Abstract

Hyperparameter (HP) tuning in deep learning is an expensive process, prohibitively so for neural networks (NNs) with billions of parameters. We show that, in the recently discovered Maximal Update Parametrization ($\mu P$), many optimal HPs remain stable even as model size changes. This leads to a new HP tuning paradigm we call $\mu P$ Transfer: parametrize the target model in $\mu P$, tune the HP indirectly on a smaller model, and zero-shot transfer them to the full-sized model, i.e., without directly tuning the latter at all. We verify $\mu P$ Transfer on Transformer and ResNet. For example, 1) by transferring pretraining HPs from a model of 13M parameters, we outperform published numbers of BERT-large (350M parameters), with a total tuning cost equivalent to pretraining BERT-large once; 2) by transferring from 40M parameters, we outperform published numbers of the 6.7B GPT-3 model, with tuning cost only 7% of total pretraining cost. A Pytorch implementation of our technique can be found at [github.com/microsoft/mup](https://github.com/microsoft/mup).

1 Introduction

Hyperparameter (HP) tuning is critical to deep learning. Poorly chosen HPs result in subpar performance and training instability. Many published baselines are hard to compare to one another due to varying degrees of HP tuning. These issues are exacerbated when training extremely large deep learning models, since state-of-the-art networks with billions of parameters become prohibitively expensive to tune.

Recently, [45] showed that different neural network parametrizations induce different infinite-width limits and proposed the Maximal Update Parametrization (abbreviated $\mu P$) (summarized in Table 3) that enables “maximal” feature learning in the limit. Intuitively, it ensures that each layer is updated on the same order during training regardless of width. In contrast, while the standard parametrization (SP) ensures activations are of unit order at initialization, it actually causes

Figure 1: Training loss against learning rate on Transformers of varying $d_{model}$ trained with Adam. Conventionally and in contrast with our technique, different widths do not share the same optimal hyperparameter; wider networks do not always perform better than narrower ones; in fact they underperform the same-width networks in our technique even after tuning learning rate. See Sections 3 and 4 for experimental setup.

\[\text{Reference}\]

1Work done partly during Microsoft AI Residency Program.
2Equal contribution. Order is random. Correspondence to {gregyang, edwardhu}@microsoft.com
3See arxiv.org for the full, up-to-date version of this work.
4i.e., the updates’ effect on activations becomes roughly independent of width in the large width limit.

Algorithm 1 Tuning a Large Target Model via $\mu$Transfer

1: Parametrize target model in Maximal Update Parametrization ($\mu P$)
2: Tune a smaller version (in width and/or depth) of target model
3: Copy tuned hyperparameters to target model

Table 1: Hyperparameters That Can Be $\mu$Transferred, Not $\mu$Transferred, or $\mu$Transferred Across, with a few caveats discussed in Section 5.1. * means empirically validated only on Transformers, while all others additionally have theoretical justification.

<table>
<thead>
<tr>
<th>$\mu$Transferable</th>
<th>Not $\mu$Transferable</th>
<th>$\mu$Transferred Across</th>
</tr>
</thead>
<tbody>
<tr>
<td>optimization related, init, parameter multipliers, etc</td>
<td>regularization</td>
<td>width, depth*, batch size*, training time*, seq length*</td>
</tr>
</tbody>
</table>

them to blow up in wide models during training essentially due to an imbalance of per-layer learning rate (also see Fig. 8). We leverage $\mu P$ to zero-shot transfer HPs from small models to large models in this work – that is, we obtain near optimal HPs on a large model without directly tuning it at all! While practitioners have always guessed HPs of large models from those of small models, the results are hit-or-miss at best because of incorrect parametrization. For example, as shown in Fig. 1, in a Transformer, the optimal learning rate is stable with width in $\mu P$ (right) but far from so in standard parametrization (left). In addition to width, we empirically verify that, with a few caveats, HPs can also be transferred across depth (in Section 5.1) as well as batch size, language model sequence length, and training time (in Appendix I.2.1). This reduces the tuning problem of an (arbitrarily) large model to that of a (fixed-sized) small model. Our overall procedure, which we call $\mu$Transfer, is summarized in Algorithm 1 and Fig. 2, and the HPs we cover are summarized in Tables 1 and 2.

There are several benefits to our approach: 1. **Better Performance:** $\mu$Transfer is not just about predicting how the optimal learning rate scales in SP. In general, we expect the $\mu$Transferred model to outperform its SP counterpart with learning rate optimally tuned. For example, this is the case in Fig. 1 with the width-8192 Transformer. We discuss the reason for this in Appendices B and C. 2. **Speedup:** It provides massive speedup to the tuning of large models. For example, we are able to outperform published numbers of (350M) BERT-large [9] purely by zero-shot HP transfer, with tuning cost approximately equal to 1 BERT-large pretraining. Likewise, we outperform the published numbers of the 6.7B GPT-3 model [6] with tuning cost being only 7% of total pretraining cost. For models on this scale, HP tuning is not feasible at all without our approach. 3. **Tune Once for Whole Family:** For any fixed family of models with varying width and depth (such as the BERT family or the GPT-3 family), we only need to tune a single small model and can reuse its HPs for all models in the family [1]. For example, we will use this technique to tune BERT-base (110M parameters) and BERT-large (350M parameters) simultaneously by transferring from a 13M model. 4. **Better Compute Utilization:** While large model training needs to be distributed across many GPUs, the small model tuning can happen on individual GPUs, greatly increasing the level of parallelism for tuning (and in the context of organizational compute clusters, better scheduling and utilization ratio). 5. **Painless Transition from Exploration to Scaling Up:** Often, researchers explore new ideas on small models but, when scaling up, find their HPs optimized during exploration work poorly on large models. $\mu$Transfer would solve this problem.

Nevertheless, $\mu$Transfer still has several limitations. For example, while it is very effective for pretraining, it cannot transfer regularization HPs [4], so it’s generally not applicable to the finetuning of pretrained models. We discuss other limitations carefully in Section 5.1.

---

4 but possibly not for different data and/or tasks.
5 It can transfer regularization HPs to the extent they help training but it may not transfer their effect on testing.
Table 2: Examples of μ-Transferable Hyperparameters. All of the below can also be specialized to per-layer hyperparameters.

<table>
<thead>
<tr>
<th>Optimizer Related</th>
<th>Initialization</th>
<th>Parameter Multipliers</th>
</tr>
</thead>
<tbody>
<tr>
<td>learning rate (LR), momentum, Adam beta, LR schedule, etc</td>
<td>per-layer init. variance</td>
<td>multiplicative constants after weight/biases, etc</td>
</tr>
</tbody>
</table>

Our Contributions

- We demonstrate it is possible to zero-shot transfer near optimal HPs to a large model from a small model via the Maximal Update Parametrization (μP) from [45].
- While [45] only covered SGD, here we derive μP for Adam as well (Table 3).
- We propose a new HP tuning technique, μTransfer, for large neural networks based on this observation that provides massive speedup over conventional methods and covers both SGD and Adam training;
- We thoroughly verify our method on machine translation and large language model pretraining (in Section 6.3) as well as image classification (in Appendix I.1);
- We release a PyTorch [27] package for implementing μTransfer painlessly. A sketch of this package is given in Appendix J.

Terminologies

Sometimes, to be less ambiguous, we often refer to the “large model” as the target model, as it is the model we wish to ultimately tune, while we refer to the “small model” as the proxy model, as it proxies the HP tuning process. We follow standard notation regarding dimensions in a Transformer; one can see Fig. 11 for a refresher.

Tensor Programs Series

This paper is the 5th installment of the Tensor Programs series. While the target audience here are practitioners and empirical researchers, this paper presents the first major practical payoff of the theoretical foundation built in previous works [41–46].

2 Parametrization Matters: A Primer

In this section, we give a very basic primer on why the correct parametrization can allow HP transfer across width, but see Appendices L.1 to L.3 for more (mathematical) details.

The Central Limit Theorem (CLT) says that, if \( x_1, \ldots, x_n \) are iid samples from a zero-mean, unit-variance distribution, then \( \frac{1}{\sqrt{n}} (x_1 + \cdots + x_n) \) converges to a standard Gaussian \( \mathcal{N}(0, 1) \) as \( n \to \infty \).

Therefore, we can say that \( \frac{1}{\sqrt{n}} \) is the right order of scaling factor \( c_n \) such that \( c_n (x_1 + \cdots + x_n) \) converges to something nontrivial. In contrast, if we set \( c_n = 1/n \), then \( c_n (x_1 + \cdots + x_n) \to 0 \); or if \( c_n = 1 \), then \( c_n (x_1 + \cdots + x_n) \) blows up in variance as \( n \to \infty \).

Now suppose we would like to minimize the function

\[
F_n(c) \overset{def}{=} \mathbb{E}_{x_1, \ldots, x_n} f(c(x_1 + \cdots + x_n))
\]

over \( c \in \mathbb{R} \), for some bounded continuous function \( f : \mathbb{R} \to \mathbb{R} \). If we reparametrize \( c = \alpha / \sqrt{n} \) for \( \alpha \in \mathbb{R} \), then by CLT, \( G_n(\alpha) \overset{def}{=} F_n(c) \to \mathbb{E} f(\mathcal{N}(0, \alpha^2)) \) stabilizes into a function of \( \alpha \) as \( n \to \infty \).

Then for sufficiently large \( n \), the optimal \( \alpha_n^* \) defined \( G_n(\alpha) \) should be close to \( \alpha_N^* \) for any \( N > n \), and indeed, for \( N = \infty \) — this precisely means we can transfer the optimal \( c_n^* \) or \( \alpha_n^* \) for a smaller problem (say \( F_n \)) to a larger problem (say \( F_N \)): \( G_N \) is approximately minimized by \( \alpha_n^* \) and \( F_N \) is approximately minimized by \( c_n^* \sqrt{n}/N \). Because the transfer algorithm is simply copying \( \alpha \), we say the parametrization \( c = \alpha / \sqrt{n} \) is the correct parametrization for this problem.

In the scenario studied in this paper, \( x_1, \ldots, x_n \) are akin to randomly initialized parameters of a width-\( n \) neural network, \( c \) is akin to a HP such as learning rate, and \( f \) is the test-set performance of the network after training, so that \( F_n \) gives its expectation over random initializations. Just as in this example, if we parametrize the learning rate and other HPs correctly, then we can directly copy the optimal HPs for a narrower network into a wide network and expect approximately optimal
performance — this is the hyperparameter transfer we propose here. It turns out the Maximal Update Parametrization ($\mu P$) introduced in [45] is correct (akin to the parametrization in $\alpha$ above), while the standard parametrization (SP) is incorrect (akin to the parametrization in $c$). We will review both parametrizations shortly. Theoretically, a $\mu P$ network has a well-defined infinite-width limit — akin to $(x_1 + \cdots + x_n)/\sqrt{n}$ having a $N(0, 1)$ limit by CLT — while a SP network does not (the limit will blow up) [45]. In fact, based on the theoretical foundation laid in [45], we argue in Appendix L.3 that $\mu P$ should also be the unique parametrization that allows HP transfer across width.

We emphasize that, to ensure transferability of any hyperparameter (such as learning rate), it’s not sufficient to reparameterize only that hyperparameter, but rather, we need to identify and correctly reparameterize all hyperparameters in Table 2. For example, in Fig. 1 the wide models in SP still underperform their counterparts in $\mu P$, even with learning rate tuned optimally. This is precisely because SP does not scale parameter multipliers and input/output layer learning rates correctly in contrast to $\mu P$ (see Table 3). See Appendix C for more intuition via a continuation of our example here. We shall also explain this more concretely in the context of neural networks in Appendix B.

3 Hyperparameters Don’t Transfer Conventionally

In the community there seem to be conflicting assumptions about HP stability. A priori, models of different sizes don’t have any reason to share the optimal HPs. Indeed, papers aiming for state-of-the-art results often tune them separately. On the other hand, a nontrivial fraction of papers in deep learning fixes all HPs when comparing against baselines, which reflects an assumption that the optimal HPs should be stable — not only among the same model of different sizes but also among models of different designs — therefore, such comparisons are fair. Here, we demonstrate HP instability across width explicitly in MLP and Transformers in the standard parametrization. We will only look at training loss to exclude the effect of regularization.

**MLP with Standard Parametrization** We start with a 2-hidden-layer MLP with activation function $\phi$, using the standard parametrization $\Phi$ with LeCun initialization$^6$ akin to the default in PyTorch:

$$f(\xi) = W^{3T} \phi(W^{2T} \phi(W^{1T} \xi + b^1) + b^2)$$

with init. $W^1 \sim N(0, 1/d_{in}), W^{(2,3)} \sim N(0, 1/n), b^{(1,2)} = 0,

$$W^1 \in \mathbb{R}^{d_{in} \times n}, b^1 \in \mathbb{R}^n,$$

$$W^2 \in \mathbb{R}^{n \times d_{out}}, b^2 \in \mathbb{R}^n,$$

$$W^3 \in \mathbb{R}^{n \times d_{out}},$$

and $d_{in}$, $n$, and $d_{out}$ are the input, hidden, and output dimensions. The particular MLP we use has $\phi = \text{ReLU}$ and a cross-entropy (xent) loss function. We define the width of MLP as the hidden size $n$, which is varied from 256 to 8192. The models are trained on CIFAR-10 for 20 epochs, which is more than enough to ensure convergence.

As shown on the left in Fig. 3, the optimal learning rate shifts by roughly an order of magnitude as the width increases from 256 to 8192; using the optimal learning of the smallest model on the largest model gives very bad performance, if not divergence.

**Transformer with Standard Parametrization** This perhaps unsurprising observation holds for more complex architectures such as Transformer as well, as shown in Fig. 1 (left). We define width

$^6$ The more theoretically astute reader may observe that SP with a $\Theta(1/width)$ learning rate induces a well-defined infinite-width limit exists as well. Nevertheless, this does not allow HP transfer because this limit is in kernel regime as shown in [45]. See Appendix L.3 for more discussions.

$^7$ i.e. the default parametrization offered by common deep learning frameworks. See Table 3 for a review.

$^8$ The key here is that the init. variance $\propto 1/fan\_in$, so the same insights here apply with e.g. He initialization.
Table 3: $\mu P$ and SP for General Neural Networks, Basic Form. This basic form emphasizes the scaling with width ($\text{fan}_{\text{in}}$ or $\text{fan}_{\text{out}}$); in practice, we may insert tunable multipliers in front of $\text{fan}_{\text{in}}$ and $\text{fan}_{\text{out}}$ as in Eq. (4). Notations: 1) $\eta$ is the “master” learning rate. 2) The $\text{fan}_{\text{out}}$ of a bias vector is its dimension (whereas $\text{fan}_{\text{in}}$ is 1). 3) Purple text highlights key differences from standard parametrization (SP); Gray text recalls the corresponding SP. SGD (resp. Adam) here can be replaced by variants such as SGD with momentum (resp. Adagrad, Adadelta, etc). In general, the three columns here can be interpreted as linear layers that have \{finite, infinite, infinite\} input dimension and \{infinite, finite, infinite\} output dimension in an infinite-width network; this description generalizes more readily to other parameters such as those of layernorm. Transformer $\mu P$ requires one more modification ($1/d$ attention instead of $1/\sqrt{d}$); see Definition 4.1. This version of $\mu P$ gets rid of parameter multipliers; for the version similar to that in [45], see Table 13. Also see Table 12 for a $\mu P$ formulation that is easier to implement (and compatible with input/output weight sharing).

<table>
<thead>
<tr>
<th>Init. Var.</th>
<th>Output weights</th>
<th>Hidden weights</th>
</tr>
</thead>
<tbody>
<tr>
<td>SGD LR</td>
<td>$\eta \cdot \text{fan}_{\text{out}}$ ($\eta$)</td>
<td>$\eta/fan_{\text{in}}$ ($\eta$)</td>
</tr>
<tr>
<td>Adam LR</td>
<td>$\eta$</td>
<td>$\eta/fan_{\text{in}}$ ($\eta$)</td>
</tr>
</tbody>
</table>

as $d_{\text{model}}$, with $d_k = d_v = d_{\text{model}}/n_{\text{head}}$ and $d_{f_{\text{nn}}} = 4d_{\text{model}}$. The models are trained on wikitext-2 for 5 epochs. In Fig. 18 in the appendix we also show the instability of initialization scale and other HPs.

4 Unlocking Zero-Shot Hyperparameter Transfer with $\mu P$

We show that $\mu P$ solves the problems we see in Section 3.

**MLP with $\mu P$** For the MLP in Section 3 to switch to $\mu P$, we just need to modify Eq. (2)’s initialization of the last layer and its learning rates of the first and last layer as well as of the biases. The basic form is

\[
\text{initialize } W^1 \sim \mathcal{N}(0, 1/d_{in}), W^2 \sim \mathcal{N}(0, 1/n), W^3 \sim \mathcal{N}(0, 1/n^2), b^{(1,2)} = 0
\]

with SGD learning rates $\eta_{W^1} = \eta_{b^1} = \eta_{b^2} = \eta, \eta_{W^2} = \eta, \eta_{W^3} = \eta/n^1$.

(3)

Here, $\eta$ specifies the “master” learning rate, and we highlighted in purple the differences in the two parametrizations. This basic form makes clear the scaling with width $n$ of the parametrization, but in practice we will often insert (possibly tune-able) multiplicative constants in front of each appearance of $n$. For example, this is useful when we would like to be consistent with a SP MLP at a base width $n_0$. Then we may insert constants as follows: For $n = n_0$,

\[
\text{initialize } W^1 \sim \mathcal{N}(0, 1/d_{in}), W^2 \sim \mathcal{N}(0, 1/n), W^3 \sim \mathcal{N}(0, 1/n^2), b^{(1,2)} = 0
\]

with SGD learning rates $\eta_{W^1} = \eta_{b^1} = \eta_{b^2} = \eta/n, \eta_{W^2} = \eta, \eta_{W^3} = \eta/n^1$.

(4)

Then at width $n = n_0$, all purple factors above are 1, and the parametrization is identical to SP (Eq. (2)) at width $n_0$. Of course, as $n$ increases from $n_0$, then Eq. (4) quickly deviates from Eq. (2) in other words, for a particular $n$, $\mu P$ and SP can be identical up to the choice of some constants (in this case $n_0$), but $\mu P$ determines a different “set” of networks and optimization trajectory than SP as one varies $n$. As we will see empirically in the next section, this deviation is crucial for HP transfer.

Indeed, in Fig. 3(right), we plot the CIFAR10 performances, over various learning rates and widths, of $\mu P$ MLPs with $n_0 = 128$. In contrast to SP, the optimal learning rate under $\mu P$ is stable. This means that, the best learning rate for a width-128 network is also best for a width-8192 network in $\mu P$ — i.e. HP transfer works — but not for SP. In addition, we observe performance for a fixed learning rate always weakly improves with width in $\mu P$, but not in SP.

This MLP $\mu P$ example can be generalized easily to general neural networks trained under SGD or Adam, as summarized in Table 3 which is derived in Appendix L.

---

*While superficially different, this parametrization is equivalent to the $\mu P$ defined in [45]."
Figure 4: Empirical validation of the stability of four representative hyperparameters on pre-LN Transformers in $\mu_P$: learning rate, last layer weight multiplier $\alpha_{\text{output}}$, weight initialization standard deviation, and learning rate schedule. We use the following learning rate schedules: (a) linear decay; (b) StepLR @ [5k, 8k] with a decay factor of 0.1; (c) StepLR @ [4k, 7k] with a decay factor of 0.3; (d) cosine annealing; (e) constant; (f) inverse square-root decay. All models are trained on wikitext-2 for 10k steps. When not specified in the legend, the width used is 256, depth 2, batch size 20, sequence length 256, and LR schedule constant. We sweep a particular HP, corresponding to each column, while fixing all others constant. See Section 5.1 for discussion of these results.

Transformers with $\mu_P$ We repeat the experiments with base width $n_0 = 128$ for Transformers:

Definition 4.1. The Maximal Update Parametrization ($\mu_P$) for a Transformer is given by Table 3 and $1/d$ attention instead of $1/\sqrt{d}$, i.e. the attention logit is calculated as $q^\top k/d$ instead of $q^\top k/\sqrt{d}$ where query $q$ and key $k$ have dimension $d$.

The results are shown on the right in Fig. 1, where the optimal learning rate is stable, and the performance improves monotonically as width increases.

5 Which Hyperparameters Can Be $\mu_P$ Transferred?

In this section, we explore how common HPs fit into our framework. In general, they can be divided into three kinds, summarized in Table 1:

1. those that can transfer from the small to the large model, such as learning rate (Table 2);
2. those that primarily control regularization and don’t work well with our technique; and
3. those that define training scale, such as width as discussed above as well as others like depth and batch size, across which we transfer other HPs.

Those in the first category transfer across width, as theoretically justified above in Section 2. To push the practicality and generality of our technique, we empirically explore the transfer across the other dimensions in the third category. Note that $\mu_P$ transfer across width is quite general, e.g. it allows varying width ratio of different layers or number of attention heads in a Transformer; see Appendix G.2. This will be very useful in practice. For the second category, the amount of regularization (for the purpose of controlling overfitting) naturally depends on both the model size and data size, so we should not expect transfer to work if the parametrization only depends on model size. We discuss these HPs in more detail in Appendix G.1.

---

10 This is roughly because during training, $q$ and $k$ will be correlated so $q^\top k$ actually scales like $d$ due to Law of Large Numbers, in contrast to the original motivation that $q, k$ are uncorrelated at initialization so Central Limit applies instead. See Appendix L.2.1 for a more in-depth discussion.
5.1 Empirical Validation and Limitations

Our empirical investigations focus on Transformers (here) and ResNet (in Appendix I.1.1), the most popular backbones of deep learning models today. We train a 2-layer pre-layernorm $\mu^{P}$ Transformer with 4 attention heads on Wikitext-2. We sweep one of four HPs (learning rate, output weight multiplier, initialization standard deviation, and learning rate schedule) while fixing the others and sweeping along width and depth (with additional results in Fig. 19 on transfer across batch size, sequence length, and training time). Fig. 4 shows the results averaged over 5 random seeds.

Empirically, we find that for language modeling on Transformers, HPs generally transfer across scale dimensions if some minimum width (e.g. 256), depth (e.g., 4), batch size (e.g., 32), sequence length (e.g., 128), and training steps (e.g., 5000) are met, with some caveats discussed below. While the exact optimum can shift slightly with increasing scale, this shift usually has very small impact on the loss, compared to SP (Figs. 1 and 3(left)). However, there are some caveats. For example, the best initialization standard deviation does not seem to transfer well across depth (2nd row, 3rd column), despite having a stabler optimum across width. In addition, while our results on width, batch size, sequence length, and training time still hold for post-layernorm (Fig. 17), the transfer across depth only works for pre-layernorm Transformer. Nevertheless, in practice (e.g. our results in Section 6.3) we find that fixing initialization standard deviation while tuning other HPs works well when transferring across depth.

6 Efficiency and Performance of $\mu$Transfer

Now that the plausibility of $\mu$Transfer has been established in toy settings, we turn to more realistic scenarios to see if one can achieve tangible gains. Specifically, we perform HP tuning only on a smaller proxy model, test the obtained HPs on the large target model directly, and compare against baselines tuned using the target model. We seek to answer the question: Can $\mu$Transfer make HP tuning more efficient while achieving performance on par with traditional tuning? As we shall see by the end of the section, the answer is positive. We focus on Transformers here, while experiments on ResNets on CIFAR10 and Imagenet can be found as well in Appendix I.1. All of our experiments are run on V100 GPUs.

6.1 Transformer on IWSLT14 De-En

Setup IWSLT14 De-En is a well-known machine translation benchmark. We use the default IWSLT (post-layernorm) Transformer implemented in fairseq [25] with 40M parameters, which we denote as the 1x model [13]. For $\mu$Transfer, we tune on a 0.25x model with $1/4$ of the width, amounting to 4M parameters. For this experiment, we tune via random search the learning rate $\eta$, the output layer parameter multiplier $\alpha_{\text{output}}$, and the attention key-projection weight multiplier $\alpha_{\text{attn}}$. See the grid and other experimental details in Appendix H.1.

We compare transferring from the 0.25x model with tuning the 1x model while controlling the total tuning budget in FLOPs [14]. To improve the reproducibility of our result: 1) we repeat the entire HP search process (a trial) 25 times for each setup, with number of samples as indicated in Table 4 and report the 25th, 50th, 75th, and 100th percentiles in BLEU score; 2) we evaluate each selected HP combination using 5 random initializations and report the mean performance [15].

We pick the HP combination that achieves the lowest validation loss [16] for each trial. The reported best outcome is chosen according to the validation loss during tuning. We compare against the default in fairseq, which is presumably heavily tuned. The result is shown in Table 4.

---

11 “2 layers” means the model has 2 self-attention blocks. To compare with SP Transformer, see Fig. 18.
12 In fact, post-layernorm Transformers are much more sensitive to HPs than pre-layernorm, so our technique is more crucial for them, especially for transfer across width. Fig. 1 uses post-layernorm.
13 https://github.com/pytorch/fairseq/blob/master/examples/translation/README.md
14 Ideally we would like to measure the wall clock time used for tuning. However, smaller models such as the proxy Transformer used for IWSLT are not efficient on GPUs, so wall clock time would not reflect the speedup for larger models like GPT-3. Thus, we measure in FLOPs, which is less dependent on hardware optimization.
15 We do not report the standard deviation over random initializations to avoid confusion.
16 We find this provides more reliable result than selecting for the best BLEU score.
Table 4: Transformer on IWSLT14 De-En. 1x and 0.25x refers to scaling of width only. Compared to traditional tuning (“Tuning on 1x”), \(\mu\)-transfer from 0.25x provides better and more reliable outcome given fixed amount of compute. On the other hand, naive transfer (i.e. with SP instead of \(\mu\)) fails completely. The percentiles are over independent trials, with each trial involving the entire tuning process with a new HP random search.

<table>
<thead>
<tr>
<th>Setup</th>
<th>Total Compute</th>
<th>#Samples</th>
<th>Val. BLEU Percentiles</th>
</tr>
</thead>
<tbody>
<tr>
<td>fairseq [25] default</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Tuning on 1x</td>
<td>1x</td>
<td>3</td>
<td>25.69</td>
</tr>
<tr>
<td>Naive transfer from 0.25x</td>
<td>1x</td>
<td>64</td>
<td>training diverged</td>
</tr>
<tr>
<td>(\mu)-Transfer from 0.25x (Ours)</td>
<td>1x</td>
<td>64</td>
<td>25.94 26.33 26.42</td>
</tr>
</tbody>
</table>

Performance Pareto Frontier The result above only describes a particular compute budget. Is \(\mu\)-Transfer still preferable when we have a lot more (or less) compute? To answer this question, we produce the compute-performance Pareto frontier in Fig. 5 (left), where we repeat the above experiment with different compute budgets. Evidently, our approach completely dominates conventional tuning.

Sample Quality of Proxy Model vs Target Model The Pareto frontier in Fig. 5 (right) suggests that, given a fixed number of random samples from the HP space, 1) tuning the target model directly yields slightly better results than tuning the proxy model (while taking much more compute of course), but 2) this performance gap seems to vanish as more samples are taken. This can be explained by the intuition that the narrower proxy model is a “noisy estimator” of the wide target model [25]. With few samples, this noise can distort the random HP search, but with more samples, this noise is suppressed.

6.2 Transformer on WMT14 En-De

We scale up to WMT14 En-De using the large (post-layernorm) Transformer from [37] with 211M parameters. We tune on a proxy model with 15M parameters by shrinking \(d_{\text{model}}, d_{\text{ffn}},\) and \(n_{\text{head}}\). For this experiment, we tune via random search the learning rate \(\eta\), the output layer parameter multiplier \(\alpha_{\text{output}}\), and the attention key-projection weight multiplier \(\alpha_{\text{attn}}\) following the grid in Appendix H.2. The result is shown in Table 5: While random search with 3 HP samples far underperforms the fairseq default, we are able to match it via transfer using the same tuning budget.

Table 5: Transformers on WMT14 En-De. 1x and 0.25x refers to scaling of width only. We report BLEU fluctuation over 3 independent trials, i.e., 3 independent random HP searches.

<table>
<thead>
<tr>
<th>Setup</th>
<th>Total Compute</th>
<th>#Samples</th>
<th>Val. BLEU Percentiles</th>
</tr>
</thead>
<tbody>
<tr>
<td>fairseq [25] default</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Tuning on 1x</td>
<td>1x</td>
<td>3</td>
<td>training diverged</td>
</tr>
<tr>
<td>Naive transfer from 0.25x</td>
<td>1x</td>
<td>64</td>
<td>training diverged</td>
</tr>
<tr>
<td>(\mu)-Transfer from 0.25x (Ours)</td>
<td>1x</td>
<td>64</td>
<td>25.94 26.34 26.42</td>
</tr>
</tbody>
</table>
6.3 BERT

Finally, we consider large-scale language model pretraining where HP tuning is known to be challenging. Using Megatron (pre-layernorm) BERT [32] as a baseline, we hope to recover the performance of the published HPs by only tuning a proxy model that has roughly 13M parameters, which we call BERT-prototype. While previous experiments scaled only width, here we will also scale depth, as discussed in Section 5 and validated in Fig. 4. We use a batch size of 256 for all runs and follow the standard finetuning procedures. For more details on BERT-prototype, what HPs we tune, and how we finetune the trained models, see Appendix H.3.

During HP tuning, we sample 256 combinations from the search space and train each combination on BERT-prototype for \(10^5\) steps. The total tuning cost measured in FLOPs is roughly the same as training 1 BERT-large for the full \(10^6\) steps; the exact calculation is shown in Appendix H.3. The results are shown in Table 6. Notice that on BERT-large, we obtain sizeable improvement over the well-tuned Megatron BERT-large baseline.

Table 6: BERT pretraining. HP transfer outperforms published baselines without tuning the full model directly at all. We tune BERT-base and BERT-large simultaneously via a single proxy model, BERT-prototype. The total tuning cost = the cost of pretraining a single BERT-large. Model speedup refers to the training speedup of BERT-prototype over BERT-base or BERT-large. Total speedup in addition includes time saving from transferring across training steps. Both speedups can be interpreted either as real-time speedup on V100s or as FLOPs speedup (which turn out to be empirically very similar in this case).

<table>
<thead>
<tr>
<th>Model</th>
<th>Method</th>
<th>Model Speedup</th>
<th>Total Speedup</th>
<th>Test loss</th>
<th>MNLI (m/mm)</th>
<th>QQP</th>
</tr>
</thead>
<tbody>
<tr>
<td>BERT\textsubscript{base}</td>
<td>Megatron Default</td>
<td>1x</td>
<td>1x</td>
<td>1.995</td>
<td>84.2/84.2</td>
<td>90.6</td>
</tr>
<tr>
<td>BERT\textsubscript{base}</td>
<td>Naive Transfer</td>
<td>4x</td>
<td>40x</td>
<td>training diverged</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BERT\textsubscript{base}</td>
<td>(\mu)-Transfer (Ours)</td>
<td>4x</td>
<td>40x</td>
<td>1.970</td>
<td>84.3/84.8</td>
<td>90.8</td>
</tr>
<tr>
<td>BERT\textsubscript{large}</td>
<td>Megatron Default</td>
<td>1x</td>
<td>1x</td>
<td>1.731</td>
<td>86.3/86.2</td>
<td>90.9</td>
</tr>
<tr>
<td>BERT\textsubscript{large}</td>
<td>Naive Transfer</td>
<td>22x</td>
<td>220x</td>
<td>training diverged</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BERT\textsubscript{large}</td>
<td>(\mu)-Transfer (Ours)</td>
<td>22x</td>
<td>220x</td>
<td>1.683</td>
<td>87.0/86.5</td>
<td>91.4</td>
</tr>
</tbody>
</table>

6.4 GPT-3

In order to further verify \(\mu\)-Transfer at scale, we applied it to GPT-3 6.7B [6]. This Transformer model (the target model) consists of 32 residual blocks with width 4096. We form the small proxy model by shrinking width to 256, resulting in roughly 40 million trainable parameters, 168 times smaller than the target model. HPs were then determined by a random search on the proxy model. The total tuning cost was only 7% of total pretraining cost. Details of the HP sweep can be found in Appendix H.4.

In order to exclude code difference as a possible confounder, we also re-trained GPT-3 6.7B from scratch using the original HPs from [6]. During training of the \(\mu\)-Transfer model we encountered numerical issues that lead to frequent divergences. In order to avoid them, the model was trained using FP32 precision, even though the original 6.7B model and our re-run were trained using FP16. The resulting \(\mu\)-Transfer model outperforms the 6.7B from [6], and is in fact comparable to the twice-as-large 13B model across our evaluation suite (see Table 9). Selected evaluation results can be found in Table 7 and further details are given in Table 8 and Appendix H.4.

7 Related Works

Hyperparameter Tuning Many have sought to speed up HP tuning beyond the simple grid or random search, such as via Bayesian optimization [34, 35] or multi-arm bandits [15, 18]. There are also dedicated tools such as Optuna [4] and Talos [3] which integrate with existing deep learning frameworks and provide an easy way to apply more advanced tuning techniques. Our work is

\[1\] While we are mainly focused on the efficacy of \(\mu\)-Transfer regardless of precision, it would be interesting to ablate the effect of precision in our results, but we did not have enough resources to rerun the baseline in FP32.

\[2\] It is quite interesting that \(\mu\)-Transfer identified a useful region of hyperparameters leading to much improved performance, which probably would be difficult to discover normally because 1) researchers usually change hyperparameters to accomodate precision and 2) there was no precise enough justification to go against this judgment until \(\mu\)-Transfer.
Table 7: GPT-3 6.7B Pretraining. Selected evaluation results for the GPT-3 6.7B model tuned with µTransfer (transferred from a small proxy model of 40M parameters), compared to the results published in [6] and a re-run with original HPs. Note that the perplexities in this table are based on a custom tokenization and are not comparable to the literature. The validation loss refers to the loss achieved on a random held-out part of our dataset. Zero-shot, One-Shot and Few-Shot refer to the number of additional query and answer pairs passed in the context when performing the sampling-based evaluations. See Appendix H.4 for full evaluation results.

<table>
<thead>
<tr>
<th>Task</th>
<th>Metric</th>
<th>6.7B+µP</th>
<th>6.7B re-run</th>
<th>6.7B from [6]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Validation loss</td>
<td>cross-entropy</td>
<td>1.98</td>
<td>2.03</td>
<td>-</td>
</tr>
<tr>
<td>PTB</td>
<td>perplexity</td>
<td>11.4</td>
<td>13.0</td>
<td>-</td>
</tr>
<tr>
<td>WikiText-103</td>
<td>perplexity</td>
<td>8.56</td>
<td>9.13</td>
<td>-</td>
</tr>
<tr>
<td>One Billion Words</td>
<td>perplexity</td>
<td>20.5</td>
<td>21.7</td>
<td>-</td>
</tr>
<tr>
<td>LAMBADA Zero-Shot</td>
<td>accuracy</td>
<td>73.5</td>
<td>70.8</td>
<td>70.3</td>
</tr>
<tr>
<td>LAMBADA One-Shot</td>
<td>accuracy</td>
<td>69.9</td>
<td>64.8</td>
<td>65.4</td>
</tr>
<tr>
<td>LAMBADA Few-Shot</td>
<td>accuracy</td>
<td>74.7</td>
<td>77.1</td>
<td>79.1</td>
</tr>
<tr>
<td>HellaSwag Zero-Shot</td>
<td>accuracy</td>
<td>72.0</td>
<td>66.7</td>
<td>67.4</td>
</tr>
<tr>
<td>HellaSwag One-Shot</td>
<td>accuracy</td>
<td>71.1</td>
<td>65.9</td>
<td>66.5</td>
</tr>
<tr>
<td>HellaSwag Few-Shot</td>
<td>accuracy</td>
<td>72.4</td>
<td>66.4</td>
<td>67.3</td>
</tr>
</tbody>
</table>

complementary to the above, as they can be used to tune the proxy model. it is only for scientific reasons that we primarily did random search throughout this work.

Hyperparameter Transfer Many previous works explored transfer learning of HP tuning (e.g. [12, 28, 36, 49]). However, to the best of our knowledge, our work is the first to explore zero-shot HP transfer. In addition, we focus on transferring across model scale rather than between different tasks or datasets. Some algorithms like Hyperband [19] can leverage cheap estimates of HP evaluations (like using a small model to proxy a large model) but they are not zero-shot algorithms, so would still be very expensive to apply to large model training. Nevertheless, all of the above methods are complementary to ours as they can be applied to the tuning of our proxy model.

Previously Proposed Scaling Rules of Hyperparameters [11, 23, 31, 53] investigated the right way to scale learning rate with batch size, with sometimes conflicting proposals. which we summarize in Appendix F. [26] studied how learning rate (and batch size) should scale with width for MLPs and CNNs trained with SGD in NTK or standard parametrizations. We provide a detailed comparison of our work with theirs in Appendix F.

Many previous works proposed different initialization or parametrizations with favorable properties, such as better stability for training deep neural networks [5, 10, 13, 21, 30, 47, 48, 51]. Our work differs from these in that we focus on the transferability of optimal HPs from small models to large models in the same parametrization.

8 Conclusion

Leveraging the discovery of a feature learning neural network infinite-width limit, we hypothesized and verified that the HP landscape across NNs of different width is reasonably stable if parametrized according to Maximal Update Parametrization (µP). We further empirically showed that it’s possible to transfer across depth, batch size, sequence length, and training time, with a few caveats. This allowed us to indirectly tune a very large network by tuning its smaller counterparts and transferring the HPs to the full model.

Venues of Improvement Nevertheless, our method has plenty of room to improve. For example, initialization does not transfer well across depth, and depth transfer generally still does not work for post-layernorm Transformers. This begs the question whether a more principled parametrization in depth could solve these problems. Additionally, Fig. 4 shows that the optimal HP still shifts slightly for smaller models. Perhaps by considering finite-width corrections to µP one can fix this shift. Finally, it will be interesting to consider if there’s a way to transfer regularization HPs as a function of both the model size and data size.
**Broader Impact**  Our work makes HP tuning of large models more efficient. This enables large models to be better tuned given the same compute budget, thereby increasing the performance per cost. Organizations large and small can focus their research on small models and scale up only once with reasonable confidence that the training would go well. We do not foresee any direct negative societal impact.

**References**


