Meta-learning

Медведев Алексей Владимирович

МГУ имени М. В. Ломоносова, факультет ВМК, кафедра ММП

Meta-learning

Многообещающая область машинного обучения. Стандартного определения пока что нет, но идея состоит в том, что мы строим систему, которая может улучшить качество алгоритма при решении конкретной задачи, или адаптировать алгоритм для решения новых задач. Можно сказать, что ставится задача научиться учить (learn-to-learn). Аналогия — эволюционный процесс, создавший человеческий мозг.

Neural Architecture Search

Проблема

Нейронные сети хорошо показали себя в задачах обработки изображений, речи, понимания языка. Но конструирование нейросети остается непростой задачей, требующей определенных навыков.

Идея

Создать алгоритм, который сможет описывать архитектуру нейросети для данной задачи машинного обучения.

Neural Architecture Search

Neural Architecture Search [Barret Zoph, Quoc V. Le 2017]

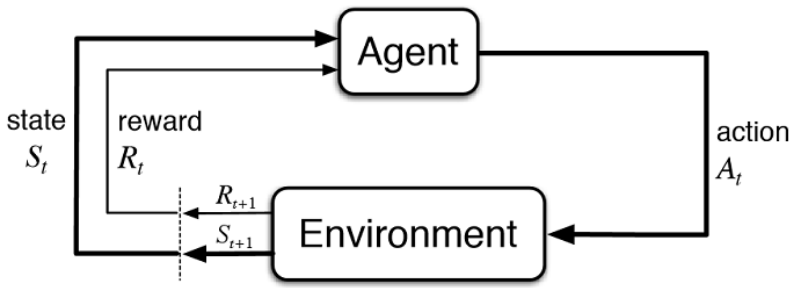
- Архитектура нейросети может быть записана как строка произвольной длины
- Можно взять RNN (controller), каждый блок которой будет отвечать за определенный параметр искомой архитектуры(Количество фильтров в слое сверточной нейросети, с какими слоями соединен данный слой(skip connections) и т.д.)
- Обучить controller с помощью reinforcement-learning, считая что вознаграждение это точность полученной архитектуры на кросс-валидации.

Reinforcement Learning

Введем некоторые обозначения:

- a — действие.
- s — состояние.
- π — стратегия $p(a|s)$.
- r_t — вознаграждение.
- задача максимизировать:

$$J(\theta) = E_{\pi_\theta}[\sum_t r_t]$$



REINFORCE

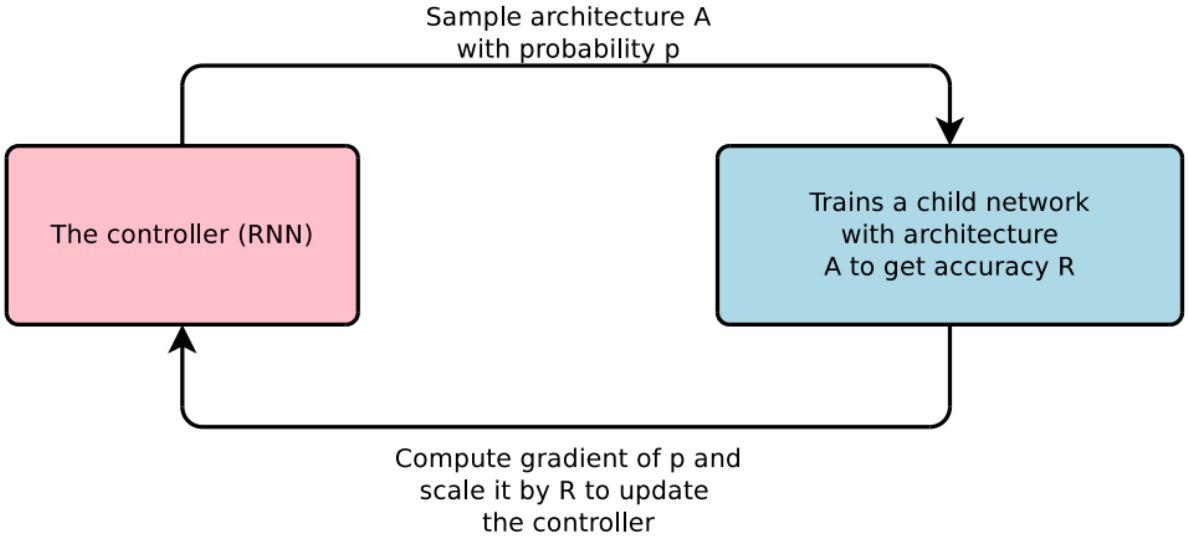
REINFORCE algorithms [Williams, 1992]

- Зачастую функция вознаграждения недифференцируема.
- REINFORCE правила оптимизируют policy-function напрямую.
- Эти правила в общем виде можно записать так:

$$\Delta\theta = \alpha\nabla \log(\pi_\theta) v_t$$

- v_t — текущее значение value функции(оценивает возможную награду).
- $v_t = Q^\pi(s_t, a_t) = E\left[\sum_{i=t} \gamma^{i-t} r_i \,\middle|\, s_t, a_t, \pi\right]$
- α - learning rate.

Neural Architecture Search

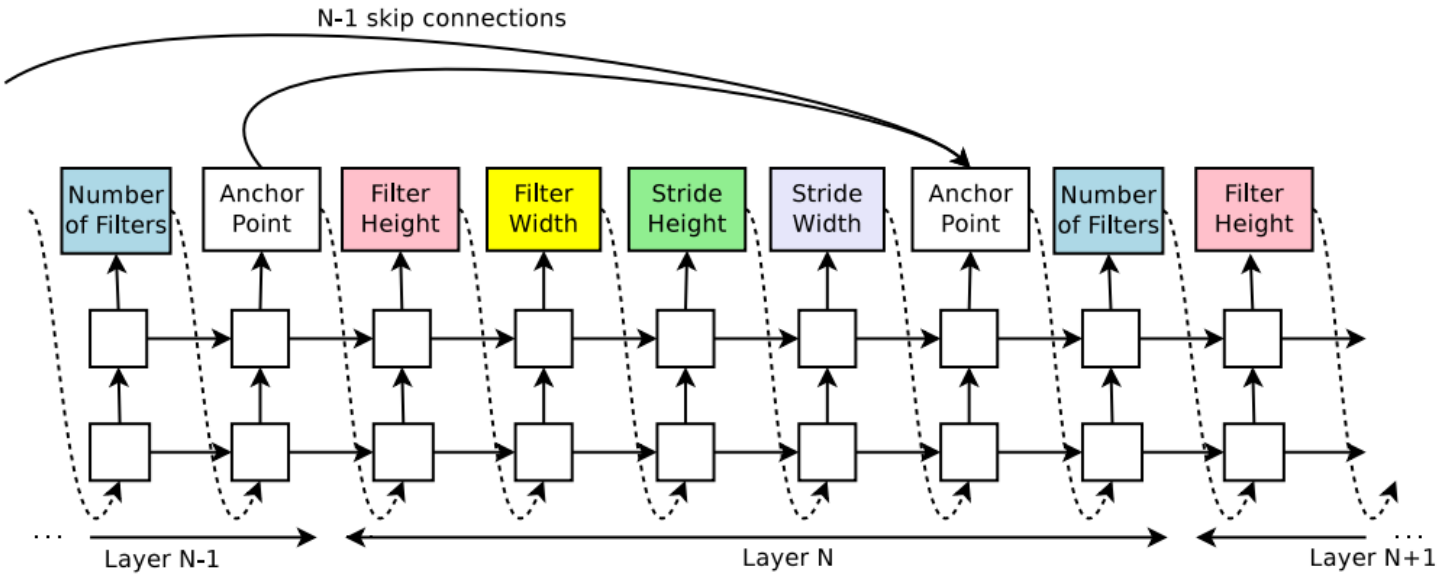


- Градиент функционала принимает следующий вид:

$$\nabla_{\theta_c} J(\theta_c) = \frac{1}{m} \sum_{k=1}^{m} \sum_{t=1}^{T} \nabla_{\theta_c} \log P(a_t | a_{(t-1):1}; \theta_c)(R_k - b)$$

- m — размер batch'а, созданных controller'ом архитектур,
- R_k — вознаграждение, которое получено для данной архитектуры,
- T — количество гиперпараметров, которые controller должен определить,
- P — вероятностная модель, которую моделирует controller.

Neural Architecture Search



- controller представляет собой RNN
- каждый выход — softmax
- каждый блок предсказывает свой параметр конструируемой архитектуры
- блоки сгруппированы по слоям

Neural Architecture Search

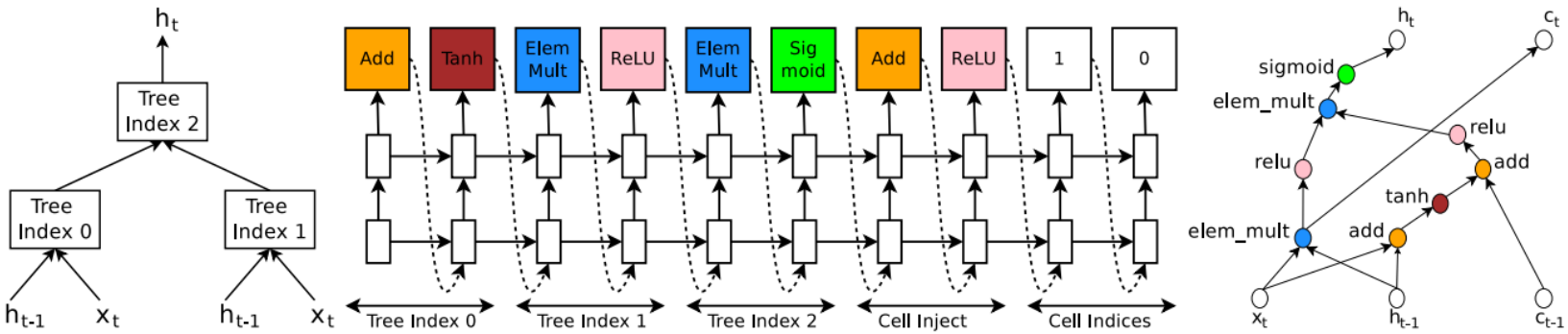


Рис.: Пример архитектуры controller'а, для рекуррентной нейросети.

Neural Architecture Search

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

Рис.: Результаты state-of-the-art моделей и найденных алгоритмом архитектур на датасете CIFAR10(классификация).

Neural Architecture Search

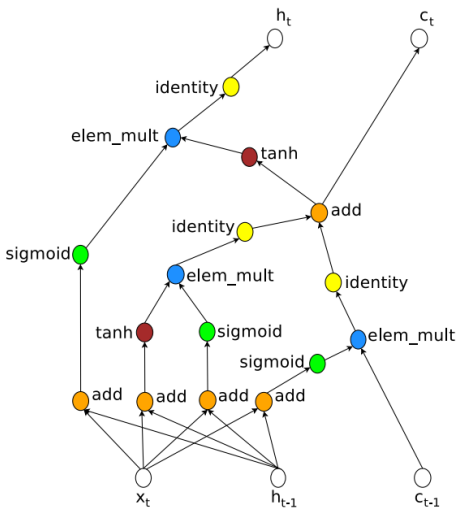
Model	Parameters	Test Perplexity
Mikolov & Zweig (2012) - KN-5	2M‡	141.2
Mikolov & Zweig (2012) - KN5 + cache	2M‡	125.7
Mikolov & Zweig (2012) - RNN	6M‡	124.7
Mikolov & Zweig (2012) - RNN-LDA	7M‡	113.7
Mikolov & Zweig (2012) - RNN-LDA + KN-5 + cache	9M‡	92.0
Pascanu et al. (2013) - Deep RNN	6M	107.5
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

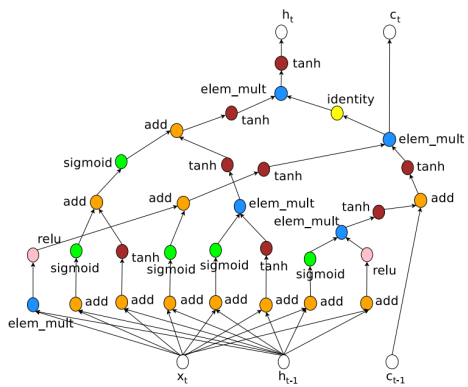
Рис.: Результаты state-of-the-art моделей и найденных алгоритмом архитектур на датасете Penn Treebank.

Neural Architecture Search

Результат

Алгоритм нашел несколько интересных архитектур как в случае классификации, так и в случае более сложной задачи моделирования языка. Более того последние показали еще и хорошие результаты на задаче машинного перевода.





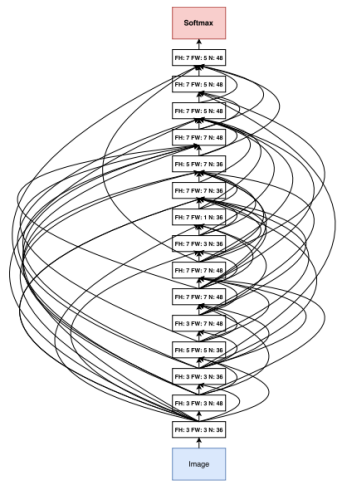


Таблица: LSTM и полученные модели: модуль рекуррентной нейросети и архитектура для CIFAR10

Sim2Real with meta learning

Проблема

- есть задачи, где обучение напрямую либо очень дорого, либо невозможно
- обучение в сложном симуляторе вычислительно затратно
- модель эксплуатирует баги симулятора

Идея

- рандомизировать параметры симулятора

Sim2Real with meta learning

Метод [Peng et al., 2017]

- Промоделируем динамику настоящей среды:

$$\hat{p}(s_{t+1}|a_t, s_t, \mu) \approx p^*(s_{t+1}|a_t, s_t)$$

- μ — множество параметров(масса конечностей робота, масса и трение шайбы, высота стола и т.д)
- Задача сводится к оптимизации:

$$\max_{\pi} E_{\mu \sim \rho_\mu} \left[E_{\tau \sim p(\tau|\pi,\mu)} \left[\sum_{t=0}^{T-1} r(s_t, a_t) \right] \right]$$

- Параметры μ не известны для настоящей среды
- Научимся оценивать μ через историю взаимодействия со средой

$$h_t = [a_{t-1}, s_{t-1}, a_{t-2}, s_{t-2} \ldots]$$

- Для этого наделим policy функцию памятью $\pi(a_t|s_t, z_t)$

Recurrent Deterministic Policy Gradient(RDPG)

Метод [N. Heess et al., 2015]

- Для обучения рекуррентных policy функций есть специальный алгоритм.
- Две обучаемые функции: policy ($\pi(a_t|s_t, z_t)$), value или omniscient critic ($Q(s_t, a_t, y_t, \mu)$), где $y_t = y(h_t)$ — внутренняя память
- Value функция обновляется согласно равенству Беллмана:

$$Q^*(s, a) = E_{s'}\left[r_t + \gamma \max_{a'} Q^*(s', a')\right]$$

Hindsight Experience Replay

Проблема

- Одна из главных проблем в RL — исследование среды(exploration)
- Рандомные действия на первых шагах должны приносить награду, иначе нечего учить
- В случае разреженной функции потерь обучение невозможно

Идея

- Построить модель, которая может попасть в любое состояние, не только целевое
- Например нужно попасть в точку A, попадаем в B, делаем вид что так и надо

Hindsight Experience Replay

Algorithm 1 Hindsight Experience Replay (HER)

Given:

- an off-policy RL algorithm $\mathbb{A}$, ▷ e.g. DQN, DDPG, NAF, SDQN
- a strategy $\mathbb{S}$ for sampling goals for replay, ▷ e.g. $\mathbb{S}(s_0, \ldots, s_T) = m(s_T)$
- a reward function $r : \mathcal{S} \times \mathcal{A} \times \mathcal{G} \to \mathbb{R}$. ▷ e.g. $r(s, a, g) = -[f_g(s) = 0]$

Initialize $\mathbb{A}$ ▷ e.g. initialize neural networks
Initialize replay buffer R
for episode = 1, M **do**
  Sample a goal g and an initial state s_0.
  for $t = 0, T-1$ **do**
    Sample an action a_t using the behavioral policy from $\mathbb{A}$:
      $a_t \leftarrow \pi_b(s_t||g)$ ▷ $||$ denotes concatenation
    Execute the action a_t and observe a new state s_{t+1}
  end for
  for $t = 0, T-1$ **do**
    $r_t := r(s_t, a_t, g)$
    Store the transition $(s_t||g,\ a_t,\ r_t,\ s_{t+1}||g)$ in R ▷ standard experience replay
    Sample a set of additional goals for replay $G := \mathbb{S}$(**current episode**)
    for $g' \in G$ **do**
      $r' := r(s_t, a_t, g')$
      Store the transition $(s_t||g',\ a_t,\ r',\ s_{t+1}||g')$ in R ▷ HER
    end for
  end for
  for $t = 1, N$ **do**
    Sample a minibatch B from the replay buffer R
    Perform one step of optimization using $\mathbb{A}$ and minibatch B
  end for
end for

Рис.: Алгоритм, позволяющий справляться с разреженными функциями вознаграждений.

Hindsight Experience Replay

Experience Replay.mp4

Sim2Real with meta learning

Algorithm 1 Dynamics Randomization with HER and RDPG

$\theta \leftarrow$ random weights
$\varphi \leftarrow$ random weights
while not done **do**
  $g \sim \rho_g$ sample goal
  $\mu \sim \rho_\mu$ sample dynamics
  Generate rollout $\tau = (s_0, a_0, ..., s_T)$with dynamics μ
  for each s_t, a_t in τ **do**
    $r_t \leftarrow r(s_t, g)$
  end for
  Store $(\tau, \{r_t\}, g, \mu)$ in M
  Sample episode $(\tau, \{r_t\}, g, \mu)$ from M
  with probability k
    $g \leftarrow$ replay new goal with HER
    $r_t \leftarrow r(s_t, g)$ for each t
  endwith

  for each t **do**
    Compute memories z_t and y_t
    $\hat{a}_{t+1} \leftarrow \pi_\theta(s_{t+1}, z_{t+1}, g)$
    $\hat{a}_t \leftarrow \pi_\theta(s_t, z_t, g)$
    $q_t \leftarrow r_t + \gamma Q_\varphi(s_{t+1}, \hat{a}_{t+1}, y_{t+1}, g, \mu)$
    $\triangle q_t \leftarrow q_t - Q_\varphi(s_t, a_t, y_t, g, \mu)$
  end for
  $\nabla_\varphi = \frac{1}{T} \sum_t \triangle q_t \frac{\partial Q_\varphi(s_t, a_t, y_t, g, \mu)}{\partial \varphi}$
  $\nabla_\theta = \frac{1}{T} \sum_t \frac{\partial Q_\varphi(s_t, \hat{a}_t, y_t, g, \mu)}{\partial a} \frac{\partial \hat{a}_t}{\partial \theta}$
  Update value function and policy with ∇_θ and ∇_φ
end while

Рис.: Основной алгоритм обучения агента

Sim2Real with meta learning

Transfer of Robotic Control with Dynamics Randomization.mp4

Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks

Проблема

- Есть задачи с небольшим количеством данных
- Алгоритм должен уметь быстро, с очень небольшим количеством обучающих примеров, адаптироваться под решение новых задач.
- Например научить классифицировать изображения чайника модель, которая уже умеет классифицировать множество объектов.

Идея

- Можно представить задачу следующим образом: модель f работает с множеством задач T, из распределения $p(T)$.
- Чтобы избежать переобучения и решить новую задачу важно найти параметры модели, сильно влияющие на функции потерь каждой задачи из $p(T)$.
- Не делается никаких предположений, кроме того, что модель параметризована, данный подход можно обобщить на широкий круг задач от классификации до reinforcement learning.

Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks

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**

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**

Рис.: Алгоритм MAML в общем виде

Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks

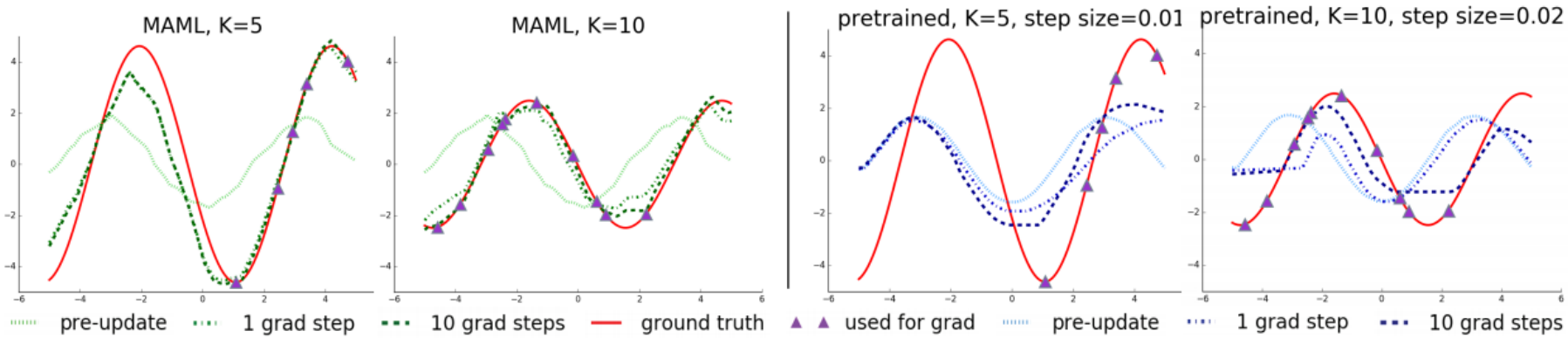


Рис.: Алгоритм MAML для регрессии

Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks

MiniImagenet (Ravi & Larochelle, 2017)	5-way Accuracy	
	1-shot	5-shot
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)	$\mathbf{48.07 \pm 1.75\%}$	$\mathbf{63.15 \pm 0.91\%}$
MAML (ours)	$\mathbf{48.70 \pm 1.84\%}$	$\mathbf{63.11 \pm 0.92\%}$

Рис.: Алгоритм MAML для классификации

Optimization as a model for few-shot learning

Идея [Sachin Ravi and Hugo Larochelle, 2017]

- Научить модель определять способ обновления параметров конечного алгоритма
- Формула шага градиентного спуска:

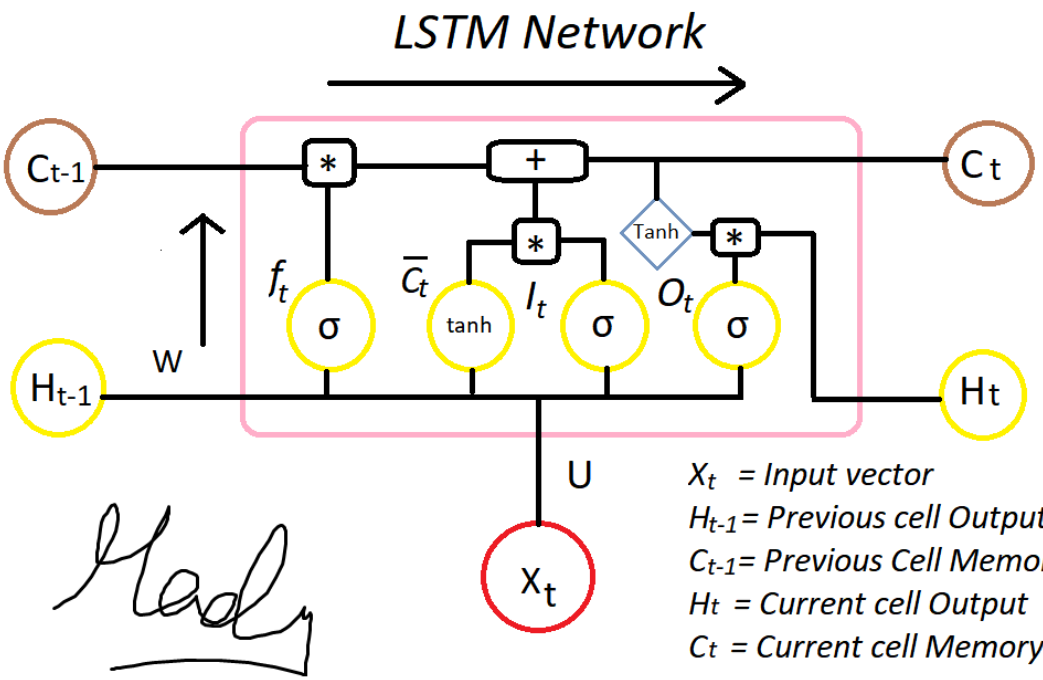
$$\theta_t = \theta_{t-1} - \alpha_t \nabla_{\theta_{t-1}} L_t$$

- Формула обновления памяти ячейки LSTM сети (cell state):

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$$

- c_t будет играть роль параметров сети θ_t, $\tilde{c}_t = \nabla_{\theta_{t-1}} L_t$
- Теперь learning rate зависит от функции потерь, ее градиента, параметров сети и своего значения на предыдущем шаге
- $i_t = \sigma(W_I \cdot \left[\nabla_{\theta_{t-1}} L_t, L_t, \theta_{t-1}, i_{t-1}\right] + b_I)$
- Константа $f_t = 1$ не оптимальна, уменьшение параметров и забывание части предыдущих их значений может оказаться полезным в точке неудачного локального оптимума
- $f_t = \sigma(W_F \cdot \left[\nabla_{\theta_{t-1}} L_t, L_t, \theta_{t-1}, f_{t-1}\right] + b_F)$

Optimization as a model for few-shot learning



X_t = Input vector
H_{t-1} = Previous cell Output
C_{t-1} = Previous Cell Memory
H_t = Current cell Output
C_t = Current cell Memory

(*) = Element-wise multiplication
(+) = Element-wise addition

$$f_t = \sigma (X_t * U_f + H_{t-1} * W_f)$$
$$\bar{C}_t = \tanh (X_t * U_c + H_{t-1} * W_c)$$
$$I_t = \sigma (X_t * U_i + H_{t-1} * W_i)$$
$$O_t = \sigma (X_t * U_o + H_{t-1} * W_o)$$

$$C_t = f_t * C_{t-1} + I_t * \bar{C}_t$$
$$H_t = O_t * \tanh (C_t)$$

W, U = weight vectors for forget gate (f), candidate (c), i/p gate (I) and o/p gate (O)

Note : These are different weights for different gates, for simpicity's sake, I mentioned W and U

Рис.: Архитектура LSTM

Optimization as a model for few-shot learning

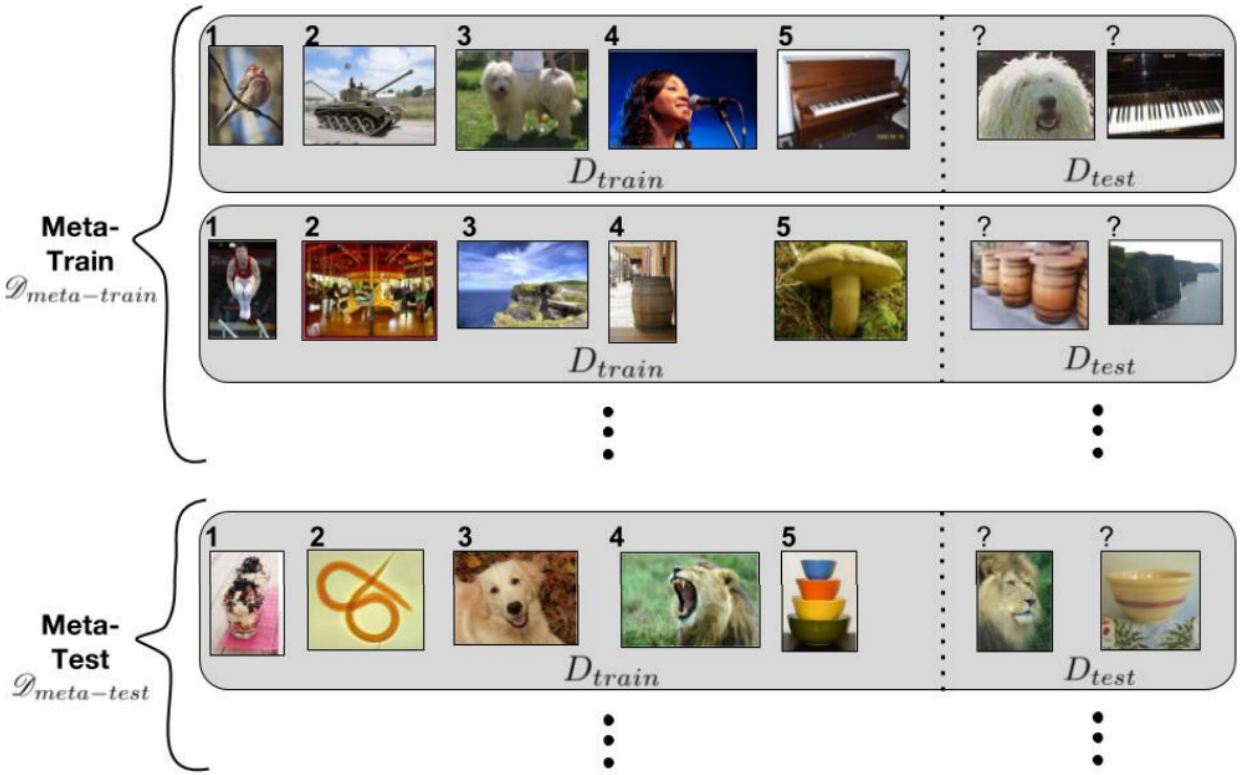


Рис.: Обучение Meta-Learner'а происходит на множестве пар датасетов, называемых эпизодами. Каждая пара состоит из тренировочной и тестовой выборки. В тренировочной выборке присутствуют k*N элементов, где N — количество классов, а k — ограничение на количество элементов из одного класса.

Optimization as a model for few-shot learning

Algorithm 1 Train Meta-Learner

Input: Meta-training set $\mathscr{D}_{meta-train}$, Learner M with parameters θ, Meta-Learner R with parameters Θ.

$\Theta_0 \leftarrow$ random initialization

for $d = 1, n$ **do**
  $D_{train}, D_{test} \leftarrow$ random dataset from $\mathscr{D}_{meta-train}$
  $\theta_0 \leftarrow c_0$ ▷ Intialize learner parameters

  for $t = 1, T$ **do**
    $\mathbf{X}_t, \mathbf{Y}_t \leftarrow$ random batch from D_{train}
    $\mathcal{L}_t \leftarrow \mathcal{L}(M(\mathbf{X}_t; \theta_{t-1}), \mathbf{Y}_t)$ ▷ Get loss of learner on train batch
    $c_t \leftarrow R((\nabla_{\theta_{t-1}} \mathcal{L}_t, \mathcal{L}_t); \Theta_{d-1})$ ▷ Get output of meta-learner using Equation 2
    $\theta_t \leftarrow c_t$ ▷ Update learner parameters
  end for

  $\mathbf{X}, \mathbf{Y} \leftarrow D_{test}$
  $\mathcal{L}_{test} \leftarrow \mathcal{L}(M(\mathbf{X}; \theta_T), \mathbf{Y})$ ▷ Get loss of learner on test batch
  Update Θ_d using $\nabla_{\Theta_{d-1}} \mathcal{L}_{test}$ ▷ Update meta-learner parameters

end for

Рис.: Алгоритм обучения Meta-Learner

Optimization as a model for few-shot learning

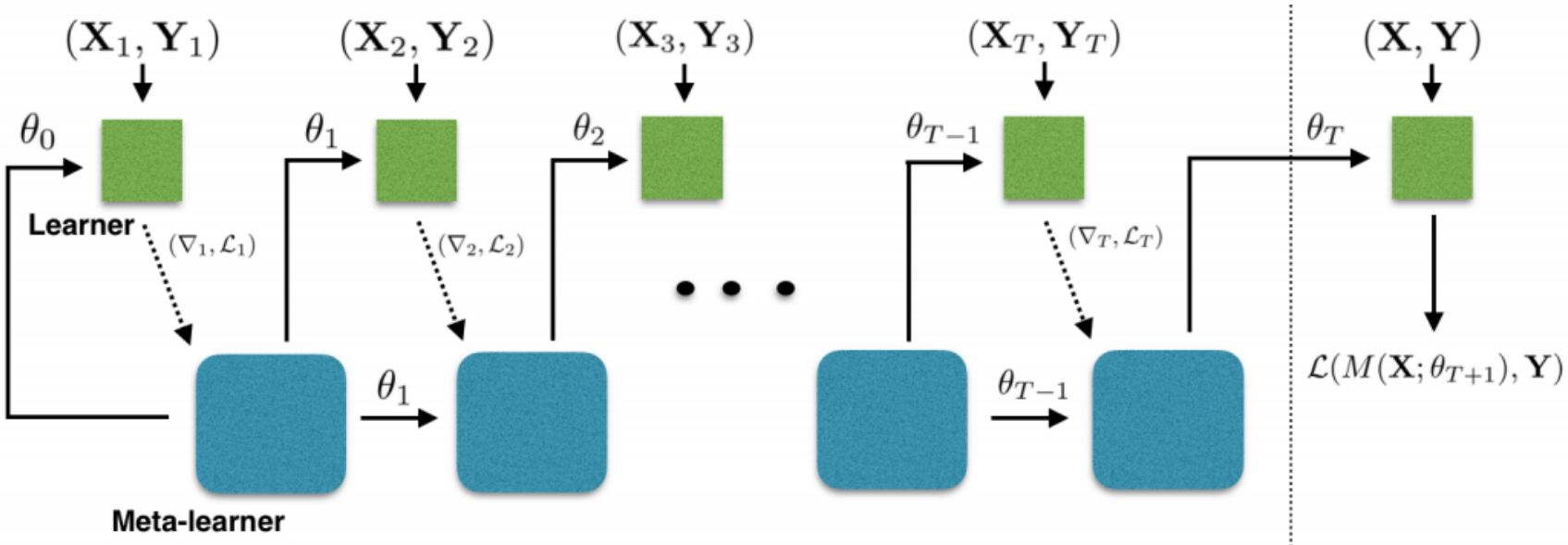


Рис.: Проход вперед(forward pass) meta learner'a. Как видно часть стрелок пунктирные, это означает, что во время обновления весов эти шаги не учитываются. Это позволяет избежать появления вторых производных и сильно упрощает вычисления. Пунктирной линией отделены шаги на тестовой и тренировочной выборках.

Limitations of meta learning

Ограничения

- Распределение задач на тренировочной выборке должно быть тем же что и на тестовой
- Пример случая, когда новая задача **фундаментально** отличается от тренировочных: если мы обучим модель математике, программированию, чтению и т.д. сможет ли модель в результате выучить химию?

References

- N. Heess, J. J. Hunt, T. P. Lillicrap, and D. Silver, «Memory-based control with recurrent neural networks», http://arxiv.org/abs/1512.04455
- Barret Zoph, Quoc V. Le, «Neural Architecture Search», https://openreview.net/pdf?id=r1Ue8Hcxg
- Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel , Wojciech Zaremba, «Hindsight Experience Replay», https://arxiv.org/pdf/1707.01495v3.pdf
- Chelsea Finn, Pieter Abbeel, Sergey Levine, «Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks», https://arxiv.org/pdf/1703.03400.pdf
- Sachin Ravi, Hugo Larochelle, «Optimization as a model for few-shot learning», https://openreview.net/pdf?id=rJY0-Kcll
- Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, Pieter Abbeel1, «Sim-to-Real Transfer of Robotic Control with Dynamics Randomization», https://arxiv.org/pdf/1710.06537.pdf