University of Hamburg

Bachelor's Thesis

Visualizing Language using Attention Head based t-SNE

by

Bent Müller
Matriculation Number *******

Supervisor and First Assessor:
Prof. Dr. Armin Iske

Co-Assessor:
Dr. Claus Goetz

Submission Date: 27.01.2023

Contents

1 Introduction **2**
1.1 Structure of the Thesis . . . 3

2 Mathematical Foundation **4**
2.1 General Foundations . . . 4
2.2 Transformer Building blocks . . . 7
2.2.1 Hyperparameters and Learned Parameters . . . 7
2.2.2 Tokenization . . . 8
2.2.3 Embeddings . . . 10
2.2.4 Tensors & Broadcasting rules . . . 10
2.2.5 Basics from Neural Networks . . . 13
2.3 The Transformer Architecture in Detail . . . 15
2.3.1 Transformer-Specific Building Blocks . . . 15
2.4 Transformer Architectures . . . 23
2.5 Dimensionality Reduction Techniques . . . 24
2.5.1 t-SNE in Detail . . . 26

3 The Algorithm **29**
3.1 Attention Head based t-SNE . . . 29

4 Numerical Findings **31**
4.1 Standard t-SNE . . . 31
4.2 Attention Head based t-SNE in Practice . . . 34
4.2.1 Comparing Attention Head based t-SNE to Standard t-SNE . . . 39
4.3 Visualizing Maximum Attention . . . 41
4.4 Examining the robustness of Attention Head based t-SNE . . . 42

5 Conclusion & Outlook **42**
5.1 Attention Head based t-SNE . . . 43
5.2 Implications for t-SNE in General . . . 43
5.3 Conclusion on Current Transformer Architectures . . . 43
5.4 Outlook for Future Work . . . 44

A Appendix **49**
A.1 Determining the Standard Deviation in t-SNE . . . 49
A.2 Parameters used in the t-SNEs . . . 49
A.3 Quantile Equidistant Rescaling of Scatter Plots . . . 49
A.4 Abstract of the Original t-SNE Paper . . . 50
A.5 More on Attention Head based t-SNE . . . 51

Visualizing Language using Attention Head based t-SNE

Bent Müller
bent.muller@icloud.com

University of Hamburg, Department of Mathematics

Abstract

In this thesis, I have looked at the Transformer Architecture, a neural network architecture which revolutionized the field of Natural Language Processing (NLP) driving AI systems to the point at which they create real value in our society. I looked at transformers from a mathematical viewpoint, clearly defined its components and deeply examined its attention mechanism. Further, I have developed a novel visualization technique, based on the well known t-distributed Stochastic Neighbor Embedding (t-SNE), that allows us to literally look inside transformers as they process text, giving us better insights in understanding their power and limitations.

1 Introduction

The Transformer Architecture, first introduced by [Vaswani et al., 2017], is a novel neural network architecture which has proven to excel in various tasks. While a standard neural network may be regarded as a parameterized function $f_\theta : \mathcal{X} \to \mathcal{Y}$, that can learn to approximate a map for which input-output pairs $(x_1, y_1), \dots, (x_m, y_m)$ are known, transformers excel particularly at sequence modeling in which a number of sequences $(x_1^{(1)}, \dots, x_{n_1}^{(1)}), \dots, (x_1^{(m)}, \dots, x_{n_m}^{(m)})$ is known. Since there are multiple ways of defining the task of sequence modeling in the case of Natural Language Processing (NLP), I will present some of the most important ones in the following, the $x_i^{(j)}$ may be regarded as words such that $(x_1^{(1)}, \dots, x_{n_1}^{(1)})$ forms a text with n_1 words.

Masked Language Modeling (MLM) is a task in which a random token (word) from a text is masked[1] and the model tries to predict it. This is

[1]Specifically, the token is replaced by the '[MASK]' token such that the model explicitly knows which token it is supposed to predict. The exact masking procedure is a little more sophisticated as multiple tokens are masked and predicted simultaneously, please see the original paper for details.

one of the tasks on which the famous BERT Architecture was trained[2] [Devlin et al., 2018].

Next Sentence Prediction (NSP) is a task in which the model is given two sentences[3] and needs to predict whether the second sentence really followed the first one in a given text corpus, or whether it was randomly chosen from it. The idea behind this task is that the model should learn to identify relationships across sentences that reach specifically beyond the token level. This was the other task on which BERT was simultaneously trained.

Natural Language Generation (NLG) is technically a special case of MLM, in which always the last token of a text is masked. Examples of large models trained on this task are the GPT series [Brown et al., 2020, Radford et al., 2019] and the recent PaLM [Chowdhery et al., 2022]. The main difference to MLM is that models trained using NLG specialize on generating text which might be more desirable in certain applications.

Language models are usually pretrained on a large corpus of text using one of the above-mentioned tasks and then fine-tuned on a different, more specific task [Reimers and Gurevych, 2019, Araci, 2019]. It turns out that models trained on these naive tasks learn valuable numerical representations of natural language which can be used in a wide variety of other NLP tasks.

Transformers are nowadays also used in Computer Vision tasks, specifically for image classification in which they surpassed the previous state-of-the-art, which were usually based on Convolutional Neural Networks (CNNs) [Dosovitskiy et al., 2020, Liu et al., 2021]. Similarly, transformers are also used in video processing tasks [Arnab et al., 2021].

Further, transformer models have also been adopted in other tasks, such as Automatic Speech Recognition (ASR) either directly as end-to-end models [Dong et al., 2018] or as hybrids [Gulati et al., 2020]. They have even proven successful for tasks like solving Partial Differential Equations (PDEs) [Cao, 2021, Kovachki et al., 2021][4].

1.1 Structure of the Thesis

The Transformer's diverse success leads one to believe that the architecture bears some significant advantage over other architectures for building

[2]training, learning, and fitting denote the process of adjusting the parameters θ of a model such that it better, according to a predefined loss metric, approximates the observations $(x_1, y_1), \ldots, (x_m, y_m)$.

[3]In the BERT paper, a sentence does not necessarily denote a single linguistic sentence but rather an arbitrary body of text.

[4]In [Kovachki et al., 2021], the transformer architecture is proposed in section 5.2

neural networks. In this work, the aim is to study where exactly this advantage comes from. Specifically, the transformer will be examined from a mathematical standpoint to discover what exactly it succeeds at where other models fail. While the functioning of the transformer is well understood, it is less known what exactly it learns and how it utilizes this knowledge.

A technique is presented which allows us to study the transformer's internal attention mechanisms in detail, specifically it is modified version of the well known dimensionality reduction technique t-SNE [Van der Maaten and Hinton, 2008] which can be used to visualize internal representations of the transformer in a way that lets us observe how it learns abstractions.

Finally, the implications of the observations will be discussed regarding our understanding of the transformer architecture, and it is hypothesized how it constructively builds abstractions.

2 Mathematical Foundation

Before introducing the transformer architecture in detail, its building blocks will be defined such that its exact workings are better understood later on. A short introduction to the mathematical concepts used in this work as well as to neural networks are given in this section.

2.1 General Foundations

Definition 2.1 (Probability Vector). *A vector $X \in \mathbb{R}^n$ is called n-dimensional* ***probability vector*** *if and only if it satisfies*

(i) $X_i \in [0,1]$ *and*

(ii) $\sum_{i=1}^{n} X_i = 1$.

Connecting this definition to our background from probability theory, we see that each such vector describes the distribution of a discrete random variable, i.e., one valued in $\{1,\ldots,n\} \subset \mathbb{N}$.

Definition 2.2 (Softmax).

$$\begin{aligned}\mathit{Softmax}: \quad & \mathbb{R}^n \to \mathbb{R}^n \\ & x \mapsto (\exp(x_1),\ldots,\exp(x_n)) \cdot \frac{1}{\sum_{i=1}^{n}\exp(x_i)}\end{aligned}$$

The Softmax function maps any $x \in \mathbb{R}^n$ to a probability vector.

To avoid confusion, if the $\exp(\cdot)$ function is applied on matrices, it does not denote the matrix exponential but is applied **element-wise**, opposed to the usual mathematical notation.

Definition 2.3 (Right-Stochastic Matrix)**.** *A matrix $A \in \mathbb{R}^{n \times n}$ is called* ***right-stochastic*** *if and only if*

(i) $A_{ij} \in [0, 1]$ *and*

(ii) $\sum_{j=1}^{n} A_{ij} = 1$ *for all* $i \in \{1, \dots, n\}$.

Every right-stochastic matrix can be interpreted as the transition matrix of a markov chain such that A_{ij} denotes the probability $P(X_{t+1} = j \mid X_t = i)$ of transitioning from state i to state j.

Definition 2.4 (Kullback-Leibler (KL) Divergence)**.** *Let X and Y be two probability vectors. Then the* ***Kullback-Leibler divergence*** *between X and Y is defined as*

$$KL(X \parallel Y) = \sum_{i=1}^{n} X_i \log \frac{X_i}{Y_i}$$

The Kullback-Leibler divergence is a measure of how well the distribution Y approximates the distribution X. It can also be defined for continuous probability distributions, but in our case we only need the discrete version.

Definition 2.5 (Activation Function)**.** *We call any map*

$$\phi : \mathbb{R} \to \mathbb{R}$$

which is differentiable almost everywhere[5] *an* ***activation function****. Recall that this means ϕ is differentiable everywhere except on any countable set of points in $\mathbb{R}$. The outputs of ϕ are called* ***activations****.*

Activation functions play an important role in neural networks as they allow the network to learn non-linear functions. For certain non-linear activation functions, neural networks unleash their power as universal function approximators [Hornik et al., 1989]. Further, we require ϕ to be differentiable almost everywhere, since we will only be evaluating the derivative of ϕ at countably many points in $\mathbb{R}$.

Definition 2.6 (Rectified Linear Unit (ReLU))**.**

$$\begin{aligned} ReLU : & \mathbb{R} \to \mathbb{R}_{\geq 0} \\ & x \mapsto \max(0, x) \end{aligned}$$

Also, in BERT a similar activation function is used called *GELU*, which is short for *Gaussian Error Linear Unit.*

[5] ϕ is thus differentiable everywhere except on any subset of $\mathbb{R}$ that has Lebesgue measure 0, specifically any countable set of points.

Definition 2.7 (GELU)**.**

$$\begin{aligned} GELU: \quad & \mathbb{R} \to \mathbb{R} \\ & x \mapsto x \cdot \Phi(x) = x \cdot \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-t^2/2}\, dt \end{aligned}$$

where $\Phi(x)$ is the cumulative distribution function of the standard normal distribution. GELU can be approximated by

$$GELU(x) \approx \frac{1}{2} \cdot \left(1 + \tanh\left(\sqrt{\frac{2}{\pi}} \cdot (x + 0.044715 \cdot x^3)\right)\right)$$

to achieve a faster computation.

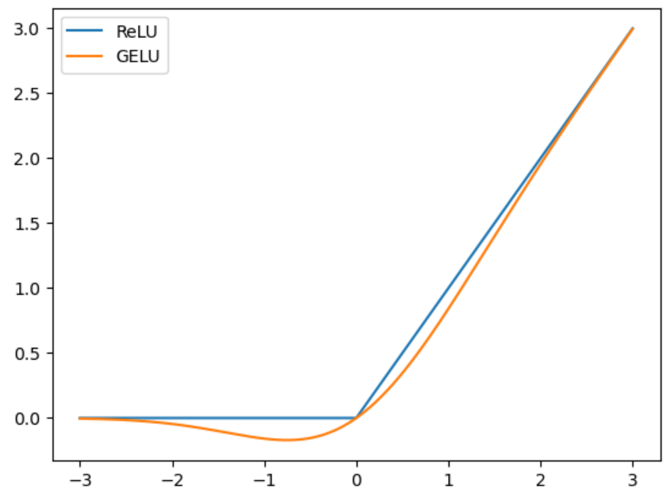

Figure 1: ReLU and GELU activation functions plotted on $[-3, 3]$

Figure 1 shows the ReLU and GELU activation functions on the interval $[-3, 3]$. It has been shown that the GELU activation function is more effective than the ReLU activation function in neural networks, since it "weights inputs by their value, rather than gates inputs by their sign as in ReLUs" [Hendrycks and Gimpel, 2016]. GELU better approximates the identity near the origin and smoothly decays its absolute activation as $x \to -\infty$, whereas the ReLU activation function cuts off any input for $x < 0$.

2.2 Transformer Building blocks

In the following, we define the building blocks that make up the transformer model itself. Since in the context of transformers a number of hyperparameters are used that lay out the architecture and size of the model, we will define them here as well. Also, the notion of hyperparameters and learned parameters in general will be introduced here.

2.2.1 Hyperparameters and Learned Parameters

Learned parameters are part of the model and will later be optimized such that our model can fit actual data. A simple example is a weight matrix, whose entries will be optimized, so its corresponding linear transformation helps solve some part inside our bigger language model which will consist of many such transformations. For example this provides us the ability to *learn linear transformations.*

Hyperparameters are constant and usually define the architecture, size and sometimes numerical stability of the model. An example might be the dimension of some weight matrix, such parameters will also determine the computational cost of running the resulting model. Below, all important hyperparameters used in the context of this thesis are defined. How and where these parameters are used, will become more clear after all the components are defined.

To keep definitions short, the learned variables will sometimes be shortened to θ, which in that case denotes the tuple of all learned variables, and the hyperparameters assumed to be fixed.

- $n_v \in \mathbb{N}; \sim$ vocabulary size; The total number of tokens in the *vocabulary* of our model.
- $n_s \in \mathbb{N}; \sim$ sequence length; The maximum number of tokens our model can process at one step.
- $d_m \in \mathbb{N}; \sim$ Model dimension, sometimes d_{model} in the literature; The dimension in which each token will be embedded. Hence, our embeddings will be elements in $\mathbb{R}^{d_m}$.
- $d_k = d_q \in \mathbb{N}; \sim$ Keys and Queries dimension; The dimension to which keys and queries are projected in the attention mechanism.
- $d_v \in \mathbb{N}; \sim$ Values dimension; The dimension to which values are projected in the attention mechanism.
- $n_l \in \mathbb{N}; \sim$ Number of layers
- $h \in \mathbb{N}; \sim$ Number of attention heads
- $\varepsilon \in \mathbb{R}_{>0}; \sim$ Layer Normalization epsilon
- $d_f \in \mathbb{N}; \sim$ Feed forward network dimension, sometimes d_{ffn} in the literature

Subsequently, each of the important fundamental building blocks will now be defined. From these building blocks, the transformer model will then be defined.

2.2.2 Tokenization

In Natural Language Processing (NLP) we encounter the problem of modeling arbitrary words such that they can be effectively represented and understood by a computer, this usually involves finding numerical representations for words. To handle this fundamental problem, tokenizers are commonly used [Mielke et al., 2021].

A tokenizer breaks apart a text into a sequence of tokens, such that the more frequently used words are represented by single tokens (e.g., the word *'the'* directly becomes 'the') and the less frequently used words are represented by multiple tokens (e.g., *'university'* might become 'univers' and '##ity'), modeling every possible text as a sequence of tokens. Each token is encoded by its position in the vocabulary, an integer, such that more common tokens have a lower indices and less common tokens have higher indices.

In this case the '##' indicate that the token is not preceded by a space. This technique has the advantage that the token space is kept relatively[6] small and certain tokens are reused within multiple multi-token words, for example the token 'univers' is also used in 'universe', 'universal'.

Initially a tokenizer maps a token to an integer, which represents the token's index in the vocabulary. In a next step, an embedding vector will be learned for each of the tokens in the vocabulary.

Table 1: Tokenization of the sentence *"Tokenizers are great."* as encoded by BERT's tokenizer.

Token	Token Index
[CLS]	101
token	19204
##izer	17629
##s	2015
are	2024
great	2307
.	1012
[SEP]	102

Table 1 shows an example of how BERT's [Devlin et al., 2018] tokenizer processes the sentence *"Tokenizers are great."*. The actual input to BERT will then be the integer sequence of token indices 101, 19204, . . . , 1012, 102. Observe that the work 'are', which is a common word, is indexed at 2024

[6]relative to the total number of words this technique can model; BERT uses a vocabulary of 30,522 tokens

while ‘token’ is indexed at 19204, since it is much less common in the data BERT was trained on.

Notice the special tokens *‘[CLS]’* and *‘[SEP]’*, which did not appear in the original sentence. These tokens are specific to BERT since they were used in the task of Next Sentence Prediction (NSP). The *‘[SEP]’* token defines the end of a sentence such that the NSP task is not bound to actual *linguistic sentences* and can thus be applied to two arbitrary texts [Devlin et al., 2018, Input/Output Representations in Chapter 3].

The *‘[CLS]’* token marks the final embedding which will be used in classification tasks. This is necessary because after embedding the tokens into $\mathbb{R}^{d_m}$, we finally want to use a single vector to represent the[7] tokens during training, we cannot use any of the tokens’ embeddings for classification.

After tokenization, the sequence of tokens is either truncated, if it is too long, or padded, if it is too short, to a fixed length, as specified by the maximum sequence length n_s. Padding is done by appending an appropriate number of *‘[PAD]’* tokens to the sequence. Tokens that have not been added to the vocabulary are mapped to the *‘[UNK]’* token.

For the sake of simplicity, we will assume that a tokenizer is a function that maps a text to a fixed-length integer sequence, specifically

$$\text{Tokenizer}: \quad \text{Text} \to \mathbb{N}^{n_s}_{\leq n_v} := \{n \in \mathbb{N} : n \leq n_v\}^{n_s}$$

without defining the exact details since it doesn’t concern us in this work.

We can see that a tokenizer maps text of arbitrary length containing arbitrary characters to a fixed-length integer sequence which is bounded by the vocabulary size n_v. A tokenizer is also ‘trained’ by learning the vocabulary from a corpus of text and implicitly their word frequencies using which it then decides which words are being split into multiple tokens. For example BERT’s tokenizer even learned tokens for certain chinese symbols, however it is questionable, since these were likely underrepresented in the training data, if BERT can understand chinese as well as english. So tokenization, while being very successful in english, is still an open area of research especially regarding multilingual models.

2.2.3 Embeddings

Tokenizers solve some very basic but central problems, however they don’t yield good representations of the words in the text. To overcome this problem, transformers use *learned embeddings* in form of vectors in $\mathbb{R}^{d_m}$.

[7]Masking in BERT is done by randomly replacing tokens (1) with the *‘[MASK]’* token, (2) with random tokens, or (3) with their original tokens (not masking effectively) and then predicting those tokens using their final embeddings. For more details, please see [Devlin et al., 2018, Chapter 3.1 Task #1: Masked LM].

Definition 2.8 (Token Embedding)**.**

$$\begin{aligned} \textit{Token Embedding}_{\theta_E} :& \mathbb{N}_{\leq n_v} \to \mathbb{R}^{d_m}, \quad \theta_E \in \mathbb{R}^{n_v \times d_m} \\ & t \mapsto (\theta_E)_{t,\cdot} = ((\theta_E)_{t,1}, \ldots, (\theta_E)_{t,d_m}) \end{aligned}$$

The token embedding is a parameterized function which maps an integer to a vector in $\mathbb{R}^{d_m}$ by selecting the corresponding row of the learned embedding matrix $\theta_E \in \mathbb{R}^{n_v \times d_m}$.

Notice that the embedding matrix θ_E is a learned parameter. This means that it is initialized randomly, usually with elements drawn from a standard normal distribution, and then updated during training. How exactly this initialization is done is rather irrelevant given that it provides adequate numerical stability throughout the model so it can move the random embeddings to form meaningful representations. One can imagine that during training, the model can freely choose where to place each embedding in $\mathbb{R}^{d_m}$.

2.2.4 Tensors & Broadcasting rules

In this thesis, we will often encounter maps between cartesian products of high-dimensional spaces. To enable convenient notation, we define structured multidimensional objects, tensors, in the following.

Definition 2.9 (Tensor)**.** *Let F be an arbitrary non-empty set and $d_1, \ldots, d_n \in \mathbb{N}$ be non-negative integers for some $n \in \mathbb{N}_0$. Then we call the collection*

$$x \in F^{d_1 \times \cdots \times d_n}, \quad \textit{where } F^{d_1 \times \cdots \times d_n} := F^{d_1} \times \cdots \times F^{d_n}$$

*a **tensor** of rank n and shape $(d_1, \ldots, d_n)$. For $k_i \in \{1, \ldots, d_i\}, 1 \leq i \leq n$, we call the n-tuple $(k_1, \ldots, k_n)$ a **multi index** which can be used to access the elements of x as follows:*

$$x_{k_1,\ldots,k_n} \in F$$

*In the special case of $n = 0$ this formulation yields a rank-0 tensor, which cannot be indexed since its shape is the empty set. Such a tensor is known as a **scalar** and is just an element in F. To make this definition more explicit, we can flatten any tensor x yielding $\hat{x}$ which is a vector in $F^{\prod_{i=1}^{n} d_i}$. This makes indexing tensors concrete:*

$$x_{k_1,\ldots,k_n} := \hat{x}_{\hat{k}}, \quad \textit{where } \hat{k} := \sum_{i=1}^{n} k_i \prod_{j=1}^{i-1} d_j$$

Further we recognize that, because of flattening, tensors are just vectors that can be indexed in a structured way.

This kind of flattening is also known as *column-major* ordering of tensors, sometimes also referred to as *Fortran*[8] ordering. Note that there are other ways of flattening (indexing) multidimensional arrays (tensors) [Morton, 1966, Beckmann et al., 1990] which we will not discuss in this thesis [Knuth, 1968, Chapter 2.2.6, Page 299]. In general, the idea of multidimensional arrays exists for a long time.

Mostly we will be concerned with real-valued tensors, in which case $F = \mathbb{R}$. Connecting tensors to our knowledge from linear algebra, we recognize that we are already familiar with tensors of rank 1, also known as vectors which are indexed with single integers, and of rank 2, which are matrices indexed with tuples of two integers (commonly i, j).

Higher-rank tensors will simply be understood as structured collections of elements in F. For example, a tensor of rank 3 can be understood as a collection (or stack) of matrices, that is indexed with a 3-tuple. Fixing any of the three indices will yield a matrix indexed in the remaining two indices.

Definition 2.10 (Partial Indexing)**.** *Let x be a tensor of rank $n \in \mathbb{N}$ and shape $(d_1, \dots, d_n)$ with values in F. Then we can* ***partially index*** *x with any multi index $(k_1, \dots, k_m)$ for $n \geq m \in \mathbb{N}$ such that $k_i \in \{1, \dots, d_i\}$ for all $i \in \{1, \dots, m\}$. This yields*

$$x_{k_1,\dots,k_m} \in F^{d_{m+1} \times \cdots \times d_n},$$

which is a tensor of rank $n - m$ and shape $(d_{m+1}, \dots, d_n)$ also with values in F.

Note that to avoid confusion, in this thesis when using partial indexing as demonstrated above, we always index the leading dimensions first, from the left in this case. When different indexing is desired, a transpose operation can be applied. If one deals with tensors of rank greater than 2, we call the transpose operation a permutation of dimensions since it is not obvious how dimensions are permuted, which is unambiguous for matrices.

Tensors allow us to define certain *broadcasting rules* which will later enable us to conveniently express operations on tensors that we want to apply independently along certain dimensions.

Definition 2.11 (Broadcasting)**.** *Let $f : V \to W$ be a map between two non-empty sets. Then we can* ***broadcast*** *f to yield the map*

$$\begin{aligned} \hat{f} :& V^{d_1 \times d_2 \times \cdots \times d_n} \to W^{d_1 \times d_2 \times \cdots \times d_n}, \textit{ for arbitrary } d_i \in \mathbb{N} \\ (\hat{f}(v))_{k_1,\dots,k_n} &= f(v_{k_1,\dots,k_n}), \quad \textit{for } k_i \in \{1, \dots, d_i\}. \end{aligned}$$

We explicitly say that f is ***broadcasted*** *over the dimensions $d_1, \dots, d_n$ when applying f to some $v \in V^{d_1 \times d_2 \times \cdots \times d_n}$ and in that case actually apply $\hat{f}$.*

[8]Fortran is a programming language which was developed in the 1950s which has been used in scientific computing.

Broadcasting and tensors as defined above, are inspired by the python library *NumPy* [Harris et al., 2020] which uses *ND Arrays* (n-dimensional arrays) to represent tensors and broadcasting naturally when applying element-wise maps to tensors.

This definition allows us to apply the token embedding, which was only defined for single integers, to map sequences of integers, the tokens, to sequences of vectors in $\mathbb{R}^{d_m}$ (i.e., to matrices in $\mathbb{R}^{n_s \times d_m}$). In this case we have $V = \mathbb{N}_{\leq n_v}$, $W = \mathbb{R}^{d_m}$ and $d = (d_1) = (n_s)$ and we yield the map $\hat{f} : \mathbb{N}^{n_s}_{\leq n_v} \to \mathbb{R}^{d_m \times n_s}$ which we will also denote as a regular token embedding map.

Often we define a map on $\mathbb{R}^{d_l \times \cdots \times d_n}$ for some $n \geq l \in \mathbb{N}$, but want to broadcast it only over a subset of the dimensions, usually some leading dimensions. Technically, our current definition of broadcasting allows this simply by letting $V = \mathbb{R}^{d_l \times \cdots \times d_n}$, which yields a broadcasted map on $(\mathbb{R}^{d_l \times \cdots \times d_n})^{d_1 \times \cdots \times d_{l-1}}$ for any f defined on V. However, this gets confusing and inaccurate quickly, so we want to define *partial broadcasting* to help us with this.

Definition 2.12 (Partial Broadcasting)**.** *Let $f : V^{d_l \times \cdots \times d_n} \to W^{d_l \times \cdots \times d_n}$ be a map between two arbitrary cartesian products of sets where $n \geq l \in \mathbb{N}$. Then we can **partially broadcast** f to yield the broadcasted map*

$$\begin{aligned}\hat{f} :& V^{d_1 \times \cdots \times d_l \times \cdots \times d_n} \to W^{d_1 \times \cdots \times d_l \times \cdots \times d_n}, \text{ for some } d_i \in \mathbb{N} \\ (\hat{f}(v))_{k_1,\ldots,k_n} &= f(v_{k_1,\ldots,k_{l-1}})_{k_l,\ldots,k_n}, \quad \text{for } k_i \in \{1,\ldots,d_i\}\end{aligned}$$

*for which we say that f is **partially broadcasted** over the dimensions $d_1,\ldots,d_{l-1}$. Equivalently, we say that f is **broadcasted** over the first $l-1$ leading dimensions of V. Further, if f is applied to any tensor v of shape $(d_1,\ldots,d_l,\ldots,d_n)$, it is implicitly broadcasted over the leading dimensions $d_1,\ldots,d_{l-1}$.*

Partial broadcasting allows for dependence of f within the not broadcasted dimensions $d_l,\ldots,d_n$. For example, we can now apply the *softmax function* (see 2.2) which was originally defined on $\mathbb{R}^n$ and apply it to tensors of arbitrary shape and rank ≥ 1 by broadcasting it over all dimensions except the last one.

Definition 2.13 (Point-wise Tensor Product)**.** *Let x, y be two tensors of shape $(d_1,\ldots,d_n)$ with values in $\mathbb{R}$. Then we define the **point-wise tensor product** of x and y as a map*

$$\begin{aligned}\odot : \mathbb{R}^{d_1 \times \cdots \times d_n} \times \mathbb{R}^{d_1 \times \cdots \times d_n} &\to \mathbb{R}^{d_1 \times \cdots \times d_n} \\ \text{such that } (x \odot y)_{k_1,\ldots,k_n} &= x_{k_1,\ldots,k_n} \cdot y_{k_1,\ldots,k_n}.\end{aligned}$$

for every multi-index $k_1,\ldots,k_n, k_i \in \{1,\ldots,d_i\}$.

Definition 2.14 (Concatenation)**.** *Let x be a tensor of shape $(d_1, \ldots, d_q, \ldots, d_n)$ and y be a tensor of shape $(l_1, \ldots, l_q, \ldots, l_n)$, whose shapes match in all dimensions except for q-th, i.e., $d_i = l_i$ for all $i \in \{1, \ldots, n\} \backslash \{q\}$. Both tensors are valued in some non-empty set F. Then we define the* ***concatenation*** *of x and y along the q-th dimension as a map*

$$Concat_q : F^{d_1 \times \cdots \times d_q \times \cdots \times d_n} \times F^{l_1 \times \cdots \times l_q \times \cdots \times l_n} \to F^{d_1 \times \cdots \times (d_q + l_q) \times \cdots \times d_n}$$

such that

$$(Concat_q(x, y))_{k_1, \ldots, k_n} = \begin{cases} x_{k_1, \ldots, k_q, \ldots, k_n} & \text{if } 1 \leq k_q \leq d_q \\ y_{k_1, \ldots, k_q, \ldots, k_n} & \text{if } d_q \leq k_q \leq d_q + l_q \end{cases}$$

for every multi index $(k_1, \ldots, k_q, \ldots, k_n)$ with $1 \leq k_q \leq d_q + l_q$ and $1 \leq k_i \leq n$ if $i \neq q$. In this case, we say that x and y are ***concatenated along the q-th dimension*** *(or sometimes axis). For a set of tensors $\{x_1, \ldots, x_m\}$, with pairwise compatible shapes as above, we define the* ***concatenation*** *of all tensors as*

$$Concat_q(x_1, \ldots, x_m) = Concat_q(Concat_q(\ldots Concat_q(x_1, x_2), x_3 \ldots), x_m).$$

2.2.5 Basics from Neural Networks

We now introduce some basic concepts that are commonly used in building artificial neural networks (ANNs). However, we will solely define components used in the Transformer model, specifically in BERT.

Definition 2.15 (Layer Normalization)**.** *For a given $\varepsilon > 0$ and $n \in \mathbb{N}$, we define* ***Layer Normalization*** *as a vector-valued map as follows:*

$$\begin{aligned} LayerNorm_{\gamma,\beta,\varepsilon} : \mathbb{R}^n &\to \mathbb{R}^n \qquad \gamma, \beta \in \mathbb{R}^n, \varepsilon \in \mathbb{R}_{>0} \\ x &\longmapsto \gamma \odot \frac{x - \mu}{\sigma + \varepsilon} + \beta \qquad \text{with} \\ \mu &= \frac{1}{n} \sum_{i=1}^{n} x_i \quad \text{the mean of } x \text{ and} \\ \sigma &= \sqrt{\frac{1}{n} \sum_{i=1}^{n} (x_i - \mu)^2} \quad \text{its standard deviation} \end{aligned}$$

Where γ and β are learned parameters and ε is a small constant to avoid numerical instability when dividing by small σ. ε is a hyperparameter and commonly set to 10^{-8}.

The *learned* parameters γ and β are sometimes called *gain* and *bias* respectively. These can rescale or shift specific elements of the input vector. Notice that the above product $\gamma \odot \frac{x-\mu}{\sigma+\varepsilon}$ is a *point-wise tensor product* as defined in 2.13. Later, the transformer uses layer normalization in combination with the previously defined partial broadcasting rules (see 2.12) as a matrix-valued map which is broadcasted along the rows (leading dimension). Layer Normalization greatly improves numerical stability and thus training of neural networks, where it is commonly used to stabilize internal representations (i.e., activations) [Ba et al., 2016]. This helps the neural network keep the distributions of its internal representations stable.

Definition 2.16 (Feed-Forward Layer)**.** *Let $d_{in}, d_{out} \in \mathbb{N}$ be two given positive integers called the input and output dimensions respectively, $\phi : \mathbb{R} \to \mathbb{R}$ a given activation function. Then we define a* ***Feed-Forward Layer*** *as a map*

$$\begin{aligned} FFN_{d_{in},d_{out},W,b,\phi} \ : \ & \mathbb{R}^{d_{in}} \to \mathbb{R}^{d_{out}} \qquad \textit{where } W \in \mathbb{R}^{d_{out}\times d_{in}}, b \in \mathbb{R}^{d_{out}} \\ & x \longmapsto \phi\left(W \cdot x + b\right) \end{aligned}$$

where W, the weight matrix and b, the bias vector are learned parameters. By partial broadcasting, we can apply this map to any tensor $x \in \mathbb{R}^{d_1\times\cdots\times d_{n-1}\times d_{in}}$ by appropriately broadcasting the dot product and implicitly assuming that the last dimension denotes d_{in}.

Like in layer normalization and softmax, the feed-forward layer will be applied as a matrix-valued map in the transformer by partial broadcasting. Sometimes this map is called a *fully-connected layer* or *feed-forward network* (FFN). Classical neural networks are often compositions of feed-forward layers followed by an output layer that makes some prediction, *layering* the single components together by composition. This is known as the *multi-layer perceptron* (MLP) [Rumelhart et al., 1985]. Building on this, we now define the *Feed-Forward Block* as a composition of two feed-forward layers in the following. This block is the one that's used in the transformer model.

Definition 2.17 (Feed-Forward Block)**.** *Let $d_{in}, d_{hidden}, d_{out} \in \mathbb{N}$ specify the given input, hidden and output dimensions respectively. Also let $\phi : \mathbb{R} \to \mathbb{R}$ be an activation function. Then we define the* ***Feed-Forward Block*** *as a parameterized map*

$$FFNBlock_{d_{in},d_{hidden},d_{out},W_1,b_1,W_2,b_2,\phi} : \mathbb{R}^{d_{in}} \to \mathbb{R}^{d_{out}}$$

with learned parameters

$$W_1 \in \mathbb{R}^{d_{hidden}\times d_{in}}, b_1 \in \mathbb{R}^{d_{hidden}}, W_2 \in \mathbb{R}^{d_{out}\times d_{hidden}}, b_2 \in \mathbb{R}^{d_{out}}$$

that maps any input $x \in \mathbb{R}^{d_{in}}$ as follows:

$$\begin{aligned} x \longmapsto FFNBlock(x) &:= FFN_{d_{hidden},d_{out},W_2,b_2,\hat{\phi}}(FFN_{d_{in},d_{hidden},W_1,b_1,\phi}(x)) \\ &= W_2 z + b_2 = W_2 \cdot \phi\left(W_1 x + b_1\right) + b_2 \end{aligned}$$

*In this context $z \in \mathbb{R}^{d_{hidden}}$ is called the **hidden representation** which is a possibly*[9] *non-linear transformation of x. Note that the second activation function is the identity i.e. $\hat{\phi}(x) = x$.*

A feed-forward block is a stack of two feed-forward layers, in which the first layer ideally learns a well-suited non-linear hidden representation which is then affinely transformed back to the embedding space. In BERT, this block is used with $d_{\text{in}} = d_{\text{out}} = d_{\text{m}}$, $d_{\text{hidden}} = 4d_m$ and $\phi = \text{GELU}$ (as defined in 2.7).

2.3 The Transformer Architecture in Detail

Extending our foundations, we now define the transformer-specific building blocks, especially the transformer block, which forms the basis for all transformer-based models including BERT.

2.3.1 Transformer-Specific Building Blocks

The defining innovation brought forward by the transformer architecture is certainly its attention mechanism [Vaswani et al., 2017]. This attention mechanism allows the model to dynamically "attend" to certain parts of the input sequence and move information in form of linearly projected embeddings within the sequence.

Definition 2.18 (Scaled Dot-Product Attention)**.** *Let $n_s, d_k, d_v, h \in \mathbb{N}$ be given positive integers. Then we define the **Scaled Dot-Product Attention** as a map as follows:*

$$\begin{aligned} Attention_{d_k,d_v,n_s} &: \mathbb{R}^{n_s \times d_k} \times \mathbb{R}^{n_s \times d_k} \times \mathbb{R}^{n_s \times d_v} \to \mathbb{R}^{n_s \times d_v} \\ &(Q, K, V) \longmapsto Softmax\left(\frac{QK^T}{\sqrt{d_k}}\right) V \end{aligned}$$

*Where the softmax function is **broadcasted along the first dimension** i.e., it is applied to each row of $\frac{QK^T}{\sqrt{d_k}}$ separately. The matrix $A = Softmax\left(\frac{QK^T}{\sqrt{d_k}}\right) \in \mathbb{R}^{n_s \times n_s}$ is called **attention matrix**.*

$$Attention_{d_k,d_v,n_s}(Q, K, V)_i = \sum_{j=1}^{n_s} A_{ij} V_j$$

*Since $\forall 1 \leq i, j \leq n_s : A_{ij} \in (0, 1)$, $\forall 1 \leq i \leq n_s : \sum_{j=1}^{n_s} A_{ij} = 1$ and $V_j \in \mathbb{R}^{d_v}$, the output is a weighted sum of the rows of V. We may interpret A as a dynamic weighting function, since its weights solely depend on Q and K. The **attention weight** A_{ij} shows how much of V_j was moved to the i-th row of the output matrix.*

[9]depending on ϕ since the identity is also a valid activation function

Attention matrices are always right-stochastic matrices and thus can be interpreted as transition matrices of Markov chains. One can then find the eigenvalues of that attention matrix, which in the Markov chain context, indicate the invariant distribution of the states in the chain. This has been tried in this work, but since it was not obvious what these invariant distributions mean in the context of the attention mechanism, I decided not to include it in the thesis. Still, further research in this direction is still plausible. Since attention matrices are used deterministically, in this thesis I prefer the interpretation that attention matrices describe relative information flow instead of a Markov chain. Further, every right-stochastic matrix has a spectral radius of 1, which ensures that during the mixing of embeddings within the sequence, the embeddings stay within the same numerical range as before. This provides good numerical stability of the attention mechanism.

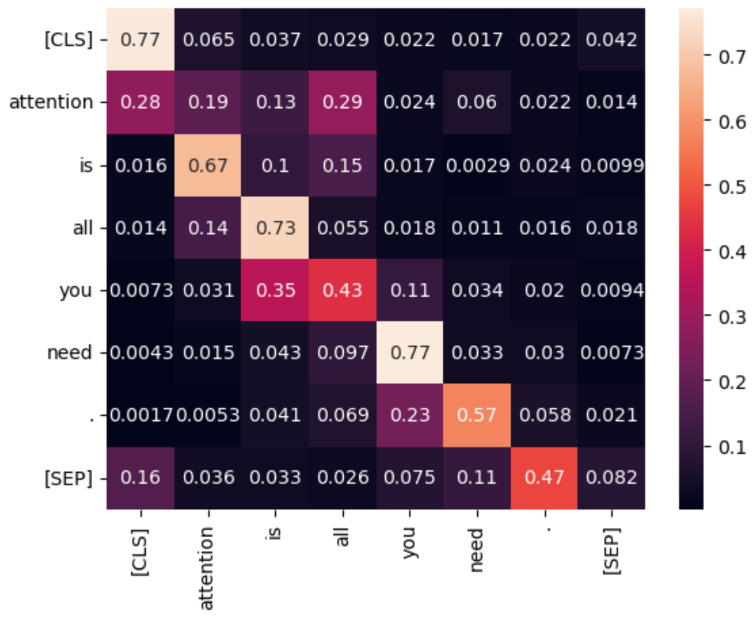

Figure 2: Example of an attention matrix as found in BERT visualizing a part of the attention mechanism on the sentence “Attention is all you need.“

In figure 2 we can see how a real attention matrix looks like within BERT when applied to the sentence “Attention is all you need.”. Note the high activations connecting the words “you” and “need” as well as the ones connecting “is” with “attention” and “all”. Visualizing this clearly leads us to believe the model understood the important context-defining words in this sentence.

It has been shown in [Wang et al., 2020] that attention matrices are often of lower rank than n_s, leading to their proposed **Linformer** which reduces these $O(n_s^2)$ dot-products to $O(n_s k)$ dot-products by computing a reduced attention matrix in $\mathbb{R}^{n_s \times k}$, as well as reduced values $V \in \mathbb{R}^{k \times d_v}$ for some fixed k.[10] In that case, not all of the tokens attend to each other but only to a linear transformation of the original embeddings. The approach is justified if tokens are not equally important in the sequence.

Q, K and V are usually called the **queries**, **keys**, and **values** respectively. Now observe that $QK^T \in \mathbb{R}^{n_s \times n_s}$ and subsequently each row of $\text{Softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)$ must sum to one. Further, imagine that in the k-th row, one of the n_s dot products in $(QK^T)_k$ is particularly large (relative to the others in that row), then the softmax of the k-th row will likely have one entry close to one while all the others are close to zero. If j is that entry close to one, then the k-th row of the output will consist mostly of V_j, thus information has been moved from the j-th row of V to the k-th row of the output.

Because of this property, the attention mechanism may function like a database lookup which can dynamically query information from any token matching a number of features to tokens matching a different number of features. The power is that at any point, the mechanism may query information from any other point in the sequence solely based on features in the relevant keys. Thus, it is able to quickly retrieve the necessary information to understand the current token correctly. Modeling long range dependencies, which is especially required for NLP, has always been a challenge for other neural network architectures [Vaswani et al., 2017].

One of the main advantages that come with this attention mechanism is that at its center lies a large parallelizable matrix multiplication, which is what most modern computer hardware is optimized for [Vaswani et al., 2017]. This enables transformers to be trained much faster than other models which possibly rely on sequential computations.

Definition 2.19 (Multi-Head Attention)**.** *Let $n_s, d_m, d_k, d_v, h \in \mathbb{N}$ be given positive integers. Then we define the* ***Multi-Head Attention*** *Mechanism as a matrix-valued map as follows*

$$MHA_{d_m,d_k,d_v,h,\theta_H,W_O} : \mathbb{R}^{n_s \times d_m} \to \mathbb{R}^{n_s \times d_m}$$

in which the learned parameters are

$$W_O \in \mathbb{R}^{hd_v \times d_m} \text{ and } \theta_H = \left(\left(W_Q^{(1)}, W_K^{(1)}, W_V^{(1)}\right), \dots, \left(W_Q^{(h)}, W_K^{(h)}, W_V^{(h)}\right)\right)$$
$$\text{such that for } 1 \le i \le h : W_Q^{(i)}, W_K^{(i)} \in \mathbb{R}^{d_m \times d_k} \text{ and } W_V^{(i)} \in \mathbb{R}^{d_m \times d_v}.$$

[10] They achieve this by projecting K and V to $\mathbb{R}^{k \times d_k}$ and $\mathbb{R}^{k \times d_v}$ respectively, and then querying from these reduced keys and values using $Q \in \mathbb{R}^{n_s \times d_k}$. After the projections we thus have $QK^T \in \mathbb{R}^{n_s \times k}$ and finally $\text{Softmax}(QK^T/\sqrt{d_k})V \in \mathbb{R}^{n_s \times d_v}$. This reduces the time and space complexity to $O(n_s k)$ at the expense of learning another projection matrix in $\mathbb{R}^{k \times n_s}$. The idea behind this approach is that not all of the n_s original keys and values are equally important for the queries.

Further, the actual map is defined as follows:

$$MHA : x \longmapsto Concat_2(H_1, \dots, H_h)W_O$$

Which concatenates the outputs of the h heads $H_1, \dots, H_h$ along their last dimension and then applies a linear transformation W_O. The H_i are defined as follows:

$$\begin{aligned} H_i &= Attention_{d_k,d_v,n_s}\left(x \cdot W_Q^{(i)}, x \cdot W_K^{(i)}, x \cdot W_V^{(i)}\right) \\ &= Softmax\left(\frac{\left(x \cdot W_Q^{(i)}\right) \cdot \left(x \cdot W_K^{(i)}\right)^T}{\sqrt{d_k}}\right) \cdot \left(x \cdot W_V^{(i)}\right) \end{aligned}$$

Regarding its input X as a sequence of vectors in $\mathbb{R}^{d_m}$, we notice that the Multi-Head Attention Mechanism is permutation-invariant, i.e., for any permutation $\pi \in \mathcal{S}_n$ of the set $\{1, \dots, n\}$ we have

$$\pi(\text{MHA}(X)) = \text{MHA}(\pi(X)).$$

Lemma 2.1 (Multi-head Attention is permutation-agnostic)**.** *Let $MHA_{d_m,d_k,d_v,h,\theta_H,W_O}$ be a Multi-Head Attention Mechanism, $X \in \mathbb{R}^{n_s \times d_m}$ and $\pi \in \mathcal{S}_{n_s}$ be a permutation of the set $\{1, \dots, n_s\}$. Then we have*

$$\pi\left(MHA(X)\right) = MHA(\pi(X)).$$

Proof: Without Loss of Generality, assume $\pi = \pi_{\hat{i}\hat{j}}$ be a permutation that swaps the $\hat{i}$-th and $\hat{j}$-th element of the sequence such that

$$\pi_{\hat{i}\hat{j}} : (x_1, \dots, x_{\hat{i}}, \dots, x_{\hat{j}}, \dots, x_{n_s}) \mapsto (x_1, \dots, x_{\hat{j}}, \dots, x_{\hat{i}}, \dots, x_{n_s})$$

and for any matrix $B \in \mathbb{R}^{n \times m}$ we have

$$\pi_{\hat{i}\hat{j}} : \begin{pmatrix} b_{11} & b_{12} & \dots & b_{1m} \\ \vdots & \vdots & \ddots & \vdots \\ b_{\hat{i}1} & b_{\hat{i}2} & \dots & b_{\hat{i}m} \\ \vdots & \vdots & \ddots & \vdots \\ b_{\hat{j}1} & b_{\hat{j}2} & \dots & b_{\hat{j}m} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \dots & b_{nm} \end{pmatrix} \mapsto \begin{pmatrix} b_{11} & b_{12} & \dots & b_{1m} \\ \vdots & \vdots & \ddots & \vdots \\ b_{\hat{j}1} & b_{\hat{j}2} & \dots & b_{\hat{j}m} \\ \vdots & \vdots & \ddots & \vdots \\ b_{\hat{i}1} & b_{\hat{i}2} & \dots & b_{\hat{i}m} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \dots & b_{nm} \end{pmatrix}$$

for any $1 \leq \hat{i}, \hat{j} \leq n$. Then we observe the following.

$$MHA(\pi_{\hat{i}\hat{j}}(X)) = Concat_2\left(H_1, \dots, H_h\right) \cdot W_O$$

where

$$\begin{aligned}
H_k &= \mathit{Softmax}\left(\frac{\left(\pi_{\hat{i}\hat{j}}(X)\cdot W_Q^{(k)}\right)\cdot\left(\pi_{\hat{i}\hat{j}}(X)\cdot W_K^{(k)}\right)^T}{\sqrt{d_k}}\right)\cdot\left(\pi_{\hat{i}\hat{j}}(X)\cdot W_V^{(k)}\right)\\
&= \underbrace{\mathit{Softmax}\left(\frac{\pi_{\hat{i}\hat{j}}\left(X\cdot W_Q^{(k)}\right)\cdot\left(\pi_{\hat{i}\hat{j}}\left(X\cdot W_K^{(k)}\right)\right)^T}{\sqrt{d_k}}\right)}_{\hat{A}^{(k)}}\cdot\left(\pi_{\hat{i}\hat{j}}\left(X\cdot W_V^{(k)}\right)\right)
\end{aligned}$$

where $\hat{A}^{(k)}$ denotes the permuted attention matrix for the k-th attention head. We can easily see that

$$\hat{A}^{(k)} = \pi_{\hat{i}\hat{j}}\left(\pi_{\hat{i}\hat{j}}\left(A^{(k)}\right)^T\right)^T$$

since in the dot product both the rows and columns of the queries and keys have been permuted. Finally, we can see that

$$\begin{aligned}
H_k &= \begin{pmatrix}
\hat{A}^{(k)}_{11} & \dots & \hat{A}^{(k)}_{1\hat{i}} & \dots & \hat{A}^{(k)}_{1\hat{j}} & \dots & \hat{A}^{(k)}_{1n_s}\\
\vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots\\
\hat{A}^{(k)}_{\hat{i}1} & \dots & \hat{A}^{(k)}_{\hat{i}\hat{i}} & \dots & \hat{A}^{(k)}_{\hat{i}\hat{j}} & \dots & \hat{A}^{(k)}_{\hat{i}n_s}\\
\vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots\\
\hat{A}^{(k)}_{\hat{j}1} & \dots & \hat{A}^{(k)}_{\hat{j}\hat{i}} & \dots & \hat{A}^{(k)}_{\hat{j}\hat{j}} & \dots & \hat{A}^{(k)}_{\hat{j}n_s}\\
\vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots\\
\hat{A}^{(k)}_{n_s1} & \dots & \hat{A}^{(k)}_{n_s\hat{i}} & \dots & \hat{A}^{(k)}_{n_s\hat{j}} & \dots & \hat{A}^{(k)}_{n_sn_s}
\end{pmatrix}\cdot\begin{pmatrix}
\hat{V}_{11} & \hat{V}_{12} & \dots & \hat{V}_{1d_v}\\
\vdots & \vdots & \ddots & \vdots\\
\hat{V}_{\hat{i}1} & \hat{V}_{\hat{i}2} & \dots & \hat{V}_{\hat{i}d_v}\\
\vdots & \vdots & \ddots & \vdots\\
\hat{V}_{\hat{j}1} & \hat{V}_{\hat{j}2} & \dots & \hat{V}_{\hat{j}d_v}\\
\vdots & \vdots & \ddots & \vdots\\
\hat{V}_{n_s1} & \hat{V}_{n_s2} & \dots & \hat{V}_{n_sd_v}
\end{pmatrix}\\
&= \begin{pmatrix}
A^{(k)}_{11} & \dots & A^{(k)}_{1\hat{j}} & \dots & A^{(k)}_{1\hat{i}} & \dots & A^{(k)}_{1n_s}\\
\vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots\\
A^{(k)}_{\hat{i}1} & \dots & A^{(k)}_{\hat{i}\hat{j}} & \dots & A^{(k)}_{\hat{i}\hat{i}} & \dots & A^{(k)}_{\hat{i}n_s}\\
\vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots\\
A^{(k)}_{\hat{j}1} & \dots & A^{(k)}_{\hat{j}\hat{j}} & \dots & A^{(k)}_{\hat{j}\hat{i}} & \dots & A^{(k)}_{\hat{j}n_s}\\
\vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots\\
A^{(k)}_{n_s1} & \dots & A^{(k)}_{n_s\hat{j}} & \dots & A^{(k)}_{n_s\hat{i}} & \dots & A^{(k)}_{n_sn_s}
\end{pmatrix}\cdot\begin{pmatrix}
V_{11} & V_{12} & \dots & V_{1d_v}\\
\vdots & \vdots & \ddots & \vdots\\
V_{\hat{j}1} & V_{\hat{j}2} & \dots & V_{\hat{j}d_v}\\
\vdots & \vdots & \ddots & \vdots\\
V_{\hat{i}1} & V_{\hat{i}2} & \dots & V_{\hat{i}d_v}\\
\vdots & \vdots & \ddots & \vdots\\
V_{n_s1} & V_{n_s2} & \dots & V_{n_sd_v}
\end{pmatrix}
\end{aligned}$$

where of course $A^{(k)} = \mathit{Softmax}\left(\frac{\left(X\cdot W_Q^{(k)}\right)\cdot\left(X\cdot W_K^{(k)}\right)^T}{\sqrt{d_k}}\right)$ and $V = X\cdot W_V^{(k)}$ are the attention matrix and the values of the unpermuted input sequence, *respectively. We may now see that $H_k(\pi_{\hat{i}\hat{j}}(X)) = \pi_{\hat{i}\hat{j}}(H_k(X))$ when regarding the k-th attention head's output as a function in X. The subsequent concatenation and matrix multiplication are also permutation agnostic since they are applied independently along the sequence dimension n_s. Thus we have that $MHA(\pi_{\hat{i}\hat{j}}(X)) = \pi_{\hat{i}\hat{j}}(MHA(X))$.*

Because we can construct any permutation of more than two elements π of multiple two-element permutations, we have proven the lemma. ■

This property counteracts the inductive biases[11] many other architectures imply on the data, however it also means that the model cannot detect any ordering in the input data since it essentially sees a bag of items. For example, convolutional neural networks which have also been used in NLP tasks, are sensitive to the order of the input sequence [Conneau et al., 2016].

In some models, like BERT, a slight variation of the classical Multi-Head Attention mechanism is used that masks tokens to prohibit backward-facing information flow, i.e., at each point in the input sequence, masked MHA can only attend to previous items up to and including the current position. This is easily achieved by adding a masking matrix point-wise to the attention matrix before applying the Softmax.

Definition 2.20 (Masked Multi-Head Attention)**.** *Let MHA_θ be a multi-head attention mechanism with parameters as defined before. Then we define* ***masked multi-head attention*** *as follows:*

$$MaskedMHA_\theta : \mathbb{R}^{n_s \times d_m} \to \mathbb{R}^{n_s \times d_m}$$

with learned parameters

$$\theta = (\{W_Q^{(i)}, W_K^{(i)}, W_V^{(i)}\}_{i=1}^h, W_O)$$

that maps any input sequence $X \in \mathbb{R}^{n_s \times d_m}$ as follows:

$$\begin{aligned} MaskedMHA_\theta(X) &= MHA_\theta(X, X, X) \\ &= Concat_2(H_1(X), \ldots, H_h(X)) \end{aligned}$$

Now the difference to the classical MHA lies in the calculation of H_i:

$$H_i(X) = Softmax\left(\frac{\left(X \cdot W_Q^{(i)}\right) \cdot \left(X \cdot W_K^{(i)}\right)^T}{\sqrt{d_k}} + M \right) \cdot \left(X \cdot W_V^{(i)}\right)$$

[11]an inductive bias is a prior assumption that a model implicitly makes about the data and the problem it aims to solve; E.g., if a model clusters data points into groups of points that are close to each other, it implicitly assumes that nearby points have more similar features than distant points. Inductive biases sometimes set barriers for models to learn specific types of tasks while introducing errors which are not always obvious to the modeler.

where $M \in \{0, \Lambda\}^{n_s \times n_s}$ is a masking matrix.

$$M_{ij} = \begin{cases} 0 & \textit{if } j \leq i \\ \Lambda & \textit{otherwise} \end{cases}$$

Λ is, technically also a hyperparameter, a large negative number, commonly $\Lambda = -10^9$ is used. This way, the $\exp(\cdot)$ afterwards in the Softmax will map any entry $j > i$ very close to zero, since Λ was added to it. Notice that this way, each row still sums to one while making each attention only able to attend to previous tokens.

The idea is to incorporate a natural "reading direction" into the model, which might be beneficial for generative modeling. This masked MHA is the attention mechanism used in the famous GPT series [Radford et al., 2019, Brown et al., 2020]. To incorporate positional information into the model, positional encoding needs to be added explicitly to the embeddings before they are fed to the model.

Definition 2.21 (Positional Encoding)**.** *Let $n_s, d_m \in \mathbb{N}$ be given positive integers. Then we define the **Positional Encoding** as a map as follows:*

$$\begin{aligned} PEncoding_{d_m, n_s} : \mathbb{R}^{n_s \times d_m} &\to \mathbb{R}^{n_s \times d_m} \\ X &\longmapsto X + p \\ p_{i,j} &= \begin{cases} \sin\left(i \cdot 10000^{-\frac{j}{d_m}}\right) & \textit{if } j \textit{ is even} \\ \cos\left(i \cdot 10000^{-\frac{j-1}{d_m}}\right) & \textit{if } j \textit{ is odd} \end{cases} \end{aligned}$$

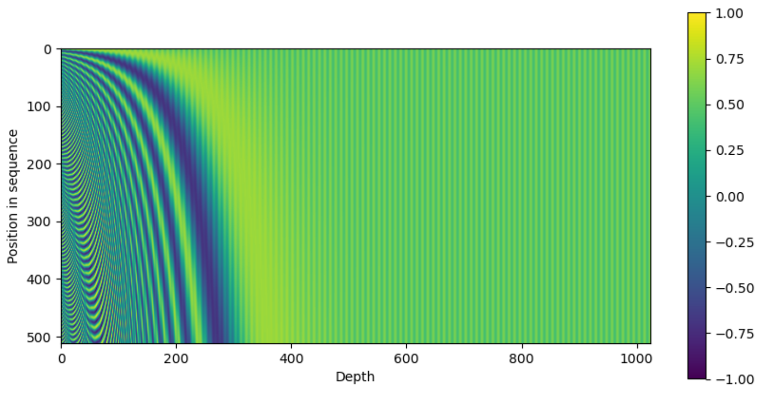

Figure 3: Example positional encoding (p) for $n_s = 512$ and $d_m = 1024$ as is found in the original Transformer paper

Above, we have introduced the positional encoding from the original transformer paper [Vaswani et al., 2017]. BERT for example uses a learned positional encoding, which they call **positional embedding**, where p is initialized randomly and then learned during training.

Definition 2.22 (Positional Embedding)**.** *Let $n_s, d_m \in \mathbb{N}$ be given positive integers. Then we define the **Positional Embedding** as a map as follows:*

$$\begin{aligned} PEmbedding_{d_m, n_s, \theta_P} : \mathbb{R}^{n_s \times d_m} &\to \mathbb{R}^{n_s \times d_m} \\ X &\longmapsto X + \theta_P \end{aligned}$$

where $\theta_P \in \mathbb{R}^{n_s \times d_m}$ is a learned parameter which is initialized with samples from $\mathcal{N}(0, 1)$.

The main problem positional embedding faces is that the resulting transformer can not be applied to token sequences longer than n_s, whereas the standard transformer may be applied to sequences of arbitrary length since we can simply generate $p_{i,j}$ for any $i > n_s$. However, positional embedding may still find a more optimal way of encoding positional information since it is learned during training. Further, there are also more sophisticated ways of encoding positional information which will not be discussed here [Su et al., 2021]. To visualize positional encoding, an example of what p looks like is shown in Figure 3.

As we can see, the classical transformer encodes position using sinusoids of different frequencies. This encoding has the special property that for any $k \in \mathbb{N}$ there exists a unique linear transformation $T_k \in \mathbb{R}^{d_m \times d_m}$ that maps every p_i close to p_{i+k}. The authors hypothesized that using this encoding, the transformer can learn to attend to relative positions in the sequence using one of the linear transformations in the MHA.

2.4 Transformer Architectures

Finally, we are able to define the basic building block of various transformer models, the **Transformer Block** which essentially composes, using layer normalization in between, the MHA and the FFN Block. The FFN Block enables the transformer to perform additional point-wise processing after the MHA was applied. To improve numerical stability, especially since the transformer blocks should be stacked very deep, layer normalization is always used when applying a layer. For any sublayer f, we then have

$$y = \text{LayerNorm}(x + f(x))$$

such that y is the input to the next layer. These residual connections make sure that gradients do not vanish or explode when stacking the transformer blocks very deep[12].

Definition 2.23 (Transformer Block)**.** *Let $n_s, d_m, d_{ffn}, h, d_k, d_v \in \mathbb{N}$ be given positive integers and $\phi : \mathbb{R} \to \mathbb{R}$ an activation function. Then we*

[12]GPT-3 for example consists of 96 stacked transformer blocks each of which consists of $h = 96$ attention heads of dimension $d_k = 128$ and a model dimension $d_m = 12,288$.

*define the **Transformer Block** as the following map:*

$$\begin{aligned} \textit{Transformer Block}_\theta : \mathbb{R}^{n_s \times d_m} &\longrightarrow \mathbb{R}^{n_s \times d_m} \\ x &\longmapsto LN_{\theta_1}\left(FFNBlock_{\theta_2}\left(LN_{\theta_3}\left(x + MHA_{\theta_4}(x,x,x)\right)\right)\right) \end{aligned}$$

Where $\theta := (\theta_1, \theta_2, \theta_3, \theta_4)$ *is shorthand for the parameters of the relevant sublayers and LN is short for layer normalization (see 2.15) such that all parameters are*

$$\begin{aligned} &\theta_1 = (\gamma_1, \beta_1, \varepsilon_1) && \text{(LayerNorm \# 1)}, \\ &\theta_2 = (d_m, d_{hidden}, d_m, W_1, b_1, W_2, b_2, \phi) && \text{(FFNBlock)}, \\ &\theta_3 = (\gamma_2, \beta_2, \varepsilon_2) && \text{(LayerNorm \# 2)}, \\ &\theta_4 = (d_m, d_k, d_v, h, \theta_H, W_O) && \text{(MHA)} \end{aligned}$$

as defined before. The addition $x + Sublayer(x)$ *is called a **residual connection** since it leaves some residual input to the output.*

Residual connections help keep gradients stable when stacking neural networks very deep [He et al., 2016]. Since we have now defined its basic building block, we can now define BERT.

Definition 2.24 (BERT)**.** *Let* $n_s, n_l, d_m, d_{ffn}, h, d_k, d_v \in \mathbb{N}$ *be given positive integers such that* $h = \frac{d_m}{d_k}$, $d_k = d_v$ *and of course* d_k *divides* d_m. *Then we define **BERT** as the following map:*

$$\begin{aligned} BERT_\theta \ &: \mathbb{R}^{n_s \times d_m} \longrightarrow \mathbb{R}^{n_s \times d_m} \\ &x \longmapsto E \circ (T_{\theta_1} \circ T_{\theta_2} \circ \cdots \circ T_{\theta_{n_l}})(x) \end{aligned}$$

Where $E = \textit{Token Embedding}_{\theta_E} \circ \textit{Positional Embedding}_{\theta_P}$ *is the first embedding layer and* T_{θ_i} *is the i-th transformer block in the stack. For layer* $1 \leq l \leq n_l + 1$*, we denote the **hidden state*** $X^{(l)} \in \mathbb{R}^{n_s \times d_m}$ *as follows:*

$$X^{(l)} = \begin{cases} E(x) & \text{if } l = 1, \\ (E \circ T_{\theta_1} \circ \cdots \circ T_{\theta_{l-1}})(x) & \text{otherwise.} \end{cases}$$

And for attention head $1 \leq k \leq h$ *on layer* $1 \leq l \leq n_l$*, we denote that attention head's attention matrix with* $A^{(l,k)} \in \mathbb{R}^{n_s \times n_s}$ *as is obtained from the relevant MHA.*

Figure 4 gives an overview of how BERT works when it is applied to classification tasks, in which an entire sentence, or multiple sentences, are mapped to one class. In that task only the left-most hidden state $X^{(n_l+1)}$, which represents the embedded *'[CLS]'* token, is used as the entire sentence's numerical representation. In the center of the figure we can see how BERT's attention mechanism connects every token to every other token in the sentence.

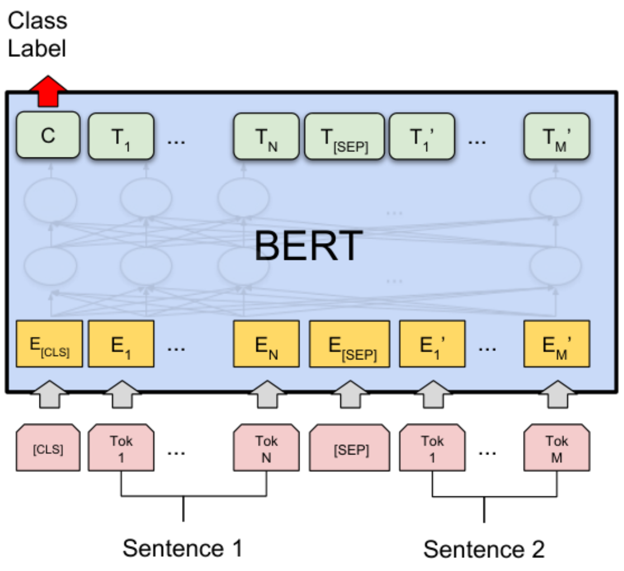

Figure 4: The BERT architecture and how it can be used in classification tasks

2.5 Dimensionality Reduction Techniques

In this thesis, I mainly extend the t-SNE algorithm so it can be used to visualize contextualized information as found when running actual text through BERT. To lay the foundation for this extension, the basic t-SNE is first defined and explained in detail so as to give the reader a good understanding of the algorithm. Since the embeddings learned by BERT, or other transformers, are very high-dimensional, we need ways to visualize them in order to understand what the models learned. A very successful algorithm used for this purpose is **t-distributed Stochastic Neighbor Embedding** (t-SNE), first introduced in [Van der Maaten and Hinton, 2008], which projects our embeddings from $\mathbb{R}^{d_m}$ to $\mathbb{R}^2$ so we can easily visualize them using a scatter plot. Further, t-SNE retains much of the original structure of the high-dimensional data such that the visualizations give us a good understanding of how embeddings are actually placed in $\mathbb{R}^{d_m}$. t-SNE is an improved version of the previously proposed Stochastic Neighbor Embedding (SNE) [Hinton and Roweis, 2002]. Note that t-SNE finds a non-parametric map, so after running the algorithm we do not yield a transformation to the low-dimensional space, as is sometimes the case for other dimensionality reduction techniques[13].

[13]There are modifications of t-SNE that do yield a parametric map, which we will not discuss here, see [Van Der Maaten, 2009].

2.5.1 t-SNE in Detail

Later, an extension to t-SNE is proposed based on the representations learned within the transformer model, specifically its attention heads, which allows us to visualize contextualized information in a meaningful way. Also, in this section we will be using the notation from the original paper to make clear exactly how the algorithm has been used here.

Given a set of data points $\mathcal{X} = \{x_1, x_2, \ldots, x_n\}$ where $x_i \in \mathbb{R}^{d_m}$, t-SNE tries to find low-dimensional representations $\mathcal{Y} = \{y_1, y_2, \ldots, y_n\}$ where $y_i \in \mathbb{R}^{d_c}$ such that the y_i preserve much of the local as well as global structure[14] of our original high-dimensional data $\mathcal{X}$. In our case, we set $d_c = 2$.

In a first step, for each pair of data points x_i and x_j, t-SNE computes a conditional probability $p_{j|i}$ that represents "the probability that x_i would choose x_j as its neighbor if neighbors were chosen using a normal distribution centered at x_i". This conditional probability is computed as follows:

$$p_{j|i} = \frac{\exp\left(\frac{-\|x_i - x_j\|^2}{2\sigma_i^2}\right)}{\sum_{k \neq i} \exp\left(\frac{-\|x_i - x_k\|^2}{2\sigma_i^2}\right)} \quad \text{if } j \neq i \text{ and} \quad p_{j|j} = 0 \text{ for } 1 \leq j \leq n \tag{1}$$

Note that each σ_i induces an individual probability distribution P_i over all other data points from the data set. How exactly this σ_i is chosen is not relevant here, for an exact explanation see Appendix A.1. One might imagine that this σ_i is chosen such that the data point x_i has a certain number of neighbors which is specified by the perplexity parameter by the user. These neighbors have a high probability compared to the other data points since the gaussian nature of the distribution quickly assigns very low probabilities to data points that lie further away.

Also notice that because each data point gets its own probability distribution, the conditional probabilities are not symmetric (i.e., $p_{j|i} \neq p_{i|j}$). Classical Stochastic Neighbor Embedding (SNE) [Hinton and Roweis, 2002] uses this asymmetric matrix of conditional probabilities directly whereas the improved t-SNE algorithm symmetrizes this matrix to yield a symmetric matrix of joint probabilities as follows:

$$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2n} \qquad \text{and} \qquad p_{ii} = 0 \text{ for } 1 \leq i \leq n$$

Notice here that in the matrix of conditional probabilities with entries $p_{j|i}$ (j-th row and i-th column), each column sums to one so that the entire matrix sums to n. For this reason, we need to scale $(p_{j|i} + p_{i|j})$ by $\frac{1}{2n}$ to make all p_{ij} sum to one as we would expect with joint probabilities.

Subsequently, the algorithm initializes the low-dimensional representations y_i with samples from $\mathcal{N}(0, 10^{-4} \cdot I_{d_c})$ where I_{d_c} denotes the identity

[14]structure in the sense of the pairwise distances between the data points

matrix in $\mathbb{R}^{d_c \times d_c}$. t-SNE now computes another matrix of joint probabilities q_{ij} which model the probability that y_i chooses y_j as a neighbor if y_i's neighbors in the low-dimensional space are chosen following a Student-t distribution with one degree of freedom centered at y_i. In this case, we naturally have $q_{ij} = q_{ji}$ given since the Student-t distribution only has one parameter and thus each data point uses the same probability distribution for choosing its neighbors. We thus have:

$$q_{ij} = \frac{(1 + \|y_i - y_j\|^2)^{-1}}{\sum_{k \neq l}(1 + \|y_k - y_l\|^2)^{-1}} \quad \text{if } i \neq j \text{ and} \quad q_{ii} = 0 \text{ for } 1 \leq i \leq n \tag{2}$$

The algorithm now tries to minimize the Kullback-Leibler Divergence between Q and P, where Q denotes the matrix of q_{ij} and P the matrix with entries p_{ij}. A low KL Divergence indicates that the probabilities q_{ij} model the original p_{ij} well, i.e., if x_i likely chooses x_j as its neighbor so should y_i also chose y_j as its neighbor in the low-dimensional space with the same likelihood. The KL Divergence can finally also be used as a measure of the quality of the low-dimensional embedding and thus the visualizations. This problem is now formulated as minimizing the following cost function C:

$$C := KL(P\|Q) = \sum_{i=1}^{n}\sum_{j=1}^{n} p_{ij} \log\left(\frac{p_{ij}}{q_{ij}}\right)$$

If we now plug in the formulas[15] for p_{ij} and q_{ij}, we will see that C is differentiable with respect to y_i with the following gradient.

$$\frac{\partial C}{\partial y_i} = \nabla_{y_i} C = 4 \sum_{j=1}^{n} \frac{(p_{ij} - q_{ij})(y_i - y_j)}{1 + \|y_i - y_j\|^2} \tag{3}$$

Using a gradient descent[16] algorithm, the y_i are then updated to minimize the cost function C. This yields the final algorithm as defined below:

Algorithm 2.1. *t-Distributed Stochastic Neighbor Embedding (t-SNE)*
Data: *set of data points* $\mathcal{X} = \{x_1, x_2, \ldots, x_n\}$,
parameters: perplexity Perp, number of iterations T, *number of low-dimensional components* d_c, *learning rate* η, *momentum* $\alpha(t)$
Result: *low-dimensional data representation* $\mathcal{Y}^{(T)} = \{y_1, y_2, \ldots, y_n\}$

```
1: procedure t-SNE($\mathcal{X}$)
```

[15] See the original t-SNE paper's appendix for a full derivation of this gradient [Van der Maaten and Hinton, 2008, Appendix A, Derivation of the t-SNE gradient].

[16] Gradient Descent is a family optimization techniques which iteratively update parameters θ of a function f_θ to minimize some predefined loss function $L(f_\theta, y)$ using the negative gradient $\nabla_\theta L(f_\theta, y)$ of the loss function with respect to the parameters θ. In our case, we simply have $f_\theta(x_i) = y_i, \forall 1 \leq i \leq n$ and the loss, or cost function, is C.

```
2:  for $i \in \{1, \ldots, n\}$ do
3:      $\sigma_i \leftarrow$ FindStandardDeviation($\mathcal{X}$, Perp)
4:      for $j \in \{1, \ldots, n\}$ do
5:          compute $p_{j|i}$ using equation 1
6:      $p_{i|i} \leftarrow 0$
7:  for all $i, j \in \{1, \ldots, n\}$ do
8:      $p_{ij} \leftarrow \frac{1}{2n} \cdot (p_{j|i} + p_{i|j})$          ▷ This is our matrix P
9:  sample $\mathcal{Y}^{(0)} = \{y_1^{(0)}, y_2^{(0)}, \ldots, y_n^{(0)}\}$ from $\mathcal{N}(0, 10^{-4} \cdot I_{d_c})$
10: for $t \in \{1, \ldots, T\}$ do
11:     for all $i, j \in \{1, \ldots, n\}$ do
12:         compute $q_{ij}$ using equation 2
13:     for $i \in \{1, \ldots, n\}$ do
14:         compute gradient $\nabla_{y_i} C(y_i^{(t-1)})$ using equation 3
15:         if $t = 1$ then
16:             $y_i^{(t)} = y_i^{(t-1)} - \eta \nabla_{y_i} C(y_i^{(t-1)})$
17:         else
18:             $y_i^{(t)} = y_i^{(t-1)} - \eta \nabla_{y_i} C(y_i^{(t-1)}) + \alpha(t)(y_i^{(t-1)} - y_i^{(t-2)})$
19: return $\{y_1^{(T)}, y_2^{(T)}, \ldots, y_n^{(T)}\}$
```

Please see the Appendix A.1 for details how σ_i is computed.

Note that the last term $\alpha(t)(y_i^{(t-1)} - y_i^{(t-2)})$ adds the scaled previous update of y_i to the current one. This speeds up the convergence of the algorithm if the current and the last gradient point in the same direction. This technique is called **momentum** and the parameter $\alpha(t) \in [0, 1]$ controls the impact at which momentum speeds up the gradient descent algorithm. It is also common to decay the momentum term to achieve larger updates of y_i in the beginning and smaller ones near the end of the optimization, for this reason α is defined as a function in t here. This momentum technique introduces an exponentially decaying moving average of historical updates as follows:

$$\begin{aligned} y_i^{(t)} &= y_i^{(t-1)} - \eta \nabla_{y_i} C(y_i^{(t-1)}) + \alpha(t)(y_i^{(t-1)} - y_i^{(t-2)}) \\ &= y_i^{(t-1)} - \eta \nabla_{y_i} C(y_i^{(t-1)}) + \alpha(t)(-\eta \nabla_{y_i} C(y_i^{(t-2)}) + \alpha(t-1)(y_i^{(t-2)} - y_i^{(t-3)})) \end{aligned}$$

Which turns out to be

$$y_i^{(t)} - y_i^{(t-1)} = \eta \left(-\nabla_{y_i^{(t-1)}} C - \alpha(t) \nabla_{y_i^{(t-2)}} C - \alpha(t)\alpha(t-1) \nabla_{y_i^{(t-3)}} C - \cdots \right)$$

if using the shorthand notation $\nabla_{y_i^{(t-1)}} C$ for the gradient $\nabla_{y_i} C(y_i^{(t-1)})$. Consider now that if in the beginning of the optimization, gradients are very noisy but somehow all point in a similar direction, this common direction quickly becomes the dominant term in the updates.

3 The Algorithm

As we now have defined all the relevant building blocks, we can finally proceed to the main part of this thesis, which is to understand how the model processes text data and what exactly it learns from it.

Further, an extensions to the t-SNE algorithm is proposed in this section which can be used to better understand how the attention mechanism works and specifically what each attention head learns, since this novel algorithm can be used to visualizes contextual information, which has not been possible before. Also, we will be looking at metrics we can use to unveil the attention mechanism as it processes text data.

3.1 Attention Head based t-SNE

t-SNE as defined in section 2 assumes, in the neighbor embedding context, that neighbors are chosen using normal distributions with diagonal covariance matrices (i.e., each x_i chooses its neighbors using $\mathcal{N}(x_i, \sigma_i \cdot I_{d_m})$). Given our knowledge that the transformer heavily uses dot products to detect and process features, this assumption might absolutely not hold for the model's learned embeddings, which possibly produces suboptimal results using vanilla t-SNE.

At a basic level, t-SNE only uses this normal distribution to calculate the neighboring probabilities p_{ij} from the euclidian distances. The transformer however implicitly learns to produce a right-stochastic matrix for each attention head on each layer. We can thus assume that given a set of embedded tokens $\mathcal{X} = \{x_1, x_2, \ldots, x_n\}$, we may use the attention mechanism to calculate the neighboring probabilities $p_{j|i}$ directly as the entries of the attention matrix. Artificially setting the diagonal elements to zero, symmetrizing its entries and normalizing the matrix such that it sums to one, as is done in t-SNE, gives us a joint probability matrix p_{ij}.

$$p_{ij} = \frac{(1 - \delta_{ij}) \cdot (A_{ij}^{(k,l)} + A_{ji}^{(k,l)})}{\sum_{m=1}^{n_s} \sum_{n=1}^{n_s} (1 - \delta_{nm}) \cdot (A_{nm}^{(k,l)} + A_{mn}^{(k,l)})}$$

Where $A^{(k,l)} \in [0, 1]^{n_s \times n_s}$ denotes the attention matrix of the k-th attention head on the l-th layer and $\delta_{ij} = 1$ if $i = j$ and 0 otherwise, denotes the Kronecker delta function. Although this algorithm theoretically also works for any set of tokens, I have found it to work much better using an actual piece of text as input, so that the tokens include positional encoding, and we can visualize the contextual information.

$$A^{(k,l)} = \begin{pmatrix} 0 & p_{2|1} & p_{3|1} & \cdots & p_{n|1} \\ p_{1|2} & 0 & p_{3|2} & \cdots & p_{n|2} \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ p_{1|n} & p_{2|n} & p_{3|n} & \cdots & 0 \end{pmatrix}$$

Note that the probability $p_{j|i}$, that embedding x_i chooses to embed x_j as its neighbor, can now be interpreted[17] as the amount of information that flows from embedding x_j to embedding x_i (after projecting them with $W_V^{(k)}$ of course) and is certainly not symmetric.

After symmetrizing this matrix however, this probability becomes the average information flow from x_j to x_i and from x_i to x_j, which might not always be optimal. However, in the worst case one of the probabilities might be close to zero while the other one is larger, which results in the larger probability being about halved. Nevertheless, even in this case the final joint probability will still indicate some information flow after all.

A great advantage this algorithm has over the standard t-SNE is that the p_{ij} are simply calculate using euclidian distances but directly from relevant features from inside the transformer. This is advantageous since nearby embeddings, which would have a small euclidian distance, might certainly have distinguishing features, especially after context is added to the embeddings, and may thus unlikely be chosen as neighbors by the improved algorithm whereas the standard algorithm would place them close to each other.

4 Numerical Findings

In this section, we will first use the standard t-SNE algorithm to visualize embeddings and their resepective words. Finally, we will be looking at the visualizations produced by the proposed attention head based t-SNE algorithm and try to compare it to the standard t-SNE algorithm. Further, another simpler visualization technique is shown which can be used in combination with attention head based t-SNE to quickly discover the important attention heads in the model. In all of the following experiments, BERT[18] has been used since it is one of the model widely used NLP models [Wolf et al., 2019].

[17]or in the Markov Chain context, the probability of being in state x_j given that we are in state x_i at the previous time step

[18]Specifically bert-base-uncased from HuggingFace's transformers library [Wolf et al., 2019]

4.1 Standard t-SNE

To visualize t-SNE, we first need to define a way to embed text using BERT. In order to provide embeddings for multiple tokens, since single tokens are quite abstract in some cases, we define the multi-token embedding of a text in the following.

Definition 4.1 (Multi-Token Embedding using BERT)**.** *Let t be a text of arbitrary length and $X^{(l)} \in \mathbb{R}^{n_s \times d_m}$ the hidden state at layer l when applying BERT on t. Then we define the multi-token embedding of t at layer $l \in \{1, \dots, n_l + 1\}$ as the embedding of the '[CLS]' token on layer l. Then the multi-token embedding of t is defined as*

$$MTE_l(t) = X_0^{(l)} \in \mathbb{R}^{d_m}.$$

This multi-token embedding conveniently maps arbitrary-length text to a single vector in $\mathbb{R}^{d_m}$. This is advantageous if we want to embed a word which itself consists of multiple tokens (e.g., 'New York'). The same map can also be used to embed entire sentences or paragraphs.

Figure 6 shows a t-SNE visualization of embeddings obtained by the multi-token embedding algorithm applied on a set of words describing occupations. Notice the semantically meaningful clustering of occupations like 'musician', 'dancer', 'singer', 'choreographer' and 'composer' near $x = -4, y = 0$ as well as the clustering of 'woodworker', 'farmworker', 'rancher', 'millwright' and closely 'locksmith' and 'brickmason' near $x = 2, y = -7$. Figure 5 shows a zoomed version of the same plot.

This visualization shows that the model clearly learned relationships between different professions that reach beyond simple similarity of words which might even be obtained using a simple character-based similarity measure between words (like the Levenshtein Distance [Levenshtein, 1966]). Such a similarity measure would place 'millwright' far away from 'woodworker', whereas BERT clearly understands that both are related since they both are professions in the construction sector.

4.2 Attention Head based t-SNE in Practice

As defined in section 3, we present an extension to the standard t-SNE algorithm that allows us to visualize the attention heads of the transformer model.

I found the algorithm works best when visualizing longer pieces of text, rather than single sentences. In the following example it was applied to the abstract of the original t-SNE paper, the full text of which can be found in the Appendix A.4.

The main advantages of this algorithm are (1) it visualizes context as opposed to visualizations common of embeddings distributions and (2) it

./images/tsne_emb_zoomed_professions.png

Figure 5: A zoomed version of Figure 6, focusing only on the aforementioned professions

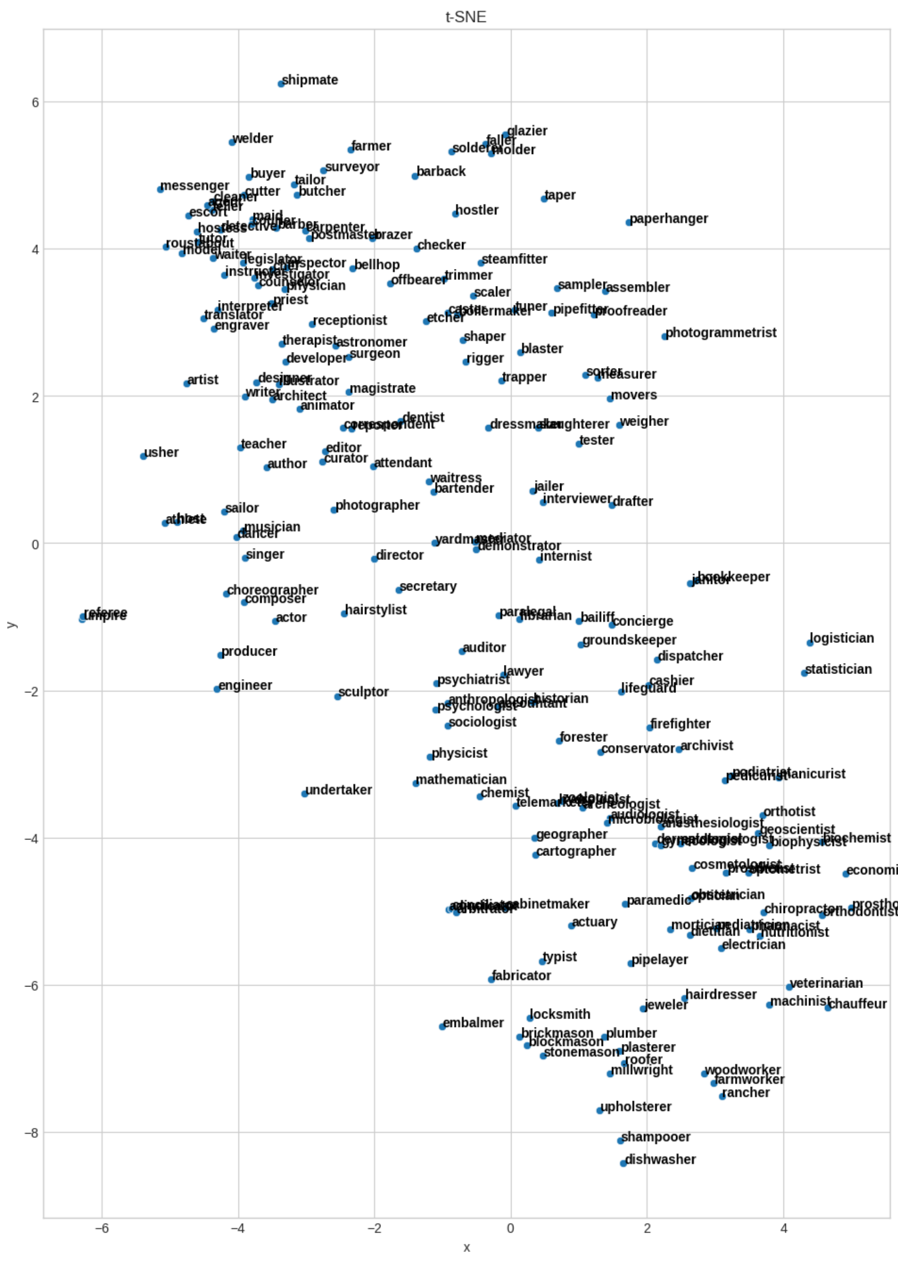
Figure 6: An example t-SNE visualization of multi-token embeddings for words describing occupations; To reduce the sample size, only words without spaces were considered.

allows us to look into the inner workings of each attention head at each layer of the transformer model. For some attention heads it becomes relatively clear to see what they are looking at, while for others it is quite tricky. Also, attention heads on higher layers tend to have worked out the significant ideas of the text, while attention heads on lower layers curate the details such as the ordering of words or work out the exact contextualized meaning of single words.

./images/layer_1_head_2_zoomed_part.png

Figure 7: A zoomed version of Figure 8

Figure 8 shows the visualizations of attention head 2 on layer 1 of BERT using attention head based t-SNE. Notice the groupings of the words 'linear', 'techniques', 'mapping' and 'locally' indicating that the attention head groups the central concepts employed in the paper near $(x = -0.5, y = -1.3)$. Also, near $(x = -1.5, y = -0.6)$, it grouped the words 'manifold', 'dimensional', 'objects', 'structure' and '##point'. Notice how sometimes, the same word is grouped with different words at different places indicating that the attention head is actually looking at the context of each word rather than the words themselves. At the same time, it also placed all the prepositions and words that do not give meaning to the text more spread out in the plot.

In Figure 9 we can clearly see that attention head 2 on layer 2 tries to dissect the text into logical statements that summarize it briefly. Specifically, it strongly attends to positional information mostly retaining the order of words to the level where parts of the text can still be read in the scatter plot. Further, it breaks the text apart such as to form chains of words that, in this case, describe the features of the t-SNE algorithm. Starting from the bottom left, near $(x = -20, y = -60)$, we can see the chain "produced by t-SNE are significantly better than those produced by the other techniques" and above

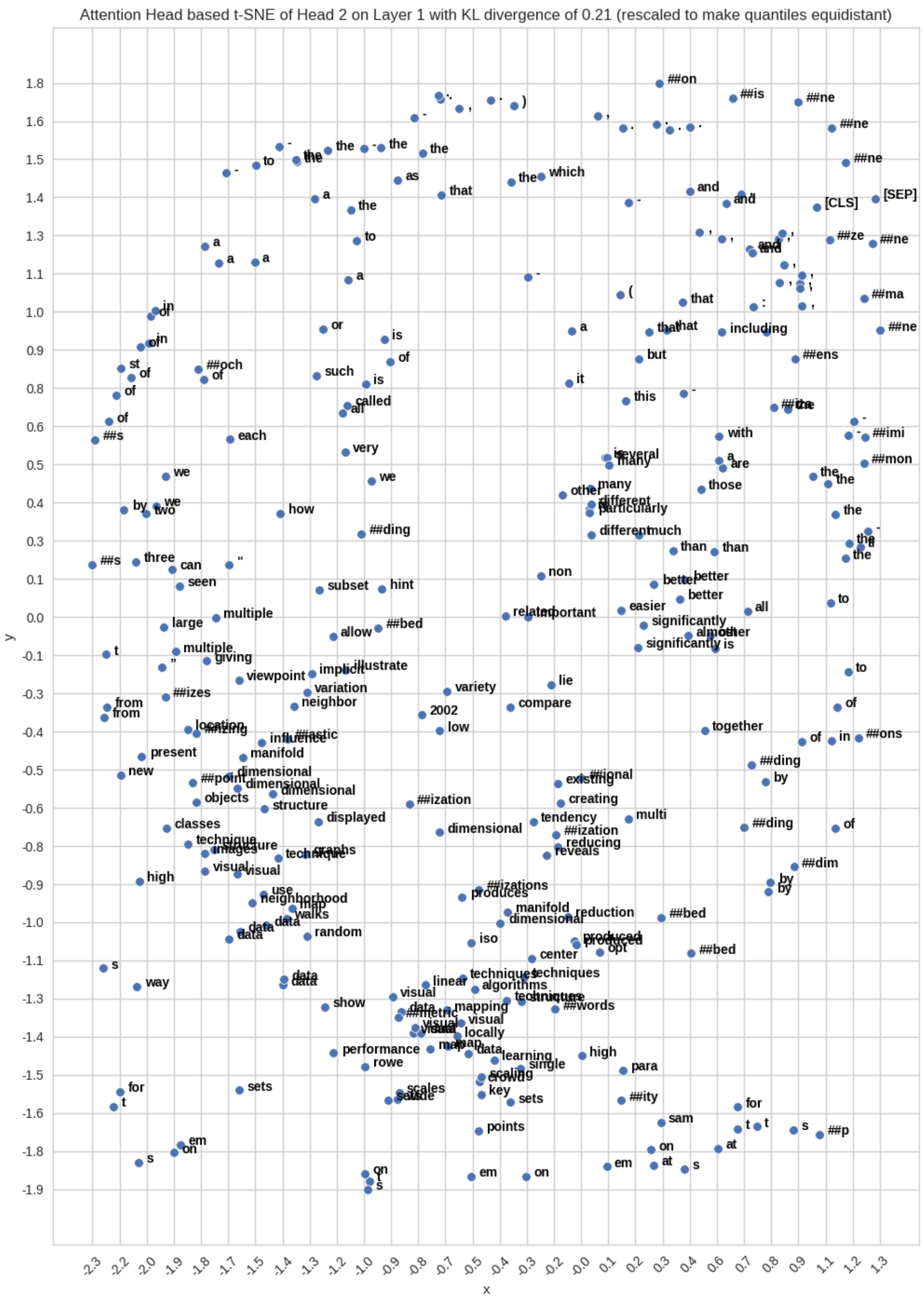

Figure 8: Visualizations of Attention Head 2 at Layer 1, applied to the abstract of the original t-SNE paper; axis have been rescaled making quantiles equidistant for readability, the original plot can be found in the Appendix 23; Details on quantile equidistant rescaling can be found in Appendix A.3

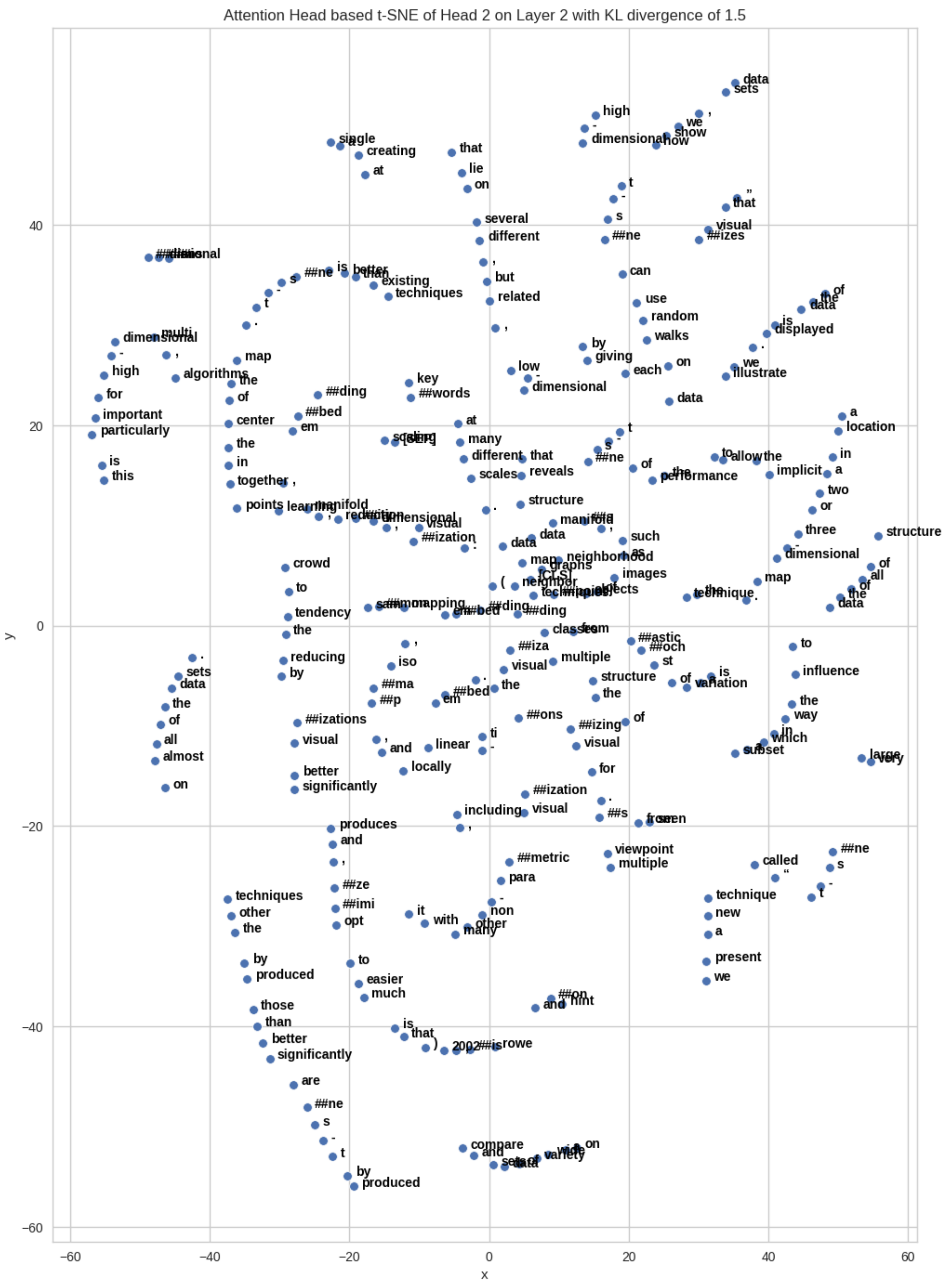

Figure 9: Visualizations of Attention Head 2 at Layer 2 with attention head based t-SNE

that, with some space we see “on almost all of the data sets.”. To the right of that chain, starting near ($x = -20, y = -40$), we see “that is much easier to optimize, and produces significantly better visualizations” and above that with a space, “by reducing the tendency to crowd”, and again with another space, “points together in the center of the map.”. The latter two chains refer to the crowding problem[19] the original SNE algorithm suffered from which has been addressed by the t-SNE algorithm. Notice that the spaces between the chains logically separate the different, but related, ideas presented in the text. For example, t-SNE produces better visualizations because it reduces the tendency to crowd points together in the center of the map. The last chain and the first one are really only connected through the chain in the middle which forms the relationship, this constellation is well shown using attention head based t-SNE. Throughout the plot, we can see more chains of words that describe certain ideas conveyed in the text, such as

- “t-sne can use random walks”, near ($x = 20, y = 40$),
- “a location in a two or three-dimensional map”, near ($x = 50, y = 20$),
- “we present a new technique called ”t-sne”, near ($x = 30, y = -35$).

Figure 10 shows that attention head 10 on layer 9 of BERT clearly incorporates much more structure[20] into its representations. Following the same observations from the previous figure, this visualization shows point clouds which represent the individual related ideas conveyed in the text, indicating that the previous attention chains have already been processed to form more complex structures. In the top right corner, we can see a large cluster consisting of multiple words, all of which are related to the original SNE algorithm, mainly these are

1. “stochastic neighbor embedding” and “variation”, near ($x = 0.7, y = 10$),
2. “hinton and roweis (2002)”, above and slightly to the left of the previous words, and
3. “reducing the”, “tendency to crowd points together”, and “center of the map”, near ($x = 13.1, y = 7.4$) describing SNE’s main drawback.

[19]The original SNE also assumed a normal distribution for choosing neighbors in the low-dimensional space, which led to the crowding problem in which all of the points formed a single cluster in the center of the map. This was due to the fact that KL divergence becomes very large if the embedding places two points y_i and y_j more far apart than their corresponding high-dimensional counterparts x_i and x_j were placed, i.e., $p_{ij} \log \frac{p_{ij}}{q_{ij}}$ becomes very large if p_{ij} is large and q_{ij} is small.

[20]as it forms more clusters which are distinct but placed in a way that shows relationships between them fairly well

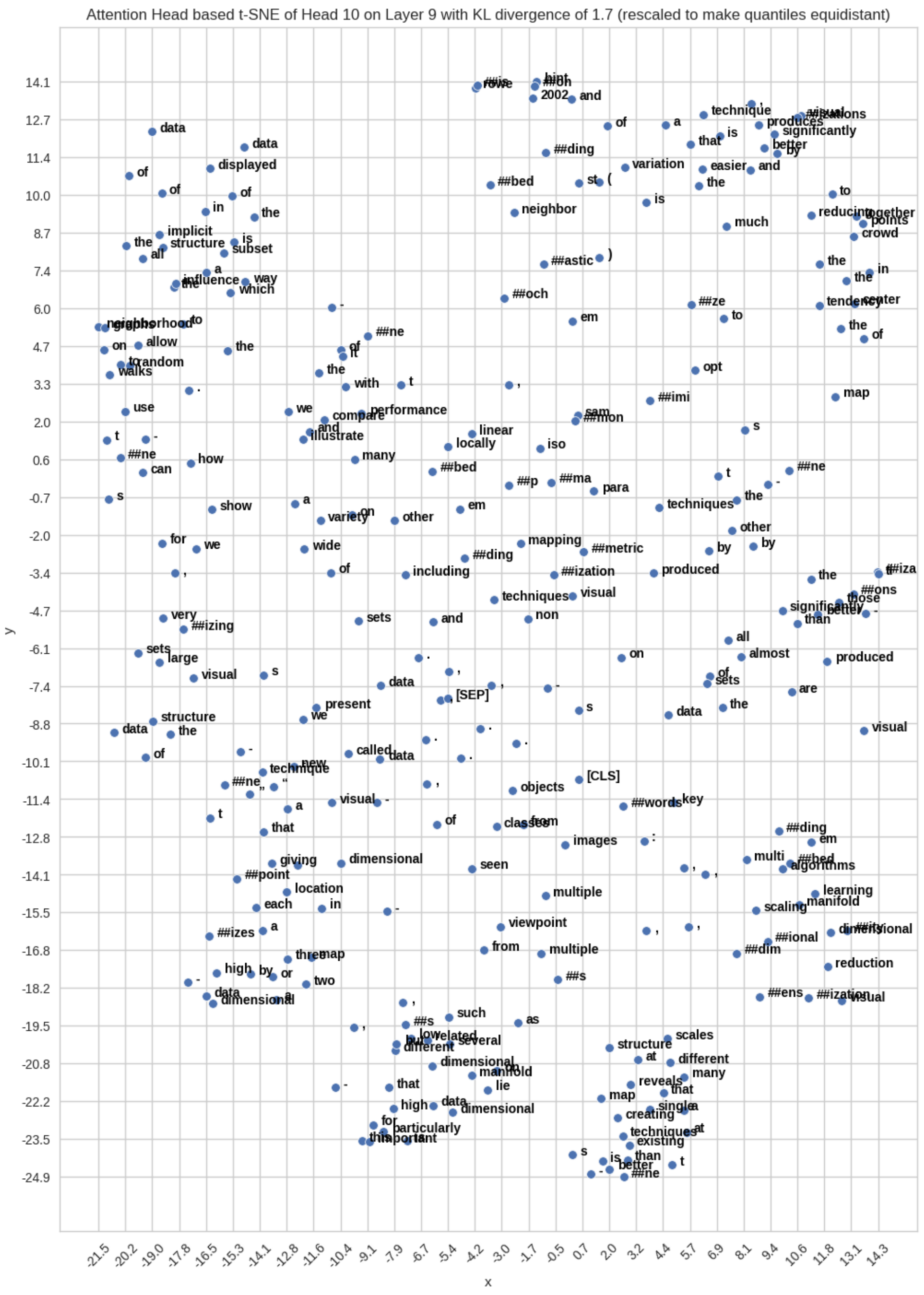

Figure 10: Visualizations of Attention Head 10 at Layer 9 produced by the attention head based t-SNE; axis have been rescaled making quantiles equidistant for readability, the original plot can be found in Appendix 24.

We can see that this head doesn't see the text as a bunch of attention chains anymore but rather as related ideas that are encoded in only a few of the defining words, e.g., reducing, tendency, crowd, points, etc.

4.2.1 Comparing Attention Head based t-SNE to Standard t-SNE

Firstly, we notice that the standard t-SNE algorithm cannot visualize any contextualized information it cannot find in the hidden states. However, judging from the regular t-SNE visualization of the hidden states at the last layer, as can be seen in Figure 11, we can see that the standard t-SNE algorithm does somehow find some contextualized information in that hidden state. Still, we see that since in the text the words appear multiple times, the regular t-SNE algorithm places them very close to each other indicating that BERT has incorporated little contextual information into the hidden state.

From a numerical standpoint, we can compare the KL divergences of the algorithms to evaluate how well they preserve the original distributions. In Figure 12 we can see that the attention head based t-SNE algorithm mostly has a higher KL divergence than the standard t-SNE algorithm on every but the first layer. This indicates that the low-dimensional embeddings found by the standard t-SNE better fit the distribution of the hidden states than those found by the attention head based t-SNE algorithm [21] We may also notice the much higher variance of the KL divergences in attention head based t-SNE.

We can now conclude that the new attention head based t-SNE algorithm allows us to look into the inner workings of the multi head attention mechanism of transformer models. For this reason, it can produce much better visualizations than the standard t-SNE algorithm. Further, it gives us the chance to try to understand how each attention head sees the text from a different perspective. However, it is often difficult to grasp what some attention heads do, especially as we cannot yet interpret how each attention head transforms[22] the sequence of tokens. Further visualizations can be found in Appendix A.5.

4.3 Visualizing Maximum Attention

The problem with the above approach is that it produces one plot per attention head, which in the case of BERT yields 12 plots per layer on 12

[21]Representations within the attention heads may possibly be more sophisticated than those in the hidden states, which may explain why t-SNE is able to find a better fit for the hidden states.

[22]since for each attention head the $W_V^{(k)}$ is different and thus different features are moved within the sequence

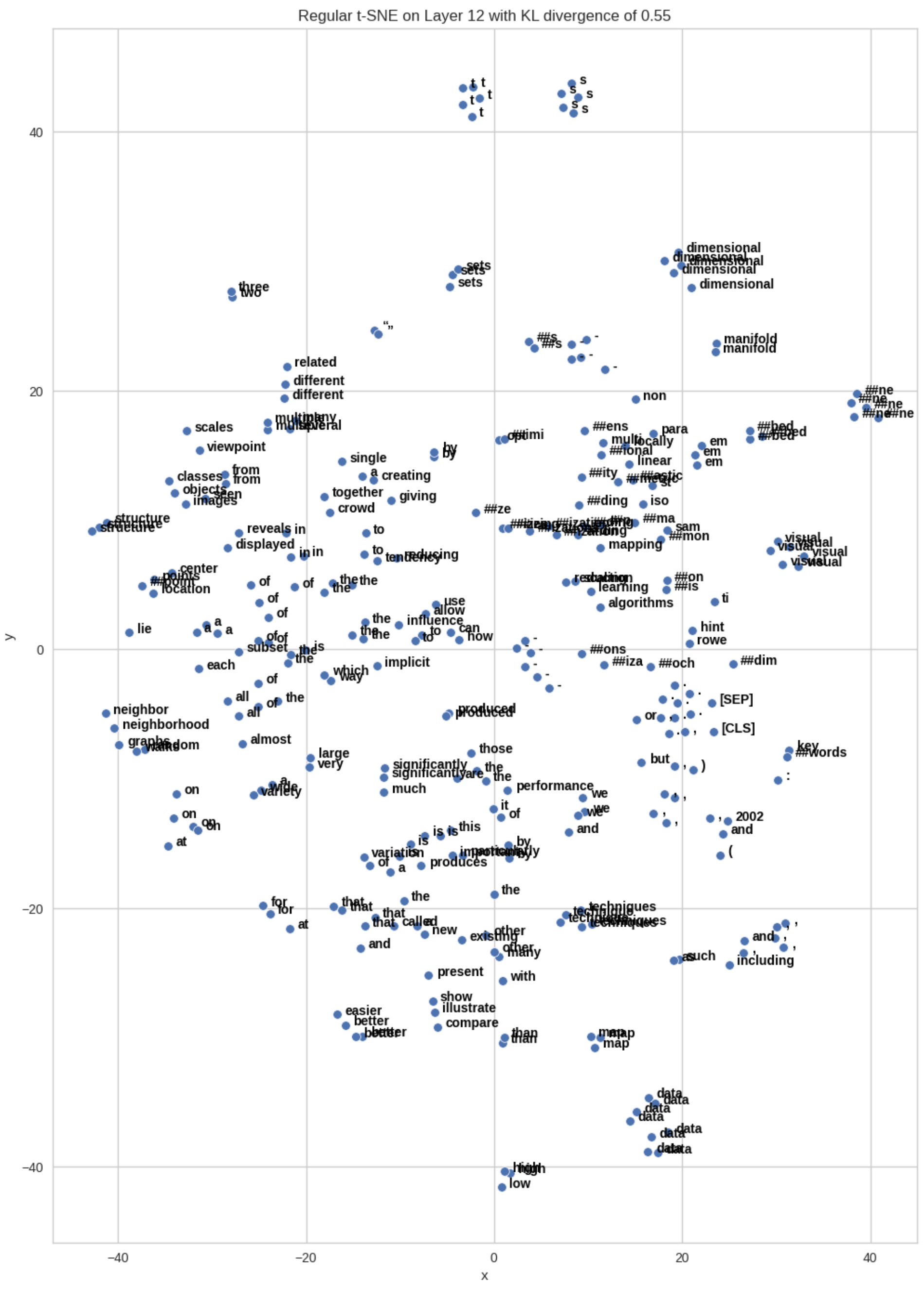

Figure 11: Visualizations of the hidden states at Layer 12 produced by the standard t-SNE algorithm; Details on the parameters can be found in Appendix A.2

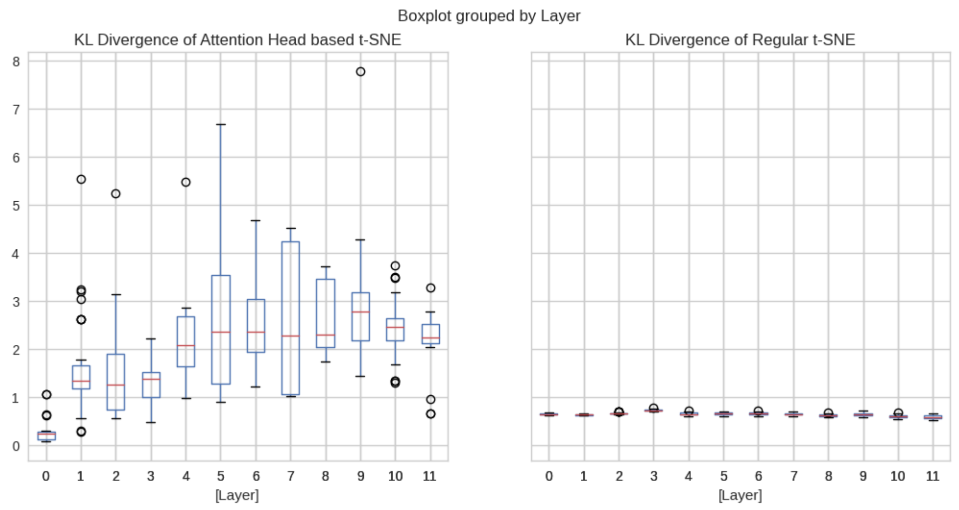

Figure 12: Boxplots showing the distributions of the KL divergence of the attention head based t-SNE and the standard t-SNE algorithm applied to the hidden states or the attention heads on each layer; Layers are 0-indexed in this plot; Both algorithms were run 100 times

layers, and inspecting 144 plots is inconvenient when trying to understand what the attention mechanism is processing. To overcome this problem, one may try to reduce each attention matrix to one a single vector, yielding one matrix per layer and for BERT 12 $(h \times n_s)$ matrices that show attention activity throughout the entire model. In a first step, it was tried to average the attention matrices along their columns, but it has been found that reducing attention matrices using the maximum of each column produces better results. This motivates the definition of *maximum attention*:

Definition 4.2 (Maximum Attention (MA))**.** *Let $A^{(k,l)} \in \mathbb{R}^{n_s \times n_s}$ be the attention matrix produced by attention head k on layer l when processing a text using BERT. Then the* ***maximum attention*** *of $A^{(k,l)}$ is defined as follows:*

$$\begin{aligned} MA \quad &: \mathbb{R}^{n_s \times n_s} \to \mathbb{R}^{n_s} \\ MA(A^{(k,l)})_j &= \max_{i \in \{1,\dots,n_s\}} A^{(k,l)}_{i,j} \quad \forall j \in \{1, \dots, n_s\} \end{aligned}$$

Which reduces the attention matrix to a single vector in $(0,1)^{n_s}$. Further, ***layer-wise maximum attention metric*** *is defined component-wise as*

$$\begin{aligned} \textit{layer-wise } MA(A^{(1,l)}, \dots, A^{(h,l)})_{ij} &= MA(A^{(i,l)})_j \\ &= \max_{m \in \{1,\dots,n_s\}} A^{(i,l)}_{m,j} \end{aligned}$$

which reduces an entire attention mechanism's activity, on layer l, to a single matrix in $(0,1)^{h \times n_s}$. Each row of this matrix shows the respective attention head's activity.

One may interpret $\mathrm{MA}(A^{(k,l)})_j$ as the maximum amount of information that attention head k on layer l queried from token j. Using this technique, we can visualize *the entire model's attention* in 12 plots, one for each layer.

As we can see in Figure 13, this visualization allows us much more easily to identify the attention heads that do most as well as understand the attention patterns the different heads produce. Interestingly, we can see that on each of the lower layers in the model, there are usually one or two attention heads which attend to consecutive tokens in the sequence. As we have seen before, attention head 2 on layer 2 seems to process positional information using exactly this technique. It mostly attends to consecutive token blocks consisting of two to five tokens and then breaks this chain of attention by giving a low attention score to the next token in order to separate them in the representation space. I suspect that this attention head tries to isolate certain ideas in the text forming a kind of summary of the statements in the text. It can then move information within this attention chain such that a later attention head, which likely operates using the same procedure at a higher level, may query this summary to combine it with other extracted summaries. Another example is the attention head 10 on layer 3, which shows a high average attention score for almost every token in the sequence.

As we can see in Figure 14, attention head 10 on layer 3 seems to be the most active attention head in the model as it forms the longest attention chains. We can nicely combine maximum attention to find the patterns in the interesting attention heads and then visualize them using attention head based t-SNE.

4.4 Examining the robustness of Attention Head based t-SNE

Using the KL divergence between the embedded representations found by t-SNE (q_{ij}) and the original representations (p_{ij}), we can compare the quality of the embeddings produced by t-SNE as well as those produced by the Attention Head based t-SNE. This enables a numerical robustness comparison of the embeddings produced by both algorithms [23]. Instead of only looking at the KL divergences, we disturb the input sequence by randomly replacing a certain percentage of the tokens with a random token from the set of all tokens found in the text. After disturbing 5% of the tokens, we evaluate both the classical t-SNE on layer 3 as well as the Attention Head based t-SNE on attention head 10 on layer 3 using their KL divergences. To

[23]Such disturbances may occur if typos are found in the text

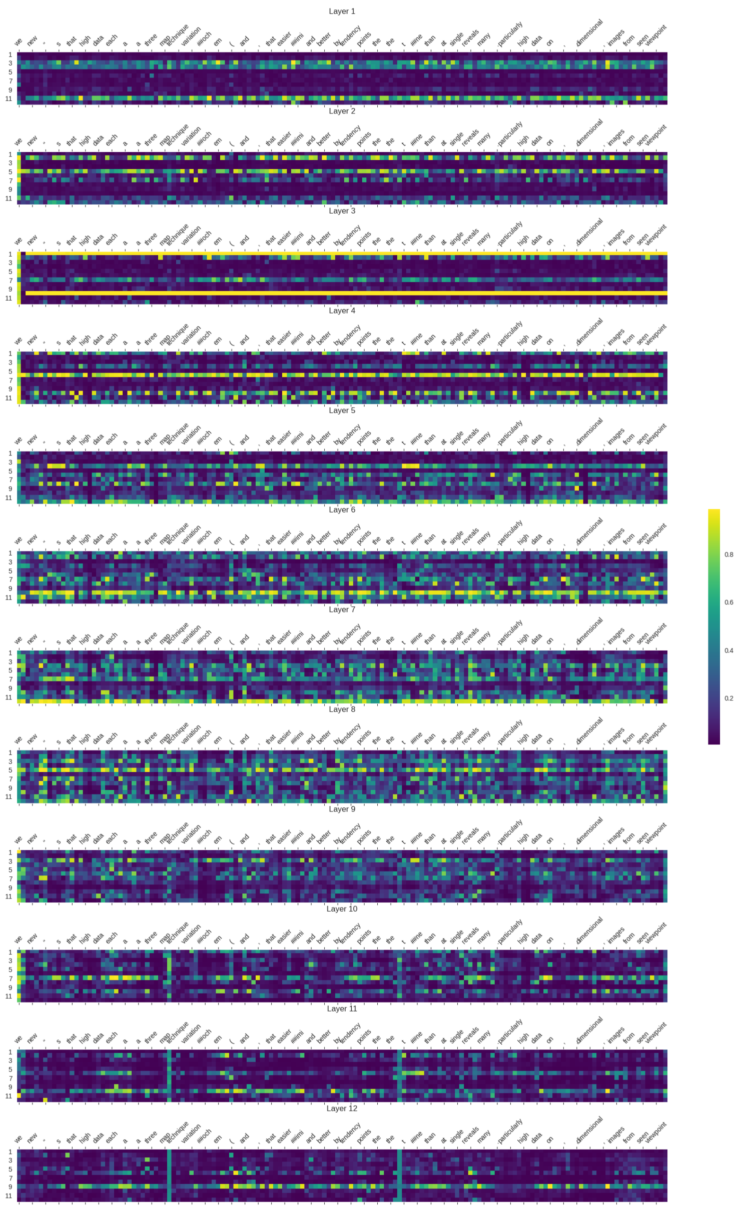

Figure 13: layer-wise Maximum Attention Metric for BERT applied on the abstract of the original t-SNE paper showing maximum attention for each attention head on each token in the sequence; Only every third token and every second attention head is labeled on the axis for readability.

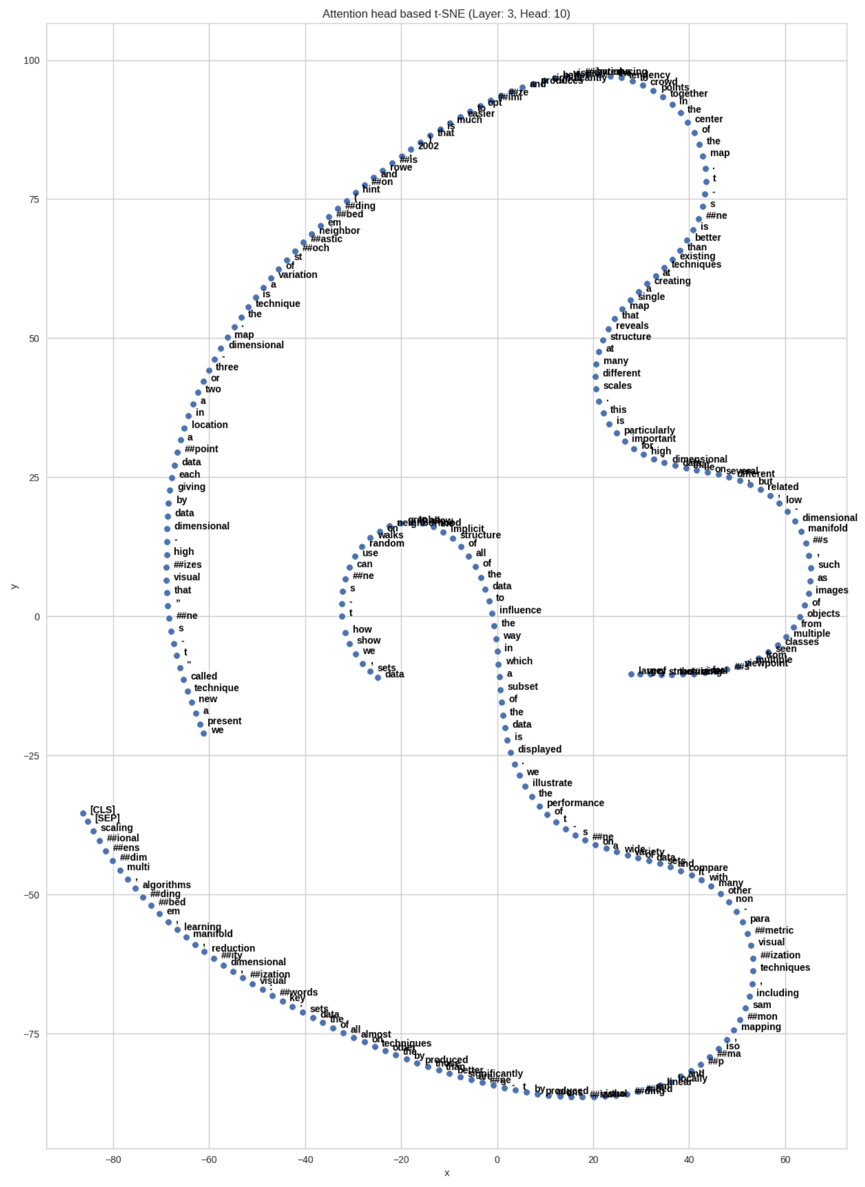

Figure 14: Visualizations of Attention Head 10 at Layer 3 produced by the Attention Head based t-SNE

maintain a certain statistical significance, we repeat this experiment 1000 times.

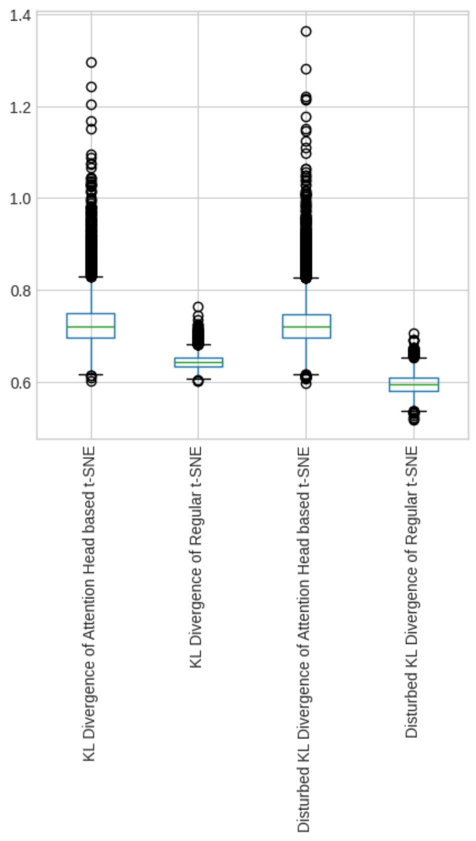

Figure 15: Boxplot showing the KL divergences of both Algorithms both with and without disturbance by token replacement

Interestingly, in Figure 15 we see that the KL divergence of the standard t-SNE improves when the input sequence is disturbed. This may be explained by the fact that embeddings after disturbance form a less complex structure, which might be easier to embed using t-SNE. More importantly however, in Figure 16 we see that the standard deviation of the KL divergences of the standard t-SNE increased significantly more than that of the Attention Head based t-SNE after disturbance. This shows that the Attention Head based t-SNE is more robust to changes in the input sequence than the standard t-SNE.

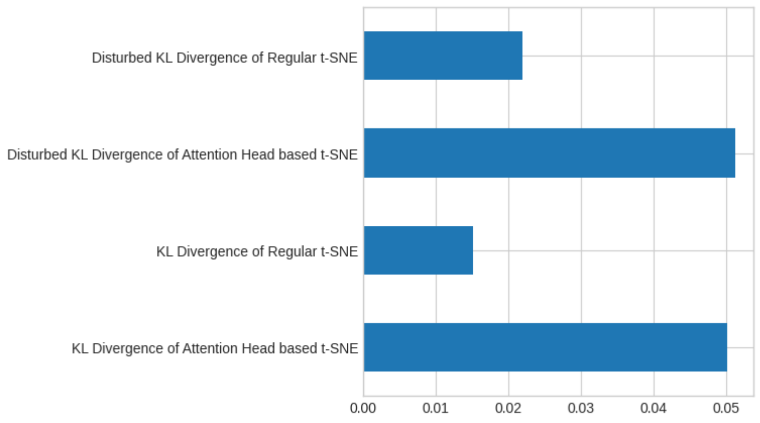

Figure 16: Bar plot showing the standard deviations of the KL divergences of both Algorithms with and without disturbance by token replacement

5 Conclusion & Outlook

Concluding my study, I now want to summarize what the results of my experiments have shown me and what might be the next steps to take. Specifically, I want to explain the implications my findings have on the t-SNE algorithm, especially its problems regarding its use of the normal distribution when determining the neighboring probabilities.

5.1 Attention Head based t-SNE

As we have seen in Section 4.2, attention head based t-SNE allows us, for the first time, to visualize contextualized representations of text in a meaningful and interpretable way. It can be used in studying various learned representations of text and mainly to better visually understand the success of the attention mechanism inside the transformer. Further, I hope it becomes a useful tool in making transformers and neural networks more transparent and interpretable as well as making the transformer architecture more accessible to researchers.

5.2 Implications for t-SNE in General

The performance of attention head based t-SNE, specifically its superior embeddings compared to the standard t-SNE, shows that using a better prior probability distribution for determining the neighboring probabilities in the high-dimensional space is crucial since it allows the algorithm to better recreate the original structure in the low-dimensional space. As is known from the literature, normality assumptions can be problematic if they are not met by the data, especially if the data is known not to follow this assumption.

Unfortunately, a prior right-stochastic matrix is not always easily found for some data sets. Still, this technique can likely be applied once a suitable embedding[24] is found for which such a matrix can easily be constructed using this same algorithm since attention matrices in transformers are simply softmaxed scaled dot products.

5.3 Conclusion on Current Transformer Architectures

One of the interesting observations from maximum attention was that not all attention heads are equally active, which is nothing new. This suggests that transformers may benefit from having more dynamic attention heads, in which the keys and values[25] are for example transformed more sophisticated than just a linear transformation, perhaps another feed-forward network. This is especially important since the MHA performs the expensive $\mathcal{O}(n_s^2)$ dot-products, so additional processing beforehand is computationally cheap.

Also, I think the transformer may benefit from additional post-processing of the output of the attention heads.

$$\mathrm{MHA}(X, X, X) = \mathrm{Concat}_2\left(H_1, H_2, \ldots, H_h\right) \cdot W_O$$

As we can see, the output of the *expensive* attention mechanism is only linearly transformed after the concatenation. A more sophisticated post-processing step may be beneficial here, possibly directly composing the feed-forward network instead of W_O at this point which would also save a matrix multiplication as well as one layer normalization step such that the MHA looks like the following.

$$\mathrm{MHA}(X, X, X) = \mathrm{FFNBlock}\left(\mathrm{Concat}_2\left(H_1, H_2, \ldots, H_h\right)\right)$$

5.4 Outlook for Future Work

Understanding black box models such as transformers is very important for the reliable development of such technologies and I believe there is still a long way to go but also lots of possibilities to explore. I hope that my work will be a step in the right direction and that it will inspire others to explore the transformer architecture further. Undoubtedly, language models, and especially transformers, will literally transform the way we interact with computers and I hope that this work sheds some light on the inner workings of these models so they can be understood, optimized and built in a more reliable and fair way.

[24]hidden representations from neural networks likely work well as embeddings for this technique

[25]transforming the queries would more dynamically change the positions where features flow to, which intuitively should stay more static as opposed to the feature sources

References

[Araci, 2019] Araci, D. (2019). Finbert: Financial sentiment analysis with pre-trained language models. *CoRR*, abs/1908.10063.

[Arnab et al., 2021] Arnab, A., Dehghani, M., Heigold, G., Sun, C., Lučić, M., and Schmid, C. (2021). Vivit: A video vision transformer. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 6836–6846.

[Ba et al., 2016] Ba, J. L., Kiros, J. R., and Hinton, G. E. (2016). Layer normalization. *arXiv preprint arXiv:1607.06450*.

[Beckmann et al., 1990] Beckmann, N., Kriegel, H.-P., Schneider, R., and Seeger, B. (1990). The r*-tree: An efficient and robust access method for points and rectangles. In *Proceedings of the 1990 ACM SIGMOD international conference on Management of data*, pages 322–331.

[Brown et al., 2020] Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. (2020). Language models are few-shot learners. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H., editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc.

[Cao, 2021] Cao, S. (2021). Choose a transformer: Fourier or galerkin. In Ranzato, M., Beygelzimer, A., Dauphin, Y., Liang, P., and Vaughan, J. W., editors, *Advances in Neural Information Processing Systems*, volume 34, pages 24924–24940. Curran Associates, Inc.

[Chowdhery et al., 2022] Chowdhery, A., Narang, S., Devlin, J., Bosma, M., Mishra, G., Roberts, A., Barham, P., Chung, H. W., Sutton, C., Gehrmann, S., Schuh, P., Shi, K., Tsvyashchenko, S., Maynez, J., Rao, A., Barnes, P., Tay, Y., Shazeer, N., Prabhakaran, V., Reif, E., Du, N., Hutchinson, B., Pope, R., Bradbury, J., Austin, J., Isard, M., Gur-Ari, G., Yin, P., Duke, T., Levskaya, A., Ghemawat, S., Dev, S., Michalewski, H., Garcia, X., Misra, V., Robinson, K., Fedus, L., Zhou, D., Ippolito, D., Luan, D., Lim, H., Zoph, B., Spiridonov, A., Sepassi, R., Dohan, D., Agrawal, S., Omernick, M., Dai, A. M., Pillai, T. S., Pellat, M., Lewkowycz, A., Moreira, E., Child, R., Polozov, O., Lee, K., Zhou, Z., Wang, X., Saeta, B., Diaz, M., Firat, O., Catasta, M., Wei, J., Meier-Hellstern, K., Eck, D., Dean, J., Petrov, S., and Fiedel, N. (2022). PaLM: Scaling language modeling with pathways.

[Conneau et al., 2016] Conneau, A., Schwenk, H., Barrault, L., and Lecun, Y. (2016). Very deep convolutional networks for text classification. *arXiv preprint arXiv:1606.01781*.

[Devlin et al., 2018] Devlin, J., Chang, M., Lee, K., and Toutanova, K. (2018). BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805.

[Dong et al., 2018] Dong, L., Xu, S., and Xu, B. (2018). Speech-transformer: A no-recurrence sequence-to-sequence model for speech recognition. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5884–5888.

[Dosovitskiy et al., 2020] Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., and Houlsby, N. (2020). An image is worth 16x16 words: Transformers for image recognition at scale. *CoRR*, abs/2010.11929.

[Gulati et al., 2020] Gulati, A., Qin, J., Chiu, C.-C., Parmar, N., Zhang, Y., Yu, J., Han, W., Wang, S., Zhang, Z., Wu, Y., and Pang, R. (2020). Conformer: Convolution-augmented transformer for speech recognition.

[Harris et al., 2020] Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., del Río, J. F., Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C., and Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, 585(7825):357–362.

[He et al., 2016] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

[Hendrycks and Gimpel, 2016] Hendrycks, D. and Gimpel, K. (2016). Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*.

[Hinton and Roweis, 2002] Hinton, G. E. and Roweis, S. (2002). Stochastic neighbor embedding. *Advances in neural information processing systems*, 15.

[Hornik et al., 1989] Hornik, K., Stinchcombe, M., and White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366.

[Knuth, 1968] Knuth, D. E. (1968). *The Art of Computer Programming, Volume 1: Fundamental Algorithms*. Addison-Wesley.

[Kovachki et al., 2021] Kovachki, N. B., Li, Z., Liu, B., Azizzadenesheli, K., Bhattacharya, K., Stuart, A. M., and Anandkumar, A. (2021). Neural operator: Learning maps between function spaces. *CoRR*, abs/2108.08481.

[Levenshtein, 1966] Levenshtein, V. I. (1966). Binary Codes Capable of Correcting Deletions, Insertions and Reversals. *Soviet Physics Doklady*, 10:707.

[Liu et al., 2021] Liu, Z., Lin, Y., Cao, Y., Hu, H., Wei, Y., Zhang, Z., Lin, S., and Guo, B. (2021). Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 10012–10022.

[Mielke et al., 2021] Mielke, S. J., Alyafeai, Z., Salesky, E., Raffel, C., Dey, M., Gallé, M., Raja, A., Si, C., Lee, W. Y., Sagot, B., and Tan, S. (2021). Between words and characters: A brief history of open-vocabulary modeling and tokenization in NLP. *CoRR*, abs/2112.10508.

[Morton, 1966] Morton, G. M. (1966). A computer oriented geodetic data base and a new technique in file sequencing.

[Radford et al., 2019] Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., et al. (2019). Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

[Reimers and Gurevych, 2019] Reimers, N. and Gurevych, I. (2019). Sentence-bert: Sentence embeddings using siamese bert-networks. *CoRR*, abs/1908.10084.

[Rumelhart et al., 1985] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1985). Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science.

[Su et al., 2021] Su, J., Lu, Y., Pan, S., Wen, B., and Liu, Y. (2021). Roformer: Enhanced transformer with rotary position embedding. *arXiv preprint arXiv:2104.09864*.

[Van Der Maaten, 2009] Van Der Maaten, L. (2009). Learning a parametric embedding by preserving local structure. In *Artificial intelligence and statistics*, pages 384–391. PMLR.

[Van der Maaten and Hinton, 2008] Van der Maaten, L. and Hinton, G. (2008). Visualizing data using t-SNE. *Journal of machine learning research*, 9(11).

[Vaswani et al., 2017] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need. *CoRR*, abs/1706.03762.

[Wang et al., 2020] Wang, S., Li, B. Z., Khabsa, M., Fang, H., and Ma, H. (2020). Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768*.

[Wolf et al., 2019] Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., and Brew, J. (2019). Huggingface's transformers: State-of-the-art natural language processing. *CoRR*, abs/1910.03771.

A Appendix

A.1 Determining the Standard Deviation in t-SNE

In one sentence, t-SNE finds σ_i such that the perplexity of x_i's induced probability distribution P_i is equal to the perplexity specified by the user, as a parameter. P_i's perplexity is defined as follows:

$$\text{Perplexity}(P_i) = 2^{H(P_i)}$$

where $H(P_i)$ is the Shannon entropy of P_i which is defined as follows:

$$H(P_i) = - \sum_{j=1,\, j \neq i}^{n} p_{j|i} \log_2 p_{j|i}$$

And of course n is the number of data points.

Specifically, t-SNE performs a binary search over the values of σ_i for each x_i so as to best approximate the desired perplexity.

A.2 Parameters used in the t-SNEs

To find the best visualizations, all t-SNEs, also the attention head based t-SNEs, have been run 100 times and the best visualization according to the KL divergence has been chosen. For the regular t-SNE the following parameters have been used.

- perplexity: Perp = 20
- learning rate: $\eta = 5$
- number of iterations: $N = 1000$

In attention head based t-SNE, the same parameters, as far as they still apply, have been used.

A.3 Quantile Equidistant Rescaling of Scatter Plots

Assume that the data points $(x_1, y_1), \dots, (x_n, y_n)$ we want to display in a scatter plot are not uniformly distributed within the rectangle the scatter plot displays $[\min_{1\leq i\leq n} x_i, \max_{1\leq i\leq n} x_i] \times [\min_{1\leq i\leq n} y_i, \max_{1\leq i\leq n} y_i]$. As an example, we draw data points from 5 multivariate normal distributions with different means and covariances. Plotting such data directly using a scatter plot represents the euclidian distances well, but certain parts of the plot may get very crowded and in the case of our embeddings become unreadable in such parts.

We can now imagine that it becomes quite difficult to examine to local structure of the data points in cluster C. Further, we may notice that the

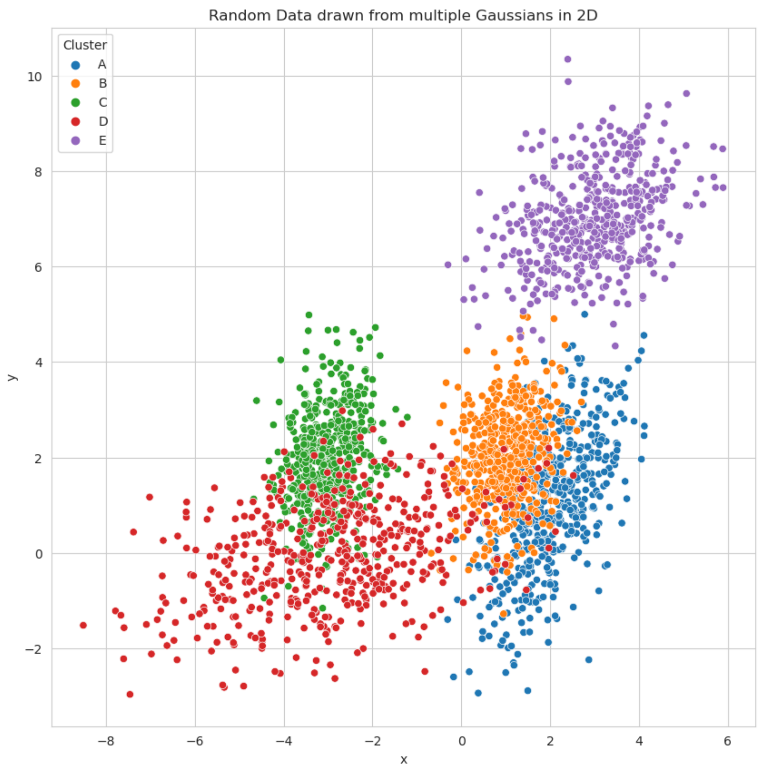

Figure 17: Scatter plot without any rescaling applied; The colors indicate from which distribution the data points originate

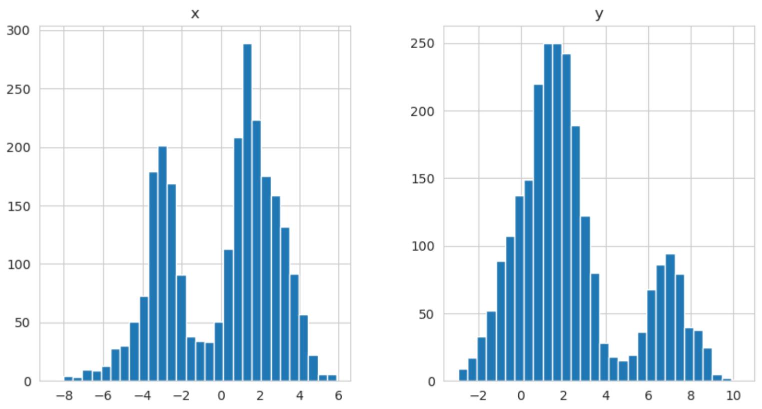

Figure 18: Histograms of the original, unscaled components

components x and y of the data points are also not uniformly distribute and thus do not optimally fill the geometry of the scatter plot.

Quantile equidistant rescaling alleviates this problem by non-linearly transforming the data points so as to better utilize the rectangular geometry of the scatter plot. It does so by first measuring a number of quantiles $1 < k < n$ for each axis, and then transforms this axis by linearly interpolating values that fall between those quantiles. For each axis, this yields a part-wise linear mapping φ from $[[\min_{1\leq i\leq n} x_i, \max_{1\leq i\leq n} x_i]]$ to $[0, 1]$, and similarly for y.

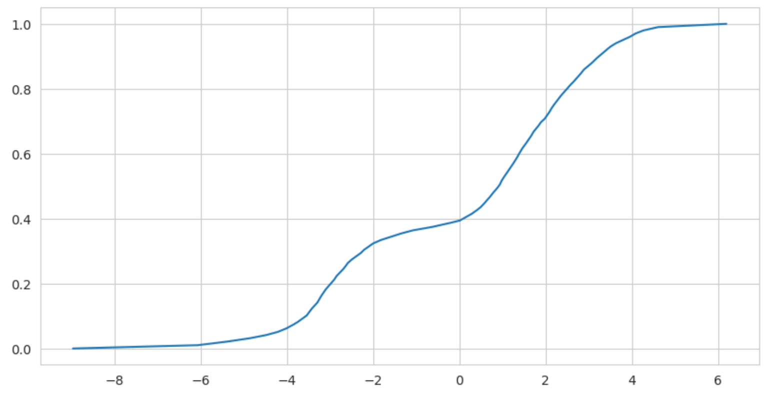

Figure 19: The interpolated rescaling map φ for the x-Axis from the above example for $k = 100$

Figure 19 shows this map for the x-Axis from before. Notice that for $k = n$ we obviously just yield the emipirical cumulative distribution function (ECDF). The advantage of this approach is that the parameter k, can be used to calibrate the strength at which the rescaling is done, so that lower k preserve more of the global structure keeping most parts of the axis linear. In all of our applications I chose $k = 100$ for simplicity. k can also be chosen relative to n of course, which results in roughly the interpolation between every n/k-th point.

To still indicate the original scale somehow, a number of ticks, which indicate another number of quantiles are projected onto the original scale using the inverse rescaling function φ^{-1}. These ticks are then *equidistant* on each axis but take on different values which indicate the original scale between each quantile.

In the Figures 20 and 21 we can see how the quantile equidistant rescaling better utilizes the rectangular shape of the scatter plot by projecting the components such that final axes' distributions are roughly uniform. If one wants to preserve more of the original scale, we show how quantile equidistant rescaling looks like for $k = 5$, interpolating the axis only between 5

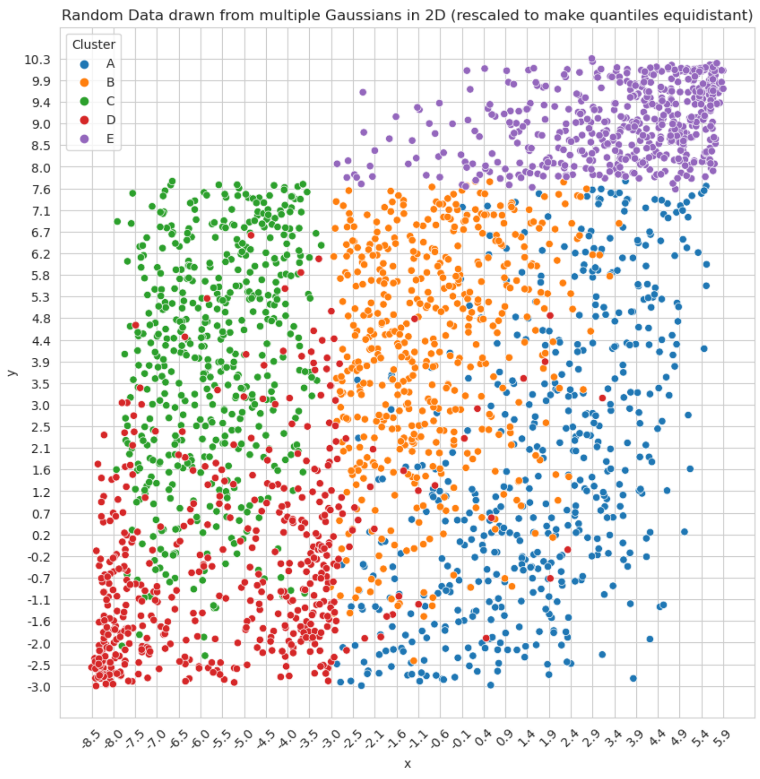

Figure 20: The rescaled scatter plot

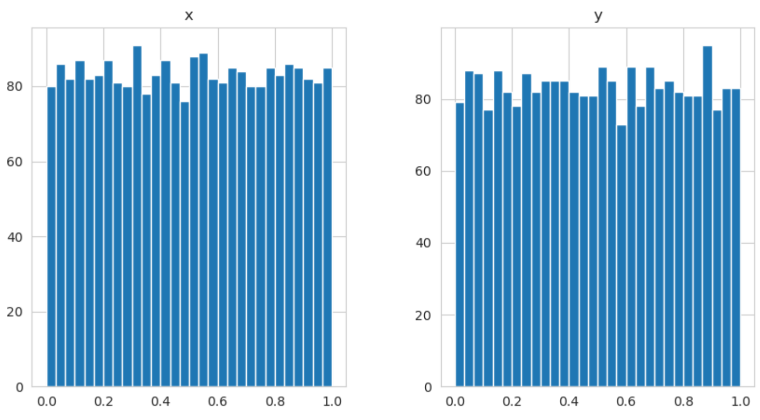

Figure 21: Histograms of the rescaled axis

quantiles in Figure 22.

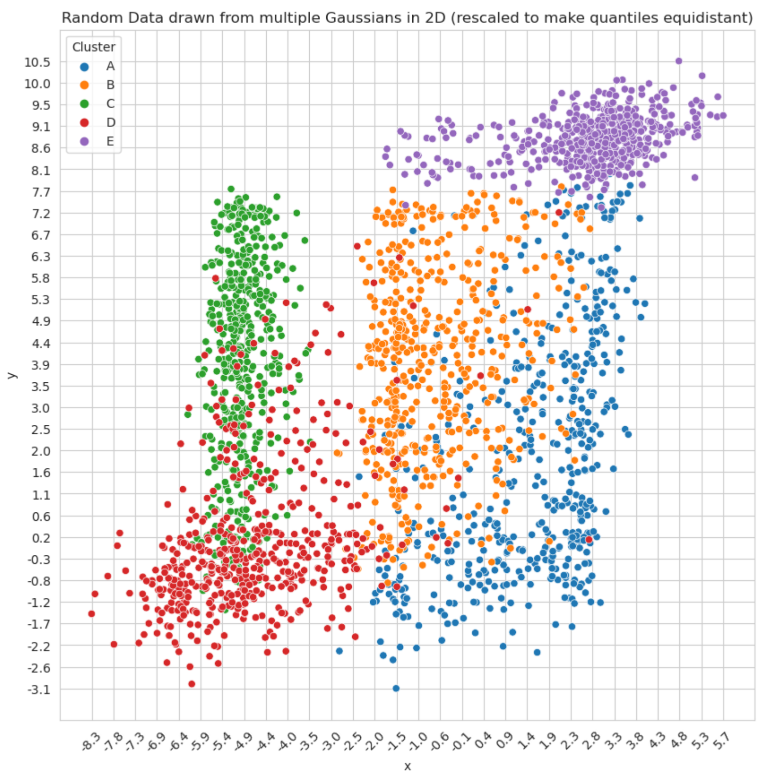

Figure 22: Quantile Equidistant Rescaling for $k = 5$ applied on the example from above

One notable characteristic of quantile equidistant rescaling is that if most of the data is clustered near the origin, it tends to zoom the center while squeezing the outside points into the corners of the plot. However, mostly in our plots we have much less points on the outside so this does not pose a problem. And of course, for high values of k, the algorithm strongly flattens out the original structure which may lead to confusion, so it should only be applied when readability is desired over structure.

A.4 Abstract of the Original t-SNE Paper

We present a new technique called "t-SNE" that visualizes high-dimensional data by giving each datapoint a location in a two or three-dimensional map. The technique is a variation of Stochastic Neighbor Embedding (Hinton and Roweis, 2002) that is much easier to optimize, and produces significantly better visualizations by reducing the tendency to crowd points together in the center of the map. t-SNE is better than existing techniques at creating a single map that reveals structure at many different scales. This is particularly important for high-dimensional data that lie on several different, but related, low-dimensional manifolds, such as images of objects from multiple classes seen from multiple viewpoints. For visualizing the structure of very large data sets, we show how t-SNE can use random walks on neighborhood graphs to allow the implicit structure of all of the data to influence the way in which a subset of the data is displayed. We illustrate the performance of t-SNE on a wide variety of data sets and compare it with many other non-parametric visualization techniques, including Sammon mapping, Isomap, and Locally Linear Embedding. The visualizations produced by t-SNE are significantly better than those produced by the other techniques on almost all of the data sets.

A.5 More on Attention Head based t-SNE

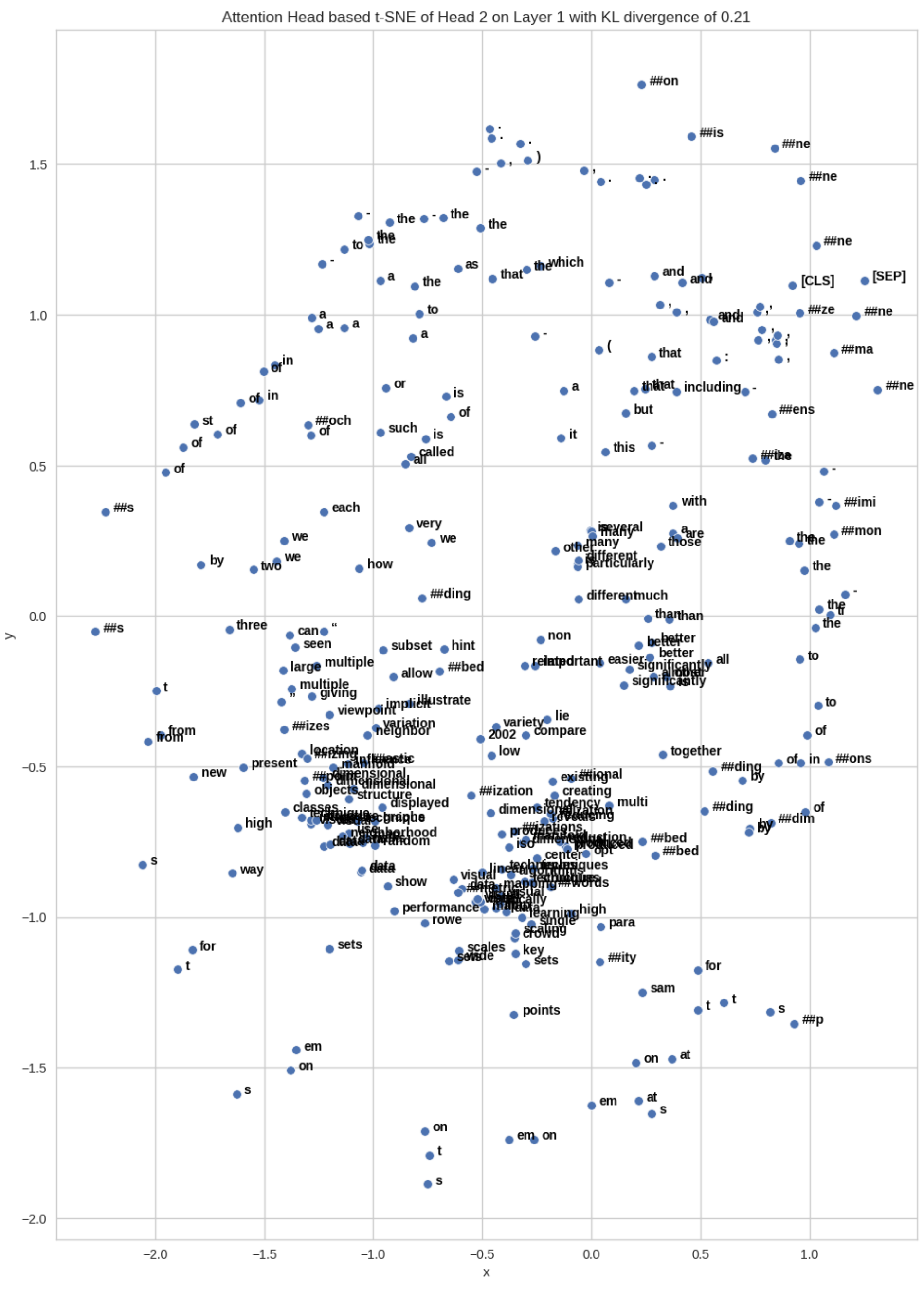

Figure 23: Visualizations of Attention Head 2 at Layer 1, applied to the abstract of the original t-SNE paper; with linear axis scales

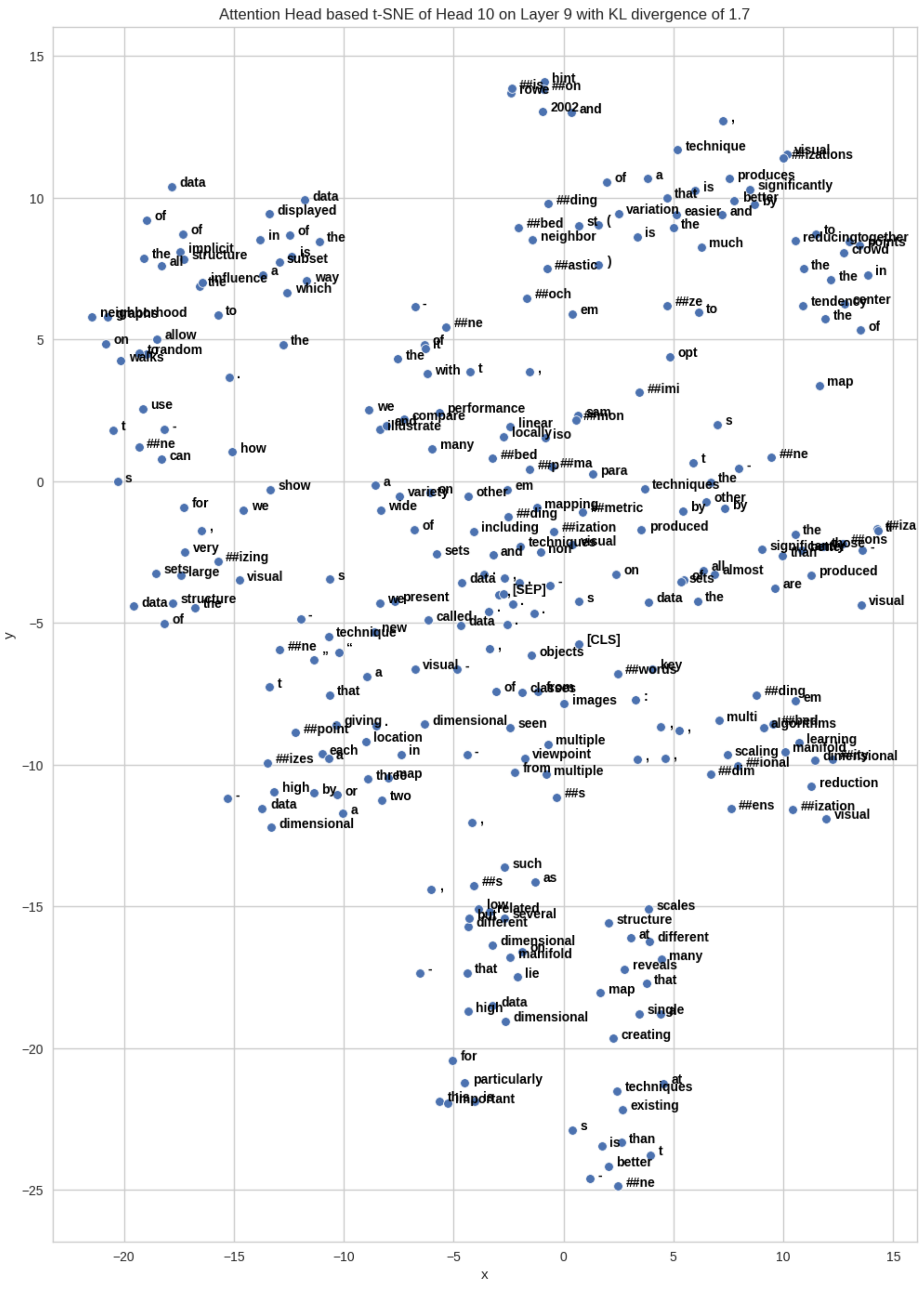

Figure 24: Visualizations of Attention Head 10 at Layer 9 produced by the attention head based t-SNE; with linear axis scales