Neural Network Compression

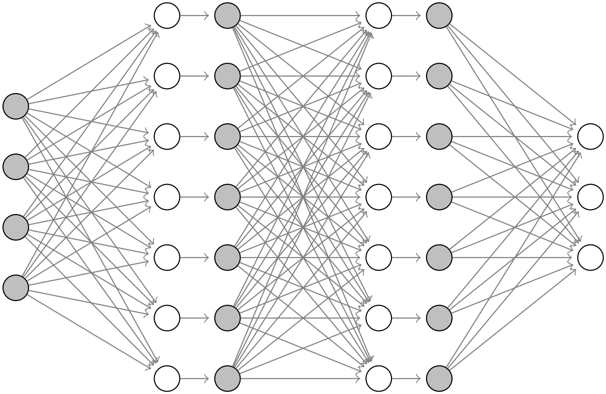
Linear Neural Reconstruction

David A. R. Robin

Nov 29, 2019

Neural networks

$$X \in \mathbb{R}^d \mapsto W_2 \cdot \sigma_1(W_1 \cdot \sigma_0(W_0 \cdot X))$$



Notations

$w \in \mathbb{R}^n$	: weights of a full neural network
$\mathcal{L} : \mathbb{R}^n \to \mathbb{R}$	: loss function
$\odot : (x, y) \mapsto (x_i y_i)_i$	: pointwise product
$\mathbb{1}_A : x \mapsto \mathbb{1}(x \in A)$	: set indicator function
$d \in \mathbb{N}$	: number of inputs of a layer
$h \in \mathbb{N}$	: number of outputs of a layer
$W \in \mathbb{R}^{h \times d}$	: weights of a single layer
$\mathcal{D} \quad$ (over $\mathbb{R}^d$)	: distribution of inputs to a layer
$X \sim \mathcal{D}$	: input to a layer (random variable)

Previously in Network Compression

Previously in Network Compression : Pruning

Too many weights. Which one would you remove ?

$$\mathcal{L}(w+\delta w)-\mathcal{L}(w) \underset{0}{=} \nabla\mathcal{L}(w)^T\cdot\delta w+\frac{1}{2}\delta w^T\cdot\nabla^2\mathcal{L}(w)\cdot\delta w+\mathcal{O}(\|\delta w\|^3)$$

Cost of pruning a single weight : $\Delta\mathcal{L}(w_q \cdot e_q) \approx \frac{1}{2}w_q^2 \cdot (\nabla^2\mathcal{L})_{qq}$

Previously in Network Compression : Pruning

Too many weights. Which one would you remove ?

$$\mathcal{L}(w+\delta w)-\mathcal{L}(w) \underset{0}{=} \nabla\mathcal{L}(w)^T \cdot \delta w + \frac{1}{2}\delta w^T \cdot \nabla^2\mathcal{L}(w) \cdot \delta w + \mathcal{O}(\|\delta w\|^3)$$

Cost of pruning a single weight : $\Delta\mathcal{L}(w_q \cdot e_q) \approx \dfrac{1}{2} w_q^2 \cdot (\nabla^2\mathcal{L})_{qq}$

Previously in Network Compression : Pruning

Too many weights. Which one would you remove ?

$$\mathcal{L}(w+\delta w)-\mathcal{L}(w) \underset{0}{=} \nabla \mathcal{L}(w)^T \cdot \delta w + \frac{1}{2} \delta w^T \cdot \nabla^2 \mathcal{L}(w) \cdot \delta w + \mathcal{O}(\|\delta w\|^3)$$

Cost of pruning a single weight : $\Delta \mathcal{L}(w_q \cdot e_q) \approx \frac{1}{2} w_q^2 \cdot (\nabla^2 \mathcal{L})_{qq}$

Previously in Network Compression : Pruning

Too many weights. Which one would you remove ?

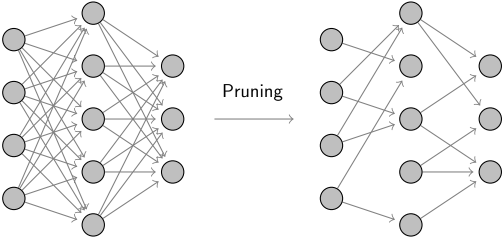
$$\mathcal{L}(w+\delta w)-\mathcal{L}(w) \underset{0}{=} \nabla\mathcal{L}(w)^T\cdot\delta w+\frac{1}{2}\delta w^T\cdot\nabla^2\mathcal{L}(w)\cdot\delta w+\mathcal{O}(\|\delta w\|^3)$$

Cost of pruning a single weight : $\Delta\mathcal{L}(w_q\cdot e_q)\approx\frac{1}{2}w_q^2\cdot(\nabla^2\mathcal{L})_{qq}$

Previously in Network Compression : Pruning

Pruning : Remove weights (i.e. connections)

Assumption : small magnitude $|w_q|$ pruned $\rightarrow$ small loss increase
(even when pruning several weights at once)

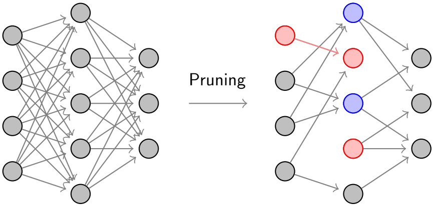


Algorithm : Prune, Retrain, Repeat

Result : 90% of weights removed, same accuracy (high compressibility)

Previously in Network Compression : Explaining Pruning

Magnitude-based pruning requires retraining.



Neurons with no inputs or no outputs (in red) can be kept[1], as well as redundant neurons (in blue) that could be discarded at no cost.

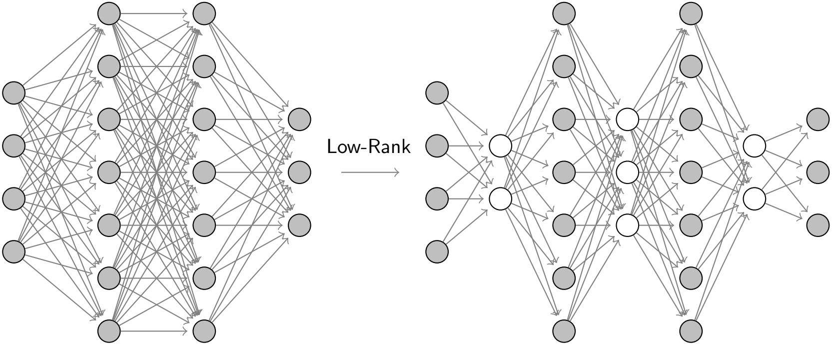
Redundancy is not leveraged

Can we take advantage of redundancies ?

[1] given enough retraining with weight decay, these will be discarded

Previously in Network Compression : Low-rank

$$\min_{P \in \mathbb{R}^{h \times r}, Q \in \mathbb{R}^{d \times r}} \left\| W - PQ^T \right\|_2$$



Problems : keeps hidden neuron count intact, data-agnostic

Contribution

Activation reconstruction

L-layer feed-forward flow:

- Z_0 input to the network
- $Z_{k+1} = \sigma_k(W_k \cdot Z_k)$
- Use Z_L as prediction

Weight approximation *(theirs)*:

- $\hat{W}_k \approx W_k$

Activation reconstruction *(ours)*:

- $\hat{Z}_k \approx Z_k$

We have more than weights, we have activations

We only need $\sigma_k(\hat{W}_k Z_k) \approx \sigma_k(W_k Z_k)$

$$\hat{Z}_k \approx Z_k \, , \, \sigma_k(\hat{W}_k Z_k) \approx \sigma_k(W_k Z_k) \quad \Rightarrow \quad \hat{Z}_{k+1} \approx Z_{k+1}$$

$$\sigma_k(\hat{W}_k Z_k) \approx \sigma_k(W_k Z_k) \quad \not\Rightarrow \quad \hat{W}_k \approx W_k$$

Linear activation reconstruction

$$\hat{W}_k \approx W_k \quad \Rightarrow \quad \hat{W}_k Z_k \approx W_k Z_k \quad \Rightarrow \quad \sigma_k(\hat{W}_k Z_k) \approx \sigma_k(W_k Z_k)$$

The first ($\hat{W}_k \approx W_k$) is sub-optimal because data-agnostic

The third ($\sigma_k(\hat{W}_k Z_k) \approx \sigma_k(W_k Z_k)$) is non-convex, non-smooth

Let's try to get $\hat{W}_k \cdot Z_k \approx W_k \cdot Z_k$

Low-rank inspiration

Low-rank with activation reconstruction gives

$$\min_{P\in\mathbb{R}^{h\times r},\, Q\in\mathbb{R}^{n\times r}} \mathbb{E}_X \left\| WX - PQ^T X \right\|_2^2$$

Q : feature extractor,
P : linear reconstruction from extracted features

Knowing the right rank r to use is hard.
Soft low-rank would use the nuclear norm $\|\cdot\|_*$ instead

$$\min_M \mathbb{E}_X \| WX - MX \|_2^2 + \lambda \cdot \|M\|_*$$

where λ controls the tradeoff between compression and accuracy

Neuron removal

$$\mathcal{C}_i(M) = 0 \Rightarrow X_i \text{ is never used} \Rightarrow \text{we can remove neuron n}^\circ i$$

Column-sparse matrices remove neurons.

Caracterization of such matrices reminiscent of low-rank : PC^T

Low-Rank : $M = PQ^T$

- $P \in \mathbb{R}^{h \times r_Q}$
- $Q \in \mathbb{R}^{d \times r_Q}$

Column-sparse : $M = PC^T$

- $P \in \mathbb{R}^{h \times r_C}$
- $C \in \{0,1\}^{d \times r_C}$, $C^T 1_d = 1_r$

Q the feature extractor becomes a feature selector C

Leveraging consecutive layers

Restricting to feature selectors, we gain an interesting property

feature selector's action commute with non-linearities

For a three-layer network:

$$\begin{array}{rrrrl} & W_3 \cdot \sigma_2(& W_2 \cdot \sigma_1(& W_1 \cdot X &)) \\ \approx & P_3 C_3^T \cdot \sigma_2(& P_2 C_2^T \cdot \sigma_1(& P_1 C_1^T \cdot X &)) \\ = & P_3 \cdot \sigma_2(& C_3^T P_2 \cdot \sigma_1(& C_2^T P_1 \cdot C_1^T X &)) \\ = & \hat{W}_3 \cdot \sigma_2(& \hat{W}_2 \cdot \sigma_1(& \hat{W}_1 \cdot C_1^T X &)) \end{array}$$

Memory footprint:

- original : $h_3 \times h_2 + h_2 \times h_1 + h_1 \times d$
- compressed : $h_3 \times r_3 + r_3 \times r_2 + r_2 \times r_1 + \alpha \cdot \log_2 \binom{d}{r_1}$

h_2 and h_1 are gone ! Only h_3 (#outputs) and d (#inputs) remain

Optimality of feature selectors

feature selector's action commute with non-linearities:

$$C \in \{0,1\}^{r\times d}\,,\ C^T 1_d = 1_r \quad \Rightarrow \quad PC^T \cdot \sigma(U) = P \cdot \sigma(C^T U)$$

We only need the commutation property.
Can we maybe use something less extreme than feature selectors ?

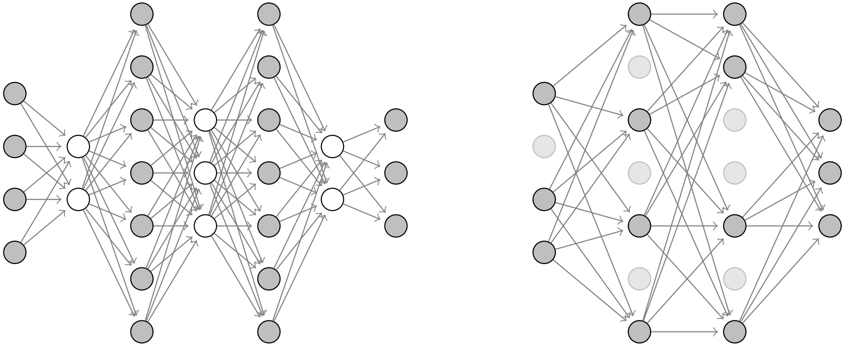
Lemma (commutation lemma)

Let C be a linear operator
Let $\sigma : x \mapsto \max(0,x)$ be the pointwise ReLU
C's action commutes with σ $\Rightarrow$ C is a feature selector

Answer : No, not even if all σ_k are ReLU

Comparison with low-rank

Hidden neurons are deleted



Note how this doesn't suffer pruning drawbacks discussed before

Comparison with low-rank

Low-Rank : $M = PQ^T$

- $P \in \mathbb{R}^{h \times r_Q}$
- $Q \in \mathbb{R}^{d \times r_Q}$

Column-sparse : $M = PC^T$

- $P \in \mathbb{R}^{h \times r_C}$
- $C \in \{0,1\}^{d \times r_C}$, $C^T 1_d = 1_r$

For the same ℓ_2 error, low-rank is less constrained, hence $r_Q \leq r_C$
But it doesn't remove hidden neurons, which may dominate its cost

Two regimes:

- Heavy overparameterization $(r_C \ll d)$: use column-sparse
- Light overparameterization $(r_C \approx d)$: use low-rank

Once neurons have been removed, it is still possible to apply low-rank approximation on top of the first compression

Solving for column-sparse

Linear Neural Reconstruction Problem

Using the $\ell_{2,1}$ norm as a proxy for the number of non-zero columns, we can consider the following distinct relaxation

$$\min_{M} \ \mathbb{E}_X \| WX - MX \|_2^2 + \lambda \cdot \|M\|_{2,1} \tag{1}$$

where $\|M\|_{2,1} = \sum_j \sqrt{\sum_i M_{i,j}^2}$ is the $\ell_{2,1}$ norm of M, i.e. the sum of the ℓ_2-norms of its columns.

Auto-correlation factorization

The sum over the training set can be factored away

Using $A = W - M$, we have

$$\mathbb{E}_X \| AX \|_2^2 = \mathbb{E}_X \operatorname{Tr}\left(A \cdot XX^T \cdot A^T\right) = \operatorname{Tr}\left(A \cdot (\mathbb{E}_X XX^T) \cdot A^T\right)$$

$R = \mathbb{E}_X[XX^T] \in \mathbb{R}^{d \times d}$ is the auto-correlation matrix.

The objective can then be evaluated in $\mathcal{O}(hd^2)$, which does not depend on the number of samples.

Efficient solving

Our problem is strictly convex $\rightarrow$ solvable to global optimum

We solve it with Fast Iterative Shrinkage-Thresholding,
an accelerated proximal gradient method (quadratic convergence).

Lemma (quadratic convergence)

Let $\mathcal{L} : M \mapsto \frac{1}{2} \cdot \mathbb{E}_X \| WX - MX \|_2^2 + \lambda \cdot \|M\|_{2,1}$*,*
$(M_k)_k$ *the iterates obtained by the FISTA algorithm,*
M^* *the global optimum, and* $L = \lambda_{\max}(\mathbb{E}_X[XX^T])$*. Then*

$$\mathcal{L}(M_k) - \mathcal{L}(M^*) \leq \frac{2L}{k^2} \|M_0 - M^*\|_F^2$$

Extension to convolutional layers

For each output position (u, v) in output channel j,
we write $X_i^{(u,v)}$ the associated input,
that will be multiplied by W_j to get $(W * X_i)_{j,u,v}$

$$\|W * X_i\|_2^2 = \sum_j \sum_{u,v} \left\| W_j \odot X_i^{(u,v)} \right\|_2^2$$

hence

$$R \propto \sum_i \sum_{u,v} \text{vec}(X_i^{(u,v)}) \cdot \text{vec}(X_i^{(u,v)})^T$$

This rewriting holds for any stride, padding or dilation values

Then use more general Group-Lasso instead of $\ell_{2,1}$

Tackling Lasso bias

Lasso regularization $\rightarrow$ shrinkage effect $\rightarrow$ bias in the solution

We limit this effect by solving twice

- Solve for (P, C^T) and retain only C
- Solve for P with fixed C without penalty

The second is just a linear regression

Influence of debiasing

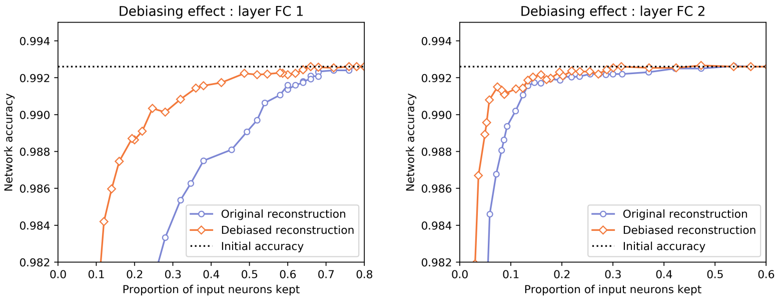


Figure: Influence of debiasing on reconstruction quality (LeNet-5 Caffe)

Results

General results

Network		Error		Comp. rate	Size
Architecture	Type	Top-1	Top-5		
LeNet-300-100	Baseline	1.68 %	-	-	1.02 MiB
	Compressed	1.71 %	-	46 %	482 KiB
	Retrained (1)	1.64 %	-	29 %	307 KiB
LeNet-5 (Caffe)	Baseline	0.74 %	-	-	1.64 MiB
	Compressed	0.78 %	-	16 %	276 KiB
	Retrained (1)	0.78 %	-	10 %	177 KiB
AlexNet	Baseline	43.48 %	20.93 %	-	234 MiB
	Compressed	45.36 %	21.90 %	39 %	91 MiB

Reconstruction chaining

Extension to arbitrary output

We can extend the previous problem to reconstruct arbitrary output Y

$$\min_M \frac{1}{2N} \sum_i \| Y_i - MX_i \|_2^2 + \lambda \cdot \|M\|_{2,1} \tag{2}$$

FISTA is adapted by simply changing the gradient step $dA = YX^T - AXX^T$, where YX^T can be precomputed as well

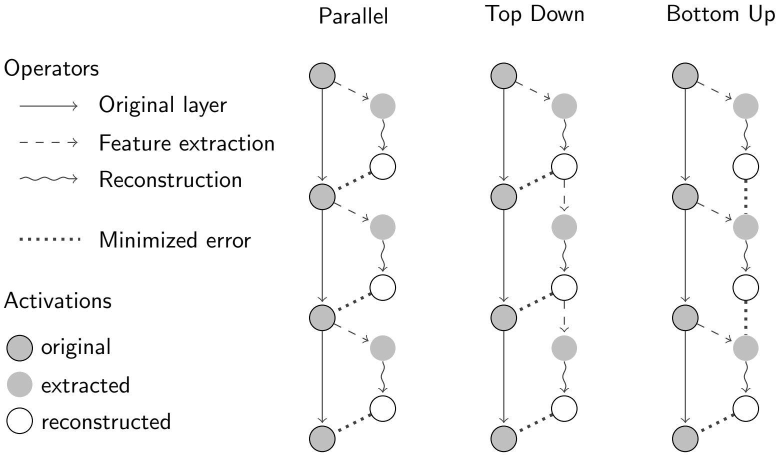
Three chaining strategies

Consider a feed-forward fully connected network
Input Z_0, weights $(W_k)_k$ and non-linearities $(\sigma_k)_k$

$$Z_{k+1} = \sigma_k(W_k \cdot Z_k)$$

- Parallel : $Y = W_k \cdot Z_k\,, \quad X = Z_k$
- Top-down : $Y = W_k \cdot Z_k\,, \quad X = \hat{Z}_k$
- Bottom-up : $Y = C_{k+1}^T W_k \cdot Z_k\,, \quad X = Z_k$

Three chaining strategies



Reconstruction chaining

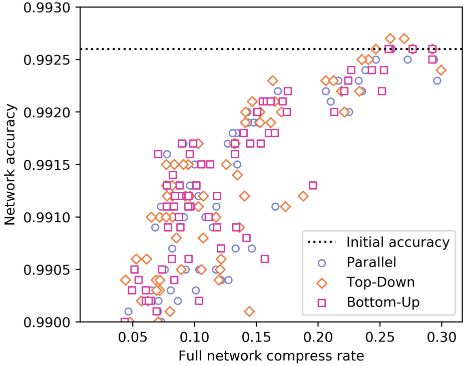


Figure: Performances of reconstruction chainings (LeNet-5 Caffe)

Practical Session

`www.robindar.com/teaching/lnr_practical_session.ipynb`

Appendix

Fast Iterative Shrinkage-Thresholding

Algorithm 1 FISTA with fixed step size

input: $X \in \mathbb{R}^{h \times N}$: input to the layer,
$W \in \mathbb{R}^{o \times h}$: weight to approximate,
λ : hyperparameter

output: $M \in \mathbb{R}^{o \times h}$: reconstruction

$R \leftarrow XX^T/N$

$L \leftarrow$ largest eigenvalue of R

$M \leftarrow 0 \in \mathbb{R}^{o \times h}$, $P \leftarrow 0 \in \mathbb{R}^{o \times h}$

$t \leftarrow \lambda/L$, $k \leftarrow 1$, $\theta \leftarrow 1$

repeat

$\theta \leftarrow (k-1)/(k+2)$, $k \leftarrow k+1$

$A \leftarrow M + \theta(M - P)$

$dA \leftarrow (W - A)R$

$P \leftarrow M,\ M \leftarrow prox_{t\|\cdot\|_{2,1}}(A - dA/L)$

until desired convergence

Convergence guarantees

Lemma

Let $\mathcal{L} : M \mapsto \frac{1}{2} \cdot \mathbb{E}_X \| WX - MX \|_2^2 + \lambda \cdot \|M\|_{2,1}$,
$(M_k)_k$ *the iterates obtained by FISTA as described above,*
M^* *the global optimum, and* $L = \lambda_{\max}(\mathbb{E}_X[XX^T])$. *Then*

$$\mathcal{L}(M_k) - \mathcal{L}(M^*) \leq \frac{2L}{k^2} \|M_0 - M^*\|_F^2$$

Choosing $M_0 = 0$, we can refine this bound with the following

$$\|M^*\|_F^2 \leq \|M^*\|_{2,1} \cdot \min\left(\sqrt{d}, \|M^*\|_{2,1}\right)$$

and by definition of M^*, we have $\forall M$, $\|M^*\|_{2,1} \leq \frac{1}{\lambda}\mathcal{L}(M)$