CS229 Project Final Report

Reproduce and Explore Variations of SNAPSHOT ENSEMBLES

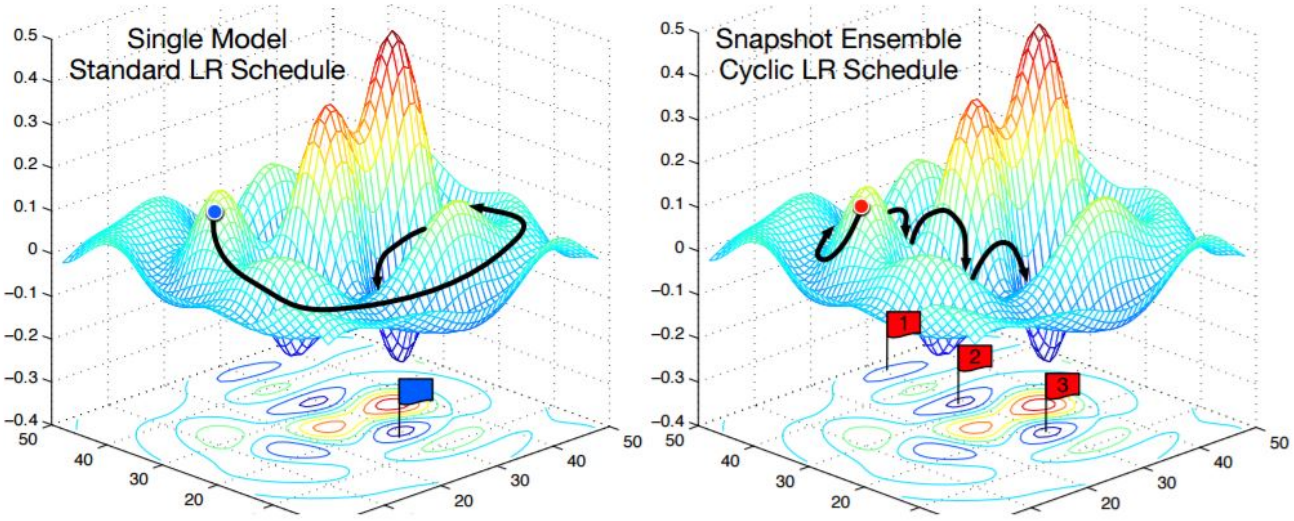
Jiyang Li (jiyangli@stanford.edu), Jin Xi (xijin@stanford.edu)
Project Category: General Machine Learning

Abstract

Ensembles of neural networks are known to be much more robust and accurate than individual networks. However, training multiple deep networks for model averaging is computationally expensive. In this report, we reproduced a method raised in SNAPSHOT ENSEMBLES: TRAIN 1, GET M FOR FREE, which is to obtain the seemingly contradictory goal of ensembling multiple neural networks at no additional training cost. This paper achieve this goal by training a single neural network, converging to several local minima along its optimization path and saving the model parameters. To obtain repeated rapid convergence, this paper leverage recent work on cyclic learning rate schedules. We successfully reproduced their work by implementing the same algorithm across multiple neural network architecture and multiple datasets based on Tensorflow backend. Given the original paper only average all outputs from the M snapshots naively, furthermore, we investigated two different model ensemble methods including weighted averaging ensemble method and Locally weighted ensemble method. By applying these two new ensemble we achieved **better accuracy across all tested datasets/models** compared to the original naive averaging ensemble method. In some cases(CIFAR-100/WRN-16), it could provide **0.5% more accuracy** than original ensemble method.

1. Introduction

SNAPSHOT ENSEMBLES: TRAIN 1, GET M FOR FREE is published as a conference paper at ICLR 2017. In this paper, authors proposed a method to ensemble multiple neural networks at no additional training cost. They achieved this goal by training a single neural network, converging to



several local minima along its optimization path and saving the model parameters.

It can be properly described using the following image([1] Gao & Yixuan et al., 2017). Left image is an illustration of SGD optimization with a typical learning rate schedule. The model converges to a minimum at the end of training. Right image is an illustration of Snapshot Ensembling. The model undergoes several learning rate annealing cycles, converging to and escaping from multiple local minima. We take a snapshot at each minimum for test-time ensembling.

Based on this paper, our team project is to reproduce the their work on Tensorflow backend(the original paper ran experiments on Torch) and verify the results declared in the paper. Furthermore, we also investigated two different model ensemble methods (including weighted averaging ensemble method and Locally weighted ensemble method) compared to the naive averaging ensemble method in the original paper. Which is proved to achieve better accuracy across all tested datasets/models.

2. Related work

SNAPSHOT ENSEMBLES: TRAIN 1, GET M FOR FREE is the paper we want to reproduce where the authors proposed a method to ensemble multiple neural networks at no additional training cost.

The original paper takes a regular deep learning architecture, run it with cyclic learning rate schedules, take several snapshots of the local minima, and ensemble these network snapshots to provide a lower error rate.

The original paper evaluated the Snapshot Ensembles on three state-of-the-art deep learning architectures for object recognition: ResNet, Wide-ResNet and DenseNet. And then it uses Cyclic Cosine Annealing as learning rate schedule.

The original work is evaluated on CIFAR datasets (Krizhevsky & Hinton, 2009), The Street View House Numbers (SVHN) dataset (Netzer et al., 2011) and Tiny ImageNet dataset.

https://github.com/titu1994/Snapshot-Ensembles

is a code framework which implements the method raised in previous paper based on tensorflow backend. The author also raised up weighted averaging ensemble method to better ensemble M models, but his experiment results perform even worse on some cases compared to the original naive averaging ensemble method, we identified the source reason is that he reused the same training datasets(used already for generating the M snapshot ensembles) to learn the averaging weights improperly. We identified this error and allocated another validation set from test set to train the weighted averaging ensemble model. Which proved to achieve better accuracy across all tested datasets/models compared to original paper.

3. Dataset and Features

CIFAR10. Dataset of 50,000 32x32 color training images, labeled over 10 categories, and 10,000 test images.

CIFAR100. Dataset of 50,000 32x32 color training images, labeled over 100 categories, and 10,000 test images.

4. Methods

The Snapshot Ensemble algorithm has three major components: neural network, learning rate schedule and snapshot selection strategy, and model ensemble algorithm. It first run a training process to train neural network models in various architecture. During the training, it keeps several snapshots of the model (weights of the model). And assemble these model snapshots into a single model. In order to avoid very similar snapshots, the original work uses

a cosine annealing cycles learning rate schedule, in which the learning rate is decreased during each cycle, and reset to a large value when a new cycle starts. As a result, during each cycle, the learning algorithm will get close to some local minima, and escape the local minima when next cycle starts. This schedule helps to find snapshot models that's less correlated to each other. And finally ensemble the model snapshots together, by simply taking average model outputs.

The original work has experiments to evaluate Snapshot Ensemble on various Neural Network frameworks: ResNet ([2], He et al., 2016), Wide ResNet ([3], Zagoruyko & Komodakis, 2016) and DenseNet ([4], Huang et al., 2016). It also compared the cosine annealing cycles learning rate schedule. But the original work didn't check efficiency of different model ensemble methods, which is the focus of our project.

The method used in original paper is simple averaging all outputs from the M snapshots:

$$\hat{y}_{\text{ensemble}} = \frac{1}{M}\sum_{i=1}^{M}\hat{y}_i$$

In this method, every model contribute to the final ensemble output evenly. It doesn't make much sense as the first several snapshots are tend to be underfitting (and have lower accuracy, comparing to following snapshots). We can assign a weight to each model, and use the weighted average as ensemble output:

$$\hat{y}_{\text{ensemble}} = \frac{\sum_{i=1}^{M} w_i\hat{y}_i}{\sum_{i=1}^{M} w_i}$$

That can be modeled as a linear regression problem. The features are M snapshots' output. The optimal weights can be learned using linear regression learning algorithms.

Besides the simple averaging and weighted averaging, we also proposed our own methods, to ensemble the snapshots.

Locally weighted ensemble method. This is inspired by the locally weighted linear regression, which was taught on class. For a test sample, the M snapshot models may give different results, if we found some training sample have the same set of prediction results as the test sample, then they are similar. An imaginary example is: if model A category the sample as a cat, and model B category it as a bird, then it's likely an owl. In practice, we define a similarity function for two examples a and b, using the prediction outputs from the M snapshot models:

$$\text{sim}(a,b) = e^{-\sum_{i=1}^{M}||\hat{y}_a - \hat{y}_b||_2^2}$$

Then for each test sample, we can go through all training samples, and take the weighted average of training labels:

$$\hat{y}_{\text{ensemble}}(a) = \frac{\sum_{i=1}^{m_{train}} sim(a, train_i)y_i}{\sum_{i=1}^{m_{train}} sim(a, train_i)}$$

Note that this method need to go through all training samples for each test sample. If we need to improve the efficiency, we can do some approximation: First, map the continuous predictions into discrete buckets, then use this similarity function that output 1 if a and b are in same bucket, and 0 otherwise. Using this optimization, we can preprocess the training samples, put them into the buckets, and for each test, all we need to do is to read the majority label from its corresponding bucket.

We have another idea that we didn't have a chance to implement and do experiment on it. The method is: for each model, calculate the Gaussian distribution of all samples where it provides correct prediction; then for each test sample, find the model whose distribution best match the test sample, and use that model's output.

5. Experiments

5.1 Experiment Configuration

Architectures. We used both Wide Residual Network (WRN-16, a 16 layer ResNet with 4 times as many convolutional features per layer as a standard ResNet; and WRN-22, a 22 layer ResNet with 4 times as many convolutional features per layer as a standard ResNet). We also tried a DenseNet model, which has 40 layers, and its growth rate is 12.

Baselines. We commit to the baselines that all the model went through the same amount of training.

Training Budget. B = 200 epochs. M = 5 cycles. Each snapshot variant is trained with B/M = 40 epochs.

Learning Rate Schedule. We use the Cyclic Cosine Annealing schedule, same as the paper proposed. At the beginning of each cycle, restart learning rate α_0 = 0.1 ([5], Loshchilov & Hutter 2016)

$$\alpha(t) = \frac{\alpha_0}{2}\left(\cos\left(\frac{\pi \mod (t-1, \lceil T/M \rceil)}{\lceil T/M \rceil}\right) + 1\right)$$

Backend. Keras and Tensorflow.

5.2 Results

Experiment 1: In the first experiment, we tried to verify that Snapshot Ensemble can help improve the prediction accuracy.

	CIFAR10	CIFAR100
Single Best Model	8.53%	31.58%
Snapshot Ensemble (α_0 = 0.1)	8.36%	30.62%

Table 1: Error rates (%) on CIFAR-10 and CIFAR-100 datasets.

In this experiment, we use WRN-16 model, Cyclic Cosine Annealing schedule with α_0 = 0.1. We achieved different error rate compared to the author, but it could be two reasons: author is building model on 32-layer Wide ResNet, while we are building it on 16-layer Wide ResNet to save training time; author is using the backend known as Torch, while we are using Keras on Tensorflow. But we can still see error rate decrease when switch from single model to Snapshot Ensemble.

Experiment 2: In the second experiment, we tried different ensemble methods: no ensemble (take the best single model), simple averaging, weighted averaging, and the locally weighted ensemble method. The results are shown in table 2.

Error rate	CIFAR-10 / WRN-16	CIFAR-10 / WRN-22	CIFAR-100 / WRN-16	CIFAR-100 / WRN-22
Single Model Best	8.53%	8.71%	31.58%	29.82%
Avg	8.39%	**8.64%**	**30.91%**	**29.0%**
Weighted Avg	**8.34%**	8.67%	31.17%	29.2%
Locally weighted	8.42%	8.7%	30.95%	29.03%

Table 2: Comparison of different ensemble methods.

In the table, the bolded green numbers are best result for given dataset and neural network architecture. Note that for each column, we only train the model once, and saved the predictions for ensemble.

The experiment result shows that simple averaging performs better than more complex methods in 3 out of 4 cases.

In theory, weighted averaging should perform better than simple averaging by assigning smaller weights to earlier models. However, in this experiment, the weights are learned using the training sample again. And for some model, the learned weight are extremely small, which is equivalent to ignoring these snapshots. As a result, for 3 out of 4 cases, the weighted averaging gives a worse error rate than the simple averaging. The problem is that these snapshot

models are trained using training data, and if we optimize the weights using the same set of training data, there will be overfitting problem. We'll show more about this in the third experiment.

The locally weighted method we proposed also failed to compete over simple averaging. One possible reason is that as these models are generated sequentially, their predictions are highly correlated. So the similarity function we defined doesn't provide a good estimate on the similarity of different samples.

Experiment 3: In the third experiment, we further evaluate the weighted averaging method. To avoid overfitting we saw in experiment 2, we allocated a dedicate validation set, and optimize the weights using only the validation set.

Error rate	CIFAR-10 / WRN-16	CIFAR-10 / WRN-22	CIFAR-100 / WRN-16	CIFAR-100 / WRN-22
Single Model Best	8.35%	8.76%	31.82%	29.79%
Avg	8.34%	8.58%	31.13%	29.01%
Weighted Avg (with validation set)	**8.28%**	**8.52%**	**30.61%**	**28.96%**

Table 3: Comparing weighted averaging (with validation set) and simple averaging.

In this experiment, we split the test set with 10000 examples into validation set with 2000 examples and test set with 8000 examples. All rows in table 3 are evaluated on the same 8000 examples.

This time, we can see that weighted averaging gives a more accurate ensemble model.

6. Conclusion and Future Work

In this project, we reproduced the original paper's snapshot ensemble method. Because we use different neural network architecture (our experiments are running on smaller model due to limited computation resources), we didn't reach same level of accuracy as the original paper, but we showed that snapshot ensemble does improve the accuracy.

In addition to that, we compared different ensemble methods, which is not covered in the original paper. We proposed several new ideas to improve the ensemble model, and showed that weighted averaging (if trained using a stand-alone validation set) can further improve the accuracy of ensemble model. Our experiments also showed that when weights are trained using training set, there is a overfitting problem, that makes the weighted averaging result worse than the simple averaging.

Because of limited time and limited computation resource we have, there are some future work not covered:

- Implement our idea that uses Gaussian distribution (see last paragraph in section 4), and try other ideas to assemble the snapshot models.
- Explore other strategies to take the snapshots.
- Apply the locally weighted method to assemble models trained by different methods, and evaluate the result.

Contributions

Jin: Built personal machine with one GTX 1080 GPU, configured CUDA Tensorflow with GPU environments, investigated the github code Snapshot Ensembles, raised up Gaussian distribution ensemble method, reproduced experiments on Wide-ResNet-16 model on CIFAR-10 and CIFAR-100 datasets, worked on experiments result analysis, contributed in write-up for project milestone, contributed in write-up for project final report.

Jiyang: investigated the github code Snapshot Ensembles, raised up Locally weighted ensemble method, identified improvement for weighted averaging ensemble method, reproduced experiments on Wide-ResNet-22 model on CIFAR-10 and CIFAR-100 datasets, worked on experiments result analysis, contributed in write-up for project milestone, contributed in write-up for project final report.

References

[1] Gao Huang∗ , Yixuan Li∗ , Geoff Pleiss, Zhuang Liu, John E. Hopcroft, Kilian Q. Weinberger. *SNAPSHOT ENSEMBLES: TRAIN 1, GET M FOR FREE*. In ICLR 2017.

[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. *Deep residual learning for image recognition*. In CVPR, 2016

[3] Sergey Zagoruyko and Nikos Komodakis. *Wide residual networks*. arXiv preprint arXiv:1605.07146, 2016.

[4] Gao Huang, Zhuang Liu, and Kilian Q Weinberger. *Densely connected convolutional networks*. arXiv preprint arXiv:1608.06993, 2016

[5] Ilya Loshchilov and Frank Hutter. *Sgdr: Stochastic gradient descent with restarts*. arXiv preprint arXiv:1608.03983, 2016.

[6] titu1994. Snapshot Ensembles in Keras. https://github.com/titu1994/Snapshot-Ensembles