Deep Learning KU (DAT.C302UF), WS24
Assignment 3
Autoregressive Language Modeling with Transformers

Thomas Wedenig
thomas.wedenig@tugraz.at

Teaching Assistant:	Patrick Ebnicher
Points to achieve:	25 pts
Deadline:	15.01.2025 23:59
Hand-in procedure:	You can work in groups of **at most two people**. **Exactly one** team member uploads two files to TeachCenter: **The report (.pdf)** and **the Jupyter Notebook (.ipynb)**. The first page of the report must be the **cover letter**. Do not upload a folder. Do not zip the files.
Plagiarism:	If detected, 0 points for all parties involved. If this happens twice, we will grade the group with "Ungültig aufgrund von Täuschung"

General Remarks

- In the classes `CausalSelfAttention`, `MLP`, and `GPT`, you are *only* allowed to add code to the respective `TODO` sections. Do not change the method signatures.
- Code in comments will not be executed or graded.
- Make sure all of your plots have labeled axes and are clearly described in your report.
- For all the tasks below, create the appropriate code and clearly discuss all findings in the PDF report. **The submitted code should be able to reproduce your results**.

Autoregressive Language Modeling

Autoregressive language models aim to model the probability distribution over sequences of discrete *tokens*[1]. Given a sequence $\mathbf{x} = (x_1, \ldots, x_t)^\top$, we can always decompose the joint probability $p_\theta(\mathbf{x})$ using the chain rule of probability:

$$p_\theta(\mathbf{x}) = p_\theta(x_1) \prod_{k=2}^{t} p_\theta(x_k | \mathbf{x}_{<k}) \tag{1}$$

where $\mathbf{x}_{<k} = (x_1, \ldots, x_{k-1})^\top$ represents all tokens before position k. This decomposition allows us to model the probability of each token conditioned on all previous tokens in the sequence. Generative Pre-trained Transformers (GPTs) essentially use Transformer-based neural networks to model $p_\theta(x_t | \mathbf{x}_{<t})$. Transformers utilize the self-attention mechanisms to capture long-range dependencies between tokens, making them particularly effective for generating coherent text by sampling one token at a time from these conditional distributions.

[1] Tokens are usually sub-word units – but for simplicity we will consider each *character* to be a token in this assignment.

Task details:

a) (1 pts) : Get familiar with the dataset and briefly analyze its structure. What is returned when we fetch a batch of data from this dataset using a dataloader? Describe what the `block_size` parameter controls.

a) (8 pts) : Using only `PyTorch` primitives[2], implement the `CausalSelfAttention` class for multiple attention heads. This class will receive a batch of sequences of embeddings and performs causally masked, multi-head scaled dot-product self-attention.

Follow the `TODOs` in the code. Explain in your report what Q, K, and V represent in this context and how they interact to produce attention weights. Write down the dimensionality of Q, K, and V in the multi-head attention setting. What would happen if we would *not* apply the causal mask?

c) (2 pts) : Implement the `MLP` class that follows each attention block. Follow the `TODOs` in the code. For the `GELU` activation function you can use `nn.GELU`[3] from `PyTorch`.

d) (3 pts) : A `Block` module which consists of `CausalSelfAttention` and `MLP` modules (as well as `LayerNorm` and residual connections) is already provided for you. Carefully read the code of the `GPT` class and complete the forward pass.

e) (4 pts) : The `GPT` model $f_\theta(\mathbf{x}_{<t})$ not only outputs the *logits* $\mathbf{z}_t$ for the probability mass function $p_\theta(x_t \,|\, \mathbf{x}_{<t})$, but also the logits for all distributions $\{p_\theta(x_k \,|\, \mathbf{x}_{<k})\}_{k=2}^{t}$. Recall that given the logits $\mathbf{z}_t$, we can compute the corresponding probability mass function as

$$p_\theta(x_t \,|\, \mathbf{x}_{<t}) = softmax(\mathbf{z}_t)_{x_t}$$

Given a temperature value $\tau > 0$, we define

$$p_\theta^\tau(x_k \,|\, \mathbf{x}_{<k}) = softmax(\mathbf{z}_k/\tau)_{x_k}$$

Implement the `sample` method in the `GPT` class, which receives a temperature τ and a starting sequence $(x_1, \ldots, x_t)$ and autoregressively samples $x_{t+i} \sim p_\theta^\tau(x_{t+i} \,|\, \mathbf{x}_{<t+i})$ for $i = 1, \ldots,$ `max_new_tokens`. Can we condition on arbitrarily long sequences $\mathbf{x}_{<t+i}$?

f) (2 pts) : Train the model using appropriate hyperparameters. The default hyperparameters should serve as a good starting point, but feel free to change them if you think this is necessary. Create a plot showing the training and validation loss over iterations. Analyze the training dynamics and comment on the model's convergence behavior.

g) (2 pts) Generate samples from your trained model using temperatures $\tau \in \{1.5, 1.0, 0.8, 0.5, 0.1, 0.0001\}$. Include representative samples for each temperature in your report and comment how τ influences the generated text.

h) (3 pts) Experiment with increasing the model size (number of blocks, embedding dimension, number of heads, etc.) and block size. Find appropriate training hyperparameters (learning rate, batch size, etc.). Report the number of model parameters and your choice of hyperparameters in your report. Create a plot showing the training and validation loss over iterations for this larger model. Generate samples with an appropriate value of τ and compare the performance with the baseline model. Briefly comment on the computational requirements (i.e., which hardware you have used and how long a training run took).

[2] i.e., do *not* use `PyTorch` functions that directly compute the attention mechanism

[3] `https://pytorch.org/docs/stable/generated/torch.nn.GELU.html`