


Summary: In this workshop you will gain some practical experience with Recurrent Neural Networks (RNNs). First you will get acquainted with the calculation of the number of parameters (weights) in RNNs. Then you will look at a particular application and train an RNN-based chatbot.

Task 1: Number of trainable parameters in RNNs

Assume you are interested in finding a neural architecture for a language processing system. It has an input layer with 1000 inputs, an embedding layer with 64 units, a hidden layer of 128 hidden units, and fully connected layer with 10 outputs. You are required to calculate the number of parameters used by the neural architectures below. Verify that you get the following parameters:

Architecture	Your Worked Solution	Num. Params
Vanilla RNN	$\text{EmbeddingLayer} = \text{inputs} * \text{outputs} =$ $\text{RNNLayer} = \text{Wxh} + \text{Whh} + \text{b} =$ $\text{FCLayer} = \text{Why} + \text{b} =$ 	89994
LSTM	$\text{EmbeddingLayer} = \text{inputs} * \text{outputs} =$ $\text{LSTMLayer} = \text{Wc} + \text{Wo} + \text{Wi} + \text{Wf} + \text{Uc} + \text{Uo} + \text{Ui} + \text{Uf} + \text{bc} + \text{bo} + \text{bi} + \text{bf} =$ $\text{FCLayer} = \text{Why} + \text{b} =$	164106
BiLSTM	$\text{EmbeddingLayer} = \text{inputs} * \text{outputs} =$ $\text{fLSTMLayer} = \text{Wc} + \text{Wo} + \text{Wi} + \text{Wf} + \text{Uc} + \text{Uo} + \text{Ui} + \text{Uf} + \text{bc} + \text{bo} + \text{bi} + \text{bf} =$ $\text{bLSTMLayer} = \text{Wc} + \text{Wo} + \text{Wi} + \text{Wf} + \text{Uc} + \text{Uo} + \text{Ui} + \text{Uf} + \text{bc} + \text{bo} + \text{bi} + \text{bf} =$ $\text{FCLayer} = (\text{Why} + \text{b}) * 2 =$	264202
2-Stacked BiLSTM	$\text{EmbeddingLayer} = \text{inputs} * \text{outputs} =$ $\text{fLSTMLayer1} = \text{Wc} + \text{Wo} + \text{Wi} + \text{Wf} + \text{Uc} + \text{Uo} + \text{Ui} + \text{Uf} + \text{bc} + \text{bo} + \text{bi} + \text{bf} =$ $\text{bLSTMLayer1} = \text{Wc} + \text{Wo} + \text{Wi} + \text{Wf} + \text{Uc} + \text{Uo} + \text{Ui} + \text{Uf} + \text{bc} + \text{bo} + \text{bi} + \text{bf} =$ $\text{fLSTMLayer2} = \text{Wc} + \text{Wo} + \text{Wi} + \text{Wf} + \text{Uc} + \text{Uo} + \text{Ui} + \text{Uf} + \text{bc} + \text{bo} + \text{bi} + \text{bf} =$ $\text{bLSTMLayer2} = \text{Wc} + \text{Wo} + \text{Wi} + \text{Wf} + \text{Uc} + \text{Uo} + \text{Ui} + \text{Uf} + \text{bc} + \text{bo} + \text{bi} + \text{bf} =$ $\text{FCLayer} = (\text{Why} + \text{b}) * 2 =$	658442

Task 2: RNN-based chatbot

Download the data and baseline chatbot from Blackboard (under workshop materials of Week 6), which correspond to the following files: data_task2.zip, BaselineChatbot1.1.zip. Open the program “BaselineChatbot-v1.1.py”, look for ‘path_to_data_’, and update the corresponding paths as in your PC. If the model’s folder (training_checkpoints_gru) does not exist, create it please. Have a look at the data files. The file “dstc8-train.txt” has sentences separated by a tab character (left: previous sentence, right: following sentence). The files “dstc8-*-candidates” contain the previous sentence called ‘CONTEXT’ and the sentence to choose from referred to as ‘CANDIDATE**’. The word tokens starting with a dollar sign refer to abstracted words to reduce word diversity. The goal of your trained model is to predict the correct sentence (first candidate) from all available candidates. Just be aware that the task gets harder as the amount of training and test sentences increases. Run the code as follows (arguments: program name, execution mode, number of epochs):

- python BaselineChatbot-v1.1.py train 100
- python BaselineChatbot-v1.1.py val 0



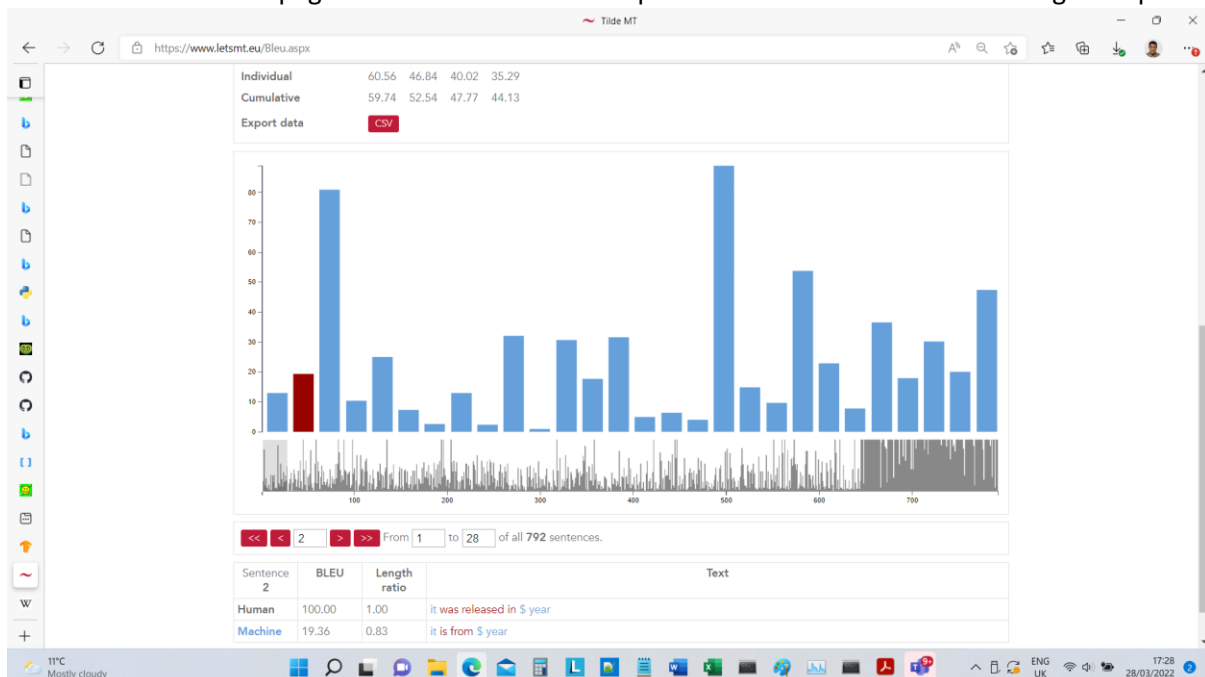
Whilst the model is training, have a look at the following classes and work out task number 3

- class Encoder(tf.keras.Model)
- class BahdanauAttention(tf.keras.layers.Layer)
- class Decoder(tf.keras.Model):

The output of your training should be the following: one file with the source sentences (dstc8-sgd-reference.txt), and one file with the predicted outputs/responses (dstc8-sgd-predicted.txt). Once the model has trained+validated, load the output files (*.txt) from above in the following link to observe the performance of your Chatbot – the higher the better: <https://www.letsmt.eu/Bleu.aspx>.

The screenshot shows the 'Tilde Custom Machine Translation' website. The main heading is 'Interactive BLEU score evaluator'. Below this, there is a description: 'Perform comparative quality evaluations of files translated with one or more MT systems. This allows you to compare MT output with human translations and compare the BLEU scores of various MT systems. Click here to learn more.' The interface is divided into four steps for file selection: Step 0 (source file), Step 1 (human translated file), Step 2 (machine translated file), and Step 3 (second machine translated file). Each step has a 'Choose File' button and a text input field. Below the steps, there are checkboxes for 'Calculate BLEU' and 'Display', and a 'Score' button. The bottom of the page shows a Windows taskbar with the date 28/03/2022 and time 17:26.

Look at the correct and incorrect words predicted by the encoder-decoder, which can be found at the bottom of the webpage. Click a blue bar to see a pair of sentences as in the following example.



Task 3: Calculation of attention in RNNs

Consider the following input words: <start>, how, are, you, <end>

... and the following output words: <start>, fine, thanks, <end>

Encoder output shape: (batch size, sequence length, units) (1, 5, 3)

Encoder Hidden state shape: (batch size, units) (1, 3)

(a) Given the following scores,

scores=[[0.01341882], [0.01776753], [0.01350384], [-0.00414546], [-0.00804073]]

as per Bahdanau's style: $\text{score}(h_t, \bar{h}_s) = \begin{cases} h_t^\top W \bar{h}_s & [\text{Luong's multiplicative style}] \\ v_a^\top \tanh(W_1 h_t + W_2 \bar{h}_s) & [\text{Bahdanau's additive style}] \end{cases}$

verify the calculation of attention scores according to

$$\text{attention-weights } a_{ts} = \frac{e^{\text{score}(h_t, h_s)}}{\sum_{s'=1}^s e^{\text{score}(h_t, h_{s'})}}$$

a = [[0.20137736], [0.20225501], [0.2013945], [0.1978712], [0.19710194]]

(b) Given the following encoder outputs being attended (h_s),

h_s = [[0.01152591, -0.01253617, 0.01942099],
[0.01529713, -0.01739413, 0.02411043],
[0.01658512, -0.01774601, 0.02075228],
[-0.00456512, -0.00694389, 0.00476691],
[0.01493649, -0.00596176, 0.00019239]]

verify the calculation of context vectors according to

$$\text{context vector } c_t = \sum_s a_{ts} h_s$$

c_t= [0.01079584 -0.01216557 0.01394795]