Department of Computing

Imperial College of Science, Technology and Medicine

Deep learning for GPU poor

Author: Anton Zhitomirsky

Contents

1 Calculating statistics of a model

1.1 Foundation model parameter count

Commonly when you look online for models, you will see their parameter counts. *Transformers are typically described by the #parameters* which impacts the computational requirements to store and run these models.

1.2 Floating Point System

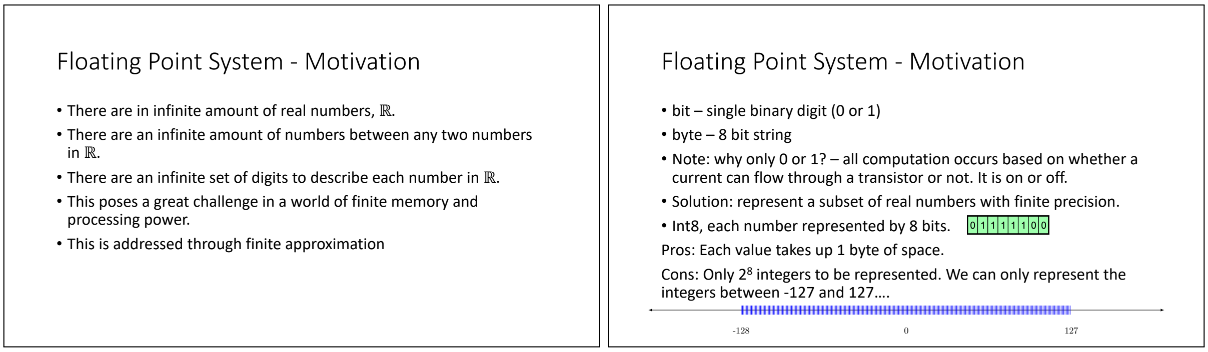
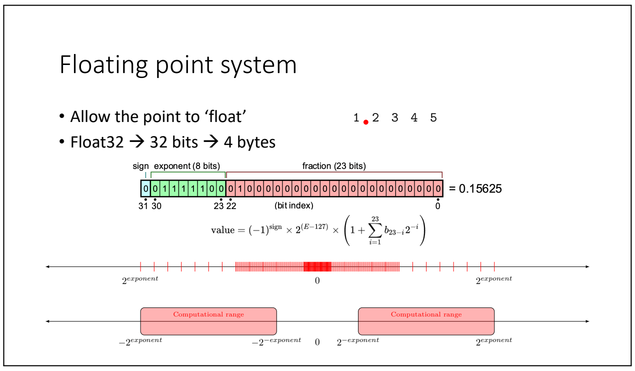
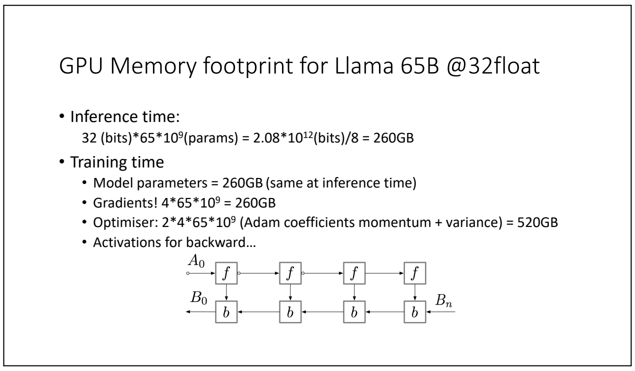


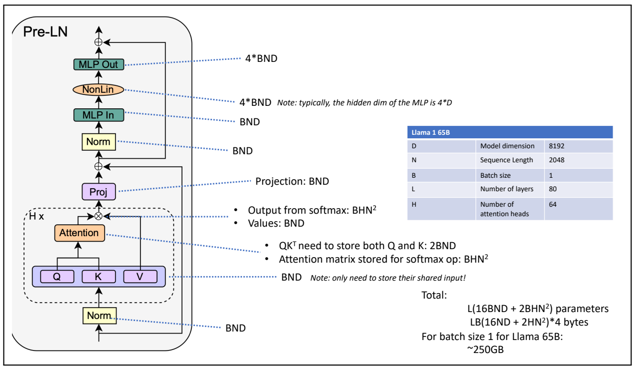
Figure 1: We store data in computers in bits with transistors. The more bits, the more precision.



- **Floating point** is a way to represent numbers in a computer where the decimal point can float.
- A certain section is allocated to represent the precision (actual number) the exponent and the sign.
- still has the problem of underflow or overflow.

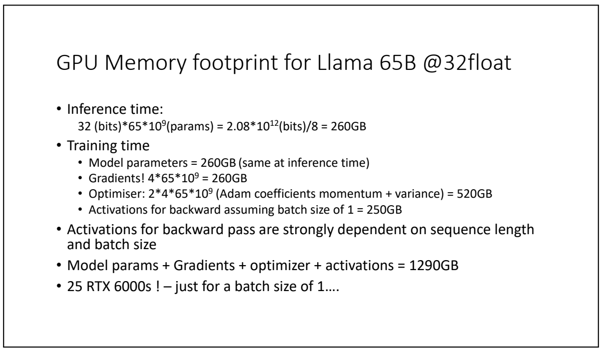


- At inference time, just to store the data it is already 260GB
- At training time, we also require gradients and optimiser values (momentum, etc) which can be 10x the size of the model and the activations.

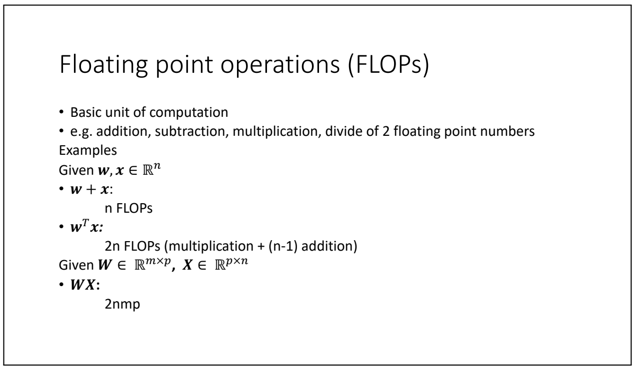


So if we pass through the network, we need to store the inputs to every single layer that theres a change in the computational graph. Here:

- The first is the batch times the number of tokens or the number of patches times by that dimensionality.
- For the query key value, they all get the same value.
- The attention matrix requires query with key-values
- We also need to store the n^2 complexity of the attention matrix.
- Also we store the result of the soft max which is the batch times the number of heads times the sequence dimension.
- With multi-head attention we usually have another layer that learns how to combine these values together.
- All in all, we have 250GB for a batch size of 1.

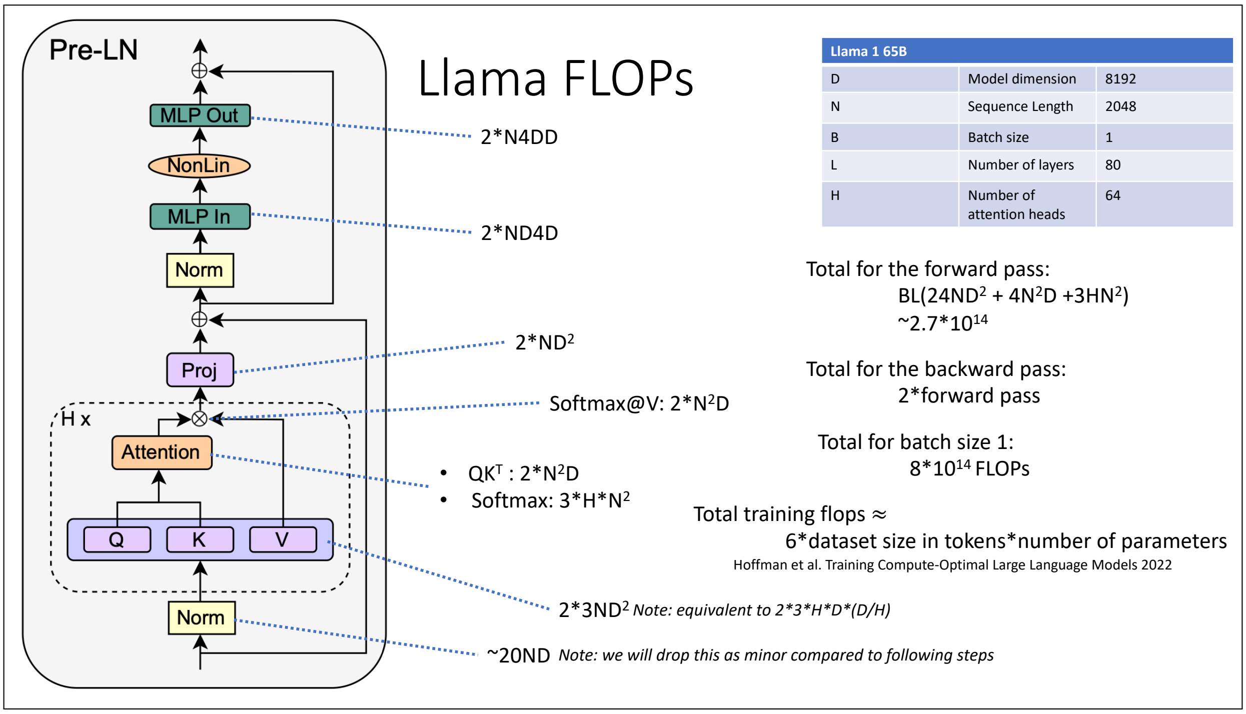


1.3 Floating Point Operations (FLOPs)



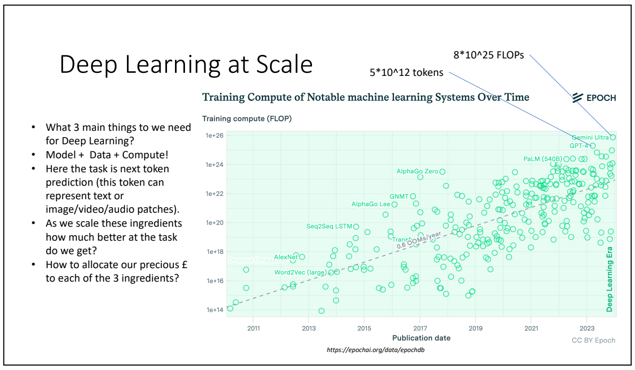
Computational requirements:

- given two vectors, if we add this will reuqire n operations.
- For the dot products, there is $2n$ flops.
- For the matrix multiplication, there is $2nmp$ multiplications

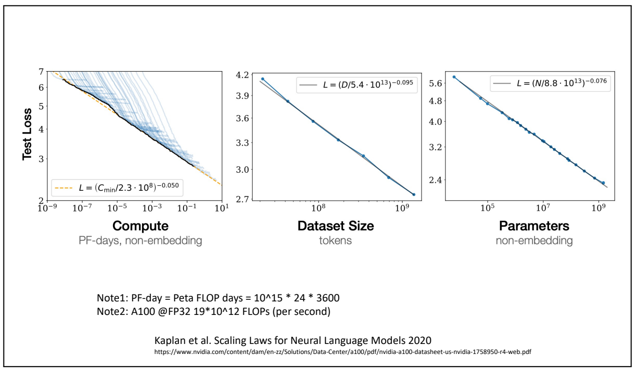


1. the layer norm is roughly 20 operations for every feature and token
2. for the query key values we have it 3 times, and 2 becuase its a dot product.
3. The softmax: for every single query we dot product it for every single value.
4. projectino is same as query key values (but only 1)

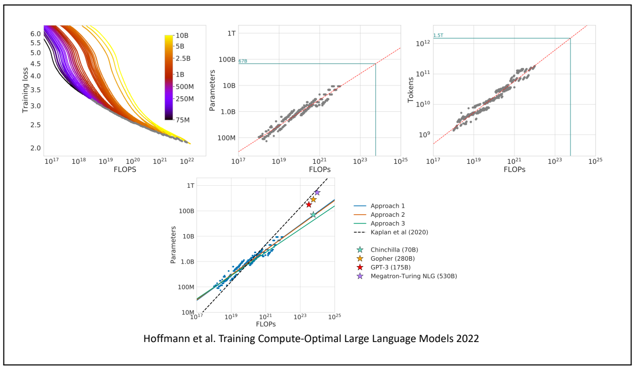
2 Deep Learning At Scale



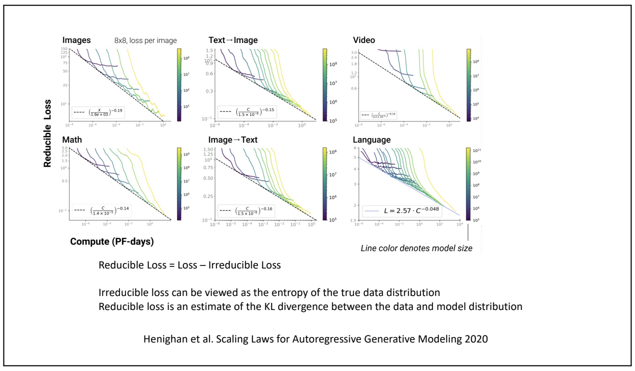
- For deeplearning we need model data and compute.
- At scale, you can't train the model many times for hyper-parmaeter tuning. A single model may take many dollars to train just once.



"When they have limitless compute and limitless datasets, they saw this really nice power law trend where it sticks very closely to the trend. This says that generally you can predict quite well, if you have limited compute and limited dataset size what loss you would get at the end." The values here were proven to be wrong because the smaller models were not fully optimized; they used the same learning rate schedule as they did for the larger models. Therefore, the line was too steep.



"openAI published finidngs about colour representing the size of the model, and they fit for given flops and compute budgets which model gets the lowest loss." Here, we see that chinchilla is favoured since it prefers the number of data points over the model size.



Here we see that 'deep learning has bene sovled'. "This is showing that this happens in all modalities; these losses can just keep going until we get to super human performance."

Definition 2.1 (Reducible Loss)**.** It is the actual loss - the irreductible loss, where irreducible loss is viewed as the true entropy of the data i.e. if I had a perfect fit of the data, what would be my loss even if I had a perfect fit?
It can be viewed as the KL divergence between the mainfold of the dataspace and the manafold of the learnt model.

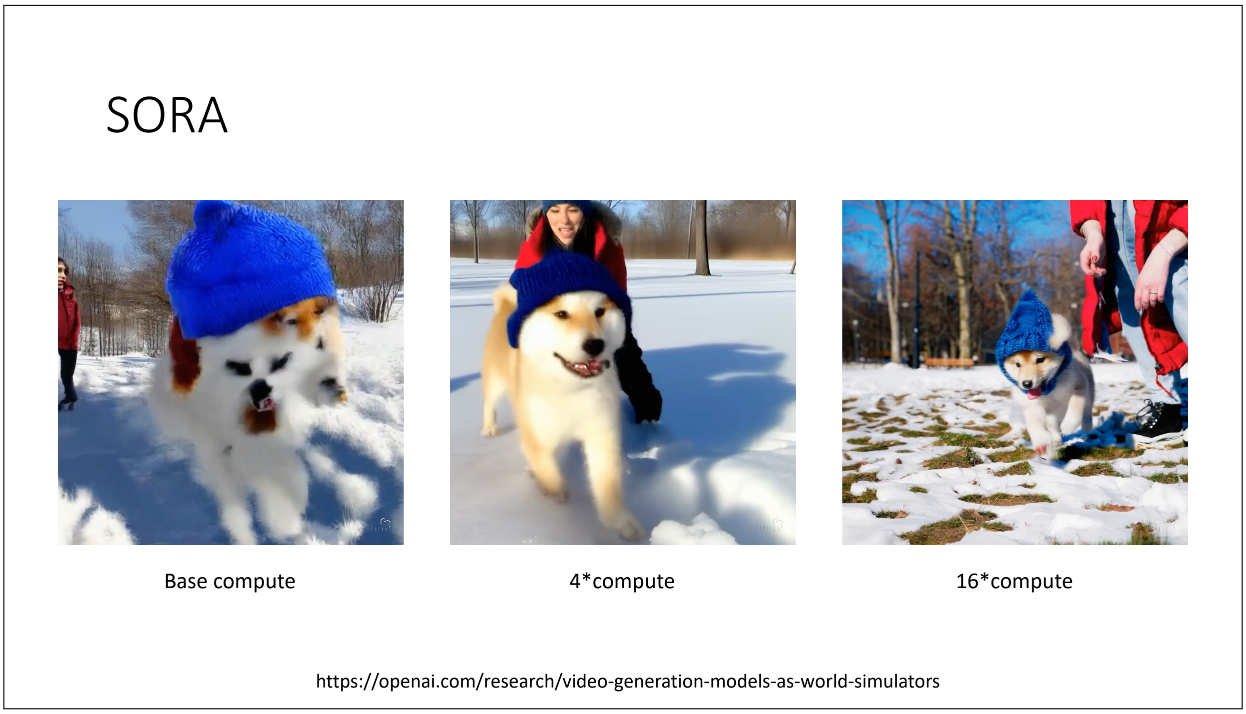
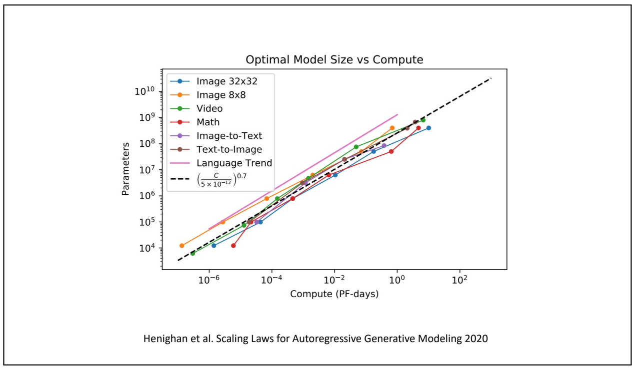
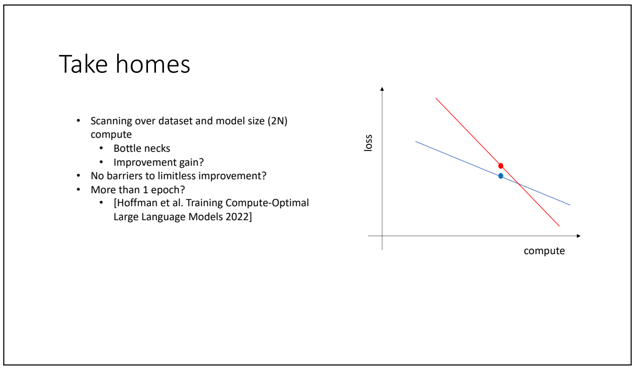


Figure 2: SORA AI just used more compute!



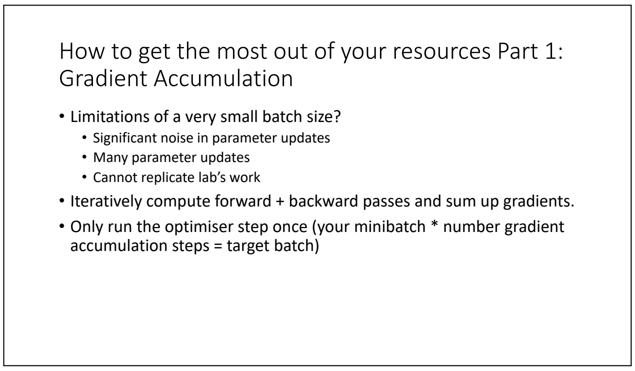
"When we plot all modalities the number of compute days vs the number of parameters for the optimal setting. There are correlation between what it is like to compute potimal size model for videa and for language model"



- if blue is state of the art and red is my implementation,
- then we can do a scaling analysis and conclude that since the curve is steeper then it may perform better at high levels of compute.

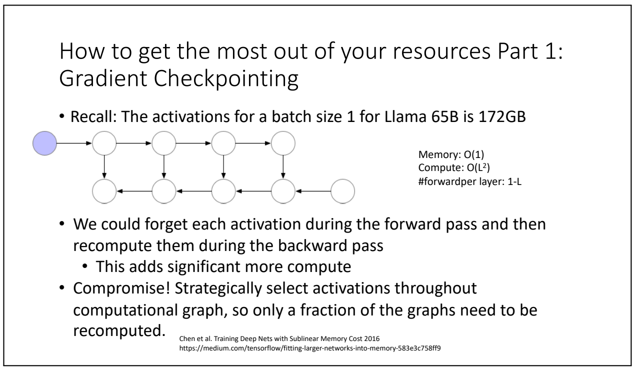
3 Methods for reducing computational requirements

3.1 Gradient Accumulation



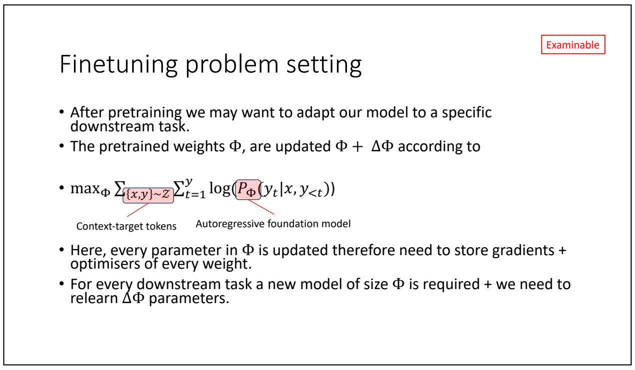
- As we increase the batch size our computation requirements increase
- If we use 1 batch, this is just Stochastic Gradient Descent so you're approximating the true gradient with one sample; your loss will be jumping around wihtout convergence
- *accumulate the gradients without updating the optimizer, then when the mini-batch size times the number of cumulative gradient accumulation steps equals the target batch then we do the step in the loss surface.*
- Useful if you want a larger batch size

3.2 Gradient Checkpointing



- idea is to initially throw out the activation in the forward pass after we've done it, but in the backpropogation step we redo the entire forward computation step.
- This is computationally expensive, but the memory requirements stay low; it doesn't scale with the number of layers.
- The forward pass is much faster than the backwards pass
- Therefore, we select checkpoints from which we continue the forward pass.

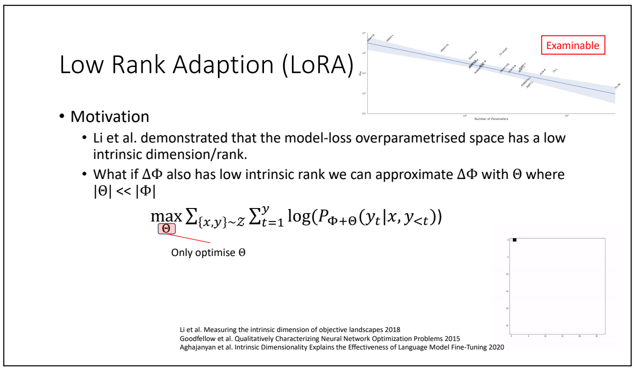
4 Finetuning problem setting



(e) "For every downstream task" we need to fine tune (lots of flops of updating) and store each separate model for each downstream task. This is computationally burdesome

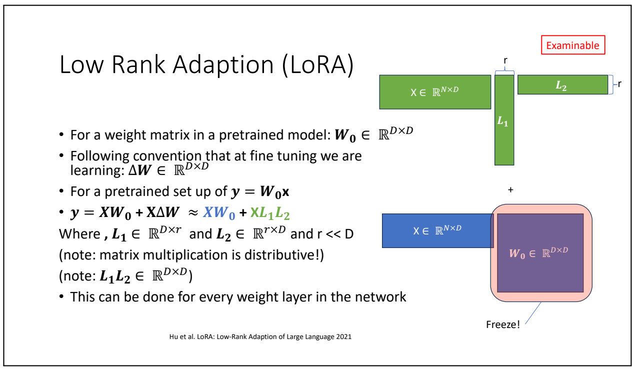
- We're talking about autoregressive models; we have a pre-trained auto-regression model, and for our fine tuning datset, we select some context target pairs (the tokens before the token you're trying to predict).
- We're trying to min/max this over each of the pre-trained parameters.
- This requires us to store teh gradients and optimisations for every single one of these parameteters.
- We also require more leaning signal to provide enough signal to update every single one of these parmaeters (scalability issue as shown above with scalling laws).

4.1 Low Rank Adaptation (LoRA)

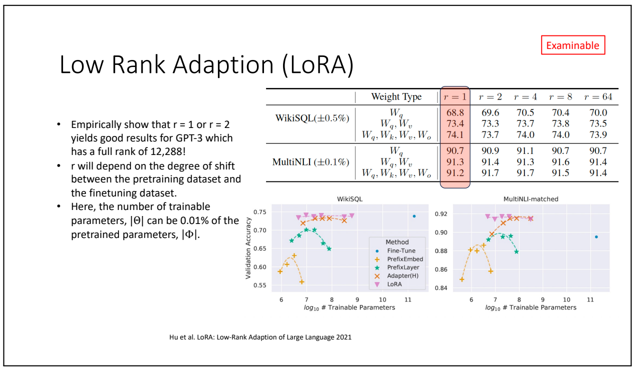


(f) Diagram: "The intrinsic low rank of the over-parameterised space decreases as the number of parameters of our pre-trained model increases; the larger the model, the simpler the loss parameter space is".

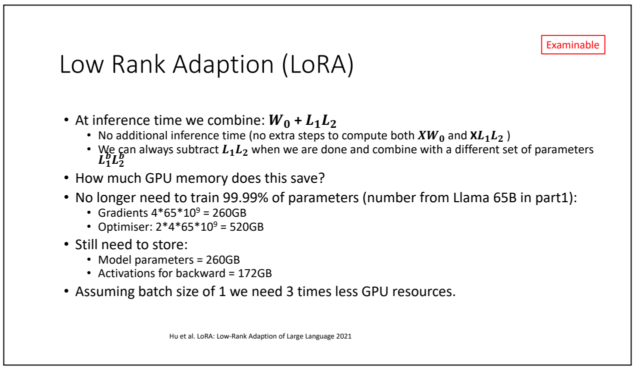
- During fine-tuning, researchers constrained the number of parameters that are available for fine-tuning and saw that the training performance actually didn't drop.
- If we increase the number of parameters, the likelihood of getting stuck in a local minima 'saddle' increases.
- The authors: What happes if $\Delta\Phi$'s rank is intrinsicly low? We can instead update Θ that is significantly less than Φ.
- TLDR: update only a subset of parameters.



- If before, we learn that the pre-trained projection and also the finetuned projection Δ can we decompose ΔW into a low rank version.
- in blue: the original pre-trained model where we take the input and multiply it by the weight matrix. We freeze these, so we don't learn anything about them.
- In green, in parallel we have an approximation of the weight matrix that is low rank. We only learn this.



- intuitively it works because we project $L_1 \times L_2$ into a $D \times D$ matrix.
- they found that the low rank approximation of the weight matrix was able to capture the majority of the information in the original weight matrix even at $r = 1$.
- This is susceptible to initialisation bias. They set L_2 to be 0s; they don't add any new information in the first couple of iterations – its only the pre-trained model that is doing the work that is itsself forzen. Then the model slowly introduces the new information.

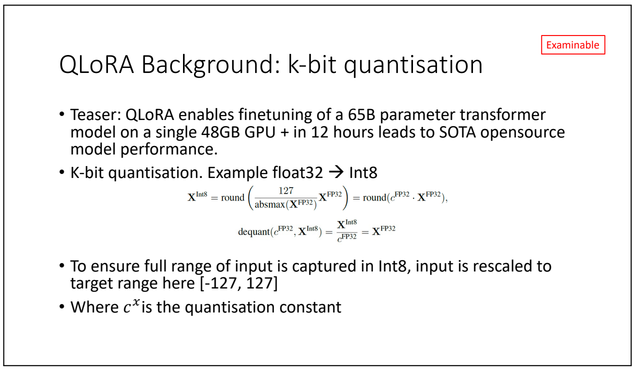


- At inference time we have a lot of flops from multiplying the matrix together. What they do is they fuse the kernels togeher.
- We can then swap in the very small memory footprints in easily.
- We can then apply some methods from Gradient Accumulation/Checkpointing.

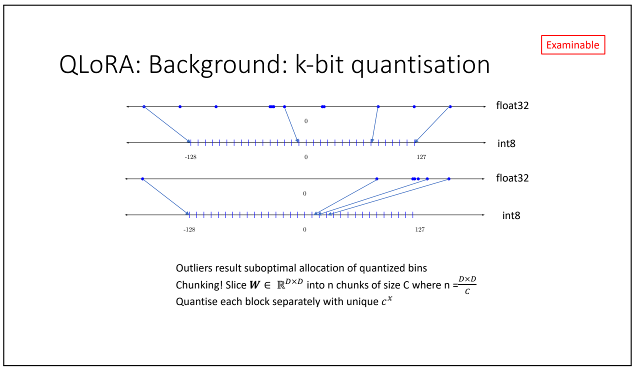
Therefore the big drawback with LORA is that we still need to store the W_0 matrix.

4.2 QLoRA

4.2.1 K-Bit Quantisation

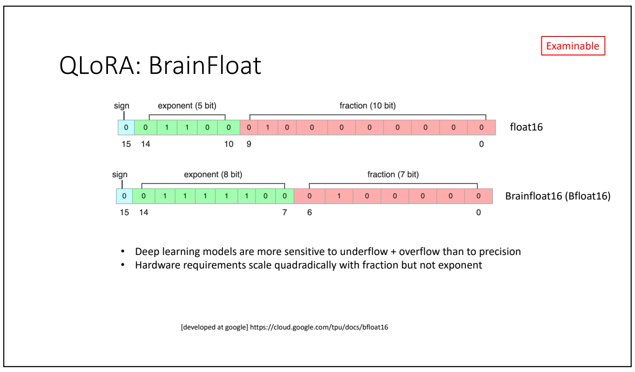


- K-bit quantisation: this is going from a data type of higher precision or a higher bit rate to a lower bit rate
- we accomplish this by scaling by the absolute max of the input data



- We have an example with datapoints in float in blue. We then qunatise it down to int8
- We scale the absolute max to the lowest possible value and scale everything else equally by using the quantisation constant.
- This is a lossy process. We can't go back to the original data, but we can go back to the original data type.
- This has the setback that if we have outliers, we scale things to a bin, and the represntation becomes very sparse.
- To solve this issue you perform chunks, into small pieces so that the chances of ther being an outlier is very small.
- Each chunk therefore has a unique qunatisation constant.

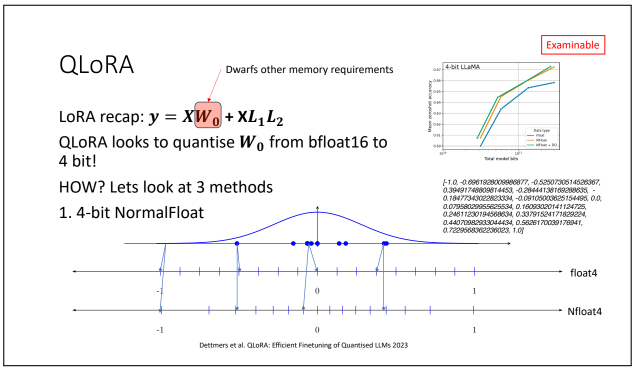
4.2.2 BrainFloat



- This type favours the exponent significantly.
- We found that the models were more robust to the actual precision of the bodies than they were to a thing called under or overflow. (where gradients are so small that they become 0)
- "models performed significantly better when we worried more about underflow than overflow".

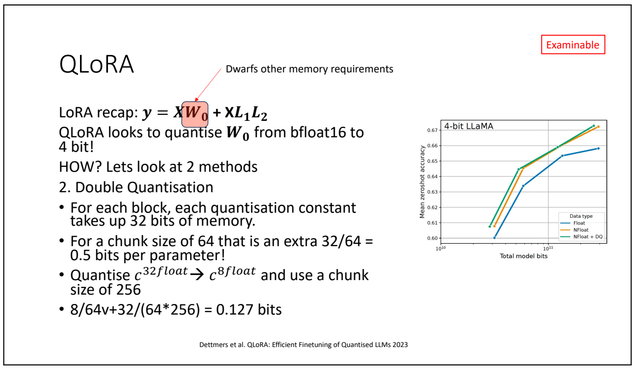
4.3 Quantising W_0 from bfloat16 to 4 bit

4.3.1 4-bit NormalFloat



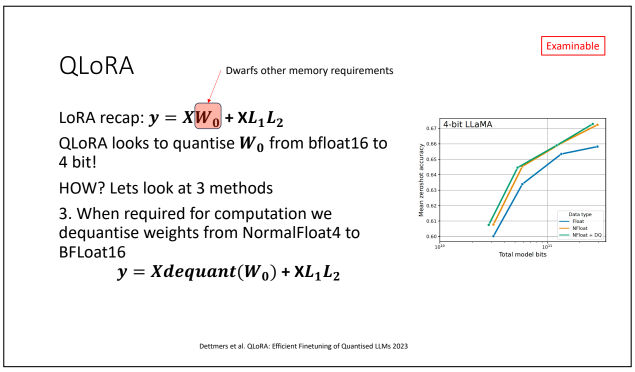
- A big drawback with LORA is that we still need to store the W_0 matrix.
- We can use BrainFloat to reduce the memory footprint of the model to 4 bits!
- Here, we rely on the idea that most data in deep learning is a normal distribution.
- Majority of values map to the same central point normally
- Therefore, they introduce a new type where values are normally distributed in the new type also. The idea is that they are wider at the edges
- The numbers are all the numbers that an Nfloat4 can represent, and in the graph we see Nfloat performs better.

4.3.2 Double Quantisation

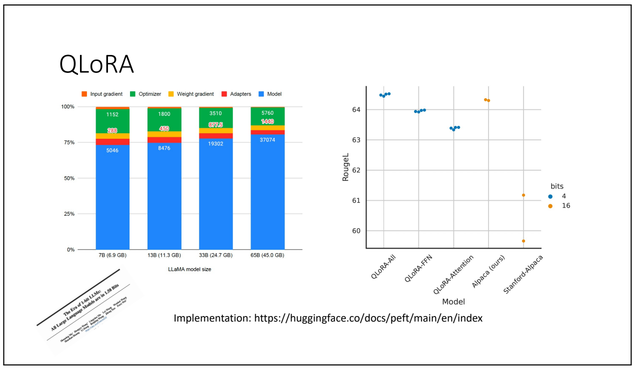


- In K-bit quantisation, we store quantisation constants that is stored in 32 bit precision.
- If we quantise 65 billion parameters, then we have a lot of qunatisation constants.
- Therefore, we quantise the quantisation constant.
- Quantisation and de-qunatisation is a cheap operation and fits nicely into consumer-grade hardware.

4.3.3 Dequnatisation



- The noraml float is a static memory type.
- When we train the model, the vast majority of the pre-trained model is stored in 4 bit.
- When we matrix multiply we de-quantise it into 16 bit brainfloat space and perform the matrix mulitplicaiton.
- (L_1 and L_2 are still in 16 bit representation)



- PEFT good for project.