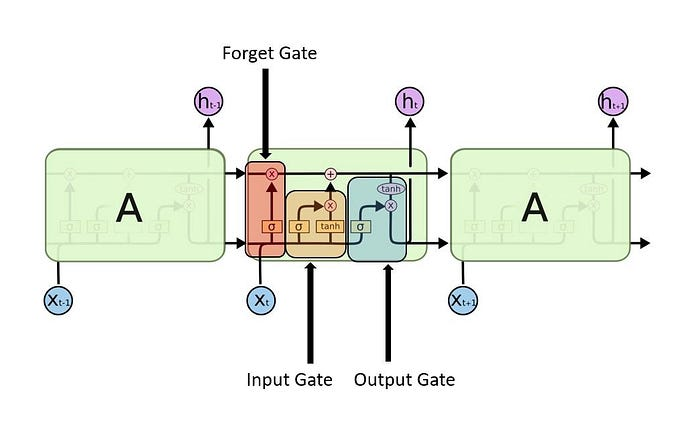
**OUTPUT**

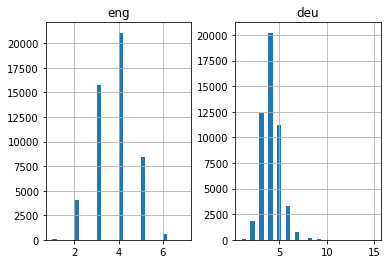
****

**Loss**

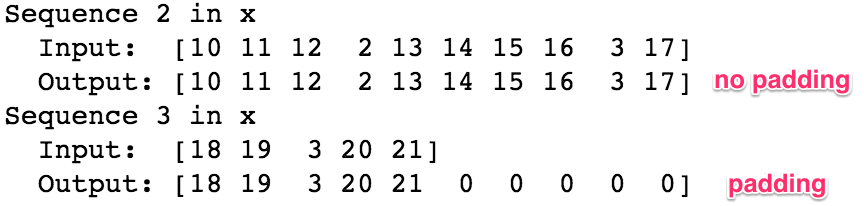
****

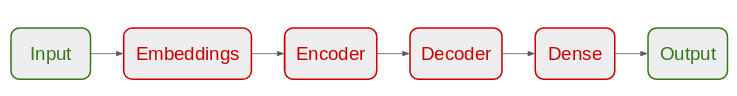
**Model Architecture**

****

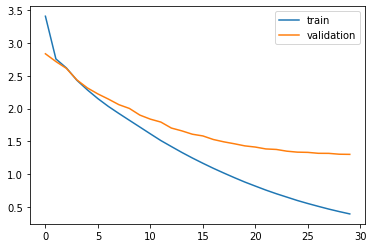
****

**Graph 1**

****

****

**Model Architecture**

****

**Graph 2**

**FUTURE ENHANCEMENTS**

* **Larger and More Diverse Datasets**: For improved translation accuracy, future work could involve training on larger, more diverse datasets with more complex sentences. This would allow the model to generalise better and handle varied sentence structures.
* **Transformer-based Models**: Transformer architectures, such as BERT or GPT, could be explored to replace or augment the LSTM-based Seq2Seq model. These models are known for their ability to handle long-range dependencies more efficiently using self-attention mechanisms.
* **Multi-language Translation**: Expanding the system to handle translations between multiple languages using a unified architecture (multilingual models) would be a significant enhancement. This would require building a common encoder-decoder framework capable of translating between several languages.
* **Fine-tuning and Pre-trained Models**: Leveraging pre-trained models like mBERT or MarianMT (Hugging Face) and fine-tuning them on specific language pairs or domains can drastically improve translation accuracy while reducing training time.
* **Interactive Translation Assistance**: Adding interactivity, such as allowing users to correct translations in real-time and retrain the model based on feedback, could improve translation personalization for various domains or languages.

**CONCLUSIONS**

This project demonstrates the use of Seq2Seq models with LSTMs and attention mechanisms for machine translation from German to English. By leveraging the attention mechanism, the model is able to overcome the limitations of basic Seq2Seq architectures, providing more accurate translations, particularly for longer sentences. The model is trained and tested on a German-English parallel corpus, and its performance is evaluated using the BLEU score. The proposed system improves upon traditional translation methods, providing a more effective approach to handling the complexities of language translation.

**REFERENCES**

1. Cho, K., van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014). Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. *arXiv preprint arXiv:1406.1078*.
2. Bahdanau, D., Cho, K., & Bengio, Y. (2015). Neural Machine Translation by Jointly Learning to Align and Translate. *Proceedings of the International Conference on Learning Representations (ICLR)*.
3. Sutskever, I., Vinyals, O., & Le, Q. V. (2014). Sequence to Sequence Learning with Neural Networks. *Advances in Neural Information Processing Systems* (NIPS).
4. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is All You Need. *Advances in Neural Information Processing Systems*.
5. Graves, A. (2013). Generating Sequences with Recurrent Neural Networks. *arXiv preprint arXiv:1308.0850*.