Spline Theory of Deep Network

Jérôme Gilles

Department of Mathematics and Statistics, SDSU
jgilles@sdsu.edu
http://jegilles.sdsu.edu

Deep Learning today

Tremendous results of “AI” or Deep Learning in many fields

but ...

Deep Learning today

Tremendous results of “AI” or Deep Learning in many fields

but ...

- how to choose the appropriate architecture?

Deep Learning today

Tremendous results of “AI” or Deep Learning in many fields

but ...

- how to choose the appropriate architecture?
- how many hidden layers?

Deep Learning today

Tremendous results of “AI” or Deep Learning in many fields

but ...

- how to choose the appropriate architecture?
- how many hidden layers?
- how many neurons?

Deep Learning today

Tremendous results of “AI” or Deep Learning in many fields

but ...

- how to choose the appropriate architecture?
- how many hidden layers?
- how many neurons?
- how to choose the appropriate size of filters and their number in convolutional layers?

Deep Learning today

Tremendous results of “AI” or Deep Learning in many fields

but ...

- how to choose the appropriate architecture?
- how many hidden layers?
- how many neurons?
- how to choose the appropriate size of filters and their number in convolutional layers?
- how to choose the appropriate learning objective function?

Deep Learning today

Tremendous results of “AI” or Deep Learning in many fields

but ...

- how to choose the appropriate architecture?
- how many hidden layers?
- how many neurons?
- how to choose the appropriate size of filters and their number in convolutional layers?
- how to choose the appropriate learning objective function?
- how big should be the training set?

Deep Learning today

Tremendous results of “AI” or Deep Learning in many fields

but ...

- how to choose the appropriate architecture?
- how many hidden layers?
- how many neurons?
- how to choose the appropriate size of filters and their number in convolutional layers?
- how to choose the appropriate learning objective function?
- how big should be the training set?
- what are really doing deep neural networks?

Deep Learning today

Tremendous results of “AI” or Deep Learning in many fields

but ...

- how to choose the appropriate architecture?
- how many hidden layers?
- how many neurons?
- how to choose the appropriate size of filters and their number in convolutional layers?
- how to choose the appropriate learning objective function?
- how big should be the training set?
- what are really doing deep neural networks?
- why do they work?

Deep Learning today

Tremendous results of “AI” or Deep Learning in many fields

but ...

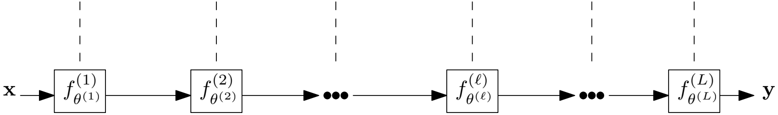
- how to choose the appropriate architecture?
- how many hidden layers?
- how many neurons?
- how to choose the appropriate size of filters and their number in convolutional layers?
- how to choose the appropriate learning objective function?
- how big should be the training set?
- what are really doing deep neural networks?
- why do they work?

$\Rightarrow$ need of a mathematical theory of deep learning

Spline Theory of Deep Network

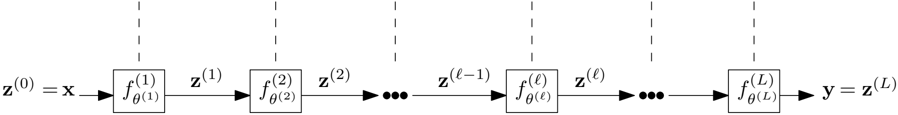
Work from Richard Baraniuk (ECE Dept - Rice University)
and his collaborators

Deep Learning - Notations (1/2)



Architecture:
$\mathbb{R}^C \ni \mathbf{y} = f_\Theta(\mathbf{x}) = \left(f^{(L)}_{\theta_{(L)}} \circ f^{(L-1)}_{\theta_{(L-1)}} \circ \ldots \circ f^{(\ell)}_{\theta_{(\ell)}} \circ \ldots \circ f^{(1)}_{\theta_{(1)}}\right)(\mathbf{x})\ ,\ \mathbf{x} \in \mathbb{R}^D$
where $\Theta = \{\theta^{(1)}, \theta^{(2)}, \ldots, \theta^{(L)}\}$ are the parameters (must be learned), and $f^{(l)}_{\theta^{(l)}}$ represent layer ℓ ($\in \{1, \ldots, L\}$).

Deep Learning - Notations (1/2)



Architecture:

$\mathbb{R}^C \ni \mathbf{y} = f_\Theta(\mathbf{x}) = \left(f^{(L)}_{\theta_{(L)}} \circ f^{(L-1)}_{\theta_{(L-1)}} \circ \ldots \circ f^{(\ell)}_{\theta_{(\ell)}} \circ \ldots \circ f^{(1)}_{\theta_{(1)}}\right)(\mathbf{x})\ ,\ \mathbf{x} \in \mathbb{R}^D$

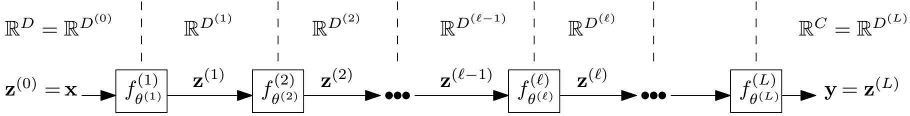
where $\Theta = \{\theta^{(1)}, \theta^{(2)}, \ldots, \theta^{(L)}\}$ are the parameters (must be learned), and $f^{(l)}_{\theta^{(l)}}$ represent layer ℓ ($\in \{1, \ldots, L\}$).

Intermediate variables:

$\mathbf{z}^{(\ell)}(\mathbf{x})$ is the output of layer ℓ

$= \left(f^{(\ell)}_{\theta_{(\ell)}} \circ \ldots \circ f^{(1)}_{\theta_{(1)}}\right)(\mathbf{x})\ ,\ \mathbf{z}^{(0)}(\mathbf{x}) = \mathbf{x}\ ,\ \mathbf{z}^{(L)}(\mathbf{x}) = \mathbf{y}$

Deep Learning - Notations (1/2)



Architecture:
$\mathbb{R}^C \ni \mathbf{y} = f_\Theta(\mathbf{x}) = \left(f^{(L)}_{\theta_{(L)}} \circ f^{(L-1)}_{\theta_{(L-1)}} \circ \ldots \circ f^{(\ell)}_{\theta_{(\ell)}} \circ \ldots \circ f^{(1)}_{\theta_{(1)}}\right)(\mathbf{x})\ ,\ \mathbf{x} \in \mathbb{R}^D$
where $\Theta = \{\theta^{(1)}, \theta^{(2)}, \ldots, \theta^{(L)}\}$ are the parameters (must be learned), and $f^{(l)}_{\theta^{(l)}}$ represent layer ℓ ($\in \{1, \ldots, L\}$).

Intermediate variables:
$\mathbf{z}^{(\ell)}(\mathbf{x})$ is the output of layer ℓ
$= \left(f^{(\ell)}_{\theta_{(\ell)}} \circ \ldots \circ f^{(1)}_{\theta_{(1)}}\right)(\mathbf{x})\ ,\ \mathbf{z}^{(0)}(\mathbf{x}) = \mathbf{x}\ ,\ \mathbf{z}^{(L)}(\mathbf{x}) = \mathbf{y}$

Assume that $\mathbf{z}^{(\ell)} \in \mathbb{R}^{D^{(\ell)}}$
(take the convention that $D^{(0)} = D$ and $D^{(L)} = C$)

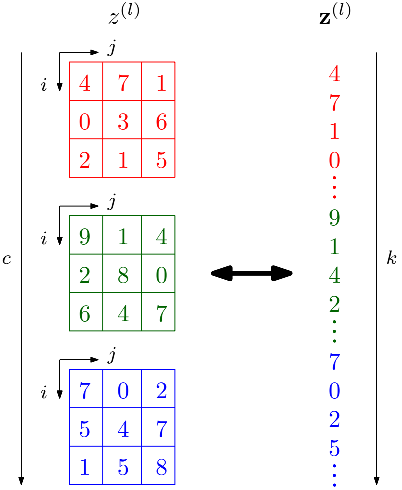
Deep Learning - Notations (2/2)

x can be any type of signal (audio, image,...).
Consider multichannel images: $z^{(\ell)}$ of size $C^{(\ell)} \times I^{(\ell)} \times J^{(\ell)} = D^{(\ell)}$

Deep Learning - Notations (2/2)

x can be any type of signal (audio, image,...).
Consider multichannel images: $z^{(\ell)}$ of size $C^{(\ell)} \times I^{(\ell)} \times J^{(\ell)} = D^{(\ell)}$

Samples can be indexed either in the tensor or flattened vector notation: $[z^{(\ell)}(x)]_{c,i,j} = [\mathbf{z}^{(\ell)}(\mathbf{x})]_k$



Deep Network (DN) basic operators (1/2)

- **fully connected operator**: $f_W^{(\ell)}(\mathbf{z}^{(\ell-1)}(\mathbf{x})) = W^{(\ell)}\mathbf{z}^{(\ell-1)}(\mathbf{x}) + b_W^{(\ell)}$ where $W^{(\ell)} \in \mathbb{R}^{D^{(\ell)} \times D^{(\ell-1)}}$ is a dense matrix and $b_W^{(\ell)} \in \mathbb{R}^{D^{(\ell)}}$ is a bias vector.

Deep Network (DN) basic operators (1/2)

- **fully connected operator**: $f_W^{(\ell)}(\mathbf{z}^{(\ell-1)}(\mathbf{x})) = W^{(\ell)}\mathbf{z}^{(\ell-1)}(\mathbf{x}) + b_W^{(\ell)}$ where $W^{(\ell)} \in \mathbb{R}^{D^{(\ell)} \times D^{(\ell-1)}}$ is a dense matrix and $b_W^{(\ell)} \in \mathbb{R}^{D^{(\ell)}}$ is a bias vector.
- **convolutional operator**: $f_C^{(\ell)}(\mathbf{z}^{(\ell-1)}(\mathbf{x})) = C^{(\ell)}\mathbf{z}^{(\ell-1)}(\mathbf{x}) + b_C^{(\ell)}$ where $C^{(\ell)} \in \mathbb{R}^{D^{(\ell)} \times D^{(\ell-1)}}$ is a multichannel block-circulant convolution matrix and $b_C^{(\ell)} \in \mathbb{R}^{D^{(\ell)}}$ is a bias vector.

Deep Network (DN) basic operators (1/2)

- **fully connected operator**: $f_W^{(\ell)}(\mathbf{z}^{(\ell-1)}(\mathbf{x})) = W^{(\ell)}\mathbf{z}^{(\ell-1)}(\mathbf{x}) + b_W^{(\ell)}$ where $W^{(\ell)} \in \mathbb{R}^{D^{(\ell)} \times D^{(\ell-1)}}$ is a dense matrix and $b_W^{(\ell)} \in \mathbb{R}^{D^{(\ell)}}$ is a bias vector.
- **convolutional operator**: $f_C^{(\ell)}(\mathbf{z}^{(\ell-1)}(\mathbf{x})) = C^{(\ell)}\mathbf{z}^{(\ell-1)}(\mathbf{x}) + b_C^{(\ell)}$ where $C^{(\ell)} \in \mathbb{R}^{D^{(\ell)} \times D^{(\ell-1)}}$ is a multichannel block-circulant convolution matrix and $b_C^{(\ell)} \in \mathbb{R}^{D^{(\ell)}}$ is a bias vector.
- **activation operator**: pointwise nonlinearity σ, $\left[f_\sigma^{(\ell)}(\mathbf{z}^{(\ell-1)}(\mathbf{x}))\right]_k = \sigma([\mathbf{z}^{(\ell-1)}(\mathbf{x})]_k)$ where
  - $\sigma_{ReLU}(u) = \max(0, u)$,
  - $\sigma_{LReLU}(u) = \max(\eta u, u)\ ,\ \eta > 0$,
  - $\sigma_{abs}(u) = |u|$,
  - $\sigma_{sig}(u) = \frac{1}{1+e^{-u}}$,
  - $\sigma_{\tanh}(u) = 2\sigma_{sig}(2u) - 1$.

Deep Network (DN) basic operators (2/2)

- **pooling operator**: policy ρ to reduce dimension. Collection of indices $\left\{\mathcal{R}_k^{(\ell)}\right\}_{k=1}^{D^{(\ell)}}$.
  Example: *max pooling*: $\left[f_\rho^{(\ell)}\left(\mathbf{z}^{(\ell-1)}(\mathbf{x})\right)\right]_k = \max_{d\in\mathcal{R}_k^{(\ell)}}\left[\mathbf{z}^{(\ell-1)}(\mathbf{x})\right]_d$.

Deep Network (DN) basic operators (2/2)

- **pooling operator**: policy ρ to reduce dimension. Collection of indices $\left\{\mathcal{R}_k^{(\ell)}\right\}_{k=1}^{D^{(\ell)}}$.
  Example: *max pooling*: $\left[f_\rho^{(\ell)}\left(\mathbf{z}^{(\ell-1)}(\mathbf{x})\right)\right]_k = \max_{d\in\mathcal{R}_k^{(\ell)}}\left[\mathbf{z}^{(\ell-1)}(\mathbf{x})\right]_d$.

Definition

A DN layer $f^{(\ell)}$ comprises a single nonlinear DN operator composed with any preceeding affine operator lying between it and the preceding nonlinear operator.

Example: CNN have two types of layers: 1) convolution-activation and 2) max-pooling.

Deep Network (DN) basic operators (2/2)

- **pooling operator**: policy ρ to reduce dimension. Collection of indices $\left\{\mathcal{R}_k^{(\ell)}\right\}_{k=1}^{D^{(\ell)}}$.
  Example: *max pooling*: $\left[f_\rho^{(\ell)}\left(\mathbf{z}^{(\ell-1)}(\mathbf{x})\right)\right]_k = \max_{d\in\mathcal{R}_k^{(\ell)}}\left[\mathbf{z}^{(\ell-1)}(\mathbf{x})\right]_d$.

Definition

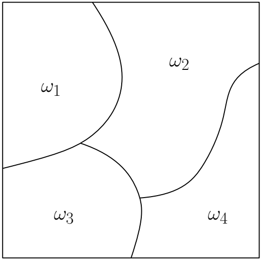
A DN layer $f^{(\ell)}$ comprises a single nonlinear DN operator composed with any preceeding affine operator lying between it and the preceding nonlinear operator.

Example: CNN have two types of layers: 1) convolution-activation and 2) max-pooling.

Output operator: $\mathbf{y} = g(f_\Theta(\mathbf{x}))$ where $g : \mathbb{R}^C \to \mathbb{R}^C$.
Example: *softmax* for classification problems, nothing for regression problems.

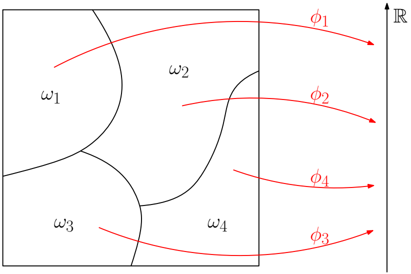
Multivariate Spline Operators

Partition $\mathbb{R}^D$ into R regions: $\Omega = \{\omega_1, \ldots, \omega_R\}$,



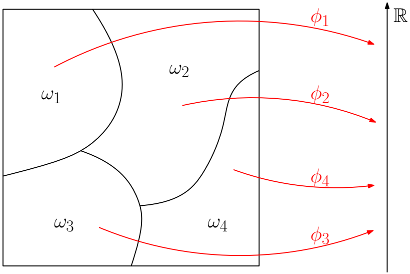
Multivariate Spline Operators

Partition $\mathbb{R}^D$ into R regions: $\Omega = \{\omega_1, \dots, \omega_R\}$,
Local mappings: $\Phi = \{\phi_1, \dots, \phi_R\}, \phi_r : \omega_r \to \mathbb{R}$ s.t $\phi_r(\mathbf{x}) := \langle [\alpha]_{r,.}, \mathbf{x} \rangle + [\beta]_r$
where $\alpha \in \mathbb{R}^{R \times D}$ and $\beta \in \mathbb{R}^R$ define hyperplanes in $\mathbb{R}^D$



Multivariate Spline Operators

Partition $\mathbb{R}^D$ into R regions: $\Omega = \{\omega_1, \dots, \omega_R\}$,
Local mappings: $\Phi = \{\phi_1, \dots, \phi_R\}, \phi_r : \omega_r \to \mathbb{R}$ s.t $\phi_r(\mathbf{x}) := \langle [\alpha]_{r,.}, \mathbf{x} \rangle + [\beta]_r$
where $\alpha \in \mathbb{R}^{R \times D}$ and $\beta \in \mathbb{R}^R$ define hyperplanes in $\mathbb{R}^D$



Definition (Multivariate Spline Operator)

$$s[\alpha, \beta, \Omega](\mathbf{x}) = \sum_{r=1}^{R} \left(\langle [\alpha]_{r,.}, \mathbf{x} \rangle + [\beta]_r \right) \mathbf{1}(\mathbf{x} \in \omega_r) =: \langle \alpha[\mathbf{x}], \mathbf{x} \rangle + \beta[\mathbf{x}]$$

Note: piecewise but not globally affine and convex (except for $R = 1$, degenerate spline)

Max-Affine Spline Operators (1/2)

Issue with general splines: to find the best spline approximation, we need to optimize wrt α, β and $\Omega \rightarrow$ minimizing wrt Ω is cumbersome to perform.

Max-Affine Spline Operators (1/2)

Issue with general splines: to find the best spline approximation, we need to optimize wrt α, β and $\Omega \rightarrow$ minimizing wrt Ω is cumbersome to perform.

$\Rightarrow$ constrain the multivariate spline to be **globally convex** $\rightarrow$ max-affine spline functions: $s[\alpha, \beta, \Omega](\mathbf{x}) = \max_{r=1,\dots,R} \langle [\alpha]_{r,.}, \mathbf{x} \rangle + [\beta]_r$

Max-Affine Spline Operators (1/2)

Issue with general splines: to find the best spline approximation, we need to optimize wrt α, β and $\Omega \rightarrow$ minimizing wrt Ω is cumbersome to perform.

$\Rightarrow$ constrain the multivariate spline to be **globally convex** $\rightarrow$ max-affine spline functions: $s[\alpha, \beta, \Omega](\mathbf{x}) = \max_{r=1,\dots,R}\langle[\alpha]_{r,.}, \mathbf{x}\rangle + [\beta]_r$

Properties:

- α and β define Ω (adaptive partitioning splines) thus we simply denote them $s[\alpha, \beta](\mathbf{x})$,
- always piecewise affine and globally convex (hence continuous),
- conversely

Max-Affine Spline Operators (1/2)

Issue with general splines: to find the best spline approximation, we need to optimize wrt α, β and $\Omega \rightarrow$ minimizing wrt Ω is cumbersome to perform.

$\Rightarrow$ constrain the multivariate spline to be **globally convex** $\rightarrow$ max-affine spline functions: $s[\alpha, \beta, \Omega](\mathbf{x}) = \max_{r=1,\dots,R} \langle [\alpha]_{r,.}, \mathbf{x} \rangle + [\beta]_r$

Properties:

- α and β define Ω (adaptive partitioning splines) thus we simply denote them $s[\alpha, \beta](\mathbf{x})$,
- always piecewise affine and globally convex (hence continuous),
- conversely

Theorem

Any $h \in \mathcal{C}^0(\mathbb{R}^D)$ that is piecewise affine and globally convex, $\exists \alpha, \beta$ s.t $h(\mathbf{x}) = s[\alpha, \beta](\mathbf{x})$.

Max-Affine Spline Operators (2/2)

Generalization to operators: max-affine spline operators (MASO) $S[A, B] : \mathbb{R}^D \to \mathbb{R}^K$:

$$S[A, B](\mathbf{x}) = \begin{bmatrix} \max_{r=1,\dots,R} & \langle [A]_{1,r,.}, \mathbf{x} \rangle + [B]_{1,r} \\ & \vdots \\ \max_{r=1,\dots,R} & \langle [A]_{K,r,.}, \mathbf{x} \rangle + [B]_{K,r} \end{bmatrix} =: A[\mathbf{x}]\mathbf{x} + B[\mathbf{x}]$$

where $A \in \mathbb{R}^{K \times R \times D}$ and $B \in \mathbb{R}^K \times \mathbb{R}^R$.

Max-Affine Spline Operators (2/2)

Generalization to operators: max-affine spline operators (MASO) $S[A,B]:\mathbb{R}^D \rightarrow \mathbb{R}^K$:

$$S[A,B](\mathbf{x}) = \begin{bmatrix} \max_{r=1,\dots,R} \quad \langle [A]_{1,r,.}, \mathbf{x}\rangle + [B]_{1,r} \\ \vdots \\ \max_{r=1,\dots,R} \quad \langle [A]_{K,r,.}, \mathbf{x}\rangle + [B]_{K,r} \end{bmatrix} =: A[\mathbf{x}]\mathbf{x} + B[\mathbf{x}]$$

where $A \in \mathbb{R}^{K \times R \times D}$ and $B \in \mathbb{R}^K \times \mathbb{R}^R$.

Theorem

Any operator $H(\mathbf{x}) = [h_1(\mathbf{x}), \dots, h_K(\mathbf{x})]^T$ with $\forall k, h_k \in \mathcal{C}^0(\mathbb{R}^D)$ piecewise affine and globally convex, $\exists A, B$ s.t $H(\mathbf{x}) = S[A,B](\mathbf{x})$.

Extension: a MASO can approximate arbitrarily closely any (nonlinear) operator that is convex in each output dimension.

Simplified Max-Affine Spline Operators

Use the same bias β' for all dimension:

$$S'[A,\beta'](\mathbf{x}) = \begin{bmatrix} \max_{r=1,\dots,R} & \langle [A]_{1,r,.}, (\mathbf{x}+\beta') \rangle \\ & \vdots \\ \max_{r=1,\dots,R} & \langle [A]_{K,r,.}, (\mathbf{x}+\beta') \rangle \end{bmatrix}$$

This simplified MASO is still sufficient to model most activation functions like ReLU, leaky-ReLU, absolute value; and linearly independent filters.

Deep networks and MASOs

Basic DN operators are MASOs!

- **fully connected** $f_W^{(\ell)}$: $S[A_W^{(\ell)}, B_W^{(\ell)}]$ where $R = 1, [A_W^{(\ell)}]_{k.1,.} = [W^{(\ell)}]_{k,.}$ and $[B_W^{(\ell)}]_{k,1} = [b_W^{(\ell)}]_k$,
- **convolutional** $f_C^{(\ell)}$: $S[A_C^{(\ell)}, B_C^{(\ell)}]$ where $R = 1, [A_C^{(\ell)}]_{k.1,.} = [C^{(\ell)}]_{k,.}$ and $[B_C^{(\ell)}]_{k,1} = [b_C^{(\ell)}]_k$,
- **activation** $f_\sigma^{(\ell)}$: $S[A_\sigma^{(\ell)}, B_\sigma^{(\ell)}]$ where $R = 2, [B_\sigma^{(\ell)}]_{k,1} = [B_\sigma^{(\ell)}]_{k,2} = 0 \quad \forall k$, and
  - ReLU: $[A_\sigma^{(\ell)}]_{k,1,.} = 0 \quad ; \quad [A_\sigma^{\ell}]_{k,2,.} = \mathbf{e}_k \ \forall k$,
  - leaky-ReLU: $[A_\sigma^{(\ell)}]_{k,1,.} = \nu \mathbf{e}_k \quad ; \quad [A_\sigma^{(\ell)}]_{k,2,.} = \mathbf{e}_k \ \forall k, \nu > 0$
  - absolute value: $[A_\sigma^{(\ell)}]_{k,1,.} = -\mathbf{e}_k \quad ; \quad [A_\sigma^{(\ell)}]_{k,2,.} = \mathbf{e}_k \ \forall k$

  ($\mathbf{e}_k$ is the $k-$th canonical basis element of $\mathbb{R}^{D^\ell}$)
- **pooling** $f_\rho^{(\ell)}$: $[A_\rho^{(\ell)}, B_\rho^{(\ell)}]$
  - max-pooling: $R = \#\mathcal{R}_k, [A_\rho^{(\ell)}]_{k,.,.} = \{\mathbf{e}_i, i \in \mathcal{R}_k\}$ and $[B_\rho^{(\ell)}]_{k,r} = 0, \forall k, r$
  - average-pooling: $R = 1, [A_\rho^{(\ell)}]_{k,1,.} = \frac{1}{\#\mathcal{R}_k} \sum_{i \in \mathcal{R}_k} \mathbf{e}_i$ and $[B_\rho^{(\ell)}]_{k,1} = 0, \forall k$

Composition of MASOs

Proposition

A DN layer constructed from an arbitrary composition of fully connected/convolution operators followed by one activation or pooling operator is a MASO $S[A^{(\ell)}, B^{(\ell)}]$ such that

$$f^{(\ell)}(\mathbf{z}^{(\ell-1)}) = A^{(\ell)}[\mathbf{x}]\mathbf{z}^{(\ell-1)}(\mathbf{x}) + B^{(\ell)}[\mathbf{x}].$$

Example: fully connected operator $S[A_W^{(\ell)}, B_W^{(\ell)}]$ followed by an activation operator $S[A_\sigma^{(\ell)}, B_\sigma^{(\ell)}]$ is a MASO $S[A^{(\ell)}, B^{(\ell)}]$ where $[A^{(\ell)}]_{k,r,.} = W^{(\ell)T}[A_\sigma^{(\ell)}]_{k,r,.}$ and $[B^{(\ell)}]_{k,r} = [B_\sigma^{(\ell)}]_{k,r} + b_W^{(\ell)T}[A_\sigma^{(\ell)}]_{k,r,.}$.

Composition of MASOs

Proposition

A DN layer constructed from an arbitrary composition of fully connected/convolution operators followed by one activation or pooling operator is a MASO $S[A^{(\ell)}, B^{(\ell)}]$ such that

$$f^{(\ell)}(\mathbf{z}^{(\ell-1)}) = A^{(\ell)}[\mathbf{x}]\mathbf{z}^{(\ell-1)}(\mathbf{x}) + B^{(\ell)}[\mathbf{x}].$$

Example: fully connected operator $S[A_W^{(\ell)}, B_W^{(\ell)}]$ followed by an activation operator $S[A_\sigma^{(\ell)}, B_\sigma^{(\ell)}]$ is a MASO $S[A^{(\ell)}, B^{(\ell)}]$ where $[A^{(\ell)}]_{k,r,.} = W^{(\ell)T}[A_\sigma^{(\ell)}]_{k,r,.}$ and $[B^{(\ell)}]_{k,r} = [B_\sigma^{(\ell)}]_{k,r} + b_W^{(\ell)T}[A_\sigma^{(\ell)}]_{k,r,.}$.

Theorem

A DN constructed from an arbitrary composition of fully connected/convolution, activation, and pooling operators of the previous types is a composition of MASOs. Moreover, the overall composition itself an ASO.

DN are Signal-Dependent Affine Transformations

Consequence of previous theorem:

the mapping from $\mathbf{x}$ to $\mathbf{z}^{(\ell)}(\mathbf{x})$ is an ASO,

DN are Signal-Dependent Affine Transformations

Consequence of previous theorem:

the mapping from $\mathbf{x}$ to $\mathbf{z}^{(\ell)}(\mathbf{x})$ is an ASO,

$\Leftrightarrow \mathbf{z}^{(\ell)}(\mathbf{x})$ is a signal-dependent, piecewise affine transformation of $\mathbf{x}$

DN are Signal-Dependent Affine Transformations

Consequence of previous theorem:

the mapping from $\mathbf{x}$ to $\mathbf{z}^{(\ell)}(\mathbf{x})$ is an ASO,

$\Leftrightarrow$ $\mathbf{z}^{(\ell)}(\mathbf{x})$ is a signal-dependent, piecewise affine transformation of $\mathbf{x}$

$\Leftrightarrow$ this particular affine mapping depends on which partition of the spline $\mathbf{x}$ falls in $\mathbb{R}^D$

DN are Signal-Dependent Affine Transformations

Consequence of previous theorem:

the mapping from $\mathbf{x}$ to $\mathbf{z}^{(\ell)}(\mathbf{x})$ is an ASO,

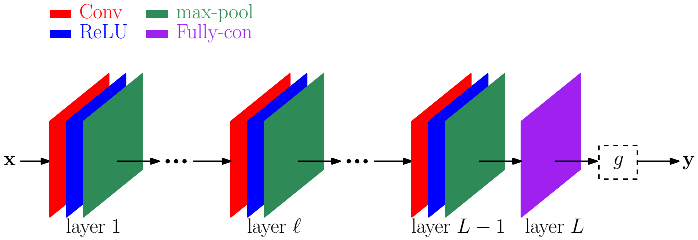
$\Leftrightarrow$ $\mathbf{z}^{(\ell)}(\mathbf{x})$ is a signal-dependent, piecewise affine transformation of $\mathbf{x}$

$\Leftrightarrow$ this particular affine mapping depends on which partition of the spline $\mathbf{x}$ falls in $\mathbb{R}^D$

(Note that in the case we use operators which are convex but not piecewise affine, we can show that such mapping can be approximated arbitrarily closely by a MASOs. In the case, the operators are not convex then the same result hold with the use of ASOs.)

Explicit input/ouput formula (1/2)

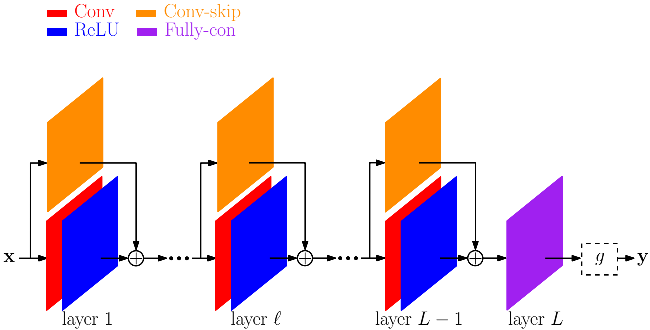
CNN affine mapping formula:



$$\mathbf{z}_{CNN}^{(L)}(\mathbf{x}) = W^{(L)} \left(\prod_{\ell=L-1}^{1} A_{\rho}^{(\ell)}[\mathbf{x}] A_{\sigma}^{(\ell)}[\mathbf{x}] C^{(\ell)} \right) \mathbf{x}$$
$$+ W^{(L)} \sum_{\ell=1}^{L-1} \left(\prod_{j=L-1}^{\ell+1} A_{\rho}^{(j)}[\mathbf{x}] A_{\sigma}^{(j)}[\mathbf{x}] C^{(j)} \right) \left(A_{\rho}^{(\ell)}[\mathbf{x}] A_{\sigma}^{(\ell)}[\mathbf{x}] b_{C}^{(\ell)} \right) + b_{W}^{(L)}$$

Explicit input/ouput formula (2/2)

ResNet affine mapping formula:



$$\mathbf{z}^{(L)}_{RES}(\mathbf{x}) = W^{(L)} \left(\prod_{\ell=L-1}^{1} \left(A_{\sigma}^{(\ell)}[\mathbf{x}] C^{(\ell)} + C_{skip}^{(\ell)} \right) \right) \mathbf{x}$$

$$+ W^{(L)} \sum_{\ell=1}^{L-1} \left(\prod_{j=L-1}^{\ell+1} \left(A_{\sigma}^{(j)}[\mathbf{x}] C^{(j)} + C_{skip}^{(\ell)} \right) \right) \left(A_{\sigma}^{(\ell)}[\mathbf{x}] b_{C}^{(\ell)} + b_{skip}^{(\ell)} \right) + b_{W}^{(L)}$$

Template Matching Machines (1/3)

The explicit formula can be rewritten in a general form:

$$\mathbf{z}^{(L)}(\mathbf{x}) = \left(W^{(L)}A[\mathbf{x}]\right)\mathbf{x} + \left(W^{(L)}B[\mathbf{x}] + b_W^{(L)}\right)$$

Template Matching Machines (1/3)

The explicit formula can be rewritten in a general form:

$$\mathbf{z}^{(L)}(\mathbf{x}) = \left(W^{(L)} A[\mathbf{x}]\right) \mathbf{x} + \left(W^{(L)} B[\mathbf{x}] + b_W^{(L)}\right)$$

- interpretation 1: $\mathbf{z}^{(L)}(\mathbf{x})$ is the output of bank of linear matched filters (i.e inner product of $\mathbf{x}$ with each row of $W^{(L)} A[\mathbf{x}]$) + set of biases (prior probability over the classes)

Template Matching Machines (1/3)

The explicit formula can be rewritten in a general form:

$$\mathbf{z}^{(L)}(\mathbf{x}) = \left(W^{(L)}A[\mathbf{x}]\right)\mathbf{x} + \left(W^{(L)}B[\mathbf{x}] + b_W^{(L)}\right)$$

- interpretation 1: $\mathbf{z}^{(L)}(\mathbf{x})$ is the output of bank of linear matched filters (i.e inner product of $\mathbf{x}$ with each row of $W^{(L)}A[\mathbf{x}]$) + set of biases (prior probability over the classes)
- interpretation 2: hierarchical matched filters:

$$\mathbf{z}^{(L)} = W^{(L)} \max_{r^{(L-1)}} \left(A^{(L-1)}_{r^{(L-1)}} \dots \max_{r^{(2)}} \left(A^{(2)}_{r^{(2)}} \max_{r^{(1)}} \left(A^{(1)}_{r^{(1)}}\mathbf{x} + B^{(1)}_{r^{(1)}}\right) + B^{(2)}_{r^{(2)}}\right) \dots + B^{(L-1)}_{r^{(L-1)}}\right) + b_W^{(L)}$$

Template Matching Machines (1/3)

The explicit formula can be rewritten in a general form:

$$\mathbf{z}^{(L)}(\mathbf{x}) = \left(W^{(L)}A[\mathbf{x}]\right)\mathbf{x} + \left(W^{(L)}B[\mathbf{x}] + b_W^{(L)}\right)$$

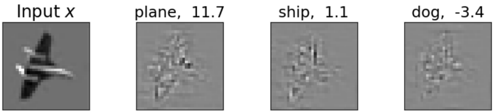
- interpretation 1: $\mathbf{z}^{(L)}(\mathbf{x})$ is the output of bank of linear matched filters (i.e inner product of $\mathbf{x}$ with each row of $W^{(L)}A[\mathbf{x}]$) + set of biases (prior probability over the classes)
- interpretation 2: hierarchical matched filters:

$$\mathbf{z}^{(L)} = W^{(L)} \max_{r^{(L-1)}} \left(A^{(L-1)}_{r^{(L-1)}} \dots \max_{r^{(2)}} \left(A^{(2)}_{r^{(2)}} \max_{r^{(1)}} \left(A^{(1)}_{r^{(1)}}\mathbf{x} + B^{(1)}_{r^{(1)}}\right) + B^{(2)}_{r^{(2)}}\right) \dots + B^{(L-1)}_{r^{(L-1)}}\right) + b_W^{(L)}$$

Visualization: extract (via the backpropagation algorithm) the templates

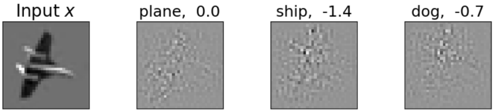
$$A[\mathbf{x}]_c = \frac{d[\mathbf{z}^{(L)}(\mathbf{x})]_c}{\mathbf{x}}$$

Template Matching Machines (2/3)

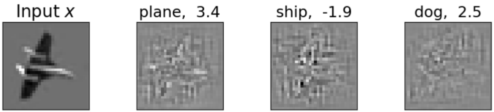


(a) *largeCNN*, ReLU activation, no BN

(b) *largeCNN*, absolute value activation, no BN

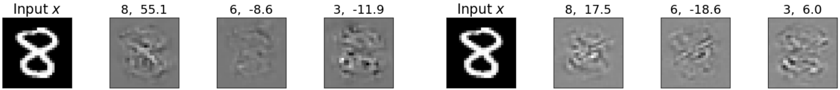


(c) *largeCNN*, ReLU activation, BN

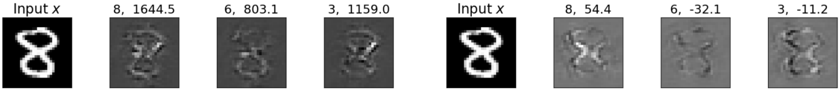


(d) *largeCNN*, absolute value activation, BN

Template Matching Machines (3/3)



(a) *smallResNet*, ReLU activation, no BN (b) *smallResNet*, ReLU activation, BN



(c) *smallResNet*, absolute value activation, no BN (d) *smallResNet*, absolute value activation, BN

See Balestriero and Baraniuk, “Mad Max: Affine Spline Insights into Deep Learning”

A simple way of boosting DN performances

It is known that a matched filterbank is optimal to classify signals immersed in additive white Gaussian noise $\rightarrow$ not realistic in practice.

A simple way of boosting DN performances

It is known that a matched filterbank is optimal to classify signals immersed in additive white Gaussian noise $\rightarrow$ not realistic in practice.

Eldar & Oppenheim proposed to use orthogonal templates $\rightarrow$ can be achieved in DN by penalizing non-zero off-diagonal entries in $W^{(L)}(W^{(L)})^T$ via the learning objective function:

$$\mathcal{L}_{CE} + \lambda \sum_{c_1 \neq c_2} \left| \left\langle [W^{(L)}]_{c_1,.}, [W^{(L)}]_{c_2,.} \right\rangle \right|^2$$

A simple way of boosting DN performances

It is known that a matched filterbank is optimal to classify signals immersed in additive white Gaussian noise $\rightarrow$ not realistic in practice.

Eldar & Oppenheim proposed to use orthogonal templates $\rightarrow$ can be achieved in DN by penalizing non-zero off-diagonal entries in $W^{(L)}(W^{(L)})^T$ via the learning objective function:

$$\mathcal{L}_{CE} + \lambda \sum_{c_1 \neq c_2} \left| \left\langle [W^{(L)}]_{c_1,.}, [W^{(L)}]_{c_2,.} \right\rangle \right|^2$$

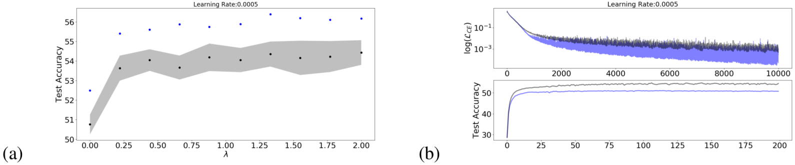
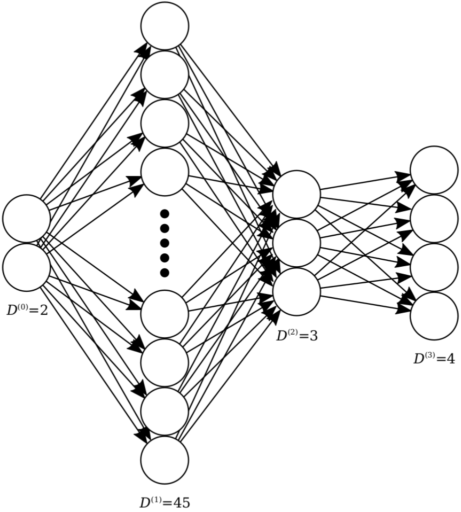
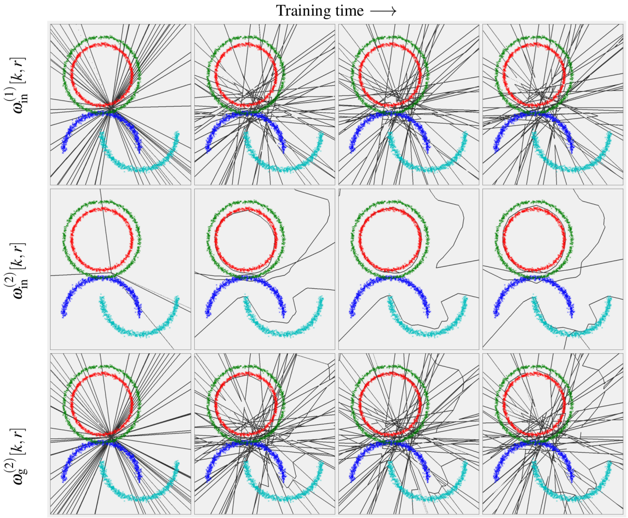


FIG. 5. Orthogonal templates significantly boost DN performance with *no change to the architecture*. (a) Classification performance of the *largeCNN* trained on CIFAR100 for different values of the orthogonality penalty λ in (5.11). We plot the average (back dots), standard deviation (gray shade), and maximum (blue dots) of the test set accuracy over 15 runs. (b, top) Training set error. The blue/black curves corresponds to $\lambda = 0/1$. (b, bottom) Test set accuracy over the course of the learning.

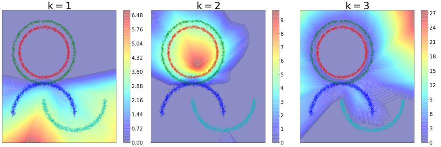
On the geometry of the partitioning



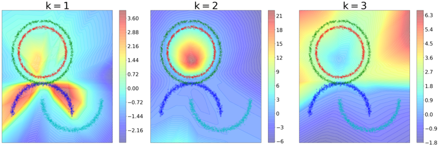
On the geometry of the partitioning



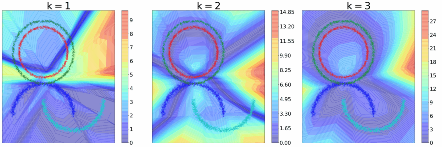
On the geometry of the partitioning



(a) ReLU



(b) Leaky ReLU



(c) Absolute value

Conclusion - Perspectives

- MASO appear to be a well-adapted tool to study DN
- Different point of views (operator, template matched filters,. . .)
- Universality of MASO DNs (i.e approximation theory)
- Stability and Lipschitz constant
- Colinear template and Data set memorization
- Multiscale partitions
- Connections with Vector Quantization (information theory), K-means (statistics) and Voronoi tiling (geometry)

- new constraints on the templates
- further study of the case of non-convex activation functions
- link with wavelet theory
- approximation theory (smoothness spaces of decision boundaries, optimal approximation)

Randall Balestriero, Richard Baraniuk, “Mad Max: Affine Spline Insights into Deep Learning,”

arxiv.org/abs/1805.06576, 2018