Lecture 23: Lego World of Deep Learning, Convolutional Neural Networks, etc

Instructor: Prof. Ganesh Ramakrishnan

What Changed with Neural Networks?

Hardly using image proc knowledge

- Origin: Computational Model of Threshold Logic from Warren McCulloch and Walter Pitts (1943)
- Big Leap: For ImageNet Challenge, AlexNet acheived 85 % accuracy (NIPS 2012). Previous best was 75 % (CVPR 2011). → using image proc knowledge
- Subsequent best was 96.5 % MSRA (arXiv 2015). Comparable to human level accuracy.
- Challenges involved were varied background, same object with different colors (*e.g.*, cats), varied sizes and postures of same objects, varied illuminated conditions. How??
- Tasks like OCR, Speech recognition are now possible without segmenting the word image/signal into character images/signals.

LeNet(1989 and 1998) v/s AlexNet(2012)

old

	LeeNet 1989	LeeNet 1998	AlexNet 2012
Task	Digit Recognition	Digit Recognition	Object Recognition
# Classes	10	10	1k
Image Size	16 X 16	28 X 28	256 X 256 X 3
# Examples	7k	60k	1.2 M
units	1256	8084	658 k
parameters	9760	60k	60 M
connections	65k	344k	652M
Summation	Sigmoid	Sigmoid	ReLU
GPU/ Non-GPU	Non-GPU based.	Non-GPU based.	GPU based.
Operations	11 billion	412 billions	200 quadrillions

Aspects other than accuracy

Reasons for Big Leap

- Why LeeNet was not as successful as AlexNet, though the algorithm was same?
- Right algorithm at wrong time. → No GPUs better algorithms for (batch) linear alg. operations
- Modern features.
- Advancement in Machine learning. → Dropout regularizations, pretrain, precondition
- Realistic data collection in huge amount due to: regular competitions, evaluation metrics or challenging problem statements. [Owing to web explosion]
- Advances in Computational Resources: GPUs, industrial scale clusters.
- Evolution of tasks: Classification of 10 objects to 100 objects to structure of classes.

Challenges with Neural Networks

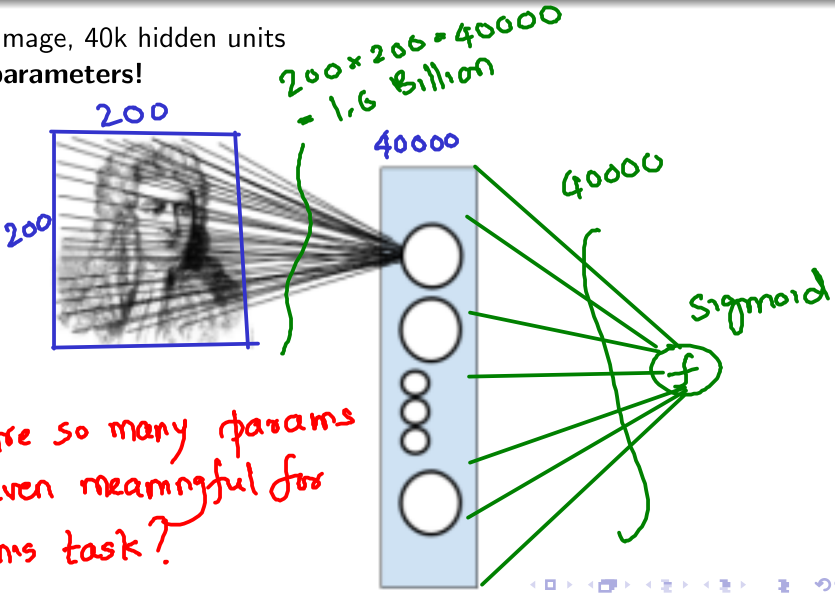
As # boolean vars increases, fraction of separable fns drastically shrinks

Recall from Tutorial 6, Curse of Dimensionality: If dimension is large, number of samples may be too small for accurate parameter estimation.
Otherwise, we may end up in using too a complicated model for the data, *i.e.*, we may over-fit

Challenges with Neural Networks (contd)

200 X 200 grayscale image, 40k hidden units

around 1.6 Billion parameters!



Are so many params even meaningful for this task?

Challenges with Neural Networks (contd)

Npw consider the task of Colored Image Recognition

Input Image Size: 200 X 200 X 3 (RGB)

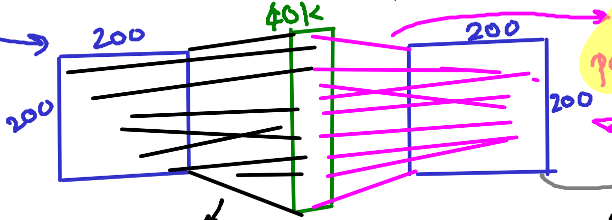
Multi-Layer Perceptron (MLP): Hidden Layer with 40k neurons results in 1.6 billion $\times$ 3 = 4.8 billion! parameters.

Question: How many neurons (location specific)?

Answer: 40000 + (o/p)

Challenges and Opportunities with Neural Networks

1. Local Optima: Only Approximately Correct Solution. But stochastic gradient descent by avoiding even local minima often gives good generalization (avoids overfitting)
2. Training data: Need for large number of training instances. Pre-training[1]
3. Extensive Computations and Numerical Precision: With lots of gradient computations and backpropagation, errors can be compounding. Advances in numerical computing tricks/matrix multiplication and GPUs!



Standby parameters to be removed on seeing labeled data

Treat the i/p image itself as class

Error measured is "reconstruction" error pixelwise evaluated

Need to "pre" train on unclassified i/p data itself

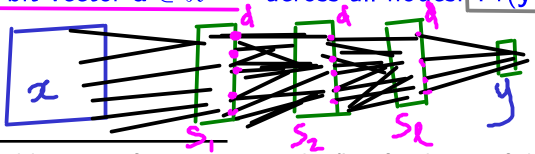
[1] Unsupervised learning of parameters in the first few layers of the NN

Challenges and Opportunities with Neural Networks

1. Local Optima: Only Approximately Correct Solution. But stochastic gradient descent by avoiding even local minima often gives good generalization
2. Training data: Need for large number of training instances. Pre-training[1]
3. Extensive Computations and Numerical Precision: With lots of gradient computations and backpropagation, errors can be compounding. Advances in numerical computing tricks/matrix multiplication and GPUs!
4. Architecture Design: How many nodes and edges in each hidden layer? How many layers? Network structures can be overestimated and then regularized using Dropout, *i.e.*, randomly multiply the output of a node with 0 using a random dropout bit vector $\mathbf{d} \in \Re^{\sum_l s_l}$ across all nodes: $\boxed{\Pr(\mathbf{y}|\mathbf{x}) = \sum_{\mathbf{d}} \Pr(\mathbf{d}) \Pr(\mathbf{y}|\mathbf{x}, \mathbf{d})}$

Extreme form of regularization

Bagging technique



$d_i \in \{0, 1\}$

In practice you average across some configurations of d

[1]Unsupervised learning of parameters in the first few layers of the NN

Challenges and Opportunities with Neural Networks

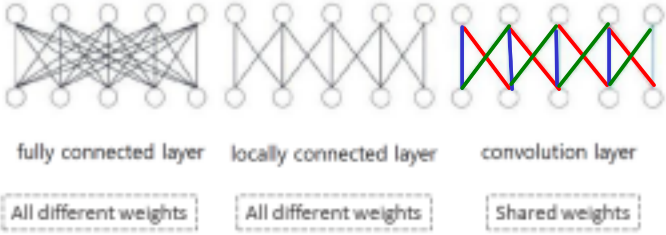
1. Local Optima: Only Approximately Correct Solution. But stochastic gradient descent by avoiding even local minima often gives good generalization
2. Training data: Need for large number of training instances. Pre-training[1]
3. Extensive Computations and Numerical Precision: With lots of gradient computations and backpropagation, errors can be compounding. Advances in numerical computing tricks/matrix multiplication and GPUs!
4. Architecture Design: How many nodes and edges in each hidden layer? How many layers? Network structures can be overestimated and then regularized using Dropout, *i.e.*, randomly multiply the output of a node with 0 using a random dropout bit vector $\mathbf{d} \in \Re^{\sum_l s_l}$ across all nodes: $\Pr(\mathbf{y}|\mathbf{x}) = \sum_{\mathbf{d}} \Pr(\mathbf{d}) \Pr(\mathbf{y}|\mathbf{x}, \mathbf{d})$
5. Associating Semantics: Architectures to suit particular tasks? Architectures can be designed keeping the problem in mind: Examples of CNNs, RNNs, Memory Cells, LSTMs, BiLSTMs, Embeddings, Inception, Attention Networks, *etc.*

Our next focus

[1] Unsupervised learning of parameters in the first few layers of the NN

Convolutional Neural Network

- Variation of multi layer feedforward neural network designed to use minimal preprocessing with wide application in image recognition and natural language processing
- Traditional multilayer perceptron(MLP) models do not take into account spatial structure of data and suffer from curse of dimensionality
- Convolution Neural network has smaller number of parameters due to **local connections** and **weight sharing** (Let us see how & why)



Consider the task of Colored Image Recognition
Input Image Size: 200 X 200 X 3 (RGB)
MLP: Hidden Layer with 40k neurons results in 4.8 Billion parameters.
With Convolutional Neural Networks?

How do CNNs reduce # params?

The Lego Blocks in Modern Deep Learning

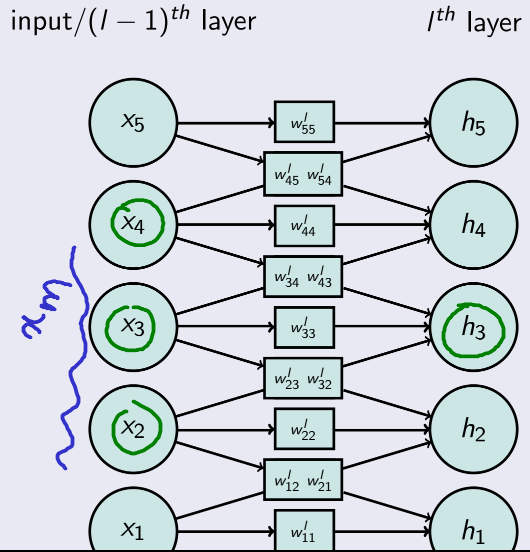
(Libraries)

1. **Depth/Feature Map**
2. **Patches/Kernels (provide for spatial interpolations)**
3. **Strides (enable downsampling)**
4. **Padding (shrinking across layers)**
5. Pooling
6. Inception
7. Memory cell and Backpropagation through time
8. Embeddings

Convolution: Sparse Interactions through Kernels (for Single Feature Map)

(k(i-m) could also be another kernel filter)

(x w) ok [convolution filter]



1) Each input x_m is connected to some h_i's OR

h_i is connected to x_m for all $m \in I_i$

$$h_i = \sum_{m \in I_i} x_m \omega_{mi}$$

OR $h_i = \sum_m x_m \omega_{mi} k(i-m)$

nearest nbr

where: $K(i-m) = 1$ if $|i-m| < \theta$

$\& = 0$ o/w

$$h_i = \sum_m x_m w_{mi} k(i-m)$$

$\theta = 2$

Toeplitz matrix K

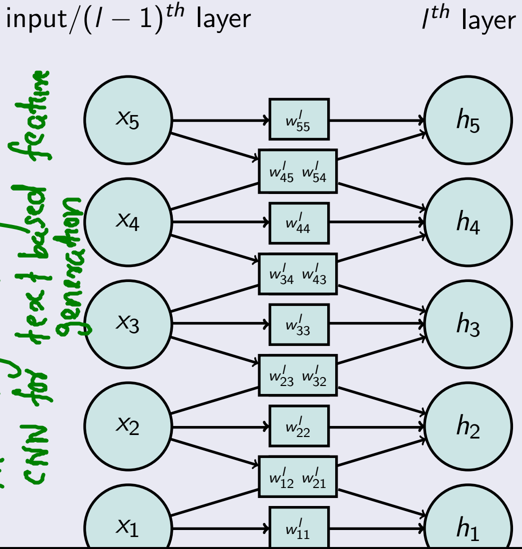
Has same structure

$$\vec{h} = \vec{x}^T (w \cdot K)$$

Can be computed efficiently for forward & backward propagation

Convolution: Sparse Interactions through Kernels (for Single Feature Map)

H/W: Try & interpret 1-D CNN for text based feature generation

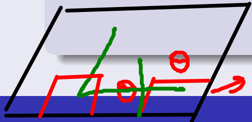


2D convolution filter

- $h_i = \sum_m x_m w_{mi} K(i-m)$
- On RHS, $K(i-m) = 1$ *iff* $|m-i| \leq 1$
- For 2-D inputs (such as images):

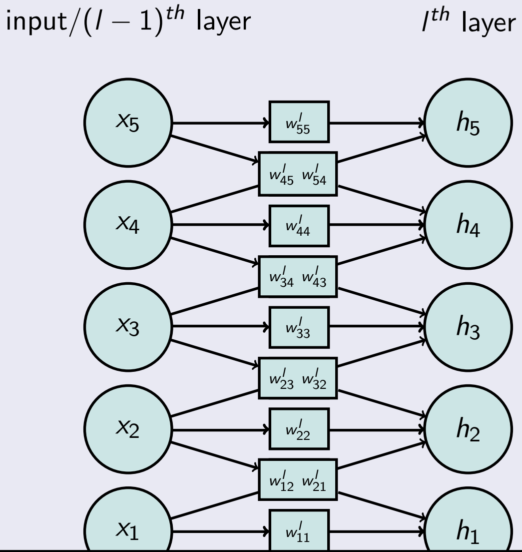
$$h_{ij} = \sum_m \sum_n x_{mn} w_{mnij} \, \underbrace{K(i-m, j-n)}$$

Toeplitz Tensor



→ patch level interactions dealt with in subsequent NN layers.

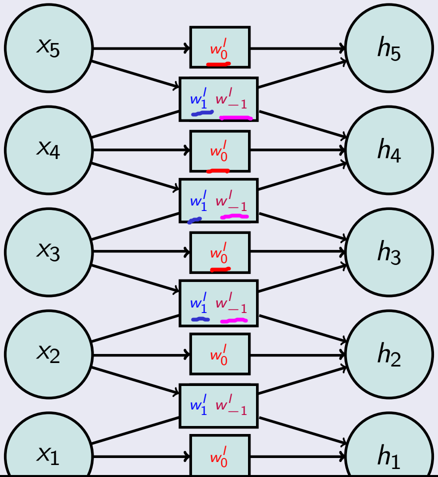
Convolution: Sparse Interactions through Kernels (for Single Feature Map)



- $h_i = \sum_m x_m w_{mi} K(i-m)$
- On RHS, $K(i-m) = 1$ *iff* $|m-i| \leq 1$
- For 2-D inputs (such as images):
  $h_{ij} = \sum_m \sum_n x_{mn} w_{ij,mn} K(i-m, j-n)$
- Intuition: Neighboring signals x_m (or pixels x_{mn}) more relevant than one's further away, reduces prediction time
- Can be viewed as multiplication with a Toeplitz matrix K (which has each row as the row above shifted by one element)
- Further, K is often sparse (eg: $K(i-m) = 1$ *iff* $|m-i| \leq \theta$)

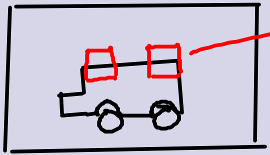
Convolution: Shared parameters and Patches (for Single Feature Map)

input/$(l-1)^{th}$ layer $\qquad$ l^{th} layer



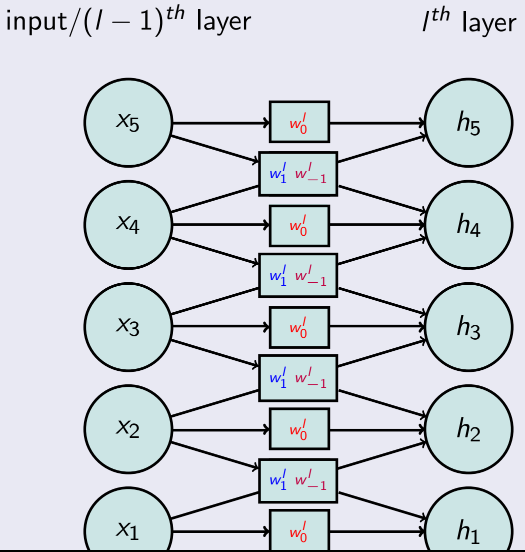
Param sparsity & sharing gives equivariance

$$f(g(x)) = g(f(x))$$



Param sharing! Captures uniformity in continuity & contrast with background

Convolution: Shared parameters and Patches (for Single Feature Map)



- $h_i = \sum_m x_m w_{i-m} K(i-m)$
- On LHS, $K(i-m) = 1$ *iff* $|m-i| \leq 1$
- For 2-D inputs (such as images):

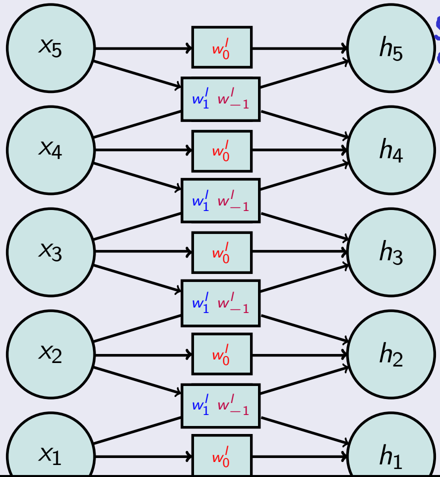
$$h_{ij} = \sum_{m,n} x_{mn} w_{i-m,j-n} K(i-m,j-n)$$

Convolution: Shared parameters and Patches (for Single Feature Map)

f & g are equivariant if

$f(g(x)) = g(f(x))$ → convolution

Shift operation

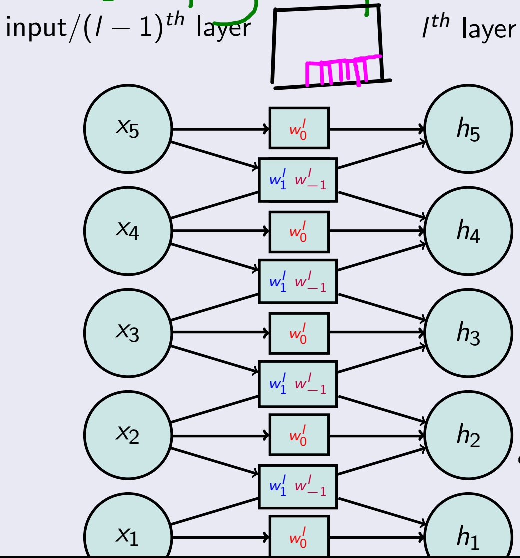


- $h_i = \sum_m x_m w_{i-m} K(i-m)$
- On LHS, $K(i-m) = 1$ *iff* $|m-i| \leq 1$
- For 2-D inputs (such as images):
  $h_{ij} = \sum_m \sum_n x_{mn} w_{i-m,j-n} K(i-m, j-n)$
- **Intuition:** Neighboring signals x_m (or pixels x_{mn}) affect in similar way irrespective of location (*i.e.*, value of m or n)
- **More Intuition:** Corresponds to moving **patches around the image**
- Further reduces *storage* requirement; does not affect prediction time
- Further, K is often sparse (eg: $K(i-m) = 1$ *iff* $|m-i| \leq \theta$)

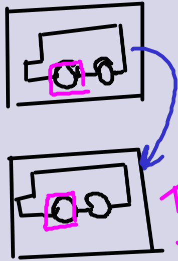
Already reduced by sparsity

Convolution: Strides and Padding (for Single Feature Map)

Sampling is implemented (implicitly through strides)



Equivariance: $f(g(x)) = g(f(x))$ is useful



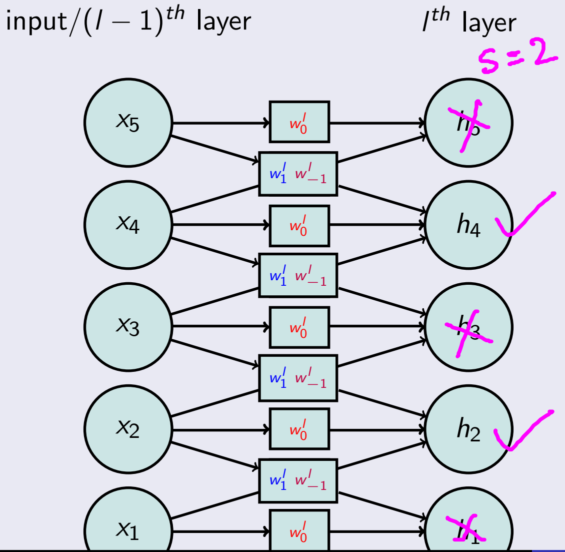
shifting "f"

To get same representation for type or its parts

sample$(f(g(x)))$

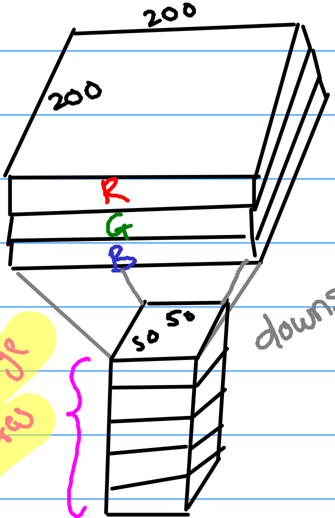
$\text{sample}(f(g(x))) = \text{sample}(g(f(x)))$

Convolution: Strides and Padding (for Single Feature Map)



- Consider only h_i's where i is a multiple of s.
- **Intuition:** Stride of s corresponds to moving the patch by s steps at a time
- **More Intuition:** Stride of s corresponds to downsampling by s
- What to do at the ends/corners: Ans: **Pad** with either 0's (**same padding**) or let the next layer have fewer nodes (**valid padding**)
- Reduces *storage* requirement as well as prediction time

Birds eye view of convolution (param sharing, sparsity, downsampling)



downsampling/striding (s=4)

These are multiple feature maps in the same hidden layer

each layer intends to capture image features

Image feature:

1. Does it have heads
2. Does it have hands
3. Color is contrasting or natural ?!