Advanced Machine Learning

Paper: *Learning without forgetting* [10]

Romain Mormont

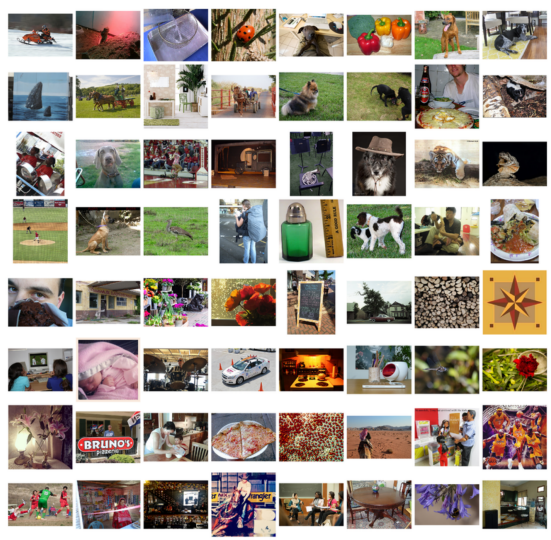
Montefiore Institute, ULiège

18 March 2019

Image classification and transfer learning

Deep learning works well with images...
as long as we have enough of them !

What if we have a small dataset ?
$\Rightarrow$ **transfer learning** from a larger database (e.g. ImageNet) !



https://goo.gl/zv8P35

Transfer learning

> "**Transfer learning** *aims to produce an effective model for a target task with limited or no labeled training data by leveraging and exploiting knowledge from a different, but related source domain to predict the truth label for an unseen target instance.*"
>
> *[3]*

We consider homogeneous transfer learning:

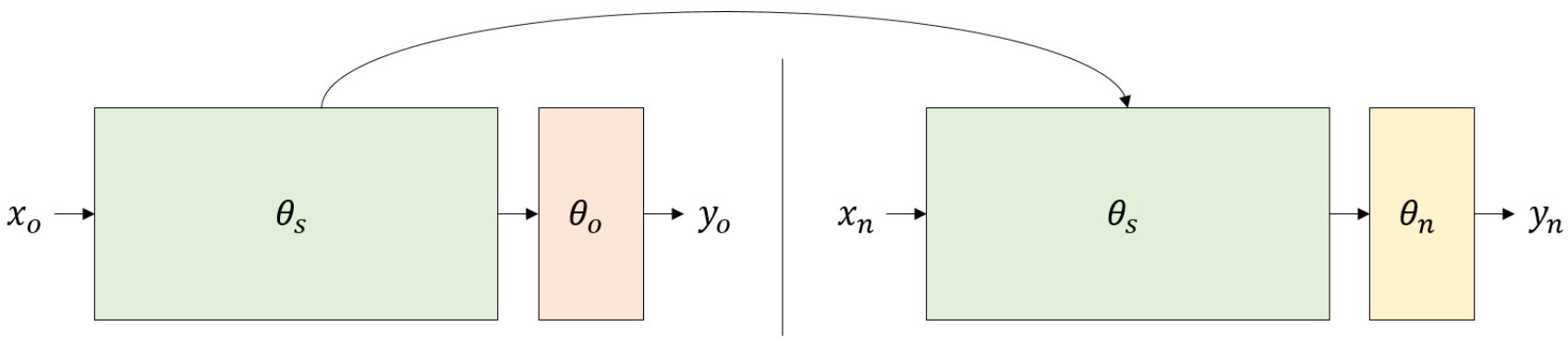
- a **source** task $(\mathcal{X}_o, \mathcal{Y}_o, p_o(x, y))$ (**o**ld task)
- a **target** task $(\mathcal{X}_n, \mathcal{Y}_n, p_n(x, y))$ (**n**ew task)
- homogeneity: $\mathcal{X}_o = \mathcal{X}_n$ (images) but $p(x_o) \neq p(x_n)$
- different tasks: $\mathcal{Y}_o \neq \mathcal{Y}_n$
- usually: $N_n \ll N_o$

Transfer learning

We consider the following network components:

- θ_s: shared parameters
- θ_o: source task-specific parameters
- θ_n: target task-specifc parameters

θ_s and θ_o are **pre-trained on the source task** and θ_n is **initialized randomly**.



How to transfer ?

There exists several transfer techniques:

- **feature extraction**: θ_s and θ_o are unchanged. Features are extracted from one or more layers to train θ_n or another classifier.
- **fine-tuning**: both θ_s and θ_n are optimized on the target task (θ_o fixed).
- **partial fine-tuning**: similar to fine-tuning but early layers of the shared network are frozen. Only a part of θ_s is optimized along with θ_n.
- **joint/multitask training**: θ_s, θ_n and θ_o are all optimized with both tasks at the same time (e.g. by interleaving samples).
- ...

Feature extraction I

How to transfer ?

θ_s and θ_o are unchanged. Features are extracted from one or more layers to train θ_n or another classifier.

The idea of extracting features from pre-trained networks arrived not so long after the deep learning revolution:

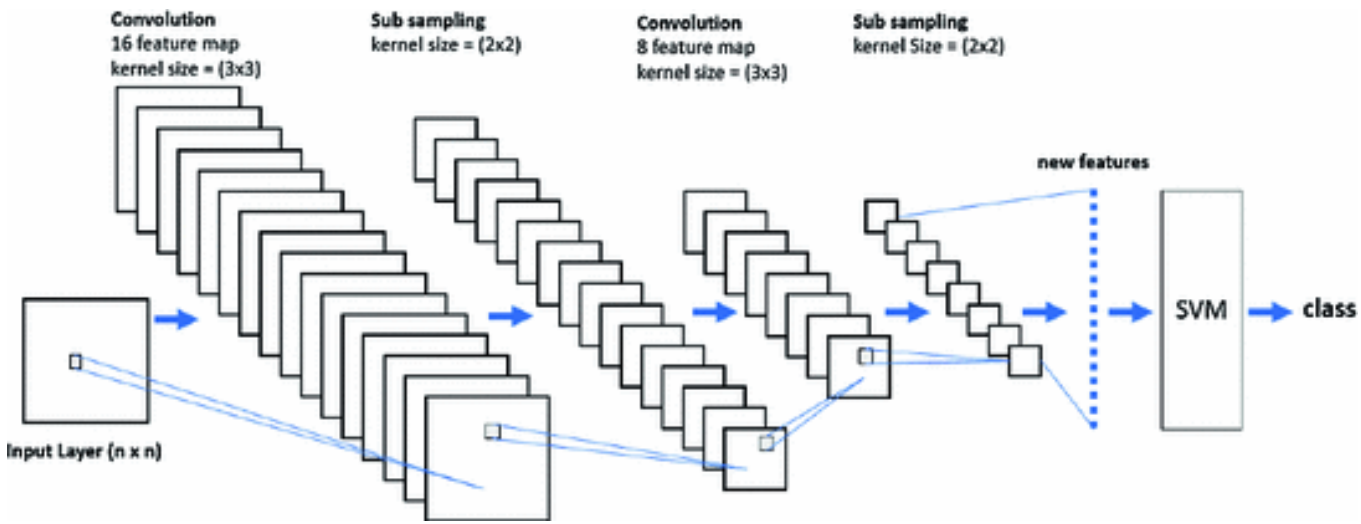
- Decaf [4] (2013.10): feature extractor based on AlexNet
- Overfeat [11] (2013.12): another feature extractor based on AlexNet

Nowadays, all modern deep learning frameworks provide pre-trained weights for **well-known architectures** (e.g. ResNet, DenseNet, GoogLeNet,...) which can then be used as feature extractors.

Feature extraction II

One usually extracts **one or more vector of features** per input image.

The extracted vectors can then be used to train classifier with **traditional machine learning** methods (e.g. SVM, ANN, random forests,...). Simple linear models are often enough to obtain competitive performances.



Feature extraction III

Advantages

- no need for training the network $\Rightarrow$ fast to apply
- benefit from complex features learnt by the transferred network
- deep features often yield better performances than handcrafted features

Drawbacks

- features are not task-specific $\Rightarrow$ **under-performing**

Conclusion

Feature extraction provides a **good baseline** for image classification and transfer learning.

Fine-tuning I

How to transfer ?

Both θ_s and θ_n are optimized on the target task (θ_o fixed).

Fine-tuning has been applied with great success in many applications and fields [5, 7].

All parameters are trained to **minimize a loss on a new task**, usually with a **small learning rate**. Fine-tuning is a way of making the features more discriminative for the new task. Using a small learning rate is a way of preserving the representations learnt from the source task.

Fine-tuning II

Advantages

- accelerate convergence at training
- learned features are more task-specific
- often **outperform features extraction** or learning from random weights

Drawbacks

- subject to **catastrophic forgetting**

Conclusion

One of the most efficient approaches when tackling a new classification task.

Catastrophic forgetting

> *"[Catastrophic forgetting] refers to the catastrophic loss of previously learned responses, whenever an attempt is made to train the network with a single new (additional) response. Affected networks include, for example, backpropagation learning networks."*
>
> *[2]*

Catastrophic forgetting (CF):

- a neural network has no built-in mechanism to retain previously learnt information
- when fine-tuning, CF is actually a consequence of overfitting the new task [8]
- mechanisms aimed at handling this problem would actually act as regularization during training and would **reduce overfitting**

Less/no catastrophic forgetting means better transfer !

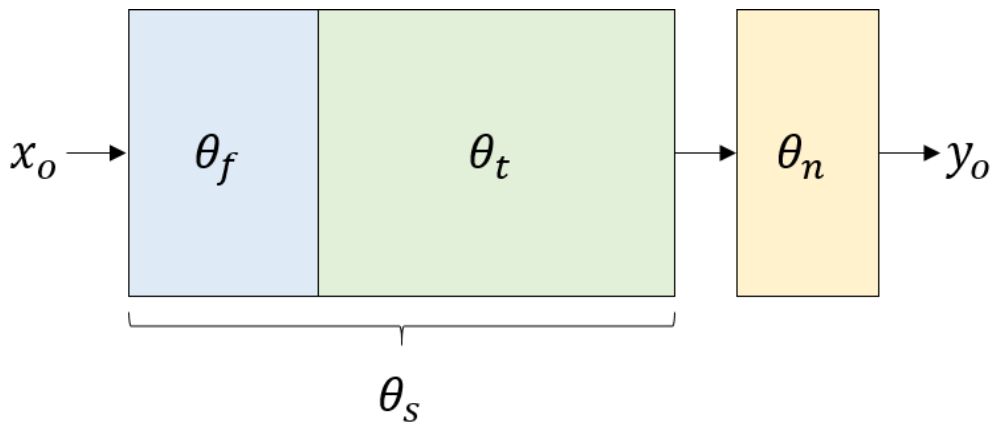
Partial fine-tuning I

How to transfer ?

Similar to fine-tuning but early layers of the shared network are frozen. Only a part of θ_s is optimized along with θ_n.

Let $\theta_f \subseteq \theta_s$ be the set of frozen parameters and $\theta_t = \theta_s \setminus \theta_f$ be the set of trainable parameters. Partial fine-tuning is a **compromise between features extraction and fine-tuning**, as it falls back to:

- fine-tuning when $\theta_f = \emptyset$
- features extraction when $\theta_f = \theta_s$



Partial fine-tuning II

Similarly to fine-tuning, the parameters θ_t and θ_n are trained to **minimize a loss on the new task**.

Partial fine-tuning offers a mechanism for mitigating **catastrophic forgetting and overfitting** by fixing a part of the network.

However, it doesn't solve the problem especially when few parameters are frozen.

Partial fine-tuning III

Advantages

- mechanism for dealing with catastrophic forgetting
- can accelerate training, as less parameters to optimize

Drawbacks

- the extent of θ_f is task-specific
- catastrophic forgetting is not avoided

Conclusion

Partial fine-tuning offers a mechanism for dealing with catastrophic forgetting but is not the final solution

Joint training I

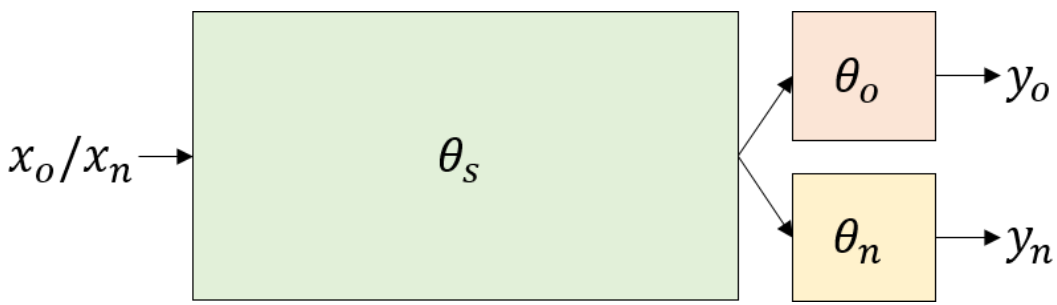
How to transfer ?

θ_s, θ_n and θ_o are all optimized with both tasks at the same time.

Also known as **multi-task training** [1]. Both tasks are provided at the same time (instead of sequentially).

Regarding training, several questions arise:

- interleaving samples ? interleaving tasks ?
- how to combine task-specific losses ?
- what about imbalance when $N_o \gg N_n$?



Joint training II

Advantages

- catastrophic forgetting is not an issue, or is greatly mitigated

Drawbacks

- more data $\Rightarrow$ longer training time
- require access to the (potentially large) source dataset

Conclusion

Joint training deals with catastrophic forgetting at the expense of longer training time and is not possible if the source dataset is unavailable.

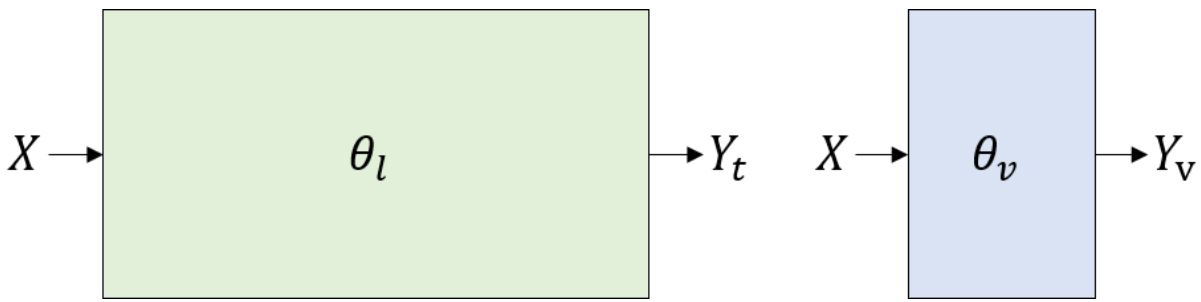
Summary

	Feature Extraction	Fine Tuning	Joint Training
new task performance	X medium	good	best
original task performance	good	X bad	good
training efficiency	fast	fast	X slow
storage requirement	medium	medium	X large
requires previous task data	no	no	X yes

Note: knowledge distillation

There exists single task transfer techniques that consists in tranferring knowledge learnt by a large network θ_l into a significantly smaller network θ_v with $|\theta_l| \gg |\theta_v|$.

One of such techniques is called **distillation** [9] and consists in optimizing the small network **to replicate the probabilities** produced by the larger network for a same input sample using a distillation loss.



Learning without forgetting

Learning without Forgetting

Zhizhong Li, Derek Hoiem, *Member, IEEE*

Abstract—When building a unified vision system or gradually adding new capabilities to a system, the usual assumption is that training data for all tasks is always available. However, as the number of tasks grows, storing and retraining on such data becomes infeasible. A new problem arises where we add new capabilities to a Convolutional Neural Network (CNN), but the training data for its existing capabilities are unavailable. We propose our Learning without Forgetting method, which uses only new task data to train the network while preserving the original capabilities. Our method performs favorably compared to commonly used feature extraction and fine-tuning adaption techniques and performs similarly to multitask learning that uses original task data we assume unavailable. A more surprising observation is that Learning without Forgetting may be able to replace fine-tuning with similar old and new task datasets for improved new task performance.

Index Terms—Convolutional Neural Networks, Transfer Learning, Multi-task Learning, Deep Learning, Visual Recognition

1 Introduction

Many practical vision applications require learning new visual capabilities while maintaining performance on existing ones. For example, a robot may be delivered to someone's house with a set of default object recognition capabilities, but new site-specific object models need to be added. Or for construction safety, a system can identify whether a worker is wearing a safety vest or hard hat, but a superintendent may wish to add the ability to detect improper footware. Ideally, the new tasks could be learned while sharing parameters from old ones without

network could be duplicated and fine-tuned for each new task to create a set of specialized networks.

It is also possible to use a variation of fine-tuning where part of θ_s – the convolutional layers – are frozen to prevent overfitting, and only top fully connected layers are fine-tuned. This can be seen as a compromise between fine-tuning and feature extraction. In this work we call this method **Fine-tuning FC** where FC stands for fully connected.

Joint Training (e.g. [7]): All parameters θ_s, θ_o, θ_n are

Learning without forgetting

Timeline:

2016.09 - First publication on *arXiv*

2017.11 - Published in *IEEE Transactions on Pattern Analysis and Machine Intelligence, Volume 40*

Summary:

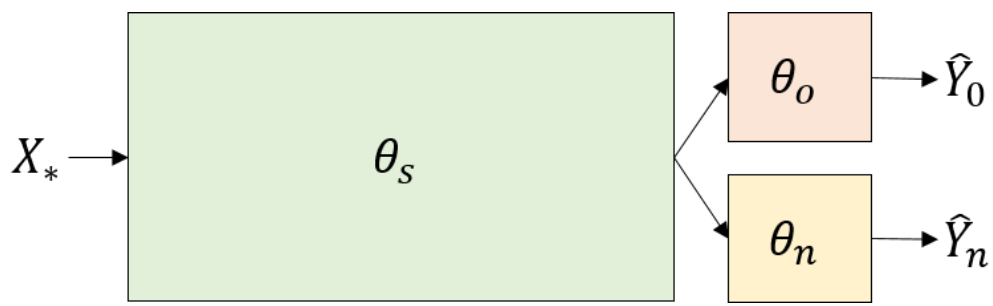
- transfer learning for image classification with deep convolutional network
- new method for training a pre-trained network on a new task
- **goal**: avoid "*forgetting*" what was learnt on the source task
- **idea**: a loss function designed to preserve the network response for the old task during training on the new task

Method I

Inputs:

- shared parameters θ_s and θ_o pre-trained on the source task
- target task data X_n and Y_n
- new layer(s) with parameters θ_n attached to the last layer of the shared network

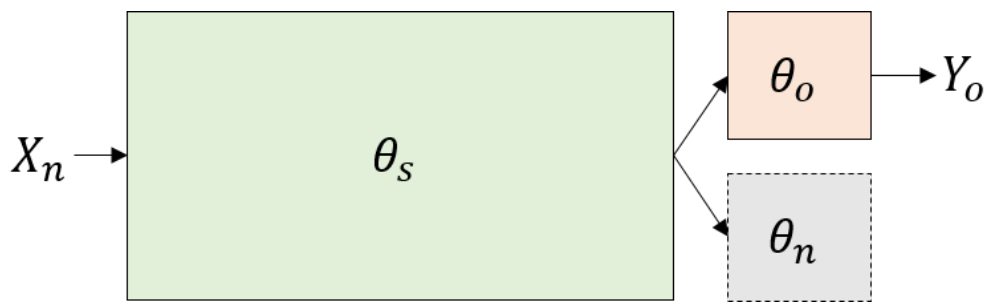
No need for source task data !



Method II

Initialization:

- extraction of "old" network responses Y_o (i.e. old task classes probabilities/softmaxes) to the new data X_n
- θ_n parameters random initialization



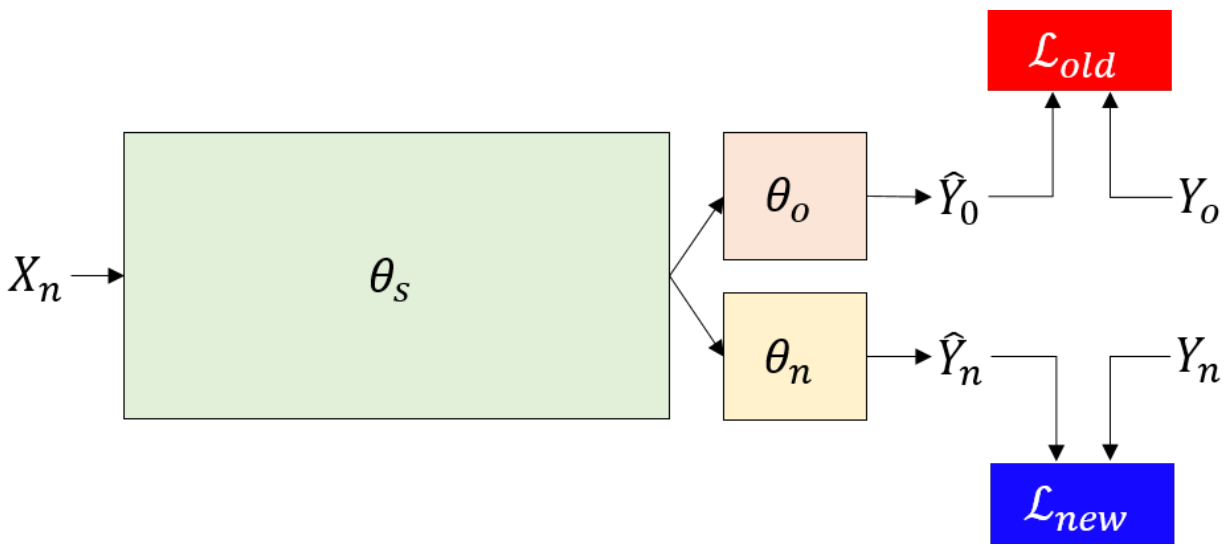
Method III

Training: jointly optimize θ_s, θ_o and θ_n with stochastic gradient descent:

$$\hat{\theta}_s^*, \hat{\theta}_o^*, \hat{\theta}_n^* = \underset{\hat{\theta}_s, \hat{\theta}_o, \hat{\theta}_n}{\operatorname{argmin}} \left(\lambda_0 \mathcal{L}_{old} \left(Y_0, \hat{Y}_0 \right) + \mathcal{L}_{new} \left(Y_n, \hat{Y}_n \right) + \mathcal{R} \left(\hat{\theta}_s, \hat{\theta}_o, \hat{\theta}_n \right) \right)$$

where:

- $\lambda_0 \in \mathbb{R}$ is a loss balance weight
- $\mathcal{L}_{old}$ is the old task loss
- $\mathcal{L}_{new}$ is the new task loss
- $\mathcal{R}$ is a regularization term (weight decay)

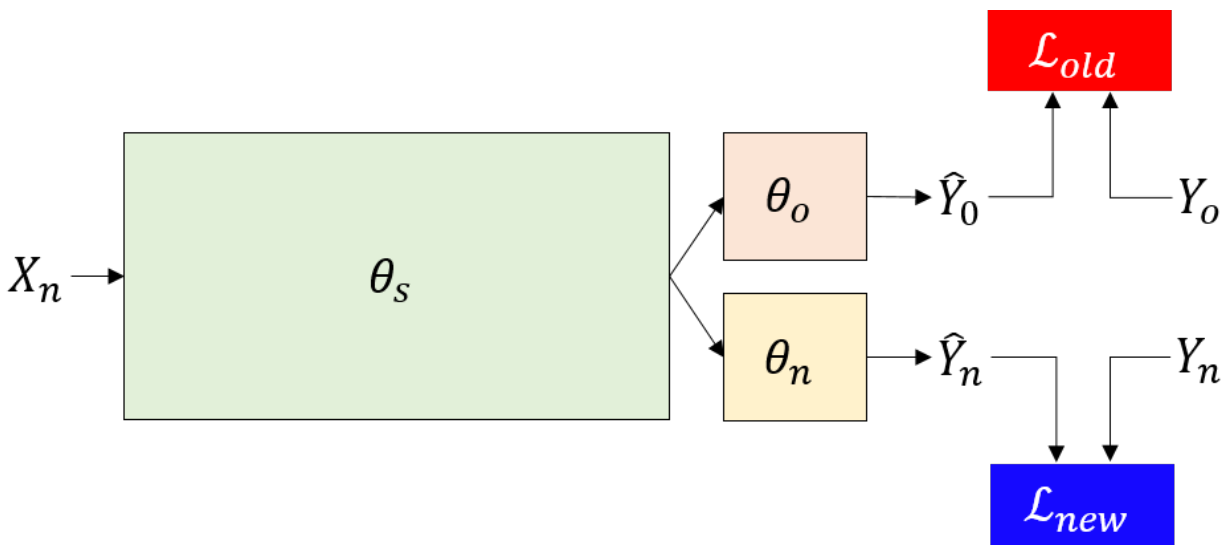


$\mathcal{L}_{new}$ (new task loss)

Goal: should encourage predicted class probabilities to be consistent with ground truth. An obvious pick for obtaining this behavior is the **multinomial logistic loss** (averaged over a training batch):

$$\mathcal{L}_{new}\left(\mathbf{y}_n, \hat{\mathbf{y}}_n\right) = -\mathbf{y}_n \cdot \log \hat{\mathbf{y}}_n$$

where $\mathbf{y}_n$ is the one-hot ground-truth class vector and $\hat{\mathbf{y}}_n$ is the softmax output of the network.



$\mathcal{L}_{old}$ (old task loss)

Goal: output probabilities $\hat{Y}_o$ should be close to the recorded outputs probabilities Y_0. They use the distillation loss of [9]:

$$\mathcal{L}_{new}\left(\mathbf{y}_o, \hat{\mathbf{y}}_o\right) = -\sum_{i=1}^{C} y_0'^{(i)} \log \hat{y}_0'^{(i)}$$

where C is the number of classes and $y_0'^{(i)}$ and $\hat{y}_0'^{(i)}$ are modified version of the softmax function:

$$y_0'^{(i)} = \frac{\left(y_o^{(i)}\right)^{1/T}}{\sum_j \left(y_o^{(i)}\right)^{1/T}} \qquad \hat{y}_0'^{(i)} = \frac{\left(\hat{y}_o^{(i)}\right)^{1/T}}{\sum_j \left(\hat{y}_o^{(i)}\right)^{1/T}}$$

where T is the "*temperature*" hyper-parameter of the distillation loss.

Algorithm

LEARNINGWITHOUTFORGETTING:

Start with:

θ_s: shared parameters

θ_o: task specific parameters for each old task

X_n, Y_n: training data and ground truth on the new task

Initialize:

$Y_o \leftarrow \text{CNN}(X_n, \theta_s, \theta_o)$ // *compute output of old tasks for new data*

$\theta_n \leftarrow \text{RANDINIT}(|\theta_n|)$ // *randomly initialize new parameters*

Train:

Define $\hat{Y}_o \equiv \text{CNN}(X_n, \hat{\theta}_s, \hat{\theta}_o)$ // *old task output*

Define $\hat{Y}_n \equiv \text{CNN}(X_n, \hat{\theta}_s, \hat{\theta}_n)$ // *new task output*

$$\theta_s^*,\ \theta_o^*,\ \theta_n^* \leftarrow \underset{\hat{\theta}_s, \hat{\theta}_o, \hat{\theta}_n}{\text{argmin}} \left(\lambda_o \mathcal{L}_{old}(Y_o, \hat{Y}_o) + \mathcal{L}_{new}(Y_n, \hat{Y}_n) + \mathcal{R}(\hat{\theta}_s, \hat{\theta}_o, \hat{\theta}_n) \right)$$

Implementation details I

Base networks:

- AlexNet
- VGG

Related work: A-LTM

Related work: Less Forgetting Learning

References I


Rich Caruana. "Multitask learning". In: *Machine learning* 28.1 (1997), pp. 41–75.


Catastrophic forgetting. http://standoutpublishing.com/g/catastrophic-forgetting.html. Accessed: 2019-03-19.


Oscar Day and Taghi M Khoshgoftaar. "A survey on heterogeneous transfer learning". In: *Journal of Big Data* 4.1 (2017), p. 29.


Jeff Donahue et al. "Decaf: A deep convolutional activation feature for generic visual recognition". In: *International conference on machine learning*. 2014, pp. 647–655.


Andre Esteva et al. "Dermatologist-level classification of skin cancer with deep neural networks". In: *Nature* 542.7639 (2017), p. 115.

References II


Xavier Glorot and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks". In: *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. 2010, pp. 249–256.


Varun Gulshan et al. "Development and validation of a deep learning algorithm for detection of diabetic retinopathy in retinal fundus photographs". In: *Jama* 316.22 (2016), pp. 2402–2410.


Steven Gutstein and Ethan Stump. "Reduction of catastrophic forgetting with transfer learning and ternary output codes". In: *2015 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2015, pp. 1–8.


Geoffrey Hinton, Oriol Vinyals and Jeff Dean. "Distilling the knowledge in a neural network". In: *arXiv preprint arXiv:1503.02531* (2015).

References III

Zhizhong Li and Derek Hoiem. "Learning without forgetting". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 40.12 (2018), pp. 2935–2947.

Pierre Sermanet et al. "Overfeat: Integrated recognition, localization and detection using convolutional networks". In: *arXiv preprint arXiv:1312.6229* (2013).