1. Seq-to-seq Model Configuration

Q1. Describe the network architecture for the encoder, and for the decoder model: what kind of RNNs are used? What are the dimensions of the various layers? What are the non-linearity functions used? How is the attention computed?

The encoder employs a GRU architecture with one hidden layer. The shape of input layer is (|V|, hidden_size). |V| here represents the length of vocabulary of Bulgarian characters, which is 85. The hidden_size here represents the dimension of word embedding, which is 256 in the script. The shape of hidden layer is (hidden_size, hidden_size), in this case (256, 256). Since not specified, the non-linearity functions used in the default GRU are sigmoid and tanh.

There are two decoder models.

The simple RNN decoder model also employs a GRU architecture with one hidden layer. The initial input here is the output of the encoder. The shape of input layer is (|V|, hidden_size). |V| here represents the length of vocabulary of Bulgarian characters, which is 31. The hidden_size here represents the dimension of word embedding, which is also 256 in the script. The shape of hidden layer is (hidden_size, hidden_size), in this case (256, 256). The output layer then uses softmax to find the target English character with the highest score.

The attention decoder model employs an attention architecture with one hidden layer. The initial input here is the output of the encoder. It is fed to calculate the first set of attention weights. At every step, a weighted combination is calculated as the matrix-matrix product of attention weights and encoder's outputs. The attention weights are calculated by multiplying the decoder's output and hidden state at each step. The weighted combination is then transformed using ReLU and fed into the GRU with hidden state to generate the next output with a log softmax transformation. The dimensions of this model is the same as the previous simple RNN decoder except that it has an additional attention layer to focus on the different parts of the encoder's outputs at each step in the decoder model.

2. Training Algorithm

Q2. Two important hyperparameters control training: *n_iters* and *learning_rate*. Explain what role each of them plays

In the overall training process, the parameter n_iters is used to control how many times to randomly choose the word pair from the original 4997 word pairs, so that the number of

elements in the training pairs is equal to the number of n_iters. And then we call function 'train' for each training pair in the training pairs to train the overall encoder-decoder model.

From the function 'train', we found that the model uses stochastic gradient descent algorithm to update the parameters, so the learning rate is used to control the step size of convergence.

Q3. Select values for these hyperparameters using the validation set. Describe the experiments you ran to select those values, and explain your reasoning.

To generate the appropriate combination of these two hyperparameters, we employed the grid search method. The following table shows the edit distance for validation set with different combinations of n_iters and learning_rate. From the results, we found that when we randomly selected 25000 word pairs and picked 0.001 as learning rate, the edit distance for validation set would be the lowest, 0.7137.

By comparing to the result when n_iters is equal to 20000 and learning rate is equal to 0.001, we concluded that the encoder-decoder model will have a greater probability to converge and better result only with a small learning rate and enough training examples. In other words, a model with a small learning rate need more training examples to converge to global minima for estimating model parameters.

Due to the limitation of computation, we only ran the model one time with each combination of hyperparameters. The result will be more robust with running more times.

n_iter/learning_rate	**0.001**	**0.01**	**0.05**	**0.1**
5000	5.7600	1.9074	4.1242	12.9053
10000	3.1832	1.2463	3.6947	18.6253
20000	1.4505	1.2884	4.6211	18.8547
25000	0.7137	1.4611	3.7032	9.2779

3. Understanding techer_forcing

Q4. Explain how training works if teacher_forcing is set to 0, and if teacher_forcing is set to 1.

If teacher_forcing is set to 0, during training, each input at the decoding phase is generated purely by the decoder's prediction. If teacher forcing is set to 1, during training, each input at the decoding phase is generated by the real target output.

Q5. Investigate the impact of teacher forcing empirically. Report learning curves for 0.1, 0.5 and 0.9, and explain what you observe.

With the help of "teacher forcing", the model converges faster but too much "teacher forcing" may cause the network to be unstable. Following are three learning curves for 0.1, 0.5, and 0.9, respectively. We can observe from the graphs that a higher ratio of "teacher forcing", the training process indeed converges faster. For example, it takes the network with teacher_forcing set to 0.1 around 5000 iterations (examples) to reach a edit distance of 1.0, while it takes the network with teacher_forcing set to 0.9 around only 2500 iterations (examples) to reach a edit distance of 1.0. However, the downside is also obvious in terms of instability. For example, the network with teacher_forcing set to 0.9 experiences a major fluctuation between 12500 and 15000 iterations (examples).

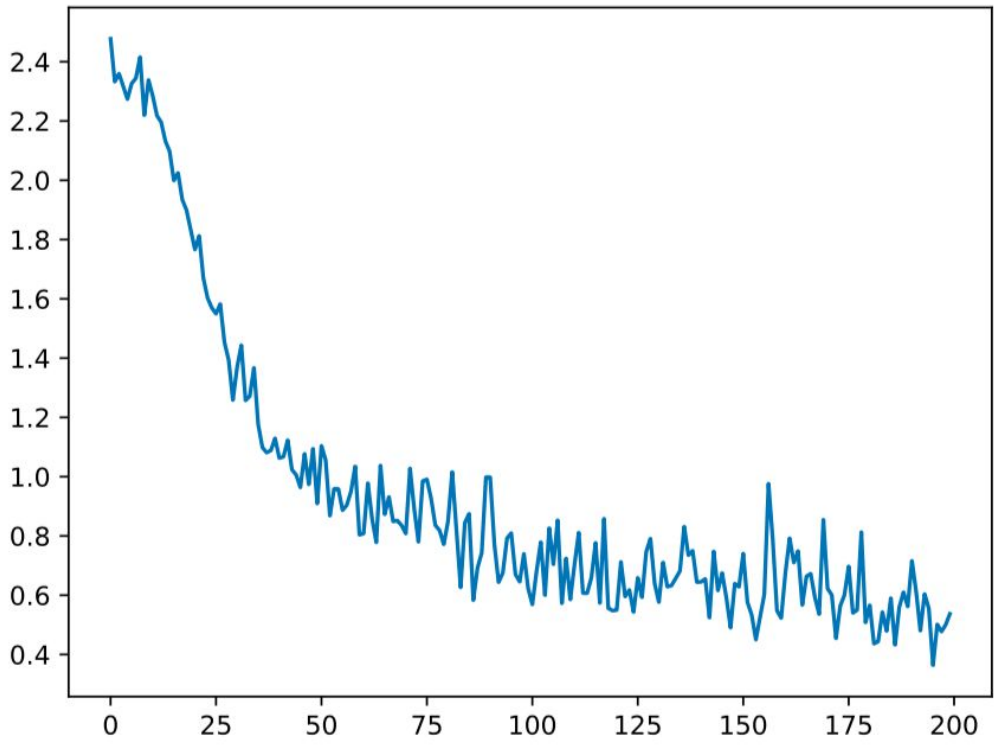


Figure 1 Learning Curve of network with teacher_forcing = 0.1

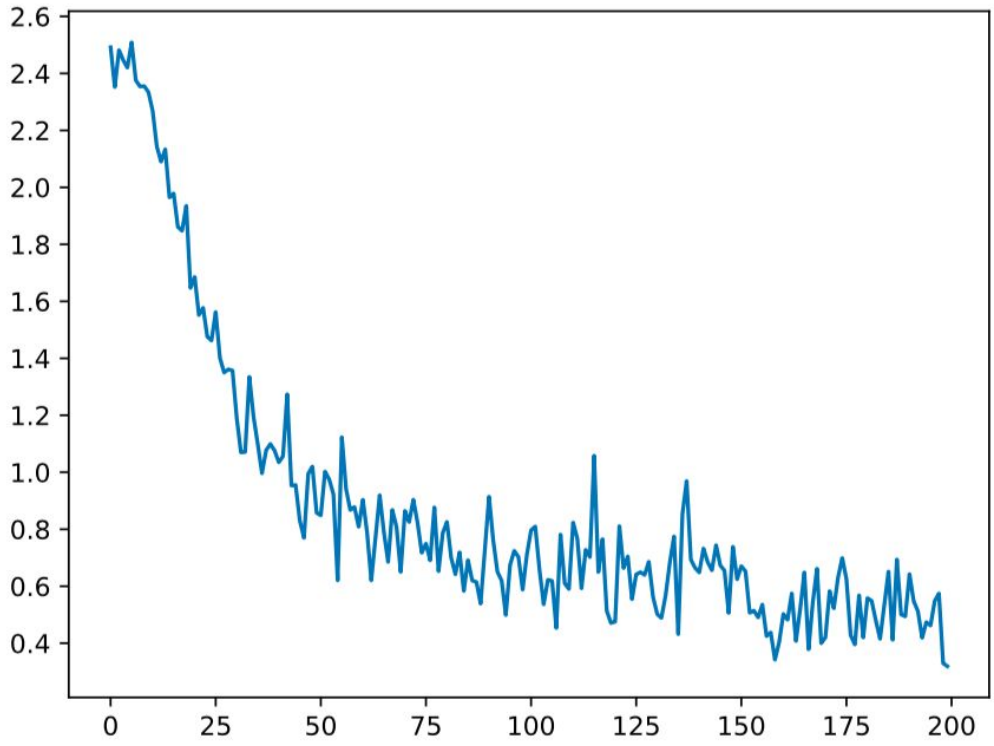


Figure 2 Learning Curve of network with teacher_forcing = 0.5

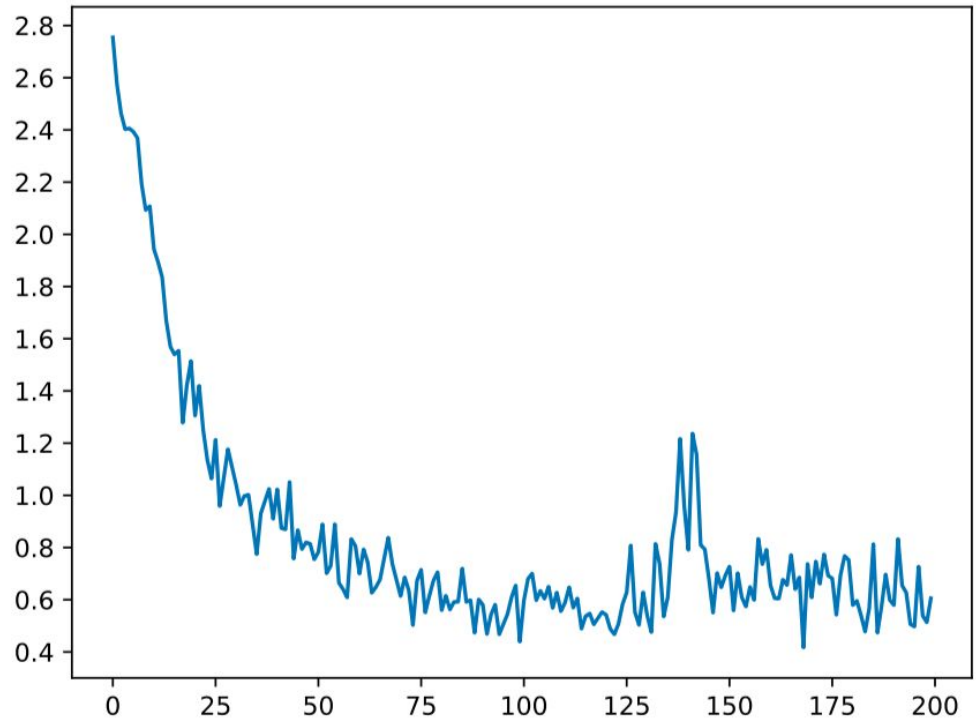


Figure 3 Learning Curve of network with teacher_forcing = 0.9

4. Impact of attention mechanism （Q6- Q8）

Transliteration is the process of converting characters from the input language alphabet to the output language alphabet, in which the process will transliterate one character and only focus on that character or the nearby characters without any attention for other characters within the input words. Therefore, similar to the idea of transliteration process, the attention model is more useful to model transliteration than the regular encoder-decoder model.

Because of randomly generating training word pairs, we ran the script three times with and without attention model. The following table is the result for the models with and without attention model in terms of edit distance and running time. All models have the same hyperparameters (n_iters=20000 and learning_rate=0.01).

The results prove our hypothesis that attention model is more useful for transliteration. The average edit distance on validation set for attention model is 1.0175, compared to 1.5088 for regular encoder-decoder model. On training set, the attention model also gets a lower edit distance than regular encoder-decoder model (0.5371 vs 0.6955). However, with lower edit distance on validation set and training set, the attention mode also needs more time to train.

With attention model	**Edit distance on training set**	**Edit distance on validation set**	**Running time**
1st time	0.5274	1.0526	14 min 14s
2nd time	0.5331	0.9600	14 min 18s
3rd time	0.5508	1.0400	14 min 18s
Average	0.5371	1.0175	14 min 67s

Regular encode-decoder model	**Edit distance on training set**	**Edit distance on validation set**	**Running time**
1st time	0.6701	1.4821	10 min 37s
2nd time	0.6828	1.5032	10 min 38s
3rd time	0.7335	1.5411	10min 35s
Average	0.6955	1.5088	11min 2s

5. Transliteration from Chinese to English (Q9-Q11)

As shown in the requirement of this project, it is very interesting to observe the transliteration from Chinese words to English characters based on pronunciation. However, the Chinese language is extremely different from English and Bulgarian, in which individual English characters represent sounds rather than concepts. A single character in the Chinese language will be represented as one or a few words in English, which is a combination of the letters. Therefore, the smallest unit in formal Chinese language is a word, not a letter, making the vocabulary set extremely large when transliterating Chinese to English. In other words, there is no difference for vocabulary set between Chinese transliteration and regular translation, because both vocabulary sets represent the number of unique Chinese characters. The training data we used in this part is from the CJK Dictionary Institute[1]. In the sample dataset[2], there are 20,000 word pairs with 3,212 unique simplified Chinese characters. We transformed the dataset into the same format as the example datasets. We splitted 20,000 word pairs into training, validation, and testing sets with sizes of (16000, 2000, 2000). Because of the huge vocabulary size, running 20,000 iterations (examples), the average edit distance we obtained was very large (5.9326).

The problem we mentioned above is very characteristic to the Chinese written language. The writing system that Chinese language uses is known as Logographic writing system, the earliest writing system, which was widely used in the first historical civilizations. Therefore, the method we use to decode Chinese characters should be also applicable for other languages that have Logographic writing system.

To solve the problem of the huge vocabulary size, we experimented the 'zhengma' method[3] to re-decode Chinese characters. The idea is to use English letters to represent different Chinese character components. For example, although '沙' and '纱' are different characters in the Chinese language, both characters share a same pronunciation 'sha' and a same right component '少'. It is very common in Chinese language that some words with a shared component share the same or similar pronunciation. Therefore, with 'zhengma' decoding method, our solution aims to leverage the above phenomenon and thus decrease the overall vocabulary set. After applying 'zhengma' decoding method, the above examples '沙' and '纱' will be decoded as 'VKM' and 'ZKM', in which the first character, 'V' and 'Z', will represent the left component of these two Chinese characters and 'KM' will represent the shared right component. As a result, 'VKM' and 'ZKM' can represent the internal constitution of these two Chinese characters. Our assumption is that by applying 'zhengma' method, we could reduce the edit distance because of the much

[1] Website: http://www.cjk.org/cjk/samples/hanzipin.htm
[2] Source: http://www.cjk.org/cjk/samples/pinyin_sample_20000.txt
[3] https://en.wikipedia.org/wiki/Zhengma_method

smaller vocabulary size. After applying 'zhengma' method for all characters in the original vocabulary set, theoretically, the new vocabulary set will be the same as the English alphabet.

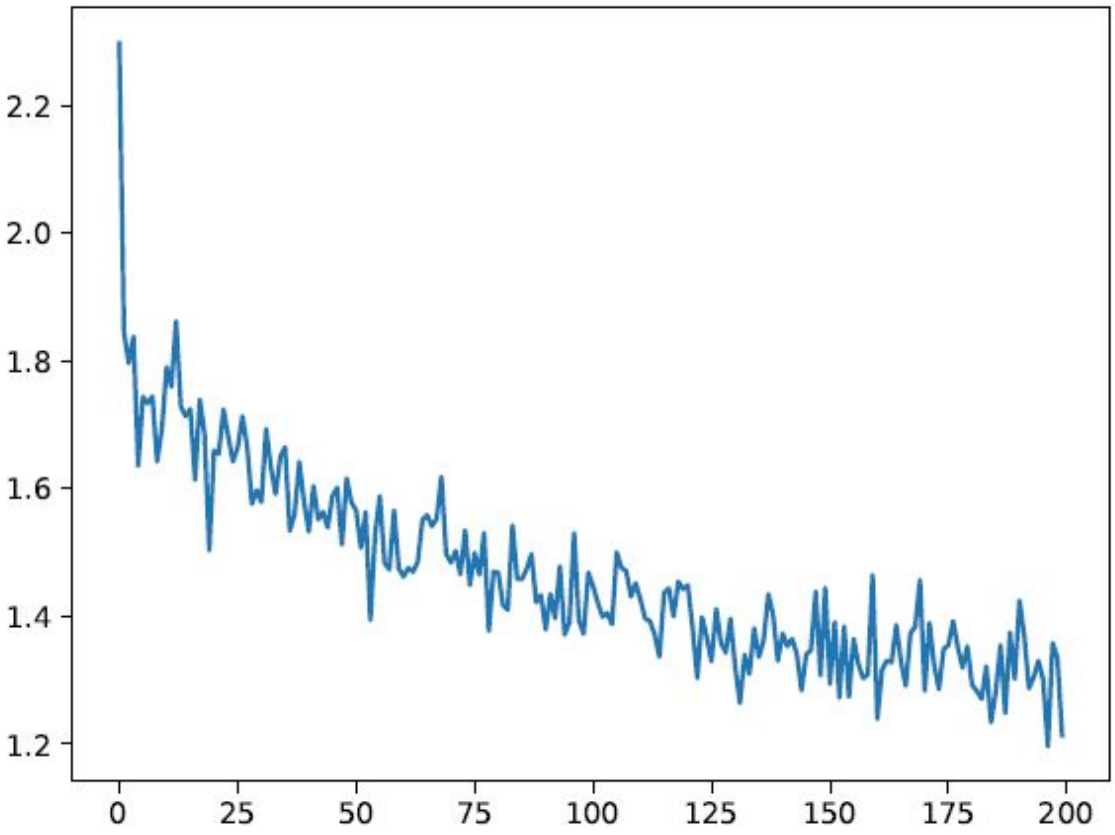
To generate the 'zhengma' code for the Chinese characters in the original vocabulary set, we collected the 'zhengma' codes from a dictionary website[4] with a web crawler, and obtained 'zhengma' codes for about 3400 unique characters in training, validation and testing sets. After that, we used 'zhengma' codes to replace the original Chinese characters and trained a new attention model.

The following table shows the results of transliteration from Chinese to English with and without 'zhengma' method. Because the training examples are randomly chosen, we ran the script two times with and without 'zhengma' code. All models use the same hyperparameters as n_iters = 20000 and learning_rate = 0.01.

Model with 'zhengma'	n_iters	Edit distance on training set	Edit distance on validation set	Running time	Vocabulary statistics
1st Time	20000	1.3298	6.0323	25 min 28s	40
2nd Time	20000	1.3415	6.1678	26 min 45 s	40
Average	20000	1.33565	6.10005	26 min 6 min	40

Model without 'zhengma'	n_iters	Edit distance on training set	Edit distance on validation set	Running time	Vocabulary statistics
1st Time	20000	1.1459	5.8032	23 min 14 s	3212
2nd Time	20000	1.1230	6.3603	22 min 17 s	3212
Average	20000	1.13445	6.08175	22 min 45 s	3212

[4] Website: http://www.cilin.org



However, from the results, we found there is no significant difference for edit distance between models with and without ‘zhengma’ decoding method, although the second model with ‘zhengma’ decoding method has a low edit distance. Also, the running time for the model with ‘zhengma’ code is longer than the model without ‘zhengma’ code. The reason is that although the vocabulary set decreased from 3212 to 40, the representation of an individual Chinese character becomes more complex. In other words, after employing ‘zhengma’ code for all characters in our datasets, one individual Chinese character is represented as multiple letters, thus making the input words more complex. For example, the 24th line of the training example is word pair that represent coffee grinder, which the length of the input word will increase from 6 Chinese characters to 20 English letters. As a result, the complex input words make it difficult to train the attention model and thus result in a higher edit distance.