Transformers from scratch
Bachelor AI Project 2025

This is a shared Bachelor project. All students participating will work independently on the same subject. In the first half, you will all complete the same assignment. In the second half, you will build on this assignment to investigate a separate research question each.

Notes:

- This project will be very technical and lean heavily on pytorch and some advanced machine learning concepts. It's not for everybody. If you're wondering whether this is for you, please work through the 5th worksheet of the ML course and read this blog post on transformer models. You will need to understand these concepts well enough to implement them from scratch.
- Starting this year, **the project will only be available for AI students.** I'm no longer able to supervise BSc CS students, sorry.
- If you are a CS student, or the project is full, you might still be able to join the shared part if you find another supervisor to take care of the second half of the project (including grading).

Shared assignment
The main task is to build an autoregressive transformer model (think GPT) from scratch.

During the weekly meetings, we will look into the details of how pytorch works and how self-attention works. During this time, there will be fixed weekly assignments.

During this phase of the project, there is a fairly strict program with a weekly target, to make sure that everybody achieves the same goal by the end of the project. Make sure to put in enough time and not to fall behind. If you have trouble with any part, remember that the other students in the project are doing the same assignments, so you have plenty of people to ask for help.

Individual assignment
There are many directions to explore once you have the model running. Here are some ideas (in increasing order of complexity). Every student should work on a different subject, so some coordination will be required. We will spend some time during all group meetings brainstorming directions.

You can build on the code you created in the first six weeks or switch to an existing transformers library, using only the knowledge that you now have of its internals. What is wise depends on your research question.

Here are some possibilities:

- Build a (stripped down) Perceiver architecture (similar to the transformer, but with potentially much longer contexts) and compare the performance of the two models on a simple task like autoregressive modeling.

- Perform a *cramming* experiment. Pick a simple model that is a few years old (like GPT-1 or BERT) and see how close you can get to its performance in a single 12 hour run. Note that 12 hours is on the edge of how much time it takes to learn a remotely competent language model, so the tasks should be relatively simple (for instance, generative summarization won't work, because the model won't yet have learned to generate coherent language).
- Pick a simple task, like autoregressive modeling, and study different ways of *augmenting* the training data. Test the performance on different out-of-domain (OOD) datasets to see whether augmentation makes the model more robust too OOD data.
- Take a pretrained model like GPT2, and see whether it can be made "recurrent" using the following trick: for a sequence of (c-1) * n tokens, where c is the maximum context length of GPT2, run the first c tokens through the model. Then, run the next c-1 tokens, but prepend the last hidden representation from the previous run (either at the start or at every layer). Repeat this trick for the whole sequence. Does this allow the model, without retraining, to retain some of the longer context length?
- Turn a pretrained model, like GPT2, into a variational autoencoder for sentences, by inserting adapter layers. See if you can separate style from content in the latent space. (This one is a bit ambitious).
- Take a pretrained transformer model, and distill it into an old-fashioned LSTM. How many layers of LSTM do you need to maintain performance? Could the stronger inductive bias of the LSTM lead to *better* performance than the original transformer over contexts that are longer than those seen in training?

You can also come up with an idea of your own, or a variation on one of the above ideas as the project progresses. **However, we've found that students often pick topics that are too ambitious for a 6 week period.** It's best to take the above projects as a starting point and to see if you can tweak it for what you're interested in. A good project has the following properties:

- It builds on a simple model (avoid encoder/decoder models, or complex architectures).
- It poses a single, simple research question.
- It requires limited computational resources. Training a model from scratch that can generate coherent language takes at least 48 hours, and you'll need to run your experiments many times over. So this is likely already out of bounds (but you can always build on a pretrained model).

It's fine to pick a project that is a bit ambitious, but if you do, try to build it on top of a simpler, safer project that does satisfy the above requirements. That way, you can always scale back if you find you've bit off more than you can chew.

Deadlines, dates, program

You are expected to work on this project (close to) full time, so you should expect to work for at least 32 hours per week, or 4 full working days to complete the tasks set for each week.

- The project will officially start **on the first· wednesday of period 5** (April 2)
- Shared assignment: you should have a running transformer model generating sentences on **May 7st** at the latest

- Choose project topic: You should choose the topic for your individual assignment by **May 14th.**
- Final presentation: This is a poster presentation. The possible dates will be set by the BSc project organizers.
- Final deadline: Last Friday of P6 (Usually the end of June).

The final deadline is strict*ish*. A few days extra is acceptable if you need them, but don't expect to work more than a week into July, since the whole grading process needs to be finished well before July 25 if you want to graduate without paying extra tuition.

Shared program

week 1: Understanding pytorch

Reading: DLVU, MLVU backpropagation

Practical:
- Fifth MLVU worksheet
- Pytorch 60 minute blitz

Complete both the tutorials above, and take plenty of time to play around with the resulting models. Try to change things, and build a proper understanding of how pytorch works.

week 2: Data preprocessing

Practical:
- DLVU assignment 4
Load the IMDb data for sentiment classification.

week 3: Implementing attention

Reading:
- Transformers from scratch
- DLVU Self-attention

Practical:
Implement first the **simple self attention** from the materials above, and insert it into your classifier from the previous week. Then add the three features required for proper self-attention: scaling, qkv projections and multi-headedness.

See how much your classifier improves.

week 4: building a transformer

- Classifier
- Autoregressive

week 5: training your transformer

- moving to the cluster
- installing Weights and Biases
- basic training methods:
    - Finding your batch size
    - Finding your learning rate
        - Range testing.
    - Gradient norm, LR scheduling, gradient clipping

week 6: adding a special feature

If there is time left over, you can experiment with making a small tweak to your code, possibly as a proof of concept for the main project.

Choose from:
- Building a perceiver layer (not the full model, but the basic rewiring of the attention)
- Building an encoder/decoder model
- Adding relative position embeddings. This is a bit technically challenging, but if you give yourself a few hours to go through it slowly step by step, you should be able to work out the basic idea.

weeks 7 to ~12

Pick a project and implement it. You can build on the code you wrote in the first 6 weeks, or use the understanding you developed in the first six weeks to take some existing code (for instance from HuggingFace) and to modify it for your purpose.