DATASET DISTILLATION

Клименко Злата

Dataset Distillation

Идея:

- хотим перегнать знания из большого обучающего набора данных в небольшой

(генерируем новые данные)

- обучение сети на новых данных дает хорошее качество
- фиксированная архитектура

Один шаг GD

Обычная задача:

$$\theta^* = \arg\min_\theta \frac{1}{N}\sum_{i=1}^{N} \ell(x_i, \theta) \triangleq \arg\min_\theta \ell(\mathbf{x}, \theta)$$ - хотим найти решение

$$\theta_{t+1} = \theta_t - \eta \nabla_{\theta_t} \ell(\mathbf{x}_t, \theta_t)$$ -оптимизация на каждом шаге SGD

Вместо этого хотим сделать один шаг на новом наборе:

$$\theta_1 = \theta_0 - \tilde{\eta} \nabla_{\theta_0} \ell(\tilde{\mathbf{x}}, \theta_0)$$

Один шаг GD

Фиксированные веса:

- находим lr и синтетический набор как

$$\tilde{\mathbf{x}}^*, \tilde{\eta}^* = \arg\min_{\tilde{\mathbf{x}}, \tilde{\eta}} \mathcal{L}(\tilde{\mathbf{x}}, \tilde{\eta}; \theta_0) = \arg\min_{\tilde{\mathbf{x}}, \tilde{\eta}} \ell(\mathbf{x}, \theta_1) =$$

$$= \arg\min_{\tilde{\mathbf{x}}, \tilde{\eta}} \ell(\mathbf{x}, \theta_0 - \tilde{\eta} \nabla_{\theta_0} \ell(\tilde{\mathbf{x}}, \theta_0))$$

Случайные веса:

- предполагаем, что начальные веса берутся из распределения

$$\tilde{\mathbf{x}}^*, \tilde{\eta}^* = \arg\min_{\tilde{\mathbf{x}}, \tilde{\eta}} \mathbb{E}_{\theta_0 \sim p(\theta_0)} \mathcal{L}(\tilde{\mathbf{x}}, \tilde{\eta}; \theta_0)$$

Пример

модель - линейная регрессия с квадратичной функцией потерь

Хотим сжать данные, не потеряв при этом качества. Функция потерь выглядит так:

$$\ell(\mathbf{x}, \theta) = \ell((\mathbf{d}, \mathbf{t}), \theta) = \frac{1}{2N}\|\mathbf{d}\theta - \mathbf{t}\|^2$$

, где d - матрица признаков, t - таргет, N - размер датасета. Хотим N -> M и минимизировать

$$\ell(\mathbf{x}, \theta_0 - \tilde{\eta}\nabla_{\theta_0}\ell(\tilde{\mathbf{x}}, \theta_0))$$

Новые веса после обновления:

$$\theta_1 = \theta_0 - \tilde{\eta}\nabla_{\theta_0}\ell(\tilde{\mathbf{x}}, \theta_0) = \theta_0 - \frac{\tilde{\eta}}{M}\tilde{\mathbf{d}}^T(\tilde{\mathbf{d}}\theta_0 - \tilde{\mathbf{t}}) = (\mathbf{I} - \frac{\tilde{\eta}}{M}\tilde{\mathbf{d}}^T\tilde{\mathbf{d}})\theta_0 + \frac{\tilde{\eta}}{M}\tilde{\mathbf{d}}^T\tilde{\mathbf{t}}.$$

Пример

модель - линейная регрессия с квадратичной функцией потерь

Минимум $\ell(\mathbf{x}, \theta)$ достигается при $\mathbf{d}^T\mathbf{d}\theta^* = \mathbf{d}^T\mathbf{t}$

Подставим в это равенство новые веса:

$$\mathbf{d}^T\mathbf{d}(\mathbf{I} - \frac{\tilde{\eta}}{M}\tilde{\mathbf{d}}^T\tilde{\mathbf{d}})\theta_0 + \frac{\tilde{\eta}}{M}\mathbf{d}^T\mathbf{d}\tilde{\mathbf{d}}^T\tilde{\mathbf{t}} = \mathbf{d}^T\mathbf{t}$$

При независимых признаках получаем $\mathbf{I} - \frac{\tilde{\eta}}{M}\tilde{\mathbf{d}}^T\tilde{\mathbf{d}} = \mathbf{0}$

Значит, M >= количества признаков в данных.

Алгоритм

Algorithm 1 Dataset Distillation

Input: $p(\theta_0)$: distribution of initial weights; M: the number of distilled data
Input: α: step size; n: batch size; T: the number of optimization iterations; $\tilde{\eta}_0$: initial value for $\tilde{\eta}$

Initialize $\tilde{\mathbf{x}} = \{\tilde{x}_i\}_{i=1}^{M}$ randomly, $\tilde{\eta} \leftarrow \tilde{\eta}_0$
for each training step $t = 1$ to T **do**
    Get a minibatch of real training data $\mathbf{x}_t = \{x_{t,j}\}_{j=1}^{n}$
    Sample a batch of initial weights $\theta_0^{(j)} \sim p(\theta_0)$
    for each sampled $\theta_0^{(j)}$ **do**
        Compute updated parameter with GD: $\theta_1^{(j)} = \theta_0^{(j)} - \tilde{\eta}\nabla_{\theta_0^{(j)}}\ell(\tilde{\mathbf{x}}, \theta_0^{(j)})$
        Evaluate the objective function on real training data: $\mathcal{L}^{(j)} = \ell(\mathbf{x}_t, \theta_1^{(j)})$
    end for
    Update $\tilde{\mathbf{x}} \leftarrow \tilde{\mathbf{x}} - \alpha\nabla_{\tilde{\mathbf{x}}}\sum_j \mathcal{L}^{(j)}$, and $\tilde{\eta} \leftarrow \tilde{\eta} - \alpha\nabla_{\tilde{\eta}}\sum_j \mathcal{L}^{(j)}$
end for

Output: distilled data $\tilde{\mathbf{x}}$ and optimized learning rate $\tilde{\eta}$

Расширение алгоритма

- Хотим несколько шагов GD - каждый шаг на разном батче новых данных и с разным lr,

строка 6 меняется на цикл по i : $\theta_{i+1} = \theta_i - \tilde{\eta}_i \nabla_{\theta_i} \ell(\tilde{\mathbf{x}}_i, \theta_i)$

в строке 9 делаем backprop через все шаги.

- Хотим несколько эпох - каждую эпоху повторяем шаги градиентного спуска для одной и той же последовательности сжатых данных

Как считается градиент по гиперпараметрам?

Одна итерация - запускается весь внутренний цикл для оптимизации обычных параметров, оценивается лосс по данным, градиент вычисляется путем обратного распространения ошибки, обновляются гиперпараметры.

Наивное вычисление градиентов требует больших вычислительных затрат и памяти.

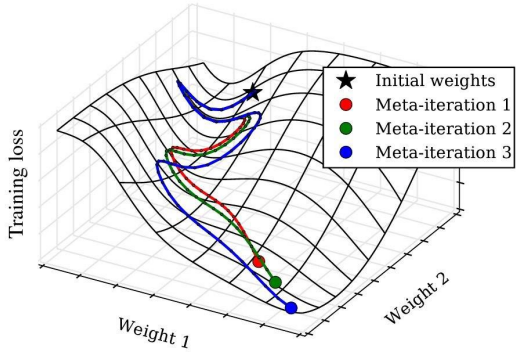


Figure 1. Hyperparameter optimization by gradient descent. Each meta-iteration runs an entire training run of stochastic gradient descent to optimize elementary parameters (weights 1 and 2). Gradients of the validation loss with respect to hyperparameters are then computed by propagating gradients back through the elementary training iterations. Hyperparameters (in this case, learning rate and momentum schedules) are then updated in the direction of this hypergradient.

Как считается градиент по гиперпараметрам?

Вместо хранения траектории обучения для распространения ошибки будем вычислять ее повторно в обратном направлении

Заменим умножение гессиана на вектор на градиент на вектор с применением шага назад + трюки для избежания потери точности

Algorithm 1 Stochastic gradient descent with momentum

input: initial $\mathbf{w}_1$, decays $\boldsymbol{\gamma}$, learning rates $\boldsymbol{\alpha}$, loss function $L(\mathbf{w}, \boldsymbol{\theta}, t)$
initialize $\mathbf{v}_1 = \mathbf{0}$
for $t = 1$ **to** T **do**
  $\mathbf{g}_t = \nabla_{\mathbf{w}} L(\mathbf{w}_t, \boldsymbol{\theta}, t)$ ▷ evaluate gradient
  $\mathbf{v}_{t+1} = \gamma_t \mathbf{v}_t - (1 - \gamma_t)\mathbf{g}_t$ ▷ update velocity
  $\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha_t \mathbf{v}_t$ ▷ update position
end for
output trained parameters $\mathbf{w}_T$

Algorithm 2 Reverse-mode differentiation of SGD

input: $\mathbf{w}_T$, $\mathbf{v}_T$, $\boldsymbol{\gamma}$, $\boldsymbol{\alpha}$, train loss $L(\mathbf{w}, \boldsymbol{\theta}, t)$, loss $f(\mathbf{w})$
initialize $d\mathbf{v} = \mathbf{0}$, $d\boldsymbol{\theta} = \mathbf{0}$, $d\alpha_t = \mathbf{0}$, $d\gamma = \mathbf{0}$
initialize $d\mathbf{w} = \nabla_{\mathbf{w}} f(\mathbf{w}_T)$
for $t = T$ **counting down to** 1 **do**
  $d\alpha_t = d\mathbf{w}^\mathsf{T} \mathbf{v}_t$
  $\mathbf{w}_{t-1} = \mathbf{w}_t - \alpha_t \mathbf{v}_t$ } exactly reverse
  $\mathbf{g}_t = \nabla_{\mathbf{w}} L(\mathbf{w}_t, \boldsymbol{\theta}, t)$ } gradient descent
  $\mathbf{v}_{t-1} = [\mathbf{v}_t + (1 - \gamma_t)\mathbf{g}_t]/\gamma_t$ } operations
  $d\mathbf{v} = d\mathbf{v} + \alpha_t d\mathbf{w}$
  $d\gamma_t = d\mathbf{v}^\mathsf{T}(\mathbf{v}_t + \mathbf{g}_t)$
  $d\mathbf{w} = d\mathbf{w} - (1 - \gamma_t) d\mathbf{v} \nabla_{\mathbf{w}} \nabla_{\mathbf{w}} L(\mathbf{w}_t, \boldsymbol{\theta}, t)$
  $d\boldsymbol{\theta} = d\boldsymbol{\theta} - (1 - \gamma_t) d\mathbf{v} \nabla_{\boldsymbol{\theta}} \nabla_{\mathbf{w}} L(\mathbf{w}_t, \boldsymbol{\theta}, t)$
  $d\mathbf{v} = \gamma_t d\mathbf{v}$
end for
output gradient of $f(\mathbf{w}_T)$ w.r.t $\mathbf{w}_1$, $\mathbf{v}_1$, $\boldsymbol{\gamma}$, $\boldsymbol{\alpha}$ and $\boldsymbol{\theta}$

Пример - отравление данных

- Можем применить один шаг GD на синтетических данных
- Хотим, чтобы классификатор предсказывал вместо класса K класс T

Как достичь?

- Берем новую $\ell_{K \to T}(\mathbf{x}, \theta_1)$, которая помогает предсказывать K вместо T, не трогая остальные классы (меняем метки в кросс-энтропии)
- Получаем данные путем минимизации

  $$\tilde{\mathbf{x}}^*, \tilde{\eta}^* = \arg\min_{\tilde{\mathbf{x}}, \tilde{\eta}} \mathbb{E}_{\theta_0 \sim p(\theta_0)} \mathcal{L}_{K \to T}(\tilde{\mathbf{x}}, \tilde{\eta}; \theta_0)$$

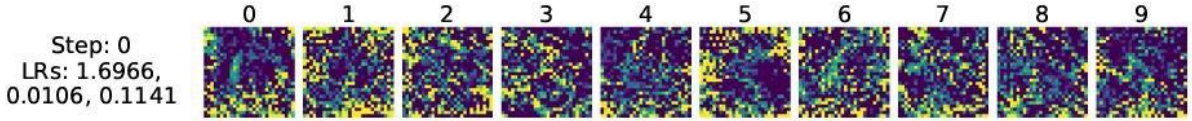
  , где p - распределение на случайных весах хорошо обученных классификаторов.

- Не требуется знать веса

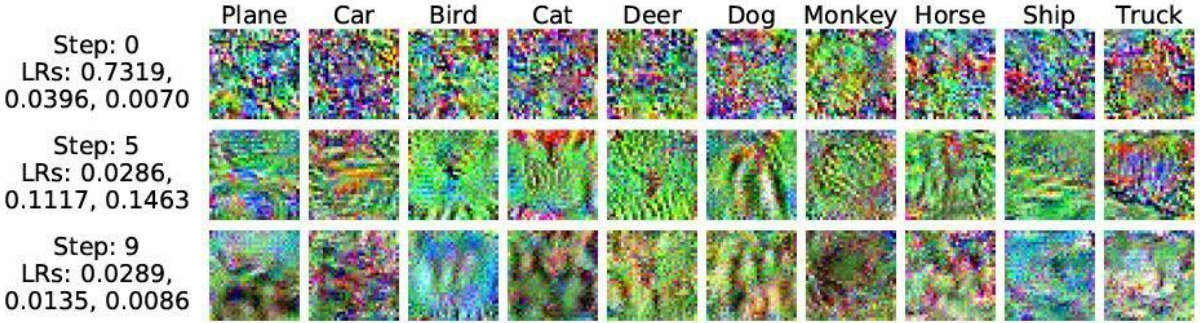
Возможная инициализация сети

- Случайная инициализация (метод Хе или метод Завьера)
- Фиксированная инициализация
- Предобученная модель со случайными весами
- Предобученная модель с фиксированными весами

Эксперименты
фиксированная инициализация

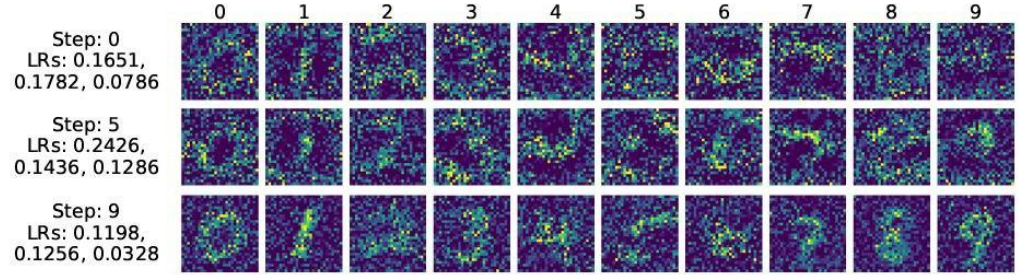


(a) MNIST. Theses distilled images train a fixed initializations from 12.90% test accuracy to 93.76%.

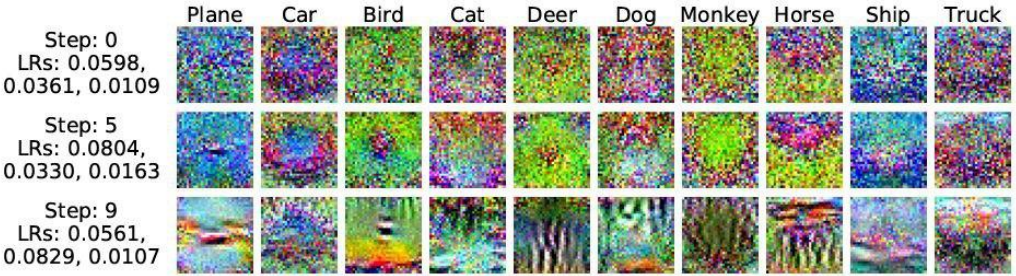


(b) CIFAR10. These distilled images train a fixed initialization from 8.82% test accuracy to 54.03%.

Эксперименты
случайная инициализация

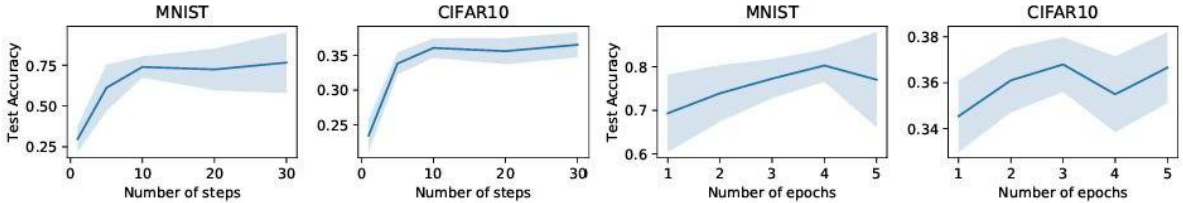


(a) MNIST. These distilled images unknown random initializations to $79.50\% \pm 8.08\%$ test accuracy.

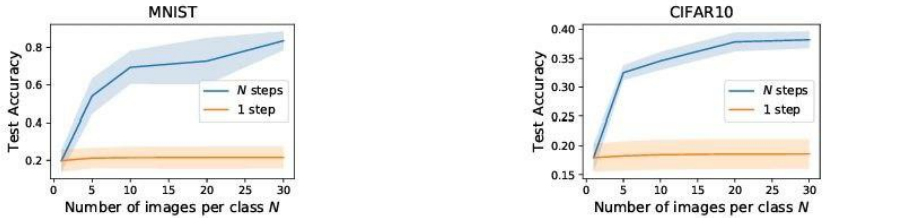


(b) CIFAR10. These distilled images unknown random initializations to $36.79\% \pm 1.18\%$ test accuracy.

Эксперименты
зависимость от гиперпараметров



(a) Test accuracy w.r.t. the number of steps (b) Test accuracy w.r.t. the number of epochs



(a) Test accuracy w.r.t. the number of steps (b) Test accuracy w.r.t. the number of epochs

Эксперименты
предобученные сети

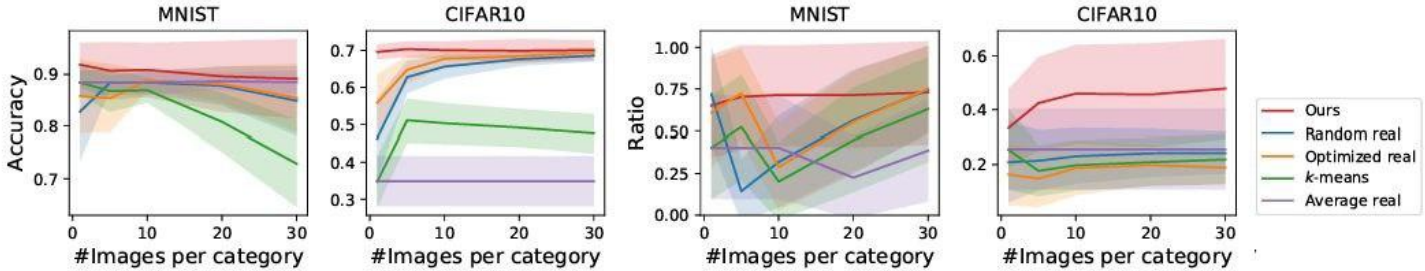
	Ours w/ fixed pre-trained	Ours w/ random pre-trained	Random real	Optimized real	k-means	Average real	Adaptation Motiian et al. (2017)	No adaptation	Train on **full** target dataset
$\mathcal{M} \rightarrow \mathcal{U}$	**97.9**	$\mathbf{95.4 \pm 1.8}$	94.9 ± 0.8	95.2 ± 0.7	92.2 ± 1.6	93.9 ± 0.8	$\mathbf{96.7 \pm 0.5}$	90.4 ± 3.0	97.3 ± 0.3
$\mathcal{U} \rightarrow \mathcal{M}$	**93.2**	$\mathbf{92.7 \pm 1.4}$	87.1 ± 2.9	87.6 ± 2.1	85.6 ± 3.1	78.4 ± 5.0	89.2 ± 2.4	67.5 ± 3.9	98.6 ± 0.5
$\mathcal{S} \rightarrow \mathcal{M}$	**96.2**	85.2 ± 4.7	84.6 ± 2.1	85.2 ± 1.2	$\mathbf{85.8 \pm 1.2}$	74.9 ± 2.6	74.0 ± 1.5	51.6 ± 2.8	98.6 ± 0.5

Table 2: Performance of our method and baselines in adapting models among MNIST ($\mathcal{M}$), USPS ($\mathcal{U}$), and SVHN ($\mathcal{S}$). 100 distilled images are trained for ten GD steps and three epochs. Our method outperforms few-shot domain adaptation (Motiian et al., 2017) and other baselines in most settings.

Target dataset	Ours	Random real	Optimized real	Average real	Fine-tune on **full** target dataset
PASCAL-VOC	**70.75**	19.41 ± 3.73	23.82 ± 3.66	9.94	75.57 ± 0.18
CUB-200	**38.76**	7.11 ± 0.66	7.23 ± 0.78	2.88	41.21 ± 0.51

Table 3: Performance of our method and baselines in adapting an AlexNet pre-trained on ImageNet to PASCAL-VOC and CUB-200. We use one distilled image per category, with one GD step repeated for three epochs. Our method significantly outperforms the baselines. We report results over 10 runs.

Эксперименты
вредоносная атака



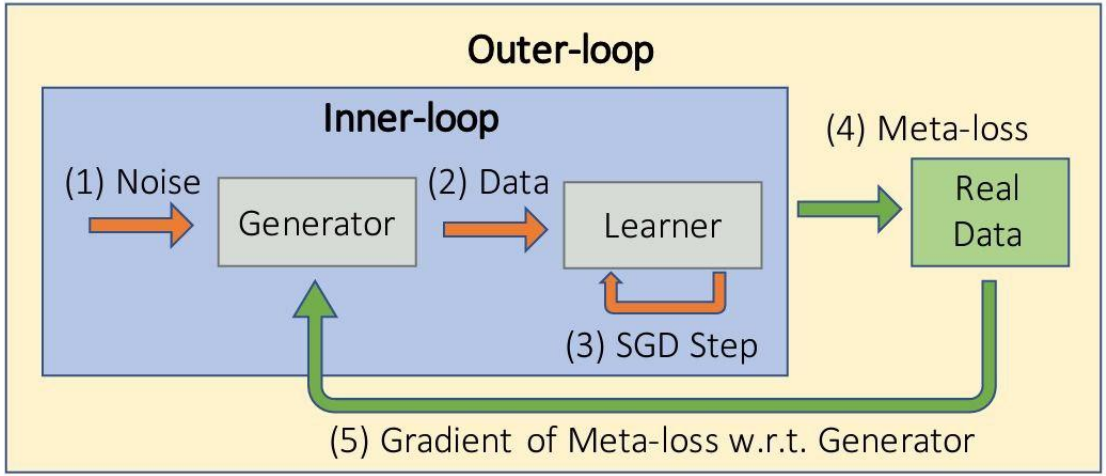
(a) Overall accuracy w.r.t. modified labels (b) Ratio of attacked category misclassified as target

Generative Teaching Networks

Идея:

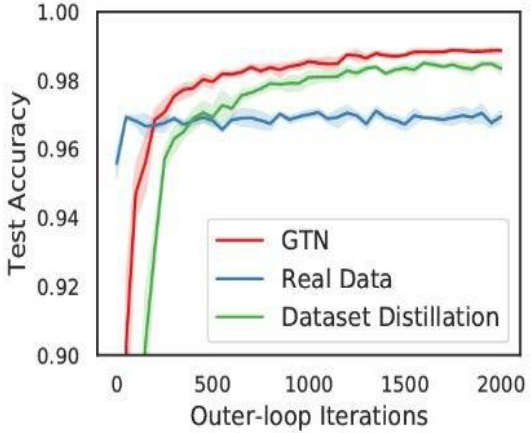
- хотим обучать сеть, генерирующую набор данных
- эти данные должны показывать хорошее качество при обучении на них основной обучаемой сети
- хотим, чтобы данные не зависели от нейронной сети
- архитектура обучаемой сети меняется, веса инициализируются случайно

Общая схема

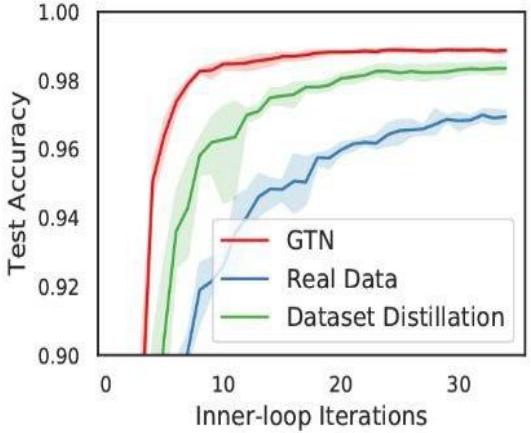


(a) Overview of Generative Teaching Networks

Сравнение методов на MNIST



(a) Meta-training curves



(b) Training curves

(c) GTN-generated samples

Список литературы

- Wang, Tongzhou et al. "Dataset Distillation". *arXiv preprint arXiv:1811.10959*. (2018).
- Felipe Petroski Such, et al. "Generative Teaching Networks: Accelerating Neural Architecture Search by Learning to Generate Synthetic Training Data". *CoRR* abs/1912.07768. (2019).
- Maclaurin, Dougal et al. "Gradient-based Hyperparameter Optimization through Reversible Learning". *arXiv e-prints* arXiv:1502.03492. (2015):