Understanding DoRA

- When we use a pre-trained model (LLaMA, BERT, etc.,), it's already been trained on a massive data to understand general language. But if we want it to do specific things, we'll need to fine-tune it for those tasks
    - Fine-tuning: updating the internal parameters (weights) just a little so that we get better performance on the specific tasks we want
    - We can just use the big model and re-train the weights on specific tasks, but this is still quite expensive
- Every layer in a neural network is basically a function with weights, numbers that the model learns during training, including attention layers
    - Many weight matrices
    - Big models like LLaMA or GPT have billions of weights. Instead of fine-tuning an entire model, we just train a few part of the model —> PEFT (parameter-efficient fine-tuning)
- LoRA (Low-Rank Adaptation) —> "instead of directly modifying a big weight matrix (full fine-tuning), let's add a small low-rank update to it"
    - Model has a trillion parameters? Well, then, you need to store a trillion gradients in full fine-tuning. This is a problem.
    - LoRA was introduced by Microsoft Research in 2021
    - Keep big matrix unchanged and learn two tiny matrices A and B such that W' = W_0 + BA
        - You only train B and A, not W_0. This saves a ton of memory and works surprisingly well
    - Big idea: instead of modifying the trillions of weights, we freeze them, instead adding our own sets of weights. While we are indeed adding more parameters, the GPU still sees the same amount of parameters (after added), but the number of *trainable parameters* is far less.
- Recall definition of rank: number of linearly independent columns in a matrix
    - Insight: if we remove linearly dependent columns we shrink the dimensionality of our matrix without losing information
    - LoRA idea: we can express our delta W, the full-rank fine-tuning weight matrix we want to add to the original W, as a product of two low-rank matrices —> this is referred to as a low-rank decomposition
        - Decompositions are more efficient: we can build a 3x3 matrix (9 numbers) from a 3x1 vector and a 1x3 row vector (6 numbers total).
    - Rank *r* is a hyper parameter that we'll have to choose.
        - Can't be too low —> will reduce dimensionality too much and implicitly delete columns that are actually linearly independent
        - Can't be too high —> no computational advantage
- Initializations:
    - A ~ Gaussian
    - B ~ 0 vector
    - Backprop will figure things out w.r.t. objective
- Why was LoRA better than alternatives?
    - There are plenty of other options, notably *adapter* blocks in LLMs
        - Problem with these is that they're quite sequential, not parallelizable.
- DoRA (Weight-Decomposed Low-Rank Adaptation—> "Let's separate weights into magnitude (how strong) and direction (where to point), then update them differently"
    - Again, take a pre-trained weight matrix W_0

        - Decompose it in magnitude and direction terms
        - Then, keep the direction mostly fixed, but update it with LoRA
        - Train the magnitude as a separate small vector
- Idea: you can decompose original weight matrix into direction and magnitude
    - W is dxd and W = m * (V) / ||v|| = ||w_c|| * w / (||w_c||)
    - The paper defines two formulas for how magnitude and direction change in a LoRA update (W')
        - delta M: average of fine_tuned magnitude - original magnitude
        - delta D: 1 - cosine similarity
- What did they notice with their approach?
    - Full fine-tuning
        - **When you plot a fine-tuned model w.r.t. delta M and delta D, you notice that an increasing change in direction leads to a decreasing magnitude (inverse relation)**
        - **High variance across the board**
    - LoRA
        - Opposite happens: low variance and magnitude and direction are proportional
        - **This is bad, because LoRA is supposed to be mimicking fine-tuning!! It doesn't in this sense**
    - DoRA claim:
        - We can get better performance if we can mimic the full fine-tuning's high variance and inverse relation
        - Indeed, that's what their results yielded
- DoRA approach revisited
    - They use the above decoupling/decomposition and train magnitude and direction independently from each other.
    - Train direction using LoRA (normally) and magnitude as a vector
        - Just adds d more parameters to train (*m*)
            - They suggest treating || V + deltaV||_c as a constant for simpler gradients
    - They actually prove that the larger directional change then the loss w.r.t. magnitude is smaller than vice versa. They prove the inverse relationship
- Results demonstrate a clear improvement on LoRA (in general) over LLaMA B training examples. In some cases, by quite a lot!
    - Introduction of parameters is very small; improvement in performance outweighs the cost