Transformers

Greg Strabel

May 21, 2023

1 Preliminaries

1.1 Matrix Multiplication

Given two matrices $X \in \mathbb{R}^{n \times m}$ and $Y \in \mathbb{R}^{m \times k}$

$$(XY)_{ij} = \sum_{l=1}^{m} X_{il} Y_{lj} = X_{i \cdot} Y_{\cdot j} \tag{1}$$

Therefore

$$(XY)_{\cdot j} = \sum_{l=1}^{m} Y_{lj} X_{\cdot l} \tag{2}$$

so that the columns of XY are linear combinations of the columns of X and

$$(XY)_{i \cdot} = \sum_{l=1}^{m} X_{il} Y_{l \cdot} \tag{3}$$

so that the rows of XY are linear combinations of the rows of Y.

1.2 Softmax

Definition 1.1 (Softmax)**.** The softmax function $\sigma : \mathbb{R}^K \to \mathbb{R}^K$ is

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \tag{4}$$

2 Attention

2.1 Dot-Product Attention

Definition 2.1 (Dot-Product Attention)**.** Given $Q \in \mathbb{R}^{d_l \times d_k}$, $K \in \mathbb{R}^{d_s \times d_k}$ and $V \in \mathbb{R}^{d_s \times d_v}$

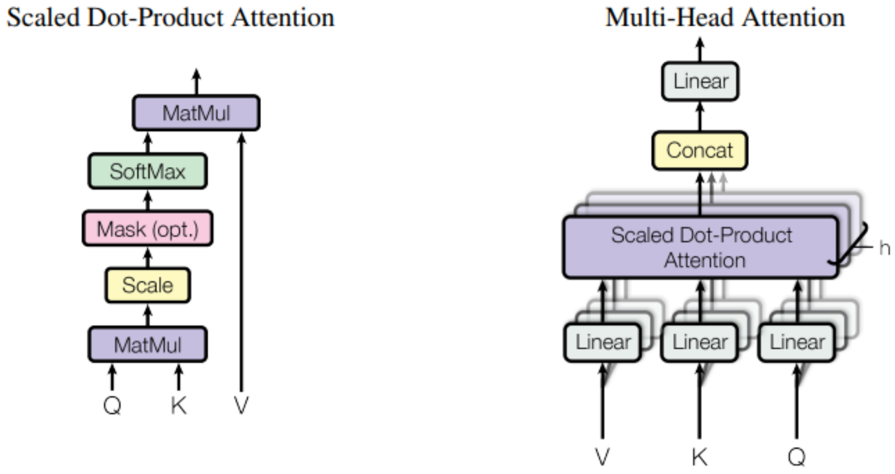


Figure 2: (left) Scaled Dot-Product Attention. (right) Multi-Head Attention consists of several attention layers running in parallel.

Figure 1: Attention Mechanism [VSP+17]

$$\text{Attention}\left(Q,K,V\right)=\text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V\in\mathbb{R}^{d_l\times d_v} \tag{5}$$

2.2 Multihead Attention

Given input matrices $X_Q\in\mathbb{R}^{d_l\times d_g}$, $X_K\in\mathbb{R}^{d_s\times d_w}$ and $X_V\in\mathbb{R}^{d_s\times d_u}$ and weight matrices

$$\begin{gathered}\{W_Q^i\in\mathbb{R}^{d_g\times d_k}\}_{i=1}^h\\ \{W_K^i\in\mathbb{R}^{d_w\times d_k}\}_{i=1}^h \qquad (6)\\ \{W_V^i\in\mathbb{R}^{d_u\times d_v}\}_{i=1}^h\\ W_O\in\mathbb{R}^{hd_v\times d_o} \qquad (7)\end{gathered}$$

we define

$$\begin{aligned}Q^i&=X_QW_Q^i\in\mathbb{R}^{d_l\times d_k}\\ K^i&=X_KW_K^i\in\mathbb{R}^{d_s\times d_k}\\ V^i&=X_VW_V^i\in\mathbb{R}^{d_s\times d_v}\end{aligned} \tag{8}$$

$$A^i = \text{Attention}\left(Q^i, K^i, V^i\right) \in \mathbb{R}^{d_l \times d_v} \tag{9}$$

$$\text{Multihead}\left(X_Q, X_K, X_V\right) = \text{concat}\left(A_1, ..., A_h\right) W_O \in \mathbb{R}^{d_l \times d_o} \tag{10}$$

2.3 Multihead Self-Attention

In Multihead Self-Attention, $X_Q = X_K = X_V$ so that $d_{model} \coloneqq d_g = d_w = d_u$, $L \coloneqq d_l = d_s$ and $\text{Multihead}\left(X_Q, X_K, X_V\right) \in \mathbb{R}^{L \times d_o}$. If we also have $d_o = d_{model}$ then we get the Multihead Self-Attention described in [VSP+17]Attention Is All You Need

2.4 Transformer Block

A single transformer block takes an input tensor X_0 and then:

1. Applies a multi-head attention layer to X_0 to produce X_1
2. Adds X_0 and X_1 and applies normalization to produce X_2
3. Applies a fully connected feed forward layer to X_2 to produce X_3
4. Adds X_2 and X_3 and applies normalization to produce the final output X_4

3 Attention is All You Need [VSP+17]

Multi-head self-attention was introduced in the paper Attention is All You Need [VSP+17], which used the architecture in 3 for a translation task.

Inputs to the model are Byte Pair Encoded (BPE, section 12) tokens, which are mapped to a learned embedding space. In order for the model to make use of the order of the input sequence, information on sequence order is injected by adding in sine and cosine function of different frequencies:

$$\begin{aligned} PE_{(pos,2i)} &= \sin(pos/10000^{2i/d_{model}}) \\ PE_{(pos,2i+1)} &= \cos(pos/10000^{2i/d_{model}}) \end{aligned} \tag{11}$$

4 Improving Language Understanding by Generative Pre-Training

[RNSS18] is the OpenAI article that introduced GPT (Generative Pre-Training). This framework uses unsupervised pre-training of a multi-layer transformer decoder with a

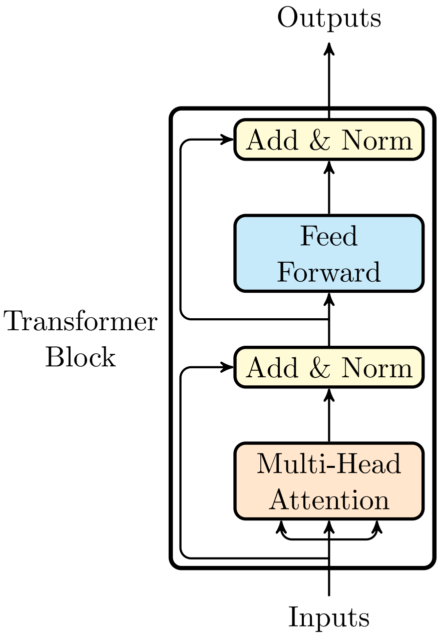


Figure 2: Single Transformer Block

masked language model objective followed by fine-tuning for specific tasks.

Unsupervised pre-training involves taking a corpus of tokens $\mathcal{U} = \{u_1, u_2, ..., u_n\}$ and training a model to maximize the likelihood objective:

$$L_1(\mathcal{U}) = \sum_i \log P(u_i | u_{i-k}, ..., u_{i-1}; \Theta) \tag{12}$$

where k is the size of the context window, P is the conditional probability of the next token and Θ are the parameters of the model.

[RNSS18] use a multi-layer transformer decoder for their model; a stack of multiple transformer blocks:

$$\begin{aligned} &h_0 = UW_e + W_p \\ &h_l = \texttt{transformer\_decoder}(h_{l-1}) \quad \forall i \in [1, ..., n] \\ &P(u) = \texttt{softmax}(h_n W_e') \end{aligned} \tag{13}$$

where $U = (u_{-k}, ..., u_{-1})$ is the context vector of tokens, n is the number of layers, W_e is the token embedding matrix, and W_p is the position embedding matrix.

Supervised fine-tuning uses labeled training data to further optimize the model for certain tasks, including classification, semantic similarity, textual entailment and question

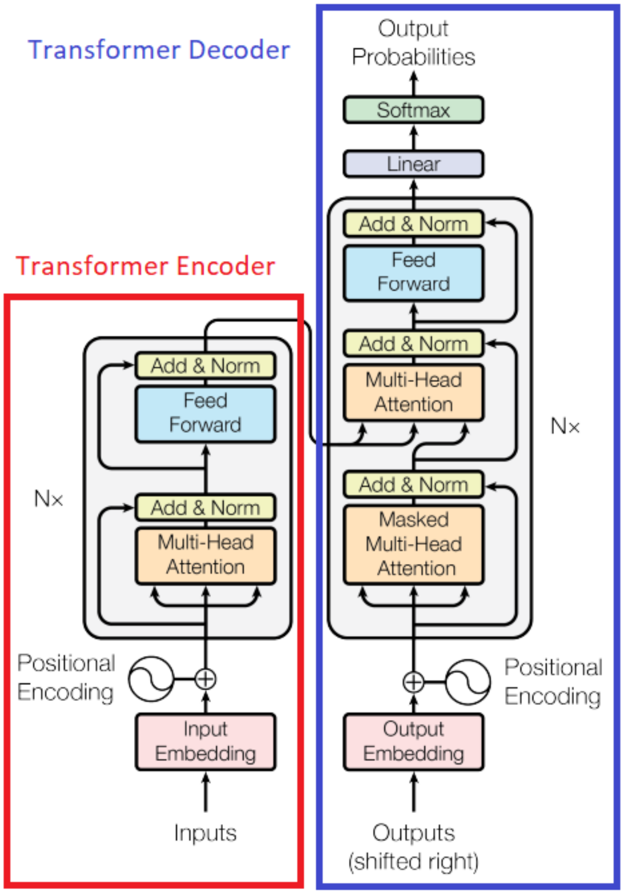


Figure 1: The Transformer - model architecture.

Figure 3: Transformer Architecture [VSP+17]

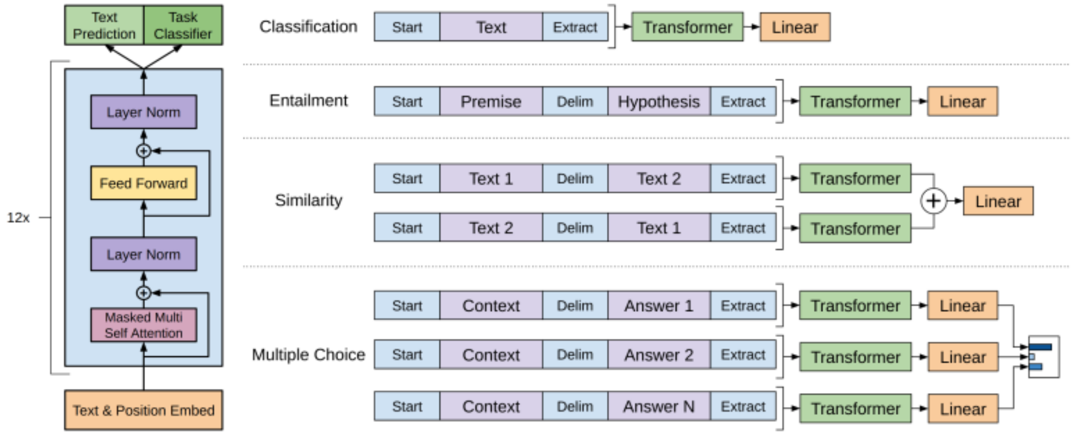


Figure 1: **(left)** Transformer architecture and training objectives used in this work. **(right)** Input transformations for fine-tuning on different tasks. We convert all structured inputs into token sequences to be processed by our pre-trained model, followed by a linear+softmax layer.

Figure 4: GPT Training Objectives [RNSS18]

answering. Given a labeled dataset $\mathcal{C}$ of inputs $\{x^1, ..., x^m\}$ and targets y, the model is trained to maximize the likelihood objective:

$$L_2(\mathcal{C}) = \sum_{(x,y)} \log P(y|x^1, ..., x^m) \tag{14}$$

5 Language Models are Unsupervised Multitask Learners [RWC+18]

[RWC+18] is the paper from OpenAI that introduced GPT-2. The principle finding of this paper was that by scaling up both model size from approximately 100M parameters to 1.5B parameters and using a significantly larger pre-training dataset (WebText - based on Common Crawl data with outbound Redit links), a Transformer-decoder type model could achieve state-of-the-art performance on many tasks without the need to fine-tune.

6 Language models are few-shot learners [BMR+20]

[BMR+20] introduced GPT-3.

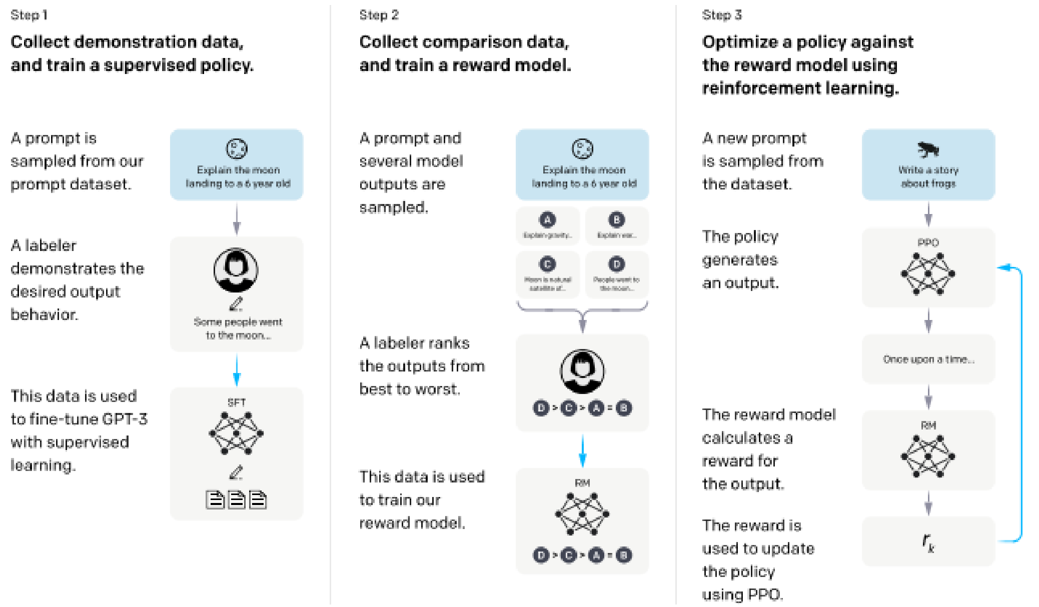

Step 1
Collect demonstration data, and train a supervised policy.
A prompt is sampled from our prompt dataset.
Explain the moon landing to a 6 year old
A labeler demonstrates the desired output behavior.
Some people went to the moon...
This data is used to fine-tune GPT-3 with supervised learning.
SFT
Step 2
Collect comparison data, and train a reward model.
A prompt and several model outputs are sampled.
Explain the moon landing to a 6 year old
A
Explain gravity...
B
Explain war...
C
Moon is natural satellite of...
D
People went to the moon...
A labeler ranks the outputs from best to worst.
D > C > A = B
This data is used to train our reward model.
RM
D > C > A = B
Step 3
Optimize a policy against the reward model using reinforcement learning.
A new prompt is sampled from the dataset.
Write a story about frogs
The policy generates an output.
PPO
Once upon a time...
The reward model calculates a reward for the output.
RM
The reward is used to update the policy using PPO.
r_k

Figure 2: A diagram illustrating the three steps of our method: (1) supervised fine-tuning (SFT), (2) reward model (RM) training, and (3) reinforcement learning via proximal policy optimization (PPO) on this reward model. Blue arrows indicate that this data is used to train one of our models. In Step 2, boxes A-D are samples from our models that get ranked by labelers. See Section 3 for more details on our method.

Figure 5: Instruct GPT Fine-Tuning [OWJ+22]

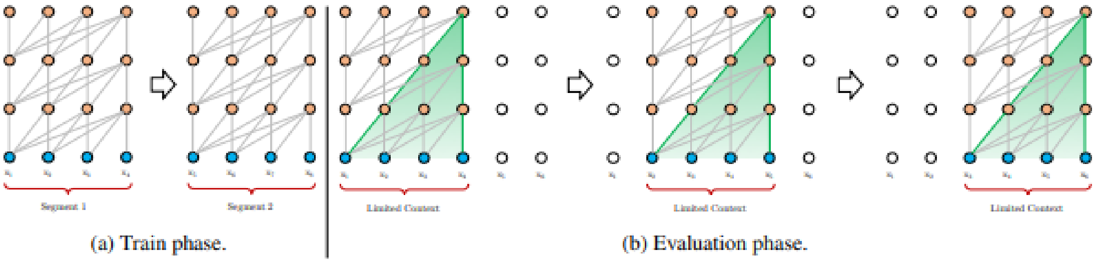
7 Training language models to follow instructions with human feedback [OWJ+22]

8 BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

[DCLT19] was the first paper to introduce BERT models - transformer encoder only models that can be fine-tuned for a variety of tasks.

9 Scaling Transformers to Longer Sequences

One limitation of the original transformers is that their computational complexity grows quadratically in the length of the input sequence; that is, they have computational complexity of $O(n^2)$, where n is the sequence length. There have been several approaches



(a) Train phase.

(b) Evaluation phase.

Figure 1: Illustration of the vanilla model with a segment length 4.

Figure 6: Transformer with Fixed Context Window [DYY+19]

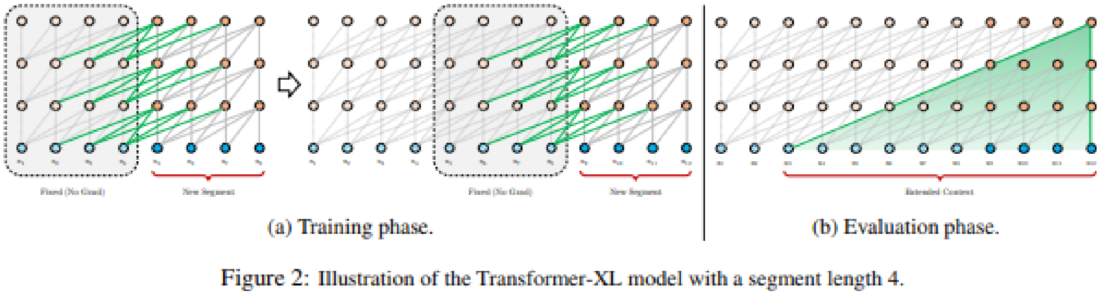
developed to reduce this complexity.

9.1 Transformer-XL [DYY+19]

One crude option to reduce computational complexity during training is to split text into segments of length L and train a model on the individual segments, ignoring all contextual information from previous segments. This reduces complexity during training but causes contextual fragmentation as no information flows across segments. This vanilla model is shown in 6. At each time step during evaluation, the model processes a text segment of length L, the last output position is recorded and then the context window is shifted to the right by one step and the process repeated. By shifting only a single time step, each prediction is able to use the context of the last L positions, alleviating the contextual fragmentation in training, but reintroducing computational complexity.

In order to address the issue of contextual fragmentation, [DYY+19] introduce a recurrence mechanism. With two consecutive segments of length L, $s_\tau = [x_{\tau,1}, ..., x_{\tau,L}]$ and $s_{\tau+1} = [x_{\tau+1,1}, ..., x_{\tau+1,L}]$, let the n-th layer hidden state sequence produced by the τ-th segment s_τ be $h_\tau^n \in \mathbb{R}^{L \times d}$, where d is the hidden dimension. Then

$$\begin{aligned} \tilde{h}_{\tau+1}^{n-1} &= \left[SG(h_\tau^{n-1}) \circ h_{\tau+1}^{n-1}\right] \\ q_{\tau+1}^n, k_{\tau+1}^n, v_{\tau+1}^n &= h_{\tau+1}^{n-1} W_q', \tilde{h}_{\tau+1}^{n-1} W_k', \tilde{h}_{\tau+1}^{n-1} W_v' \\ h_{\tau+1}^n &= \texttt{transformer\_block}(q_{\tau+1}^n, k_{\tau+1}^n, v_{\tau+1}^n) \end{aligned} \tag{15}$$



(a) Training phase.

(b) Evaluation phase.

Figure 2: Illustration of the Transformer-XL model with a segment length 4.

Figure 7: TransformerXL [DYY+19]

9.2 Generating Long Sequences with Sparse Transformers

In [CGRS19], the authors scale transformers to longer sequences by factorizing the self-attention mechanism.

10 Improving Fine-tuning

10.1 LoRA: Low-Rank Adaptation of Large Language Models [HSW+21]

11 Relative Position Embeddings

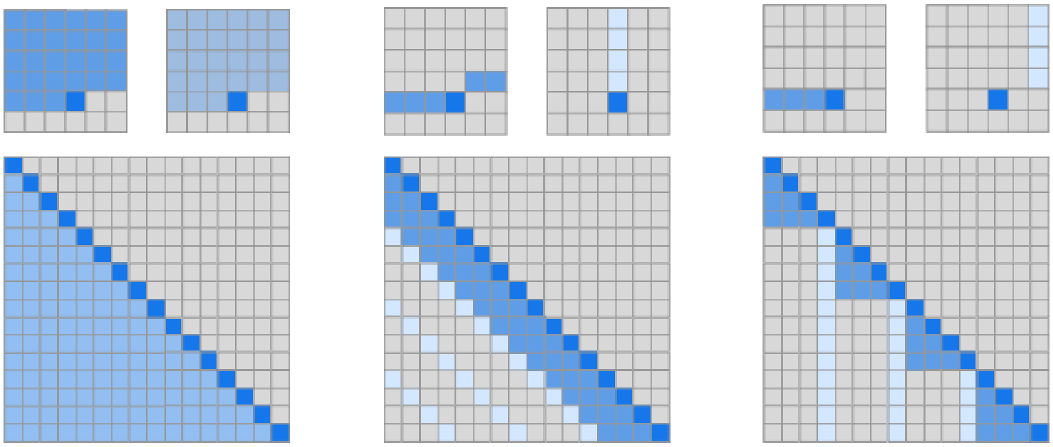
Given relative position embedding matrix $E^r \in \mathbb{R}^{L\times d_{model}}$ and matrix $X \in \mathbb{R}^{L\times d_{model}}$

12 Byte-Pair Encoding

```python
import re, collections

def get_stats(vocab):
    pairs = collections.defaultdict(int)
    for word, freq in vocab.items():
        symbols = word.split()
        for i in range(len(symbols)-1):
            pairs[symbols[i],symbols[i+1]] += freq
    return pairs

def merge_vocab(pair, v_in):
    v_out = {}
    bigram = re.escape(' '.join(pair))
    p = re.compile(r'(?<!\S)' + bigram + r'(?!\S)')
    for word in v_in:
        w_out = p.sub(''.join(pair), word)
        v_out[w_out] = v_in[word]
```



(a) Transformer

(b) Sparse Transformer (strided)

(c) Sparse Transformer (fixed)

Figure 3. Two 2d factorized attention schemes we evaluated in comparison to the full attention of a standard Transformer (a). The top row indicates, for an example 6x6 image, which positions two attention heads receive as input when computing a given output. The bottom row shows the connectivity matrix (not to scale) between all such outputs (rows) and inputs (columns). Sparsity in the connectivity matrix can lead to significantly faster computation. In (b) and (c), full connectivity between elements is preserved when the two heads are computed sequentially. We tested whether such factorizations could match in performance the rich connectivity patterns of Figure 2.

Figure 8: Sparse Transformer Attention [CGRS19]

Algorithm 1: Relative Position Embedding

```
Input: Relative embedding matrix $E^r \in \mathbb{R}^{L \times d_{model}}$
       Matrix $X \in \mathbb{R}^{L \times d_{model}}$
Output: $D \in \mathbb{R}^{L \times L}$
$A \leftarrow X E^{rT} \in \mathbb{R}^{L \times L}$
$M \leftarrow \begin{cases} m_{i,j} = 1 & i \leq L, j \leq L, i \geq j \\ m_{i,j} = 0 & i \leq L, j \leq L, i < j \end{cases} \in \mathbb{R}^{L \times L}$
$A \leftarrow A \odot M$    // element-wise product
$B \leftarrow \begin{cases} b_{i,j} = 0 & j = 1, i \leq L \\ b_{i,j} = a_{i,j-1} & i \leq L, 1 < j \leq L+1 \end{cases} \in \mathbb{R}^{L \times L+1}$
$V \leftarrow \mathrm{vec}(B^T)$
$C \leftarrow \{c_{ij} = V_{(i-1)L+j} \mid i \leq L, j \leq L\} \in \mathbb{R}^{L+1 \times L}$
$D \leftarrow \{d_{ij} = c_{i+1,j} \mid i \leq L, j \leq L\}$
return $D$
```

```
    return v_out

vocab = {'l o w </w>' : 5, 'l o w e r </w>' : 2,
         'n e w e s t </w>':6, 'w i d e s t </w>':3}

num_merges = 10

for i in range(num_merges):
    pairs = get_stats(vocab)
    best = max(pairs, key=pairs.get)
    vocab = merge_vocab(best, vocab)
    print(best)
```

Listing 1: Byte-Pair Encoding

Papers to cover:

- Attention is All You Need - BERT - Transformer XL - GPT papers - Sparse Transformers - Roformer - Roberta - albert

References

[BMR+20] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

[CGRS19] Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating long sequences with sparse transformers, 2019.

[DCLT19] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

[DYY+19] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V. Le, and Ruslan Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context, 2019.

[HSW+21] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models, 2021.

[OWJ+22] Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback, 2022.

[RNSS18] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. 2018.

[RWC$^+$18] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2018.

[VSP$^+$17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.