CS336 Assignment 3 Writeup

Marcel Rød
Stanford University
roed@stanford.edu

Parts of this writeup were discussed with Rishabh Ranjan.

2. Scaling Laws Review

2.1. Scaling Laws from IsoFLOPs profiles

2.1.1. `(chinchilla_isoflops)`

To validate the fit, I first produce the IsoFLOPs optimal loss curves for each FLOPs budget.

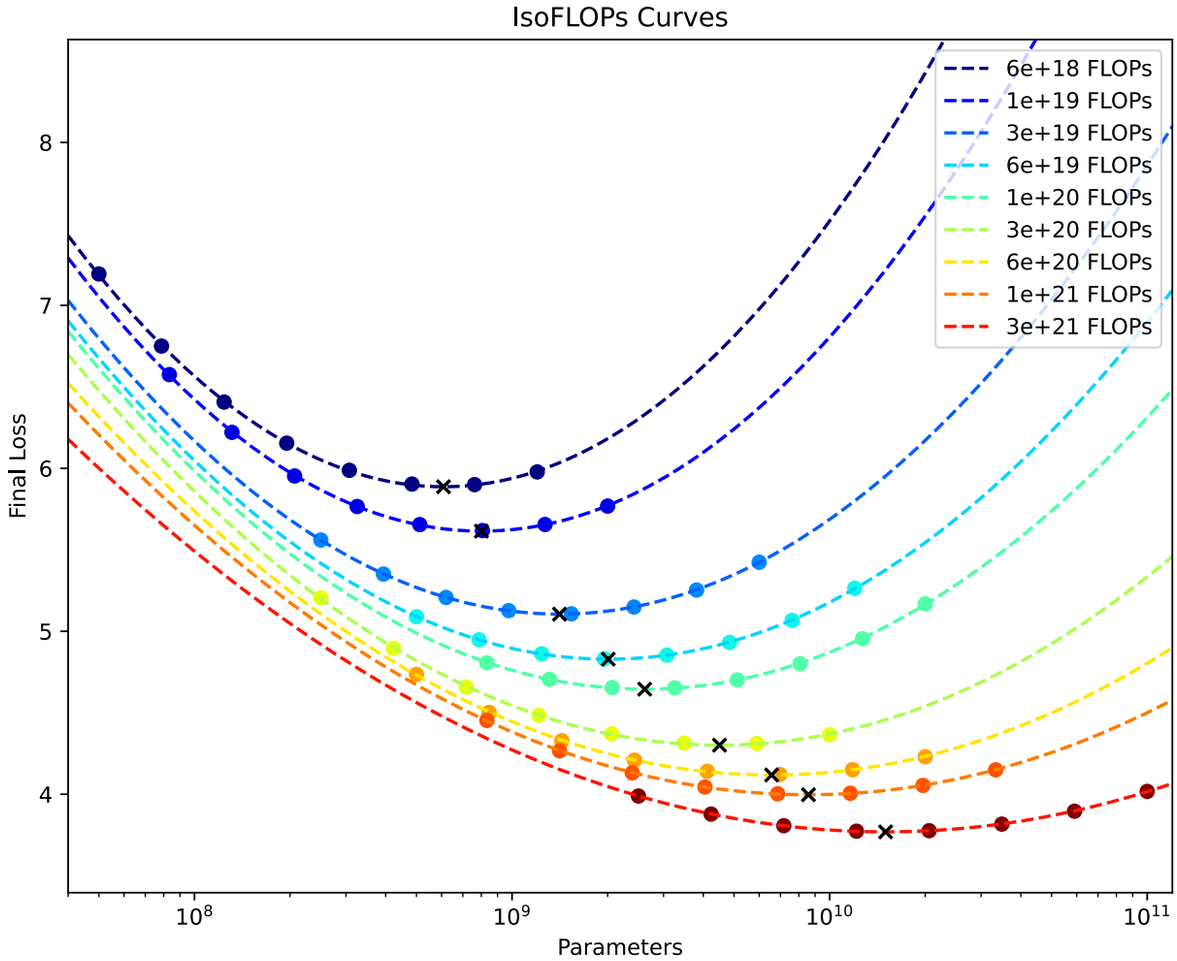

Figure 1: The IsoFLOPs for each FLOPs budget generated from the synthetic data. The black crosses indicate the optimal parameter value given the interpolation. Interestingly, these seem to lie on a straight line in the log-log plot.

1. Plotting the results we get for the best parameters for the given IsoFLOPs profiles, we see that these points lie on a line in log-log space. We fit a line to these points, where the line has a slope of roughly 0.5, indicating that the optimal number of parameters scales roughly with the square root of the FLOPs budget. The plot shows the pairs of points $\langle C_i, N_{\text{opt}(C_i)} \rangle$ as black points. The extrapolation of this line to 10^{23} and 10^{24} FLOPs nets us an estimate of 90.6B and 295B parameters, respectively.

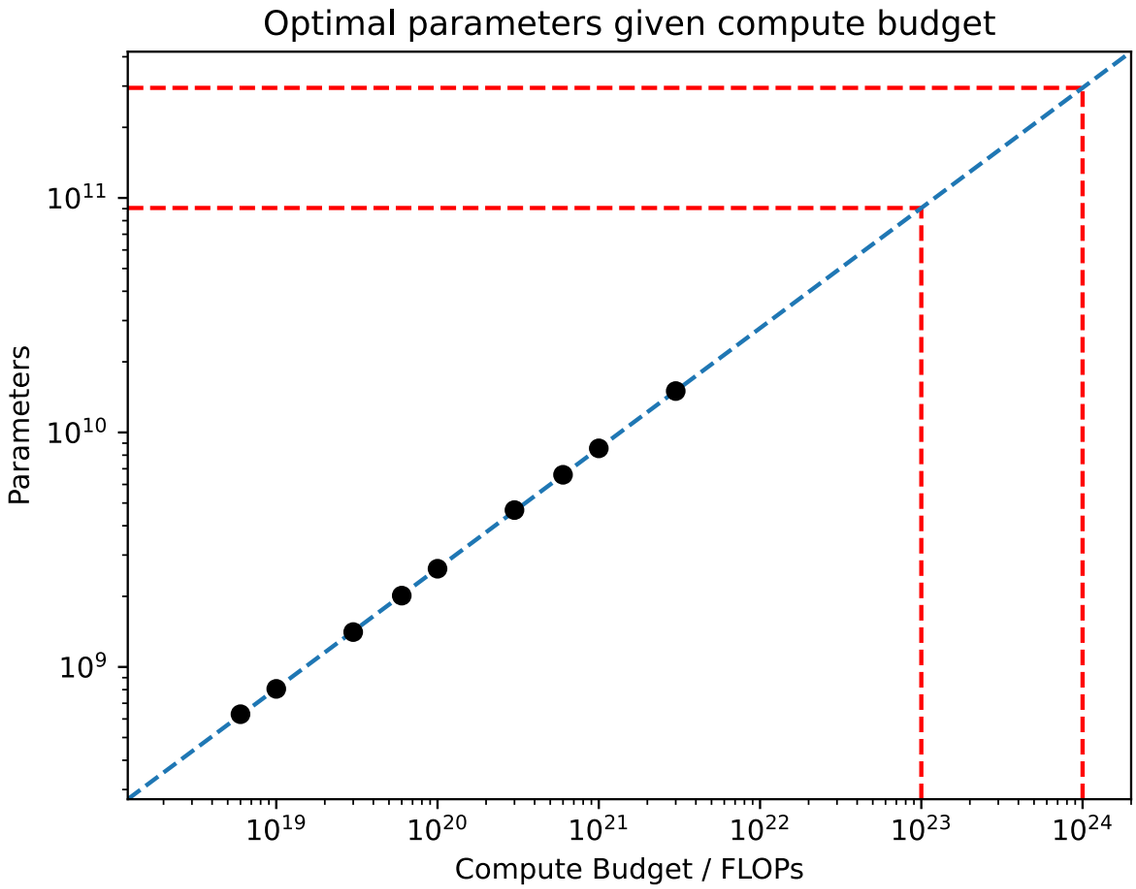

Figure 2: The black points indicate the points measured from the IsoFLOPs curves from before, and the blue line is the linear fit to these points. The red lines indicate the extrapolation to 10^{23} and 10^{24} FLOPs.

2. We can translate the results from above to the number of tokens by using the direct relationship given in the assignment. Performing this transformation, we get the points and the line in the plot below. The extrapolated number of tokens for 10^{23} and 10^{24} FLOPs are 184B and 565B, respectively.

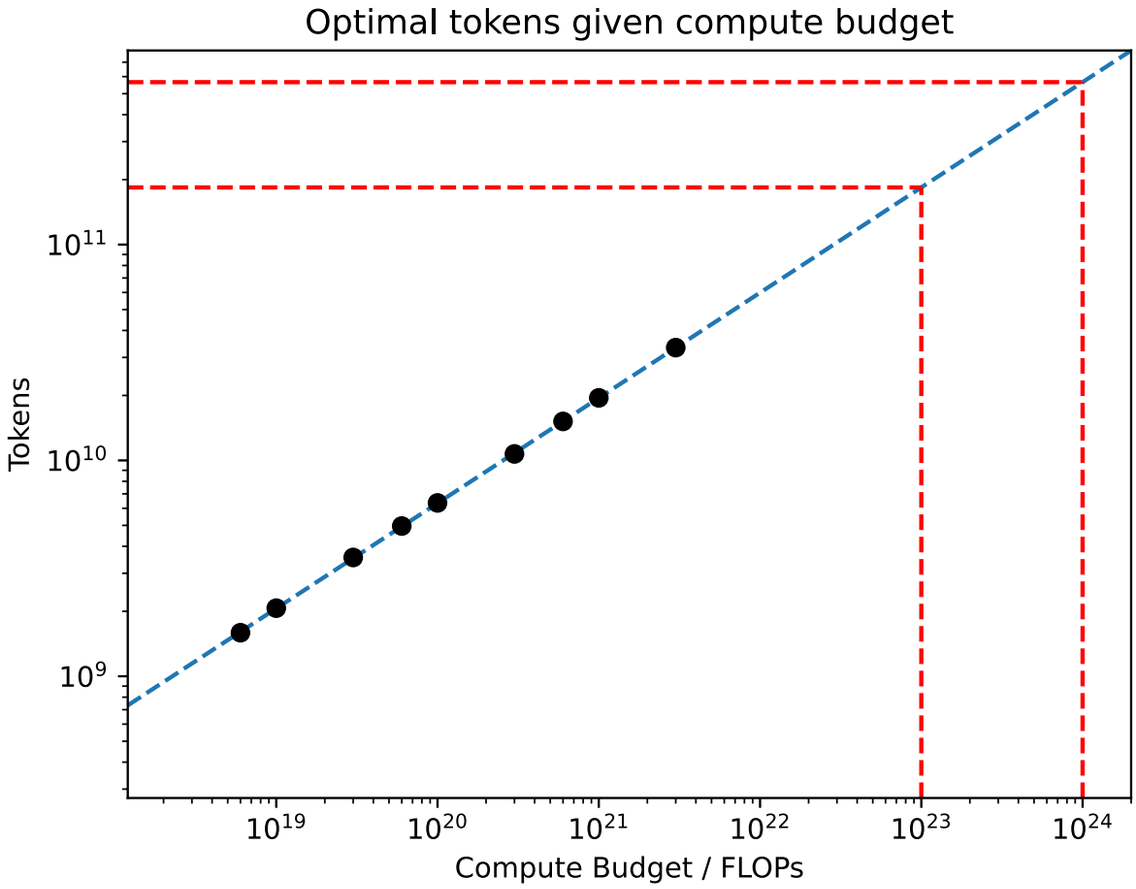

Figure 3: The same setup as in Figure 2, but transforming the points and the line to instead represent the number of tokens.

3. Constructing Scaling Laws

3.1.1. `(scaling_laws)`

The general approach for finding the best hyperparameters for the large scaling run will include the following sources:

1. Papers describing previous hyperparameter searches for similar models.
2. Papers describing methods for determining the scaling laws for models, using them to predict the best balance between parameter count and dataset size for a given compute budget.
3. Intuition gathered from previous assignments and my own experience with training similar models to the one available through the API endpoint.

For the first part of the decision process we especially refer to [1], which describes a lot of experimental results for scaling laws and shows ablations for a majority of the relevant hyperparameters for transformer models. Key findings that greatly narrow our search space are that we want the head dimension of our model to be around 128 for d_{model} greater than 512, and around 64 for d_{model} of 256 or smaller. This result can be used to determine the number of heads in the model from a smaller range given d_{model}. [1] also hints at something we know from playing with the models in previous assignments, which is that for a given FLOPs budget, the batch size should be as small as possible to increase the number of steps done in a training run. In fact, a very small batch size can be further beneficial in that it works as a method of regularization, and can help the model generalize better. One might be worried about the stability of this approach, but with the right

learning rate scheduling, optimizer and gradient clipping, this is empirically not much of a problem. This is a somewhat unrealistic result, however, since there is overhead that scales with the number of steps in a real training run, for instance the overhead of communication in a distributed setting on a GPU cluster. In this setting one might instead want to use a larger (effective) batch size to amortize the overhead of communication, and hope to get individual steps of higher quality than in the low batch-size case. The conclusion for the experiment is that we should fix the choice of batch size to the minimum of 128, since in the IsoFLOPs case there is no reason to increase the batch size above this value. Further, the learning rate we are looking to use in the final run parameters will be Targetting a small batch size also means that we should aim for a smaller learning rate, ideally one smaller than $10^{-}4$. We thus stick to $10^{-}4$ since it is the smallest available learning rate in the API. Using all these heuristics we can define our hyperparameters uniquely given a model parameter count, and we will further tweak the hyperparameters in local minima to find possibly better models. From [1], we also consider a mostly fixed aspect ratio so that we can narrow down our search space for sampling of parameters and build sensible IsoFLOPs curves. The paper shares that an aspect ratio from 32 to 128 is good for the parameter count range we are working with.

We now need to determine how to get information about the scaling laws for our particular model. Recall that the total compute budget is 2×10^{18}. There are many different approaches to this, but I find the IsoFLOPs fitting method from [2] to be the most intuitive and interpretable method.

To perform this method, we need to decide on a set of FLOPs budgets to use for sampling models to find the isoflop curves for. Since this planning can be done "online" as the points come in, we decide to do samples based on predictions with partial data.

The following steps reference points that can be seen in Figure 4:

- To start, we sample a large range of parameters at the lowest possible FLOPs budget at 10^{13}, but we notice that there is little to no difference in the loss values for these models, meaning they give us very little information. The plan was to use the lower FLOPs budgets to figure out how to better sample the higher FLOPs budgets models, but since the models at these low scales are quite noisy, they don't give us much useful information, and have served as outliers for some of the later analyses.
- Sampling a few points for 3×10^{14} and 10^{15} shows us some more variation in the model values, and we can begin to see optimal parameter counts for these budgets.
- Moving on to 6×10^{15}, we see that the optimal parameter count looks to be around 10^{6} parameters.
- Following the line inferred by the sketched optimal parameter counts, we sample around the region of 10^{7} parameters for 10^{17} FLOPs, which shows us that a parameter count of around 6×10^{7} works well at this scale.

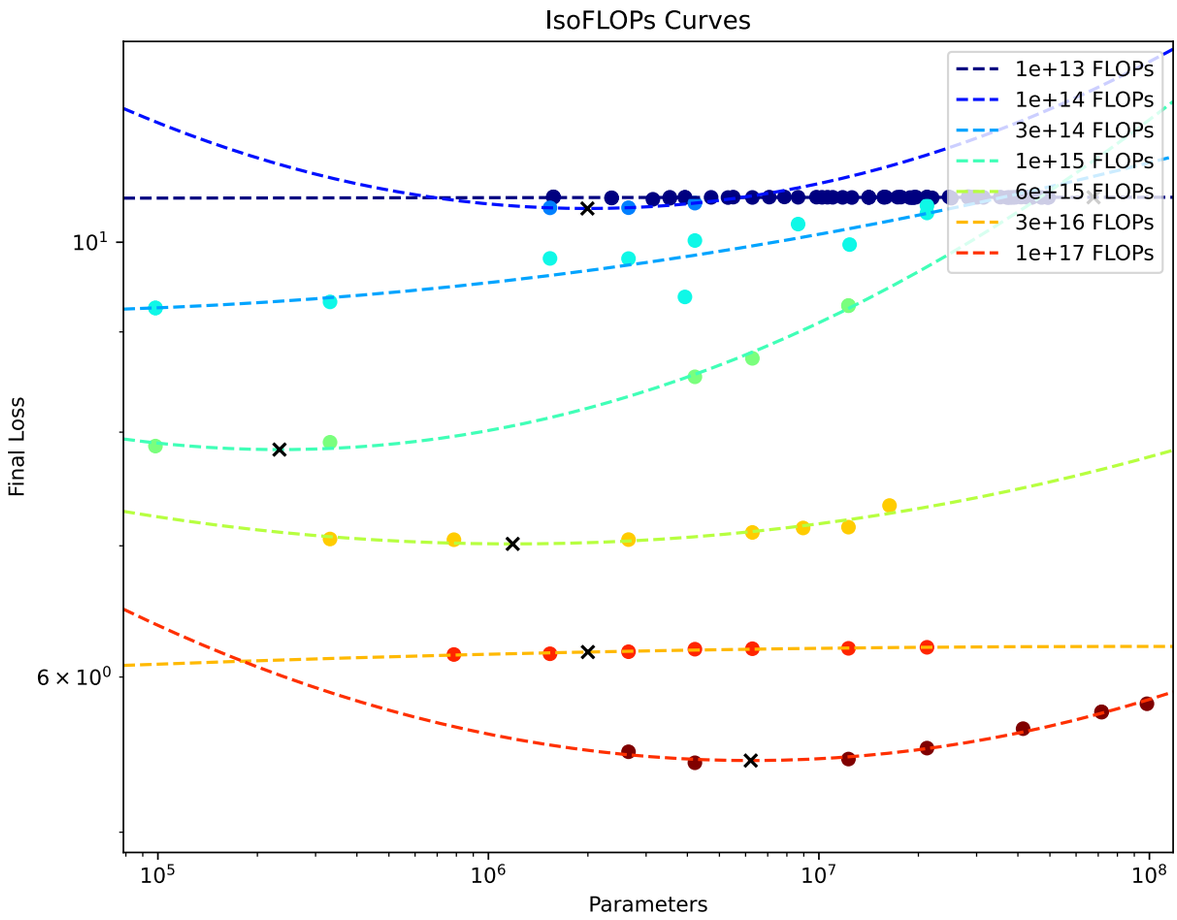

Figure 4: The IsoFLOPs curves generated by querying the API using my methodology. The curves have been fit using parabolas, and the optimal points are indicated by black crosses (adjusted for 3e16).

In hindsight, if I could make these choices over again, I would do many more samples at the lower FLOPs budgets to get an even better idea of the space before moving on to the higher FLOPs budgets. I think the odd shapes and outlier points I got at lower compute budgets could have been mitigated by trying more strategies that are specifically suited to the lower compute budgets, but I'm not sure how much this would've helped for extrapolation to large compute budgets.

We now use the optimal points found in the black crosses seen in Figure 4 to fit a few more curves based on aggregations of our data, resulting in Figure 5 and Figure 6, which show predictions from the scaling laws we've applied.

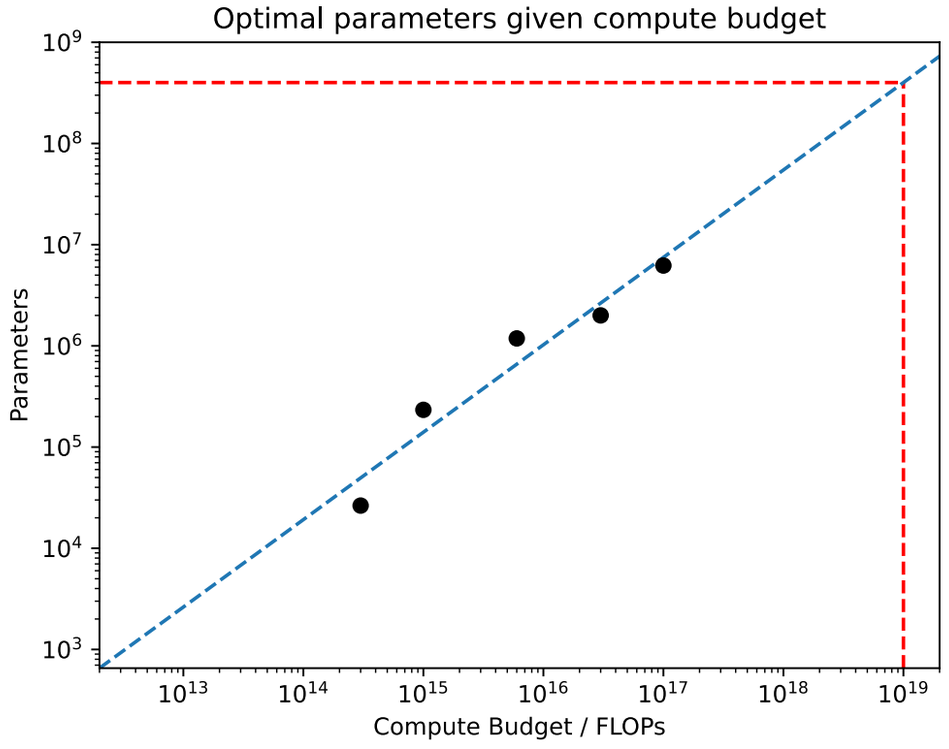

Figure 5: The black points indicate the points measured from the IsoFLOPs curves from before, and the blue line is the linear fit to these points. The red line shows extrapolation to 10^{19} FLOPs.

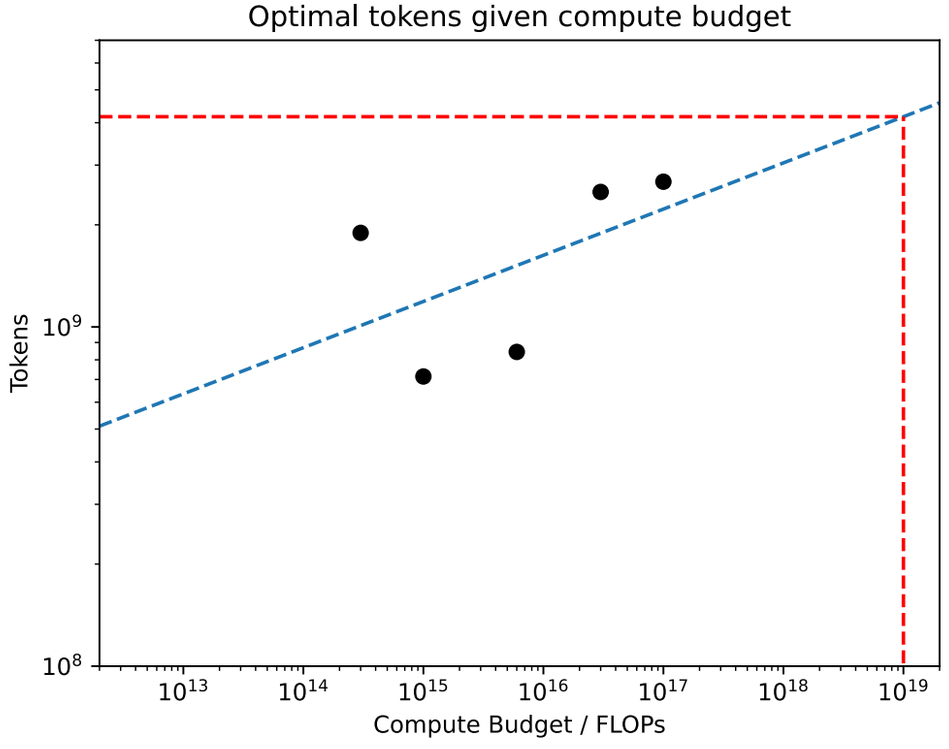

Figure 6: The same setup as in Figure 5, but transforming the points and the line to instead represent the number of tokens. The optimal value for 10^{19} FLOPs is 4 times 10^8 parameters.

We run a few experiments spread out around the FLOPs space to find good regions of the n_parameters axis to continue to measure.

It seems like the scaling laws fit poorly for low FLOPs counts, but do much better for the higher values of FLOPs. The final linear fit is a compromise, and doesn't look like it fits the data extraordinarily well, but it is the best I can do with the data I ended up measuring.

For the FLOPs budget of 1e19, the optimal model size predicted is 4e8 parameters, which given my model setup would give a model with $n_{\text{layers}} = \sqrt[3]{\frac{4\times10^8}{32^2\times12}} \Rightarrow 32$ layers with $d_{\text{model}} = 1024$.

Putting the measurements together and fitting curves gives us the following results:

Finally, the loss estimate for the model with 10^{19} FLOPs is comes to 3.59, as seen in Figure 7.

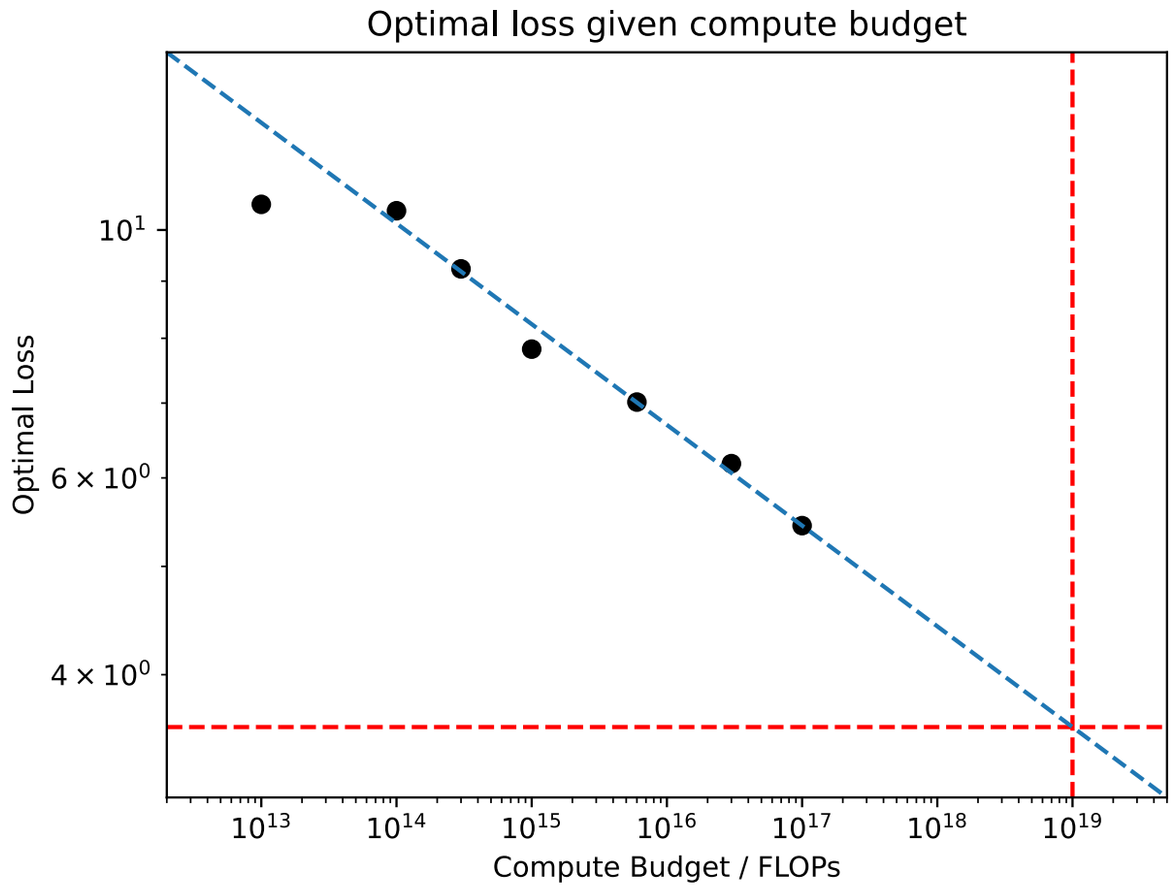

Figure 7: If we ignore the outliers at 10^{13} FLOPs, we get a pretty good fit for the loss estimates in log-log space. The final value at 10^{19} FLOPs is projected to be 3.59.

For the final hyperparameters, based on my experiments I would use the following setup:

$$\text{batch\_size} = 16 \text{ or even } 8$$
$$\text{num\_heads} = 12$$
$$d_{\text{model}} = 1024$$
$$n_{\text{layers}} = 32$$
$$\text{lr} = 10^{-4}$$

Note that using the API didn't give me the best tools to explore the hyperparameter space considering the fact that I can use parameter values outside of those available in the API. If I could, I would do more tests with a smaller batch size and potentially a smaller learning rate.

Bibliography

[1] J. Kaplan *et al.*, "Scaling Laws for Neural Language Models." Accessed: May 08, 2024. [Online]. Available: http://arxiv.org/abs/2001.08361

[2] J. Hoffmann *et al.*, "Training Compute-Optimal Large Language Models." Accessed: May 08, 2024. [Online]. Available: http://arxiv.org/abs/2203.15556