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.

REINFORCE

REINFORCE algorithms [Williams, 1992]

Чтобы найти оптимальный алгоритм с точки зрения reinforcement-learning необходимо максимизировать мат.ожидание награды: $J(\theta) = E_{\pi_\theta}[R]$.

Где π_θ — policy функция, по сути и есть алгоритм, который мы обучаем.

Зачастую функция вознаграждения(R) недеффиринцируема, поэтому было придумано множество REINFORCE правил, которые позволяют применить градиентный спуск напрямую к policy-function.

Эти правила в общем виде можно записать так:

$$\Delta\theta = \alpha \nabla \log(\pi_\theta)(r - b)$$

Где r — текущий reward, b — baseline(скользящее среднее предыдущих вознаграждений), α - learning rate.

Neural Architecture Search

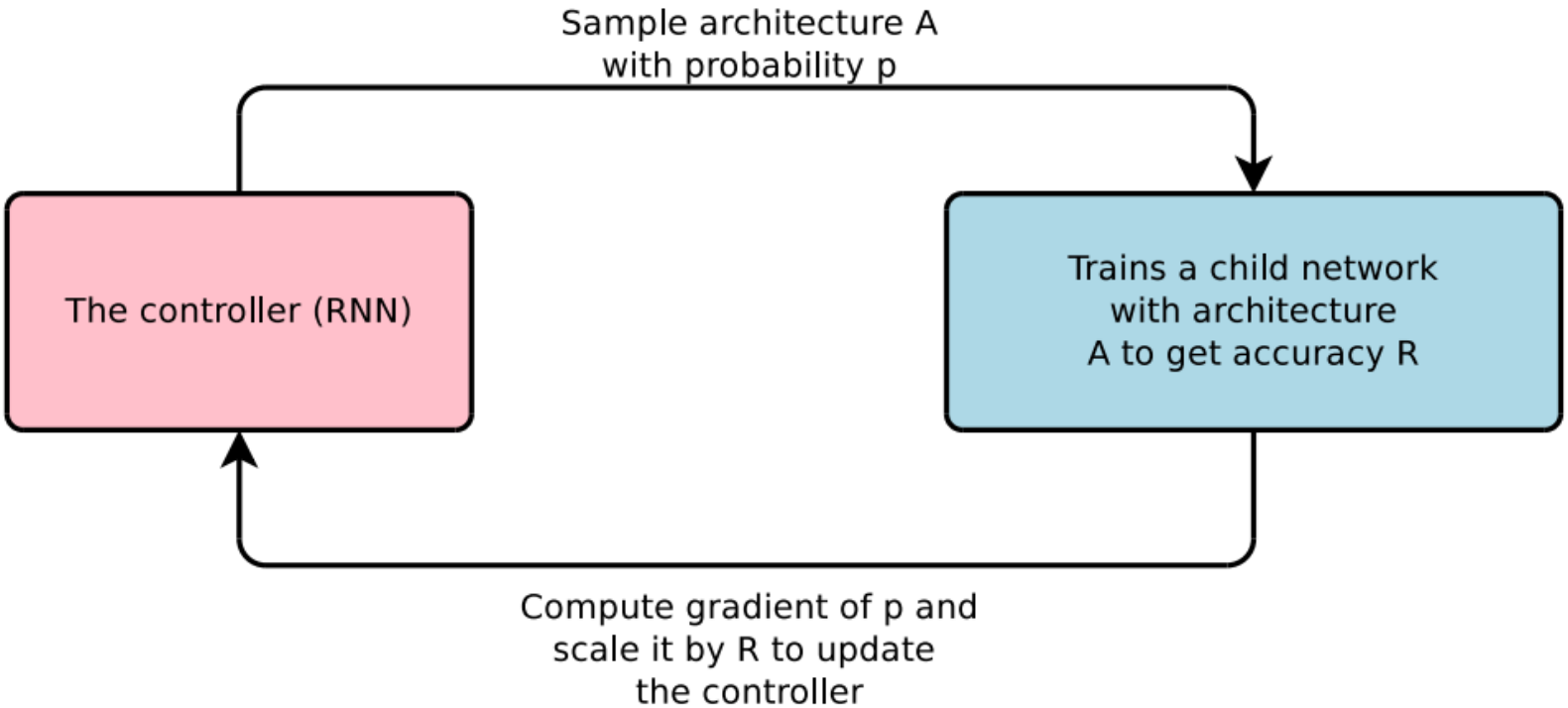


Рис.: Итоговый алгоритм обучения controller-network

Neural Architecture Search

Итоговый алгоритм

С помощью вышеупомянутого REINFORCE rule градиент нашего функционала принимает следующий вид:

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

Где 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

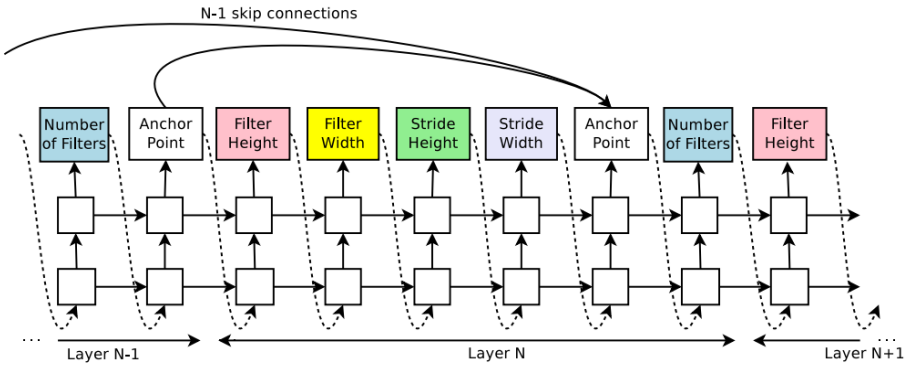


Рис.: Пример архитектуры controller'а, ищущего архитектуру сверточной нейросети.

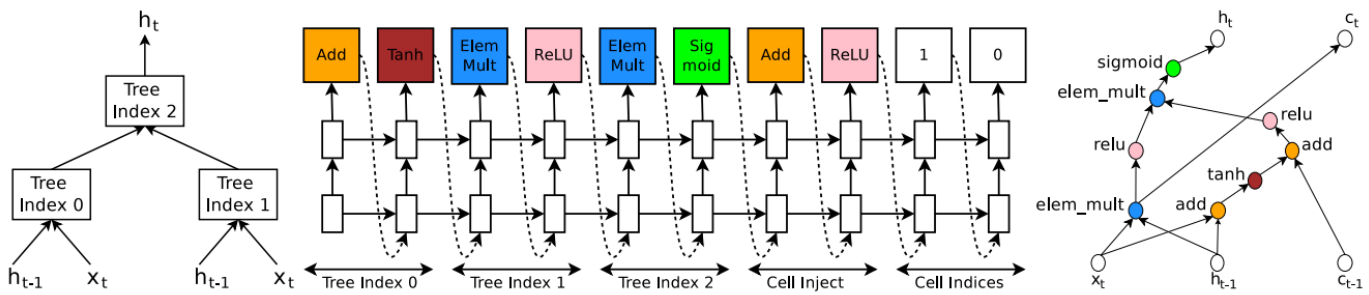


Рис.: Пример архитектуры controller'а, ищущего архитектуру, рекуррентной нейросети.

Neural Architecture Search

Архитектура controller'a

Controller представляет собой RNN, каждый выход которой представляет собой выход softmax классификатора и подается на вход следующему блоку. Каждый блок предсказывает свой параметр конструируемой архитектуры, такие блоки можно сгруппировать по слоям будущей нейросети. В более сложном случае появляются особые блоки, которые отвечают за skip-connections.

Особый случай представляет конструирование рекуррентных нейросетей, блоки на выходе предсказывают функции активации, опреации над внутренней памятью и входом блока 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

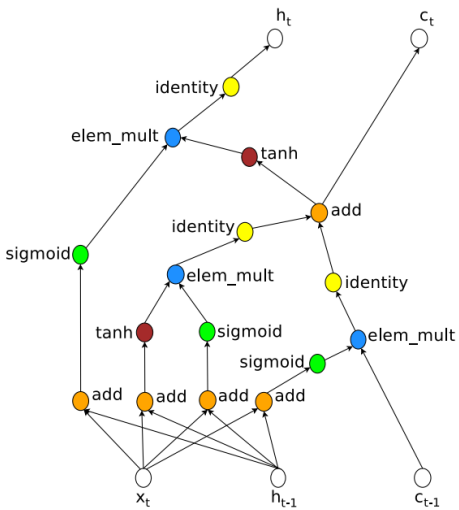
Model	Parameters	Test Perplexity
Mikolov & Zweig (2012) - KN-5	2M‡	141.2
Mikolov & Zweig (2012) - KN5 + cache	2M‡	125.7
Mikolov & Zweig (2012) - RNN	6M‡	124.7
Mikolov & Zweig (2012) - RNN-LDA	7M‡	113.7
Mikolov & Zweig (2012) - RNN-LDA + KN-5 + cache	9M‡	92.0
Pascanu et al. (2013) - Deep RNN	6M	107.5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

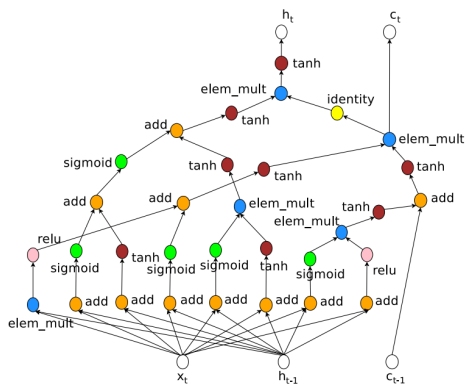
Рис.: Результаты state-of-the-art моделей и найденных алгоритмом архитектур на датасете Penn Treebank.

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.





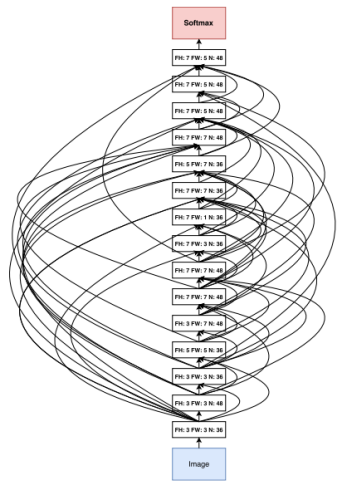


Таблица: LSTM и полученные модели: модуль рекуррентной нейросети и архитектура для CIFAR10

Sim2Real with meta learning

Проблема

Есть задачи, например создание автопилота для вертолета, обучение для которых в реальной жизни либо очень дорого, либо невозможно. В то же время обучение в симуляторе настоящей среды сопряжено с некоторыми проблемами. Чем полнее симуляция среды, тем она тяжелее с вычислительной точки зрения, а значит обучение будет проходить медленно, плюс обучаемые агенты могут использовать упрощения и баги симуляции для создания своих оптимальных, но неприменимых в реальной жизни стратегий. Хотелось бы уметь обучать агента в симуляторе, но так чтобы он преуспел в реальном мире.

Идея

Создать алгоритм, который будет адаптироваться к параметрам среды, в которой он действует. Для этого можно обучить алгоритм на симуляциях среды с рандомизированными параметрами.

Sim2Real with meta learning

Метод [Peng et al., 2017]

Наша цель натренировать policy функции, которые смогут решать задачи, в среде описанной динамикой реального мира $p^*(s_{t+1}|a_t, s_t)$. Но ее использование может быть накладным. Поэтому мы приближаем ее моделью $\hat{p}(s_{t+1}|a_t, s_t, \mu) \approx p^*(s_{t+1}|a_t, s_t)$. Где μ — множество параметров(масса конечностей робота, масса и трение шайбы, высота стола и т.д). В таких терминах можно сформулировать задачу как максимизацию следующего функционала:

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

Действия агента напрямую зависят от параметров функции динамики среды (μ). Т.е policy функция зависит от них. Если во время тренировки параметры μ нам известны, то в реальном мире дела обстоят сложнее. Предлагается оценивать эти параметры с помощью истории действий и состояний $h_t = [a_{t-1}, s_{t-1}, a_{t-2}, s_{t-2}...]$. Для этого сделаем функцию policy рекуррентной $\pi(a_t|s_t, z_t)$, где внутренняя память $z_t = z(h_t)$ и есть механизм, позволяющий определить параметры.

Recurrent Deterministic Policy Gradient(RDPG)

Метод

Для обучения рекуррентных policy функций есть специальный алгоритм. Чтобы его применить нужны 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. Формулы обновления весов будут приведены далее в описании общего алгоритма. Идея заключается в том, что value функция аппроксимирует скользящее среднее вознаграждения, и таким образом корректирует обучение policy. Value функция обновляется согласно равенству Беллмана.

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.

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

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

Проблема

Случается так, что алгоритм должен уметь быстро, с очень небольшим количеством обучающих примеров, адаптироваться под решение новых задач. Например научить классифицировать изображения чайника, имея только пару его изображений, модель, которая уже умеет классифицировать множество объектов.

Идея

Можно представить задачу следующим образом: модель f работает с множеством задач T, имеющих некоторое распределение $p(T)$. Чтобы избежать переобучения и при этом решить новую задачу важно найти параметры модели, которые сильно влияют на функции потерь каждой задачи из $p(T)$. А так как не делается никаких предположений, кроме того, что модель параметризована, то 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

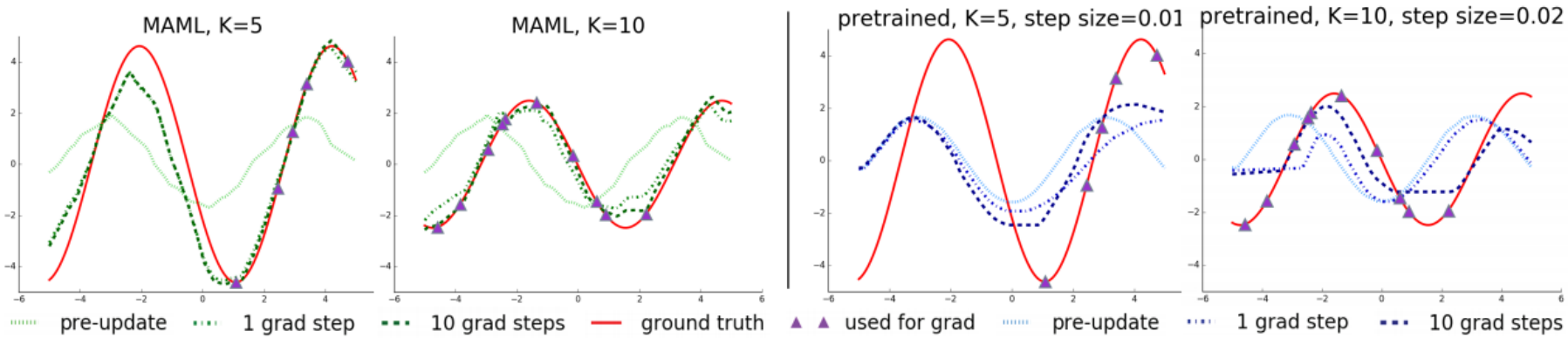


Рис.: Алгоритм MAML для регрессии

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

Создать модель, которая будет определять способ обновления параметров конечного алгоритма. Посмотрим на формулу градиентного спуска:

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

Ключевым является следующее наблюдение. Формула обновления памяти ячейки LSTM сети (cell state) очень похожа на обновление параметров с помощью градиентного спуска:

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

Где c_t будет играть роль параметров сети θ_t и $\tilde{c}_t = \nabla_{\theta_{t-1}} L_t$. В такой постановке i_t будет обучаемым параметром:

$$i_t = \sigma(W_I \cdot \left[\nabla_{\theta_{t-1}} L_t, L_t, \theta_{t-1}, i_{t-1}\right] + b_I)$$

Теперь learning rate является функцией от функции потерь, ее градиента, параметров сети и своего значения на предыдущем шаге. Константное занчение $f_t = 1$ кажется не самым оптимальным. Уменьшение параметров и забывание чати предыдущих их значений может оказаться полезным в точке неудачного локального оптимума. В ситуации когда значение градиента близко к нулю, а значение функции потерь довольно велико.

$$f_t = \sigma(W_F \cdot \left[\nabla_{\theta_{t-1}} L_t, L_t, \theta_{t-1}, f_{t-1}\right] + b_F)$$

Optimization as a model for few-shot learning

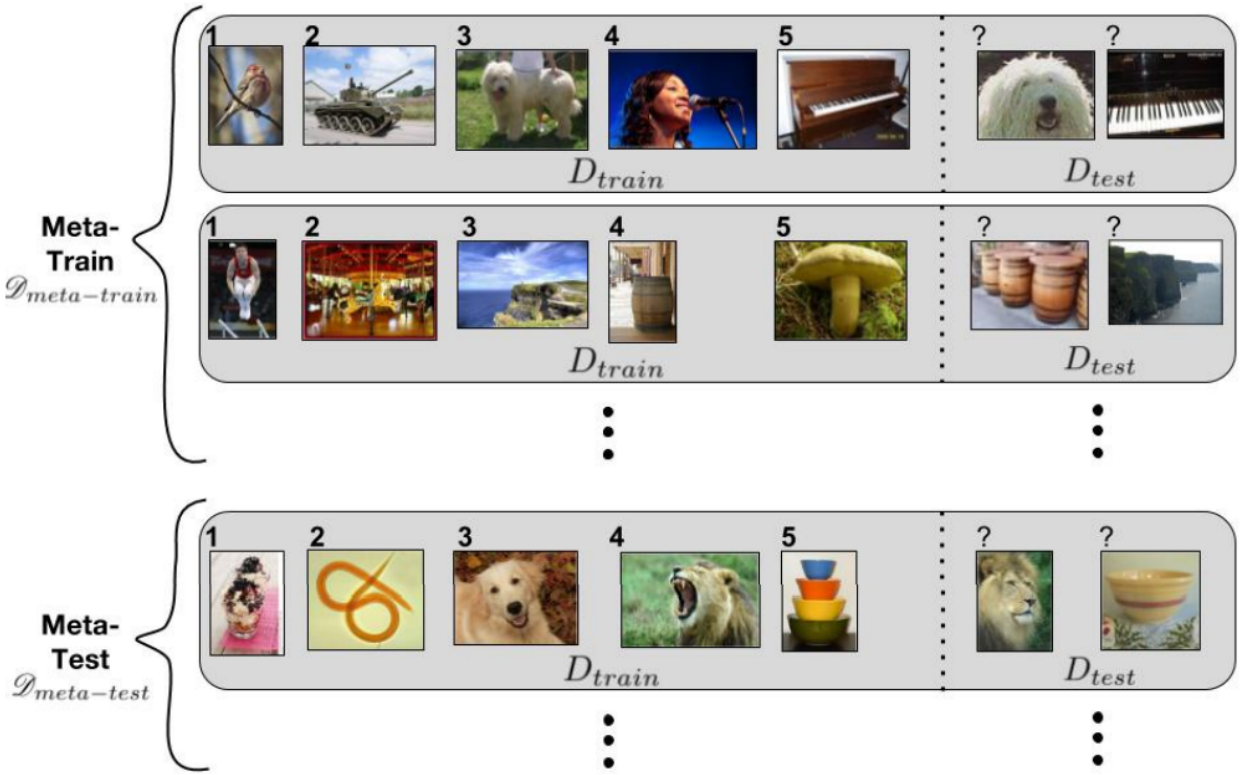


Рис.: Обучение Meta-Learner'а происходит на множестве пар датасетов, называемых эпизодами. Каждая пара состоит из тренировочной и тестовой выборки. В тренировочной выборке присутствуют k*N элементов, где N — количество классов, а k — ограничение на количество элементов из одного класса.

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

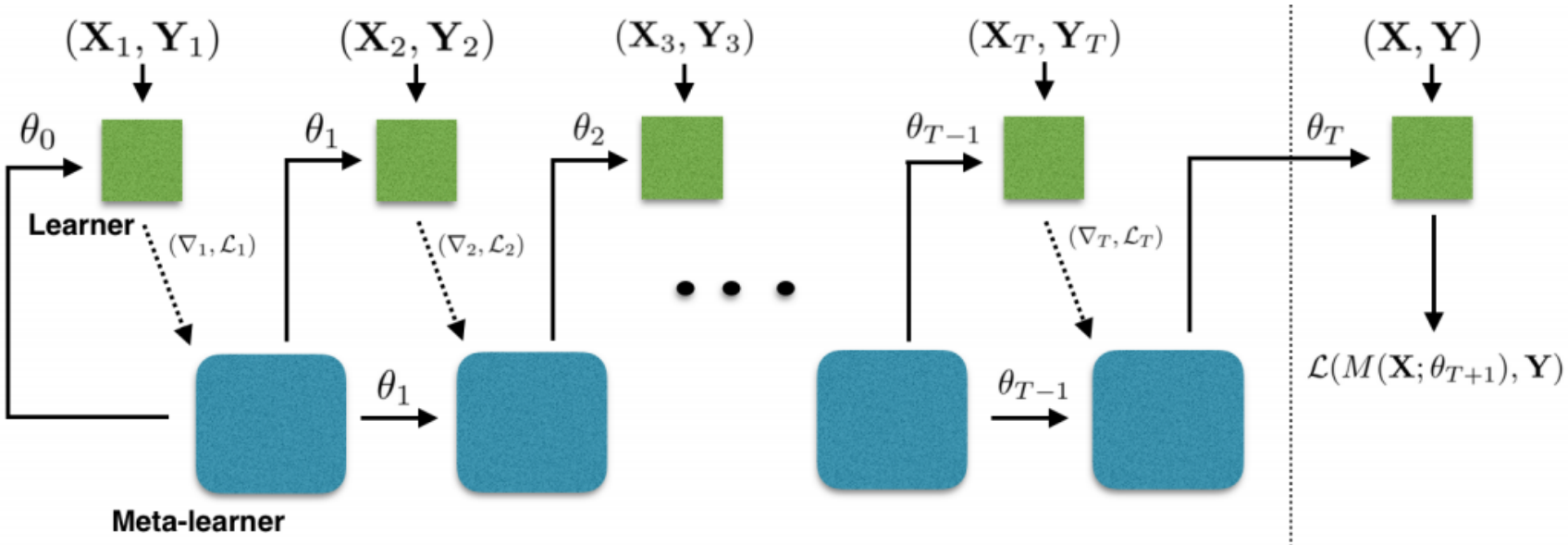


Рис.: Проход вперед(forward pass) meta learner'a. Как видно часть стрелок пунктирные, это означает, что во время обновления весов эти шаги не учитываются. Это позволяет избежать появления вторых производных и сильно упрощает вычисления. Пунктирной линией отделены шаги на тестовой и тренировочной выборках.

Limitations of meta learning

Ограничения

Наибольшим ограничением является то, что распределение задач на тренировочной выборке должно быть тем же что и на тестовой. Но на самом деле иногда новые задачи фундаментально отличаются от всех виденных ранее. Например если мы обучим модель математике, програмированию, чтению и т.д. сможет ли модель в результате выучить химию?