Development Mechanism:

A comprehensive approach to the task of enabling humans who cannot sign to communicate using sign-language would clearly require the development of a general purpose speech to sign language converter. This in turn requires the solution of the following problems:

1. Automatic speech to text conversion (speech Recognition).
2. Automatic translation of English text into a suitable representation of sign language.
3. Display of this representation as a sequence of Signs using computer graphics techniques.

As we mentioned, the most suitable sign representation is “international sign writing”, we try to found all sources that we can use to gather “English – sign writing” combination sentences for our dataset.

In first try we gathered 1500 words and small sentences combinations, and we use it to build our first experimental model.

We use a sequence to sequence (seq2seq) model and After training the model we were able to input an English word, such as \*"¿translator ", and return the sign:

\*" 𝠀񀀁񀀩񆙡񋎩񋎽񂈁񂈉񆿅񆿕񋸥𝠃𝤨𝥇񆙡𝣪𝣟񀀁𝤅𝣕񀀩𝣿𝣚񋎽𝣥𝣱񋎩𝣦𝣆񂈁𝤚𝤌񂈉𝤁𝤌񆿅𝤛𝤮񆿕𝤀𝤮񋸥𝤍𝥁 "

Which is the representation of word “translator” in American Sign Writing.

# to train our model We used Tensor flow in “google colab” and we followed steps in “Neural Machine Translation with Attention” Notebook available at:

# <https://colab.research.google.com/github/tensorflow/docs/blob/master/site/en/r2/tutorials/text/nmt_with_attention.ipynb>

the steps are as follow:

**A: prepare the dataset:**

We stored the dataset in a “.txt” file in Unicode format, every line in this file is an “English + American sign language” sentence and they are separated with tab “/t”

Example:

𝠀񀀁񀀉񈗥񈗵񋸦𝠃𝤝𝤨񀀁𝤎𝣤񀀉𝣰𝣮񈗥𝤏𝤇񈗵𝣱𝤑񋸦𝤆𝤜 ¿come

1. Add a \*start\* and \*end\* token to each sentence.

2. Clean the sentences by removing special characters.

3. Create a word index and reverse word index (dictionaries mapping from word → id and id → word).

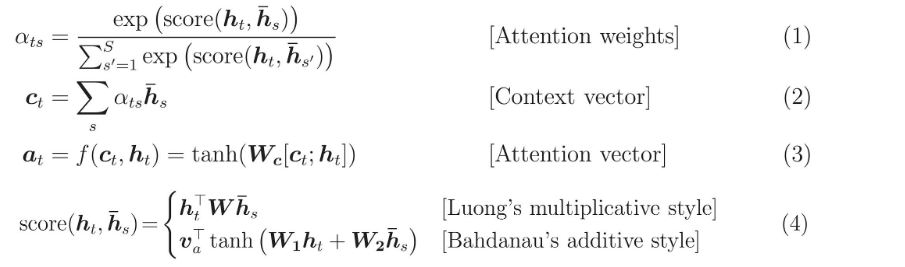
4. Pad each sentence to a maximum length.

### **B: Create a tf.data dataset**

## **C: Write the encoder and decoder model:**

The input is put through an encoder model which gives us the encoder output of shape (batch\_size, max\_length, hidden\_size) and the encoder hidden state of shape (batch\_size, hidden\_size).

Here are the equations that are implemented:



If we consider this notation:

* FC = Fully connected (dense) layer
* EO = Encoder output
* H = hidden state
* X = input to the decoder

The pseudo-code:is as follow:

* score = FC(tanh(FC(EO) + FC(H)))
* attention weights = softmax(score, axis = 1). Softmax by default is applied on the last axis but here we want to apply it on the *1st axis*, since the shape of score is *(batch\_size, max\_length, hidden\_size)*. Max\_length is the length of our input. Since we are trying to assign a weight to each input, softmax should be applied on that axis.
* context vector = sum(attention weights \* EO, axis = 1). Same reason as above for choosing axis as 1.
* embedding output = The input to the decoder X is passed through an embedding layer.
* merged vector = concat(embedding output, context vector)
* This merged vector is then given to the GRU

## **D: Define the optimizer and the loss function:**

## **E: Checkpoints (Object-based saving)**

## **F: Training:**

1. Pass the input through the encoder which return encoder output and the encoder hidden state.
2. The encoder output, encoder hidden state and the decoder input (which is the start token) is passed to the decoder.
3. The decoder returns the predictions and the decoder hidden state.
4. The decoder hidden state is then passed back into the model and the predictions are used to calculate the loss.
5. We Used teacher forcing to decide the next input to the decoder.
6. Teacher forcing is the technique where the target word is passed as the next input to the decoder.
7. The final step is to calculate the gradients and apply it to the optimizer and backpropagate.

## **G: Translate:**

* The evaluate function is similar to the training loop, except we don't use teacher forcing here. The input to the decoder at each time step is its previous predictions along with the hidden state and the encoder output.
* Stop predicting when the model predicts the end token.
* And store the attention weights for every time step.

Note: The encoder output is calculated only once for one input.

## **H: Restore the latest checkpoint and test manually**

We tested the translate function for following words and we get these results: