Lecture 2 - Deep Neural Networks Compression

Efficacy and efficiency evaluation of machine learning models
Ph.D. Course

Marco Frasca

AnacletoLab, Dipartimento di Informatica
Università degli Studi di Milano

12.06.24

Outline

1. The Need to Compress DNNs

2 Compressing for Pre-Trained Models

- 2.1 Structure-Preserving Compression Strategies
- `sHAM`
- 2.2 Structure-Altering Compression Strategies

3. Train Compressible Models

4. Exam Paper List

Outline

1. The Need to Compress DNNs

2 Compressing for Pre-Trained Models
- 2.1 Structure-Preserving Compression Strategies
- sHAM
- 2.2 Structure-Altering Compression Strategies

3. Train Compressible Models

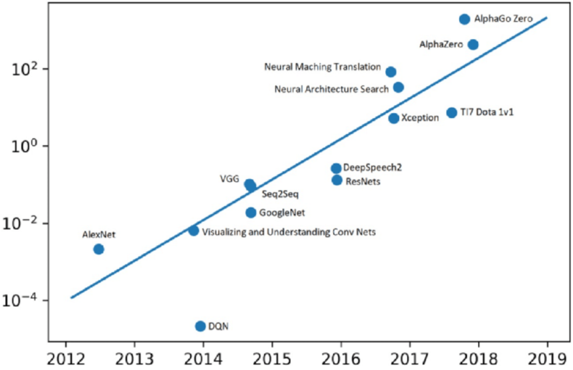
4. Exam Paper List

The Need to Compress DNNs

- The increasing number of layers and layer sizes in DNNs comes at the cost of severely increased computational complexity (and energy demand)

The Need to Compress DNNs

- The increasing number of layers and layer sizes in DNNs comes at the cost of severely increased computational complexity (and energy demand)
- Recalling the need for power-hungry hardware (petaflops/day, in 1e04)



Amodei, D. and Hernandez, D. Ai and compute. https://openai.com/index/ai-and-compute/ 2018.

Learning Issues with Too Large Models

- Overparameterization
  - Existing DNNs models are often overparameterized [Allen-Zhu et *al*. 2019]

Learning Issues with Too Large Models

- Overparameterization
  - Existing DNNs models are often overparameterized [Allen-Zhu et *al*. 2019]
  - Resulting in a considerable memory footprint: VGG16 demands around 500 MB, T5 (variant 11B) requires about 20 GB

Learning Issues with Too Large Models

- Overparameterization
  - Existing DNNs models are often overparameterized [Allen-Zhu et *al*. 2019]
  - Resulting in a considerable memory footprint: VGG16 demands around 500 MB, T5 (variant 11B) requires about 20 GB

- Compress DNNs means adopting some lossy and/or lossless transformations to reduce the memory footprint, and/or the number of parameters, and/or the overall inference computation

Learning Issues with Too Large Models

- Overparameterization
  - Existing DNNs models are often overparameterized [Allen-Zhu et *al*. 2019]
  - Resulting in a considerable memory footprint: VGG16 demands around 500 MB, T5 (variant 11B) requires about 20 GB

- Compress DNNs means adopting some lossy and/or lossless transformations to reduce the memory footprint, and/or the number of parameters, and/or the overall inference computation

- In addition to the reduction of resource demand, compressing even in necessary in some contexts
  - e.g. when resources (RAM, battery, etc.) are limited (e.g., IoT devices)
  - low-power mobile/embedded systems

Allen-Zhu, Z.et *al*. Learning and generalization in overparameterized neural networks, going beyond two layers. Advances in Neural Information Processing Systems, 32, Curran Associates Inc, 2019.

Outline

1. The Need to Compress DNNs

2 Compressing for Pre-Trained Models

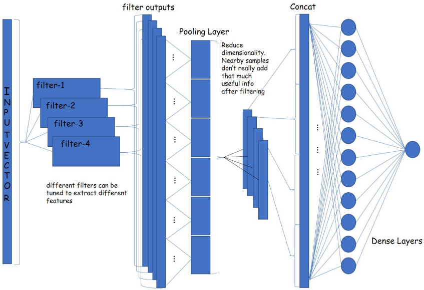
- 2.1 Structure-Preserving Compression Strategies
- `sHAM`
- 2.2 Structure-Altering Compression Strategies

3. Train Compressible Models

4. Exam Paper List

Compressing for Pre-Trained Models

Typical CNN



An illustration of Convolutional Neural Networks

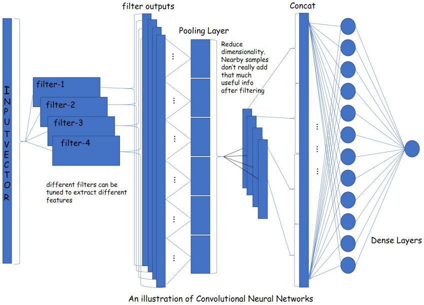
Source: https://towardsdatascience.com/acomparisonof-dnncnnandlstmusingtfkeras2191f8c77bbe

DNN compression

- Reduce the number of parameters

Compressing for Pre-Trained Models

Typical CNN



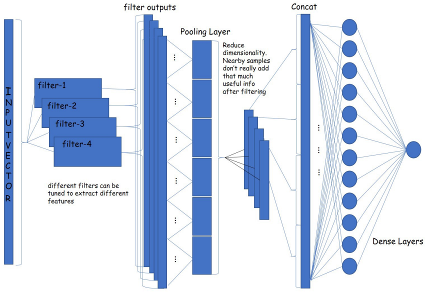
Source: https://towardsdatascience.com/acomparisonof-dnncnnandlstmusingtfkeras2191f8c77bbe

DNN compression

- Reduce the number of parameters
- This usually induces a drop in the energy demand

Compressing for Pre-Trained Models

Typical CNN



An illustration of Convolutional Neural Networks

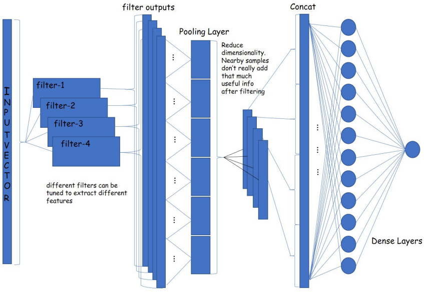
Source: https://towardsdatascience.com/acomparisonof-dnncnnandlstmusingtfkeras2191f8c77bbe

DNN compression

- Reduce the number of parameters
- This usually induces a drop in the energy demand
- It can be done by preserving the topology or not

Compressing for Pre-Trained Models

Typical CNN



An illustration of Convolutional Neural Networks

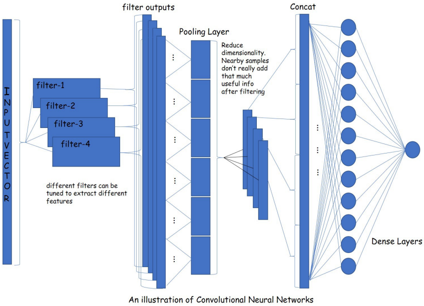
Source: https://towardsdatascience.com/acomparisonof-dnncnnandlstmusingtfkeras2191f8c77bbe

DNN compression

- Reduce the number of parameters
- This usually induces a drop in the energy demand
- It can be done by preserving the topology or not
  - Preserving the topology means do not alter the input and the output size of layers

Compressing for Pre-Trained Models

Typical CNN



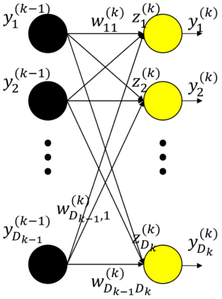
Source: https://towardsdatascience.com/acomparisonof-dnncnnandlstmusingtfkeras2191f8c77bbe

DNN compression

- Reduce the number of parameters
- This usually induces a drop in the energy demand
- It can be done by preserving the topology or not
  - Preserving the topology means do not alter the input and the output size of layers
- In most architectures proposed, fully connected (dense) layers often cover the large majority of parameters (weights and biases)

Computation in Dense layers

The computation in dense layers is performed via vector-matrix products:

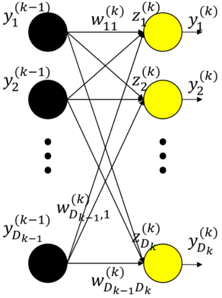


$$\mathbf{z}^{(k)} = \begin{bmatrix} z_1^{(k)} \\ z_2^{(k)} \\ \vdots \\ z_{D_k}^{(k)} \end{bmatrix} \qquad \mathbf{y}^{(k)} = \begin{bmatrix} y_1^{(k)} \\ y_2^{(k)} \\ \vdots \\ y_{D_k}^{(k)} \end{bmatrix}$$

$$\mathbf{W}^{(k)} = \begin{bmatrix} w_{11}^{(k)} & w_{21}^{(k)} & \vdots & w_{D_{k-1}1}^{(k)} \\ w_{12}^{(k)} & w_{22}^{(k)} & \vdots & w_{D_{k-1}2}^{(k)} \\ \cdots & \cdots & \ddots & \vdots \\ w_{1D_k}^{(k)} & w_{2D_k}^{(k)} & \cdots & w_{D_{k-1}D_k}^{(k)} \end{bmatrix} \qquad \mathbf{b}^{(k)} = \begin{bmatrix} b_1^{(k)} \\ b_2^{(k)} \\ \vdots \\ b_{D_k}^{(k)} \end{bmatrix}$$

Computation in Dense layers

The computation in dense layers is performed via vector-matrix products:



$$\mathbf{z}^{(k)} = \begin{bmatrix} z_1^{(k)} \\ z_2^{(k)} \\ \vdots \\ z_{D_k}^{(k)} \end{bmatrix} \qquad \mathbf{y}^{(k)} = \begin{bmatrix} y_1^{(k)} \\ y_2^{(k)} \\ \vdots \\ y_{D_k}^{(k)} \end{bmatrix}$$

$$\mathbf{W}^{(k)} = \begin{bmatrix} w_{11}^{(k)} & w_{21}^{(k)} & \vdots & w_{D_{k-1}1}^{(k)} \\ w_{12}^{(k)} & w_{22}^{(k)} & \vdots & w_{D_{k-1}2}^{(k)} \\ \cdots & \cdots & \ddots & \vdots \\ w_{1D_k}^{(k)} & w_{2D_k}^{(k)} & \cdots & w_{D_{k-1}D_k}^{(k)} \end{bmatrix} \qquad \mathbf{b}^{(k)} = \begin{bmatrix} b_1^{(k)} \\ b_2^{(k)} \\ \vdots \\ b_{D_k}^{(k)} \end{bmatrix}$$

For k-th layer, we have $\boldsymbol{y}^{(k)} = f_k(\boldsymbol{W}^k \cdot \boldsymbol{y}^{(k-1)})$ (casting the bias $\boldsymbol{b}^{(k)}$ into an additional input)

where

- $W^{(k)} \in \mathbb{R}^{D_k \times D_{k-1}}$ is the layer weight matrix
- f_k is the neuron activation function for this layer
- $\boldsymbol{y}^{(k)}$ is the output vector of k-th layer

Computation in Dense layers

Therefore, most computation in dense layers is due to the vector-matrix product: $\boldsymbol{W} \cdot \boldsymbol{y}$

Computation in Dense layers

Therefore, most computation in dense layers is due to the vector-matrix product: $\boldsymbol{W} \cdot \boldsymbol{y}$

- Most attempts tried to apply lossy transformations to $\boldsymbol{W}$ and then leverage the vast literature about lossless techniques to sparse vector-matrix multiplication

Computation in Dense layers

Therefore, most computation in dense layers is due to the vector-matrix product: $\boldsymbol{W} \cdot \boldsymbol{y}$

- Most attempts tried to apply lossy transformations to $\boldsymbol{W}$ and then leverage the vast literature about lossless techniques to sparse vector-matrix multiplication
- For example, connection pruning and/or quantization of $\boldsymbol{W}$

Deep Compression Strategies

Main techniques to reduce size and resource need of neural networks limiting or avoiding decays in accuracy

Deep Compression Strategies

Main techniques to reduce size and resource need of neural networks limiting or avoiding decays in accuracy

- Reducing the Number of Weights/Parameters
  1. Connections/Neuron Pruning

Deep Compression Strategies

Main techniques to reduce size and resource need of neural networks limiting or avoiding decays in accuracy

- Reducing the Number of Weights/Parameters
  1. Connections/Neuron Pruning
  2. Low-Rank Factorization (LRF)

Deep Compression Strategies

Main techniques to reduce size and resource need of neural networks limiting or avoiding decays in accuracy

- Reducing the Number of Weights/Parameters
  1. Connections/Neuron Pruning
  2. Low-Rank Factorization (LRF)
  3. Filter/Channel Pruning (for CNNs)

Deep Compression Strategies

Main techniques to reduce size and resource need of neural networks limiting or avoiding decays in accuracy

- Reducing the Number of Weights/Parameters
  1. Connections/Neuron Pruning
  2. Low-Rank Factorization (LRF)
  3. Filter/Channel Pruning (for CNNs)
- Quantization to Reduce the Bits per Weight
  1. Sharing weights - Layer-wise

Deep Compression Strategies

Main techniques to reduce size and resource need of neural networks limiting or avoiding decays in accuracy

- Reducing the Number of Weights/Parameters
  1. Connections/Neuron Pruning
  2. Low-Rank Factorization (LRF)
  3. Filter/Channel Pruning (for CNNs)
- Quantization to Reduce the Bits per Weight
  1. Sharing weights - Layer-wise
  2. Sharing weights Globally

Deep Compression Strategies

Main techniques to reduce size and resource need of neural networks limiting or avoiding decays in accuracy

- Reducing the Number of Weights/Parameters
  1. Connections/Neuron Pruning
  2. Low-Rank Factorization (LRF)
  3. Filter/Channel Pruning (for CNNs)
- Quantization to Reduce the Bits per Weight
  1. Sharing weights - Layer-wise
  2. Sharing weights Globally
  3. Extreme quantization: binarization

Deep Compression Strategies

Main techniques to reduce size and resource need of neural networks limiting or avoiding decays in accuracy

- Reducing the Number of Weights/Parameters
  1. Connections/Neuron Pruning
  2. Low-Rank Factorization (LRF)
  3. Filter/Channel Pruning (for CNNs)
- Quantization to Reduce the Bits per Weight
  1. Sharing weights - Layer-wise
  2. Sharing weights Globally
  3. Extreme quantization: binarization
- Knowledge distillation

Deep Compression Strategies

Main techniques to reduce size and resource need of neural networks limiting or avoiding decays in accuracy

- Reducing the Number of Weights/Parameters
  1. Connections/Neuron Pruning
  2. Low-Rank Factorization (LRF)
  3. Filter/Channel Pruning (for CNNs)
- Quantization to Reduce the Bits per Weight
  1. Sharing weights - Layer-wise
  2. Sharing weights Globally
  3. Extreme quantization: binarization
- Knowledge distillation
- Optimizing Compression Criteria during Network Training

Outline

1. The Need to Compress DNNs

2 Compressing for Pre-Trained Models
- 2.1 Structure-Preserving Compression Strategies
- sHAM
- 2.2 Structure-Altering Compression Strategies

3. Train Compressible Models

4. Exam Paper List

Weight Pruning

- Given a trained network
- Classical approach consisting in removing *weak* connections

Weight Pruning

- Given a trained network
- Classical approach consisting in removing *weak* connections
- Denoted by $\boldsymbol{W} \in \mathbb{R}^{n \times m}$ the connection matrix of a given dense layer,
  1. Define a threshold $\tau \in (0, \max(|\boldsymbol{W}|)]$

Weight Pruning

- Given a trained network
- Classical approach consisting in removing *weak* connections
- Denoted by $\boldsymbol{W} \in \mathbb{R}^{n \times m}$ the connection matrix of a given dense layer,
  1. Define a threshold $\tau \in (0, \max(|\boldsymbol{W}|)]$
  2. Cut away all connections having absolute weight less or equal to τ:

$$\overline{W}_{ij} = \begin{cases} W_{ij} & \text{if } |W_{ij}| > \tau \\ 0 & \text{otherwise} \end{cases}$$

Weight Pruning

- Given a trained network
- Classical approach consisting in removing *weak* connections
- Denoted by $\boldsymbol{W} \in \mathbb{R}^{n \times m}$ the connection matrix of a given dense layer,
  1. Define a threshold $\tau \in (0, \max(|\boldsymbol{W}|)]$
  2. Cut away all connections having absolute weight less or equal to τ:

$$\overline{W}_{ij} = \begin{cases} W_{ij} & \text{if } |W_{ij}| > \tau \\ 0 & \text{otherwise} \end{cases}$$

  3. Fine-tune the remaining connections (that is, retrain the network)

Weight sharing (WS)- Quantization

- Based on the idea to quantize connection weights

Weight sharing (WS)- Quantization

- Based on the idea to quantize connection weights
- Only k distinct connection weights are allowed

Weight sharing (WS)- Quantization

- Based on the idea to quantize connection weights
- Only k distinct connection weights are allowed
- The trained weight matrix $\boldsymbol{W}$ undergoes to a clustering procedure to detect the initial weight clusters $C_1, \ldots, C_k$, whose centroids are denoted by $c_1, \ldots, c_k$
- Then, all weights in cluster C_j are set to c_j (c_j is their representative)

Weight sharing (WS)- Quantization

- Based on the idea to quantize connection weights
- Only k distinct connection weights are allowed
- The trained weight matrix $\boldsymbol{W}$ undergoes to a clustering procedure to detect the initial weight clusters $C_1, \ldots, C_k$, whose centroids are denoted by $c_1, \ldots, c_k$
- Then, all weights in cluster C_j are set to c_j (c_j is their representative)
- In general this means to have the possibility to use less bits to store each weight

Weight sharing (WS)- Quantization

- Based on the idea to quantize connection weights
- Only k distinct connection weights are allowed
- The trained weight matrix $\boldsymbol{W}$ undergoes to a clustering procedure to detect the initial weight clusters $C_1, \ldots, C_k$, whose centroids are denoted by $c_1, \ldots, c_k$
- Then, all weights in cluster C_j are set to c_j (c_j is their representative)
- In general this means to have the possibility to use less bits to store each weight
- Binarization: extreme case where only two distinct weights are used (typically 0 and 1) [Hubara et *al.* 2016]

– Gamma, E. Helm, R. Johnson, R. and Vlissides, JM. Design Patterns: Elements of Reusable Object-Oriented Software. Addison Wesley, 1994

– I. Hubara, et al., Binarized neural networks. Advances in Neural Inf. Process. Syst. 2016.

iMap - Index Map [Han et *al.* 2015]

Losseless format to exploit a quantized matrix $\overline{\boldsymbol{W}} \in \mathbb{R}^{n \times m}$

iMap - Index Map [Han et *al.* 2015]

Losseless format to exploit a quantized matrix $\overline{\boldsymbol{W}} \in \mathbb{R}^{n \times m}$

- Assuming the weight matrix $\boldsymbol{W}$ has only k distinct weights $c_1, c_2, \ldots, c_k$:

iMap - Index Map [Han et *al.* 2015]

Losseless format to exploit a quantized matrix $\overline{\boldsymbol{W}} \in \mathbb{R}^{n \times m}$

- Assuming the weight matrix $\boldsymbol{W}$ has only k distinct weights $c_1, c_2, \ldots, c_k$:
  1. Create a vector that contains all the distinct weights $c_1, c_2, \ldots, c_k$, denoted by $\boldsymbol{C}$

iMap - Index Map [Han et *al.* 2015]

Losseless format to exploit a quantized matrix $\overline{\boldsymbol{W}} \in \mathbb{R}^{n \times m}$

- Assuming the weight matrix $\boldsymbol{W}$ has only k distinct weights $c_1, c_2, \ldots, c_k$:

    1. Create a vector that contains all the distinct weights $c_1, c_2, \ldots, c_k$, denoted by $\boldsymbol{C}$
    2. Create a matrix $\boldsymbol{M}$ with the same shape of $\overline{\boldsymbol{W}}$, where each element m_{ij} contains an index related to $\boldsymbol{C}$

$$\overset{\bar{\mathbf{W}}}{\begin{bmatrix} 2 & 2 & 1 \\ 6 & 4 & 1 \\ 6 & 2 & 4 \end{bmatrix}} \Rightarrow \overset{\mathbf{C}}{\begin{bmatrix} 2 \\ 6 \\ 4 \\ 1 \end{bmatrix}} \quad \overset{\mathbf{M}}{\begin{bmatrix} 0 & 0 & 3 \\ 1 & 2 & 3 \\ 1 & 0 & 2 \end{bmatrix}}$$

iMap - Index Map [Han et *al.* 2015]

Losseless format to exploit a quantized matrix $\overline{\boldsymbol{W}} \in \mathbb{R}^{n \times m}$

- Assuming the weight matrix $\boldsymbol{W}$ has only k distinct weights $c_1, c_2, \ldots, c_k$:

  1. Create a vector that contains all the distinct weights $c_1, c_2, \ldots, c_k$, denoted by $\boldsymbol{C}$
  2. Create a matrix $\boldsymbol{M}$ with the same shape of $\overline{\boldsymbol{W}}$, where each element m_{ij} contains an index related to $\boldsymbol{C}$

$$\overset{\bar{\mathbf{W}}}{\begin{bmatrix} 2 & 2 & 1 \\ 6 & 4 & 1 \\ 6 & 2 & 4 \end{bmatrix}} \Rightarrow \overset{\mathbf{C}}{\begin{bmatrix} 2 \\ 6 \\ 4 \\ 1 \end{bmatrix}} \quad \overset{\mathbf{M}}{\begin{bmatrix} 0 & 0 & 3 \\ 1 & 2 & 3 \\ 1 & 0 & 2 \end{bmatrix}}$$

- Denoted by b and $\bar{b}$ the number of bits used to store one entry of $\overline{\boldsymbol{W}}$ and $\boldsymbol{M}$, respectively: the compression ratio is $\frac{bnm}{\bar{b}nm+kb}$.

S. Han, H. Mao, W.J. Dally. Deep compression: Compressing deep neural network with pruning, trained quantization and Huffman coding. ICLR 2015.

Weight sharing: Main approaches

- **CWS** (*Clustering Weight Sharing*) exploits k-Means and maps the weights to the respective centroid
- **PWS** (*Probabilistic Weight Sharing*) is a probabilistic approach that maps the weights to representative values using a probability distribution on representatives based on their 'distance' to the weight to be quantized
- **UQ** (*Uniform Quantization*) selects representative weights uniformly in a variable size interval centered around the weight to be quantized
- **ECSQ** (*Entropy Constrained Scalar Quantization*) learns C_i and c_i by optimizing jointly distortion and entropy of the resulting distribution of representative weights

For a survey, see Marinò, G.C. et *al*. Deep neural networks compression: A comparative survey and choice recommendations. Neurocomputing, 2023, 520, pp. 152–170.

Sparse Low Rank-Factorization (SLR) [Swaminathan et *al.* 2020]

Another way to reduce the dot computation is to approximate the original weight matrix $\boldsymbol{W}$ by the product of two smaller matrices

Sparse Low Rank-Factorization (SLR) [Swaminathan et *al.* 2020]

Another way to reduce the dot computation is to approximate the original weight matrix $\boldsymbol{W}$ by the product of two smaller matrices

- Exploits low-rank factorization of rank q via truncated SVD of $\boldsymbol{W}$, i.e. $\boldsymbol{W} \simeq \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$, $\boldsymbol{U} \in \mathbb{R}^{n\times q}$, $\boldsymbol{\Sigma} \in \mathbb{R}^{q\times q}$, and $\boldsymbol{V} \in \mathbb{R}^{m\times q}$

Sparse Low Rank-Factorization (SLR) [Swaminathan et *al.* 2020]

Another way to reduce the dot computation is to approximate the original weight matrix $\boldsymbol{W}$ by the product of two smaller matrices

- Exploits low-rank factorization of rank q via truncated SVD of $\boldsymbol{W}$, i.e. $\boldsymbol{W} \simeq \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$, $\boldsymbol{U} \in \mathbb{R}^{n\times q}$, $\boldsymbol{\Sigma} \in \mathbb{R}^{q\times q}$, and $\boldsymbol{V} \in \mathbb{R}^{m\times q}$
- The truncated matrices $\boldsymbol{U}$ and $\boldsymbol{V}$ are further compressed by assuming some input and output neurons in the layer are less important, utilizing for them a reduced rank $\bar{q} < q$

Sparse Low Rank-Factorization (SLR) [Swaminathan et *al.* 2020]

Another way to reduce the dot computation is to approximate the original weight matrix $\boldsymbol{W}$ by the product of two smaller matrices

- Exploits low-rank factorization of rank q via truncated SVD of $\boldsymbol{W}$, i.e. $\boldsymbol{W} \simeq \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$, $\boldsymbol{U} \in \mathbb{R}^{n\times q}$, $\boldsymbol{\Sigma} \in \mathbb{R}^{q\times q}$, and $\boldsymbol{V} \in \mathbb{R}^{m\times q}$
- The truncated matrices $\boldsymbol{U}$ and $\boldsymbol{V}$ are further compressed by assuming some input and output neurons in the layer are less important, utilizing for them a reduced rank $\bar{q} < q$
- The corresponding row and column entries in $\boldsymbol{U}$ and $\boldsymbol{V}^T$, respectively, are set to zero (*sparsification*)

Sparse Low Rank-Factorization (SLR) [Swaminathan et *al.* 2020]

Another way to reduce the dot computation is to approximate the original weight matrix $\boldsymbol{W}$ by the product of two smaller matrices

- Exploits low-rank factorization of rank q via truncated SVD of $\boldsymbol{W}$, i.e. $\boldsymbol{W} \simeq \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$, $\boldsymbol{U} \in \mathbb{R}^{n\times q}$, $\boldsymbol{\Sigma} \in \mathbb{R}^{q\times q}$, and $\boldsymbol{V} \in \mathbb{R}^{m\times q}$
- The truncated matrices $\boldsymbol{U}$ and $\boldsymbol{V}$ are further compressed by assuming some input and output neurons in the layer are less important, utilizing for them a reduced rank $\bar{q} < q$
- The corresponding row and column entries in $\boldsymbol{U}$ and $\boldsymbol{V}^T$, respectively, are set to zero (*sparsification*)
- The efficiency is increased by exploring blockwise matrix-matrix product

Sparse Low Rank-Factorization (SLR) [Swaminathan et *al.* 2020]

Another way to reduce the dot computation is to approximate the original weight matrix $\boldsymbol{W}$ by the product of two smaller matrices

- Exploits low-rank factorization of rank q via truncated SVD of $\boldsymbol{W}$, i.e. $\boldsymbol{W} \simeq \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$, $\boldsymbol{U} \in \mathbb{R}^{n\times q}$, $\boldsymbol{\Sigma} \in \mathbb{R}^{q\times q}$, and $\boldsymbol{V} \in \mathbb{R}^{m\times q}$
- The truncated matrices $\boldsymbol{U}$ and $\boldsymbol{V}$ are further compressed by assuming some input and output neurons in the layer are less important, utilizing for them a reduced rank $\bar{q} < q$
- The corresponding row and column entries in $\boldsymbol{U}$ and $\boldsymbol{V}^T$, respectively, are set to zero (*sparsification*)
- The efficiency is increased by exploring blockwise matrix-matrix product
- No retraining needed

S. Swaminathan, D. Garg, R. Kannan, et al., Sparse low rank factorization for deep neural network compression, Neurocomputing 398 (2020) 185–196

Lossless representation of the layer

Once 'simplified' the layer matrix, lossless format supporting the dot product can be leveraged:

1. E.g. CSC, CSR, CER, CSER, Index Map, SLR, HAM, sHAM

HAM - Huffman Address Map

Lossless format supporting dot product leveraging quantization in weight matrix $\overline{\boldsymbol{W}}$

- Depending on k, weights $c_1, c_2, \ldots, c_k$ might have a high frequency

HAM - Huffman Address Map

Lossless format supporting dot product leveraging quantization in weight matrix $\overline{\boldsymbol{W}}$

- Depending on k, weights $c_1, c_2, \ldots, c_k$ might have a high frequency

- Apply canonical Huffman coding to quantized weights

HAM - Huffman Address Map

Lossless format supporting dot product leveraging quantization in weight matrix $\overline{\boldsymbol{W}}$

- Depending on k, weights $c_1, c_2, \ldots, c_k$ might have a high frequency

- Apply canonical Huffman coding to quantized weights

- $\texttt{HAM}(\overline{\boldsymbol{W}})$ is obtained by coding weights in $\overline{\boldsymbol{W}}$ by column

HAM - Huffman Address Map

Lossless format supporting dot product leveraging quantization in weight matrix $\overline{\boldsymbol{W}}$

- Depending on k, weights $c_1, c_2, \ldots, c_k$ might have a high frequency

- Apply canonical Huffman coding to quantized weights

- $\mathtt{HAM}(\overline{\boldsymbol{W}})$ is obtained by coding weights in $\overline{\boldsymbol{W}}$ by column

- $\mathtt{HAM}(\overline{\boldsymbol{W}})$ is split into $\left\lceil \frac{|\mathtt{HAM}(\overline{\boldsymbol{W}})|}{B} \right\rceil$ integers (B word size)

- To decompress we also need to store the inverse Huffman code

Marinò, G.C. et *al.* Efficient and Compact Representations of Deep Neural Networks via Entropy Coding. IEEE Access, 2023, 11, pp. 106103–106125

sHAM - Sparse Huffman Address Map

Lossless format supporting dot product exploiting the quantization and sparsity in $\overline{\boldsymbol{W}}$

- The weight 0 (most frequent) is excluded from the code, by first representing $\overline{\boldsymbol{W}}$ in Compressed Sparse Column (CSC) or CSR format

sHAM - Sparse Huffman Address Map

Lossless format supporting dot product exploiting the quantization and sparsity in $\overline{\boldsymbol{W}}$

- The weight 0 (most frequent) is excluded from the code, by first representing $\overline{\boldsymbol{W}}$ in Compressed Sparse Column (CSC) or CSR format

- Then compress the CSC vector containing the non-zero values with `HAM` format

sHAM - Sparse Huffman Address Map

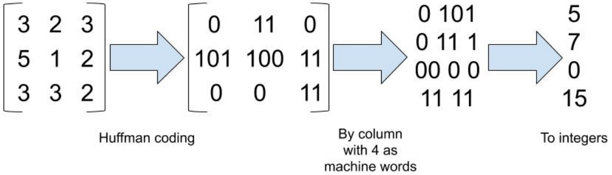
Lossless format supporting dot product exploiting the quantization and sparsity in $\overline{\boldsymbol{W}}$

- The weight 0 (most frequent) is excluded from the code, by first representing $\overline{\boldsymbol{W}}$ in Compressed Sparse Column (CSC) or CSR format
- Then compress the CSC vector containing the non-zero values with `HAM` format
- More efficient than `HAM` for very sparse input matrices

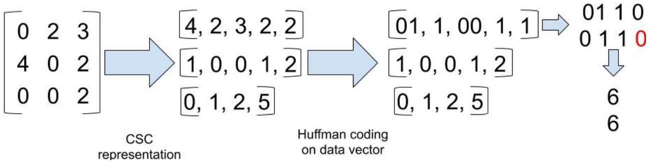
Marinò, G.C. et *al*. Efficient and Compact Representations of Deep Neural Networks via Entropy Coding. IEEE Access, 2023, 11, pp. 106103–106125

HAM/sHAM – Example

Example of HAM format



Example of sHAM format



Outline

1. The Need to Compress DNNs

2 Compressing for Pre-Trained Models

- 2.1 Structure-Preserving Compression Strategies
- sHAM
- 2.2 Structure-Altering Compression Strategies

3. Train Compressible Models

4. Exam Paper List

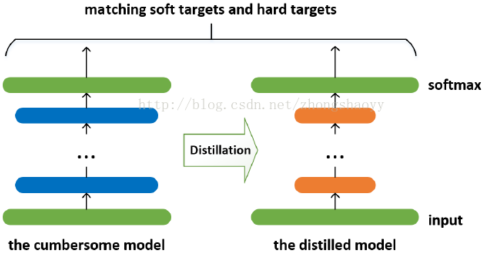
Knowledge Distillation (KD) in Deep Neural Networks

- Knowledge distillation [Hinton et *al.* 2015] transfers knowledge from a large, accurate model (teacher) to a smaller model (student).

Knowledge Distillation (KD) in Deep Neural Networks

- Knowledge distillation [Hinton et *al.* 2015] transfers knowledge from a large, accurate model (teacher) to a smaller model (student).

- The teacher model has already been trained on the task at hand (e.g., image classification), and its predictions are considered "gold standard"

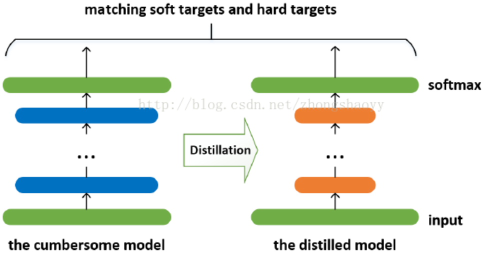


Source: https://796t.com/content/1548498443.html

Knowledge Distillation (KD) in Deep Neural Networks

- Knowledge distillation [Hinton et *al.* 2015] transfers knowledge from a large, accurate model (teacher) to a smaller model (student).

- The teacher model has already been trained on the task at hand (e.g., image classification), and its predictions are considered "gold standard"



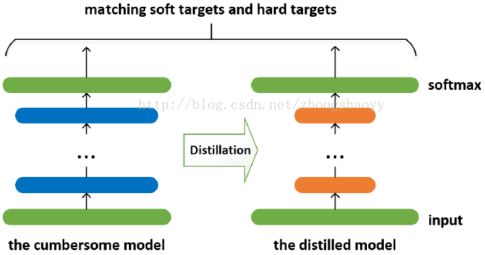
Source: https://796t.com/content/1548498443.html

– Soft targets (probabilities) from the teacher guide the student's learning.

Knowledge Distillation (KD) in Deep Neural Networks

- Knowledge distillation [Hinton et *al.* 2015] transfers knowledge from a large, accurate model (teacher) to a smaller model (student).

- The teacher model has already been trained on the task at hand (e.g., image classification), and its predictions are considered "gold standard"



Source: https://796t.com/content/1548498443.html

– Soft targets (probabilities) from the teacher guide the student's learning.

– Soft Targets: Instead of using the teacher's hard class labels (e.g., "cat"," dog"), we use soft targets–probability distributions over classes

Hinton, G.E., Vinyals, O., and Dean, J. (2015). Distilling the Knowledge in a Neural Network. ArXiv, abs/1503.02531.

Distillation Process

- For each training input, the teacher provides a vector of class probabilities
  - e.g., [0.8, 0.1, 0.1] for “cat”,“dog”, “bird” (the soft targets btw)

Distillation Process

- For each training input, the teacher provides a vector of class probabilities
  - e.g., [0.8, 0.1, 0.1] for “cat”,“dog”, “bird” (the soft targets btw)
- Distillation Loss: The student model is trained to mimic the teacher’s behavior by minimizing a distillation loss.
  - The KD loss function compares the student’s predicted probabilities with the soft targets provided by the teacher

Distillation Process

- For each training input, the teacher provides a vector of class probabilities
  - e.g., [0.8, 0.1, 0.1] for “cat”,“dog”, “bird” (the soft targets btw)
- Distillation Loss: The student model is trained to mimic the teacher’s behavior by minimizing a distillation loss.
  - The KD loss function compares the student’s predicted probabilities with the soft targets provided by the teacher
- The goal is for the student to learn not only which class is correct but also the confidence associated with it

Temperature

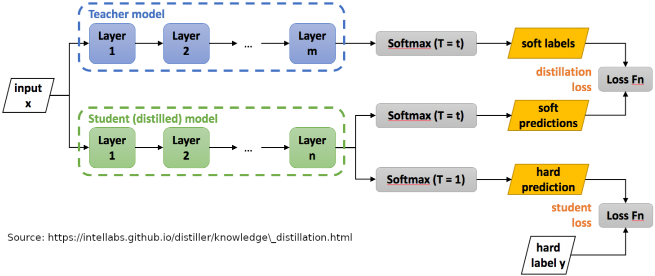
- The soft targets are often scaled using a temperature parameter (usually denoted as T)
  - If i is the i-th output unit, whose logit is denoted by z_i, the soft probabilities are then computed as $q_i = \frac{\exp(z_i/T)}{\sum_j \exp(z_j/T)}$

Temperature

- The soft targets are often scaled using a temperature parameter (usually denoted as T)
  - If i is the i-th output unit, whose logit is denoted by z_i, the soft probabilities are then computed as $q_i = \frac{\exp(z_i/T)}{\sum_j \exp(z_j/T)}$
- Higher T values make the probabilities more uniform, lower T values sharpen the probabilities (more peaky)

Temperature

- The soft targets are often scaled using a temperature parameter (usually denoted as T)
  - If i is the i-th output unit, whose logit is denoted by z_i, the soft probabilities are then computed as $q_i = \frac{\exp(z_i/T)}{\sum_j \exp(z_j/T)}$
- Higher T values make the probabilities more uniform, lower T values sharpen the probabilities (more peaky)
- The overall loss combines the KD and the task-specific losses, using two values for T: $t > 1$ for teacher and student KD loss output, and $T = 1$ for student classification (same logits for both student losses)



Source: https://intellabs.github.io/distiller/knowledge_distillation.html

KD Conclusions

Benefits

- Model Compression: Student becomes smaller and efficient

KD Conclusions

Benefits

- Model Compression: Student becomes smaller and efficient
- Generalization: Insights from teacher improve student's performance

KD Conclusions

Benefits

- Model Compression: Student becomes smaller and efficient
- Generalization: Insights from teacher improve student's performance
- Robustness: Student learns from teacher's robust predictions

KD Conclusions

Benefits

- Model Compression: Student becomes smaller and efficient
- Generalization: Insights from teacher improve student's performance
- Robustness: Student learns from teacher's robust predictions

Drawbacks

- Choice of Teacher Models: The accuracy of student models depends on the choice of teacher models

KD Conclusions

Benefits

- Model Compression: Student becomes smaller and efficient
- Generalization: Insights from teacher improve student's performance
- Robustness: Student learns from teacher's robust predictions

Drawbacks

- Choice of Teacher Models: The accuracy of student models depends on the choice of teacher models
- Accuracy Degradation: Student models may not consistently achieve the same high accuracy as teacher models during inference

Other structure-Altering Compression Strategies: Channel/filter pruning

- Finding the optimal number of filters to be used in convolutional layers (apart from the filter size, the stride and other hyperparameters) is hard

Other structure-Altering Compression Strategies: Channel/filter pruning

- Finding the optimal number of filters to be used in convolutional layers (apart from the filter size, the stride and other hyperparameters) is hard
- Thus, many studies focused on trying to detect useless/irrelevant filters to be dropped out

Other structure-Altering Compression Strategies: Channel/filter pruning

- Finding the optimal number of filters to be used in convolutional layers (apart from the filter size, the stride and other hyperparameters) is hard
- Thus, many studies focused on trying to detect useless/irrelevant filters to be dropped out
- Among them, it worth to mention ThiNet [Luo et *al* 2017]

Other structure-Altering Compression Strategies: Channel/filter pruning

- Finding the optimal number of filters to be used in convolutional layers (apart from the filter size, the stride and other hyperparameters) is hard
- Thus, many studies focused on trying to detect useless/irrelevant filters to be dropped out
- Among them, it worth to mention ThiNet [Luo et *al* 2017]
  - Key insight: ThiNet differs from other methods for the fact that it prunes filters is banking upon statistics from the next layer (not the current layer)

Other structure-Altering Compression Strategies: Channel/filter pruning

- Finding the optimal number of filters to be used in convolutional layers (apart from the filter size, the stride and other hyperparameters) is hard
- Thus, many studies focused on trying to detect useless/irrelevant filters to be dropped out
- Among them, it worth to mention ThiNet [Luo et *al* 2017]
  - Key insight: ThiNet differs from other methods for the fact that it prunes filters is banking upon statistics from the next layer (not the current layer)
- Filter Pruning is set as an Optimization Problem

Luo, J.H. et *al.* ThiNet: A Filter Level Pruning Method for Deep Neural Network Compression. 2017. arXiv:1707.06342.

ThiNet

- Detecting filters to be discarded: if a subset of input channels in layer $i+1$ (i.e filters in layer i) are enough to approximate the output in layer $i+1$, then the other channels (filters in layer i) can be safely removed

ThiNet

- Detecting filters to be discarded: if a subset of input channels in layer $i + 1$ (i.e filters in layer i) are enough to approximate the output in layer $i + 1$, then the other channels (filters in layer i) can be safely removed

- Fine Tuning: Necessary to retrain for some epochs to recover the generalization ability damaged by filter pruning.
  - Authors suggests, for time-saving considerations, to fine-tune one or two epochs after the pruning of one layer

ThiNet

- Detecting filters to be discarded: if a subset of input channels in layer $i + 1$ (i.e filters in layer i) are enough to approximate the output in layer $i + 1$, then the other channels (filters in layer i) can be safely removed
- Fine Tuning: Necessary to retrain for some epochs to recover the generalization ability damaged by filter pruning.
  - Authors suggests, for time-saving considerations, to fine-tune one or two epochs after the pruning of one layer
- Optimization Process:
  1. Compute the output of each layer in the feed-forward step;

ThiNet

- Detecting filters to be discarded: if a subset of input channels in layer $i + 1$ (i.e filters in layer i) are enough to approximate the output in layer $i + 1$, then the other channels (filters in layer i) can be safely removed

- Fine Tuning: Necessary to retrain for some epochs to recover the generalization ability damaged by filter pruning.
  - Authors suggests, for time-saving considerations, to fine-tune one or two epochs after the pruning of one layer

- Optimization Process:
  1. Compute the output of each layer in the feed-forward step;
  2. For each channel, fix a number of channel r to be pruned away

ThiNet

- Detecting filters to be discarded: if a subset of input channels in layer $i+1$ (i.e filters in layer i) are enough to approximate the output in layer $i+1$, then the other channels (filters in layer i) can be safely removed
- Fine Tuning: Necessary to retrain for some epochs to recover the generalization ability damaged by filter pruning.
  - Authors suggests, for time-saving considerations, to fine-tune one or two epochs after the pruning of one layer
- Optimization Process:
  1. Compute the output of each layer in the feed-forward step;
  2. For each channel, fix a number of channel r to be pruned away
  3. Select and drop the r channels yielding to the smallest contribution to the activation

ThiNet

- Detecting filters to be discarded: if a subset of input channels in layer $i+1$ (i.e filters in layer i) are enough to approximate the output in layer $i+1$, then the other channels (filters in layer i) can be safely removed
- Fine Tuning: Necessary to retrain for some epochs to recover the generalization ability damaged by filter pruning.
  - Authors suggests, for time-saving considerations, to fine-tune one or two epochs after the pruning of one layer
- Optimization Process:
  1. Compute the output of each layer in the feed-forward step;
  2. For each channel, fix a number of channel r to be pruned away
  3. Select and drop the r channels yielding to the smallest contribution to the activation
- Solving point 3 is NP-hard, thus authors adopt a greedy approximate solution growing the set of r filters one at a time, adding the one having the smallest objective values

Outline

1. The Need to Compress DNNs

2 Compressing for Pre-Trained Models
- 2.1 Structure-Preserving Compression Strategies
- sHAM
- 2.2 Structure-Altering Compression Strategies

3. Train Compressible Models

4. Exam Paper List

Train Compressible Models

Category of methods that attempts to directly learn a model whose parameters are optimized to be compressed next according to predifined schemes

Train Compressible Models

Category of methods that attempts to directly learn a model whose parameters are optimized to be compressed next according to predifined schemes

- Simpler approaches for instance just add penalties (e.g. L1 norm of layer weights) to induce sparsity in the trained network (see https://arxiv.org/pdf/1611.06694 for instance)

Train Compressible Models

Category of methods that attempts to directly learn a model whose parameters are optimized to be compressed next according to predifined schemes

- Simpler approaches for instance just add penalties (e.g. L1 norm of layer weights) to induce sparsity in the trained network (see https://arxiv.org/pdf/1611.06694 for instance)
- Perpiñán et *al.* 2017 instead propose to learn a model along with its compression via constrained optimization frameworks

Train Compressible Models

Category of methods that attempts to directly learn a model whose parameters are optimized to be compressed next according to predifined schemes

- Simpler approaches for instance just add penalties (e.g. L1 norm of layer weights) to induce sparsity in the trained network (see https://arxiv.org/pdf/1611.06694 for instance)
- Perpiñán et *al.* 2017 instead propose to learn a model along with its compression via constrained optimization frameworks
- Penalty terms are included into the loss, each for every compression scheme

Train Compressible Models

Category of methods that attempts to directly learn a model whose parameters are optimized to be compressed next according to predifined schemes

- Simpler approaches for instance just add penalties (e.g. L1 norm of layer weights) to induce sparsity in the trained network (see https://arxiv.org/pdf/1611.06694 for instance)
- Perpiñán et *al*. 2017 instead propose to learn a model along with its compression via constrained optimization frameworks
- Penalty terms are included into the loss, each for every compression scheme
- Supports different schemes, including pruning and quantization

Train Compressible Models

Category of methods that attempts to directly learn a model whose parameters are optimized to be compressed next according to predifined schemes

- Simpler approaches for instance just add penalties (e.g. L1 norm of layer weights) to induce sparsity in the trained network (see https://arxiv.org/pdf/1611.06694 for instance)
- Perpiñán et *al*. 2017 instead propose to learn a model along with its compression via constrained optimization frameworks
- Penalty terms are included into the loss, each for every compression scheme
- Supports different schemes, including pruning and quantization
- Alternated training and compression steps to reduce the complexity and tackle intractable problems

M. Á. Carreira-Perpiñán, Y. Idelbayev. Model compression as constrained optimization, with application to neural nets. 2017. arXiv:1707.04319

Outline

1. The Need to Compress DNNs

2 Compressing for Pre-Trained Models
- 2.1 Structure-Preserving Compression Strategies
- sHAM
- 2.2 Structure-Altering Compression Strategies

3. Train Compressible Models

4. Exam Paper List

List of papers about DNN compression to discuss for the exam

- https://dl.acm.org/doi/10.1145/3617688. Smart-DNN+: A Memory-efficient Neural Networks Compression Framework for the Model Inference;
- https://dl.acm.org/doi/abs/10.5555/3491440.3491534. Channel pruning via automatic structure search;
- https://arxiv.org/abs/2012.03096. Parallel Blockwise Knowledge Distillation for Deep Neural Network Compression;
- https://dl.acm.org/doi/10.1145/3459637.3482005. LC: A Flexible, Extensible Open-Source Toolkit for Model Compression;
- Link. ECC: Platform-Independent Energy-Constrained Deep Neural Network Compression via a Bilinear Regression Model.