PyTorch + Slurm Tutorial on Pawsey

Protein Language Model Training

George Bouras

University of Adelaide
Australia

@GB13Faithless gbouras13

THE UNIVERSITY of ADELAIDE

Flinders Accelerator for Microbiome Exploration

GitHub Repository

gbouras13

- Follow along https://github.com/gbouras13/ABACBS2024-GPU-workshop-demo-plm-training-slurm-setonix

(Masked) Language Models 101

1. **Split your data** into “training” and ”evaluation” sets
2. “**Tokenise**” your input data into small chunks, make them numbers, pad them to a consistent length, batch them (i.e. make a tensor)
3. Hide some tokens (“**masking**”)
4. Multiply these numbers with weights (i.e. another tensors) to get **representations**
5. Multiply these representations with more weights (i.e. other tensors) to **predict** the masked input
6. Measure difference between your prediction & actual output (“**loss**”)
7. **Backpropagate** the loss into your model to **optimise** the weights
8. **Repeat** on more data

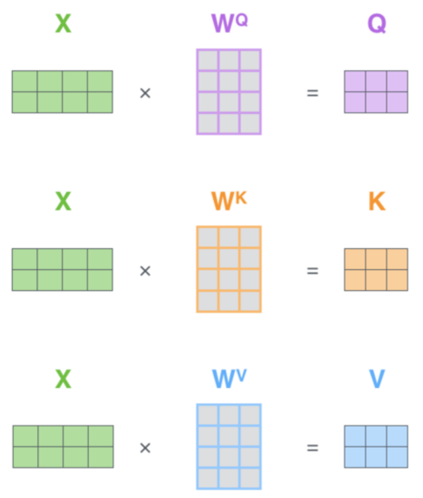
Protein Language Models (pLMs)

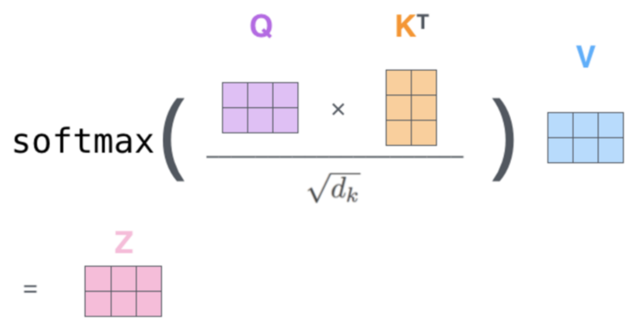
- Tokens – individual amino acids
- Loss – predicting randomly masked amino acids
- Dataset – millions of proteins –> hold a few thousand out for evaluation
- Representations – "Embeddings" – contain evolutionary, structural and functional information about proteins
- Famous models:
  - ESM-2 (Lin et al 2023 *Science*)
  - ProtT5 (Elnaggar et al 2021 *IEEE Trans Pattern Anal Mach Intell)*
  - ProstT5 (Heinzinger et al 2023 *bioRxiv*)
  - ProtBERT (Brandes et al 2022 *Bioinformatics*)

M K T A Y I A K

20 15 11 5 19 12 5 15

20 32 11 32 19 12 5 15





20 15 11 14 19 12 5 15

Loss back propagation

Tokenisation

Masking (32)

Attention mechanism (i.e. lots of tensor multiplication)

Prediction & Loss

Images taken from http://jalammar.github.io/illustrated-transformer/

After Using Your GPUs For a While

- After the loss flattens off “enough” for “a while", you are “done”
- Use your trained embeddings to predict or understanding things about your protein(s)

PyTorch

- Nice framework for implementing tensor multiplication
- Almost everything you need for (protein) language model training is implemented in PyTorch functions “out of the box”
- Hides the low-level GPU code from you
  - Mostly well optimized
- PyTorch DDP – makes distributing PyTorch across multiple GPUs easy

Scaling Language Model Training

- For each step, a copy of model is distributed to each GPU with a (different) batch of data
- During training, each GPU computes the associated loss for its batch
- After each step, loss from all GPUs are pooled and averaged via inter-GPU communication, which then updates the parameters across all GPU nodes as it is back-propagated through the model

Some Practical Things

- "batch size" – number of sequences per batch
- Maximise batch size to use all GPU VRAM
    - `nvidia-smi` or `rocm-smi` or wait for crashes
- Save intermediate models often ("checkpointing")
- "Rank" = GPU number for all GPUs
- "Local rank" = GPU number on your GPU node
- "World size" = Total number of GPUs
- "epoch" – 1 total pass through all training data

Live Demo: ESM2-Finetuning

Other Comments + Conclusions

- Wrapping everything in containers is ideal on HPC
- Print out your tensors in testing and debugging
- Do overfitting experiments on small datasets before scaling
- Often, data-loading (CPU) or filesystem limitations need more optimisation/thought than GPU training code
- Popular frameworks (PyTorch DDP) make it easy to run multi-GPU jobs
- ROCm (AMD) support is improving for popular frameworks
- Training big models on lots of GPUs can be unstable and tricky – go with the flow and do your best