BIRLA INSTITUTE OF TECHNOLOGY

MESRA - 835215

NLP ASSIGNMENT : Topic-Translator

Members:

Tulika Gupta(MCA/10019/22)

Saakshi Priya(MCA/10002/22)

Kriti Anand(MCA/10005/22)

Pratibha Roy(MCA/10015/22)

In [1]:
```
import string
import re
import pandas as pd
from keras.models import Sequential
from keras.layers import Embedding, LSTM, Dense, RepeatVector
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from sklearn.model_selection import train_test_split
from keras.callbacks import ModelCheckpoint
from keras.models import load_model
from keras import optimizers
```

In [2]:
```
data_path = 'C:/Users/91947/OneDrive/Desktop/fra-eng/fra.txt'
with open(data_path, 'r', encoding='utf-8') as f:
    lines = f.read()
```

In [4]:
```
#splitting into lines and words for preprocessing
def to_lines(text):
    sents = text.strip().split('\n')
    sents = [i.split('\t') for i in sents]
    return sents
```

In [ ]:
```
lines
```

In [5]:
```
fra_eng = to_lines(lines)
fra_eng[:5]
```

Out[5]:
```
[['Go.', 'Va !'],
 ['Hi.', 'Salut !'],
 ['Run!', 'Cours\u202f!'],
 ['Run!', 'Courez\u202f!'],
 ['Who?', 'Qui ?']]
```

In [6]:
```
#Converting into array
import numpy as np

fra_eng = np.array(fra_eng)
fra_eng[:5]
```

Out[6]:
```
array([['Go.', 'Va !'],
       ['Hi.', 'Salut !'],
       ['Run!', 'Cours\u202f!'],
       ['Run!', 'Courez\u202f!'],
       ['Who?', 'Qui ?']], dtype='<U349')
```

In [7]:
```
#Selecting only 50000 records for fast processing as the data set is too large
fra_eng = to_lines(lines)
fra_eng = np.array(fra_eng)[:50000, [0, 1]]
```

In [8]:
```
#DATACLEANING
#remove punctuation
fra_eng[:, 0] = [s.translate(str.maketrans('', '', string.punctuation)).lower() for
fra_eng[:, 1] = [s.translate(str.maketrans('', '', string.punctuation)).lower() for
```

In [9]:
```
#TEXT TO SEQUENCE CONVERSION (WORD TO INDEX MAPPING)

#function to build a tokenizer
# Tokenization
def tokenization(lines):
    tokenizer = Tokenizer()
    tokenizer.fit_on_texts(lines)
```

```
    return tokenizer

eng_tokenizer = tokenization(fra_eng[:, 0])
eng_vocab_size = len(eng_tokenizer.word_index) + 1
eng_length = 8

fra_tokenizer = tokenization(fra_eng[:, 1])
fra_vocab_size = len(fra_tokenizer.word_index) + 1
fra_length = 8
```

In [10]:
```
# Data encoding
def encode_sequences(tokenizer, length, lines):
    seq = tokenizer.texts_to_sequences(lines)
    seq = pad_sequences(seq, maxlen=length, padding='post')
    return seq
```

In [11]:
```
# Split data into train and test sets
train, test = train_test_split(fra_eng, test_size=0.2, random_state=12)
trainX = encode_sequences(fra_tokenizer, fra_length, train[:, 1])
trainY = encode_sequences(eng_tokenizer, eng_length, train[:, 0])
testX = encode_sequences(fra_tokenizer, fra_length, test[:, 1])
testY = encode_sequences(eng_tokenizer, eng_length, test[:, 0])
```

In [12]:
```
# Define the NMT model
def define_model(input_vocab, output_vocab, input_timesteps, output_timesteps, unit
    model = Sequential()
    model.add(Embedding(input_vocab, units, input_length=input_timesteps, mask_zerc
    model.add(LSTM(units))
    model.add(RepeatVector(output_timesteps))
    model.add(LSTM(units, return_sequences=True))
    model.add(Dense(output_vocab, activation='softmax'))
    return model
```

In [13]:
```
#creating an encoder-decoder architecture for neural machine translation.
model = define_model(fra_vocab_size, eng_vocab_size, fra_length, eng_length, units=
```

In [14]:
```
# Compile the model
optimizer = optimizers.RMSprop(learning_rate=0.001)
model.compile(optimizer=optimizer, loss='sparse_categorical_crossentropy')
```

In [15]:
```
# Train the model
model.fit(trainX, trainY, epochs=10, batch_size=512, validation_split=0.2)
```

```
Epoch 1/10
63/63 [==============================] - 177s 3s/step - loss: 4.4844 - val_loss:
3.3870
Epoch 2/10
63/63 [==============================] - 163s 3s/step - loss: 3.1783 - val_loss:
3.0796
Epoch 3/10
63/63 [==============================] - 165s 3s/step - loss: 3.0180 - val_loss:
2.9943
Epoch 4/10
63/63 [==============================] - 162s 3s/step - loss: 2.9661 - val_loss:
2.9873
Epoch 5/10
63/63 [==============================] - 12174s 196s/step - loss: 2.9408 - val_los
s: 2.9611
Epoch 6/10
63/63 [==============================] - 157s 2s/step - loss: 2.9211 - val_loss:
2.9299
Epoch 7/10
63/63 [==============================] - 160s 3s/step - loss: 2.9047 - val_loss:
2.9179
Epoch 8/10
63/63 [==============================] - 160s 3s/step - loss: 2.8905 - val_loss:
2.9248
Epoch 9/10
63/63 [==============================] - 161s 3s/step - loss: 2.8777 - val_loss:
2.8974
Epoch 10/10
63/63 [==============================] - 163s 3s/step - loss: 2.8673 - val_loss:
2.8984
```

Out[15]:

```
<keras.src.callbacks.History at 0x219060890d0>
```

In [16]:

```python
preds = model.predict(testX)
```

```
313/313 [==============================] - 43s 123ms/step
```

In [17]:

```python
#these predictions are sequences of integers. We need to convert these integers to
def get_word(n, tokenizer):
    return tokenizer.index_word.get(n)
```

In [18]:

```python
#convert predictions into sentences(English)
max_length = eng_length
preds_text = []

for i in preds:
    temp = []
    for j in range(max_length):
        if j < len(i):
            t = get_word(np.argmax(i[j]), eng_tokenizer)
            if j > 0:
                if (t == get_word(np.argmax(i[j - 1]), eng_tokenizer)) or (t is Nor
                    temp.append(' ')
                else:
                    temp.append(t)
            else:
                if t is None:
                    temp.append(' ')
                else:
                    temp.append(t)
        else:
            temp.append(' ')
    preds_text.append(' '.join(temp))
```

In [19]:
```
#LET'S PUT THE ORIGINAL ENGLISH SENTENCES IN THE TEST DATASET AND THE PREDICTED SEQ
```

In [20]:
```
pred_df = pd.DataFrame({'actual': test[:, 0], 'predicted': preds_text})
```

In [21]:
```
#print 15 rows randomly
pred_df.sample(15, replace=True)
```

Out[21]:

	actual	predicted
3819	lets have a good time	i is a
2347	youre very observant	youre you
2188	tom has green eyes	i is a
1321	i have to stop	i is a
2162	thats a big deal	i is a
6176	is this love	i you
5349	tie your shoe	i you
7678	we cant save everyone	i not a to
8470	you must stop him	youre you
5500	life is too short	i is a
9822	this is my family	i is
6470	whats wrong with you	i is a
459	she has gone shopping	i not to
1601	im not a criminal	i not a to
3587	i think we should quit	i not a to

In [25]:
```
import string
import numpy as np
import pandas as pd
from keras.models import Sequential
from keras.layers import Embedding, LSTM, Dense, RepeatVector
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences

# Load and preprocess the data (similar to the provided code)

# Tokenization (similar to the provided code)

# Data encoding (similar to the provided code)

# Define the NMT model (similar to the provided code)

# Compile the model (similar to the provided code)

# Train the model (similar to the provided code)

# Take user input
user_input = "comment allez-vous"  # Enter the input in French

# Preprocess user input
user_input = user_input.lower()
user_input = user_input.translate(str.maketrans('', '', string.punctuation))
```

```
user_input_sequence = fra_tokenizer.texts_to_sequences([user_input])
user_input_sequence = pad_sequences(user_input_sequence, maxlen=fra_length, padding

# Translate user input
predicted_sequence = model.predict(user_input_sequence)
translated_text = ""

for i in predicted_sequence[0]:
    word = eng_tokenizer.index_word.get(np.argmax(i))
    if word:
        translated_text += word + " "

# Display the translation
print("Translated: " + translated_text)
```

```
1/1 [==============================] - 0s 206ms/step
Translated: youre you
```

In [ ]:

NATURAL LANGUAGE PROCESSING

Topic: Translator

Members:

- Tulika Gupta(MCA/10019/22)
- Saakshi Priya(MCA/10002/22)
- Kriti Anand(MCA/10005/22)
- Pratibha Roy(MCA/10015/22)

MAJOR POINTS

- Introduction to Translator
- Seq2seq
- RNN
- Our Approach for the project

LANGUAGE TRANSLATOR

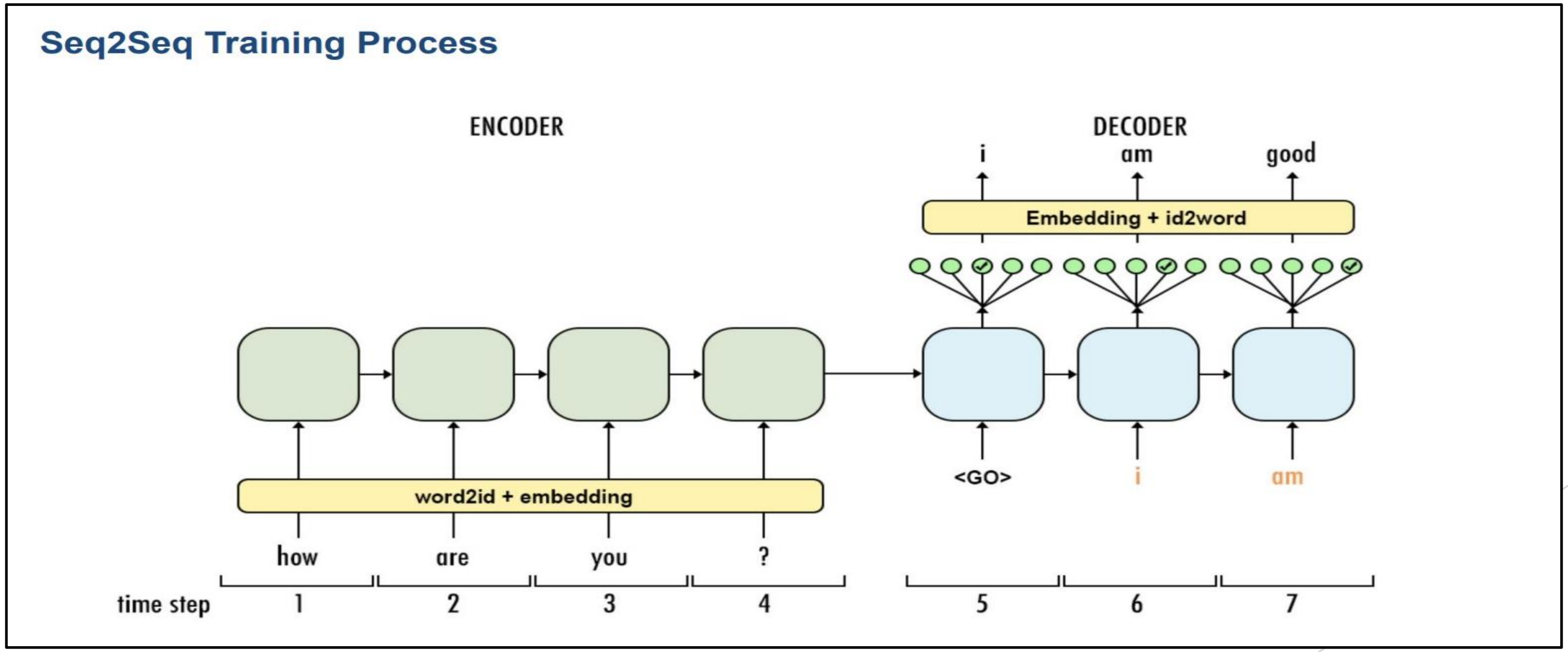
Introduction:

- As the name suggests Language translator is that which translates sentences from one language to another.
- Translators play a vital role in bridging language barriers and is essential in various fields, including literature, business, diplomacy, travel, and more.
- So basically the objective of this Language Translator project is that we are going to translate french sentences to english sentences with the help of machine learning.

SEQ2SEQ MODEL:

In this project, we will be using **Sequence-to-Sequence (Seq2Seq) Modeling** to translate given sentences from one language to another.

- In Sequence-to-Sequence learning (Seq2Seq) we have to create training models that convert sequences from one domain(e.g. sentences in French) to sequences in another domain (e.g. the same sentences translated to English).



- Sequence-to-Sequence (Seq2Seq) models are widely used in Natural Language Processing (NLP) for tasks like machine translation, text summarization, and chatbot development.

- here are the brief steps for building a Seq2Seq model:

1. Data Collection and Preprocessing:

   - Gather paired sequences (e.g., source and target language sentences).

   - Preprocess the data, including tokenization and cleaning.

2. Vocabulary Building:

   - Create vocabularies for both source and target languages.

3. Data Sequencing:

   - Convert text data into sequences of integers using vocabularies.

4. Sequence Padding:

   - Ensure all sequences have the same length by padding shorter sequences.

5. Model Architecture:

- Design a Seq2Seq model with an encoder and decoder.

6. Embeddings:

- Use word embeddings to represent words in continuous vector spaces.

7. Model Training:

- Train the model to minimize the difference between predicted and actual sequences.

8. Inference:

- During inference, feed a source sequence to the encoder and generate the target sequence.

9. Decoding:

- Choose a decoding strategy (e.g., greedy decoding or beam search) to generate the output sequence.

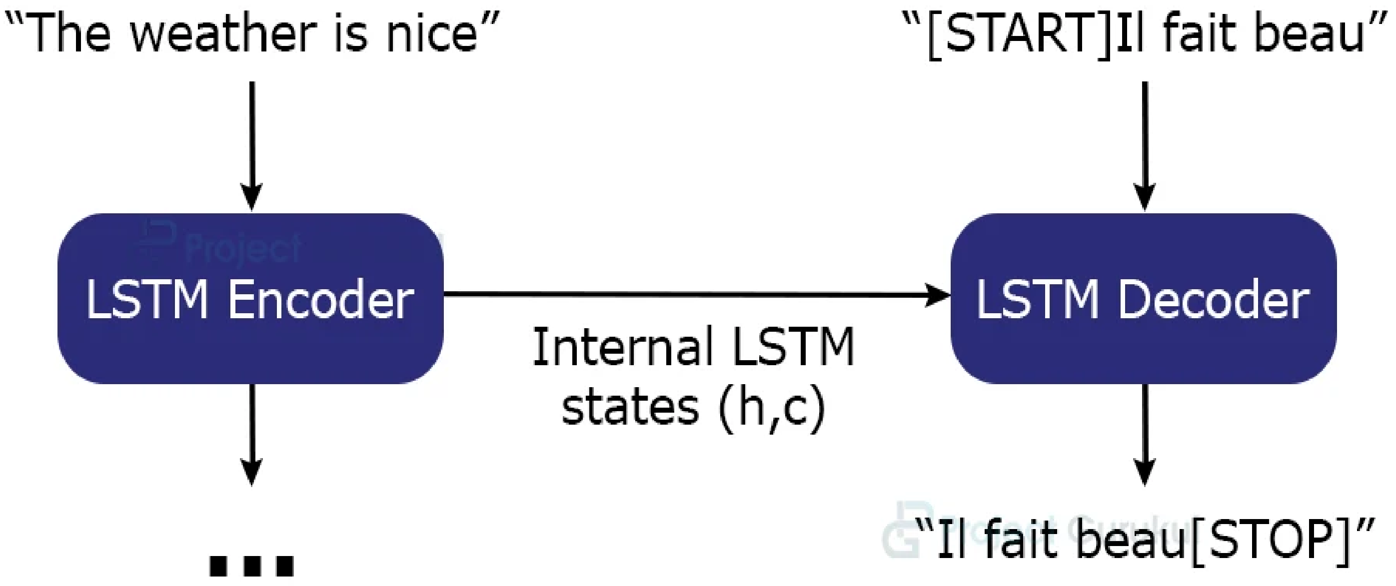
RECURRENT NEURAL NETWORK[RNN]:

- A Recurrent Neural Network (RNN) is a type of neural network designed to handle sequential data. It's significant in translator NLP (Natural Language Processing) because:

1. **Sequence Processing:** RNNs excel at processing sequences, making them ideal for tasks like translation, where the order of words matters.
2. **Contextual Understanding:** RNNs maintain a hidden state, preserving context from previous words, which is crucial for understanding and translating sentences.
3. **Language Models:** RNNs can serve as the foundation for language models that predict the likelihood of a word given its context, a key component of translation.

- For implementing, we will create two RNN layer :

One RNN layer will act as ‘encoder’: In this we give our english sentence as an input.
And other RNN layer will act as ‘decoder’: which will give us the output (translated sentence in french)



Long Short Term Memory Network[LSTM]:

- Long Short-Term Memory Networks, it is a type of Recurrent Neural Network (RNN). RNN is basically used for sequential data, as it is the first algorithm that remembers its input, due to an internal memory.

- LSTM are capable of learning order dependence in sequence prediction problems. This is basically used when you have complex problem domains like speech recognition, machine translation etc.
  we will be creating two RNN layers, one will be for encoder and other will be for decoder.

OUR APPROACH:

- 1. Data collection and preprocessing: We start with the collection of a bilingual dataset, consisting of paired sequences in the source and target languages. To prepare the data for training, we perform essential preprocessing tasks like tokenization, sequence padding, and data splitting."

```
In [18]: #splitting into lines and words for preprocessing
         def to_lines(text):
             sents = text.strip().split('\n')
             sents = [i.split('\t') for i in sents]
             return sents

In [19]: fra_eng = to_lines(lines)
         fra_eng[:5]

Out[19]: [['Go.', 'Va !'],
          ['Hi.', 'Salut !'],
          ['Run!', 'Cours\u202f!'],
          ['Run!', 'Courez\u202f!'],
          ['Who?', 'Qui ?']]
```

```
In [23]: #Converting into array
         import numpy as np

         fra_eng = np.array(fra_eng)
         fra_eng[:5]

Out[23]: array([['Go.', 'Va !'],
                ['Hi.', 'Salut !'],
                ['Run!', 'Cours\u202f!'],
                ['Run!', 'Courez\u202f!'],
                ['Who?', 'Qui ?']], dtype='<U349')

In [85]: #Selecting only 50000 records for fast processing as the data set is too large
         fra_eng = to_lines(lines)
         fra_eng = np.array(fra_eng)[:50000, [0, 1]]
```

•

```
In [86]: #DATACLEANING
         #remove punctuation
         fra_eng[:, 0] = [s.translate(str.maketrans('', '', string.punctuation)).lower() for s in fra_eng[:, 0]]
         fra_eng[:, 1] = [s.translate(str.maketrans('', '', string.punctuation)).lower() for s in fra_eng[:, 1]]
```

- **Tokenization:** Tokenization involves converting text into numerical values to make it suitable for machine learning. Separate tokenizers are used for the source and target languages.

```
In [87]: #TEXT TO SEQUENCE CONVERSION (WORD TO INDEX MAPPING)

         #function to build a tokenizer
         # Tokenization
         def tokenization(lines):
             tokenizer = Tokenizer()
             tokenizer.fit_on_texts(lines)
             return tokenizer

         eng_tokenizer = tokenization(fra_eng[:, 0])
         eng_vocab_size = len(eng_tokenizer.word_index) + 1
         eng_length = 8

         fra_tokenizer = tokenization(fra_eng[:, 1])
         fra_vocab_size = len(fra_tokenizer.word_index) + 1
         fra_length = 8
```

```
In [88]: # Data encoding
         def encode_sequences(tokenizer, length, lines):
             seq = tokenizer.texts_to_sequences(lines)
             seq = pad_sequences(seq, maxlen=length, padding='post')
             return seq
```

```
In [89]: # Split data into train and test sets
         train, test = train_test_split(fra_eng, test_size=0.2, random_state=12)
         trainX = encode_sequences(fra_tokenizer, fra_length, train[:, 1])
         trainY = encode_sequences(eng_tokenizer, eng_length, train[:, 0])
         testX = encode_sequences(fra_tokenizer, fra_length, test[:, 1])
         testY = encode_sequences(eng_tokenizer, eng_length, test[:, 0])
```

```
In [90]: # Define the NMT model
         def define_model(input_vocab, output_vocab, input_timesteps, output_timesteps, units):
             model = Sequential()
             model.add(Embedding(input_vocab, units, input_length=input_timesteps, mask_zero=True))
             model.add(LSTM(units))
             model.add(RepeatVector(output_timesteps))
             model.add(LSTM(units, return_sequences=True))
             model.add(Dense(output_vocab, activation='softmax'))
             return model
```

```
In [91]: #creating an encoder-decoder architecture for neural machine translation.
         model = define_model(fra_vocab_size, eng_vocab_size, fra_length, eng_length, units=512)
```

```
In [92]: # Compile the model
         optimizer = optimizers.RMSprop(learning_rate=0.001)
         model.compile(optimizer=optimizer, loss='sparse_categorical_crossentropy')
```

Training: The model learns to minimize the difference between its predictions and the actual target sequences during the training process.

```
In [93]: # Train the model
         model.fit(trainX, trainY, epochs=10, batch_size=512, validation_split=0.2)

         Epoch 1/10
         63/63 [==============================] - 74s 1s/step - loss: 4.4891 - val_loss: 3.4017
         Epoch 2/10
         63/63 [==============================] - 69s 1s/step - loss: 3.1764 - val_loss: 3.0581
         Epoch 3/10
         63/63 [==============================] - 70s 1s/step - loss: 3.0187 - val_loss: 2.9973
         Epoch 4/10
         63/63 [==============================] - 69s 1s/step - loss: 2.9697 - val_loss: 2.9846
         Epoch 5/10
         63/63 [==============================] - 69s 1s/step - loss: 2.9424 - val_loss: 2.9441
         Epoch 6/10
         63/63 [==============================] - 70s 1s/step - loss: 2.9202 - val_loss: 2.9276
         Epoch 7/10
         63/63 [==============================] - 69s 1s/step - loss: 2.9055 - val_loss: 2.9254
         Epoch 8/10
         63/63 [==============================] - 69s 1s/step - loss: 2.8921 - val_loss: 2.9123
         Epoch 9/10
         63/63 [==============================] - 72s 1s/step - loss: 2.8792 - val_loss: 2.9201
         Epoch 10/10
         63/63 [==============================] - 72s 1s/step - loss: 2.8692 - val_loss: 2.8893

Out[93]: <keras.src.callbacks.History at 0x2448e1e4a50>
```

```
In [84]: preds = model.predict(textX.reshape((textX.shape[0], textX.shape[1])))
         #these predictions are sequences of integers. We need to convert these integers to their corresponding words.

         63/63 [==============================] - 10s 109ms/step
```

```
In [71]: #these predictions are sequences of integers. We need to convert these integers to their corresponding words
def get_word(n, tokenizer):
    return tokenizer.index_word.get(n)
```

```
In [72]: #convert predictions into sentences(English)
max_length = eng_length
preds_text = []

for i in preds:
    temp = []
    for j in range(max_length):
        if j < len(i):
            t = get_word(np.argmax(i[j]), eng_tokenizer)
            if j > 0:
                if (t == get_word(np.argmax(i[j - 1]), eng_tokenizer)) or (t is None):
                    temp.append(' ')
                else:
                    temp.append(t)
            else:
                if t is None:
                    temp.append(' ')
                else:
                    temp.append(t)
        else:
            temp.append(' ')
    preds_text.append(' '.join(temp))
```

FINAL RESULTS IN A DATAFRAME:

```
In [73]: pred_df = pd.DataFrame({'actual': test[:, 0], 'predicted': preds_text})

In [74]: #print 15 rows randomly
         pred_df.sample(15, replace=True)
```

Out[74]:

	actual	predicted
3220	did i just say that	i not a
1085	am i clear	youre you
680	i went to school	i not a
3696	i want to talk	i is
2371	they finished 13th	i you
3227	ill lend it to you	i not a
1836	they asked him	i you
5385	lets make a cake	i you
2452	she loves tom	i not a
3485	go have fun	youre you
2079	youre the oldest	youre you
5024	he killed himself	i you
62	i won the raffle	i not a
5251	do you trust her	i you
4901	ill check again	i you

THANK YOU