Learning to Reweight Examples for Denoising

Presenter: Shilin HE

6/13/19

Motivation

DNN is powerful but can easily overfit to training set biases and label noises, e.g.,

1. Different distribution among train/valid/test (e.g., Class imbalance problem)
2. Corrupted data label (label noise)

Zhang et al. (ICLR2017 best paper):
Deep CNNs can memorize the entire dataset even with corrupted labels
(Parameters more than training size by orders-of-magnitude)

Deep CNNs achieve SOTA results but perform poor when trained on corrupted data.

Q: Why do Transfomer and BERT not overfit easily?

Zhang et al. Understanding deep learning requires rethinking generalization

Motivation

To handle noisy labels:

1. Adding regularization either explicitly or implicitly (learned classier barely reaches the optimal performance).
2. Estimating the label transition matrix (hard to estimate when class number is huge)
3. Data selection (reweighting), train on small-loss instances.

MentorNet

MentorNet: Learning Data-Driven Curriculum for Very Deep Neural Networks on Corrupted Labels

Lu Jiang [1] Zhengyuan Zhou [2] Thomas Leung [1] Li-Jia Li [1] Li Fei-Fei [1 2]

The main idea is to use an additional NN (MentorNet) to supervise the training of the base NN (StudentNet)

i.e., a curriculum (sample weight) for StudentNet to focus on samples of correct labels

MentorNet

Traditionally,

1) Curriculum learning (CL) is usually predefined and fixed during training, ignoring the feedback from the student (Heuristics)

2) Alternating Minimization, difficult for training deep CNNs

Key novelty in this paper:

1. Learn the curriculum from the data by a network (MentorNet)
2. Jointly optimize the network (MentorNet and StudentNet)

Approach

Curriculum Learning

weight

$$\min_{\mathbf{w}\in\mathbb{R}^d,\mathbf{v}\in[0,1]^{n\times m}} \mathbb{F}(\mathbf{w},\mathbf{v}) = \frac{1}{n}\sum_{i=1}^{n} \mathbf{v}_i^T \mathbf{L}(\mathbf{y}_i, g_s(\mathbf{x}_i,\mathbf{w})) + G(\mathbf{v};\lambda) + \theta\|\mathbf{w}\|_2^2$$

StudentNet

Loss

Curriculum, can be a function, e.g., $G(\mathbf{v}) = -\lambda\|\mathbf{v}\|_1$

w and **v** are alternatively minimized, update one while the other is fixed

Curriculum cannot be adjusted with student feedback

Approach

MentorNet g_m: compute time-varying weights for each training sample

$$g_m(\mathbf{z}_i; \Theta^*) = \arg \min_{v_i \in [0,1]} \mathbb{F}(\mathbf{w}, \mathbf{v}), \forall i \in [1, n]$$

$\mathbf{z}_i = \phi(\mathbf{x}_i, y_i, \mathbf{w})$ Input to the MentorNet about i–th sample

Θ Parameters of g_m

MentorNet can be learned to

1) approximate existing curriculums
2) discover new curriculums from data.

   Θ Learned on another dataset D' (sampled from training $\{(\phi(\mathbf{x}_i, y_i, \mathbf{w}), v_i^*)\}$

   Clean sample, $v_i = 1$

   Mock exam, for the teacher (MentorNet) to learn to update her teaching strategy (curriculum)

Approach

MentorNet g_m: compute time-varying weights for each training sample

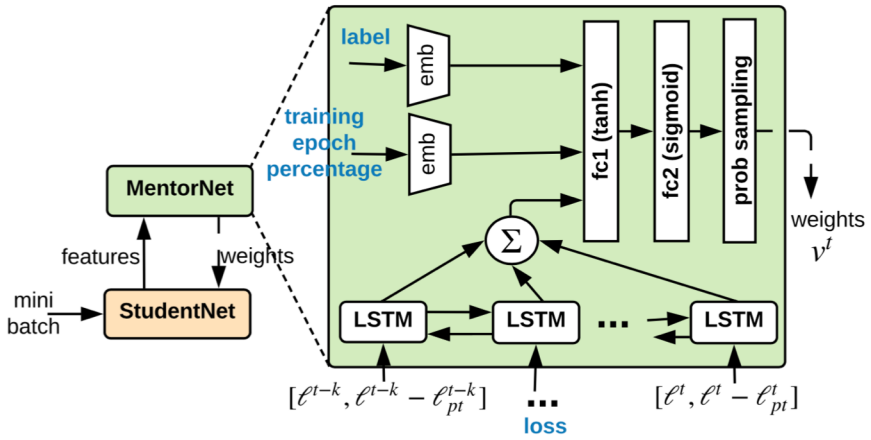

Figure 1. The MentorNet architecture used in experiments. *emb*, *fc* and *prob sampling* stand for the embedding, fully-connected and probabilistic sampling layer.

Input: Mini-batch of samples
Output: Sample Weights

ℓ_{pt} maintains an exponential moving average on the p-th percentile of the loss in each mini-batch

$\ell - \ell_{pt}$ loss difference over last few epochs

However, in experiments, only consider loss and loss difference in current epoch

2-layer perceptron can approximate the existing curriculum

Approach

Algorithm fix v and update w => takes too long to converge

Algorithm 1 SPADE for minimizing Eq. (1)

Input : Dataset $\mathcal{D}$, a predefined G or a learned $g_m(\cdot;\Theta)$
Output : The model parameter $\mathbf{w}$ of StudentNet.

Initialize $\mathbf{w}^0, \mathbf{v}^0, t = 0$
while *Not Converged* **do**
  Fetch a mini-batch Ξ_t uniformly at random
  For every $(\mathbf{x}_i, y_i)$ in Ξ_t compute $\phi(\mathbf{x}_i, y_i, \mathbf{w}^t)$
  if *update curriculum* **then**
    $\Theta \leftarrow \Theta^*$, where Θ^* is learned in Sec. 3.1
  end
  if *G is used* **then**
    $\mathbf{v}_\Xi^t \leftarrow \mathbf{v}_\Xi^{t-1} - \alpha_t \nabla_{\mathbf{v}} \mathbb{F}(\mathbf{w}^{t-1}, \mathbf{v}^{t-1})|_{\Xi_t}$
  end
  else $\mathbf{v}_\Xi^t \leftarrow g_m(\phi(\Xi_t, \mathbf{w}^{t-1}); \Theta)$;
  $\mathbf{w}^t \leftarrow \mathbf{w}^{t-1} - \alpha_t \nabla_{\mathbf{w}} \mathbb{F}(\mathbf{w}^{t-1}, \mathbf{v}^t)|_{\Xi_t}$
  $t \leftarrow t + 1$
end
return $\mathbf{w}^t$

Approximate existing curriculums

$$\mathbf{v}_\Xi^t = g_m(\phi(\Xi_t, \mathbf{w}^{t-1})) = \arg\min_{\mathbf{v}_\Xi} \mathbb{F}(\mathbf{w}^{t-1}, \mathbf{v}^{t-1})$$

V is updated on the small subset D’

V is computed on-the-fly within a mini-batch

No need to allocate a v_i for each example i, computed by the MentorNet on the mini-batch

Experiments

Web-Vision dataset: 2.4 million images of real-world noisy labels, crawled from the web

Table 5. Validation accuracy on the ImageNet ILSVRC12 and Web-Vision validation set. The number outside (inside) the parentheses denotes top-1 (top-5) classification accuracy (%). * marks the method trained using additional verification labels.

Dataset	Method	ILSVRC12	WebVision
Entire	Li *et al.* (2017a)	0.476 (0.704)	0.570 (0.779)
Entire	Forgetting	0.590 (0.808)	0.666 (0.856)
Entire	Lee *et al.* (2017)*	0.602 (0.811)	0.685 (0.865)
Entire	MentorNet	0.625 (0.830)	0.708 (0.880)
Entire	**MentorNet***	**0.642** (**0.848**)	**0.726** (**0.889**)
Mini	FullModel	0.585 (0.818)	-
Mini	Forgetting	0.562 (0.816)	-
Mini	Reed Soft	0.565 (0.827)	-
Mini	Self-paced	0.576 (0.822)	-
Mini	**MentorNet**	**0.638** (**0.858**)	-

Learning to Reweight

Learning to Reweight Examples for Robust Deep Learning

Mengye Ren [1 2] Wenyuan Zeng [1 2] Bin Yang [1 2] Raquel Urtasun [1 2]

Reweighting examples can provide robustness against training set bias

Existing work mainly focus on the loss of each sample (e.g., MentorNet), however,

1. In noisy label problems, we prefer examples with smaller training losses as they are more likely to be clean images

2. in class imbalance problems, algorithms such as hard negative mining (Malisiewicz et al., 2011) prioritize examples with higher training loss since they are more likely to be the minority class

This paper assigns weights to training examples based on ***their gradient directions***.

Learning to Reweight

Weighted Loss:

$$\theta^*(w) = \arg\min_{\theta} \sum_{i=1}^{N} w_i f_i(\theta),$$

W can be understood as hyper-parameters

Updated on validation set

$$w^* = \arg\min_{w, w \geq 0} \frac{1}{M} \sum_{i=1}^{M} f_i^v(\theta^*(w)).$$

Calculating the optimal w requires two-nested optimization loop, *expensive*

The idea is:

for each training iteration

1. inspect the descent direction of some training examples on the training loss surface
2. reweight them according to their similarity to the descent direction of the validation loss surface

Learning to Reweight

Vanilla SGD:

$$\theta_{t+1} = \theta_t - \alpha \nabla \left(\frac{1}{n} \sum_{i=1}^{n} f_i(\theta_t) \right)$$

the impact of example i on validation set?

Perturb the weight by ϵ for sample in mini-batch

$$f_{i,\epsilon}(\theta) = \epsilon_i f_i(\theta),$$

$$\hat{\theta}_{t+1}(\epsilon) = \theta_t - \alpha \nabla \sum_{i=1}^{n} f_{i,\epsilon}(\theta) \Big|_{\theta=\theta_t}.$$

Idea from the Influence Function (Statistics)

Find the optimal ϵ on validation set

$$\epsilon_t^* = \arg\min_{\epsilon} \frac{1}{M} \sum_{i=1}^{M} f_i^v(\theta_{t+1}(\epsilon))$$

Time consuming… =>
Take a mini-batch of validation set only

$$u_{i,t} = -\eta \frac{\partial}{\partial \epsilon_{i,t}} \frac{1}{m} \sum_{j=1}^{m} f_j^v(\theta_{t+1}(\epsilon)) \Big|_{\epsilon_{i,t}=0}$$

$$\tilde{w}_{i,t} = \max(u_{i,t}, 0).$$

"Understanding black-box predictions via influencefunctions." *ICML* (2017)

Learning to Reweight

Algorithm 1 Learning to Reweight Examples using Automatic Differentiation

Require: $\theta_0, \mathcal{D}_f, \mathcal{D}_g, n, m$

Ensure: θ_T

for $t = 0 \dots T-1$ **do**

  $\{X_f, y_f\} \leftarrow \text{SampleMiniBatch}(\mathcal{D}_f, n)$

  $\{X_g, y_g\} \leftarrow \text{SampleMiniBatch}(\mathcal{D}_g, m)$

  $\hat{y}_f \leftarrow \text{Forward}(X_f, y_f, \theta_t)$

  $\epsilon \leftarrow 0; l_f \leftarrow \sum_{i=1}^{n} \epsilon_i C(y_{f,i}, \hat{y}_{f,i})$

  $\nabla\theta_t \leftarrow \text{BackwardAD}(l_f, \theta_t)$

  $\hat{\theta}_t \leftarrow \theta_t - \alpha\nabla\theta_t$

  $\hat{y}_g \leftarrow \text{Forward}(X_g, y_g, \hat{\theta}_t)$

  $l_g \leftarrow \frac{1}{m}\sum_{i=1}^{m} C(y_{g,i}, \hat{y}_{g,i})$

  $\nabla\epsilon \leftarrow \text{BackwardAD}(l_g, \epsilon)$

  $\tilde{w} \leftarrow \max(-\nabla\epsilon, 0); w \leftarrow \frac{\tilde{w}}{\sum_j \tilde{w} + \delta(\sum_j \tilde{w})}$

  $\hat{l}_f \leftarrow \sum_{i=1}^{n} w_i C(y_i, \hat{y}_{f,i})$

  $\nabla\theta_t \leftarrow \text{BackwardAD}(\hat{l}_f, \theta_t)$

  $\theta_{t+1} \leftarrow \text{OptimizerStep}(\theta_t, \nabla\theta_t)$

end for

Validate and compute ϵ

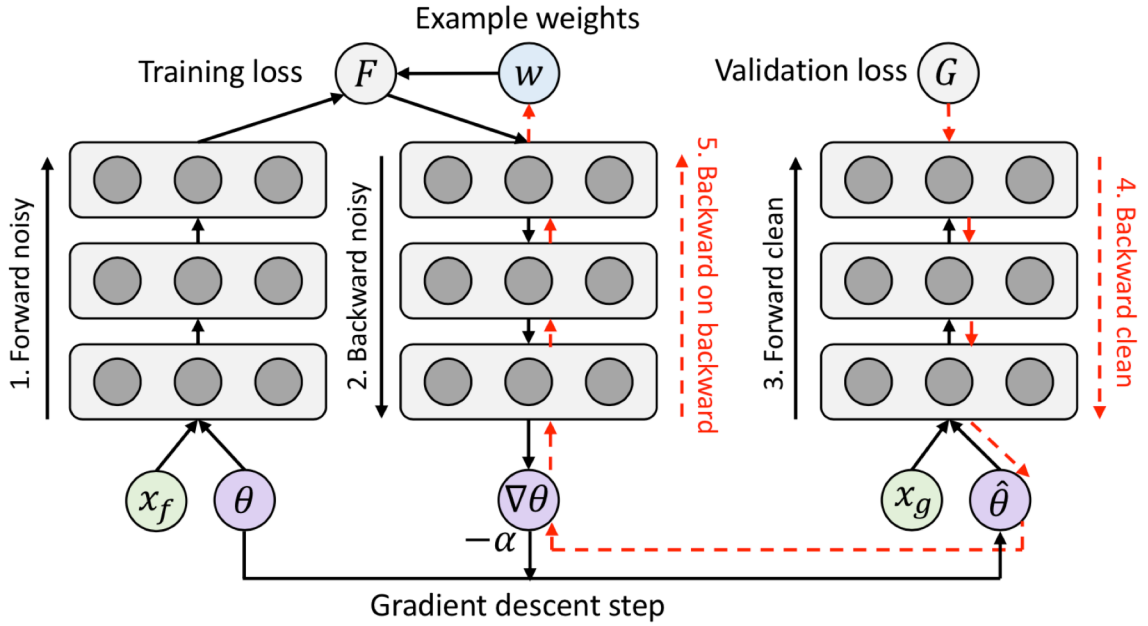

Figure 1. Computation graph of our algorithm in a deep neural network, which can be efficiently implemented using second order automatic differentiation.

Learning to Reweight

Theoretical proof in a FFN shows that:
The gradient of l_g w.r.t ϵ can be decomposed as the product of

1. similarity between training inputs and validation inputs
2. Similarity between training and validation gradient directions

Drawbacks:

1. 3x training time,
   two forward and backward, and another backward on backward process, and a backward to calculate the reweighted objective.

2. Evaluated on MNIST and CIFAR, not very convincing.

Questions

Most of existing work focus on Computer Vision (Classification) area and they share very similar paradigm,

1. Relations among noise, generalization, accuracy? Is noise good or bad?

2. What makes denoising or data reweighting on NLP tasks different? Are noise different? (for example, in NMT)

3. Is it feasible to assign weight to individual words?


THANKS