TRANSFORMER

Hallvard Høyland Lavik
hallvard.hoyland.lavik@nmbu.no

Contents

Figures

Listings

Motivation

The transformer architecture, introduced in *Attention is All You Need* [1], aimed to overcome limitations of existing sequence-to-sequence models based on recurrent neural networks and convolutional neural networks. These models struggled with capturing long-range dependencies, processing long sequences, and efficiently utilizing parallel computation.

In order to overcome these challenges, the team [1] proposed a new architecture relying on attention mechanisms, to capture global dependencies between input and output elements. This design allows for more effectively attending long-range dependencies, and is the reason behind the transformers huge success in performing tasks related to natural language processing, among other things.

Note: in this report, only natural language processing is studied. It is however fully possible to apply the transformer architecture to other domains – such as images – as well. The theory does however remain quite similar, with minor modifications depending on the use-case.

In this study, the focus is solely on the application of natural language processing – as it was initially developed for. However, it's important to note that the transformer architecture can be (and has been successfully) adapted to other fields, such as image processing, with equal effectiveness. The underlying theory remains largely consistent across different applications, with only minor adjustments made to accommodate the specific requirements of each use-case [2].

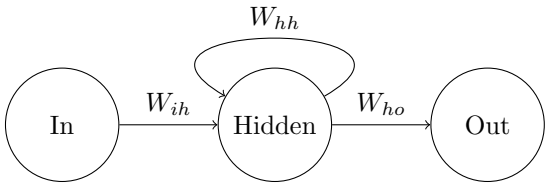
Traditional sequence based deep learning

Recurrent neural networks (RNNs) are a type of neural network designed for sequence-based data. Unlike traditional feedforward neural networks, RNNs have connections that loop back within the network, providing a way to maintain an internal state (*i.e.*, its memory).

BASIC STRUCTURE

A basic RNN has an input layer (In), a hidden recurrent layer (Hidden), and an output layer (Out). The recurrent layer processes sequences of data by looping its output back into its input, allowing it to learn from past information.

$$h_t = f_{\text{activation}}(W_{hh}h_{t-1} + W_{ih}x_t + b_h)$$
$$o_t = f_{\text{activation}}(W_{ho}h_t + b_o)$$



Where t represents the position in the sequence (*e.g.*, time), o the output, i the input and b the bias. h_{t-1} therefore represents the hidden output for the previous time-step, and h_t the current hidden output. $f_{\text{activation}}$ for the hidden and output layers may differ, and represent their activation functions.

LONG SHORT-TERM MEMORY

A Long Short-Term Memory (LSTM) model is a special type of RNN that can better learn long-term dependencies in the data compared to a simple RNN. It has a more complex internal structure involving gates that control the flow of information.

The LSTM structure consists of four gates, which combine or remove information. The operations done are linear, being

$$\oplus \text{ Element-wise addition. } \begin{bmatrix}0.8\\0.8\\0.8\end{bmatrix} \oplus \begin{bmatrix}1.0\\0.5\\0.0\end{bmatrix} = \begin{bmatrix}0.8+1.0\\0.8+0.5\\0.8+0.0\end{bmatrix} = \begin{bmatrix}1.8\\1.3\\0.8\end{bmatrix}$$

$$\otimes \text{ Element-wise multiplication. } \begin{bmatrix}0.8\\0.8\\0.8\end{bmatrix} \otimes \begin{bmatrix}1.0\\0.5\\0.0\end{bmatrix} = \begin{bmatrix}0.8\cdot 1.0\\0.8\cdot 0.5\\0.8\cdot 0.0\end{bmatrix} = \begin{bmatrix}0.8\\0.4\\0.0\end{bmatrix}$$

By inspecting these operations, we can see that gates using $\otimes$ is able to either block or allow information to pass through (respectively through values of 0.0 or 1.0), or something in-between. This means, that the network can learn previous state values, and take these into account when filtering values of new inputs.

An LSTM network has a *cell state* C, which acts as the memory of the network. This state is being transferred across the time-steps, thus allowing for previous inputted information to be retained in future time-steps.

Γ_f Forget gate layer

The first step in an LSTM is the *forget gate* layer. This is a neural network which takes in the previous output along with current input. This layer has a sigmoid activation function, $\sigma(\cdot)$, where inputs resulting in 0's lead to variables being left out of the cell state C.

$$\Gamma_f = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \qquad\qquad C_t^* = \Gamma_f \otimes C_{t-1}$$

Γ_u Update/input gate layer

The next step is to decide what new information to store in the cell state.

$$\Gamma_u = \sigma(W_u \cdot [h_{t-1}, x_t] + b_u) \qquad\qquad \tilde{C}_t = \texttt{tanh}(W_C \cdot [h_{t-1}, x_t] + b_C)$$

These two are then pairwise multiplied together, and added to the forgotten state.

$$\textbf{CELL STATE} \qquad \begin{aligned} C_t &= C_{t-1}^* \oplus (\Gamma_u \otimes \tilde{C}_t) \\ &= (\Gamma_f \otimes C_{t-1}) \oplus (\Gamma_u \otimes \tilde{C}_t) \end{aligned}$$

Γ_o Output gate layer

The output of the model is then calculated based on both the cell state and previous ouput as well as input.

$$\Gamma_o = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$

$$\textbf{OUTPUT} \qquad h_t = \Gamma_o \otimes \texttt{tanh}(C_t)$$

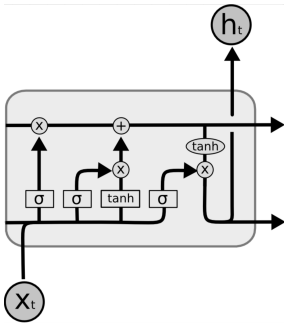


Figure 1: LSTM block (from [3]).

Although models like the mentioned Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) enable the retention of some prior information, they inherently struggle with handling long sequences. For example, when an LSTM cell state, C, is presented with an input sequence of length L, it is unable to maintain information from all previous steps when evaluating step l. This is because the cell state progressively becomes more abstract as it processes each step in the sequence, making it challenging to retain information from the distant past.

Natural language processing

TOKENIZATION

Tokenization, in terms of language processing, involves breaking down text into small pieces called tokens. Said tokens vary between the tokenization methods, and could include phrases, words, or even single characters. These tokens are then mapped to a numerical representation, as state-of-the-art artificial intelligence language models rely on numerical input. Tokenization methods are therefore used to preprocess the inputted text. [4] Thus, a tokenization model is in simple terms a dictionary of mappings between tokens and their corresponding numerical representations.

Vocabulary size

> When creating a tokenization model, its vocabulary is of great importance. Intuitively, the smaller the vocabulary, the bigger the tokenized sequences are. *I.e.*, a greater number of individual tokens are needed to compose the text it represents. Therefore, a tokenizer with a sufficiently large vocabulary, results in an ability to convert text into its numerical representation without leading to an unnecessary long and complicated representation.
>
> That is, if a tokenizers vocabulary consists of the individual letters of the alphabet, the tokenization of "word" becomes `[23, 15, 18, 4]` (assuming `a = 1`, `b = 2`, etc.), which leads to long sentences being mapped to even longer token representations. Oppositely, if a tokenizers vocabulary contains every English word, the tokenization of "word" is mapped to a single numerical representation, thus leading to smaller time complexity of computations in the forward pass of the transformer.

Byte-pair encoding

> Byte-pair encoding (BPE) is a type of tokenization method that is commonly used in large language models [5]. It works by initially representing text as a sequence of characters, and then iteratively replacing the most frequent pair of bytes with a single, new byte. This process continues until the desired vocabulary size is reached. The result is a set of tokens that represent common sequences of characters, which can be more efficient and effective for language modeling than word-based tokenization. [6, 4, 7]
>
> BPE is commonly used because it strikes a balance between character-level and word-level tokenization. It can handle out-of-vocabulary words and rare words more effectively than word-level tokenization, while still capturing meaningful linguistic units. [6, 7]

Regular expression

> In order to simplify the vocabulary creation, regular expressions are useful. The regular expressions are increasingly complex [7], but help filter out redundant characters. The regular expression seen in Table 1 is used by GPT-4 when tokenizing inputs (according to Karpathy [4]) [7].
>
> ```
> '(?i:[sdmt]|ll|ve|re)|[^\r\n\p{L}\p{N}]?+\p{L}+|\p{N}{1,3}| ?[^\s\p{L}\p{N
>    }]++[\r\n]*|\s*[\r\n]|\s+(?!\S)|\s+
> ```

COMPONENT	DESCRIPTION
`'(?i:[sdmt]|ll|ve|re)`	Uses a case-insensitive inline modifier `(?i:...)` to match common abbreviations and contractions in English text (`'`). Matches either a single character that is one of `s`, `d`, `m`, or `t`, or the two-letter sequences `ll`, `ve`, or `re`.
`[^\r\n\p{L}\p{N}]?+\p{L}+`	Uses a possessive quantifier `?+` to match one or more Unicode letters (`\p{L}`) that are preceded by zero or more characters that are not (`^`) Unicode letters, line breaks (`\r\n`), or Unicode numbers (`\p{N}`). Matches words that start with non-letter characters, such as hyphenated words.
`\p{N}{1,3}`	Matches up to three Unicode numbers (`\p{N}`). Matches numbers in the text.
`␣?[^\s\p{L}\p{N}]++[\r\n]*`	Matches zero or one space character (␣), followed by one or more characters that are not (`^`) whitespace (`\s`), Unicode letters (`\p{L}`), or Unicode numbers (`\p{N}`), followed by zero or more line breaks (`[\r\n]*`). The `?` at the beginning of the pattern makes the preceding space optional. Matches punctuation and symbols that are not part of words or numbers.
`\s*[\r\n]`	Matches zero or more whitespace characters (`\s*`) followed by a line break (`[\r\n]`). Matches line breaks in the text.
`\s+(?!\S)`	Matches one or more whitespace characters (`\s+`) that are not followed by a non-whitespace character (`(?!\S)`). Matches whitespace at the end of lines.
`\s+`	Matches one or more whitespace characters, *i.e.*, spaces between words.

Training algorithm

When creating the BPE tokenizer, elements are continuously merged. The following code (from [4]) handles this merging; given a list of integers, `ids`, it replaces all consecutive occurences of `pair` with the new token `idx`:

```
def merge(ids, pair, idx):
    newids = []
    i = 0
    while i < len(ids):
        if ids[i] == pair[0] and i < len(ids) - 1 and ids[i+1] == pair[1]:
            newids.append(idx)
            i += 2
        else:
```

```
            newids.append(ids[i])
            i += 1
    return newids
```

With the helper-function defined, the tokenizer (from [4]) can begin training on a given string, `text`, and a regular expression pattern, `pattern` (*e.g.*, the pattern seen above):

```
merges = {}
vocabulary = {}

ids = [list(pt.encode("utf-8")) for pt in regex.findall(pattern, text)]

for i in range(vocabulary_size):
    stats = {}
    for chunk_ids in ids:
        for pair in zip(ids, ids[1:]):
            stats[pair] = counts.get(pair, 0) + 1

    pair = max(stats, key=stats.get)

    ids = [merge(chunk_ids, pair, i) for chunk_ids in ids]

    merges[pair] = i
    vocabulary[i] = vocabulary[pair[0]] + vocabulary[pair[1]]
```

The objective is to iteratively merge the most common pairs of characters or tokens in the given text to create new tokens until the desired `vocabulary_size` is achieved.

First, the `text` to be trained on is split into chunks (`pt`) based on the `pattern`, and each chunk is encoded into a list of bytes.

In each iteration, the frequency of consecutive pairs in the encoded chunks is calculated, and the pair with the highest frequency is merged into a new token. All occurrences of the pair in the encoded chunks are replaced with the new token, which is also added to the `vocabulary` and the `merges` dictionary. The updated `vocabulary` and `merges` dictionary are then stored, and is what is used when encoding or decoding new text or tokens, respectively.

EMBEDDING

Word (*i.e.*, token) embedding is a technique in natural language processing where words from the vocabulary are mapped to vectors of real numbers. These vectors capture the semantic properties of the words. For instance, words that are semantically similar are generally mapped to vectors that are close to each other in the vector space.

word2vec

In order to achieve this vectorized word representation, word2vec is commonly used, as it is highly generalizable. The reason for this generalizability is that word2vec is a neural network-based method that comes in two variants: continuous bag of words (CBoW) (based on Cloze process from Taylor [8]) and Skip-gram.

The CBoW model predicts a target word given its context words, while the Skip-gram model predicts the context words given a target word. In both models, the input words are represented as one-hot vectors, which are then multiplied by a (learned) weight matrix to get the corresponding word embeddings. During training, the output words are also represented as one-hot vectors, which serve as the ground truth during training. The goal of the models is to learn a weight matrix that can map each word in the vocabulary to a dense, low-dimensional vector that captures its semantic and syntactic relationships with other words. [9]

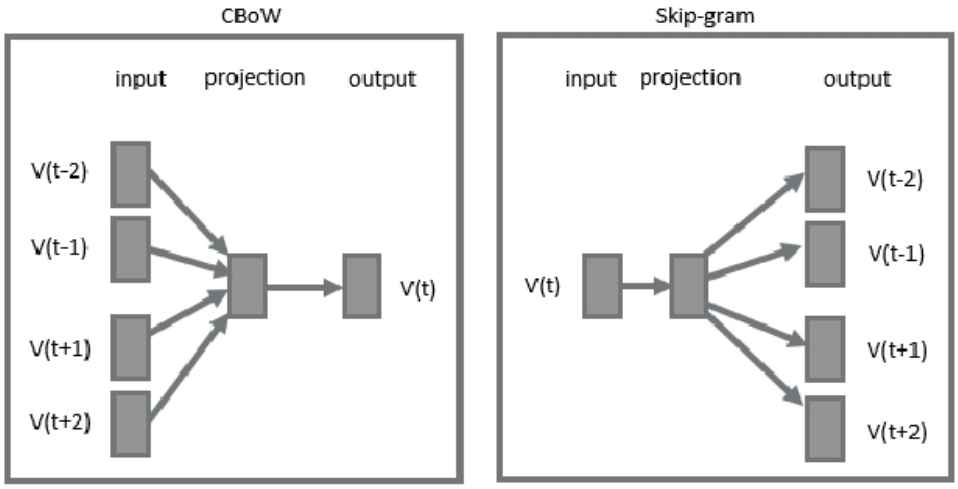


Figure 2: word2vec: CBoW and Skip-gram (from [10]).

For pure embedding models (*i.e.*, models which is used for embedding only), word2vec is a good approach. For large language models, however, the embedding is typically represented as a single dense layer (`torch.nn.Embedding`, which is in simple terms the same as `torch.nn.Linear`, except its handling of index representations instead of raw values), mapping token indices of dimensions `vocabulary_size` $\mapsto$ `embedding_size`.

Comparing embeddings of GPT-2

In Figure 3, the embedding of various word and sentences are compared. Here, the pretrained GPT-2 model is used to get the embeddings, and the cosine similarity between two embeddings is compared. Here, we see that the similarity between `boy` and `girl` is relatively large, whereas `man` and `girl` is small(er). Thus, we clearly see that semantically similar words (or sentences) tend to be relatively similar. As an example, the sentence `a one a two a three` and `!!!` are compared, where we see a low similarity – corresponding to the aforesaid intuition. It is however worth noting that there is no concrete reason that similar words should align, but here we observe that they in fact (to some extent) does.

When comparing embeddings, it is common to probe pure embedding models. However, as this is done by countless others, and because transformers are of interest here, the inspection of GPT-2 was done (as seen in Figure 3) Interestingly, we also see that some sense of semantic meaning is captured in large language models, although not as much as for pure embedding models [11].

[1]:
```
import torch
from transformers import GPT2Tokenizer, GPT2Model
```

[2]:
```
tokenizer = GPT2Tokenizer.from_pretrained('gpt2')
tokenizer.pad_token = tokenizer.eos_token
model = GPT2Model.from_pretrained('gpt2')
```

[3]:
```
embedding = {"man": None, "boy": None, "teenager": None, "girl": None,
             "a one a two a three": None, "!!!": None}
for word in embedding:
    ids = tokenizer(word, return_tensors='pt')["input_ids"]
    with torch.no_grad():
        outs = model.wte(ids).sum(dim=1).flatten()
    embedding[word] = outs
```

[4]:
```
def compare(one, two):
    sim = torch.nn.functional.cosine_similarity(
        one,
        two,
        dim=0
    ).item()
    print(f"{int(sim * 100)} % cosine similarity")
```

[5]:
```
compare(embedding["boy"], embedding["girl"])
```

```
62 % cosine similarity
```

[6]:
```
compare(embedding["man"], embedding["girl"])
```

```
39 % cosine similarity
```

[7]:
```
compare(embedding["boy"] + embedding["man"], embedding["teenager"])
```

```
44 % cosine similarity
```

[8]:
```
compare(embedding["a one a two a three"], embedding["!!!"])
```

```
19 % cosine similarity
```

Figure 3: Cosine similarity between GPT-2 embeddings.

Attention

Arguably, the most important core function of information processing in the brain is to selectively attend important impressions [12], which has also become the core feature (and the reason for success) of the transformer model [13]. For instance, when reading a document, much of its content is redundant, and the human brain is able to attend to the important parts and their connection – thus disregarding most of the redundant information. Likewise, the attention mechanism of the transformer aims at selectively attending different parts of the input sequence when generating an output [1].

The attention mechanism used in transformers can, in simple terms, be thought of as the idea of each word of a sequence containing parts of relevant preceding words; how much should each of the prior words in the sentence influence the current word? Thus, when the model is predicting the next word, given the example sequence;

$$\texttt{At my dairy farm we always get our milk from \_\_\_},$$

the model would first apply attention to `from`, leading to its updated representation:

$$\texttt{from}' = \texttt{from} + 0.3 \times \texttt{dairy} + 0.6 \times \texttt{farm} + 0.9 \times \texttt{milk}.$$

Thereafter, when the model is to predict the next word after `from`, it is perhaps nudged towards `cows`, as the attended `from`$'$ contains some fraction of `milk` *et cetera*. Not unlike the gates of the LSTM.

ORIGIN OF ATTENTION IN DEEP LEARNING

When trying to solve problems related to machine translation, Bahdanau *et al.* [14] began experimenting with how the context of a given sentence may be used when predicting an output. In the paper *ibid.*, they came up with a method where, for any given word i in the sequence, its context is composed of a weighed sum of the other words in the sequence.

While recent methods has optimized this approach by using matrices [1] instead of a neural network [14], the theory remains the same. It is, however, worth noting that it is possible to obtain the (additive) attention through a neural network, but that it is deemed more computational efficient to use (multiplicative attention through) optimized matrix multiplication algorithms [1].

INPUTS

When training a transformer model, the inputs are represented as a tensor $\mathbf{X}$ of (token) embeddings of a sequence, hereafter thought of as a matrix of words to simplify the intuition;

$$\mathbf{X} = \begin{bmatrix} \texttt{sequence 0} \\ \vdots \\ \texttt{sequence N} \end{bmatrix} = \begin{bmatrix} \texttt{word}_{0,0} & \dots & \texttt{word}_{0,M} \\ \vdots & \ddots & \vdots \\ \texttt{word}_{N,0} & \dots & \texttt{word}_{N,M} \end{bmatrix}. \tag{1}$$

Here, a sequence could represent a cutout from a text, *i.e.*, `sequence 0 = text[0:M]`. Thus, in a similar manner, we can also express the targets, $\mathbf{Y}$, as the right-shifted version of $\mathbf{X}$ (where a sequence therefore becomes `text[1:M+1]`);

$$\mathbf{Y} = \begin{bmatrix} \texttt{right-shifted sequence 0} \\ \vdots \\ \texttt{right-shifted sequence N} \end{bmatrix} = \begin{bmatrix} \texttt{word}_{0,1} & \dots & \texttt{word}_{0,M+1} \\ \vdots & \ddots & \vdots \\ \texttt{word}_{N,1} & \dots & \texttt{word}_{N,M+1} \end{bmatrix}. \tag{2}$$

Visualizing this, by inspecting a row of $\mathbf{X}$ and the corresponding row of $\mathbf{Y}$;

$$\begin{aligned} \mathbf{X}_{\text{row } 0} = \mathbf{x} &= \begin{bmatrix} \texttt{At}, & \texttt{my}, & \texttt{dairy}, & \texttt{farm}, & \ldots \end{bmatrix} \\ \mathbf{Y}_{\text{row } 0} = \mathbf{y} &= \begin{bmatrix} \texttt{my}, & \texttt{dairy}, & \texttt{farm}, & \texttt{we}, & \ldots \end{bmatrix}, \end{aligned} \tag{3}$$

we see that the targets of $\mathbf{x}_{0 \to i}$ is $\mathbf{y}_i$.

Decoder-only models

For a next-word prediction model, such as GPT-2 [15], the architecture consists of solely the decoder of the transformer, the $\mathbf{X}$ and $\mathbf{Y}$ seen in Equations (1) and (2) are then used during training.

Encoder-decoder models

When the goal of the transformer is to map some inputs to a specified output (*e.g.*, translation between languages), the previously explained procedure (of using the shifted inputs as targets) does not work. The matrices $\mathbf{X}$ and $\mathbf{Y}$ thus become;

$$\mathbf{X} = \begin{bmatrix} \texttt{source 0} \\ \vdots \\ \texttt{source N} \end{bmatrix} \quad \text{and} \quad \mathbf{Y} = \begin{bmatrix} \texttt{target 0} \\ \vdots \\ \texttt{target N} \end{bmatrix}, \tag{4}$$

where `source` and `target` represents the inputs and expected outputs (*e.g.*, a sentence in one language as the source and its translation as the target).

The transformer thus has to construct the target from scratch, whereas the decoder-only models build on the input by iteratively predicting the next word. This auto-regressive (*i.e.*, iterative) nature is also present in the encoder-decoder models, but here needing to generate the output from scratch (based on the input) instead of the inputs continuation.

QUERY, KEY AND VALUE

When calculating what to attend based on some input, $\mathbf{X}$, three new quantities; query, key and value, are introduced, respectively;

$$\begin{aligned} \mathbf{Q} &= \mathbf{X}\mathbf{W}_Q \\ \mathbf{K} &= \mathbf{X}\mathbf{W}_K, \\ \mathbf{V} &= \mathbf{X}\mathbf{W}_V \end{aligned} \tag{5}$$

along with their respective sets of weights, $\mathbf{W}_i$. In practice, these three matrices are obtained by a single fully-connected layer (`self.c_attn`) which takes the inputs, $\mathbf{X}$, and outputs $\texttt{concat}(\mathbf{Q}, \mathbf{K}, \mathbf{V})$. Thus, the individual quantities may be obtained through:

```
B, T, C = x.size()
q, k, v = self.c_attn(x).split(self.n_embd, dim=2)
k = k.view(B, T, self.n_head, C // self.n_head).transpose(1, 2)
q = q.view(B, T, self.n_head, C // self.n_head).transpose(1, 2)
v = v.view(B, T, self.n_head, C // self.n_head).transpose(1, 2)
```

where `self.n_embed` and `self.n_head` are the dimension of the embeddings and number of attention heads, respectively.

The matrices containing the queries, keys and values can thus be seen as three separate representations of the inputs, which is used to enhance or diminish certain aspects of the context when predicting the next element of the sequence. An important note here, is to not forget the encoder-decoder connections. More on this below.

While *Geometry of Deep Learning* tries to provide a biological analogy as to what the query and key represents, its actual representation in terms of the transformer is rather abstracted [16, 17].

SIMILARITY SCORE

The score of an arbitrary query column vector, $\mathbf{q}_i$, and the key column vectors $\mathbf{k}_j$ `for` $\mathbf{k}_j \in \mathbf{K}$, is found by taking the dot product between them;

$$\texttt{score}_{i,j} = \mathbf{q}_i \cdot \mathbf{k}_j \qquad \texttt{for } j = 1, \cdots, M, \tag{6}$$

where M is the dimension of $\mathbf{k}$. Here, we get a score for each key element in the sequence $\mathbf{k}_j$ and the chosen query element, $\mathbf{q}_i$. The intuition behind this score is to find the importance of the other elements contained in the sequence with respect to the current element.

These scores are then (typically) scaled by the root of their dimension, M, and normalized using the softmax function such that a probabilistic representation is obtained;

$$\texttt{weighting}_{i,j} = \texttt{softmax}\left(\frac{\mathbf{q}_i \cdot \mathbf{k}_j}{\sqrt{M}}\right) \qquad \texttt{for } j = 1, \cdots, M. \tag{7}$$

Which in simple terms explain how much of the previous words should be added to the current word, as mentioned above.

Similarity functions

While the mentioned approach using the scaled dot product is the most common [16, 18, 17, 19], other functions for calculating the score may be used. Other such functions include the non-scaled dot product, $\mathbf{q}_i \cdot \mathbf{k}_j$, and the cosine similarity, $\frac{\mathbf{q}_i \cdot \mathbf{k}_j}{||\mathbf{q}_i|| ||\mathbf{k}_j||}$, as presented in *Geometry of Deep Learning* [16].

Masking

Intuitively, (during training,) the attention scores of the decoder need to be masked in order to prevent the influence of future words in preceding words. That is, when a model is generating a sequence, it does not know the future generated words before having generated them.

Relating this to Equation (3), the second element of $\mathbf{x}$ (*i.e.*, `my`) does not contain any information about the succeeding words (`dairy`, *etc.*), but may contain information about preceding words (`At`).

Therefore, when training, the scores of all words after the "current" word is set to $-\infty$, such that it is lost when `softmax` is performed;

$$\texttt{score} = \begin{bmatrix} \texttt{score}_{0,0} & \dots & -\infty \\ \vdots & \ddots & \vdots \\ \texttt{score}_{N,1} & \dots & \texttt{score}_{N,M} \end{bmatrix}. \tag{8}$$

This can be seen in practice through Listing 4.5. Note that this is only needed to be done in the first step of the decoder, seen in Figure 4.

When the softmax score has been calculated, it is multiplied with the value representation of the input, $\mathbf{V}$, much like the gates in the LSTM which enhance or diminish the focus on certain elements of the sequence. The output of the attention layer for element i is then the sum of all j products. That is;

$$\begin{aligned} \texttt{attention}_i &= \sum_{j=1}^{N} \texttt{weighting}_{i,j} \mathbf{v}_j \\ &= \sum_{j=1}^{N} \texttt{softmax}\left(\frac{\mathbf{q}_i \cdot \mathbf{k}_j}{\sqrt{M}}\right) \mathbf{v}_j \qquad \texttt{for } j = 1, \cdots, M. \end{aligned} \tag{9}$$

Which can be rewritten in terms of matrix notation [1]:

$$\texttt{attention} = \texttt{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{M}}\right)\mathbf{V}. \tag{10}$$

This allows the transformer model to selectively focus on different parts of the input sequence when generating each output element, improving its ability to capture long-range dependencies and context compared to just transferring the cell state C as is done in LSTMs.

PYTHON EQUIVALENCE OF ATTENTION

> The code implementation of Equation (10) here presented, with masking (Equation (8)):

```
score = (q @ k.transpose(-2, -1)) / math.sqrt(N)
mask = torch.tril(torch.ones(M, M))
score = score.masked_fill(mask[:T, :T] == 0, float('-inf'))
score = torch.nn.functional.softmax(score, dim=-1)
attention = score @ v
```

PARALLELIZATION

> Unlike recurrent neural networks, which process input tokens one at a time, the attention mechanism calculates the weighted sum of value vectors for each input by considering all key and value vectors at once, enabling parallel computation, as seen in Equation (10).

MULTI-HEAD ATTENTION

> Multi-head attention is an extension of attention that allows the model to attend each position of the sequence simultaneously. This is achieved by linearly projecting the input tokens into multiple attention heads, computing the attention scores for each head independently, and then concatenating the results. This process enables the model to capture a diverse range of contextual information and improve its overall understanding of the input sequence [1].

Architecture

The transformer architecture consists of two primary components: the encoder and the decoder, both of which contain multiple layers stacked after each other. The encoder and decoder can be seen in the left and right part of Figure 4, respectively.

While the architecture displayed in Figure 4 is the originally presented transformer [1, 20], variations of the core architecture which aim at overcoming certain disadvantages can be found throughout the literature (such as [21, 22, 23, 24, 25], among countless others).

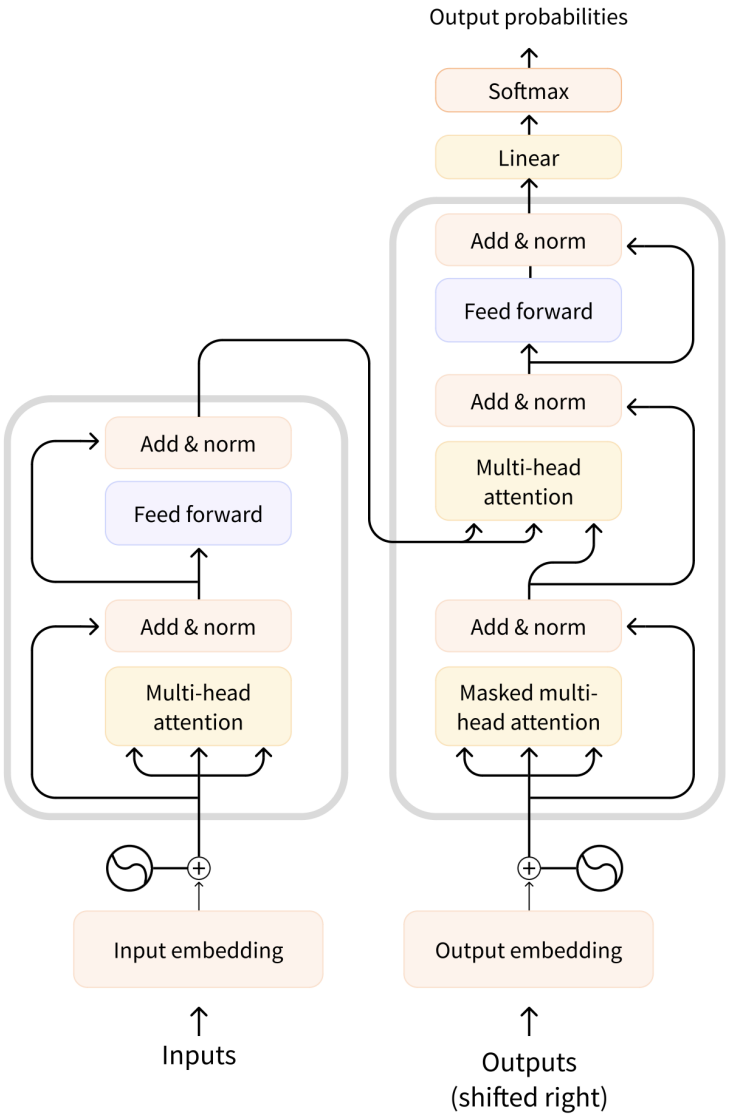


Figure 4: Transformer architecture (from Huggingface [20]).

POSITIONAL ENCODING

Since the transformer does not inherently consider the position or order of input tokens, positional encodings are added to provide this information. These encodings can be learned or fixed, and are added to the input embeddings before being passed through the model (visualized in Figure 4 right after the input- and output embedding) [1]. Positional encoding thus allow the model to maintain a sense of sequence order, which is important for understanding the context and relationships between tokens in the input data.

As seen in *Attention Is All You Need* [1], the positional encoding was done similarly to the code below, with modifications, by Karpathy [19].

```python
class PositionalEncoding(torch.nn.Module):
    def __init__(self, n_embd: int, dropout: float, maxlen: int = 5000):
        super().__init__()

        den = torch.exp(
            -torch.arange(0, n_embd, 2) * math.log(10000) / n_embd
        )
        pos = torch.arange(0, maxlen).reshape(maxlen, 1)

        pos_embedding = torch.zeros((maxlen, n_embd))
        pos_embedding[:, 0::2] = torch.sin(pos * den)
        pos_embedding[:, 1::2] = torch.cos(pos * den)
        pos_embedding = pos_embedding.unsqueeze(-2)

        self.dropout = torch.nn.Dropout(dropout)
        self.register_buffer('pos_embedding', pos_embedding)

    def forward(self, x: torch.Tensor):
        return self.dropout(x + self.pos_embedding[:x.size(0), :])
```

While there are many ways to encode positions, the reasoning behind the chosen sinusoidal encoding is that "[t]he wavelengths form a geometric progression from 2π to $10000 \times 2\pi$[,] [...] because we hypothesized it would allow the model to easily learn to attend by relative positions, since for any fixed offset k, [$\texttt{pos\_embedding}_{\text{pos}+k}$] can be represented as a linear function of [$\texttt{pos\_embedding}_{\text{pos}}$]." [1]

ENCODER

The encoder takes a sequence of (*e.g.*, tokenized word) position embeddings as input, processes all elements of the sequence simultaneously using attention, and outputs a new sequence of encoded vectors that represent the input sequence with the added context of the full sequence.

Here, the name "encoder" becomes apparent, as the function of the encoder is to encode some input sequence to a representation containing the context important for each of the words. In order to achieve this, both a multi-head attention mechanism and a position-wise feed-forward network is used, the latter being a fully connected neural network applied independently to each position in the sequence [1], as seen in Figure 4.

The output of the encoder (and what is used by the decoder) are therefore the $\mathbf{K}$ and $\mathbf{V}$ matrices previously discussed, which is used in the attention of the decoder – as can be seen in Figure 4.

DECODER

The decoder, responsible for generating output given some input, initiates the prediction process with a special beginning-of-sequence (BOS) token, which is commonly used in encoder-decoder models as no output has been generated yet. The decoder's attention mechanism in the second layer takes into account the

encoded **K** and **V**. The three sub-layers of the decoder being; a masked multi-head attention mechanism, an encoder-decoder attention mechanism, and a position-wise feed-forward network, as seen in Figure 4.

During the prediction of a new token, the previously predicted tokens serve as input to the decoder, along with the encoder output, in an auto-regressive manner. This process continues sequentially until the desired output sequence length is reached or an end-of-sequence (EOS) token is predicted.

Special tokens

Special tokens, such as BOS and EOS, are commonly added to the training data to help the model identify the start and end of a sentence during the training process. This leads to a more natural final model, as it can determine when to end a sequence. However, during experimentation, some of the trained models does not contain special tokens. As a result, these models predict a fixed number of tokens and do not stop naturally on their own, as can be observed in the results section.

Masked multi-head attention

As explained when introducing attention, masking ensure that the predictions for position i are only dependent on positions before i. This is necessary during training to prevent the model from "cheating" by looking at the sequence in its entirety (thus "knowing" which word it is supposed to predict). The mask therefore ensures that the model only attends to previous positions when generating each token of the output sequence. [1] Intuitively, this masking has no effect when generating new text, but is important during training (where the output is known).

This masking may, for instance, be applied in the following manner, (as opposed to the alternative approach in Listing 4.5,) based on [20]:

```
def square_mask(self, dim):
    mask = (
        torch.tril(torch.ones((dim, dim),
        device=self.config.device)) == 1
    )
    mask = mask.float().masked_fill(
        mask == 0,
        float('-inf')
    ).masked_fill(mask == 1, float(0.0))
    return mask
```

Where the `square_mask` method creates a square mask of a given dimension. This mask is used to prevent future tokens from being used in the prediction of the current token during training. The method takes an integer `dim` as input, which represents the dimension of the square mask. It then creates a square matrix of ones with the same dimension, and applies the `torch.tril` function to it, which returns the lower triangular part of the matrix, and the values are replaced with $-\infty$ for zeros and 0.0 for ones, as per [1].

The `masking` method creates four different masks for the `src` (source) and `tgt` (target) data: the source mask, target mask, source padding mask, and target padding mask. The source and target masks are square masks created using the `square_mask` method above, with dimensions equal to the sequence lengths of the source and target data, respectively. The padding masks are created by comparing the source and target data to the padding token (here fetched from the tokenizer model), and transposing the resulting boolean tensors.

```
def masking(self, src, tgt):
    src_seq_len = src.shape[0]
    tgt_seq_len = tgt.shape[0]

    tgt_m = self.square_mask(tgt_seq_len).to(self.config.device)
    src_m = torch.zeros(
        (src_seq_len, src_seq_len),
```

```
        device=self.config.device
    ).type(torch.bool)

    src_pad_m = (
        src == self.config.tokenizer["special_symbols"]["[PAD]"]
    ).transpose(0, 1).to(self.config.device)
    tgt_pad_m = (
        tgt == self.config.tokenizer["special_symbols"]["[PAD]"]
    ).transpose(0, 1).to(self.config.device)

    return src_m, tgt_m, src_pad_m, tgt_pad_m
```

These masks are then used by the model to ignore padding tokens in the input data, and to ensure that the prediction for each token is only dependent on the previous tokens in the sequence.

Encoder-decoder attention

Seen in Figure 4 where the output of the encoder (left) is connected to the decoder (right). The attention mechanism here allows the decoder to focus on different parts of the input sequence when generating each token in the output sequence. It does this by combining the output of the encoder (namely, the **K** and **V**) and the output of the masked multi-head attention mechanism (**Q**) as input, thus being able to consider both the context of the input sequence and the previously generated tokens in the output sequence.

The decoder's output is a sequence of vectors, each of which represents a token in the output sequence. The final linear layer and softmax function are then applied to each vector to produce a probability distribution over the target vocabulary, which is used to generate the output sequence.

Each of these sub-layers also includes a residual connection (or skip connection) around it, followed by layer normalization [1].

ATTENTION

See section regarding attention above for the theoretical explanation of attention.

MULTI-LAYER PERCEPTRON

In the transformer architecture seen in Figure 4, each attention block also consists of a feed-forward layer. This layer, often referred to as "multi-layer perceptron" (MLP) can be represented through the class:

```
class MLP(nn.Module):
    def __init__(self, config):
        super().__init__()
        self.c_fc = nn.Linear(config.n_embd, 4 * config.n_embd,
                              bias=config.bias)
        self.gelu = nn.GELU()
        self.c_proj = nn.Linear(4 * config.n_embd, config.n_embd,
                                bias=config.bias)
        self.dropout = nn.Dropout(config.dropout)

    def forward(self, x):
        x = self.c_fc(x)
        x = self.gelu(x)
        x = self.c_proj(x)
        x = self.dropout(x)
        return x
```

Which each element is passed through after the attention mechanism has been performed, such that each block can be concisely contained in the class:

```
class Block(nn.Module):
    def __init__(self, config):
        super().__init__()
        self.ln_1 = LayerNorm(config.n_embd, bias=config.bias)
        self.attn = Attention(config)
        self.ln_2 = LayerNorm(config.n_embd, bias=config.bias)
        self.mlp = MLP(config)

    def forward(self, x):
        x = x + self.attn(self.ln_1(x))
        x = x + self.mlp(self.ln_2(x))
        return x
```

The two classes being fetched from Karpathy [19].

THE TRANSFORMER IN ACTION

See the GIF from [17] for a great visual illustration of the transformer in action.

See [26, 27, 28, 19, 20] among others for further implementations of the transformer.

References

[1] Ashish Vaswani et al. *Attention Is All You Need*. 2023. arXiv: 1706.03762 [cs.CL].

[2] Alexey Dosovitskiy et al. *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale*. 2021. arXiv: 2010.11929 [cs.CV].

[3] Cecile Liu. *Why LSTM cannot prevent gradient exploding?* 2019. URL: https://medium.com/@CecileLiu/why-lstm-cannot-prevent-gradient-exploding-17fd52c4d772.

[4] Andrej Karpathy. *minbpe*. 2024. URL: https://github.com/karpathy/minbpe.

[5] Thomas Wolf et al. "Transformers: State-of-the-Art Natural Language Processing". In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Online: Association for Computational Linguistics, Oct. 2020, pp. 38–45. URL: https://www.aclweb.org/anthology/2020.emnlp-demos.6.

[6] Wikipedia contributors. *Byte pair encoding*. 2024. URL: https://en.wikipedia.org/wiki/Byte_pair_encoding.

[7] OpenAI. *tiktoken*. 2024. URL: https://github.com/openai/tiktoken.

[8] Wilson L. Taylor. ""Cloze Procedure": A New Tool for Measuring Readability". In: *Journalism Quarterly* 30.4 (1953), pp. 415–433. DOI: 10.1177/107769905303000401. eprint: https://doi.org/10.1177/107769905303000401. URL: https://doi.org/10.1177/107769905303000401.

[9] TensorFlow. *word2vec*. 2024. URL: https://www.tensorflow.org/text/tutorials/word2vec.

[10] Swimm. *What Is Word2Vec and How Does It Work?* URL: https://swimm.io/learn/large-language-models/what-is-word2vec-and-how-does-it-work.

[11] Lisa Zhang. *GloVe vectors*. URL: https://www.cs.toronto.edu/~lczhang/321/lec/glove_notes.html.

[12] George R. Mangun. *The Neuroscience of Attention: Attentional Control and Selection*. Oxford University Press, Jan. 2012. ISBN: 9780195334364. DOI: 10.1093/acprof:oso/9780195334364.001.0001. URL: https://doi.org/10.1093/acprof:oso/9780195334364.001.0001.

[13] Rick Merritt. *What is a Transformer Model?* 2022. URL: https://blogs.nvidia.com/blog/what-is-a-transformer-model/.

[14] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. *Neural Machine Translation by Jointly Learning to Align and Translate*. 2016. arXiv: 1409.0473 [cs.CL].

[15] Alec Radford et al. "Language Models are Unsupervised Multitask Learners". In: (2019).

[16] Jong Chul Ye. *Geometry of Deep Learning: A Signal Processing Perspective*. Springer Nature Singapore, 2022. ISBN: 9789811660467. DOI: https://doi.org/10.1007/978-981-16-6046-7.

[17] Jay Alammar. *The Illustrated Transformer*. 2018. URL: jalammar.github.io/illustrated-transformer/.

[18] Mary Phuong and Marcus Hutter. *Formal Algorithms for Transformers*. 2022. arXiv: 2207.09238 [cs.LG].

[19] Andrej Karpathy. *nanoGPT*. 2023. URL: https://github.com/karpathy/nanogpt.

[20] Huggingface. *How do Transformers work?* URL: https://huggingface.co/learn/nlp-course/chapter1/4.

[21] Jacob Devlin et al. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2019. arXiv: 1810.04805 [cs.CL].

[22] Albert Gu and Tri Dao. *Mamba: Linear-Time Sequence Modeling with Selective State Spaces*. 2023. arXiv: 2312.00752 [cs.LG].

[23] Tom B. Brown et al. *Language Models are Few-Shot Learners*. 2020. arXiv: 2005.14165 [cs.CL].

[24] Zihang Dai et al. *Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context*. 2019. arXiv: 1901.02860 [cs.LG].

[25] Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. *Reformer: The Efficient Transformer*. 2020. arXiv: 2001.04451 [cs.LG].

[26] PyTorch. *Language Modeling with nn.Transformer and torchtext*. 2024. URL: https://pytorch.org/tutorials/beginner/transformer_tutorial.html.

[27] Sasha Rush et al. *The Annotated Transformer*. 2022. URL: http://nlp.seas.harvard.edu/annotated-transformer/#embeddings-and-softmax.

[28] Kevin Ko. *Transformer: PyTorch Implementation of "Attention Is All You Need"*. 2019. URL: https://github.com/hyunwoongko/transformer/tree/master.

[29] Colin Raffel et al. "Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer". In: *Journal of Machine Learning Research* 21.140 (2020), pp. 1–67. URL: http://jmlr.org/papers/v21/20-074.html.

Results

The following pages are notebooks displaying the results of the various trained models. A full implementation of the training-algorithms can be found in the source code.

1 Oversettelse fra norsk bokmål til nynorsk

OBS: Ekstremt dårlig datasett. Derfor dårlig modell.

Hyperparameters:

```
vocab_size: int = 50000
n_encoder_layer: int = 8
n_decoder_layer: int = 12
n_head: int = 8
n_embd: int = 512
dropout: float = 0.1
bias: bool = False

epochs: int = 50
batch_size: int = 128
device: str = "cuda" if torch.cuda.is_available() else "cpu"

from_lang: str = "nb"
from_path: str = "NbAiLab/norwegian-paws-x"
to_lang: str = "nn"
to_path: str = "NbAiLab/norwegian-paws-x"

loss_fn: torch.nn.CrossEntropyLoss = torch.torch.nn.CrossEntropyLoss(
    ignore_index=self.tokenizer["special_symbols"]["[PAD]"]
)
optimizer: dict = {'lr': 0.0001, 'betas': (0.9, 0.98), 'eps': 1e-09}
output_path: str = "./output/"
```

Transformer architecture:

```
Transformer(
  (transformer): Transformer(
    (encoder): TransformerEncoder(
      (layers): ModuleList(
        (0-5): 6 x TransformerEncoderLayer(
          (self_attn): MultiheadAttention(
            (out_proj): NonDynamicallyQuantizableLinear(
                in_features=512, out_features=512, bias=True
            )
```

```
        )
        (linear1): Linear(in_features=512, out_features=2048, bias=True)
        (dropout): Dropout(p=0.1, inplace=False)
        (linear2): Linear(in_features=2048, out_features=512, bias=True)
        (norm1): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
        (norm2): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
        (dropout1): Dropout(p=0.1, inplace=False)
        (dropout2): Dropout(p=0.1, inplace=False)
      )
    )
    (norm): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
  )
  (decoder): TransformerDecoder(
    (layers): ModuleList(
      (0-7): 8 x TransformerDecoderLayer(
        (self_attn): MultiheadAttention(
          (out_proj): NonDynamicallyQuantizableLinear(
              in_features=512, out_features=512, bias=True
          )
        )
        (multihead_attn): MultiheadAttention(
          (out_proj): NonDynamicallyQuantizableLinear(
              in_features=512, out_features=512, bias=True
          )
        )
        (linear1): Linear(in_features=512, out_features=2048, bias=True)
        (dropout): Dropout(p=0.1, inplace=False)
        (linear2): Linear(in_features=2048, out_features=512, bias=True)
        (norm1): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
        (norm2): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
        (norm3): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
        (dropout1): Dropout(p=0.1, inplace=False)
        (dropout2): Dropout(p=0.1, inplace=False)
        (dropout3): Dropout(p=0.1, inplace=False)
      )
    )
    (norm): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
  )
)
(src_tok_emb): Embedding(
  (embedding): Embedding(50000, 512)
)
(tgt_tok_emb): Embedding(
  (embedding): Embedding(50000, 512)
)
(positional_encoding): PositionalEncoding(
  (dropout): Dropout(p=0.1, inplace=False)
)
```

```
  (generator): Linear(in_features=512, out_features=50000, bias=True)
)
```

[1]:
```python
import torch

import sys
sys.path.append("..")
```

[2]:
```python
prompts = [
    "Universet",
    "En stol.",
    "Boken min er på bordet.",
    "Jeg liker å sitte på skolen når jeg jobber.",
    "Ved å høre på forelesninger i fysikk ble jeg veldig trøtt.",
    "Hvem kan vel unnslippe døden?",
    "Norges Miljø- og Biovitenskapelige Universitet"
]
models = {
    "10 epochs": torch.load("./model-10.pth", map_location=torch.device('cpu')),
    "30 epochs": torch.load("./model-30.pth", map_location=torch.device('cpu')),
    "60 epochs": torch.load("./model-60.pth", map_location=torch.device('cpu')),
}
```

2 Eksempler fra modellene under trening

[3]:
```python
for state, model in models.items():
    model.config.device = "cpu"

    print(f"Modell: {state}")
    for prompt in prompts:
        print(f" Fra: {prompt}")
        print(f"   > {model(prompt)}")
    print()
```

```
Modell: 10 epochs
 Fra: Universet
   >  Universetet vart sett i ein av tidlegare hus for Universetet
 Fra: En stol.
   >  Ein annan stad i byen. Han var ein stad i det regionale byen.
 Fra: Boken min er på bordet.
   >  Boka er mykje på å sjå på Manchester.
 Fra: Jeg liker å sitte på skolen når jeg jobber.
   >  Eg prøvde å ha ein god på skulen, men eg held han på ein
 Fra: Ved å høre på forelesninger i fysikk ble jeg veldig trøtt.
   >  Ved å arbeide på fysikk på fysikk er eg mykje mykje enn det.
 Fra: Hvem kan vel unnslippe døden?
   >  Det kan vera ut ein måte å sjå ut ut ut når ho kjem ut.
```

```
 Fra: Norges Miljø- og Biovitenskapelige Universitet
   >  Eckhoff brukte eit organisasjon og kulturane som blei sett inn i fengsel
og klosterdelagd.

Modell: 30 epochs
 Fra: Universet
   >  Universet som inkluderer fotstader i Universet Universet Universtaris
 Fra: En stol.
   >  Den 100re borgmesteren i ein plass i Den statlege plassen.
 Fra: Boken min er på bordet.
   >  Boka er på bordet.
 Fra: Jeg liker å sitte på skolen når jeg jobber.
   >  Eg likte å sitjande på skulen når eg jobbar han.
 Fra: Ved å høre på forelesninger i fysikk ble jeg veldig trøtt.
   >  Ved å høyre høyre senter i fysikk var eg kjent.
 Fra: Hvem kan vel unnslippe døden?
   >  Styremedlemmar går frå døden?
 Fra: Norges Miljø- og Biovitenskapelige Universitet
   >  CLpressa og eit lovleg tårn ved vår

Modell: 60 epochs
 Fra: Universet
   >  Universetishet er geet som Universet som Universetis
 Fra: En stol.
   >  Ein kan bli ein statar i skal handla.
 Fra: Boken min er på bordet.
   >  Boka er oms bordet på bordet.
 Fra: Jeg liker å sitte på skolen når jeg jobber.
   >  Eg likar å sitja på skulen når eg jobbar.
 Fra: Ved å høre på forelesninger i fysikk ble jeg veldig trøtt.
   >  Ved å høyre forsking på klinisk forsking vart alle synleg.
 Fra: Hvem kan vel unnslippe døden?
   >  Det kan ta livet av døden å ta resten av farga.
 Fra: Norges Miljø- og Biovitenskapelige Universitet
   >  Tour-kontoret og biografiar ved New York held seg på
```

3 Eksempler fra treningsdata

[4]:
```
dataset, _ = models["10 epochs"]._data()
```

```
python(96737) MallocStackLogging: can't turn off malloc stack logging because it
was not enabled.
huggingface/tokenizers: The current process just got forked, after parallelism
has already been used. Disabling parallelism to avoid deadlocks…
To disable this warning, you can either:
        - Avoid using `tokenizers` before the fork if possible
```

```
        - Explicitly set the environment variable TOKENIZERS_PARALLELISM=(true |
false)
Map:   0%|          | 0/49401 [00:00<?, ? examples/s]
Map:   0%|          | 0/2000 [00:00<?, ? examples/s]
Map:   0%|          | 0/2000 [00:00<?, ? examples/s]
Map:   0%|          | 0/49401 [00:00<?, ? examples/s]
Map:   0%|          | 0/2000 [00:00<?, ? examples/s]
Map:   0%|          | 0/2000 [00:00<?, ? examples/s]
```

[5]:
```
for i in torch.random.default_generator.get_state().tolist()[:9]:
    print(dataset["train"]["sentence1"][i])
    print()
```

```
Pluto ble klassifisert som planeten da Grand Tour ble foreslått og ble lansert
på den tiden " New Horizons ".

Den 31 oktober 2012 kjøpte Watson Actavis og tok Actavis-navnet.

Han har blitt trent av sin bestefar, Nick Dakin, og er nå trent av Geoff
Barraclough.

Han tilbrakte flere år i New York som en datamaskin konsulent og programvare
ingeniør, deretter flyttet til Atlanta i 1998.

Elizabeth II var en forfader til dronninger Edzard II og Beatrix av Nederland.

Hos rotter hemmer det også den sentrale, om ikke perifere, utskillelsen av
oksytocin og vasopressin.

Etter at han ble sparket, flyttet han fra Tyskland til New Mexico og deretter
til Los Angeles, deretter til San Francisco.

Dette er en liste over grotter i Storbritannia, inkludert informasjon om de
største og dypeste grotter i Storbritannia.

NBA-sesongen 1975-76 var den 30. sesongen av National Basketball Association.
```

1 Translation from English to Norwegian

Hyperparameters:

```
vocab_size: int = 65536
n_feedforward: int = 512
n_encoder_layer: int = 3
n_decoder_layer: int = 3
n_head: int = 8
n_embd: int = 512
dropout: float = 0.1
bias: bool = False

epochs: int = 50
batch_size: int = 96
device: str = "cuda" if torch.cuda.is_available() else "cpu"

data_lang: str = "en-no"
data_path: str = "Helsinki-NLP/opus-100"

loss_fn: torch.nn.CrossEntropyLoss = torch.torch.nn.CrossEntropyLoss(
    ignore_index=self.tokenizer["special_symbols"]["[PAD]"]
)
optimizer: dict = {'lr': 0.0001, 'betas': (0.9, 0.98), 'eps': 1e-09}

output_path: str = "./output/"
```

Transformer architecture:

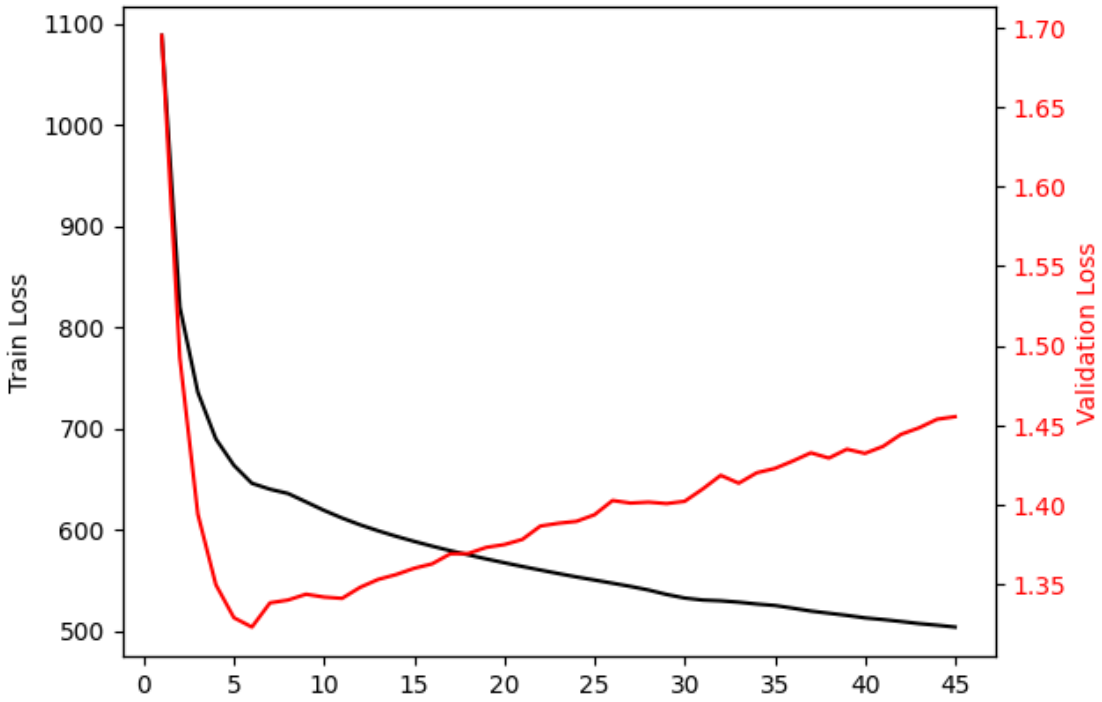
```
Transformer(
  (transformer): Transformer(
    (encoder): TransformerEncoder(
      (layers): ModuleList(
        (0-2): 3 x TransformerEncoderLayer(
          (self_attn): MultiheadAttention(
            (out_proj): NonDynamicallyQuantizableLinear(
              in_features=512, out_features=512, bias=True
            )
          )
```

```
        (linear1): Linear(in_features=512, out_features=512, bias=True)
        (dropout): Dropout(p=0.1, inplace=False)
        (linear2): Linear(in_features=512, out_features=512, bias=True)
        (norm1): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
        (norm2): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
        (dropout1): Dropout(p=0.1, inplace=False)
        (dropout2): Dropout(p=0.1, inplace=False)
      )
    )
    (norm): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
  )
  (decoder): TransformerDecoder(
    (layers): ModuleList(
      (0-2): 3 x TransformerDecoderLayer(
        (self_attn): MultiheadAttention(
          (out_proj): NonDynamicallyQuantizableLinear(
            in_features=512, out_features=512, bias=True
          )
        )
        (multihead_attn): MultiheadAttention(
          (out_proj): NonDynamicallyQuantizableLinear(
            in_features=512, out_features=512, bias=True
          )
        )
        (linear1): Linear(in_features=512, out_features=512, bias=True)
        (dropout): Dropout(p=0.1, inplace=False)
        (linear2): Linear(in_features=512, out_features=512, bias=True)
        (norm1): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
        (norm2): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
        (norm3): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
        (dropout1): Dropout(p=0.1, inplace=False)
        (dropout2): Dropout(p=0.1, inplace=False)
        (dropout3): Dropout(p=0.1, inplace=False)
      )
    )
    (norm): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
  )
)
(src_tok_emb): Embedding(
  (embedding): Embedding(65536, 512)
)
(tgt_tok_emb): Embedding(
  (embedding): Embedding(65536, 512)
)
(positional_encoding): PositionalEncoding(
  (dropout): Dropout(p=0.1, inplace=False)
)
(generator): Linear(in_features=512, out_features=65536, bias=True)
```

)

[1]:
```
import torch
import pandas as pd
import matplotlib.pyplot as plt

import sys
sys.path.append("..")
```

[2]:
```
loss = pd.read_csv('./loss.csv', header=0, index_col=0)

fig, ax1 = plt.subplots()
ax1.plot(loss.index, loss["train_loss"], color='black')
ax1.set_ylabel('Train Loss', color='black')
ax1.tick_params(axis='y', labelcolor='black')

ax2 = ax1.twinx()
ax2.plot(loss.index, loss["val_loss"], color='red')
ax2.set_ylabel('Validation Loss', color='red')
ax2.tick_params(axis='y', labelcolor='red')

plt.xticks(range(0, max(loss.index)+1, 5))
plt.show()
```



[3]:
```
prompts = [
    "Universe.",
    "A chair.",
    "My friend.",
    "My book is on the table.",
    "I like sitting at school when working.",
    "Attending physics-lectures made me sleep.",
    "Who could possibly avoid death?",
    "Yes, well how do you, as an AI, handle such long inputs as this? Are you␣
 ↪able to translate "
    "them? I have not looked in the dataset thoroughly enough to see wheter␣
 ↪there are any long "
    "pairs."
]
models =  {
    "5": torch.load("./model-5.pth", map_location=torch.device('cpu')),
    "10": torch.load("./model-10.pth", map_location=torch.device('cpu')),
}
for model in models.values():
    model.config.device = "cpu"
```

2 Example translations from the untrained and checkpointed models

[4]:
```
for prompt in prompts:
    print(f"\n \033[91m {prompt}\033[0m")
    for which, model in models.items():
        print(f" \033[91m   {which:>2}:\033[0m {model(prompt)}")
```

```
 Universe.
    5:  - Alle sammen.
   10:  - Jeg er verdens beste.

 A chair.
    5:  En stol.
   10:  En stol.

 My friend.
    5:  Vennen min.
   10:  Min venn.

 My book is on the table.
    5:  Min bok er på bordet.
   10:  Min bok ligger på bordet.
```

I like sitting at school when working.
5: Jeg liker å sitte på skolen når jeg jobber.
10: Jeg liker å sitte på skolen når jeg jobber.

Attending physics-lectures made me sleep.
5: Hun gjorde meg til å sove.
10: -Jeg fikk meg til å sove.

Who could possibly avoid death?
5: Hvem kan unngå døden?
10: Hvem kan kanskje unngå døden?

Yes, well how do you, as an AI, handle such long inputs as this? Are you able to translate them? I have not looked in the dataset thoroughly enough to see wheter there are any long pairs.
5: - Hvordan kan du oversette dem?
10: - Hvordan har du det?

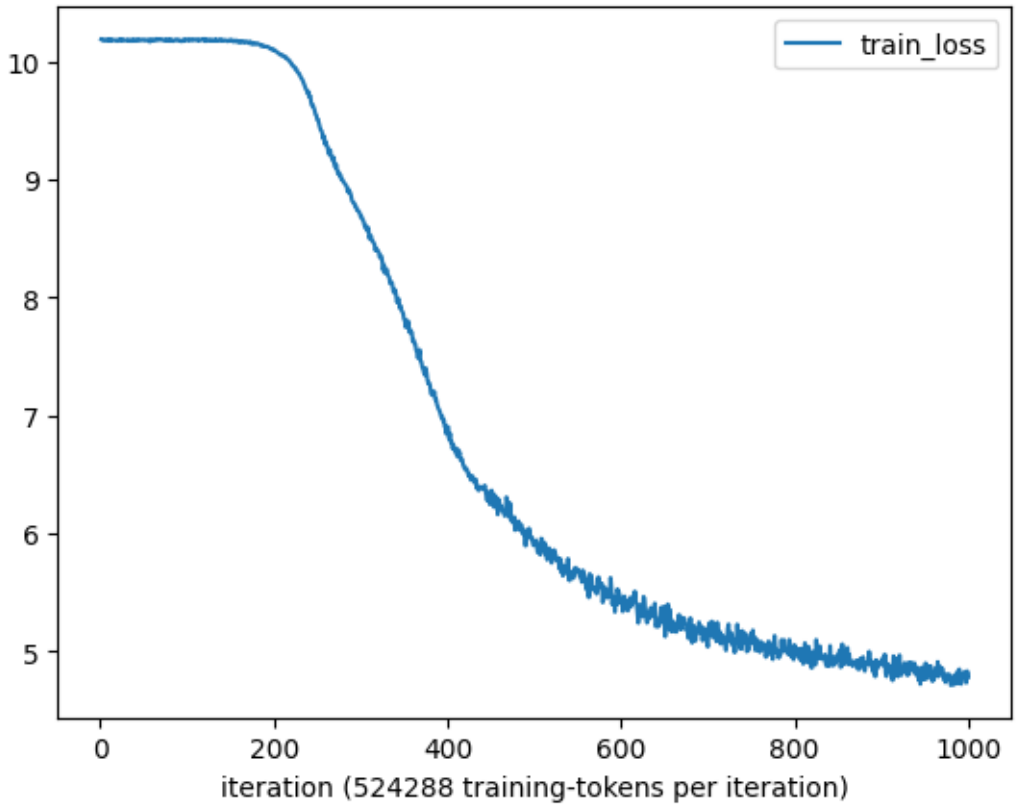
1 Large version of the Don Quixote generative model

Training cancelled at 1000 iterations, see `quixote-small` for the better (smaller) model.

[1]:
```
import torch
import pandas as pd

import sys
sys.path.append("..")
```

[2]:
```
loss = pd.read_csv('loss.csv', header=0, index_col=0, usecols=[0, 1])
loss.index.name = f"iteration ({8 * 4 * 256 * 64} training-tokens per␣
 ↪iteration)"
loss.plot()
```

[2]:
```
<Axes: xlabel='iteration (524288 training-tokens per iteration)'>
```



[3]:
```
model = torch.load("./quixote-large.pth", map_location=torch.device('cpu'))
model.config.device = "cpu"
```

[4]:
```
prompt = ""Here is the apple of which I spoke," said Don Quixote, to which␣
 ↪Sancho replied"
generate = 100
top_k = None
```

[5]:
```
print(f"\n\033[91m Prompt:\033[0m {prompt}")
for temperature in [1.0, 0.8, 0.5, 0.1]:
    print(f"\n\033[91m Temperature:\033[0m {temperature}")
    output = model(prompt, generate=generate, temperature=temperature,␣
 ↪top_k=top_k)
    print(f"\033[91m quixote-large:\033[0m {output} ...")
```

```
 Prompt: "Here is the apple of which I spoke," said Don Quixote, to
which Sancho replied

 Temperature: 1.0
 quixote-large: "Here is the apple of which I spoke," said Don Quixote,
```

to which Sancho replied Don Quixote of bargain." "In death. "Dulcinea," said Sancho, "for I will he were so for all month, and liberty under Rocinante, now in this time, for a heartfelt Dapple without required (for I was or the duchess, but palm of. There is the books that she is no opportunity of hearing the city; and his think that plan be seen a good once, that the king and his power to the rinsings, Sancho; where enchanted love of its Dulcinea del Toboso. …

Temperature: 0.8
quixote-large: "Here is the apple of which I spoke," said Don Quixote, to which Sancho replied Sancho, "but I am an end of the lady, enemy of the feet, and the peerless Dulcinea are all the other, though there will fall and at least may be no urinary out of some other, the farmer; and you are her meadow which she has ever told him, as well to make up too, of my master had been, I would burn me! For no more must listening to dare to the risk, and he said, I am ‘Give me about to see her, …

Temperature: 0.5
quixote-large: "Here is the apple of which I spoke," said Don Quixote, to which Sancho replied Don Quixote was to the same time, "I am not only in the ground, or the duke and so little, I am, and the sake of the first, and the old and the ground and the first, and the others; and the duchess has been made a good for the world, as he was that of the one of the others of the world. Don Quixote, I can do not to the time, and his master of those who, and the host of the landlord, and …

Temperature: 0.1
quixote-large: "Here is the apple of which I spoke," said Don Quixote, to which Sancho replied Don Quixote, "I am a good fortune, and I have been a man, and I have been a great and I have been a great and I have been a good, and I have been a man of the same time to be a little, and I have been a great and I have been a little to the same time to be a good, and I have been a knight, and I have been a good as I am a great and I have been a man of the world. …

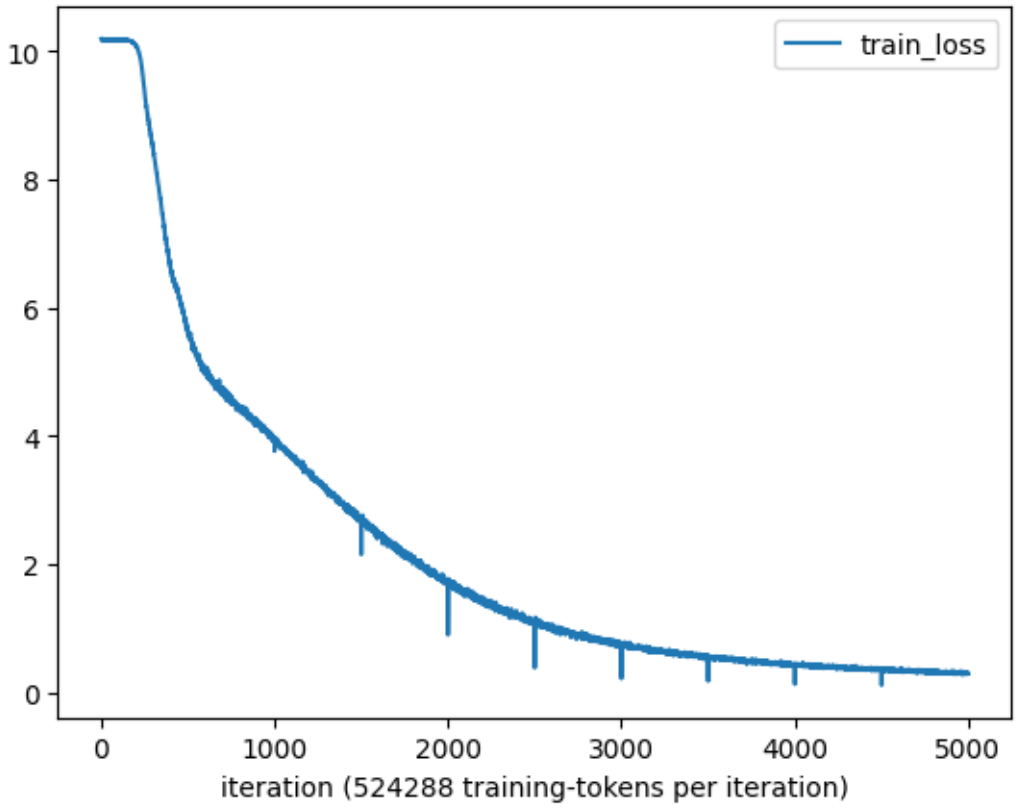
1 Small version of the Don Quixote generative model

See `quixote-large` for a bigger version (stopped early during training).

[1]:
```
import torch
import pandas as pd

import sys
sys.path.append("..")
```

[2]:
```
loss = pd.read_csv('loss.csv', header=0, index_col=0)
loss.index.name = f"iteration ({8 * 4 * 256 * 64} training-tokens per␣
 ↪iteration)"
loss.plot(y="train_loss")
```

[2]:
```
<Axes: xlabel='iteration (524288 training-tokens per iteration)'>
```



[3]:
```
models = {
    "2000": torch.load("./model-2000.pth", map_location=torch.device('cpu')),
    "3000": torch.load("./model-3000.pth", map_location=torch.device('cpu')),
    "4000": torch.load("./model-4000.pth", map_location=torch.device('cpu')),
    "5000": torch.load("./model-5000.pth", map_location=torch.device('cpu')),
}
for model in models.values():
    model.config.device = "cpu"
```

[4]:
```
prompts = [""Here is the apple of which I spoke," said Don Quixote,",
           ""My name is Hallvard, and I'm a student.""]
temperatures = [1.0, 0.8, 0.5]
generate = 100
top_k = None
```

[5]:
```
for prompt in prompts:
    print(f"\n\033[91m Prompt:\033[0m {prompt}")
    for temperature in temperatures:
        print(f"\n\033[91m Temperature:\033[0m {temperature}")
        for which, model in models.items():
```

```
            output = model(prompt, generate=generate, temperature=temperature,␣
↪top_k=top_k)
            print(f"\033[91m {which}:\033[0m {output} ...")
```

Prompt: "Here is the apple of which I spoke," said Don Quixote,

Temperature: 1.0
2000: "Here is the apple of which I spoke," said Don Quixote, "is it rare and pat to the purpose that you would gladly feel something to eat and labour to judge's pledge.'" "At any rate," observed Sancho, "and I might tell you more time without waiting for quarrel, as far from your worship is next to be marjoram and neither got up." "Now I, señora," said the price, "for you value on a heap of me." "I do not know not tell you, Sancho," said the curate, "that you had not …
3000: "Here is the apple of which I spoke," said Don Quixote, "for I entreat you to call you your enemy to allow the expression of my own person in high degree, though I have a respect as my taste and at least to retain me, and support myself for even and even were it been the privilege of beauty to be breast without showing myself in the chaste fidelity of beauty;" and with great rage he began to weep bitterly, and with a heavy penalties upon them, that was prevented them from falling to arm and to fly. The duke came with a couple …
4000: "Here is the apple of which I spoke," said Don Quixote, "and I should not give crumbs to a cat, my wits are so confused and upset." With this permission Sancho said to the peasants who stood clustered round him, waiting with open mouths for the decision to come from his, "Brothers, what the fat man requires is not in reason, nor has it a shadow of justice in it; because, as they say, if it be true, that the challenged may choose the weapons, the other has no right to choose such as will prevent and keep …
5000: "Here is the apple of which I spoke," said Don Quixote, "and I cannot call this service to you, no one or mine ever so great or divine as to be brought before God's will." "I am afraid of," said Sancho, "but your worship will let me eat this evening or a cudgel, and not a squire's curse." Don Quixote was listening to all such good subjects, he took the cloth upon Sancho to carry on his heart as to his master; and so much to carry out the scrutiny he had made of them, and of …

Temperature: 0.8
2000: "Here is the apple of which I spoke," said Don Quixote, "for those knocks on the head has been said, that you would not be valid, and even though it excited in a rich farmer to rise to each way, he resolved to go looking for someone to-day. Your servant was astonished to see whether the Knight of the Rueful Countenance, so loud that when he heard him speak from his knees and not a message from Montesinos, which the duke's letter achieved, if so rationally that the duke and duchess were to hear them. Among the duke and duchess …
3000: "Here is the apple of which I spoke," said Don Quixote, "and I

should not give infinite mercy of my eye, and the injury has been done to you;" and so saying he began to move from that position, and that time he might have been weary with the oath he had sworn falsely, and by the same law he swore to die on that he should return; and by the faith of what offences he was going to such a sorry business. He made right mind to know what Tom Cecial?" "How now, Sancho," answered Don Quixote; " …

4000: "Here is the apple of which I spoke," said Don Quixote, "for no one of the eye, and I he won't give any pleasure in more favourable answer from the lady Dulcinea del Toboso, for it is to me of the way of endeavouring to become an emperor, or at least a monarch; for it had been so settled between them, and with his personal worth and the might of his arm it was an easy matter to come to be one: and how on becoming one his lord was to make a marriage for him (for he would be a widower …

5000: "Here is the apple of which I spoke," said Don Quixote, "and I cannot call myself fifth (which may be included in the end of your guest's defence of my honour. To these five, as it were capital causes, there may be added some others that may be just and reasonable, and make it a duty to take up arms; but to take them up for trifles and things to laugh at and be amused by rather than offended, looks as though he who did so was altogether wanting in common sense. Moreover, to take an unjust revenge (and …

Temperature: 0.5

2000: "Here is the apple of which I spoke," said Don Quixote, "if thou wilt not give it to me a beard of my poor, for I am not fit to give thee a punch that will leave my fist sunk in his skull; for cirimonies and soapings of this sort are more like jokes than the polite attentions of one's host." As soon as he was ready to die with his laughter; but Don Quixote to die with his laughter at his stirrups and pressed his morion; but, as has been said, he came out to him a couple of slaps on …

3000: "Here is the apple of which I spoke," said Don Quixote, "and I entreat you to go to rest assured that you are knights-errant to return to El Toboso and rest assured that he cannot lie closer in this castle; and if I am willing to condemn myself in however, let it pass over for this bed in costs little more than putting it over minute." He approved of what Sancho proposed, and resolved to wait for him very anxious to follow him until he brought back. Sancho pushed on into the glens of the Sierra, leaving them in one through which there …

4000: "Here is the apple of which I spoke," said Don Quixote, "and I entreat you to open your eyes, for I shall not be able to make a moment in the apple of your highness's eye, and I'd as soon stab myself as consent to it; for though my master says that in civilities it is better to lose by a card too many than a card too few, when it comes to civilities to asses we must mind what we are about and keep within due bounds." "Take him to your government, Sancho," said the duchess, "and …

5000: "Here is the apple of which I spoke," said Don Quixote, "for no one of my eye, and I cannot call it; but though in truth to be told, I ought not to be censured. The first is some words that I have read in the preface; the next that the language is Aragonese, for sometimes he writes without articles;

and the third, which above all stamps him as ignorant, is that he goes wrong and departs from the truth in the most important part of the history, for here he says that my squire Sancho Panza's wife is …

Prompt: "My name is Hallvard, and I'm a student."

Temperature: 1.0
2000: "My name is Hallvard, and I'm a student." To which the duke made no answer, opening him in a strain he was making a very reverse of a person of rank. They called to him, and asked the curate if he had told them all to be strippedSancho, he found a question by breaking his mercies he listened to even if they had received of Clara." On hearing these words received her of her face hastily and ran thus: "Ill luck betide thee, I swear heartily and compassion for thee, Sancho!" said Don Quixote, "Worthy …
3000: "My name is Hallvard, and I'm a student." "What are you perchance?" said Don Quixote. "What, being entirely," replied Sancho, "your worship is to blame for you left buried (for I never saw a head of it), and went back to the ass was not very fond of him, as he can be always natural and proper to know when he calls himself, protects and prizes a famous castle of famous as you are caught in the noose of fineness, it is one of envy itself, and the twelve more exposed himself to the …
4000: "My name is Hallvard, and I'm a student." With this the words Don Quixote turned his pike to Sancho and said to him, "It is my belief, señor, that I suppose the ass was born to be found me take a tavern, and not on the grandest occasion the past or present has seen, or the future can hope to see. If my wounds have no beauty to the beholder's eye, they are, at least, honourable in the estimation of those who know where they were received; for the soldier shows to greater advantage dead in …
5000: "My name is Hallvard, and I'm a student." With this permission Sancho made him, and he pacified his master and urging him earnestly to accompany him. The bachelor, to accompany him, and they at having ordered him to accompany him whither Don Quixote, took him; and their departure, Sancho afterwards answered very much the reply, Don Quixote asked the servants who were all earnestly and in the Moorish lady's father's house, as they were not a Christian gentleman who had been seen by this time. "Anything else, Christian, I might hope for or …

Temperature: 0.8
2000: "My name is Hallvard, and I'm a student." "And yet, mother," said Don Quixote, "and I do not mean to be able to do any squire that has carried him away from some oak-galls show; for the peerless Clavileño the pair of those that direct male and not very gravely and solemnly as solemnly as fair lady, "Worthy duenna, only enjoin hand ye cannibals;" and not open your feet to her hands, but only open your eyes? But I will only so hold your mouth, shall only open your eyes and let her all …
3000: "My name is Hallvard, and I'm a student." "Well, sirs," said Don Quixote, "but you see how worthy folk have never brought you to this minute, for if they did not open a lips or even-but never mind-it only thing it occurs to me

that I say when they say of the blow of the stone." "I swear to thee!" said Sancho Panza. "that now seated on the pack-saddle softer than a tie it to a tree, and with a couple of dozen carobs and as many more filberts and walnuts; thanks …

4000: "My name is Hallvard, and I'm a student." "That is the true," said Don Quixote; "say what thou wilt, only say, then?" "Well, here I have another mole on the middle of your backbone, which is the mark of a strong man." "That is enough," said Sancho; "for with friends we must not look too closely into trifles; and whether it be on the shoulder or on the backbone matters little; it is enough if there is a mole, be it where it may, for it is all the …

5000: "My name is Hallvard, and I'm a student." Hereupon, smiling slightly, Don Quixote exclaimed, "Lion-whelps to me! to me whelps of lions, and at such a time! Then, by God! those gentlemen who send them here shall see if I am a man to be frightened by lions. Get down, my good fellow, and as you are the keeper open the cages, and turn me out those beasts, and in the midst of this plain I will let them know who Don Quixote of La Mancha is, in spite and in the …

Temperature: 0.5

2000: "My name is Hallvard, and I'm a student." "There is a bad Christian," said Don Quixote; "for if the name of my name is nothing of the proverb that says ‘what difference between my lord,’ and ‘the Pope, for though he is not a woman like a countess, but a woman’s daughter does not look for her a wife; and ‘he who knows that ‘the ‘the fool knows her?’ ‘he who has the mother,’ applies. Thou, thou, to my thinking, art venturing to it, to it …

3000: "My name is Hallvard, and I'm a student." "That I have already said on my part," said the duke, "but you see how worthy gentleman is a very fat man, and he who is a married with a lining to match, and I know not what trimmings of impertinence and roguery? Who asked thee to meddle in my affairs, or to inquire whether I am a wise man or a blockhead? Hold thy peace; saddle Rocinante if he be unsaddled; and let us go to put my offer into execution; for with the right that I …

4000: "My name is Hallvard, and I'm a student." The lion has a close at last moment, as he was about to answer a word, not which was his companion carried upon him, but as he did not dare to suggest to him, fearing that he would not consent to do so; not because he did not know perfectly well the rank, goodness, virtue, and beauty of Luscinda, and that she had qualities that would do honour to any family in Spain, but because I was aware that he did not wish me to marry so soon, before …

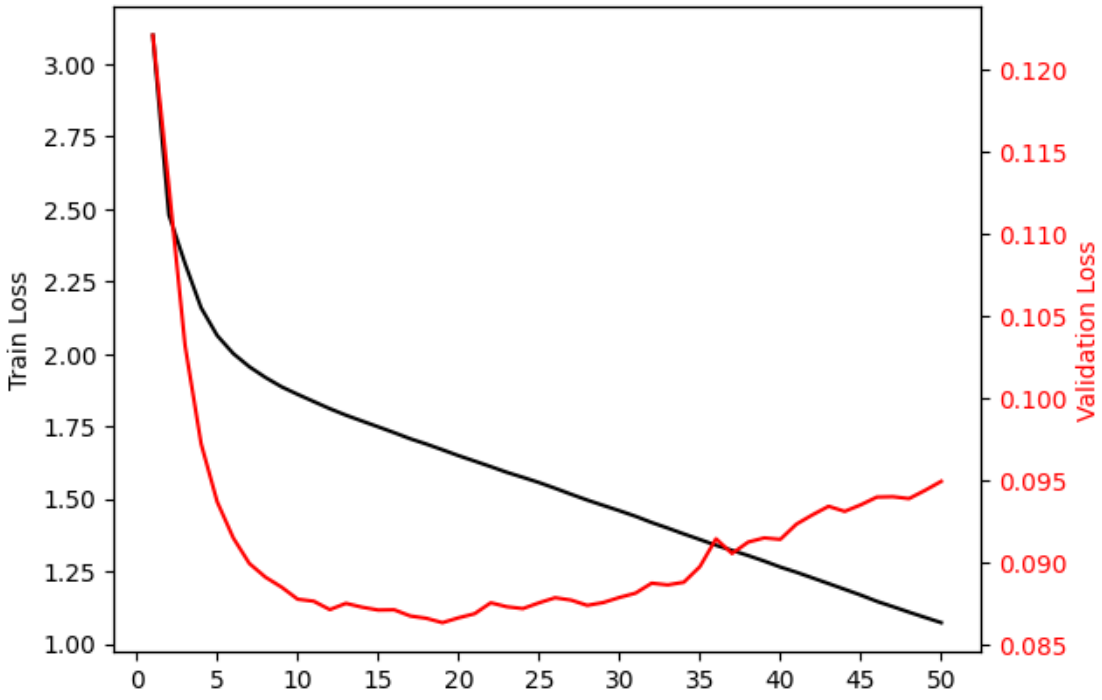
5000: "My name is Hallvard, and I'm a student." With this, at last words Don Quixote, proceeded to say to him, "In the height of misadventure, is to fall in with you who would have been sure to take these adventures; so that, when he is a man who has something in his senses, has turned into a piece of bread and cheese; and seeing what my lady Dulcinea says, what my belief is that he would turn and feel the risk of death for another thou hast been going about in the depths of thy thoughts." " …

1 Small version of the Don Quixote generative model

Predicts the next sentence(s) instead of an infinite amount of text.

[1]:
```
import torch
import pandas as pd
import matplotlib.pyplot as plt

import sys
sys.path.append("..")
```

[2]:
```
loss = pd.read_csv('./loss.csv', header=0, index_col=0)

fig, ax1 = plt.subplots()
ax1.plot(loss.index, loss["train_loss"], color='black')
ax1.set_ylabel('Train Loss', color='black')
ax1.tick_params(axis='y', labelcolor='black')

ax2 = ax1.twinx()
ax2.plot(loss.index, loss["val_loss"], color='red')
ax2.set_ylabel('Validation Loss', color='red')
ax2.tick_params(axis='y', labelcolor='red')

plt.xticks(range(0, max(loss.index)+1, 5))
plt.xlabel('Epoch')
plt.show()
```



[3]:
```
models = {
    "10": torch.load("./model-10.pth", map_location=torch.device('cpu')),
    "15": torch.load("./model-15.pth", map_location=torch.device('cpu')),
    "20": torch.load("./model-20.pth", map_location=torch.device('cpu')),
    "25": torch.load("./model-25.pth", map_location=torch.device('cpu')),
    "30": torch.load("./model-30.pth", map_location=torch.device('cpu')),
    "40": torch.load("./model-40.pth", map_location=torch.device('cpu')),
}
for model in models.values():
    model.config.device = "cpu"
```

[4]:
```
prompts = [""Here is the apple of which I spoke," said Don Quixote,",
           ""My name is Hallvard, and I'm a student.""]
maximum = 200
```

[5]:
```
for prompt in prompts:
    for which, model in models.items():
        output = model(prompt, maximum)
        print(f"\033[91m {which}:\033[0m \033[92m {prompt}\033[0m {output}")
    print()
```

```
 10:  "Here is the apple of which I spoke," said Don Quixote,
"I have been said Sancho," said Don Quixote; "for I have been a good as I have
been a man of the world."
```

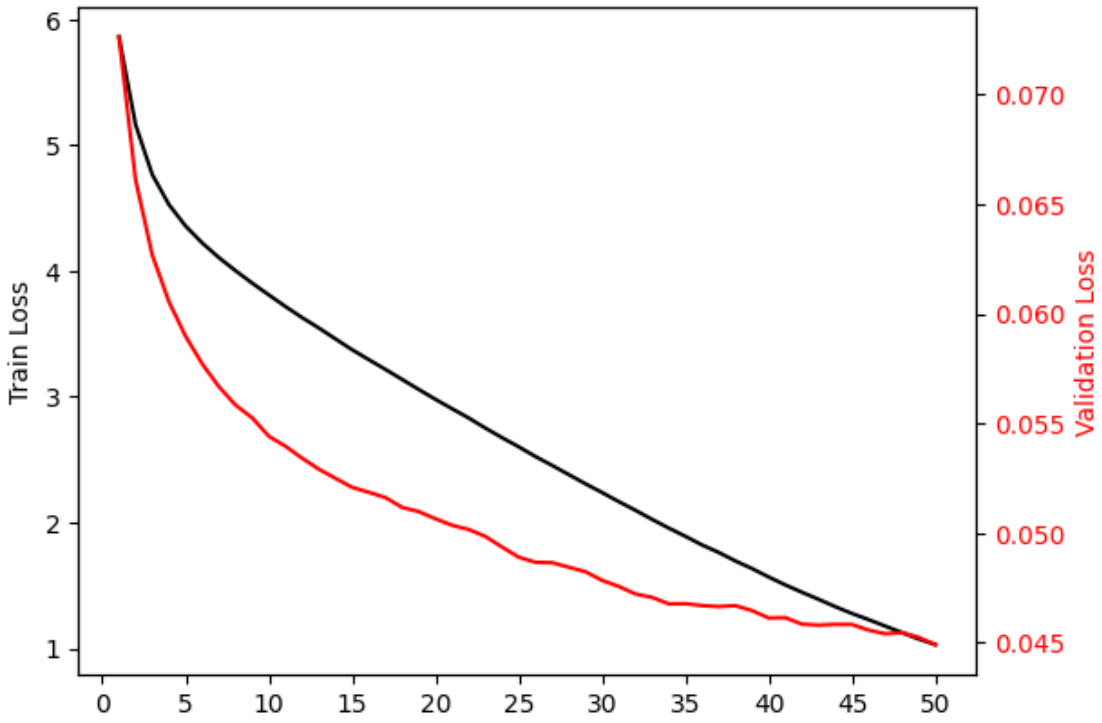
```
 15:  "Here is the apple of which I spoke," said Don Quixote,
"I am not," said Don Quixote, "for I am not to be so much as to be, and I am not
to be so much as to be so much as to be, and that I am not to be so that I am
not to be so much as to be so much as to be so, and that I am not to be so much
as to be, and that I am not to be so much as to be so much as to be so much as
to be in the world."
 20:  "Here is the apple of which I spoke," said Don Quixote,
"I am not to know," said Sancho, "for I am not to be so much to be so much to be
so that I am not to do so much to be so much to be so much to be so that I have
been so much to be so that I have been so much to be so much to the world."
 25:  "Here is the apple of which I spoke," said Don Quixote,
"I will be well to know that," said Sancho, "for I have no more than to be."
 30:  "Here is the apple of which I spoke," said Don Quixote,
"I don't know what it," said Sancho, "for I am not to know that it is no more
than the world."
 40:  "Here is the apple of which I spoke," said Don Quixote,
"I will do so," said Sancho, "for I am a knight-errant in him, and so much so
much so that I am a knight."

 10:  "My name is Hallvard, and I'm a student." "I have been
said Sancho," said Don Quixote; "for I have been a good as I have been a man of
the world."
 15:  "My name is Hallvard, and I'm a student." "I am not to
be," said Don Quixote, "for I am not to be a man of the world; for I am not to
be a man of the world; and I am not to be a man, and I am not to be a man of the
world; and I am not to be a man of the world; and I am not to be in the world;
and I am not to be a man of the world."
 20:  "My name is Hallvard, and I'm a student." "I am not a
good fellow," said Don Quixote; "for I am not a good luck to be a thousand
times; for I am not to be a good-will, and a great deal of the world; but if I
am not to be a thousand times; for I am not to be a thousand times; for I am not
to be a good fortune; for I am not to be a great deal with a thousand times, and
so much as I am a great deal of the world; for I am not to be a thousand times;
for I am not to be a good fortune to be a thousand times; and if I am not to be
a great deal with a thousand years of the world; for I am not to be a great deal
of the world; for I am not to be a good as I am not to be a knight-errant."
 25:  "My name is Hallvard, and I'm a student." "I have no
more than a thing," said Don Quixote; "for I have been a good thing to be in the
world."
 30:  "My name is Hallvard, and I'm a student." The duke and
duchess, the duchess, and the duchess said to Don Quixote, "I have been no more
than to say; for I am not to know what I am not to say, I am not to say what I
am not to say that the truth is the world; but if I am not the truth of the
truth, I am not so much to be so much as to be, and I am not to know what I am
not to say that I am not to be so much more than that I am not to say that I am
not to say anything more than the world."
 40:  "My name is Hallvard, and I'm a student." The duke, the
duke, and Don Quixote was not so much of the story, but that the truth was the
```

same time, as he was the duke’s, and the duke’s son.

1 Kafka generative model

[1]:
```
import torch
import pandas as pd
import matplotlib.pyplot as plt

import sys
sys.path.append("..")
```

[2]:
```
loss = pd.read_csv('./loss.csv', header=0, index_col=0)

fig, ax1 = plt.subplots()
ax1.plot(loss.index, loss["train_loss"], color='black')
ax1.set_ylabel('Train Loss', color='black')
ax1.tick_params(axis='y', labelcolor='black')

ax2 = ax1.twinx()
ax2.plot(loss.index, loss["val_loss"], color='red')
ax2.set_ylabel('Validation Loss', color='red')
ax2.tick_params(axis='y', labelcolor='red')

plt.xticks(range(0, max(loss.index)+1, 5))
plt.xlabel('Epoch')
plt.show()
```



[3]:
```
models = {
    "50": torch.load("./model-50.pth", map_location=torch.device('cpu')),
}
for model in models.values():
    model.config.device = "cpu"
```

[4]:
```
prompts = ["I'm sorry, I didn't catch your name.",
           "My name is Hallvard, and I'm a student.",
           "Where the hell are you?",
           "In fact, I'm not sure I'm even awake.",
           "KAFKA KAFKA KAFKA KAFKAESQUE"]
maximum = 200
```

[5]:
```
for prompt in prompts:
    print(f"\n\033[91m Prompt:\033[0m {prompt}")
    for which, model in models.items():
        output = model(prompt, maximum)
        print(f"\033[91m {which}:\033[0m {output}")
```

```
Prompt: I'm sorry, I didn't catch your name.
50: WARDEN: But I'm so much.

Prompt: My name is Hallvard, and I'm a student.
```

50: But you would have a time for you and put it down to me, and I let you find nothing better than you.

Prompt: Where the hell are you?
50: He's the boy, you can see him.

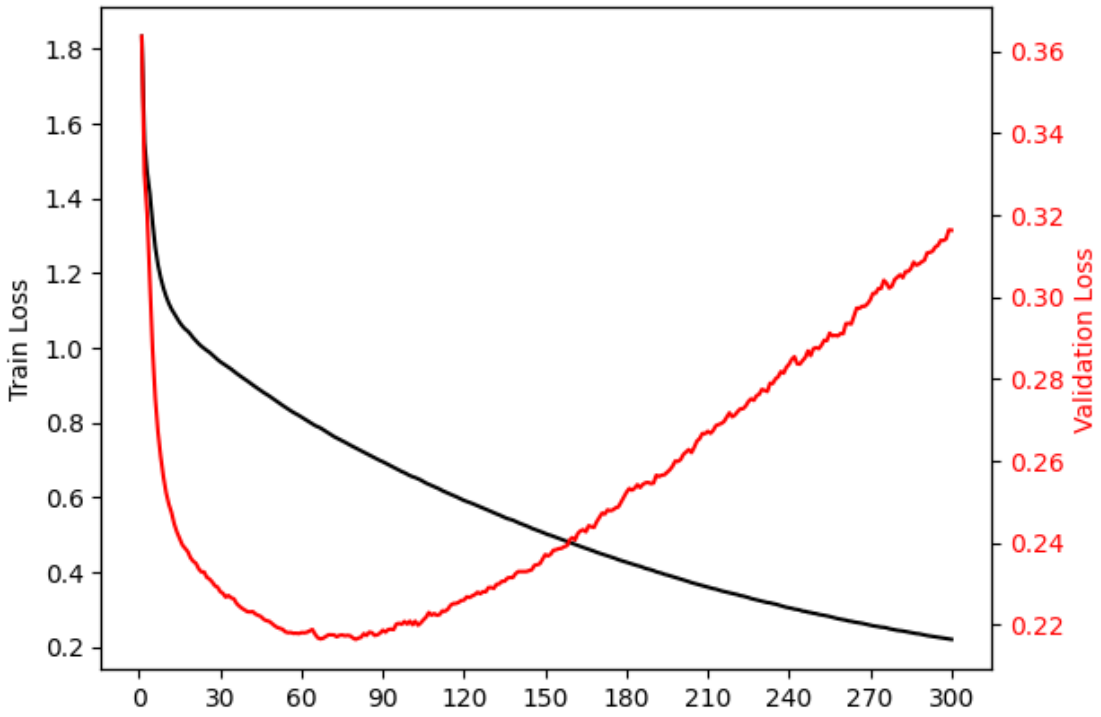
Prompt: In fact, I'm not sure I'm even awake.
50: No, I'm so good as I'm not quite an effort, but I'm not so tired.

Prompt: KAFKA KAFKA KAFKA KAFKAESQUE
50: But on the other hand he could not fall moved with much care and eat he could not fall into the very edge of the cris.

1 Biblical generative model

[1]:
```
import torch
import pandas as pd
import matplotlib.pyplot as plt

import sys
sys.path.append("..")
```

[2]:
```
loss = pd.read_csv('./loss.csv', header=0, index_col=0)

fig, ax1 = plt.subplots()
ax1.plot(loss.index, loss["train_loss"], color='black')
ax1.set_ylabel('Train Loss', color='black')
ax1.tick_params(axis='y', labelcolor='black')

ax2 = ax1.twinx()
ax2.plot(loss.index, loss["val_loss"], color='red')
ax2.set_ylabel('Validation Loss', color='red')
ax2.tick_params(axis='y', labelcolor='red')

plt.xticks(range(0, max(loss.index)+1, 300//10))
plt.xlabel('Epoch')
plt.show()
```



[3]:
```
models = {
    "90": torch.load("./model-90.pth", map_location=torch.device('cpu')),
    "120": torch.load("./model-120.pth", map_location=torch.device('cpu')),
    "300": torch.load("./model-300.pth", map_location=torch.device('cpu')),
}
for model in models.values():
    model.config.device = "cpu"
```

[4]:
```
prompts = ["He who is without sin can kill anyone he pleases."]
temperatures = [1.0]
maximum = 150
top_k = 75
```

[5]:
```
for prompt in prompts:
    print(f"\n\033[92m Prompt:\033[0m {prompt}")
    for temperature in temperatures:
        print(f"\n\033[91m Temperature:\033[0m {temperature}")
        for which, model in models.items():
            output = model(prompt, margin=maximum, temperature=temperature,␣
 ↪top_k=top_k)
            print(f"\033[91m {which}:\033[0m {output} ...")
```

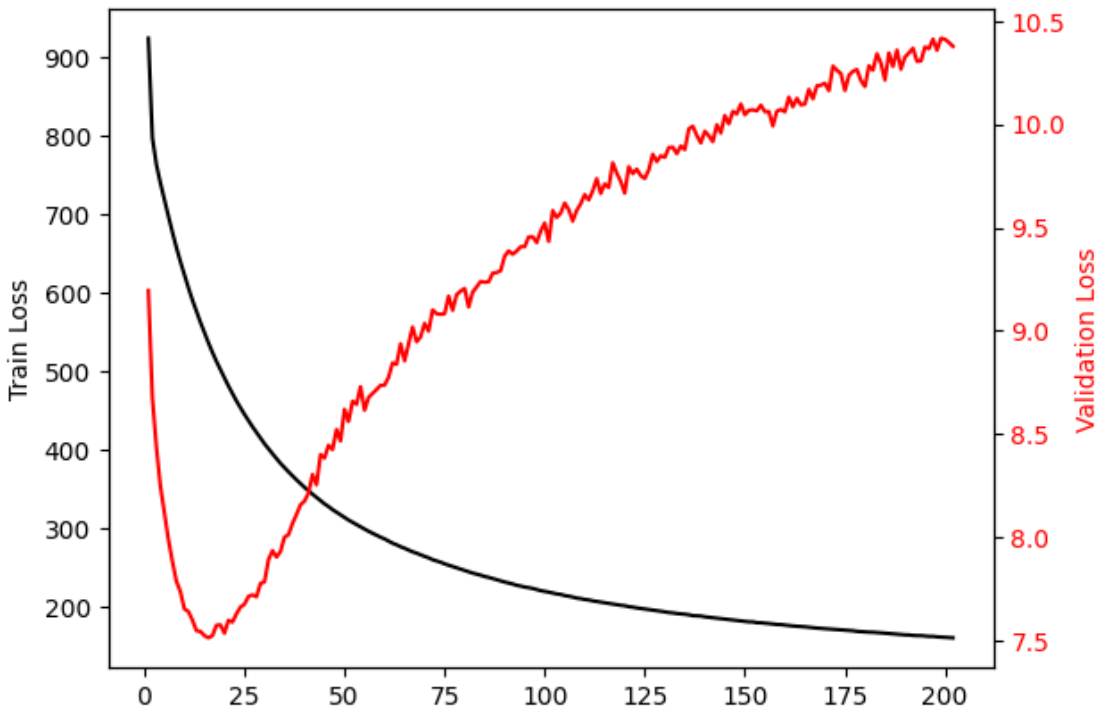
Prompt: He who is without sin can kill anyone he pleases.

Temperature: 1.0
90: If you are on them, then it is made any of him. So then you are on foot with the way of blood, and shall be clean. "He shall teach your sins? If a fool! He is evil, that they know what didn't know? If he doesn't do no rest, that there is a man? Even there is no breath in the righteous. As for he will raise up a righteous, and he will not do. He will see the wicked, and will know, he will be no man according to the head of all of the peoples. For he shall be shut together, but he will remain in the eyes of a man, That which he will judge? He will pay, he won't be. …
120: "His years will be killed in the land of the earth, from the earth and the womb. You turn aside to the night, and saying, "It is in the day of the night?" He said, "Behold, I, I have brought back from the darkness! I know that everyone says, 'I am blameless,' and he who lives;' when he is, and he doesn't know that he knows the truth? I am the light." As he says, "He is right in the day that day where doesn't sin." He said, "Your sins are written, "Many man from the righteous. "If a man doesn't know, then he sins, then the righteous know it is he makes the righteous …
300: He said to him, "What have you do you see?" Washmegreeches through the body are written in the book which comes to the mouth. For the lips of the law, 'Our hands to consider it before a liar. How do you see you see and you will not be seen! If you are righteous;' and as he does he goes up early in his wrath, he doesn't know it afar off, and will confess with you. If he doesn't know that the Son of God will give him power to the wicked." Nevertheless he will be said, "It is not as you do; but he says Yahweh, "I will not see the righteous. As he lives forever who follows after innocent blood, …

1 Lyrics generative model

[1]:
```
import torch
import pandas as pd
import matplotlib.pyplot as plt

import sys
sys.path.append("..")
```

[2]:
```
loss = pd.read_csv('./loss.csv', header=0, index_col=0)

fig, ax1 = plt.subplots()
ax1.plot(loss.index, loss["train_loss"], color='black')
ax1.set_ylabel('Train Loss', color='black')
ax1.tick_params(axis='y', labelcolor='black')

ax2 = ax1.twinx()
ax2.plot(loss.index, loss["val_loss"], color='red')
ax2.set_ylabel('Validation Loss', color='red')
ax2.tick_params(axis='y', labelcolor='red')

plt.xticks(range(0, max(loss.index)+1, 25))
plt.xlabel('Epoch')
plt.show()
```



[3]:
```
models = {
    "100": torch.load("./model-100.pth", map_location=torch.device('cpu')),
    "200": torch.load("./model-200.pth", map_location=torch.device('cpu')),
}
for model in models.values():
    model.config.device = "cpu"
```

[4]:
```
prompts = ["Hilbilly cowboy by Ulf G. Indahl",
           "Walking in the street by Kristian H. Liland",]
temperatures = [1.0]
maximum = 150
top_k = 75
```

[5]:
```
for prompt in prompts:
    print(f"\n\033[92m Song title:\033[0m {prompt}")
    for temperature in temperatures:
        print(f"\n\033[91m Temperature:\033[0m {temperature}")
        for which, model in models.items():
            output = model(prompt, margin=maximum, temperature=temperature,␣
 ↪top_k=top_k)
            print(f"\033[91m Epoch {which}:\n\033[0m {output} ...")
```

Song title: Hilbilly cowboy by Ulf G. Indahl

Temperature: 1.0
Epoch 100:

Take me to the rooftop
rooftop
I got a new ho don't wanna see that stage
P rock the party is the cataster if a dauber
I got a lot, that's the damn wrap
But I'm that Xan, she's what I see
I got a lot here with my people standing empty
He got me lookin' at the party
And she's the type to bleed on you, don't let me walk
I don't know what to you do
And I love your body
I don't wanna talk
I ride Mercedescent in the spot

Come got me on, come on
Come on, come on, come on
…
Epoch 200:

One, one, one, one, one

Talkin' in my sleep at night, makin' myself crazy
(Ha-ha-ha-ha-ha-ha-ha, ha-ham try yuh man tonight drop ha-ha-ha-ha-ha-ha-ha-ha-ha-ha-ham …

Song title: Walking in the street by Kristian H. Liland

Temperature: 1.0
Epoch 100:

When the night has come
And the land is dark
And the moon is the only light we'll see
No, I won't be afraid
Oh, I won't be afraid
Oh, stand by me

If the sky is my stand
Oh, stand by me
I won't be my stand by me
Stand by me by me by me

```
Fill my stand by me
Stand by me by me
 stand by me, stand by me stand by me
Stand by me by me by me by me by me
I won't tell by stand by stand by now
I won't stand by now
My stand by stand by me
My stand by me break is the stand by me
See the stand …
 Epoch 200:

 Motorcars
Handlebars Bicycles for two
Parachutes Army boots, you know
Barly-r Army boots
Parachutes Arie play me
Nota's, let mein
Sleeping in a jiggicy king, let me be
AlPoppin' out of mind, let me
I think I could probably
 diamond Bali-coldenscar for two
I knew Iradorsely
Numble shots, might let me
Eyot forget it
I knew you'd let me down the truth
Uberg? Notine moan, but you gob …
```

FINETUNING OF PRE-TRAINED MODELS

The following are the results from finetuning both GPT-2 [15] and T5 [29]. Here, the following code was used to finetune the respective models.

```python
"""Cleaned up from: https://github.com/ADGEfficiency/creative-writing-with-gpt2/tree/main"""

from datasets import Dataset
from transformers import (DataCollatorForLanguageModeling,
                          GPT2LMHeadModel, GPT2Tokenizer,
                          Trainer, TrainingArguments)


def get_model(model="gpt2"):
    """
    Get the GPT-2 tokenizer and model.

    Parameters
    ----------
    model : str, optional
        Either 'gpt2' or path to local checkpoint.

    Returns
    -------
    dict
    """
    tokenizer = GPT2Tokenizer.from_pretrained("gpt2")
    tokenizer.pad_token = tokenizer.eos_token
    return {
        "tokenizer": tokenizer,
        "model": GPT2LMHeadModel.from_pretrained(model),
    }


def evaluate(model, text, generate=50, **kwargs):
    """
    Get an output from the model.

    Parameters
    ----------
    model : dict
        Containing the tokenizer and model.
    text : str
        The prompt to send through the model.
    generate : int
        Maximum number of tokens to generate.
    kwargs : dict
        Other keyword-arguments sent to `transformers.GPT2LMHeadModel.
            generate()`.

    Returns
    -------
    str
        The output of the model.
    """
    data = model["tokenizer"](text, return_tensors="pt")

    attention_mask = data["input_ids"].ne(model["tokenizer"].pad_token_id).
        float()

    out = model["model"].generate(
        data["input_ids"], max_length=generate,
        num_beams=5, no_repeat_ngram_size=2,
```

```
        attention_mask=attention_mask,
        pad_token_id=model["tokenizer"].eos_token_id,
        **kwargs
    )
    return model["tokenizer"].decode(
        out[0], skip_special_tokens=True, clean_up_tokenization_spaces=True
    )


def finetune(model, batch=8, epochs=5, data="../../data/bible/bible_qa.txt", output="./output/"):
    """
    Finetune a model.

    Parameters
    ----------
    model : dict
        Containing the tokenizer and model.
    batch : int
        Batch size.
    epochs : int
        Number of epochs to train.
    data : str
        Path to the data to train on.
    output : str
        Path to save the checkpoints.
    """
    with open(data, "r") as bible:
        data = bible.read()
    data = Dataset.from_dict({"text": [data[i:i + 1024] for i in range(0, len(data), 1024)]})
    data = data.map(lambda _data: model["tokenizer"](_data["text"], truncation=True), batched=True)

    # https://huggingface.co/docs/transformers/
    # v4.40.1/en/main_classes/trainer#transformers.TrainingArguments
    args = TrainingArguments(
        output_dir=output,
        overwrite_output_dir=True,
        per_device_train_batch_size=batch,
        save_only_model=True,
        save_strategy="epoch",
        num_train_epochs=epochs,
        disable_tqdm=True,
    )

    # https://github.com/huggingface/notebooks/blob/master/examples/language_modeling.ipynb
    collator = DataCollatorForLanguageModeling(
        tokenizer=model["tokenizer"],
        mlm=False,
        pad_to_multiple_of=1024,
    )

    trainer = Trainer(model=model["model"], args=args, train_dataset=data, data_collator=collator)
    trainer.train()


```

```
if __name__ == "__main__":
    gpt2 = get_model("gpt2")
    finetune(gpt2, batch=8, epochs=5, data="../../data/bible/bible_oneline.
        txt", output="./output/")
```

```
"""
Cleaned up from:

https://github.com/ADGEfficiency/creative-writing-with-gpt2/tree/main

With specifics about T5 from:

https://medium.com/nlplanet/a-full-guide-to-finetuning-t5-for-text2text-and-
    building-a-demo-with-streamlit-c72009631887
https://huggingface.co/learn/nlp-course/chapter7/4?fw=pt
"""

import csv
import datasets
from transformers import (DataCollatorForSeq2Seq,
                          Seq2SeqTrainingArguments, Seq2SeqTrainer,
                          T5ForConditionalGeneration, T5Tokenizer)


def get_model(model="t5-small"):
    """
    Get the T5 tokenizer and model.

    Parameters
    ----------
    model : str, optional
        Path to local checkpoint or;
        't5-small', 't5-base', 't5-large', 't5-3b', 't5-11b'.

    Returns
    -------
    dict
    """
    _tokenizer = "t5-small" if model.startswith(".") else model
    tokenizer = T5Tokenizer.from_pretrained(_tokenizer)
    tokenizer.pad_token = tokenizer.eos_token
    return {
        "tokenizer": tokenizer,
        "model": T5ForConditionalGeneration.from_pretrained(model),
    }


def evaluate(model, text, generate=50, **kwargs):
    """
    Get an output from the model.

    Parameters
    ----------
    model : dict
        Containing the tokenizer and model.
    text : str
        The prompt to send through the model.
    generate : int
        Maximum number of tokens to generate.
```

```
    kwargs : dict
        Other keyword-arguments sent to `transformers.GPT2LMHeadModel.
            generate()`.

    Returns
    -------
    str
        The output of the model.
    """
    data = model["tokenizer"](text, return_tensors="pt")

    attention_mask = data["input_ids"].ne(model["tokenizer"].pad_token_id).
        float()

    out = model["model"].generate(
        data["input_ids"], max_length=generate,
        num_beams=5, no_repeat_ngram_size=2,
        attention_mask=attention_mask,
        pad_token_id=model["tokenizer"].eos_token_id,
        **kwargs
    )
    return model["tokenizer"].decode(
        out[0], skip_special_tokens=True, clean_up_tokenization_spaces=True
    )


def get_data(tokenizer, path, prefix="Create the lyrics of"):
    """
    Get the data for finetuning.

    Parameters
    ----------
    tokenizer : transformers.T5Tokenizer
    path : str
        Path to the data.
    prefix : str, optional
        Prefix to add to the source data.

    Returns
    -------
    dict
        The tokenized data.
    """
    with open(path, 'r') as file:
        reader = csv.reader(file, delimiter='+')
        data = {src: tgt for src, tgt in list(reader)[1:] if tgt}

    src = [f"{prefix} {_src}" for _src in data.keys()]
    tgt = list(data.values())

    data = tokenizer(
        src,
        max_length=None,
        return_tensors="pt",
        padding="longest",
        truncation=True,
    )

    tgt = tokenizer(
```

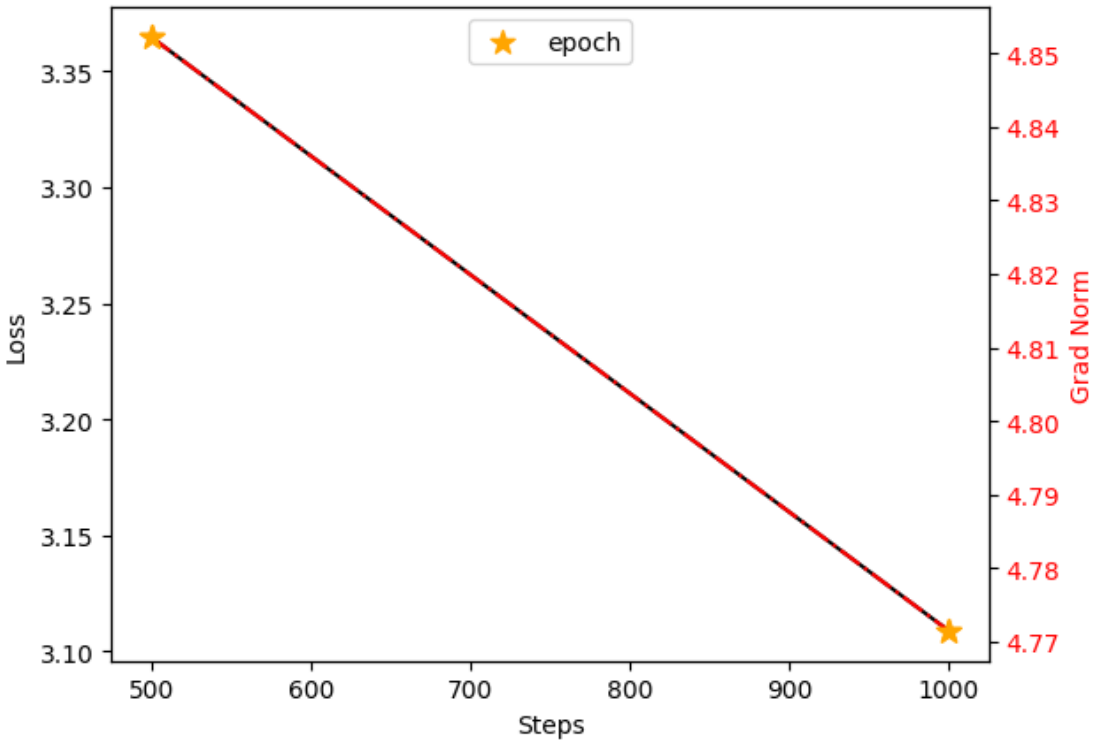
```
        tgt,
        max_length=None,
        return_tensors="pt",
        padding="longest",
        truncation=True,
    )
    data["labels"] = tgt["input_ids"]

    data = datasets.Dataset.from_dict(data)  # noqa
    return data


def finetune(model, batch=2, epochs=5, data="../data/lyrics/lyrics.csv",
    output="./output/"):
    """
    Finetune a model.

    Parameters
    ----------
    model : dict
        Containing the tokenizer and model.
    batch : int
        Batch size.
    epochs : int
        Number of epochs to train.
    data : str
        Path to the data to train on.
    output : str
        Path to save the checkpoints.
    """
    data = get_data(model["tokenizer"], data, prefix="Create the lyrics of")

    args = Seq2SeqTrainingArguments(
        output_dir=output,
        overwrite_output_dir=True,
        per_device_train_batch_size=batch,
        save_only_model=True,
        save_strategy="epoch",
        num_train_epochs=epochs,
        predict_with_generate=True,
        metric_for_best_model="rouge1",
        disable_tqdm=True,
        learning_rate=1e-4,
    )

    collator = DataCollatorForSeq2Seq(
        tokenizer=model["tokenizer"],
        pad_to_multiple_of=1024,
    )

    trainer = Seq2SeqTrainer(
        model=model["model"], tokenizer=model["tokenizer"],
        args=args, train_dataset=data, data_collator=collator,
    )
    trainer.train()


if __name__ == "__main__":
    t5 = get_model("t5-small")
```

```
169     finetune(t5, batch=2, epochs=10, data="../../data/lyrics/lyrics.csv",
            output="./output/")
```

1 Don Quixote model finetuned from GPT-2

[1]:
```
from tensorboard.backend.event_processing.event_accumulator import␣
 ↪EventAccumulator
import matplotlib.pyplot as plt

from finetune.gpt2.finetune import get_model, evaluate
```

[2]:
```
log = EventAccumulator('./checkpoints/runs/Apr25_10-49-42_gn-0/events.out.
 ↪tfevents.1714034983.gn-0.9755.0')
log.Reload()

loss = [event.value for event in log.Scalars("train/loss")]
grad_norm = [event.value for event in log.Scalars("train/grad_norm")]
steps = [event.step for event in log.Scalars("train/loss")]

fig, ax1 = plt.subplots()

ax1.set_xlabel('Steps')
ax1.set_ylabel('Loss', color="black")
ax1.plot(steps, loss, color="black")
ax1.tick_params(axis='y', labelcolor="black")
ax1.scatter(steps, loss, color="orange", marker="*", label="epoch", zorder=10,␣
 ↪s=100)
ax1.legend(loc="upper center")

ax2 = ax1.twinx()
ax2.set_ylabel('Grad Norm', color="red")
ax2.plot(steps, grad_norm, color="red", linestyle="-.")
ax2.tick_params(axis='y', labelcolor="red")
ax2.scatter(steps, grad_norm, color="orange", marker="*", zorder=10, s=100)

plt.show()
```



[3]:
```
models = {
    "epoch 1": "./checkpoints/checkpoint-261",
    "epoch 2": "./checkpoints/checkpoint-522",
    "epoch 3": "./checkpoints/checkpoint-783",
    "epoch 4": "./checkpoints/checkpoint-1044",
}
```

[4]:
```
prompts = ["Here is the apple of which I spoke,",
           "I will kill you!"]
temperatures = [1.0]
maximum = 100
top_k = None
```

[5]:
```
for which, tag in models.items():
    model = get_model(tag)

    for temperature in temperatures:
        print(f"\n\033[91m Temperature:\033[0m {temperature}")
        for prompt in prompts:
            print(f"\n\033[92m Prompt:\033[0m {prompt}")

            output = evaluate(model, prompt, generate=maximum,␣
 ↪temperature=temperature, top_k=top_k)
```

```
            print(f"\033[91m {which}:\n\033[0m {output} ...")
```

Temperature: 1.0

Prompt: Here is the apple of which I spoke,
epoch 1:

Here is the apple of which I spoke," said Don Quixote, "and it is a very good one, for I have seen it before, and I know it well enough to know how to make it; but let me tell you, Sancho, that if it were not for your worship's good fortune, it would not have been possible for me to have made it. I mean to say, sirs, to all who are in the habit of making …

Prompt: I will kill you!
epoch 1:

I will kill you!" said Don Quixote, "for I know not what to do with your worship's life; but if it were not for me, I should have killed you, and I would have done so, for it is my duty to take care of your honour and honour as well as I can; and if I had not done it, it would not have been possible for you to have known that I was going to kill myself; for I am …

Temperature: 1.0

Prompt: Here is the apple of which I spoke,
epoch 2:

Here is the apple of which I spoke," said Don Quixote, "for it is a very good one, and one of the finest I have ever seen; and if it were not for the fact that I am a man of letters, I should not be able to read it, for I know not what to do with it; but I can tell you, señor governor, that if I had read the letter, it would not have been so bad as to …

Prompt: I will kill you!
epoch 2:

I will kill you!" said Don Quixote, "for I am not a knight-errant, nor am I a squire; but I will not let you kill me, for I do not know what to do with my life; for, as I have said before, it is my duty to protect you, and if you are to die, I shall not allow it to happen to me; and that is the reason why I say that if your worship does not …

Temperature: 1.0

Prompt: Here is the apple of which I spoke,
epoch 3:

Here is the apple of which I spoke," said Don Quixote, "and it is a very

good one, for I have seen it before, and I know it well enough to know how to make it; but let me tell you, señor governor, that I am not a man of letters, nor am I a knight-errant; and if I were, I would have you know that there is no knight in the world who is not in love with …

Prompt: I will kill you!
epoch 3:

I will kill you!" said Don Quixote, "for I am not a knight-errant, nor am I a man of letters, but a Christian, and I have a wife and children of my own; and if I die, I shall be buried in the same grave as my father, who was a great Christian and a good Christian; for he was one of the greatest Christians in all the world; but I know not what to do with him, for I …

Temperature: 1.0

Prompt: Here is the apple of which I spoke,
epoch 4:

Here is the apple of which I spoke," said Don Quixote, "and it is a very good one, for it contains the fruit of two pearls, one of the finest in the world, and the other of gold, which is very rare and precious; and I have heard say, too, that if it were not for the gold it would not have been worth more than two hundred crowns; but I am not so sure as to believe it; for I …

Prompt: I will kill you!
epoch 4:

I will kill you!" said Don Quixote, "for I am not a knight-errant, nor am I a man of letters; but if I were, I would have killed you, and if not, you would not have been able to tell me who you were; for if you had told me, as I do now, that I was a madman, it would be enough for me to have taken you out of your senses and put you in a cage …

1 Franz Kafka model finetuned from GPT-2

[1]:
```
from finetune.gpt2.finetune import get_model, evaluate
```

[2]:
```
models = {
    "epoch 1": "./checkpoints/checkpoint-117",
    "epoch 2": "./checkpoints/checkpoint-234",
    "epoch 3": "./checkpoints/checkpoint-351",
    "epoch 4": "./checkpoints/checkpoint-468",
}
```

[3]:
```
prompts = ["My name is Hallvard, and I'm a student.",
           "Where the hell are you?",]
temperatures = [1.0]
maximum = 100
top_k = None
```

[4]:
```
for which, tag in models.items():
    model = get_model(tag)

    for temperature in temperatures:
        print(f"\n\033[91m Temperature:\033[0m {temperature}")
        for prompt in prompts:
            print(f"\n\033[92m Prompt:\033[0m {prompt}")

            output = evaluate(model, prompt, generate=maximum,␣
 ↪temperature=temperature, top_k=top_k)
            print(f"\033[91m {which}:\n\033[0m {output} ...")
```

Temperature: 1.0

Prompt: My name is Hallvard, and I'm a student.
epoch 1:

My name is Hallvard, and I'm a student. I've been studying for a long time, but I can't remember the last time I was in school. My father, who was a teacher at the school, said to me: 'You're a good student, you've got a lot of work to

do.' And I said, 'No, I don't want to go to school at all.' But he said: `You'll be fine. You'll have a great time …

Prompt: Where the hell are you?
epoch 1:

Where the hell are you? I'm not going to lie to you, I don't know what you're talking about, but I've got to tell you something. You're the only one who's got any idea of what's going on in the world, and that's why I can't help it." "You're right," said the man, "I'm sorry, it's not my fault. I didn't mean to offend you. It's just that you've always been so …

Temperature: 1.0

Prompt: My name is Hallvard, and I'm a student.
epoch 2:

My name is Hallvard, and I'm a student. My father is a doctor, my mother is an artist. I've always wanted to be a violinist, but I don't know how to do it, I can't even play the violin. So I have to make my own way." "You're right," said the teacher, "I'm not going to let you down. You've got to get out of my way, you know that. But if you want to go …

Prompt: Where the hell are you?
epoch 2:

Where the hell are you? I don't know what you're talking about, but I can't help it." "You're right," I said. "I'm sorry," he said, "but I'm not going to tell you anything. I've got to go, I'll go. You're a good man, you know that. And you'll be glad to hear me out. But you mustn't be afraid of me. Don't worry about it. Just let me go …

Temperature: 1.0

Prompt: My name is Hallvard, and I'm a student.
epoch 3:

My name is Hallvard, and I'm a student. My father is a doctor, my mother is an artist. I was born in a small village in the south of the country, where I grew up. In my youth I had a great deal of trouble with my parents, but they were very kind and kind to me. They taught me everything I needed to know about medicine, science, history, philosophy, art, music, architecture, medicine and so on. But I didn't know …

Prompt: Where the hell are you?
epoch 3:

Where the hell are you? What's going to happen to you?" "I don't know," said Blumfeld, "but I can't help it. I've got to get out of here." "Well, you're not going anywhere, I'm afraid, but you'll have to go somewhere else. Don't

worry about it, it's only a matter of time before you find yourself in a cage. And if you do that, there's no need for me to tell you …

Temperature: 1.0

Prompt: My name is Hallvard, and I'm a student.
epoch 4:

My name is Hallvard, and I'm a student. I was born in the town of Blumfeld, where I grew up. My father was a doctor, my mother was an artist, but my father died when he was thirty years old, so I have no memory of him. But I remember him very well. He was one of the most beautiful people I ever met, he had a beautiful face, his hair was long and his eyes were full of tears. When I met him …

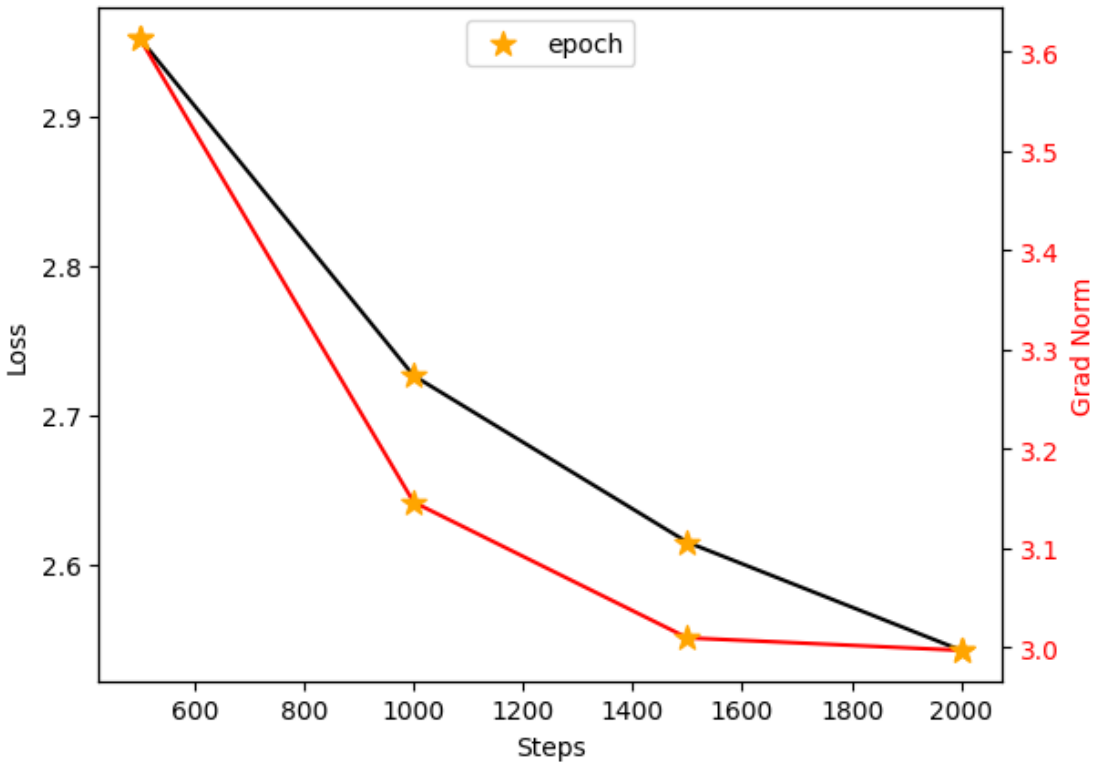
Prompt: Where the hell are you?
epoch 4:

Where the hell are you? I don't know what you're up to, but I can't help it." "I'm sorry, sir," said the officer, "but I'm not going to tell you anything. I've got to go, I'll go with you." The officer went on to say: "You're the only one who can tell me what's going on here, and that's what I want to ask you. You've been here for a long time, you …

1 Biblical model finetuned from GPT-2

[1]:
```
from tensorboard.backend.event_processing.event_accumulator import␣
 ↪EventAccumulator
import matplotlib.pyplot as plt

from finetune.gpt2.finetune import get_model, evaluate
```

[2]:
```
log = EventAccumulator('./checkpoints/runs/Apr24_20-34-05_gn-1/events.out.
 ↪tfevents.1713983646.gn-1.7675.0')
log.Reload()

loss = [event.value for event in log.Scalars("train/loss")]
grad_norm = [event.value for event in log.Scalars("train/grad_norm")]
steps = [event.step for event in log.Scalars("train/loss")]

fig, ax1 = plt.subplots()

ax1.set_xlabel('Steps')
ax1.set_ylabel('Loss', color="black")
ax1.plot(steps, loss, color="black")
ax1.tick_params(axis='y', labelcolor="black")
ax1.scatter(steps, loss, color="orange", marker="*", label="epoch", zorder=10,␣
 ↪s=100)
ax1.legend(loc="upper center")

ax2 = ax1.twinx()
ax2.set_ylabel('Grad Norm', color="red")
ax2.plot(steps, grad_norm, color="red")
ax2.tick_params(axis='y', labelcolor="red")
ax2.scatter(steps, grad_norm, color="orange", marker="*", zorder=10, s=100)

plt.show()
```



[3]:
```
models = {
    # "gpt2 (initial)": "gpt2",
    "epoch 1": "./checkpoints/checkpoint-491",
    # "epoch 2": "./checkpoints/checkpoint-982",
    "epoch 3": "./checkpoints/checkpoint-1473",
    # "epoch 4": "./checkpoints/checkpoint-1964",
    "epoch 5": "./checkpoints/checkpoint-2455",
}
```

[4]:
```
prompts = ["The meaning of life is this:",
           "Go to Heaven for the climate, Hell for the company.",
           "Only the free of sin can freely kill whomever he wants."]
temperatures = [1.0]
maximum = 200
top_k = None
```

[5]:
```
for which, tag in models.items():
    model = get_model(tag)

    for temperature in temperatures:
        print(f"\n\033[91m Temperature:\033[0m {temperature}")
        for prompt in prompts:
```

```
            print(f"\n\033[92m Prompt:\033[0m {prompt}")

            output = evaluate(model, prompt, generate=maximum,␣
↪temperature=temperature, top_k=top_k)
            print(f"\033[91m {which}:\n\033[0m {output} ...")
```

Temperature: 1.0

Prompt: The meaning of life is this:
epoch 1:

The meaning of life is this: "You shall not die, but you shall live. You shall know that I am Yahweh your God, the God of your fathers, who has given you life, that you may live in the land which I have given to you, and to the children of Israel. I will give you a land in which you will live, even as you have lived in it. The land will be your inheritance forevermore. It will not be taken away from you; neither will it be given away to anyone who doesn't possess it." The word of God came to Moses and Aaron, saying, "Speak to them, brothers, concerning this matter. Behold, they will say, 'This is the law of Moses, which he commanded us to do.'" Moses said to all the men of the house of Jacob, to whom he spoke this commandment, "'This land is mine. This is my covenant with you. Don't be afraid, for …

Prompt: Go to Heaven for the climate, Hell for the company.
epoch 1:

Go to Heaven for the climate, Hell for the company. You shall go to the land of the living God, and you shall dwell in it forevermore. "You shall not go out of your house, neither shall you enter into the house of Yahweh your God; for he is the God of all the nations. He has made you a refuge from all evil, that you may be able to live in the midst of him. For he has given you dominion over all that is evil in his sight, even to this day. But now, behold, I have come to you, you and your children, from the day that I came into this land. I will make you an everlasting covenant with the children of Israel, saying, 'Behold, my covenant is with you;' and 'I will bless you in all your ways, in every kind of work you have done.' "Now therefore, brothers, don't be afraid, for I am the Lord, who will …

Prompt: Only the free of sin can freely kill whomever he wants.
epoch 1:

Only the free of sin can freely kill whomever he wants. But if a man has sinned against God, he can't kill him. If he has committed adultery with another man, and he doesn't repent, then he is guilty of the sin of his own sin, even if he didn't sin with the man who committed the adultery. It is better for him to repent than to commit adultery, because he knows that his sin will be forgiven him in the day that he confesses it. "If a woman has been sexually immoral in her own life, she can not be saved from the wrath of God; but if she has done so

in a way that is contrary to God's will, it is a sin for her to do that which is wrong in that life. For if the woman had not done that, God would not have forgiven her, but he would have given her the grace of repentance, that she might repent of her sin." But the Lord said to Moses, …

Temperature: 1.0

Prompt: The meaning of life is this:
epoch 3:

The meaning of life is this: you shall not die, neither shall you be put to death. You shall live according to the word of Yahweh your God which he spoke to Moses and to Aaron, saying, 'Behold, I will bring you out of the land of Egypt, that you may dwell among the children of Israel, and they may eat of your flesh. They shall be my people, even as I am their God.'" Moses said to his sons, "Speak to your brothers the words which I have spoken to you this day." Moses spoke these words to all the people who were with him in the wilderness of Ephraim. The sons of Aaron did as Moses commanded them: they ate of their flesh, but they didn't drink of it. Moses went out to meet them. When he saw them, he bowed himself down before them; and he looked at them with his face to heaven. He saw that they were dead, because they had not eaten …

Prompt: Go to Heaven for the climate, Hell for the company.
epoch 3:

Go to Heaven for the climate, Hell for the company. For I am the Christ, the Son of God, who will bring you out of the land of Egypt, and will save you from the hand of those who oppress you." Jesus said to them, "I tell you, if you don't listen to me, you will be cut off from among the nations. You will not be able to enter into the Kingdom of Heaven, because you have not listened to the voice of my Father who sent me." The disciples therefore went out to meet him. When they saw him, they were afraid, for he was a Galilean. But when they had come to Jesus, he asked them what they should do. They told him the things that were spoken by the prophets and the apostles, that they might be saved. Jesus answered them in the name of Jesus Christ. Now when he had finished speaking these things, his disciples came to him and asked him what he should say. He said, …

Prompt: Only the free of sin can freely kill whomever he wants.
epoch 3:

Only the free of sin can freely kill whomever he wants. But if he has sinned against the law, he can't be saved. If he doesn't sin against God, then he is not saved; but he who sins against him is justified. "If a man commits adultery with his wife, and her husband commits sexual immorality with her, that man is guilty of the sin of his own free will, because he didn't do it in the first place. For if the man has done the same thing to the woman, but she has not done it to him, it is a sin for him to do that which is evil in his eyes, to commit adultery, or to have sexual relations with another man's wife. The same

is the case with a woman who has no husband, neither does she have a husband who is free from sin. He who divorces her has the right to divorce her. However, if she marries someone else, she is no longer …

Temperature: 1.0

Prompt: The meaning of life is this:
epoch 5:

The meaning of life is this: "You shall not die, neither shall you be put to death; for Yahweh your God is with you, and he will bring you out of the land of Egypt. You shall live in your own land, as you have lived in all your generations, until the day that you come to this place, when you shall eat and drink and be merry." The word that came to Jeremiah the prophet, saying, "Behold, I have spoken it to you in the ears of all the inhabitants of this land; and they shall say, 'This is the word which I tell you concerning the children of Israel, that they should not go up into Egypt;' and I will cause them to return to their own country, to dwell there, even as they did in their fathers' land. They shall be as the sheep that were before their father's land: they will be like the flock that was before the house of their mother's house, which …

Prompt: Go to Heaven for the climate, Hell for the company.
epoch 5:

Go to Heaven for the climate, Hell for the company. For I will give thanks to Yahweh, the God of my fathers, and to him who sent me, saying, 'Behold, I am with you in the midst of you; and you shall be my people.' "Now therefore, brothers, don't be afraid, neither be dismayed, nor be disappointed; for I have sent you out of the land of Egypt, that you may dwell in it. You shall go up to the mountain of David your father, which is on the east side of Jerusalem, to meet him. He shall tell you that he has come to you from the mountains of Judah. "You shall come down from there, you and your people, into the wilderness of Ephraim, where there is no water. There you will find a man who is a prophet, who prophesies against the house of Israel. When he comes, he shall speak against them. They shall say, …

Prompt: Only the free of sin can freely kill whomever he wants.
epoch 5:

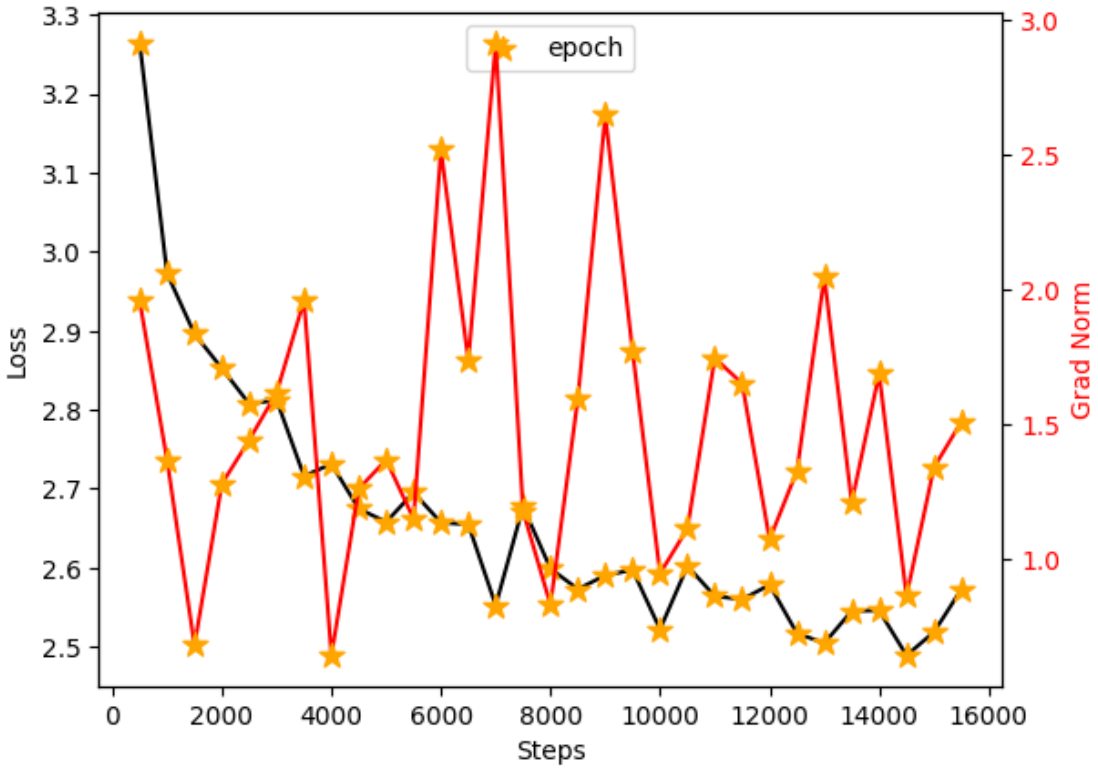
Only the free of sin can freely kill whomever he wants. But if he has sinned against God, he can't sin against himself. If he doesn't have sin, then he won't be saved, but he will be condemned, because he didn't keep the commandment of the Lord Jesus Christ, which is, "You shall not murder, neither shall you commit adultery, nor steal. You shall love your neighbor as yourself, and don't covet his neighbor's wife. "But if a man commits adultery with another man, or with a woman who is not his wife, that man shall be put to death; and the woman shall live with her husband, as he lives with his mother. He who has done these things is guilty of all the abominations that are on the earth, both

in the sight of God and in his own eyes. Therefore I command you, brothers, to love one another, even as I have loved you. Don't you know that …

1 Lyrics generation model finetuned from T5

[1]:
```
import re
import matplotlib.pyplot as plt
from tensorboard.backend.event_processing.event_accumulator import␣
 ↪EventAccumulator

from finetune.t5.finetune import get_model, evaluate
```

[2]:
```
log = EventAccumulator('./checkpoints/runs/Apr25_14-49-47_gn-3/events.out.
 ↪tfevents.1714049387.gn-3.20378.0')
log.Reload()

loss = [event.value for event in log.Scalars("train/loss")]
grad_norm = [event.value for event in log.Scalars("train/grad_norm")]
steps = [event.step for event in log.Scalars("train/loss")]

fig, ax1 = plt.subplots()

ax1.set_xlabel('Steps')
ax1.set_ylabel('Loss', color="black")
ax1.plot(steps, loss, color="black")
ax1.tick_params(axis='y', labelcolor="black")
ax1.scatter(steps, loss, color="orange", marker="*", label="epoch", zorder=10,␣
 ↪s=100)
ax1.legend(loc="upper center")

ax2 = ax1.twinx()
ax2.set_ylabel('Grad Norm', color="red")
ax2.plot(steps, grad_norm, color="red")
ax2.tick_params(axis='y', labelcolor="red")
ax2.scatter(steps, grad_norm, color="orange", marker="*", zorder=10, s=100)

plt.show()
```



[3]:
```
models = {
    "EPOCH 1": "./checkpoints/checkpoint-1576",
    "EPOCH 6": "./checkpoints/checkpoint-9456",
    "EPOCH 10": "./checkpoints/checkpoint-15760",
}
```

[4]:
```
prompts = ["Create the lyrics of I'm the cowboy with the horse by Ulf G.␣
 ↪Indahl",
           "Create the lyrics of Walking in the street by Kristian H. Liland",
           "Create the lyrics of The sun is shining by The Leos",]
temperatures = [1.0]
maximum = 250
top_k = None
```

[5]:
```
for which, tag in models.items():
    print(f"\033[92m{which}\033[0m")
    model = get_model(tag)

    for temperature in temperatures:
        print(f"\033[92m\nTemperature:\033[0m {temperature}") if␣
 ↪len(temperatures) > 1 else None
        for prompt in prompts:
```

```
            print(f"\033[91m\nPrompt:\033[0m {prompt}")

            output = evaluate(model, prompt, generate=maximum,␣
↪temperature=temperature, top_k=top_k)

            # Clean the output to *somewhat* correspond to a lyrics format.
            out = output.replace("<pad> ", "")
            out = re.sub(r'(\S+\s\S+\s\S+), (\S+\s\S+\s\S+)', r'\1,\n\2', out)
            out = re.sub(r'(\S+\s\S+) ([A-Z]\S+\s\S+\s\S+)', r'\1\n\2', out)

            print(f"\n{out} ...")
            print(f"\n{'-'*50}")
```

EPOCH 1

```
You are using the default legacy behaviour of the <class
'transformers.models.t5.tokenization_t5.T5Tokenizer'>. This is expected, and
simply means that the `legacy` (previous) behavior will be used so nothing
changes for you. If you want to use the new behaviour, set `legacy=False`. This
should only be set if you understand what it means, and thoroughly read the
reason why this was added as explained in
https://github.com/huggingface/transformers/pull/24565
Special tokens have been added in the vocabulary, make sure the associated word
embeddings are fine-tuned or trained.


Prompt: Create the lyrics of I'm the cowboy with the horse by Ulf G. Indahl

I'm the cowboy with the horse,
'cause it's a fuckin' thing, I don't know what you're going to do,
but if you want me to go,
you gotta get me in the sand,
and I know I can do it,
so I just know that I have to say that he'll tell you,
"I've got to give you" - I love you when I get to know me,
(I like you) I hate you and my nigga,
my horse and the horses, in my head,
it is oh-oo-ahl-yead ya-e-sheep-tee) (Also, yeah,
yes, no, there is no nassy)
And I like to be tayy,
she'd love to come out and tell me that you are gonna give me the same thing as
I do),
I want you to get you in your dick,
or I think I should have been able to make me feel better when you say I …

--------------------------------------------------
```

Prompt: Create the lyrics of Walking in the street by Kristian H. Liland

I'm walking in the street,
walkin' on the streets of the city,
walking with the sands of a taverna,
'Cause I don't know what you're going to do,
I know how to make it happen,
and I just know that it's not the only thing that I can do if I want it,
but I think I need it to be the best thing I could do
If I wanted it for the sake of my life I got it all done
If you want to see me in my heart,
you can see the way I walk in,
the road,walkin' around the town,
Walking in and yeah,
it is gonna be my fucking nigga
And I love you,
me and the people I have been in for so long,
take it out of there, just like I said,
"I've never been there" (I love it) I like to go out and see you in front of me,
with my wife and my daughter,
she'd love to hear me singing,
love love, we'll be able to listen to the music, …

Prompt: Create the lyrics of The sun is shining by The Leos

The sun is shining by the sun shines by
The Leoos,
the leoes and the sands of the sky,
'Cause
I'm a sun shining star,
and I can't wait to see it,
but it's not too long for me to be shining,
And I know that you're gonna see me,
I will see you,
You'll see us,
we've seen you all,
We'd love to hear you say,
"Oh, yeah, yes, no,
No, there is nothing to do with it" (Also yeah) I love you so much more than
just if you want it to shine,
The rays are shining in the air,
it is shined
By the light,

this sun, shining
By The light shining from the darkness,
you can hear it all the time,
when it comes to my mind,
what is it?
Is it too late to come?,
But I just want you to know you and your soul,
my souls were shining I want to go out of my heart,
It is too early for you? (Baby, …

EPOCH 6

Special tokens have been added in the vocabulary, make sure the associated word embeddings are fine-tuned or trained.

Prompt: Create the lyrics of I'm the cowboy with the horse by Ulf G. Indahl

I'm the cowboy with the horse 'Cause I don't know what you're gonna do,
oh, yeah, you gotta get a fuckin' nigga,
I sat in the car, and he's in my head,
but I know that you are takin to the dick,
so you know I love you so much I like you too much (I've got to do)
And if you want me to go,
then you can see me in your mouth,
my mouth is full of ev'em y'all gon' be gimmick (Yeah)
You'll never know how to make you feel like I was the
Cowboy,
with my horse And I got my hands on my knees and I get my cuffs on the back of my hand,
it'd be so good to see you,
that I want you to know you and me I just wanna know uh-huh,
hahaha-ayy-ee-hu-ya-na) I have to say …

Prompt: Create the lyrics of Walking in the street by Kristian H. Liland

I'm walking in the street,
'Cause I don't know what you're going to do,
I know you can do it,
but it's not the only thing that I can say,
"I've got a lot of things to say," he said (Yeah,
yeah)
He said, "I want you to be the one that you want me to go" (Woo)
She said that she'd like to stay in my mind,

and she said "You'll be in love with me" (Ooh) I want to see you in your eyes (Ayy)
You know,
you know I love you, right?
Is it all right now?
You can see the way it works,
just like you could do? (You can get it)
And I just want it to take me home (I just wanted it),
I got it right, it got me wrong (Hey!)
He told me that it was gonna take you back to the streets (Have you got the shit)
Then I went to bed with you (What?) I wanted to keep it up (Oh) (…

Prompt: Create the lyrics of The sun is shining by The Leos

The sun is shining,
the sun shines the sky, sun shining the skies,
and the clouds are shining
The skies of the sea,
oh
I'm a savage sun, it's shining 'Cause we're all shining in the sunshine,
but we don't know what to say
We'll be shining on the moon,
we can see what we see,
what it is, how it feels,
when it comes to us, where we are,
Where we will,Where we shall,
We will be shine,
come and go, I will come to you,
you and me
And I am letting you know that you are so far away from me,
this sun comes, light shine shine the light,
shine
The light is the shining shining shine shining bright
The sky is bright,
bright and bright (Ooo)
And if you'd like to see you in your eyes,
then we have been frightened by the darkness,
that we grew up with, like we all living together
We are in love with each other,
who we must,
Who will we? What is it?
Why do we come together?
Is it time to shine? …

```
--------------------------------------------------
EPOCH 10
Special tokens have been added in the vocabulary, make sure the associated word
embeddings are fine-tuned or trained.


Prompt: Create the lyrics of I'm the cowboy with the horse by Ulf G. Indahl

I'm the cowboy with the horse 'Cause I don't know if it's gonna be a fuckin'
nigga,
oh uh (Oh)
You're smokin', you've gotta eat it (Yeah,
yeah)
Yeah, I know you y'all gon' get tucked up (Woo)
And he'll be in the car (Hahaha)
Oh,
my liar (Ayy) He'd like to go to the park (Oh,
wow) I got the dancin' in my head (Hey!) I just wanna make it out of the way
(Uh-huh)
Then you got to see me in your head,
and you get it, but you know that you are givin' with me (Mmm-mmmm)
If you want me to do that,
that mhm-do-ma-loaf (Baby)
Y'know, what you have to say, "I' …

--------------------------------------------------


Prompt: Create the lyrics of Walking in the street by Kristian H. Liland

I'm walking in the street,
I don't know what you're going to do,
oh, yeah, you know how to make me feel like a shit,
but it's not the only thing that I can do 'Cause I know that you are gonna be
the one I want to be
Then you can see me in your eyes,
and I just want you to see you in my eyes
And if you want me to know me,
then I will be able to tell you,
"I'll be in love with you" (It'd be like you) I love you when I go to the gym,
we've got to go out of town (Yeah)
You know,
how you got me on my knees
And I gotta get up and run up,
runnin' on the floor (You know it)
And you just know when you fuck me up (Hey)
Y'all have to take me down (Hahaha) (Ayy),
```

yay-a-hh-huh)
Yeah, my nigga, that is what I …

Prompt: Create the lyrics of The sun is shining by The Leos

The sun is shining,
the sun shines and the light of the skies
And the clouds are shining (The sun shining)
And if you want to see me,
I'm a sun-shined sun 'Cause I don't know what to say,
it's gonna be, oh, you're savage,
and I can see you in the sky
And I know that you are so good,
but I am so bad, so I have to be able to do it,
now I will be there for you,
this sun, shine, shining the and light from the shining sun (Leosse) and shine
(Alright) I love you and my sons (I love)
You love me so much,
my love is and it is so strong (Hey!)
And when you think of me I think that I should be with you (Sunday,
day, night)
Oh, what he did? (Yeah)
So I want you to hear me in your eyes
And you will hear your voice,
then you can hear you out of my mind
And my heart will grow and frightened …

Source code

The source code behind this report can be found here.

Deep learning libraries

During implementation, PyTorch was used.

Assistance

The author acknowledge the Orion High Performance Computing Center (OHPCC) at the Norwegian University of Life Sciences (NMBU) for providing computational resources that have contributed to the research results reported within this paper.

During creation of the codebase, GitHub Copilot along with Mistral was used –mostly helping with tensor dimension issues.

Prompts equivalent to

```
How do I reshape the tensor [...]  to correspond with this [...]
Modify this code such that shapes match [...]
Smooth and plot the contents of a csv-file using pandas.
```

were used. *I.e.*, sparring when I was stuck on something.

During writing of this report, **no** artificial intelligence tools were used to generate text. Mistral was however used when either shortening or correcting/rephrasing certain paragraphs/sentences.

Prompts equivalent to

```
Shorten the paragraph [...]
Highlight errors in this paragraph [...]
```

were used.