AutoLR Layer-wise Pruning and Auto-tuning of Learning Rates in Fine-tuning of Deep Networks

Summary

AutoLR: Layer-wise Pruning and Auto-tuning of Learning Rates in Fine-tuning of Deep Networks 논문리뷰. (ZoteroReview)

Review

Abstract

> Existing fine-tuning methods use a single learning rate over all layers. In this paper, first, we discuss that trends of layer-wise weight variations by fine-tuning using a single learning rate do not match the well-known notion that lower-level layers extract general features and higher-level layers extract specific features. Based on our discussion, we propose an algorithm that improves fine-tuning performance and reduces network complexity through layer-wise pruning and auto-tuning of layer-wise learning rates. The proposed algorithm has verified the effectiveness by achieving state-of-the-art performance on the image retrieval benchmark datasets (CUB-200, Cars-196, Stanford online product, and Inshop). Code is available at https://github.com/youngminPIL/AutoLR.
>
> - 동일한 learning rate를 사용하는 fine-tuning을 사용하면, 일반적으로 CNN에서 잘 알려져 있는 낮은 layer에서 general feature를 추출하고 깊은 layer에서는 task-specific한 feature를 추출한다는 개념과 일치하지 않는다.
> - 본 논문에서는 계층별 pruning / 계층별 learning rate 자동 조정을 사용하여 fune-tuning의 성능을 향상시키고 복잡성을 줄이는 알고리즘을 제안한다.

Introduction

실제로는 ImageNet처럼 대량의 데이터를 수집하는 것이 쉽지 않기 때문에 pre-train된 모델을 fine-tuning하는 방식이 많이 사용되고 있다. (Fu, Zheng, and Mei 2017; Zheng et al. 2017; Suh et al. 2019; Guo et al. 2019; Wang et al. 2019)

Fine-tuning의 성능은 일반적으로 다음의 요소에 기반한다.

- similarity between the source and target tasks (Azizpour et al. 2015; Cui et al. 2018),
- choice of deep network model (Kornblith, Shlens, and Le 2018)
- tuning strategy (Tajbakhsh et al. 2016; Guo et al. 2019; Ro et al. 2019)

→ 본 연구는 더 효과적인 tuning strategy를 제안하는 것을 목적으로 한다.

Tuning Strategy는 다음과 같은 종류가 있다.

- Partial tuning (Tajbakhsh et al. 2016): 오직 higher-level layer만 학습시킨다.
- Weight-reverting (Guo et al. 2019; Ro et al. 2019): Fine-tuning 학습 도중에 원래 모델(pre-trained model)의 가중치로 되돌아 간다.
- Learning rate를 점점 줄여나가는 방식 (Smith 2017), (Loshchilov and Hutter 2016)

→ 그러나 대부분의 LR 조절 방식은 모두 레이어에 관계없이 동일한 단일 learning rate를 사용한다.

본 논문에서는, **각 레이어의 LR이 레이어의 역할에 따라 자동으로 조정되는 레이어별 LR 자동 조정 알고리즘**을 제안한다. 또한, 새로운 작업에 대해 사전 학습된 각 레이어의 유용성에 따라 레이어를 제거하는 레이어별 pruning 알고리즘을 제안한다.

단일 LR을 사용하는 기존 방식을 사용하여 Fine-tuning을 하는 경우, low-level layer가 일반적인 특징을 추출하고 high-level layer가 specific한 특징을 추출한다고 주장하는 CNN에 대한 이전 연구(Yosinski et al. 2014; Zeiler and Fergus 2014)와 모순된다.

⇒ 이 점에서 착안하여, LR을 각 level layer가 하는 기능에 맞도록 조절할 수 있는 알고리즘을 제안한다.

Related Works

Fine-tuning

- Azizpour et al. (2015): pre-trained 모델 학습에 사용한 task와 Fine-tuning에 사용한 task가 유사할 수록 Fine-tuning이 잘 동작한다는 연구.
- And Cui et al. (2018): 여러개의 유사한 task를 사용하여 knowledge transfer를 사는 방식을 제안한 연구.

DNN/CNN explain

- Kornblith et al. (2018): Image Net이 매우 많은 이미지 데이터를 가지고 있기 때문에 대부분의 task(=few-shot task)에 대해서도 상관관계를 가지고 있다는 연구.
- Yosinski et al. (2014): DNN 각 계층의 일반화/특이성 정도를 정량화하기 위한 실증적 연구
- Zeiler et al. (2014): hidden layer의 각 feature를 시각하하여 분석한 연구.

위 선행연구들에 따르면, DNN에서 하위 수준 레이어가 일반적인 특징을 추출하고 상위 수준 레이어가 specific feature를 추출한다고 주장한다.

(Layerwise) Fine-tuning

- Tajbakhsh et al. 2016: 모든 레이어를 튜닝하는 것보다 몇 개의 상위 레벨 레이어만 튜닝하는 것이 더 효과적임을 보인 연구
- Guo et al. (2019): pre-trained weight를 사용할 지, 아님 fine-tuning weight를 사용할 것인지를 결정하는 policy network를 제안하는 연구
- Ro et al. (2019): fine tune model의 일부 weight를 pre-trained state로 되돌려서 성능을 향상시키는 방식을 제안하는 연구
- Li et al. 2017; Ren et al. 2018: network weight의 최적 업데이트를 찾는 것을 목표로 진행한 연구
- Smith (2017): 학습률을 선형적으로 감소시켰다가 다시 증가시키는 삼각형 형태로 학습률을 조정하는 방법을 제안
- Loshchilov et al. (2016): 일정 기간 동안 학습률이 기하급수적으로 감소하고 다시 증가하는 방식을 제안

Idea

> **Important**
>
> lower-level layers extract general features and higher-level layers extract specific features.

Hypothesis

> Hypothesis 1: The pre-trained high-level layers may not be helpful to a new target task because they are specific to the source task.

일반적인 CNN laye별 분석에 따르면, 하위 레이어가 일반적인 feature를 추출하고 상위 레이어에서 specific feature를 추룰하는데 specific feature는 주로 특정한 이미지 집합의 세부적인 특징을 의미한다. 즉, Task가 바뀌는 fine-tuning에서는 high-level feature가 많이 변화될 것이기 때문에 high-level layer의 weight는 의미없을 수 있다.

> Hypothesis 2: The weight variations of pre-trained lowlevel layers would be small because they are generally valid for most tasks, whereas those in high-level layers would be large because they should adopt themselves to a new task specifically.

즉, 제대로 된 Fine-tuning이라면, **low-lever weight의 변화는 크지 않을 수 있지만 high-level에서는 새로운 task가 가지는 특징에 맞게 새로운 feature를 뽑아내야하기 때문에 weight의 변화가 커야할 것**이다.

$$v_t^1 \leq v_t^2 \leq \cdots \leq v_t^K \tag{1}$$

Disadvantages of traditional fine tuning

Experiment 1의 실험 결과에 따르면, 일반적인 fine-tuning을 사용했을 때 오히려 high-level에서 weight variation이 low-level의 weight variation 보다 작은 것을 알 수 있다. 즉, 기존의 fine-tuning 방식이 Hypothesis 2를 위배하기 때문에 학습에 도움을 주지 못하고 있다.

또한 Experiment 2의 실험 결과에 따르면, high-level layer를 pruning 하면 더 좋은 성능을 보이고, low-level layer의 LR은 작게 high-level layer의 LR은 크도록 조절하면 더 높은 성능을 가진다.

⇒ 따라서 Hypothesis 1, 2를 기반으로 layer-wise pruning & auto tuning of LR을 수행하는 알고리즘을 제안한다.

Idea of Auto LR

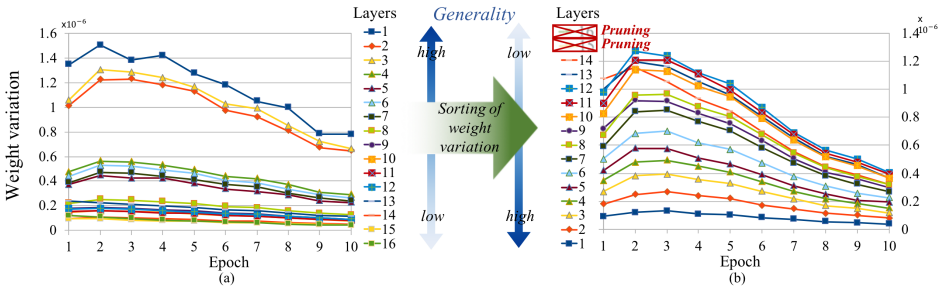
1. layer별 weight variation이 **Hypothesis를 만족하는 배열로 정렬되도록**, 각 epoch마다 layer별 이번 업데이트에 사용하고자하는 learning rate로 조절한다.
2. **weight variation과 LR의 관계를 SGD 업데이트 식에서부터 유도**한다.

Experiment

weight variation for each layer

(a) lr을 고정하는 기본적인 fine-tuning method를 사용했을 때 layer별 weight 변화를 나타낸다. layer가 낮을 수록 generality에 미치는 영향이 크지만 lr을 고정하면 low-level layer가 오히려 high-level layer보다 더 많은 변화가 발생하는 것을 알 수 있다.

(b) 가장 high-level layer 2개를 pruning하고 나머지에 Auto lr을 활용하여 훈련한 것인데 high-level일수록 low-layer보다 weight variation이 더 큰 것을 확인할 수 있다. (=> 즉, 성능개선이 이루어졌다)



pruning

Hypothesis 1에 따라 high-level layer를 pruning 하는 것이 더 좋은 성능을 보이는 걸 알 수 있다. 실험에서 layer 14번까지 잘라버리면 성능이 떨어지기 때문에 1~14번째 layer들이 성능에 영향을 미친다는 것을 알 수 있다. 또한 low-level lr을 줄이거나 high-level lr을 증가시키면 성능이 좋아지는걸 볼 수 있다.

	R@1	R@2	R@4	R@8
Original	63.49	75.03	84.00	90.58
Pruning 16	66.95	77.63	86.39	92.00
Pruning 15, 16	67.00	78.49	86.85	92.07
Pruning 14, 15, 16	51.00	64.16	75.44	85.94
Pruning 15, $16^{\dagger}$	67.15	78.61	86.83	92.08
Pruning 15, $16^{\ddagger}$	67.29	79.15	87.36	92.66
Pruning 15, $16^{\star}$	**68.33**	**79.88**	**87.69**	**92.71**

AutoLR Algorithm

Hypothesis 2를 만족하는 "target weight variation"을 설정하는 방법은 다음과 같다.
먼저 현재 weight variation이 얼마나 Hypothesis (2)를 만족하는지를 평가하기 위한 지표로서 sorting quality를 정의한다.

Definition 1. Sorting Quality

- σ: weight variation을 오름차순으로 mapping하여 그 index를 반환하는 함수이다.
- K: sorting quality가 0, 1사이 값에 있게 만드는 nomalize factor이다.

k가 실제 layer의 index이고 $\sigma(v_t^k)$는 layer 를 오름차순으로 정렬했을 때 나오는 index이다. 즉, $k=1$이면 weigh variation v의 index가 작아야 한다. layer index와 sorting idex가 비슷한 곳에 있을 수록 1에서 빼지는 값이 줄어들기에 Sorting quality가 높을 수록 현재 weight variation이 Hypothesis 2를 만족시키고 있다고 판단할 수 있다.

$$= 1 - \frac{2}{K^2}\sum_{k=1}^{K}|k - \sigma(v_t^k)|$$

Definition 2. Initial target weight variation

매번 epoch의 첫번째 단계에서는 이전 iteration $l-1$이 존재하지 않기 때문에 target weight를 다음과 같이 미리 정해두고 계산한다. 이때, α와 β는 target weight variation의 range를 정해주기 위한 hyperparameter이다.

$$d_t = \frac{1}{K-1}\left(\beta \max_{1\le k\le K} v_t^{(k)} - \alpha \min_{1\le k\le K} v_t^{(k)}\right)$$

$$\bar{v}_t^{(k)} \leftarrow \min_{1\le i\le K} v_t^{(k)} + (k-1)d_t,\ k = 1, \cdots, K$$

Algorith: setting target weight variation

sorting quality가 사전에 정의해둔 threshold τ_s보다 작다면, 현재 weight variation이 Hypothesis를 만족하는 정도가 높지 않기 때문에 weight variation을 수정해줘야한다.

이때, 한쪽 방향으로만 변화하는걸 막기 위하여 layer의 center ($\hat{k}$) 를 시작으로, low-layer방향과 high-layer 방향을 번갈아가면서 수행한다.

- For $k = \hat{k}$

$$\bar{v}_t^{(k)} \leftarrow v_t^{(k)} \tag{12}$$

- For $k = \hat{k}+1, \hat{k}+2, \cdots, K$

$$\bar{v}_t^{(k)} \leftarrow \begin{cases} v_t^{(k)} & \bar{v}_t^{(k-1)} \leq v_t^{(k)} \\ \bar{v}_t^{(k-1)} + d_t & \text{otherwise,} \end{cases} \tag{13}$$

- For $k = \hat{k}-1, \hat{k}-2, \cdots, 1$

$$\bar{v}_t^{(k)} \leftarrow \begin{cases} v_t^{(k)} & \bar{v}_t^{(k+1)} \geq v_t^{(k)} \\ \bar{v}_t^{(k-1)} + d_t & \text{otherwise,} \end{cases} \tag{14}$$

위 방식을 사용하여 target weight variation을 결정하면, weight variation과 LR의 관계식에서부터 target LR을 유도해낼 수 있다.

target weight variation은 LR과 weight variation의 relationship에 의해 아래와 같은 관계를 가지며, 이로부터 우리는 target weight variation이 되도록 SGD update를 수행하는 "target LR"을 계산할 수 있다.

$$\hat{v}_t^k = \frac{\hat{\eta}^k}{n_k} \|\nabla \mathcal{L}_{acc}(w_t^k)\|$$

Finally we can get the renewed LR as

$$\hat{\eta}_t^k \approx \frac{\eta_t^k \hat{\eta}_t^k}{v_t^k}$$

전반적인 AutoLR 알고리즘의 수도코드는 다음과 같다.

Algorithm 2 AutoLR: Auto-tuning of learning rates

Notation:
η^k : learning rate of k-th block
$\bar{\eta}^k$: target learning rate of k-th block
v_t^k : weight variation of k-th block in t-th epoch
$\bar{v}_t^k$: target weight variation of k-th block in t-th epoch
network : network with tuned weights
trial-network : trial network for tuning of η^k

Auto-tuning:
Initialize $\{\eta_k\}$ with $\{\eta_k^0\}$
Initialize *network* with *pre-trained network*
Set *sorting quality* $= 0$.
for epoch $\leftarrow 1$ to T **do**
  trial-network $\leftarrow$ *network*
  while *sorting quality* $\leq \tau_s$ **do**
    Fine-tune *trial-network* for target dataset
    Calculate *weight variation* in (1)
    Calculate *sorting quality* in (9)
    if *sorting quality* $> \tau_s$ **then**
      network $\leftarrow$ *trial-network*
    else
      if epoch $== 1$ **then**
        Renew $\bar{v}_t^k$ by (11)
      else
        Renew $\bar{v}_t^k$ by (12) and (13)
      Renew $\bar{\eta}^k$ by (15)
      $\eta^k \leftarrow \bar{\eta}^k$
  epoch++

Formulation + Proof

Notation

- amount of weight changes in the k-th layer during t-th epoch:

$$\Delta w_t^k = w_t^k - w_{t-1}^k$$

- number of weights in k-th layer: n_k
- weight variation of k-th layer during t-th epoch:

$$\begin{aligned} v_t^k &= \frac{1}{n_k} \|\Delta w_t^k\| \\ &= \frac{1}{n_k} \|w_t^k - w_{t-1}^k\| \end{aligned} \tag{2}$$

- target weight variation of k-th layer during t-th epoch: $\hat{v}_t^k$
  → 이때 target이란, Hypothesis 2를 만족하는 weight variation ordering을 지킬 때 값을 의미한다. (e,g, target weight variation, target

learning rate...)

- loss for $w^k_{t,l-1}$: $\mathcal{L}(w^k_{t,l-1})$
  - iteration index: l
  - epoch index: t
- momentum coefficient : ρ
- LR of k-th layer: η^k
- **target** LR of k-th layer: $\hat{\eta}^k$

Relationship LR between weight variation

SGD with the momentum of k-th layer as t-th epoch during l-th iteration for a randomly chosen mini-batch:

$$\Delta w^k_{t,l} \leftarrow \rho \Delta w^k_{t,l-1} - \eta^k \nabla \mathcal{L}(w^k_{t,l-1}) \tag{3}$$

$\Delta w^k_{t,l}$ 는 과거의 업데이트들을 이용하여 "재귀적으로" 표현할 수 있기에 다음과 같이 나타난다.

$$\Delta w^k_{t,l} = -\eta^k [\nabla \mathcal{L}(w^k_{t,l-1}) + \rho \nabla \mathcal{L}(w^k_{t,l-2}) + \rho^2 \nabla \mathcal{L}(w^k_{t,l-3}) + \rho^3 \nabla \mathcal{L}(w^k_{t,l-4}) + \cdots] \tag{4}$$

Loss Function의 Gradient는 이번 epoch t에서 minibatch L개에 대해 각각의 iteration l에서의 업데이트에서 사용한 Gradient들의 합으로 표현할 수 있다. (L : number of minibatch)

$$\nabla \mathcal{L}_{acc}(w^k_t) = \sum_{l=1}^{L} [\nabla \mathcal{L}(w^k_{t,l-1}) + \rho \nabla \mathcal{L}(w^k_{t,l-2}) + \rho^2 \nabla \mathcal{L}(w^k_{t,l-3}) + \rho^3 \nabla \mathcal{L}(w^k_{t,l-4}) + \cdots] \tag{5}$$

$Eq5$를 이용하면 다음과 같이 weight change를 표현할 수 있다.

$$\Delta w^k_t = \sum_{l=1}^{L} \Delta w^k_{t,l} = -\eta^k \nabla \mathcal{L}_{acc}(w^k_t) \tag{6}$$

따라서 양변에 norm을 취한 값도 동일하게 되고.

$$\|\Delta w^k_t\| = \eta^k \|\nabla \mathcal{L}_{acc}(w^k_t)\| \tag{7}$$

$Eq2, Eq7$에 의해 relationship between weight variation and learning rate은 다음과 같다. 이때 $\|\nabla \mathcal{L}_{acc}(w^k_t)\|$는 무시 가능한 수준이기에 실제 구현시에는 고려하지 않는다.

$$v^k_t = \frac{\eta^k}{n_k} \|\nabla \mathcal{L}_{acc}(w^k_t)\| \tag{8}$$

Target LR

$$\hat{v}^k_t - v^k_t \approx \frac{\hat{\eta}^k_t - \eta^k_t}{n_k} \|\Delta \mathcal{L}_{acc}(w^k_t)\| = \frac{\hat{\eta}^k_t - \eta^k_t}{\eta^k} v^k_t$$

Note

배워갈 점

CNN의 layer별 feature를 뽑는 것의 차이점을 이용하여 weight variation을 맞춘다는 아이디어가 참신했다. learning rate를 조절하여 효율적인 성능을 보이는 것도 새로웠다.

얻은 아이디어 및 생각

단순히 layer-wise로 성능을 높이는 것 뿐만 아니라, initial weight(=pre-trained model's weight)와의 distance에 대한 제약을 이용하여 일반화 성능을 높이는 방식을 이 연구에 도입하면 더 좋은 알고리즘을 만들 수 있을 것 같다는 생각이 든다.