Supplementary Material for the Paper: 'Local to Global Learning: Gradually Adding Classes for Training Deep Neural Networks'

Hao Cheng[1*], Dongze Lian[1*], Bowen Deng[1], Shenghua Gao[1], Tao Tan[2], Yanlin Geng[3†]
[1] School of Information Science and Technology, ShanghaiTech University, Shanghai 201210, China
[2] Dept of Mathematics and Computer Science, Centre of Analysis, Eindhoven University of Technology
[3] State Key Lab. of ISN, Xidian University, Xi'an 710071, China
{chenghao, liandz, dengbw, gaoshh}@shanghaitech.edu.cn, t.tan1@tue.nl, ylgeng@xidian.edu.cn

Abstract

This is the supplementary material for the paper "Local to Global Learning: Gradually Adding Classes for Training Deep Neural Networks". Section 1 derives the training time of LGL. Section 2 explains the LGL algorithm from an information-theoretic perspective.

1. Training Time of LGL

Proposition 1 *Suppose the training set has K clusters and each cluster has an equal number of samples. We continually add $\frac{K}{m}$ clusters to DNN and train DNN for an equal number of epochs at each step when performing the LGL algorithm. Then the whole training time of LGL is $\frac{1}{2}t(m+1)$, where t is the training time of the baseline which trains DNN on all the training set from the beginning without LGL strategy.*

Proof 1 *LGL trains DNN for m times. At the beginning of step c, $c \in \{1, 2, \ldots, m\}$, there are $\frac{Kc}{m}$ clusters in the training set, which takes $\frac{tc}{m}$ time to train. The whole training time is*

$$\sum_{c=1}^{m} \frac{tc}{m} = \frac{tm(m+1)}{2m} = \frac{t(m+1)}{2}. \tag{1}$$

We can see the training time of LGL is linear in t and m. We can set m to control the training time when performing the LGL algorithm. Also in practice, we do not need to train DNN for an equal number of epochs as the baseline at each step since DNN can converge faster when there are few clusters in the training set. Thus the practical training time of LGL can be less than $\frac{1}{2}t(m+1)$.

2. Information-Theoretic Perspective of LGL for Deep Neural Networks

The term 'entropy' first comes from thermodynamic theory to represent the system's disorganisation. The system tends to be stable when it has a low entropy. Later, entropy was introduced to the information theory to measure unpredictability of the state. Inspired by the definition of entropy, we use $H(T|X)$ to represent DNN's stability. We want to emphasize that the meaning of entropy here is different from the entropy defined in Section 4.4 from our paper. In Section 4.4, the entropy is used to represent the similarity of trained clusters and untrained clusters. While in this section, we utilize entropy to represent the stability of DNN. The former concentrates on the data and the latter concentrates on the model. $H(T|X)$ is written as:

$$\begin{aligned} H(T|X) &= \sum_{x\in X} p(x) H(T|X=x) && (2)\\ &= -\sum_{x\in X} p(x) \sum_{t\in T} p(t|x) \log_2 p(t|x) && (3)\\ &\leq \log_2 C(X) && (4) \end{aligned}$$

$C(X)$ denotes the number of classs of the training set X. Suppose X has M samples, then for each $x \in X$, $p(x) = \frac{1}{M}$. T denotes the output of the softmax layer. For each $t \in \mathcal{T}$, $p(t|x)$ denotes the probability of x belonging to the class with respect to t. From the property of the entropy:

- $H(T|X)$ achieves the maximum, which is $\log_2 C(X)$ from (4), if for each $x \in X$, $p(t|x)$ follows the uniform distribution on $\mathcal{T}$. This means that DNN assigns x to each class with equal probability (this is the most unstable state, since we do not know which class x belongs to).
- $H(T|X)$ achieves the minimum, which is 0, if for each $x \in X$, there exists a t such that $p(t|x) = 1$. This means that DNN assigns x to a class with probability

1, and other classes with probability 0 (the most stable state).

Figure 1 depicts the changes of $H(T|X)$ during training DNN.

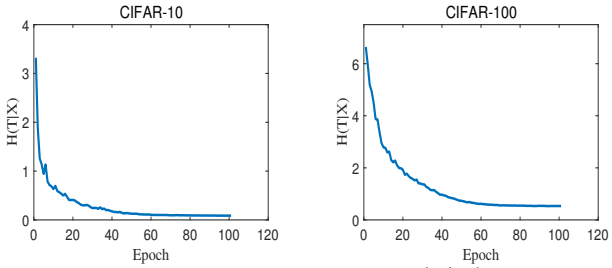

Figure 1. This figure depicts the changes of $H(T|X)$ during training neural networks. X represents the validation data of CIFAR-10 and CIFAR-100. The model is VGG-16.

From Figure 1, DNN can be seen as a system with a large $H(T|X)$ (high unstability) at initial state, then a training algorithm is performed to decrease the $H(T|X)$ of DNN to a stable state (convergence, low $H(T|X)$). Thus $H(T|X)$ is a proper criterion to measure the stability of DNN.

For random initialization, the initial state of DNN (blue circles in Figure 2) almost lies on the maximum bound of $H(T|X)$ since the weights of DNN do not have any information of the training data, the output of the softmax layer tends to form a uniform distribution. Whereas for LGL, the initial $H(T|X)$ of DNN is lower than random initialization each time we add new clusters (yellow circles of Figure 2), which means the training of DNN starts with a more stable point. Section 5.1 in the paper shows the stability of LGL when the model or data distribution varies. Thus the benefit of LGL algorithm is that *LGL can lower the initial $H(T|X)$ to make the training of DNN starts at a more stable state*.

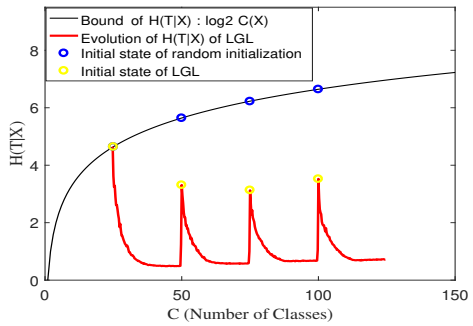

Figure 2. This figure depicts the initial $H(T|X)$ of traditional initialization method and LGL. X represents the validation data of CIFAR-100. The model is VGG-16. The red line contains 4 segments showing the evolution of $H(T|X)$ with training epochs when performing LGL (starts at 25 classes and add 25 each time). Within each segment, the abscissa axis is the training epochs (100 epochs each time). (Best viewed in color)

It is worth noticing that $H(T|X)$ is not completely related to the accuracy since $H(T|X)$ does not include the label. We only use $H(T|X)$ to represent the DNN's stability, and then understand why LGL works better.