

Мета-обучение

НИС МОП 2020

Артем Стрельцов

Что такое мета-обучение

«Учимся учиться»

- Хотим, чтобы модель умела «адаптироваться» к новым данным и сохранять на них обобщающую способность. Другими словами, чтобы она умела решать новые задачи («училась учиться»), за счет «мини-сессий» обучения на ограниченном наборе данных.
- Пример: классификатор обучили на изображениях, на которых нет кошек, который впоследствии, увидев лишь небольшую выборку картинок с ними, может отличать кошку от не-кошки.
- Каждая задача ассоциируется со своими данными D — векторы признаков и метки.
- В контексте мета-обучения оптимальные параметры модели можно записать как $\theta^* = \arg \min_{\theta} \mathbb{E}_{D \sim p(D)} [L_{\theta}(D)]$. Иными словами один набор данных теперь интерпретируется как один пример.

Пример: few shot learning

Training

Train dataset #1: "cat-bird"

cats



birds



Train dataset #2: "flower-bike"

flowers



bikes



Testing

Test dataset: "dog-otter"

dogs

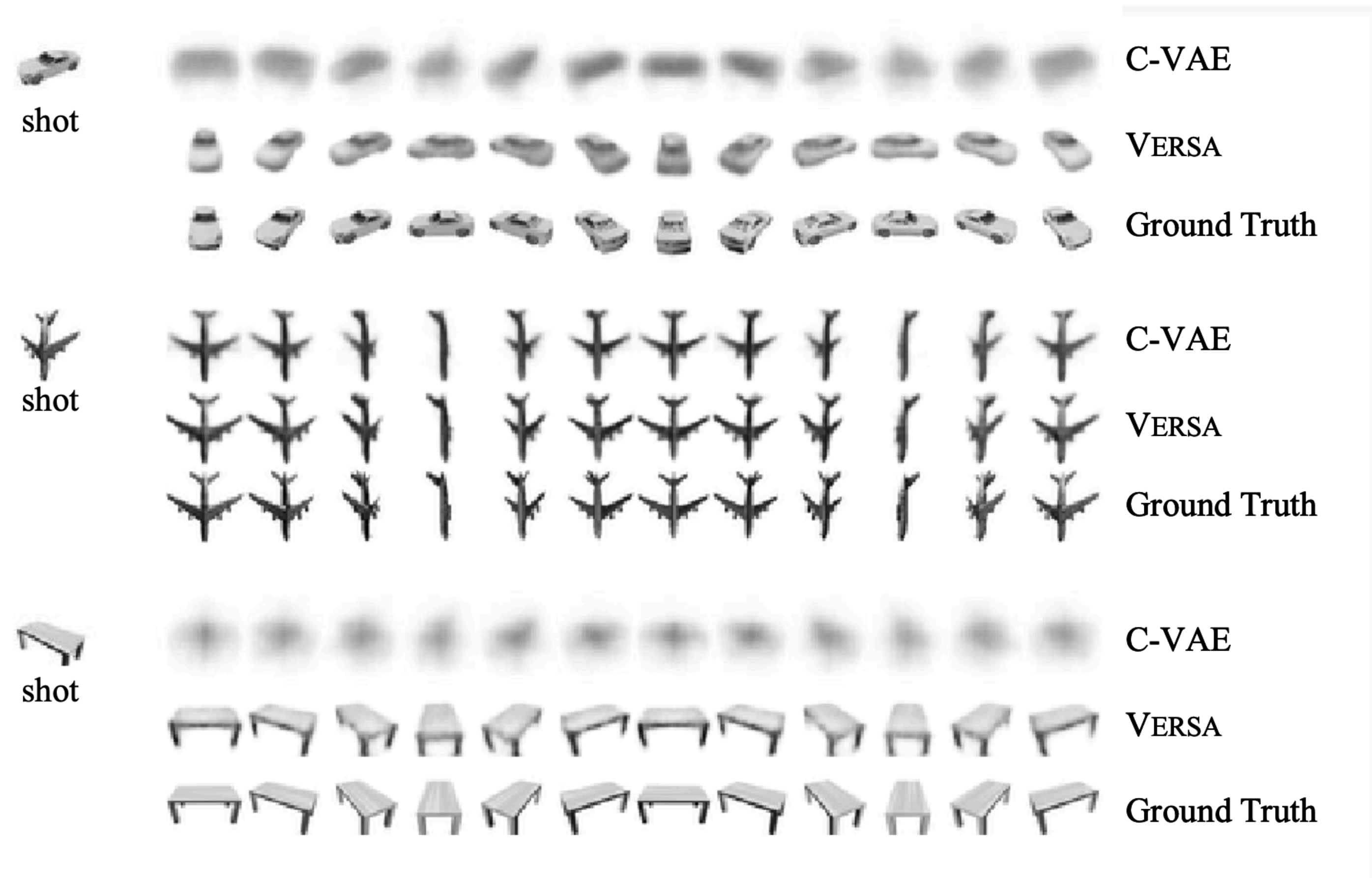


otters



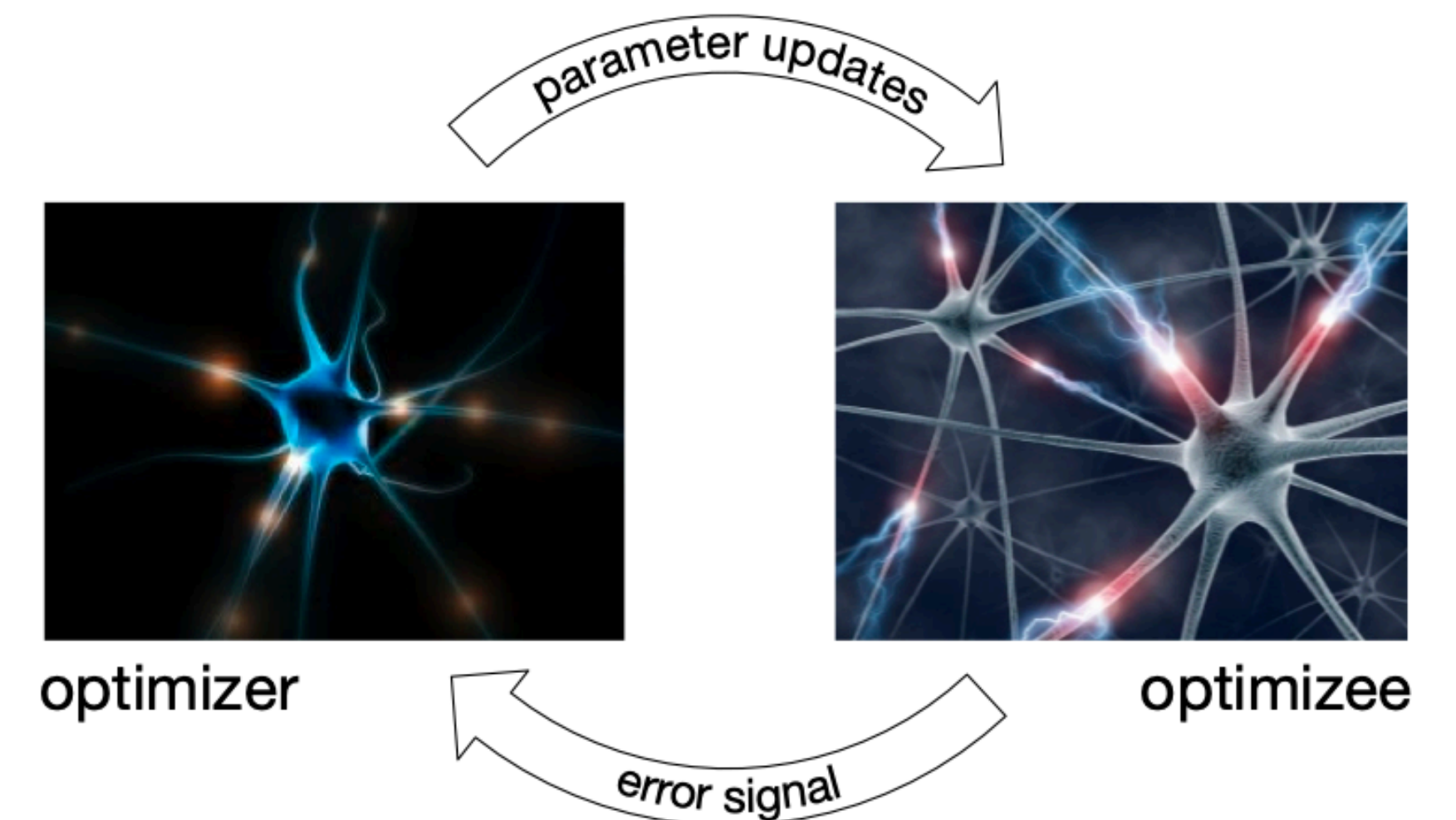
Пример: VERSA

Реконструкция объекта под разными углами



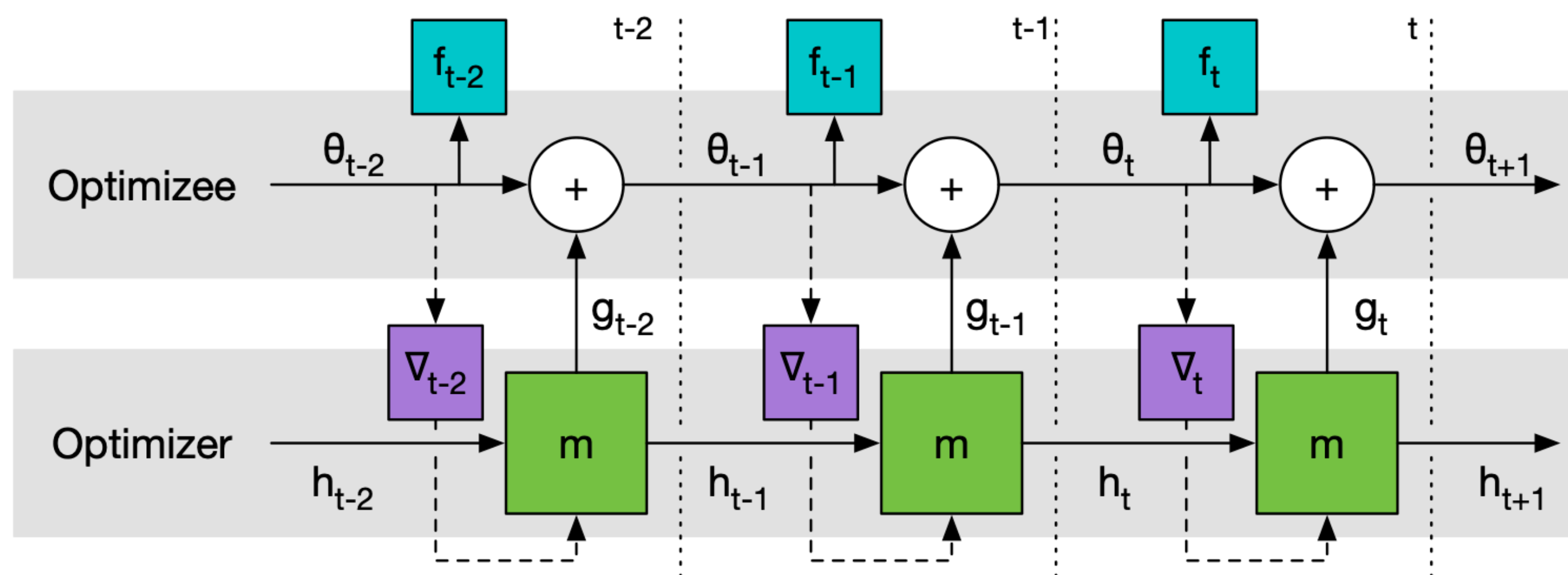
Градиентный спуск

- Классический градиентный спуск: $\theta_{t+1} = \theta_t - \alpha_t \nabla f(\theta_t)$.
- Вместо этого будем обновлять как $\theta_{t+1} = \theta_t + g_t(\nabla f(\theta_t), \phi)$. Где g — оптимизатор с набором параметров ϕ .
- Ищем $\theta^*(f, \phi)$, где f — оптимизируемая функция, ϕ — параметры оптимизатора.
- Функция потерь: $L(\phi) = \mathbb{E}_f[f(\theta^*(f, \phi))]$.
- В качестве оптимизатора g — RNN.



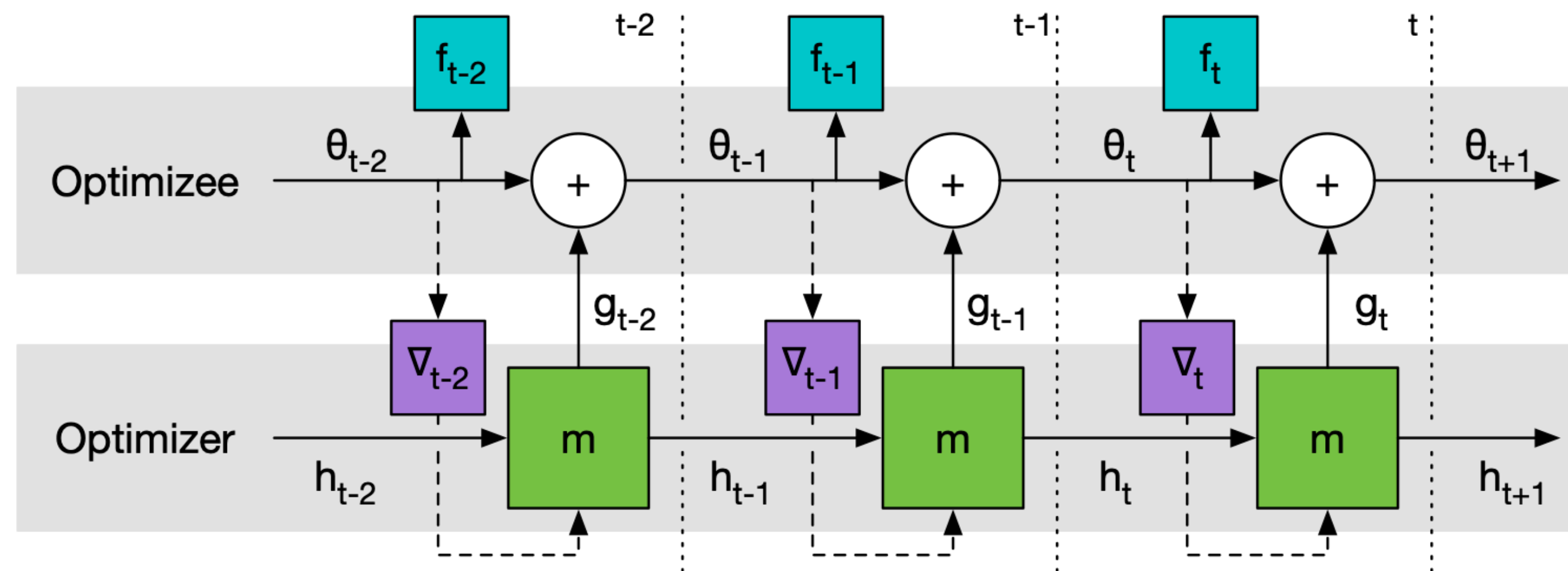
Градиентный спуск

- $L(\phi) = \mathbb{E}_f \left(\sum_{t=1}^T w_t f(\theta_t) \right)$. Частный случай, когда $w = [t = T]$
- $\theta_{t+1} = \theta_t + g_t, \begin{bmatrix} g_t \\ h_{t+1} \end{bmatrix} = m(\nabla_t, h_t, \phi), \nabla_t = \nabla_{\theta} f(\theta_t)$



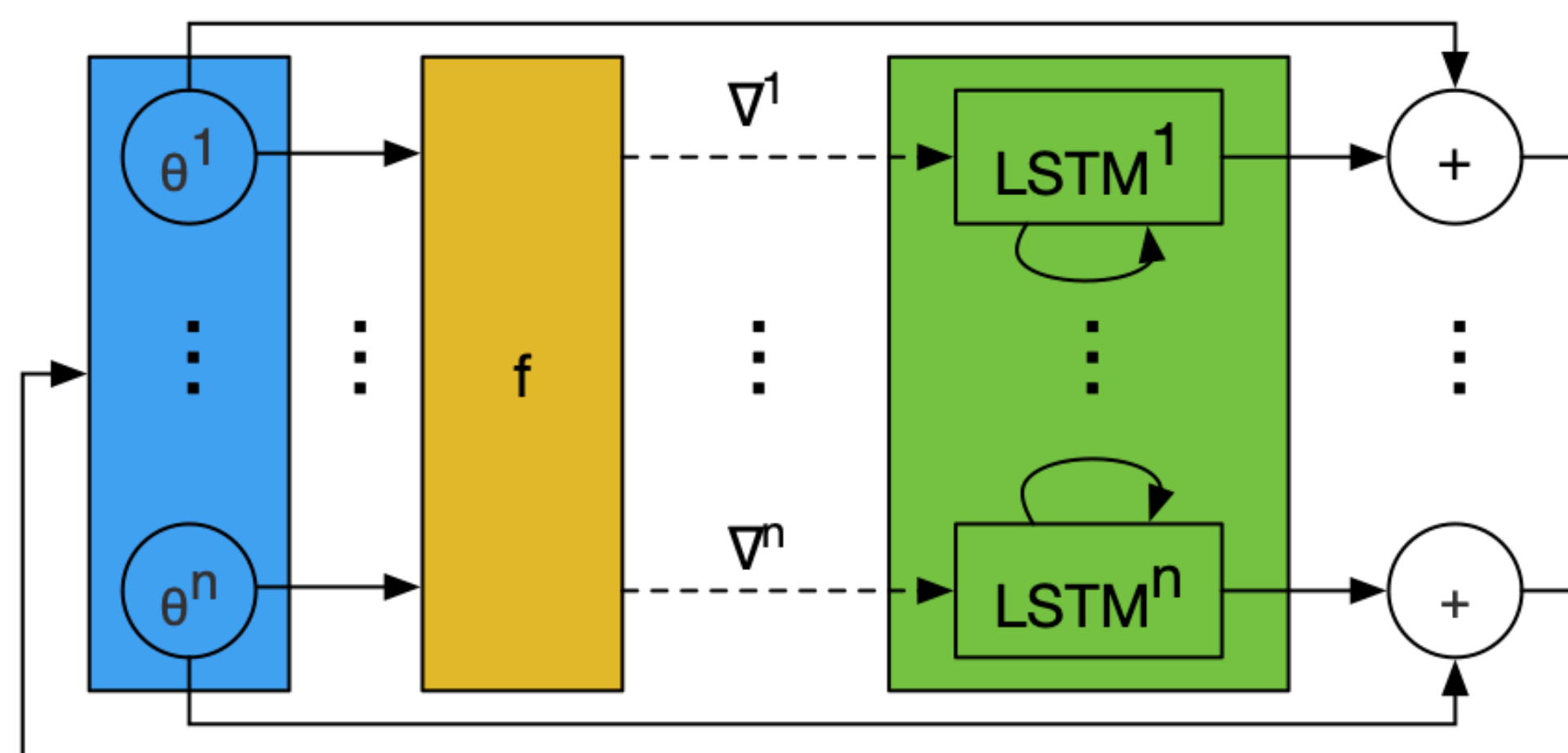
Градиентный спуск

- Минимизируем $L(\phi)$: градиент $\frac{\partial L(\phi)}{\partial \phi}$ считаем с помощью сэмплирования случайной функции f и далее применяем обратное распространение ошибки. При этом можно считать, что градиент f не зависит от параметров ϕ модели
- Значит, $\partial \nabla_t / \partial \phi = 0$, то есть вторые производные и далее можно не считать.

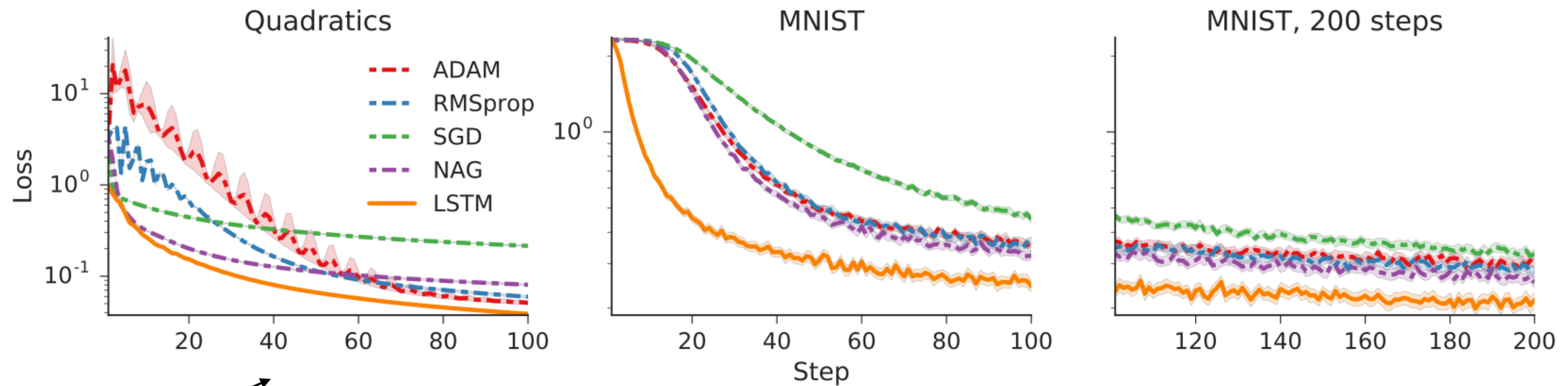


Покоординатная LSTM

- Зачастую оптимизировать надо много, иногда и десятки тысяч параметров.
- Поэтому полносвязная RNN не подходит. Зато можно действовать покоординатно. Для каждого параметра будет своя LSTM, со своими скрытыми состояниями, однако все параметры — доступны для всех LSTM.
- За счет использования RNN можно «выучиться» обновлять градиент, поскольку мы фактически знаем историю его обновлений. Идейно получается очень похоже на Momentum.

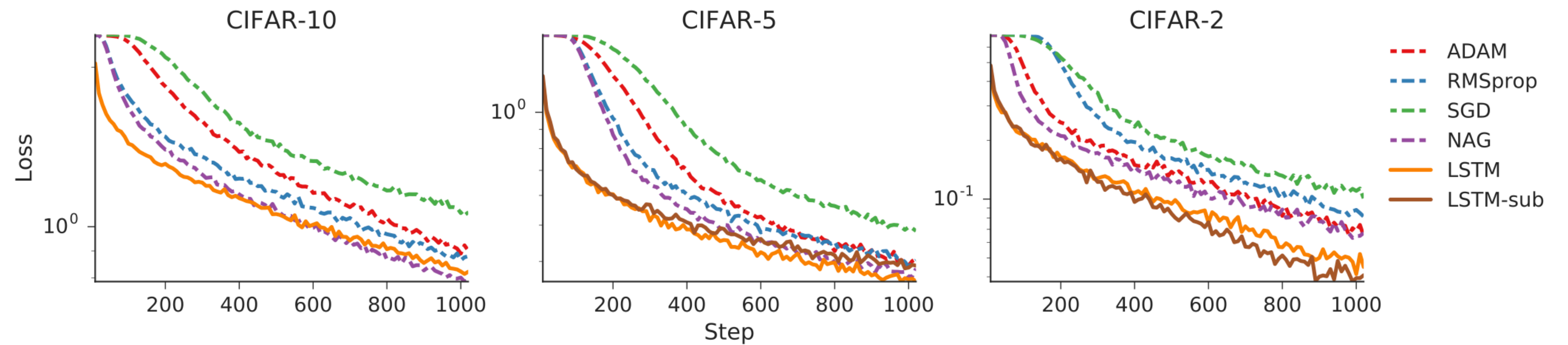


Результаты



$$f(\theta) = \|W\theta - y\|_2^2$$

$$W \in \mathbb{R}^{10 \times 10}, y \in \mathbb{R}^{10}$$



MAML

Model-agnostic Meta-Learning

- Хотим, чтобы модель умела быстро переключаться на новые задачи
- Есть распределение задач $\mathcal{T}_i \sim p(\mathcal{T})$.
- Пусть есть модель f_θ . Адаптируясь к новой задаче, будем получать новый параметр:

$$\theta'_i = \theta - \alpha \nabla_{\theta} \mathcal{L}_{\mathcal{T}_i}(f_\theta).$$

- И тогда задача сводится к следующей задаче оптимизации:

$$\min_{\theta} \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta'_i}) = \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta - \alpha \nabla_{\theta} \mathcal{L}_{\mathcal{T}_i}(f_\theta)})$$

MAML

Реализация

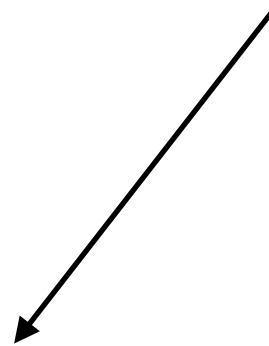
Algorithm 1 Model-Agnostic Meta-Learning

Require: $p(\mathcal{T})$: distribution over tasks

Require: α, β : step size hyperparameters

- 1: randomly initialize θ
 - 2: **while** not done **do**
 - 3: Sample batch of tasks $\mathcal{T}_i \sim p(\mathcal{T})$
 - 4: **for all** \mathcal{T}_i **do**
 - 5: Evaluate $\nabla_{\theta} \mathcal{L}_{\mathcal{T}_i}(f_{\theta})$ with respect to K examples
 - 6: Compute adapted parameters with gradient descent: $\theta'_i = \theta - \alpha \nabla_{\theta} \mathcal{L}_{\mathcal{T}_i}(f_{\theta})$
 - 7: **end for** Note: the meta-update is using different set of data.
 - 8: Update $\theta \leftarrow \theta - \beta \nabla_{\theta} \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta'_i})$
 - 9: **end while**
-

Можно показать, что MAML зависит от вторых производных, поэтому на практике могут использовать FOMAML — MAML первого порядка.


$$g_{MAML} = \nabla_{\theta} L^{(1)}(\theta_k) = [\text{chain rule}] = \nabla_{\theta_k} L^{(1)}(\theta_k) \cdot \prod_{i=1}^k (I - \alpha \nabla_{\theta_{i-1}} (\nabla_{\theta} L^{(0)}(\theta_{i-1})))$$

$$g_{FOMAML} = \nabla_{\theta_k} L^{(1)}(\theta_k)$$

MAML

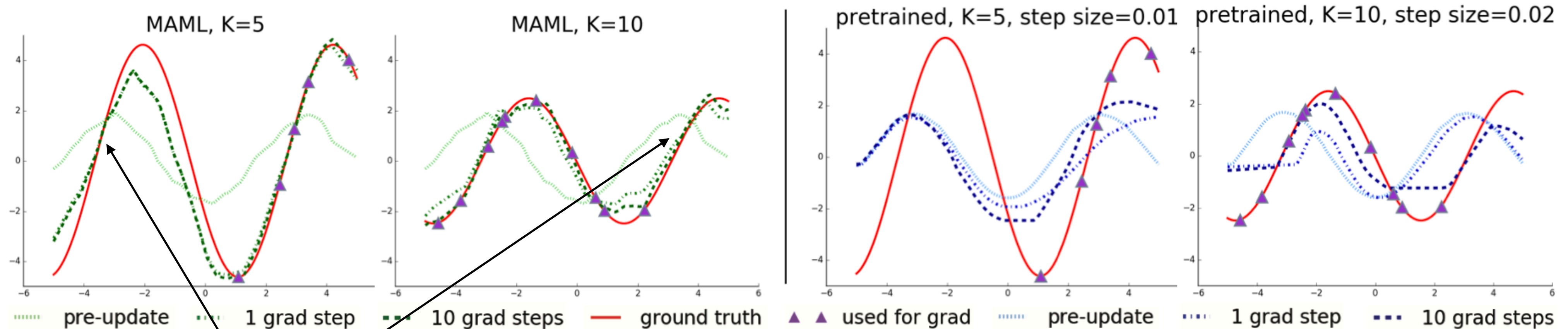
Модификации

- Для регрессии с MSE

$$\mathcal{L}_{\mathcal{T}_i}(f_\phi) = \sum_{\mathbf{x}^{(j)}, \mathbf{y}^{(j)} \sim \mathcal{T}_i} \|f_\phi(\mathbf{x}^{(j)}) - \mathbf{y}^{(j)}\|_2^2,$$

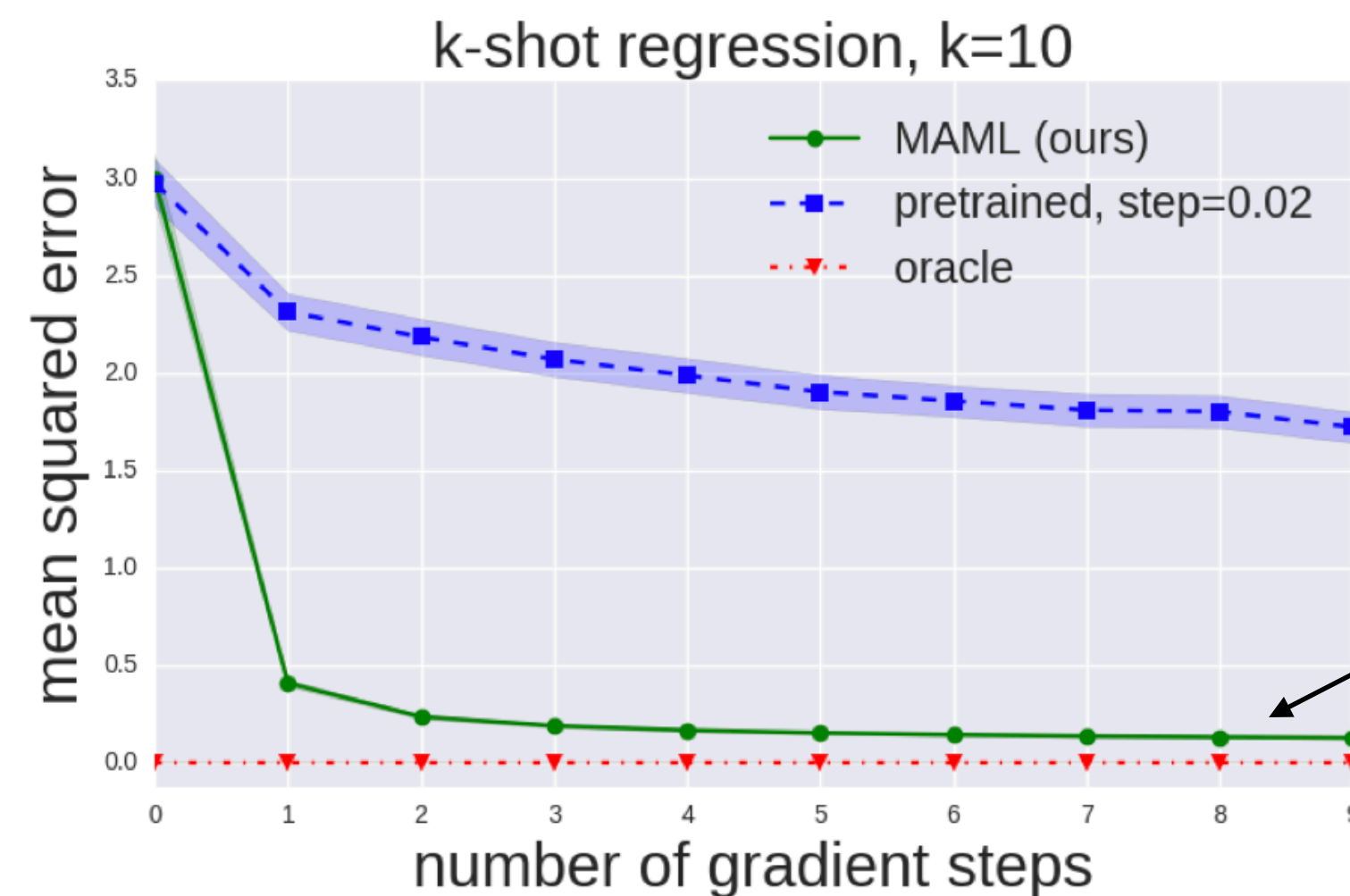
- Можно также K-shot learning, тогда для каждой задачи подаем по K примеров $x^{(j)}, y^{(j)}$
- Классификация с кросс-энтропией:
$$\mathcal{L}_{\mathcal{T}_i}(f_\phi) = \sum_{\mathbf{x}^{(j)}, \mathbf{y}^{(j)} \sim \mathcal{T}_i} \mathbf{y}^{(j)} \log f_\phi(\mathbf{x}^{(j)}) + (1 - \mathbf{y}^{(j)}) \log(1 - f_\phi(\mathbf{x}^{(j)}))$$
- Можно использовать и в обучении с подкреплением.

MAML: результаты



Тут нет точек, но модель умеет экстраполировать

Предобученная
на синусоидных функциях
модель без MAML



При увеличении количества шагов
градиентного спуска во время
мета-тестирования (новой задачи)
не происходит переобучение

MAML: результаты

	5-way Accuracy		20-way Accuracy	
	1-shot	5-shot	1-shot	5-shot
Omniglot (Lake et al., 2011)				
MANN, no conv (Santoro et al., 2016)	82.8%	94.9%	–	–
MAML, no conv (ours)	$89.7 \pm 1.1\%$	$97.5 \pm 0.6\%$	–	–
Siamese nets (Koch, 2015)	97.3%	98.4%	88.2%	97.0%
matching nets (Vinyals et al., 2016)	98.1%	98.9%	93.8%	98.5%
neural statistician (Edwards & Storkey, 2017)	98.1%	99.5%	93.2%	98.1%
memory mod. (Kaiser et al., 2017)	98.4%	99.6%	95.0%	98.6%
MAML (ours)	$98.7 \pm 0.4\%$	$99.9 \pm 0.1\%$	$95.8 \pm 0.3\%$	$98.9 \pm 0.2\%$

	5-way Accuracy	
	1-shot	5-shot
MiniImagenet (Ravi & Larochelle, 2017)		
fine-tuning baseline	$28.86 \pm 0.54\%$	$49.79 \pm 0.79\%$
nearest neighbor baseline	$41.08 \pm 0.70\%$	$51.04 \pm 0.65\%$
matching nets (Vinyals et al., 2016)	$43.56 \pm 0.84\%$	$55.31 \pm 0.73\%$
meta-learner LSTM (Ravi & Larochelle, 2017)	$43.44 \pm 0.77\%$	$60.60 \pm 0.71\%$
MAML, first order approx. (ours)	$48.07 \pm 1.75\%$	$63.15 \pm 0.91\%$
MAML (ours)	$48.70 \pm 1.84\%$	$63.11 \pm 0.92\%$

Reptile

Algorithm 1 Reptile (serial version)

Initialize ϕ , the vector of initial parameters

for iteration = 1, 2, ... **do**

 Sample task τ , corresponding to loss L_τ on weight vectors $\tilde{\phi}$

 Compute $\tilde{\phi} = U_\tau^k(\phi)$, denoting k steps of SGD or Adam

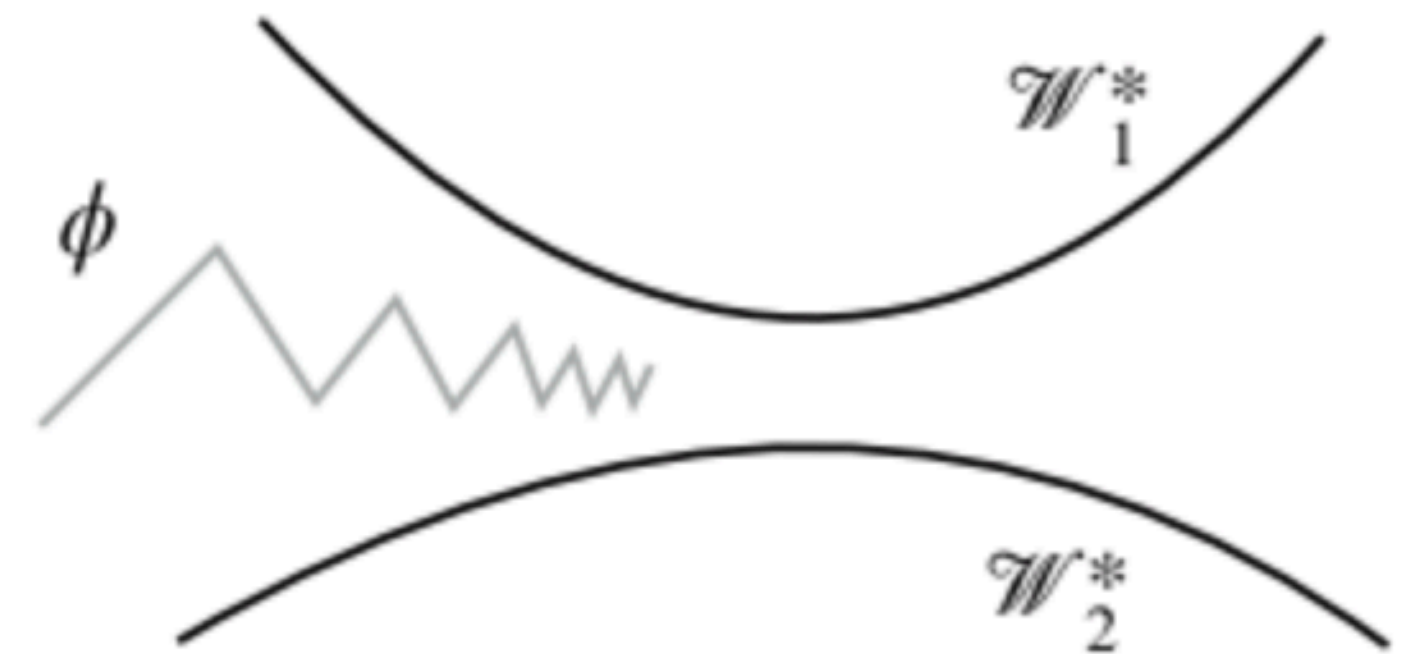
 Update $\phi \leftarrow \phi + \epsilon(\tilde{\phi} - \phi)$

end for

- Здесь U_τ^k — оператор, обновляющий ϕ , используя данные из τ , k раз.
- В нашем алгоритме U соответствует использованию градиентного спуска (Adam или SGD).
- Главная идея: $(\tilde{\phi} - \phi)$ будем считать чем-то вроде градиента.
- Для батчей: $\frac{1}{n} \sum_{i=1}^n (\tilde{\phi} - \phi)$

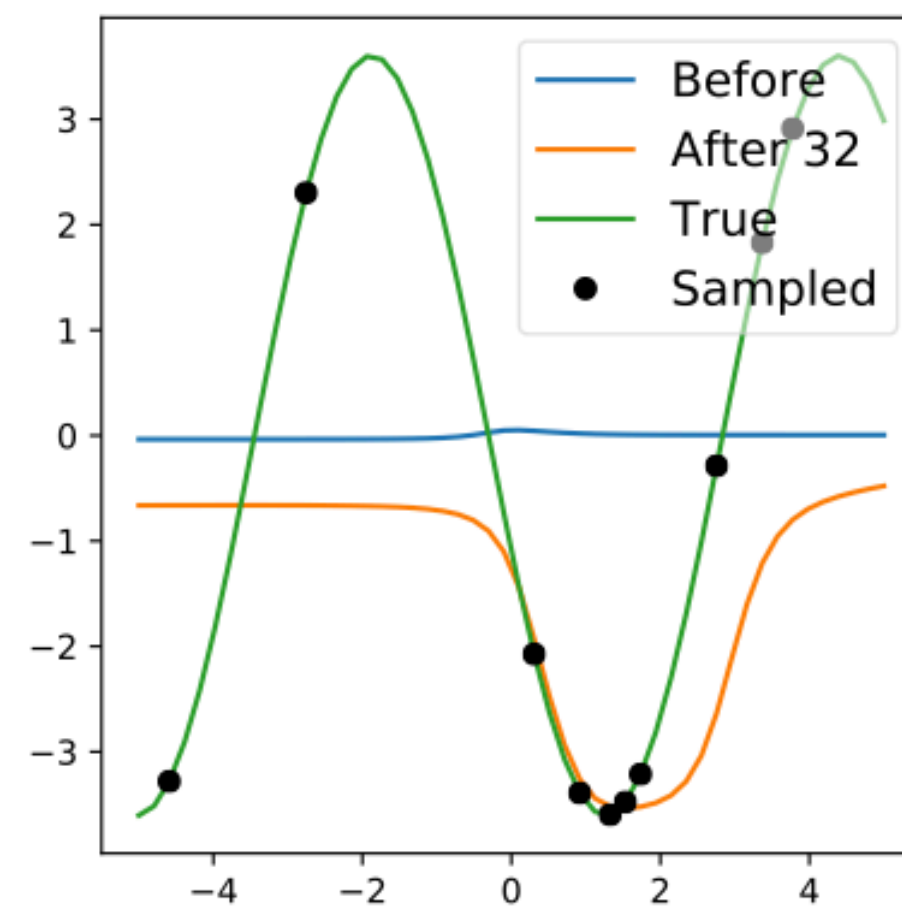
Reptile

- **Важно:** решение как правило отличается от получаемого посредством минимизации $\mathbb{E}_\tau[L_\tau]$. На самом деле, можно показать, что это эквивалентно SGD только при выполнении одной итерации, иначе, как и в MAML, (неявно) зависим от второй производной.
- **Почему Reptile при этом работает?** Для каждой задачи есть свой набор оптимальных весов, при этом, вероятно, далеко не один. Из этого следует предположение, что найдется какой-нибудь, который очень близок к какому-то оптимальному набору весов для каждой из задач.

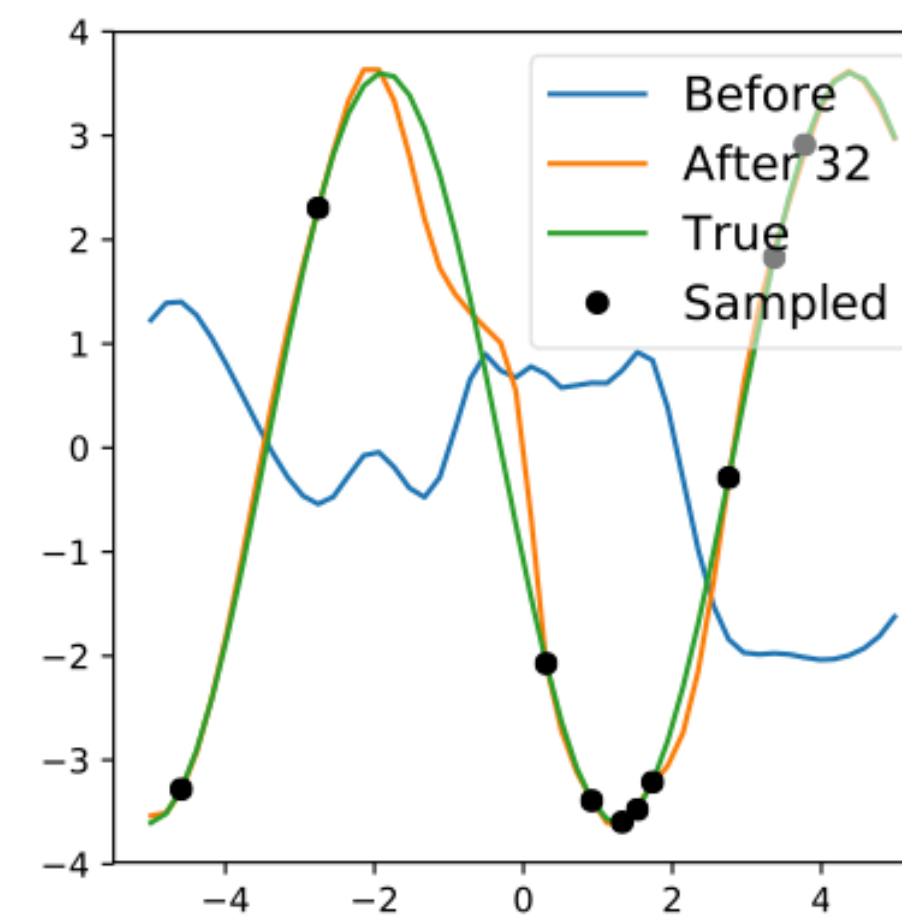


Reptile

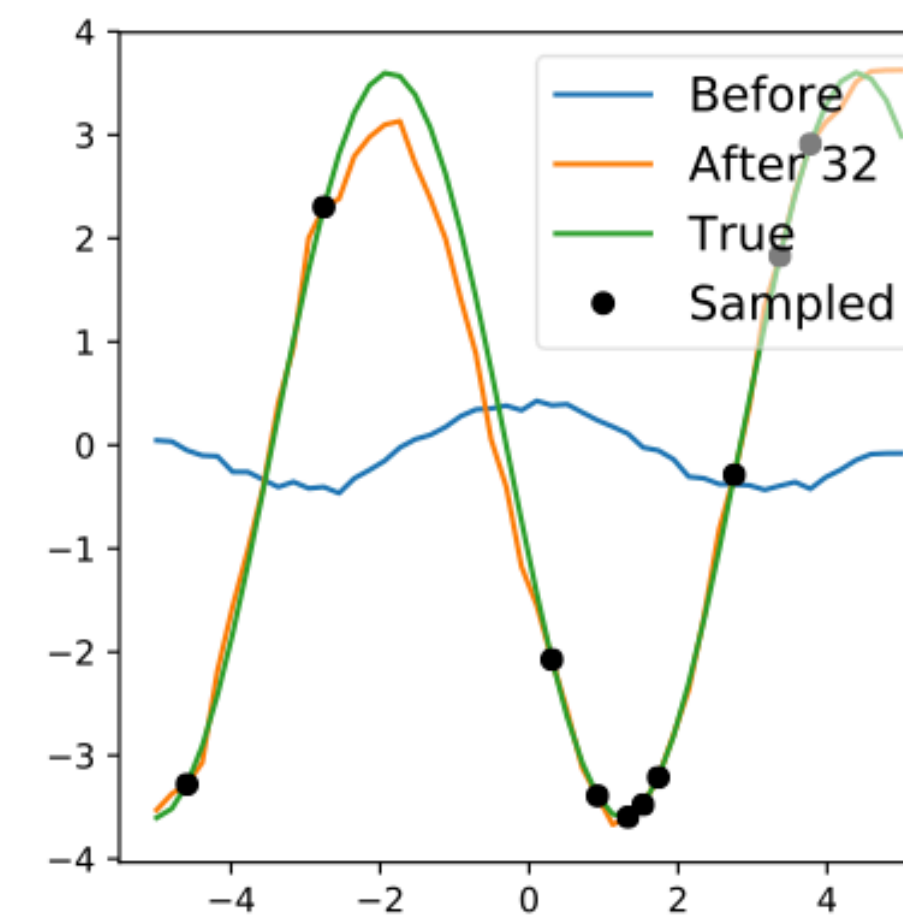
Результаты



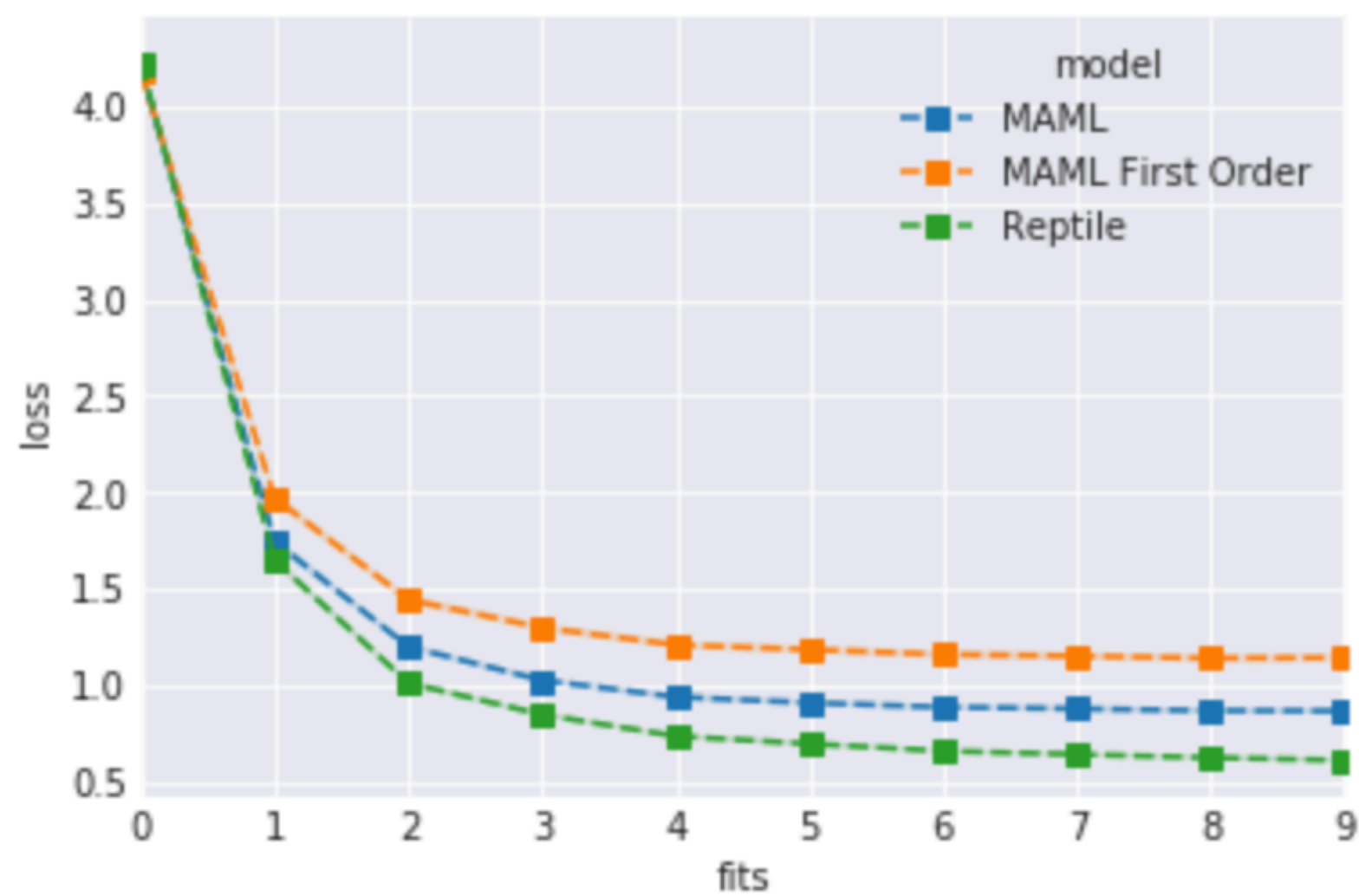
(a) Before training



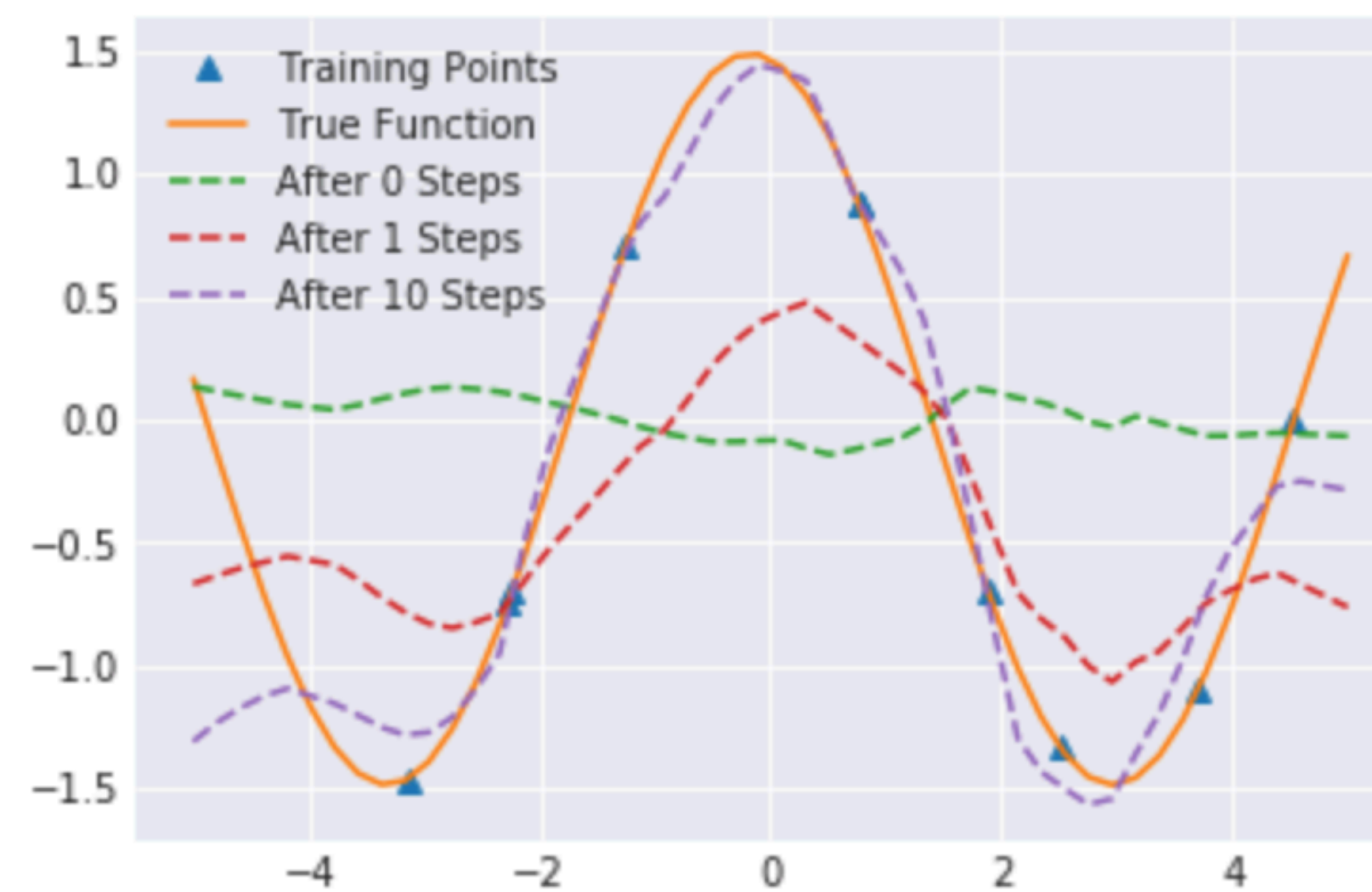
(b) After MAML training



(c) After Reptile training



Learning curve of Reptile, MAML and MAML first-order



MAML vs Reptile

Algorithm	1-shot 5-way	5-shot 5-way
MAML + Transduction	$48.70 \pm 1.84\%$	$63.11 \pm 0.92\%$
1 st -order MAML + Transduction	$48.07 \pm 1.75\%$	$63.15 \pm 0.91\%$
Reptile	$45.79 \pm 0.44\%$	$61.98 \pm 0.69\%$
Reptile + Transduction	$48.21 \pm 0.69\%$	$66.00 \pm 0.62\%$

Table 1: Results on Mini-ImageNet

В таких случаях (трансдукция) модель обращает внимание на специфичные объекты

Algorithm	1-shot 5-way	5-shot 5-way	1-shot 20-way	5-shot 20-way
MAML + Transduction	$98.7 \pm 0.4\%$	$99.9 \pm 0.1\%$	$95.8 \pm 0.3\%$	$98.9 \pm 0.2\%$
1 st -order MAML + Transduction	$98.3 \pm 0.5\%$	$99.2 \pm 0.2\%$	$89.4 \pm 0.5\%$	$97.9 \pm 0.1\%$
Reptile	$95.32 \pm 0.05\%$	$98.87 \pm 0.02\%$	$88.27 \pm 0.30\%$	$97.07 \pm 0.12\%$
Reptile + Transduction	$97.97 \pm 0.08\%$	$99.47 \pm 0.04\%$	$89.36 \pm 0.20\%$	$97.47 \pm 0.10\%$

Table 2: Results on Omniglot

Итог

- Мета-обучение позволяет хорошо подстраиваться под похожие, но ранее неизвестные задачи
- Самые известные методы — MAML и Reptile
- MAML хорош на практике, но требует подсчета вторых производных, что не очень удобно с точки зрения сложности вычислений
- Reptile — всего лишь эвристика
- Трудно сказать, что лучше, MAML или Reptile. Очень сильно зависит от задачи

Список источников

- <https://lilianweng.github.io/lil-log/2018/11/30/meta-learning.html#define-the-meta-learning-problem>
- <https://arxiv.org/pdf/1805.09921.pdf>
- <https://dl.acm.org/doi/pdf/10.5555/3157382.3157543>
- <https://arxiv.org/pdf/1703.03400.pdf>
- <https://arxiv.org/pdf/1803.02999.pdf>