UpGrad

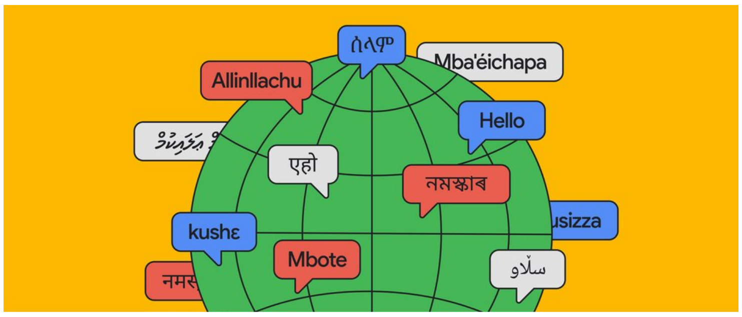
Session Summary

Introduction to Machine Translation

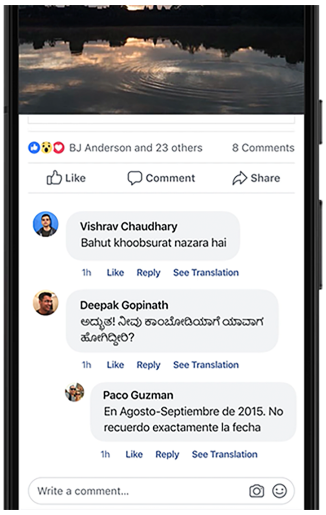
- Effective language translation is necessary for communication.
- **Machine translation (MT)** provides quick translations without a lot of human involvement, thereby eliminating language barriers.

Examples of Machine Translation in Real-time

- Google Translate



- Comment/post/speech translation on Facebook and other applications



BJ Anderson and 23 others 8 Comments

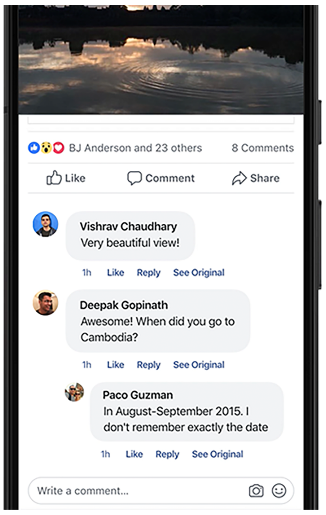
Like Comment Share

Vishrav Chaudhary
Bahut khoobsurat nazara hai
1h Like Reply See Translation

Deepak Gopinath
ಅದ್ಭುತ! ನೀವು ಕಾಂಬೋಡಿಯಾಗೆ ಯಾವಾಗ ಹೋಗಿದ್ದೀರಿ?
1h Like Reply See Translation

Paco Guzman
En Agosto-Septiembre de 2015. No recuerdo exactamente la fecha
1h Like Reply See Translation

Write a comment...



BJ Anderson and 23 others 8 Comments

Like Comment Share

Vishrav Chaudhary
Very beautiful view!
1h Like Reply See Original

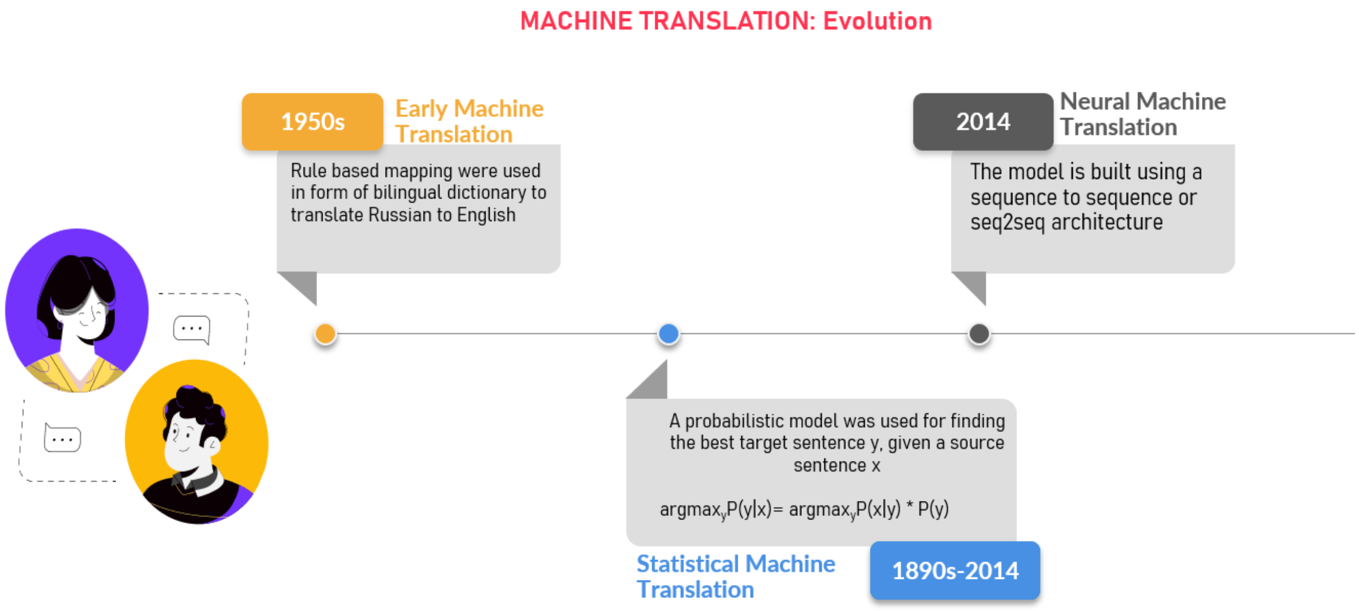
Deepak Gopinath
Awesome! When did you go to Cambodia?
1h Like Reply See Original

Paco Guzman
In August-September 2015. I don't remember exactly the date
1h Like Reply See Original

Write a comment...

© Copyright UpGrad Education Pvt. Ltd. All rights reserved

Here are the highlights of how MT evolved over the years:

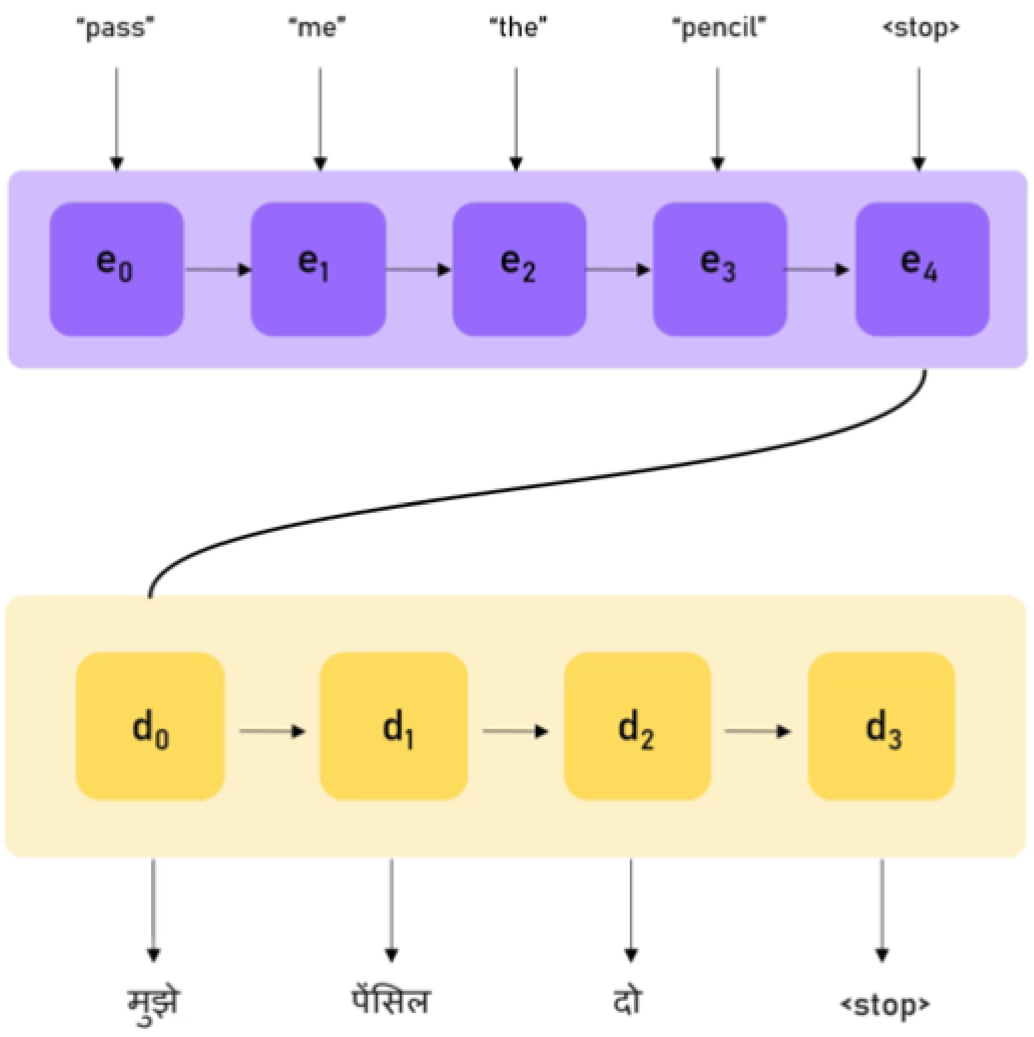

MACHINE TRANSLATION: Evolution
1950s
Early Machine Translation
Rule based mapping were used in form of bilingual dictionary to translate Russian to English
2014
Neural Machine Translation
The model is built using a sequence to sequence or seq2seq architecture
A probabilistic model was used for finding the best target sentence y, given a source sentence x
argmax_y P(y|x)= argmax_y P(x|y) * P(y)
Statistical Machine Translation
1890s-2014

- **RBMT**: Rule-based mapping was used in the form of a bilingual dictionary to translate Russian into English.
- **EBMT**: It uses a database of bilingual sentence pairs to translate by finding a similar sentence in the database and using its corresponding target sentence as the translation.
- **SMT**: A probabilistic model was used for finding the best target sentence y, given a source sentence x.
- **NMT**: A neural network-based model was used to generate a target sentence in the target language, given a source sentence in the source language. The model is built using a **seq2seq architecture.**

Understanding the Encoder-Decoder Architecture for NMT

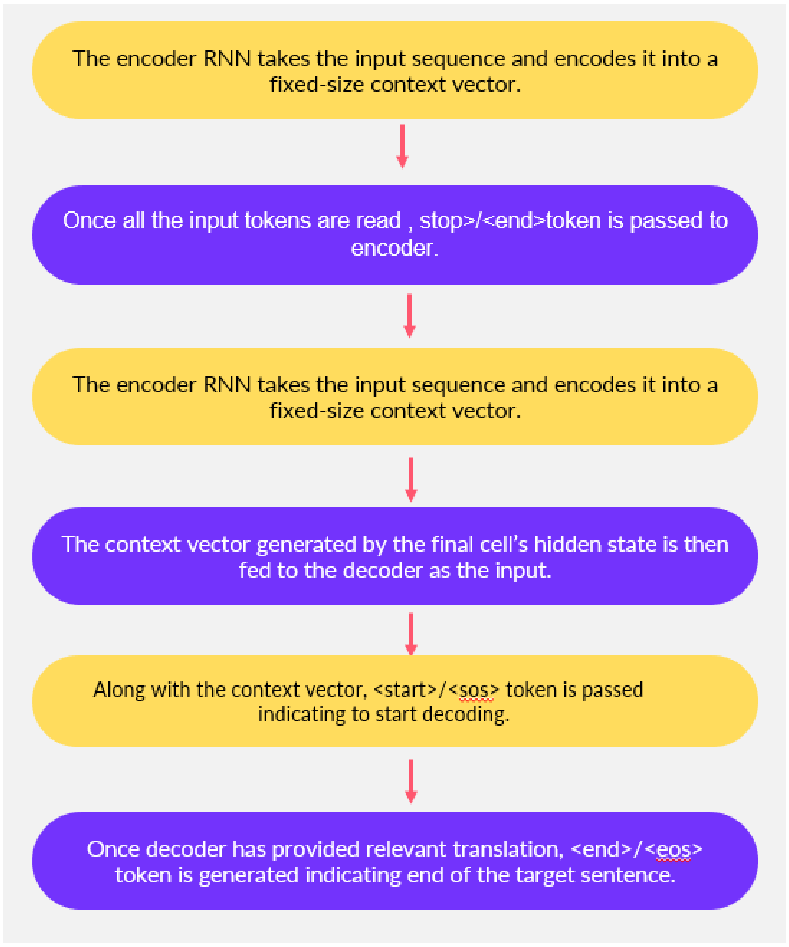
- Sequence-to-sequence (seq2seq) model follows an encoder-decoder architecture, where it is made up of two RNNs.
- Both the encoder and the decoder consist of a series of RNN cells where each layer's output is the input to the next layer.

© Copyright UpGrad Education Pvt. Ltd. All rights reserved



The following process is followed in this architecture.

© Copyright UpGrad Education Pvt. Ltd. All rights reserved


The encoder RNN takes the input sequence and encodes it into a fixed-size context vector.
Once all the input tokens are read , stop>/<end>token is passed to encoder.
The encoder RNN takes the input sequence and encodes it into a fixed-size context vector.
The context vector generated by the final cell's hidden state is then fed to the decoder as the input.
Along with the context vector, <start>/<sos> token is passed indicating to start decoding.
Once decoder has provided relevant translation, <end>/<eos> token is generated indicating end of the target sentence.

NMT is a **conditional language model** because the decoder predicts the next word based on the context input from the encoder and previous predictions, making it conditional and language-based.

© Copyright UpGrad Education Pvt. Ltd. All rights reserved

Requirements of NMT Architecture

Here are the required characteristics for an NMT architecture to produce a good translation.

01 It should handle variable length sequence

02 It should be able to maintain information about order

03 It should share parameters across the sequence

04 It should be able to track long term dependency

RNN Vanilla, LSTM, GRU Model

Vanilla RNN

RNN Vanilla model is unable to **generate long sequences** due to the problem of **vanishing and exploding gradients** in RNNs

LSTM

Vanishing gradient problem is solved by **LSTM**. It has **gating mechanism**.

GRU

GRU is a variation of LSTM and is simpler than LSTM by design and training

LSTM

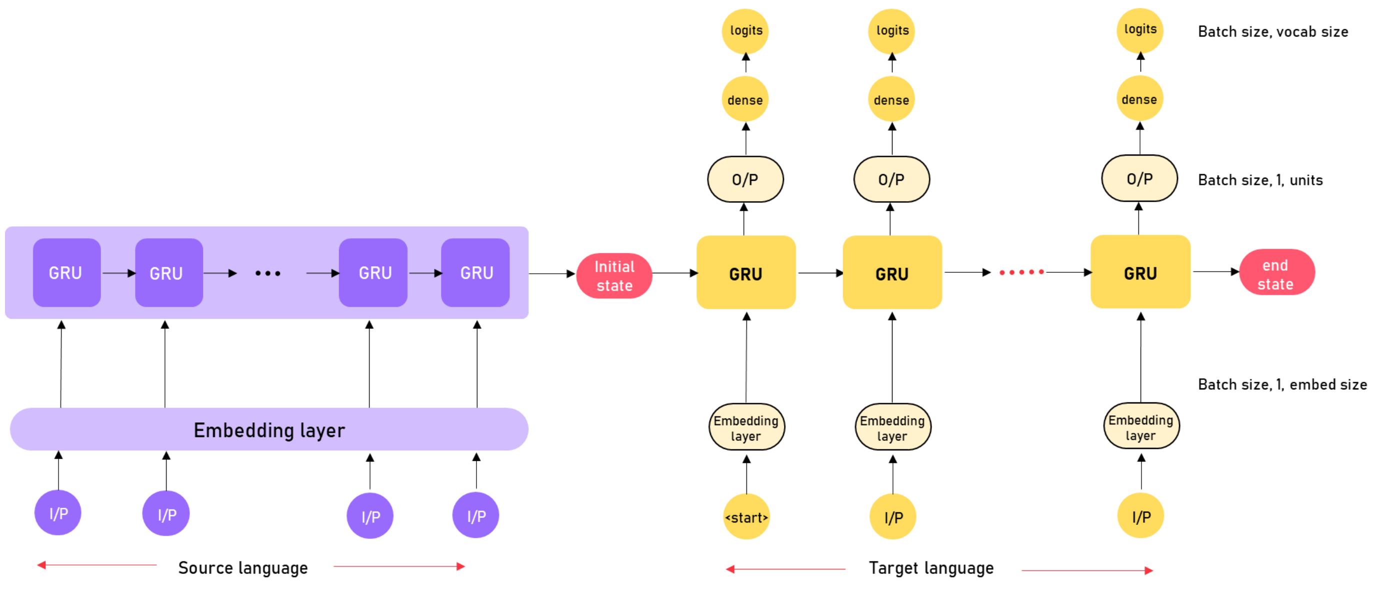
- Different gating mechanisms are used in LSTM
  - Controlled updating/deleting of information
  - Ensures only relevant information is kept
- Helps in solving the problem of exploding/vanishing gradients and retains important information

© Copyright UpGrad Education Pvt. Ltd. All rights reserved

Parameters	Vanilla RNN	LSTM	GRU
Gates	NA	● 3 (Input, Output, Forget) ● 2 activation functions	● 2 (Reset, Update) ● 1 activation function
Can it handle long-term dependency?	No	Yes	Yes
Does it possess internal memory?	No	Yes	No

Model Training and Prediction

The encoder-decoder training architecture for NMT is illustrated below.



Encoder-Decoder training architecture for NMT

Here are the requirements for training a module:

- NMT models require **fixed-length input sequences**, usually limited by the **longest sentence** in the data.
- The shape of the input data should be **(batch_size, max_length)**.

© Copyright UpGrad Education Pvt. Ltd. All rights reserved

- For sentences shorter than the maximum length, paddings are added.

Inserting Paddings:

- Here is an example of inserting paddings that consist of tokenized words.

[<Start>	Those	are	sunflowers	<end>			
<Start>	Tom	bought	a	new	car	<end>	
<Start>	Can	you	Please	call	me	tomorrow	<end>]

- Converting these sentences into their respective tokens, the data will look like this.

[2	71	1331	4231	3			
2	73	8	3215	55	927	3	
2	83	91	1	645	1253	927	3

The maximum length for the samples is 8. So, samples shorter than 8 must be padded with empty(0) tokens.

You can pad the samples to max length by using **tf.keras.preprocessing.sequence.pad_sequences.**

```
padded_inputs = tf.keras.preprocessing.sequence.pad_sequences(tokens,
padding="post") print(padded_inputs)
```

[2	71	1331	4231	3	**0**	**0**	**0**
2	73	8	3215	55	927	3	**0**
2	83	91	1	645	1253	927	3]

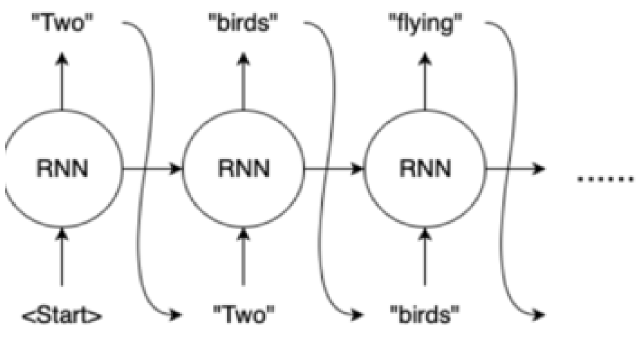
Note: Since the padding is kept as **"post**," all the empty tokens are added after each sequence. It is recommended to keep post padding when working with RNN cells.

© Copyright UpGrad Education Pvt. Ltd. All rights reserved

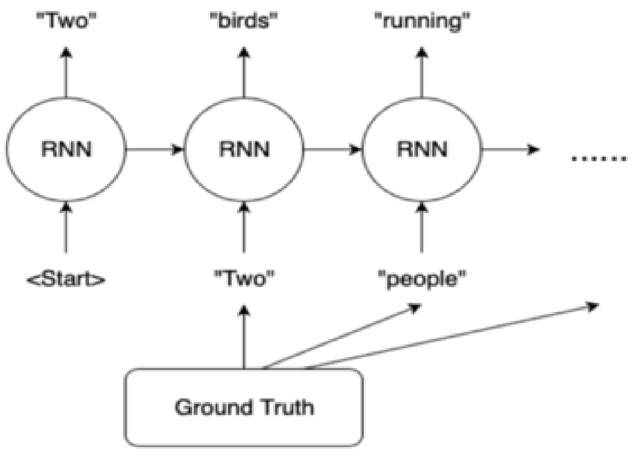
Training Process

Since the decoder is trained to predict the next characters of the target sequence given the previous prediction, if the model produces a wrong prediction at a certain timestep, all the following predictions can be incorrect.

This problem can be solved by training the model using **teacher forcing**, where the model is fed with the correct target token at each timestep, regardless of the model's predictions at the previous timestep, making the model more robust.



Without Teacher Forcing

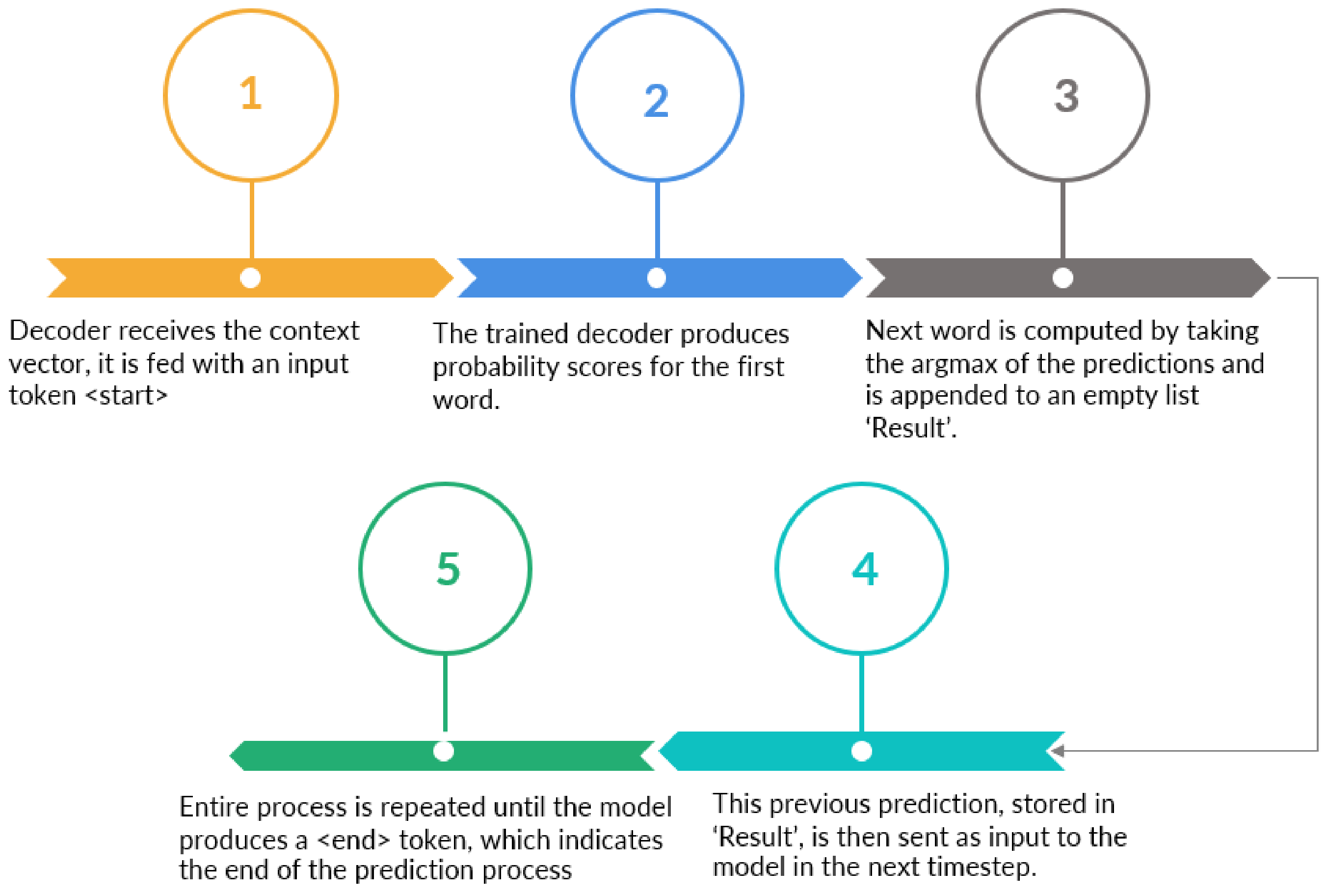


With Teacher Forcing

Source: https://blog.naver.com/sooftware/221790750668

© Copyright UpGrad Education Pvt. Ltd. All rights reserved

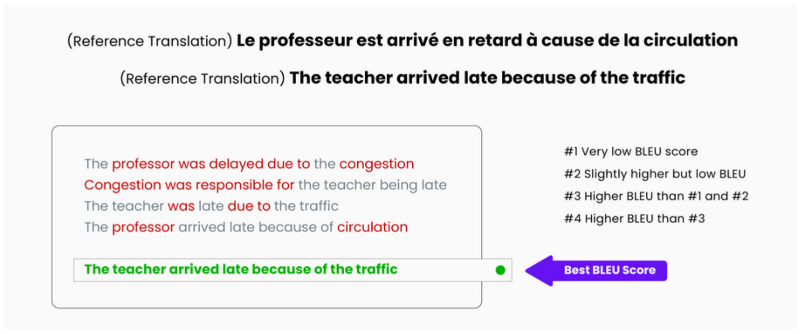
The process followed for training the model is illustrated below.



1. Decoder receives the context vector, it is fed with an input token <start>
2. The trained decoder produces probability scores for the first word.
3. Next word is computed by taking the argmax of the predictions and is appended to an empty list 'Result'.
4. This previous prediction, stored in 'Result', is then sent as input to the model in the next timestep.
5. Entire process is repeated until the model produces a <end> token, which indicates the end of the prediction process

BLEU Score:

- The BLEU score is a well-acknowledged evaluation metric employed for model prediction.
- It determines the "difference" between the predicted sentence and the high-quality reference translations.
- Its value lies between 0 and 1. A metric close to 1 means that the two are very similar.



(Reference Translation) **Le professeur est arrivé en retard à cause de la circulation**

(Reference Translation) **The teacher arrived late because of the traffic**

The professor was delayed due to the congestion
Congestion was responsible for the teacher being late
The teacher was late due to the traffic
The professor arrived late because of circulation
The teacher arrived late because of the traffic

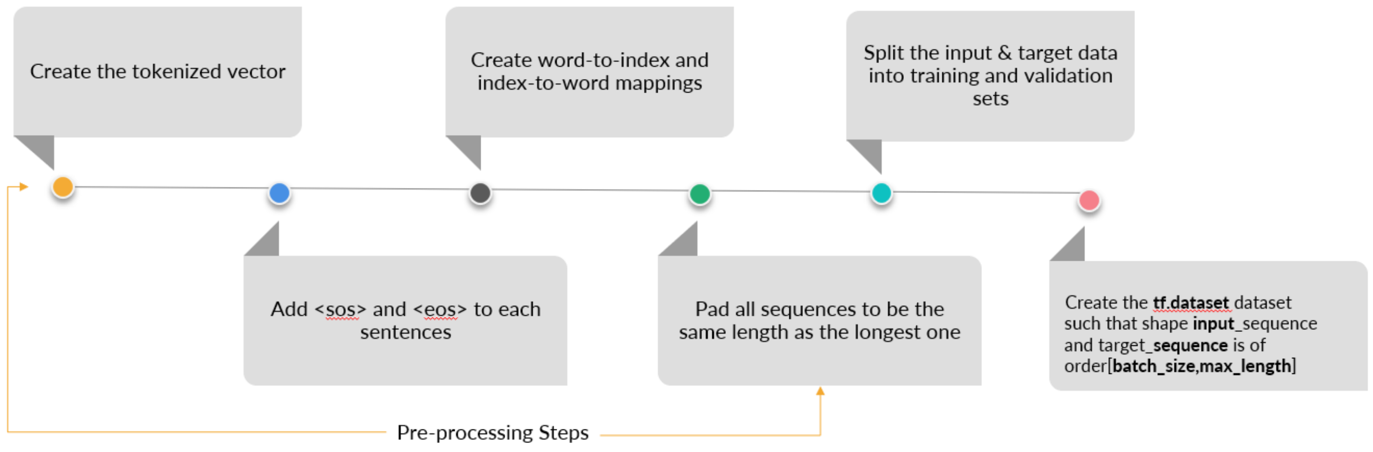
#1 Very low BLEU score
#2 Slightly higher but low BLEU
#3 Higher BLEU than #1 and #2
#4 Higher BLEU than #3

Source: https://blog.modernmt.com/understanding-mt-quality-bleu-scores/

© Copyright UpGrad Education Pvt. Ltd. All rights reserved

Implementing NMT using TensorFlow

The goal is to convert a sentence from the source language into the target language. Here is a summary of the steps.


Create the tokenized vector
Create word-to-index and index-to-word mappings
Split the input & target data into training and validation sets
Add <sos> and <eos> to each sentences
Pad all sequences to be the same length as the longest one
Create the tf.dataset dataset such that shape input_sequence and target_sequence is of order[batch_size,max_length]
Pre-processing Steps

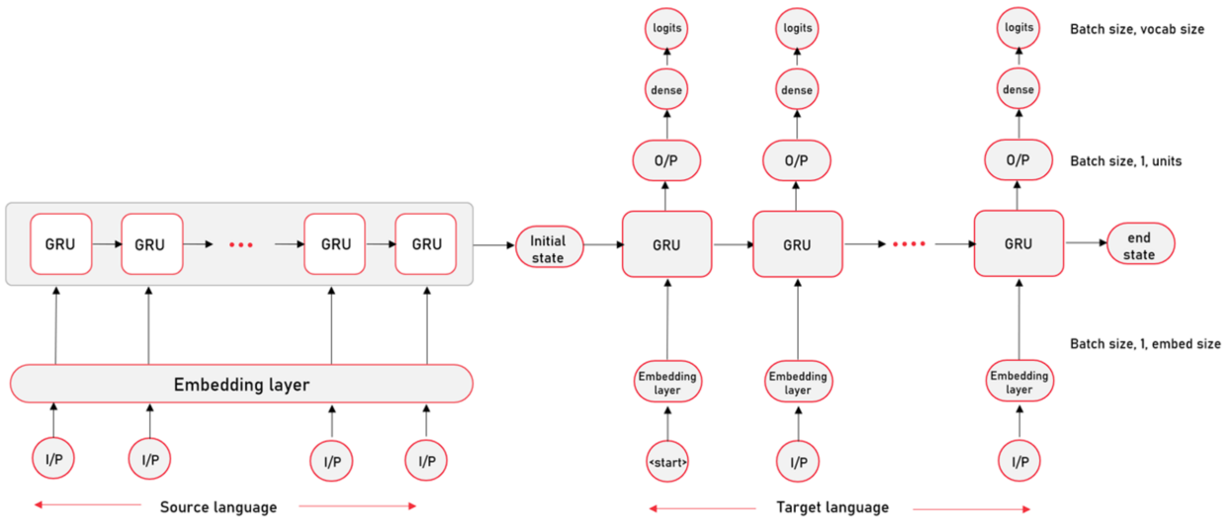
The following code will ensure the shapes of input_sequence and target_sequence are in the required order.

```
example_input_batch, example_target_batch = next(iter(dataset))
example_input_batch.shape, example_target_batch.shape
```

```
(TensorShape([64, 72]), TensorShape([64, 69]))
```

Here, 64 is the batch_size, 72 is the max_length of the input_sequence, and 69 is the max_length of the output_sequence.

The training process for the seq2seq model is illustrated below.



Encoder-Decoder training architecture for NMT

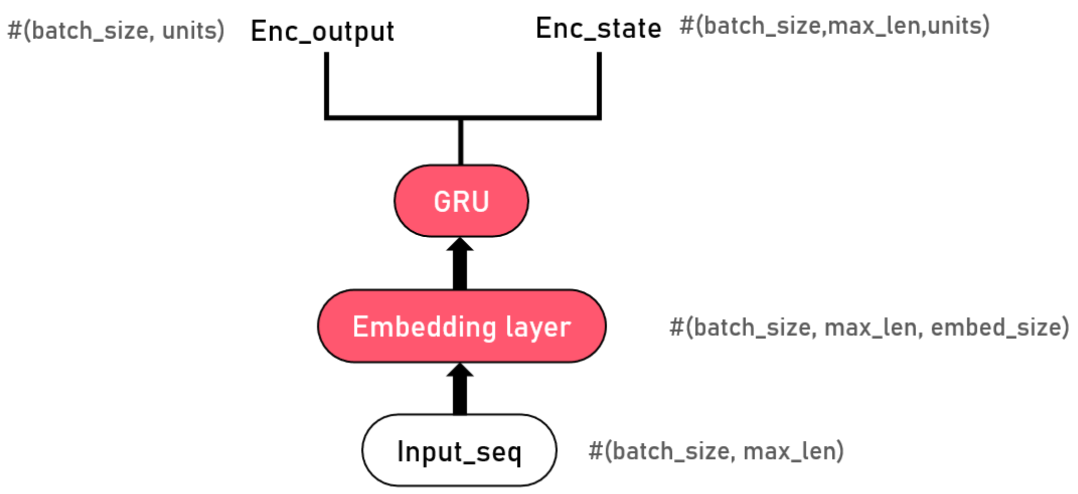
Let's break down the encoder and the decoder separately to explore their components.

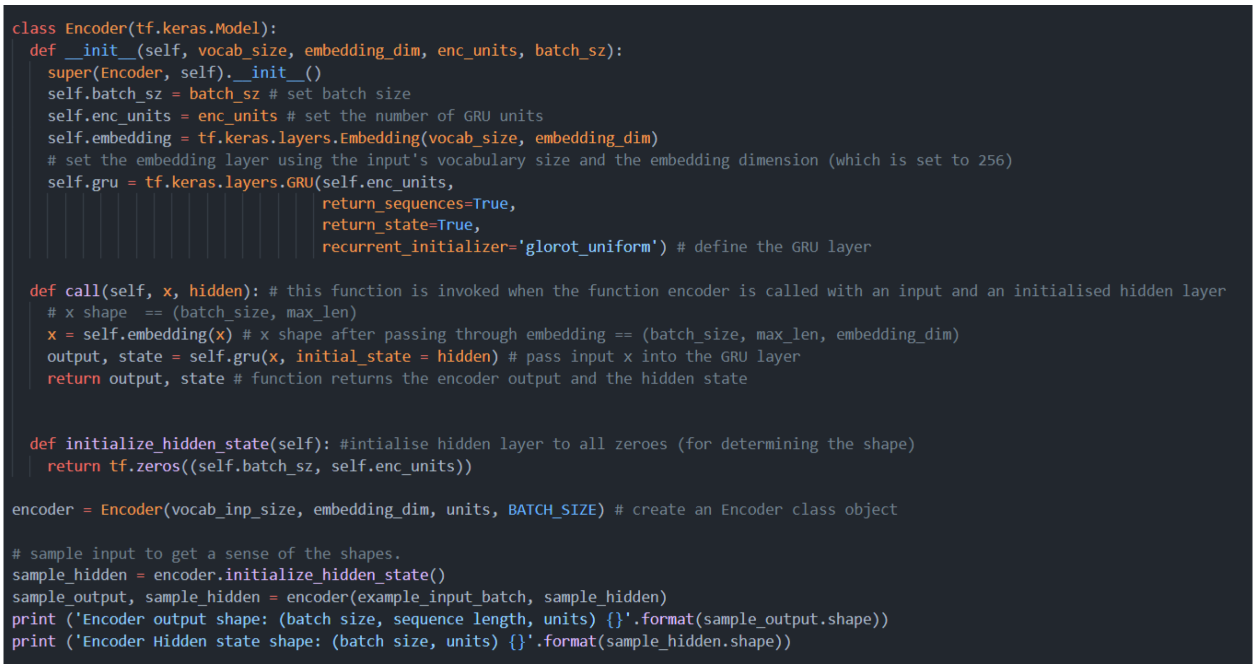
Encoder Model

- It consists of an embedding layer (layers.Embedding), which creates an embedding vector for each token ID.
- Once the embeddings are generated, the GRU (layers.GRU) units process them into a new sequence.

© Copyright UpGrad Education Pvt. Ltd. All rights reserved

After processing the entire sequence, the model returns the encoder output and the hidden state.





```python
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 # set batch size
    self.enc_units = enc_units # set the number of GRU units
    self.embedding = tf.keras.layers.Embedding(vocab_size, embedding_dim)
    # set the embedding layer using the input's vocabulary size and the embedding dimension (which is set to 256)
    self.gru = tf.keras.layers.GRU(self.enc_units,
                                   return_sequences=True,
                                   return_state=True,
                                   recurrent_initializer='glorot_uniform') # define the GRU layer

  def call(self, x, hidden): # this function is invoked when the function encoder is called with an input and an initialised hidden layer
    # x shape  == (batch_size, max_len)
    x = self.embedding(x) # x shape after passing through embedding == (batch_size, max_len, embedding_dim)
    output, state = self.gru(x, initial_state = hidden) # pass input x into the GRU layer
    return output, state # function returns the encoder output and the hidden state


  def initialize_hidden_state(self): #intialise hidden layer to all zeroes (for determining the shape)
    return tf.zeros((self.batch_sz, self.enc_units))

encoder = Encoder(vocab_inp_size, embedding_dim, units, BATCH_SIZE) # create an Encoder class object

# sample input to get a sense of the shapes.
sample_hidden = encoder.initialize_hidden_state()
sample_output, sample_hidden = encoder(example_input_batch, sample_hidden)
print ('Encoder output shape: (batch size, sequence length, units) {}'.format(sample_output.shape))
print ('Encoder Hidden state shape: (batch size, units) {}'.format(sample_hidden.shape))
```

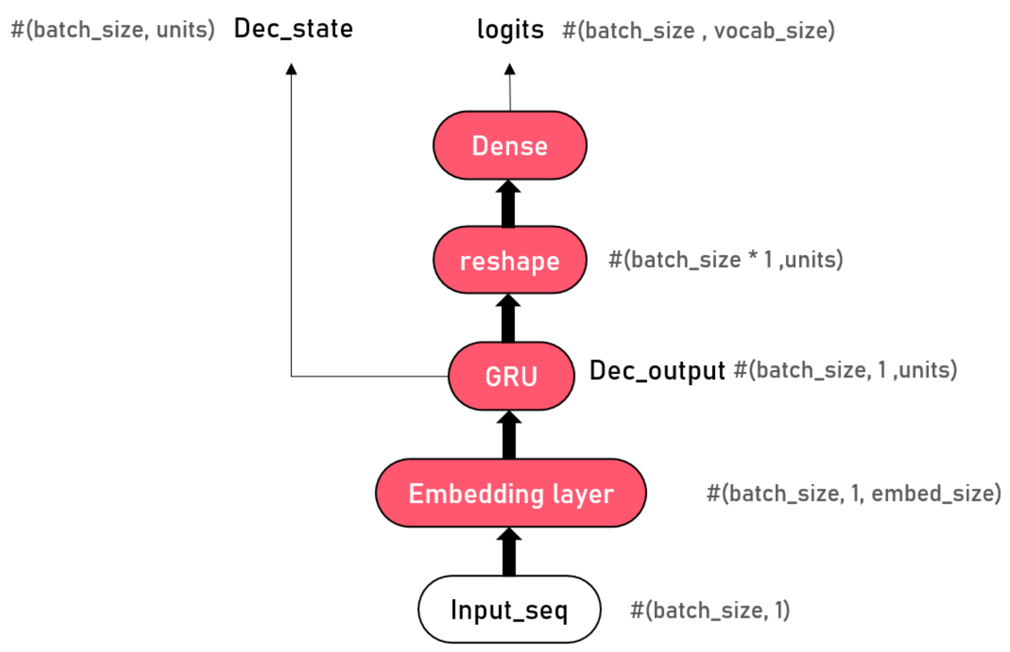
Once you have built your encoder successfully, you can observe the following:

- Encoder output shape: **(batch size, sequence length, units)** = (64, 72, 1024)
- Encoder hidden state shape: (batch size, units) = (64, 1024)

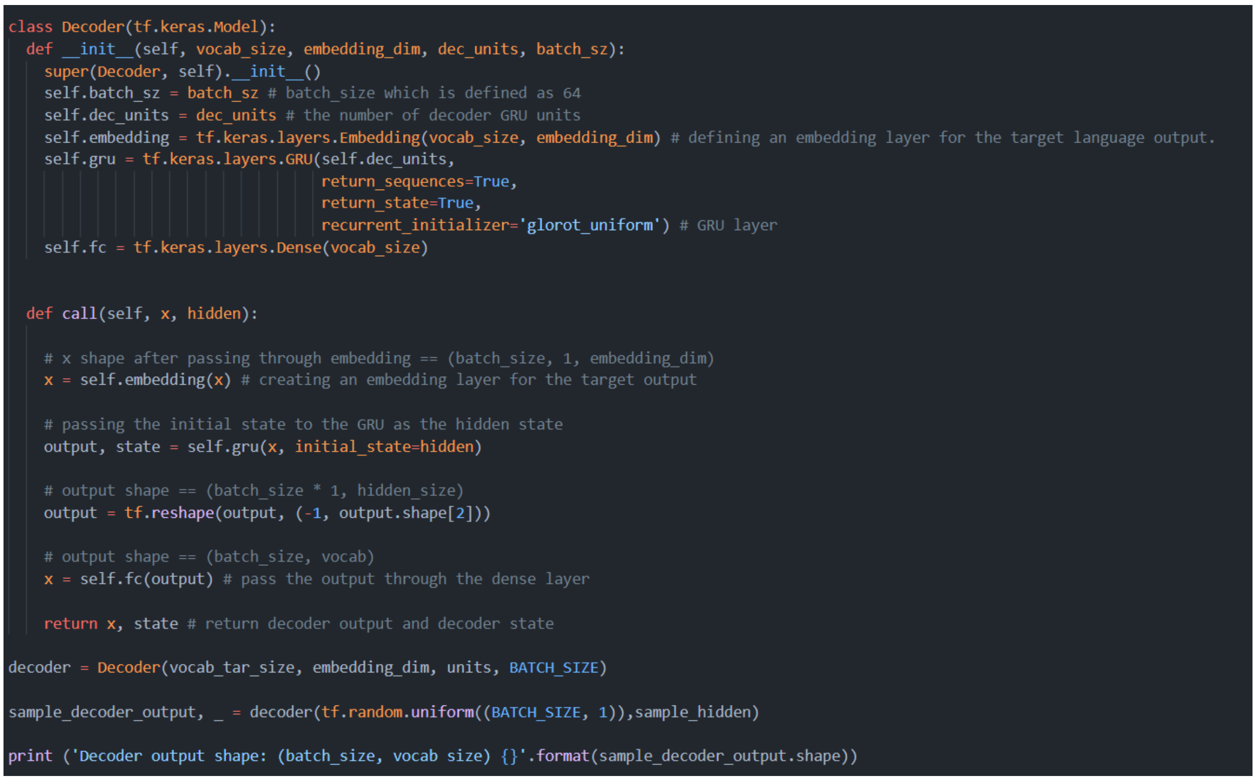
© Copyright UpGrad Education Pvt. Ltd. All rights reserved

Decoder Model

- It is initialized with the hidden state from the encoder.
- The embedding layer present in it creates an embedding vector for the target output. The initial input to the layer will be the **<sos>** tag in the shape of (**batch_size,1**), as you are passing one token at each timestep.
- These vectors are then passed on to GRU units, which create the output and the hidden state.
- The hidden state is fed to the next GRU cell, and the output is passed through a dense layer.



© Copyright UpGrad Education Pvt. Ltd. All rights reserved



```python
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 # batch_size which is defined as 64
    self.dec_units = dec_units # the number of decoder GRU units
    self.embedding = tf.keras.layers.Embedding(vocab_size, embedding_dim) # defining an embedding layer for the target language output.
    self.gru = tf.keras.layers.GRU(self.dec_units,
                                   return_sequences=True,
                                   return_state=True,
                                   recurrent_initializer='glorot_uniform') # GRU layer
    self.fc = tf.keras.layers.Dense(vocab_size)


  def call(self, x, hidden):

    # x shape after passing through embedding == (batch_size, 1, embedding_dim)
    x = self.embedding(x) # creating an embedding layer for the target output

    # passing the initial state to the GRU as the hidden state
    output, state = self.gru(x, initial_state=hidden)

    # output shape == (batch_size * 1, hidden_size)
    output = tf.reshape(output, (-1, output.shape[2]))

    # output shape == (batch_size, vocab)
    x = self.fc(output) # pass the output through the dense layer

    return x, state # return decoder output and decoder state

decoder = Decoder(vocab_tar_size, embedding_dim, units, BATCH_SIZE)

sample_decoder_output, _ = decoder(tf.random.uniform((BATCH_SIZE, 1)),sample_hidden)

print ('Decoder output shape: (batch_size, vocab size) {}'.format(sample_decoder_output.shape))
```

After creating the decoder model successfully, you can observe the following:

- Decoder output shape: (batch_size, vocab size) = (64, 22224)

After building all the encoder and decoder models, you need to set up the training process for the entire NMT architecture. For this, you have to do the following:

- Set the optimizer(Adam) and loss object(SparseCategoricalCrossentropy).
- Create your checkpoint path.
- Create your training step functions, which will define how the model will be updated for each input/target batch.

The entire sequence of the model training can be summarized as follows:

Overall the implementation for the Model.train_step method is as follows:

1. input_sequence, target_sequence is received as a batch from the training dataset and passed to train_step.
2. The input_sequence is fed to the encoder along with the hidden state to produce enc_output and enc_hidden.
3. The decoder is initialized with the enc_hidden and is employed with teacher forcing where the target is fed as the next input. Therefore, the prediction process is looped over the entire target_sequence:
    - The decoder is run one step at a time. Once the predictions are created, the loss is calculated for each step/word.
    - Teacher forcing is applied to change the decoder input to targ[: , t], which denotes the target word at the tth timestep present in the batch.

© Copyright UpGrad Education Pvt. Ltd. All rights reserved

4. The average loss is accumulated (by dividing the loss by sentence length) to calculate the batch_loss.
5. The gradient is calculated using the tape.gradient and all thetrainable_variables (Encoder and Decoder) are updated using the optimizer.
6. The model is saved at the checkpoint path after every 2 epochs.

Once the model is trained, you need to create an evaluate function to run the model inference.

Overall the function is similar to **train_step** except that the input to the decoder at each time step is the last prediction from the decoder.

The major changes to the function are as follows:

- The input text is processed first by preprocess_sentence and padded using the keras.preprocessing.sequence.pad_sequences .
- Once the text is converted into the respective token IDs, the trained decoder will produce a list of predictions (probability scores) for the first word. By taking the argmax of the predictions, the next word is computed and appended to an empty list "Result."
- This previous prediction, stored in "Result," is then sent as an input to the model in the next timestep.

The entire process is repeated until the model produces an <eos> token, which indicates the end of the prediction process.

This brings us to the end of NMT implementation.

© Copyright UpGrad Education Pvt. Ltd. All rights reserved

Disclaimer: All content and material on the upGrad website is copyrighted material, either belonging to UpGrad or its bonafide contributors and is purely for the dissemination of education. You are permitted to access print and download extracts from this site purely for your own education only and on the following basis:

- You can download this document from the website for self-use only.
- Any copies of this document, in part or full, saved to disc or to any other storage medium may only be used for subsequent, self-viewing purposes or to print an individual extract or copy for non-commercial personal use only.
- Any further dissemination, distribution, reproduction, copying of the content of the document herein or the uploading thereof on other websites or use of the content for any other commercial/unauthorized purposes in any way which could infringe the intellectual property rights of UpGrad or its contributors, is strictly prohibited.
- No graphics, images, or photographs from any accompanying text in this document will be used separately for unauthorized purposes.
- No material in this document will be modified, adapted, or altered in any way.
- No part of this document or UpGrad content may be reproduced or stored in any other website or included in any public or private electronic retrieval system or service without UpGrad's prior written permission.
- Any rights not expressly granted in these terms are reserved.

© Copyright UpGrad Education Pvt. Ltd. All rights reserved