In [ ]:

```
## Model: GRU with Attention
## Conversion: English to Hindi
## BLEU 1-gram Score: 0.42
## Loss @ Epoch 12: 0.0872
```

In [99]:

```
import pandas as pd
from nltk.translate.bleu_score import SmoothingFunction, sentence_bleu
```

In [35]:

```
num_samples_en = 84557
lines_en = pd.read_csv('trainen.txt',encoding='utf8', sep='delimiter', names
lines_en = lines_en[0:num_samples_en]
input_texts_en=len(lines_en)
print(input_texts_en)
```

```
C:\Users\Matt\Anaconda3\lib\site-packages\ipykernel_launcher.py:2: ParserWarning:

Falling back to the 'python' engine because the 'c' engine does not support regex separators (separa
+' are interpreted as regex); you can avoid this warning by specifying engine='python'.


84557
```

In [36]:

```
lines_en.head()
```

	eng
0	And what is their Sigil?
1	I do not want to die.
2	It's the same country I think.
3	Then they'll be crying like babies.
4	- No, I need power up!

In [37]:

```
lines_en.shape
```

```
(84557, 1)
```

In [38]:

```
num_samples_hi = 84557
lines_hi = pd.read_csv('trainhi.txt',encoding='utf8', sep='delimiter', names
lines_hi = lines_hi[0:num_samples_hi]
input_texts_hi=len(lines_hi)
print(input_texts_hi)
```

```
C:\Users\Matt\Anaconda3\lib\site-packages\ipykernel_launcher.py:2: ParserWarning:

Falling back to the 'python' engine because the 'c' engine does not support regex separators (separa
+' are interpreted as regex); you can avoid this warning by specifying engine='python'.


84557
```

In [39]:

```
lines_hi.head()
```

	hin
0	और उनके Sigil क्या है?
1	मैं मरना नहीं चाहता.
2	यह मुझे लगता है कि एक ही देश है.
3	फिर ये नन्हें बच्चों की तरह रोएँगे।
4	नहीं, मुझे पावर की जरुरत है !

In [40]:

```
lines_hi.shape
```

```
(84557, 1)
```

In [41]:

```
data = pd.merge(lines_en, lines_hi, left_index=True, right_index=True)
data.shape
```

```
(84557, 2)
```

In [42]:

```
data.head()
```

	eng	hin
0	And what is their Sigil?	और उनके Sigil क्या है?
1	I do not want to die.	मैं मरना नहीं चाहता.
2	It's the same country I think.	यह मुझे लगता है कि एक ही देश है.
3	Then they'll be crying like babies.	फिर ये नन्हें बच्चों की तरह रोएँगे।
4	- No, I need power up!	नहीं, मुझे पावर की जरुरत है !

In [46]:

```
df = data[~data['hin'].str.contains("[a-zA-Z]").fillna(False)]
df.shape
```

```
(78475, 2)
```

In [47]:

```
df.head()
```

	eng	hin
1	I do not want to die.	मैं मरना नहीं चाहता.
2	It's the same country I think.	यह मुझे लगता है कि एक ही देश है.
3	Then they'll be crying like babies.	फिर ये नन्हें बच्चों की तरह रोएँगे।
4	- No, I need power up!	नहीं, मुझे पावर की जरुरत है !
5	I will not eat him.	मैं उसे नहीं खा जाएगा.

In [48]:

```
df.to_csv('cleaned_data.csv', sep='\t')
```

In [49]:

```
# Importing necessary modules
import tensorflow as tf
tf.enable_eager_execution()
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
import unicodedata
import re
import numpy as np
import os
import time
import string
from chart_studio.plotly import plot, iplot
import plotly
import chart_studio.plotly as py
from plotly.offline import init_notebook_mode, iplot
plotly.offline.init_notebook_mode(connected=True)
import plotly.graph_objs as go
```

In [50]:

```
# Providing location of input data
file_path = 'cleaned_data.txt'
```

In [51]:

```
# Opening file, reading it and delimiting it with tabs
lines = open(file_path, encoding='UTF-8').read().strip().split('\n')
lines[3000:3010]
```

```
['3258\tTeach me? Oh, no!\tमुझे सिखाने के लिए?',
 '3259\tHe deserves to know.\tहकदार करने के लिए पता है.',
 '3260\tI was just getting to the ice cream.\tमैं सिर्फ आइसक्रीम के लिए जा रहा था.',
 "3261\t-First off, I'm a big fan.\t- सबसे पहले, मैं एक बड़ा प्रशंसक रहा हूँ।",
 '3262\tShiva will be born as someone who helps him.\tशिव कोई है जो उसे मदद करता है के रूप में पैदा हो
 '3263\tHas my good time begun?\tमेरा अच्छा समय शायद शुरू हो गया है?',
 '3264\tYeah, for the minute.\tहां, कुछ पल के लिए।',
 '3265\tWould you read it to me, please?\tक्या तुम कृपया मुझे यह पढ़कर सुनाओगी?',
 "3266\tIt's not going to kill me.\tयह मुझे मारने के लिए नहीं जा रहा है.",
 '3267\tOkay, take care buddy.\tठीक है, ध्यान दोस्त ले.']
```

In [52]:

```
# Performing basic cleanup
exclude = set(string.punctuation) # Set of all special characters
remove_digits = str.maketrans('', '', string.digits) # Set of all digits
```

In [53]:

```
# Function to preprocess English sentences in file
def preprocess_eng_sentence(sent):
    sent = sent.lower() # lower casing
    sent = re.sub("'", '', sent) # remove the quotation marks if any
    sent = ''.join(ch for ch in sent if ch not in exclude)
    sent = sent.translate(remove_digits) # remove the digits
    sent = sent.strip()
    sent = re.sub(" +", " ", sent) # remove extra spaces
    sent = '<start> ' + sent + ' <end>' # add <start> and <end> tokens
    return sent
```

In [54]:

```
# Function to preprocess Hindi sentences in file
def preprocess_hin_sentence(sent):
    sent = re.sub("'", '', sent) # remove the quotation marks if any
    sent = ''.join(ch for ch in sent if ch not in exclude)
    sent = re.sub("[२३०८१५७९४६]", "", sent) # remove the digits
    sent = sent.strip()
    sent = re.sub(" +", " ", sent) # remove extra spaces
    sent = '<start> ' + sent + ' <end>' # add <start> and <end> tokens
    return sent
```

In [55]:

```
# Generate pairs of cleaned English and Hindi sentences
sent_pairs = []
for line in lines:
    sent_pair = []
    index, eng, hin = line.split('\t')
    eng = preprocess_eng_sentence(eng)
    sent_pair.append(eng)
    hin = preprocess_hin_sentence(hin)
    sent_pair.append(hin)
    sent_pairs.append(sent_pair)
sent_pairs[3000:3010]
```

```
[['<start> teach me oh no <end>', '<start> मुझे सिखाने के लिए <end>'],
 ['<start> he deserves to know <end>',
  '<start> हकदार करने के लिए पता है <end>'],
 ['<start> i was just getting to the ice cream <end>',
  '<start> मैं सिर्फ आइसक्रीम के लिए जा रहा था <end>'],
 ['<start> first off im a big fan <end>',
  '<start> सबसे पहले मैं एक बड़ा प्रशंसक रहा हूँ। <end>'],
 ['<start> shiva will be born as someone who helps him <end>',
  '<start> शिव कोई है जो उसे मदद करता है के रूप में पैदा हो जाएगा <end>'],
 ['<start> has my good time begun <end>',
  '<start> मेरा अच्छा समय शायद शुरू हो गया है <end>'],
 ['<start> yeah for the minute <end>', '<start> हां कुछ पल के लिए। <end>'],
 ['<start> would you read it to me please <end>',
  '<start> क्या तुम कृपया मुझे यह पढ़कर सुनाओगी <end>'],
 ['<start> its not going to kill me <end>',
  '<start> यह मुझे मारने के लिए नहीं जा रहा है <end>'],
 ['<start> okay take care buddy <end>', '<start> ठीक है ध्यान दोस्त ले <end>']]
```

In [56]:

```
# This class creates a word -> index mapping (e.g,. "dad" -> 5) and vice-ver
# (e.g., 5 -> "dad") for each language,
class LanguageIndex():
    def __init__(self, lang):
        self.lang = lang
        self.word2idx = {}
        self.idx2word = {}
        self.vocab = set()
        self.create_index()

    def create_index(self):
        for phrase in self.lang:
            self.vocab.update(phrase.split(' '))
        self.vocab = sorted(self.vocab)
        self.word2idx['<pad>'] = 0
        for index, word in enumerate(self.vocab):
            self.word2idx[word] = index + 1
        for word, index in self.word2idx.items():
            self.idx2word[index] = word
```

In [57]:

```
def max_length(tensor):
    return max(len(t) for t in tensor)
```

In [58]:

```
# Using the tf.data input pipeline to create dataset
# and then load it in mini-batches
def load_dataset(pairs, num_examples):
    # pairs => already created cleaned input, output pairs
    # index language using the class defined above
    inp_lang = LanguageIndex(en for en, hi in pairs)
    targ_lang = LanguageIndex(hi for en, hi in pairs)
    # Vectorize the input and target languages
    # English sentences
    input_tensor = [[inp_lang.word2idx[s] for s in en.split(' ')] for en, hi
    # Hindi sentences
    target_tensor = [[targ_lang.word2idx[s] for s in hi.split(' ')] for en,
    # Calculate max_length of input and output tensor
    # Here, we'll set those to the longest sentence in the dataset
    max_length_inp, max_length_tar = max_length(input_tensor), max_length(ta
    # Padding the input and output tensor to the maximum length
    input_tensor = tf.keras.preprocessing.sequence.pad_sequences(input_tenso
                                                                 maxlen=max_
                                                                 padding='po
    target_tensor = tf.keras.preprocessing.sequence.pad_sequences(target_ten
                                                                  maxlen=max
                                                                  padding='p
    return input_tensor, target_tensor, inp_lang, targ_lang, max_length_inp,
```

In [59]:

```
input_tensor, target_tensor, inp_lang, targ_lang, max_length_inp, max_length
```

In [60]:

```
# Creating training and validation sets using an 80-20 split
input_tensor_train, input_tensor_val, target_tensor_train, target_tensor_val
                                                          random_state = 101

# Show length
len(input_tensor_train), len(target_tensor_train), len(input_tensor_val), le
```

```
(62780, 62780, 15695, 15695)
```

In [61]:

```
    if tf.test.is_gpu_available():
        print("Yes")
```

```
Yes
```

In [62]:

```
BUFFER_SIZE = len(input_tensor_train)
BATCH_SIZE = 8
N_BATCH = BUFFER_SIZE//BATCH_SIZE
embedding_dim = 256
units = 1024
vocab_inp_size = len(inp_lang.word2idx)
vocab_tar_size = len(targ_lang.word2idx)
dataset = tf.data.Dataset.from_tensor_slices((input_tensor_train,
                                              target_tensor_train)).shuffle(
dataset = dataset.batch(BATCH_SIZE, drop_remainder=True)
```

In [63]:

```
def gru(units):
    if tf.test.is_gpu_available():
        return tf.keras.layers.CuDNNGRU(units,
                                        return_sequences=True,
                                        return_state=True,
                                        recurrent_initializer='glorot_unifor
    else:
        return tf.keras.layers.GRU(units,
                                   return_sequences=True,
                                   return_state=True,
                                   recurrent_activation='sigmoid',
                                   recurrent_initializer='glorot_uniform')
```

In [64]:

```
class Encoder(tf.keras.Model):
    def __init__(self, vocab_size, embedding_dim, enc_units, batch_sz):
        super(Encoder, self).__init__()
        self.batch_sz = batch_sz
        self.enc_units = enc_units
        self.embedding = tf.keras.layers.Embedding(vocab_size, embedding_dim
        self.gru = gru(self.enc_units)

    def call(self, x, hidden):
        x = self.embedding(x)
        output, state = self.gru(x, initial_state = hidden)
        return output, state

    def initialize_hidden_state(self):
        return tf.zeros((self.batch_sz, self.enc_units))
```

In [65]:

```
class Decoder(tf.keras.Model):
    def __init__(self, vocab_size, embedding_dim, dec_units, batch_sz):
        super(Decoder, self).__init__()
        self.batch_sz = batch_sz
        self.dec_units = dec_units
        self.embedding = tf.keras.layers.Embedding(vocab_size, embedding_dim
        self.gru = gru(self.dec_units)
        self.fc = tf.keras.layers.Dense(vocab_size)
        # used for attention
        self.W1 = tf.keras.layers.Dense(self.dec_units)
        self.W2 = tf.keras.layers.Dense(self.dec_units)
        self.V = tf.keras.layers.Dense(1)
    def call(self, x, hidden, enc_output):
        # enc_output shape == (batch_size, max_length, hidden_size)
        # hidden shape == (batch_size, hidden size)
        # hidden_with_time_axis shape == (batch_size, 1, hidden size)
        # we are doing this to perform addition to calculate the score
        hidden_with_time_axis = tf.expand_dims(hidden, 1)
        # score shape == (batch_size, max_length, 1)
        # we get 1 at the last axis because we are applying tanh(FC(EO) + FC
        score = self.V(tf.nn.tanh(self.W1(enc_output) + self.W2(hidden_with_
        # attention_weights shape == (batch_size, max_length, 1)
        attention_weights = tf.nn.softmax(score, axis=1)
        # context_vector shape after sum == (batch_size, hidden_size)
        context_vector = attention_weights * enc_output
        context_vector = tf.reduce_sum(context_vector, axis=1)
        # x shape after passing through embedding == (batch_size, 1, embeddi
        x = self.embedding(x)
        # x shape after concatenation == (batch_size, 1, embedding_dim + hid
        x = tf.concat([tf.expand_dims(context_vector, 1), x], axis=-1)
        # passing the concatenated vector to the GRU
        output, state = self.gru(x)
        # output shape == (batch_size * 1, hidden_size)
        output = tf.reshape(output, (-1, output.shape[2]))
        # output shape == (batch_size * 1, vocab)
        x = self.fc(output)
        return x, state, attention_weights
    def initialize_hidden_state(self):
        return tf.zeros((self.batch_sz, self.dec_units))
```

In [66]:

```
# Defining Encoder and Decoder
encoder = Encoder(vocab_inp_size, embedding_dim, units, BATCH_SIZE)
decoder = Decoder(vocab_tar_size, embedding_dim, units, BATCH_SIZE)
```

In [67]:

```
optimizer = tf.train.AdamOptimizer()

def loss_function(real, pred):
    mask = 1 - np.equal(real, 0)
    loss_ = tf.nn.sparse_softmax_cross_entropy_with_logits(labels=real, logi
    return tf.reduce_mean(loss_)
```

In [68]:

```
checkpoint_dir = './training_checkpoints'
checkpoint_prefix = os.path.join(checkpoint_dir, "ckpt")
checkpoint = tf.train.Checkpoint(optimizer=optimizer,
                                 encoder=encoder,
                                 decoder=decoder)
```

In [69]:

```
# Training the network for 12 epochs using Eager Execution
EPOCHS = 12
for epoch in range(EPOCHS):
    start = time.time()
    hidden = encoder.initialize_hidden_state()
    total_loss = 0
    for (batch, (inp, targ)) in enumerate(dataset):
        loss = 0
        with tf.GradientTape() as tape:
            enc_output, enc_hidden = encoder(inp, hidden)
            dec_hidden = enc_hidden
            dec_input = tf.expand_dims([targ_lang.word2idx['<start>']] * BAT
            # Teacher forcing - feeding the target as the next input
            for t in range(1, targ.shape[1]):
                # passing enc_output to the decoder
                predictions, dec_hidden, _ = decoder(dec_input, dec_hidden,
                loss += loss_function(targ[:, t], predictions)
                # using teacher forcing
                dec_input = tf.expand_dims(targ[:, t], 1)
        batch_loss = (loss / int(targ.shape[1]))
        total_loss += batch_loss
        variables = encoder.variables + decoder.variables
        gradients = tape.gradient(loss, variables)
        optimizer.apply_gradients(zip(gradients, variables))
        if batch % 100 == 0:
            print('Epoch {} Batch {} Loss {:.4f}'.format(epoch + 1,
                                                         batch,
                                                         batch_loss.numpy())
    # saving (checkpoint) the model every epoch
    checkpoint.save(file_prefix = checkpoint_prefix)

    print('Epoch {} Loss {:.4f}'.format(epoch + 1,
                                        total_loss / N_BATCH))
    print('Time taken for 1 epoch {} sec\n'.format(time.time() - start))
```

```
Epoch 1 Batch 3700 Loss 0.6780
Epoch 1 Batch 3800 Loss 0.6381
Epoch 1 Batch 3900 Loss 0.6312
Epoch 1 Batch 4000 Loss 0.5332
Epoch 1 Batch 4100 Loss 0.9339
Epoch 1 Batch 4200 Loss 0.5294
Epoch 1 Batch 4300 Loss 0.4298
Epoch 1 Batch 4300 Loss 0.4298
Epoch 1 Batch 4400 Loss 0.4381
Epoch 1 Batch 4500 Loss 0.6522
Epoch 1 Batch 4600 Loss 1.0877
Epoch 1 Batch 4700 Loss 0.8076
Epoch 1 Batch 4800 Loss 0.7929
Epoch 1 Batch 4900 Loss 0.6858
Epoch 1 Batch 5000 Loss 0.4694
Epoch 1 Batch 5100 Loss 0.5341
Epoch 1 Batch 5200 Loss 0.9201
Epoch 1 Batch 5300 Loss 0.5246
Epoch 1 Batch 5400 Loss 0.7496
Epoch 1 Batch 5500 Loss 0.4711
Epoch 1 Batch 5600 Loss 0.6425
Epoch 1 Batch 5700 Loss 0.5657
Epoch 1 Batch 5800 Loss 0.8112
Epoch 1 Batch 5900 Loss 0.5228
Epoch 1 Batch 6000 Loss 0.6331
Epoch 1 Batch 6100 Loss 0.7275
```

In [70]:

```
# restoring the latest checkpoint in checkpoint_dir
checkpoint.restore(tf.train.latest_checkpoint(checkpoint_dir))
```

```
<tensorflow.python.training.checkpointable.util.CheckpointLoadStatus at 0x19792cf6b70>
```

In [71]:

```
def evaluate(inputs, encoder, decoder, inp_lang, targ_lang, max_length_inp,
    attention_plot = np.zeros((max_length_targ, max_length_inp))
    sentence = ''
    for i in inputs[0]:
        if i == 0:
            break
        sentence = sentence + inp_lang.idx2word[i] + ' '
    sentence = sentence[:-1]
    inputs = tf.convert_to_tensor(inputs)
    result = ''
    hidden = [tf.zeros((1, units))]
    enc_out, enc_hidden = encoder(inputs, hidden)
    dec_hidden = enc_hidden
    dec_input = tf.expand_dims([targ_lang.word2idx['<start>']], 0)
    for t in range(max_length_targ):
        predictions, dec_hidden, attention_weights = decoder(dec_input, dec_
        # storing the attention weights to plot later on
        attention_weights = tf.reshape(attention_weights, (-1, ))
        attention_plot[t] = attention_weights.numpy()
        predicted_id = tf.argmax(predictions[0]).numpy()
        result += targ_lang.idx2word[predicted_id] + ' '
        if targ_lang.idx2word[predicted_id] == '<end>':
            return result, sentence, attention_plot
        # the predicted ID is fed back into the model
        dec_input = tf.expand_dims([predicted_id], 0)
    return result, sentence, attention_plot
```

In [72]:

```
  f predict_random_val_sentence():
    actual_sent = ''
    k = np.random.randint(len(input_tensor_val))
    random_input = input_tensor_val[k]
    random_output = target_tensor_val[k]
    random_input = np.expand_dims(random_input,0)
    result, sentence, attention_plot = evaluate(random_input, encoder, decoder,
    print('Input: {}'.format(sentence[8:-6]))
    print('Predicted translation: {}'.format(result[:-6]))
    for i in random_output:
        if i == 0:
            break
        actual_sent = actual_sent + targ_lang.idx2word[i] + ' '
    actual_sent = actual_sent[8:-7]
    print('Actual translation: {}'.format(actual_sent))
    attention_plot = attention_plot[:len(result.split(' '))-2, 1:len(sentence.sp
    sentence, result = sentence.split(' '), result.split(' ')
    sentence = sentence[1:-1]
    result = result[:-2]
    # use plotly to generate the heat map
    trace = go.Heatmap(z = attention_plot, x = sentence, y = result, colorscale=
    data=[trace]
    iplot(data)
```

In [73]:

```
predict_random_val_sentence()
```

```
Input: i mean its an old sad story
Predicted translation: मेरा मतलब है यह एक पुराने और एक पुराने और एक बहुत पुरानी है
Actual translation: मेरा मतलब है यह एक पुराने दुखद कहानी है।
```

In [74]:

```
predict_random_val_sentence()
```

```
Input: well hes a criminal and a killer
Predicted translation: खैर वह एक अंगूठी और एक हत्यारा है
Actual translation: खैर वह एक अपराधी है और एक हत्यारा है
```

In [75]:

```
predict_random_val_sentence()
```

```
Input: and take out the vermin
Predicted translation: और कीड़े बाहर ले
Actual translation: और कीड़े बाहर ले
```

In [76]:

```
predict_random_val_sentence()
```

```
Input: thats how you brought down the odyssey
Predicted translation: यही कारण है कि आप बेवकूफ हो
Actual translation: कि आप ओडिसी नीचे लाया कैसे है
```

In [77]:

```
predict_random_val_sentence()
```

```
Input: a large one
Predicted translation: एक बड़ी एक
Actual translation: बड़ा वाला
```

In [78]:

```
predict_random_val_sentence()
```

```
Input: ask me again any time
Predicted translation: फिर से और समय
Actual translation: बढीया है।
```

In [79]:

```
predict_random_val_sentence()
```

```
Input: pull back
Predicted translation: पीछे हटो
Actual translation: वापस खींचो
```

In [80]:

```
predict_random_val_sentence()
```

```
Input: actually dont say that all right
Predicted translation: नहीं यह सब ठीक नहीं है कि
Actual translation: बहुत अच्छे। वैसे वह मत कहना ठीक
```

In [81]:

```
predict_random_val_sentence()
```

```
Input: i knew who it was
Predicted translation: मैं जानता हूँ कि यह वास्तव में था
Actual translation: मुझे पता था की वो आप थे
```

In [82]:

```
predict_random_val_sentence()
```

```
Input: we have to find another way
Predicted translation: हम एक और रास्ता खोजने के लिए है
Actual translation: हम दूसरा रास्ता खोजने के लिए है
```

In [84]:

```
predict_random_val_sentence()
```

```
Input: hey youve done a stretch in cashman right
Predicted translation: अरे तुम उस में उस पर आया है
Actual translation: अरे तुम ने कैशमन में भी सजा काटी है है ना
```

In [85]:

```
predict_random_val_sentence()
```

```
Input: tyrant they yell so easily
Predicted translation: आलसी लोगों को जीतने के
Actual translation: तानाशाह वे इतनी आसानी से चिल्लाना
```

In [86]:

```
predict_random_val_sentence()
```

```
Input: please tell your sir
Predicted translation: कृपया अपनी टीम साहब
Actual translation: कृपया अपने सर बता
```

In [87]:

```
predict_random_val_sentence()
```

```
Input: hey i got an appointment
Predicted translation: अरे हाँ मैं एक अपॉइंटमेंट है
Actual translation: अरे मैं एक नियुक्ति है
```

In [88]:

```
predict_random_val_sentence()
```

```
Input: id better run im on nights
Predicted translation: मैं बेहतर रात लाम के लिए यात्रा कर रहा हूँ
Actual translation: मुझे जाना होगा मेरी रात की पाली है।
```

In [163]:

```
predict_random_val_sentence()
```

```
Input: but theyre not
Predicted translation: लेकिन वे नहीं कर रहे हैं।
Actual translation: लेकिन वे नहीं कर रहे हैं
```

In [155]:

```
predict_random_val_sentence()
```

```
Input: get out from here
Predicted translation: यहाँ से चले जाओ
Actual translation: निकल जाओ यहाँ से
```

In [151]:

```
predict_random_val_sentence()
```

```
Input: cale
Predicted translation: केल
Actual translation: केल
```

In [92]:

```
predict_random_val_sentence()
```

```
Input: utilities online
Predicted translation: शेर्लोट ऑनलाइन
Actual translation: ऑनलाइन यूटिलिटीज।
```

In [93]:

```
predict_random_val_sentence()
```

```
Input: watch your three
Predicted translation: वहाँ तीन तीन देखो
Actual translation: अपने तीन देखो
```

In [94]:

```
predict_random_val_sentence()
```

```
Input: but we need to start doing things that will directly impact on those offenders
Predicted translation: लेकिन हम पर उपयोगी समय की आवश्यकता होगी तो वे उन स्थानों पर गांठ के लिए तैयार करते हैं
त है
Actual translation: लेकिन हमें कुछ ऐसा करने की ज़रूरत है जो इन पर्यावरण अपराधियों पर सीधे असर डाले
```

In [164]:

```
def predict_random_val_sentence_bleu():
    actual_sent = ''
    k = np.random.randint(len(input_tensor_val))
    random_input = input_tensor_val[k]
    random_output = target_tensor_val[k]
    random_input = np.expand_dims(random_input,0)
    result, sentence, attention_plot = evaluate(random_input, encoder, decod
    print('Input: {}'.format(sentence[8:-6]))
    print('Predicted translation: {}'.format(result[:-6]))
    for i in random_output:
        if i == 0:
            break
        actual_sent = actual_sent + targ_lang.idx2word[i] + ' '
    actual_sent = actual_sent[8:-7]
    print('Actual translation: {}'.format(actual_sent))
    reference = actual_sent.split()
    candidate = result[:-6].split()
    print('Actual translation split: ',actual_sent.split())
    print('Predicted tanslation split: ',result[:-6].split())
    attention_plot = attention_plot[:len(result.split(' '))-2, 1:len(sentenc
    sentence, result = sentence.split(' '), result.split(' ')
    sentence = sentence[1:-1]
    result = result[:-2]
    # use plotly to generate the heat map
    trace = go.Heatmap(z = attention_plot, x = sentence, y = result, colorsc
    data=[trace]
    iplot(data)
```

In [104]:

```
predict_random_val_sentence_bleu()
```

```
Input: foxtrot
Predicted translation: फ़ाक्सत्रोट
Actual translation: फ़ाक्सत्रोट
['फ़ाक्सत्रोट']
['फ़ाक्सत्रोट']
```

```
0
```

In [188]:

```
predict_random_val_sentence_bleu()
```

```
Input: i believe you can do it
Predicted translation: मैं आप यह कर सकते हैं
Actual translation: मैं आप यह कर सकते हैं
Actual translation split:  ['मैं', 'आप', 'यह', 'कर', 'सकते', 'हैं']
Predicted tanslation split:  ['मैं', 'आप', 'यह', 'कर', 'सकते', 'हैं']
```

In [191]:

```
predict_random_val_sentence_bleu()
```

```
Input: mission
Predicted translation: मिशन
Actual translation: मिशन
Actual translation split:  ['मिशन']
Predicted tanslation split:  ['मिशन']
```

In [198]:

```
predict_random_val_sentence_bleu()
```

```
Input: and a puppy
Predicted translation: और एक पिल्ला
Actual translation: और एक पिल्ला।
Actual translation split:  ['और', 'एक', 'पिल्ला।']
Predicted tanslation split:  ['और', 'एक', 'पिल्ला']
```

In [202]:

```
predict_random_val_sentence_bleu()
```

```
Input: yes indu
Predicted translation: हाँ इंदु
Actual translation: हाँ इंदु
Actual translation split:  ['हाँ', 'इंदु']
Predicted tanslation split:  ['हाँ', 'इंदु']
```

In [ ]: