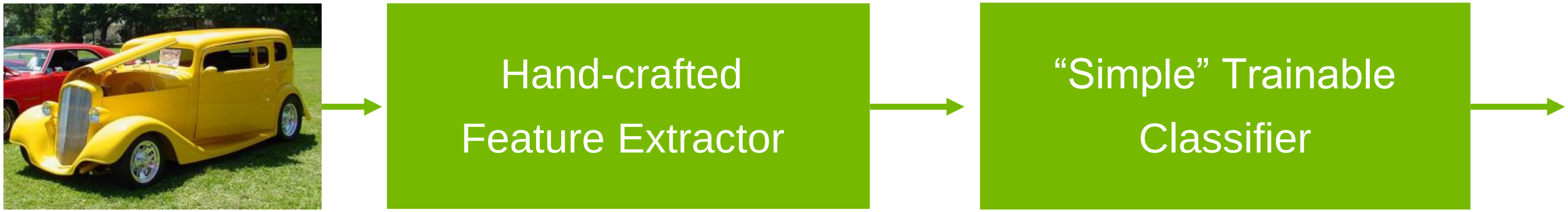
Linear Machines, Feature Learning and Deep Learning Basics

Y.LeCun and MA Ranzato

Deep learning =
Learning representations/features

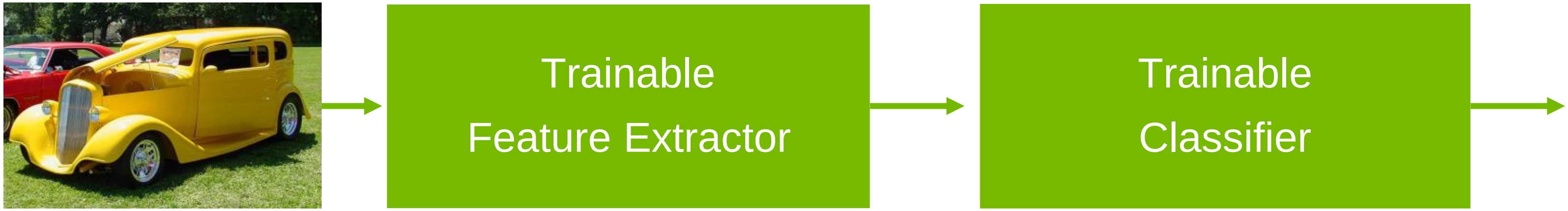
- The traditional model of pattern recognition (since the late 50's)
  - Fixed/engineered features (or fixed kernel) + trainable classifier



Hand-crafted Feature Extractor

“Simple” Trainable Classifier

- End-to-end learning / Feature learning / Deep learning
  - Trainable features (or kernel) + trainable classifier

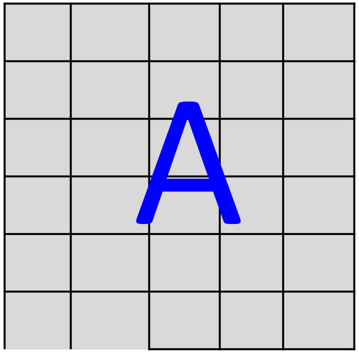


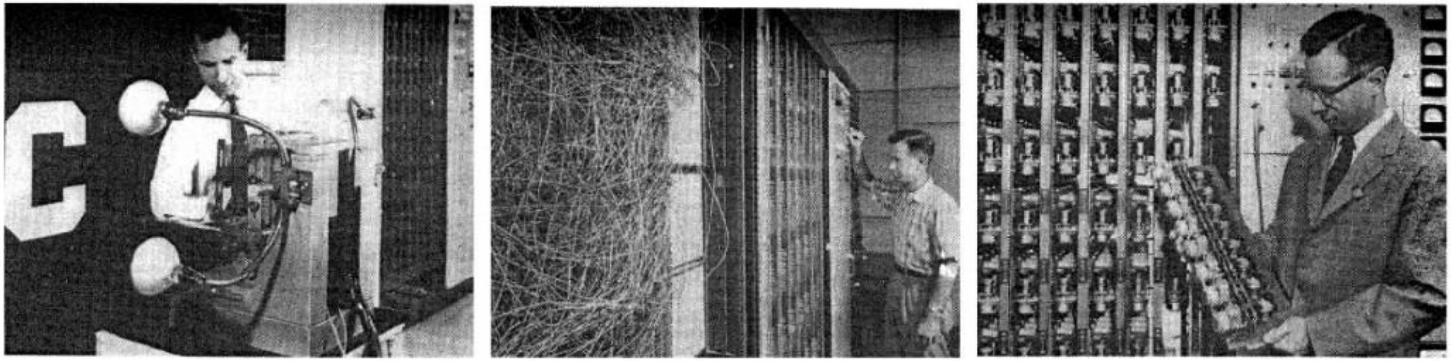
Trainable Feature Extractor

Trainable Classifier

This basic model has not evolved much since the 50's

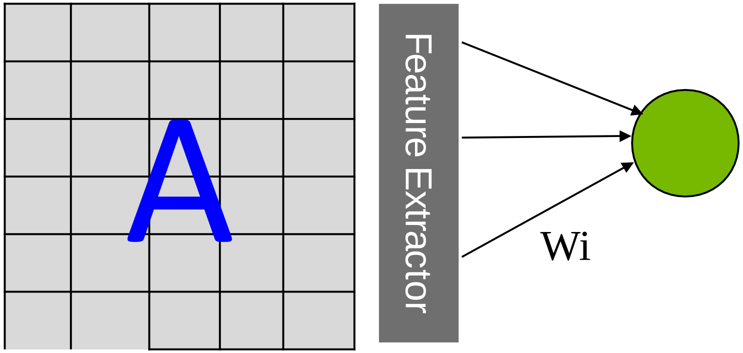
- The first learning machine: the Perceptron
  - Built at Cornell in 1958
- The perceptron was intended to be a machine, rather than a program,
- Its first implementation was in software for the IBM 704, it was subsequently implemented in custom-built hardware as the "Mark 1 perceptron".
- This machine was designed for image recognition: it had an array of 400 photocells, randomly connected to the "neurons". Weights were encoded in potentiometers, and weight updates during learning were performed by electric motors





This basic model has not evolved much since the 50's

- The Perceptron was a linear classifier on top of a simple feature extractor
- The vast majority of practical applications of ML today use glorified linear classifiers or glorified template matching.
- Designing a feature extractor requires considerable efforts by experts.



$$y = sign\left(\sum_{i=1}^{N} W_i F_i(X) + b\right)$$

This basic model has not evolved much since the 50's

- Although the perceptron initially seemed promising, it was quickly proved that perceptrons could not be trained to recognise many classes of patterns.

- Single layer perceptrons are only capable of learning linearly separable patterns; in 1969 a famous book entitled “Perceptrons” Marvin Minsky and Seymour Papert showed that it was impossible for these classes of network to learn an XOR function.

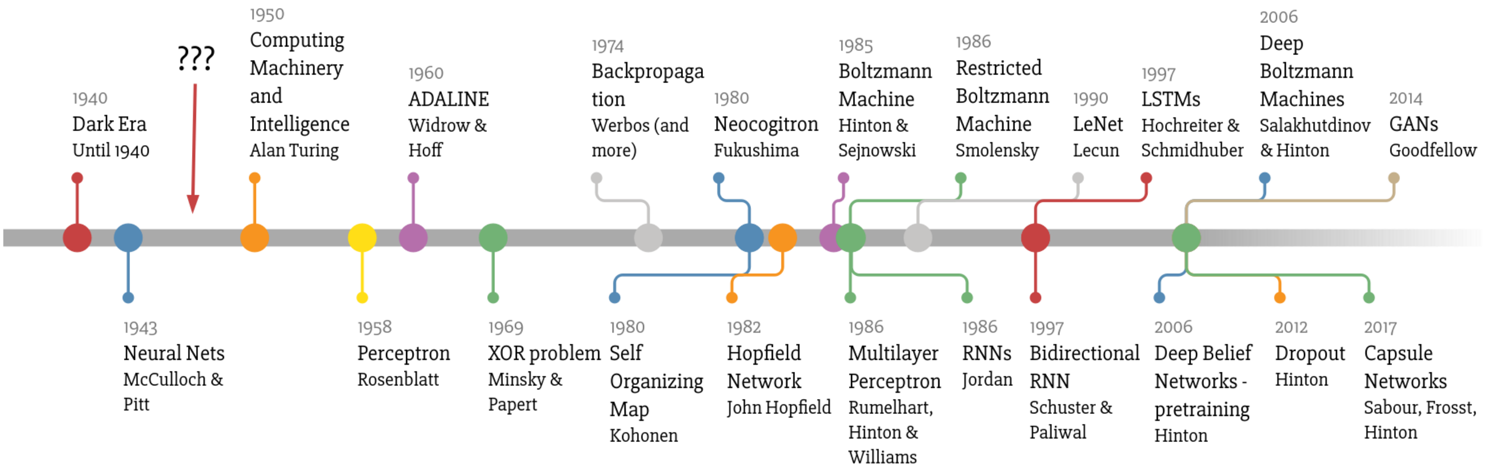
- This caused the field of neural network research to stagnate for many years, before it was recognised that a feedforward neural network with two or more layers (also called a multilayer perceptron) had far greater processing power than perceptrons with a single layer.

This basic model has not evolved much since the 50's

– It took ten more years until neural network research experienced a resurgence in the 1980s.

– This text was reprinted in 1987 as "Perceptrons - Expanded Edition" where some errors in the original text are shown and corrected.

Deep Learning Timeline



1940
Dark Era
Until 1940

???

1943
Neural Nets
McCulloch &
Pitt

1950
Computing
Machinery
and
Intelligence
Alan Turing

1958
Perceptron
Rosenblatt

1960
ADALINE
Widrow &
Hoff

1969
XOR problem
Minsky &
Papert

1974
Backpropagation
Werbos (and
more)

1980
Neocogitron
Fukushima

1980
Self
Organizing
Map
Kohonen

1982
Hopfield
Network
John Hopfield

1985
Boltzmann
Machine
Hinton &
Sejnowski

1986
Restricted
Boltzmann
Machine
Smolensky

1986
Multilayer
Perceptron
Rumelhart,
Hinton &
Williams

1986
RNNs
Jordan

1990
LeNet
Lecun

1997
LSTMs
Hochreiter &
Schmidhuber

1997
Bidirectional
RNN
Schuster &
Paliwal

2006
Deep
Boltzmann
Machines
Salakhutdinov
& Hinton

2006
Deep Belief
Networks -
pretraining
Hinton

2012
Dropout
Hinton

2014
GANs
Goodfellow

2017
Capsule
Networks
Sabour, Frosst,
Hinton

Made by Favio Vázquez

Linear machines and their limitations

First, Some Definitions

Loss Function:

- Loss function is a method of evaluating how well your algorithm models your dataset.

- If your predictions are totally off, your loss function will output a higher number.

- If they’re pretty good, it’ll output a lower number. As you change pieces of your algorithm to try and improve your model, your loss function will tell you if you’re getting anywhere.

First, Some Definitions

Mean Squared Error (MSE) Loss Function:

– Mean Squared Error (MSE) is easy to understand and implement and generally works pretty well.

– To calculate MSE, you take the difference between your predictions and the ground truth, square it, and average it out across the whole dataset.

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y_i})^2$$

First, Some Definitions

Likelihood Loss Function:

The function takes the predicted probability for each input example and multiplies them.

For example, consider a model that outputs probabilities of: [0.4, 0.7, 0.8, 0.1] for the ground truth labels of [0, 1, 1, 0].

Since the model outputs probabilities for TRUE (or 1) only, when the ground truth label is 0 we take (1-p) as the probability.

The likelihood loss would be computed as (1-0.4) * 0.7 * 0.8 * (1-0.1) = 0.6*0.7*0.8*0.9 = 0.3024.

Although the output isn’t exactly human interpretable, it’s useful for comparing models

First, Some Definitions

Log Loss (Log Likelihood Loss) Function:

$$-(y\log(p) + (1-y)\log(1-p))$$

This is the same formula as the regular likelihood function, but with logarithms added in.

When the true class is 1, the second half of the function is 0, and when the true class is 0, the first half is 0.

It basically multiplies the log of the actual predicted probability for the ground truth class.

Linear machines: regression with mean square

Linear regression, mean square loss:

- Decision rule: $y = W'X$
- Loss function: $L(W, y^i, X^i) = \frac{1}{2}(y^i - W'X^i)^2$
- Gradient of loss: $\frac{\partial L(W, y^i, X^i)}{\partial W}' = -(y^i - W(t)'X^i)X^i$
- Update rule: $W(t+1) = W(t) + \eta(t)(y^i - W(t)'X^i)X^i$
- Direct solution: solve linear system $[\sum_{i=1}^{P} X^i X^{i\prime}]W = \sum_{i=1}^{P} y^i X^i$

Linear machines

Perceptron

- Perceptron is a type of linear classifier that makes its predictions based on a linear predictor function combining a set of weights with the feature vector
- It learns a threshold function: a function that maps its input *X* (a real-valued vector) to an output value F(*X*) (a single binary value).

$$F(X) = \begin{cases} 1, if\ W.X + b > 0, \\ 0, \quad\quad otherwise \end{cases}$$

F(*X*) is a thresholding function

W is a vector of real-valued weights

b is the bias

Effectively, F(X) classifies X as a positive or negative instance

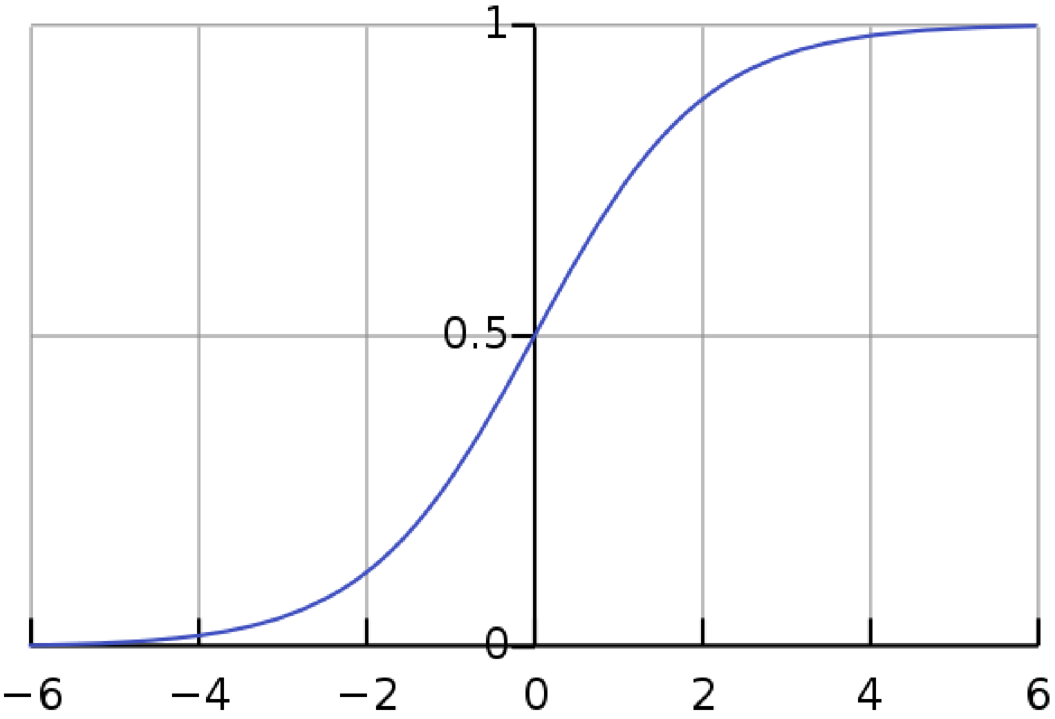
Linear machines

Perceptron

- Decision rule: $y = F(W'X)$ (F is the threshold function)
- Loss function: $L(W, y^i, X^i) = (F(W'X^i) - y^i)W'X^i$
- Gradient of loss: $\frac{\partial L(W, y^i, X^i)}{\partial W}' = -(y^i - F(W(t)'X^i))X^i$
- Update rule: $W(t+1) = W(t) + \eta(t)(y^i - F(W(t)'X^i))X^i$
- Direct solution: find W such that $-y^i F(W'X^i) < 0 \quad \forall i$

Note about the Loss function: The perceptron only updates the weight when the output produced is different from the target labels.
However, if the label and output are different, then the loss function will return $W'X^i$ corresponding to the amount of error.

Logistic Function

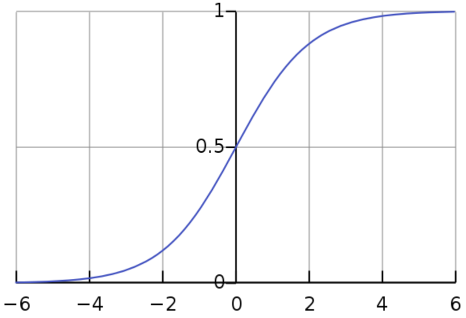


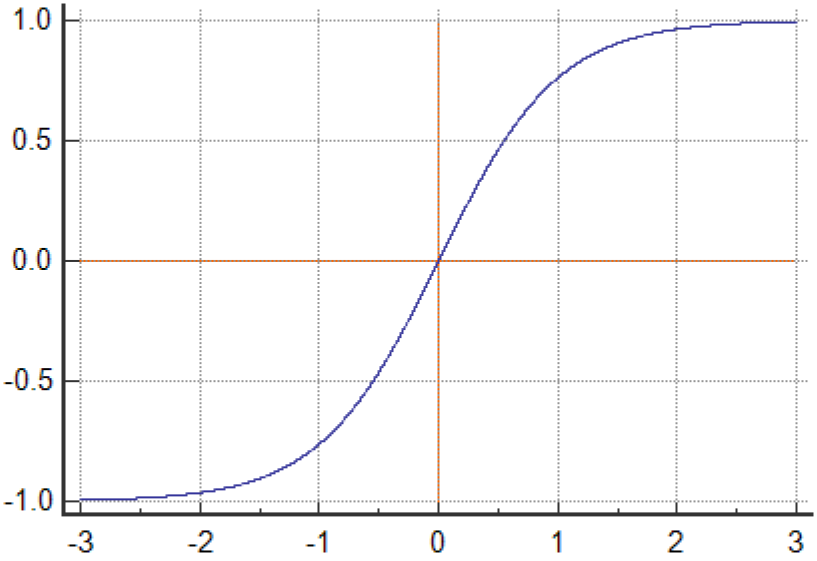
$$f(x) = \frac{L}{1 + e^{-k(x-x_0)}}$$

Hyperbolic Tangent (tanh)

$$f(x) = \frac{1}{1 + e^{-x}} = \frac{e^x}{e^x + 1} = \frac{1}{2} + \frac{1}{2}\tanh(\frac{x}{2})$$

Tanh: Shifted and scaled version of the logistic function





$$f(x) = \tanh x = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

Linear machines: logistic regression

Logistic regression, negative log-likelihood loss function:

– Decision rule: $y = F(W'X)$, with $F(a) = \tanh(a) = \frac{1-\exp(a)}{1+\exp(a)}$ (sigmoid function).

– Loss function: $L(W, y^i, X^i) = 2\log(1 + \exp(-y^i W' X^i))$

– Gradient of loss: $\frac{\partial L(W, y^i, X^i)}{\partial W}' = -\left(Y^i - F(W'X)\right) X^i$

– Update rule: $W(t+1) = W(t) + \eta(t)(y^i - F(W(t)'X^i))X^i$

General gradient-based supervised learning machine

Neural nets, and many other models:

– Decision rule: $y = F(W,X)$, where F is some function, and W some parameter vector.

– Loss function: $L(W, y^i, X^i) = D(y^i, F(W, X^i))$, where $D(y, f)$ measures the “discrepancy” between A and B.

– Gradient loss: $\frac{\partial L(W,y^i,X^i)}{\partial W}' = \frac{\partial D(y^i,f}{\partial f} \frac{\partial F(W,X^i)}{\partial W}$

– Update rule: $W(t+1) = W(t) - \eta(t) \frac{\partial D(y^i,f}{\partial f} \frac{\partial F(W,X^i)}{\partial W}$

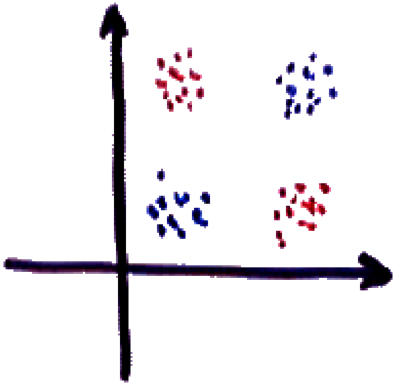
Three questions:

– What architecture $F(W,X)$.

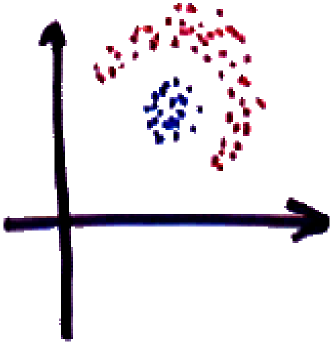
– What loss function $L(W, y^i, X^i)$.

– What optimization method.

Limitations of linear machines



The *linearly separable* dichotomies are the partitions that are realizable by a linear classifier

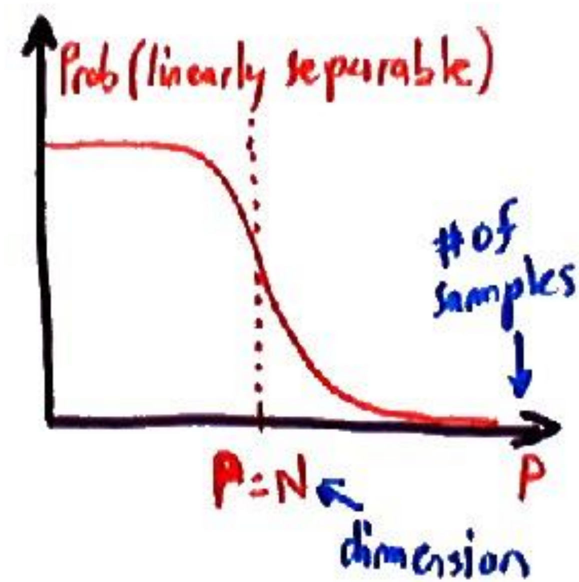


The boundary between the classes is a hyperplane.

Number of linearly separable dichotomies

The probability that P samples of dimension N are linearly separable goes to zero very quickly as P grows larger than N

(Cover's theorem, 1966).

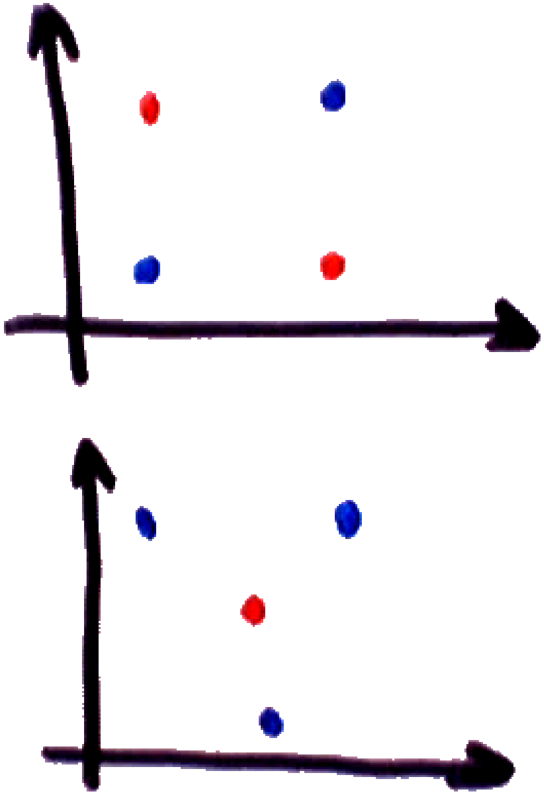


– Problem: there are 2^P possible dichotomies of P points

– Only about N are linearly separable

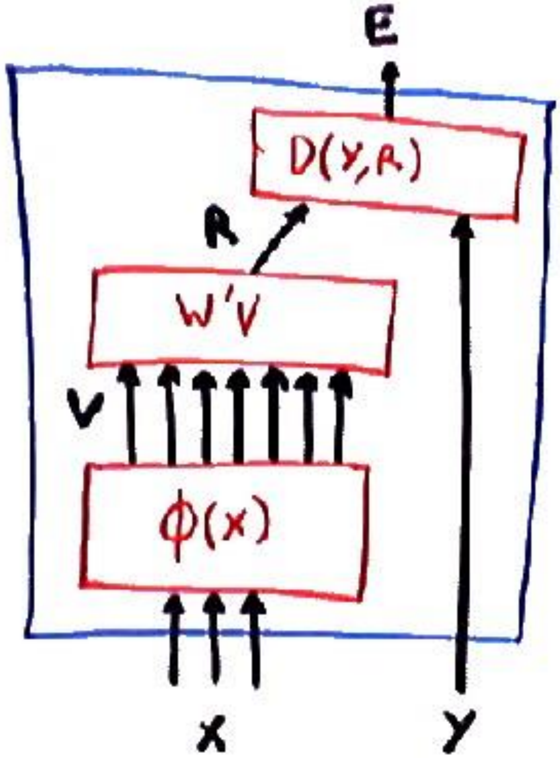
– If P is larger than N, the probability that a random dichotomy linearly separable is very, very small

Example of non-linearly separable dichotomies



- Some seemingly simple dichotomies are not linearly separable
- Question: how do we make a given problem linearly separable?

Preprocessing: Making N (dimension) larger:



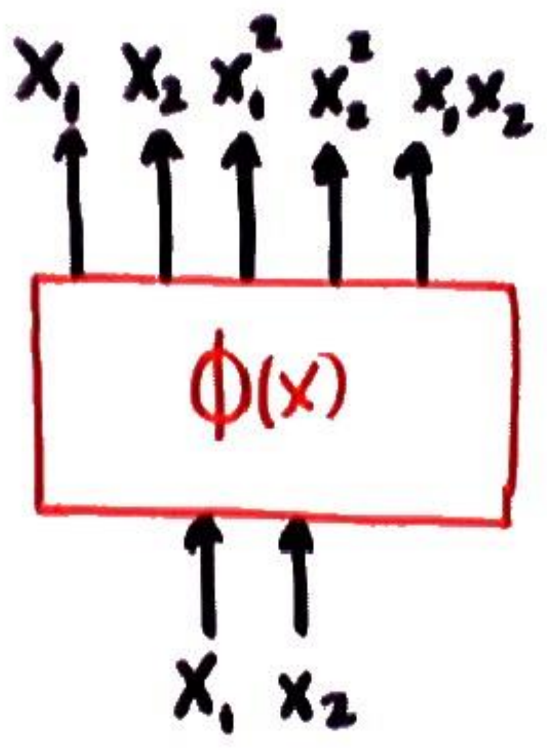
We can make N larger by augmenting the input variables with new “features”.

We map/project X from its original N-dimensional space into a higher dimensional space using a vector function $\Phi(X)$.

In this higher dimensional space things are more likely to be linearly separable,

$$E(Y,X,W) = D(Y,R)$$
$$R = f(W'V)$$
$$V = \Phi(X)$$

Adding cross-product terms



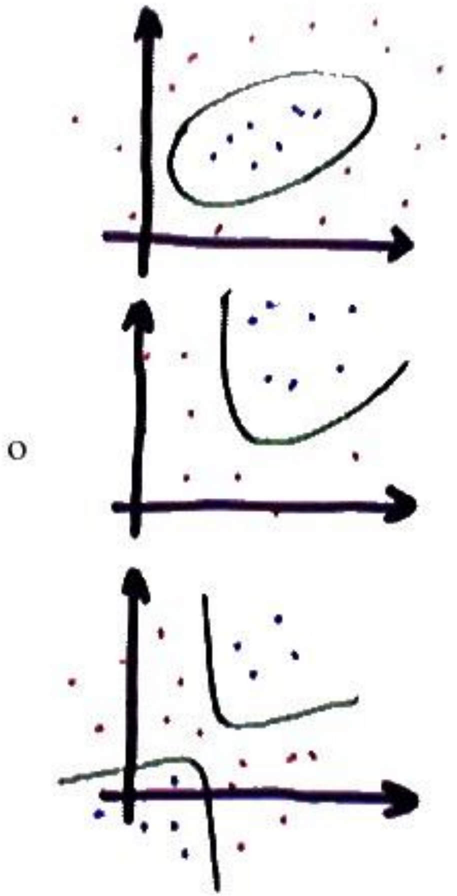
- Polynomial expansion
- If our original input variables are *(1, x1, x2)*, we construct a new feature vector with the following components:

$$\Phi(1, x_1, x_2) = (1, x_1, x_2, x_1^2, x_2^2, x_1 x_2)$$

- i.e. we add all the cross-products of the original variables

- We map/project *X* from its original *N*-dimensional space into a higher dimensional space with *N (N+1)/2* dimensions

Polynomial mapping



- Many new functions are now separable with the new architecture

- With cross-product features, the family of class boundaries in the original space is the conic sections (ellipse, parabola, hyperbola)

- To each possible boundary in the original space corresponds a linear boundary in the transformed space

- We can use standard linear learning algorithms (perception, linear regression, logistic regression…)

Problems with polynomial mapping

- We can generalize this idea to higher degree polynomials, adding cross-product terms with 3, 4 or more variables
- Unfortunately, the number of terms is the number of combinations d choose N, which grows like N^d, where d is the degree, and N the number of original variables
- In particular, the number of free parameters that must be learned is also of order N^d
- This is impractical for large N and for $d > 2$

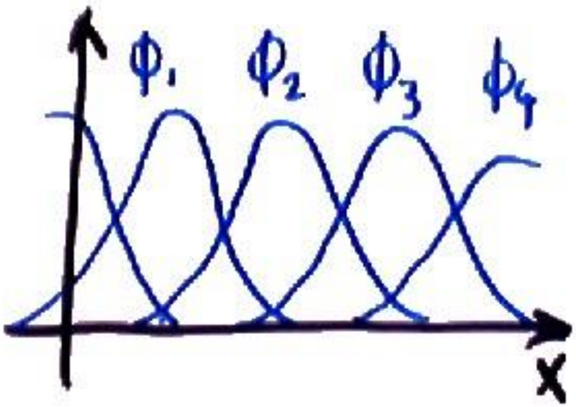
Problems with polynomial mapping

Example: handwritten digit recognition (16x16 pixel images).
How many variables do we have?

Number of variables: 256.
Degree 2: 32,896 variables
Degree 3: 2,796,160
Degree 4: 247,460,160…

Next idea: tile the space

– Place a number of equally-spaced “bumps” that cover the entire input space



– For classification, the bumps can be Gaussians

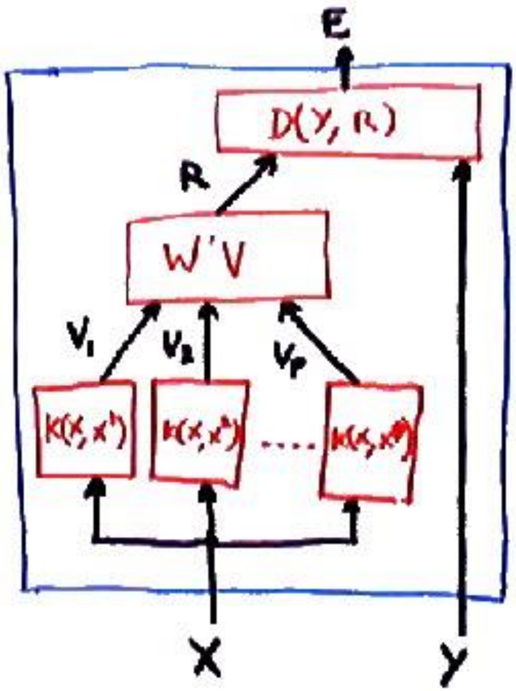
– For regression, the basis functions can be wavelets, sine/cosine, splines (pieces of polynomials)…

– Problem: this does not work with more than a few dimensions

– The number of bumps necessary to cover an N dimensional space grows exponentially with N.

Sample-centered basis functions (kernels)

- Place the center of a basis function around each training sample. That way, we only spend resources on regions of the space where we actually have training samples.



- Discriminant function:

$$f(X,W) = \sum_{k=1}^{k=P} W_k K(X, X^k)$$

- *K(X,X')* often takes the form of a *radial basis function*

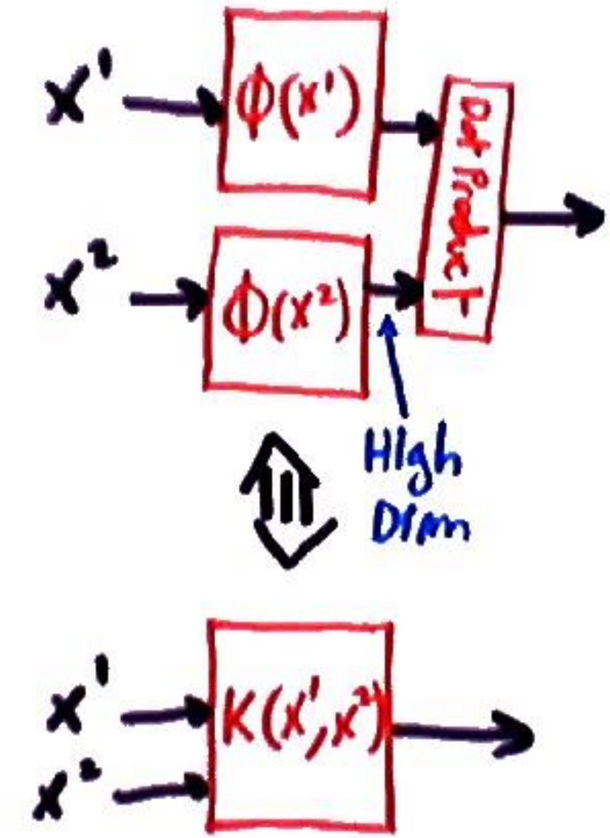
$$K(X,X') = \exp(b||X - X'||^2)$$

- *or a polynomial*:

$$K(X,X') = (X.X' + 1)^m$$

- This is a very common architecture, which can be used with a number of energy functions.
- In particular, this is the architecture of the Support Vector Machine (SVM)

The kernel trick



- If the kernel function *K(X,X')* verifies the Mercer conditions, then there exist a mapping Φ, such that

$$\Phi(X).\Phi(X') = K(X, X').$$

- The Mercer conditions are that *K* must be symmetric, and must be positive definite (i.e. *K(X,X)* must be positive for all *X*)
- In other words, if we want to map our *X* into a high-dimensional space (so as to make them linearly separable), and all we have to do in that space is compute dot products, we can take a shortcut and simply compute $K(X^1,X^2)$ without going through the high-dimensional space
- This is called the "kernel trick". It is used in many so-called Kernel-based methods, including Support Vector Machines.

Examples of Kernels

- Quadratic kernel: $\Phi(X) = (1, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_1 x2, x_1^2, x_2^2)$ then

$$K(X, X') = \Phi(X).\Phi(X') = (X.X' + 1)^2$$

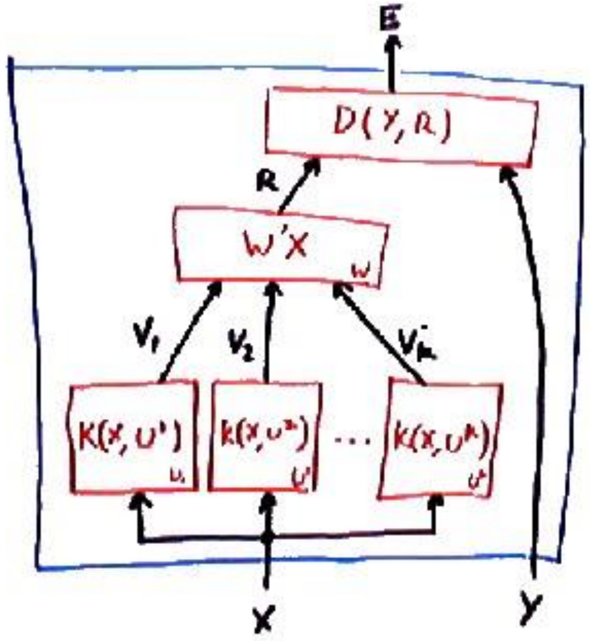
- Polynomial kernel: this generalizes to any degree *d*. the kernel that corresponds to $\Phi(X)$ being a polynomial of degree *d* is

$$K(X, X') = \Phi(X).\Phi(X') = (X.X' + 1)^d.$$

- Gaussian Kernel: $K(X, X') = \exp(-b||X - X'||^2)$

- This kernel, sometimes called the Gaussian Radial Basis Function (RBF), is very commonly used

Sparse basis functions



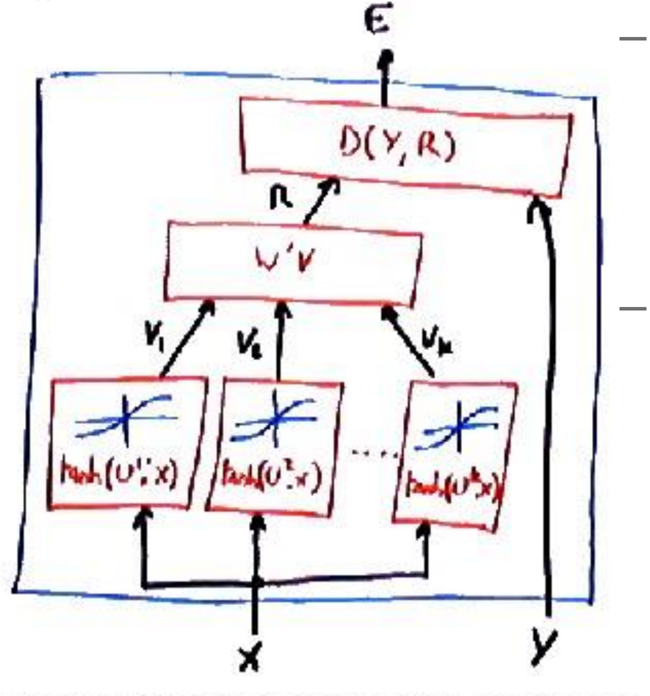
The discriminant function F is:

- Place the center of a basis function around areas containing training samples
- Idea 1: use an unsupervised clustering algorithm (such as K-means or mixture of Gaussians) to place the centers of the basis functions in areas of high sample density
- Idea 2: adjust the basis functions centers through gradient descent in the loss function

$$F(X, W, U^1, \ldots, U^K) = \sum_{k=1}^{k=K} W_k K(X, U^k)$$

Neural net with a single hidden layer

– A particularly interesting type of basis function is the sigmoid unit: $V_k = tanh(U'^k X)$



– A network using these basis functions, whose output is $R = \sum_{k=1}^{k=K} W_k V_k$

is called a *single hidden-layer neural network*

– Similarly to the RBF network, we can compute the gradient of the loss function with respect to the U^k.

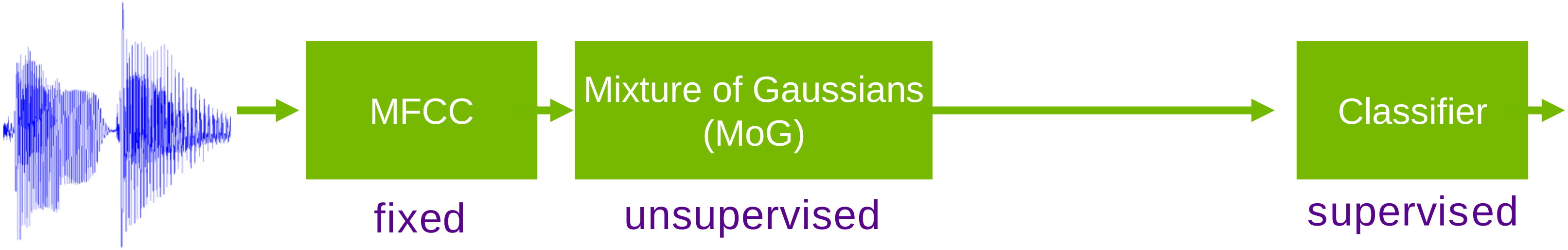
$$\frac{\partial L(W)}{\partial U^j} = \frac{\partial L(W)}{\partial R} W_j \frac{\partial tanh(U_j' X)}{\partial U_j}$$

$$= \frac{\partial L(W)}{\partial R} W_j tanh'(U_j' X) X'$$

– Any well-behaved function can be approximated as close as we wish by such networks (but K might be very large).

Architecture of “mainstream” pattern recognition systems

- Modern architecture for pattern recognition
  - Speech recognition: early 90's – 2011



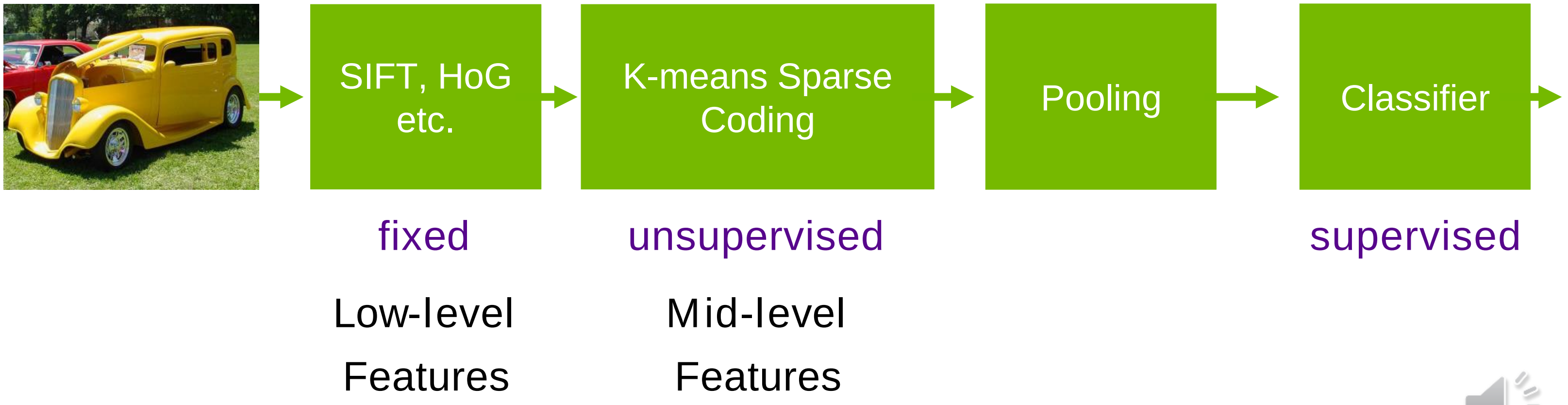
MFCC → Mixture of Gaussians (MoG) → Classifier

fixed

unsupervised

supervised

  - Object Recognition: 2006 - 2012



SIFT, HoG etc. → K-means Sparse Coding → Pooling → Classifier

fixed

unsupervised

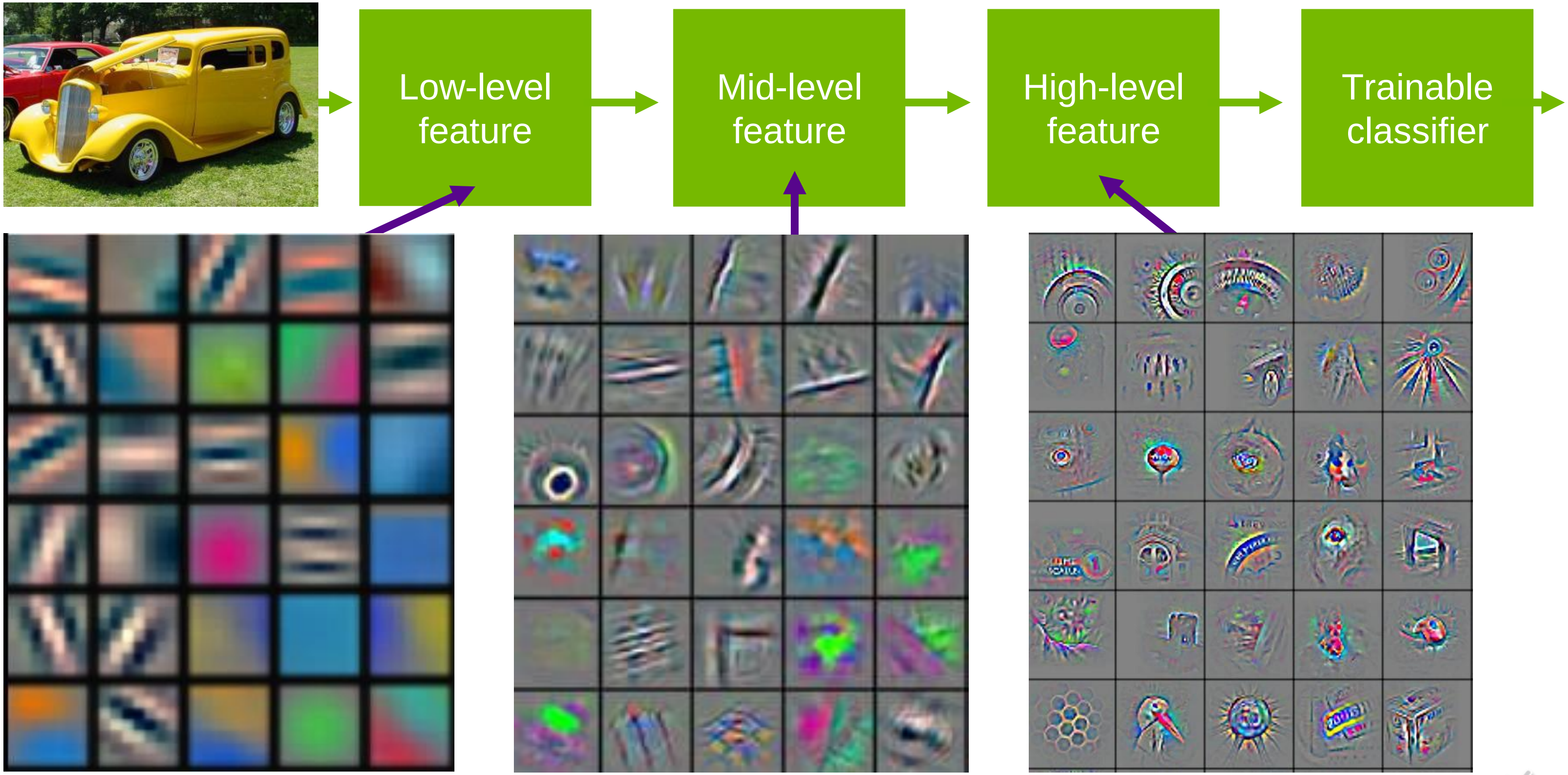
supervised

Low-level Features

Mid-level Features

Deep learning = learning hierarchical representations

It's deep if it has more than one stage of non-linear feature transformation


Low-level feature

Mid-level feature

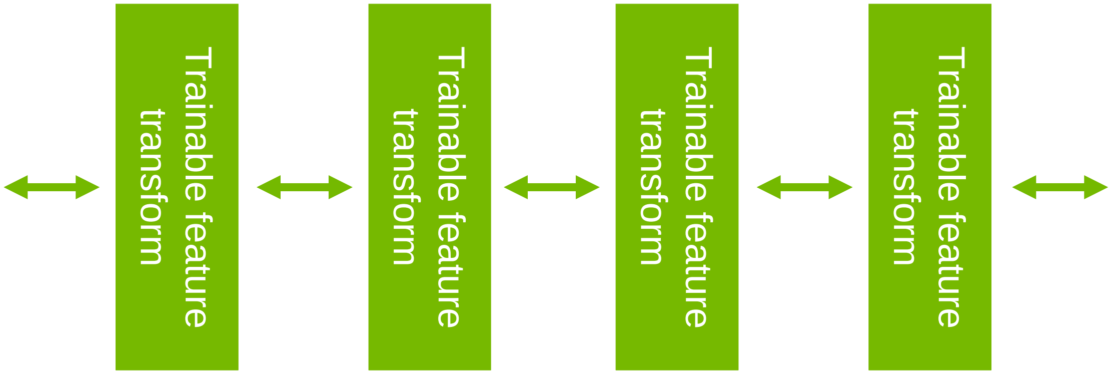
High-level feature

Trainable classifier

Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

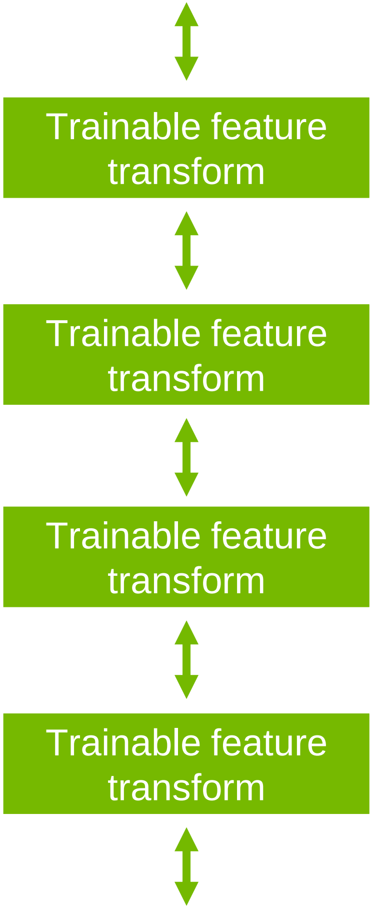
Trainable feature hierarchy

- Hierarchy of representations with increasing level of abstraction Each stage is a kind of trainable feature transform
- Image recognition
  - Pixel → edge → texton → motif → part → object
- Text
  - Character → word → word group → clause → sentence → story
- Speech
  - Sample → spectral band → sound → … → phone → phoneme → word


Trainable feature transform
Trainable feature transform
Trainable feature transform
Trainable feature transform

Learning representations: a challenge for ML, CV, AI, neuroscience, cognitive science...

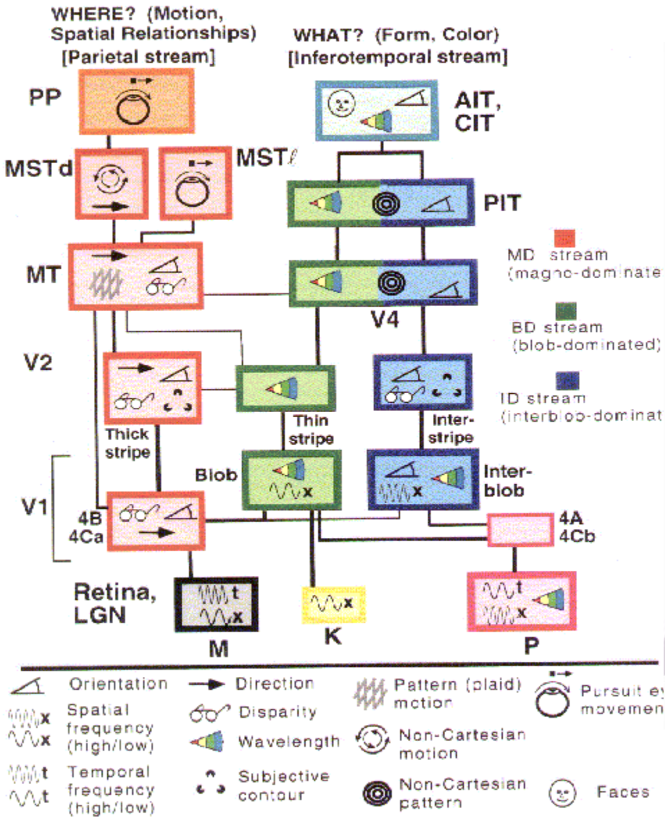
- CV: How do we learn representations of the perceptual world?
  - How can a perceptual system build itself by looking at the world?
  - How much prior structure is necessary
- ML/AI: How do we learn features or feature hierarchies?
  - What is the fundamental principle? What is the learning algorithm? What is the architecture?
- Neuroscience: How does the cortex learn perception?
  - Does the cortex "run" a single, general learning algorithm? (or a small number of them)
- CogSci: How does the mind learn abstract concepts on top of less abstract ones?
- Deep Learning addresses the problem of learning hierarchical representations with a single algorithm
  - Or perhaps with a few algorithms

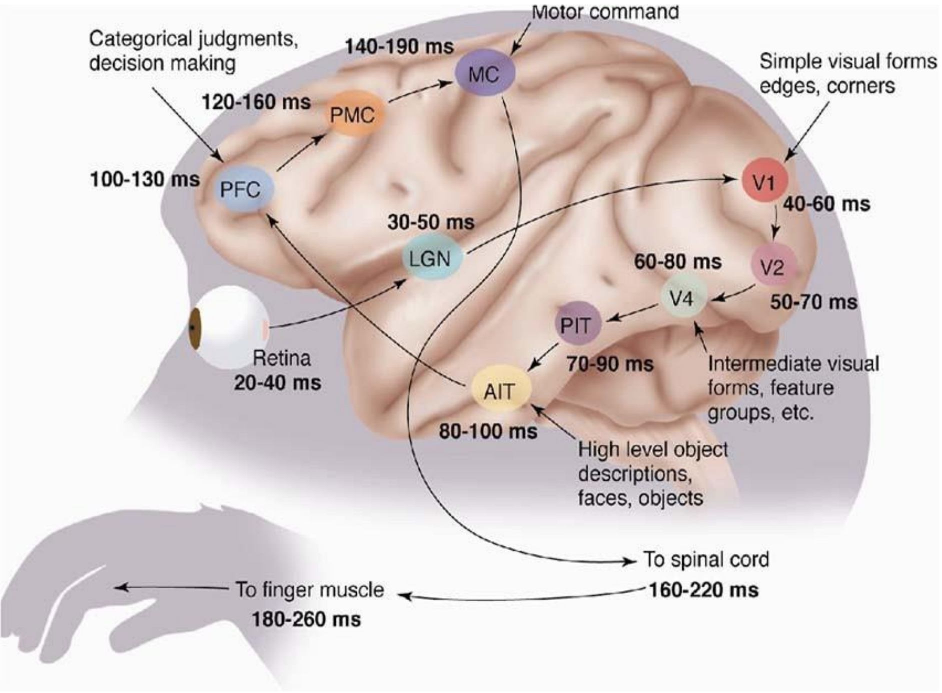

Trainable feature transform
Trainable feature transform
Trainable feature transform
Trainable feature transform

The mammalian visual cortex is hierarchical

– The ventral (recognition) pathway in the visual cortex has multiple stages Retina - LGN - V1 - V2 - V4 - PIT - AIT ....

– Lots of intermediate representations





[picture from Simon Thorpe]

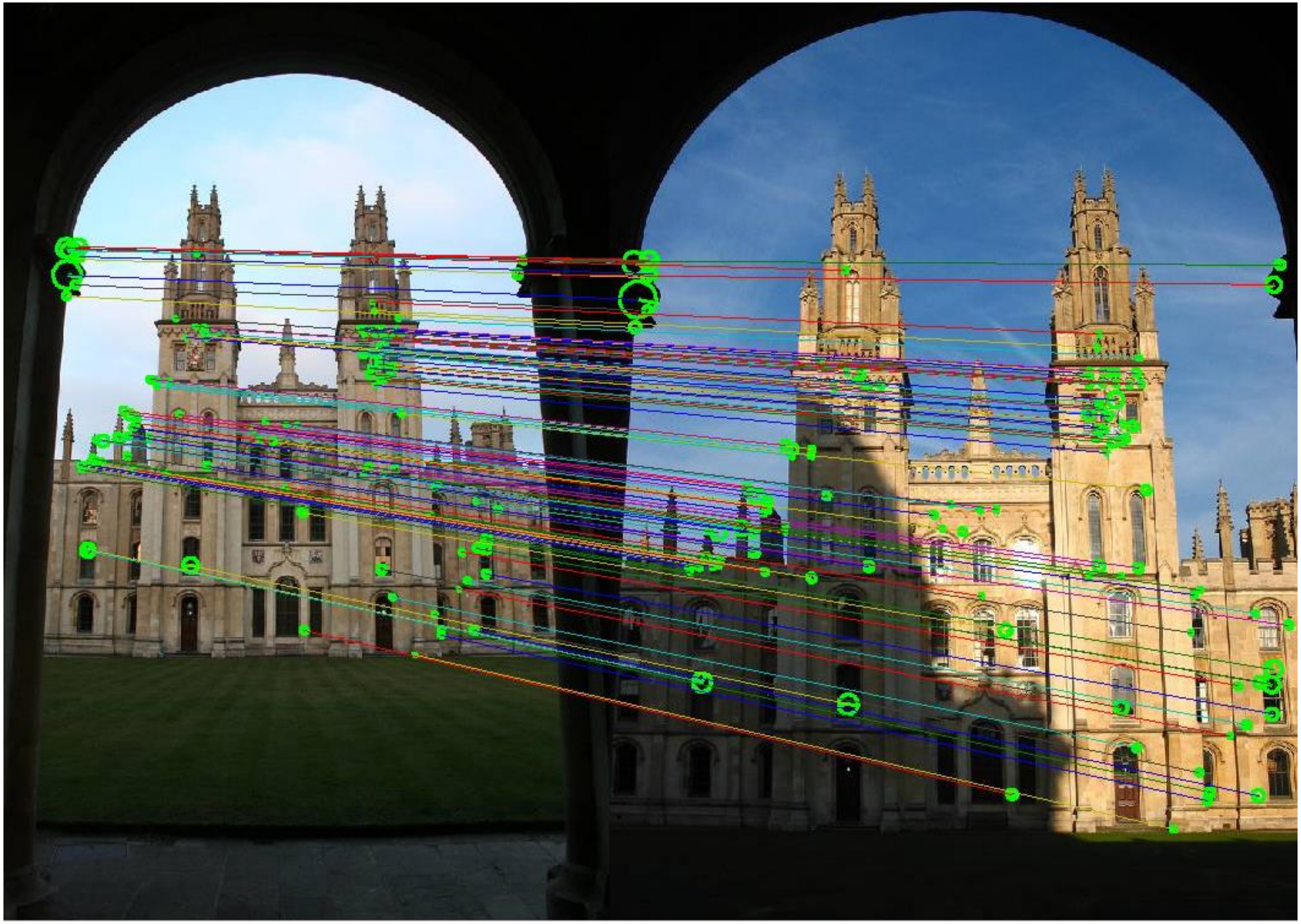
[Gallant & Van Essen]

Let's be inspired by nature, but not too much

- It's nice to imitate Nature, but we also need to understand
  - How do we know which details are important?
  - Which details are merely the result of evolution, and the constraints of biochemistry?
- For airplanes, we developed aerodynamics and compressible fluid dynamics.
  - We figured that feathers and wing flapping weren't crucial

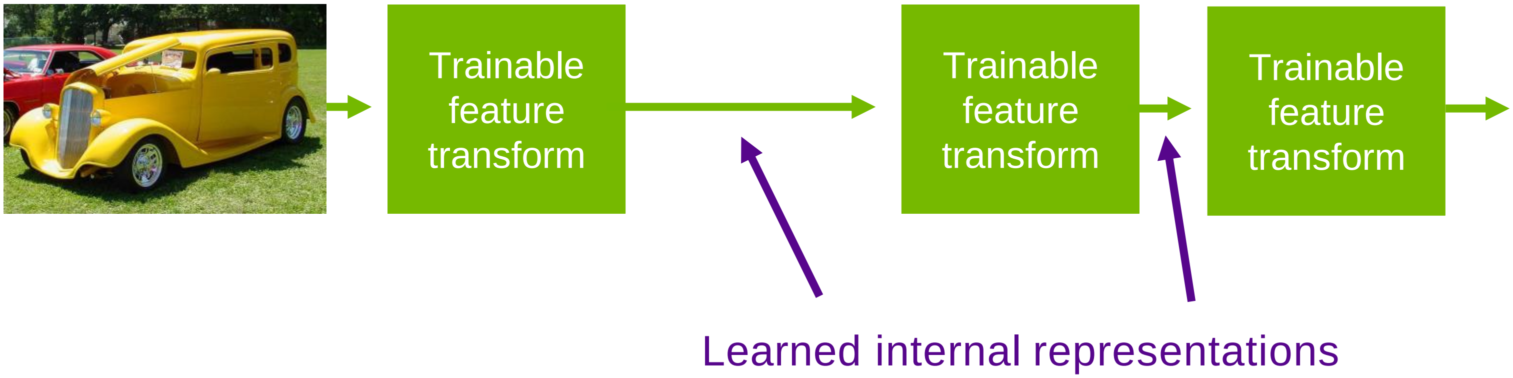
What is the equivalent of aerodynamics for understanding intelligence?

Invariant Features



Trainable feature hierarchies: end-to-end learning

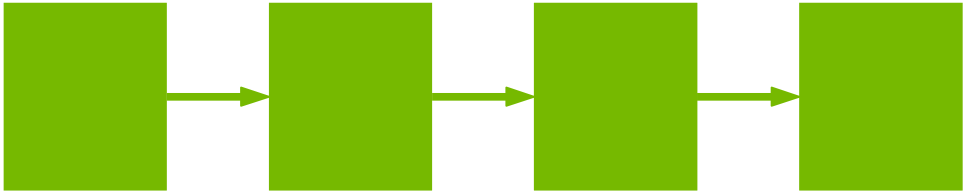
- A hierarchy of trainable feature transforms
  - Each module transforms its input representation into a higher-level one.
  - High-level features are more global and more invariant
  - Low-level features are shared among categories


Trainable feature transform
Trainable feature transform
Trainable feature transform
Learned internal representations

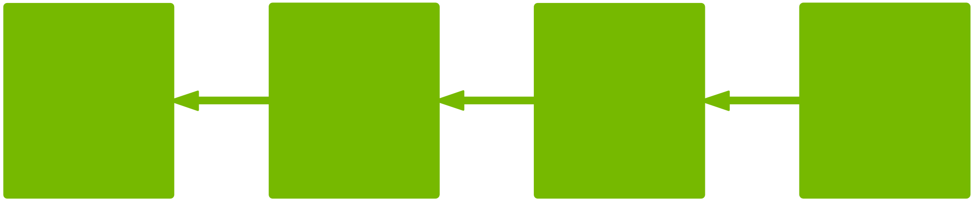
- How can we make all the modules trainable and get them to learn appropriate representations?

Three types of deep architectures

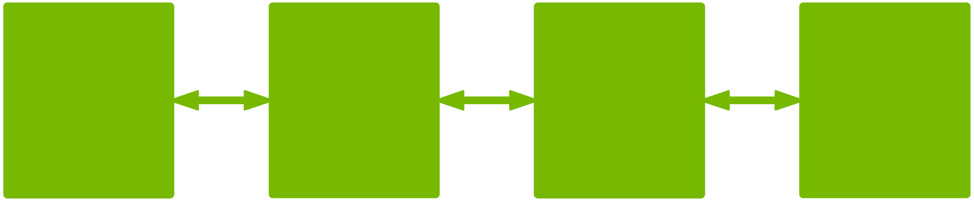
– Feed-forward: multilayer neural nets, convolutional nets



– Feed-back: stacked sparse coding, deconvolutional nets



– Bi-directional: Deep Boltzmann Machines, stacked auto-encoders



Do we really need deep architectures?

- Theoretician's dilemma: "We can approximate any function as close as we want with shallow architecture. Why would we need deep ones?"

$$y = \sum_{i=1}^{P} \alpha_i K(X, X^i) \qquad y = F(W^1.F(W^0.X))$$

  - kernel machines (and 2-layer neural nets) are "universal".

- Deep learning machines

$$y = F(W^K.F(W^{K-1}.F(....F(W^0.X)...)))$$

- Deep machines are more efficient for representing certain classes of functions, particularly those involved in visual recognition
  - They can represent more complex functions with less "hardware"
- We need an efficient parameterization of the class of functions that are useful for "AI" tasks (vision, audition, NLP...)

Why would deep architectures be more efficient?

[Bengio & LeCun 2007 “Scaling Learning Algorithms Towards AI”]

- A deep architecture trades space for time (or breadth for depth)
  - More layers (more sequential computation),
  - But less hardware (less parallel computation).

- Example1: N-bit parity
  - Requires N-1 XOR gates in a tree of depth log(N).
  - Even easier if we use threshold gates
  - If we restrict ourselves to 2 layers: Requires an exponential number of gates

    (DNF formula with exponential number of minterms).

Which models are deep?

- 2-layer models are not deep (even if you train the first layer)
    - Because there is no feature hierarchy

- Neural nets with 1 hidden layer are not deep

Which models are deep?

SVMs and Kernel methods are not deep

Layer1: kernels

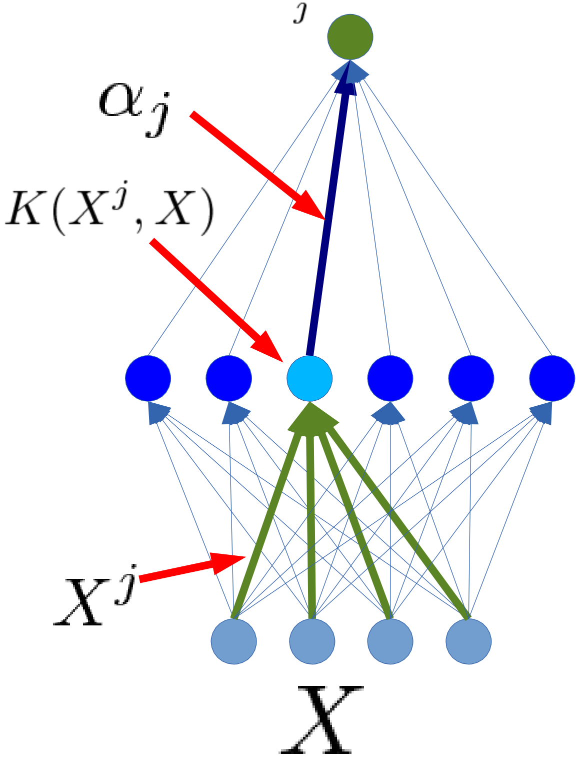
Layer2: linear

The first layer is “trained” in with the simplest unsupervised method ever devised: using the samples as templates for the kernel functions.

Decision (Classification) trees are not deep:

No hierarchy of features. All decisions are made in the input space

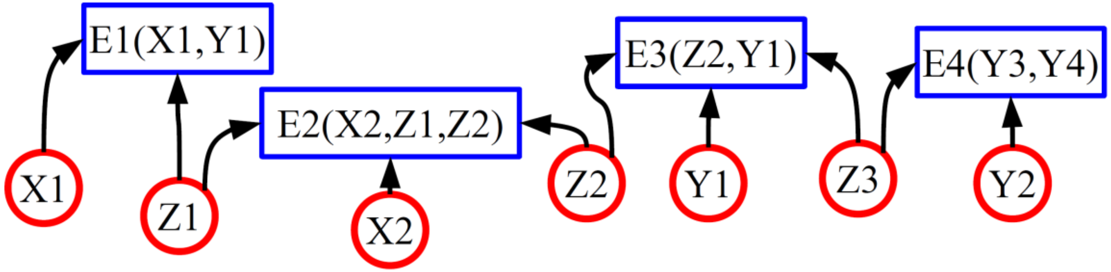
$$G(X,\alpha) = \sum_j \alpha_j K(X^j, X)$$



Are graphical models deep?

- There is no opposition between graphical models and deep learning.
  - Many deep learning models are formulated as factor graphs
  - Some graphical models use deep architectures inside their factors
- Graphical models can be deep (but most are not). Factor graph: sum of energy functions
  - Over inputs X, outputs Y and latent variables Z. Trainable parameters: W

$$-\log P(X,Y,Z/W) \propto E(X,Y,Z,W) = \sum_i E_i(X,Y,Z,W_i)$$



- Each energy function can contain a deep network
- The whole factor graph can be seen as a deep network

Deep learning: A theoretician's nightmare?

- Deep Learning involves non-convex loss functions
  - With non-convex losses, all bets are off
  - Then again, every speech recognition system ever deployed has used non-convex optimization (GMMs are non convex).

- But to some of us all “interesting” learning is non convex
  - Convex learning is invariant to the order in which sample are presented (only depends on asymptotic sample frequencies).
  - Human learning isn't like that: we learn simple concepts before complex ones. The order in which we learn things matter.

Deep learning: A theoretician's paradise?

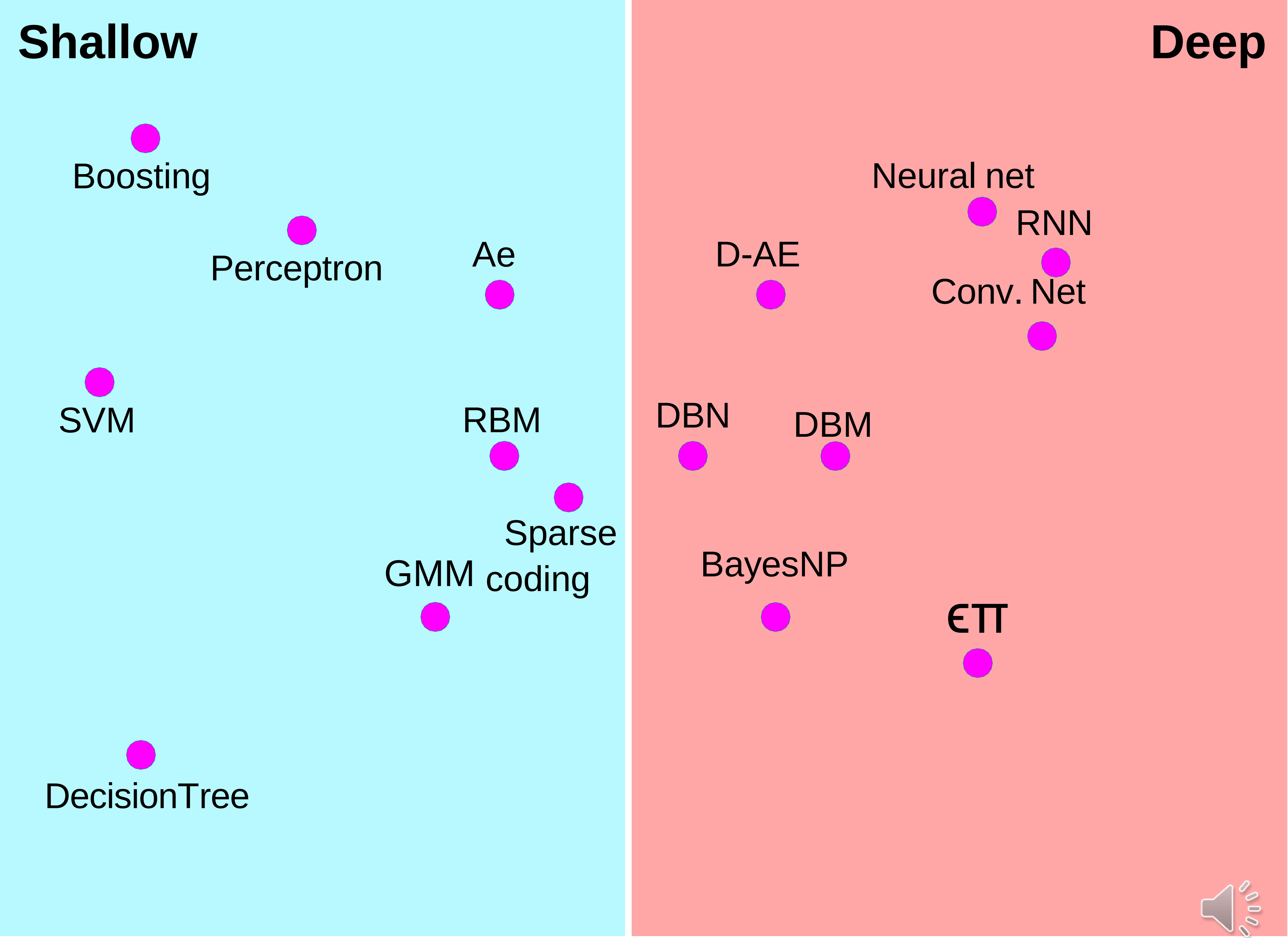
- Deep learning is about representing high-dimensional data
  - There has to be interesting theoretical questions there what is the geometry of natural signals?
  - Is there an equivalent of statistical learning theory for unsupervised learning?
  - What are good criteria on which to base unsupervised learning?

- Deep learning systems are a form of latent variable factor graph
  - Internal representations can be viewed as latent variables to be inferred, and deep belief networks are a particular type of latent variable models.
  - The most interesting deep belief nets have intractable loss functions: how do we get around that problem?

- Lots of theory at the 2012 IPAM summer school on deep learning
  - Wright's parallel SGD methods
  - Mallat's “scattering transform”
  - Osher's “split Bregman” methods for sparse modeling
  - Morton's “algebraic geometry of DBN”,....

Deep learning and feature learning today

- Deep learning has been the hottest topic in speech recognition in the last 2 years
  - A few long-standing performance records were broken with deep learning methods
  - Microsoft and google have both deployed dl-based speech recognition system in their products
  - Microsoft, google, IBM, nuance, AT&T, and all the major academic and industrial players in speech recognition have projects on deep learning
- Deep learning is the hottest topic in computer vision
  - Feature engineering is the bread-and-butter of a large portion of the CV community, which creates some resistance to feature learning
  - But the record holders on ImageNet and semantic segmentation are convolutional nets
- Deep learning is becoming hot in natural language processing
- Deep learning/feature learning in applied mathematics
  - The connection with applied math is through sparse coding, non-convex optimization, stochastic gradient algorithms, etc...



Shallow

Deep

Boosting

Perceptron

Ae

SVM

RBM

Sparse coding

GMM

DecisionTree

Neural net

RNN

D-AE

Conv. Net

DBN

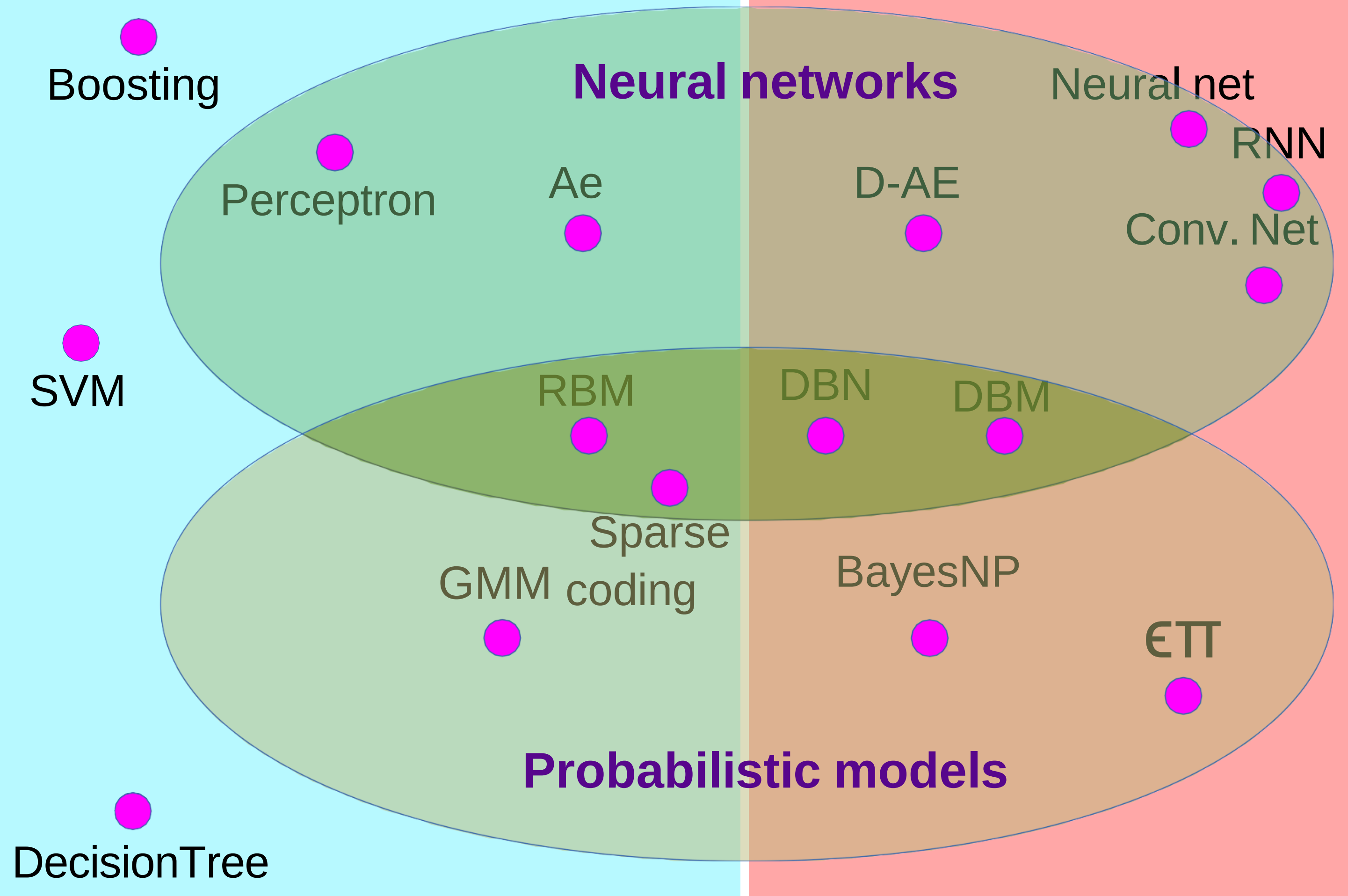
DBM

BayesNP

ϵπ

Shallow

Deep



Neural networks

Probabilistic models

Boosting

Perceptron

Ae

D-AE

Neural net

RNN

Conv. Net

SVM

RBM

DBN

DBM

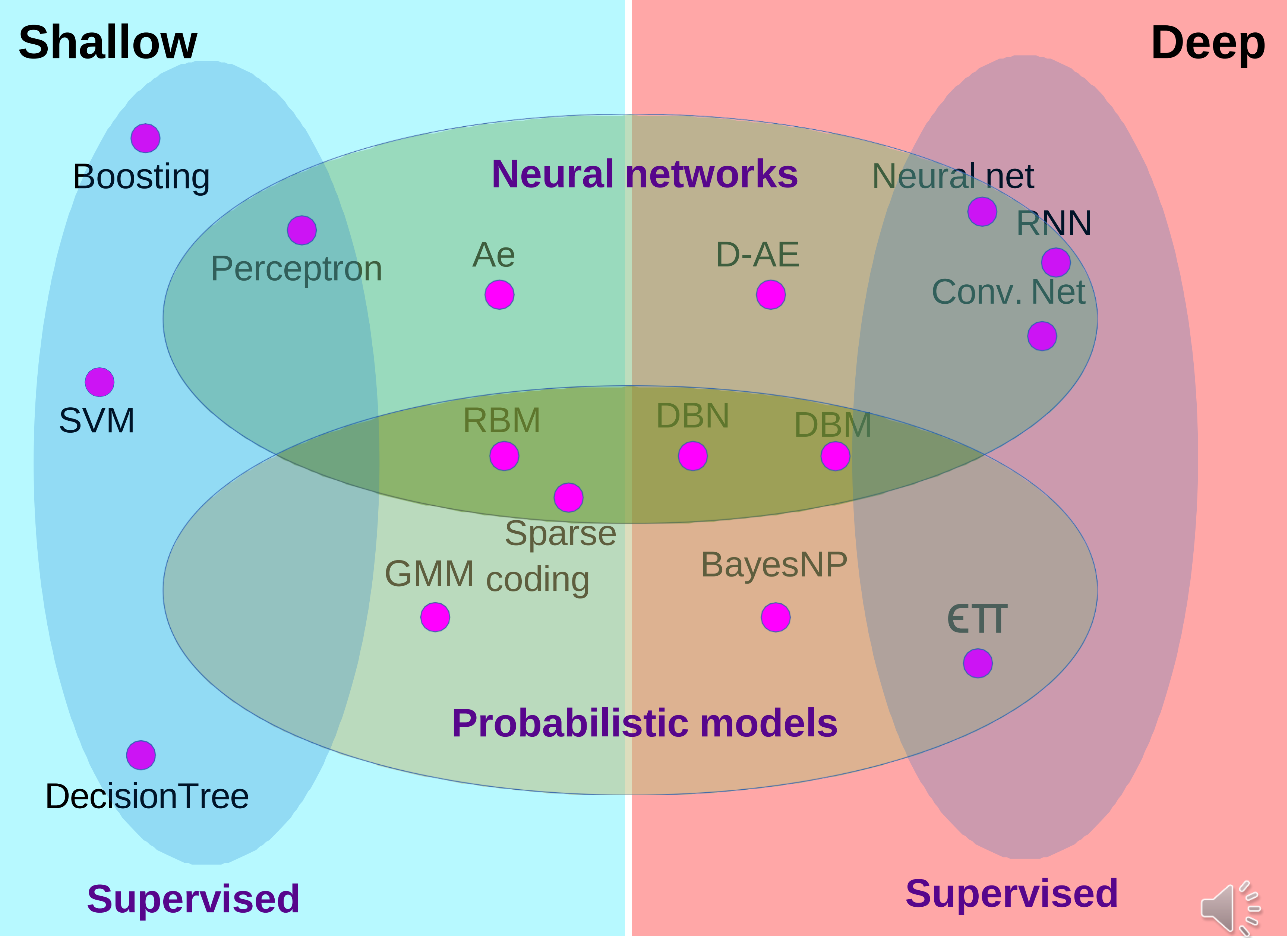
Sparse coding

GMM

BayesNP

ϵΠ

DecisionTree



Shallow

Deep

Neural networks

Probabilistic models

Boosting

Perceptron

SVM

DecisionTree

Ae

D-AE

Neural net

RNN

Conv. Net

RBM

DBN

DBM

Sparse coding

GMM

BayesNP

ЄΠ

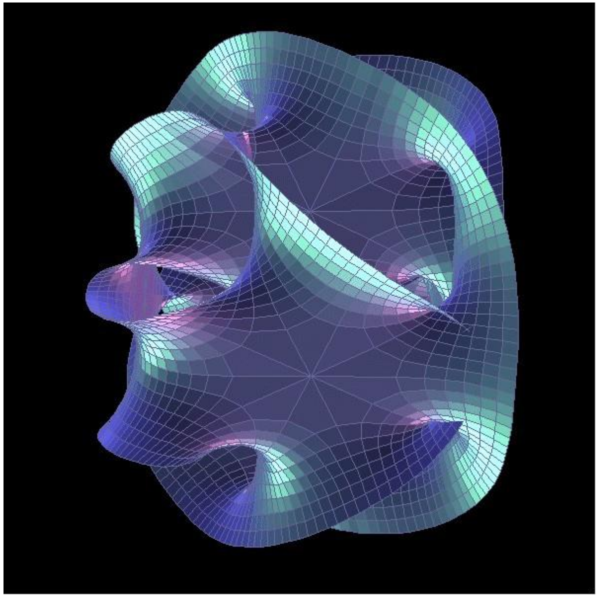
Supervised

Supervised

What are good features?

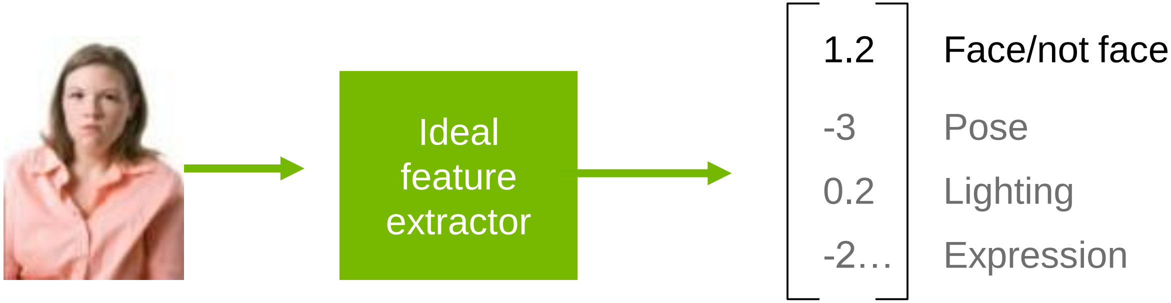
Discovering the hidden structure in high-dimensional data the manifold hypothesis

- Learning representations of data:
  - Discovering & disentangling the independent explanatory factors
- The manifold hypothesis:
  - Natural data lives in a low-dimensional (non-linear) manifold
  - Because variables in natural data are mutually dependent



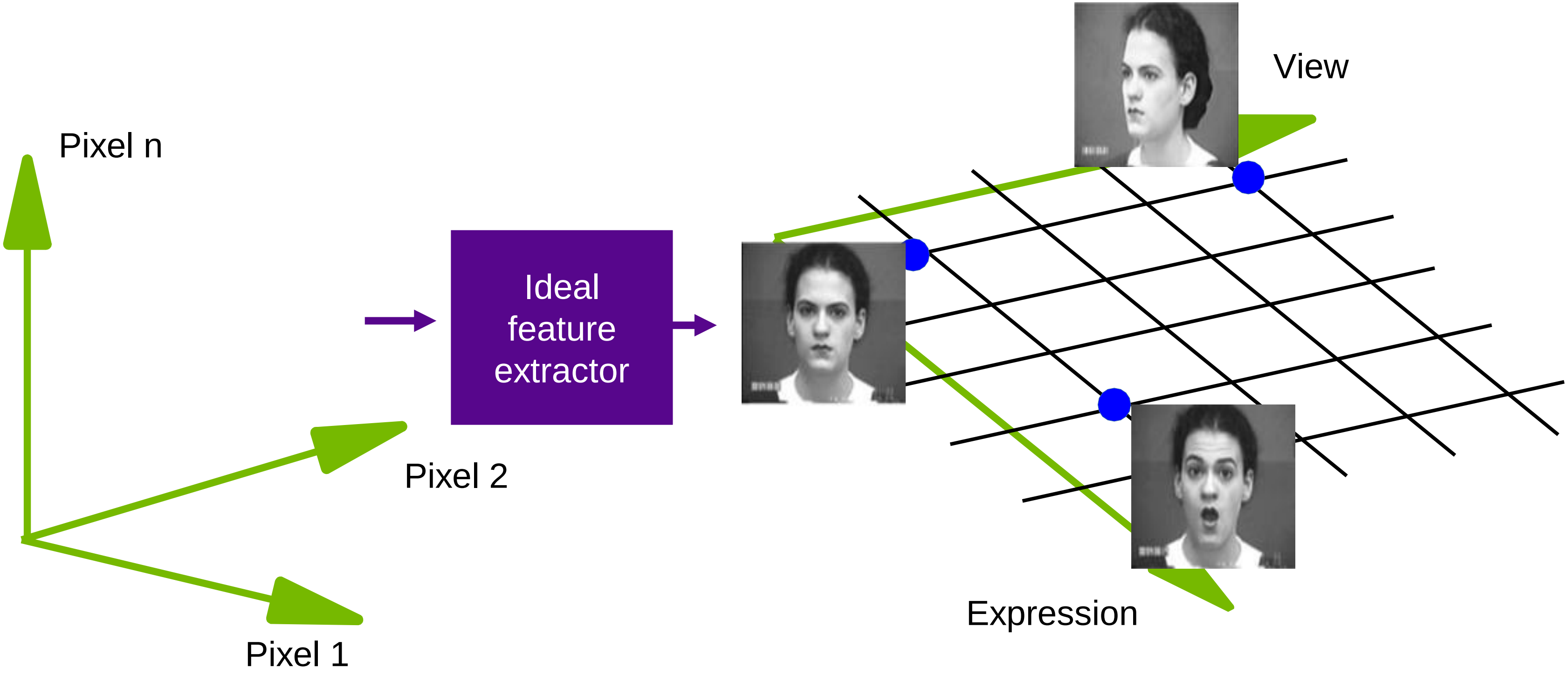
Discovering the hidden structure in high-dimensional data

- Example: all face images of a person
  - 1000x1000 pixels = 1,000,000 dimensions
  - But the face has 3 Cartesian coordinates and 3 Euler angles and humans have less than about 50 muscles in the face
  - Hence the manifold of face images for a person has <56 dimensions
- The perfect representations of a face image:
  - Its coordinates on the face manifold
  - Its coordinates away from the manifold
- We do not have good and general methods to learn functions that turns an image into this kind of representation



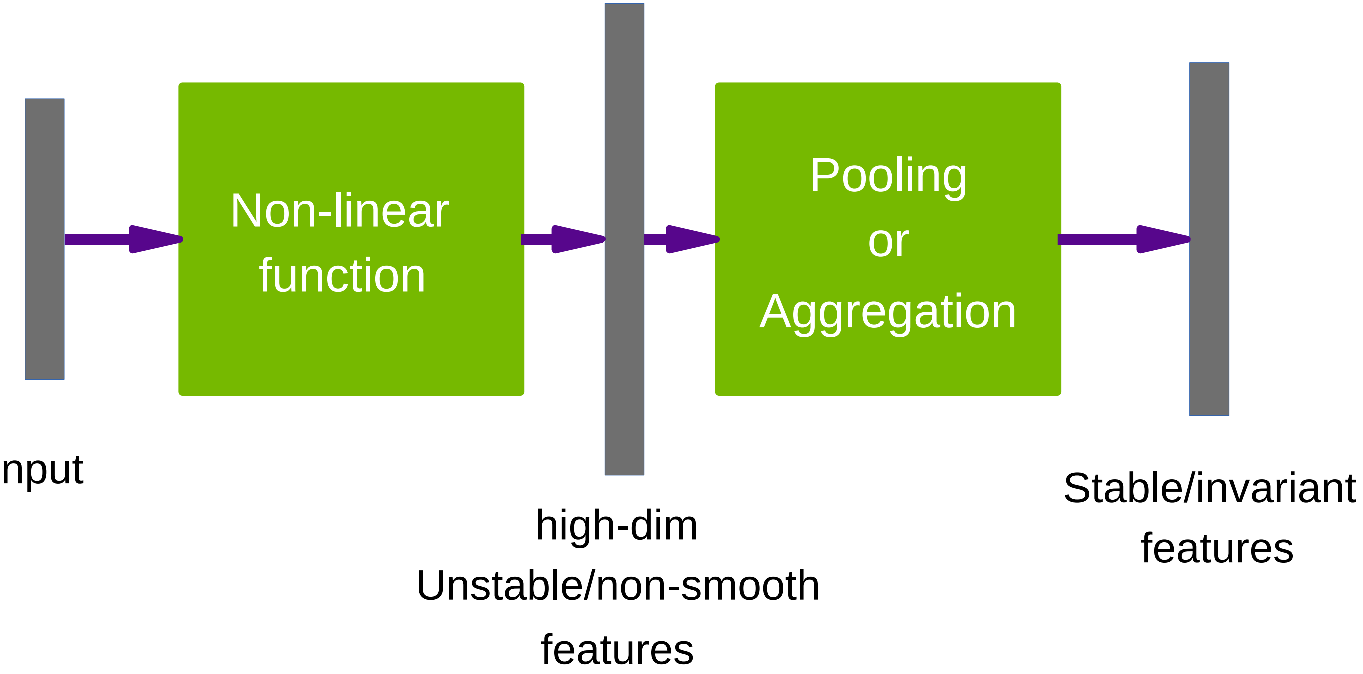
Disentangling factors of variation

The ideal disentangling feature extractor



Basic idea for invariant feature learning

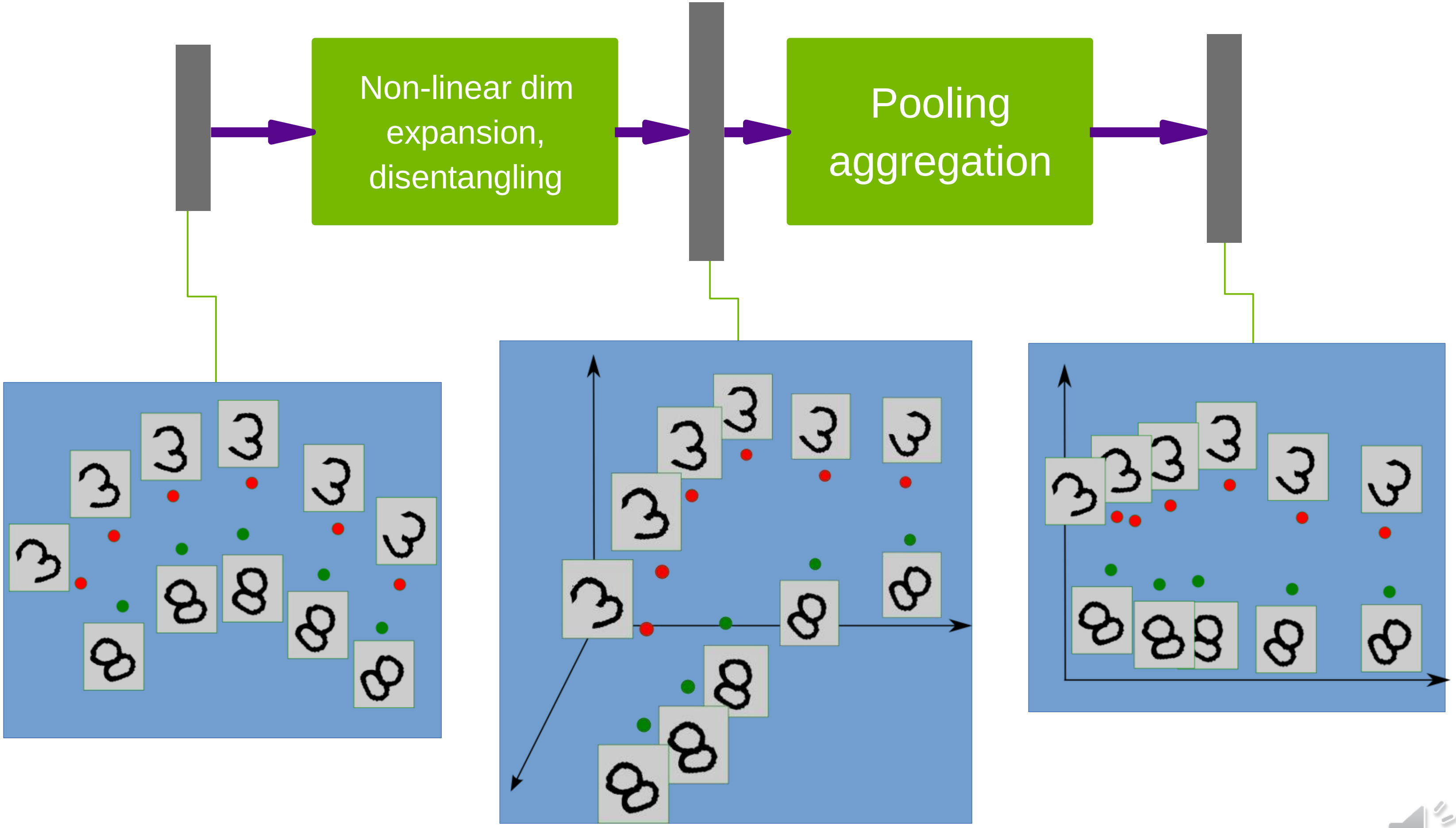
- Embed the input non-linearly into a high(er) dimensional space
  - In the new space, things that were non separable may become separable
- Pool regions of the new space together
  - Bringing together things that are semantically similar.



Non-linear function
Pooling or Aggregation
Input
high-dim Unstable/non-smooth features
Stable/invariant features

Non-linear expansion → pooling

Entangled data manifolds

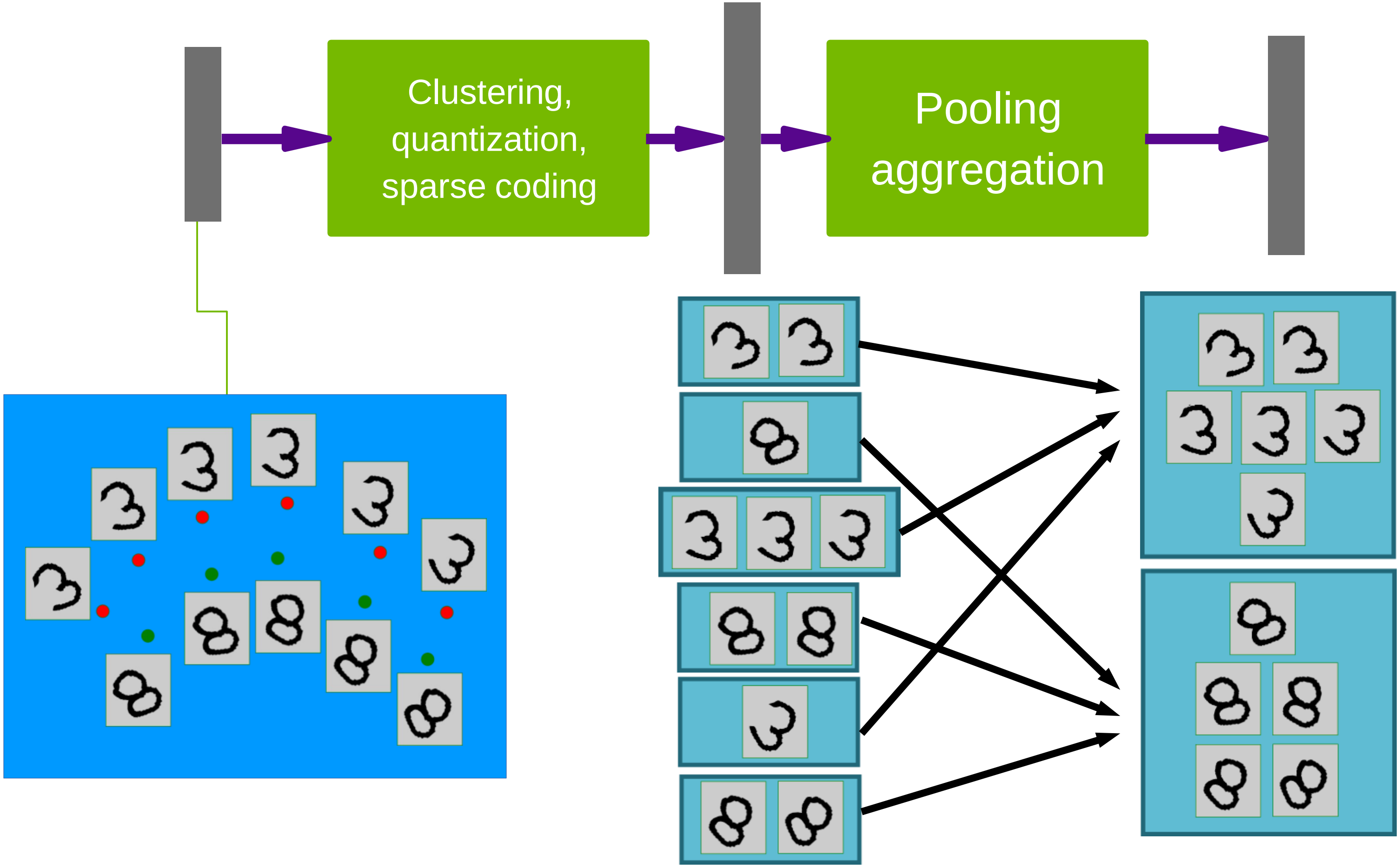


Non-linear dim expansion, disentangling

Pooling aggregation

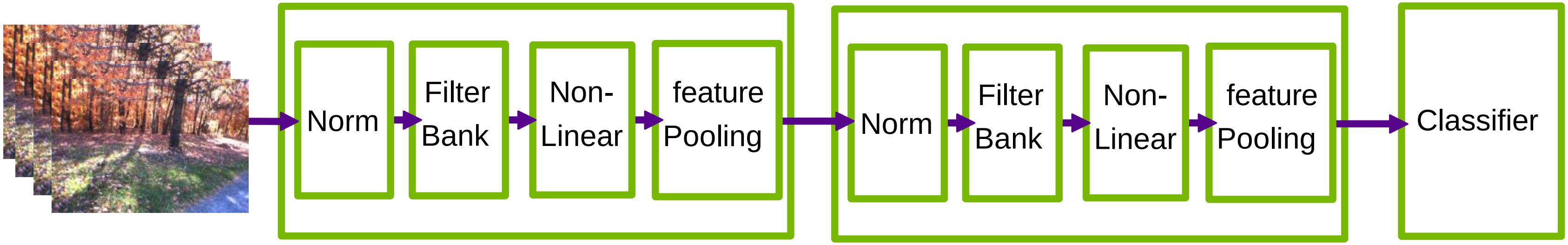
Sparse non-linear expansion → pooling

Use clustering to break things apart, pool together similar things



Overall architecture:

Normalization → filter bank → non-linearity → pooling



Norm

Filter Bank

Non-Linear

feature Pooling

Norm

Filter Bank

Non-Linear

feature Pooling

Classifier

- Stacking multiple stages of
  - [Normalization → filter bank → non-linearity → pooling].
- Normalization:
  - Subtractive: average removal, high pass filtering
  - Divisive: local contrast normalization, variance normalization
- Filter bank: dimension expansion, projection on overcomplete basis
- Non-linearity: sparsification, saturation, lateral inhibition....
  - Rectification (relu), component-wise shrinkage, tanh, winner-takes-all
- Pooling: aggregation over space or feature type
  - $X_i; \quad L_p: \sqrt[p]{X_i^p}; \quad PROB: \frac{1}{b}\log\left(\sum_i e^{bX_i}\right)$

References

Slide Sources

- NVIDIA Deep LearningTeaching Kit, Y.LeCun and M.A. Ranzato.

- https://en.wikipedia.org/wiki/Perceptron

- https://blog.algorithmia.com/introduction-to-loss-functions/

References

Convolutional nets

- LeCun, Bottou, Bengio and Haffner: Gradient-Based Learning Applied to Document Recognition, Proceedings of the IEEE, 86(11):2278-2324, November 1998
- Krizhevsky, Sutskever, Hinton “ImageNet Classification with deep convolutional neural networks” NIPS 2012
- Jarrett, Kavukcuoglu, Ranzato, LeCun: What is the Best Multi-Stage Architecture for Object Recognition?, Proc. International Conference on Computer Vision (ICCV'09), IEEE, 2009
- Kavukcuoglu, Sermanet, Boureau, Gregor, Mathieu, LeCun: Learning Convolutional Feature Hierarchies for Visual Recognition, Advances in Neural Information Processing Systems (NIPS 2010), 23, 2010
- see yann.lecun.com/exdb/publis for references on many different kinds of convnets.
- see http://www.cmap.polytechnique.fr/scattering/ for scattering networks (similar to convnets but with less learning and stronger mathematical foundations)

References

Applications of RNNs

- Mikolov “Statistical language models based on neural networks” PhD thesis 2012
- Boden “A guide to RNNs and backpropagation” Tech Report 2002
- Hochreiter, Schmidhuber “Long short term memory” Neural Computation 1997
- Graves “Offline arabic handwrting recognition with multidimensional neural networks” Springer 2012
- Graves “Speech recognition with deep recurrent neural networks” ICASSP 2013

References

Applications of convolutional nets

- Farabet, Couprie, Najman, LeCun, “Scene Parsing with Multiscale Feature Learning, Purity Trees, and Optimal Covers”, ICML 2012
- Pierre Sermanet, Koray Kavukcuoglu, Soumith Chintala and Yann LeCun: Pedestrian Detection with Unsupervised Multi-Stage Feature Learning, CVPR 2013
- D. Ciresan, A. Giusti, L. Gambardella, J. Schmidhuber. Deep Neural Networks Segment Neuronal Membranes in Electron Microscopy Images. NIPS 2012
- Raia Hadsell, Pierre Sermanet, Marco Scoffier, Ayse Erkan, Koray Kavackuoglu, Urs Muller and Yann LeCun: Learning Long-Range Vision for Autonomous Off-Road Driving, Journal of Field Robotics, 26(2):120-144, February 2009
- Burger, Schuler, Harmeling: Image Denoising: Can Plain Neural Networks Compete with BM3D?, Computer Vision and Pattern Recognition, CVPR 2012,

References

Deep learning & energy-based models

- Deep learning & energy-based models
  - Y. Bengio, Learning Deep Architectures for AI, Foundations and Trends in Machine Learning, 2(1), pp.1-127, 2009.
  - LeCun, Chopra, Hadsell, Ranzato, Huang: A Tutorial on Energy-Based Learning, in Bakir, G. and Hofman, T. and Schölkopf, B. and Smola, A. and Taskar, B. (Eds), Predicting Structured Data, MIT Press, 2006
  - M. Ranzato Ph.D. Thesis “Unsupervised Learning of Feature Hierarchies” NYU 2009
- Practical guide
  - Y. LeCun et al. Efficient BackProp, Neural Networks: Tricks of the Trade, 1998
  - L. Bottou, Stochastic gradient descent tricks, Neural Networks, Tricks of the Trade Reloaded, LNCS 2012.
  - Y. Bengio, Practical recommendations for gradient-based training of deep architectures, ArXiv 2012