Learning without Forgetting – summary

Matěj Nikl

August 6, 2016

This paper is about a method called Learning without Forgetting (LwF), which focuses on the task of incremental learning of new capabilities without forgetting the old ones. On top of that, this method can accomplish this task without the need for the training data the old capabilities were trained with.

It resembles the combination of *Knowledge Distilling* and transfer learning strategy *fine-tuning*.

1 The Architecture

The LwF's view on a model consists of:

- shared parameters θ_s
- task-specific parameters for previously learned tasks θ_o
- randomly initialized task-specific parameters for new tasks θ_n

It is useful to think of θ_o and θ_n as classifiers (fully connected layers) that operate on features parameterized by θ_s (see Figure 1)

2 Learning without Forgetting

Given a (C)NN with θ_s and θ_o, the goal is to add θ_n for new tasks and learn all the parameters that work well on both old and new tasks, using only labeled data for the new tasks:

1. record responses (probability distributions over classes) y_o for each example of the new dataset from the original network (defined by θ_s and θ_o)
2. add new fully connected classifiers θ_n on top of θ_s, that will compute the new tasks' class probability distributions
3. fine-tune θ_n using supervised cross-entropy loss 2.1 (sum over losses if multiple new tasks) until convergence (having θ_s (and θ_o) frozen)
4. train all parameters jointly using a (weighted) sum of supervised cross-entropy loss and Knowledge Distillation loss 2.2 (sum over losses if multiple old tasks) until convergence

2.1 Cross-entropy loss

$$\mathcal{L}_{\text{new}}(y_n, \hat{y}_n) = -y_n \log \hat{y}_n$$

2.2 Knowledge Distillation loss

The Knowledge Distillation loss adds the objective of keeping the outputs of the old task classifiers for all of the new dataset inputs the same as they were, before the learning of the new tasks began.

$$\begin{aligned}\mathcal{L}_{\text{old}}(y_o, \hat{y}_o) &= D_{\text{KL}}(y'_o \| \hat{y}'_o) \\ &= -\sum_{i=1}^{l} y_o'^{(i)} \log(\hat{y}_o'^{(i)}) \\ &= -\sum_{i=1}^{l} \left(\frac{e^{y_o^{(i)}/T}}{\sum_j e^{y_o^{(j)}/T}} \log \left(\frac{e^{\hat{y}_o^{(i)}/T}}{\sum_j e^{\hat{y}_o^{(j)}/T}} \right) \right)\end{aligned}$$

where l is the number of classes and T is the temperature of the softmax function. The recommended setting is $T > 1$, because then the weight of smaller logit is bigger and encourages the network to better encode similarities among classes.

3 Principles of modularization

Each of the θ_o output classifiers can be seen as a module. There is no need for the network to always output all θ_o outputs – only a selected output classifier for the selected task at hand can be connected to produce the desired output. The output classifiers, however, ale strongly tied to the shared parameters θ_s with whom they have been trained with. The output modules are thus not transferable to a different model with different θ_s.

4 Principles of growing

For each new task a new output layer θ_n is crated. The whole set of parameters θ is adjusted:

- θ_s and θ_o in such a way that they allow for the new task to be learned, while also retaining the old knowledge
- θ_n in such a way that it performs well on the new task

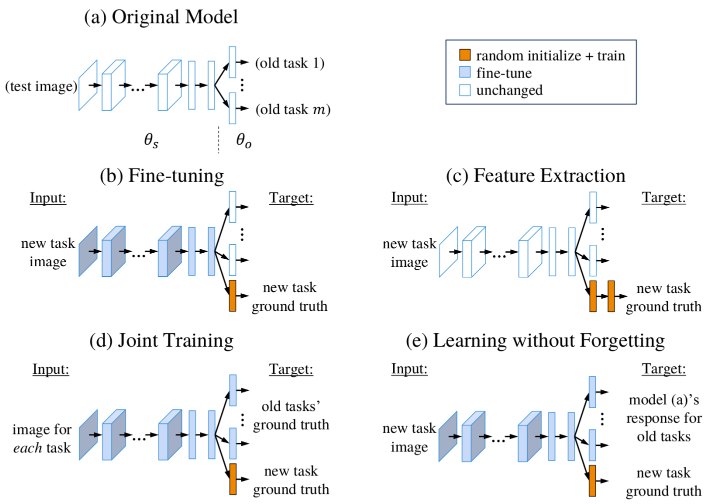
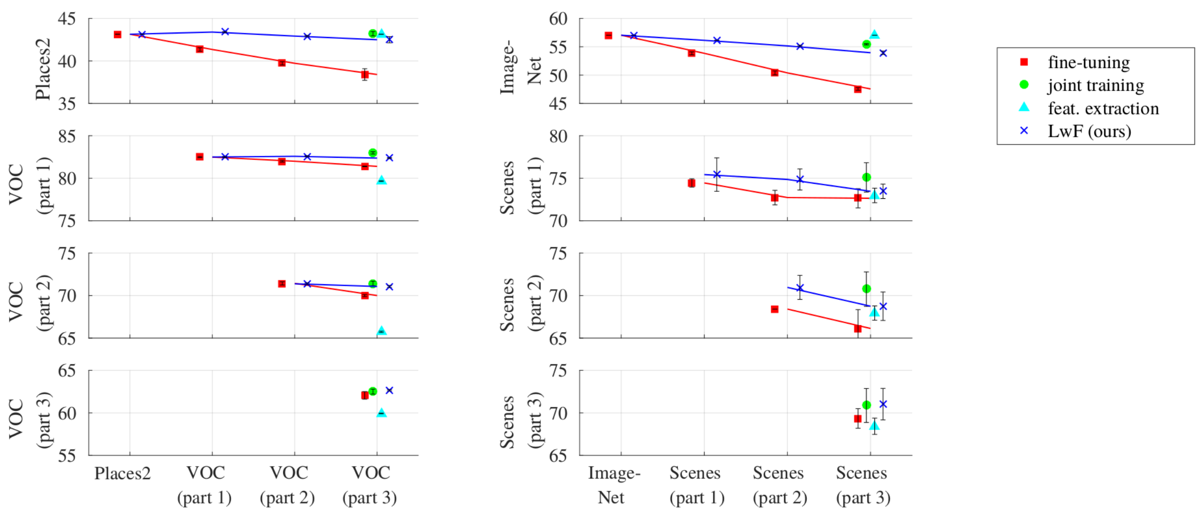


Figure 1: Illustration of LwF method (e) and other methods (b-d).

	ImageNet→VOC		ImageNet→CUB		ImageNet→Scenes		Places2→VOC		Places2→CUB		Places2→Scenes		ImageNet→MNIST	
	old	new	old	new	old	new	old	new	old	new	old	new	old	new
LwF (ours)	56.5	75.8	55.1	57.5	55.9	64.5	43.3	72.1	38.4	41.7	43.0	75.3	52.1	99.0
fine-tuning	-1.4	-0.3	-5.1	-1.5	-3.4	-1.0	-1.8	-0.1	-9.1	-0.8	-4.1	-0.8	-4.9	0.2
feat. extraction	0.5	-1.1	2.0	-5.3	1.2	-3.7	-0.2	-3.9	4.7	-19.4	0.2	-0.5	5.0	-0.8
joint training	0.2	0.0	0.5	-0.9	0.5	-0.6	-0.1	0.1	3.3	-0.2	0.2	0.1	4.7	0.2

Table 1: Performance for the single new task scenario using AlexNet structure. The difference of methods' performance with LwF is reported to facilitate comparison.



(a) Places2→VOC (b) ImageNet→Scenes

Figure 2: Performance of each task when gradually adding new tasks to a pre-trained network. The x-axis labels indicate the new task added to the network each time. Error bars shows ± 2 standard deviations for 3 runs with different θ_n random initializations. Markers are jittered horizontally for visualization, but line plots are not jittered to facilitate comparison.