Unit-Scaled μP: Simple, Stable Scaling

Based on the papers:

Blake, C., Eichenberg, C., Dean, J., Balles, L., Prince, L.Y., Deiseroth, B., Cruz-Salinas, A.F., Luschi, C., Weinbach, S. and Orr, D. *"u-μP: The Unit-Scaled Maximal Update Parametrization."* arXiv:2407.17465, 2024.

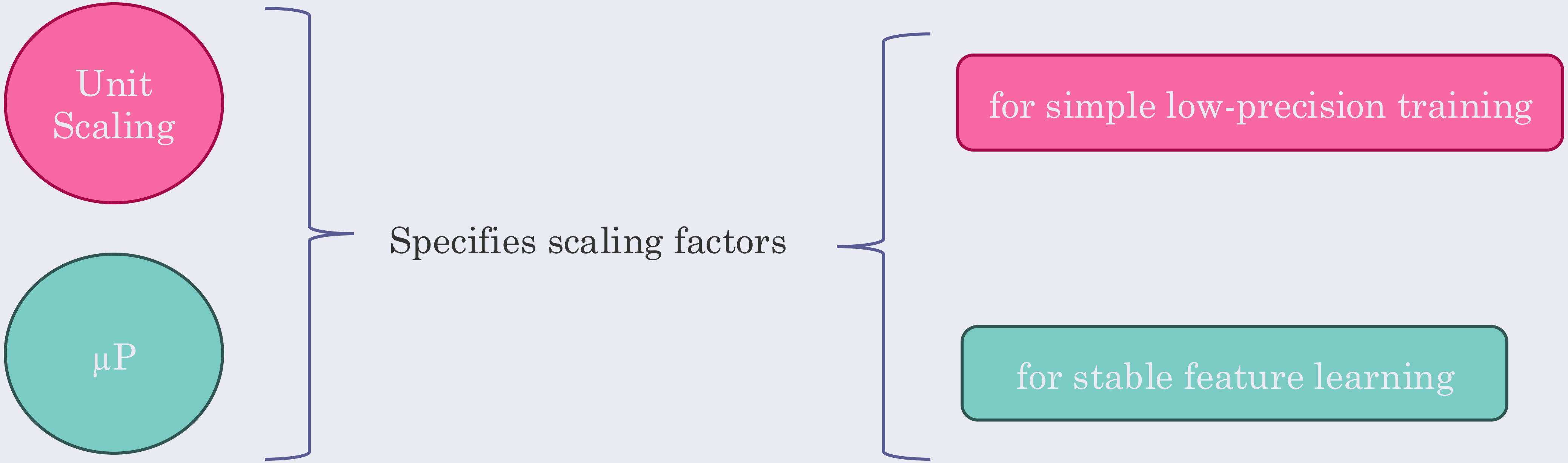
Blake, C., Orr, D., and Luschi, C. *"Unit scaling: Out-of-the-box low-precision training."* ICML, 2023.

Contents

Our goal

Can we make µP more effective, useable and efficient in practice?

Motivation



Starting point: derive a combined scheme giving the benefits of both

Then: make this practically effective for large-model training

Stability

Across all model-widths:
(and possibly depths)

Stable feature-learning:

Different parts of the model learn at the same relative rate

µP

Stable hyperparameters:

A model's optimal HPs are the same

u-µP

Stable numerics:

Floating-point representations stay within range during training

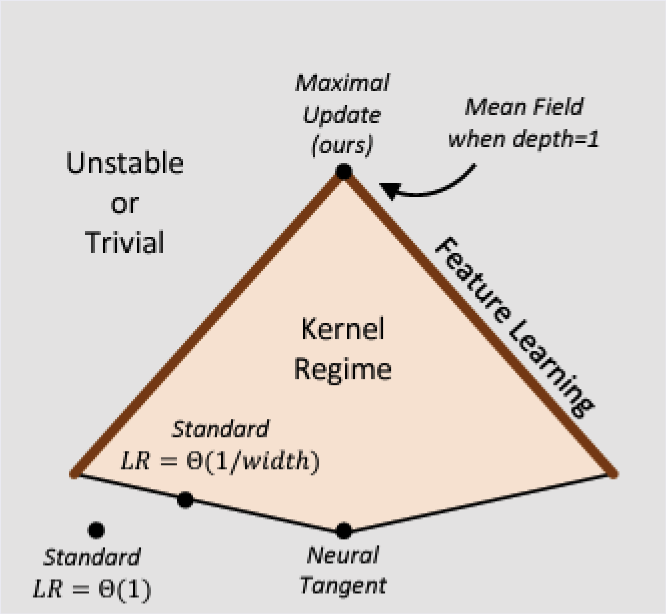
Contributions

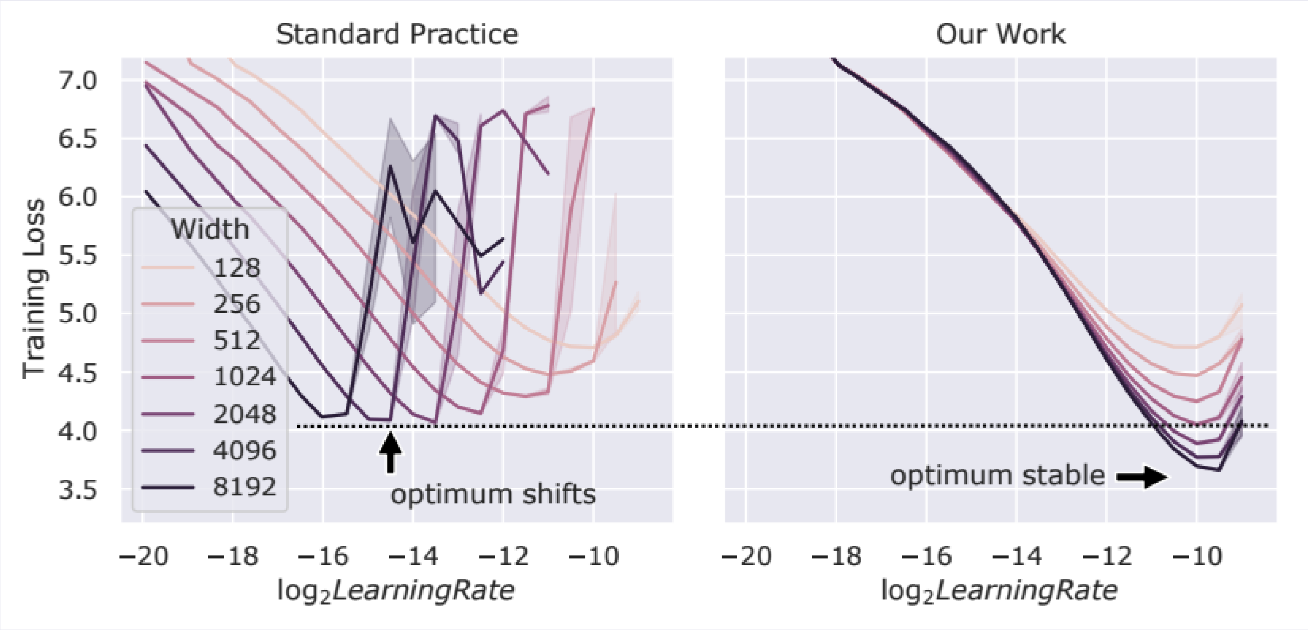
1. Drawbacks of standard µP
2. A simpler set of scaling rules: **u-µP**
3. A more principled set of HPs
4. Improved HP transfer
5. More efficient HP search
6. **Out-of-the-box FP8** training

Contents

Background: µP

Infinite-width limit and HP Transfer





Yang, G., et al. *"Feature Learning in Infinite-Width Neural Networks."* https://arxiv.org/abs/2011.14522, 2022.

Yang, G., et al. *"Tensor Programs V: Tuning large neural networks via zero-shot hyperparameter transfer."* arXiv:2203.03466, 2022.

ABC-parametrizations

During model training, the time evolution of weights is given by:

$$W_t = A_w \cdot w_t$$
$$w_0 \sim \mathcal{N}(0, B_W^2)$$
$$w_{t+1} = w_t - C_W \cdot \Phi_t(\nabla\mathcal{L}_0, \ldots, \nabla\mathcal{L}_t)$$

where t is the time-step and $\Phi_t(\nabla\mathcal{L}_0, \ldots, \nabla\mathcal{L}_t)$ the weight-update.

A parametrization (such as μP) is defined by how A_W, B_W, C_W scale with the model-width.

ABC-parametrizations

$$W_t = A_w \cdot w_t$$
$$w_0 \sim \mathcal{N}(0, B_W^2)$$
$$w_{t+1} = w_t + C_W \cdot \Phi_t(\nabla\mathcal{L}_0, \dots, \nabla\mathcal{L}_t)$$

By considering $a_w \propto A_W$, $b_w \propto B_W$, $c_w \propto C_w$, we can define μP in the following way:

ABC-multiplier			Weight (W) Type		
			Input	Hidden	Output
μP	parameter	(a_w)	1	1	$1/\text{fan-in}(W)$
	initialization	(b_w)	1	$1/\sqrt{\text{fan-in}(W)}$	1
	Adam LR	(c_w)	1	$1/\text{fan-in}(W)$	1

Background: µP

HPs for µP

Each A_W, B_W, C_w comprises an HP, a scaling factor, and a base shape:

$$A_w \leftarrow \alpha_w \frac{a_W}{a_{W_{base}}} \qquad B_w \leftarrow \sigma_w \frac{b_W}{b_{W_{base}}} \qquad C_w \leftarrow \eta_w \frac{c_W}{c_{W_{base}}}$$

ABC-multiplier			Weight (W) Type		
			Input	Hidden	Output
µP	parameter	(a_w)	1	1	$1/\text{fan-in}(W)$
	initialization	(b_w)	1	$1/\sqrt{\text{fan-in}(W)}$	1
	Adam LR	(c_w)	1	$1/\text{fan-in}(W)$	1

In practice this "full" set of HPs is too large.

A global σ and η are typically used, plus some selection of α_ws.

Contents

Making μP work in practice

Prerequisites for effective transfer

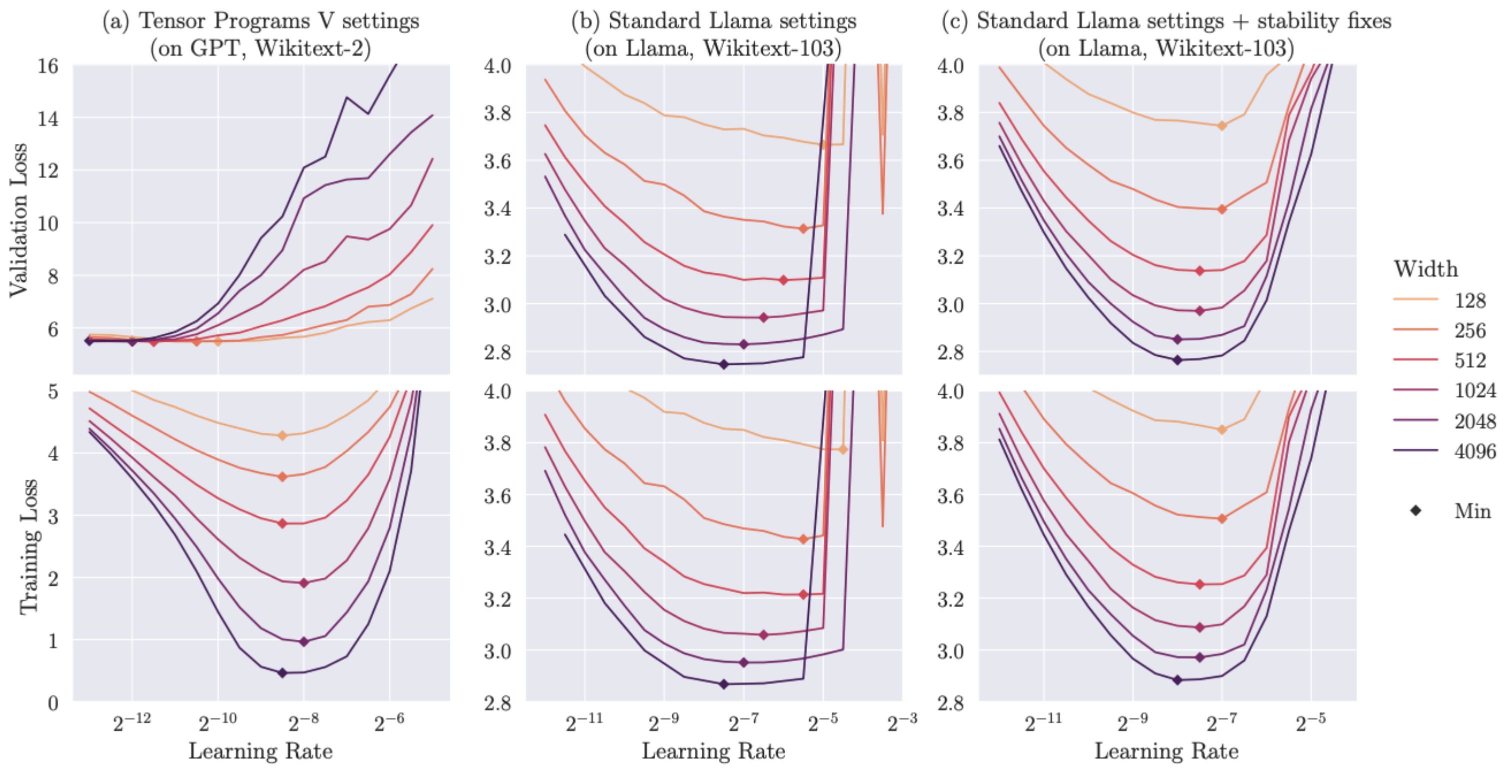
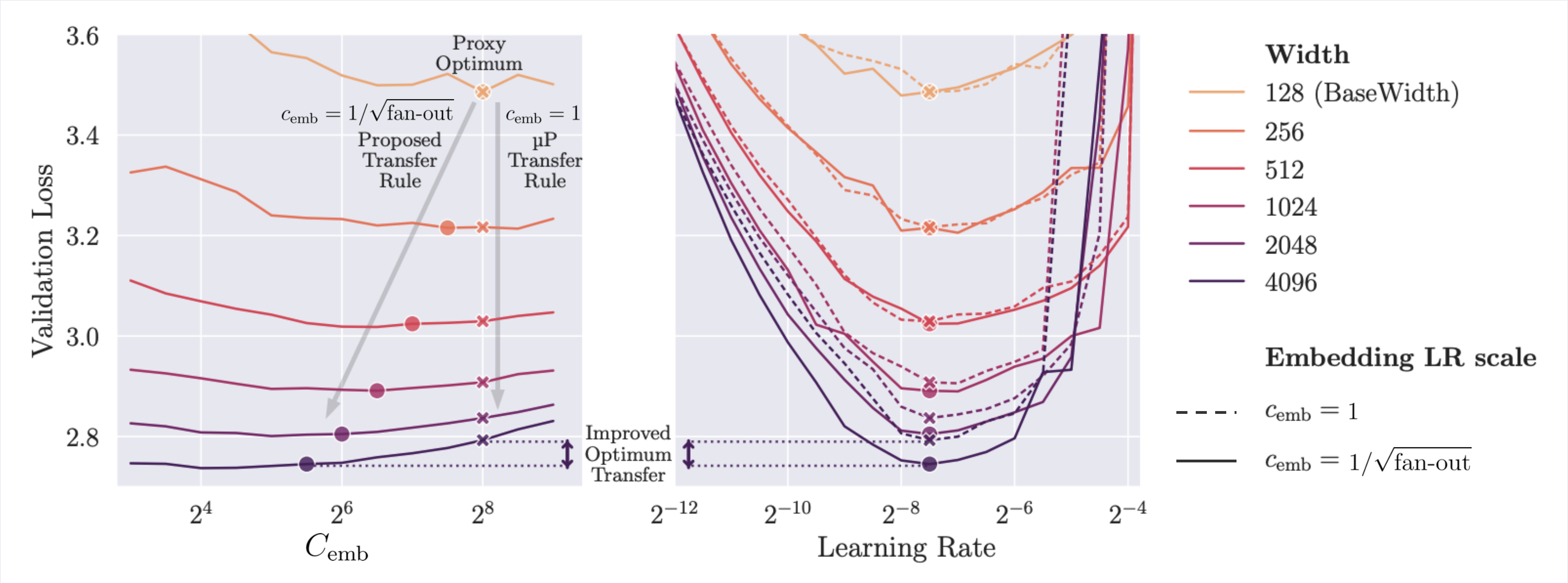


Figure 2: Effective μTransfer does not hold across all training setups. **(a)** We show strong transfer for the unrealistic setup used in Tensor Programs V (too many epochs; constant LR). **(b)** Moving to a more standard Llama training setup, transfer breaks down. **(c)** This is restored by the introduction of two stability fixes: non-parametric norms and independent weight decay.

Lingle, L. *"A Large-Scale Exploration of μ-Transfer."* arXiv:2404.05728. (recommend non-parametric norms)

Wortsman, M., et al. *"Small-scale proxies for large-scale Transformer training instabilities."* ICLR, 2024. (recommend independent weight decay)

An improved embedding LR rule



recall: $w_{t+1} = w_t + C_W \cdot \Phi_t(\nabla\mathcal{L}_0, \ldots, \nabla\mathcal{L}_t),$

Contents

The principle of unit scale

We require standard deviation ≈ 1 for all

- Weights
- Activations
- Gradients (wrt. both activations and weights)

at initialization—i.e. for the first forward and backward pass of the model.

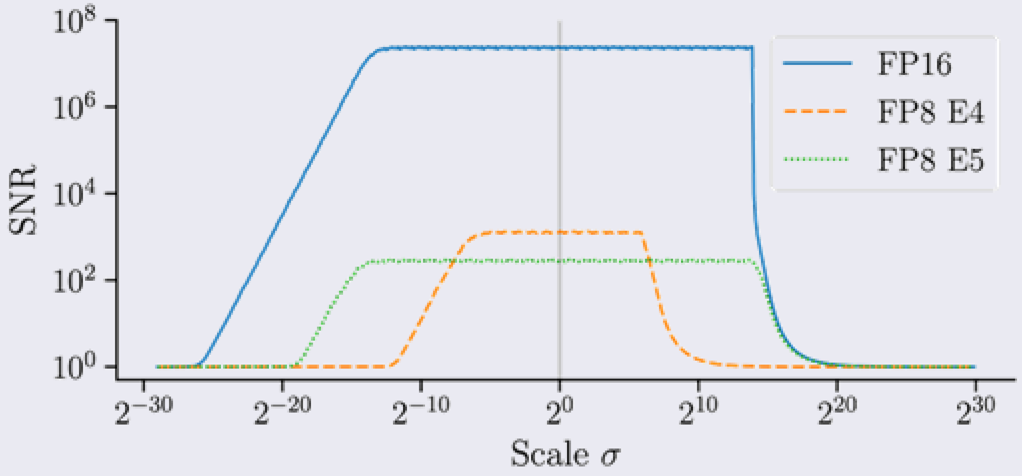


Figure 2. The signal to noise ratio (SNR) of samples from a normal distribution, quantised in FP16 and FP8, as a function of the distribution's scale.

Can we predict scale?

If we can predict the scale of tensors, we can normalize them.
This is hard over all of training, but is easier at initialization:

- Consider a simple linear op:

$$W \in \mathbb{R}^{n\times m} \sim \mathcal{N}(0, \sigma_w^2) \qquad x \in \mathbb{R}^{m} \sim \mathcal{N}(0, \sigma_x^2)$$

$$y = Wx$$

$$\sigma_y = \sigma_w \sigma_x \sqrt{m}$$

$$y \in \mathbb{R}^{n} \sim \mathcal{N}\left(0, \sigma_y\right)$$

- A unit-scaled linear layer uses a $1/\sqrt{m}$ forward scaling factor

- Similar analysis can be performed for other ops, or we can use empirically-found scaling rules.

Existing scale-control mechanisms

We're already familiar with two mechanisms of this kind:

Glorot initialization [3]

$$\sigma_w = \sqrt{\frac{2}{m+n}}$$

(an average of the ideal y and ∇x scales)

Scaled dot-product attention [4]

$$\text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Unit Scaling poses the question: what if we put them throughout the model?

[3] Glorot, X., and Bengio, Y. *"Understanding the difficulty of training deep feedforward neural networks."* AISTATS, 2010.

[4] Vaswani, A., Shazeer, N.M., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., & Polosukhin, I. *"Attention is All you Need."* NeurIPS, 2017.

Contents

A version of μP that (almost) satisfies Unit Scaling

All models under the ABC parametrization are subject to *abc-symmetry:*

For a fixed $\theta > 0$, training under Adam(W) is invariant to changes of the kind:

$$A_W = A_W \,/\, \theta$$

$$B_W = B_W \cdot \theta$$

$$C_W = C_W \cdot \theta$$

	ABC-multiplier		Weight (W) Type Input	Hidden	Output
μP	parameter	(a_W)	1	1	$1/\text{fan-in}(W)$
	initialization	(b_W)	1	$1/\sqrt{\text{fan-in}(W)}$	1
	Adam LR	(c_W)	1	$1/\text{fan-in}(W)$	1

$$\theta = \sqrt{\text{fan–in}} \downarrow$$

	ABC-multiplier		Weight (W) Type Input	Hidden	Output
u-μP	parameter	(a_W)	1	$1/\sqrt{\text{fan-in}(W)}$	$1/\text{fan-in}(W)$
	initialization	(b_W)	1	1	1
	Adam LR	(c_W)	1	$1/\sqrt{\text{fan-in}(W)}$	1

The u-μP scheme

	ABC-multiplier		Weight Type		
			Input	Hidden	Output
μP	parameter	(A_W)	α_{emb}	1 (or α_{attn})	$\alpha_{\text{out}} \frac{\text{base-fan-in}}{\text{fan-in}}$
	initialization	(B_W)	σ_{init}	$\sigma_{\text{init}} \sqrt{\frac{\text{base-fan-in}}{\text{fan-in}}}$	σ_{init}
	Adam LR	(C_W)	$\eta \, \hat{\eta}_{\text{emb}}$	$\eta \, \frac{\text{base-fan-in}}{\text{fan-in}}$	η
u-μP	parameter†	(A_W)	1	$\frac{1}{\sqrt{\text{fan-in}}}$	$\frac{1}{\text{fan-in}}$ *
	initialization	(B_W)	1	1	1
	Adam LR	(C_W)	$\eta \, \frac{1}{\sqrt{\text{fan-out}}}$	$\eta \, \frac{1}{\sqrt{\text{fan-in}}}$	η

†u-μP's α HPs are associated with operations, not weights, so are not included here

* We apply $1/\sqrt{\text{fan}-\text{in}}$ scaling in the bwd pass

Recalling: $A_w \leftarrow \alpha_w \frac{a_W}{a_{W_{base}}}$ $\quad B_w \leftarrow \sigma_w \frac{b_W}{b_{W_{base}}}$ $\quad C_w \leftarrow \eta_w \frac{c_W}{c_{W_{base}}}$

A principled set of HPs

Previously we showed the "full" set of 3 HPs (α, σ, η) for each weight tensor.

We reduce this in a principled way by:

1. Dropping σ_{init} via abc-symmetry
2. Pushing α_s from weights into subsequent ops
3. Using one global η

For a transformer this gives:

SP	μP	u-μP
$\boldsymbol{\eta}$	$\boldsymbol{\eta}$	$\boldsymbol{\eta}$
σ-scheme	$\boldsymbol{\sigma_{\text{init}}}$	
	$\boldsymbol{\alpha_{\text{emb}}} \vert \boldsymbol{\eta_{\text{emb}}}$	$\alpha_{\text{ffn-act}}$
	α_{attn}	$\alpha_{\text{attn-softmax}}$
	α_{out}	α_{res}
	base-width	$\alpha_{\text{res-attn-ratio}}$
	base-depth	$\alpha_{\text{loss-softmax}}$

Contents

Experiments

HP interdependence

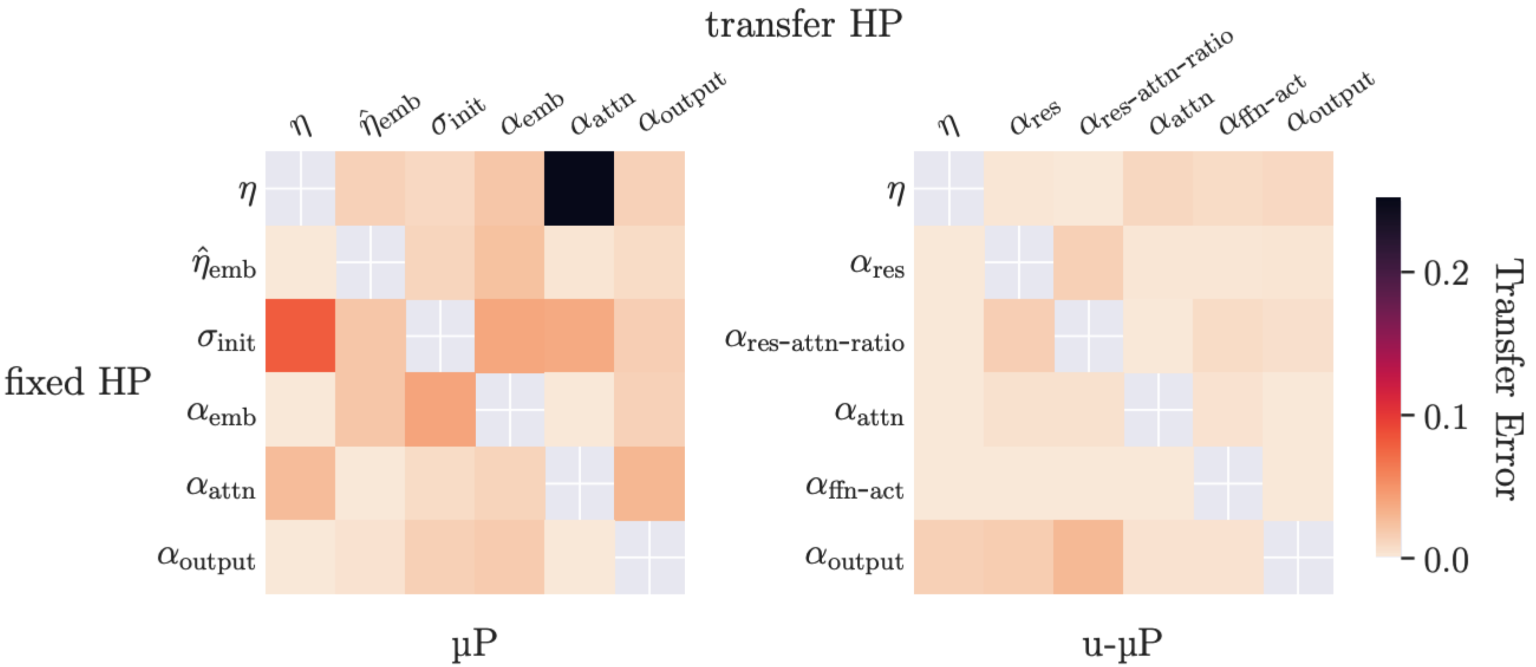


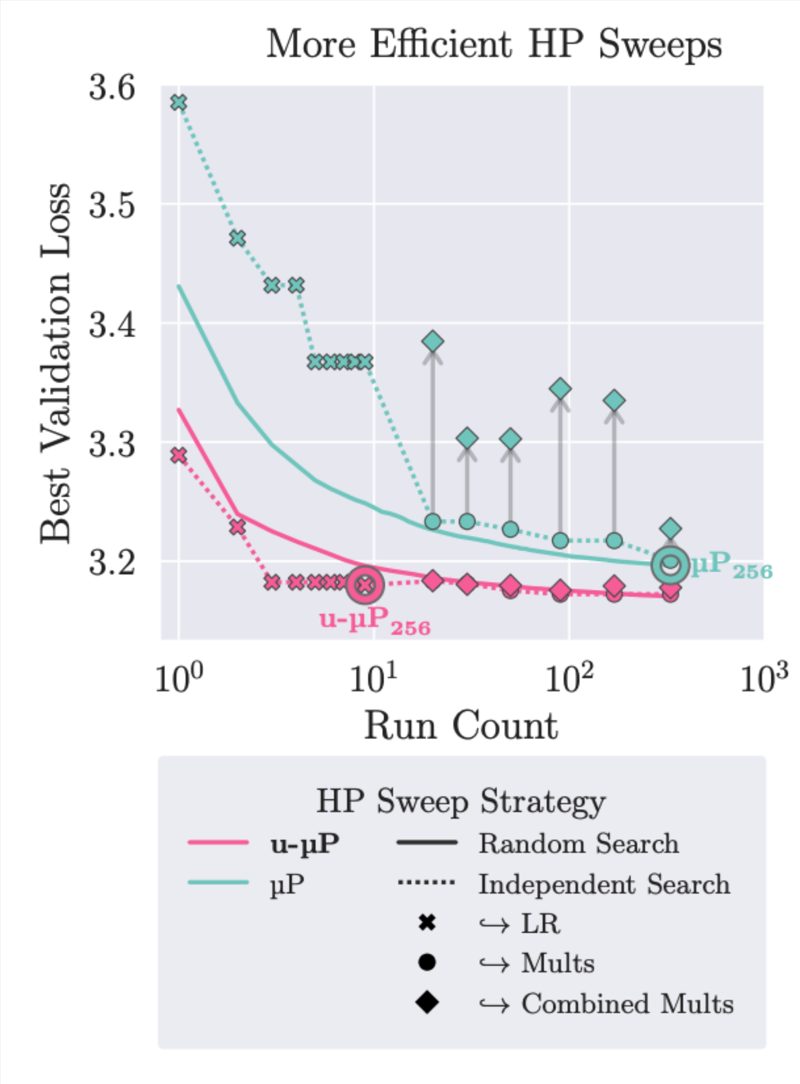
Figure 4: A visualization of the dependencies between pairs of HPs under each scheme. Transfer error measures the extent to which the optimal value of the transfer HP depends on the fixed HP (see Algorithm 1). On average, μP has a transfer error of 0.03, whereas u-μP has 0.005.

HP search

Independent search:

1. 1D line search to find optimal η. Other HPs are set to their default values (i.e. 1)
2. For all other HPs, in parallel perform a 1D line search (using best η).
3. Combine the best HPs from step 2.

For u-µP just sweeping η gives near-optimal loss.



Our FP8 Scheme

We only focus on FP8 in linear layer matmuls. Communication, optimizer states and other operations are left for future work.

We use **E4M3** for activations and weights and **E5M2** for gradients, except:

1. The final FFN down projection
2. The final SA dense projection

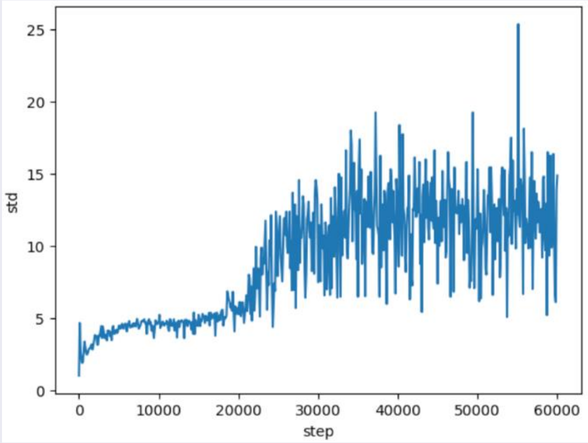
For which we use **BF16**.

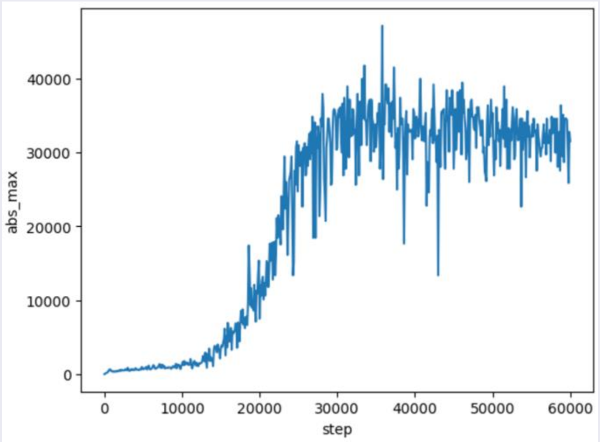
70% of our matmul flops are in FP8

Numerical insights I

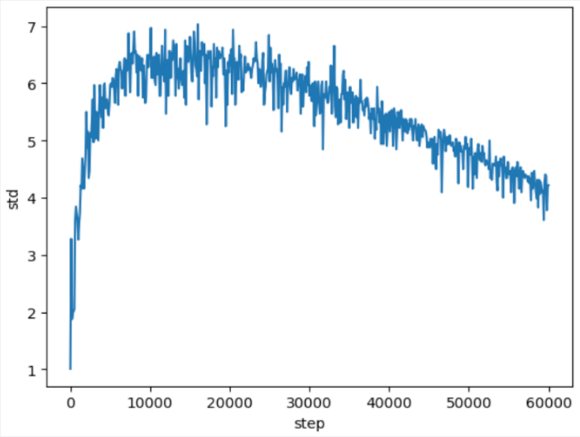
In some layers the scale of the input tensor to the critical matmul suddenly increases after stagnating for a certain number of steps

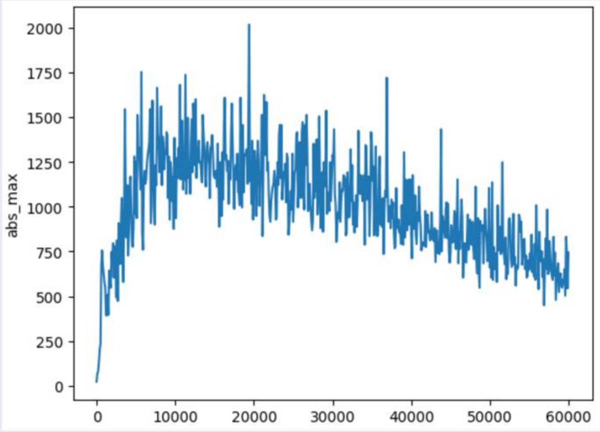
Layer 3





Layer 6





Numerical insights II

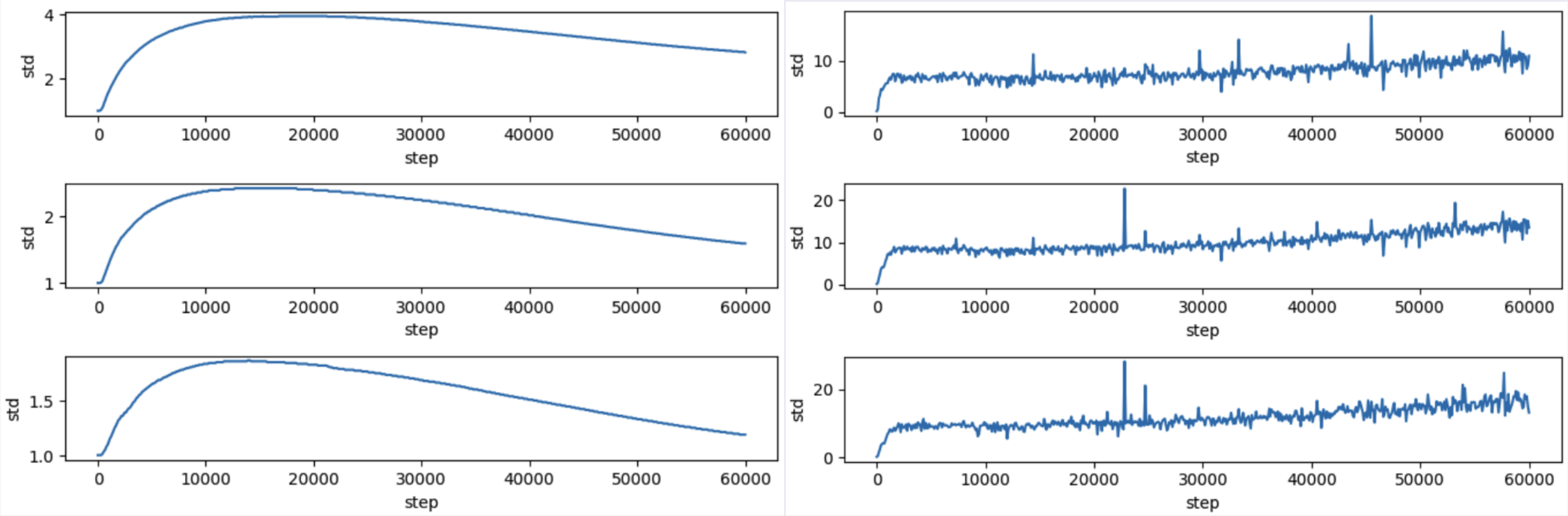
For non-critical matmuls, the numerical scales are relatively stable across model size.

Example: FFN up projection

1B

3B

7B



weight

grad_y

Large-scale u-μP FP8 results

Full 1-7B results (standard Llama architecture; 300B tokens of SlimPajama)

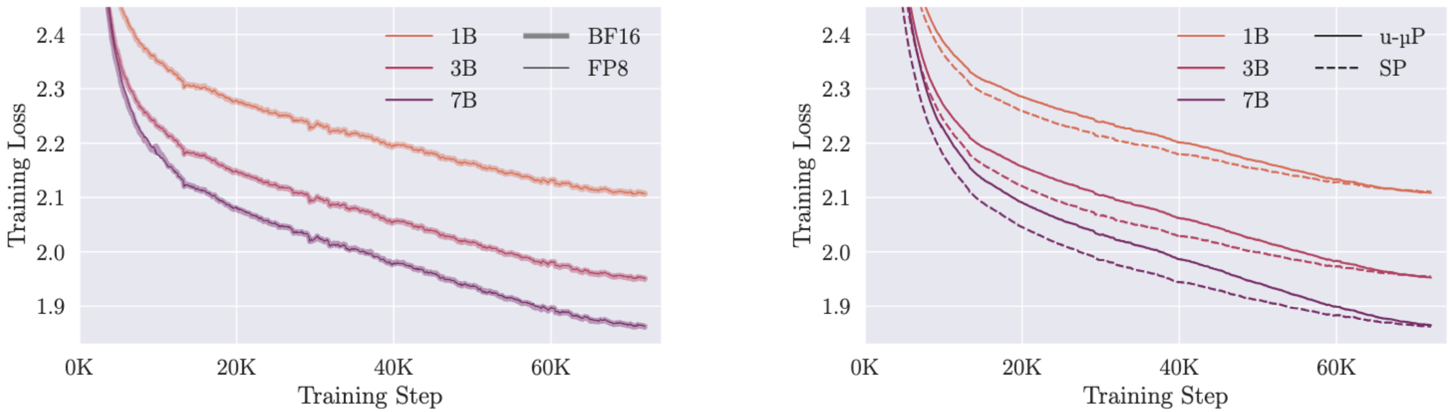


Figure 6: Large-scale training runs. (Left) u-μP BF16 vs u-μP FP8. (Right) u-μP BF16 vs SP BF16.

Table 4: 0-shot benchmark results at 7B scale.

Scheme	Format	MMLU	HellaSwag	OpenBook QA	PIQA	TriviaQA	WinoGr
SP	BF16	29.6	52.4	27.8	76.5	22.2	63.3
u-μP	BF16	29.0	**53.4**	**31.6**	77.1	**23.4**	63.7
u-μP	FP8	**31.2**	**53.4**	29.6	**77.6**	21.3	**65.7**

Contents

Conclusions

1. There is a version of µP which is compatible with Unit Scaling
2. … this leverages µP's feature stability for low-precision training
3. An embedding LR rule of $1/\sqrt{\text{fan-out}}$ is better than µP's 1 rule
4. The standard µP HPs are sub-optimal & over-complex
5. A more independent set of HPs gives more efficient search

If you want all of the above, use u-µP!

https://github.com/graphcore-research/unit-scaling
(library for u-µP)

https://github.com/Aleph-Alpha/scaling
(codebase for large-scale training)

Extra Slides

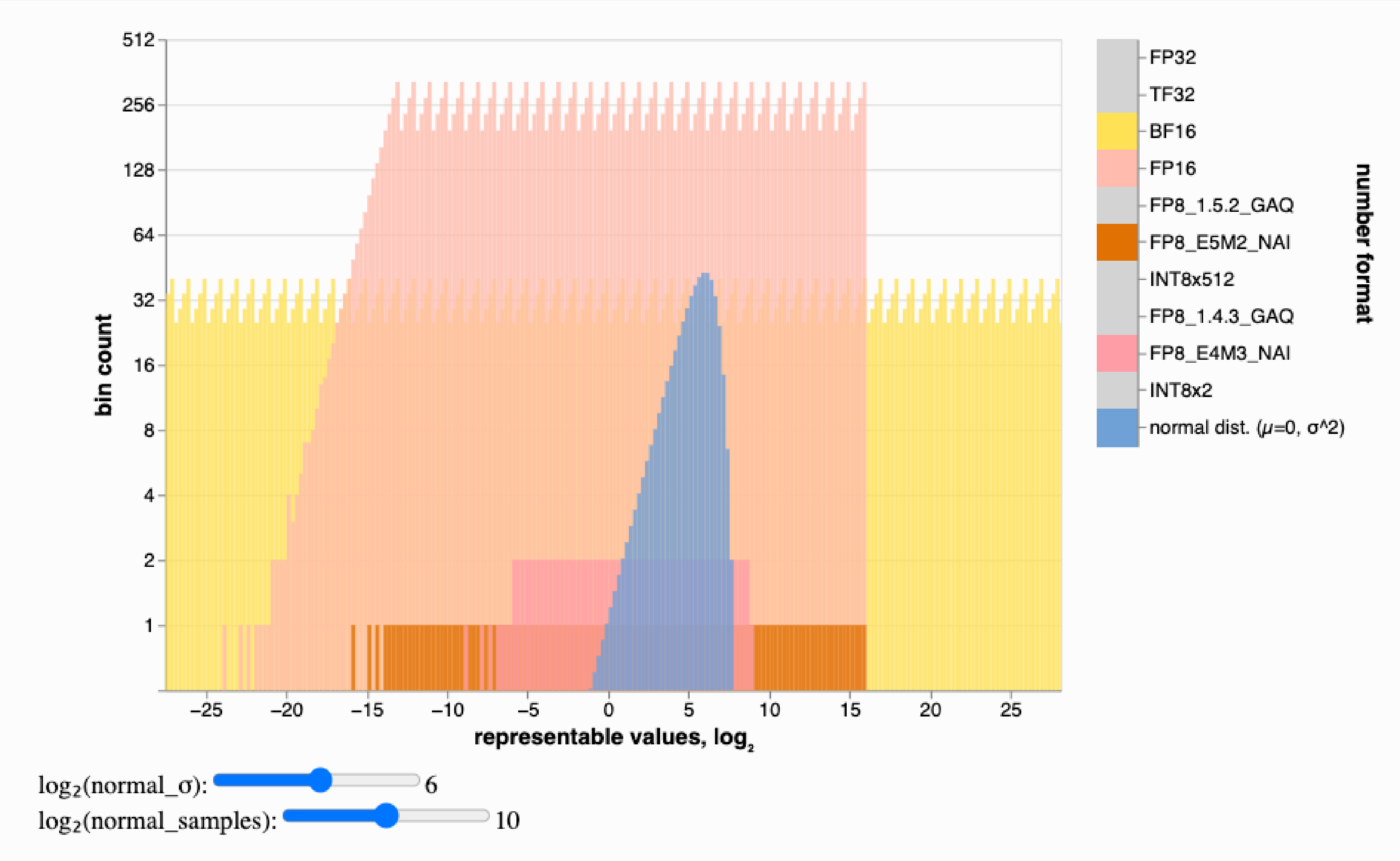
Contents

Floating-point formats

Values represented by
sign / exponent / mantissa bit-strings

$$
\begin{aligned}
\text{sign} &= (-1)^{b_{\text{sign}}}, \\
\text{exponent} &= b_{\text{exp}} - \text{bias}, \quad (\text{bias} = 2^{E-1} - 1) \\
\text{mantissa} &= 1 + \frac{b_{\text{mant}}}{2^M}, \\
\text{value} &= \text{sign} \times 2^{\text{exponent}} \times \text{mantissa}
\end{aligned}
$$

Floating-point formats



source: https://thecharlieblake.co.uk/visualising-ml-number-formats

µP and low-precision

[1] Yang, G., et al. *"Tensor Programs V: Tuning large neural networks via zero-shot hyperparameter transfer."* arXiv:2203.03466, 2022.

[1]

Desiderata J.1. At any time during training

1. Every (pre)activation vector in a network should have $\Theta(1)$-sized coordinates[32]
2. Neural network output should be $O(1)$.
3. All parameters should be updated as much as possible (in terms of scaling in width) without leading to divergence.

Let's briefly justify these desiderata. For the desideratum 1, if the coordinates are $\omega(1)$ or $o(1)$, then for sufficiently wide networks their values will go out of floating point range.

[1]

during training of the μTransfer 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.

Methods for controlling scale

Method	Fine-grained scaling	No tuning required	Adapts during training	Pass	Complexity
Loss scaling	✗	✗	✗	bwd	low
Dynamic loss scaling	✗	✓	✓	bwd	medium
Transformer engine scaling	✓	~	✓	bwd + fwd	medium/high
Unit scaling	✓	✓	✗	bwd + fwd	low

μP as a form of Unit Scaling

All models under the ABC parametrization are subject to *abc-symmetry:*

For a fixed $\theta > 0$, training under Adam(W) is invariant to changes of the kind:

$$A_w = A_w \,/\, \theta$$
$$B_w = B_w \cdot \theta$$
$$C_w = C_w \cdot \theta$$

Setting $\theta = \sqrt{\text{fan-in}}$ for hidden weights results in Unit Scaling:

	ABC-multiplier		Weight (W) Type Input	Hidden	Output
μP	parameter	(a_w)	1	1	$1/\text{fan-in}(W)$
	initialization	(b_w)	1	$1/\sqrt{\text{fan-in}(W)}$	1
	Adam LR	(c_w)	1	$1/\text{fan-in}(W)$	1

	ABC-multiplier		Weight (W) Type Input	Hidden	Output
u-μP	parameter	(a_w)	1	$1/\sqrt{\text{fan-in}(W)}$	$1/\text{fan-in}(W)$ *
	initialization	(b_w)	1	1	1
	Adam LR	(c_w)	1	$1/\sqrt{\text{fan-in}(W)}$	1

* We apply $1/\sqrt{\text{fan-in}}$ scaling in the bwd pass

An improved HP scheme

	ABC-multiplier		Weight Type			Residual
			Input	Hidden	Output	
μP	parameter	(A_W)	α_{emb}	1 (or α_{attn})	$\alpha_{\text{out}} \frac{\text{base-fan-in}}{\text{fan-in}}$	$\sqrt{\frac{\text{base-depth}}{\text{depth}}}$
	initialization	(B_W)	σ_{init}	$\sigma_{\text{init}} \sqrt{\frac{\text{base-fan-in}}{\text{fan-in}}}$	σ_{init}	—
	Adam LR	(C_W)	$\eta\, \hat{\eta}_{\text{emb}}$	$\eta\, \frac{\text{base-fan-in}}{\text{fan-in}}$	η	$\sqrt{\frac{\text{base-depth}}{\text{depth}}}$
u-μP	parameter†	(A_W)	1	$\frac{1}{\sqrt{\text{fan-in}}}$	$\frac{1}{\text{fan-in}}$	$\frac{1}{\sqrt{\text{depth}}}$
	initialization	(B_W)	1	1	1	—
	Adam LR	(C_W)	$\eta\, \frac{1}{\sqrt{\text{fan-out}}}$	$\eta\, \frac{1}{\sqrt{\text{fan-in}}}$	η	$\frac{1}{\sqrt{\text{depth}}}$

†u-μP's α HPs are associated with operations, not weights, so are not included here

<u>The u-μP HP scheme</u>

- No σ_{init}
- No base shapes
- αs moved from weight to ops

Recalling: $A_w \leftarrow \alpha_w \frac{a_W}{a_{W_{base}}}$ $\quad B_w \leftarrow \sigma_w \frac{b_W}{b_{W_{base}}}$ $\quad C_w \leftarrow \eta_w \frac{c_W}{c_{W_{base}}}$

Numerical Properties

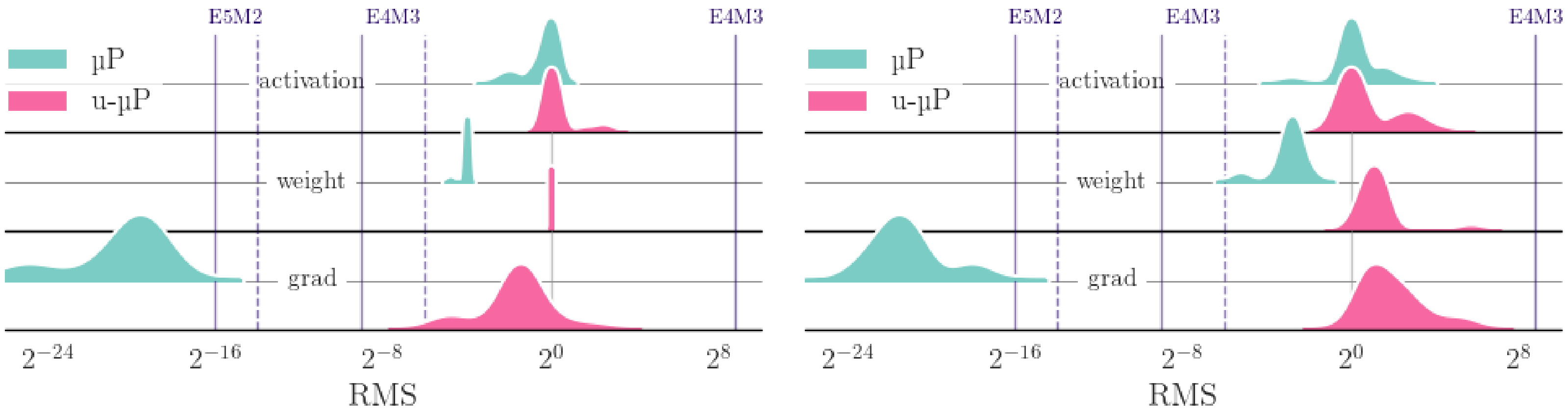


Figure 6: Per-tensor RMS $= \sqrt{\sigma^2 + \mu^2}$ across u-µP and µP models at initialization (left) and after training (right). u-µP tensors have RMS that starts close to 1 and remains within E4M3 range at the end of training. Dashed and solid red lines show each format's min. normal and subnormal values.

Our FP8 Scheme

We only focus on FP8 matmuls. Putting comms / optimizer state / nonlinearities into FP8 is out-of-scope.

We use **E4M3** for everything (activations, weights *and gradients*), except:

1. The input to the final FFN layer (fwd pass)
2. The input to the final SA layer (fwd pass)

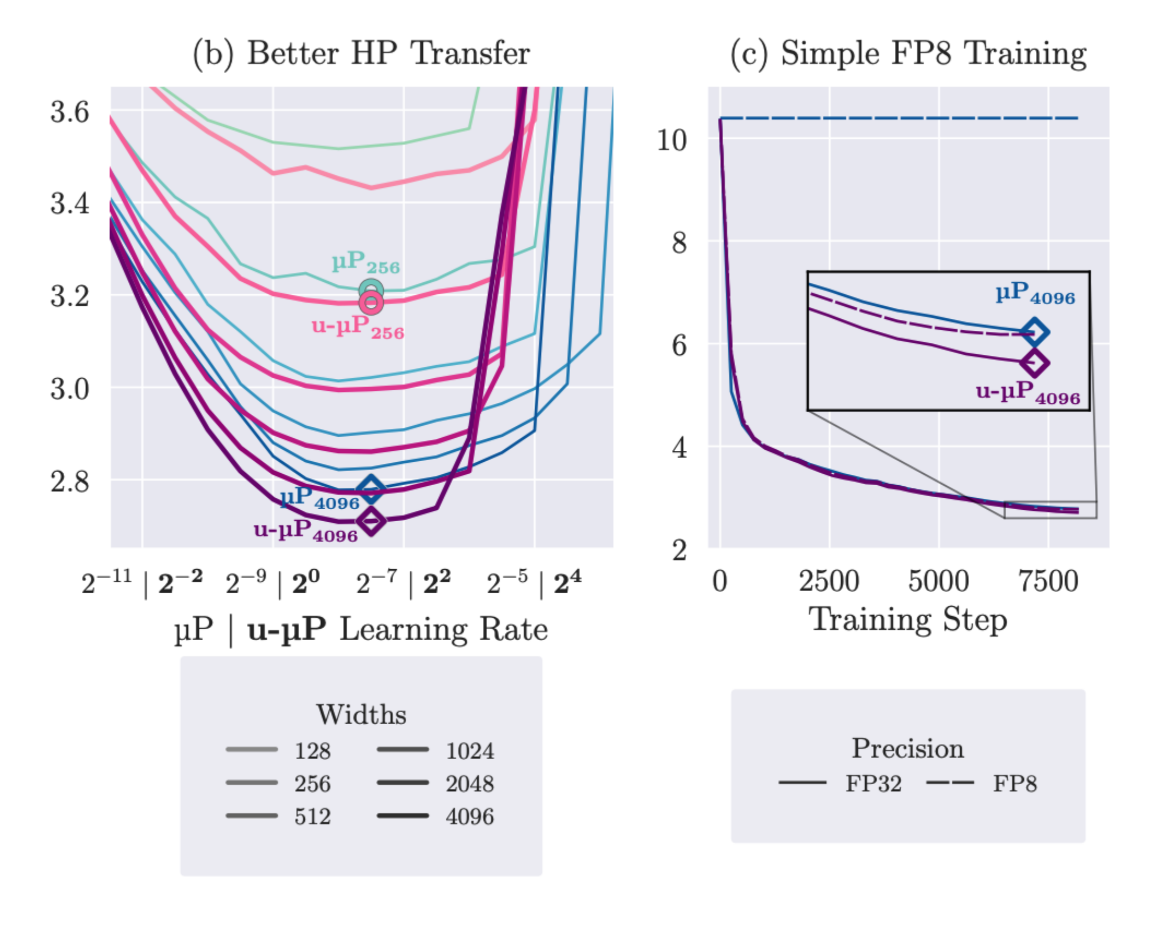
For which we use **E5M2**.

A note on the status of our experiments:

- Large-scale experimentation is still ongoing
- Particularly in the case of downstream analysis

Out-of-the-box FP8

Llama up to 1B (u-µP; WikiText-103)



Standard BERT (original Unit Scaling)

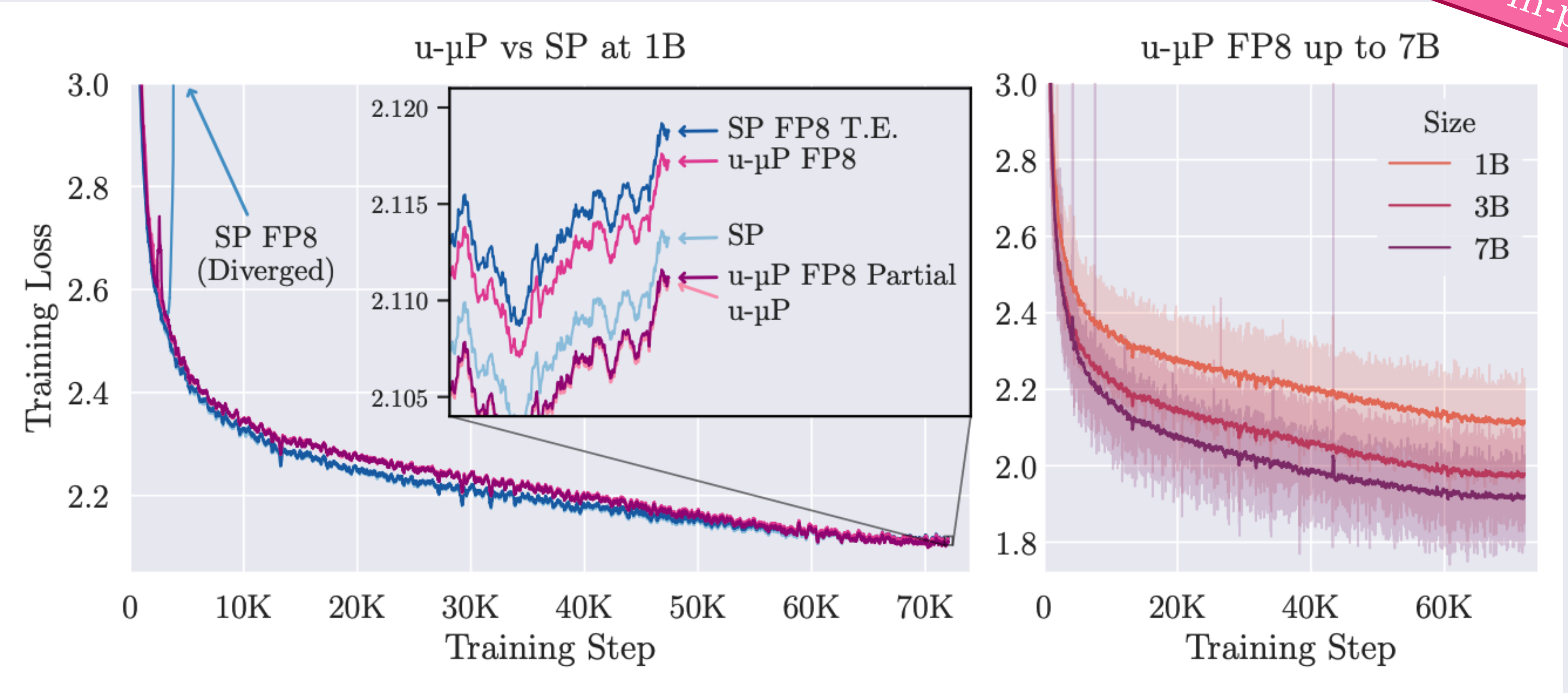
Model	Method	Precision	SQuAD v2.0 EM	SQuAD v2.0 F1
Base	No Scaling †	FP32	—	—
	Loss Scaling	FP16	73.36 (±0.27)	76.47 (±0.23)
	Unit Scaling	FP16	72.31 (±0.60)	75.70 (±0.53)
	Unit Scaling	FP8	72.28 (±0.02)	75.67 (±0.01)
Large	No Scaling †	FP32	78.7	81.9
	Loss Scaling	FP16	77.52 (±0.63)	80.54 (±0.61)
	Loss Scaling ‡	FP8	—	—
	Unit Scaling	FP16	79.94 (±0.10)	82.97 (±0.09)
	Unit Scaling	FP8	79.29 (±0.31)	82.29 (±0.29)

Experiments

Out-of-the-box FP8

Full 1-7B results (standard Llama setup; 300B tokens of SlimPajama)

Work-in-progress



- *Partial* scheme puts two layers that were in E5, in BF16.
- Gradients also now have to be in E5.

Example: uu.Linear

github.com/graphcore-research/unit-scaling

We use a different scaling factor for each of the three matmuls.

Under certain circumstances we require the same factor for y and ∇x, to ensure valid gradients.

More interesting unit-scaled ops include:

- Residual-add
- Scaled dot-product attention

```python
import torch; from torch import nn
from unit_scaling.scale import scale_bwd, scale_fwd

class Linear(nn.Linear):
    def __init__(self, fan_in, fan_out, bias=False):
        super().__init__(fan_in, fan_out, bias)

    def reset_parameters(self) -> None:
        nn.init.normal_(self.weight, 0, 1)

    def forward(self, x):
        fan_out, fan_in = self.weight.shape
        batch_size = x.numel() // fan_in

        x = scale_bwd(x, fan_out**-0.5)
        weight = scale_bwd(self.weight, batch_size**-0.5)
        output = nn.functional.linear(x, weight)
        return scale_fwd(output, fan_in**-0.5)

b, d = 2**12, 2**10
x = torch.randn(b, d).requires_grad_()
linear = Linear(d, 4 * d)  # regular: nn.Linear(d, 4 * d)
y = linear(x)
y.sum().backward()
(                                 # unit-scaled | regular
    y.std(),                      # 1.0         | 0.57
    linear.weight.std(),          # 1.0         | 0.18
    x.grad.std(),                 # 1.0         | 1.17
    linear.weight.grad.std(),     # 1.0         | 64.0
)
```

Unit-Scaled gradients

Why are we allowed to have separate forward and backward scaling factors?

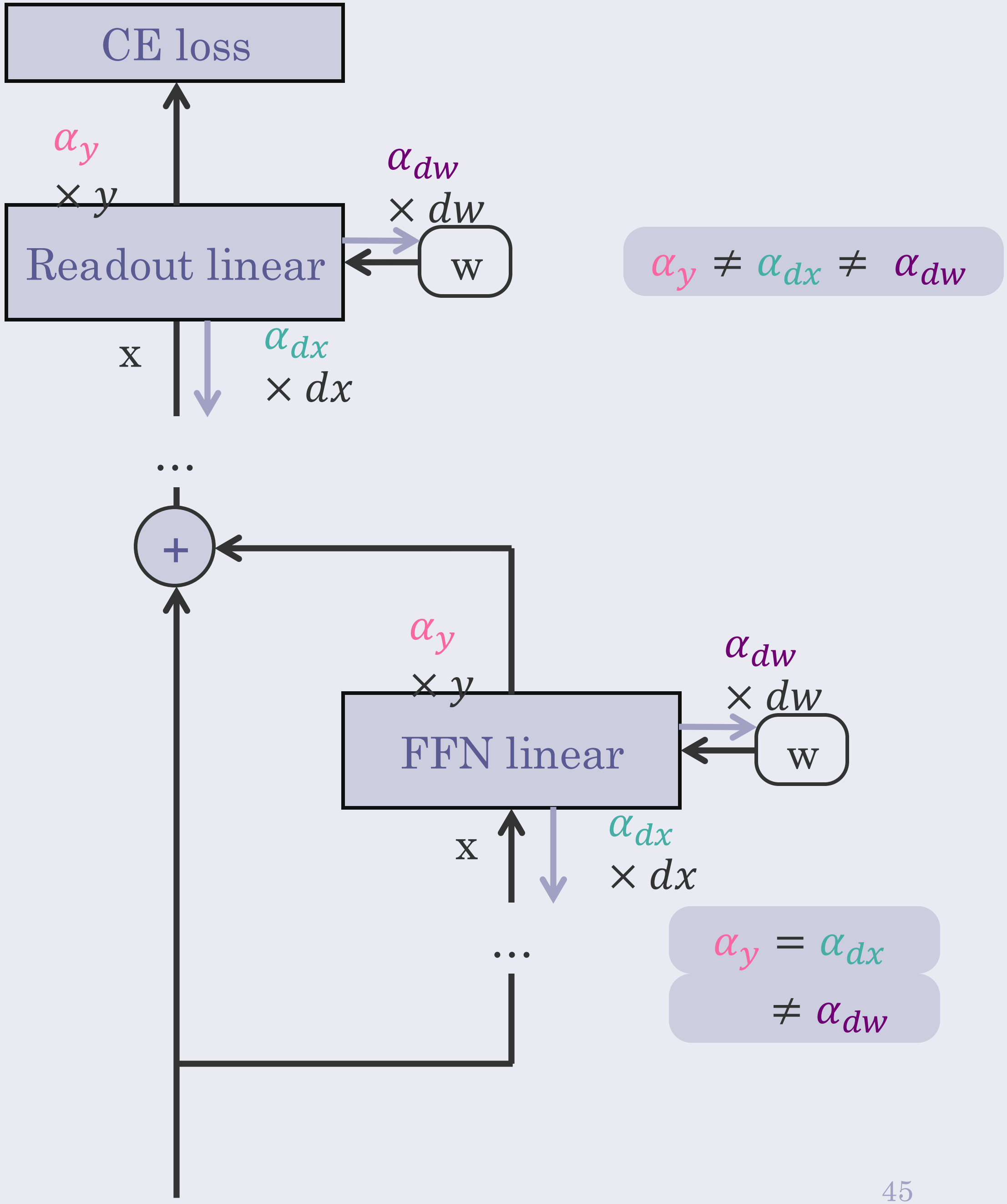
Fixed scales cancel in the Adam update (ignoring ϵ):

$\times \alpha$

$$\theta_{t+1} = \theta_t - \eta \left(\frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} \right)$$

$\times \alpha^2$

So we can use different scales, as long as we're not on a residual



How has the model changed?

1. The scale going into non-linear operations has changed

2. Our weight update has effectively become smaller

This term hasn't changed scale

$$\theta_{t+1} = \theta_t - \eta\left(\frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}\right)$$

But we now use larger ($\sigma = 1$) weights

Which HPs should be swept?

µTransfer: the process of sweeping HPs on a small model, to transfer to a large one.

How should we select/group our α_W, σ_W, η_W in order to sweep them?

A global σ and η are commonly used. However, this creates problems:

Grouping-conflicts:

$\mathrm{std}(x_{swish}) \propto \sigma_{W_{gate}}$ $\quad$ $\mathrm{std}(x_{attn}) \propto \sigma_{W_Q}\sigma_{W_K}$

HP interdependence:

the relative size of a weight update is determined by σ_W/η_W

Table 7: Sweeping configurations used for a selection of µP models from the literature. The sweeping process is similar across models, the only differences being the choice of discrete or continuous distributions and their ranges.

Model	proxy/target tokens used	proxy/target model size	sweep size	base width	HPs swept
T.P.V WMT14 [2]	100%	7.1%	64		$\eta, \alpha_{out}, \alpha_{attn}$
T.P.V BERT_{large} [2]	10%	3.7%	256	?	$\eta, \eta_{emb}, \alpha_{out}, \alpha_{attn}, \alpha_{LN}, \alpha_{bias}$
T.P.V GPT-3 [2]	1.3%	0.6%	350		$\eta, \sigma, \alpha_{emb}, \alpha_{out}, \alpha_{attn}, \alpha_{pos}$
MiniCPM [6]	0.008%	0.45%	400	256	$\eta, \sigma, \alpha_{emb}, \alpha_{residual}$
Cerebras-GPT [3]	1.1%	1.5%	200	256	$\eta, \sigma, \alpha_{emb}$
SµPar [63]	6.6%	6.4%	350	256	$\eta, \sigma, \alpha_{emb}$

Base shapes

```python
import mup

proxy_model = MupModel(d_model=128, ...)       # proxy width
base_model = MupModel(d_model=256, ...)        # base width
mup.set_base_shapes(proxy_model, base_model)   # re-initialize proxy_model
```

Recall: $$A_W \leftarrow \alpha_W \frac{a_W}{a_{W_{\text{base}}}}, \quad B_W \leftarrow \sigma_W \frac{b_W}{b_{W_{\text{base}}}}, \quad C_W \leftarrow \eta_W \frac{c_W}{c_{W_{\text{base}}}}$$

Existing approaches to σ_{init}

"Overall scheme"	Per-tensor schemes	Users
Constant	constant init	OpenLLaMA; Deepseek
Megatron	constant init + depth-residual init	GPT-2; GPT-3; Megatron-LM; OPT; Cerebras-GPT; OLMo-Llama
GPT-neo-X	small init + wang init	GPT-neo-X; TinyLlama; Pythia; StableLM; Luminous
PaLM	LeCun init + palm-readout	PaLM
Mitchell	mitchell-scale	OLMo

Scheme	Rule
Constant	Scale weights by constant std
LeCun	Scale weights by $1/\sqrt{d\_in}$
Glorot	Scale weights by $2/\sqrt{(d\_in + d\_out)}$
Small	Scale all weights by $2/\sqrt{(5d)}$
Mitchell-scale	Scale weights by $1/\sqrt{(2Ld)}$
Depth-residual	Scale residual by $1/\sqrt{(2L)}$
Wang	Scale residual by $2/(L\sqrt{d})$
Palm-readout	Scale embedding proj by $1/\sqrt{v}$

Combining µP with Unit Scaling

Basic µP scaling rules

	ABC-multiplier		Weight (W) Type		
			Input	Hidden	Output
µP	parameter	(a_W)	1	1	$^1/_{\text{fan-in}(W)}$
	initialization	(b_W)	1	$^1/\sqrt{\text{fan-in}(W)}$	1
	Adam LR	(c_W)	1	$^1/_{\text{fan-in}(W)}$	1

Full µP implementation

	ABC-multiplier		Weight Type			Residual
			Input	Hidden	Output	
µP	parameter	(A_W)	α_{emb}	1 (or α_{attn})	$\alpha_{\text{out}} \frac{\text{base-fan-in}}{\text{fan-in}}$	$\sqrt{\frac{\text{base-depth}}{\text{depth}}}$*
	initialization	(B_W)	σ_{init}	$\sigma_{\text{init}} \sqrt{\frac{\text{base-fan-in}}{\text{fan-in}}}$	σ_{init}	—
	Adam LR	(C_W)	$\eta \, \hat{\eta}_{\text{emb}}$	$\eta \, \frac{\text{base-fan-in}}{\text{fan-in}}$	η	$\sqrt{\frac{\text{base-depth}}{\text{depth}}}$

*Residual multipliers are applied to the end of each branch, rather than the output of linear layers.

Recall:

$$A_W \leftarrow \alpha_W \frac{a_W}{a_{W_{\text{base}}}},$$

$$B_W \leftarrow \sigma_W \frac{b_W}{b_{W_{\text{base}}}},$$

$$C_W \leftarrow \eta_W \frac{c_W}{c_{W_{\text{base}}}}$$

Weight Type

Residual Residual

A principled approach to HPs

Table 3: Typical transformer HPs used under different schemes. *Basic* HPs in **bold** are considered most impactful and are commonly swept. *Extended* HPs in non-bold are not always swept, often set heuristically or dropped.

SP	μP	u-μP
$\boldsymbol{\eta}$	$\boldsymbol{\eta}$	$\boldsymbol{\eta}$
σ-scheme	$\boldsymbol{\sigma_{\text{init}}}$	
	$\boldsymbol{\alpha_{\text{emb}}}\vert\boldsymbol{\eta_{\text{emb}}}$	$\alpha_{\text{ffn-act}}$
	α_{attn}	$\alpha_{\text{attn-softmax}}$
	α_{out}	α_{res}
	base-width	$\alpha_{\text{res-attn-ratio}}$
	base-depth	$\alpha_{\text{loss-softmax}}$

Deriving the u-μP HP scheme:

1. We drop σ_Ws entirely: only need α_W, η_W under abc-symmetry
2. Global η, granular α_W grouped across layers
3. Shift αs from weights to ops
4. Special residual-add formulation, which
    - Gives unit-scale at initialization
    - α_{res} defines contribution of the residual vs skip scale
    - $\alpha_{res-attn-ratio}$ defines the relative contribution of attention versus FFN branches

Experiments

Numerical Properties

Regular µP

u-µP

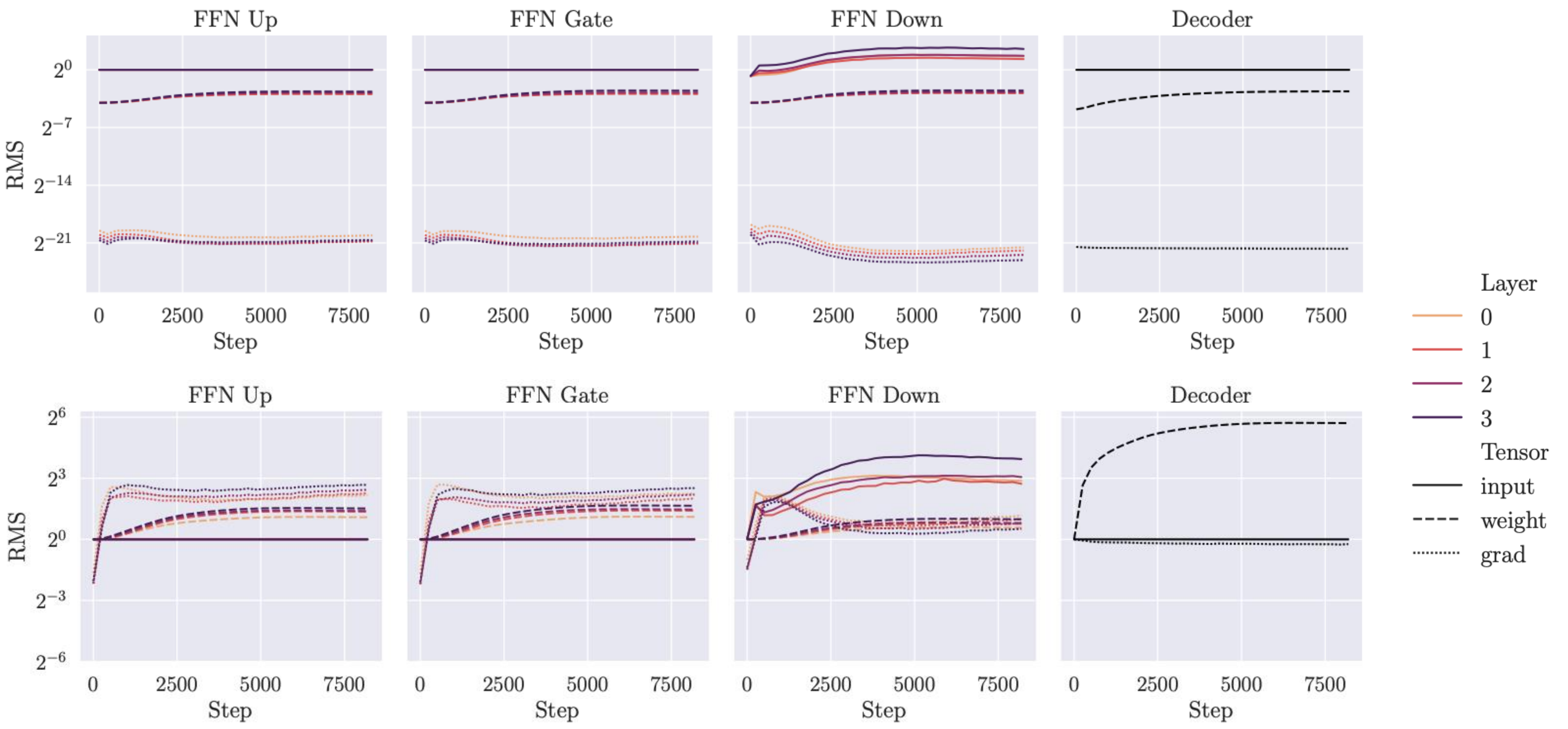
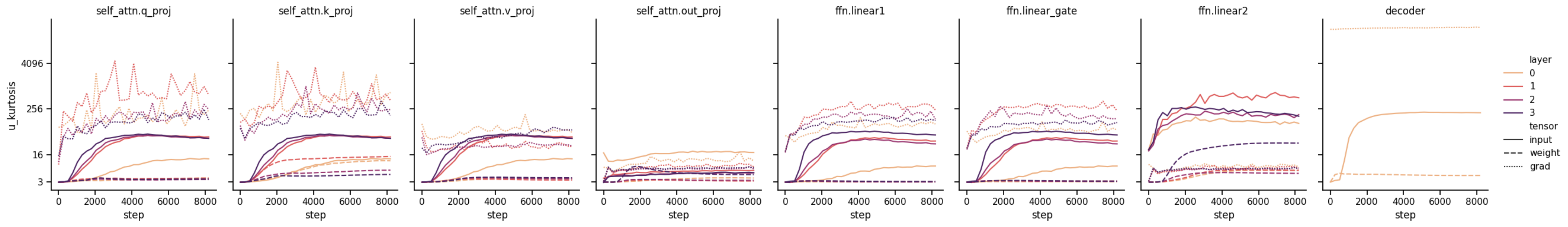


Figure 13: RMS during training, for all parametrized matmul inputs, for µP (top) and u-µP (bottom). Model width 256, default hyperparameters, $\eta = (2^1, 2^{-8})$ for (u-µP, µP).

Heavy-tail analysis



Factors affecting scale growth

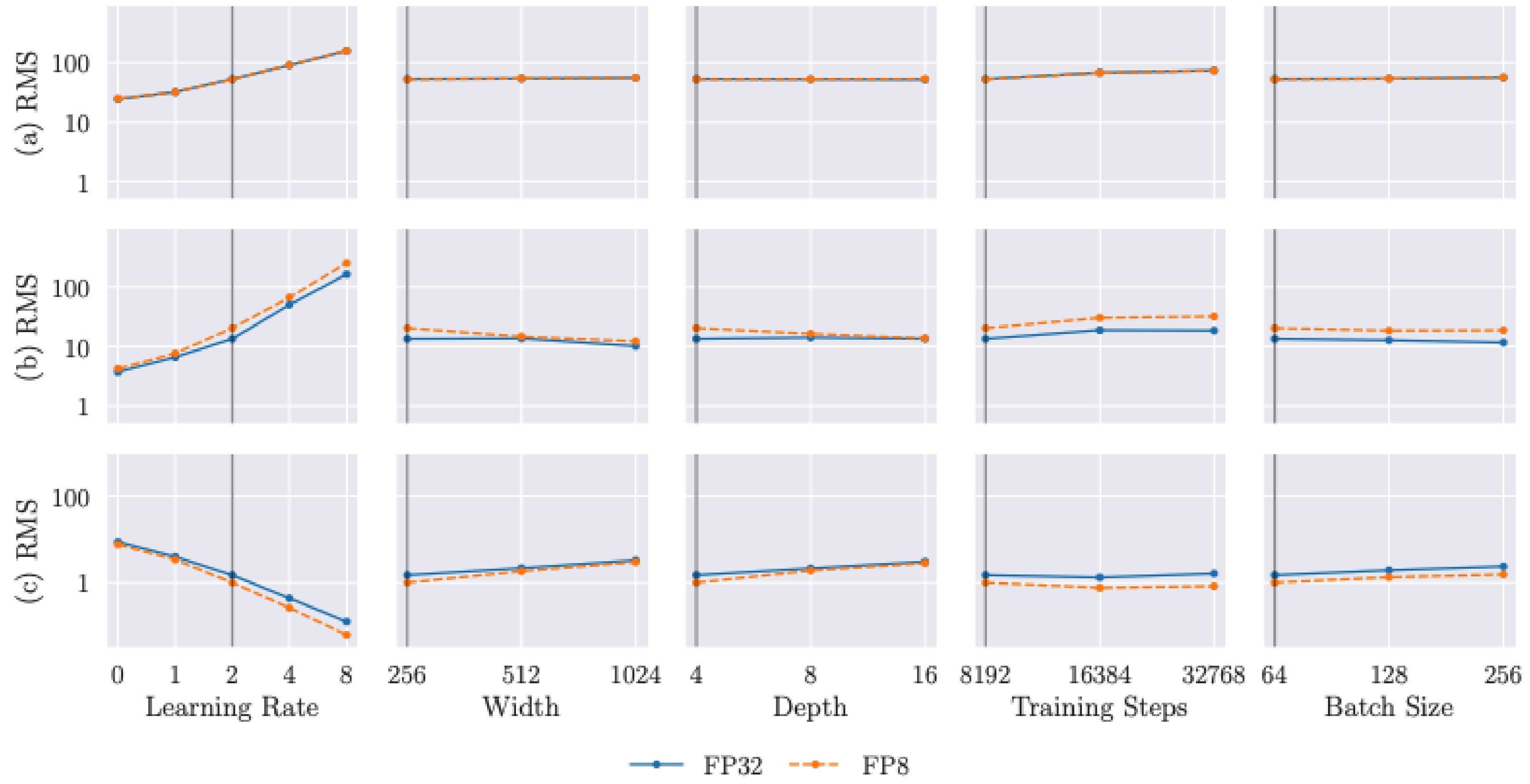


Figure 14: The effect of hyperparameters on FP8 training loss and on the end-training RMS of various tensors: (a) decoder weight, (b) last-layer FFN down-projection input and (c) last-layer FFN down-projection output gradient. Only learning rate has a substantial effect on the end-training RMS. Vertical lines show the default setting of that hyperparameter, as used for all other plots.

Takeaway:

- The only factor significantly affecting scale-growth is LR