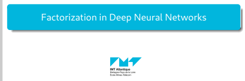
Factorization in Deep Neural Networks

IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

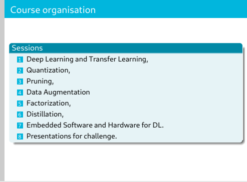
2022-03-01 Factorization in Deep Neural Networks



Course organisation

Sessions

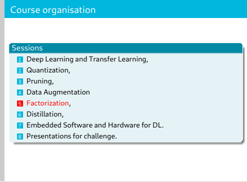
1. Deep Learning and Transfer Learning,
2. Quantization,
3. Pruning,
4. Data Augmentation
5. Factorization,
6. Distillation,
7. Embedded Software and Hardware for DL.
8. Presentations for challenge.


Course organisation
Sessions
1. Deep Learning and Transfer Learning,
2. Quantization,
3. Pruning,
4. Data Augmentation
5. Factorization,
6. Distillation,
7. Embedded Software and Hardware for DL.
8. Presentations for challenge.

Course organisation

Sessions

1. Deep Learning and Transfer Learning,
2. Quantization,
3. Pruning,
4. Data Augmentation
5. Factorization,
6. Distillation,
7. Embedded Software and Hardware for DL.
8. Presentations for challenge.


Course organisation

Sessions

1. Deep Learning and Transfer Learning,
2. Quantization,
3. Pruning,
4. Data Augmentation
5. Factorization,
6. Distillation,
7. Embedded Software and Hardware for DL.
8. Presentations for challenge.

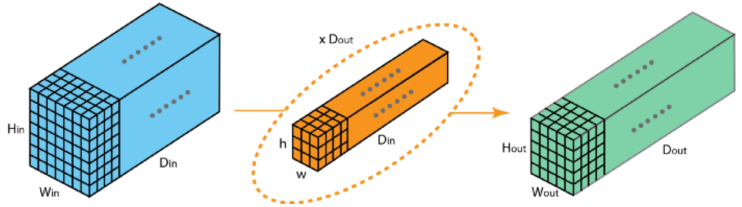
Factorization of Convolutional Networks

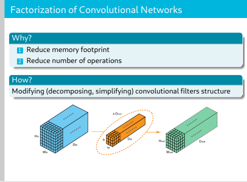
Why?

1. Reduce memory footprint
2. Reduce number of operations

How?

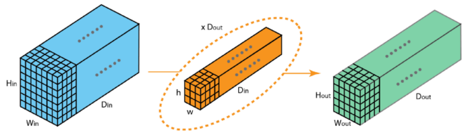
Modifying (decomposing, simplifying) convolutional filters structure





Terminology: kernel: small filter of dimension h x w, filter: ensemble of kernels channels-features map: Din, Dout

General principle



Complexity of 2D Convolutions

$N_{ops} = H_{out} \cdot W_{out} \cdot h \cdot w \cdot D_{in} \cdot D_{out}$
with kernel size (h, w), D_{in} the number of input feature maps, D_{out} the number of output feature maps of height H_{out} and width W_{out}.

To reduce the number of parameters, we can :

- Reduce the size of kernels
- Reduce the number of feature maps

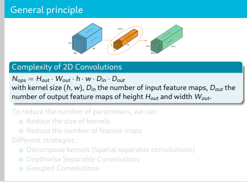
Different strategies :

- Decompose kernels (Spatial separable convolutions)
- Depthwise Separable Convolutions
- Grouped Convolutions

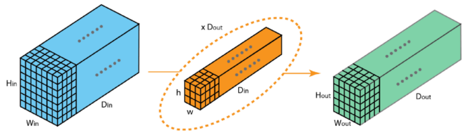
2022-03-01

Factorization in Deep Neural Networks

└General principle



General principle



Complexity of 2D Convolutions

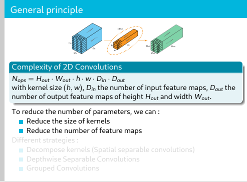
$N_{ops} = H_{out} \cdot W_{out} \cdot h \cdot w \cdot D_{in} \cdot D_{out}$
with kernel size (h, w), D_{in} the number of input feature maps, D_{out} the number of output feature maps of height H_{out} and width W_{out}.

To reduce the number of parameters, we can :

- Reduce the size of kernels
- Reduce the number of feature maps

Different strategies :

- Decompose kernels (Spatial separable convolutions)
- Depthwise Separable Convolutions
- Grouped Convolutions

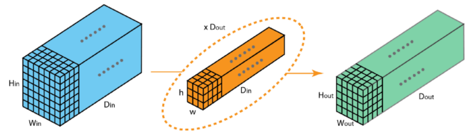

General principle

Complexity of 2D Convolutions
$N_{ops} = H_{out} \cdot W_{out} \cdot h \cdot w \cdot D_{in} \cdot D_{out}$
with kernel size (h, w), D_{in} the number of input feature maps, D_{out} the number of output feature maps of height H_{out} and width W_{out}.

To reduce the number of parameters, we can :
- Reduce the size of kernels
- Reduce the number of feature maps

Different strategies :
- Decompose kernels (Spatial separable convolutions)
- Depthwise Separable Convolutions
- Grouped Convolutions

General principle



Complexity of 2D Convolutions

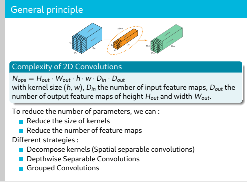
$N_{ops} = H_{out} \cdot W_{out} \cdot h \cdot w \cdot D_{in} \cdot D_{out}$
with kernel size (h, w), D_{in} the number of input feature maps, D_{out} the number of output feature maps of height H_{out} and width W_{out}.

To reduce the number of parameters, we can :

- Reduce the size of kernels
- Reduce the number of feature maps

Different strategies :

- Decompose kernels (Spatial separable convolutions)
- Depthwise Separable Convolutions
- Grouped Convolutions


General principle

Complexity of 2D Convolutions
$N_{ops} = H_{out} \cdot W_{out} \cdot h \cdot w \cdot D_{in} \cdot D_{out}$
with kernel size (h, w), D_{in} the number of input feature maps, D_{out} the number of output feature maps of height H_{out} and width W_{out}.

To reduce the number of parameters, we can :
- Reduce the size of kernels
- Reduce the number of feature maps

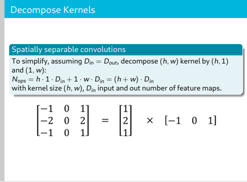
Different strategies :
- Decompose kernels (Spatial separable convolutions)
- Depthwise Separable Convolutions
- Grouped Convolutions

Decompose Kernels

Spatially separable convolutions

To simplify, assuming $D_{in} = D_{out}$, decompose (h, w) kernel by $(h, 1)$ and $(1, w)$:
$N_{ops} = h \cdot 1 \cdot D_{in} + 1 \cdot w \cdot D_{in} = (h + w) \cdot D_{in}$
with kernel size (h, w), D_{in} input and out number of feature maps.

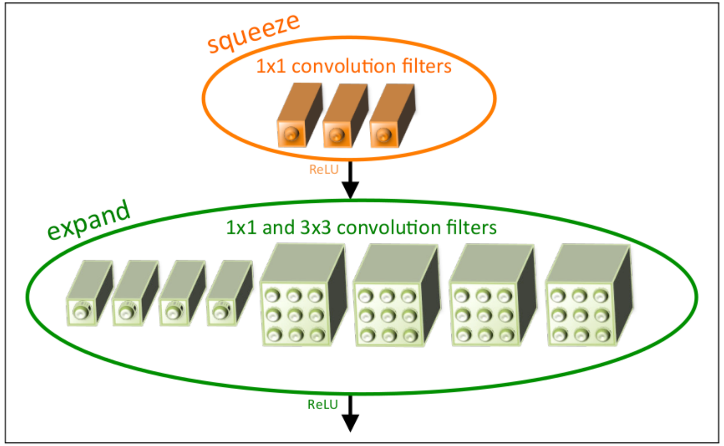
$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \times \begin{bmatrix} -1 & 0 & 1 \end{bmatrix}$$


Decompose Kernels
Spatially separable convolutions
To simplify, assuming $D_{in} = D_{out}$, decompose (h, w) kernel by $(h, 1)$ and $(1, w)$:
$N_{ops} = h \cdot 1 \cdot D_{in} + 1 \cdot w \cdot D_{in} = (h + w) \cdot D_{in}$
with kernel size (h, w), D_{in} input and out number of feature maps.
$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \times \begin{bmatrix} -1 & 0 & 1 \end{bmatrix}$

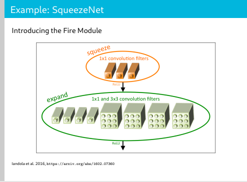
In convolution, the 3x3 kernel directly convolves with the image. In spatially separable convolution, the 3x1 kernel first convolves with the image. Then the 1x3 kernel is applied. This would require 6 instead of 9 parameters while doing the same operations. Although decomposing kernels (spatially separable convolutions) saves cost, it is rarely used in deep learning. One of the main reason is that not all kernels can be divided into two, smaller kernels

Example: SqueezeNet

Introducing the Fire Module



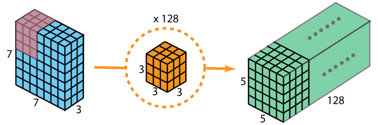
Iandola et al. 2016, `https://arxiv.org/abs/1602.07360`



We can however cite an example based on the same principle of reducing kernel size SqueezeNet. Two main strategies: replace 3x3 filters with 1x1 filters and reduce the channels in input to 3x3 filters -squeeze layers- then in the expand layers has a mix of 1x1 and 3x3 filters. They also use stride > 1 to downsample activation maps later towards the end of the architecture to maximize accuracy on a limited budget of parameters. Reduce x50 number of parameters with AlexNet level accuracy.

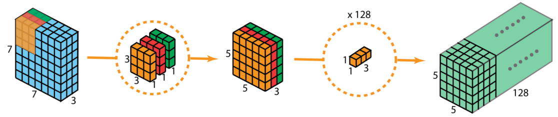
Depthwise separable convolutions

Instead of learning parameters that recombine all input feature maps to compute each output feature map:



$$N^N_{mul} = H_{out} \cdot W_{out} \cdot h \cdot w \cdot D_{in} \cdot D_{out} = 5 \cdot 5 \cdot 3 \cdot 3 \cdot 3 \cdot 128 = 86400$$

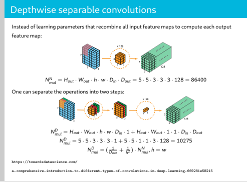
One can separate the operations into two steps:



$$N^D_{mul} = H_{out} \cdot W_{out} \cdot h \cdot w \cdot D_{in} \cdot 1 + H_{out} \cdot W_{out} \cdot 1 \cdot 1 \cdot D_{in} \cdot D_{out}$$
$$N^D_{mul} = 5 \cdot 5 \cdot 3 \cdot 3 \cdot 3 \cdot 1 + 5 \cdot 5 \cdot 1 \cdot 1 \cdot 3 \cdot 128 = 10275$$
$$N^D_{mul} = \left(\frac{1}{D_{out}} + \frac{1}{h^2}\right) \cdot N^N_{mul}, \; h = w$$

`https://towardsdatascience.com/`

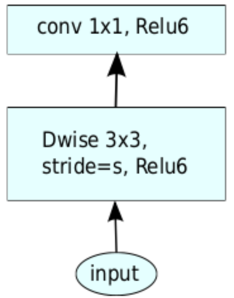
`a-comprehensive-introduction-to-different-types-of-convolutions-in-deep-learning-669281e58215`


Depthwise separable convolutions
Instead of learning parameters that recombine all input feature maps to compute each output feature map:
One can separate the operations into two steps:

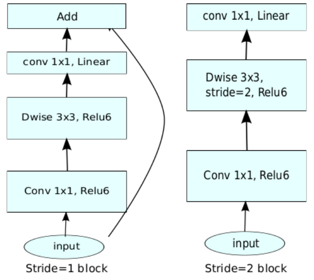
A standard convolution both filters and combines inputs into a new set of outputs in one step. The depthwise separable convolution splits this into two layers, depthwise convolutions and pointwise convolutions. We use depthwise convolutions to apply a single filter per each input channel (input depth). Pointwise convolution, a simple 1×1 convolution, is then used to create a linear combination of the output of the depthwise layer.Drawbacks: Depthwise separable convolutions reduces the number of parameters in the convolution. As such, for a small model, the model capacity may be decreased significantly if the 2D convolutions are replaced by depthwise separable convolutions. As a result, the model may become sub-optimal. However, if properly used, depthwise separable convolutions can give you the efficiency without dramatically damaging your model performance.

Example: MobileNet

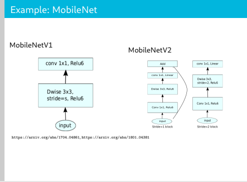
MobileNetV1



MobileNetV2



`https://arxiv.org/abs/1704.04861`, `https://arxiv.org/abs/1801.04381`



Depthwise separable convolutions were first introduced in the MobileNet paper (2017), that is a still a reference for embedded and mobile applications (small and low latency networks). For SqueezeNet and MobileNet factorized convolutions are used to drastically reduce parameters and operations wrt their counterpart.In MobileNet2, on the assumption that there is a lower dimension manifold in with the features space can be mapped, a linear bottleneck layer is added in the convolutional blocks (maps into lower dim, but without non linearity that destroy too much info). Also, shortcuts between bottleneck are added (same reason that for classical residual connections). MobileNet2 further decrease memory footprint wtr V1.

Example: MobileNet

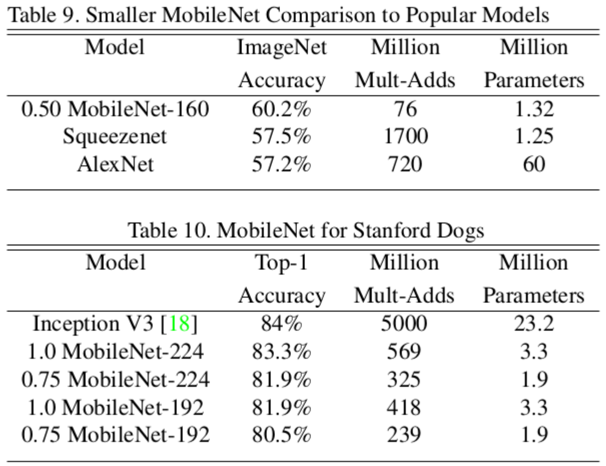


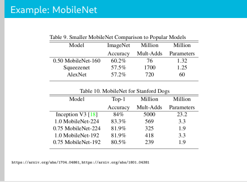
Table 9. Smaller MobileNet Comparison to Popular Models

Model	ImageNet Accuracy	Million Mult-Adds	Million Parameters
0.50 MobileNet-160	60.2%	76	1.32
Squeezenet	57.5%	1700	1.25
AlexNet	57.2%	720	60

Table 10. MobileNet for Stanford Dogs

Model	Top-1 Accuracy	Million Mult-Adds	Million Parameters
Inception V3 [18]	84%	5000	23.2
1.0 MobileNet-224	83.3%	569	3.3
0.75 MobileNet-224	81.9%	325	1.9
1.0 MobileNet-192	81.9%	418	3.3
0.75 MobileNet-192	80.5%	239	1.9

https://arxiv.org/abs/1704.04861, https://arxiv.org/abs/1801.04381

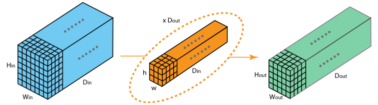


Example: MobileNet
Table 9. Smaller MobileNet Comparison to Popular Models
Table 10. MobileNet for Stanford Dogs
https://arxiv.org/abs/1704.04861, https://arxiv.org/abs/1801.04381

Depthwise separable convolutions were first introduced in the MobileNet paper (2017), that is a still a reference for embedded and mobile applications (small and low latency networks). For SqueezeNet and MobileNet factorized convolutions are used to drastically reduce parameters and operations wrt their counterpart.In MobileNet2, on the assumption that there is a lower dimension manifold in with the features space can be mapped, a linear bottleneck layer is added in the convolutional blocks (maps into lower dim, but without non linearity that destroy too much info). Also, shortcuts between bottleneck are added (same reason that for classical residual connections). MobileNet2 further decrease memory footprint wtr V1.

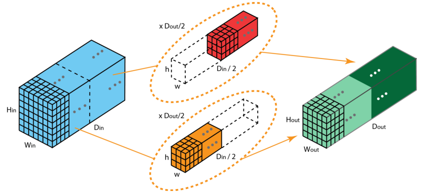
Grouped Convolutions

Instead of learning parameters that recombine all input feature maps to compute each output feature map:



$$N^N_{mul} = H_{out} \cdot W_{out} \cdot h \cdot w \cdot D_{in} \cdot D_{out}$$

One can divide the kernels into multiple groups:

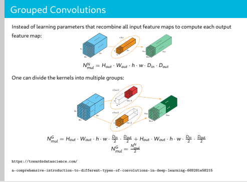


$$N^G_{mul} = H_{out} \cdot W_{out} \cdot h \cdot w \cdot \frac{D_{in}}{2} \cdot \frac{D_{out}}{2} + H_{out} \cdot W_{out} \cdot h \cdot w \cdot \frac{D_{in}}{2} \cdot \frac{D_{out}}{2}$$

$$N^G_{mul} = \frac{N^N_{mul}}{2}$$

https://towardsdatascience.com/

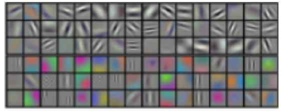
a-comprehensive-introduction-to-different-types-of-convolutions-in-deep-learning-669281e58215


Grouped Convolutions
Instead of learning parameters that recombine all input feature maps to compute each output feature map:
One can divide the kernels into multiple groups:
https://towardsdatascience.com/
a-comprehensive-introduction-to-different-types-of-convolutions-in-deep-learning-669281e58215

The first filter group (yellow) convolves with the first half of the input layer ([:, :, 0:Din/2]), while the second filter group (red) convolves with the second half of the input layer ([:, :, Din/2:Din]). As a result, each filter group creates Dout/2 channels that are stacked to obtain the output layer with Dout channels.

Examples

AlexNet filters



ResNeXt block

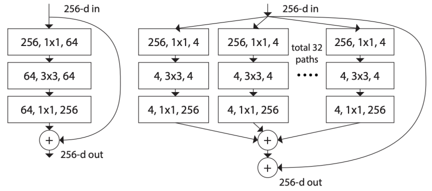
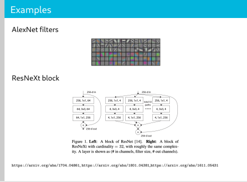


Figure 1. **Left**: A block of ResNet [14]. **Right**: A block of ResNeXt with cardinality = 32, with roughly the same complexity. A layer is shown as (# in channels, filter size, # out channels).

https://arxiv.org/abs/1704.04861, https://arxiv.org/abs/1801.04381,https://arxiv.org/abs/1611.05431


Examples
AlexNet filters
ResNeXt block

Grouped convolutions were first used in AlexNet to use 2 GPUs (parallelize efficiently). Authors showed that groups of filters were learning salient features (black-and-white and color filters). Grouped convolutions are used ResNeXt architecture from facebook: compared to a resnet block, an additional (to width and depth) dimension (cardinality) is added that exploits grouped convolutions and correspond to the number of groups. While for SqueezeNet and MobileNet grouped convolutions are used to reduce parameters and operation, in ResNeXt to boost accuracy (better than ResNet counterpart).

Combining Factorization with other Techniques: Attention based Pruning

Introducing Shift Attention Layer

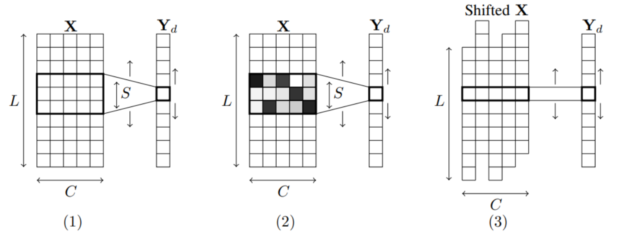
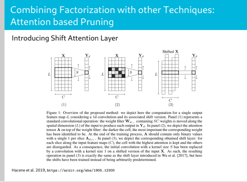


Figure 1: Overview of the proposed method: we depict here the computation for a single output feature map d, considering a 1d convolution and its associated shift version. Panel (1) represents a standard convolutional operation: the weight filter $\mathbf{W}_{d,\cdot,\cdot}$ containing SC weights is moved along the spatial dimension (L) of the input to produce each output in $\mathbf{Y}_d$. In panel (2), we depict the attention tensor $\mathbf{A}$ on top of the weight filter: the darker the cell, the most important the corresponding weight has been identified to be. At the end of the training process, $\mathbf{A}$ should contain only binary values with a single 1 per slice $\mathbf{A}_{d,c,\cdot}$. In panel (3), we depict the corresponding obtained shift layer: for each slice along the input feature maps (C), the cell with the highest attention is kept and the others are disregarded. As a consequence, the initial convolution with a kernel size S has been replaced by a convolution with a kernel size 1 on a shifted version of the input $\mathbf{X}$. As such, the resulting operation in panel (3) is exactly the same as the shift layer introduced in Wu et al. [2017], but here the shifts have been trained instead of being arbitrarily predetermined.

Hacene et al. 2019, `https://arxiv.org/abs/1905.12300`



A recent work from our lab that propose an alternative to classical convolutions using shift convolutions that are a combination of a shift layer + 1d convolution. This figure illustrates the principle for a 1d convolution example is considered. A shift layer is obtained when connections in W are pruned so that exactly one connection remains for each slice. Novelty of this work: method that learns which shifts to perform during the training process (attention based pruning during training, matrix of attention A is then binarized) instead of fixed shift. The Figure (7) is from the same paper compares SAL with some other compression methods. In our evaluation we compare SAL with Binary Connect (BC) to Resnet-20, MobileNetV2 and Squeezenet. SAL is also compared with another version denoted SAL2, in which we keep two weights per kernel instead of one. We use Resnet-20 with different number of weights and activations as baseline. SAL and SAL2 outperform all other methods

Combining Factorization with other Techniques: Attention based Pruning

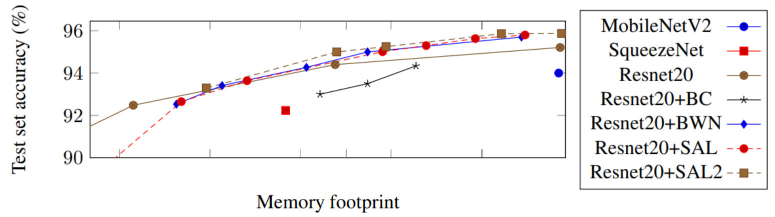
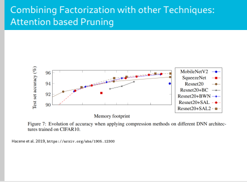


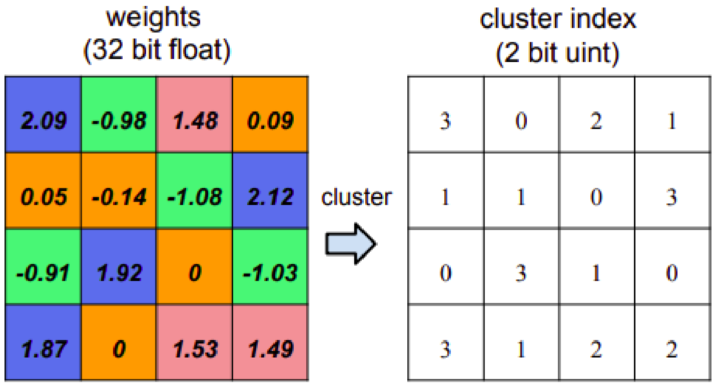
Figure 7: Evolution of accuracy when applying compression methods on different DNN architectures trained on CIFAR10.

Hacene et al. 2019, `https://arxiv.org/abs/1905.12300`


Combining Factorization with other Techniques: Attention based Pruning

A recent work from our lab that propose an alternative to classical convolutions using shift convolutions that are a combination of a shift layer + 1d convolution. This figure illustrates the principle for a 1d convolution example is considered. A shift layer is obtained when connections in W are pruned so that exactly one connection remains for each slice. Novelty of this work: method that learns which shifts to perform during the training process (attention based pruning during training, matrix of attention A is then binarized) instead of fixed shift. The Figure (7) is from the same paper compares SAL with some other compression methods. In our evaluation we compare SAL with Binary Connect (BC) to Resnet-20, MobileNetV2 and Squeezenet. SAL is also compared with another version denoted SAL2, in which we keep two weights per kernel instead of one. We use Resnet-20 with different number of weights and activations as baseline. SAL and SAL2 outperform all other methods

Combining Factorization with other Techniques: using clustering to share kernel weights

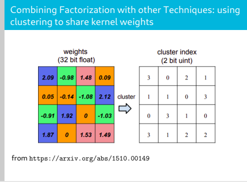


2.09	-0.98	1.48	0.09
0.05	-0.14	-1.08	2.12
-0.91	1.92	0	-1.03
1.87	0	1.53	1.49

3	0	2	1
1	1	0	3
0	3	1	0
3	1	2	2

from `https://arxiv.org/abs/1510.00149`

└Combining Factorization with other Techniques: using clustering to share kernel


Combining Factorization with other Techniques: using clustering to share kernel weights
from https://arxiv.org/abs/1510.00149

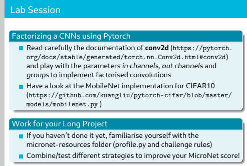
Lab Session

Factorizing a CNNs using Pytorch

- Read carefully the documentation of **conv2d** (`https://pytorch.org/docs/stable/generated/torch.nn.Conv2d.html#conv2d`) and play with the parameters *in channels*, *out channels* and *groups* to implement factorised convolutions
- Have a look at the MobileNet implementation for CIFAR10 (`https://github.com/kuangliu/pytorch-cifar/blob/master/models/mobilenet.py`)

Work for your Long Project

- If you haven't done it yet, familiarise yourself with the micronet-resources folder (profile.py and challenge rules)
- Combine/test different strategies to improve your MicroNet score!


Lab Session

Factorizing a CNNs using Pytorch
- Read carefully the documentation of **conv2d** (`https://pytorch.org/docs/stable/generated/torch.nn.Conv2d.html#conv2d`) and play with the parameters *in channels*, *out channels* and *groups* to implement factorised convolutions
- Have a look at the MobileNet implementation for CIFAR10 (`https://github.com/kuangliu/pytorch-cifar/blob/master/models/mobilenet.py`)

Work for your Long Project
- If you haven't done it yet, familiarise yourself with the micronet-resources folder (profile.py and challenge rules)
- Combine/test different strategies to improve your MicroNet score!

Factorization in Deep Neural Networks

1 Overview of unsupervised learning
- Clustering
- Decomposition using Sparse Dictionary Learning

2022-03-01

1 Overview of unsupervised learning
- Clustering
- Decomposition using Sparse Dictionary Learning

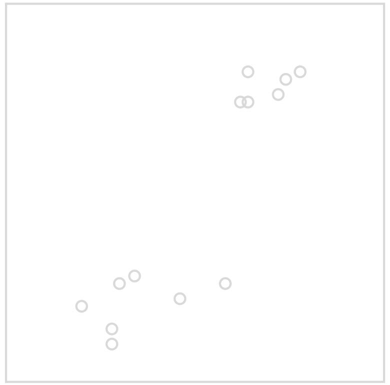
Unsupervised learning

Goal

Discover patterns/structure in X,

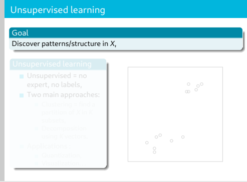
Unsupervised learning

- Unsupervised = no expert, no labels,
- Two main approaches:
  - Clustering = find a partition of X in K subsets,
  - Decomposition using K vectors.
- Applications :
  - Quantization,
  - Visualization...



2022-03-01

Factorization in Deep Neural Networks
└─Overview of unsupervised learning
└─Unsupervised learning


Unsupervised learning
Goal
Discover patterns/structure in X,
Unsupervised learning
Unsupervised = no expert, no labels,
Two main approaches:

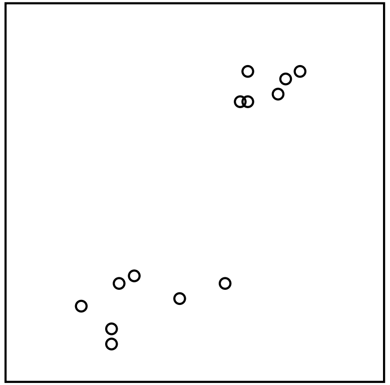
Unsupervised learning

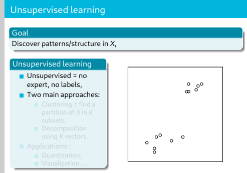
Goal

Discover patterns/structure in X,

Unsupervised learning

- Unsupervised = no expert, no labels,
- Two main approaches:
  - Clustering = find a partition of X in K subsets,
  - Decomposition using K vectors.
- Applications :
  - Quantization,
  - Visualization...





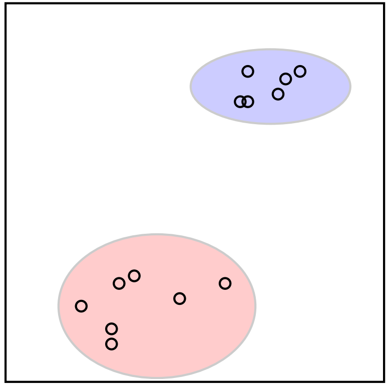
Unsupervised learning

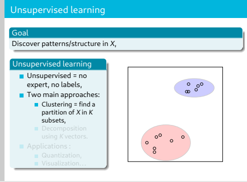
Goal

Discover patterns/structure in X,

Unsupervised learning

- Unsupervised = no expert, no labels,
- Two main approaches:
  - Clustering = find a partition of X in K subsets,
  - Decomposition using K vectors.
- Applications :
  - Quantization,
  - Visualization...





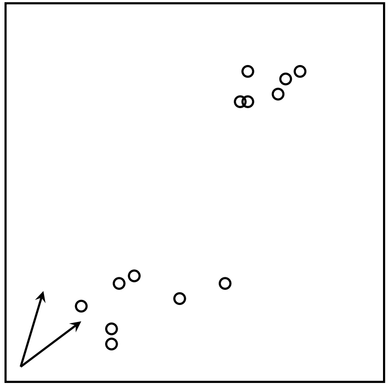
Unsupervised learning

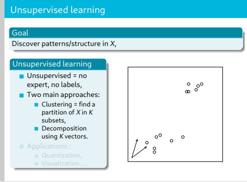
Goal

Discover patterns/structure in X,

Unsupervised learning

- Unsupervised = no expert, no labels,
- Two main approaches:
  - Clustering = find a partition of X in K subsets,
  - Decomposition using K vectors.
- Applications :
  - Quantization,
  - Visualization…





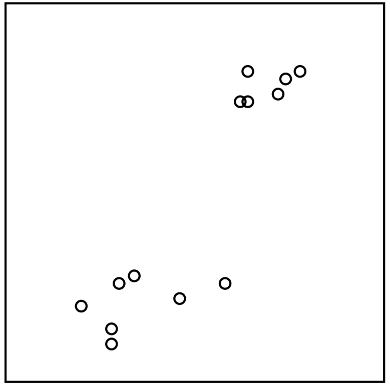
Unsupervised learning

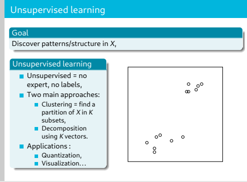
Goal

Discover patterns/structure in X,

Unsupervised learning

- Unsupervised = no expert, no labels,
- Two main approaches:
  - Clustering = find a partition of X in K subsets,
  - Decomposition using K vectors.
- Applications :
  - Quantization,
  - Visualization…





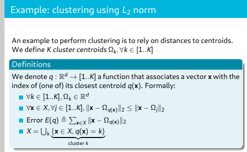
Example: clustering using L_2 norm

An example to perform clustering is to rely on distances to centroids. We define K *cluster centroids* $\Omega_k, \forall k \in [1..K]$

Definitions

We denote $q : \mathbb{R}^d \to [1..K]$ a function that associates a vector $\mathbf{x}$ with the index of (one of) its closest centroid $q(\mathbf{x})$. Formally:

- $\forall k \in [1..K], \Omega_k \in \mathbb{R}^d$
- $\forall \mathbf{x} \in X, \forall j \in [1..K], \|\mathbf{x} - \Omega_{q(\mathbf{x})}\|_2 \leq \|\mathbf{x} - \Omega_j\|_2$
- Error $E(q) \triangleq \sum_{\mathbf{x} \in X} \|\mathbf{x} - \Omega_{q(\mathbf{x})}\|_2$
- $X = \bigcup_k \underbrace{\{\mathbf{x} \in X, q(\mathbf{x}) = k\}}_{\text{cluster } k}$


Example: clustering using L_2 norm

An example to perform clustering is to rely on distances to centroids. We define K *cluster centroids* $\Omega_k, \forall k \in [1..K]$

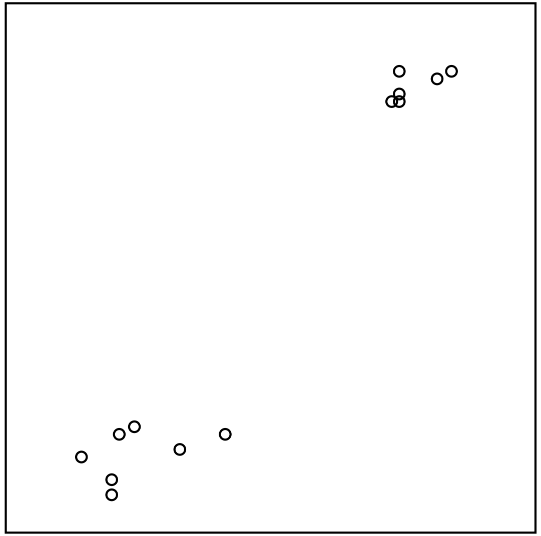
Definitions

We denote $q : \mathbb{R}^d \to [1..K]$ a function that associates a vector $\mathbf{x}$ with the index of (one of) its closest centroid $q(\mathbf{x})$. Formally:

- $\forall k \in [1..K], \Omega_k \in \mathbb{R}^d$
- $\forall \mathbf{x} \in X, \forall j \in [1..K], \|\mathbf{x} - \Omega_{q(\mathbf{x})}\|_2 \leq \|\mathbf{x} - \Omega_j\|_2$
- Error $E(q) \triangleq \sum_{\mathbf{x} \in X} \|\mathbf{x} - \Omega_{q(\mathbf{x})}\|_2$
- $X = \bigcup_k \underbrace{\{\mathbf{x} \in X, q(\mathbf{x}) = k\}}_{\text{cluster } k}$

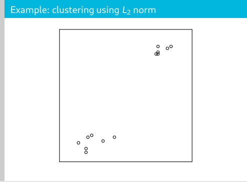
Here, we provide a formal definition of clustering using centroids. Note that there are other ways to define clustering, using regions, using density of spaces, using probabilities, etc...
The second point is the way to define the closest centroid.
The important point to note here is the definition of the error, which can be defined as the sum of all distances between points and their closest cluster centroid.

Example: clustering using L_2 norm

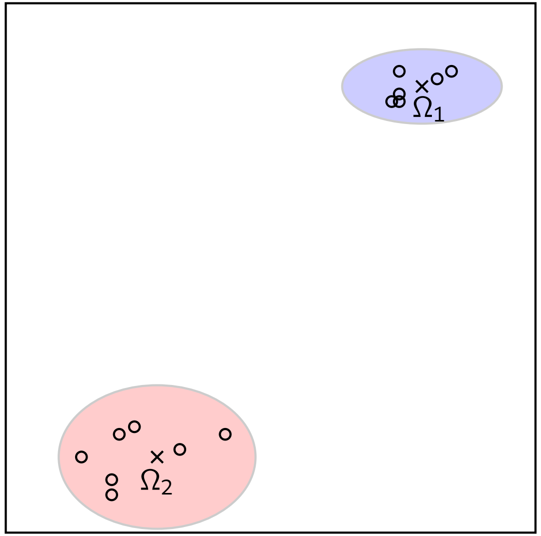


2022-03-01

Factorization in Deep Neural Networks
└Overview of unsupervised learning
└Clustering
└Example: clustering using L_2 norm

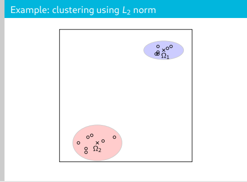


Example: clustering using L_2 norm



2022-03-01

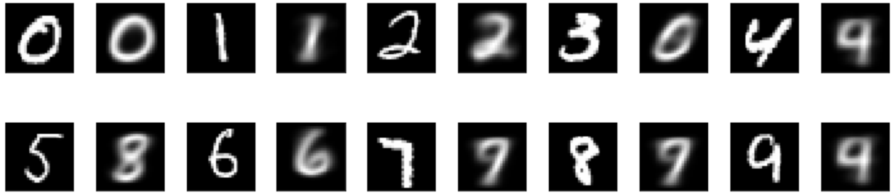
Factorization in Deep Neural Networks
└Overview of unsupervised learning
└Clustering
└Example: clustering using L_2 norm

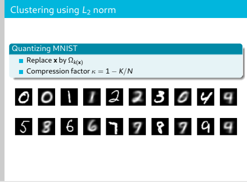


Clustering using L_2 norm

Quantizing MNIST

- Replace $\mathbf{x}$ by $\Omega_{k(\mathbf{x})}$
- Compression factor $\kappa = 1 - K/N$

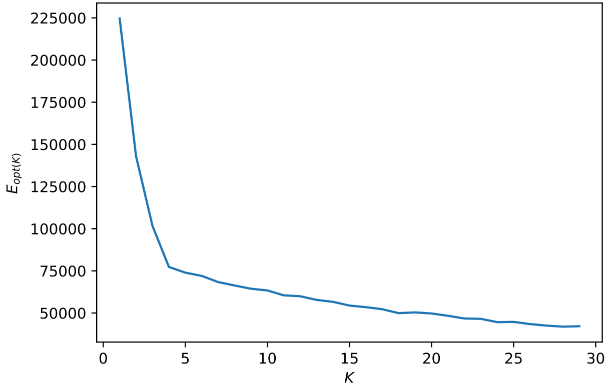


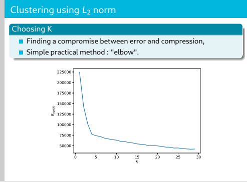


Clustering using L_2 norm

Choosing K

- Finding a compromise between error and compression,
- Simple practical method : "elbow".



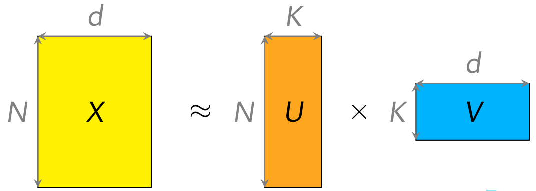


Sparse Dictionary Learning

Definitions

Dictionary learning solves the following matrix factorization problem:

- The set X is considered as a matrix $X \in \mathcal{M}_{N\times d}(\mathbb{R})$,
- We consider decompositions using a dictionary $V \in \mathcal{M}_{K\times d}(\mathbb{R})$ and a code $U \in \mathcal{M}_{N\times k}(\mathbb{R})$, with the lines of V being with norm 1,
- Error $E(U,V) \triangleq \|X - UV\|_2 + \alpha\|U\|_1$
- Training: find U^*, V^* that minimizes $E(U^*, V^*)$
- α is a sparsity control parameter that enforces codes with soft (ℓ_1) sparsity



2022-03-01

Factorization in Deep Neural Networks
└Overview of unsupervised learning
└Decomposition using Sparse Dictionary Learning
└Sparse Dictionary Learning