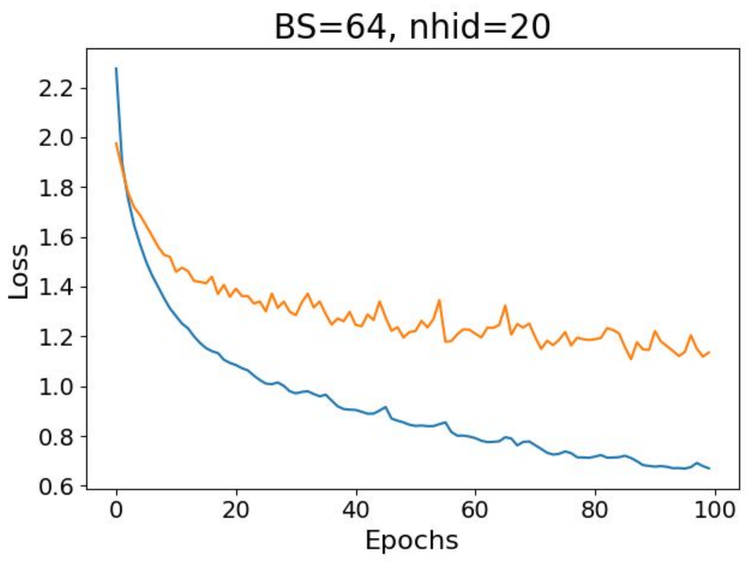
Programming Homework 3

Alexander Harnisch

Since it seems like there aren't any equations or drawings involved here, I decided it's probably easier for you to grade if I switch to typing instead of handwriting.

1. Gated Recurrent Unit



Final loss: train=0.6701413144858984 val=1.1352065165164107

The loss plot at first sight seems at least somewhat promising: The orange validation loss is consistently larger than the blue training loss, without showing any sign of diverging. So there is no sign of overfitting and the loss appears to converge after being significantly reduced.

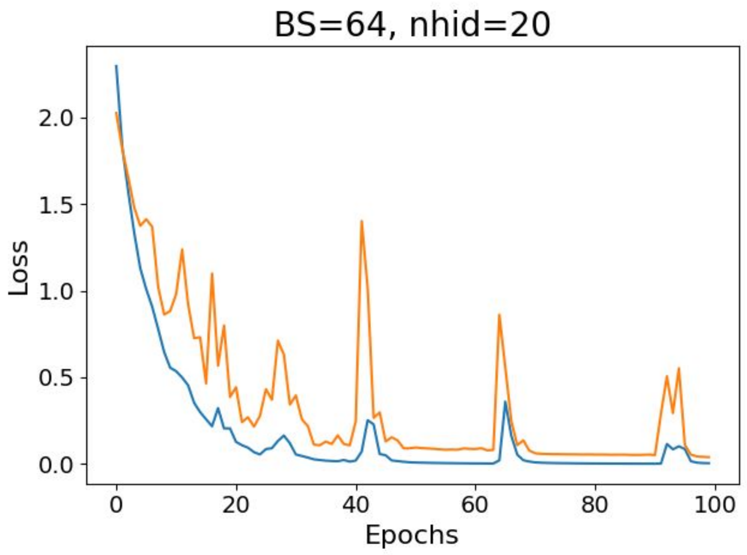
However, the sequences produced by the model are far from perfect. Yet, some types of words are better translated than others:

- The model does not handle long words well at all. Example: “incomprehensibilities” -> “incorpicesteresscay”.
- It also struggles with words containing dashes: “well-mannered” -> “ellway-eateray”
- It often does not correctly remember all intermediate letters and also appends incorrect suffixes.: “artificial” -> “artiasionclay”

- It also fails to move the correct letters from the beginning of the word to the end before appending the suffix “fake” -> “akesday”
- Generally for this model it is much easier to say that the only words that are consistently translated correctly, are very short words like “is” -> “isway”, “the” -> “ethay”, “way” -> “ayway”
-

source: the air conditioning is working
translated: ethay airway oningingday ishay odhughschay
source: roomba incomprehensibilities brink fake aardvark lebensfreude
translated: oorsshay instiestuationssay infbay akesday araodbay-ingsray eneffay-orterway
Source: well-mannered yyyyyyyyyyyy way artificial intelligence systems shopping
Translated: ellway-eateray yy-ydueredday ayway artitionfay-away-eta intelingway ossedsway-away-etay optionscay

2. RNN with Additive Attention



Final loss: train=0.004548936185626072 val=0.039627365403038524

This model performs much better compared to the previous one. Both quantitatively in terms of its loss, as well as qualitatively looking at the sequences it produces.

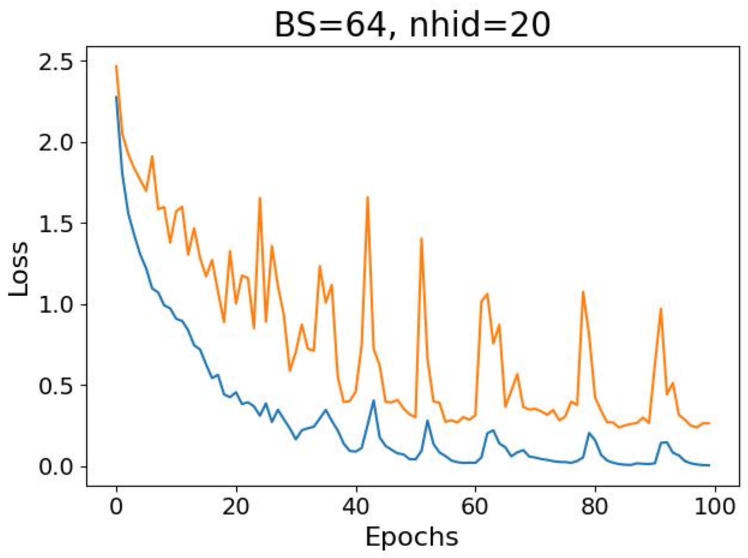
Most failure cases seem to be resolved: The attention model handles long sequences much better and makes significantly fewer copying mistakes. The only consistent failure case I can still identify are **very** long sequences “incomprehensibilities” -> “incomprehensitieiswa”. Here are a few of my test cases:

source:	roomba incomprehensibilities brink fake aardvark lebensfreude
translated:	oombay incomprehensitieiswa inkbray akefay awrerkrarmay ebensfreudeway
source:	well-mannered yyyyyyyyyyyy way artificial intelligence systems
translated:	ellway-annereday ylylylyay ayway artificialway intelligenceway ystemssay

It seems to be thrown off a little bit by my test case of just a number of “y”s. However, this is not surprising at all since the sequence “yy” does not exist in english and the model never encountered it during training. In this case it fails to correctly copy the sequence and inserts “l”s for every other “y”.

3. Transformer with Scaled Dot-Product Attention

3.5 Encoder with Causal Attention

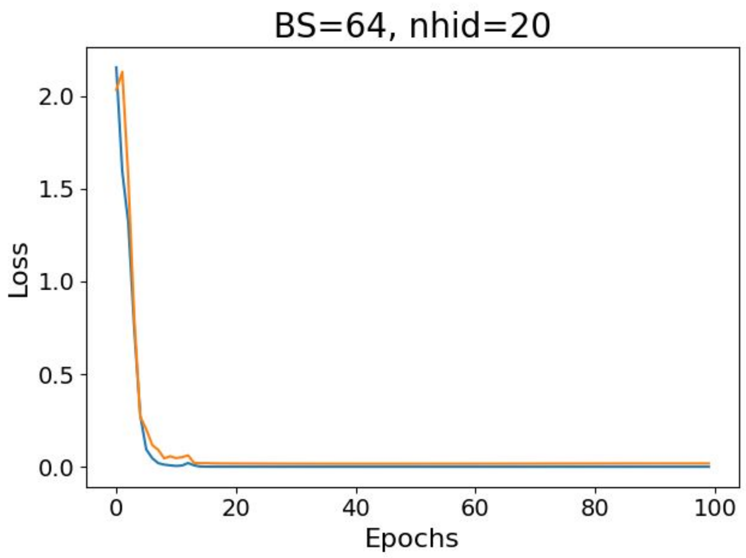


Final loss: train=0.0032218097584872625 val=0.2624206721419406

source: the air conditioning is working
translated: ethay airway onditioningcay isway orkingway
source: roomba incomprehensibilities brink fake aardvark lebensfreude
translated: oombaray incomprehinsibi inkbray akefay ardvarkway ebensfreudedlay
source well-mannered yyyyyyyyyyyy way artificial intelligence systems
translated: ellway-aneredmay yyyyyylyway ayway artificialway intelligenceway ystemssay

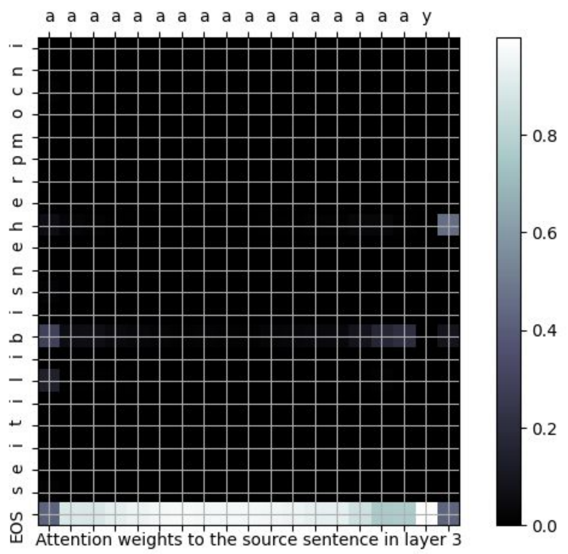
The results are very close to the previous model and still much better compared to the RNN decoder without attention. It has the significant advantage of much faster training over the RNN encoder (with and without attention) due to the transcoder's parallel nature. It also struggles with very long sequences, and often fails to copy the correct character to the end of the word. More analysis in Section 4, where we look at the attention visualization.

3.6 Encoder with Non-Causal Attention



Final Loss: Train=3.8524714970921557**e-07** val=0.017805870629819698

This model is a complete failure. It learns to cheat the system through its ability to view future elements in the sequence due to the missing causal filter. It is not an autoregressive model anymore. I think it might just learn to store the entire training data, which might explain its incredibly low loww. The visualized attention shows us that it almost exclusively pays attention to the "end of sequence" character:



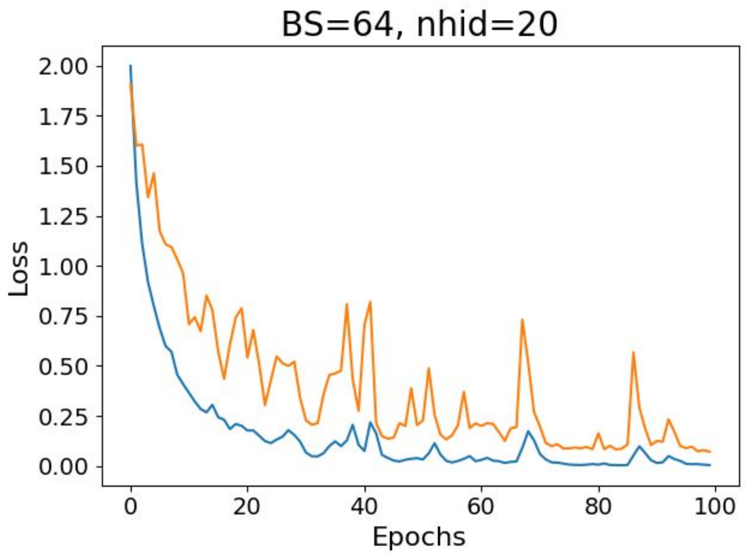
In the test cases, it always just produces a sequence of "a"s followed by a y at the end. Which might be due to the fact that on new data the softmax output is equal for the entire vocabulary, and a is the first letter in it, except for the last character. However, I have not verified if that is actually the case.

3.7 Additive Attention vs Scaled Dot-Product Attention

Generally one advantage additive attention has over dot-product attention is that it performs better, when we don't scale the dot product. Obviously, the scaling factor is an intended countermeasure to prevent softmax activation in its low gradient boundary regions. However, even with a scaling factor in place, dot product attention is still a linear transformation while additive attention is non-linear, which is probably the reason for its better performance. The biggest disadvantage of additive attention is its computational complexity, which makes training slow.

On the other hand, the primary disadvantage of scaled dot product attention is its sub-optimal performance. However, it can leverage highly optimized matrix multiplication, which results in much better performance.

Extra: RNN Encoder with Causal Transformer Decoder



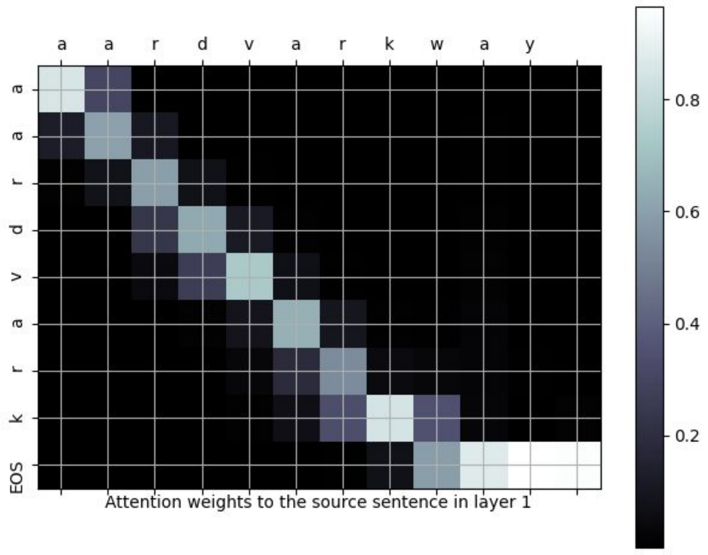
Final Loss: Train=0.004559061088735662 Val=0.07124420738424912

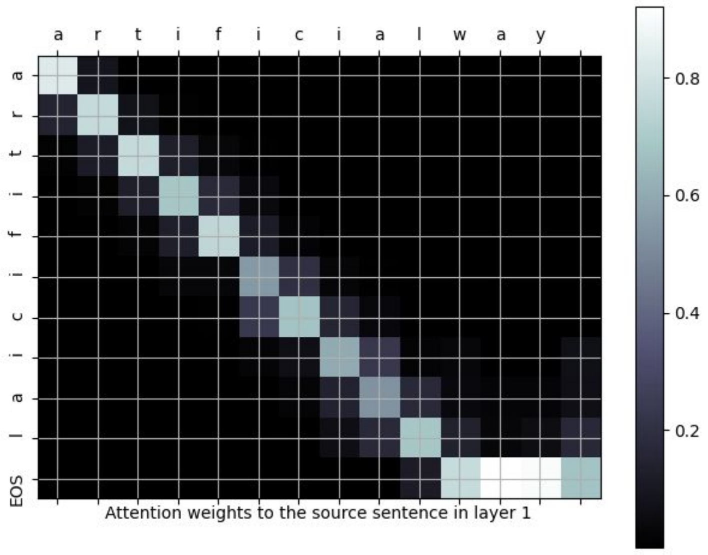
I played around a little more (and just like you suggested) I saw the best transformer decoder performance when using the RNN Encoder in combination with the causal transformer decoder. However, the RNN with additive attention still seems to be the best model over all, albeit also the slowest to train.

4. Visualizing Attention

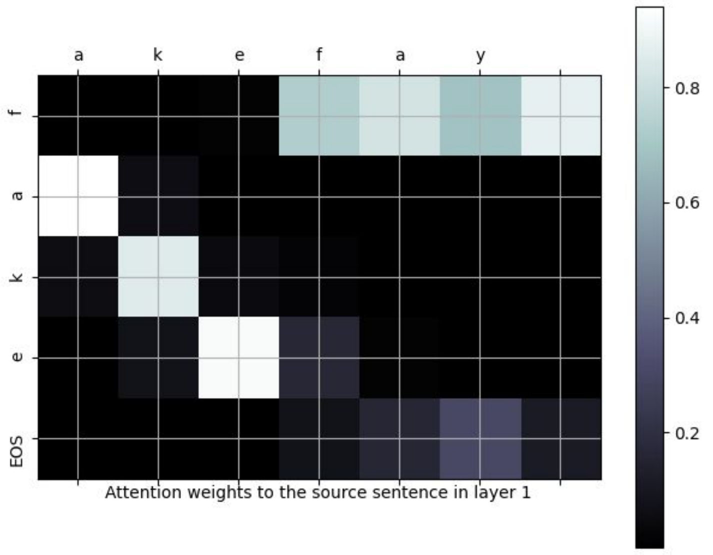
Again, the RNN with additive attention shows the best behavior. In most cases it works perfectly by paying attention to the input sequence character at the same position as the current position, to copy it. Then, to append the correct suffix, it pays attention to the first letters of the input sequence, if the first character is a consonant. It seems like it has learned the default behavior of appending “way” for vowels, in which case it does not go back to look at the beginning of the word. An initial consonant is treated as a special case, in which case attention is used to “remind” the model of appending the special case suffix “ay” instead of the default “way”. The same is true for the other special case of consonant sequences like “sh” and “th”.

The default behavior is clearly visible in the following plots:

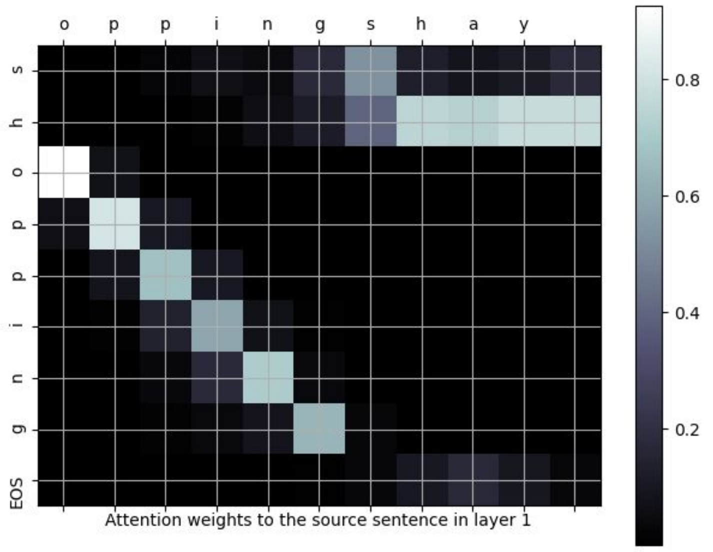




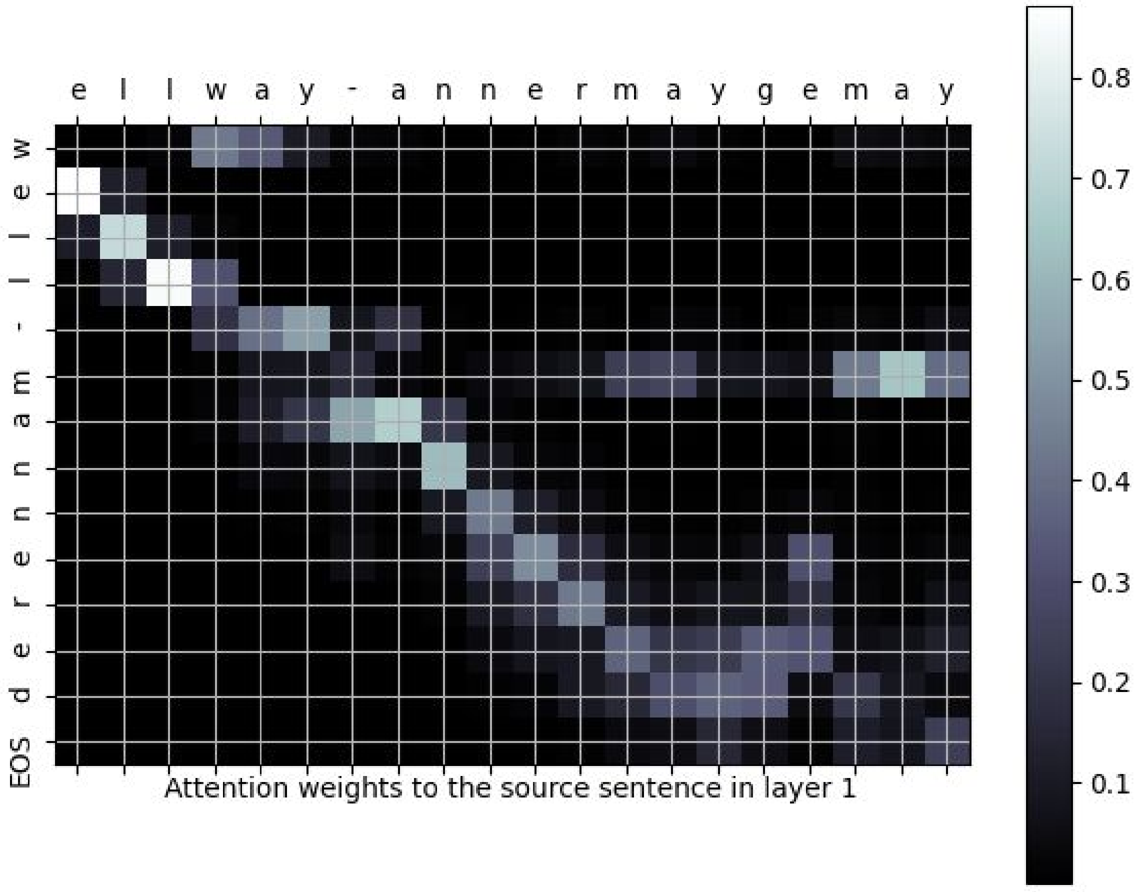
In the special case of a consonant as the first character, the model pays attention to the first characters, instead of the EOS character, when appending the suffix:



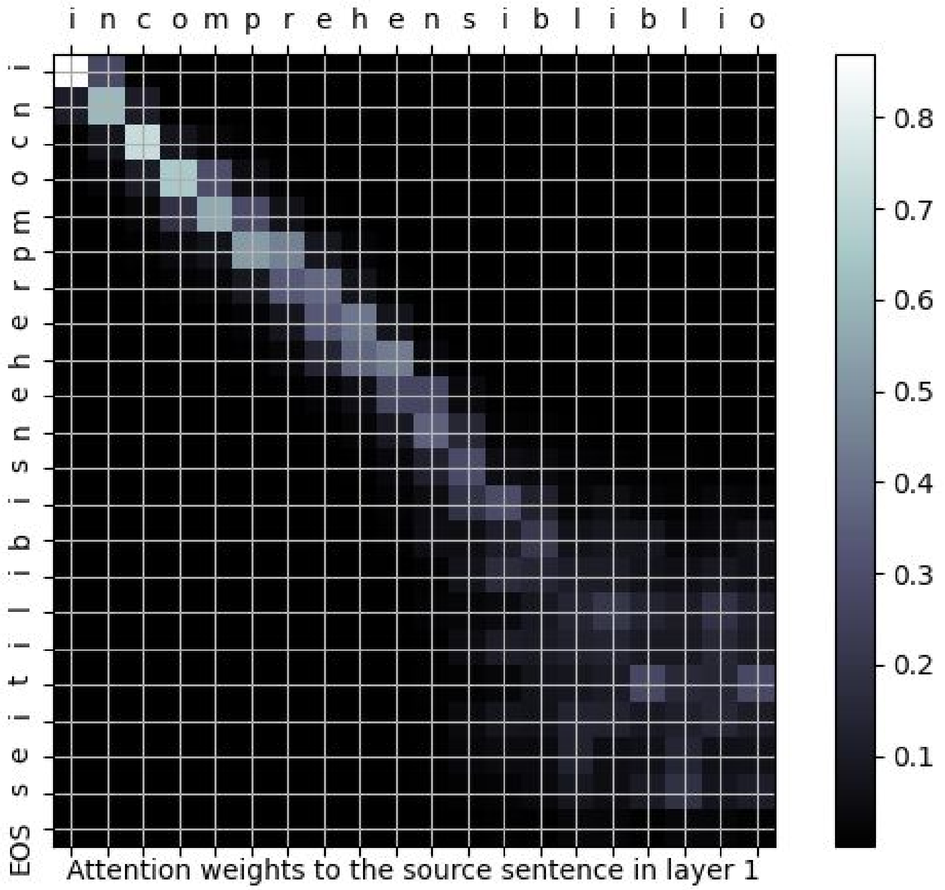
And we observe similar behavior for special consonant sequences at the beginning of the word, except that here it first pays attention to the initial consonant when moving it to the end, and then to the second constant for appending the suffix:



It even learned to handle the dash correctly:

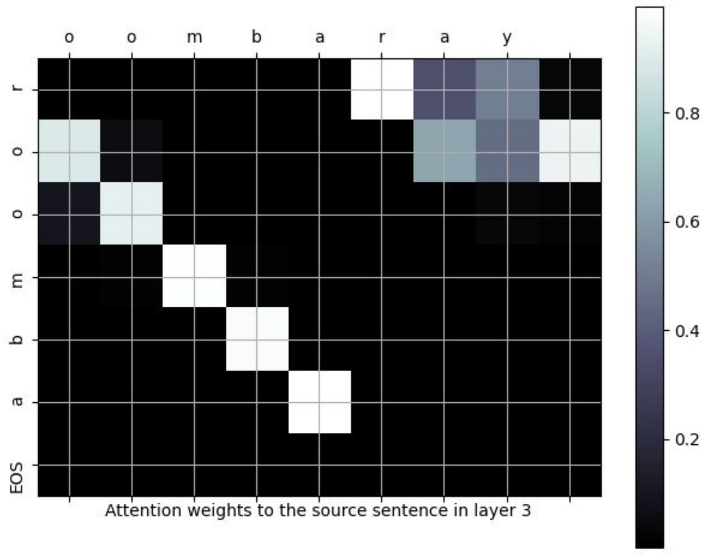


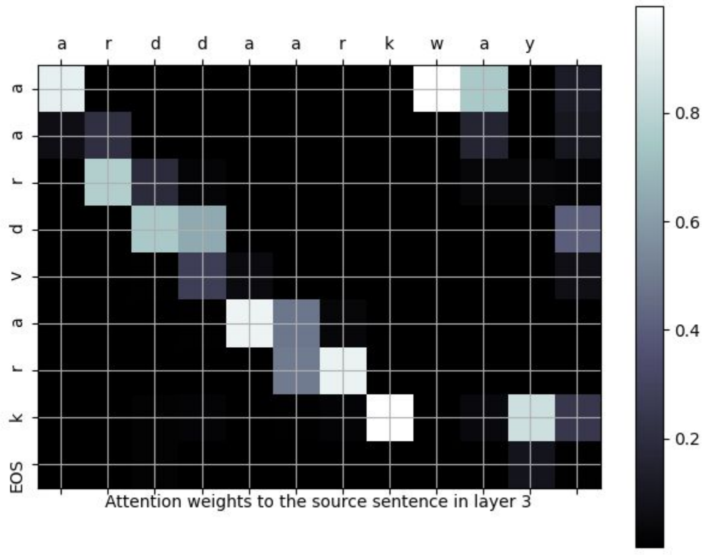
However, here we also see its biggest weakness: Unusually long sequences. Here the attention does not seem focused on the diagonal anymore, it starts to fade to the sub diagonals. Here is an extreme example:

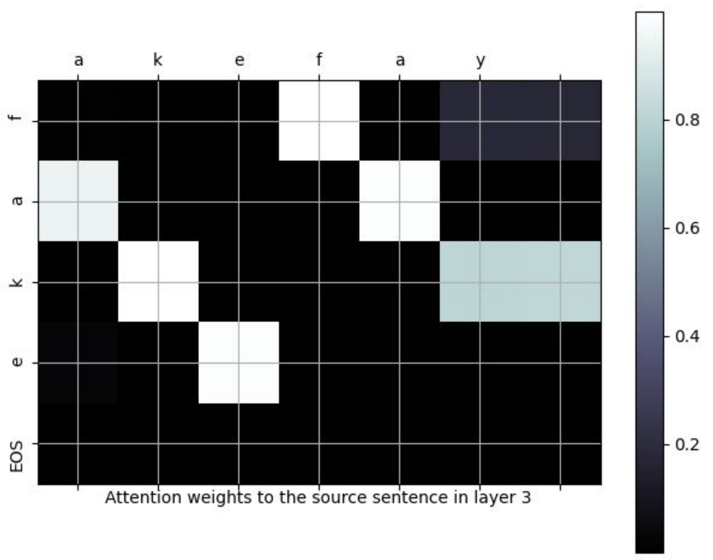


At the very end, the attention is all over the place and the model even fails to append the suffix. I believe this is simply due to insufficient training on long sequences. The attention mechanism just never learns how to handle characters further back in a long sequence. The problem could probably easily be fixed by applying data augmentation e.g. by inserting random letters in the middle of words, which does not affect the rules of pig-latin but would expose the model to more long sequences.

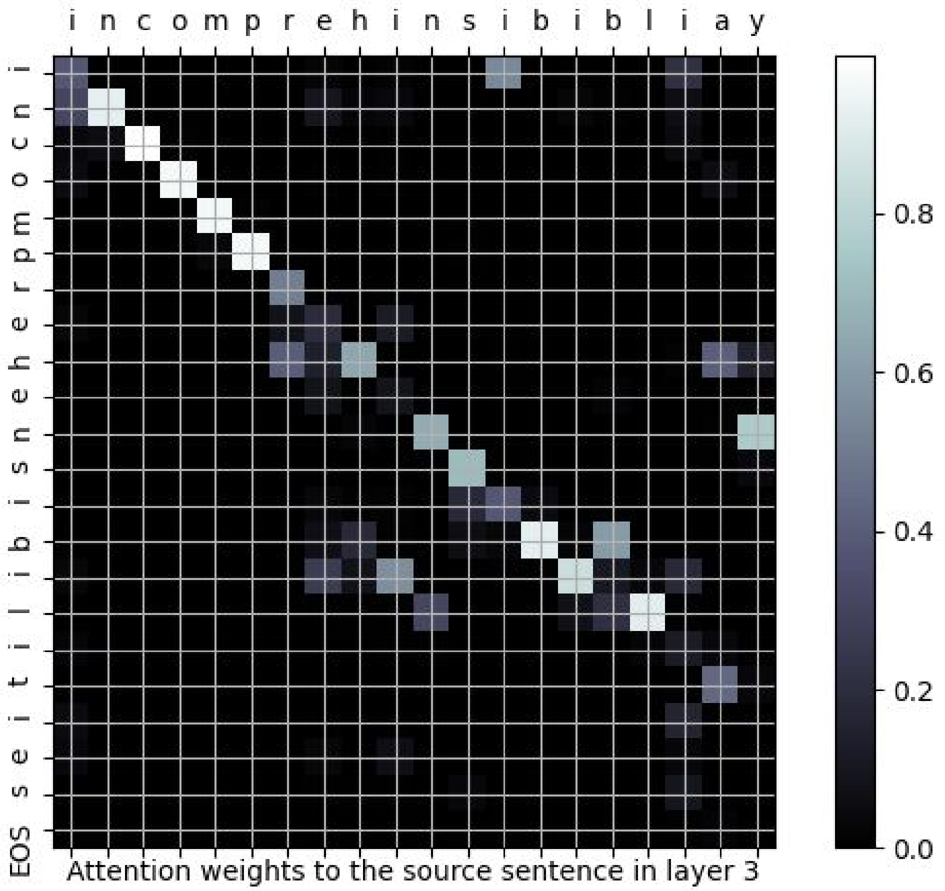
The transformer with causal dot-product attention behaves largely in the same way, but the cases I described earlier for the RNN are not handled as efficiently. This model apparently always diverts its attention back to the first characters, it has not learned the very helpful default behavior, like the RNN has. Additionally, its focus is much more concentrated on the diagonal. This might simply be a sign of the more efficient training. The RNN might show the same behavior, if trained longer.

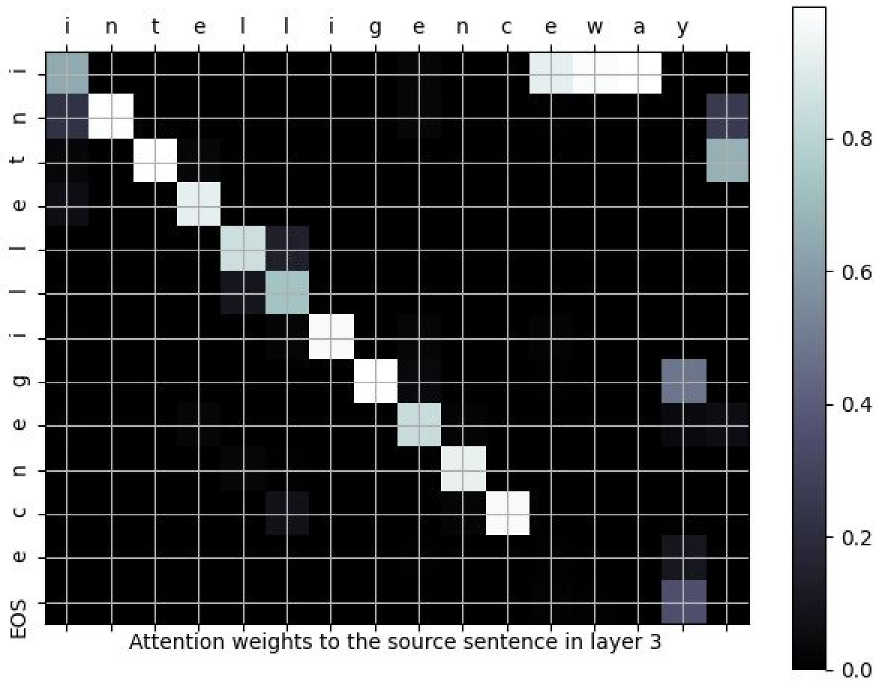

o
o
m
b
a
r
a
y
r
o
o
m
b
a
EOS
Attention weights to the source sentence in layer 3
0.8
0.6
0.4
0.2


a
r
d
d
a
a
r
k
w
a
y
a
a
r
d
v
a
r
k
EOS
Attention weights to the source sentence in layer 3
0.8
0.6
0.4
0.2

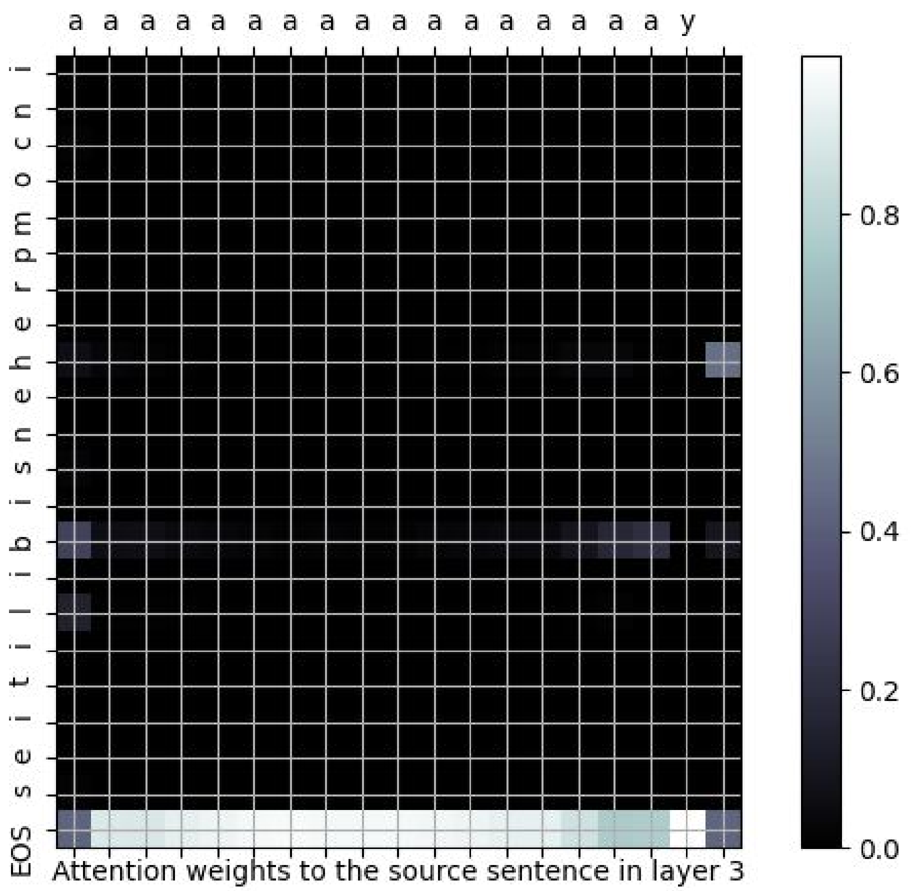


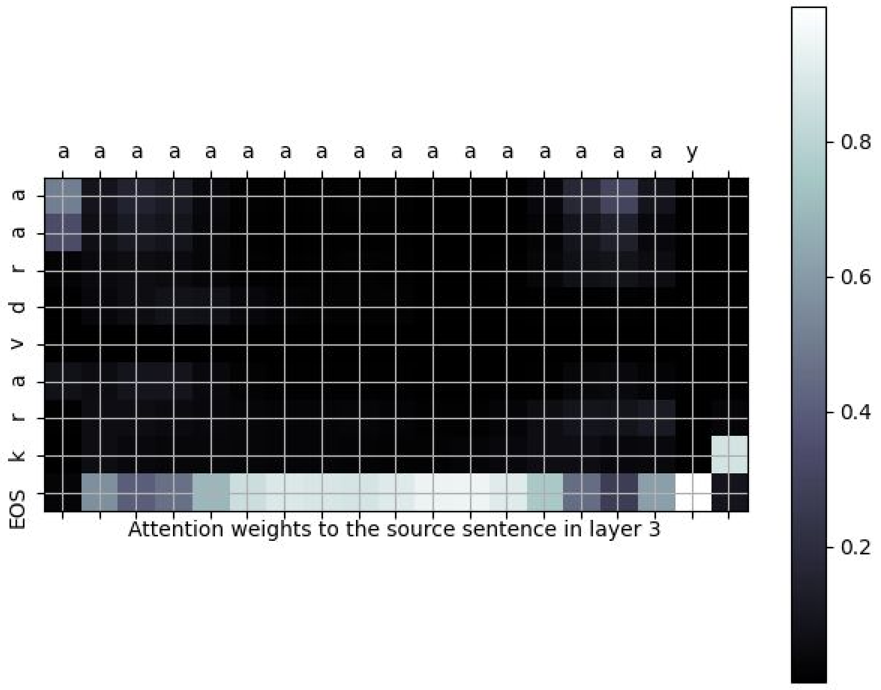
It also struggles with very long sequences, in an even more catastrophic way (again: could probably be resolved through data augmentation):


i n c o m p r e h i n s i b i b l i a y
i n c o m p r e h e n s i b i l i t i e s EOS
0.0
0.2
0.4
0.6
0.8
Attention weights to the source sentence in layer 3



Finally, the transformer model with non-causal attention, has not learned any useful attention mechanism whatsoever. It simply always only looks at the EOS, yet it doesn't even get the number of characters right:


a a a a a a a a a a a a a a a a a y
i n c o m p r e h e n s i b i l i t i e s EOS
0.8
0.6
0.4
0.2
0.0
Attention weights to the source sentence in layer 3



Attention weights to the source sentence in layer 3