neural network models for transliteration

Etamar Romano 205389711, Aryeh Klein 323746016

Submitted as final project report for the Deep Learning course, IDC, 2020

1 Introduction

Transliteration, "the act or process of writing words using a different alphabet"(Cambridge dictionary), is an important task in multilingual text processing, useful in application like online mapping, reputation monitoring and as a key component in any machine translation system. Transliteration is determent by a few major factors.
First of all, The origin and target languages. many language pairs have adopted different rules for transliteration over time. secondly "historical accidents", not always transliterating is done by the straight forward or the obvious way, many time it's a result of different conventions and historical errors which have been made and stayed over the years. And finally statistical regularities.
These properties have made us think that its important to have a high quality automated ML solution to the problem, and made us wonder which model would do it best.
In this project we compared four different neural-network based sequence-to-sequence models to transliteration. The first model is a vanilla multi-layered GRU. The second and the third models are two different interpretation of the attention principle, and the forth on is a transformer as seen in class. We would like to asses this three models performances and compare them.
Our Data set is composed of 87,564 English spelled first names, last names, cities names, mountains names and villages names with the corresponding Hebrew spelled label taken from "Wikipedia" using "Wikidata". We devided the data as follows:
80% training set, 10% testing set, 10% devset.
In these project we decided to use characters error rate as a comparison metric and report the difference of the three models.

1.1 Related Works

Arabic Machine Transliteration using an Attention-based Encoder-decoder Model [1]

A Deep Learning Based Approach to Transliteration [2]

Sequence-to-sequence neural network models for transliteration by Mihaela Rosca and Thomas Breuel [3]

2 Solution

2.1 General approach

"Transliration" of names is a task that is a combination of transliteration and translation. In this task we focus on combining the powers of machine translation with machine transliteration.
One must notice that transliterating is not very different from translating.
Transliteration seems a simpler task then translation but with one difference.
while in translating context is very important ("apple" can mean both type of fruit or a huge corporation) transliterating requires mainly to find correspondents from the origin to the target alphabet with a few exceptions. As the alphabets are a finite set of symbols that almost never change their meaning due to context ('a' in one word means the same 'a' in the other) its does seem easier to find these connections.
As we went on we realized that the task is no small feat and context plays a bigger role then expected.
For example, biblical names have different transliteration then modern names.
The transformer model has been shown to lead to better results in translation and similarly we would like to see how the transformer preforms in the "Transliration" of names.
That is why we have decided to take a similar approach to translation.
In this task we use seq2seq architecture that will be "fed" letter by letter instead of word by word.
The models we chose to implement are as follows:
1:vanilla GRU.
2,3: GRU with two types of attention mechanisms.
4: The transformer model.
As we mentioned before, We plan to try four different models based on GRU.
We would like to see how these two interpretations of attention mechanism compare while the vanilla model will serve as a "control group".
We will see also how the transformer preforms in name "transliration" and if it superior to the GRU's.
During the project the GRU models will have constant values of 2 layers, hidden layer size of 300, and embedding size of 64 so that the comparison will not be biased.

2.2 Design

Our models all use the Pytorch library in python,
We ran our models on one NVIDIA GEFORCE GTX GPU, using the anaconda framework.
Our data was pulled from wikipedia and consists of first names/last names/city's/villages/mountains.
Because we couldn't get all the data at once due to server limitation, We wrote a script that uses the wiki-data API and gets the names year by year of people born from -2000 to 2020.
We then took the data and did some prepossessing to eliminate as much noise as possible. After data cleanup and removing duplicates we remain with 87,564 names.
After that we encode each letter to an id/index.
In all models we put the inputs through an embedding size of 64.

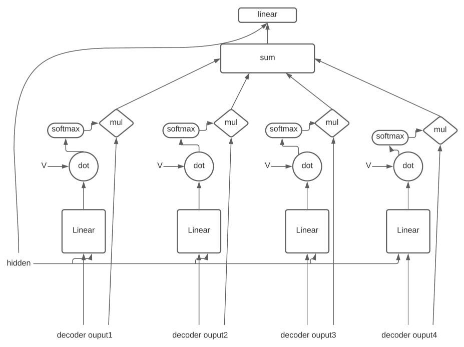
The optimizer we used in all models is Adam with a learning rate of 0.0001.
The loss function we used is pytorch's Cross entropy loss, and therefore the softmax is implemented in the loss function of the models.

1: The vanilla GRU model consists of an encoder 2 layer GRU the feeds the last hidden vector to the Decoder 2 layer GRU.
We "feed" the encoder as of the decoder letter by letter. The decoder gets as input the first "SOW" token and then the next guessed letter from the past prediction of the GRU.
(sirtut)

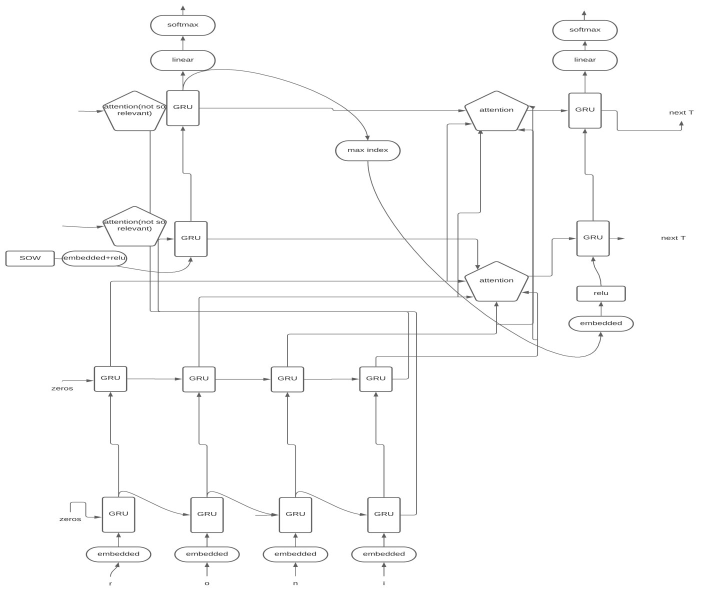
The main issue we have with the GRU models is how to implement the various attention mechanisms with layers.
After much research(articles+class and various testing) we fixated on these two models.

2: The first attention GRU model consists of a two layer encoder GRU and a two layer decoder GRU with an additive attention mechanism.
We "feed" the encoder letter by letter.
The first hidden in the encoder is the last hidden of the decoder.
we then take all the outputs of the encoder and for each timestamp(letter) in the decoder calculate the weights to get the context vector which we then concatenate with the last hidden vector and then pass it through a linear layer and use this vector as the hidden vector for this timestamp. The calculation of the attention is as follows for the hidden vector:
at time t(letter number t):
for every output$_i$: $score(output_i, hidden_{t-1}) = v^T * tanh(W[output_i; hidden_t])$.

and then take the scores and do between them a softmax for the given t.
$\alpha_{i,t} = softmax(score(output_i, hidden_{t-1}))$.
$context_t = \sum_i \alpha_{i,t} * output_i$
after that hidden = W'[context$_t$, $hidden_{t-1}$]



and then hidden goes into the GRU in time t as hidden vevctor.
In this model the attention takes into account the context from each output vector of the decoder and passes it on together with the hidden vector from the previous timestamp as a hidden vector to this timestamp.
The decoder gets as input the first "SOW" token and afterwords the guessed letter from the GRU in previous timestamp.



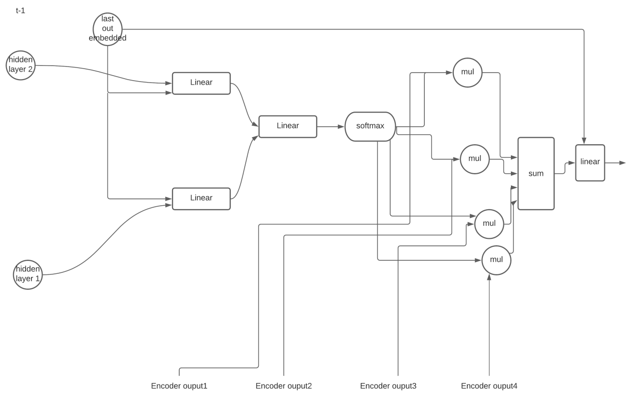
3: The second attention GRU model consists of a two layer encoder GRU and a two layer decoder GRU with an additive attention mechanism.
We "feed" the encoder letter by letter.
The first hidden in the encoder is the last hidden of the decoder.
The input for each timestamp is appended to the context vector from the previous timestmap. At first there is self attention mechanism between the hidden vectors from the previous timestamp and the embedded guessed letter from previous timestamp, lets call this the pre-context.

Then there is another additive cross attention mechanism with the output vectors of the encoder and the pre- context .

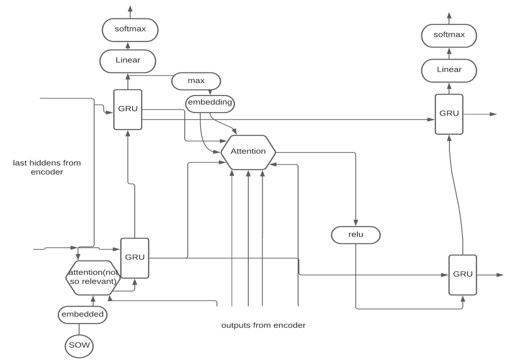
at time t:
The self attention:
selfattn = W'[W[hidden$_{t-1-1}, ouput_{t-1}], W[hidden_{t-1-2}, ouput_{t-1}]]$

where there are 2 layers of the model.
at the end it returns weights that we would want to apply to each encoder output vector.
weights = softmax(selfattn) then context$_T = \sum_i outputencoder_i * weights_i$

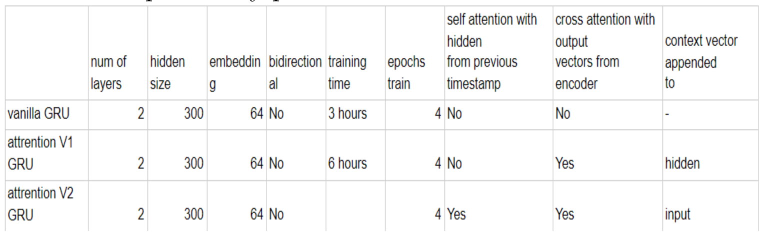
the input for time t is:
input$_t = W"[context_t, embedded(max(ouput_{t-1})]$



In this model the attention calculates wights based on past hidden vector and then calculates the weighted context from the output of the encoder, it then passes this context vector as an input for the next timestamp.
and the hidden form the last timestamp is passed on regularly.



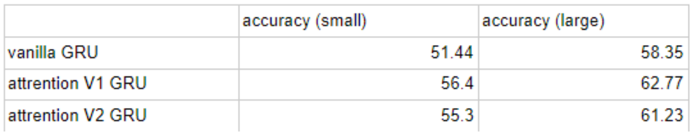
these two models are two interpretations of attention with GRU, One key difference is that in the first model the context vector is passed on as hidden vectors and in the second model as an input vector.
For these three models we also implemented with a bidirectional encoder, But

since we did not see a significant improvement (if any), we decided to keep to one directional.
4: The transformer model is based on the article ”attention is all you need”[4]. We implemented the transformer with 4 layers of both decoder and encoder.
The input of the encoder is an english and the target is a hebrew word with an added ”SOW” token to the beginning.
The loss was calculated with the output of the transformer and the original hebrew name with an added ”EOW” token.
That way the output is shifted right.
We also add a mask for the target word so the attention will only be calculated with letters already seen.
During writing the code for the transformer we had two major mistakes that took us a while to figure out.
1:We realized that softmax (logsoftmax) was happening in the loss function and we did not have to add it to the model.
2:We need to shift right the target word manually so it can learn seq2seq style. We do this by feeding the transformer decoder the target word with an added ”SOW” to the beginning, We then compute the loss between the output of the encoder and the real word without the ”SOW” and with an added ”EOW” at the end so they are the same length.
So for example ””: ”’SOW’” would be the target word fed to the encoder and the output would be compared in the loss to ”’EOW’”.
that together with the mask ensures that the encoder can only ”see” the letters before that are previously placed in the name.


	num of layers	hidden size	embeddin g	bidirection al	training time	epochs train	self attention with hidden from previous timestamp	cross attention with output vectors from encoder	context vector appended to
vanilla GRU	2	300	64	No	3 hours	4	No	No	-
attrention V1 GRU	2	300	64	No	6 hours	4	No	Yes	hidden
attrention V2 GRU	2	300	64	No		4	Yes	Yes	input

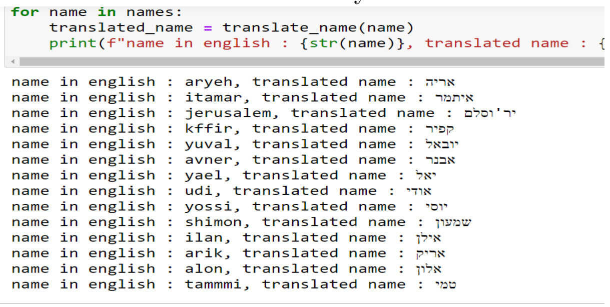
3 Experimental results

We split the data into 80% train, 10% test and 10% devset. In order to determine our chosen models and hyper parameters we created a small train set that consisted of 10% of the training data and checked the results on the devset. In our final runs we merged the devset with the train set and then checked our model on the test set.
We trained each model for four epochs. The metric we used to determine the accuracy of our model is 1 - CER (character error rate). We will notice that in some models we have used padding for either the Hebrew name or English name or both. In those cases when calculating the accuracy we don’t take into account the padding as to not get a ”fantasy” accuracy. Here are our results:



	accuracy (small)	accuracy (large)
vanilla GRU	51.44	58.35
attrention V1 GRU	56.4	62.77
attrention V2 GRU	55.3	61.23

*accuracy (small) is the accuracy when training on the small data set and testing on the devset.
*accuracy (large) is the accuracy when training on the merged train and devset and testing on the test set.
Here are some translated names by the vanilla GRU model:



```
for name in names:
    translated_name = translate_name(name)
    print(f"name in english : {str(name)}, translated name : {
```

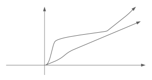
```
name in english : aryeh, translated name : אריה
name in english : itamar, translated name : איתמר
name in english : jerusalem, translated name : יר'וסלם
name in english : kffir, translated name : קפיר
name in english : yuval, translated name : יובאל
name in english : avner, translated name : אבנר
name in english : yael, translated name : יאל
name in english : udi, translated name : אודי
name in english : yossi, translated name : יוסי
name in english : shimon, translated name : שמעון
name in english : ilan, translated name : אילן
name in english : arik, translated name : אריק
name in english : alon, translated name : אלון
name in english : tammmi, translated name : טמי
```

when training on the transformer model on the small data we got a loss of 0.78. Even when printing the predictions during training on the regular training set and compare them to the real labels, we got reasonable results, as low loss as well, but when we tested the model we did not get any accuracy(around 1%). After hours and hours of thought and testing we were not able to figure out the issue. we will address this further ahead.

4 Discussion

As expected the accuracy on the testset is higher when applying attention mechanism over the "vanilla" seq2seq GRU models.
Our explanation to this is that over time the weights for each of the decoder outputs learn to adapt to the most relevant letter, without losing the context of the whole word. so we assume the weights are bigger on relevant letters are usualy around then same relative location in the hebrew word, thats why applying attention increased the accuracy. We think that if we had time to run the model for 30-40 epochs, the difference in accuracy between the model with attention and the model wothout attentopm would be greater. Our assumption is that the attention models have a much larger amount parameters and therefore will take longer to learn. So why do we have bigger delta in the small training?
One possible explanation is that since the context in "transliration" plays a smaller role then it does in translation, the accuracy in the attention models and the vanilla models will converge to a similar value but the attention models learn quicker(in contradiction to our assumptions).
An alternative explanation is that the correlation of the learning curves of the

attention models and the learning curve of the vanilla model is not linear, and we think that if we had more time to train both of the attention models, we would have a bigger accuracy difference as seen in the theoretical graph below.



Even though we didn't see any accuracy in the transformer we decided to add it to our final project as the saying goes "Success is not fatal, it is the courage to continue that counts"(Winston Churchill). We have learned a lot about the transformer model and feel competent on all its inner workings.
The reasons for our failure could be 1: lack of understanding of the testing method. We did understand that it is different from the training part as in the test you need to "feed" the transformer the prediction it made so far to the decoder, we may have failed to implement it properly. 2:The transformer needs way more data and epochs. 3: The transformer is not fit for the "transliration" task(less likely). In terms of the two different attention models, we think that adding context to the hidden vector for each timestamp (as shown in class) is more effective than adding context to the input vector. since the hidden vector is the one that we expect to have to carry the information about the models throughout each timestamp. (that is also the reason we feed the last output/hidden from the encoder as the first hidden of the decoder and not as the first input of the decoder).

5 Code

https://github.com/aryehlev/name$_translirations$

References

[1] Mohamed Seghir Hadj Ameur, Farid Meziane, and Ahmed Guessoum. Arabic machine transliteration using an attention-based encoder-decoder model. *Procedia Computer Science*, 117:287–297, 2017.

[2] Soumyadeep Kundu, Sayantan Paul, and Santanu Pal. A deep learning based approach to transliteration. In *Proceedings of the seventh named entities workshop*, pages 79–83, 2018.

[3] Mihaela Rosca and Thomas Breuel. Sequence-to-sequence neural network models for transliteration. *arXiv preprint arXiv:1610.09565*, 2016.

[4] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *arXiv preprint arXiv:1706.03762*, 2017.