Network Sparsification

reduce redundancy in the number of weights

Feature Selection

use only a small subset of the features

Advantages

- better generalization
- smaller memory footprint
- faster prediction
- less expensive to collect features

Challenging

Example

- w: 1000-dimensional; $B = 5$
- number of possible choices: $C_{1000}^{5} = 8.25 \times 10^{12}$ (expensive)

heuristic methods

- forward selection
  - the best single-feature is picked first
  - add the next best feature, ...
  - $\{\} \to \{A_1\} \to \{A_1, A_4\} \to \cdots$
- backward elimination
  - repeatedly eliminate the worst feature
  - $\{A_1, A_2, A_3, A_4, A_5, A_6\} \to \{A_1, A_3, A_4, A_5, A_6\} \to \{A_1, A_3, A_4, A_6\} \to \cdots$
- ...

Regularization

Training samples $\{(x_1, y_1), \ldots, (x_n, y_n)\}$

empirical risk minimization

$$\min_w \ \text{loss}$$

prior knowledge about the model $\rightarrow$ regularizer

regularized risk minimization

$$\min_w \text{loss} + \lambda \ \underbrace{r(w)}_{\text{regularizer}}$$

What Regularizer?

prior knowledge: many model parameters should be small

Example (ℓ_2-regularizer)

$$r(w) = \|w\|_2^2 = \sum_{i=1}^{d} w_i^2$$

square loss$+$ ℓ_2 regularizer $\rightarrow$ ridge regression

Model with Large Number of Parameters

prior knowledge: many parameters are not useful

minimize loss + sparsity-inducing regularizer

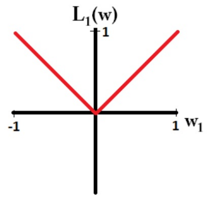
Example (ℓ_0 regularizer)

- $\|w\|_0$: number of nonzero elements in w
- feature selection: $\|w\|_0 \leq B$

ℓ_1 Regularizer

ℓ_1 as a surrogate for ℓ_0

$$r(w) = \|w\|_1 = \sum_{i=1}^{d} |w_i|$$



Example (Lasso (Tibshirani, 1996))

use square loss: $\min \|y - Xw\|^2$ s.t. $\|w\|_1 \leq t$

Pruning of Deep Networks

what to prune?

1. unstructured pruning
2. structured pruning

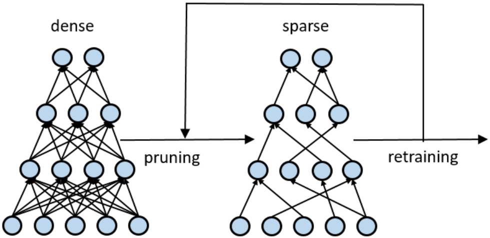
how does the pruned network look like?

Unstructured Pruning of Deep Networks

Prune Weights

(Han et al., 2015)

Learning both Weights and Connections for Efficient Neural Networks



three-step process

1. learn the connectivity via normal network training
2. prune the connections (dense network → sparse network)
3. retrain the network learn the final weights for the remaining sparse connections

Which Connections to Prune?

based on the

1. weight magnitude
   - assumption: small weights are less important
   - all connections with weights below a threshold are removed from the network
   - encourage unimportant weights to smaller values
     $\rightarrow$ usually also use ℓ_1 or ℓ_2 regularization
2. hidden unit activation
   - small activation value $\rightarrow$ this feature detector is less important for prediction
3. change in loss after removing weights
4. ...

Experimental Results

Lenet on MNIST, AlexNet/VGG-16 on ImageNet

Network	Top-1 Error	Top-5 Error	Parameters	Compression Rate
LeNet-300-100 Ref	1.64%	-	267K	
LeNet-300-100 Pruned	1.59%	-	**22K**	**12**×
LeNet-5 Ref	0.80%	-	431K	
LeNet-5 Pruned	0.77%	-	**36K**	**12**×
AlexNet Ref	42.78%	19.73%	61M	
AlexNet Pruned	42.77%	19.67%	**6.7M**	**9**×
VGG-16 Ref	31.50%	11.32%	138M	
VGG-16 Pruned	31.34%	10.88%	**10.3M**	**13**×

[from (Han et al., 2015)]

- network pruning can save 9x to 13x parameters with no drop in predictive performance

Regularizing Deep Network Weights: ℓ_2

(Collins et al., 2014)

Memory Bounded Deep Convolutional Networks

ℓ_2 regularizer

$R(\mathbf{w}) = \|\mathbf{w}\|_2^2$ (weight decay)

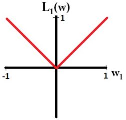
reduce the weight magnitude, but not exactly to zero

Regularizing Network Weights: ℓ_1

ℓ_1 regularizer

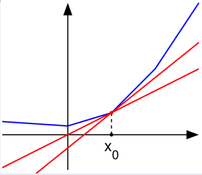
$$R(\mathbf{w}) = \|\mathbf{w}\|_1$$

- nonsmooth
- replace gradient by subgradient



subgradient

- g is a subgradient of f at x iff $f(y) \geq f(x) + g'(y - x)$
- may not be unique
- if f is differentiable $\rightarrow$ gradient

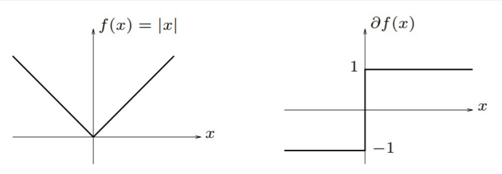


subgradient descent

$x_{t+1} = x_t - \eta g_t$, for some subgradient g_t of f at x_t

Subgradient Descent for ℓ_1

Example ($f(x) = |x|$)



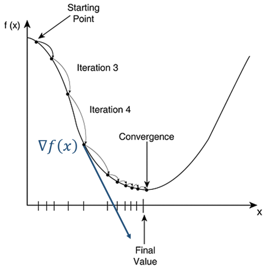
$$w_i \leftarrow w_i - \eta \operatorname{sign}(w_i)$$

√ produce a large number of weights that are very near zero

- but almost never output weights that are exactly zero
- $\rightarrow$ need to threshold away the very small weights to obtain a sparse solution

Gradient Descent

$$\min_x \underbrace{f(x)}_{\text{smooth}}$$



LOOP

1. find descent direction
2. descent

$$x_{t+1} = x_t - \eta \nabla f(x_t)$$

easy to implement

$$\begin{aligned} x_{t+1} &= \arg\min_x \nabla f(x_t)^T (x - x_t) + \frac{1}{2\eta} \|x - x_t\|^2 \\ &= \arg\min_x \underbrace{f(x_t) + \nabla f(x_t)^T (x - x_t)}_{\text{linear approximation}} + \underbrace{\frac{1}{2\eta} \|x - x_t\|^2}_{\text{proximal term}} \end{aligned}$$

Proximal Gradient Descent

$$\min_{\mathbf{x}} \underbrace{f(\mathbf{x})}_{\text{smooth}} + \underbrace{g(\mathbf{x})}_{\text{nonsmooth}}$$

$$\begin{aligned} w_{t+1} &= \arg\min_w f(w_t) + \nabla f(w_t)'(w - w_t) + \frac{1}{2\eta}\|w - w_t\|^2 + g(w) \\ &= \underbrace{\arg\min_w g(w) + \frac{1}{2\eta}\|w - z_t\|^2}_{\text{proximal step}} \ \text{(where } z_t = w_t - \eta\nabla f(w_t)) \end{aligned}$$

$$\text{prox}_{\eta g}(z) \equiv \arg\min_w \tfrac{1}{2}\|w - z\|^2 + \eta g(w)$$

$\text{prox}_{\eta g}(z)$ often has simple closed-form solution!

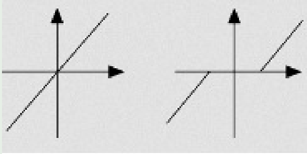
Proximal Step

Example (ℓ_2 regularizer)

$$\begin{aligned} \mathrm{prox}_{\eta\frac{\lambda}{2}\|\cdot\|^2}(z) &= \arg\min_w \eta\frac{\lambda}{2}\|w\|^2 + \frac{1}{2}\|w-z\|^2 \\ &= \frac{1}{1+\eta\lambda}\cdot z \quad (\text{shrinkage}) \end{aligned}$$

Example (ℓ_1 regularizer)

$$\begin{aligned} \mathrm{prox}_{\eta\lambda\|\cdot\|_1}(z) &= \arg\min_w \eta\lambda\|w\|_1 + \frac{1}{2}\|w-z\|^2 \\ &= [\mathrm{sign}(z_i)\cdot\max(|z_i|-\eta\lambda,0)] \quad (\text{soft-thresholding}) \end{aligned}$$



Empirically, in Deep Learning: ℓ_1 vs ℓ_2

ℓ_1 regularization

- results in more zero parameters than ℓ_2 regularization
- better accuracy after pruning, but before retraining

ℓ_2 regularization

- higher accuracy than ℓ_1 regularization after retraining

Regularizing Network Weights: ℓ_0

ℓ_0 regularizer

$$R(\mathbf{w}) = \|\mathbf{w}\|_0$$

heuristic update procedure

- every n updates: set to zero all but the t elements of the parameter vector with the largest magnitude

Deep Compression

combine pruning with other techniques for further compression

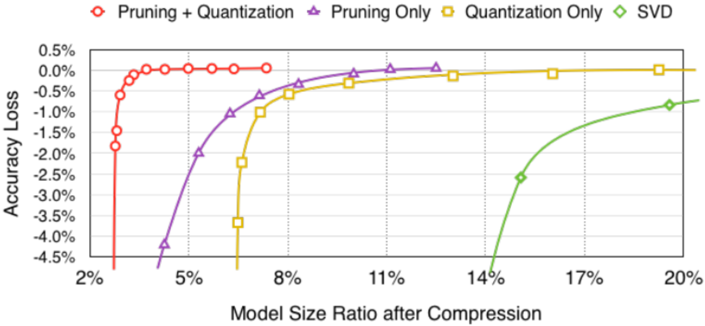
(Han et al. 2016)

Deep compression: Compressing deep neural network with pruning, trained quantization and Huffman coding

1. pruning
   - train $\rightarrow$ prune small-weight connections $\rightarrow$ retrain
2. weight sharing
   - cluster the weights (k-means clustering)
     $\rightarrow$ generate codebook
     $\rightarrow$ multiple weights share the same codebook
     $\rightarrow$ retrain codebook
3. Huffman coding of clustered weights

Experimental Results

Accuracy vs compression rate



[from (Han et al. 2016)]

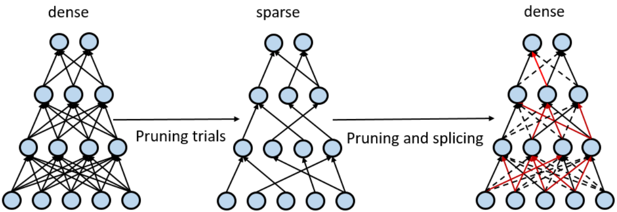
Problem with Pruning

- connections may be wrongly pruned
- once pruned, connections gone forever

(Guo et al., 2017)

Dynamic Network Surgery for Efficient DNNs

splicing: enable recovery of important connections



Which Connections to Prune or Splice?

cannot just look at the weight magnitude

- associate an importance measure t_i to each weight w_i
  - $t_i = 0$: unimportant $\rightarrow$ prune
  - $t_i = 1$: important $\rightarrow$ keep
- use thresholds a, b and update t_i as

$$t_i \leftarrow \begin{cases} 0 & a > |w_i| \quad \text{(prune)} \\ t_i & a \leq |w_i| < b \\ 1 & b \leq |w_i| \quad \text{(splice)} \end{cases}$$

Dynamic Network Surgery

procedure

1. forward propagation with $\mathbf{w} \odot \mathbf{t}$
2. backward propagation
3. update t_i
   - but only with probability $\sigma(\text{iter})$
   - slow down pruning and splicing frequencies
4. update w_i (including those with zero t_is)

$$w_i = w_i - \eta \frac{\partial}{\partial(w_i t_i)} \ell(\mathbf{w} \odot \mathbf{t})$$

Experimental Results

AlexNet

Layer	Params.	Params.% [9]	Params.% (Ours)
conv1	35K	$\sim$ 84%	53.8%
conv2	307K	$\sim$ 38%	40.6%
conv3	885K	$\sim$ 35%	29.0%
conv4	664K	$\sim$ 37%	32.3%
conv5	443K	$\sim$ 37%	32.5%
fc1	38M	$\sim$ 9%	3.7%
fc2	17M	$\sim$ 9%	6.6%
fc3	4M	$\sim$ 25%	4.6%
Total	61M	$\sim$ 11%	**5.7%**

[from (Guo et al., 2017)]

- reduces model complexity, while the prediction error rate does not increase

Another Problem

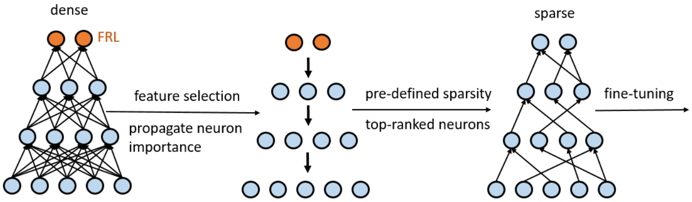
- pruning usually done layer by layer

- weights may appear unimportant in an early layer
- but may contribute significantly to important units in later layers

Neuron Importance Score Propagation (NISP)

(Yu et al., 2018)

NISP: Pruning Networks using Neuron Importance Score Propagation



1. obtain importance scores of units in the last hidden layer (using some feature ranking algorithm)
2. propagate importance scores
3. prune less important units
4. fine-tuning

Pros and Cons of Weight Pruning

pros

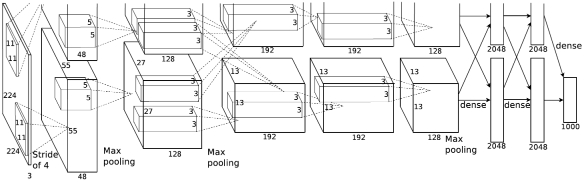
- fine-grained, most flexible
- can lead to high compression rate

cons

- standard ML frameworks do not support efficient operation on sparse weight tensors
- require special libraries or hardware for fast inference

Limitations of Weight Pruning in CNN

AlexNet



[from (Krizhevsky, 2012)]

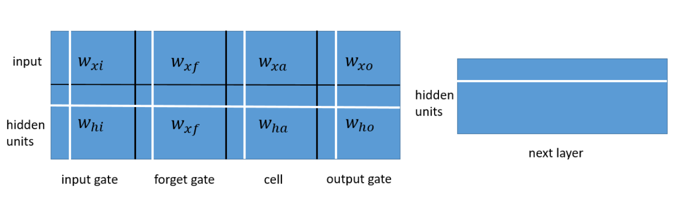
- weights removed mostly come from the fully-connected layers
  - computations in these layers are cheap
  - resultant time reduction is insignificant
  - many new deep networks use fewer fully-connected layers
- convolution is more costly $\rightarrow$ runtime during prediction is dominated by evaluation of convolution layers

Limitations of Weight Pruning in RNN

- In RNN, the basic structures interweave with each other

Example (LSTM)

- one hidden unit is associated with entries in all eight matrices, and entries in the corresponding entries in the next layer



- independently removing these structures can result in mismatch of their dimensions and then inducing invalid recurrent units

Structured Pruning

Structured Pruning in CNN

prune filters/channels directly

pros

- does not introduce sparsity $\rightarrow$ does not require sparse libraries or any specialized hardware
- easy to set a target number of filters to be pruned

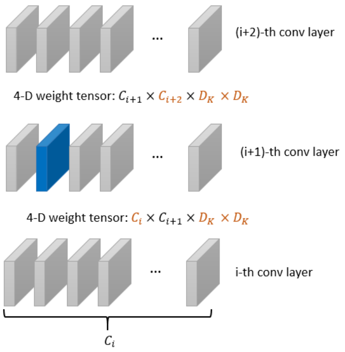
cons

- coarse-grained, less flexible

Pruning Filters: Based on Weights

(Li et al., 2017)

Pruning filters for efficient ConvNets



1. measure importance of each filter
   - e.g., sum of absolute kernel weights
2. prune some filters with smallest importance
   - remove corresponding feature map
   - kernels that apply on the removed feature maps from the filters of the next convolutional layer are also removed
3. after pruning, retrain the network

Pruning Filters: Based on Other Criteria

(Molchanov et al., 2017)

Pruning convolutional neural networks for resource efficient transfer learning

find a network (with parameter $\mathbf{w}'$) such that

- it is small
- its performance (e.g., negative log-likelihood) is close to that of the original network (with parameter $\mathbf{w}$)

$$\min_{\mathbf{w}'} |\ell(\mathbf{w}') - \ell(\mathbf{w})| \ : \ \|\mathbf{w}'\|_0 \leq B$$

Procedure

$$\min_{\mathbf{w}'} |\ell(\mathbf{w}') - \ell(\mathbf{w})| \;:\; \|\mathbf{w}'\|_0 \leq B$$

1. evaluate importance of feature maps
   - weight magnitude (assume that small weights are less important)
   - hidden unit activation
   - estimated change in loss after removing weights (use Taylor expansion)
   - based on batch normalization
2. remove the least important feature map
3. fine-tuning
4. repeat until $\|\mathbf{w}'\|_0 \leq B$

Batch Normalization (BN)

- normalizes activations for mini-batch $B = \{x_1, \ldots, x_m\}$ (usually done before nonlinearity)
- $\rightarrow$ helps training of deep networks

batch normalization

$$
\begin{aligned}
\text{mini-batch mean:} \quad & \mu_B \leftarrow \tfrac{1}{m} \textstyle\sum_{i=1}^{m} x_i \\
\text{mini-batch variance:} \quad & \sigma_B^2 \leftarrow \tfrac{1}{m} \textstyle\sum_{i=1}^{m} (x_i - \mu_B)^2 \\
\text{normalize:} \quad & \hat{x}_i \leftarrow \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \\
\text{scale and shift:} \quad & y_i \leftarrow \gamma \hat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i)
\end{aligned}
$$

- γ (scaling) and β are learnable parameters

Batch Normalization in CNN

- input X: size $B \times X \times Y \times C$

1. $\mu_c = \frac{1}{BXY} \sum_{bxy} x_{bxyc}$
2. $\sigma_c^2 = \frac{1}{BXY} \sum_{bxy} (x_{bxyc} - \mu_c)^2$
3. output

$$y_{bxyc} = \left(\frac{x_{bxyc} - \mu_c}{\sqrt{\sigma_c^2 + \epsilon}} \right) \cdot \gamma_c + \beta_c$$

Pruning Channels: Network Slimming

$$y_{bxyc} = \left(\frac{x_{bxyc} - \mu_c}{\sqrt{\sigma_c^2 + \epsilon}} \right) \cdot \gamma_c + \beta_c$$

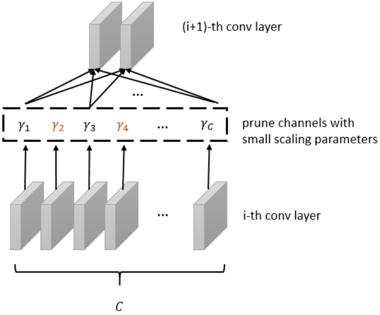
(Liu et al., 2017)

Learning Efficient Convolutional Networks through Network Slimming

idea

- magnitude of $\gamma_c \rightarrow$ importance of channel c
- to select channels
  $\rightarrow$ enforcing sparsity on these scaling parameters
  $\rightarrow$ imposes ℓ_1 regularization

Network Slimming Procedure



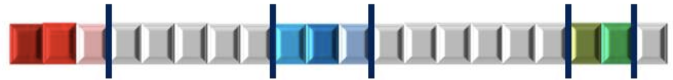
1. jointly train the network weights and BN parameters, with ℓ_1 regularization imposed on γ_c's
   - $\surd$ easy to implement without introducing any change to existing CNN architectures
2. prune channels with small factors
3. fine-tuning
4. repeat

Structured Sparsity

Features often have intrinsic structures

Example (group sparsity)

a categorical feature (e.g., nationality) is represented by a group of dummy binary features (european/american/chinese)



- select/drop whole groups
- group sparsity: a few groups are selected

Group Lasso Regularizer

- an extension of the lasso for feature selection on (predefined) groups of features
- given a number of groups G_i's
- $r(w) = \sum_i \|w_{G_i}\|_2$
  - take the ℓ_2 norm of all the groups
  - ℓ_1 norm of the above (sparsity on groups)
- reduces to the lasso if groups have cardinality one

Using Group Lasso in CNNs

(Wen et al, 2016)

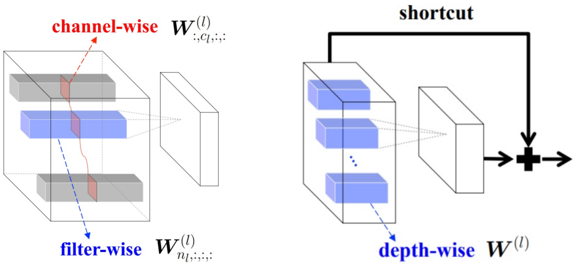
Learning structured sparsity in deep neural networks

minimize loss + sparsity-inducing regularizer

$$\min \; E_D(W) + \lambda R(W) + \lambda_g \sum_{l=1}^{L} R_g(W^{(l)})$$

- $E_D(W)$: loss on data
- $R(W)$: ℓ_2 regularizer
- R_g: group lasso regularizer

Remove Less Important Structures in CNN



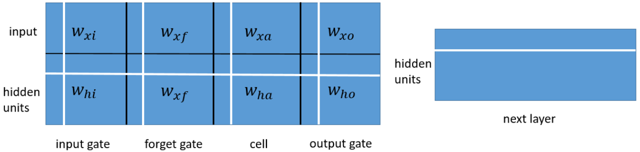
- define each channel as a group → remove less important channels
- define each filter as a group → remove less important filters
- define each depth as a group → reduce the depth

Group Lasso in LSTM

(Wen et al., 2018)

Learning intrinsic sparse structures within long short-term memory

$$\begin{aligned}
\mathbf{i}_t &= \sigma(\mathbf{x}_t \mathbf{W}_{xi} + \mathbf{h}_{t-1}\mathbf{W}_{hi} + \mathbf{b}_i) \\
\mathbf{f}_t &= \sigma(\mathbf{x}_t \mathbf{W}_{xf} + \mathbf{h}_{t-1}\mathbf{W}_{hf} + \mathbf{b}_f) \\
\mathbf{a}_t &= \tanh(\mathbf{x}_t \mathbf{W}_{xa} + \mathbf{h}_{t-1}\mathbf{W}_{ha} + \mathbf{b}_a) \\
\mathbf{o}_t &= \sigma(\mathbf{x}_t \mathbf{W}_{xo} + \mathbf{h}_{t-1}\mathbf{W}_{ho} + \mathbf{b}_o) \\
\mathbf{c}_t &= \mathbf{i}_t \odot \mathbf{a}_t + \mathbf{f}_\tau \odot \mathbf{c}_{t-1} \\
\mathbf{h}_t &= \mathbf{o}_t \odot \tanh(\mathbf{c}_\tau)
\end{aligned}$$

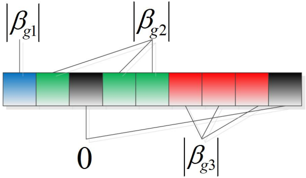


- prune a hidden state → prune corresponding row
- prune input-gate/forget-gate/cell/output-gate → prune corresponding column

Other Structured Sparsity Regularizers

Example

with unknown groups, can we obtain the group structure automatically?



feature grouping

- group relevant and highly correlated features

OSCAR

(Bondell & Reich, 2008)

Simultaneous regression shrinkage, variable selection, and supervised clustering of predictors with OSCAR

Octagonal Shrinkage and Clustering Algorithm for Regression

$$\min_w \|y - Xw\|^2 + \lambda_1 \underbrace{\|w\|_1}_{\text{sparsity}} + \underbrace{\lambda_2 \sum_{i<j} \max\{|w_i|, |w_j|\}}_{\text{grouping}}$$

- encourages equality of coefficients for correlated features $\rightarrow$ automatic grouping

(Zhang et al., 2018)

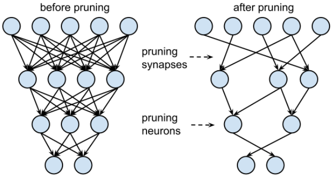
Learning to share: Simultaneous parameter tying and sparsification in deep learning

- GrOWL (group ordered weighted ℓ_1)

Topology of Pruned Networks

Pruning

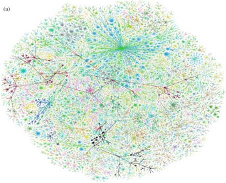
1. train a network (with full-precision weights)
2. prune unimportant weights

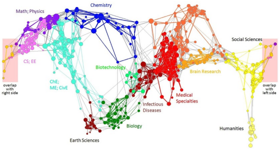


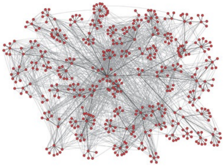
3. retrain the pruned network
4. (repeat...)

(deep networks) pruning/retraining $\leftrightarrow$ (biological networks) weakening/strengthening

Networks and Power Law



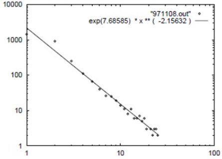




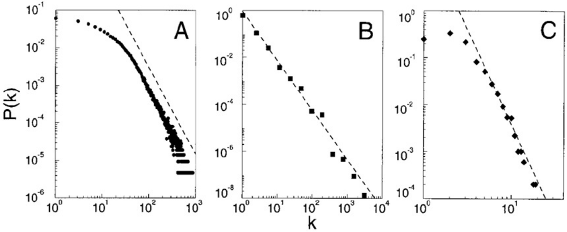
- distribution of node degree follows the power law

$$p(x) \propto x^{-\alpha}$$

- e.g., internet, collaboration, web, power grid, ...

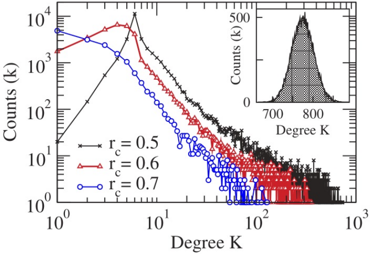


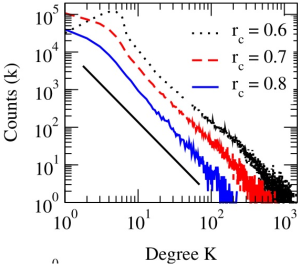
(a) Int-11-97



Scale-Free Brain Functional Networks

- distribution of functional connections also follows the power law [Eguiluz et al, 2005]





Are artificial neural networks scale-free?

(Lu and Kwok, 2018)

Power Law in Sparsified Deep Neural Networks

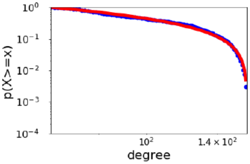
MLP on MNIST

MLP: input - 1024FC - 1024FC - 10softmax

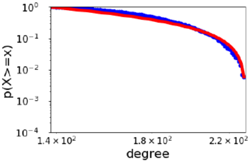
- truncated power-law distribution (TPL) [Kolyukhin and Torabi, 2013]

$$p(x) = \frac{1-\alpha}{x_{\max}^{1-\alpha} - x_{\min}^{1-\alpha}} x^{-\alpha}$$

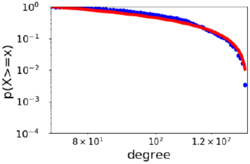
  - $[x_{\min}, x_{\max}]$: range over which the power law is valid



(a) input.



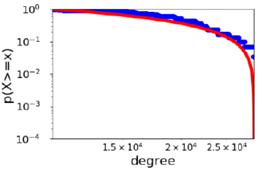
(b) fc1.



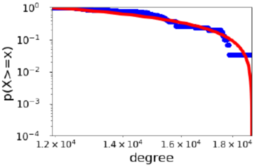
(c) fc2.

CNN on MNIST and CIFAR-10

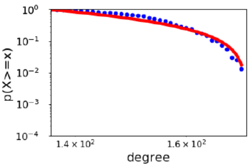
input - 32C5 - MP2 - 32C5 - MP2 - 256FC - 10softmax



(a) conv1.

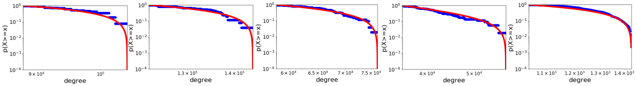


(b) conv2.



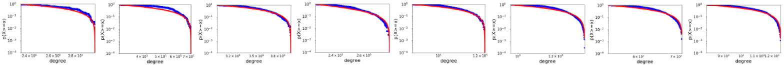
(c) fc3.

input - 32C3 - 32C3 - MP2 - 64C3 - 64C3 - MP2 - 512FC - 10softmax



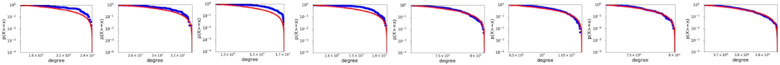
Larger CNNs on ImageNet

AlexNet

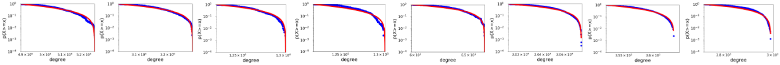


(a) conv1. (b) conv2. (c) conv3. (d) conv4. (e) conv5. (f) fc6. (g) fc7. (h) fc8.

VGG-16



(a) conv1_1. (b) conv1_2. (c) conv2_1. (d) conv2_2. (e) conv3_1. (f) conv3_2. (g) conv3_3. (h) conv4_1.



(i) conv4_2. (j) conv4_3. (k) conv5_1. (l) conv5_2. (m) conv5_3. (n) fc6. (o) fc7. (p) fc8.

Preferential Attachment

- power law distributions can originate from the process of preferential attachment [Barabasi and Albert, 1999]

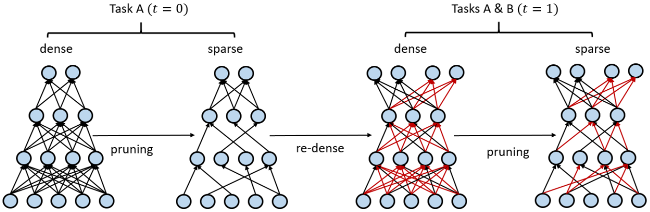
preferential attachment

- network evolves over time
- new nodes are more likely to link to nodes that already have high degrees
- rich gets richer

- when a new edge is added to the network, existing nodes with high degrees are more likely to connect to each other

Do we have preferential attachment in artificial neural networks?

Continual Learning



Time $t = 0$

- task A: MNIST image classification
- a dense network is trained, pruned and re-trained

Time $t = 1$

- new task B: MNIST image classification, but pixels inside a central square of each image are permuted
- add back pruned connections, re-train on task B
- prune, and re-train on task B

Number of Connections Created

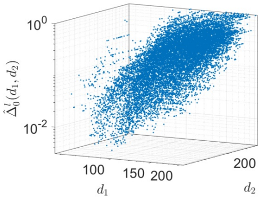
consider two nodes in consecutive layers

- one with degree d_1, the other with degree d_2

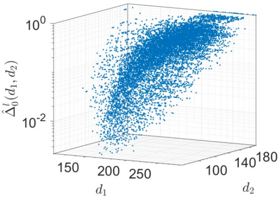
if there is preferential attachment ...

number of connections created between these two nodes

$\propto d_1 d_2$



(a) input-to-fc1.



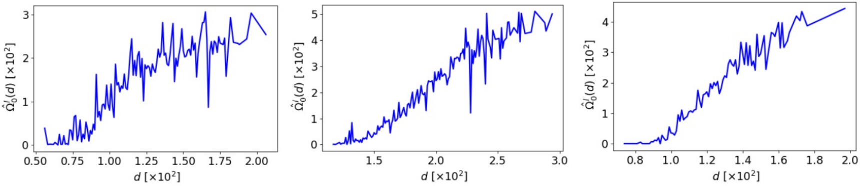
(b) fc1-to-fc2.

Increase in Node Degree

- $d_i(t)$: degree of node i at time t

increase in node degree

$$\frac{\mathrm{d}d_i(t)}{\mathrm{d}t} \propto d_i(0)$$

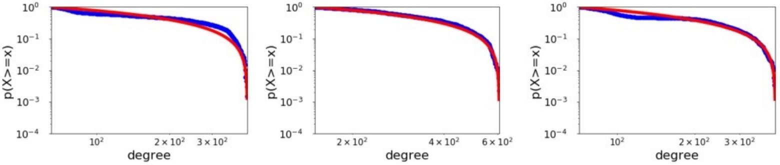


(a) input. (b) fc1. (c) fc2.

Degree Distribution

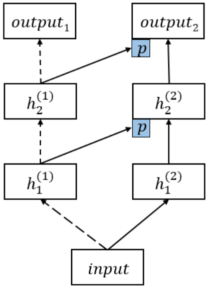
degree distribution before/after adding new task

If initial degree distribution follows the power law, the final degree distribution follows the same power law



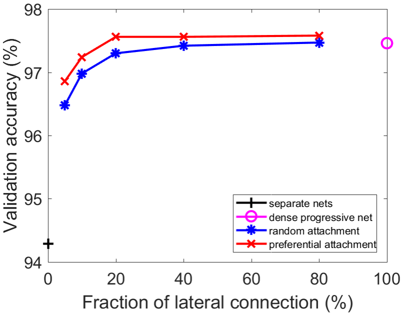
(a) input. (b) fc1. (c) fc2.

Progressive Network



1. train on task A using the left part of the network, prune, re-train
2. fix the parameters
3. train on task B: add the right part of the network
    - standard progressive network: $h_2^{(2)}$ is fully connected to $h_1^{(1)}$, ...
    - using preferential attachment: probability of connecting to a node in $h_1^{(1)}$ is proportional to its degree

Results on MNIST



- preferential attachment always outperforms random attachment
- 5% lateral connections to task A
  - preferential attachment has significantly better accuracy than using separate networks
- 20% lateral connections
  - preferential attachment has similar or even better accuracies than the dense progressive neural network