Lingoistic

DL Project 07.05.2021

—

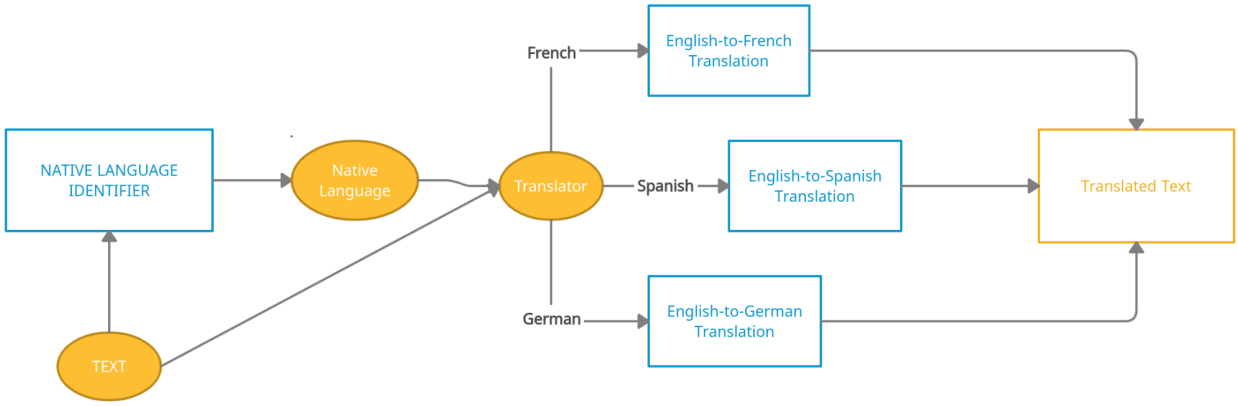
Contents

Group S21DL13

Nethra Gunti	S20180010061
Aditi Verma	S20180010006
Naga Harshita	S20180010100
Meda Rukmini	S20180010102

Overview

Lingoistic is a deep learning project that identifies the native language of an author from *French, Spanish and German*, and translates it to the claimed native language. We used a BERT-based sequence classifier for NLI, an attention based transformer model for text-to-text translation.




```
--------------------------------------------------------------------------------
TRANSLATING 5 RANDOM SENTENCES TO GERMAN

Hello,  =>      Hallo!
So i began to learn japanese - by myself.       =>      Also fing ich an, Japanisch zu lernen - von mir.
At the moment I can all the kana and about 200 kanji.   =>      Im Moment kann ich alle kana und etwa 200 kanji.
My english teacher can't really help me...       =>      Mein Englischlehrer kann mir nicht wirklich helfen...
こんばんは       =>      ► Nachrichten

--------------------------------------------------------------------------------


--------------------------------------------------------------------------------
TRANSLATING 5 RANDOM SENTENCES TO FRENCH

I had for about four years english now. =>      j ai les des des des . . . <EOS>
I feel guilty, but I don't know why.    =>      je ai simplement pourquoi je n connais . <EOS>

--------------------------------------------------------------------------------
```

Part 1: Native Language Identification

❖ Theory

Native Language Identification is the identification of an author's native language, using his writings. Traditional approaches include using grammar models and SVM classifiers but recent advancements in NLP have made it possible to solve NLI using deep learning techniques. One such technique used in this implementation is the BERT-based model for sequence classification. We used the *Transformers* package for all the models.

Dataset: We used the Lang-8 English Learner's Corpora for this task. There are two versions of the corpora available: one cleaned dataset containing only the sentences in English and their corrections; and one with all the raw data from the website till 2011. The English-only dataset does not have the L1 (native language) tag for the learner, and thus, was not suited to this task. So, we used the raw data after parsing, cleaning and structuring it.

The data is in json format. The structure is:

```
["journal_id",
  "sentence_id",
  "learning_language",
  "native_language",
  ["learner_sentence1","learner_sentence2",...],
  [["correction1_to_sentence1","correction2_to_sentence1",...],
   ["correction1_to_sentence2","correction2_to_sentence2",...],
   ...],
]
```

Given below is an example of the corpora in its raw format:

```
["772869","227504","English","Spanish",["My prefer color","Hello
people,","Today I didn't know to tell us.","My prefer color is
red.","Because
is funny and diferent.","The red can to pretend dangeruis and it's
sexy.","The
color red is chosen for people with self-confidence."],[[],[],["Today I
didn't
know how to say it this:"],["My favourite color is red."],["Because it is
funny and different."],["Red can pretend to be dangerous and it's
sexy."],["The color red is chosen by people with self confidence."]]]
```

As is evident, this data, while useful, is not readily usable in a DL mode. So, we decided to parse the .dat file (each line representing a JSON object) and created a csv which could be easily loaded and used.

	journal_id	sentence_id	learning_language	native_language	sentence_text
0	728457	216037	English	Japanese	About winter
1	728457	216037	English	Japanese	This is my second post.
2	728457	216037	English	Japanese	I will appreciate it if you correct my sentences.
3	728457	216037	English	Japanese	It's been getting colder these days here in Ja...
4	728457	216037	English	Japanese	The summer weather in Japan is not agreeable t...

Above is the cleaned dataset. As it can be seen below, the dataset has 65+ foriegn language speakers' sentences written in English with a total of `6404718` sentences.

```
df['native_language'].unique()
```

```
array(['Japanese', 'Korean', 'Vietnamese', 'Mandarin', 'German',
       'Traditional Chinese', 'Russian', 'Thai', 'Spanish', 'Hebrew',
       'Portuguese(Brazil)', 'Cantonese', 'French', 'Italian',
       'Mongolian', 'Polish', 'English', 'Afrikaans', 'Greek', 'Turkish',
       'Farsi', 'Indonesian', 'Ukrainian', 'Catalan', 'Urdu', 'Latvian',
       'Arabic', 'Other language', 'Hindi', 'Croatian', 'Malaysian',
       'Romanian', 'Hungarian', 'Finnish', 'Portuguese(Portugal)',
       'Czech', 'Swedish', 'Dutch', 'Lithuanian', 'Tagalog', 'Danish',
       'Norwegian', 'Javanese', 'Tamil', 'Bosnian', 'Telugu', 'Sinhalese',
       'Bulgarian', 'Azerbaijanian', 'Uzbek', 'Serbian', 'Slovak',
       'Laotian', 'Icelandic', 'Armenian', 'Khmer', 'Georgian',
       'Esperanto', 'Slovenian', 'Estonian', 'Bengali', 'Panjabi',
       'Swahili', 'Albanian', 'Latin', 'Welsh'], dtype=object)
```

Then a subset of the dataset is taken, keeping only the required languages (the ones we eventually use for MTL). This reduces the dataset size significantly. Finally, we clean this new dataset to remove a majority of non-Latin characters (as these could be a dead giveaway for the language) and the cleaned dataset has `96916/2=48458` sentences only.

```
df['native_language'].value_counts()
```

```
Spanish    26642
French     12500
German      9316
Name: native_language, dtype: int64
```

```
df.size
```

```
96916
```

Label Encoding: We encoded the labels from strings to integers for multi class classification.

```
{'French': 2, 'German': 0, 'Spanish': 1}
```

Tokenization: We used the pretrained BERT Tokenizer with attention mask to generate the word embeddings for both train and test datasets.

```
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased', do_lower_case=True)

encoded_data_train = tokenizer.batch_encode_plus(
    df[df.data_type=='train'].sentence_text.values,
    add_special_tokens=True,
    return_attention_mask=True,
    padding='longest',
    max_length=512,
    truncation=True,
    return_tensors='pt'
)
```

Model: We finally used a BERT-based model for sequence classification for training and evaluating the dataset.

```
model = BertForSequenceClassification.from_pretrained(
    'bert-base-uncased',
    num_labels=len(labels_dict),
    output_attentions=False,
    output_hidden_states=False
).to(device)
```

❖ Observations

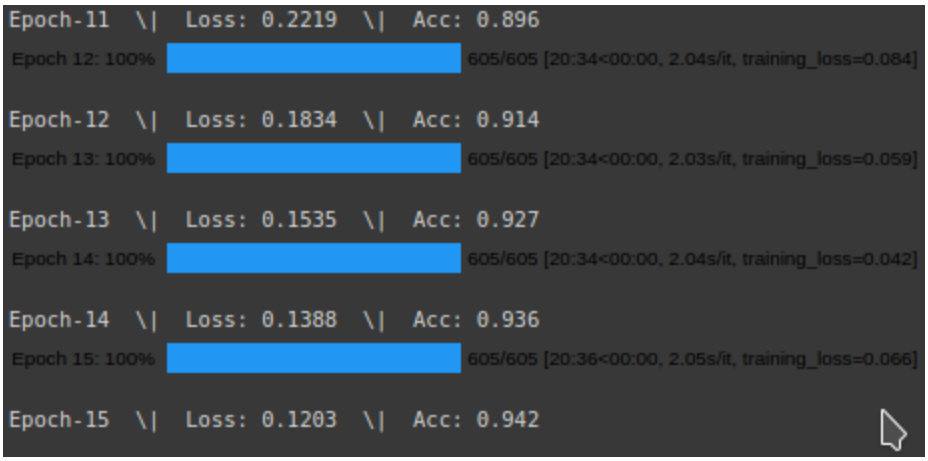
One of the most important things we noticed was that our dataset was biased towards Spanish as it had more than twice as many samples in French and more than thrice as many as in German. This resulted in the greatest class accuracy for Spanish, slightly lower accuracy for French, and the lowest for German. This has led us to believe that the number of samples to learn from in this Supervised model is of paramount importance. The proof can be inferred from the below images.

```
Class: German   |  Accuracy:  0.546

Class: Spanish  |  Accuracy:  0.797

Class: French   |  Accuracy:  0.602

Total Accuracy: 0.648
```

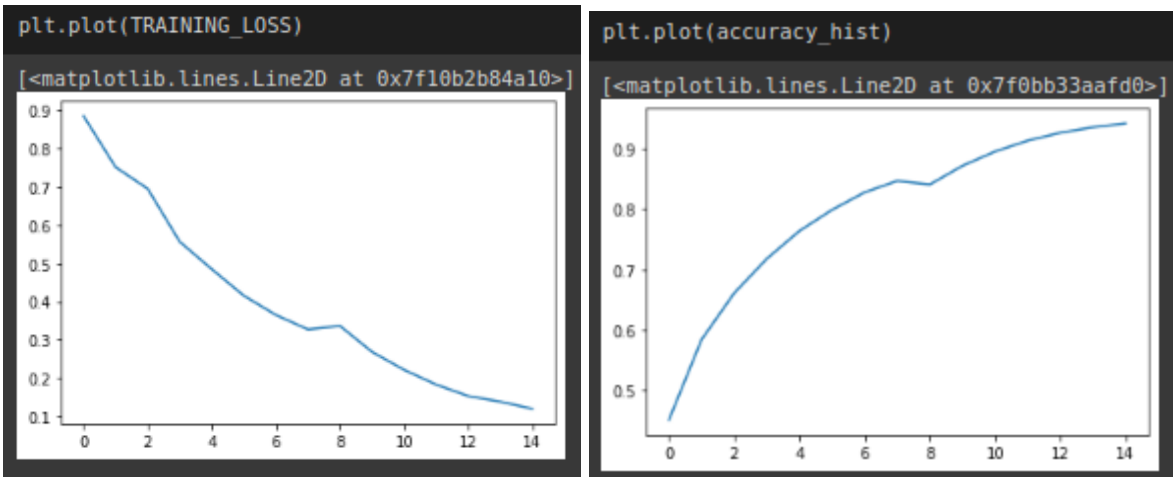
```
df['native_language'].value_counts()

Spanish    26642
French     12500
German      9316
Name: native_language, dtype: int64

df.size

96916
```

We also noticed that after 5 or so epochs, there isn't much increase in validation accuracy, but the train accuracy keeps increasing, reaching a final value of 94.5%, whereas the validation accuracy hovers around 65%. This shows that beyond a certain point, the model tries to overfit the learning data instead of learning meaningful features from it.

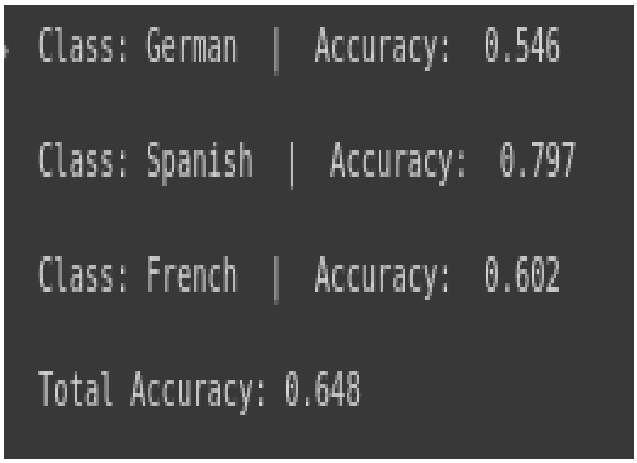
❖ Results



Training scores for the last 5/15 epochs



Loss and Accuracy Graphs for 15 epochs



Accuracy/Class and the total accuracy of the Validation Set

Part 2: Neural Machine Translation for French

❖ Theory

Machine Translation is a field with many different avenues of application. This had led to the development of several cutting-edge deep learning models which use different techniques such as Recurrent Neural Networks (RNNs), Long-Short-Term Memory (LSTM) units, Gated Recurrent Units (units), Transformers and attention in order to produce extremely accurate results.

In the interest of learning and having a little variety in the project, we decided to implement RNNs with Attention for English-French translation.

The dataset used was the Tatoeba Dataset which contains 185583 English-French sentence pairs. We only used samples which had num_words <= 30 (including the ending punctuation).

Here we created two models, an Encoder and a Decoder with Attention.

```
class EncoderRNN(nn.Module):
    def __init__(self, input_size, hidden_size):
        super(EncoderRNN, self).__init__()
        self.hidden_size = hidden_size

        self.embedding = nn.Embedding(input_size, hidden_size)
        self.gru = nn.GRU(hidden_size, hidden_size)

    def forward(self, input, hidden):
        embedded = self.embedding(input).view(1, 1, -1)
        output = embedded
        output, hidden = self.gru(output, hidden)
        return output, hidden

    def initHidden(self):
        return torch.zeros(1, 1, self.hidden_size, device=device)
```

```
class AttnDecoderRNN(nn.Module):
    def __init__(self, hidden_size, output_size, dropout_p=0.1,
max_length=MAX_LENGTH):
        super(AttnDecoderRNN, self).__init__()
        self.hidden_size = hidden_size
        self.output_size = output_size
        self.dropout_p = dropout_p
        self.max_length = max_length

        self.embedding = nn.Embedding(self.output_size, self.hidden_size)
        self.attn = nn.Linear(self.hidden_size * 2, self.max_length)
        self.attn_combine = nn.Linear(self.hidden_size * 2,
self.hidden_size)
        self.dropout = nn.Dropout(self.dropout_p)
        self.gru = nn.GRU(self.hidden_size, self.hidden_size)
        self.out = nn.Linear(self.hidden_size, self.output_size)

    def forward(self, input, hidden, encoder_outputs):
        embedded = self.embedding(input).view(1, 1, -1)
```

```
        embedded = self.dropout(embedded)

        attn_weights = F.softmax(
            self.attn(torch.cat((embedded[0], hidden[0]), 1)), dim=1)
        attn_applied = torch.bmm(attn_weights.unsqueeze(0),
                                 encoder_outputs.unsqueeze(0))

        output = torch.cat((embedded[0], attn_applied[0]), 1)
        output = self.attn_combine(output).unsqueeze(0)

        output = F.relu(output)
        output, hidden = self.gru(output, hidden)

        output = F.log_softmax(self.out(output[0]), dim=1)
        return output, hidden, attn_weights

    def initHidden(self):
        return torch.zeros(1, 1, self.hidden_size, device=device)
```

❖ Observations

The train loss started at around 4.76 and settled at 3.21 after 3 epochs.

```
1m 57s (- 64m 45s) (5000 -- 2.9308323563892147%) 4.7653
3m 47s (- 60m 57s) (10000 -- 5.861664712778429%) 4.2278
5m 39s (- 58m 37s) (15000 -- 8.792497069167643%) 3.9511
7m 31s (- 56m 42s) (20000 -- 11.723329425556859%) 3.7830
9m 24s (- 54m 49s) (25000 -- 14.654161781946073%) 3.6443
11m 18s (- 53m 1s) (30000 -- 17.584994138335286%) 3.5431
13m 12s (- 51m 11s) (35000 -- 20.5158264947245%) 3.4790
15m 7s (- 49m 23s) (40000 -- 23.446658851113718%) 3.4222
17m 0s (- 47m 28s) (45000 -- 26.377491207502928%) 3.3284
18m 54s (- 45m 37s) (50000 -- 29.308323563892145%) 3.2946
20m 48s (- 43m 44s) (55000 -- 32.23915592028136%) 3.2390
22m 43s (- 41m 52s) (60000 -- 35.16998827667057%) 3.2077
24m 36s (- 39m 58s) (65000 -- 38.10082063305979%) 3.2214
26m 30s (- 38m 5s) (70000 -- 41.031652989449%) 3.1747
28m 24s (- 36m 12s) (75000 -- 43.96248534583822%) 3.1458
30m 18s (- 34m 19s) (80000 -- 46.893317702227435%) 3.1426
32m 12s (- 32m 26s) (85000 -- 49.824150058616645%) 3.1093
34m 5s (- 30m 32s) (90000 -- 52.754982415005856%) 3.0825
```

One thing to note is the sentence length. The model might have had fewer losses if we were to limit the sentence length to 20 or even 10. It would have resulted in better translations as well.

Further we came to the realization that the <start> and <end> tokens are very important for the model. Training without them led to significantly longer outputs than desired with a lot of repetition.

Finally, the RNN model performs worse than a Transformer model (NMT) would have given the same data. This is primarily due to the application of multi-head attention in the latter.

The pretrained Transformer en2fr model provided by PyTorch is a superior choice in terms of translation quality.

❖ Results

Here are a few translation results of the model.

```
> we actually saw the accident .
= nous avons reellement vu l accident .
< nous avons vu un accident . <EOS>
```

```
> is it dangerous ?
= est ce perilleux ?
< est ce il dangereux ? <EOS>
```

```
> how do you always do it ?
= comment le faites vous toujours ?
< comment vous toujours ca ? <EOS>
```

```
> have you seen my new car ?
= as tu vu ma nouvelle auto ?
< as tu vu vu mon voiture ? <EOS>
```

```
> i don t know what you mean .
= je ne sais pas ce que vous voulez dire .
< je ne sais pas ce que vous savez pas <EOS>
```

```
> how s the water ?
= comment est l eau ?
< comment est a t eau ? <EOS>
```

```
> i lost my car keys .
= j ai perdu mes cles de voiture .
< j ai perdu de ma cles de ma cles voiture <EOS>
```

```
> i have two questions for you .
= j ai deux questions pour toi .
< je vous ai deux questions . <EOS>
```

```
> there is no hope .
= il n y a pas d espoir .
< il n y pas pas pas pas pas pas pas pas pas pas pas pas pas pas
```

An example of bad translation.

Part 3: Neural Machine Translation for Spanish

❖ Theory

- This part mainly focuses on the translation to Spanish if it is the author's native language. For training the translation model, I used an attention based transformer

with position encodings. In transformers, the sense of time is not taken as in RNN's. Instead we use position encodings for the sake of sequential information.

Dataset used: http://www.statmt.org/europarl/

No of sentences: **1,965,734** in both languages

Tradeoff taken between the training and validation data is 80: 20

Libraries used: Transformers, Robertatokenizer

Positional Encoding:

To capture the sequential information, positional encodings have come into picture.

The relative position of the words, sin and cosine functions are used.

```
self.dropout = nn.Dropout(p=dropout)
pe = torch.zeros(max_len, d_model)
position = torch.arange(0, max_len, dtype=torch.float).unsqueeze(1)
div_term = torch.exp(
    torch.arange(0, d_model, 2).float() * (-math.log(10000.0) / d_model)
)
pe[:, 0::2] = torch.sin(position * div_term)
pe[:, 1::2] = torch.cos(position * div_term)
```

Transformer Model:

It has a encoder-decoder network, where input passes through the encoder and the translated sentence that we get from the decoder.

Initially tokenisation is done and convert the input into word embeddings. For this, I have used RobertaTokenizer and nn.Embedding

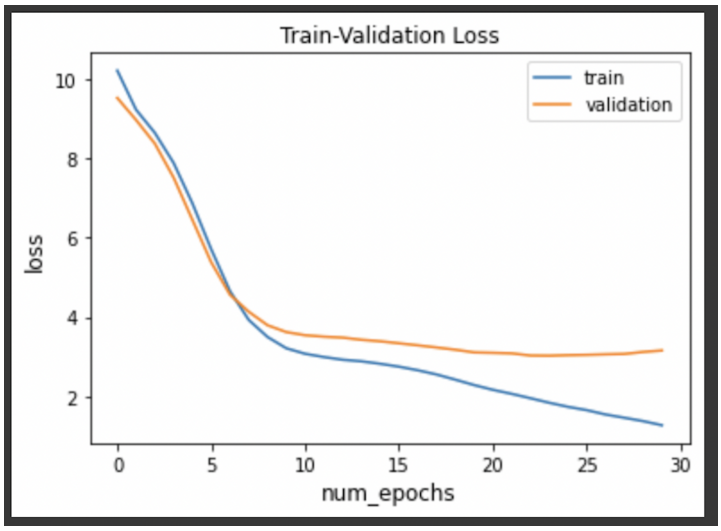
For these embedded sentences, adding the position encoding with sin and cos functions. Similar steps are taken in the decoder network.

```
self.d_model = d_model
self.embed_src = nn.Embedding(vocab_size, d_model)
self.embed_tgt = nn.Embedding(vocab_size, d_model)
self.pos_enc = PositionalEncoding(d_model, pos_dropout, max_seq_length)

self.transformer = nn.Transformer(
    d_model,
    nhead,
    num_encoder_layers,
    num_decoder_layers,
    dim_feedforward,
    trans_dropout,
```

```
        )
        self.fc = nn.Linear(d_model, vocab_size)
```

❖ Results

- Here the model is trained for 30 epochs where the loss for the validation dataset remained almost constant at 3 from 15 to 30 epochs.



Part 4: Neural Machine Translation for German

❖ Theory

Introduction

This part is the neural machine translation for english to german. In our project, if the native language identified by the NLI - BERT system is german, the pipeline routes the input to this translator to convert the text from english to german. For this task, a transformer with attention is implemented.

Dataset

The dataset used is the Statistical Machine Translation dataset for english to german translation. The Europarl Parliament Proceedings, Common Crawl and News Commentary corpora of the dataset are used.

Europarl Parliament Proceedings corpus has nearly 2,176,537 sentences and 47,236,849 words. The remaining corpora also have an approximate size as of Europarl.

Implementation details

The dataset is preprocessed by tokenizing. For this, an unsupervised tokenizer called youtokentome is used to learn the Byte Pair Encodings (BPE) model.

```
print("\nLearning BPE...")
youtokentome.BPE.train(data=os.path.join(data_folder, "train.ende"), vocab_size=37000,
                       model=os.path.join(data_folder, "bpe.model"))

print("\nLoading BPE model...")
bpe_model = youtokentome.BPE(model=os.path.join(data_folder, "bpe.model"))
```

fig1. Learning and loading the youtokentome BPE model

The transformer model has multi - headed attention modules linked to normalization and linear layers in the encoder and decoder networks. Positional encoding is used after the input encodings in encoder and decoder networks

```
class Transformer(nn.Module):

    def __init__(self, vocab_size, positional_encoding, d_model=512, n_heads=8, d_queries=64, d_values=64,
                 d_inner=2048, n_layers=6, dropout=0.1):
        super(Transformer, self).__init__()

        self.vocab_size = vocab_size
        self.positional_encoding = positional_encoding
        self.d_model = d_model
        self.n_heads = n_heads
        self.d_queries = d_queries
```

fig2. Transformer model definition

```
class MultiHeadAttention(nn.Module):

    def __init__(self, d_model, n_heads, d_queries, d_values, dropout, in_decoder=False):
        super(MultiHeadAttention, self).__init__()

        self.d_model = d_model
        self.n_heads = n_heads
```

fig3. Multi head attention

Label smoothed cross entropy loss is computed. An Adam optimizer is used. Training and validation is performed at each epoch. BLEU scores are computed on the test or evaluation dataset.

❖ Results

An approximate BLEU score of 20 is obtained.

```
3 best_translation, all_translations = translate("I am a girl")
4 print(best_translation)
```

```
Ich bin ein Mädchen.
```

Fig4. English to German Translation output of a sample english sentence

Part 5: Individual Contributions

Native Language Identification: Nethra & Aditi

MTL English-French: Aditi

MTL English-German: Rukmini

MTL English-Spanish: Naga Harshita