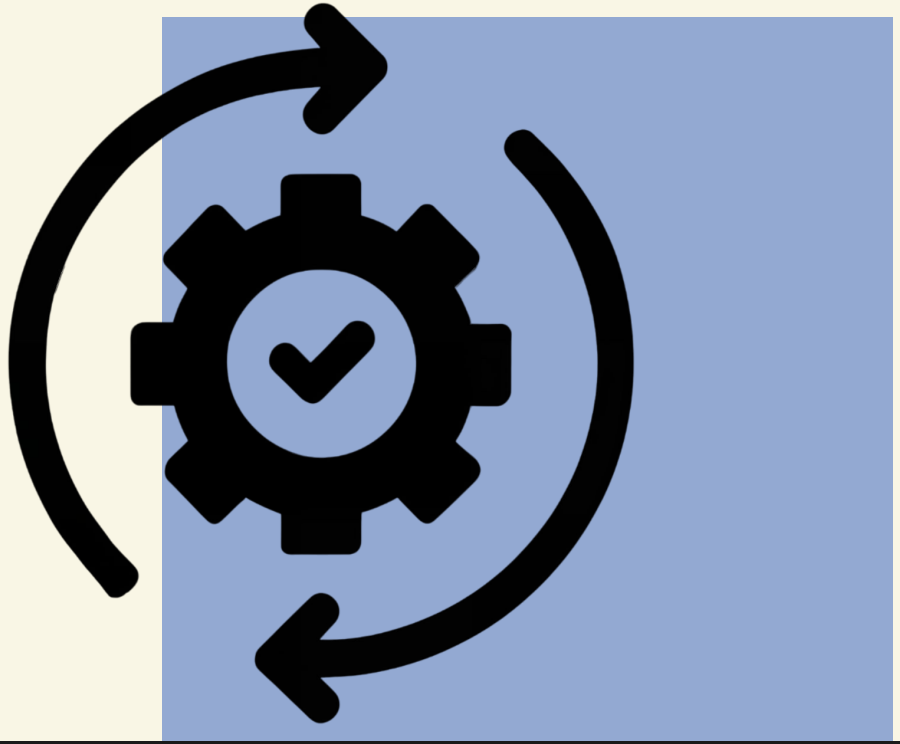
Эффективные системы машинного обучения



Лекция 3

“PEFT & NAS”

Преподаватель

Феоктистов Дмитрий

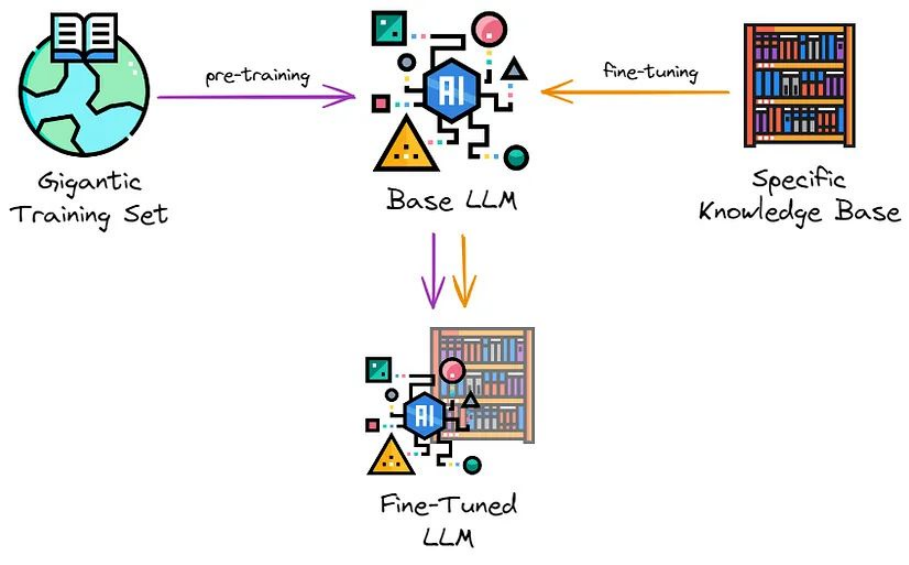
11 октября 2024

ВМК МГУ

PEFT

1. PEFT
2. LoRA
3. DoRA
4. Fast Forwarding
5. QLoRA, QDoRA

Мотивация



Промты – сложно; FT – просто

PEFT

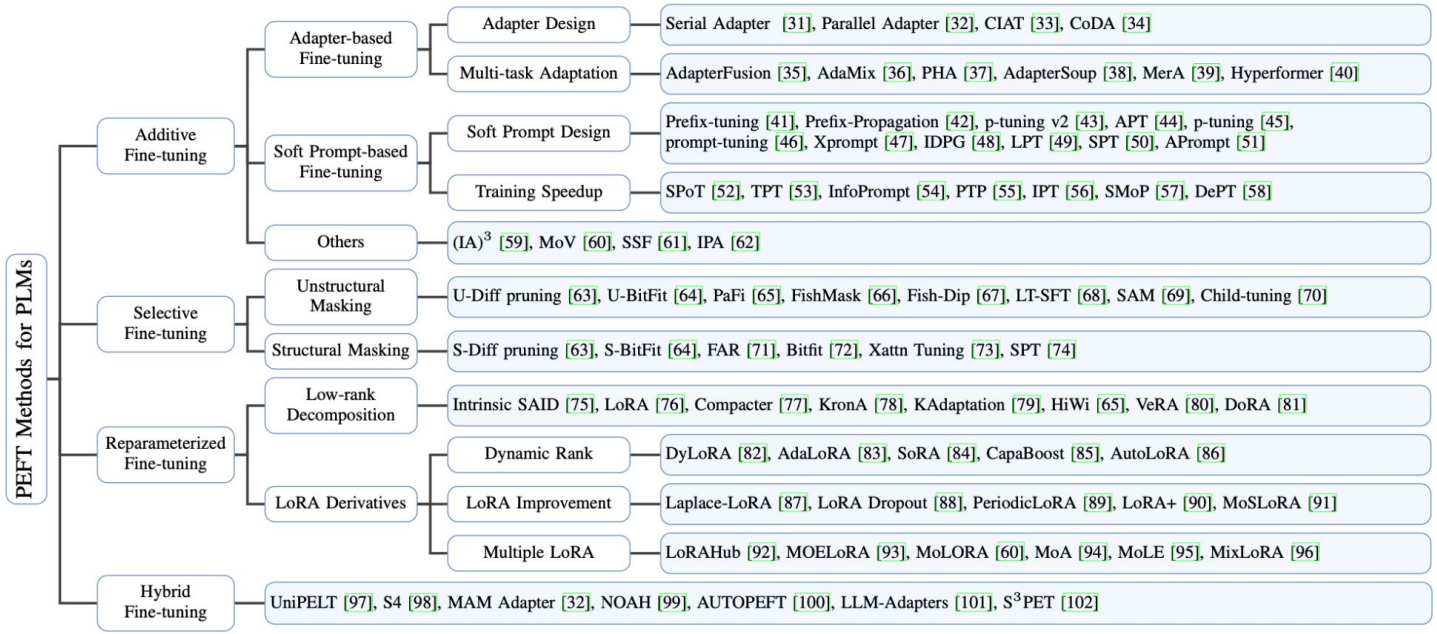


Fig. 3: Taxonomy of Parameter-Efficient Fine-Tuning Methods for Large Models.

Parameter-Efficient Fine-Tuning for Large Models: A Comprehensive Survey, Zeyu Han et al, arxiv preprint, 2024

PEFT

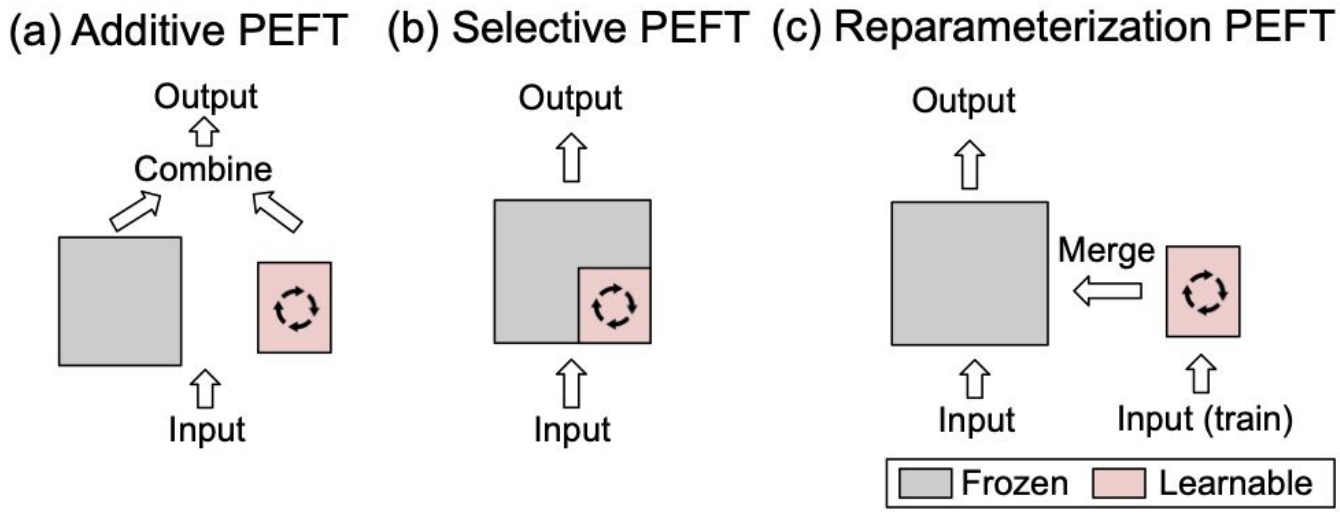


Fig. 4: Different types of PEFT algorithms.

Parameter-Efficient Fine-Tuning for Large Models: A Comprehensive Survey, Zeyu Han et al, arxiv preprint, 2024

PEFT

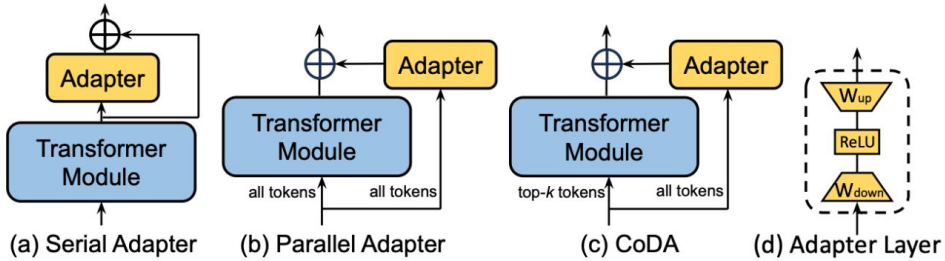


Fig. 5: Illustration of three representative adapter-based fine-tuning algorithms. Blue represents frozen, while yellow represents trainable.

Parameter-Efficient Fine-Tuning for Large Models: A Comprehensive Survey, Zeyu Han et al, arxiv preprint, 2024

PEFT

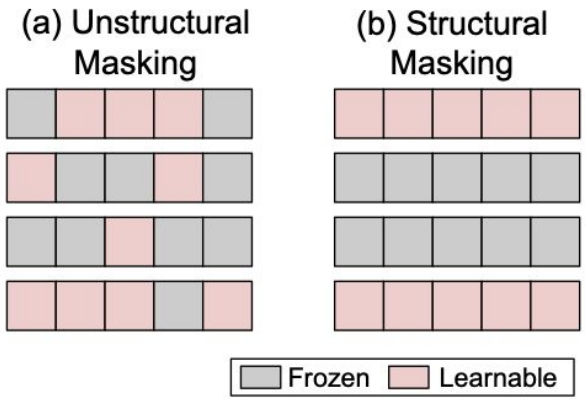


Fig. 7: Illustration of two parameter masking methods.

Parameter-Efficient Fine-Tuning for Large Models: A Comprehensive Survey, Zeyu Han et al, arxiv preprint, 2024

LoRA

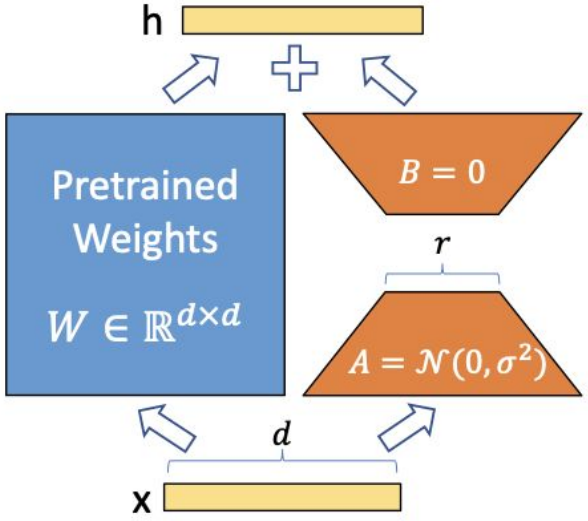


Figure 1: Our reparametrization. We only train A and B.

LoRA: Low-Rank Adaptation of Large Language Models, Edward Hu et al, ICLR 2022

LoRA

Model & Method	# Trainable Parameters	MNLI	SST-2	MRPC	CoLA	QNLI	QQP	RTE	STS-B	Avg.
RoB_{base} (FT)*	125.0M	**87.6**	94.8	90.2	**63.6**	92.8	**91.9**	78.7	91.2	86.4
RoB_{base} (BitFit)*	0.1M	84.7	93.7	**92.7**	62.0	91.8	84.0	81.5	90.8	85.2
RoB_{base} (Adpt^{D})*	0.3M	$87.1_{\pm.0}$	$94.2_{\pm.1}$	$88.5_{\pm1.1}$	$60.8_{\pm.4}$	$93.1_{\pm.1}$	$90.2_{\pm.0}$	$71.5_{\pm2.7}$	$89.7_{\pm.3}$	84.4
RoB_{base} (Adpt^{D})*	0.9M	$87.3_{\pm.1}$	$94.7_{\pm.3}$	$88.4_{\pm.1}$	$62.6_{\pm.9}$	$93.0_{\pm.2}$	$90.6_{\pm.0}$	$75.9_{\pm2.2}$	$90.3_{\pm.1}$	85.4
RoB_{base} (LoRA)	0.3M	$87.5_{\pm.3}$	$\mathbf{95.1}_{\pm.2}$	$89.7_{\pm.7}$	$63.4_{\pm1.2}$	$\mathbf{93.3}_{\pm.3}$	$90.8_{\pm.1}$	$\mathbf{86.6}_{\pm.7}$	$\mathbf{91.5}_{\pm.2}$	**87.2**
$\text{RoB}_{\text{large}}$ (FT)*	355.0M	90.2	**96.4**	**90.9**	68.0	94.7	**92.2**	86.6	92.4	88.9
$\text{RoB}_{\text{large}}$ (LoRA)	0.8M	$\mathbf{90.6}_{\pm.2}$	$96.2_{\pm.5}$	$\mathbf{90.9}_{\pm1.2}$	$\mathbf{68.2}_{\pm1.9}$	$\mathbf{94.9}_{\pm.3}$	$91.6_{\pm.1}$	$\mathbf{87.4}_{\pm2.5}$	$\mathbf{92.6}_{\pm.2}$	**89.0**
$\text{RoB}_{\text{large}}$ (Adpt^{P})†	3.0M	$90.2_{\pm.3}$	$96.1_{\pm.3}$	$90.2_{\pm.7}$	$\mathbf{68.3}_{\pm1.0}$	$\mathbf{94.8}_{\pm.2}$	$\mathbf{91.9}_{\pm.1}$	$83.8_{\pm2.9}$	$92.1_{\pm.7}$	88.4
$\text{RoB}_{\text{large}}$ (Adpt^{P})†	0.8M	$\mathbf{90.5}_{\pm.3}$	$\mathbf{96.6}_{\pm.2}$	$89.7_{\pm1.2}$	$67.8_{\pm2.5}$	$\mathbf{94.8}_{\pm.3}$	$91.7_{\pm.2}$	$80.1_{\pm2.9}$	$91.9_{\pm.4}$	87.9
$\text{RoB}_{\text{large}}$ (Adpt^{H})†	6.0M	$89.9_{\pm.5}$	$96.2_{\pm.3}$	$88.7_{\pm2.9}$	$66.5_{\pm4.4}$	$94.7_{\pm.2}$	$92.1_{\pm.1}$	$83.4_{\pm1.1}$	$91.0_{\pm1.7}$	87.8
$\text{RoB}_{\text{large}}$ (Adpt^{H})†	0.8M	$90.3_{\pm.3}$	$96.3_{\pm.5}$	$87.7_{\pm1.7}$	$66.3_{\pm2.0}$	$94.7_{\pm.2}$	$91.5_{\pm.1}$	$72.9_{\pm2.9}$	$91.5_{\pm.5}$	86.4
$\text{RoB}_{\text{large}}$ (LoRA)†	0.8M	$\mathbf{90.6}_{\pm.2}$	$96.2_{\pm.5}$	$\mathbf{90.2}_{\pm1.0}$	$68.2_{\pm1.9}$	$\mathbf{94.8}_{\pm.3}$	$91.6_{\pm.2}$	$\mathbf{85.2}_{\pm1.1}$	$\mathbf{92.3}_{\pm.5}$	**88.6**
DeB_{XXL} (FT)*	1500.0M	91.8	**97.2**	92.0	72.0	**96.0**	92.7	93.9	92.9	91.1
DeB_{XXL} (LoRA)	4.7M	$\mathbf{91.9}_{\pm.2}$	$96.9_{\pm.2}$	$\mathbf{92.6}_{\pm.6}$	$\mathbf{72.4}_{\pm1.1}$	$\mathbf{96.0}_{\pm.1}$	$\mathbf{92.9}_{\pm.1}$	$\mathbf{94.9}_{\pm.4}$	$\mathbf{93.0}_{\pm.2}$	**91.3**

Table 2: $\text{RoBERTa}_{\text{base}}$, $\text{RoBERTa}_{\text{large}}$, and $\text{DeBERTa}_{\text{XXL}}$ with different adaptation methods on the GLUE benchmark. We report the overall (matched and mismatched) accuracy for MNLI, Matthew's correlation for CoLA, Pearson correlation for STS-B, and accuracy for other tasks. Higher is better for all metrics. * indicates numbers published in prior works. † indicates runs configured in a setup similar to Houlsby et al. (2019) for a fair comparison.

LoRA: Low-Rank Adaptation of Large Language Models, Edward Hu et al, ICLR 2022

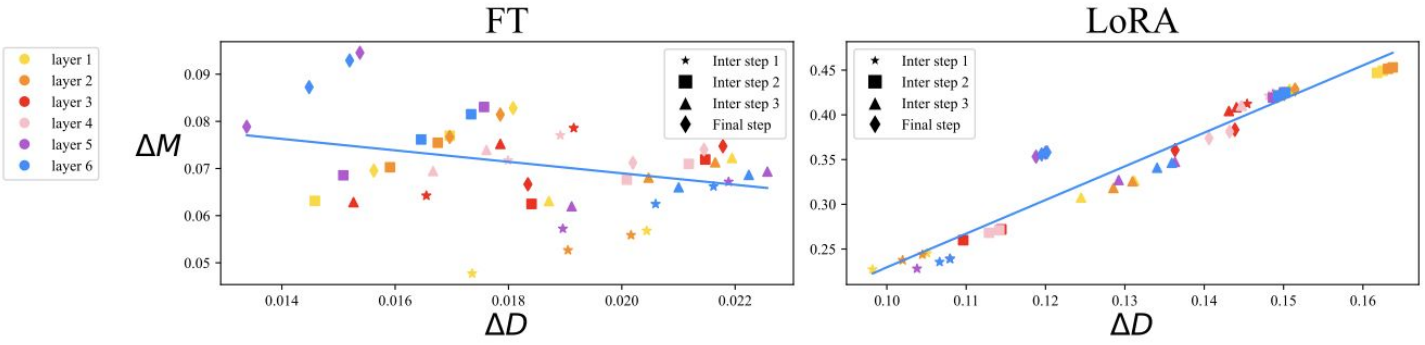
LoRA

Model & Method	# Trainable Parameters	E2E NLG Challenge BLEU	NIST	MET	ROUGE-L	CIDEr
GPT-2 M (FT)*	354.92M	68.2	8.62	46.2	71.0	2.47
GPT-2 M (Adapter^{L})*	0.37M	66.3	8.41	45.0	69.8	2.40
GPT-2 M (Adapter^{L})*	11.09M	68.9	8.71	46.1	71.3	2.47
GPT-2 M (Adapter^{H})	11.09M	$67.3_{\pm.6}$	$8.50_{\pm.07}$	$46.0_{\pm.2}$	$70.7_{\pm.2}$	$2.44_{\pm.01}$
GPT-2 M (FT^{Top2})*	25.19M	68.1	8.59	46.0	70.8	2.41
GPT-2 M (PreLayer)*	0.35M	69.7	8.81	46.1	71.4	2.49
GPT-2 M (LoRA)	0.35M	$\mathbf{70.4}_{\pm.1}$	$\mathbf{8.85}_{\pm.02}$	$\mathbf{46.8}_{\pm.2}$	$\mathbf{71.8}_{\pm.1}$	$\mathbf{2.53}_{\pm.02}$
GPT-2 L (FT)*	774.03M	68.5	8.78	46.0	69.9	2.45
GPT-2 L (Adapter^{L})	0.88M	$69.1_{\pm.1}$	$8.68_{\pm.03}$	$46.3_{\pm.0}$	$71.4_{\pm.2}$	$\mathbf{2.49}_{\pm.0}$
GPT-2 L (Adapter^{L})	23.00M	$68.9_{\pm.3}$	$8.70_{\pm.04}$	$46.1_{\pm.1}$	$71.3_{\pm.2}$	$2.45_{\pm.02}$
GPT-2 L (PreLayer)*	0.77M	70.3	8.85	46.2	71.7	2.47
GPT-2 L (LoRA)	0.77M	$\mathbf{70.4}_{\pm.1}$	$\mathbf{8.89}_{\pm.02}$	$\mathbf{46.8}_{\pm.2}$	$\mathbf{72.0}_{\pm.2}$	$2.47_{\pm.02}$

Table 3: GPT-2 medium (M) and large (L) with different adaptation methods on the E2E NLG Challenge. For all metrics, higher is better. LoRA outperforms several baselines with comparable or fewer trainable parameters. Confidence intervals are shown for experiments we ran. * indicates numbers published in prior works.

LoRA: Low-Rank Adaptation of Large Language Models, Edward Hu et al, ICLR 2022

DoRA



$$W = m\frac{V}{||V||_c} = ||W||_c\frac{W}{||W||_c}$$

$$\Delta M_{\mathrm{FT}}^t = \frac{\sum_{n=1}^{k} |m_{\mathrm{FT}}^{n,t} - m_0^n|}{k} \quad (3)$$

$$\Delta D_{\mathrm{FT}}^t = \frac{\sum_{n=1}^{k} (1 - \cos(V_{\mathrm{FT}}^{n,t}, W_0^n))}{k} \quad (4)$$

DoRA: Weight-Decomposed Low-Rank Adaptation, Shih-Yang Liu et al,ICML 2024

DoRA

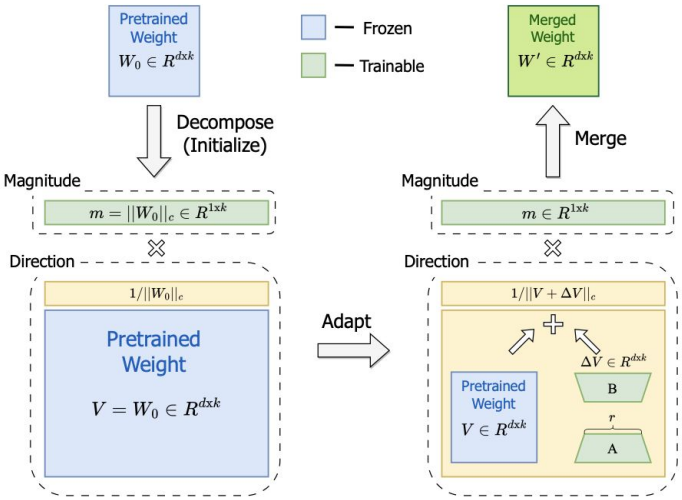


Figure 1. An overview of our proposed DoRA, which decomposes the pre-trained weight into *magnitude* and *direction* components for fine-tuning, especially with LoRA to efficiently update the direction component. Note that $|| \cdot ||_c$ denotes the vector-wise norm of a matrix across each column vector.

DoRA: Weight-Decomposed Low-Rank Adaptation, Shih-Yang Liu et al,ICML 2024

DoRA

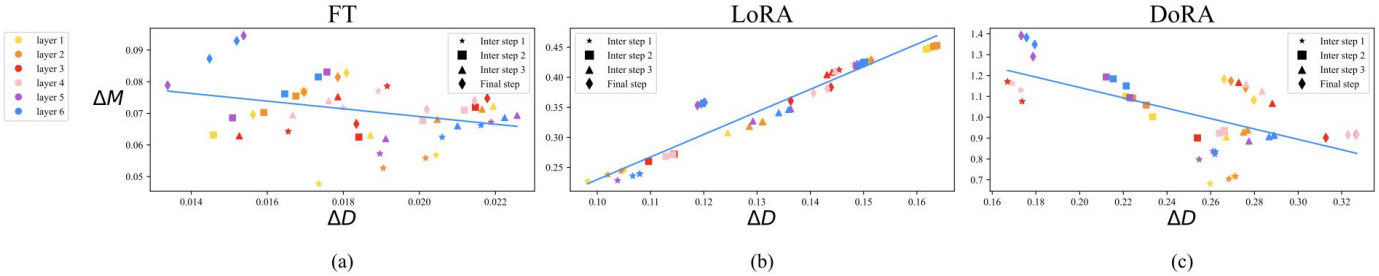


Figure 2. Magnitude and direction updates of (a) FT, (b) LoRA, and (c) DoRA of the query matrices across different layers and intermediate steps. Different markers represent matrices of different training steps and different colors represent the matrices of each layer.

DoRA: Weight-Decomposed Low-Rank Adaptation, Shih-Yang Liu et al,ICML 2024

DoRA

$$W' = \underline{m}\frac{V + \Delta V}{||V + \Delta V||_c} = \underline{m}\frac{W_0 + \underline{BA}}{||W_0 + \underline{BA}||_c} \tag{5}$$

$$\nabla_{V'}\mathcal{L} = \frac{m}{||V'||_c}\left(I - \frac{V'V'^{\mathbf{T}}}{||V'||_c^2}\right)\nabla_{W'}\mathcal{L} \tag{6}$$

$$\nabla_m\mathcal{L} = \frac{\nabla_{W'}\mathcal{L} \cdot V'}{||V'||_c} \tag{7}$$

Eq. (6) reveals that the weight gradient $\nabla_{W'}\mathcal{L}$ is scaled by $m/||V'||_c$ and is projected away from the current weight matrix. These two effects contribute to aligning the gradient's covariance matrix more closely with the identity matrix, which is advantageous for optimization (Salimans

DoRA: Weight-Decomposed Low-Rank Adaptation, Shih-Yang Liu et al,ICML 2024

DoRA

In Eq. (1), the gradients of W' and ΔW are the same. However, with DoRA, which redirects the low-rank adaptation towards the directional component, the gradient of the low-rank updates differs from that of W', as illustrated in Eq. (6). This divergence necessitates extra memory during backpropagation. To address this, we suggest treating $||V + \Delta V||_c$ in Eq. (5) as a constant, thereby detaching it from the gradient graph. This means that while $||V + \Delta V||_c$ dynamically reflects the updates of ΔV, it won't receive any gradient during backpropagation. With this modification, the gradient w.r.t m remains unchanged, and $\nabla_{V'}\mathcal{L}$ is redefined as:

$$\nabla_{V'}\mathcal{L} = \frac{m}{C}\nabla_{W'}\mathcal{L} \text{ where } C = ||V'||_c \qquad (11)$$

DoRA: Weight-Decomposed Low-Rank Adaptation, Shih-Yang Liu et al,ICML 2024

DoRA

Model	PEFT Method	# Params (%)	BoolQ	PIQA	SIQA	HellaSwag	WinoGrande	ARC-e	ARC-c	OBQA	Avg.
ChatGPT	-	-	73.1	85.4	68.5	78.5	66.1	89.8	79.9	74.8	77.0
LLaMA-7B	Prefix	0.11	64.3	76.8	73.9	42.1	72.1	72.9	54.0	60.6	64.6
	Series	0.99	63.0	79.2	76.3	67.9	75.7	74.5	57.1	72.4	70.8
	Parallel	3.54	67.9	76.4	78.8	69.8	78.9	73.7	57.3	75.2	72.2
	LoRA	0.83	68.9	80.7	77.4	78.1	78.8	77.8	61.3	74.8	74.7
	DoRA† (Ours)	0.43	70.0	82.6	79.7	83.2	80.6	80.6	65.4	77.6	**77.5**
	DoRA (Ours)	0.84	69.7	83.4	78.6	87.2	81.0	81.9	66.2	79.2	**78.4**
LLaMA-13B	Prefix	0.03	65.3	75.4	72.1	55.2	68.6	79.5	62.9	68.0	68.4
	Series	0.80	71.8	83	79.2	88.1	82.4	82.5	67.3	81.8	79.5
	Parallel	2.89	72.5	84.9	79.8	92.1	84.7	84.2	71.2	82.4	81.4
	LoRA	0.67	72.1	83.5	80.5	90.5	83.7	82.8	68.3	82.4	80.5
	DoRA† (Ours)	0.35	72.5	85.3	79.9	90.1	82.9	82.7	69.7	83.6	**80.8**
	DoRA (Ours)	0.68	72.4	84.9	81.5	92.4	84.2	84.2	69.6	82.8	**81.5**
LLaMA2-7B	LoRA	0.83	69.8	79.9	79.5	83.6	82.6	79.8	64.7	81.0	77.6
	DoRA† (Ours)	0.43	72.0	83.1	79.9	89.1	83.0	84.5	71.0	81.2	**80.5**
	DoRA (Ours)	0.84	71.8	83.7	76.0	89.1	82.6	83.7	68.2	82.4	**79.7**
LLaMA3-8B	LoRA	0.70	70.8	85.2	79.9	91.7	84.3	84.2	71.2	79.0	80.8
	DoRA† (Ours)	0.35	74.5	88.8	80.3	95.5	84.7	90.1	79.1	87.2	**85.0**
	DoRA (Ours)	0.71	74.6	89.3	79.9	95.5	85.6	90.5	80.4	85.8	**85.2**

DoRA: Weight-Decomposed Low-Rank Adaptation, Shih-Yang Liu et al,ICML 2024

DoRA

5.2. Image/Video-Text Understanding

Table 2. The multi-task evaluation results on VQA, GQA, NVLR^2 and COCO Caption with the VL-BART backbone.

Method	# Params (%)	VQA^{v2}	GQA	NVLR^2	COCO Cap	Avg.
FT	100	66.9	56.7	73.7	112.0	77.3
LoRA	5.93	65.2	53.6	71.9	115.3	76.5
DoRA (Ours)	5.96	65.8	54.7	73.1	115.9	**77.4**

Table 3. The multi-task evaluation results on TVQA, How2QA, TVC, and YC2C with the VL-BART backbone.

Method	# Params (%)	TVQA	How2QA	TVC	YC2C	Avg.
FT	100	76.3	73.9	45.7	154	87.5
LoRA	5.17	75.5	72.9	44.6	140.9	83.5
DoRA (Ours)	5.19	76.3	74.1	45.8	145.4	**85.4**

Table 4. Visual instruction tuning evaluation results for LLaVA-1.5-7B on a wide range of seven vision-language tasks. We directly use checkpoints from (Liu et al., 2023a) to reproduce their results.

Method	# Params(%)	Avg.
FT	100	66.5
LoRA	4.61	66.9
DoRA (Ours)	4.63	**67.6**

DoRA: Weight-Decomposed Low-Rank Adaptation, Shih-Yang Liu et al,ICML 2024

Fast Forwarding

Our proposal, Fast Forward, is a procedure for accelerating training at regular intervals by selecting an optimal step size with a line search. Following warmup, we apply Fast Forward every $T_{\text{interval}} = 6$ optimizer steps, as seen in Figure 1. During a Fast Forward stage, for each trainable parameter, the difference between weights in the current and previous timesteps is calculated:

$$\Delta_{\mathbf{W}} = \mathbf{W}_t - \mathbf{W}_{t-1} \tag{2}$$

The direction $\Delta_{\mathbf{W}}$ is used to iteratively update $\mathbf{W}_t$. In the τ-th Fast Forward step, the updated weight matrix is given by $\mathbf{W}_t + \tau\Delta_{\mathbf{W}}$. The recursive updates continue until the model's loss on a small validation set $L(X_{\text{val}})$ stops improving. When a Fast Forward step causes this validation loss to increase, the Fast Forward stage concludes, and regular optimization resumes for the next T_{interval} steps before reapplying Fast Forward.

Fast Forwarding Low-Rank Training, Adir Rahamim, ACL Submission, 2024

Fast Forwarding

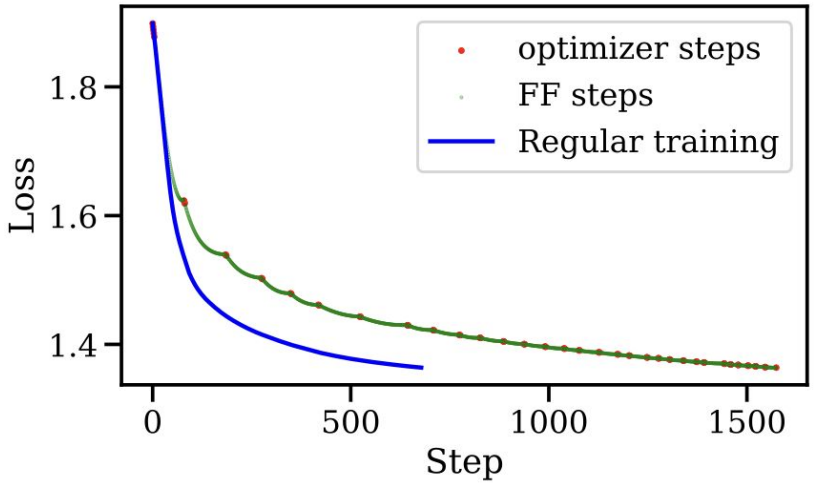
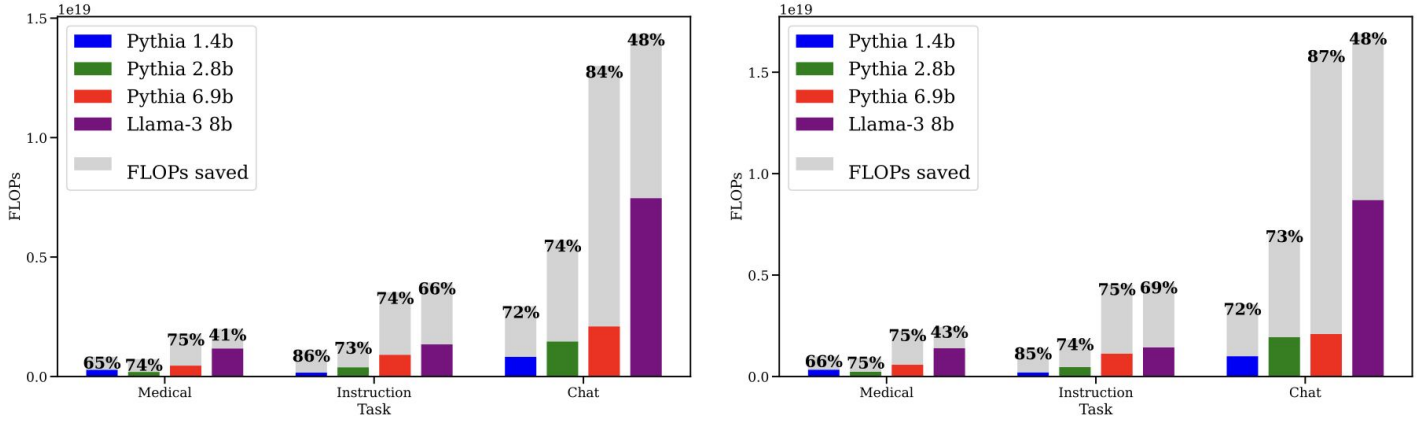


Figure 4: Training Pythia-6.9B on the chat tuning task, with other models in Appx A. Red dots represent SGD steps and green dots represent Fast Forward steps. The blue line shows vanilla Adam SGD training.

Fast Forwarding Low-Rank Training, Adir Rahamim, ACL Submission, 2024

Fast Forwarding



(a) LoRA finetuning (b) DoRA finetuning

Figure 2: The percentage of FLOPs saved during (a) LoRA and (b) DoRA finetuning with Fast Forward to match test loss after 5 epochs of regular Adam SGD training. Fast Forward saves 41–87% FLOPs, depending on the task.

Fast Forwarding Low-Rank Training, Adir Rahamim, ACL Submission, 2024

Fast Forwarding

5.1 FF does not harm long-term accuracy.

Many efficiency methods accelerate training until a fixed threshold accuracy, but ultimately harm final performance. We find that Fast Forward has no such disadvantage. To check, we finetune the Pythia 1.4B model on the medical domain dataset until the loss stopped improving on the test set. We permanently switch to standard Adam SGD after Fast Forward fails to improve the loss on the 32-sample tiny validation set $L(X_{\text{val}})$ three times in a row—though at this point, training ends after only 6 more SGD steps. Fast Forward converges to a slightly better loss while allowing us to save 56% of FLOPs.

5.2 FF does not harm performance on a standard benchmark.

We evaluated whether FF-trained models match the performance of regularly-trained models on a standard benchmark. On the PubMedQA dataset,

we evaluated two Llama-3 8b models that were fine-tuned on the medical domain, one that was regularly trained and one with FF. We tested the models on a subset of 1K examples using few-shot in-context learning. As context, all models received the same prompt of 3 randomly chosen examples, one with the answer yes, one with the answer no and one with the answer maybe, in an arbitrary order. The regularly trained model achieved an accuracy of 49.75%, whereas FF trained model achieved an accuracy of 50.95%. This shows that FF does not harm standard benchmark results.

Fast Forwarding Low-Rank Training, Adir Rahamim, ACL Submission, 2024

Fast Forwarding

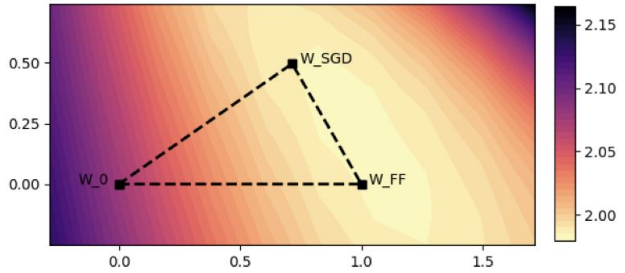


Figure 5: Test loss on the plane intersecting the pre-trained model $\mathbf{W}_0$ and the models trained with Adam SGD $\mathbf{W}_{\text{SGD}}$, and with Fast Forward $\mathbf{W}_{\text{FF}}$. Axis scale corresponds to the norm of differences $\|\mathbf{W}_{\text{FF}} - \mathbf{W}_0\|_2$.

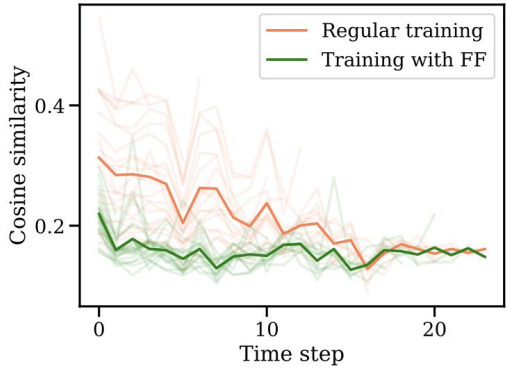


Figure 6: The cosine similarity between gradients in different time steps, in regular training and training with Fast Forward. At each timestep, we measure similarity between the current gradient and all previous saved gradients (shown individually in transparent lines). Fast Forward leads to lower average similarity (shown in opaque lines) with previous gradients.

Fast Forwarding Low-Rank Training, Adir Rahamim, ACL Submission, 2024

Fast Forwarding

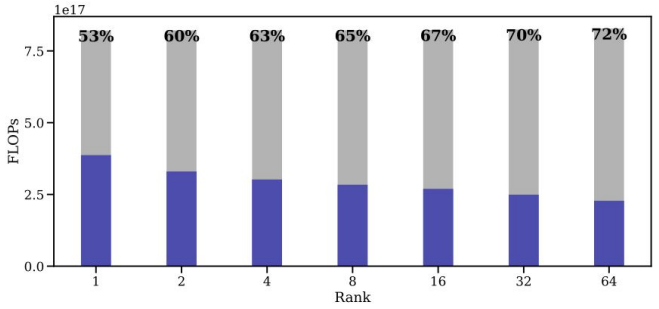


Figure 7: The total number of FLOPs consumed during training Pythia 1.4B on the clinical finetuning task for different LoRA ranks. Gray area is the compute saved with Fast Forward.

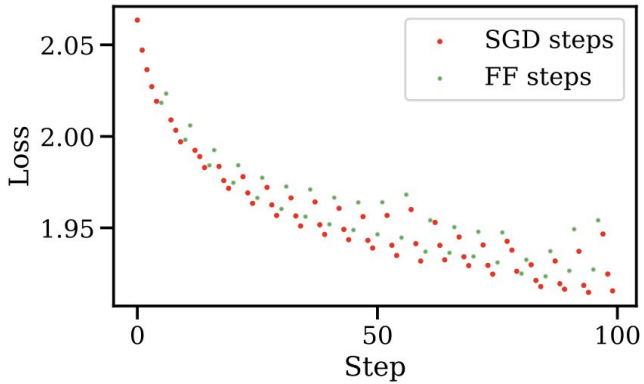


Figure 8: Test loss for full-rank standard finetuning restricted to attention layers. Each time we Fast Forward, loss increases immediately at the first simulated step.

Fast Forwarding Low-Rank Training, Adir Rahamim, ACL Submission, 2024

QLoRA

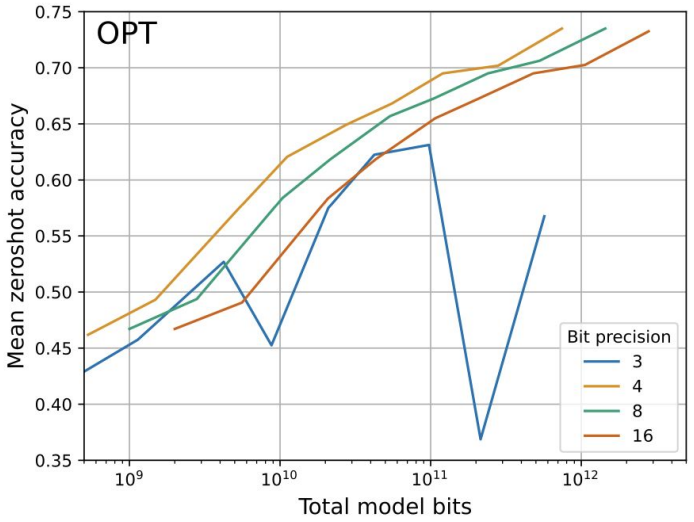


Figure 1. Bit-level scaling laws for mean zero-shot performance across four datasets for 125M to 176B parameter OPT models. Zero-shot performance increases steadily for fixed model bits as we reduce the quantization precision from 16 to 4 bits. At 3-bits, this relationship reverses, making 4-bit precision optimal.

Можем лучше сжиматься

А можем плохо сжиматься и дешево тюниться => LoRA

Так еще и адаптируем сжатие под конкретную задачу

The case for 4-bit precision: k-bit Inference Tim Dettmers, Luke Zettlemoyer, ICML, 2023

QLoRA

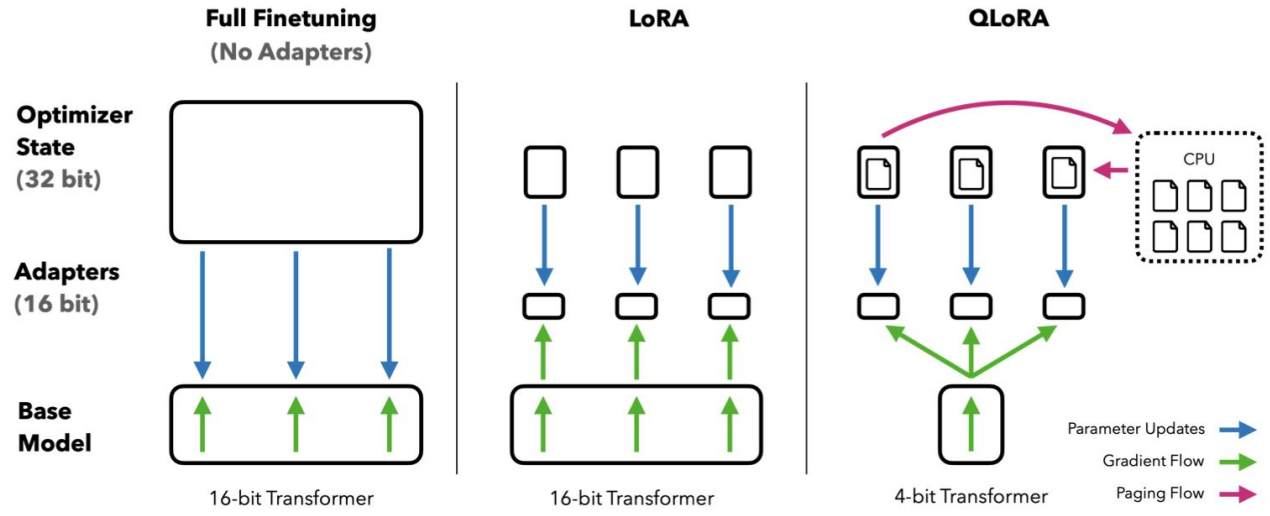
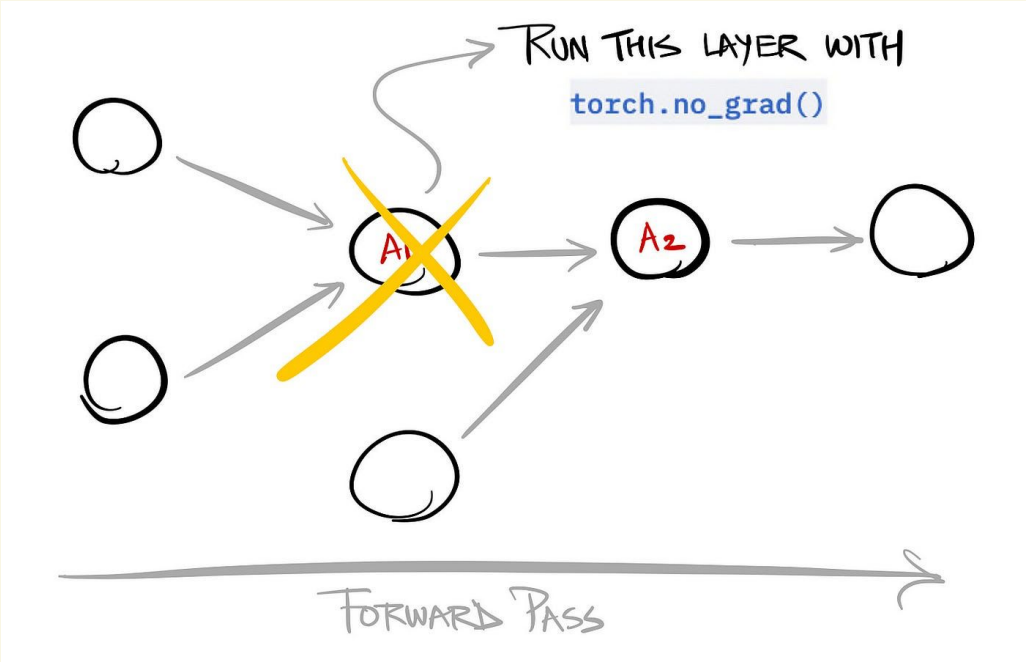


Figure 1: Different finetuning methods and their memory requirements. QLoRA improves over LoRA by quantizing the transformer model to 4-bit precision and using paged optimizers to handle memory spikes.

QLoRA: Efficient Finetuning of Quantized LLMs, Tim Dettmers et a, INeurIPS 2023

Gradient checkpointing



QLoRA

QLoRA. Using the components described above, we define QLoRA for a single linear layer in the quantized base model with a single LoRA adapter as follows:

$$\mathbf{Y}^{\text{BF16}} = \mathbf{X}^{\text{BF16}}\text{doubleDequant}(c_1^{\text{FP32}}, c_2^{\text{k-bit}}, \mathbf{W}^{\text{NF4}}) + \mathbf{X}^{\text{BF16}}\mathbf{L}_1^{\text{BF16}}\mathbf{L}_2^{\text{BF16}}, \tag{5}$$

where $\text{doubleDequant}(\cdot)$ is defined as:

$$\text{doubleDequant}(c_1^{\text{FP32}}, c_2^{\text{k-bit}}, \mathbf{W}^{\text{k-bit}}) = \text{dequant}(\text{dequant}(c_1^{\text{FP32}}, c_2^{\text{k-bit}}), \mathbf{W}^{\text{4bit}}) = \mathbf{W}^{\text{BF16}}, \tag{6}$$

4-bit NormalFloat Quantization The NormalFloat (NF) data type builds on Quantile Quantization [15] which is an information-theoretically optimal data type that ensures each quantization bin has an equal number of values assigned from the input tensor. Quantile quantization works by estimating the quantile of the input tensor through the empirical cumulative distribution function.

QLoRA: Efficient Finetuning of Quantized LLMs, Tim Dettmers et a, INeurIPS 2023

QLoRA

```python
import torch
from torchtune.modules.peft import LoRALinear

torch.set_default_device("cuda")
qlora_linear = LoRALinear(512, 512, rank=8, alpha=0.1, quantize_base=True)
print(torch.cuda.memory_allocated())  # 177,152 bytes
del qlora_linear
torch.cuda.empty_cache()
lora_linear = LoRALinear(512, 512, rank=8, alpha=0.1, quantize_base=False)
print(torch.cuda.memory_allocated()) # 1,081,344 bytes
```

QLoRA: Efficient Finetuning of Quantized LLMs, Tim Dettmers et a, INeurIPS 2023

QLoRA

Table 3: Experiments comparing 16-bit BrainFloat (BF16), 8-bit Integer (Int8), 4-bit Float (FP4), and 4-bit NormalFloat (NF4) on GLUE and Super-NaturalInstructions. QLoRA replicates 16-bit LoRA and full-finetuning.

Dataset	GLUE (Acc.)	Super-NaturalInstructions (RougeL)				
Model	RoBERTa-large	T5-80M	T5-250M	T5-780M	T5-3B	T5-11B
BF16	88.6	40.1	42.1	48.0	54.3	62.0
BF16 replication	88.6	40.0	42.2	47.3	54.9	-
LoRA BF16	88.8	40.5	42.6	47.1	55.4	60.7
QLoRA Int8	88.8	40.4	42.9	45.4	56.5	60.7
QLoRA FP4	88.6	40.3	42.4	47.5	55.6	60.9
QLoRA NF4 + DQ	-	40.4	42.7	47.7	55.3	60.9

QLoRA: Efficient Finetuning of Quantized LLMs, Tim Dettmers et a, INeurIPS 2023

QDoRA

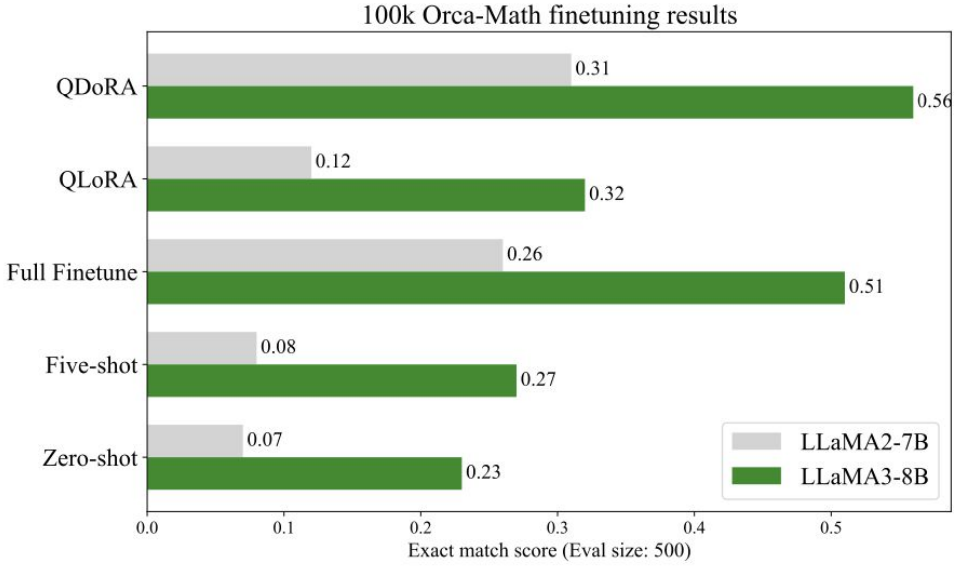


Figure 6. Accuracy comparison of LLaMA2-7B/LLaMA3-8B with QDoRA, QLoRA and FT on Orca-Math (Mitra et al., 2024).

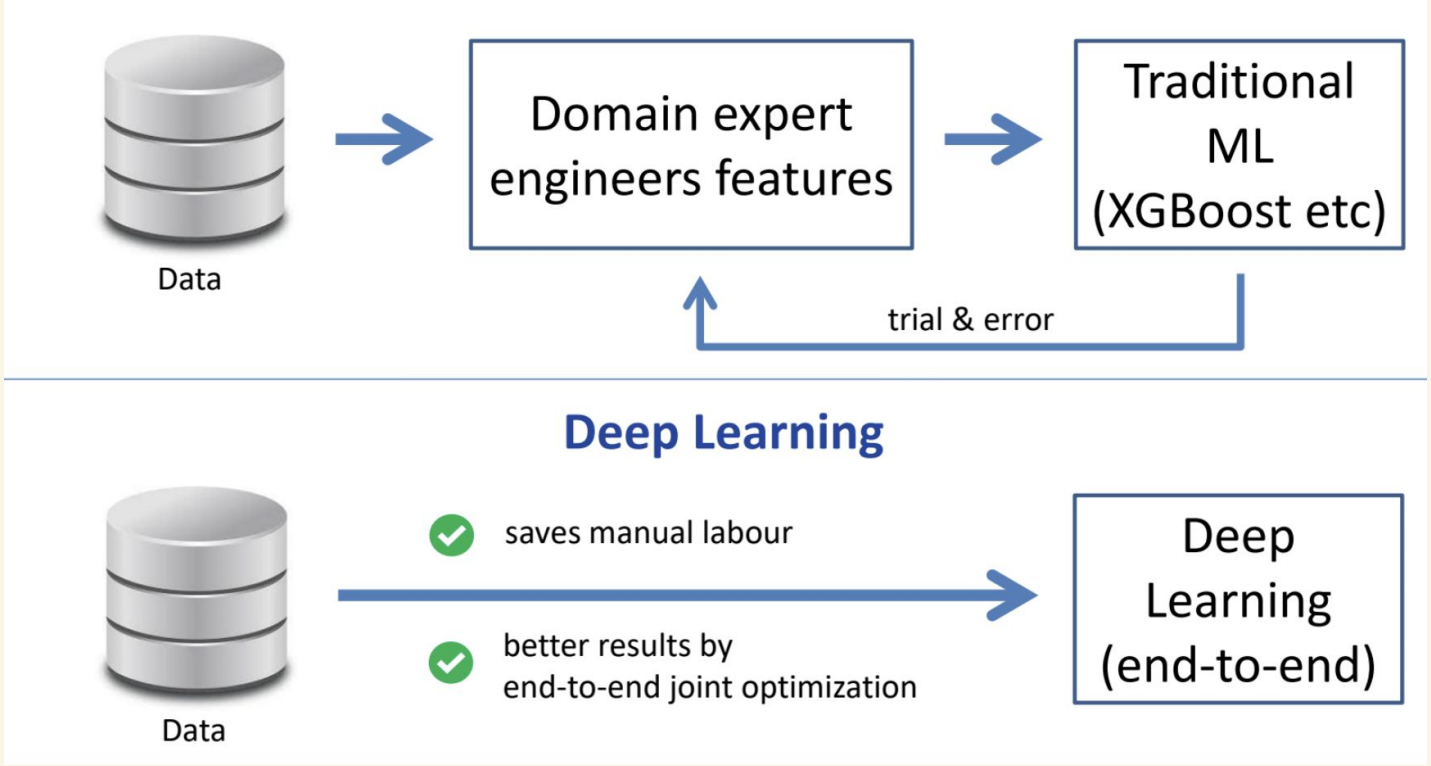
DoRA: Weight-Decomposed Low-Rank Adaptation, Shih-Yang Liu et al,ICML 2024

NAS

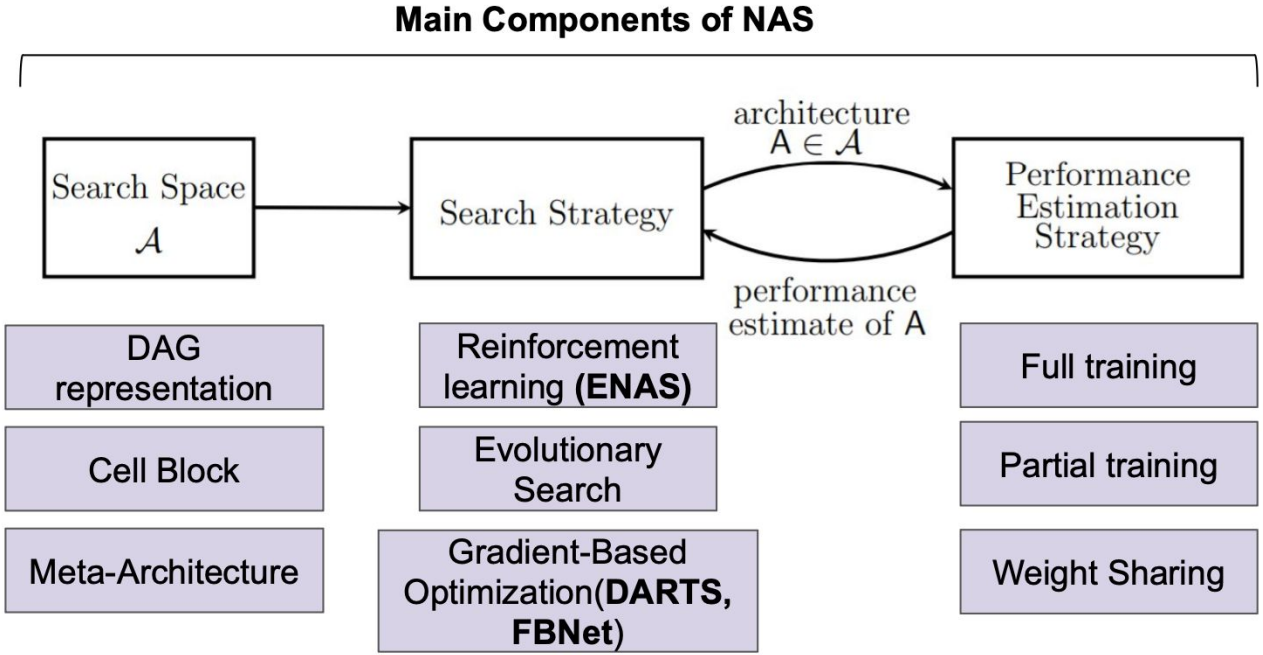
1. Мотивация
2. Постановка задачи
3. Sample-based подход
4. Дифференцируемый NAS
5. Приложения

На основе материалов Ильи Трофимова, Сколтех

Мотивация


Data
Domain expert engineers features
Traditional ML (XGBoost etc)
trial & error
Deep Learning
saves manual labour
better results by end-to-end joint optimization
Deep Learning (end-to-end)
Data

Постановка задачи



Sample based

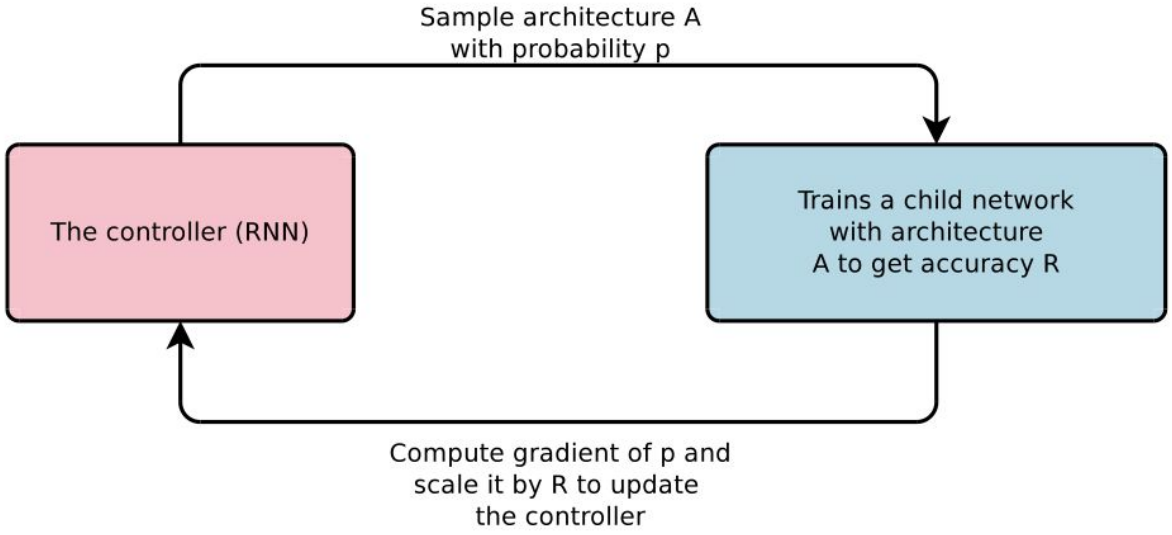


Figure 1: An overview of Neural Architecture Search.

Neural architecture search with reinforcement learning, Zoph, B., & Le, Q. V, ICLR 2017

Sample based

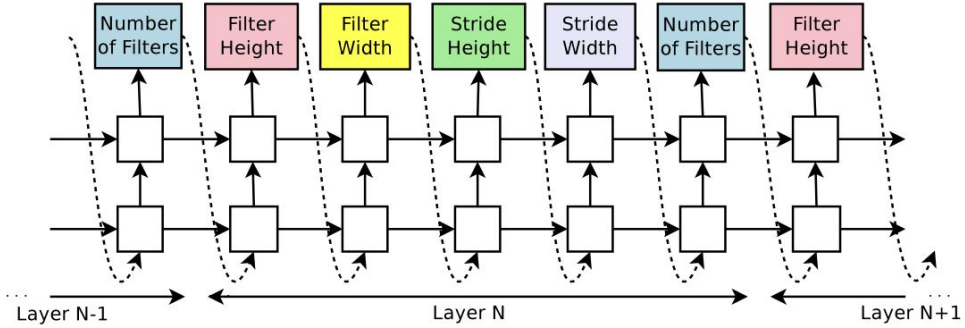


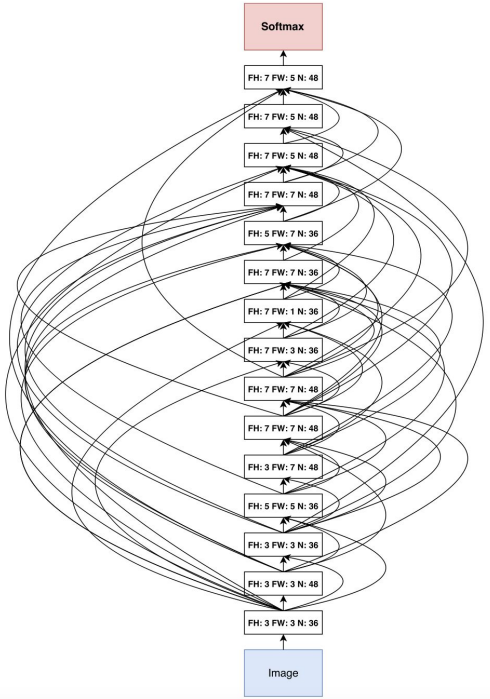
Figure 2: How our controller recurrent neural network samples a simple convolutional network. It predicts filter height, filter width, stride height, stride width, and number of filters for one layer and repeats. Every prediction is carried out by a softmax classifier and then fed into the next time step as input.

Neural architecture search with reinforcement learning, Zoph, B., & Le, Q. V, ICLR 2017

Sample based

Model	Depth	Parameters	Error rate (%)
Network in Network (Lin et al., 2013)	-	-	8.81
All-CNN (Springenberg et al., 2014)	-	-	7.25
Deeply Supervised Net (Lee et al., 2015)	-	-	7.97
Highway Network (Srivastava et al., 2015)	-	-	7.72
Scalable Bayesian Optimization (Snoek et al., 2015)	-	-	6.37
FractalNet (Larsson et al., 2016)	21	38.6M	5.22
with Dropout/Drop-path	21	38.6M	4.60
ResNet (He et al., 2016a)	110	1.7M	6.61
ResNet (reported by Huang et al. (2016c))	110	1.7M	6.41
ResNet with Stochastic Depth (Huang et al., 2016c)	110	1.7M	5.23
	1202	10.2M	4.91
Wide ResNet (Zagoruyko & Komodakis, 2016)	16	11.0M	4.81
	28	36.5M	4.17
ResNet (pre-activation) (He et al., 2016b)	164	1.7M	5.46
	1001	10.2M	4.62
DenseNet ($L = 40, k = 12$) Huang et al. (2016a)	40	1.0M	5.24
DenseNet($L = 100, k = 12$) Huang et al. (2016a)	100	7.0M	4.10
DenseNet ($L = 100, k = 24$) Huang et al. (2016a)	100	27.2M	3.74
DenseNet-BC ($L = 100, k = 40$) Huang et al. (2016b)	190	25.6M	3.46
Neural Architecture Search v1 no stride or pooling	15	4.2M	5.50
Neural Architecture Search v2 predicting strides	20	2.5M	6.01
Neural Architecture Search v3 max pooling	39	7.1M	4.47
Neural Architecture Search v3 max pooling + more filters	39	37.4M	3.65

Table 1: Performance of Neural Architecture Search and other state-of-the-art models on CIFAR-10.



Neural architecture search with reinforcement learning, Zoph, B., & Le, Q. V, ICLR 2017

Sample based

Model	Parameters	Test Perplexity
Mikolov & Zweig (2012) - KN-5	2M‡	141.2
Mikolov & Zweig (2012) - KN5 + cache	2M‡	125.7
Mikolov & Zweig (2012) - RNN	6M‡	124.7
Mikolov & Zweig (2012) - RNN-LDA	7M‡	113.7
Mikolov & Zweig (2012) - RNN-LDA + KN-5 + cache	9M‡	92.0
Pascanu et al. (2013) - Deep RNN	6M	107.5
Cheng et al. (2014) - Sum-Prod Net	5M‡	100.0
Zaremba et al. (2014) - LSTM (medium)	20M	82.7
Zaremba et al. (2014) - LSTM (large)	66M	78.4
Gal (2015) - Variational LSTM (medium, untied)	20M	79.7
Gal (2015) - Variational LSTM (medium, untied, MC)	20M	78.6
Gal (2015) - Variational LSTM (large, untied)	66M	75.2
Gal (2015) - Variational LSTM (large, untied, MC)	66M	73.4
Kim et al. (2015) - CharCNN	19M	78.9
Press & Wolf (2016) - Variational LSTM, shared embeddings	51M	73.2
Merity et al. (2016) - Zoneout + Variational LSTM (medium)	20M	80.6
Merity et al. (2016) - Pointer Sentinel-LSTM (medium)	21M	70.9
Inan et al. (2016) - VD-LSTM + REAL (large)	51M	68.5
Zilly et al. (2016) - Variational RHN, shared embeddings	24M	66.0
Neural Architecture Search with base 8	32M	67.9
Neural Architecture Search with base 8 and shared embeddings	25M	64.0
Neural Architecture Search with base 8 and shared embeddings	54M	62.4

Neural architecture search with reinforcement learning, Zoph, B., & Le, Q. V, ICLR 2017

Sample based

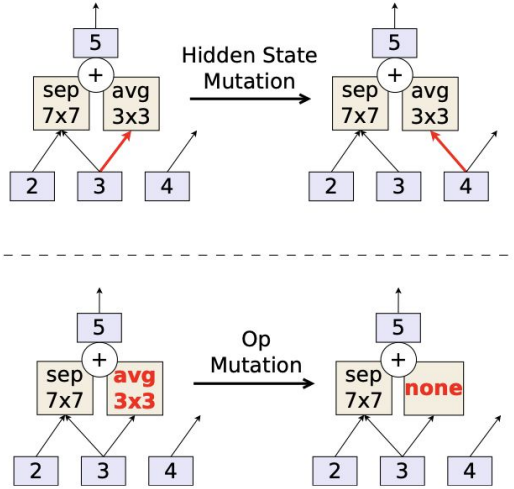


Figure 2: Illustration of the two mutation types.

Algorithm 1 Aging Evolution

population ← empty queue ▷ The population.
history ← ∅ ▷ Will contain all models.
while |*population*| < *P* **do** ▷ Initialize population.
  model.arch ← RANDOMARCHITECTURE()
  model.accuracy ← TRAINANDEVAL(*model.arch*)
  add *model* to right of *population*
  add *model* to history
end while
while |*history*| < *C* **do** ▷ Evolve for *C* cycles.
  sample ← ∅ ▷ Parent candidates.
  while |*sample*| < *S* **do**
    candidate ← random element from *population*
      ▷ The element stays in the *population*.
    add *candidate* to *sample*
  end while
  parent ← highest-accuracy model in *sample*
  child.arch ← MUTATE(*parent.arch*)
  child.accuracy ← TRAINANDEVAL(*child.arch*)
  add *child* to right of *population*
  add *child* to *history*
  remove *dead* from left of *population* ▷ Oldest.
  discard *dead*
end while
return highest-accuracy model in *history*

Regularized Evolution for Image Classifier Architecture Search, Esteban Real et al, AAAI 2019

Sample based

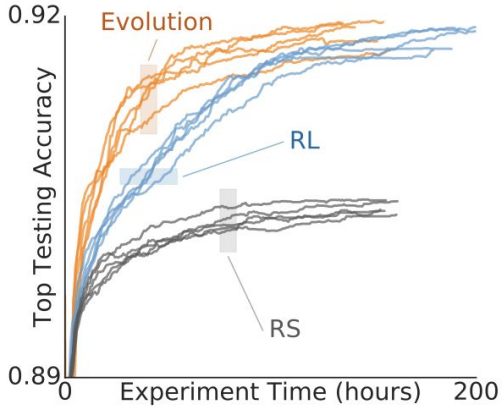


Figure 3: Time-course of 5 identical large-scale experiments for each algorithm (evolution, RL, and RS), showing accuracy before augmentation on CIFAR-10. All experiments were stopped when 20k models were evaluated, as done in the baseline study. Note this plot does not show the compute cost of models, which was higher for the RL ones.

Regularized Evolution for Image Classifier Architecture Search, Esteban Real et al, AAAI 2019

DARTS

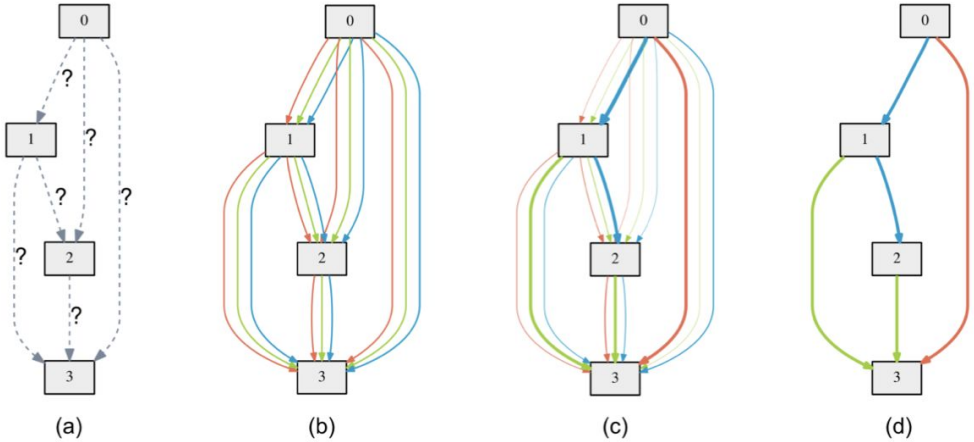


Figure 1: An overview of DARTS: (a) Operations on the edges are initially unknown. (b) Continuous relaxation of the search space by placing a mixture of candidate operations on each edge. (c) Joint optimization of the mixing probabilities and the network weights by solving a bilevel optimization problem. (d) Inducing the final architecture from the learned mixing probabilities.

DARTS: Differentiable architecture search, Hanxiao Liu et al, ICLR 2019

DARTS

$$\min_{\alpha} \quad \mathcal{L}_{val}(w^*(\alpha), \alpha)$$
$$\text{s.t.} \quad w^*(\alpha) = \text{argmin}_w \ \mathcal{L}_{train}(w, \alpha)$$

Let's approximate $w^*(\alpha) \approx w' = w - \xi \nabla_w L_{train}(w, \alpha)$.
Then

$$\begin{aligned}
\frac{d}{d\alpha} L_{val}(w^*(\alpha), \alpha) &\approx \nabla_\alpha L_{val}(w - \nabla_w L_{train}(w, \alpha), \alpha) \\
&= \nabla_w L_{val}(w', \alpha) \frac{\partial w'}{\partial \alpha} + \nabla_\alpha L_{val}(w', \alpha) \\
&= -\xi \nabla^2_{\alpha,w} L_{train}(w, \alpha) \nabla_w L_{val}(w', \alpha) + \nabla_\alpha L_{val}(w', \alpha)
\end{aligned}$$

$$\nabla^2_{\alpha,w} \mathcal{L}_{train}(w, \alpha) \nabla_{w'} \mathcal{L}_{val}(w', \alpha) \approx \frac{\nabla_\alpha \mathcal{L}_{train}(w^+, \alpha) - \nabla_\alpha \mathcal{L}_{train}(w^-, \alpha)}{2\epsilon}$$

DARTS: Differentiable architecture search, Hanxiao Liu et al, ICLR 2019

DARTS

Algorithm 1: DARTS – Differentiable Architecture Search

Create a mixed operation $\bar{o}^{(i,j)}$ parametrized by $\alpha^{(i,j)}$ for each edge (i, j)

while *not converged* **do**

1. Update architecture α by descending $\nabla_\alpha \mathcal{L}_{val}(w - \xi \nabla_w \mathcal{L}_{train}(w, \alpha), \alpha)$ ($\xi = 0$ if using first-order approximation)
2. Update weights w by descending $\nabla_w \mathcal{L}_{train}(w, \alpha)$

Derive the final architecture based on the learned α.

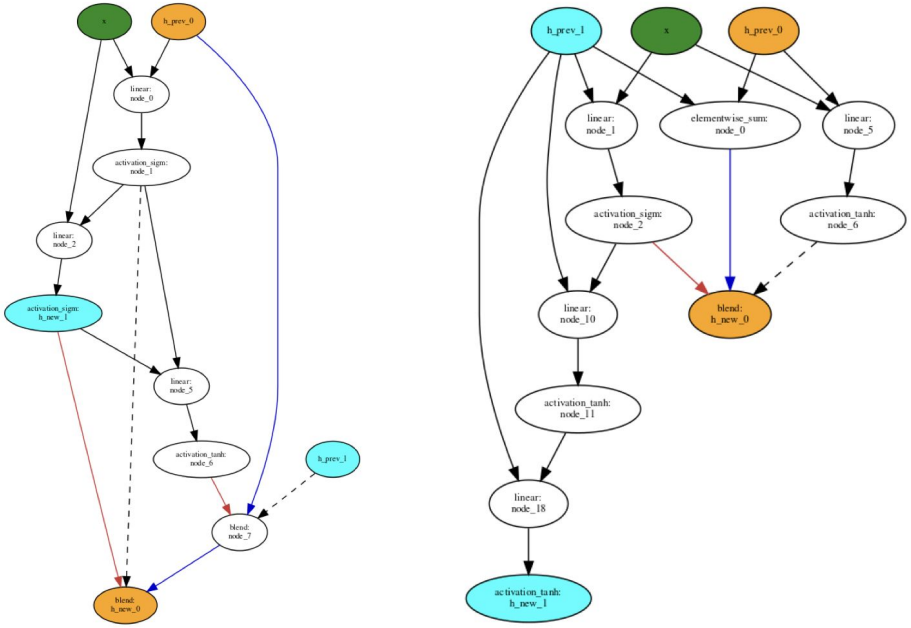
DARTS: Differentiable architecture search, Hanxiao Liu et al, ICLR 2019

NAS benchmarking

Method	Search (seconds)	CIFAR-10		CIFAR-100		ImageNet-16-120	
		validation	test	validation	test	validation	test
RSPS	8007.13	80.42±3.58	84.07±3.61	52.12±5.55	52.31±5.77	27.22±3.24	26.28±3.09
DARTS-V1	11625.77	39.77±0.00	54.30±0.00	15.03±0.00	15.61±0.00	16.43±0.00	16.32±0.00
DARTS-V2	35781.80	39.77±0.00	54.30±0.00	15.03±0.00	15.61±0.00	16.43±0.00	16.32±0.00
GDAS	31609.80	89.89±0.08	93.61±0.09	71.34±0.04	70.70±0.30	41.59±1.33	41.71±0.98
SETN	34139.53	84.04±0.28	87.64±0.00	58.86±0.06	59.05±0.24	33.06±0.02	32.52±0.21
ENAS	14058.80	37.51±3.19	53.89±0.58	13.37±2.35	13.96±2.33	15.06±1.95	14.84±2.10
RSPS†	7587.12	84.16±1.69	87.66±1.69	59.00±4.60	58.33±4.34	31.56±3.28	31.14±3.88
DARTS-V1†	10889.87	39.77±0.00	54.30±0.00	15.03±0.00	15.61±0.00	16.43±0.00	16.32±0.00
DARTS-V2†	29901.67	39.77±0.00	54.30±0.00	15.03±0.00	15.61±0.00	16.43±0.00	16.32±0.00
GDAS†	28925.91	90.00±0.21	93.51±0.13	71.14±0.27	70.61±0.26	41.70±1.26	41.84±0.90
SETN†	31009.81	82.25±5.17	86.19±4.63	56.86±7.59	56.87±7.77	32.54±3.63	31.90±4.07
ENAS†	13314.51	39.77±0.00	54.30±0.00	15.03±0.00	15.61±0.00	16.43±0.00	16.32±0.00
REA	0.02	91.19±0.31	93.92±0.30	71.81±1.12	71.84±0.99	45.15±0.89	45.54±1.03
RS	0.01	90.93±0.36	93.70±0.36	70.93±1.09	71.04±1.07	44.45±1.10	44.57±1.25
REINFORCE	0.12	91.09±0.37	93.85±0.37	71.61±1.12	71.71±1.09	45.05±1.02	45.24±1.18
BOHB	3.59	90.82±0.53	93.61±0.52	70.74±1.29	70.85±1.28	44.26±1.36	44.42±1.49
ResNet	N/A	90.83	93.97	70.42	70.86	44.53	43.63
optimal		91.61	94.37	73.49	73.51	46.77	47.31

Nas-Bench-201: Extending the scope of reproducible neural architecture search, Xuanyi Dong and Yi Yang, ICLR 2020

NAS benchmarks



(a) Top-2 (b) Top-4

Figure 11: The top non-conventional architectures according to test perplexity.

NAS-Bench-NLP: Neural Architecture Search Benchmark for Natural Language Processing, Klyuchnikov et al, IEEE 2022

Результаты*

Задача	Датасет	Человек	NAS	Прирост, %
Машинный перевод	WMT’14	BLEU=28.8 Perplexity=4.05	29.0 3.94	+0.7 -3.0**
Классификация картинок	CIFAR-10	Accuracy=96.54	97.24	+0.7
Семантическая сегментация	Cityscapes	mIOU=71.8	80.4	**+8.6**
Моделирование естественного языка	Penn Treebank	Perplexity=56.0	55.8	-0.3**

* Легко могли устареть

**Для perplexity меньшее значение лучше

Результаты*

Задача	Датасет	Человек	NAS	Прирост, %
Классификация графов	Citeseer	Accuracy=73.0	73.8	+1
RL	Atari	Average reward=172.8	181.8	**+5.2**
Детекция объектов	CoCo	Average precision=0.064	0.078	+1.4

* Легко могли устареть

Результаты

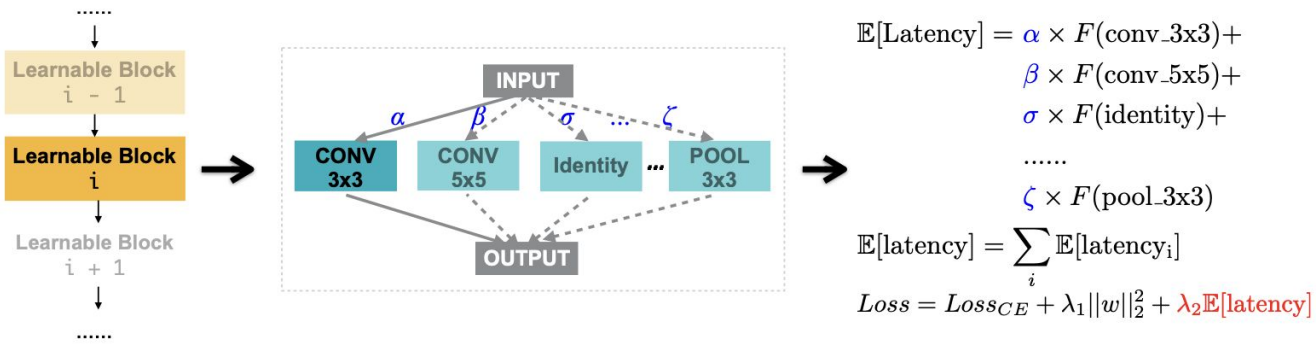


Figure 3: Making latency differentiable by introducing latency regularization loss.

ProxylessNAS: Direct Neural Architecture Search On Target Task And Hardware, Han Cai Dong et al, ICLR 2019

Результаты

To make latency differentiable, we model the latency of a network as a continuous function of the neural network dimensions [3]. Consider a mixed operation with a candidate set $\{o_j\}$ and each o_j is associated with a path weight p_j which represents the probability of choosing o_j. As such, we have the expected latency of a mixed operation (i.e. a learnable block) as:

$$\mathbb{E}[\text{latency}_i] = \sum_j p_j^i \times F(o_j^i), \tag{5}$$

where $\mathbb{E}[\text{latency}_i]$ is the expected latency of the i^{th} learnable block, $F(\cdot)$ denotes the latency prediction model and $F(o_j^i)$ is the predicted latency of o_j^i. The gradient of $\mathbb{E}[\text{latency}_i]$ w.r.t. architecture parameters can thereby be given as: $\partial\mathbb{E}[\text{latency}_i] \,/\, \partial p_j^i = F(o_j^i)$.

For the whole network with a sequence of mixed operations (Figure 3 left), since these operations are executed sequentially during inference, the expected latency of the network can be expressed with the sum of these mixed operations' expected latencies:

$$\mathbb{E}[\text{latency}] = \sum_i \mathbb{E}[\text{latency}_i], \tag{6}$$

ProxylessNAS: Direct Neural Architecture Search On Target Task And Hardware, Han Cai Dong et al, ICLR 2019

Результаты

Model	Params	Test error (%)
DenseNet-BC (Huang et al., 2017)	25.6M	3.46
PyramidNet (Han et al., 2017)	26.0M	3.31
Shake-Shake + c/o (DeVries & Taylor, 2017)	26.2M	2.56
PyramidNet + SD (Yamada et al., 2018)	26.0M	2.31
ENAS + c/o (Pham et al., 2018)	4.6M	2.89
DARTS + c/o (Liu et al., 2018c)	3.4M	2.83
NASNet-A + c/o (Zoph et al., 2018)	27.6M	2.40
PathLevel EAS + c/o (Cai et al., 2018b)	14.3M	2.30
AmoebaNet-B + c/o (Real et al., 2018)	34.9M	2.13
Proxyless-R + c/o (ours)	5.8M	2.30
Proxyless-G + c/o (ours)	5.7M	**2.08**

Table 1: ProxylessNAS achieves state-of-the-art performance on CIFAR-10.

ProxylessNAS: Direct Neural Architecture Search On Target Task And Hardware, Han Cai Dong et al, ICLR 2019

Результаты

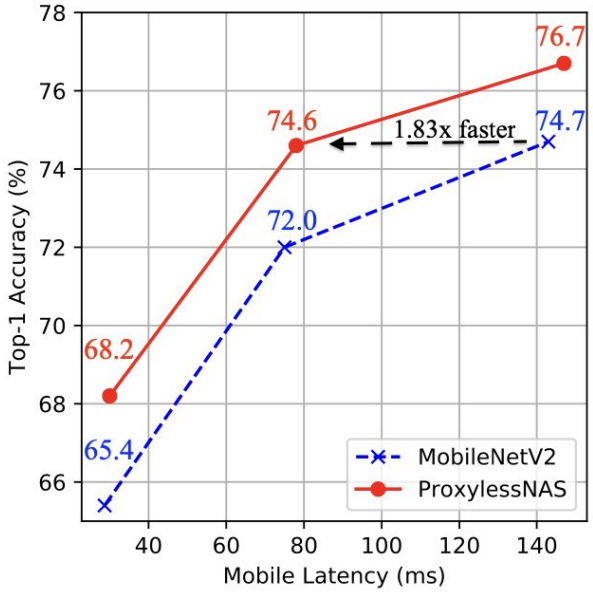
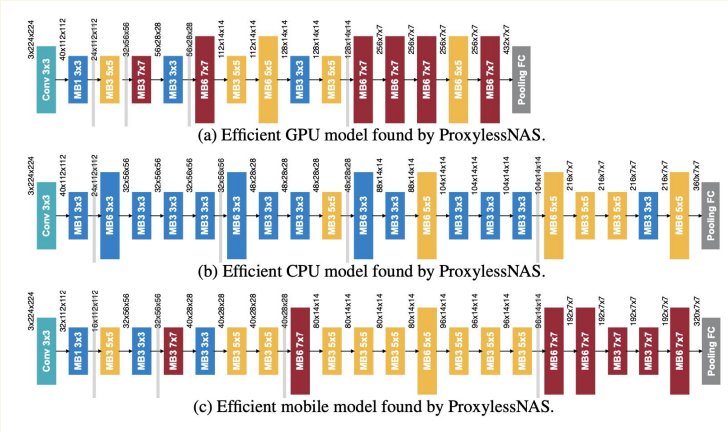


Figure 4: ProxylessNAS consistently outperforms MobileNetV2 under various latency settings.



Model	Top-1 (%)	GPU latency	CPU latency	Mobile latency
Proxyless (GPU)	75.1	5.1ms	204.9ms	124ms
Proxyless (CPU)	75.3	7.4ms	138.7ms	116ms
Proxyless (mobile)	74.6	7.2ms	164.1ms	78ms

ProxylessNAS: Direct Neural Architecture Search On Target Task And Hardware, Han Cai Dong et al, ICLR 2019

Спасибо за внимание!

Если остались вопросы, то задавайте их в чат или пишите преподавателю в tg: @trandelik