LoRA with Granite 3.3

Granite 3.3

- Announcement
- Huggingface
  - granite-3.3-8b-instruct
    - Works with: **M4 Max Pro 128GB**
    - Over 40GB of memory should be able to use this.
  - granite-3.3-2b-instruct — Use for lower spec machines
- Ollama — Not for LoRA. Just testing locally

What is LoRA?

- Low Risk Adaptation
- Allows you to fine tune LLLMs
  - Locks the overall model
  - Uses small trainable layers — Put into the attention layer
  - Allows to tweak easier.
  - You can swap out different LoRAs
  - Mitigates model collapse — It can still happen if your data is bad.
- Don't have to retrain the whole LLM
  - Cheaper
  - Less resources required
  - Works on laptop!

Dataset

- Use watsonx Assistant Manual
  - Used to test InstructLab
  - Granite doesn't know the topic well enough.
- Docling
  - IBM product (open source)
    - Based off Watson Document Understanding (WDU)
    - WDU has better OCR and understand flows of document.
  - Easy to setup and run.
  - Took ages to process PDF!
    - Probably because of 888 pages with tables/images.
    - If I customised the pipeline it might run faster — Oh well
- Creating JSON list file
  - Create markdown using docling
  - Convert the whole markdown to tokens
  - Chunk tokens to 512 blocks with 50 overlap
  - Convert tokens back to text
  - Save as: `{ "text": " ... " }`

Training LoRA

- **Setup**
  - AutoTokenizer.from_pretrained(model_id) — Handles converting text to tokens and back again.
  - tokenizer.pad_token — This is used to make all inputs the same length
    - Allows batching to work.
  - AutoModelForCausalLM.from_pretrained()
    - Loads Causal model — Predicts next token based on previous tokens
    - device_type
      - "auto" will adjust to system.
      - mps = Apple silicon
    - torch_dtype
      - Determine precision
      - "auto" tries to pick the best
      - eg. float16
    - trust_remote_code — Allows code to execute if model needs it.
- **Lora Config** — config = LoraConfig()
  - r
    - Replaces part of the frozen weights matrix with learnable ones.
    - `W' = W + (A @ B) * (alpha / r)`
      - W = Frozen weights
      - A is a learnable matrix of shape (out_features, r)
      - B is a learnable matrix of shape (r, in_features)
      - A @ B approximates a full (out_features, in_features) matrix.
    - Lower number (4-8)
      - Fewer parameters
      - Faster and less memory
      - May underfit
    - Higher number (64+)
      - More expressive
      - Slower and loads more memory
      - Can overfits if not much data or not diverse enough.
  - lora_alpha
    - Scaling factor
    - How strong the LoRA will impact the overall model.
    - Setting
      - Too high = might overshoot and cause model instability.
      - Too low = not influence the model to have an impact.
  - target_modules
    - Tells loRA where to insert adapters
      - Allows you to get performance with little changes.
      - "where you're allowed to put post-it notes in a textbook"
    - You can get a full list using: **model.named_modules()**
      - There is where most of the models thinking happens.
      - self-attention
        - q_proj = Query
        - k_proj = Key
        - v_proj = Value
        - o_proj = Output projections
      - Feed forward network (FFN)
        - gate_proj
        - up_proj
        - down_proj
  - lora_dropout
    - randomly zeroes out parts of the (A @ B) output during training. — Tries to prevent overfitting.
    - 0.05 (5%) is most common to use.
    - 0.0 if you have lots of data or want full capacity.
    - Higher values if overfitting or noisy behaviours.
  - bias
    - "Do I let the model nudge its outputs slightly, or keep them locked?"
    - "none"
      - If memory constrained — M4 Max for example
      - strict loRA isolation
      - Cleaner adapter
    - "lora_only"
      - To allow small boosts in adaptability.
      - Use if you believe bias shifts are needed to improve the model.
    - "all"
      - Use rarely
      - Close to full fine tuning
  - task_type
    - What are you adapting the model for?
    - TaskType
      - CAUSAL_LM — Left to right generation (text prediction)
      - SEQ_CLS — Sequence classification — eg. Sentiment analysis
      - TOKEN_CLS — Token level classification — eg. Named entity recognition (NER)
      - QA — Question and Answering
      - MT — Machine translation
      - SEQ_2_SEQ_LM — encoder-decoder generation
    - Why?
      - Know which layers to modify.
      - Configures the loss functions
      - Sets up the trainer internally
  - inference_mode
    - Controls if in training or inference mode
    - False
      - training mode — Learn
      - Gradients are calculated.
      - lora_dropout is active.
      - Used for fine tuning.
    - True
      - inference mode — Answer from what you know
      - No gradients
      - dropout is disabled
      - used for running inference on already trained adapters.
  - use_dora
    - Weight-Decomposed Low-Rank Adaptation
    - adjusts the weights direction while keeping same magnitude of original weights. — "Keep the strength of the original model, but steer it in a better direction."
    - Helps mitigate overfitting / instability
    - useful on small datasets — Tends to produce more stable and sometimes higher-quality fine-tuning.
  - use_rslora
    - Rank-Stabilized LoRA
    - Improves regular LoRA when the rank (r) is very low (e.g. 2 or 4).
    - Makes the adapter output more stable and expressive
    - Use when
      - r <= 8 and you want better performance
      - You don't want to increase r
      - You care about stability and generalisation
- **model = get_peft_model(model, config)** — Modifies your base model with the configuration.
- **Dataset**
  - Dataset.from_list(texts) — texts is the list of JSON from earlier
  - Then create tokenized dataset.
- **TrainingArguments**
  - output_dir — where to save checkpoints, logs and final model.
  - per_device_train_batch_size
    - One training example per batch per device
    - Low batch size = minimal memory use, good for M1/M2/M4 chips.
  - num_train_epochs — How many times to loop over all training dataset.
  - save_steps — Save a checkpoint every x training steps.
  - save_total_limit — How many checkpoints to keep
    - Older ones are deleted
  - learning_rate
    - 1e-4
      - 0.0001
      - Default for LoRA
    - How fast to learn
    - Higher number is faster but riskier.
  - fp16 — float16 precision
    - Switch on for CUDA
    - Switch off for Mx Macs
  - bf16
    - Used for M4 macs
    - Similar memory/performance benefits to fp16, but safer numerically.
  - report_to — "none" disables logging.
  - logging_strategy
    - "no" prevents notebook running report.
    - Removing will cause the notebook to run out of memory on large datasets.

Testing LoRA

- Loading a LoRA adapter will overwrite the base model it is applied to. — To test base + LoRA, base has to be loaded twice.
- Not as intelligent as GenAI chat model — Prompt has to be specific.
- It is able to understand watsonx Assistant (wxA) questions without the context of wxA in the question.
- It can still hallucinate — Should greatly improve RAG on the content.
- Base model can't even get close to a correct answer.

Timings

- PDF to chunked markdown
  - 17 minutes
  - Could probably be greatly improved with custom pipeline
  - WDU is much faster regardless!
- Training LoRA model — 1 hour 40 minutes — **FAST!**
- Transfer learning
  - Train all 8 billion parameters
  - 24-48+ hours to run.
  - HW requirements far exceed my laptop. — would need a datacenter