Understanding DL Models

Thomas Breuel
NVIDIA Research

Plan

- Overview of different approaches to understanding DNNs
- Focus
  - linear methods and models (PCA, ICA, RICA, linear layers)
  - convolutional layers and 2D signal analysis
  - Bayesian decision theory
  - Maximum Likelihood estimates and their limitations
- Petascale Training

Why?

These are topics that…

- are fundamentally important to working effectively with DNNs
- are often not taught in sufficient details
- are important to our work at NVIDIA Research

Goals

Get you *started* with these topics if you haven't seen them before.

If you have seen something before, this may still be a nice review, in particular since I tend to explain methods with actual, calculated examples.

Slides and Worksheets At...

https://github.com/tmbdev/dl-2018


A GENTLEMAN'S HATS
TOP HAT
FEDORA
TRILBY
NEWSBOY
DRIVING/IVY/FLAT
BOWLER/DERBY
COWBOY
PORK PIE
TRAPPER
BEARINGS

DEEP LEARNING VIEW

Deep Learning Expert

What is an L1 loss and when would you use it?

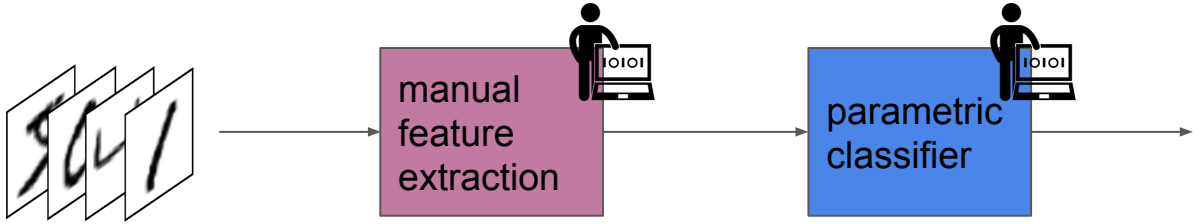
How does batch normalization work?

Describe the architecture of a GAN and the major difficulties in training it.

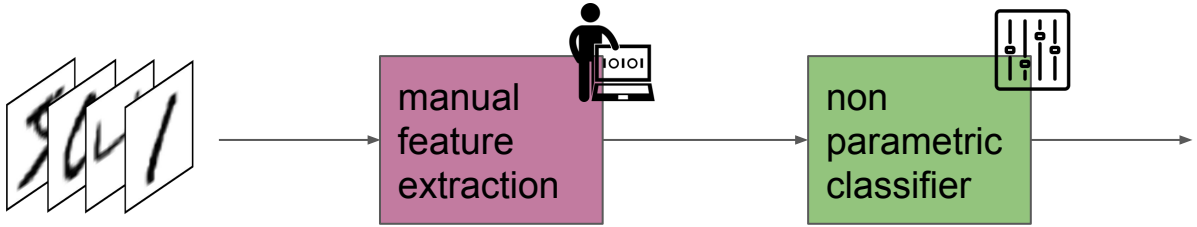
How do you use autoencoding for pretraining?

the road to deep learning

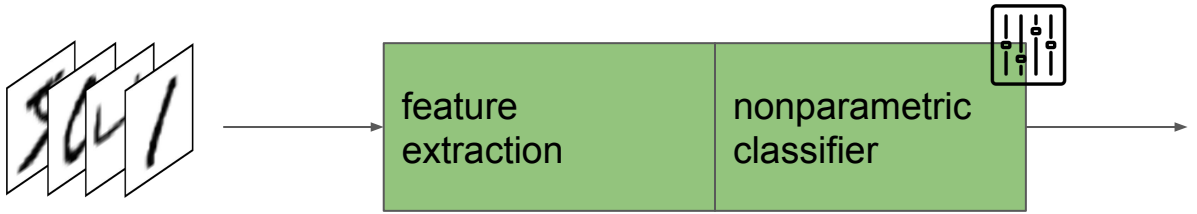
pattern recognition
machine learning



early neural
networks



deep learning



end to end training

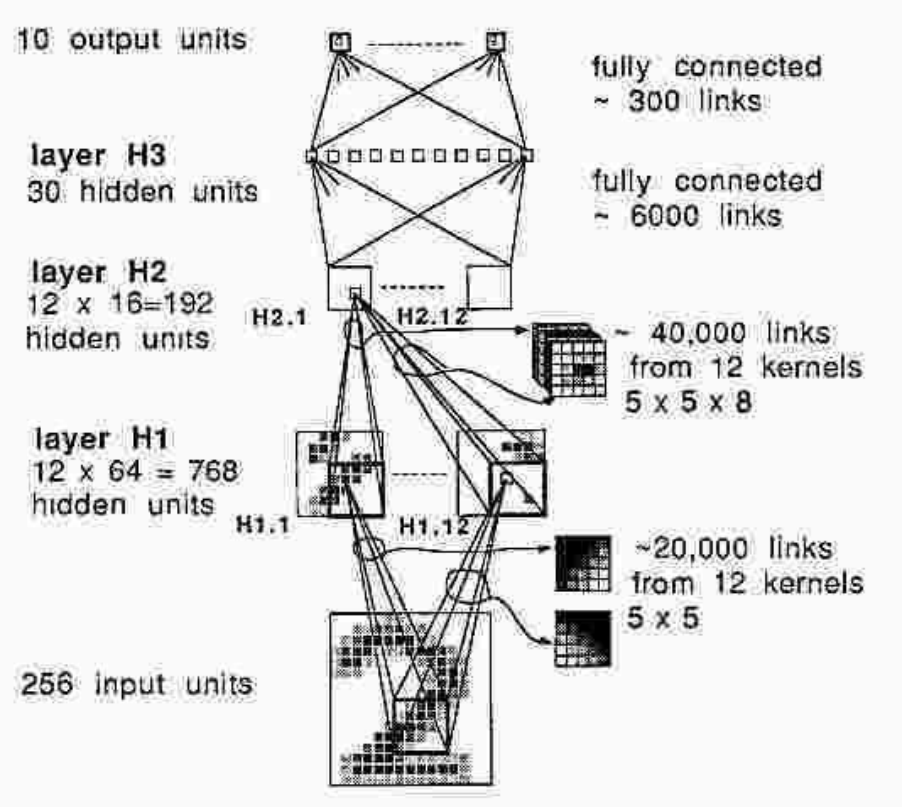
LeNet

A high impact paper that demonstrated end-to-end learning:

- multilayer convolutions for feature extraction
- multilayer perceptrons for the final classification step

Got off to a slow start because:

- computations were costly
- datasets were small
- manually designed features usually still did better



LeCun, Yann, et al. "Backpropagation applied to handwritten zip code recognition." *Neural computation* 1.4 (1989): 541-551.

AlexNet

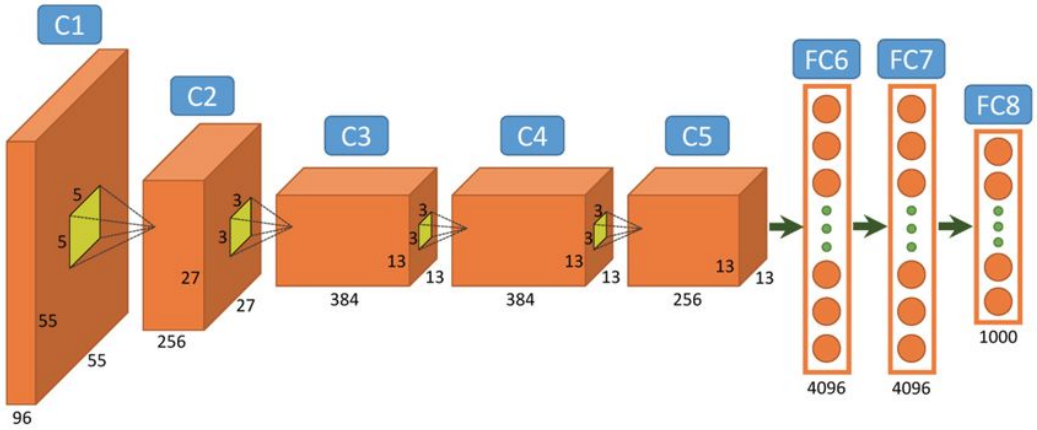
Very similar to LeNet

Massive impact: “Cited by 13136”

Put together a whole bunch of (existing) techniques:

- GPU computing
- ReLU nonlinearities
- pooling layers
- local response normalization
- data augmentation
- dropout

Demonstrated much improved performance



https://www.researchgate.net/figure/312303454_fig2_Fig-4-Architecture-of-Alexnet-From-left-to-right-input-to-output-five-convolutional

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." *Advances in neural information processing systems*. 2012.

Primary Deep Learning Frameworks

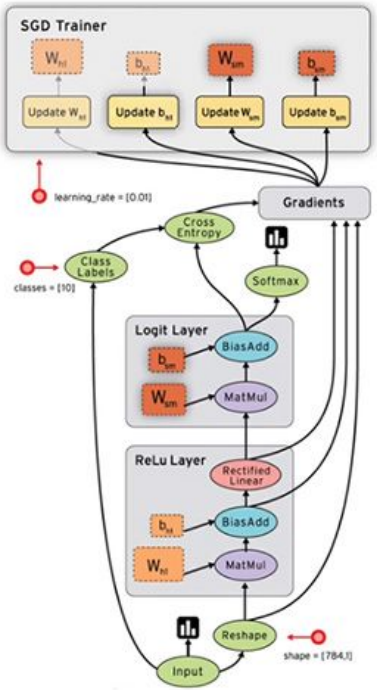
PyTorch

- interactive numerical computing for CPU/GPU
- automatic, dynamic differentiation when desired
- all of Python available at runtime

```
In [1]: import PyTorch

In [2]: # create matrix with 3 rows an 5 columns
        # initialize the values to be evenly random between 0 and 1
        syn0 = PyTorch.FloatTensor(3,5).uniform(0,1)

        # create a matrix with 3 rows and 5 columns
        # initialize the matrix to have all 0.1 values
        syn1 = PyTorch.FloatTensor(3,5).uniform(0.1,0.1)

        # create a vector, initialize it to be all 1s
        l1 = PyTorch.FloatTensor(5).uniform(1,1)

In [3]: # elemntwise scalar addition
        syn1 + 0.1

Out[3]: 0.2 0.2 0.2 0.2 0.2
        0.2 0.2 0.2 0.2 0.2
        0.2 0.2 0.2 0.2 0.2
        [torch.FloatTensor of size 3x5]
```

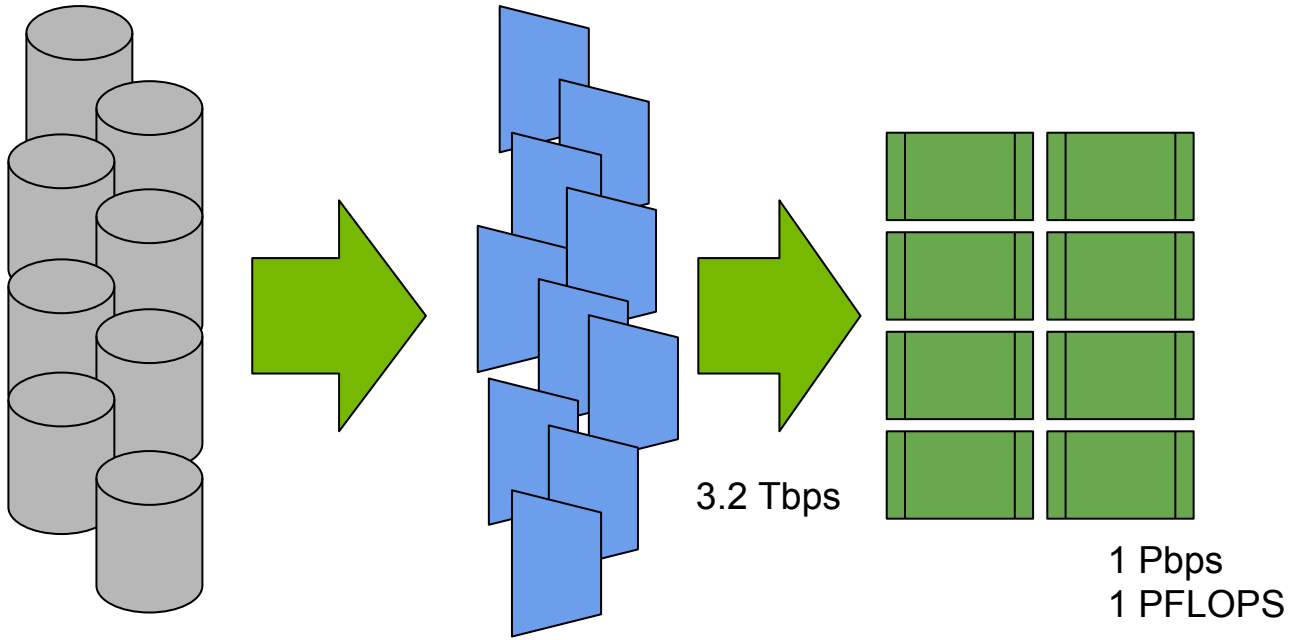
TensorFlow

- compute graph execution engine
- programmatic compute graph construction in Python
- runtime primitives limited to available graph primitives



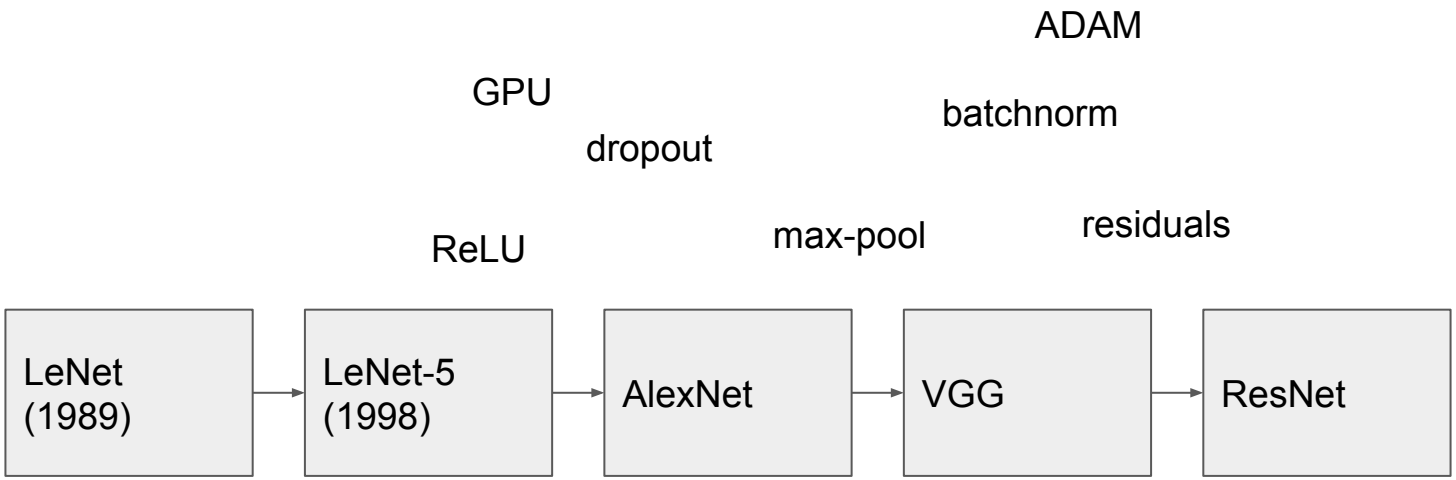
Question: who has used which?

distributed training



IS DL ALL YOU NEED?

domain-independent advances



example: domain-independent reasoning

Dropout:

- training many models and averaging them often gives better results
- this may be viewed as an approximation to a fully Bayesian procedure
- dropping out units and then averaging approximates this behavior

Can we solve all problems in deep learning this way?

No Free Lunch Theorems

If an algorithm performs well on a certain class of problems then it necessarily pays for that with degraded performance on the set of all remaining problems.

Wolpert, David H. "The supervised learning no-free-lunch theorems." *Soft Computing and Industry*. Springer London, 2002. 25-42.

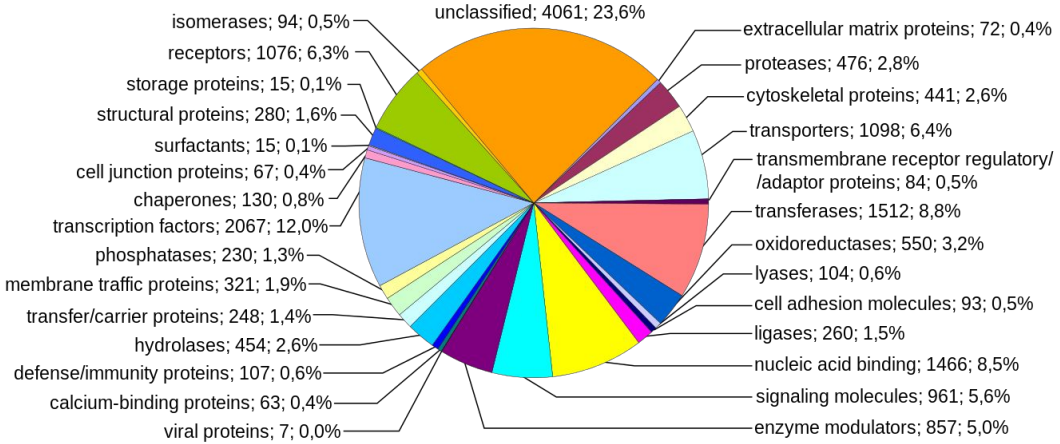
Related of Bayesian Theorems

For most decision rules, you can find some prior that makes that decision rule Bayes-optimal.

Berger, James O. *Statistical decision theory and Bayesian analysis*. Springer Science & Business Media, 2013.

Conclusion: There is no single universal neural network: the domain matters.

Biologist: “It sure looks to me like there is a universal neural network to me.”

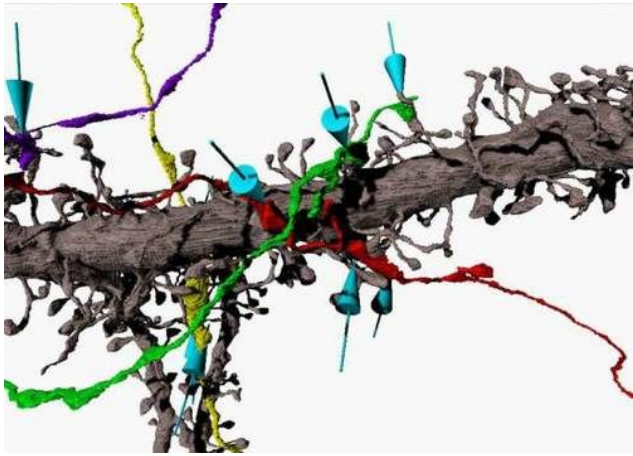


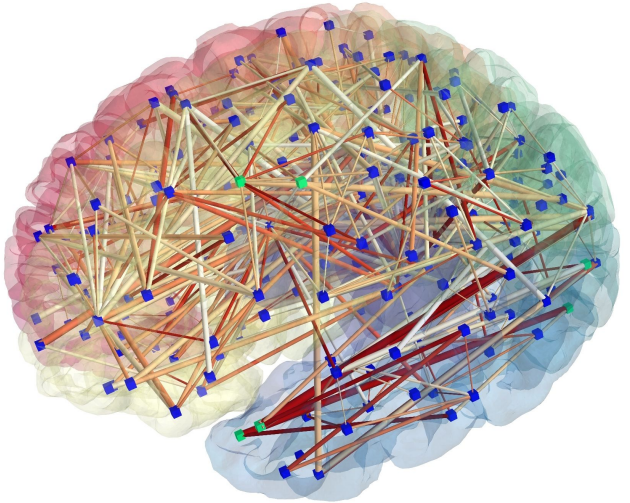
Domain Knowledge in the Brain

Brains / biology encode domain knowledge in…

- large scale connectomics
- replicated microcircuits
- properties of receptors / synapses
- effectors / sensors
- environment and culture

How… we don’t know yet.





Nature has had a billion years.

You have to be smart.

rest of today

- traditional computer vision
  … and why DL is beating it
- computer science view
- pattern recognition view
- signal processing view
- decision theoretic view

There are more: Bayesian, game theoretic, ...

COMPUTER VISION VIEW

Computer Vision (Traditional)

What is a specularity?

Describe the Canny edge detector.

What is a Hough transform?

What is a Bayer pattern?

Edge Detection with Canny

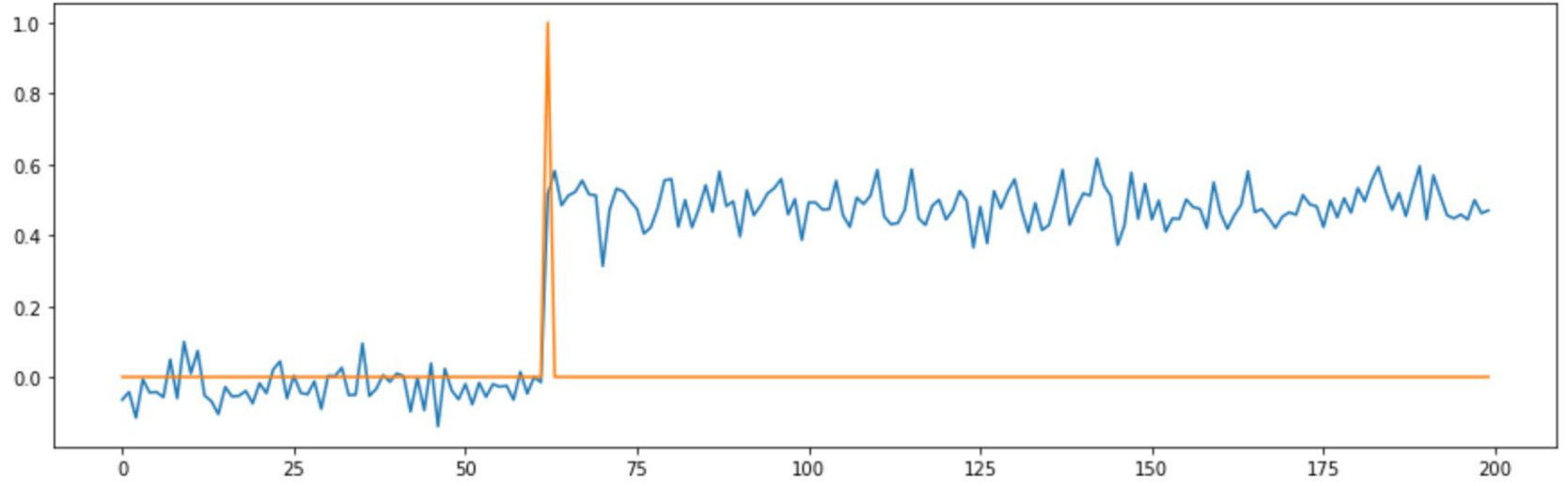
Canny, John. "A computational approach to edge detection." *IEEE Transactions on pattern analysis and machine intelligence* 6 (1986): 679-698. (27000+ citations)

Canny:

- assume noisy step edges
- construct an edge detector using an optimal linear filter

This is actually a simple neural network...

Noisy Step Edge Model

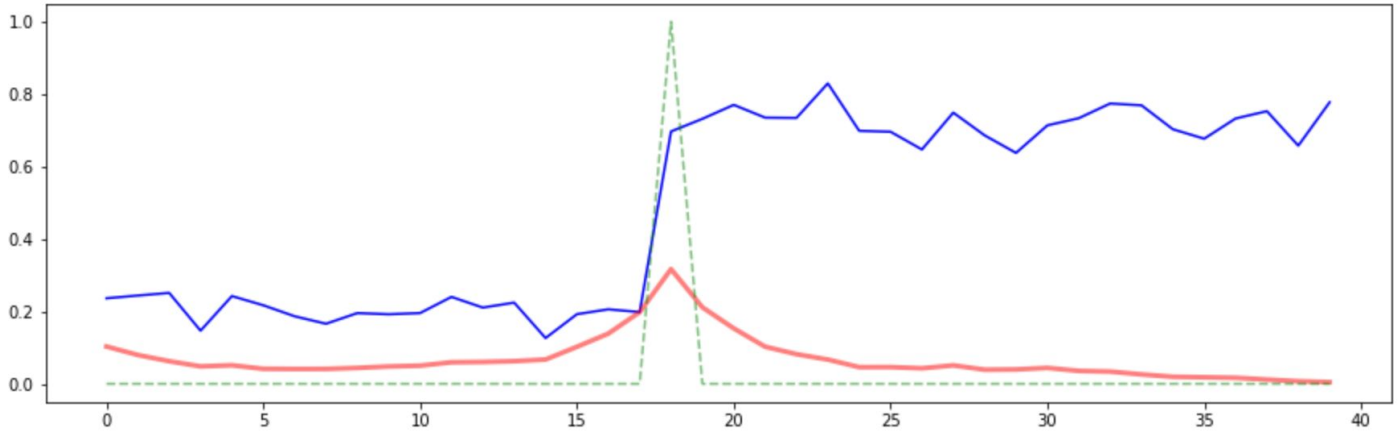


The Canny Model

Cl(1, 33) | Sigmoid()

```
Sequential (
  (0): BSD->BDS
  (1): Conv1d(1, 1, kernel_size=(33,), stride=(1,), padding=(16,))
  (2): BDS->BSD
  (3): Sigmoid ()
)
```

Canny Output

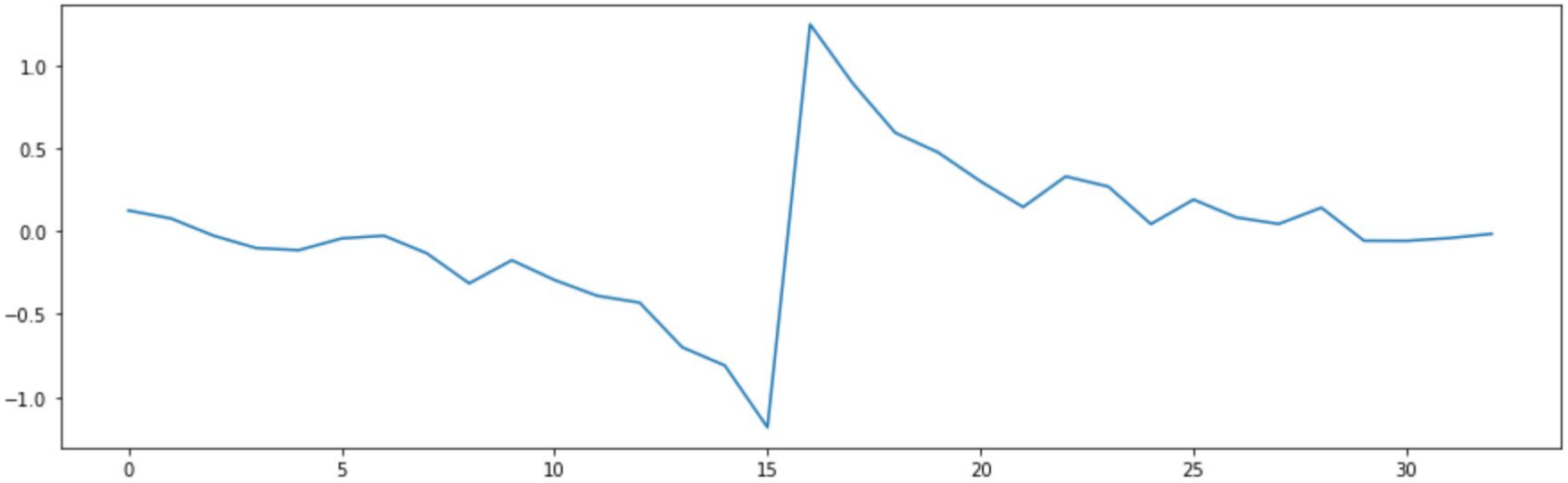


Canny considers different tradeoffs between localization and false alarms.

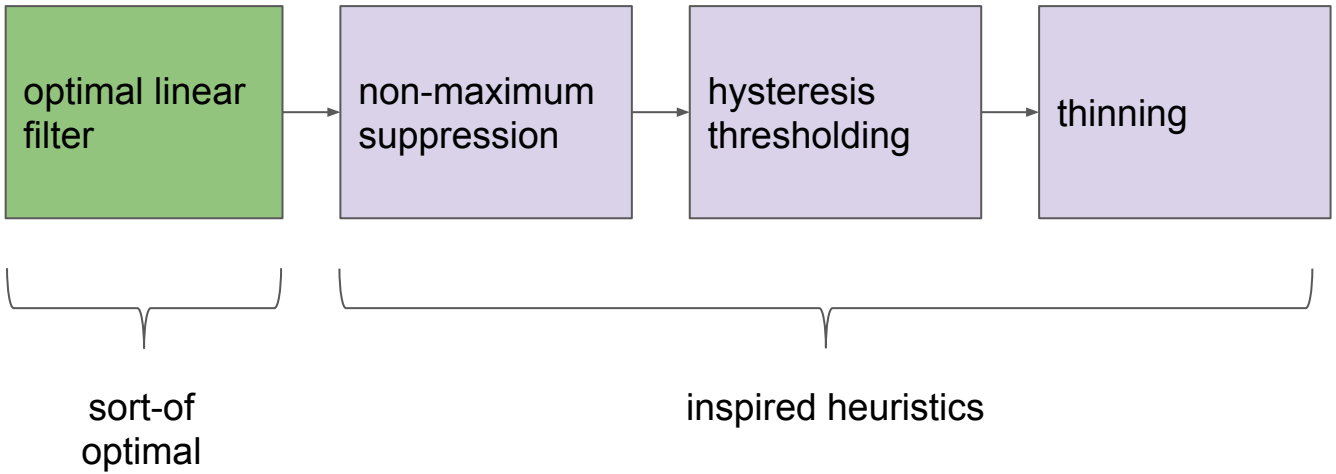
The peak is in the right place here, but the response is fairly wide.

Note the response due to zero boundary conditions (excluded from loss)

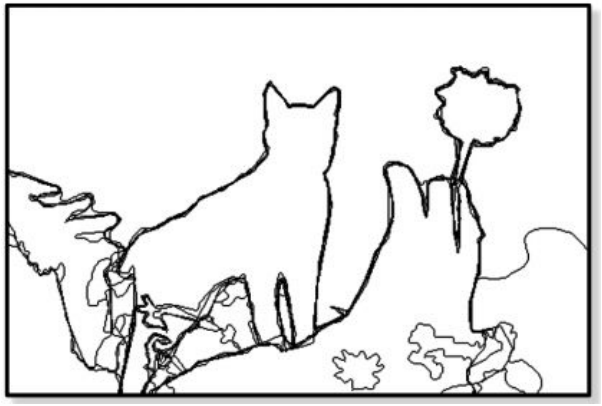
Canny Kernel



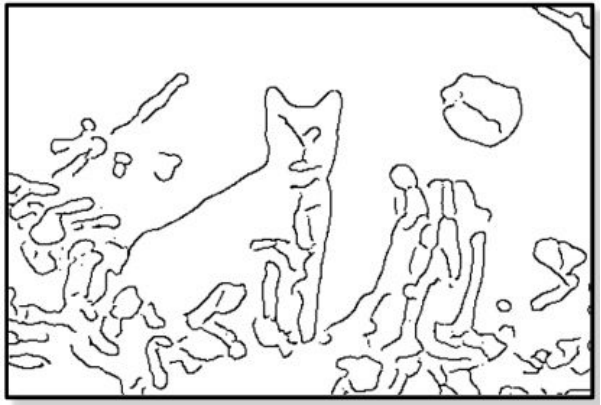
actual Canny edge detector



objective:



Canny:

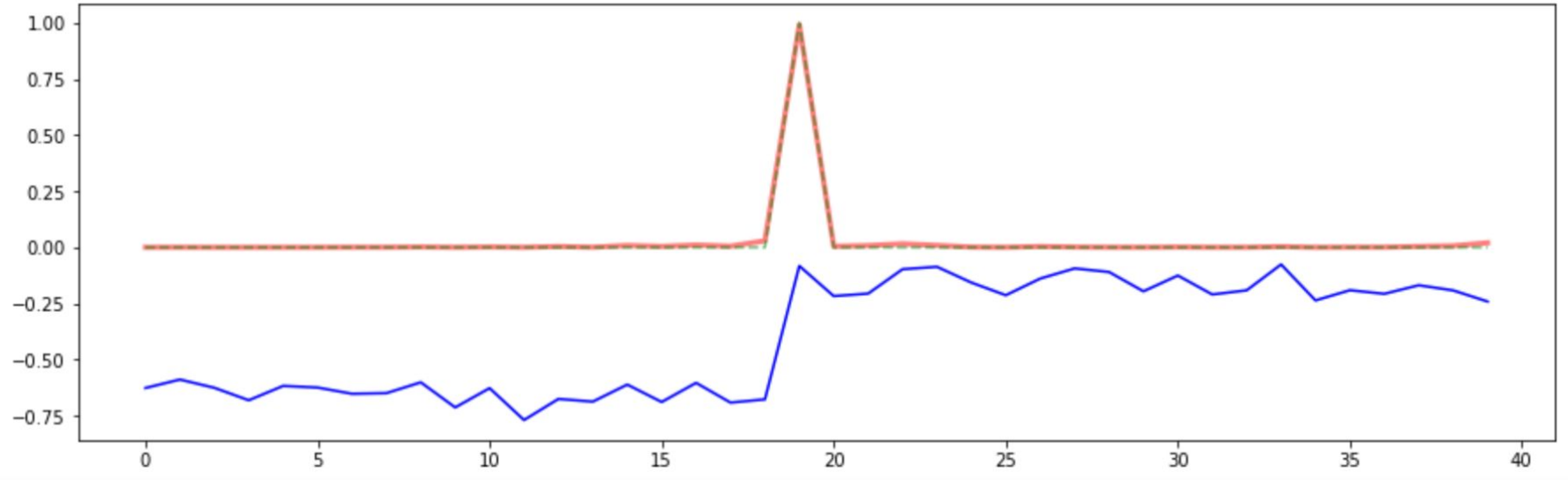


Simple DL Model

Cbr(4, 7) | Cs(1, 7)

```
Sequential (
  (0): BDS->BSD
  (1): Conv1d(1, 4, kernel_size=(7,), stride=(1,), padding=(3,))
  (2): BatchNorm1d(4, eps=1e-05, momentum=0.1, affine=True)
  (3): ReLU ()
  (4): Conv1d(4, 1, kernel_size=(7,), stride=(1,), padding=(3,))
  (5): BSD->BDS
  (6): Sigmoid ()
)
```

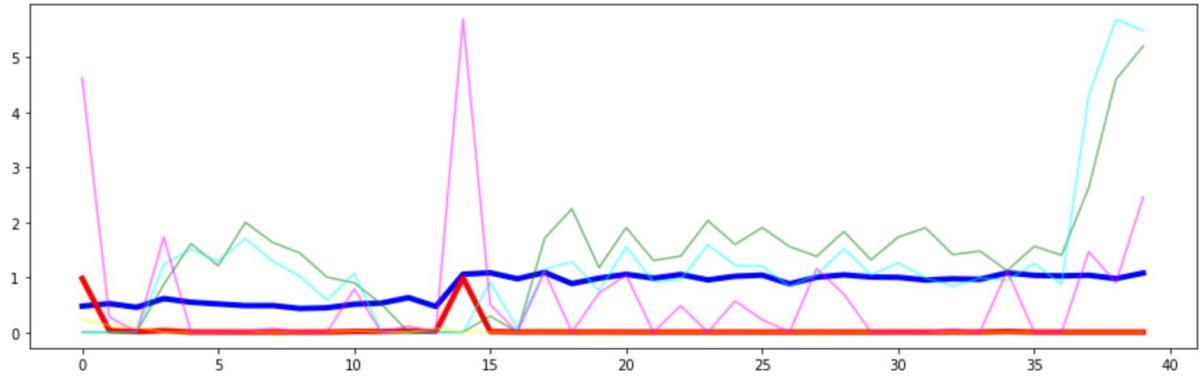
Simple DL Output



How does it work?

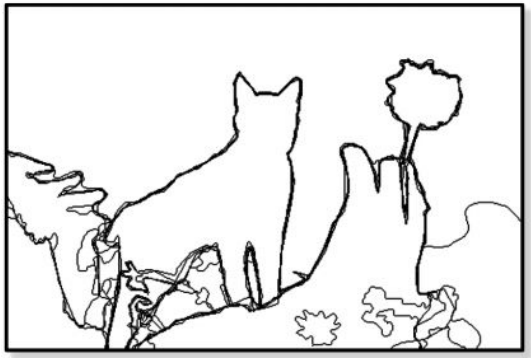
Instead of one filter, train multiple filters, one optimized for localization, the rest optimized for suppression.

Neat trick: Canny could have used it!

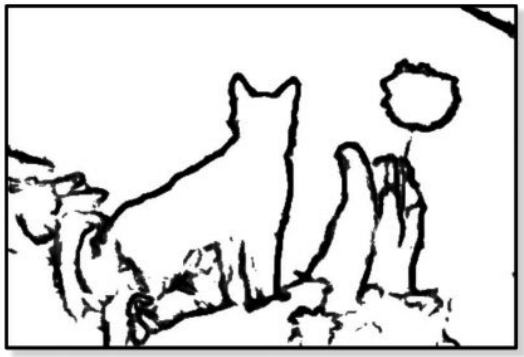


DL Edge Detector

objective:

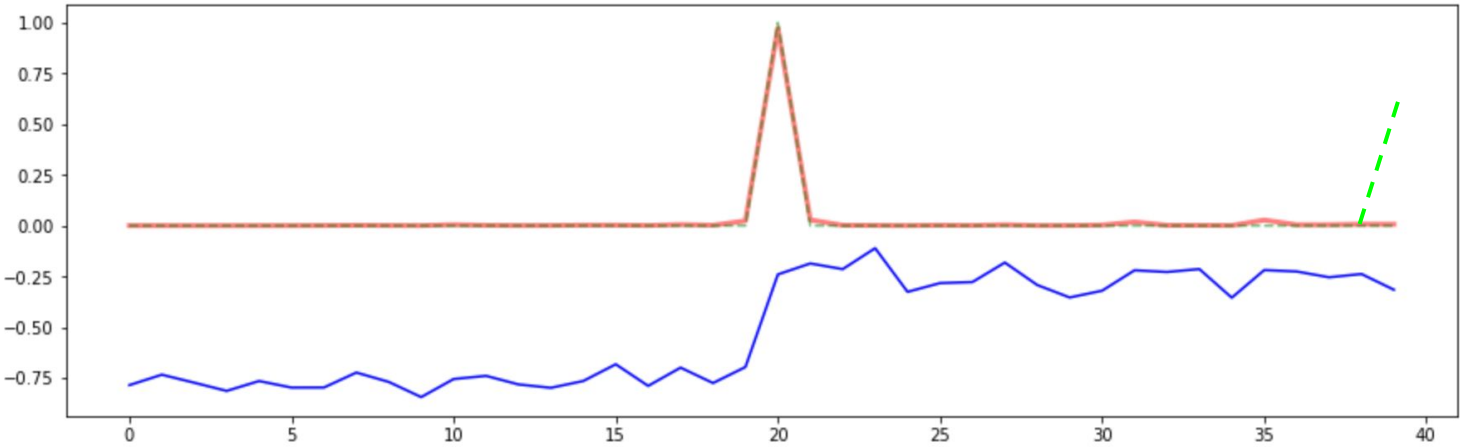


DL:



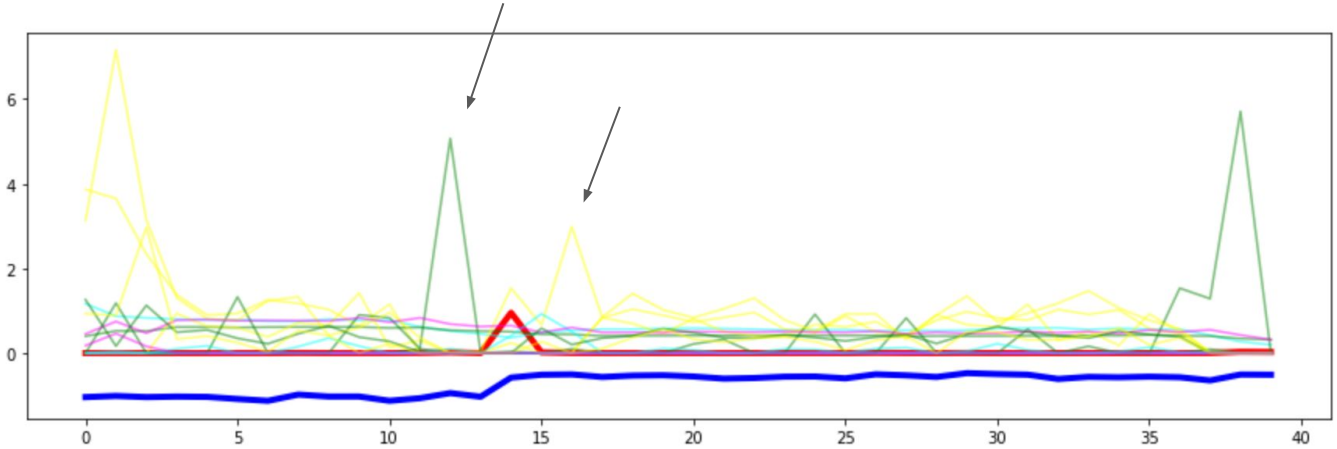
http://www.cv-foundation.org/openaccess/content_iccv_2015/papers/Xie_Holistically-Nested_Edge_Detection_ICCV_2015_paper.pdf

Bad Boundary Conditions



- Cbr(16, 7) | Cs(1, 7)
- Zero padding on the boundary creates spurious edge responses.
- DL can automatically suppress these for you (beware when benchmarking)
- How would you pick weights to get rid of such spurious responses?

Suppression of Spurious Boundary Responses



- DL discovered another neat trick: Instead of a single edge localizer, train two localization filters in the first layer (plus multiple suppression filters again).
- Each localization filter is offset from the desired peak by one pixel.
- The padding on the second convolutional layer means that the spurious edges on the boundary are only surrounded by one peak.

Lessons from Canny Example

Many old style learning algorithms:

- picked a bad model
- then found an optimal solution to that bad model
- assumption: an optimal linear model is better than a suboptimal nonlinear one

The assumption turned out to be wrong:

- Even simple suboptimal nonlinear models outperform optimal models easily.

And… by analyzing deep models carefully, we can get interesting ideas.

Traditional Machine Learning

COMPUTER SCIENCE VIEW

Computer Science

What is the difference between an NP-complete and an NP-hard problem?

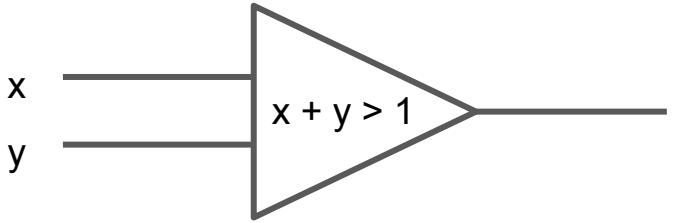
What is computational complexity of matrix multiplication?

Give an example of an uncomputable function.

What is the class of languages recognizable by a pushdown automaton?

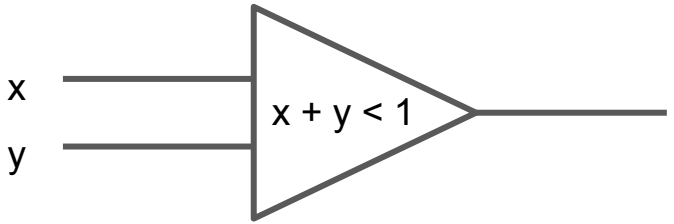
Linear Threshold Units = Logic Gates

$x \wedge y$



$x \wedge y$

$x \vee y$

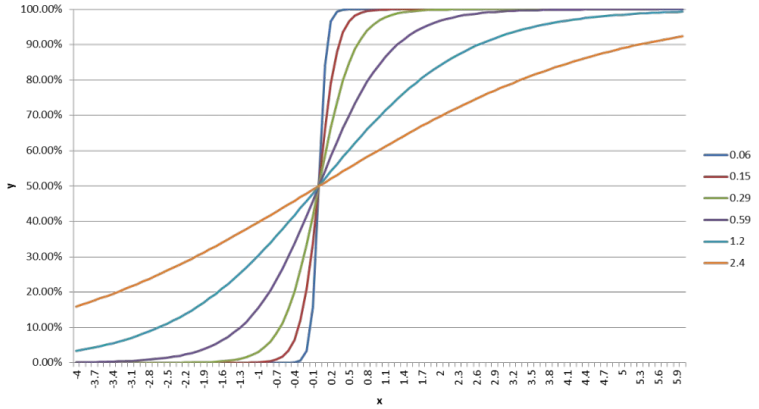


$x \vee y$

sigmoidal units converge to linear threshold units

$$y = \sigma(M \cdot x + b)$$

$$y = \sigma(\alpha M x + \alpha b)$$



$$\lim_{\alpha \to \infty} \sigma(\alpha M x + \alpha b) = \lfloor M x + b > 0 \rfloor$$

neural networks as boolean circuits

Any boolean circuit has a neural network equivalent.

Any boolean formula has a neural network equivalent.

Question: does every neural network have a boolean circuit equivalent?

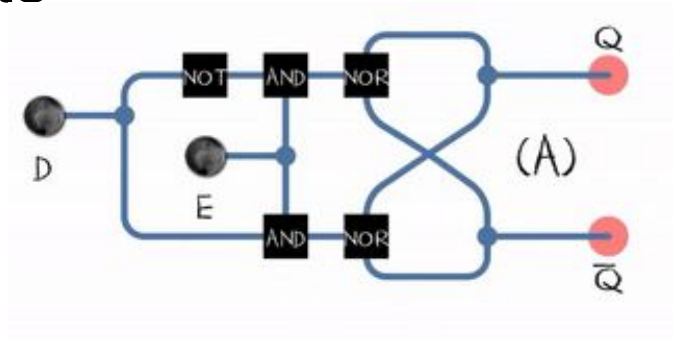
neural networks as boolean circuits

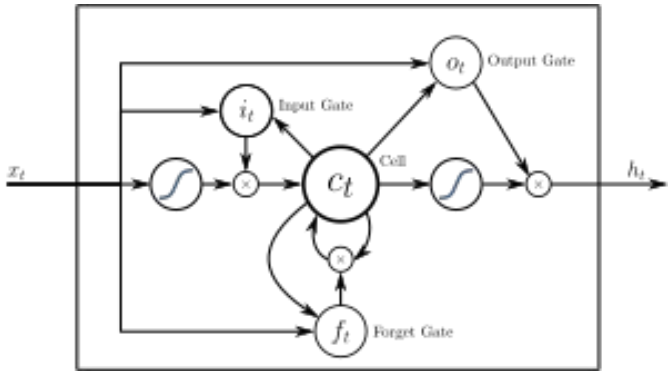
approach:

- start with an algorithm or boolean circuit
- binary → real, and → multiply, or → add+sigmoid

examples:

- perceptrons → deep learning
- one bit storage device → LSTM
- Turing machine → neural Turing machine





Images: Wikipedia

neural networks as boolean circuits

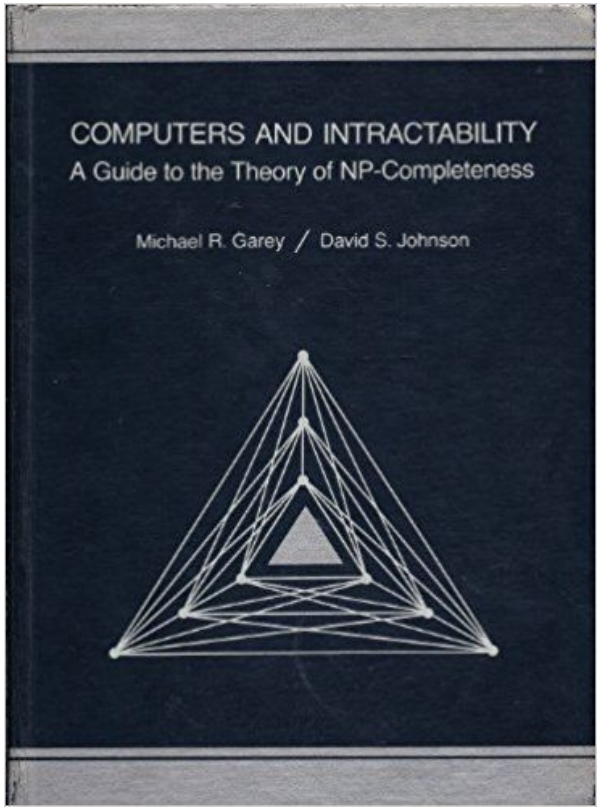
Computer science has powerful results for boolean circuits:

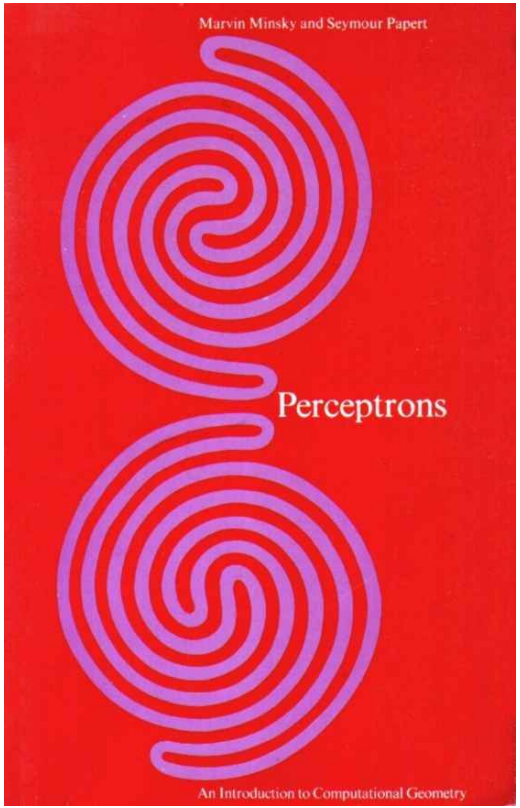
- boolean circuit complexity
- learnability of boolean functions

Do they apply to neural networks / DL networks?

- generally yes (finite precision, noise robustness)
- there are exceptions (e.g., LSTMs using internal counters)
- but even in CS, those results are sometimes weaker than believed

complexity


COMPUTERS AND INTRACTABILITY
A Guide to the Theory of NP-Completeness
Michael R. Garey / David S. Johnson


Marvin Minsky and Seymour Papert
Perceptrons
An Introduction to Computational Geometry

heuristics about complexity classes

is in AC0	useful: it's easy to compute
NP-hard	might still be easy in most cases
NP-hard, no good approximation	might still be easy in applications
requires average circuit depth O(n)	probably a purely sequential problem
not in AC0	can't be solved well by fixed # of common layer types
exponential worst case	might still be easy in most cases
PAC learnable	good indication that it might be learnable by DL system
not PAC learnable	suggestive that it might be hard to learn in practice, not proof
uncomputable	difficult to say (might still be learnable)
Perceptron analysis	relevant to standard DL models

Turing universality … and its risks

Most recurrent neural networks are Turing universal (i.e., a universal computer).

- Proof: by reduction to Rule 110 or directly to Turing Machine
- Caveat: for fixed size and fixed precision, they are, of course finite state machines.

What does this imply?

- some of their properties are uncomputable
- they can compute functions that cannot be learned
- (surprisingly, they can sometimes also learn functions that cannot be computed)
- Question: what does it imply about the “neural Turing machine”?

How does complexity show up in DL?

Example:

- You pick a boolean satisfiability problem.
- You write down an equivalent neural network but use sigmoids and scalable weights.
- You try to solve the problem by SGD.
- Why doesn’t this get around the NP-hardness of satisfiabilty?

Answer:

Computational complexity problems turn into exponential numbers of local minima.

summary: complexity results

Complexity results are tricky: the are often misinterpreted (“NP-hard = intractable”) but also often not paid enough attention to (“not in AC0”, average case).

Complexity results can give us useful insights into the structure of a problem.

Complexity issues often show up as large numbers of local minima (learning) or simply bad results (inference).

Attempted workarounds (graduate non-convexity, algebraic continuation, annealing, etc.) don’t solve the problem, but may result in good approximation algorithms.

DL as a computational tool

We can use DL as a computational tool, without any data.

Example:

- want to invert a function $f(x)$ over some domain Ω
- sample $x \in \Omega$ according to an interest distribution
- compute $y = f(x)$ for each sample
- train a DL model g using y as input and x as output

Model Equation	Learned Model	Numerical Algorithms
$y = f(x)$	$y = \tilde{g}(x)$	function approximation, interpolation
$y = f(x) + \nu$	$\hat{x} = \tilde{g}(y)$	inversion, ill-posed problems, estimators
$f(x, y) = 0$	$\hat{x} = \tilde{g}(y, \nu)$	binary relation solver
$\frac{\partial u_i}{\partial^r x_j} = f_{ijr}(u, x)$	$u(x) = \tilde{g}(x)$	ODE, PDE solvers
$\hat{x} = \arg\max f(x)$	$\hat{x} = \tilde{g}^{\infty}(x_0)$	optimization
$(x, \omega) \sim p(x, \omega)$	$P(\omega \mid x)$	statistical testing

PATTERN RECOGNITION VIEW

Pattern Recognition (and Old Style ML)

What form do discriminant functions take for normal class conditional densities?

What is the purpose of the PCA transform?

Give a commonly used fast approximate algorithm for nearest neighbor lookup.

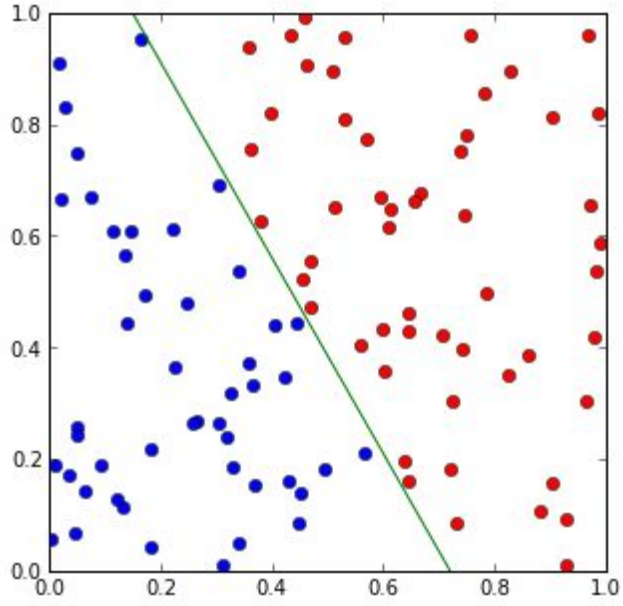
What is the bias/variance tradeoff?

fully connected layers

$$y = \sigma(M \cdot x + b)$$

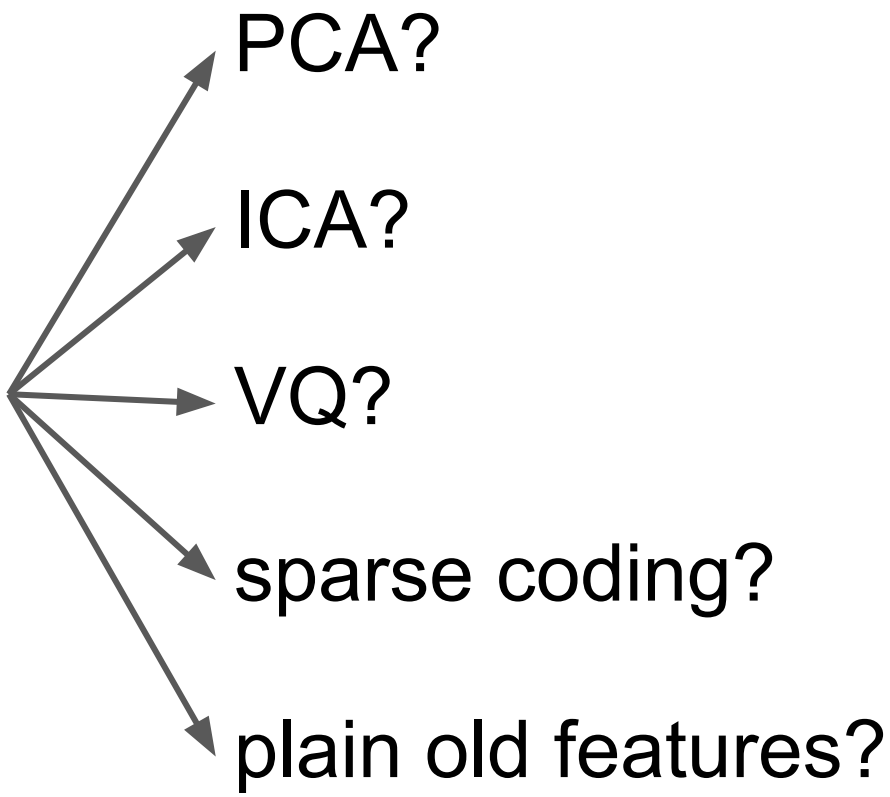
I assume this has been covered at length:

- linear classifiers
- logistic regressors



What are neural network layers like?

fully connected layer =



Why is a raven like a writing desk?

plain old features

Convolutional layers can implement common features extractors.

Reasoning: “**If it’s useful for solving the problem and our optimization algorithm works well enough, then the neural network will produce these where necessary.**”

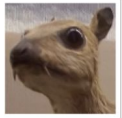
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur (approximation)	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

Figure 1. Example image processing kernels and the output on an example image of some vermin-like creature. Source: wikipedia entry on image kernels.

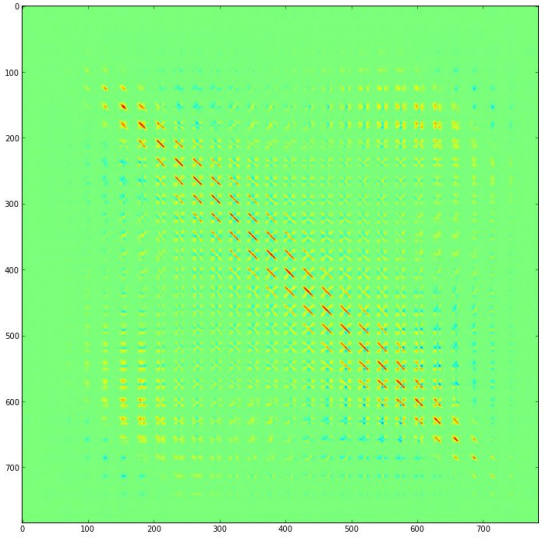
PCA

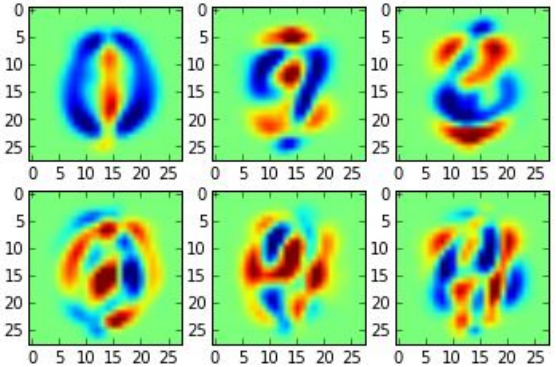
PCA is a linear decomposition of an input signal, a form of dimensionality reduction:

- best linear approximation to the data
- separates signal from noise (why?)
- tends to make the data Gaussian (why?)

Implementation:

- linear autoencoder or autoencoder in lin. regime
- 1000 dim → Fl(20) | Fl(1000) → 1000 dim
- Question: what's the difference between PCA and a linear autoencoder?





PCA and information flow

PCA can be viewed as a simple way of separating signal from noise.

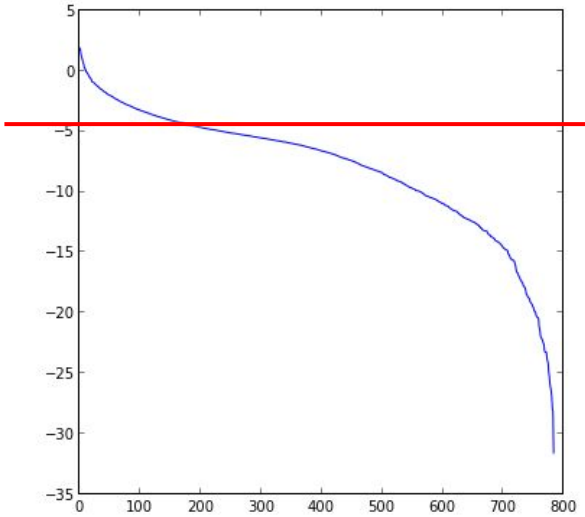
We assume that the signal has a much higher variance than the noise.

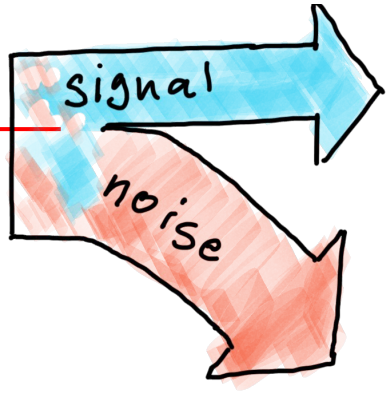
Cutting off the directions corresponding to small eigenvalues then removes the noise and little of the signal.

$$\text{observation} = \text{signal} + \text{noise}$$

```
figsize(6,6)
plot(sorted(log(evals),reverse=1))
```

```
[<matplotlib.lines.Line2D at 0x3a27ed0>]
```





ICA (Independent Component Analysis)

- a linear decomposition, like PCA
- identifies non-Gaussian components
- maximizes the entropy of the decomposition
- can separate independent sources

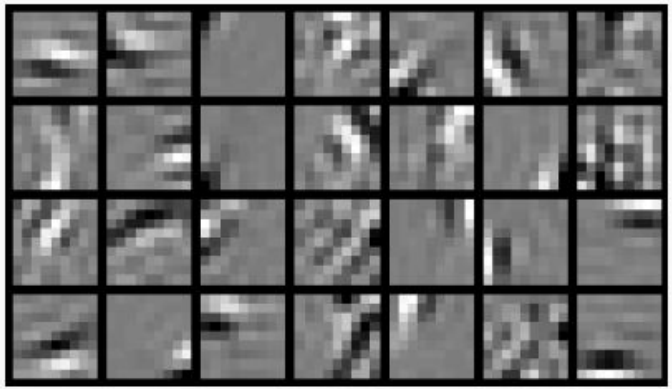
$$\arg\min_W ||Wx||_1 \, , \, WW^T = I$$

$$\arg\min_W \lambda|Wx||_1 + ||W^T Wx - x||_2^2$$

$$\arg\min_W \lambda||\sigma(Wx)||_1 + ||\sigma(W^T\sigma(Wx)) - x||_2^2$$

RICA ≈ sparse autoencoder in linear regime

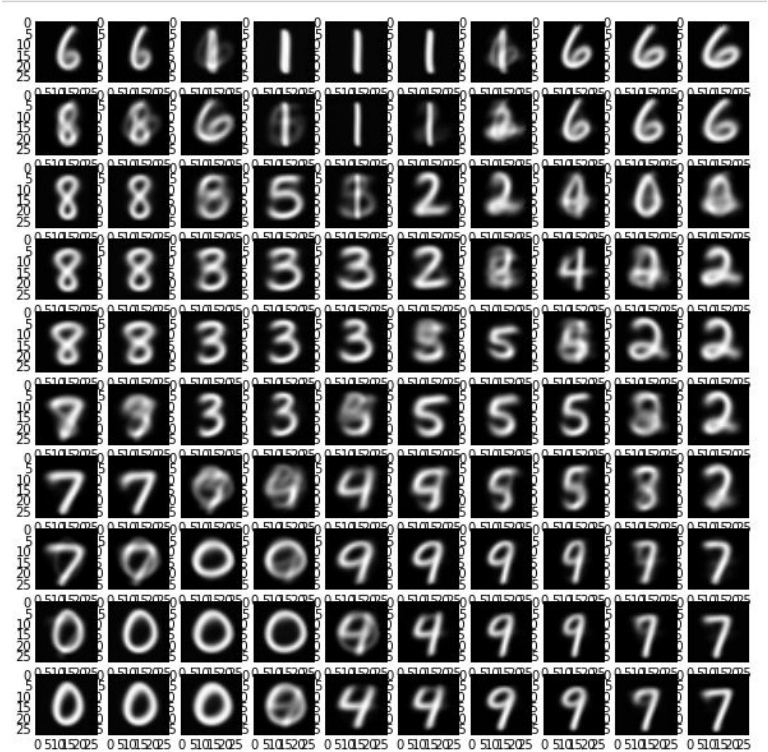
- Shape(1000) | Fl(20) | Fl(1000) + L1 loss
- Shape(1000) | Fs(20) | Fs(1000) + L1 loss
  small weights
- input should be ZCA “whitened”



k-means vs ICA

k-means can be viewed as a special case of ICA:

- require exactly one component
- reconstruction weight = 1
- input vectors are normalized (or augmented with norm)



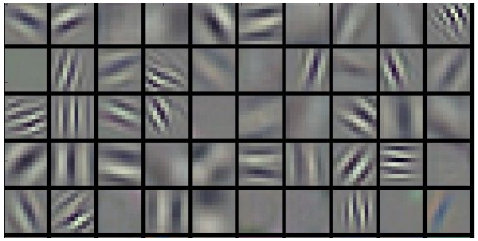
Vinnikov, Alon, and Shai Shalev-Shwartz. "K-means recovers ICA filters when independent components are sparse." *International Conference on Machine Learning*. 2014.

filter banks and image statistics

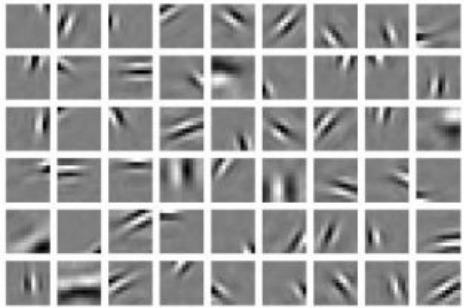
AlexNet input features look like a mix of DCT / PCA / ICA

No coincidence:

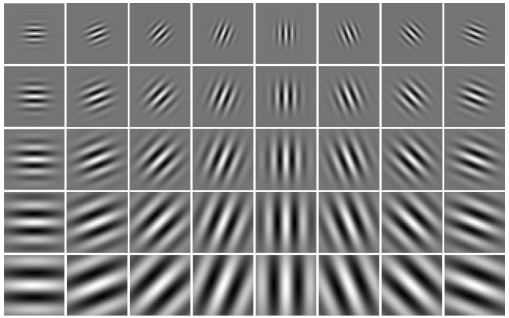
- image statistics
- mutual information



AlexNet



sparse linear decomposition



Gabor Features

Simoncelli, Eero P., and Bruno A. Olshausen. "Natural image statistics and neural representation." *Annual review of neuroscience* 24.1 (2001): 1193-1216.

DCT, PCA, ICA

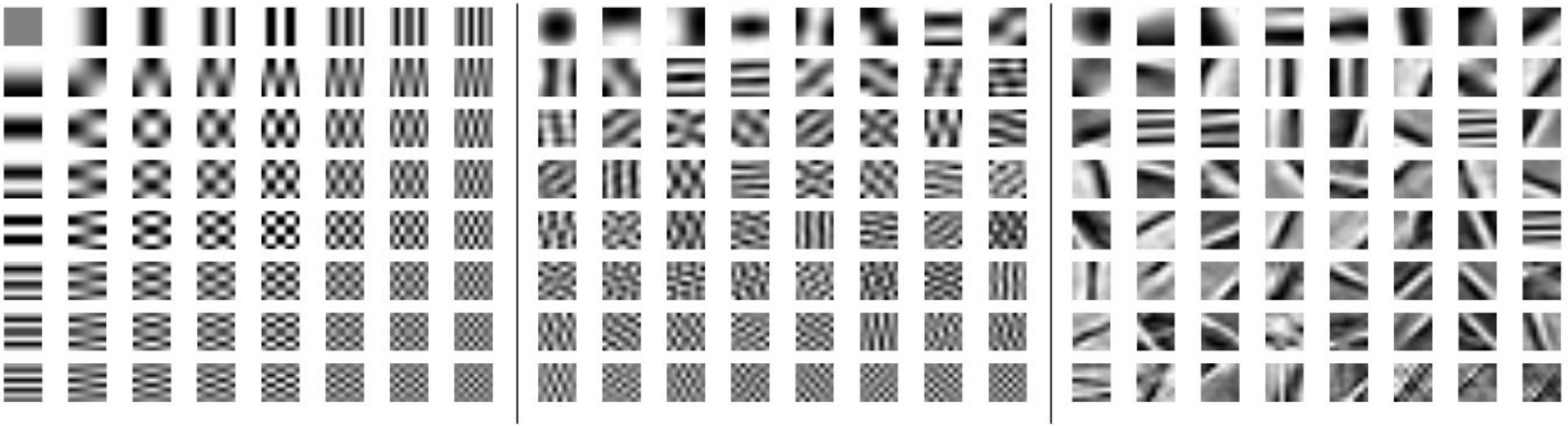


Fig. 3. Basis functions for DCT (left), PCA (center) and ICA (right).

Vasconcelos, Manuela, and Nuno Vasconcelos. "Natural image statistics and low-complexity feature selection." *IEEE Transactions on Pattern Analysis and Machine Intelligence* 31.2 (2009): 228-244.

a neural network layer...

A neural network layer is like *all of the above*: PCA, ICA, VQ (and more)

Which of these decompositions happens depends on:

- loss function (often: mutual information or entropy maximization)
- input distribution (normalize, whitened, …)
- weight magnitude (weight penalties)
- stride and pooling (for convolutional layers)

There is no “single theory” even for linear methods in pattern recognition; don’t expect there to be one for DL!

Why is a raven like a writing desk?

Because it can produce a few notes, though they are very flat; and it is NEVAR put with the wrong end in front!

Because Poe wrote on both.

Because both have inky quills.

I have no idea!

Manifolds!

Manifolds are talked about a lot in both DL and ML.

As far as DL/ML applications are concerned, a modest generalization of linear techniques (“locally, everything looks approximately linear most off the time”).

That’s why we’re going to look in much more detail at the linear theory later.

SIGNAL PROCESSING VIEW

Signal and Image Processing

What does an anti-aliasing filter do in the frequency domain?

What is an impulse response?

What is the complexity of convolution when using the FFT?

What is a morphological closing?

convolutions

convolutions are linear operations on the input

convolutional layers are just a special case of a fully connected layer

- parameter tying
- a special matrix structure (Toeplitz matrix)

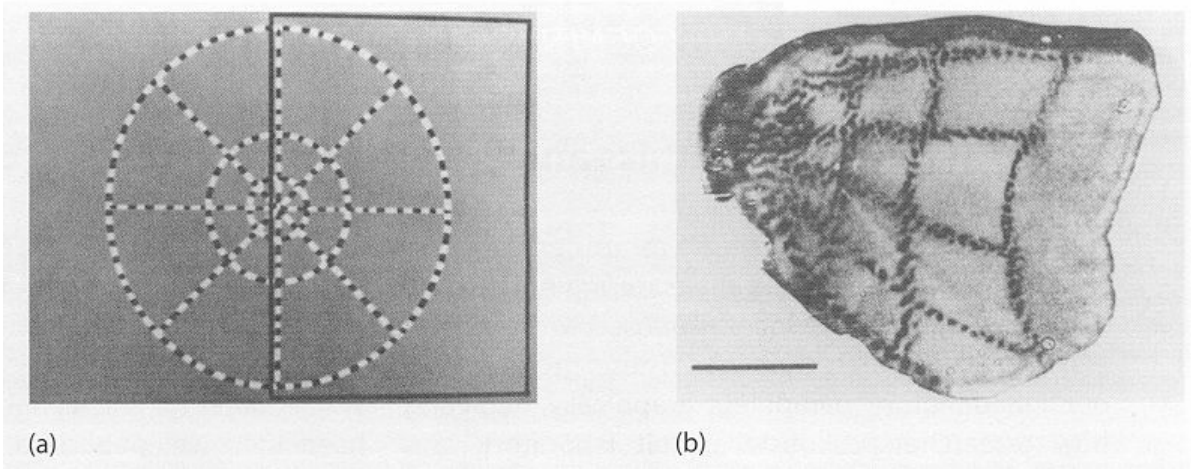
$$y = h * x = \begin{bmatrix} h_1 & 0 & \dots & 0 & 0 \\ h_2 & h_1 & \dots & \vdots & \vdots \\ h_3 & h_2 & \dots & 0 & 0 \\ \vdots & h_3 & \dots & h_1 & 0 \\ h_{m-1} & \vdots & \dots & h_2 & h_1 \\ h_m & h_{m-1} & \vdots & \vdots & h_2 \\ 0 & h_m & \dots & h_{m-2} & \vdots \\ 0 & 0 & \dots & h_{m-1} & h_{m-2} \\ \vdots & \vdots & \vdots & h_m & h_{m-1} \\ 0 & 0 & 0 & \dots & h_m \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix}$$

$$y^T = [\, h_1 \quad h_2 \quad h_3 \quad \dots \quad h_{m-1} \quad h_m \,] \begin{bmatrix} x_1 & x_2 & x_3 & \dots & x_n & 0 & 0 & 0 & \dots & 0 \\ 0 & x_1 & x_2 & x_3 & \dots & x_n & 0 & 0 & \dots & 0 \\ 0 & 0 & x_1 & x_2 & x_3 & \dots & x_n & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \dots & \vdots & \vdots & \dots & 0 \\ 0 & \dots & 0 & 0 & x_1 & \dots & x_{n-2} & x_{n-1} & x_n & \vdots \\ 0 & \dots & 0 & 0 & 0 & x_1 & \dots & x_{n-2} & x_{n-1} & x_n \end{bmatrix}.$$

convolutional layers from data

fully connected layers can learn the Toeplitz structure from data:

- train on translation invariant (or translation augmented data)
- this is the underlying mechanism of retinotopic mappings and feature map construction in brains
- note that retinas have non-uniform resolutions → uniform convolutions wouldn't work anyway

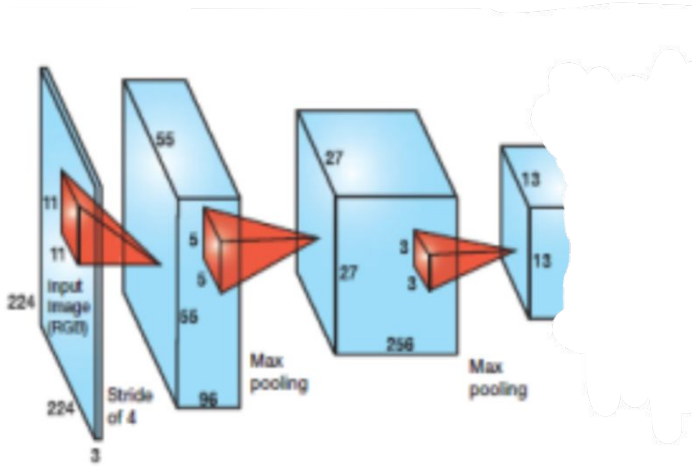


reasoning about footprint

given a stack of convolutional layers, pixels in the output (potentially) depend only a subset of the pixels of the input

this is the “footprint”

is the footprint large enough to solve your problem?



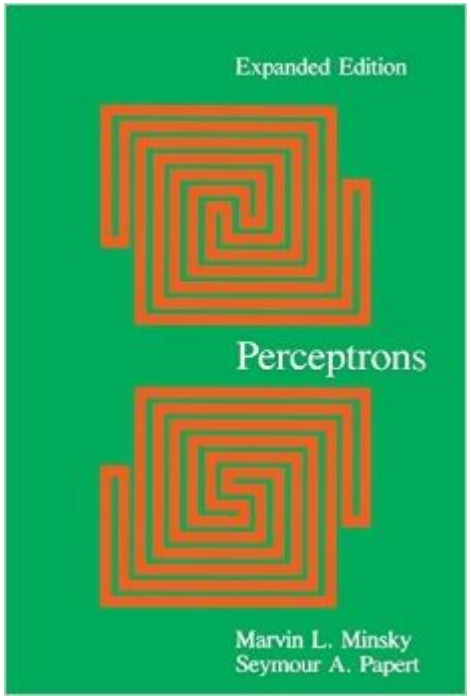
reasoning about footprint (multiple layers)

one of these figures is connected, the other is not

can a deep convolutional network solve this problem?

how would you reason this out?

(a classic problem in complexity of neural networks)



A Mystery

CIFAR-10 49 results collected

Units: accuracy %

Classify 32x32 colour images.

Result	Method	Venue
96.53%	Fractional Max-Pooling	arXiv 2015
95.59%	Striving for Simplicity: The All Convolutional Net	ICLR 2015
94.16%	All you need is a good init	ICLR 2016

Model		
Strided-CNN-C	ConvPool-CNN-C	All-CNN-C
Input 32 × 32 RGB image		
3 × 3 conv. 96 ReLU 3 × 3 conv. 96 ReLU with stride $r = 2$	3 × 3 conv. 96 ReLU 3 × 3 conv. 96 ReLU 3 × 3 conv. 96 ReLU	3 × 3 conv. 96 ReLU 3 × 3 conv. 96 ReLU
	3 × 3 max-pooling stride 2	3 × 3 conv. 96 ReLU with stride $r = 2$
3 × 3 conv. 192 ReLU 3 × 3 conv. 192 ReLU with stride $r = 2$	3 × 3 conv. 192 ReLU 3 × 3 conv. 192 ReLU 3 × 3 conv. 192 ReLU	3 × 3 conv. 192 ReLU 3 × 3 conv. 192 ReLU
	3 × 3 max-pooling stride 2	3 × 3 conv. 192 ReLU with stride $r = 2$

:

- With careful training, max pooling can be replaced by convolutions with stride.
- That's because for known distributions, linear filters + nonlinearities can approximate common non-linear filters.
- Question: Is it worth it?

Springenberg, Jost Tobias, et al. "Striving for simplicity: The all convolutional net." *arXiv preprint arXiv:1412.6806* (2014).

convolutional layers can implement nonlinear filters

standard nonlinear filters

- median
- percentile
- morphology

$$y = \sigma(F * x + b)$$
$$|F_{ij}| \gtrsim 1$$

main insight:

- convolutional layers cannot implement such filters in general
- but: for a given input distribution, they often can give good approximations
- batch norm may force the distribution sufficiently

We'll look at this in much more detail later as well.

DECISION THEORETIC VIEW

Decision Theory

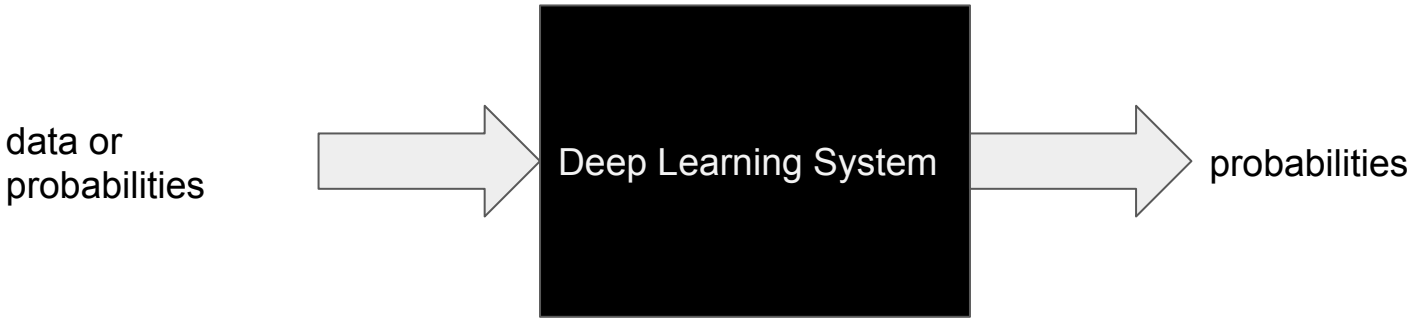
What is a decision-theoretic loss function (as opposed to a DL loss function)?

How does Bayesian parameter estimation differ from ML parameter estimation?

How are HMMs related to graphical models?

What is an inadmissible decision procedure?

DL system as black box



risk = expected loss

In general, we want to minimize expected loss. We call expected loss the *risk* of a decision.

Just like there are conditional probabilities, there are conditional risks.

$$R(\alpha_1 | x) = \lambda_{11} P(\omega_1 | x) + \lambda_{12} P(\omega_2 | x)$$

example: self-driving car

probability estimates
from neural network

P(pedestrian|image) = 0.77

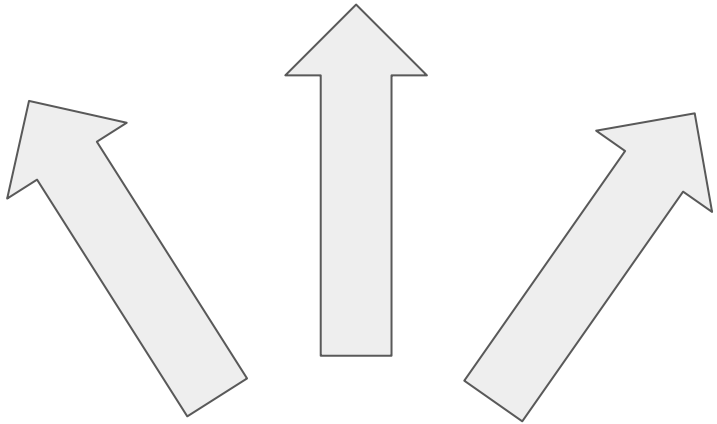
pedestrian dies: $25M

P(car|image) = 0.95

property damage: $500k

P(tree|image) = 0.98

driver dies: $50M



example: biased training set

Start again with a classification problem:

$$p(\omega, x), \quad \omega \in \{0, 1\}, \quad x \in \mathbb{R}^n$$

The prior probabilities are:

$$P(\omega) = \int p(\omega, x) dx$$

Assume our training set is highly biased, so that

$$P(\omega = 1) \ll P(\omega = 0)$$

Networks have a hard time balancing the training set, so we resample the training set such that:

$$P'(\omega = 1) \approx P'(\omega = 0)$$

example: biased training set

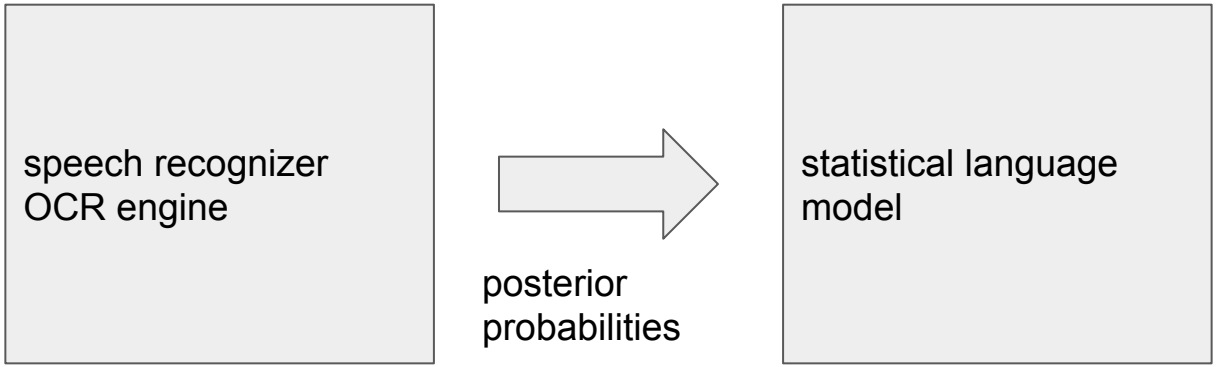
Now we train a deep network on the resampled training set, giving us posterior probability estimates:

$$P'(\omega|x)$$

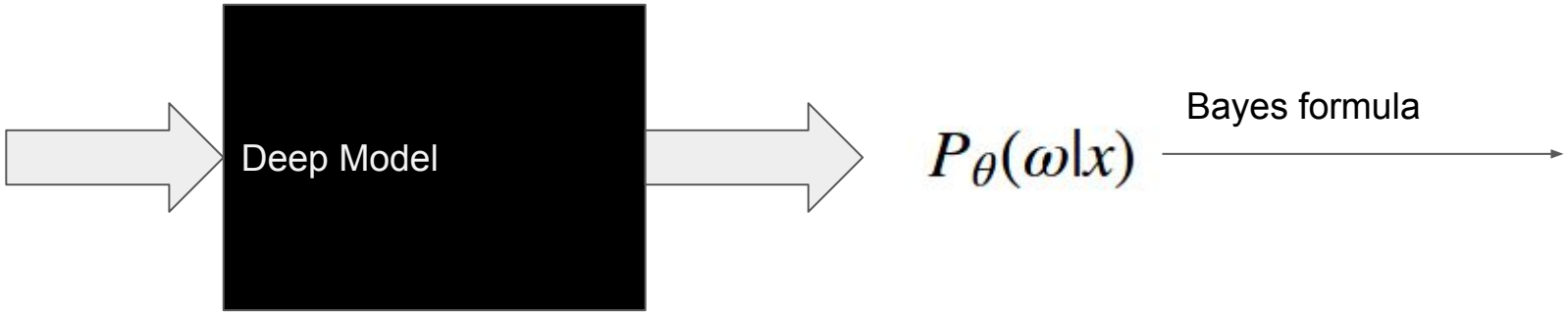
We can convert these into estimates for the original problem by applying Bayes' formula:

$$P(\omega|x) = P'(\omega|x)\frac{P(\omega)}{P'(\omega)}$$

example: language modeling



Bayesian decision theory



$$\hat{\theta} = \arg\max_{\theta} \prod p_\theta(\omega_i, x_i)$$

Bayesian neural networks

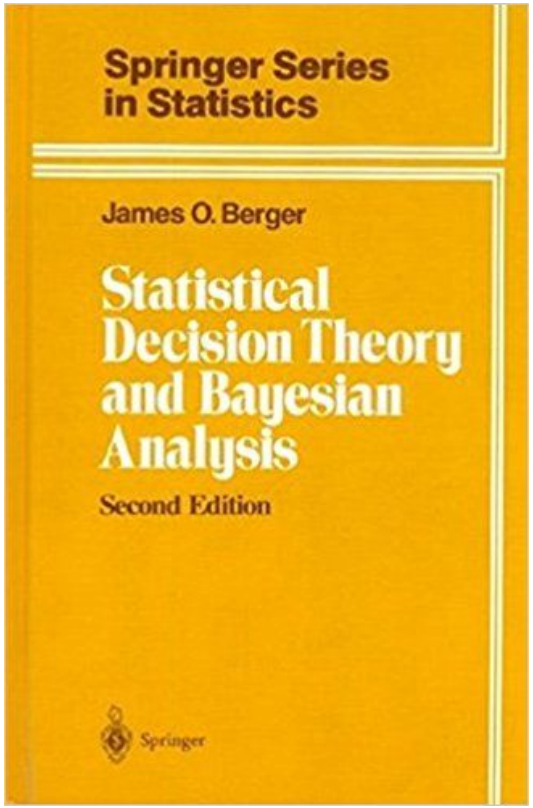
$$\tilde{p}(\omega|x) = \int p_\theta(\omega|x)\, \tilde{p}(\theta|\mathcal{D})\, d\theta$$

$$\tilde{p}(\theta|\mathcal{D}) \quad \text{where } \mathcal{D} = \{(\omega_1, x_1), \ldots, (\omega_N, x_N)\}$$

Could only scratch the surface here.

Recommendation: read Berger.

It’s not just for “weird neural network architectures”.



Conclusions

You need to wear many hats.