# Expressiveness in Deep Learning via Quantum Entanglement

**Nadav Cohen**

Tel Aviv University

*From Quantum Computing to Quantum Chemistry:*
*Theory, Platforms, and Practical Applications (TAU, CECAM, ISF)*

15 September 2019

# Sources

**Deep SimNets**
*C* + Sharir + Shashua
*Computer Vision and Pattern Recognition (CVPR) 2016*

**On the Expressive Power of Deep Learning: A Tensor Analysis**
*C* + Sharir + Shashua
*Conference on Learning Theory (COLT) 2016*

**Convolutional Rectifier Networks as Generalized Tensor Decompositions**
*C* + Shashua
*International Conference on Machine Learning (ICML) 2016*

**Inductive Bias of Deep Convolutional Networks through Pooling Geometry**
*C* + Shashua
*International Conference on Learning Representations (ICLR) 2017*

**Boosting Dilated Convolutional Networks with Mixed Tensor Decompositions**
*C* + Tamari + Shashua
*International Conference on Learning Representations (ICLR) 2018*

**Deep Learning and Quantum Entanglement:**
**Fundamental Connections with Implications to Network Design**
Levine + Yakira + *C* + Shashua
*International Conference on Learning Representations (ICLR) 2018*

**Quantum Entanglement in Deep Learning Architectures**
Levine + Sharir + *C* + Shashua
*Physical Review Letters (PRL) 2019*

# Collaborators

Yoav Levine

Or Sharir

Amnon Shashua

Ronen Tamari

David Yakira

# EVERY INDUSTRY WANTS DEEP LEARNING

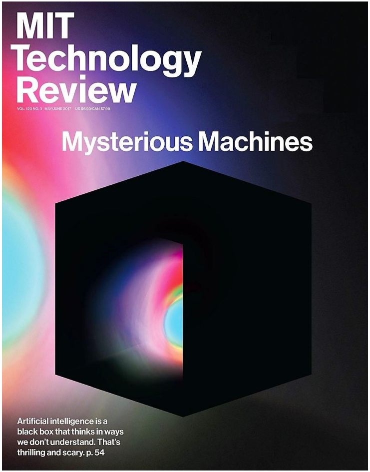
| Cloud Service Provider | Medicine | Media & Entertainment | Security & Defense | Autonomous Machines |
|---|---|---|---|---|

| | | | | |
|---|---|---|---|---|
| ➢ Image/Video classification | ➢ Cancer cell detection | ➢ Video captioning | ➢ Face recognition | ➢ Pedestrian detection |
| ➢ Speech recognition | ➢ Diabetic grading | ➢ Content based search | ➢ Video surveillance | ➢ Lane tracking |
| ➢ Natural language processing | ➢ Drug discovery | ➢ Real time translation | ➢ Cyber security | ➢ Recognize traffic sign |

Source

NVIDIA (`www.slideshare.net/openomics/the-revolution-of-deep-learning`)

# Limited Formal Understanding




**Intelligent Machines**

## The Dark Secret at the Heart of AI

No one really knows how the most advanced algorithms do what they do. That could be a problem.

by Will Knight April 11, 2017

**Last year, a strange self-driving car was released onto the quiet** roads of Monmouth County, New Jersey. The experimental vehicle, developed by researchers at the chip maker Nvidia, didn't look different from other autonomous cars, but it was unlike anything demonstrated by Google, Tesla, or General Motors, and it showed the rising power of artificial intelligence. The car didn't follow a single instruction provided by an engineer or programmer. Instead, it relied entirely on an algorithm that had taught itself to drive by watching a human do it.

# Outline

1. Deep Learning Theory: Expressiveness, Generalization and Optimization
2. Convolutional Networks as Tensor Networks
3. Expressiveness of Convolutional Networks
   - Dependencies as Quantum Entanglement
   - Analysis of Supported Entanglement
4. Extensions
5. Conclusion

# Statistical Learning Setup

# Statistical Learning Setup

$\mathcal{X}$ — instance space (e.g. $\mathbb{R}^{100\times100}$ for 100-by-100 grayscale images)

# Statistical Learning Setup

$\mathcal{X}$ — instance space (e.g. $\mathbb{R}^{100\times100}$ for 100-by-100 grayscale images)

$\mathcal{Y}$ — label space (e.g. $\mathbb{R}$ for regression or $\{1,\ldots,k\}$ for classification)

# Statistical Learning Setup

$\mathcal{X}$ — instance space (e.g. $\mathbb{R}^{100\times100}$ for 100-by-100 grayscale images)

$\mathcal{Y}$ — label space (e.g. $\mathbb{R}$ for regression or $\{1,\ldots,k\}$ for classification)

$\mathcal{D}$ — distribution over $\mathcal{X}\times\mathcal{Y}$ (unknown)

# Statistical Learning Setup

$\mathcal{X}$ — instance space (e.g. $\mathbb{R}^{100\times100}$ for 100-by-100 grayscale images)

$\mathcal{Y}$ — label space (e.g. $\mathbb{R}$ for regression or $\{1,\ldots,k\}$ for classification)

$\mathcal{D}$ — distribution over $\mathcal{X}\times\mathcal{Y}$ (unknown)

$\ell:\mathcal{Y}\times\mathcal{Y}\to\mathbb{R}_{\geq 0}$ — loss func (e.g. $\ell(y,\hat{y})=(y-\hat{y})^2$ for $\mathcal{Y}=\mathbb{R}$)

# Statistical Learning Setup

$\mathcal{X}$ — instance space (e.g. $\mathbb{R}^{100\times100}$ for 100-by-100 grayscale images)

$\mathcal{Y}$ — label space (e.g. $\mathbb{R}$ for regression or $\{1,\ldots,k\}$ for classification)

$\mathcal{D}$ — distribution over $\mathcal{X}\times\mathcal{Y}$ (unknown)

$\ell:\mathcal{Y}\times\mathcal{Y}\to\mathbb{R}_{\geq0}$ — loss func (e.g. $\ell(y,\hat{y})=(y-\hat{y})^2$ for $\mathcal{Y}=\mathbb{R}$)

**Task**

Given training set $S=\{(X_i,y_i)\}_{i=1}^m$ drawn i.i.d. from $\mathcal{D}$, return hypothesis (predictor) $h:\mathcal{X}\to\mathcal{Y}$ that minimizes population loss:

$$L_{\mathcal{D}}(h):=\mathbb{E}_{(X,y)\sim\mathcal{D}}[\ell(y,h(X))]$$

# Statistical Learning Setup

$\mathcal{X}$ — instance space (e.g. $\mathbb{R}^{100\times 100}$ for 100-by-100 grayscale images)

$\mathcal{Y}$ — label space (e.g. $\mathbb{R}$ for regression or $\{1,\ldots,k\}$ for classification)

$\mathcal{D}$ — distribution over $\mathcal{X} \times \mathcal{Y}$ (unknown)

$\ell : \mathcal{Y}\times\mathcal{Y} \to \mathbb{R}_{\geq 0}$ — loss func (e.g. $\ell(y,\hat{y}) = (y-\hat{y})^2$ for $\mathcal{Y} = \mathbb{R}$)

**Task**

Given training set $S = \{(X_i, y_i)\}_{i=1}^m$ drawn i.i.d. from $\mathcal{D}$, return hypothesis (predictor) $h : \mathcal{X} \to \mathcal{Y}$ that minimizes population loss:

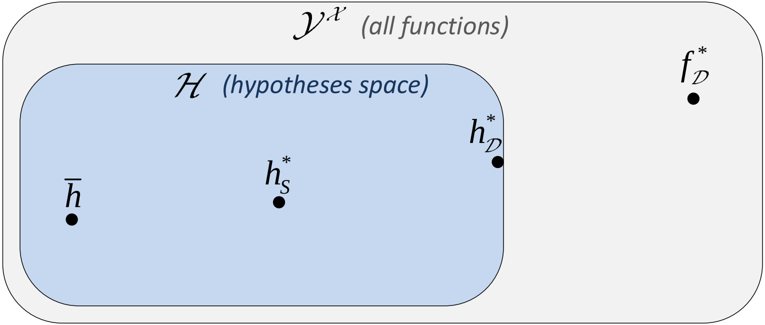
$$L_{\mathcal{D}}(h) := \mathbb{E}_{(X,y)\sim\mathcal{D}}[\ell(y, h(X))]$$

**Approach**

Predetermine hypotheses space $\mathcal{H} \subset \mathcal{Y}^{\mathcal{X}}$, and return hypothesis $h \in \mathcal{H}$ that minimizes empirical loss:

$$L_S(h) := \frac{1}{m}\sum\nolimits_{i=1}^{m} \ell(y_i, h(X_i))$$

# Three Pillars of Statistical Learning Theory: Expressiveness, Generalization and Optimization



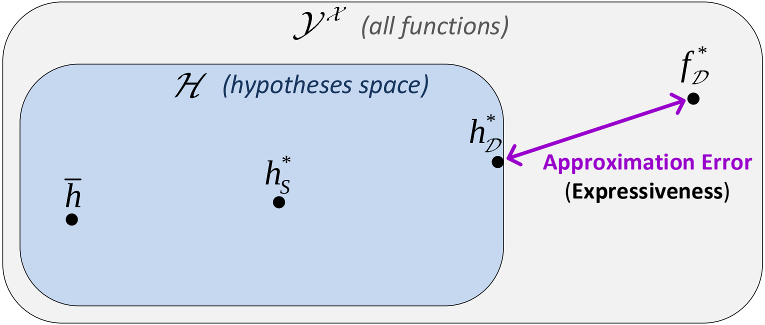

$f^*_{\mathcal{D}}$ — ground truth (minimizer of population loss over all func)

$h^*_{\mathcal{D}}$ — optimal hypothesis (minimizer of population loss over $\mathcal{H}$)

$h^*_S$ — empirically optimal hypothesis (minimizer of empirical loss over $\mathcal{H}$)

$\bar{h}$ — returned hypothesis

# Three Pillars of Statistical Learning Theory: Expressiveness, Generalization and Optimization



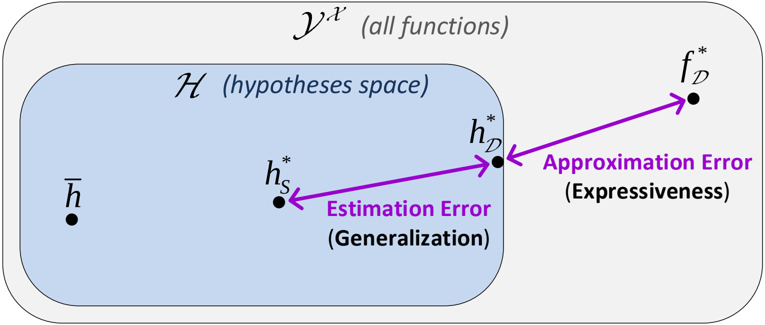

$f^*_{\mathcal{D}}$ — ground truth (minimizer of population loss over all func)

$h^*_{\mathcal{D}}$ — optimal hypothesis (minimizer of population loss over $\mathcal{H}$)

$h^*_S$ — empirically optimal hypothesis (minimizer of empirical loss over $\mathcal{H}$)

$\bar{h}$ — returned hypothesis

# Three Pillars of Statistical Learning Theory: Expressiveness, Generalization and Optimization



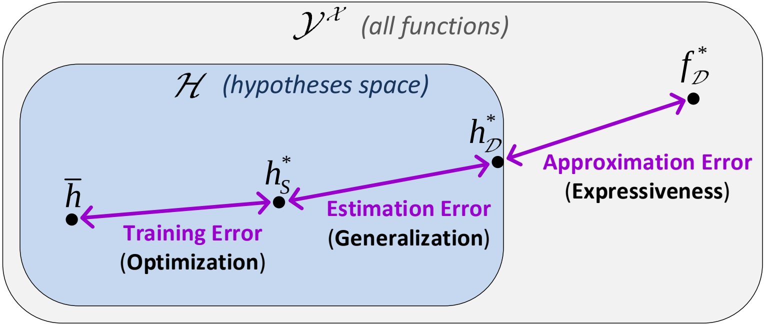

$f^*_{\mathcal{D}}$ — ground truth (minimizer of population loss over all func)

$h^*_{\mathcal{D}}$ — optimal hypothesis (minimizer of population loss over $\mathcal{H}$)

$h^*_S$ — empirically optimal hypothesis (minimizer of empirical loss over $\mathcal{H}$)

$\bar{h}$ — returned hypothesis

# Three Pillars of Statistical Learning Theory: Expressiveness, Generalization and Optimization




$f^*_{\mathcal{D}}$ — ground truth (minimizer of population loss over all func)

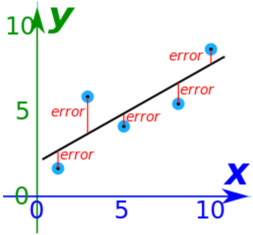
$h^*_{\mathcal{D}}$ — optimal hypothesis (minimizer of population loss over $\mathcal{H}$)

$h^*_S$ — empirically optimal hypothesis (minimizer of empirical loss over $\mathcal{H}$)

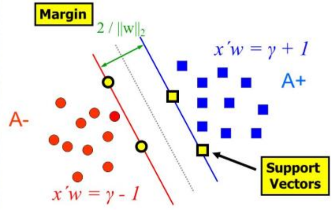
$\bar{h}$ — returned hypothesis

# Classical Machine Learning

- Euclidean instance/label spaces: $\mathcal{X} = \mathbb{R}^d$, $\mathcal{Y} = \mathbb{R}^k$
- Linear hypotheses space: $\mathcal{H} = \{\mathbf{x} \mapsto W\mathbf{x} : W \in \mathbb{R}^{k,d}\}$

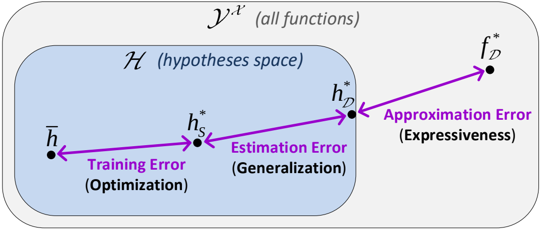



*Least Squares Regression*

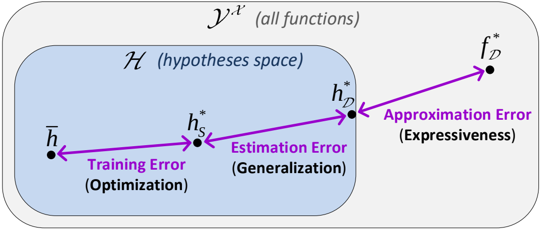



*Support Vector Machine*

# Classical Machine Learning – Three Pillars



# Classical Machine Learning – Three Pillars

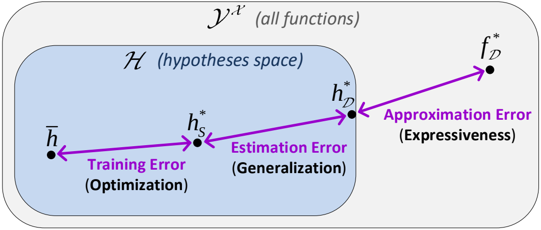



## Optimization

Empirical loss minimization is a convex program:

$$\bar{h} \approx h^*_S \quad (\text{ training err} \approx 0 \text{ })$$

# Classical Machine Learning – Three Pillars




## Optimization

Empirical loss minimization is a convex program:

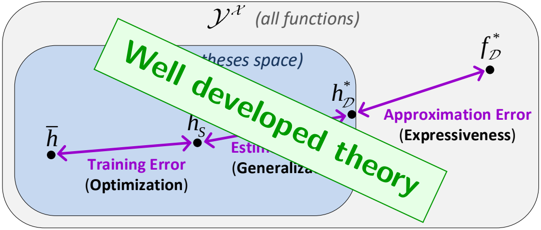
$$\bar{h} \approx h^*_S \quad (\text{ training err} \approx 0\ )$$

## Expressiveness & Generalization

Bias-variance trade-off:

| $\mathcal{H}$ | approximation err | estimation err |
|---|---|---|
| *expands* | $\searrow$ | $\nearrow$ |
| *shrinks* | $\nearrow$ | $\searrow$ |

# Classical Machine Learning – Three Pillars




## Optimization

Empirical loss minimization is a convex program:

$$\bar{h} \approx h^*_S \quad (\text{ training err} \approx 0 \text{ })$$

## Expressiveness & Generalization

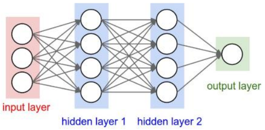
Bias-variance trade-off:

| $\mathcal{H}$ | approximation err | estimation err |
|---|---|---|
| *expands* | $\searrow$ | $\nearrow$ |
| *shrinks* | $\nearrow$ | $\searrow$ |

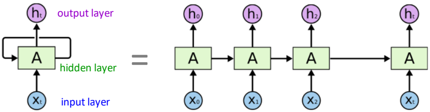
# Deep Learning

- Euclidean instance/label spaces
- Composite (non-linear) hypotheses space

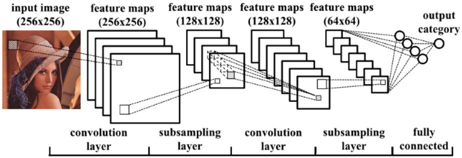
**Fully-Connected Networks**




**Recurrent Networks**



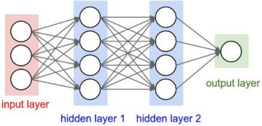

**Convolutional Networks**



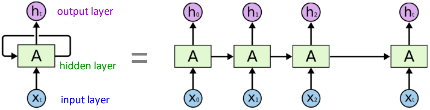
# Deep Learning

- Euclidean instance/label spaces
- Composite (non-linear) hypotheses space

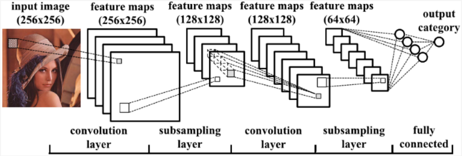
***Fully-Connected Networks***



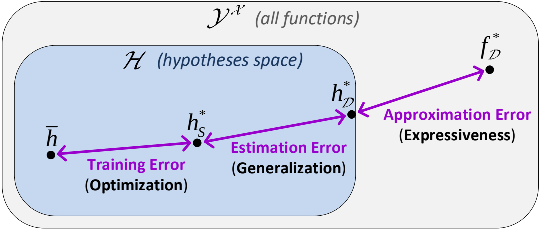

***Recurrent Networks***



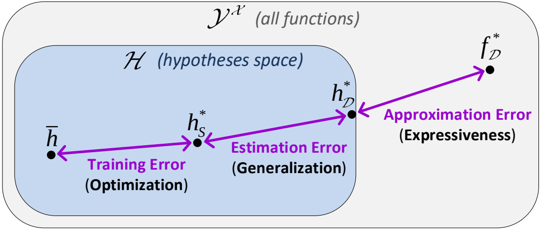

***Convolutional Networks***



# Deep Learning – Three Pillars



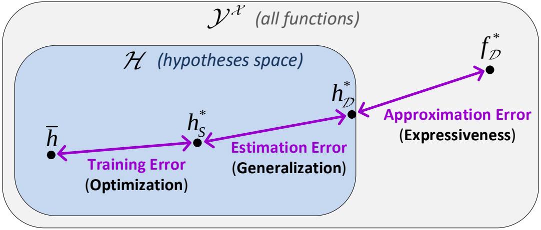
# Deep Learning – Three Pillars




## Optimization

Empirical loss minimization is a non-convex program:

# Deep Learning – Three Pillars

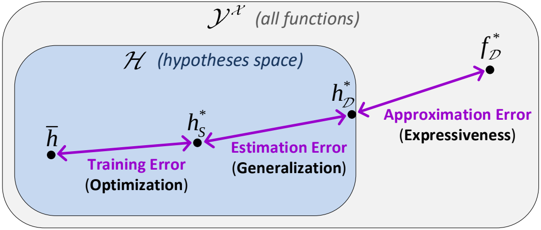



## Optimization

Empirical loss minimization is a non-convex program:

- $h^*_S$ is not unique — many hypotheses have low training err

# Deep Learning – Three Pillars

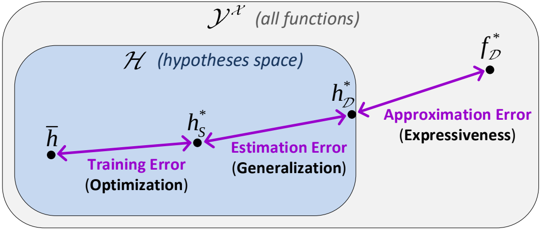



## Optimization

Empirical loss minimization is a non-convex program:

- $h^*_S$ is not unique — many hypotheses have low training err
- Gradient descent (GD) somehow reaches one of these

# Deep Learning – Three Pillars




## Optimization

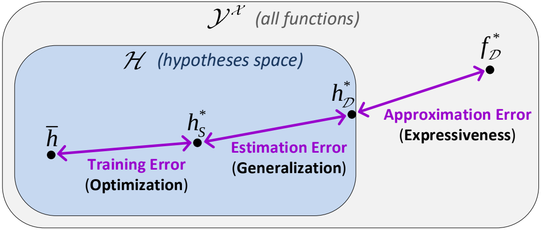
Empirical loss minimization is a non-convex program:

- $h^*_S$ is not unique — many hypotheses have low training err
- Gradient descent (GD) somehow reaches one of these

## Expressiveness & Generalization

Vast difference from classical ML:

# Deep Learning – Three Pillars




## Optimization

Empirical loss minimization is a non-convex program:

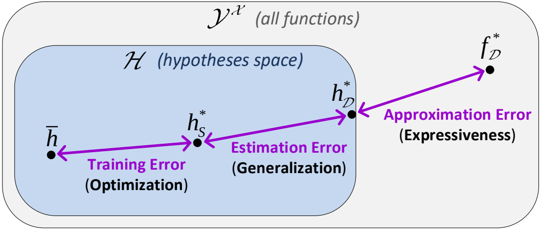
- $h^*_S$ is not unique — many hypotheses have low training err
- Gradient descent (GD) somehow reaches one of these

## Expressiveness & Generalization

Vast difference from classical ML:

- Some low training err hypotheses generalize well, others don't

# Deep Learning – Three Pillars




## Optimization

Empirical loss minimization is a non-convex program:

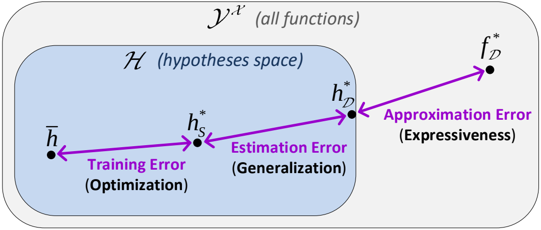
- $h^*_S$ is not unique — many hypotheses have low training err
- Gradient descent (GD) somehow reaches one of these

## Expressiveness & Generalization

Vast difference from classical ML:

- Some low training err hypotheses generalize well, others don't
- W/typical data, solution returned by GD often generalizes well

# Deep Learning – Three Pillars




## Optimization

Empirical loss minimization is a non-convex program:

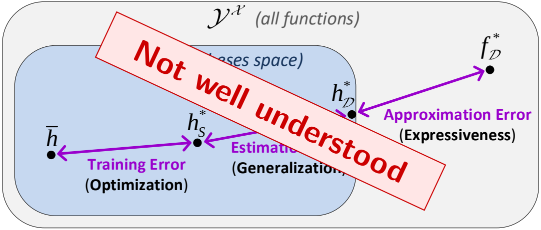
- $h^*_S$ is not unique — many hypotheses have low training err
- Gradient descent (GD) somehow reaches one of these

## Expressiveness & Generalization

Vast difference from classical ML:

- Some low training err hypotheses generalize well, others don't
- W/typical data, solution returned by GD often generalizes well
- Expanding $\mathcal{H}$ reduces approximation err, but also estimation err!

# Deep Learning – Three Pillars




## Optimization

Empirical loss minimization is a non-convex program:

- $h_S^*$ is not unique — many hypotheses have low training err
- Gradient descent (GD) somehow reaches one of these

## Expressiveness & Generalization

Vast difference from classical ML:

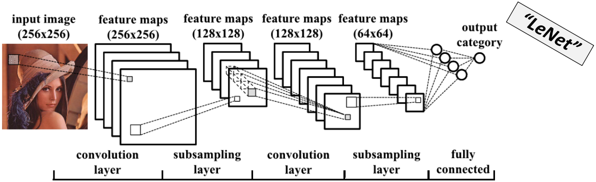
- Some low training err hypotheses generalize well, others don't
- W/typical data, solution returned by GD often generalizes well
- Expanding $\mathcal{H}$ reduces approximation err, but also estimation err!

# Outline

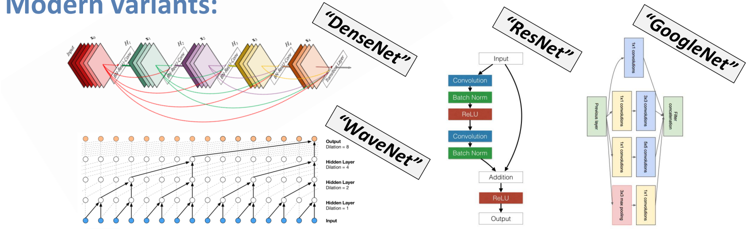
# Convolutional Networks

Most successful deep learning arch to date!

## Classic structure:




## Modern variants:




Traditionally used for images/video, nowadays for audio/text as well

# Coefficient Tensor

ConvNets realize func over many local elements (e.g. pixels)

# Coefficient Tensor

ConvNets realize func over many local elements (e.g. pixels)

Let $\mathbf{H} = span\{f_i(\mathbf{x})\}_{i=1}^{M}$ be Hilbert space of func over single element

# Coefficient Tensor

ConvNets realize func over many local elements (e.g. pixels)

Let $\mathbf{H} = span\{f_i(\mathbf{x})\}_{i=1}^{M}$ be Hilbert space of func over single element

Tensor product $\mathbf{H}^{\otimes N}$ is then Hilbert space of func over $N$ elements

# Coefficient Tensor

ConvNets realize func over many local elements (e.g. pixels)

Let $\mathbf{H} = span\{f_i(\mathbf{x})\}_{i=1}^{M}$ be Hilbert space of func over single element

Tensor product $\mathbf{H}^{\otimes N}$ is then Hilbert space of func over $N$ elements

Any $h(\cdot) \in \mathbf{H}^{\otimes N}$ can be written as:

$$h(\mathbf{x}_1, \ldots, \mathbf{x}_N) = \sum_{d_1 \ldots d_N = 1}^{M} \mathcal{A}_{d_1 \ldots d_N} \prod_{i=1}^{N} f_{d_i}(\mathbf{x}_i) = \langle \mathcal{A} \,|\, \mathcal{F}(\mathbf{x}_1, \ldots, \mathbf{x}_N) \rangle$$

where:

# Coefficient Tensor

ConvNets realize func over many local elements (e.g. pixels)

Let $\mathbf{H} = span\{f_i(\mathbf{x})\}_{i=1}^{M}$ be Hilbert space of func over single element

Tensor product $\mathbf{H}^{\otimes N}$ is then Hilbert space of func over $N$ elements

Any $h(\cdot) \in \mathbf{H}^{\otimes N}$ can be written as:

$$h(\mathbf{x}_1, \ldots, \mathbf{x}_N) = \sum_{d_1 \ldots d_N = 1}^{M} \mathcal{A}_{d_1 \ldots d_N} \prod_{i=1}^{N} f_{d_i}(\mathbf{x}_i) = \langle \mathcal{A} \,|\, \mathcal{F}(\mathbf{x}_1, \ldots, \mathbf{x}_N) \rangle$$

where:

- $\mathcal{F}(\mathbf{x}_1, \ldots, \mathbf{x}_N)$ – product (rank-1) tensor, depends only on input

# Coefficient Tensor

ConvNets realize func over many local elements (e.g. pixels)

Let $\mathbf{H} = span\{f_i(\mathbf{x})\}_{i=1}^{M}$ be Hilbert space of func over single element

Tensor product $\mathbf{H}^{\otimes N}$ is then Hilbert space of func over $N$ elements

Any $h(\cdot) \in \mathbf{H}^{\otimes N}$ can be written as:

$$h(\mathbf{x}_1, \ldots, \mathbf{x}_N) = \sum_{d_1 \ldots d_N = 1}^{M} \mathcal{A}_{d_1 \ldots d_N} \prod_{i=1}^{N} f_{d_i}(\mathbf{x}_i) = \langle \mathcal{A} \,|\, \mathcal{F}(\mathbf{x}_1, \ldots, \mathbf{x}_N) \rangle$$

where:

- $\mathcal{F}(\mathbf{x}_1, \ldots, \mathbf{x}_N)$ – product (rank-1) tensor, depends only on input

$$\left( \mathcal{F}(\mathbf{x}_1, \ldots, \mathbf{x}_N) := \mathbf{f}(\mathbf{x}_1) \otimes \cdots \otimes \mathbf{f}(\mathbf{x}_N) \;,\; \mathbf{f}(\mathbf{x}_i) := [f_1(\mathbf{x}_i), \ldots, f_M(\mathbf{x}_i)]^\top \right)$$

# Coefficient Tensor

ConvNets realize func over many local elements (e.g. pixels)

Let $\mathbf{H} = span\{f_i(\mathbf{x})\}_{i=1}^{M}$ be Hilbert space of func over single element

Tensor product $\mathbf{H}^{\otimes N}$ is then Hilbert space of func over $N$ elements

Any $h(\cdot) \in \mathbf{H}^{\otimes N}$ can be written as:

$$h(\mathbf{x}_1, \ldots, \mathbf{x}_N) = \sum_{d_1 \ldots d_N = 1}^{M} \mathcal{A}_{d_1 \ldots d_N} \prod_{i=1}^{N} f_{d_i}(\mathbf{x}_i) = \langle \mathcal{A} \,|\, \mathcal{F}(\mathbf{x}_1, \ldots, \mathbf{x}_N) \rangle$$

where:

- $\mathcal{F}(\mathbf{x}_1, \ldots, \mathbf{x}_N)$ – product (rank-1) tensor, depends only on input

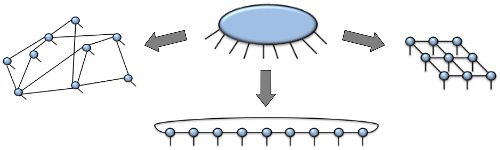
  $\left( \mathcal{F}(\mathbf{x}_1, \ldots, \mathbf{x}_N) := \mathbf{f}(\mathbf{x}_1) \otimes \cdots \otimes \mathbf{f}(\mathbf{x}_N) \;,\; \mathbf{f}(\mathbf{x}_i) := [f_1(\mathbf{x}_i), \ldots, f_M(\mathbf{x}_i)]^\top \right)$

- $\mathcal{A}$ – **coefficient tensor**, fully determines func $h(\cdot)$

# Tensor Networks

In quantum physics, high-order tensors are simulated via:

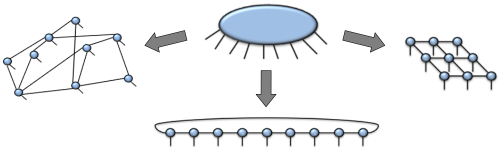
**Tensor Networks**



# Tensor Networks

In quantum physics, high-order tensors are simulated via:

**Tensor Networks**



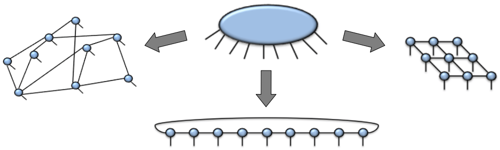
Tensor Networks (TN):

- Graphs in which: vertices $\longleftrightarrow$ tensors edges $\longleftrightarrow$ modes

# Tensor Networks

In quantum physics, high-order tensors are simulated via:

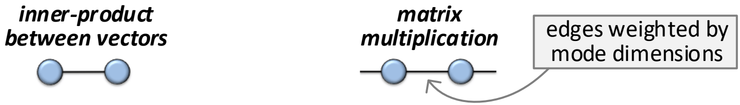
**Tensor Networks**



Tensor Networks (TN):

- Graphs in which: vertices $\longleftrightarrow$ tensors edges $\longleftrightarrow$ modes


- Edge (mode) connecting two vertices (tensors) represents contraction



# Tree Tensor Network $\longrightarrow$ Convolutional Arithmetic Circuit

$$h(\mathbf{x}_1, \ldots, \mathbf{x}_N) = \Big\langle \underbrace{\mathcal{A}}_{\text{coeff tensor}} \Big| \underbrace{\mathcal{F}(\mathbf{x}_1, \ldots, \mathbf{x}_N)}_{\text{input product tensor}} \Big\rangle$$

Coeff tensor $\mathcal{A}$ is exponential (in # of input elements $N$)

$\Longrightarrow$ directly computing general func is intractable

# Tree Tensor Network $\longrightarrow$ Convolutional Arithmetic Circuit

$$h(\mathbf{x}_1, \ldots, \mathbf{x}_N) = \Big\langle \underbrace{\mathcal{A}}_{\text{coeff tensor}} \Big| \underbrace{\mathcal{F}(\mathbf{x}_1, \ldots, \mathbf{x}_N)}_{\text{input product tensor}} \Big\rangle$$

Coeff tensor $\mathcal{A}$ is exponential (in # of input elements $N$)

$\Longrightarrow$ directly computing general func is intractable

**Observation**

Decomposing coeff tensor w/**tree TN** gives ConvNet w/linear activation and product pooling – **Convolutional Arithmetic Circuit** (ConvAC)!

# Tree Tensor Network $\longrightarrow$ Convolutional Arithmetic Circuit

$$h(\mathbf{x}_1, \ldots, \mathbf{x}_N) = \Big\langle \underbrace{\mathcal{A}}_{\text{coeff tensor}} \Big| \underbrace{\mathcal{F}(\mathbf{x}_1, \ldots, \mathbf{x}_N)}_{\text{input product tensor}} \Big\rangle$$

Coeff tensor $\mathcal{A}$ is exponential (in # of input elements $N$)

$\Longrightarrow$ directly computing general func is intractable

**Observation**

Decomposing coeff tensor w/**tree TN** gives ConvNet w/linear activation and product pooling – **Convolutional Arithmetic Circuit** (ConvAC)!

TN topology $\longleftrightarrow$ ConvAC arch

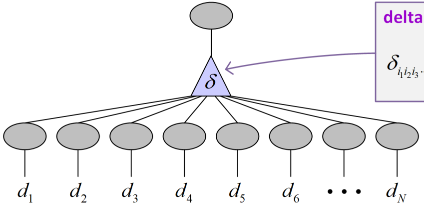
TN tensors $\longleftrightarrow$ ConvAC weights

# Example 1: Shallow Model

$$h(\mathbf{x}_1, \ldots, \mathbf{x}_N) = \Big\langle \underbrace{\mathcal{A}}_{\text{coeff tensor}} \Big| \underbrace{\mathcal{F}(\mathbf{x}_1, \ldots, \mathbf{x}_N)}_{\text{input product tensor}} \Big\rangle$$

W/star TN applied to coeff tensor:

$\mathcal{A} \;=$




**delta tensor**:

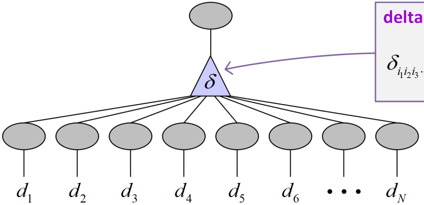
$$\delta_{i_1 i_2 i_3 \cdots} = \begin{cases} 1 & , i_1 = i_2 = i_3 = \cdots \\ 0 & , \textit{otherwise} \end{cases}$$

# Example 1: Shallow Model

$$h(\mathbf{x}_1,\ldots,\mathbf{x}_N) = \left\langle \underbrace{\mathcal{A}}_{\text{coeff tensor}} \,\middle|\, \underbrace{\mathcal{F}(\mathbf{x}_1,\ldots,\mathbf{x}_N)}_{\text{input product tensor}} \right\rangle$$

W/star TN applied to coeff tensor:

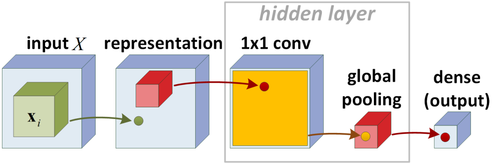
$\mathcal{A}$ =




**delta tensor**:

$$\delta_{i_1 i_2 i_3 \cdots} = \begin{cases} 1 & , i_1 = i_2 = i_3 = \cdots \\ 0 & , \textit{otherwise} \end{cases}$$

func is computed by shallow ConvAC (single hidden layer, global pooling):

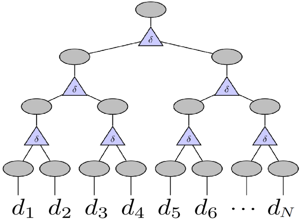


# Example 2: Deep Model

$$h(\mathbf{x}_1, \ldots, \mathbf{x}_N) = \left\langle \underbrace{\mathcal{A}}_{\text{coeff tensor}} \middle| \underbrace{\mathcal{F}(\mathbf{x}_1, \ldots, \mathbf{x}_N)}_{\text{input product tensor}} \right\rangle$$

W/binary tree TN applied to coeff tensor:

$\mathcal{A}$ =

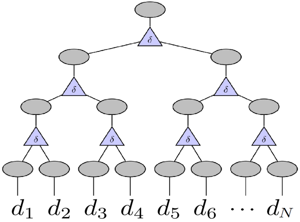


# Example 2: Deep Model

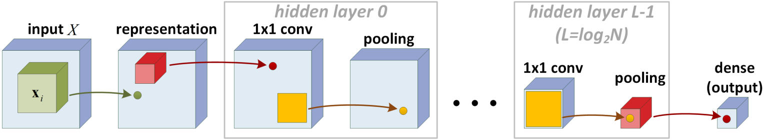
$$h(\mathbf{x}_1,\ldots,\mathbf{x}_N) = \Big\langle \underbrace{\mathcal{A}}_{\text{coeff tensor}} \Big| \underbrace{\mathcal{F}(\mathbf{x}_1,\ldots,\mathbf{x}_N)}_{\text{input product tensor}} \Big\rangle$$

W/binary tree TN applied to coeff tensor:

$\mathcal{A} \quad =$




func is computed by deep ConvAC (size-2 pooling windows):

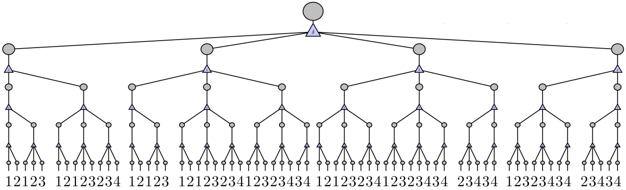


# Example 3: Deep Model with Overlaps

$$h(\mathbf{x}_1, \ldots, \mathbf{x}_N) = \Big\langle \underbrace{\mathcal{A}}_{\text{coeff tensor}} \Big| \underbrace{\mathcal{F}(\mathbf{x}_1, \ldots, \mathbf{x}_N)}_{\text{input product tensor}} \Big\rangle$$

W/"duplicated" tree TN applied to coeff tensor:

$\mathcal{A} \quad =$

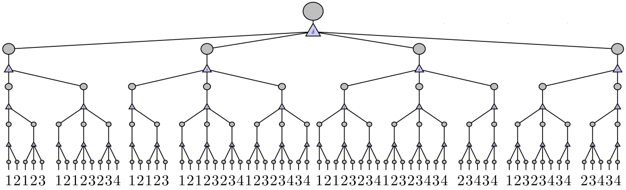


# Example 3: Deep Model with Overlaps

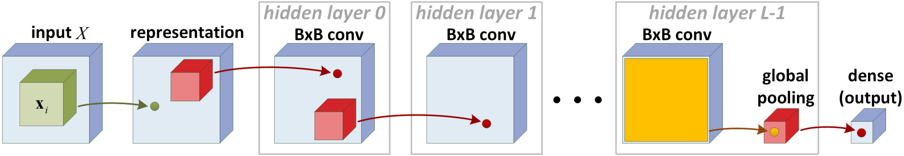
$$h(\mathbf{x}_1,\ldots,\mathbf{x}_N) = \Big\langle \underbrace{\mathcal{A}}_{\text{coeff tensor}} \Big| \underbrace{\mathcal{F}(\mathbf{x}_1,\ldots,\mathbf{x}_N)}_{\text{input product tensor}} \Big\rangle$$

W/"duplicated" tree TN applied to coeff tensor:

$\mathcal{A}$ =

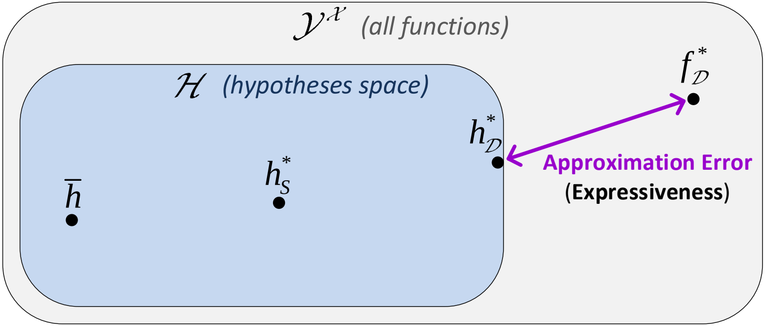



func is computed by deep ConvAC w/overlaps:



# Outline

# Expressiveness




$f^*_{\mathcal{D}}$ – ground truth (minimizer of population loss over all func)

$h^*_{\mathcal{D}}$ – optimal hypothesis (minimizer of population loss over $\mathcal{H}$)

$h^*_S$ – empirically optimal hypothesis (minimizer of empirical loss over $\mathcal{H}$)

$\bar{h}$ – returned hypothesis

# Outline

# Modeling Dependencies in Data

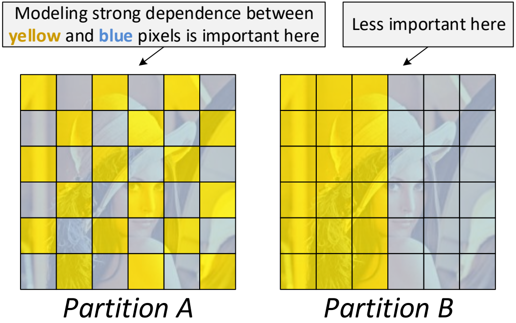
ConvNets realize func over many local elements (e.g. pixels)

# Modeling Dependencies in Data

ConvNets realize func over many local elements (e.g. pixels)

Key property of such func:

dependencies modeled between sets of input elements

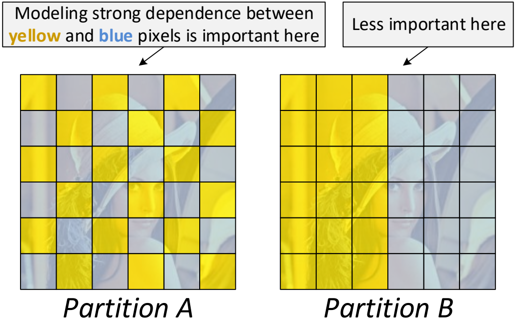


# Modeling Dependencies in Data

ConvNets realize func over many local elements (e.g. pixels)

Key property of such func:

dependencies modeled between sets of input elements



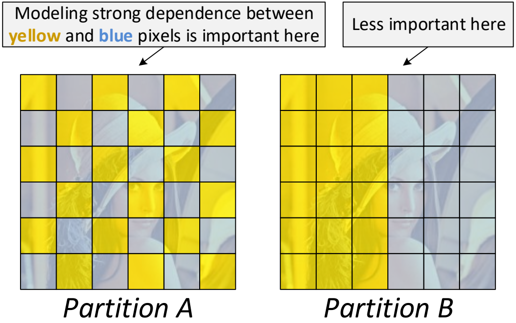

***Q:*** What kind of dependencies do ConvNets model?

# Modeling Dependencies in Data

ConvNets realize func over many local elements (e.g. pixels)

Key property of such func:

dependencies modeled between sets of input elements

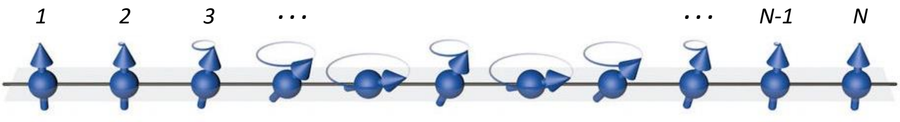



***Q:*** What kind of dependencies do ConvNets model?

***Q:*** How do these relate to network arch?

# Quantum Entanglement

# Quantum Entanglement




In quantum physics, particle is represented as vec in Hilbert space:

$$|\text{particle state}\rangle = \sum_{d=1}^{M} \underbrace{a_d}_{\text{coeff}} \cdot \underbrace{|\psi_d\rangle}_{\text{basis}} \in \mathbf{H}$$

# Quantum Entanglement

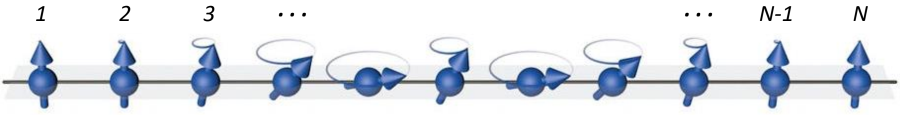

In quantum physics, particle is represented as vec in Hilbert space:

$$|\text{particle state}\rangle = \sum_{d=1}^{M} \underbrace{a_d}_{\text{coeff}} \cdot \underbrace{|\psi_d\rangle}_{\text{basis}} \in \mathbf{H}$$

System of $N$ particles is represented as vec in tensor product space:

$$|\text{system state}\rangle = \sum_{d_1 \dots d_N = 1}^{M} \underbrace{\mathcal{A}_{d_1 \dots d_N}}_{\text{coeff tensor}} \cdot |\psi_{d_1}\rangle \otimes \cdots \otimes |\psi_{d_N}\rangle \in \mathbf{H}^{\otimes N}$$

# Quantum Entanglement




In quantum physics, particle is represented as vec in Hilbert space:

$$\left|\text{particle state}\right\rangle = \sum_{d=1}^{M} \underbrace{a_d}_{\text{coeff}} \cdot \underbrace{\left|\psi_d\right\rangle}_{\text{basis}} \in \mathbf{H}$$

System of $N$ particles is represented as vec in tensor product space:

$$\left|\text{system state}\right\rangle = \sum_{d_1 \ldots d_N=1}^{M} \underbrace{\mathcal{A}_{d_1 \ldots d_N}}_{\text{coeff tensor}} \cdot \left|\psi_{d_1}\right\rangle \otimes \cdots \otimes \left|\psi_{d_N}\right\rangle \in \mathbf{H}^{\otimes N}$$

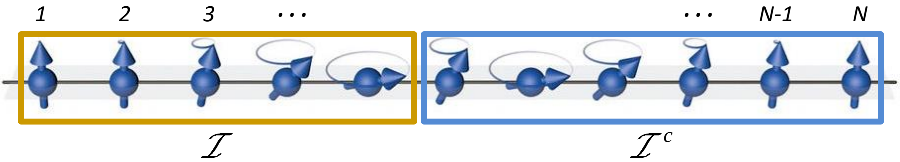
**Quantum entanglement** quantifies "dependencies" that system state models between sets of particles

# Quantum Entanglement (cont'd)

$$\boxed{|\text{system state}\rangle = \sum_{d_1 \ldots d_N = 1}^{M} \mathcal{A}_{d_1 \ldots d_N} \cdot |\psi_{d_1}\rangle \otimes \cdots \otimes |\psi_{d_N}\rangle}$$

# Quantum Entanglement (cont'd)

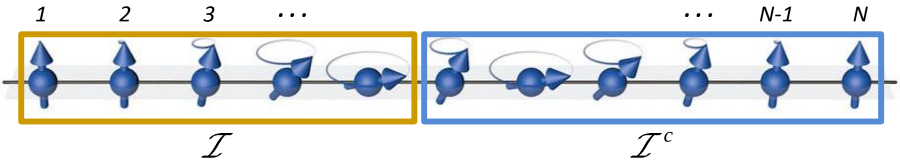
$$\boxed{|\text{system state}\rangle = \sum_{d_1 \ldots d_N = 1}^{M} \mathcal{A}_{d_1 \ldots d_N} \cdot |\psi_{d_1}\rangle \otimes \cdots \otimes |\psi_{d_N}\rangle}$$




Consider partition of particles into sets $\mathcal{I}$ and $\mathcal{I}^c$

# Quantum Entanglement (cont'd)

$$|\text{system state}\rangle = \sum_{d_1 \ldots d_N = 1}^{M} \mathcal{A}_{d_1 \ldots d_N} \cdot |\psi_{d_1}\rangle \otimes \cdots \otimes |\psi_{d_N}\rangle$$

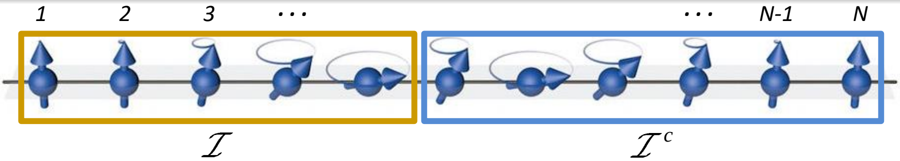



Consider partition of particles into sets $\mathcal{I}$ and $\mathcal{I}^c$

$[\![\mathcal{A}]\!]_{\mathcal{I}}$ – matricization of coeff tensor $\mathcal{A}$ w.r.t. $\mathcal{I}$:

- arrangement of $\mathcal{A}$ as matrix
- rows/cols correspond to modes indexed by $\mathcal{I}/\mathcal{I}^c$

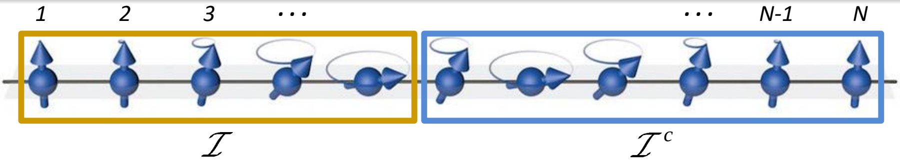
# Quantum Entanglement (cont'd)




$$|\text{system state}\rangle = \sum_{d_1 \ldots d_N = 1}^{M} \mathcal{A}_{d_1 \ldots d_N} \cdot |\psi_{d_1}\rangle \otimes \cdots \otimes |\psi_{d_N}\rangle$$

$[\![\mathcal{A}]\!]_{\mathcal{I}}$ – matricization of $\mathcal{A}$ w.r.t. $\mathcal{I}$

# Quantum Entanglement (cont'd)

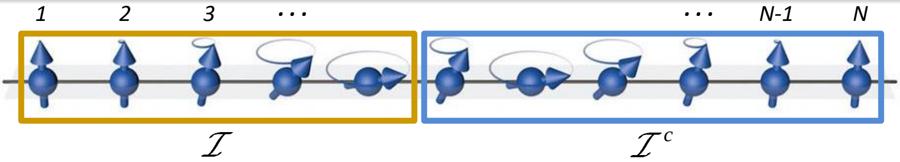



$$|\text{system state}\rangle = \sum_{d_1 \ldots d_N = 1}^{M} \mathcal{A}_{d_1 \ldots d_N} \cdot |\psi_{d_1}\rangle \otimes \cdots \otimes |\psi_{d_N}\rangle$$

$[\![\mathcal{A}]\!]_{\mathcal{I}}$ – matricization of $\mathcal{A}$ w.r.t. $\mathcal{I}$

$\boldsymbol{\sigma} = (\sigma_1, \sigma_2, \ldots, \sigma_R)$ — singular vals of $[\![\mathcal{A}]\!]_{\mathcal{I}}$

# Quantum Entanglement (cont'd)



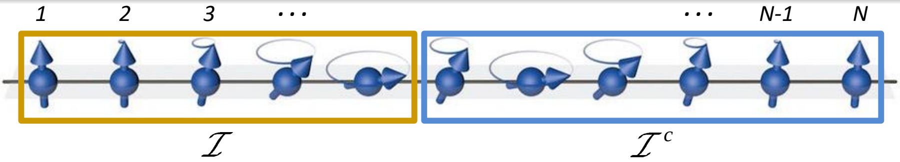

$$|\text{system state}\rangle = \sum_{d_1 \ldots d_N = 1}^{M} \mathcal{A}_{d_1 \ldots d_N} \cdot |\psi_{d_1}\rangle \otimes \cdots \otimes |\psi_{d_N}\rangle$$

$[\![\mathcal{A}]\!]_{\mathcal{I}}$ – matricization of $\mathcal{A}$ w.r.t. $\mathcal{I}$

$\boldsymbol{\sigma} = (\sigma_1, \sigma_2, \ldots, \sigma_R)$ — singular vals of $[\![\mathcal{A}]\!]_{\mathcal{I}}$

Entanglement measures between particles of $\mathcal{I}$ and of $\mathcal{I}^c$ are based on $\boldsymbol{\sigma}$:

# Quantum Entanglement (cont'd)




$$|\text{system state}\rangle = \sum\nolimits_{d_1 \ldots d_N = 1}^{M} \mathcal{A}_{d_1 \ldots d_N} \cdot |\psi_{d_1}\rangle \otimes \cdots \otimes |\psi_{d_N}\rangle$$

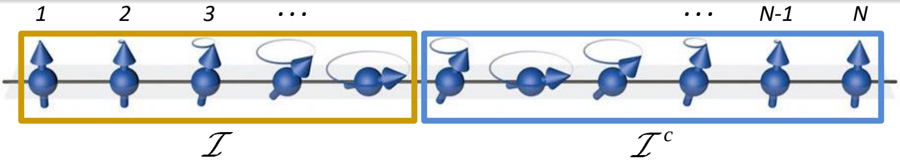
$[\![\mathcal{A}]\!]_{\mathcal{I}}$ – matricization of $\mathcal{A}$ w.r.t. $\mathcal{I}$

$\boldsymbol{\sigma} = (\sigma_1, \sigma_2, \ldots, \sigma_R)$ — singular vals of $[\![\mathcal{A}]\!]_{\mathcal{I}}$

Entanglement measures between particles of $\mathcal{I}$ and of $\mathcal{I}^c$ are based on $\boldsymbol{\sigma}$:

- Entanglement Entropy: entropy of $(\sigma_1^2, \ldots, \sigma_R^2) / \|\boldsymbol{\sigma}\|_2^2$

# Quantum Entanglement (cont'd)




$$|\text{system state}\rangle = \sum_{d_1 \ldots d_N = 1}^{M} \mathcal{A}_{d_1 \ldots d_N} \cdot |\psi_{d_1}\rangle \otimes \cdots \otimes |\psi_{d_N}\rangle$$

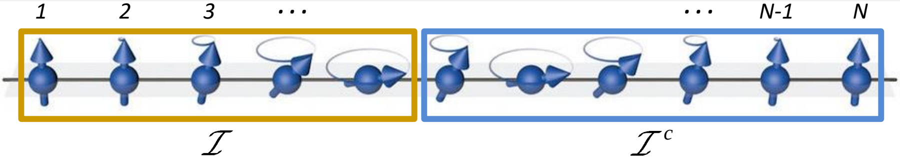
$[\![\mathcal{A}]\!]_{\mathcal{I}}$ – matricization of $\mathcal{A}$ w.r.t. $\mathcal{I}$

$\boldsymbol{\sigma} = (\sigma_1, \sigma_2, \ldots, \sigma_R)$ — singular vals of $[\![\mathcal{A}]\!]_{\mathcal{I}}$

Entanglement measures between particles of $\mathcal{I}$ and of $\mathcal{I}^c$ are based on $\boldsymbol{\sigma}$:

- Entanglement Entropy: entropy of $(\sigma_1^2, \ldots, \sigma_R^2) / \|\boldsymbol{\sigma}\|_2^2$
- Geometric Measure: $1 - \sigma_1^2 / \|\boldsymbol{\sigma}\|_2^2$

# Quantum Entanglement (cont'd)




$$|\text{system state}\rangle = \sum_{d_1 \ldots d_N = 1}^{M} \mathcal{A}_{d_1 \ldots d_N} \cdot |\psi_{d_1}\rangle \otimes \cdots \otimes |\psi_{d_N}\rangle$$

$[\![\mathcal{A}]\!]_{\mathcal{I}}$ – matricization of $\mathcal{A}$ w.r.t. $\mathcal{I}$

$\boldsymbol{\sigma} = (\sigma_1, \sigma_2, \ldots, \sigma_R)$ — singular vals of $[\![\mathcal{A}]\!]_{\mathcal{I}}$

Entanglement measures between particles of $\mathcal{I}$ and of $\mathcal{I}^c$ are based on $\boldsymbol{\sigma}$:

- Entanglement Entropy: entropy of $(\sigma_1^2, \ldots, \sigma_R^2) / \|\boldsymbol{\sigma}\|_2^2$
- Geometric Measure: $1 - \sigma_1^2 / \|\boldsymbol{\sigma}\|_2^2$
- **Schmidt Number**: $\|\boldsymbol{\sigma}\|_0 = \mathit{rank}[\![\mathcal{A}]\!]_{\mathcal{I}}$

# Measuring Dependence with Entanglement

Structural equivalence:

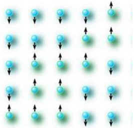
quantum many-body state

$$|\text{system state}\rangle = \sum_{d_1 \ldots d_N = 1}^{M} \underbrace{\mathcal{A}_{d_1 \ldots d_N}}_{\text{coeff tensor}} \cdot |\psi_{d_1}\rangle \otimes \cdots \otimes |\psi_{d_N}\rangle$$

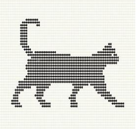
func over many local elements

$$h(\mathbf{x}_1, \ldots, \mathbf{x}_N) = \sum_{d_1 \ldots d_N = 1}^{M} \underbrace{\mathcal{A}_{d_1 \ldots d_N}}_{\text{coeff tensor}} \cdot f_{d_1}(\mathbf{x}_1) \cdots f_{d_N}(\mathbf{x}_N)$$

*state of many particles*



*func over many pixels*



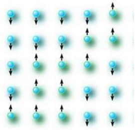
# Measuring Dependence with Entanglement

Structural equivalence:

<u>quantum many-body state</u>

$$|\text{system state}\rangle = \sum_{d_1 \ldots d_N = 1}^{M} \underbrace{\mathcal{A}_{d_1 \ldots d_N}}_{\text{coeff tensor}} \cdot |\psi_{d_1}\rangle \otimes \cdots \otimes |\psi_{d_N}\rangle$$

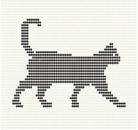
*state of many particles*



<u>func over many local elements</u>

$$h(\mathbf{x}_1, \ldots, \mathbf{x}_N) = \sum_{d_1 \ldots d_N = 1}^{M} \underbrace{\mathcal{A}_{d_1 \ldots d_N}}_{\text{coeff tensor}} \cdot f_{d_1}(\mathbf{x}_1) \cdots f_{d_N}(\mathbf{x}_N)$$

*func over many pixels*



**We may quantify dependencies func models between input sets by applying entanglement measures to its coeff tensor!**

# Measuring Dependence with Entanglement – Interpretation

$$h(\mathbf{x}_1, \ldots, \mathbf{x}_N) = \sum_{d_1 \ldots d_N = 1}^{M} \underbrace{\mathcal{A}_{d_1 \ldots d_N}}_{\text{coeff tensor}} \cdot f_{d_1}(\mathbf{x}_1) \cdots f_{d_N}(\mathbf{x}_N)$$

# Measuring Dependence with Entanglement – Interpretation

$$h(\mathbf{x}_1,\ldots,\mathbf{x}_N) = \sum_{d_1 \ldots d_N = 1}^{M} \underbrace{\mathcal{A}_{d_1 \ldots d_N}}_{\text{coeff tensor}} \cdot f_{d_1}(\mathbf{x}_1) \cdots f_{d_N}(\mathbf{x}_N)$$

When func $h(\cdot)$ is separable w.r.t. input sets $\mathcal{I}/\mathcal{I}^c$:

$$\exists g, g' \ \text{ s.t. } \ h(\mathbf{x}_1,\ldots,\mathbf{x}_N) = g\left((\mathbf{x}_i)_{i \in \mathcal{I}}\right) \cdot g'\left((\mathbf{x}_{i'})_{i' \in \mathcal{I}^c}\right)$$

it does not model any dependence between $\mathcal{I}/\mathcal{I}^c$

(in probabilistic setting, means $\mathcal{I}/\mathcal{I}^c$ are stat independent)

# Measuring Dependence with Entanglement – Interpretation

$$h(\mathbf{x}_1,\ldots,\mathbf{x}_N)=\sum_{d_1\ldots d_N=1}^{M}\underbrace{\mathcal{A}_{d_1\ldots d_N}}_{\text{coeff tensor}}\cdot f_{d_1}(\mathbf{x}_1)\cdots f_{d_N}(\mathbf{x}_N)$$

When func $h(\cdot)$ is separable w.r.t. input sets $\mathcal{I}/\mathcal{I}^c$:

$$\exists g,g' \text{ s.t. } h(\mathbf{x}_1,\ldots,\mathbf{x}_N)=g\left((\mathbf{x}_i)_{i\in\mathcal{I}}\right)\cdot g'\left((\mathbf{x}_{i'})_{i'\in\mathcal{I}^c}\right)$$

it does not model any dependence between $\mathcal{I}/\mathcal{I}^c$

(in probabilistic setting, means $\mathcal{I}/\mathcal{I}^c$ are stat independent)

Entanglement measures on $\mathcal{A}$ quantify dist of $h(\cdot)$ from separability:

- $\mathcal{A}$ has high (low) entanglement w.r.t. $\mathcal{I}/\mathcal{I}^c$
  $\Longrightarrow$ $h(\cdot)$ is far from (close to) separability w.r.t. $\mathcal{I}/\mathcal{I}^c$

# Measuring Dependence with Entanglement – Interpretation

$$h(\mathbf{x}_1,\ldots,\mathbf{x}_N)=\sum_{d_1\ldots d_N=1}^{M}\underbrace{\mathcal{A}_{d_1\ldots d_N}}_{\text{coeff tensor}}\cdot f_{d_1}(\mathbf{x}_1)\cdots f_{d_N}(\mathbf{x}_N)$$

When func $h(\cdot)$ is separable w.r.t. input sets $\mathcal{I}/\mathcal{I}^c$:

$$\exists g,g' \text{ s.t. } h(\mathbf{x}_1,\ldots,\mathbf{x}_N)=g\left((\mathbf{x}_i)_{i\in\mathcal{I}}\right)\cdot g'\left((\mathbf{x}_{i'})_{i'\in\mathcal{I}^c}\right)$$

it does not model any dependence between $\mathcal{I}/\mathcal{I}^c$

(in probabilistic setting, means $\mathcal{I}/\mathcal{I}^c$ are stat independent)

Entanglement measures on $\mathcal{A}$ quantify dist of $h(\cdot)$ from separability:

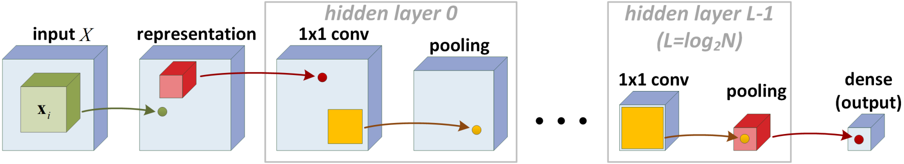
- $\mathcal{A}$ has high (low) entanglement w.r.t. $\mathcal{I}/\mathcal{I}^c$
  $\implies h(\cdot)$ is far from (close to) separability w.r.t. $\mathcal{I}/\mathcal{I}^c$
- Choice of entanglement measure determines dist metric

# Outline

# Convolutional Arithmetic Circuits $\longleftrightarrow$ Tensor Networks

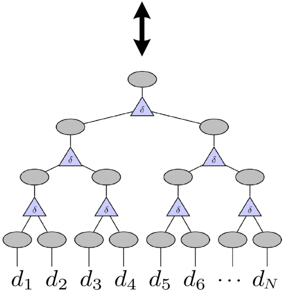
**Recap**

Func realized by ConvAC may be represented via tree TN




*structural correspondence*

| ConvAC | TN |
|---|---|
| input elements | terminal nodes |
| # of layers | tree depth |
| layer widths | bond dims |
| pool geometry | connectivity |
| overlaps | duplications |
| … | … |



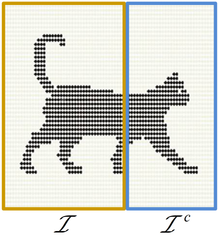
# Entanglement via Minimal Cuts

## Theorem (Quantum Max Flow/Min Cut)

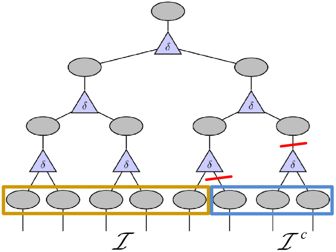
*Max Schmidt entanglement ConvAC models between input sets $\mathcal{I}/\mathcal{I}^c$ = min cut in respective TN separating nodes of $\mathcal{I}/\mathcal{I}^c$*

*ConvAC entanglement between input sets*

*TN min cut separating respective node sets*




=

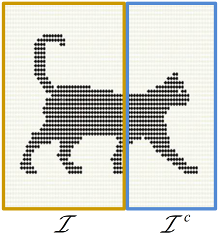


# Entanglement via Minimal Cuts

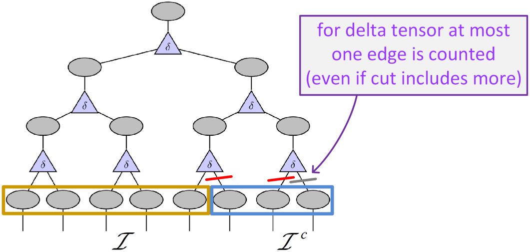
**Theorem (Quantum Max Flow/Min Cut)**

*Max Schmidt entanglement ConvAC models between input sets $\mathcal{I}/\mathcal{I}^c$ = min cut in respective TN separating nodes of $\mathcal{I}/\mathcal{I}^c$*

*ConvAC entanglement between input sets* = *TN min cut separating respective node sets*




=

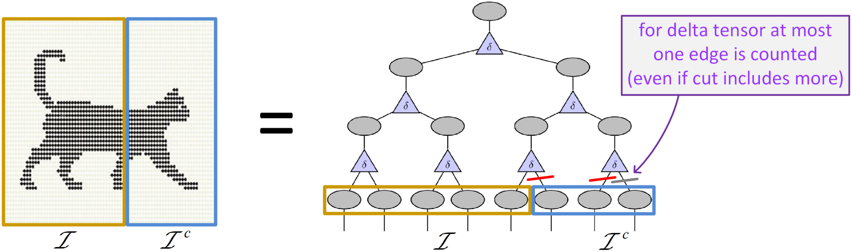


# Entanglement via Minimal Cuts

## Theorem (Quantum Max Flow/Min Cut)

*Max Schmidt entanglement ConvAC models between input sets $\mathcal{I}/\mathcal{I}^c$ = min cut in respective TN separating nodes of $\mathcal{I}/\mathcal{I}^c$*

*ConvAC entanglement between input sets* | *TN min cut separating respective node sets*



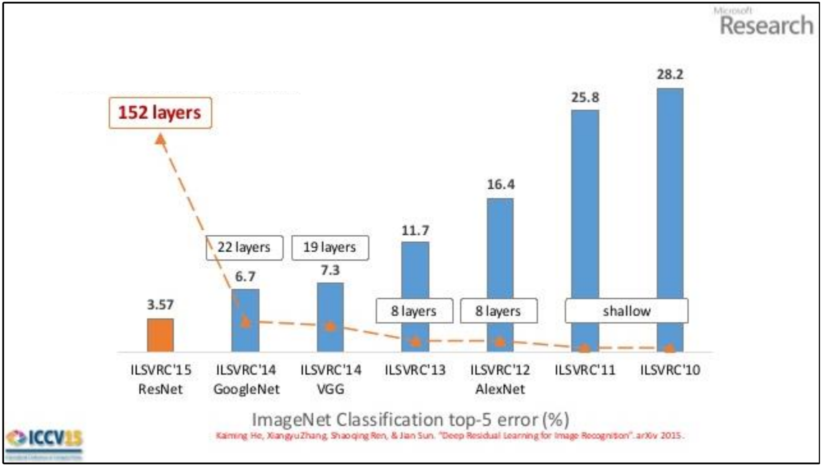

**We may analyze the effect of ConvAC arch on the dependencies (entanglement) it can model!**

# Depth

# Depth

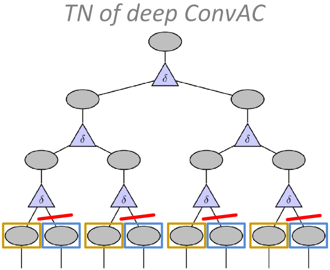
**Conjecture – depth efficiency**

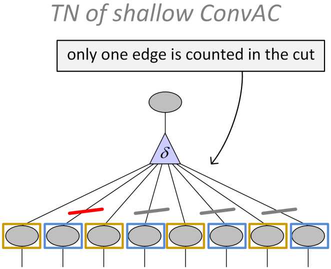
Deep ConvNets realize func requiring shallow ConvNets to grow unfeasibly



# Depth (cont'd)

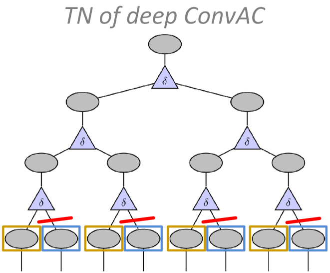
For certain partitions, min cut in TN of deep ConvAC is exponentially larger than in TN of shallow ConvAC

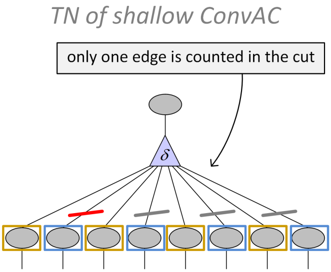


# Depth (cont'd)

For certain partitions, min cut in TN of deep ConvAC is exponentially larger than in TN of shallow ConvAC




This implies:

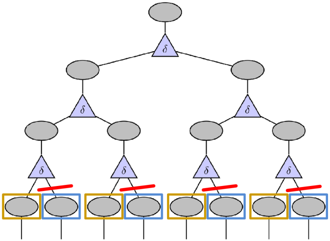
**Claim**

*Deep ConvAC can model dependencies (entanglements) requiring shallow ConvAC to have exponential width*

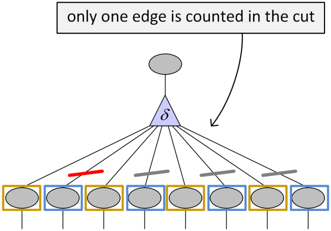
# Depth (cont'd)

For certain partitions, min cut in TN of deep ConvAC is exponentially larger than in TN of shallow ConvAC

*TN of deep ConvAC*




*TN of shallow ConvAC*




This implies:

**Claim**

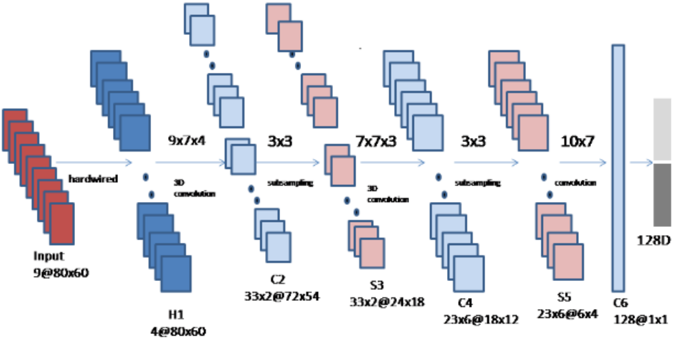
*Deep ConvAC can model dependencies (entanglements) requiring shallow ConvAC to have exponential width*

**Depth efficiency proven for ConvAC!**

# Layer Widths

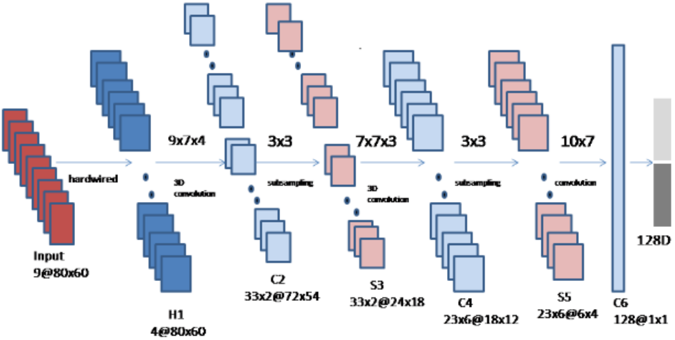
# Layer Widths

Currently no principle for setting widths (# of channels) of ConvNet layers



# Layer Widths

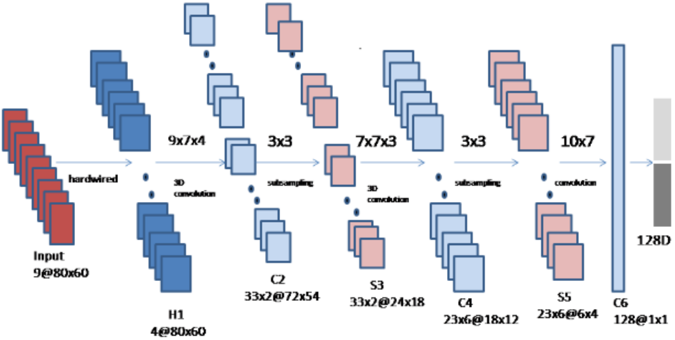
Currently no principle for setting widths (# of channels) of ConvNet layers




***Q:*** *What are implications of widening one layer vs. another?*

# Layer Widths

Currently no principle for setting widths (# of channels) of ConvNet layers




***Q:*** *What are implications of widening one layer vs. another?*

***Q:*** *Can widths be tailored for a given task?*

# Layer Widths (cont'd)

**Claim**

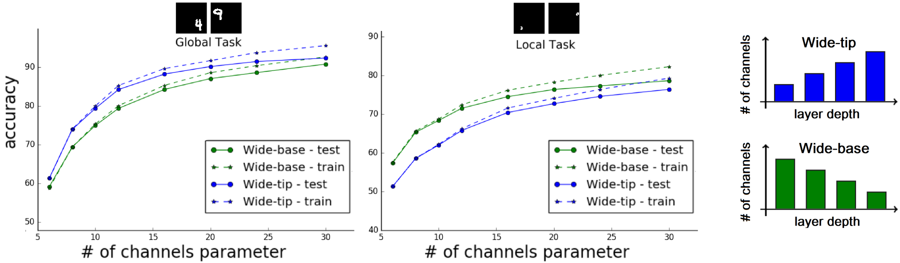
*Deep (early) layer widths are important for long (short)-range dependencies*

# Layer Widths (cont'd)

**Claim**

*Deep (early) layer widths are important for long (short)-range dependencies*

## Experiment

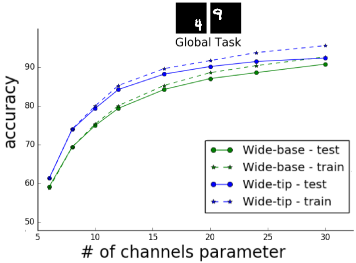


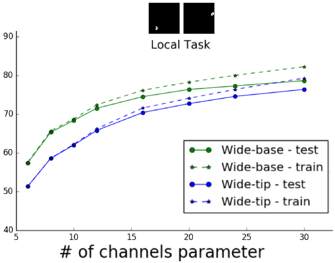
# Layer Widths (cont'd)

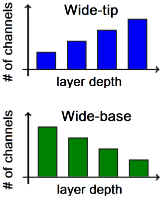
## Claim

*Deep (early) layer widths are important for long (short)-range dependencies*

## Experiment



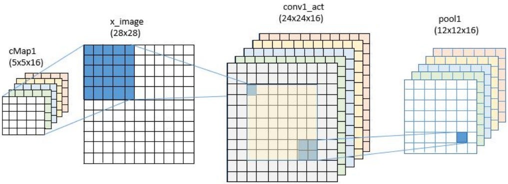




**ConvAC layer widths can be tailored to maximize dependencies (entanglements) required for given task!**

# Pooling Geometry

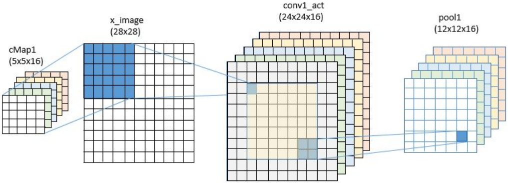
# Pooling Geometry

ConvNets typically employ square conv/pool windows

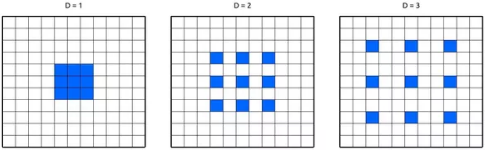


# Pooling Geometry

ConvNets typically employ square conv/pool windows

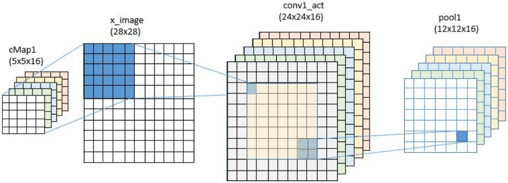



Recently, dilated windows have also become popular

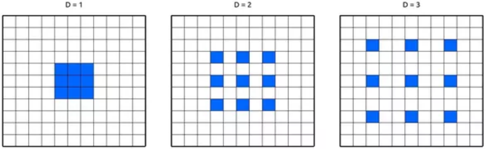


# Pooling Geometry

ConvNets typically employ square conv/pool windows



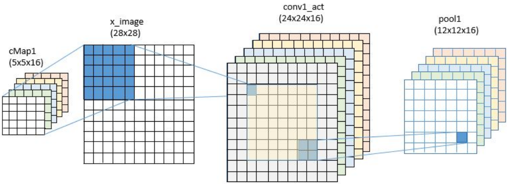

Recently, dilated windows have also become popular



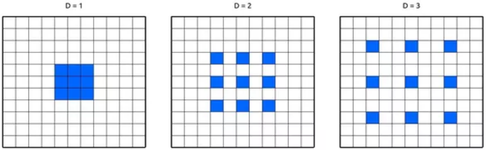

***Q:*** *What are implications of one window geometry vs. another?*

# Pooling Geometry

ConvNets typically employ square conv/pool windows




Recently, dilated windows have also become popular




***Q:*** *What are implications of one window geometry vs. another?*

***Q:*** *Can geometries be tailored for a given task?*

# Pooling Geometry (cont'd)

## Claim

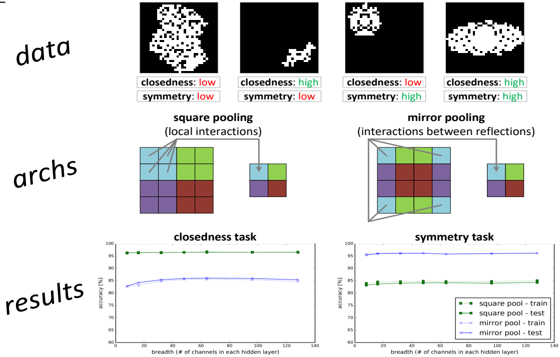
*Input elements pooled together early have stronger dependence*

# Pooling Geometry (cont'd)

**Claim**

*Input elements pooled together early have stronger dependence*

## Experiment

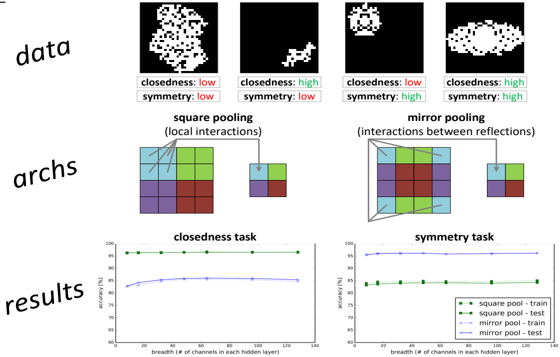


# Pooling Geometry (cont'd)

**Claim**

*Input elements pooled together early have stronger dependence*

## Experiment

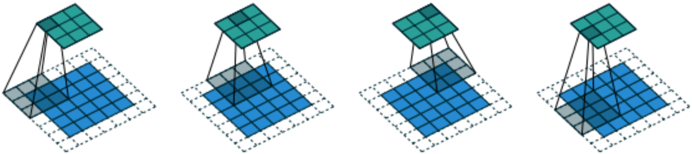


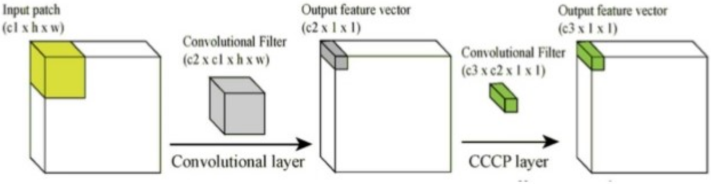

**ConvAC pooling geometry can be tailored to maximize dependencies (entanglements) required for given task!**

# Overlapping Operations

# Overlapping Operations

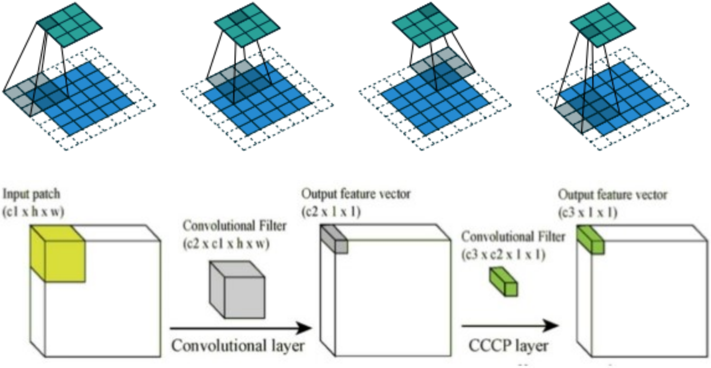
Modern ConvNets employ both overlapping and non-overlapping conv/pool operations





# Overlapping Operations

Modern ConvNets employ both overlapping and non-overlapping conv/pool operations




***Q:*** *What are implications of introducing overlaps?*

# Overlapping Operations (cont'd)

## Claim

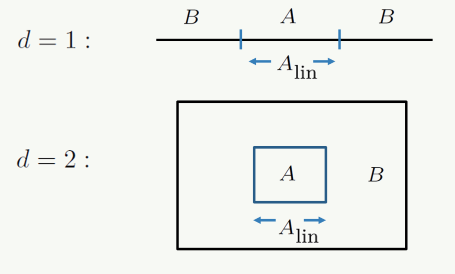
*Overlaps in conv/pool operations allow modeling dependencies that otherwise require exponential size*

# Overlapping Operations (cont'd)

**Claim**

*Overlaps in conv/pool operations allow modeling dependencies that otherwise require exponential size*

**Area/volume law**:




**area law**

entanglement $\propto A_{\text{lin}}^{d-1}$

**volume law**

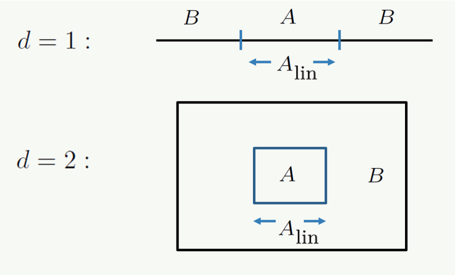
entanglement $\propto A_{\text{lin}}^{d}$

# Overlapping Operations (cont'd)

**Claim**

*Overlaps in conv/pool operations allow modeling dependencies that otherwise require exponential size*

**Area/volume law**:




**area law**

entanglement $\propto A_{\text{lin}}^{d-1}$

**volume law**

entanglement $\propto A_{\text{lin}}^{d}$

**ConvAC w/overlaps supports volume law entanglement!**

# Outline

# Other Types of Convolutional Networks

# Other Types of Convolutional Networks

We established equivalence:

$$\text{ConvAC} \longleftrightarrow \text{TN}$$

and used it to analyze dependencies (entanglement) ConvAC can model

# Other Types of Convolutional Networks

We established equivalence:

$$\text{ConvAC} \longleftrightarrow \text{TN}$$

and used it to analyze dependencies (entanglement) ConvAC can model

ConvAC delivers promising results in practice, but other ConvNets (e.g. w/ReLU activation and max pooling) are more common

# Other Types of Convolutional Networks

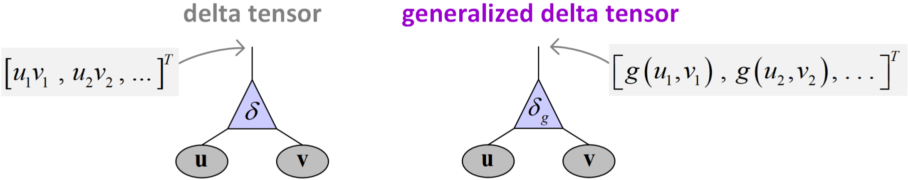
We established equivalence:

$$\text{ConvAC} \longleftrightarrow \text{TN}$$

and used it to analyze dependencies (entanglement) ConvAC can model

ConvAC delivers promising results in practice, but other ConvNets (e.g. w/ReLU activation and max pooling) are more common

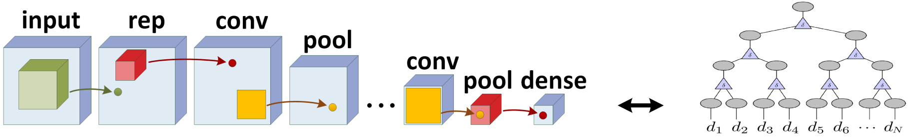
Our analysis extends to other ConvNets if we generalize delta tensor:



# Recurrent Networks

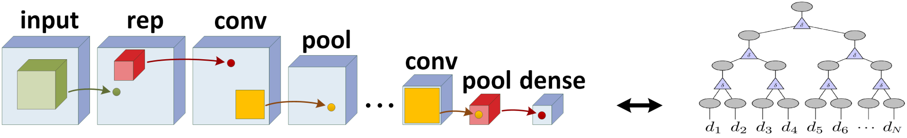
# Recurrent Networks

We analyzed **convolutional** nets via equivalence to TN w/**tree** arch

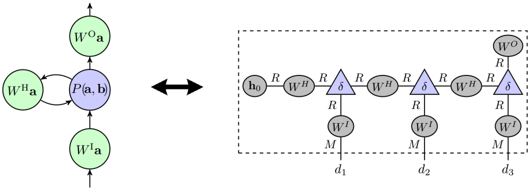


# Recurrent Networks

We analyzed **convolutional** nets via equivalence to TN w/**tree** arch

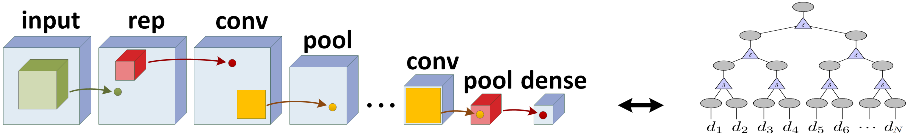



Analysis extends to **recurrent** nets via equivalence to TN w/**chain** arch

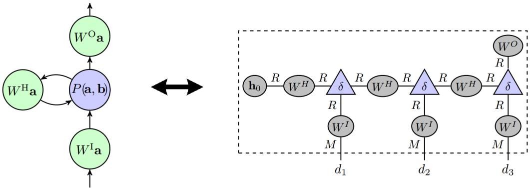


# Recurrent Networks

We analyzed **convolutional** nets via equivalence to TN w/**tree** arch




Analysis extends to **recurrent** nets via equivalence to TN w/**chain** arch




Recurrent nets process data sequentially; ability to model dependencies (entanglement) quantifies memory

# Outline

1. Deep Learning Theory: Expressiveness, Generalization and Optimization
2. Convolutional Networks as Tensor Networks
3. Expressiveness of Convolutional Networks
   - Dependencies as Quantum Entanglement
   - Analysis of Supported Entanglement
4. Extensions
5. **Conclusion**

# Recap

# Recap

- Three pillars of statistical learning theory:

  Expressiveness    Generalization    Optimization

# Recap

- Three pillars of statistical learning theory:

  Expressiveness Generalization Optimization

  - Well developed theory for classical ML

# Recap

- Three pillars of statistical learning theory:

  Expressiveness Generalization Optimization

  - Well developed theory for classical ML
  - Limited understanding for DL

# Recap

- Three pillars of statistical learning theory:

  Expressiveness    Generalization    Optimization

  - Well developed theory for classical ML
  - Limited understanding for DL

- State of the art DL arch can be represented as TN:

# Recap

- Three pillars of statistical learning theory:

  Expressiveness    Generalization    Optimization

  - Well developed theory for classical ML
  - Limited understanding for DL

- State of the art DL arch can be represented as TN:

  convolutional nets $\longleftrightarrow$ tree TN

  recurrent nets $\longleftrightarrow$ chain TN

# Recap

- Three pillars of statistical learning theory:

  Expressiveness    Generalization    Optimization

  - Well developed theory for classical ML
  - Limited understanding for DL

- State of the art DL arch can be represented as TN:

  convolutional nets $\longleftrightarrow$ tree TN

  recurrent nets $\longleftrightarrow$ chain TN

- Quantum entanglement quantifies dependencies modeled by DL arch

# Recap

- Three pillars of statistical learning theory:

  Expressiveness   Generalization   Optimization

  - Well developed theory for classical ML
  - Limited understanding for DL

- State of the art DL arch can be represented as TN:

  $$\text{convolutional nets} \quad \longleftrightarrow \quad \text{tree TN}$$
  $$\text{recurrent nets} \quad \longleftrightarrow \quad \text{chain TN}$$

- Quantum entanglement quantifies dependencies modeled by DL arch

- Quantum max flow/min cut theorem

  $\Longrightarrow$ new results on expressiveness in DL!

# Perspective

# Perspective

Understanding deep learning calls for natural sciences

# Outline

# Thank You