DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Homework Assignment No. 13:

HW No. 13: Transformers

submitted to:

Professor Joseph Picone
ECE 8527: Introduction to Pattern Recognition and Machine Learning
Temple University
College of Engineering
1947 North 12th Street
Philadelphia, Pennsylvania 19122

April 20th, 2023

prepared by:

Gavin Koma
Email: gavintkoma@temple.edu

A. INTRODUCTION

This homework is a bit different as we have not really spent time discussing transformer implementation, thus, I want to spend less of this time focusing on the code and instead explain the individual parts that make up this implementation for my own understanding. The code does work in its entirety, and I have attached it on the last few pages of this assignment. Since the code is explained within the tutorial itself, I will limit my own screenshots and include portions that I feel are imperative to my own understanding of a transformer implementation.

The tutorial itself was easy to implement and I did not have many problems albeit at the very start. To begin this tutorial, one needs to have the proper versions of modules installed. I regularly try to maintain my modules at their most contemporary version however, this code utilizes older modules that I had newer versions of. As a result, many of my dependencies would clash and I was met with a variety of errors. I am a bit grateful for this as it forced me to delve into a subject that I have not previously studied: virtual environments.

I first spend some time reading through the vituralenvironments documentation and created my own venv for this assignment that would work well given the code. To the right, one can see a few of the required modules needed and their respective versions. I previously wondered why and how engineers are able to run so many different applications without maintaining dependencies between modules, but this was a huge eureka moment for me.

```
Last login: Thu Apr 20 09:55:00 on ttys000
(venv) gavinkoma@Gavins-MBP-2 ~ % pip list
Package            Version
------------------ ---------
altair             4.2.2
attrs              23.1.0
blis               0.7.9
catalogue          2.0.8
certifi            2022.12.7
charset-normalizer 3.1.0
click              8.1.3
confection         0.0.4
cymem              2.0.7
entrypoints        0.4
filelock           3.12.0
GPUtil             1.4.0
idna               3.4
Jinja2             3.1.2
jsonschema         4.17.3
langcodes          3.3.0
MarkupSafe         2.1.2
mpmath             1.3.0
murmurhash         1.0.9
networkx           3.1
numpy              1.24.2
packaging          23.1
pandas             2.0.0
pathy              0.10.1
pip                23.1
preshed            3.0.8
pydantic           1.10.7
pyrsistent         0.19.3
python-dateutil    2.8.2
pytz               2023.3
requests           2.28.2
setuptools         67.6.1
six                1.16.0
smart-open         6.3.0
spacy              3.5.2
spacy-legacy       3.0.12
spacy-loggers      1.0.4
srsly              2.4.6
sympy              1.11.1
thinc              8.1.9
toolz              0.12.0
torch              2.0.0
```

Virtual environments are incredibly important because they allow developers to create an isolated environment that have their own individual dependencies and configurations. This allows us to manage a project's dependencies separately so distinct projects can have distinct dependencies with different package versions. Another key point regarding virtual environments is that they help to ensure reproducibility. In other words, using a VE will allow code to run in the same way on different machines while utilizing the same isolated environment for development.

As someone that doesn't have a computer science background, learning about VEs is essential for my own growth as someone that wants to work in the data science field.

B. MODEL ARCHITECTURE

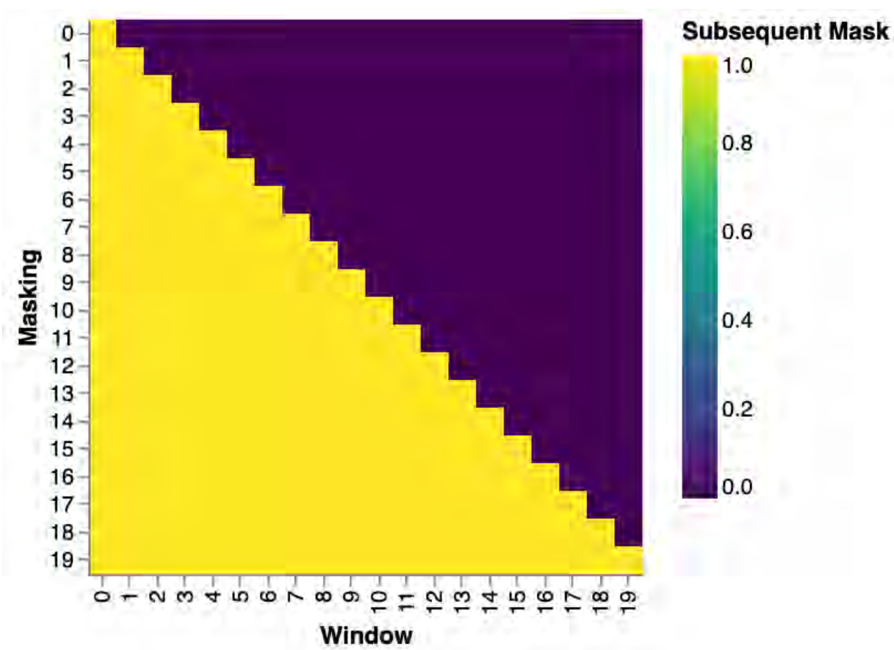
The transformer algorithm is one type of deep learning model that has most notably been used in the field of natural language processing (NLP) and was first introduced by Vaswani in 2017. The algorithm itself is based on the concept of self-attention and thus allows the model to focus on different parts of the input sequence when processing each output. The algorithm uses a series of self-attention layers (which are shown later in the code) and feedforward layers (also seen later) to process the given input data. When the model approaches a self-attention layer, the model functions to learn which parts of the input sequence to focus on and how much attention to give each part.

The first portion of this paper specifies that most competitive neural sequence transduction models rely on an encoder-decoder structure. Within said structure, the encoder will map an input sequence of symbol representations to another sequence of continuous representations. Given these values, the decoder then functions to generate an output sequence of symbols one variable at a time. One unique and important aspect in this architecture is that the model itself is auto-regressive. In other words, as the model progresses, it also consumes the previously generated output and utilizes it as an additional source of input when generating the subsequent output.

The encoder layer itself takes in the input sequence and applies self-attention to it and actively attempts to learn which parts of the sequence to focus on and how much attention to give to each part. The self-attention mechanism that we implement will allow the encoder to focus on dependencies between different parts of the input sequence. After this is completed, the encoder applies a feedforward neural network to the processed sequence and produces the previously mentioned encoded representation of the data.

The decoder layer then takes the encoded representation and applies another round of self-attention to it as well as attention to the input sequence. The decoder will use the attention mechanism to perform a similar action as the encoder: to focus on different parts of the encoded representation and input sequence to generate an output sequence. Just like the encoder, the decoder will then pass this to a feedforward neural network to process the data and output the next token in the sequence.

At this point of the tutorial, we output the same graph as shown in the post. This image shows an attention mask which shows the position that each tgt word is allowed to look at. It is used to control the attention mechanism found within the transformer algorithm and its purpose is to mask out certain elements within the input sequence so that the attention mechanism does not focus on them.



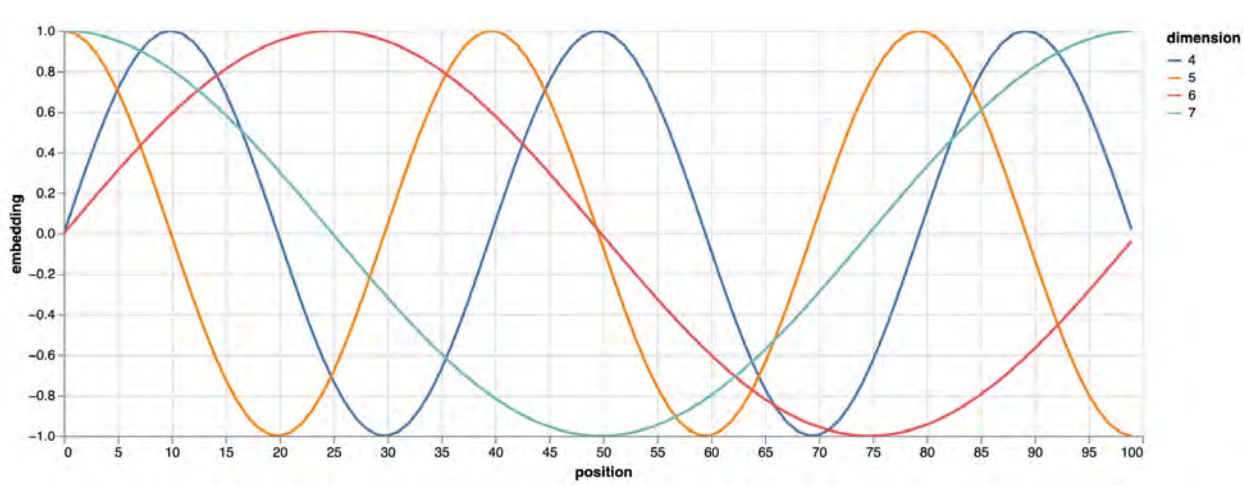
The attention function of our transformer is used to compute a weighted sum of the values. These weights are computed based on the similarity between the query and the keys themselves. Our given input consists of queries, keys, and their corresponding dimension. Our function computes the dot products of the query with all keys divided by the root of keys dimension and followed by a softmax function. The softmax function works to normalize the scores and ensures that they sum to a total of 1. These scores are then used as weights which will compute the previously mentioned weighted sum. This function is able to be visualized as a mechanism that allows the model to focus on different parts of the input sequence when processing each output. This function is used in both the encoder and decoder layers to best provide an output based on the input sequence and the encoded representation.

Our transformer specifically utilizes multi-head attention. This mechanism ensures that our model can attend to multiple positions within the input sequence at the same time. The query, keys, and value vectors are transformed to create multiple "heads" for attention. The queries come from the previous decoder layer and the memory keys and corresponding value vectors come from the output of the encoder. Doing so, allows every position in the decoder to attend over all positions in the input sequence. The encoder's attention layers have a similar element. However, in these self-attention layers, the keys, values, and queries all come from the same place: the output of the previous layer in the encoder.

Our transformer then moves onto a position-wise feedforward neural network which can be found fully connected in each encoder and decoder. The feedforward neural network contains two linear transformations with a ReLU activation in between them. I will not go in-depth regarding this portion as we have worked with these functions in previous homework assignment.

Our model does not contain recurrence or convolution and as a byproduct, we must include some information regarding the relative position of tokens in the sequence. We do so by including "positional encodings" in the input embeddings at the bottoms of the encoder and decoder stacks. Doing so, allows us to provide a method for the model to encode the order of the input sequence.

The process of positional encoding involves adding a set of fixed-length vectors to the input embeddings of each word in the sequence. These vectors are calculated using a sinusoidal method and the dimension of the corresponding embedding space. These sinusoidal functions ensure that the positional encoding vectors are unique for each position in the sequence and that they also have a smooth and continuous pattern that permits the model to interpolate between given positions. At this point in the tutorial, we are given a chance to output the following graph that shows varying sinusoidal functions of an embedding given their position:



The full model of our transformer is defined below along with a test of inference to try to generate a prediction of our model:

```
Gavin Koma
def make_model(
    src_vocab, tgt_vocab, N=6, d_model=512, d_ff=2048, h=8, dropout=0.1
):
    "Helper: Construct a model from hyperparameters."
    c = copy.deepcopy
    attn = MultiHeadedAttention(h, d_model)
    ff = PositionwiseFeedForward(d_model, d_ff, dropout)
    position = PositionalEncoding(d_model, dropout)
    model = EncoderDecoder(
        Encoder(EncoderLayer(d_model, c(attn), c(ff), dropout), N),
        Decoder(DecoderLayer(d_model, c(attn), c(attn), c(ff), dropout), N),
        nn.Sequential(Embeddings(d_model, src_vocab), c(position)),
        nn.Sequential(Embeddings(d_model, tgt_vocab), c(position)),
        Generator(d_model, tgt_vocab),
    )

    # This was important from their code.
    # Initialize parameters with Glorot / fan_avg.
    for p in model.parameters():
        if p.dim() > 1:
            nn.init.xavier_uniform_(p)
    return model
```

```
ansformers.py
1 usage   Gavin Koma *
def inference_test():
    test_model = make_model(11, 11, 2)
    test_model.eval()
    src = torch.LongTensor([[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]])
    src_mask = torch.ones(1, 1, 10)
    memory = test_model.encode(src, src_mask)
    ys = torch.zeros(1, 1).type_as(src)
    for i in range(9):
        out = test_model.decode(
            memory, src_mask, ys, subsequent_mask(ys.size(1)).type_as(src.data)
        )
        prob = test_model.generator(out[:, -1])
        _, next_word = torch.max(prob, dim=1)
        next_word = next_word.data[0]
        ys = torch.cat(
            [ys, torch.empty(1, 1).type_as(src.data).fill_(next_word)], dim=1
        )
    print("Example Untrained Model Prediction:", ys)
Gavin Koma
def run_tests():
    for _ in range(10):
        inference_test()
```

C. MODEL TRAINING

This portion of the assignment was by far the most confusing for me to understand. I will try my best to explain the methods introduced in this article and to explain their implementation. The model we implement in this tutorial contain the following essential steps to properly train:

1. Batches and Masking Creation
2. Training Loop
3. Providing Training Data and Batching
4. Hardware for Training
5. Optimizer
6. Regularization

Batches and masking are considered fairly common tools that are used for the implementation of transformer models. Batches are used to improve the efficiency of our training process. This occurs by processing multiple input sequences simultaneously. During this portion, several input sequences are grouped together into one single batch and are processed at the same time instead of processing each batch individually. Doing so, allows us to reduce the time required to not only train the model but also allows us to utilize parallel computing architectures. Batches are typically processed in a parallel fashion in which the input sequences are stacked together into a singular matrix. Unfortunately, input sequences within a batch may have different lengths and it is common for researchers to include padding when working with the shorter sequences to ensure that they are the same length as the longest sequence.

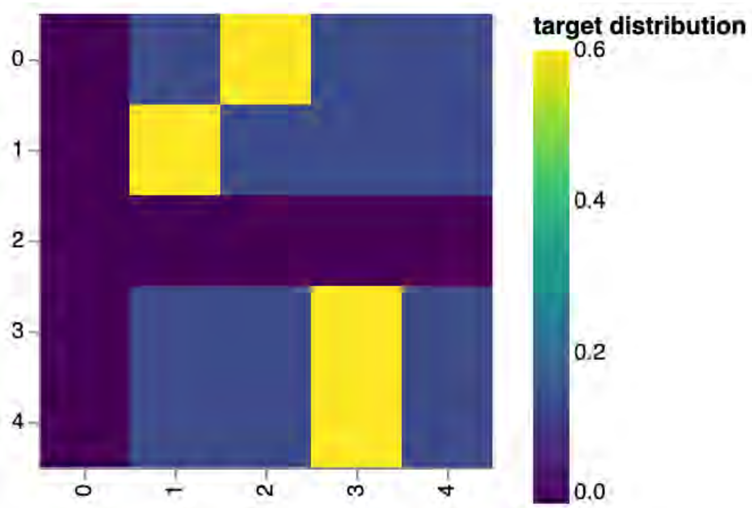
Masking is commonly used to selectively control which elements of the input sequence are attended to during the processing phase. There are two common types of masks that are used: padding masks and sequence masks. I will only focus on sequence masks as this is the type of mask that is utilized in our model. Sequence masks are binary matrices that are used to mask out future elements found in the input sequence. During the generation of an output sequence, it is important that the model only attends to the previous elements in the input sequence. This matrix is the same shape as the input or output sequence and each element within it is either a 1 or a 0. A value of 1 means that the corresponding element in the sequence *should* be attended to while a value of 0 allows the model to know that the element should be masked out.

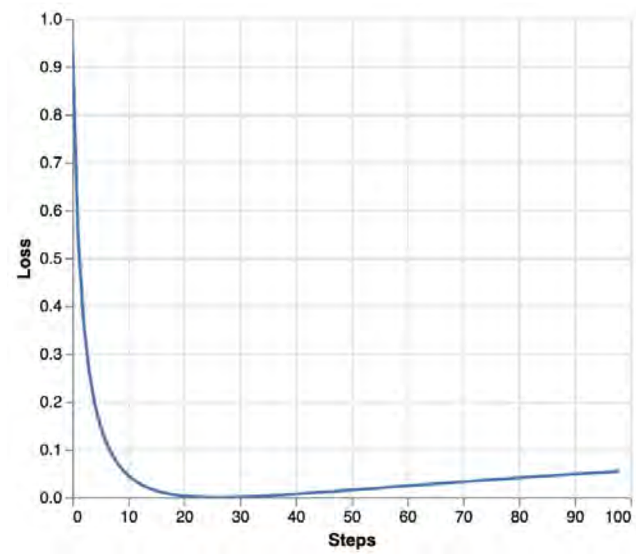
The training loop included in the program is fairly self-explanatory and will not be further explained here. The code itself is straight forward and similar methods have been implemented to train our models in the past in a loop-like model. At this point I did need to reassess my method of running this algorithm. The researchers utilized 8 NVIDIA P100 GPUs. Unfortunately, I do not have 8 GPUs casually laying around my PhD-student-only-household. I first attempted to run this code on my lab's remote GPU but training was taking well over an hour. I got a bit lucky here and have a close friend that pays for Google Collab's Premium Service which supplies great computer resources for heavy computational loads.

The final portion of training was to utilize the Adam optimizer. Adam, short for Adaptive Moment Estimation, is designed to update the weights of a neural network in an efficient and *adaptive* manner. It utilizes two main strategies to complete this task: adaptive learning rates and momentum-based updates. The adaptive learning rate strategy adjusts the learning rate for each weight in the network based on the magnitude of its gradients. Doing so helps to prevent the network from becoming stuck in steep regions when the calculated gradients are large. The momentum-based update strategy helps to accelerate the optimization process by using a moving average of the gradients to update the computed weights. This helps the optimization process to overcome small, local minima and to continue moving towards the global minimum. Adam combines these two strategies by using the first and second moments of the gradients to compute the adaptive learning rates and then to compute the moving average of the gradients to update the weights.

We conclude this training portion by performing label smoothing regularization using KL divergence loss. Label smoothing is a regularization technique that prevents our model from overfitting to the training data by artificially smoothing our target labels. Generally, label smoothing involves replacing the hard target labels with a smoothed distribution across all classes. The idea behind this concept is too introduce a small amount of noise into the training labels in an attempt to force the model to learn more robust features. In our model, we utilize the KL divergence loss to penalize the difference between the predicted probability distribution and the smoothed target distribution. This loss is then used as the objective function for training by the model. By minimizing this loss, we allow our model to complete our previously stated goal: to learn more robust features.

At this point in the tutorial, our code outputs the following example of label smoothing and loss graph:





D. SYNTHETIC EXAMPLE

The next portion of this assignment has us focus on a simple copy-task given a random set of input symbols from some small vocabulary. The goal of this portion is to generate these same symbols back to us. The code implemented generates synthetic data of some size. We introduce a simple loss compute and train functions and we predict our training using greedy decoding. Greedy decoding is a simple decoding strategy that is commonly used in sequence-to-sequence models and the strategy involves selecting the output token with the highest probability at each decoding step and appending it to the output sequence.

The code itself to worked with no problems on my GPU but the article does not spend much time discussing this portion. Therefore, I also will not spend much time regarding this synthetic example but will instead focus more on the real-world example that is performed next in much greater detail. The code that was ran on my laptop can be found at the end of this submission in .pdf format with all output results maintained and included.

E. REAL WORLD EXAMPLE

To conclude this assignment, we utilize our model on a real-world example. In this case, we are undertaking the German-English translation task. We first load the data, which is self-explanatory as we have performed this task in nearly every assignment since the beginning of the semester. Once our data is loaded, we see that we have the following vocabulary sizes for German and for English respectively: 59981 and 36745.

As previously discussed, it is incredibly important that we include batching to maximize the speed of our algorithm. We do want to evenly divide our batches and we want to do so with minimal padding. The code provided does this for us. It is important to have minimal padding in a transformer as it can lead to an increased efficiency and a faster training time. By minimizing these two factors, we are able to decrease the amount of computation required to process the sequences and it allows the model to focus more on the actual data and less on the padding. This also will hopefully lead to better accuracy and an overall faster convergence.

Training the system took a notable amount of time, but included below is a small snippet of the output to show the general trend of loss from our model:

```
| ID | GPU | MEM |
------------------
|  0 | 79% | 26% |
[GPU0] Epoch 0 Validation ====
(tensor(3.8904, device='cuda:0'), <__main__.TrainState object at 0x7fd3af1372b0>)
[GPU0] Epoch 1 Training ====
Epoch Step:      1 | Accumulation Step:   1 | Loss:   3.83 | Tokens / Sec:  1955.9 | Learning Rate: 2.4e-04
Epoch Step:     41 | Accumulation Step:   5 | Loss:   4.01 | Tokens / Sec:  1680.1 | Learning Rate: 2.6e-04
Epoch Step:     81 | Accumulation Step:   9 | Loss:   3.87 | Tokens / Sec:  1632.0 | Learning Rate: 2.7e-04
Epoch Step:    121 | Accumulation Step:  13 | Loss:   3.60 | Tokens / Sec:  1658.1 | Learning Rate: 2.8e-04
Epoch Step:    161 | Accumulation Step:  17 | Loss:   3.63 | Tokens / Sec:  1674.1 | Learning Rate: 2.9e-04
Epoch Step:    201 | Accumulation Step:  21 | Loss:   3.57 | Tokens / Sec:  1663.8 | Learning Rate: 3.0e-04
Epoch Step:    241 | Accumulation Step:  25 | Loss:   3.61 | Tokens / Sec:  1659.0 | Learning Rate: 3.1e-04
Epoch Step:    281 | Accumulation Step:  29 | Loss:   3.54 | Tokens / Sec:  1676.2 | Learning Rate: 3.2e-04
Epoch Step:    321 | Accumulation Step:  33 | Loss:   3.57 | Tokens / Sec:  1670.8 | Learning Rate: 3.3e-04
Epoch Step:    361 | Accumulation Step:  37 | Loss:   3.54 | Tokens / Sec:  1673.2 | Learning Rate: 3.4e-04
Epoch Step:    401 | Accumulation Step:  41 | Loss:   3.39 | Tokens / Sec:  1678.1 | Learning Rate: 3.5e-04
Epoch Step:    441 | Accumulation Step:  45 | Loss:   3.26 | Tokens / Sec:  1654.8 | Learning Rate: 3.6e-04
Epoch Step:    481 | Accumulation Step:  49 | Loss:   3.37 | Tokens / Sec:  1635.7 | Learning Rate: 3.7e-04
Epoch Step:    521 | Accumulation Step:  53 | Loss:   3.28 | Tokens / Sec:  1626.0 | Learning Rate: 3.8e-04
Epoch Step:    561 | Accumulation Step:  57 | Loss:   3.37 | Tokens / Sec:  1647.0 | Learning Rate: 4.0e-04
Epoch Step:    601 | Accumulation Step:  61 | Loss:   3.22 | Tokens / Sec:  1673.1 | Learning Rate: 4.1e-04
Epoch Step:    641 | Accumulation Step:  65 | Loss:   3.04 | Tokens / Sec:  1681.3 | Learning Rate: 4.2e-04
Epoch Step:    681 | Accumulation Step:  69 | Loss:   2.93 | Tokens / Sec:  1651.9 | Learning Rate: 4.3e-04
Epoch Step:    721 | Accumulation Step:  73 | Loss:   2.93 | Tokens / Sec:  1688.6 | Learning Rate: 4.4e-04
Epoch Step:    761 | Accumulation Step:  77 | Loss:   2.89 | Tokens / Sec:  1671.6 | Learning Rate: 4.5e-04
Epoch Step:    801 | Accumulation Step:  81 | Loss:   2.99 | Tokens / Sec:  1664.0 | Learning Rate: 4.6e-04
Epoch Step:    841 | Accumulation Step:  85 | Loss:   3.11 | Tokens / Sec:  1658.2 | Learning Rate: 4.7e-04
Epoch Step:    881 | Accumulation Step:  89 | Loss:   2.73 | Tokens / Sec:  1670.7 | Learning Rate: 4.8e-04
```

F. RESULTS

The results of this model can be found below. We output three separate examples prioritizing three different methods: encoder self-attention, decoder self-attention, and decoder source-attention. Doing so, allows us to compare the three different attention mechanisms. Encoder self-attention is used in the encoder to calculate the context vector for each token in the input sequence. In this situation, the input sequence is passed to the model and each token attends to all other tokens in the same input sequence in order to calculate its own context vector. In this situation, the context vector only depends on the input sequence and not on the output sequence. Decoder self-attention is used in the decoder to calculate the context vector for each token in the output sequence. Here, the model generates the output sequence one token at a time and each token will attend to all the previous tokens that were generated by the decoder in order to calculate its context vector. In this situation, the context vector depends only on the previously generated tokens and not on the input sequence. Finally, there is the decoder source-attention. This method is used in the decoder to attend to the encoder's output sequence. In this situation, the decoder will attend to all the tokens in the encoder's output sequence to calculate the context vector for the current token being generated. Here, the decoder will use information from the input sequence to generate the output sequence.

Encoder Self-Attention:

```
Example 0 ========

Source Text (Input)        : <s> Zwei Männer mit <unk> sprechen miteinander auf einem Freiluftmarkt . </s>
Target Text (Ground Truth) : <s> Two men in <unk> 's have a discussion in an outdoor market . </s>
Model Output               : <s> Two men are talking to each other in an open air market . </s>
```

Decoder Self-Attention:

```
Example 0 ========

Source Text (Input)        : <s> Ein Mann mit Rastalocken <unk> sein Ohr , um ein <unk> hören zu können . </s>
Target Text (Ground Truth) : <s> A man with dreadlocks is plugging his ear to hear a phone call . </s>
Model Output               : <s> A man with a <unk> hat is seen his ear to listen to be a music stand . </s>
```

Decoder Source-Attention:

```
Example 0 ========

Source Text (Input)        : <s> Ein Mann isst in einem Restaurant zu Mittag . </s>
Target Text (Ground Truth) : <s> A man in a restaurant having lunch . </s>
Model Output               : <s> A man eats lunch to a restaurant . </s>
```

I have initially not included the heat maps for these outputs as they can be generally confusing to read/interpret. I have included them, instead, at the very end of this document where they are found in the output of the code implementation. From the above results, it seems like the encoder self-attention is providing the best output for German-English translation. It appears to be the easiest to understand of the three. There are simple mistakes and the other two methods are still legible and their intention is clear, the encoder self-attention is the most accurate.

G. CONCLUSION

This was, difficult, to say the least. I didn't realize how in-depth some of these algorithms could be or how complicated they can get. In many classes, the algorithms implemented don't even approach half of the intense work that would be required to design something like this from scratch. As always though, I find myself learning something from these assignments. The code itself would have been miserable to comment out line-by-line but I hope my overall explanations for each section were satisfactory.

I wrote this document primarily from the intent of learning whatever I could and teach myself more about concepts that I don't know in-depth. I feel like this assignment facilitated me learning a lot regarding encoder/decoders, the importance of model training methods, and as I stated in the beginning of this document, virtual environments.

Attached on the next few pages is the entire print out of my completed and functioning implementation of this code.

```
# Uncomment for colab
#
!pip install -q torchdata==0.3.0 torchtext==0.12 spacy==3.2 altair GPUtil
!python -m spacy download de_core_news_sm
!python -m spacy download en_core_web_sm
```

```
Requirement already satisfied: pathy>=0.3.5 in /usr/local/lib/python3.9/dist-packages (from spacy<3.3.0,>=3.2.0->de-core-news
Requirement already satisfied: smart-open<7.0.0,>=5.2.1 in /usr/local/lib/python3.9/dist-packages (from pathy>=0.3.5->spacy<3
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.9/dist-packages (from pydantic!=1.8,!=1.8
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.9/dist-packages (from requests<3.0.0,>=2.13.0->sp
Requirement already satisfied: charset-normalizer~=2.0.0 in /usr/local/lib/python3.9/dist-packages (from requests<3.0.0,>=2.1
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /usr/local/lib/python3.9/dist-packages (from requests<3.0.0,>=2.13.0-
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.9/dist-packages (from requests<3.0.0,>=2.13.0->spacy<3.
Requirement already satisfied: click<9.0.0,>=7.1.1 in /usr/local/lib/python3.9/dist-packages (from typer<0.5.0,>=0.3.0->spacy
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.9/dist-packages (from jinja2->spacy<3.3.0,>=3.2.0->c
Installing collected packages: de-core-news-sm
Successfully installed de-core-news-sm-3.2.0
✔ Download and installation successful
You can now load the package via spacy.load('de_core_news_sm')
2023-04-20 21:26:38.364299: I tensorflow/core/platform/cpu_feature_guard.cc:182] This TensorFlow binary is optimized to use a
To enable the following instructions: AVX2 AVX512F FMA, in other operations, rebuild TensorFlow with the appropriate compiler
2023-04-20 21:26:39.361329: W tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-TRT Warning: Could not find TensorRT
DEPRECATION: https://github.com/explosion/spacy-models/releases/download/en_core_web_sm-3.2.0/en_core_web_sm-3.2.0-py3-none-a
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting en-core-web-sm==3.2.0
  Downloading https://github.com/explosion/spacy-models/releases/download/en_core_web_sm-3.2.0/en_core_web_sm-3.2.0-py3-none-
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 13.9/13.9 MB 37.6 MB/s eta 0:00:00
Requirement already satisfied: spacy<3.3.0,>=3.2.0 in /usr/local/lib/python3.9/dist-packages (from en-core-web-sm==3.2.0) (3.
Requirement already satisfied: pathy>=0.3.5 in /usr/local/lib/python3.9/dist-packages (from spacy<3.3.0,>=3.2.0->en-core-web-
Requirement already satisfied: spacy-legacy<3.1.0,>=3.0.8 in /usr/local/lib/python3.9/dist-packages (from spacy<3.3.0,>=3.2.(
Requirement already satisfied: preshed<3.1.0,>=3.0.2 in /usr/local/lib/python3.9/dist-packages (from spacy<3.3.0,>=3.2.0->en-
Requirement already satisfied: catalogue<2.1.0,>=2.0.6 in /usr/local/lib/python3.9/dist-packages (from spacy<3.3.0,>=3.2.0->e
Requirement already satisfied: blis<0.8.0,>=0.4.0 in /usr/local/lib/python3.9/dist-packages (from spacy<3.3.0,>=3.2.0->en-cor
Requirement already satisfied: pydantic!=1.8,!=1.8.1,<1.9.0,>=1.7.4 in /usr/local/lib/python3.9/dist-packages (from spacy<3.3
Requirement already satisfied: srsly<3.0.0,>=2.4.1 in /usr/local/lib/python3.9/dist-packages (from spacy<3.3.0,>=3.2.0->en-co
Requirement already satisfied: murmurhash<1.1.0,>=0.28.0 in /usr/local/lib/python3.9/dist-packages (from spacy<3.3.0,>=3.2.0-
Requirement already satisfied: requests<3.0.0,>=2.13.0 in /usr/local/lib/python3.9/dist-packages (from spacy<3.3.0,>=3.2.0->e
Requirement already satisfied: setuptools in /usr/local/lib/python3.9/dist-packages (from spacy<3.3.0,>=3.2.0->en-core-web-sm
Requirement already satisfied: typer<0.5.0,>=0.3.0 in /usr/local/lib/python3.9/dist-packages (from spacy<3.3.0,>=3.2.0->en-co
Requirement already satisfied: langcodes<4.0.0,>=3.2.0 in /usr/local/lib/python3.9/dist-packages (from spacy<3.3.0,>=3.2.0->e
Requirement already satisfied: jinja2 in /usr/local/lib/python3.9/dist-packages (from spacy<3.3.0,>=3.2.0->en-core-web-sm==3.
Requirement already satisfied: spacy-loggers<2.0.0,>=1.0.0 in /usr/local/lib/python3.9/dist-packages (from spacy<3.3.0,>=3.2.
Requirement already satisfied: tqdm<5.0.0,>=4.38.0 in /usr/local/lib/python3.9/dist-packages (from spacy<3.3.0,>=3.2.0->en-co
Requirement already satisfied: numpy>=1.15.0 in /usr/local/lib/python3.9/dist-packages (from spacy<3.3.0,>=3.2.0->en-core-web
Requirement already satisfied: thinc<8.1.0,>=8.0.12 in /usr/local/lib/python3.9/dist-packages (from spacy<3.3.0,>=3.2.0->en-c
Requirement already satisfied: cymem<2.1.0,>=2.0.2 in /usr/local/lib/python3.9/dist-packages (from spacy<3.3.0,>=3.2.0->en-co
Requirement already satisfied: wasabi<1.1.0,>=0.8.1 in /usr/local/lib/python3.9/dist-packages (from spacy<3.3.0,>=3.2.0->en-c
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.9/dist-packages (from spacy<3.3.0,>=3.2.0->en-core-w
Requirement already satisfied: smart-open<7.0.0,>=5.2.1 in /usr/local/lib/python3.9/dist-packages (from pathy>=0.3.5->spacy<3
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.9/dist-packages (from pydantic!=1.8,!=1.8
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.9/dist-packages (from requests<3.0.0,>=2.13.0->sp
Requirement already satisfied: charset-normalizer~=2.0.0 in /usr/local/lib/python3.9/dist-packages (from requests<3.0.0,>=2.1
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /usr/local/lib/python3.9/dist-packages (from requests<3.0.0,>=2.13.0-
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.9/dist-packages (from requests<3.0.0,>=2.13.0->spacy<3.
Requirement already satisfied: click<9.0.0,>=7.1.1 in /usr/local/lib/python3.9/dist-packages (from typer<0.5.0,>=0.3.0->spacy
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.9/dist-packages (from jinja2->spacy<3.3.0,>=3.2.0->e
Installing collected packages: en-core-web-sm
  Attempting uninstall: en-core-web-sm
    Found existing installation: en-core-web-sm 3.5.0
    Uninstalling en-core-web-sm-3.5.0:
      Successfully uninstalled en-core-web-sm-3.5.0
Successfully installed en-core-web-sm-3.2.0
✔ Download and installation successful
You can now load the package via spacy.load('en_core_web_sm')
```

```
import os
from os.path import exists
import torch
import torch.nn as nn
from torch.nn.functional import log_softmax, pad
import math
import copy
import time
from torch.optim.lr_scheduler import LambdaLR
import pandas as pd
import altair as alt
from torchtext.data.functional import to_map_style_dataset
from torch.utils.data import DataLoader
from torchtext.vocab import build_vocab_from_iterator
import torchtext.datasets as datasets
```

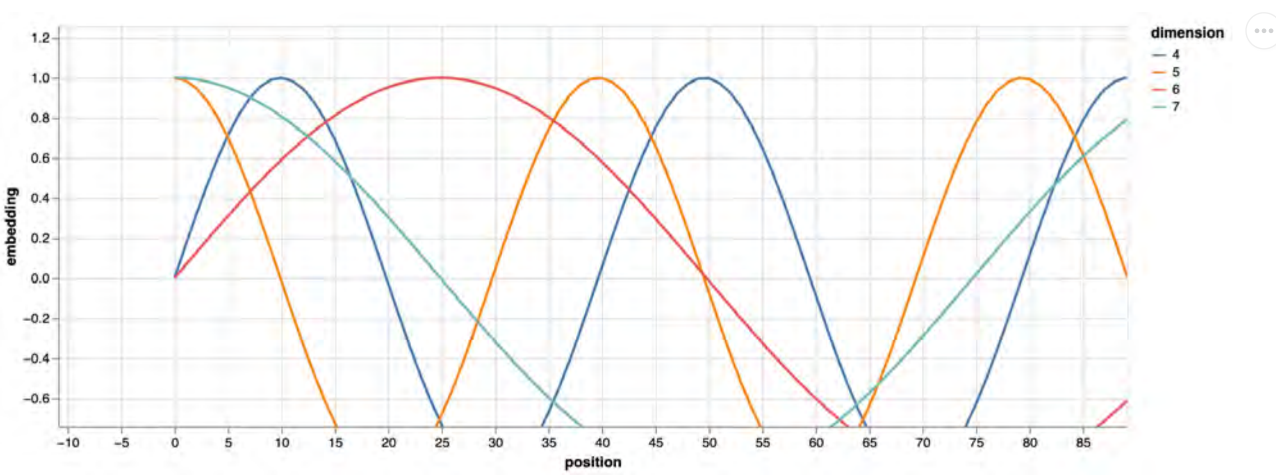
```python
import spacy
import GPUtil
import warnings
from torch.utils.data.distributed import DistributedSampler
import torch.distributed as dist
import torch.multiprocessing as mp
from torch.nn.parallel import DistributedDataParallel as DDP


# Set to False to skip notebook execution (e.g. for debugging)
warnings.filterwarnings("ignore")
RUN_EXAMPLES = True



# Some convenience helper functions used throughout the notebook


def is_interactive_notebook():
    return __name__ == "__main__"


def show_example(fn, args=[]):
    if __name__ == "__main__" and RUN_EXAMPLES:
        return fn(*args)


def execute_example(fn, args=[]):
    if __name__ == "__main__" and RUN_EXAMPLES:
        fn(*args)


class DummyOptimizer(torch.optim.Optimizer):
    def __init__(self):
        self.param_groups = [{"lr": 0}]
        None

    def step(self):
        None

    def zero_grad(self, set_to_none=False):
        None


class DummyScheduler:
    def step(self):
        None


class EncoderDecoder(nn.Module):
    """
    A standard Encoder-Decoder architecture. Base for this and many
    other models.
    """

    def __init__(self, encoder, decoder, src_embed, tgt_embed, generator):
        super(EncoderDecoder, self).__init__()
        self.encoder = encoder
        self.decoder = decoder
        self.src_embed = src_embed
        self.tgt_embed = tgt_embed
        self.generator = generator

    def forward(self, src, tgt, src_mask, tgt_mask):
        "Take in and process masked src and target sequences."
        return self.decode(self.encode(src, src_mask), src_mask, tgt, tgt_mask)

    def encode(self, src, src_mask):
        return self.encoder(self.src_embed(src), src_mask)

    def decode(self, memory, src_mask, tgt, tgt_mask):
        return self.decoder(self.tgt_embed(tgt), memory, src_mask, tgt_mask)


class Generator(nn.Module):
    "Define standard linear + softmax generation step."

    def __init__(self, d_model, vocab):
```

```
        super(Generator, self).__init__()
        self.proj = nn.Linear(d_model, vocab)

    def forward(self, x):
        return log_softmax(self.proj(x), dim=-1)


def clones(module, N):
    "Produce N identical layers."
    return nn.ModuleList([copy.deepcopy(module) for _ in range(N)])


class Encoder(nn.Module):
    "Core encoder is a stack of N layers"

    def __init__(self, layer, N):
        super(Encoder, self).__init__()
        self.layers = clones(layer, N)
        self.norm = LayerNorm(layer.size)

    def forward(self, x, mask):
        "Pass the input (and mask) through each layer in turn."
        for layer in self.layers:
            x = layer(x, mask)
        return self.norm(x)


class LayerNorm(nn.Module):
    "Construct a layernorm module (See citation for details)."

    def __init__(self, features, eps=1e-6):
        super(LayerNorm, self).__init__()
        self.a_2 = nn.Parameter(torch.ones(features))
        self.b_2 = nn.Parameter(torch.zeros(features))
        self.eps = eps

    def forward(self, x):
        mean = x.mean(-1, keepdim=True)
        std = x.std(-1, keepdim=True)
        return self.a_2 * (x - mean) / (std + self.eps) + self.b_2


class SublayerConnection(nn.Module):
    """
    A residual connection followed by a layer norm.
    Note for code simplicity the norm is first as opposed to last.
    """

    def __init__(self, size, dropout):
        super(SublayerConnection, self).__init__()
        self.norm = LayerNorm(size)
        self.dropout = nn.Dropout(dropout)

    def forward(self, x, sublayer):
        "Apply residual connection to any sublayer with the same size."
        return x + self.dropout(sublayer(self.norm(x)))


class EncoderLayer(nn.Module):
    "Encoder is made up of self-attn and feed forward (defined below)"

    def __init__(self, size, self_attn, feed_forward, dropout):
        super(EncoderLayer, self).__init__()
        self.self_attn = self_attn
        self.feed_forward = feed_forward
        self.sublayer = clones(SublayerConnection(size, dropout), 2)
        self.size = size

    def forward(self, x, mask):
        "Follow Figure 1 (left) for connections."
        x = self.sublayer[0](x, lambda x: self.self_attn(x, x, x, mask))
        return self.sublayer[1](x, self.feed_forward)


class Decoder(nn.Module):
    "Generic N layer decoder with masking."

    def __init__(self, layer, N):
        super(Decoder, self).__init__()
```

```
        self.layers = clones(layer, N)
        self.norm = LayerNorm(layer.size)

    def forward(self, x, memory, src_mask, tgt_mask):
        for layer in self.layers:
            x = layer(x, memory, src_mask, tgt_mask)
        return self.norm(x)


class DecoderLayer(nn.Module):
    "Decoder is made of self-attn, src-attn, and feed forward (defined below)"

    def __init__(self, size, self_attn, src_attn, feed_forward, dropout):
        super(DecoderLayer, self).__init__()
        self.size = size
        self.self_attn = self_attn
        self.src_attn = src_attn
        self.feed_forward = feed_forward
        self.sublayer = clones(SublayerConnection(size, dropout), 3)

    def forward(self, x, memory, src_mask, tgt_mask):
        "Follow Figure 1 (right) for connections."
        m = memory
        x = self.sublayer[0](x, lambda x: self.self_attn(x, x, x, tgt_mask))
        x = self.sublayer[1](x, lambda x: self.src_attn(x, m, m, src_mask))
        return self.sublayer[2](x, self.feed_forward)


def subsequent_mask(size):
    "Mask out subsequent positions."
    attn_shape = (1, size, size)
    subsequent_mask = torch.triu(torch.ones(attn_shape), diagonal=1).type(
        torch.uint8
    )
    return subsequent_mask == 0


def example_mask():
    LS_data = pd.concat(
        [
            pd.DataFrame(
                {
                    "Subsequent Mask": subsequent_mask(20)[0][x, y].flatten(),
                    "Window": y,
                    "Masking": x,
                }
            )
            for y in range(20)
            for x in range(20)
        ]
    )

    return (
        alt.Chart(LS_data)
        .mark_rect()
        .properties(height=250, width=250)
        .encode(
            alt.X("Window:O"),
            alt.Y("Masking:O"),
            alt.Color("Subsequent Mask:Q", scale=alt.Scale(scheme="viridis")),
        )
        .interactive()
    )


show_example(example_mask)
```


```
def attention(query, key, value, mask=None, dropout=None):
    "Compute 'Scaled Dot Product Attention'"
    d_k = query.size(-1)
    scores = torch.matmul(query, key.transpose(-2, -1)) / math.sqrt(d_k)
    if mask is not None:
        scores = scores.masked_fill(mask == 0, -1e9)
    p_attn = scores.softmax(dim=-1)
    if dropout is not None:
        p_attn = dropout(p_attn)
    return torch.matmul(p_attn, value), p_attn


class MultiHeadedAttention(nn.Module):
    def __init__(self, h, d_model, dropout=0.1):
        "Take in model size and number of heads."
        super(MultiHeadedAttention, self).__init__()
        assert d_model % h == 0
        # We assume d_v always equals d_k
        self.d_k = d_model // h
        self.h = h
        self.linears = clones(nn.Linear(d_model, d_model), 4)
        self.attn = None
        self.dropout = nn.Dropout(p=dropout)

    def forward(self, query, key, value, mask=None):
        "Implements Figure 2"
        if mask is not None:
            # Same mask applied to all h heads.
            mask = mask.unsqueeze(1)
        nbatches = query.size(0)

        # 1) Do all the linear projections in batch from d_model => h x d_k
        query, key, value = [
            lin(x).view(nbatches, -1, self.h, self.d_k).transpose(1, 2)
            for lin, x in zip(self.linears, (query, key, value))
        ]

        # 2) Apply attention on all the projected vectors in batch.
        x, self.attn = attention(
            query, key, value, mask=mask, dropout=self.dropout
        )

        # 3) "Concat" using a view and apply a final linear.
        x = (
            x.transpose(1, 2)
            .contiguous()
            .view(nbatches, -1, self.h * self.d_k)
        )
        del query
        del key
        del value
        return self.linears[-1](x)


class PositionwiseFeedForward(nn.Module):
    "Implements FFN equation."

    def __init__(self, d_model, d_ff, dropout=0.1):
        super(PositionwiseFeedForward, self).__init__()
        self.w_1 = nn.Linear(d_model, d_ff)
        self.w_2 = nn.Linear(d_ff, d_model)
        self.dropout = nn.Dropout(dropout)

    def forward(self, x):
        return self.w_2(self.dropout(self.w_1(x).relu()))


class Embeddings(nn.Module):
    def __init__(self, d_model, vocab):
        super(Embeddings, self).__init__()
        self.lut = nn.Embedding(vocab, d_model)
        self.d_model = d_model
```

```
    def forward(self, x):
        return self.lut(x) * math.sqrt(self.d_model)


class PositionalEncoding(nn.Module):
    "Implement the PE function."

    def __init__(self, d_model, dropout, max_len=5000):
        super(PositionalEncoding, self).__init__()
        self.dropout = nn.Dropout(p=dropout)

        # Compute the positional encodings once in log space.
        pe = torch.zeros(max_len, d_model)
        position = torch.arange(0, max_len).unsqueeze(1)
        div_term = torch.exp(
            torch.arange(0, d_model, 2) * -(math.log(10000.0) / d_model)
        )
        pe[:, 0::2] = torch.sin(position * div_term)
        pe[:, 1::2] = torch.cos(position * div_term)
        pe = pe.unsqueeze(0)
        self.register_buffer("pe", pe)

    def forward(self, x):
        x = x + self.pe[:, : x.size(1)].requires_grad_(False)
        return self.dropout(x)


def example_positional():
    pe = PositionalEncoding(20, 0)
    y = pe.forward(torch.zeros(1, 100, 20))

    data = pd.concat(
        [
            pd.DataFrame(
                {
                    "embedding": y[0, :, dim],
                    "dimension": dim,
                    "position": list(range(100)),
                }
            )
            for dim in [4, 5, 6, 7]
        ]
    )

    return (
        alt.Chart(data)
        .mark_line()
        .properties(width=800)
        .encode(x="position", y="embedding", color="dimension:N")
        .interactive()
    )


show_example(example_positional)
```



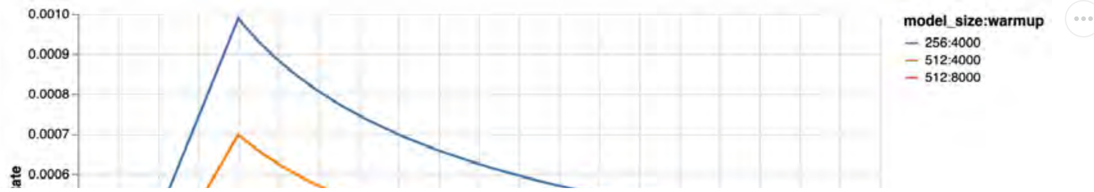
```
def make_model(
    src_vocab, tgt_vocab, N=6, d_model=512, d_ff=2048, h=8, dropout=0.1
):
    "Helper: Construct a model from hyperparameters."
    c = copy.deepcopy
    attn = MultiHeadedAttention(h, d_model)
    ff = PositionwiseFeedForward(d_model, d_ff, dropout)
    position = PositionalEncoding(d_model, dropout)
    model = EncoderDecoder(
        Encoder(EncoderLayer(d_model, c(attn), c(ff), dropout), N),
        Decoder(DecoderLayer(d_model, c(attn), c(attn), c(ff), dropout), N),
        nn.Sequential(Embeddings(d_model, src_vocab), c(position)),
        nn.Sequential(Embeddings(d_model, tgt_vocab), c(position)),
        Generator(d_model, tgt_vocab),
    )

    # This was important from their code.
    # Initialize parameters with Glorot / fan_avg.
    for p in model.parameters():
        if p.dim() > 1:
            nn.init.xavier_uniform_(p)
    return model


def·inference_test():
····test_model·=·make_model(11,·11,·2)
····test_model.eval()
····src·=·torch.LongTensor([[1,·2,·3,·4,·5,·6,·7,·8,·9,·10]])
····src_mask·=·torch.ones(1,·1,·10)

····memory·=·test_model.encode(src,·src_mask)
····ys·=·torch.zeros(1,·1).type_as(src)

····for·i·in·range(9):
········out·=·test_model.decode(
············memory,·src_mask,·ys,·subsequent_mask(ys.size(1)).type_as(src.data)
········)
········prob·=·test_model.generator(out[:,·-1])
········_,·next_word·=·torch.max(prob,·dim=1)
········next_word·=·next_word.data[0]
········ys·=·torch.cat(
············[ys,·torch.empty(1,·1).type_as(src.data).fill_(next_word)],·dim=1
········)

····print("Example·Untrained·Model·Prediction:",·ys)


def·run_tests():
····for·_·in·range(10):
········inference_test()


show_example(run_tests)
```

```
Example Untrained Model Prediction: tensor([[0, 8, 9, 9, 9, 9, 9, 9, 9, 9]])
Example Untrained Model Prediction: tensor([[0, 6, 6, 6, 6, 6, 6, 6, 6, 6]])
Example Untrained Model Prediction: tensor([[0, 2, 2, 2, 2, 2, 2, 2, 2, 2]])
Example Untrained Model Prediction: tensor([[0, 6, 0, 6, 0, 6, 0, 6, 3, 2]])
Example Untrained Model Prediction: tensor([[0, 1, 6, 1, 9, 1, 9, 1, 0, 1]])
Example Untrained Model Prediction: tensor([[ 0,  3,  5,  7,  5, 10, 10, 10, 10, 10]])
Example Untrained Model Prediction: tensor([[0, 2, 6, 6, 2, 6, 2, 6, 2, 6]])
Example Untrained Model Prediction: tensor([[0, 7, 4, 1, 6, 4, 1, 6, 4, 1]])
Example Untrained Model Prediction: tensor([[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]])
Example Untrained Model Prediction: tensor([[ 0, 10,  7, 10,  7, 10,  7, 10,  7,  0]])
```

```
class Batch:
    """Object for holding a batch of data with mask during training."""

    def __init__(self, src, tgt=None, pad=2):  # 2 = <blank>
        self.src = src
        self.src_mask = (src != pad).unsqueeze(-2)
        if tgt is not None:
            self.tgt = tgt[:, :-1]
            self.tgt_y = tgt[:, 1:]
            self.tgt_mask = self.make_std_mask(self.tgt, pad)
            self.ntokens = (self.tgt_y != pad).data.sum()

    @staticmethod
```

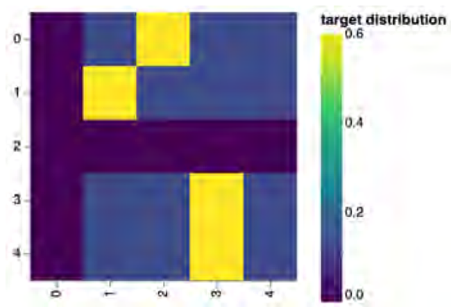
```
    def make_std_mask(tgt, pad):
        "Create a mask to hide padding and future words."
        tgt_mask = (tgt != pad).unsqueeze(-2)
        tgt_mask = tgt_mask & subsequent_mask(tgt.size(-1)).type_as(
            tgt_mask.data
        )
        return tgt_mask



class TrainState:
    """Track number of steps, examples, and tokens processed"""

    step: int = 0  # Steps in the current epoch
    accum_step: int = 0  # Number of gradient accumulation steps
    samples: int = 0  # total # of examples used
    tokens: int = 0  # total # of tokens processed



def run_epoch(
    data_iter,
    model,
    loss_compute,
    optimizer,
    scheduler,
    mode="train",
    accum_iter=1,
    train_state=TrainState(),
):
    """Train a single epoch"""
    start = time.time()
    total_tokens = 0
    total_loss = 0
    tokens = 0
    n_accum = 0
    for i, batch in enumerate(data_iter):
        out = model.forward(
            batch.src, batch.tgt, batch.src_mask, batch.tgt_mask
        )
        loss, loss_node = loss_compute(out, batch.tgt_y, batch.ntokens)
        # loss_node = loss_node / accum_iter
        if mode == "train" or mode == "train+log":
            loss_node.backward()
            train_state.step += 1
            train_state.samples += batch.src.shape[0]
            train_state.tokens += batch.ntokens
            if i % accum_iter == 0:
                optimizer.step()
                optimizer.zero_grad(set_to_none=True)
                n_accum += 1
                train_state.accum_step += 1
            scheduler.step()

        total_loss += loss
        total_tokens += batch.ntokens
        tokens += batch.ntokens
        if i % 40 == 1 and (mode == "train" or mode == "train+log"):
            lr = optimizer.param_groups[0]["lr"]
            elapsed = time.time() - start
            print(
                (
                    "Epoch Step: %6d | Accumulation Step: %3d | Loss: %6.2f "
                    + "| Tokens / Sec: %7.1f | Learning Rate: %6.1e"
                )
                % (i, n_accum, loss / batch.ntokens, tokens / elapsed, lr)
            )
            start = time.time()
            tokens = 0
        del loss
        del loss_node
    return total_loss / total_tokens, train_state


def rate(step, model_size, factor, warmup):
    """
    we have to default the step to 1 for LambdaLR function
    to avoid zero raising to negative power.
```

```
    """
    if step == 0:
        step = 1
    return factor * (
        model_size ** (-0.5) * min(step ** (-0.5), step * warmup ** (-1.5))
    )


def example_learning_schedule():
    opts = [
        [512, 1, 4000],  # example 1
        [512, 1, 8000],  # example 2
        [256, 1, 4000],  # example 3
    ]

    dummy_model = torch.nn.Linear(1, 1)
    learning_rates = []

    # we have 3 examples in opts list.
    for idx, example in enumerate(opts):
        # run 20000 epoch for each example
        optimizer = torch.optim.Adam(
            dummy_model.parameters(), lr=1, betas=(0.9, 0.98), eps=1e-9
        )
        lr_scheduler = LambdaLR(
            optimizer=optimizer, lr_lambda=lambda step: rate(step, *example)
        )
        tmp = []
        # take 20K dummy training steps, save the learning rate at each step
        for step in range(20000):
            tmp.append(optimizer.param_groups[0]["lr"])
            optimizer.step()
            lr_scheduler.step()
        learning_rates.append(tmp)

    learning_rates = torch.tensor(learning_rates)

    # Enable altair to handle more than 5000 rows
    alt.data_transformers.disable_max_rows()

    opts_data = pd.concat(
        [
            pd.DataFrame(
                {
                    "Learning Rate": learning_rates[warmup_idx, :],
                    "model_size:warmup": ["512:4000", "512:8000", "256:4000"][
                        warmup_idx
                    ],
                    "step": range(20000),
                }
            )
            for warmup_idx in [0, 1, 2]
        ]
    )

    return (
        alt.Chart(opts_data)
        .mark_line()
        .properties(width=600)
        .encode(x="step", y="Learning Rate", color="model_size:warmup:N")
        .interactive()
    )


example_learning_schedule()
```

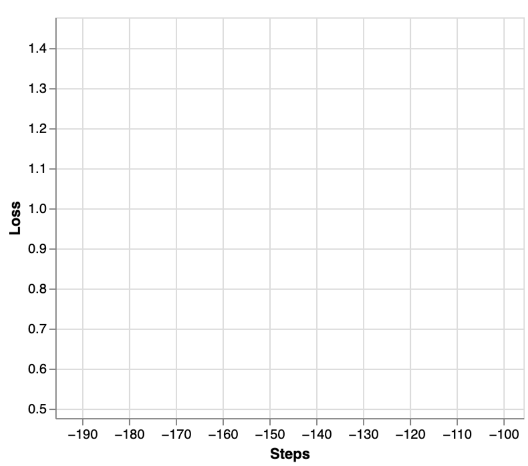



```
class LabelSmoothing(nn.Module):
    "Implement label smoothing."

    def __init__(self, size, padding_idx, smoothing=0.0):
        super(LabelSmoothing, self).__init__()
        self.criterion = nn.KLDivLoss(reduction="sum")
        self.padding_idx = padding_idx
        self.confidence = 1.0 - smoothing
        self.smoothing = smoothing
        self.size = size
        self.true_dist = None

    def forward(self, x, target):
        assert x.size(1) == self.size
        true_dist = x.data.clone()
        true_dist.fill_(self.smoothing / (self.size - 2))
        true_dist.scatter_(1, target.data.unsqueeze(1), self.confidence)
        true_dist[:, self.padding_idx] = 0
        mask = torch.nonzero(target.data == self.padding_idx)
        if mask.dim() > 0:
            true_dist.index_fill_(0, mask.squeeze(), 0.0)
        self.true_dist = true_dist
        return self.criterion(x, true_dist.clone().detach())


# Example of label smoothing.


def example_label_smoothing():
    crit = LabelSmoothing(5, 0, 0.4)
    predict = torch.FloatTensor(
        [
            [0, 0.2, 0.7, 0.1, 0],
            [0, 0.2, 0.7, 0.1, 0],
            [0, 0.2, 0.7, 0.1, 0],
            [0, 0.2, 0.7, 0.1, 0],
            [0, 0.2, 0.7, 0.1, 0],
        ]
    )
    crit(x=predict.log(), target=torch.LongTensor([2, 1, 0, 3, 3]))
    LS_data = pd.concat(
        [
            pd.DataFrame(
                {
                    "target distribution": crit.true_dist[x, y].flatten(),
                    "columns": y,
                    "rows": x,
                }
            )
            for y in range(5)
            for x in range(5)
        ]
    )

    return (
        alt.Chart(LS_data)
        .mark_rect(color="Blue", opacity=1)
        .properties(height=200, width=200)
        .encode(
            alt.X("columns:O", title=None),
            alt.Y("rows:O", title=None),
            alt.Color(
                "target distribution:Q", scale=alt.Scale(scheme="viridis")
            ),
        )
        .interactive()
    )
```

```
show_example(example_label_smoothing)
```




```
def loss(x, crit):
    d = x + 3 * 1
    predict = torch.FloatTensor([[0, x / d, 1 / d, 1 / d, 1 / d]])
    return crit(predict.log(), torch.LongTensor([1])).data


def penalization_visualization():
    crit = LabelSmoothing(5, 0, 0.1)
    loss_data = pd.DataFrame(
        {
            "Loss": [loss(x, crit) for x in range(1, 100)],
            "Steps": list(range(99)),
        }
    ).astype("float")

    return (
        alt.Chart(loss_data)
        .mark_line()
        .properties(width=350)
        .encode(
            x="Steps",
            y="Loss",
        )
        .interactive()
    )


show_example(penalization_visualization)
```




```
def data_gen(V, batch_size, nbatches):
    "Generate random data for a src-tgt copy task."
    for i in range(nbatches):
        data = torch.randint(1, V, size=(batch_size, 10))
        data[:, 0] = 1
        src = data.requires_grad_(False).clone().detach()
        tgt = data.requires_grad_(False).clone().detach()
        yield Batch(src, tgt, 0)
```

```python
class SimpleLossCompute:
    "A simple loss compute and train function."

    def __init__(self, generator, criterion):
        self.generator = generator
        self.criterion = criterion

    def __call__(self, x, y, norm):
        x = self.generator(x)
        sloss = (
            self.criterion(
                x.contiguous().view(-1, x.size(-1)), y.contiguous().view(-1)
            )
            / norm
        )
        return sloss.data * norm, sloss


def greedy_decode(model, src, src_mask, max_len, start_symbol):
    memory = model.encode(src, src_mask)
    ys = torch.zeros(1, 1).fill_(start_symbol).type_as(src.data)
    for i in range(max_len - 1):
        out = model.decode(
            memory, src_mask, ys, subsequent_mask(ys.size(1)).type_as(src.data)
        )
        prob = model.generator(out[:, -1])
        _, next_word = torch.max(prob, dim=1)
        next_word = next_word.data[0]
        ys = torch.cat(
            [ys, torch.zeros(1, 1).type_as(src.data).fill_(next_word)], dim=1
        )
    return ys


# Train the simple copy task.


def example_simple_model():
    V = 11
    criterion = LabelSmoothing(size=V, padding_idx=0, smoothing=0.0)
    model = make_model(V, V, N=2)

    optimizer = torch.optim.Adam(
        model.parameters(), lr=0.5, betas=(0.9, 0.98), eps=1e-9
    )
    lr_scheduler = LambdaLR(
        optimizer=optimizer,
        lr_lambda=lambda step: rate(
            step, model_size=model.src_embed[0].d_model, factor=1.0, warmup=400
        ),
    )

    batch_size = 80
    for epoch in range(20):
        model.train()
        run_epoch(
            data_gen(V, batch_size, 20),
            model,
            SimpleLossCompute(model.generator, criterion),
            optimizer,
            lr_scheduler,
            mode="train",
        )
        model.eval()
        run_epoch(
            data_gen(V, batch_size, 5),
            model,
            SimpleLossCompute(model.generator, criterion),
            DummyOptimizer(),
            DummyScheduler(),
            mode="eval",
        )[0]

    model.eval()
    src = torch.LongTensor([[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]])
    max_len = src.shape[1]
    src_mask = torch.ones(1, 1, max_len)
```

```
    print(greedy_decode(model, src, src_mask, max_len=max_len, start_symbol=0))


# execute_example(example_simple_model)


# Load spacy tokenizer models, download them if they haven't been
# downloaded already


def load_tokenizers():

    try:
        spacy_de = spacy.load("de_core_news_sm")
    except IOError:
        os.system("python -m spacy download de_core_news_sm")
        spacy_de = spacy.load("de_core_news_sm")

    try:
        spacy_en = spacy.load("en_core_web_sm")
    except IOError:
        os.system("python -m spacy download en_core_web_sm")
        spacy_en = spacy.load("en_core_web_sm")

    return spacy_de, spacy_en


def tokenize(text, tokenizer):
    return [tok.text for tok in tokenizer.tokenizer(text)]


def yield_tokens(data_iter, tokenizer, index):
    for from_to_tuple in data_iter:
        yield tokenizer(from_to_tuple[index])




def build_vocabulary(spacy_de, spacy_en):
    def tokenize_de(text):
        return tokenize(text, spacy_de)

    def tokenize_en(text):
        return tokenize(text, spacy_en)

    print("Building German Vocabulary ...")
    train, val, test = datasets.Multi30k(language_pair=("de", "en"))
    vocab_src = build_vocab_from_iterator(
        yield_tokens(train + val + test, tokenize_de, index=0),
        min_freq=2,
        specials=["<s>", "</s>", "<blank>", "<unk>"],
    )

    print("Building English Vocabulary ...")
    train, val, test = datasets.Multi30k(language_pair=("de", "en"))
    vocab_tgt = build_vocab_from_iterator(
        yield_tokens(train + val + test, tokenize_en, index=1),
        min_freq=2,
        specials=["<s>", "</s>", "<blank>", "<unk>"],
    )

    vocab_src.set_default_index(vocab_src["<unk>"])
    vocab_tgt.set_default_index(vocab_tgt["<unk>"])

    return vocab_src, vocab_tgt


def load_vocab(spacy_de, spacy_en):
    if not exists("vocab.pt"):
        vocab_src, vocab_tgt = build_vocabulary(spacy_de, spacy_en)
        torch.save((vocab_src, vocab_tgt), "vocab.pt")
    else:
        vocab_src, vocab_tgt = torch.load("vocab.pt")
    print("Finished.\nVocabulary sizes:")
    print(len(vocab_src))
    print(len(vocab_tgt))
    return vocab_src, vocab_tgt
```

```
if is_interactive_notebook():
    # global variables used later in the script
    spacy_de, spacy_en = show_example(load_tokenizers)
    vocab_src, vocab_tgt = show_example(load_vocab, args=[spacy_de, spacy_en])
```

```
Building German Vocabulary ...
Building English Vocabulary ...
Finished.
Vocabulary sizes:
8315
6384
```

```
def collate_batch(
    batch,
    src_pipeline,
    tgt_pipeline,
    src_vocab,
    tgt_vocab,
    device,
    max_padding=128,
    pad_id=2,
):
    bs_id = torch.tensor([0], device=device)  # <s> token id
    eos_id = torch.tensor([1], device=device)  # </s> token id
    src_list, tgt_list = [], []
    for (_src, _tgt) in batch:
        processed_src = torch.cat(
            [
                bs_id,
                torch.tensor(
                    src_vocab(src_pipeline(_src)),
                    dtype=torch.int64,
                    device=device,
                ),
                eos_id,
            ],
            0,
        )
        processed_tgt = torch.cat(
            [
                bs_id,
                torch.tensor(
                    tgt_vocab(tgt_pipeline(_tgt)),
                    dtype=torch.int64,
                    device=device,
                ),
                eos_id,
            ],
            0,
        )
        src_list.append(
            # warning - overwrites values for negative values of padding - len
            pad(
                processed_src,
                (
                    0,
                    max_padding - len(processed_src),
                ),
                value=pad_id,
            )
        )
        tgt_list.append(
            pad(
                processed_tgt,
                (0, max_padding - len(processed_tgt)),
                value=pad_id,
            )
        )

    src = torch.stack(src_list)
    tgt = torch.stack(tgt_list)
    return (src, tgt)


def create_dataloaders(
    device,
```

```
    vocab_src,
    vocab_tgt,
    spacy_de,
    spacy_en,
    batch_size=12000,
    max_padding=128,
    is_distributed=True,
):
    # def create_dataloaders(batch_size=12000):
    def tokenize_de(text):
        return tokenize(text, spacy_de)

    def tokenize_en(text):
        return tokenize(text, spacy_en)

    def collate_fn(batch):
        return collate_batch(
            batch,
            tokenize_de,
            tokenize_en,
            vocab_src,
            vocab_tgt,
            device,
            max_padding=max_padding,
            pad_id=vocab_src.get_stoi()["<blank>"],
        )

    train_iter, valid_iter, test_iter = datasets.Multi30k(
        language_pair=("de", "en")
    )

    train_iter_map = to_map_style_dataset(
        train_iter
    )  # DistributedSampler needs a dataset len()
    train_sampler = (
        DistributedSampler(train_iter_map) if is_distributed else None
    )
    valid_iter_map = to_map_style_dataset(valid_iter)
    valid_sampler = (
        DistributedSampler(valid_iter_map) if is_distributed else None
    )

    train_dataloader = DataLoader(
        train_iter_map,
        batch_size=batch_size,
        shuffle=(train_sampler is None),
        sampler=train_sampler,
        collate_fn=collate_fn,
    )
    valid_dataloader = DataLoader(
        valid_iter_map,
        batch_size=batch_size,
        shuffle=(valid_sampler is None),
        sampler=valid_sampler,
        collate_fn=collate_fn,
    )
    return train_dataloader, valid_dataloader


def train_worker(
    gpu,
    ngpus_per_node,
    vocab_src,
    vocab_tgt,
    spacy_de,
    spacy_en,
    config,
    is_distributed=False,
):
    print(f"Train worker process using GPU: {gpu} for training", flush=True)
    torch.cuda.set_device(gpu)

    pad_idx = vocab_tgt["<blank>"]
    d_model = 512
    model = make_model(len(vocab_src), len(vocab_tgt), N=6)
    model.cuda(gpu)
    module = model
```

```
    is_main_process = True
    if is_distributed:
        dist.init_process_group(
            "nccl", init_method="env://", rank=gpu, world_size=ngpus_per_node
        )
        model = DDP(model, device_ids=[gpu])
        module = model.module
        is_main_process = gpu == 0

    criterion = LabelSmoothing(
        size=len(vocab_tgt), padding_idx=pad_idx, smoothing=0.1
    )
    criterion.cuda(gpu)

    train_dataloader, valid_dataloader = create_dataloaders(
        gpu,
        vocab_src,
        vocab_tgt,
        spacy_de,
        spacy_en,
        batch_size=config["batch_size"] // ngpus_per_node,
        max_padding=config["max_padding"],
        is_distributed=is_distributed,
    )

    optimizer = torch.optim.Adam(
        model.parameters(), lr=config["base_lr"], betas=(0.9, 0.98), eps=1e-9
    )
    lr_scheduler = LambdaLR(
        optimizer=optimizer,
        lr_lambda=lambda step: rate(
            step, d_model, factor=1, warmup=config["warmup"]
        ),
    )
    train_state = TrainState()

    for epoch in range(config["num_epochs"]):
        if is_distributed:
            train_dataloader.sampler.set_epoch(epoch)
            valid_dataloader.sampler.set_epoch(epoch)

        model.train()
        print(f"[GPU{gpu}] Epoch {epoch} Training ====", flush=True)
        _, train_state = run_epoch(
            (Batch(b[0], b[1], pad_idx) for b in train_dataloader),
            model,
            SimpleLossCompute(module.generator, criterion),
            optimizer,
            lr_scheduler,
            mode="train+log",
            accum_iter=config["accum_iter"],
            train_state=train_state,
        )

        GPUtil.showUtilization()
        if is_main_process:
            file_path = "%s%.2d.pt" % (config["file_prefix"], epoch)
            torch.save(module.state_dict(), file_path)
        torch.cuda.empty_cache()

        print(f"[GPU{gpu}] Epoch {epoch} Validation ====", flush=True)
        model.eval()
        sloss = run_epoch(
            (Batch(b[0], b[1], pad_idx) for b in valid_dataloader),
            model,
            SimpleLossCompute(module.generator, criterion),
            DummyOptimizer(),
            DummyScheduler(),
            mode="eval",
        )
        print(sloss)
        torch.cuda.empty_cache()

    if is_main_process:
        file_path = "%sfinal.pt" % config["file_prefix"]
        torch.save(module.state_dict(), file_path)
```

```
#@title
def train_distributed_model(vocab_src, vocab_tgt, spacy_de, spacy_en, config):
    from the_annotated_transformer import train_worker

    ngpus = torch.cuda.device_count()
    os.environ["MASTER_ADDR"] = "localhost"
    os.environ["MASTER_PORT"] = "12356"
    print(f"Number of GPUs detected: {ngpus}")
    print("Spawning training processes ...")
    mp.spawn(
        train_worker,
        nprocs=ngpus,
        args=(ngpus, vocab_src, vocab_tgt, spacy_de, spacy_en, config, True),
    )


def train_model(vocab_src, vocab_tgt, spacy_de, spacy_en, config):
    if config["distributed"]:
        train_distributed_model(
            vocab_src, vocab_tgt, spacy_de, spacy_en, config
        )
    else:
        train_worker(
            0, 1, vocab_src, vocab_tgt, spacy_de, spacy_en, config, False
        )


def load_trained_model():
    config = {
        "batch_size": 32,
        "distributed": False,
        "num_epochs": 8,
        "accum_iter": 10,
        "base_lr": 1.0,
        "max_padding": 72,
        "warmup": 3000,
        "file_prefix": "multi30k_model_",
    }
    model_path = "multi30k_model_final.pt"
    if not exists(model_path):
        train_model(vocab_src, vocab_tgt, spacy_de, spacy_en, config)

    model = make_model(len(vocab_src), len(vocab_tgt), N=6)
    model.load_state_dict(torch.load("multi30k_model_final.pt"))
    return model


if is_interactive_notebook():
    model = load_trained_model()
```

```
[GPU0] Epoch 7 Training ====
Epoch Step:      1 | Accumulation Step:   1 | Loss:   1.14 | Tokens / Sec:  1999.9 | Learning Rate: 5.5e-04
Epoch Step:     41 | Accumulation Step:   5 | Loss:   1.13 | Tokens / Sec:  1656.8 | Learning Rate: 5.5e-04
Epoch Step:     81 | Accumulation Step:   9 | Loss:   1.04 | Tokens / Sec:  1660.9 | Learning Rate: 5.5e-04
Epoch Step:    121 | Accumulation Step:  13 | Loss:   0.98 | Tokens / Sec:  1695.5 | Learning Rate: 5.5e-04
Epoch Step:    161 | Accumulation Step:  17 | Loss:   0.99 | Tokens / Sec:  1669.0 | Learning Rate: 5.5e-04
Epoch Step:    201 | Accumulation Step:  21 | Loss:   1.01 | Tokens / Sec:  1666.6 | Learning Rate: 5.5e-04
Epoch Step:    241 | Accumulation Step:  25 | Loss:   1.08 | Tokens / Sec:  1675.9 | Learning Rate: 5.4e-04
Epoch Step:    281 | Accumulation Step:  29 | Loss:   0.82 | Tokens / Sec:  1648.3 | Learning Rate: 5.4e-04
Epoch Step:    321 | Accumulation Step:  33 | Loss:   1.12 | Tokens / Sec:  1666.6 | Learning Rate: 5.4e-04
Epoch Step:    361 | Accumulation Step:  37 | Loss:   0.93 | Tokens / Sec:  1676.4 | Learning Rate: 5.4e-04
Epoch Step:    401 | Accumulation Step:  41 | Loss:   1.07 | Tokens / Sec:  1676.0 | Learning Rate: 5.4e-04
Epoch Step:    441 | Accumulation Step:  45 | Loss:   1.10 | Tokens / Sec:  1653.9 | Learning Rate: 5.4e-04
Epoch Step:    481 | Accumulation Step:  49 | Loss:   1.21 | Tokens / Sec:  1675.2 | Learning Rate: 5.3e-04
Epoch Step:    521 | Accumulation Step:  53 | Loss:   0.96 | Tokens / Sec:  1674.4 | Learning Rate: 5.3e-04
Epoch Step:    561 | Accumulation Step:  57 | Loss:   1.10 | Tokens / Sec:  1668.3 | Learning Rate: 5.3e-04
Epoch Step:    601 | Accumulation Step:  61 | Loss:   1.14 | Tokens / Sec:  1678.2 | Learning Rate: 5.3e-04
Epoch Step:    641 | Accumulation Step:  65 | Loss:   0.99 | Tokens / Sec:  1675.3 | Learning Rate: 5.3e-04
Epoch Step:    681 | Accumulation Step:  69 | Loss:   1.07 | Tokens / Sec:  1666.3 | Learning Rate: 5.3e-04
Epoch Step:    721 | Accumulation Step:  73 | Loss:   1.17 | Tokens / Sec:  1681.1 | Learning Rate: 5.3e-04
Epoch Step:    761 | Accumulation Step:  77 | Loss:   1.16 | Tokens / Sec:  1657.9 | Learning Rate: 5.2e-04
Epoch Step:    801 | Accumulation Step:  81 | Loss:   1.19 | Tokens / Sec:  1682.5 | Learning Rate: 5.2e-04
Epoch Step:    841 | Accumulation Step:  85 | Loss:   1.18 | Tokens / Sec:  1698.1 | Learning Rate: 5.2e-04
Epoch Step:    881 | Accumulation Step:  89 | Loss:   1.05 | Tokens / Sec:  1657.0 | Learning Rate: 5.2e-04
| ID | GPU | MEM |
------------------
|  0 | 93% | 32% |
[GPU0] Epoch 7 Validation ====
(tensor(1.4417, device='cuda:0'), <__main__.TrainState object at 0x7fd3af1372b0>)
```

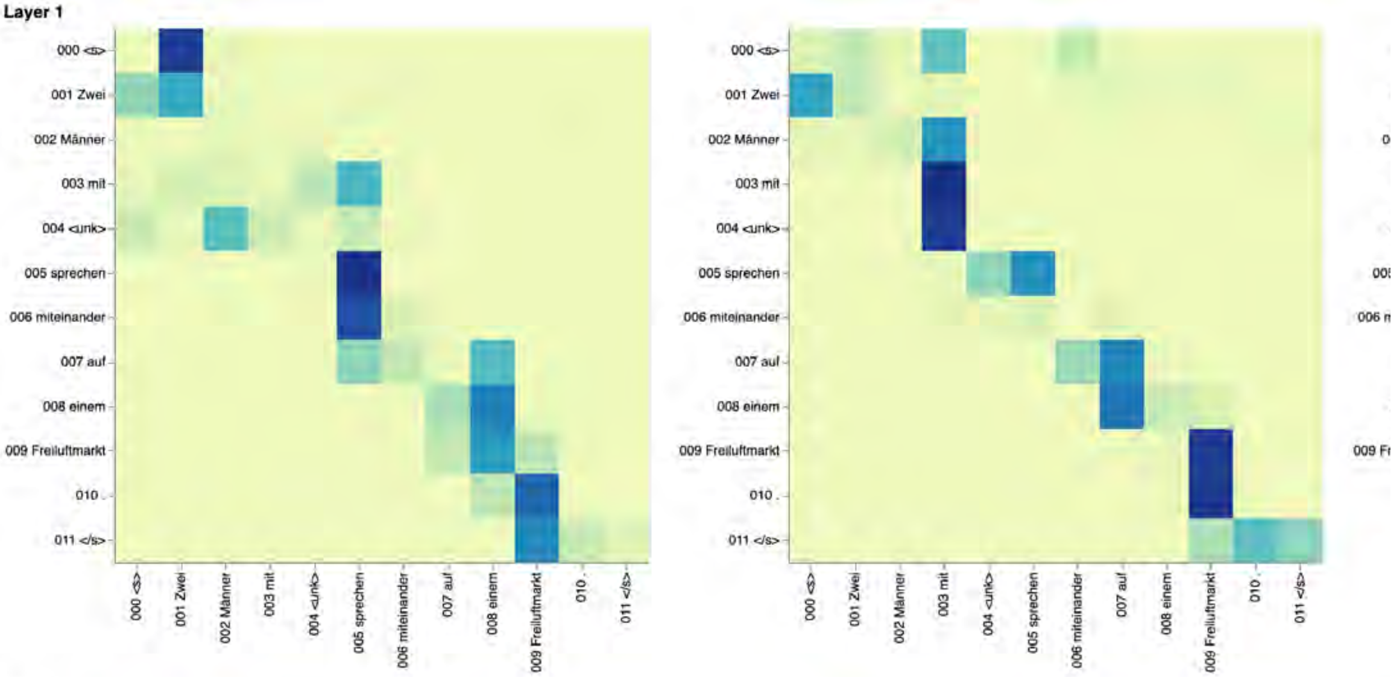
```python
#lets get into the results all down here
def check_outputs(
    valid_dataloader,
    model,
    vocab_src,
    vocab_tgt,
    n_examples=15,
    pad_idx=2,
    eos_string="</s>",
):
    results = [()] * n_examples
    for idx in range(n_examples):
        print("\nExample %d ========\n" % idx)
        b = next(iter(valid_dataloader))
        rb = Batch(b[0], b[1], pad_idx)
        greedy_decode(model, rb.src, rb.src_mask, 64, 0)[0]

        src_tokens = [
            vocab_src.get_itos()[x] for x in rb.src[0] if x != pad_idx
        ]
        tgt_tokens = [
            vocab_tgt.get_itos()[x] for x in rb.tgt[0] if x != pad_idx
        ]

        print(
            "Source Text (Input)        : "
            + " ".join(src_tokens).replace("\n", "")
        )
        print(
            "Target Text (Ground Truth) : "
            + " ".join(tgt_tokens).replace("\n", "")
        )
        model_out = greedy_decode(model, rb.src, rb.src_mask, 72, 0)[0]
        model_txt = (
            " ".join(
                [vocab_tgt.get_itos()[x] for x in model_out if x != pad_idx]
            ).split(eos_string, 1)[0]
            + eos_string
        )
        print("Model Output               : " + model_txt.replace("\n", ""))
        results[idx] = (rb, src_tokens, tgt_tokens, model_out, model_txt)
    return results


def run_model_example(n_examples=5):
    global vocab_src, vocab_tgt, spacy_de, spacy_en

    print("Preparing Data ...")
    _, valid_dataloader = create_dataloaders(
        torch.device("cpu"),
```

```
        vocab_src,
        vocab_tgt,
        spacy_de,
        spacy_en,
        batch_size=1,
        is_distributed=False,
    )

    print("Loading Trained Model ...")

    model = make_model(len(vocab_src), len(vocab_tgt), N=6)
    model.load_state_dict(
        torch.load("multi30k_model_final.pt", map_location=torch.device("cpu"))
    )

    print("Checking Model Outputs:")
    example_data = check_outputs(
        valid_dataloader, model, vocab_src, vocab_tgt, n_examples=n_examples
    )
    return model, example_data


# execute_example(run_model_example)


def mtx2df(m, max_row, max_col, row_tokens, col_tokens):
    "convert a dense matrix to a data frame with row and column indices"
    return pd.DataFrame(
        [
            (
                r,
                c,
                float(m[r, c]),
                "%.3d %s"
                % (r, row_tokens[r] if len(row_tokens) > r else "<blank>"),
                "%.3d %s"
                % (c, col_tokens[c] if len(col_tokens) > c else "<blank>"),
            )
            for r in range(m.shape[0])
            for c in range(m.shape[1])
            if r < max_row and c < max_col
        ],
        # if float(m[r,c]) != 0 and r < max_row and c < max_col],
        columns=["row", "column", "value", "row_token", "col_token"],
    )


def attn_map(attn, layer, head, row_tokens, col_tokens, max_dim=30):
    df = mtx2df(
        attn[0, head].data,
        max_dim,
        max_dim,
        row_tokens,
        col_tokens,
    )
    return (
        alt.Chart(data=df)
        .mark_rect()
        .encode(
            x=alt.X("col_token", axis=alt.Axis(title="")),
            y=alt.Y("row_token", axis=alt.Axis(title="")),
            color="value",
            tooltip=["row", "column", "value", "row_token", "col_token"],
        )
        .properties(height=400, width=400)
        .interactive()
    )


def get_encoder(model, layer):
    return model.encoder.layers[layer].self_attn.attn


def get_decoder_self(model, layer):
    return model.decoder.layers[layer].self_attn.attn


def get_decoder_src(model, layer):
```

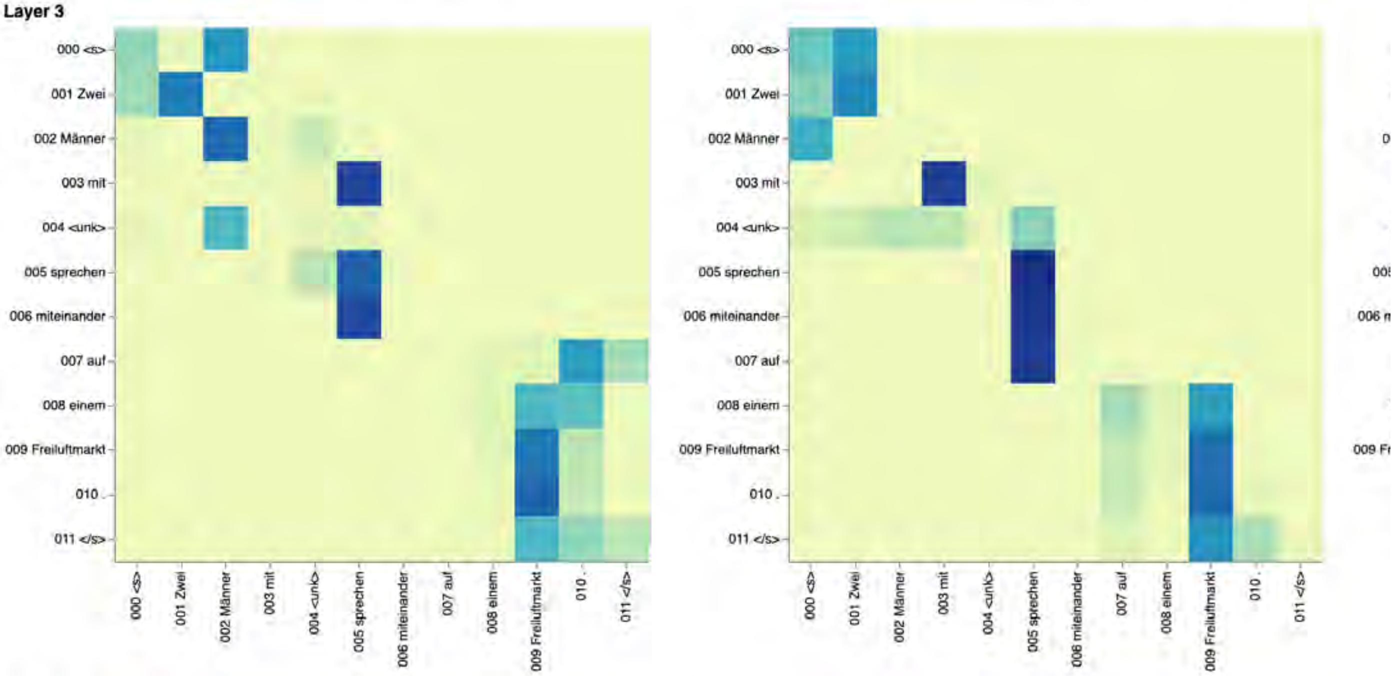
```
    return model.decoder.layers[layer].src_attn.attn


def visualize_layer(model, layer, getter_fn, ntokens, row_tokens, col_tokens):
    # ntokens = last_example[0].ntokens
    attn = getter_fn(model, layer)
    n_heads = attn.shape[1]
    charts = [
        attn_map(
            attn,
            0,
            h,
            row_tokens=row_tokens,
            col_tokens=col_tokens,
            max_dim=ntokens,
        )
        for h in range(n_heads)
    ]
    assert n_heads == 8
    return alt.vconcat(
        charts[0]
        # | charts[1]
        | charts[2]
        # | charts[3]
        | charts[4]
        # | charts[5]
        | charts[6]
        # | charts[7]
        # layer + 1 due to 0-indexing
    ).properties(title="Layer %d" % (layer + 1))


def viz_encoder_self():
    model, example_data = run_model_example(n_examples=1)
    example = example_data[
        len(example_data) - 1
    ]  # batch object for the final example

    layer_viz = [
        visualize_layer(
            model, layer, get_encoder, len(example[1]), example[1], example[1]
        )
        for layer in range(6)
    ]
    return alt.hconcat(
        layer_viz[0]
        # & layer_viz[1]
        & layer_viz[2]
        # & layer_viz[3]
        & layer_viz[4]
        # & layer_viz[5]
    )


show_example(viz_encoder_self)
```

```
Preparing Data ...
Loading Trained Model ...
Checking Model Outputs:

Example 0 ========

Source Text (Input)        : <s> Zwei Männer mit <unk> sprechen miteinander auf einem Freiluftmarkt . </s>
Target Text (Ground Truth) : <s> Two men in <unk> 's have a discussion in an outdoor market . </s>
Model Output               : <s> Two men are talking to each other in an open air market . </s>
```

Layer 1



Layer 3



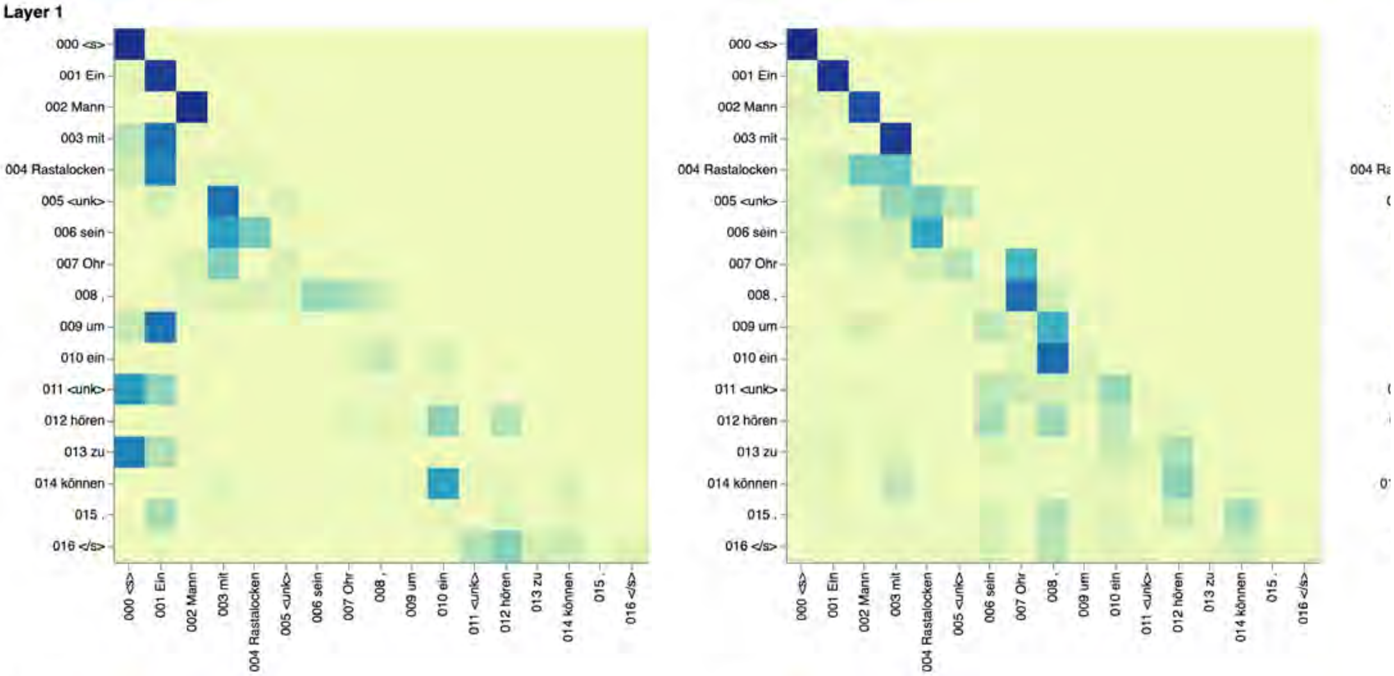
Layer 5


```
def viz_decoder_self():
    model, example_data = run_model_example(n_examples=1)
    example = example_data[len(example_data) - 1]

    layer_viz = [
        visualize_layer(
```

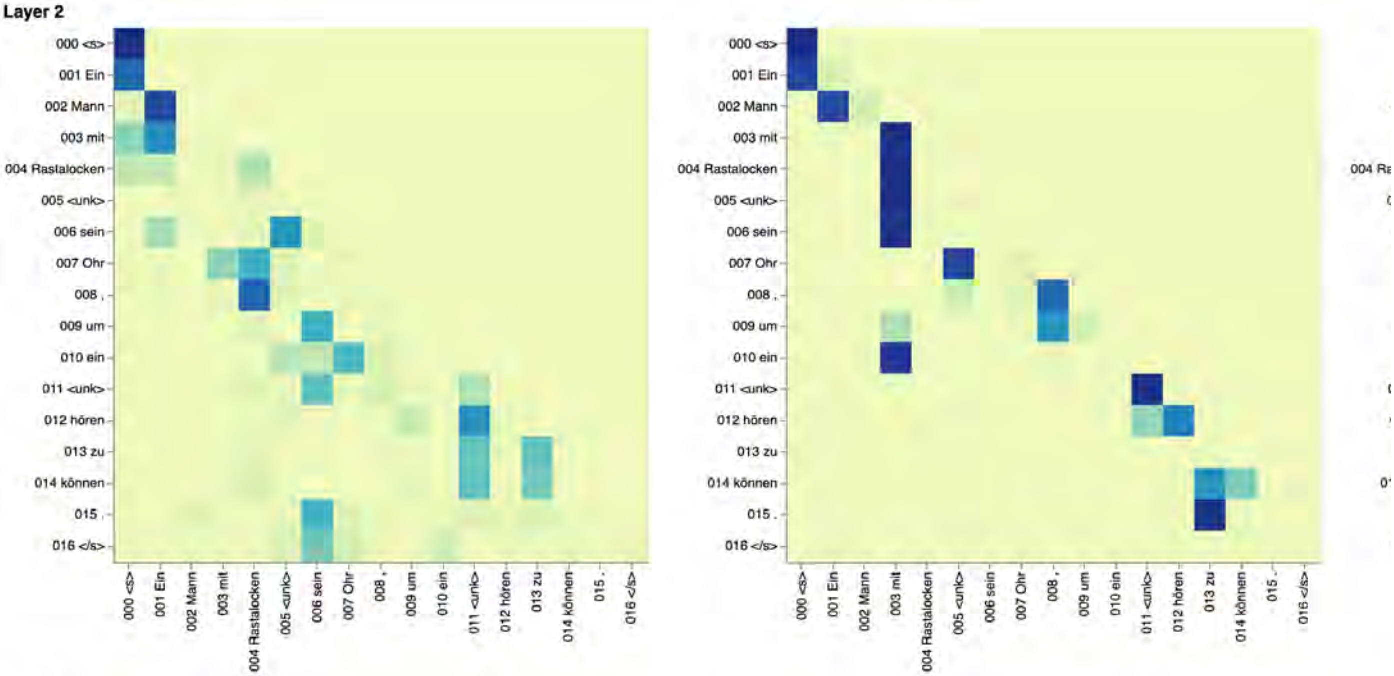
```
            model,
            layer,
            get_decoder_self,
            len(example[1]),
            example[1],
            example[1],
        )
        for layer in range(6)
    ]
    return alt.hconcat(
        layer_viz[0]
        & layer_viz[1]
        & layer_viz[2]
        & layer_viz[3]
        & layer_viz[4]
        & layer_viz[5]
    )


show_example(viz_decoder_self)
```

```
Preparing Data ...
Loading Trained Model ...
Checking Model Outputs:

Example 0 ========

Source Text (Input)        : <s> Ein Mann mit Rastalocken <unk> sein Ohr , um ein <unk> hören zu können . </s>
Target Text (Ground Truth) : <s> A man with dreadlocks is plugging his ear to hear a phone call . </s>
Model Output               : <s> A man with a <unk> hat is seen his ear to listen to be a music stand . </s>
```

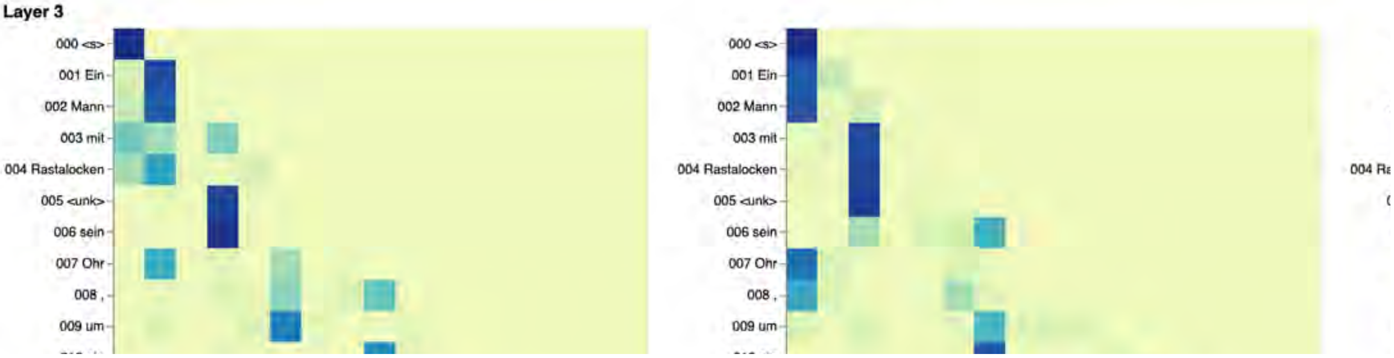
Layer 1



Layer 2



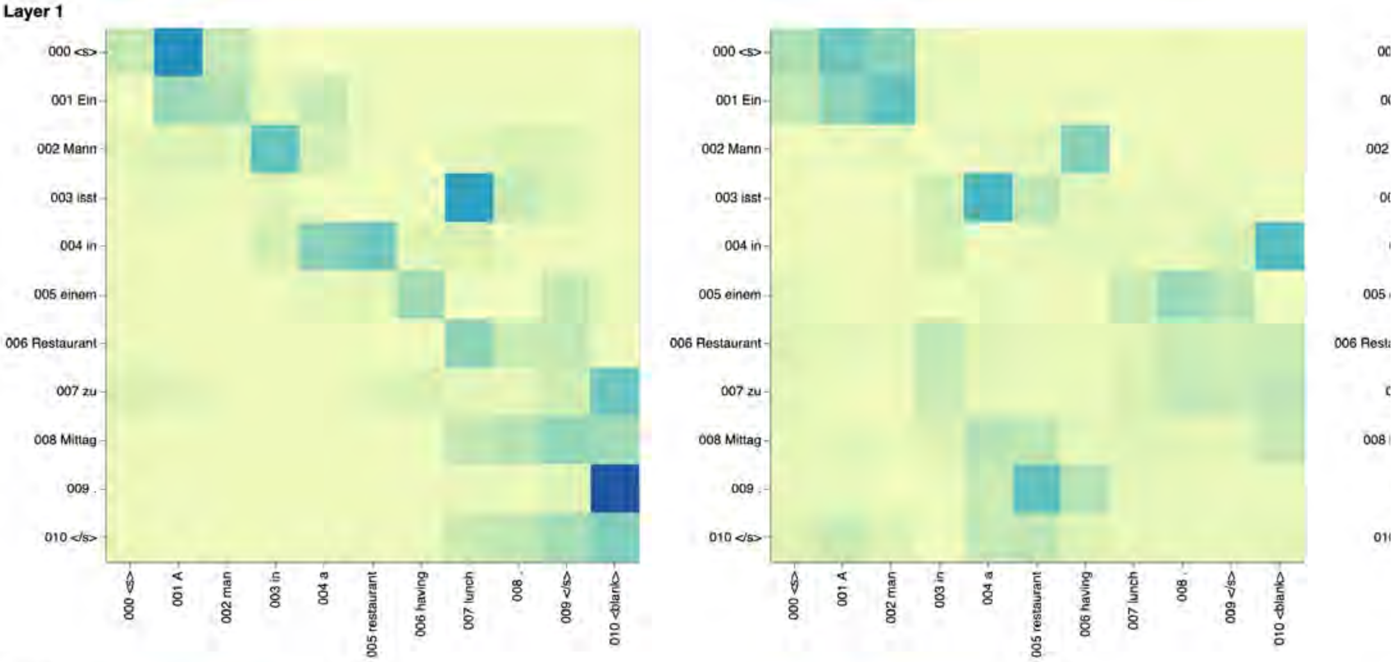
Layer 3



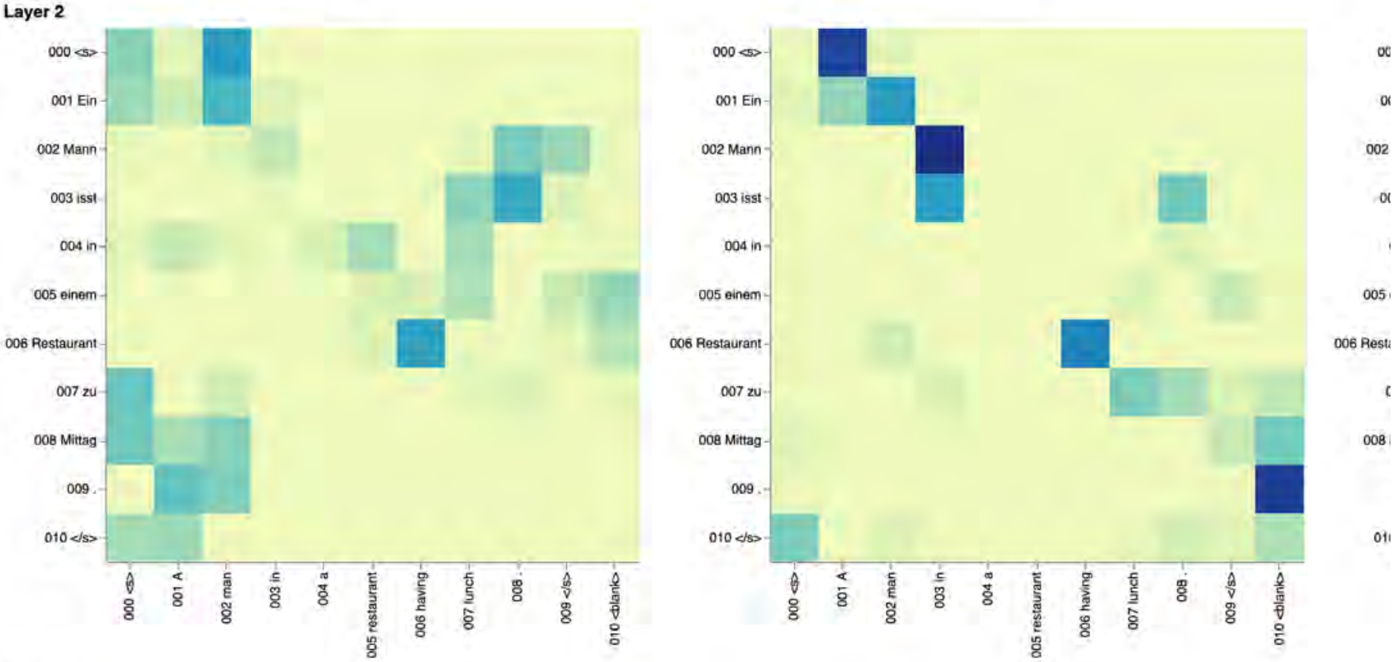

```
def viz_decoder_src():
    model, example_data = run_model_example(n_examples=1)
    example = example_data[len(example_data) - 1]

    layer_viz = [
        visualize_layer(
            model,
            layer,
            get_decoder_src,
            max(len(example[1]), len(example[2])),
            example[1],
            example[2],
        )
        for layer in range(6)
    ]
    return alt.hconcat(
        layer_viz[0]
        & layer_viz[1]
        & layer_viz[2]
        & layer_viz[3]
        & layer_viz[4]
        & layer_viz[5]
    )


show_example(viz_decoder_src)
```

```
Preparing Data ...
Loading Trained Model ...
Checking Model Outputs:

Example 0 ========

Source Text (Input)        : <s> Ein Mann isst in einem Restaurant zu Mittag . </s>
Target Text (Ground Truth) : <s> A man in a restaurant having lunch . </s>
Model Output               : <s> A man eats lunch to a restaurant . </s>
```

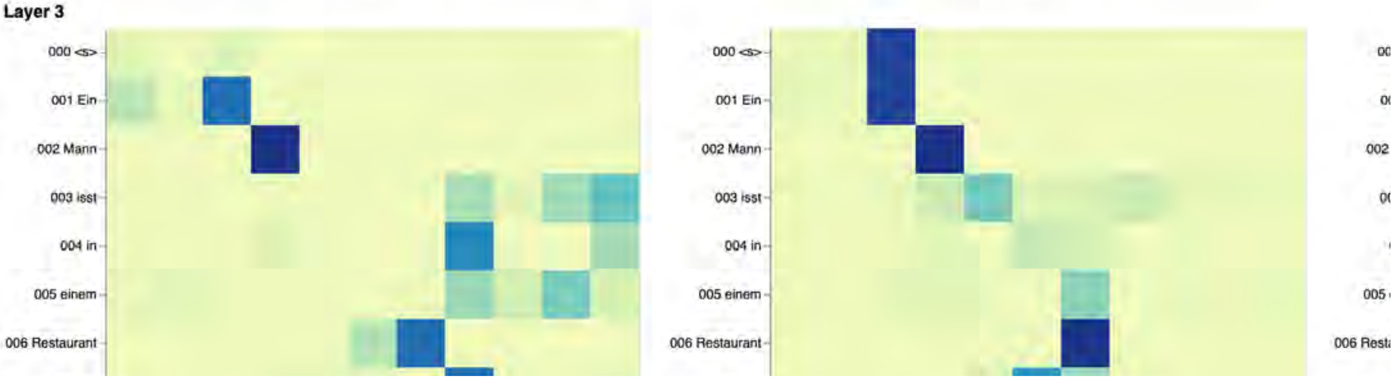
Layer 1

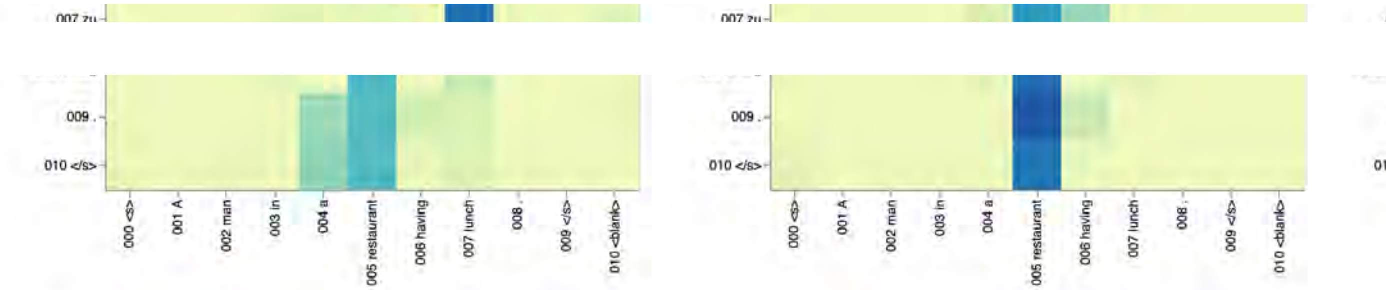


Layer 2

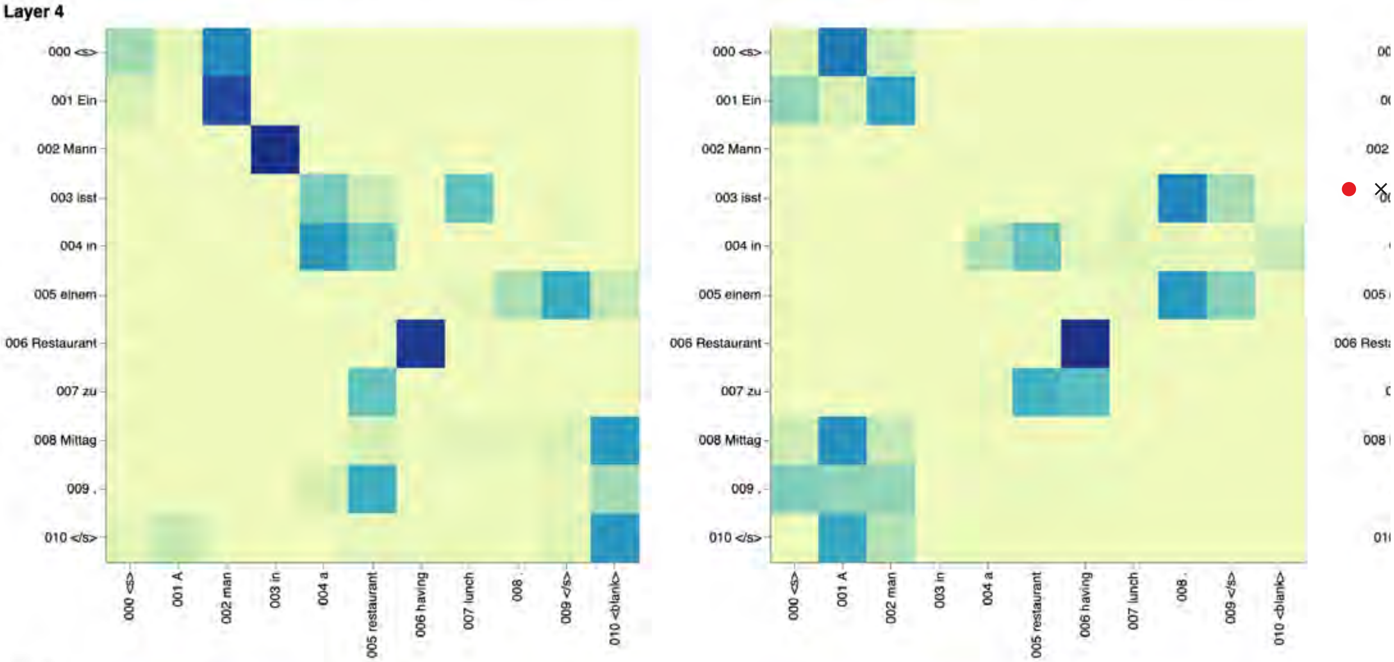


Layer 3





Layer 4



Layer 5