PyTorch implements a function for self-attention and causal attention that supports FlashAttention for efficiency. This function is beta and subject to change:

- `scaled_dot_product_attention` documentation: https://mng.bz/NRJd

PyTorch also implements an efficient `MultiHeadAttention` class based on the `scaled_dot_product` function:

- `MultiHeadAttention` documentation: https://mng.bz/DdJV

Dropout is a regularization technique used in neural networks to prevent overfitting by randomly dropping units (along with their connections) from the neural network during training:

- "Dropout: A Simple Way to Prevent Neural Networks from Overfitting" (2014) by Srivastava et al., https://jmlr.org/papers/v15/srivastava14a.html

While using the multi-head attention based on scaled-dot product attention remains the most common variant of self-attention in practice, authors have found that it's possible to also achieve good performance without the value weight matrix and projection layer:

- "Simplifying Transformer Blocks" (2023) by He and Hofmann, https://arxiv.org/abs/2311.01906

Chapter 4

The following paper introduces a technique that stabilizes the hidden state dynamics neural networks by normalizing the summed inputs to the neurons within a hidden layer, significantly reducing training time compared to previously published methods:

- "Layer Normalization" (2016) by Ba, Kiros, and Hinton, https://arxiv.org/abs/1607.06450

Post-LayerNorm, used in the original transformer model, applies layer normalization after the self-attention and feed forward networks. In contrast, Pre-LayerNorm, as adopted in models like GPT-2 and newer LLMs, applies layer normalization before these components, which can lead to more stable training dynamics and has been shown to improve performance in some cases, as discussed in the following papers:

- "On Layer Normalization in the Transformer Architecture" (2020) by Xiong et al., https://arxiv.org/abs/2002.04745
- "ResiDual: Transformer with Dual Residual Connections" (2023) by Tie et al., https://arxiv.org/abs/2304.14802

A popular variant of LayerNorm used in modern LLMs is RMSNorm due to its improved computing efficiency. This variant simplifies the normalization process by normalizing the inputs using only the root mean square of the inputs, without subtracting the mean before squaring. This means it does not center the data before computing the scale. RMSNorm is described in more detail in

- "Root Mean Square Layer Normalization" (2019) by Zhang and Sennrich, https://arxiv.org/abs/1910.07467

The Gaussian Error Linear Unit (GELU) activation function combines the properties of both the classic ReLU activation function and the normal distribution's cumulative distribution function to model layer outputs, allowing for stochastic regularization and nonlinearities in deep learning models:

- "Gaussian Error Linear Units (GELUs)" (2016) by Hendricks and Gimpel, https://arxiv.org/abs/1606.08415

The GPT-2 paper introduced a series of transformer-based LLMs with varying sizes—124 million, 355 million, 774 million, and 1.5 billion parameters:

- "Language Models Are Unsupervised Multitask Learners" (2019) by Radford et al., https://mng.bz/DMv0

OpenAI's GPT-3 uses fundamentally the same architecture as GPT-2, except that the largest version (175 billion) is 100x larger than the largest GPT-2 model and has been trained on much more data. Interested readers can refer to the official GPT-3 paper by OpenAI and the technical overview by Lambda Labs, which calculates that training GPT-3 on a single RTX 8000 consumer GPU would take 665 years:

- "Language Models are Few-Shot Learners" (2023) by Brown et al., https://arxiv.org/abs/2005.14165
- "OpenAI's GPT-3 Language Model: A Technical Overview," https://lambdalabs.com/blog/demystifying-gpt-3

NanoGPT is a code repository with a minimalist yet efficient implementation of a GPT-2 model, similar to the model implemented in this book. While the code in this book is different from nanoGPT, this repository inspired the reorganization of a large GPT Python parent class implementation into smaller submodules:

- "NanoGPT, a Repository for Training Medium-Sized GPTs, https://github.com/karpathy/nanoGPT

An informative blog post showing that most of the computation in LLMs is spent in the feed forward layers rather than attention layers when the context size is smaller than 32,000 tokens is:

- "In the Long (Context) Run" by Harm de Vries, https://www.harmdevries.com/post/context-length/

Chapter 5

For information on detailing the loss function and applying a log transformation to make it easier to handle for mathematical optimization, see my lecture video:

- L8.2 Logistic Regression Loss Function, https://www.youtube.com/watch?v=GxJe0DZvydM

The following lecture and code example by the author explain how PyTorch's cross-entropy functions works under the hood:

- L8.7.1 OneHot Encoding and Multi-category Cross Entropy, https://www.youtube.com/watch?v=4n71-tZ94yk
- Understanding Onehot Encoding and Cross Entropy in PyTorch, https://mng.bz/o05v

The following two papers detail the dataset, hyperparameter, and architecture details used for pretraining LLMs:

- "Pythia: A Suite for Analyzing Large Language Models Across Training and Scaling" (2023) by Biderman et al., https://arxiv.org/abs/2304.01373
- "OLMo: Accelerating the Science of Language Models" (2024) by Groeneveld et al., https://arxiv.org/abs/2402.00838

The following supplementary code available for this book contains instructions for preparing 60,000 public domain books from Project Gutenberg for LLM training:

- Pretraining GPT on the Project Gutenberg Dataset, https://mng.bz/Bdw2

Chapter 5 discusses the pretraining of LLMs, and appendix D covers more advanced training functions, such as linear warmup and cosine annealing. The following paper finds that similar techniques can be successfully applied to continue pretraining already pretrained LLMs, along with additional tips and insights:

- "Simple and Scalable Strategies to Continually Pre-train Large Language Models" (2024) by Ibrahim et al., https://arxiv.org/abs/2403.08763

BloombergGPT is an example of a domain-specific LLM created by training on both general and domain-specific text corpora, specifically in the field of finance:

- "BloombergGPT: A Large Language Model for Finance" (2023) by Wu et al., https://arxiv.org/abs/2303.17564

GaLore is a recent research project that aims to make LLM pretraining more efficient. The required code change boils down to just replacing PyTorch's `AdamW` optimizer in the training function with the `GaLoreAdamW` optimizer provided by the `galore-torch` Python package:

- "GaLore: Memory-Efficient LLM Training by Gradient Low-Rank Projection" (2024) by Zhao et al., https://arxiv.org/abs/2403.03507
- GaLore code repository, https://github.com/jiaweizzhao/GaLore

The following papers and resources share openly available, large-scale pretraining datasets for LLMs that consist of hundreds of gigabytes to terabytes of text data:

- "Dolma: An Open Corpus of Three Trillion Tokens for LLM Pretraining Research" (2024) by Soldaini et al., https://arxiv.org/abs/2402.00159