

3.36pt

Meta-learning


Медведев Алексей Владимирович

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

Meta-learning

- Построение системы для улучшения работы модели с конкретной задачей (подбор оптимальных гиперпараметров черного ящика).
- Адаптация алгоритма для решения новых задач.
- Аналогия — эволюционный процесс, создавший человеческий мозг.

Neural Architecture Search



`/home/alex/python/D'yakov/convnet.png`

Проблема

Нейронные сети хорошо показали себя в задачах обработки изображений, речи, понимания языка. 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

Neural Architecture Search

`/home/alex/python/D'yakov/nas_ar`

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



`/home/alex/python/D'yakov/nas_rr`

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

Reinforcement Learning

/home/alex/python/D'yakon

Введем некоторые обозначения:

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

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

Reinforcement Learning

Policy gradient

- Оптимизация π_θ напрямую
- Градиент функционала: $\nabla_\theta J(\theta) = \sum_\tau R(\tau) P(\tau, \theta) \nabla_\theta \log P(\tau, \theta)$

$$\nabla_\theta J(\theta) = \sum_\tau R(\tau) \nabla_\theta P(\tau, \theta) = \sum_\tau R(\tau) P(\tau, \theta) \frac{\nabla_\theta P(\tau, \theta)}{P(\tau, \theta)} = \sum_\tau R(\tau) P(\tau, \theta) \nabla_\theta \log P(\tau, \theta)$$

- Где: $P(\tau, \theta) = \prod_{t=0}^H P(s_{t+1} | a_t, s_t) \pi_\theta(a_t | s_t)$
- Несмещенная оценка: $\nabla_\theta J(\theta) \approx \frac{1}{m} \sum_{i=1}^m R(\tau^i) \nabla_\theta \log P(\tau^i, \theta)$
- Не зависит от динамики среды:

$$\nabla_\theta \log P(\tau, \theta) = \nabla_\theta \log \left(\prod_{t=0}^H P(s_{t+1} | a_t, s_t) \pi_\theta(a_t | s_t) \right) = \sum_{t=0}^H \nabla_\theta \log \pi_\theta(a_t | s_t)$$

REINFORCE algorithms [Williams, 1992]

- Оценка градиента хоть и несмещенная, но может иметь большой разброс.

- Чем больше $R(\tau)$ тем разброс больше:

Пусть $\log \pi_{\theta}(\tau) = [0.5, 0.2, 0.3]$, и $R(\tau) = [1000, 1001, 1002]$,

тогда $\text{Var}(0.5 \cdot 1000, 0.2 \cdot 1001, 0.3 \cdot 1002) = 23286.8$.

Если $R(\tau) = [1, 0, -1]$, то $\text{Var} = 0.1633$.

- Тогда оценка градиента принимает следующий вид:

$$\nabla_{\theta} J(\theta) \approx \frac{1}{m} \sum_{i=1}^m \nabla_{\theta} \log P(\tau^i, \theta) (R(\tau^i) - b)$$

- Доказательство несмещенности:

$$\begin{aligned} E_{\pi_{\theta}} [\nabla_{\theta} \log P(\tau, \theta) b] &= \sum_{\tau} P(\tau, \theta) \nabla_{\theta} \log P(\tau, \theta) b = \sum_{\tau} \nabla_{\theta} P(\tau, \theta) b = \\ &= b \nabla_{\theta} \sum_{\tau} P(\tau) = b \times 0 \end{aligned}$$


/home/alex/python/D'yakonov/nas_sc

- 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'a, 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



/home/alex/python/D'yakov/nas_ci

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

Neural Architecture Search



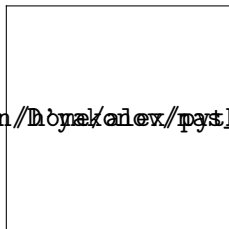
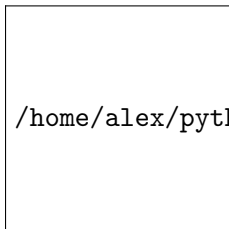
/home/alex/python/D'yakov/nas_pt

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.



`/home/alex/python/Downloads/python_lstm.py`

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

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

- 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 детерминированных policy функций есть специальный алгоритм.
- Две обучаемые функции: policy ($\pi(s_t, z_t)$), action-value или omniscient critic ($Q(s_t, a_t, y_t, \mu)$), где $y_t = y(h_t)$ — внутренняя память
- Action-value функция: $Q^\pi(s_t, a_t) = E \left[\sum_{i=t} \gamma^{i-t} r_i \mid s_t, a_t, \pi \right]$
- Action-value функция обновляется согласно равенству Беллмана:

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

Hindsight Experience Replay

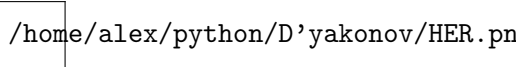
Проблема

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



The diagram shows a file path `/home/alex/python/D'yakov/HER.py`. A black rectangular box is drawn around the `home` directory name, indicating its significance in the context of the algorithm.

```
/home/alex/python/D'yakov/HER.py
```

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

Hindsight Experience Replay

Experience Replay.mp4

Sim2Real with meta learning

```
/home/alex/python/D'yakonov/sim2re
```

Рис.: Основной алгоритм обучения агента

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.

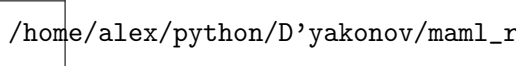
Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks



`/home/alex/python/D'yakonov/maml.p`

Рис.: Алгоритм MAML в общем виде

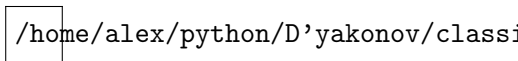
Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks



/home/alex/python/D'yakonov/maml_r

Рис.: Алгоритм MAML для регрессии

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



```
/home/alex/python/D'yakonov/classification
```

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
- $i_t = \sigma(W_I \cdot [\nabla_{\theta_{t-1}} L_t, L_t, \theta_{t-1}, i_{t-1}] + b_I)$
- Константа $f_t = 1$ не оптимальна, уменьшение параметров и забывание части предыдущих их значений может оказаться полезным в точке неудачного локального оптимума
- $f_t = \sigma(W_F \cdot [\nabla_{\theta_{t-1}} L_t, L_t, \theta_{t-1}, f_{t-1}] + b_F)$

Optimization as a model for few-shot learning



 /home/alex/python/D'yakonov/LSTM_

Рис.: Архитектура LSTM

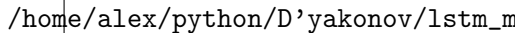
Optimization as a model for few-shot learning



```
/home/alex/python/D'yakov/lstm_n
```

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

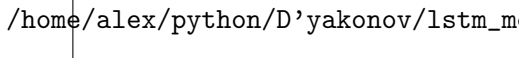
Optimization as a model for few-shot learning



`/home/alex/python/D'yakonov/lstm_m`

Рис.: Алгоритм обучения Meta-Learner

Optimization as a model for few-shot learning



```
/home/alex/python/D'yakonov/lstm_m
```

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 Abbeel, «Sim-to-Real Transfer of Robotic Control with Dynamics Randomization», <https://arxiv.org/pdf/1710.06537.pdf>