ATTENTION IN NEURAL NETWORKS

Attention Technique in Seq2Seq:

Sequence to sequence using RNNs modelling is used in -

Neural Machine Translation

Speech Recognition

Text Summarization

Image Captioning

Chatbots and

Other sequence modelling tasks

Need for Attention:

In tradition sequence to sequence model If the sequence is large It can forgets the previous words it is not able to retain all the information and interaction between the entities resulting in poor accuracy. So comes the

Attention model :

It pay attention to the part of a input sentence while generating a translation where as the sequence to sequence model uses Encoder Decoder formulation for translation .

Attention initially developed for Machine translation later it extended for many areas as well.

Idea:

In sequence to sequence model we used to discard the intermediate states(hidden and cell states) and we took only the final state which comprised of all compressed vector (Final vector/Context Vector/Thought Vector) of all the words together which later fed into decoder system.

So instead of discarding all the intermediate state Attention model takes them into consideration which are the vectors used by the decoder to generate the output sequence.

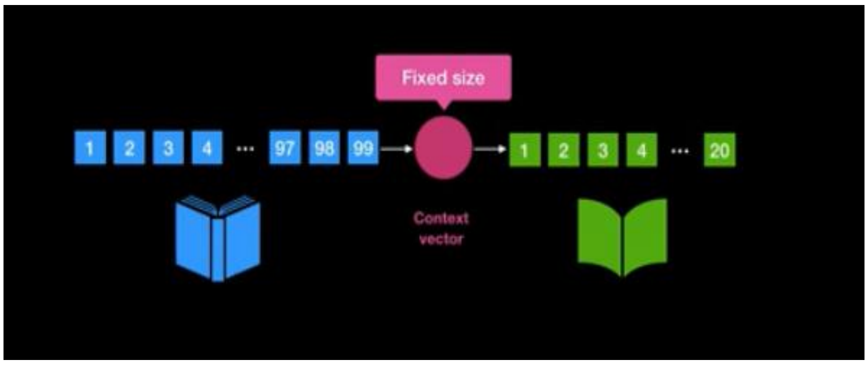
Long Sequence Problem: Attention Intuition

In Encoder Decoder model it first tries to learn the whole sequence and then start translating them.

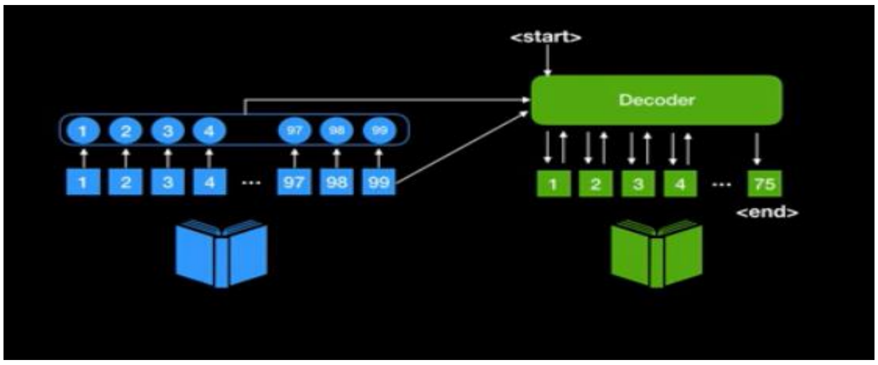
Lets look at the case how human translator will do he will first see the part of sentence and he may translate part

of translation and reads the next part and again translates them and so on.... Because it is difficult to memorize whole long sentence.

Attention model leverages this which ultimately outperforms the classical one.

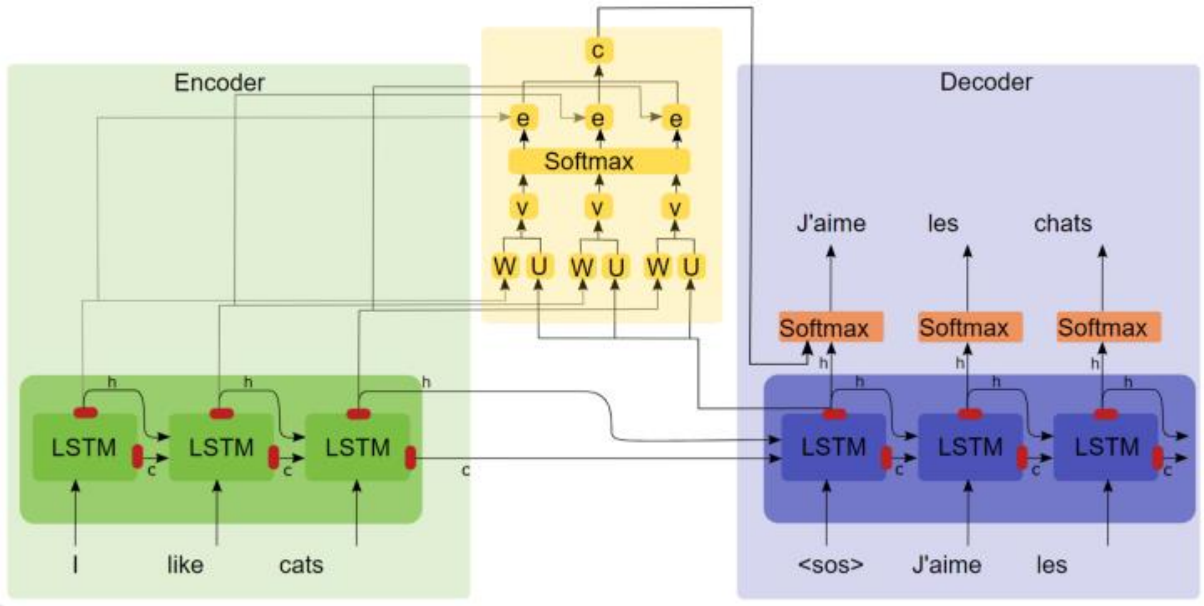


a) Traditional Seq2Seq



b) With Attention

Mechanism:



e.g.

I like cats After data preprocessing it has to be <sos>I like cats<sos> to know sentence is starting and ending.

For each hidden state passed into attention model as a vector and state vector of **<sos>** is passed and here we are applying softmax on the combined vector(which is the weighted sum) which gives the probabilities for each word

At time t=0

Weighted Vector Context Vector

[I] + [<sos>] -> [12,3,4,……….] -> Softmax -> [**0.95**,0.0.01,0.02,0.001,…………]

After getting <sos> into decoder probability for next word **'I'** is higher so the output will be **'I'**

At next time step ,

Probability of word **'like'** will be higher like -> [0.2, **0.75**,0.01,0.02,0.001,…………]

So on

It will be passed into the decoder which is learning to predict the translated word.

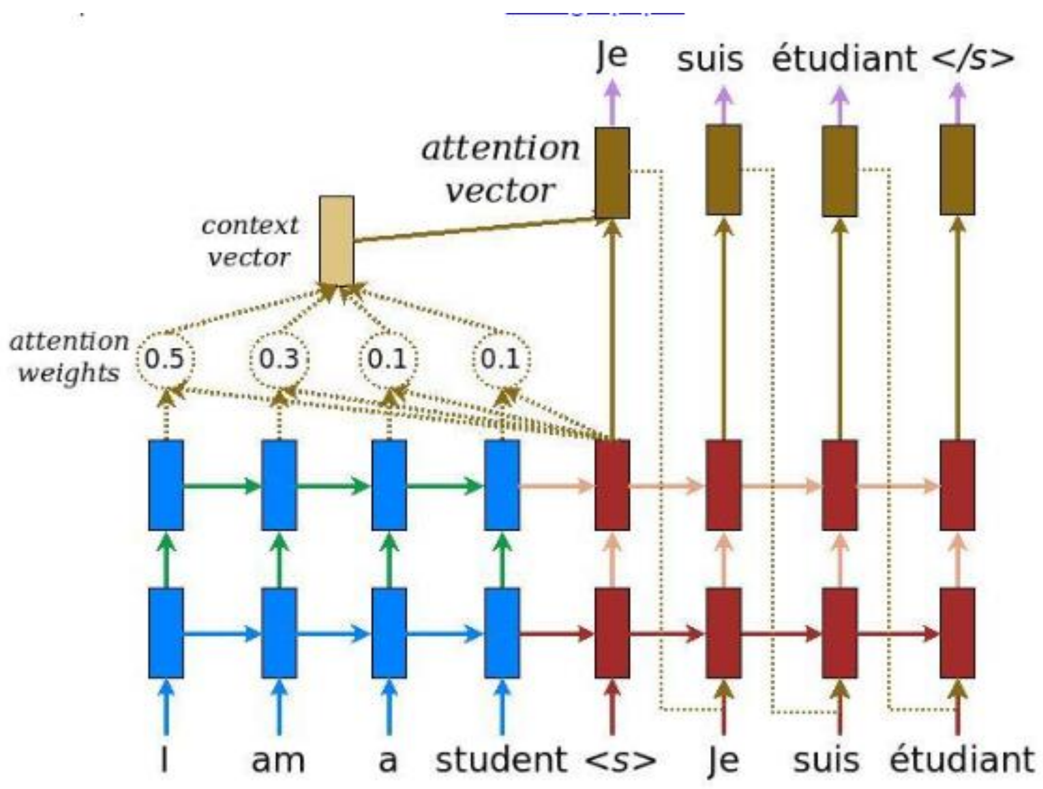
Implementation Guidelines:

1.Preprocess the data and after preprocessing and cleaning the data create each sample point of both English and Spanish in the below format .

<START> I am a student <END> Input tensor

<START> Je suis ístudiant <END> Target Tensor

Create word Embedding for them and slot them into batches and store them into dataset



Shapes:

Batch Size: 64 **Input Tensor** : 64*16 **Target Tensor** : 64*14 (14 may be maximum length of words in target)

Take GRU as Recurrent cell as it has only 1 hidden layer to avoid complexity (tf.keras.layers.GRU)

Here take no of GRU Units = 50

Here we are taking Banhadau Attention .There is one more which is called Luong Attention.

Bother differs in only how Context Vector is created . Later we will see.

2. **Encoder :**

```
output, state = gru(Input_batch , hidden)     //Initially Hidden =[ 0 0 0 0 ……. 50th]
encoder(input_batch, hidden)                  // calling
```

3. **Attention:**

```
Attention(query,value)
    Score = V( tanh ( W1 (values) + W2 ( query)) )
```

// Before passing values to W1 Change the dimensions

// W1 W2: Fully Connected Layer with n neurons V: Dense Layer : 1 neuron which gives

```
    attention_weights = softmax(score)
    context_vector = attention_weights * values
    context_vector = tf.reduce_sum(context_vector, axis=1)
```

Shapes:

Query : Hidden layer : 64* 50 (Units)

Values : Output layer of Encoder : 64 * 16 * 50(Units)

Then you might wondering how we are are adding the output from Fully connect dense layer that is where we are increasing the dimension of query vector by 1.

Attention weights : 64*16*1

Context vector : 64*50

// We are using tf.reduce sum to resuce context vector from size 64*16*50 to 64*50

4. **Decoder**

```
Decoder(x , hidden, encoder_output)
    GRU()
    context_vector, attention_weights = Attention(hidden , encoder_output)
    x=embedding(x)
    x = tf.concat ([tf.expand_dims(context_vector, 1), x], axis=-1)
    output, state = GRU(x)
    output = tf.reshape(output, (-1, output.shape[2]))
    x = FC (output)
    return x
```

Shape :

Encoder_output : 64*16*50

output shape : 64, vocab_size

Driver :

```
Foreach in epoch:
      encoder_hidden = [0 0 0 0 …… 50th ]
      Foreach (batch ,input,target) in dataset:
            encoder_output, encoder_hidden = encoder(input,encoder_hidden)
            decoder_hidden = encoder_hidden
            decoder_input = target(<START>) * 64 *1
            For i in target.shape[1]:
                  predictions, decoder_hidden, _ = decoder(decoder_input, decoder_hidden,
                                                                               enc_output)
                  dec_input = tf.expand_dims(target[:, i], 1)
            // for each i it is taking it is increasing the count to predict the next one.
```

Recap:

Pass the input through encoder now we have encoder output and encoder hidden states .Now the encoder output encoder hidden states is passed through the decoder input

Now at decoder Input we have || <START> || It will try to predict **Je**

We have now prediction and decoder hidden state now this decoder hidden state is passed through next cell to predict next word i.e. **suis** . Instead of passing prediction vector we pass the true label so the model learn fast and less prone to error this is **Teacher Forcing**.

Attention Technique in CNN:

To explain use of attention technique in CNN we would take a example **Image Captioning**

While ConvNets are use for signal processing and Image Detection/Classification they are not good in pattern recognitions .

Recurrent Neural Networks is there to learn pattern through their feed forward network .

Image Captioning

Looking at the image we have to return a free form text what image is all about . We used to take the output from final layer(Fully connected) which contained the feature of the image which is fed into the RNN network as a hidden state in previous time step which will generate the first word and this output goes again to RNN to generate second word on the next time step and so on.... .Second part is same as Machine Translation part .

Problem:

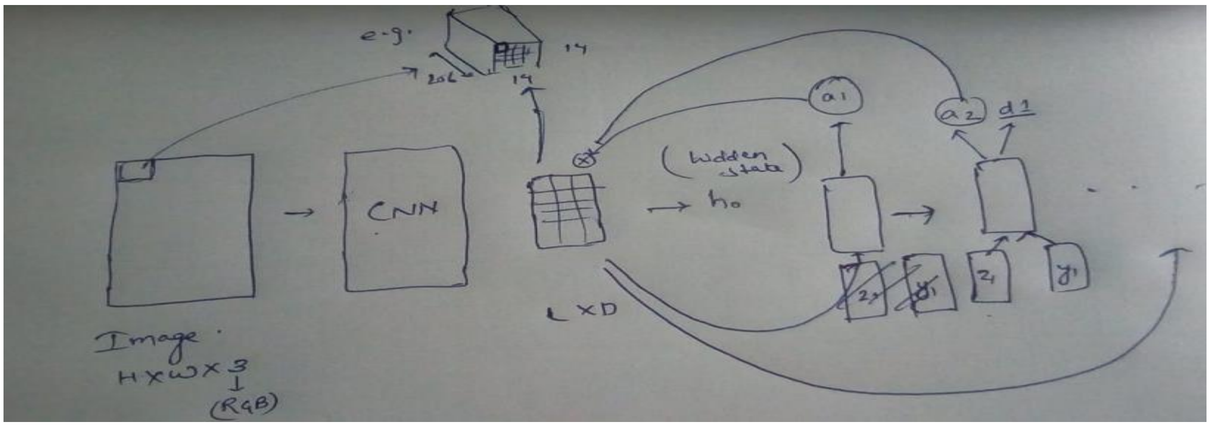
We used the output from final layer into RNN which contained the overall summary of the image in form of vector .Here the RNN does not see the whole image but only final feature generated.

Idea:

Instead of taking final feature output of convolution layer should be taken which will act as hidden state vector in

previous time step to generate the first word and rest all same.

Intutively, initially RNN was looking whole image at once(as a result from final Fully Connected Layer) and now RNN looks at different part of image at each time step.



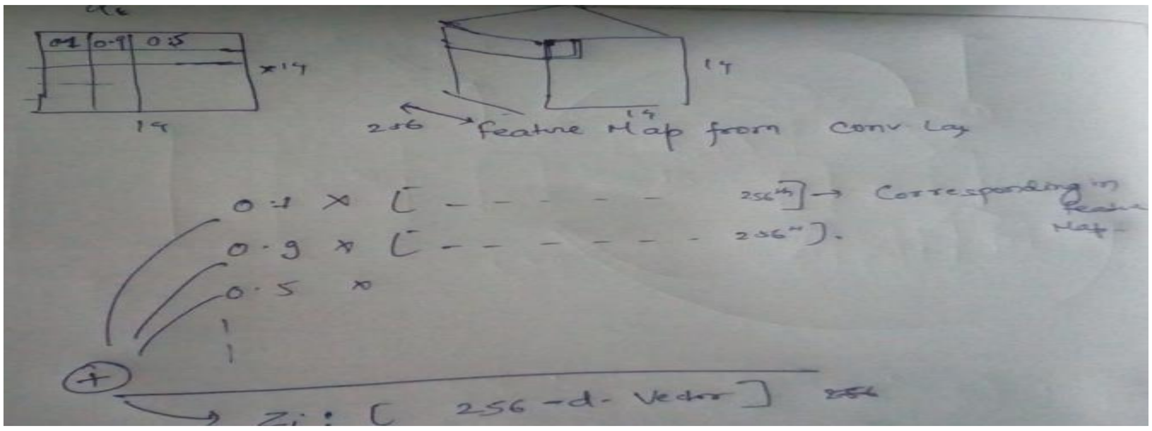
As here only one hidden state assume it as GRU hi : Hidden state

yi : Initial Word (Embedding Vector)

zi : Weighted Sum / Context Vector (here, 256 dimensional vector –depth from convolved Layer

ai : Output generated by RNN (here, 14 * 14) same as L* D such that each block contains a number between 0-1

Sum of all values = 1 , Cell which have more value gives more attention to that block.


ai
0.1
0.9
0.5
×14
14
14
14
256
Feature Map from Conv Lay
0.1 × [- - - - - - 256th] → Corresponding in feature Map
0.9 × [- - - - - - - 256th].
0.5 ×
+
Zi : [256-d- Vector]
256

Explanation through code

We are using InceptionV3 model which is pre trained on Imagenet and it is available in keras.

Input Shape: 299*299

We want output from last but new layer from the Inception model

```
image_model = tf.keras.applications.InceptionV3(include_top=False,
                                                weights='imagenet')
new_input = image_model.input
hidden_layer = image_model.layers[-1].output

image_features_extract_model = tf.keras.Model(new_input, hidden_layer)
```

The output from the convolution layer is of Size=(8,8,2048)   compress => 64 * 2048

For caption we used word embedding technique

Size for each image and caption pair I have:

Img_vector : 64 * 2048

Target _vector   : 40

Create a batch of 32 items wrap it into tensor and store it into a **dataset**.

Batch 0 :   img_tensor : 32 * 64 *2048   target_tensor : 32*40

for each epoch:

```
   for each batch in dataset:
           hidden_val =  zeros(batch_size) ,  decoder input =   word_index['<start>']*batch_size,
                                                              features = encoder(img_tensor)
           for each value in  target.shape[1]:
// Each time hidden layer is updated and goes into the model
                  prediction , hidden_val , _   =   decoder(dec_input,features,hidden)
                  loss += loss_function(target[:, i], predictions)
//Teacher Forcing : Instead of giving predicted value as next input we are passing true value to
//next RNN Cell this helps in faster learning and less mistake
                  decoder_input = tf.expand_dims(target[:, i], 1)       // Each time we are expanding
```

Here comes the core of the whole algorithm:

Encoder

It takes the img_tensor apply Activation on top and pass it into Decoder   --- 1

Decoder

```
Contect_vector ,attention_weigts  =    BahdanauAttention(features,hidden)
//Bahandau  :-  softmax(tanh(features,hidden))
x = concat(x,contex_vector)    //x is decoder input
Output , state = GRU(x)
x=Dense_layer(output)
return   x,state,attention_weights
```

Shapes:

Context_vector : 32* hidden_layer_shape

To more about Bahandau and Loung attention : https://github.com/spro/practical-pytorch/blob/master/seq2seq-translation/seq2seq-translation.ipynb

Sources:

https://machinelearningmastery.com/encoder-decoder-attention-sequence-to-sequence-prediction-keras/

https://towardsdatascience.com/light-on-math-ml-attention-with-keras-dc8dbc1fad39

https://www.tensorflow.org/beta/tutorials/sequences/nmt_with_attention

https://www.tensorflow.org/beta/tutorials/text/nmt_with_attention