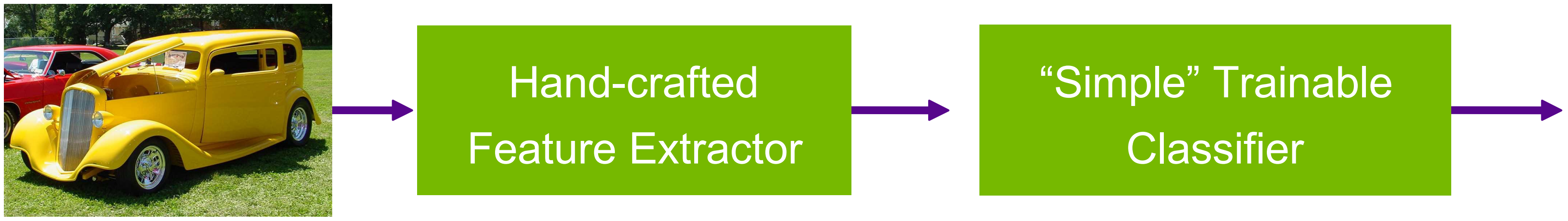
History of Convolutional Networks

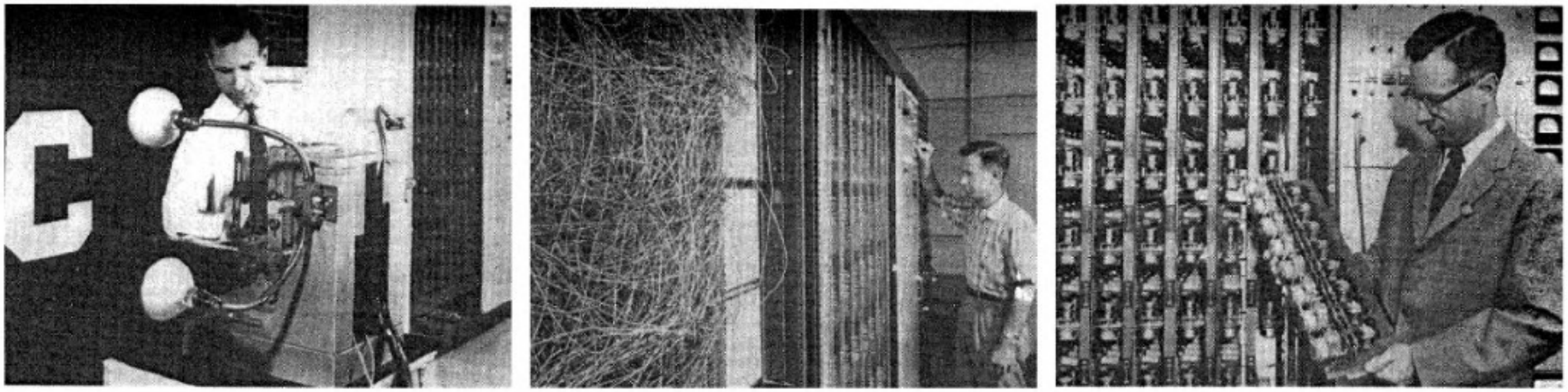
Jay Urbain, PhD
Credits: NYU, nVidia DLI

55 years of hand-crafted features

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



- Perceptron



Architecture of “classical” recognition systems

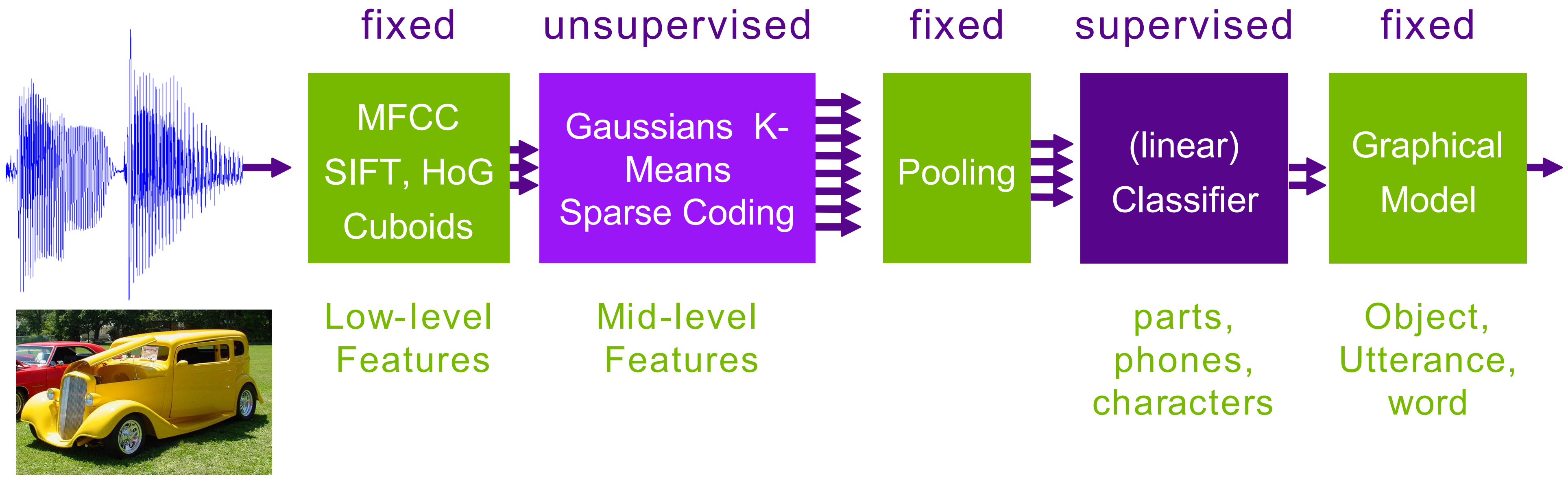
“Classic” architecture for pattern recognition

– Speech recognition: 1990-2011

– Object Recognition: 2005-2012

– Handwriting recognition (long ago)

– Graphical model has latent variables (locations of parts)



Architecture of deep learning-based recognition systems

“Deep” architecture for pattern recognition

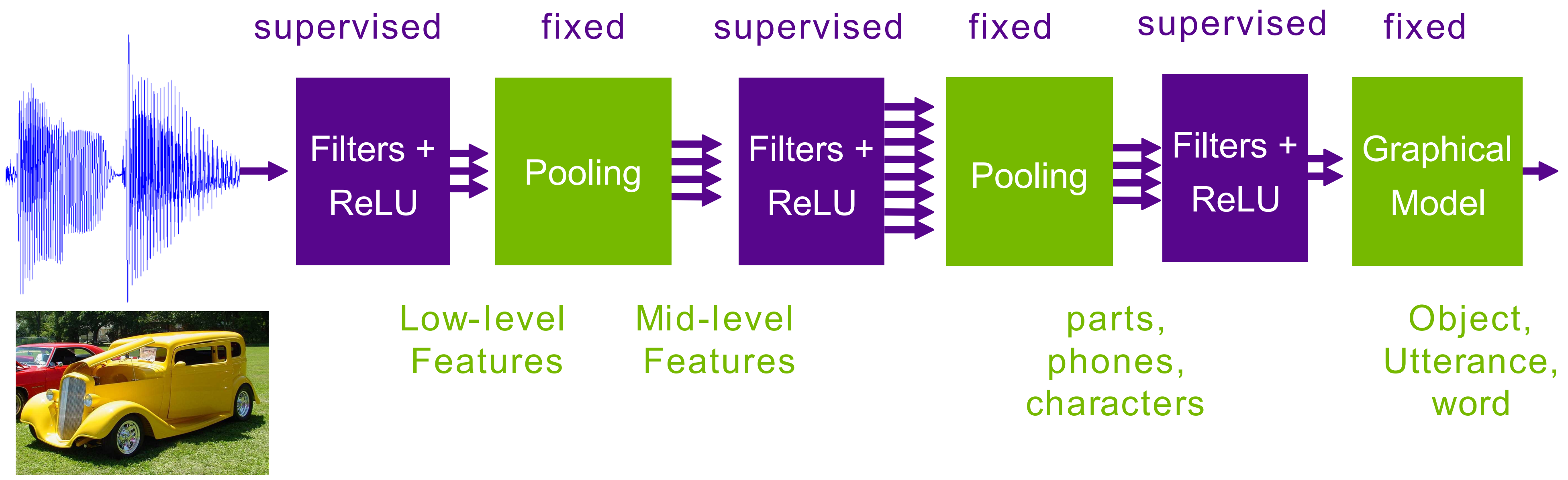
– Speech, and Object recognition: since 2011/2012

– Handwriting recognition: since the early 1990s

– Convolutional Net with optional Graphical Model on top

– Trained purely supervised

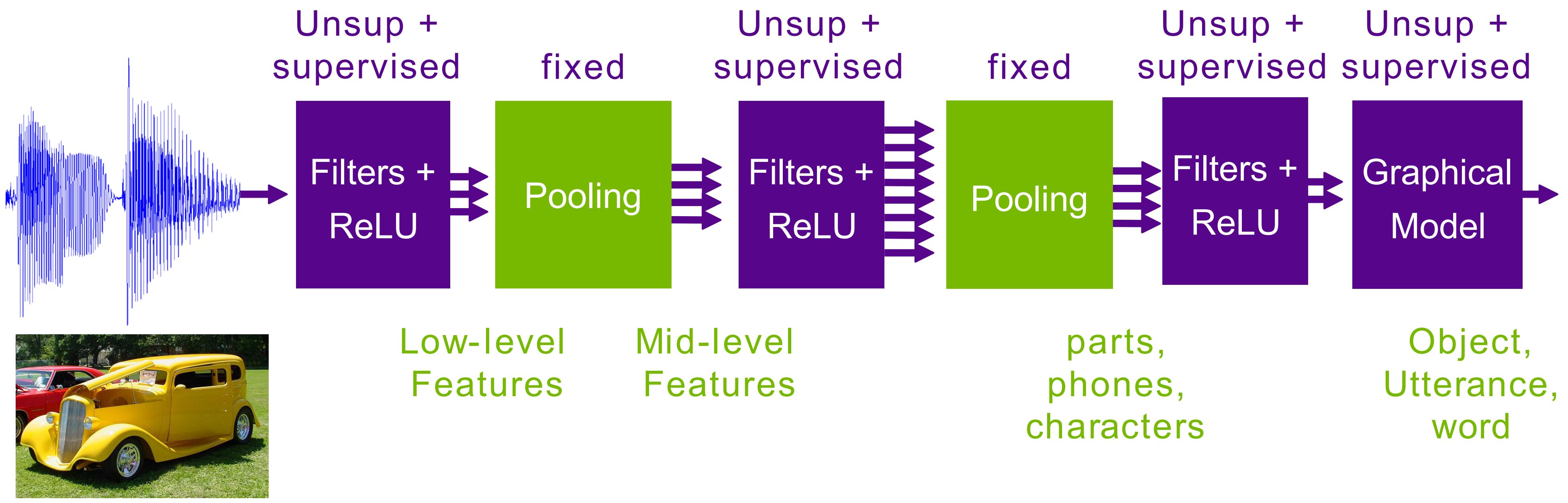
– Graphical model has latent variables (locations of parts)



Future Systems: deep learning + structured prediction

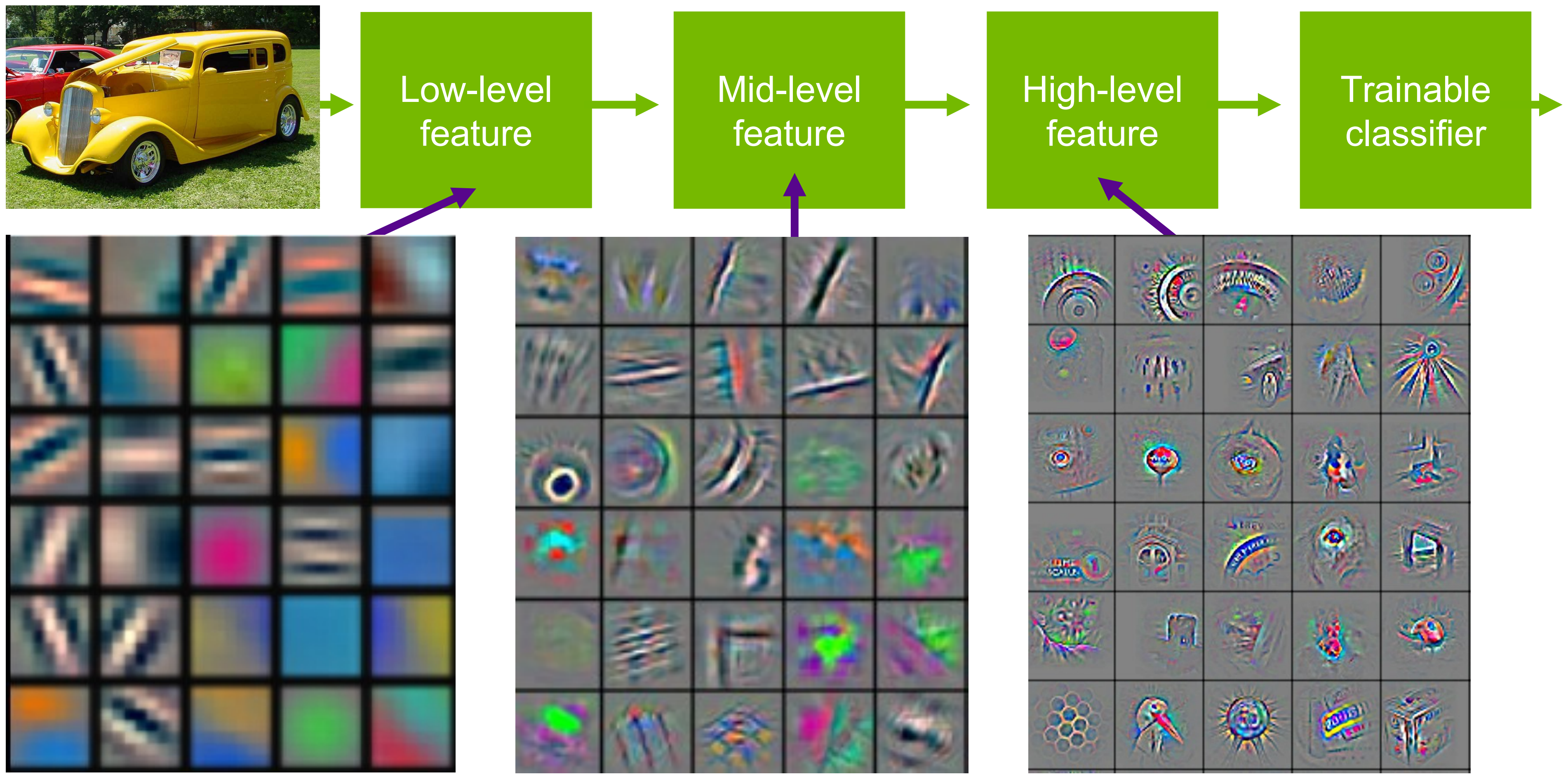
Globally-trained deep architecture

- Handwriting recognition: since the mid 1990s
- Speech Recognition: since 2011
- All the modules are trained with a combination of unsupervised and supervised learning
- **End-to-end training == deep structured prediction**



Deep learning = learning hierarchical representations

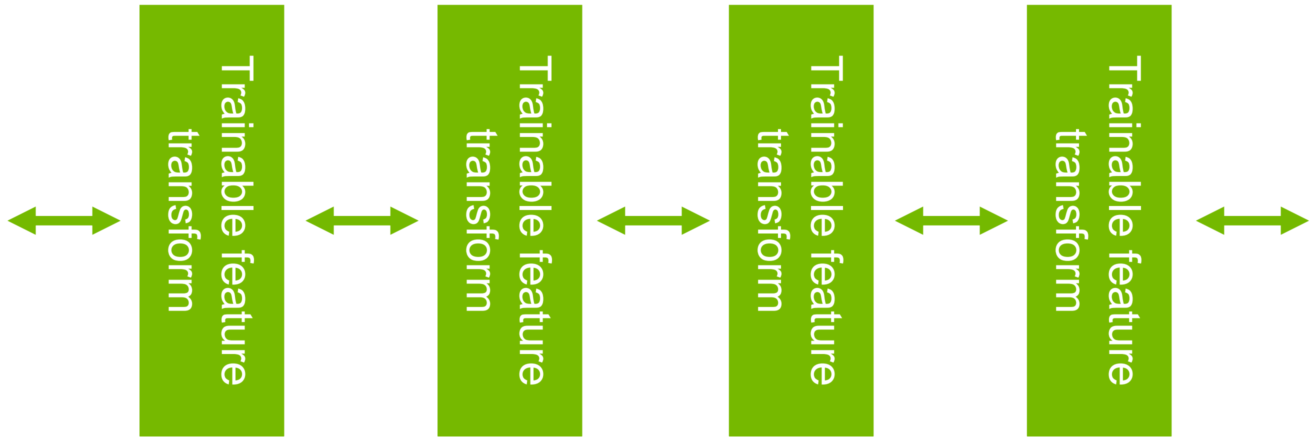
It's deep if it has more than one stage of non-linear feature transformation



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

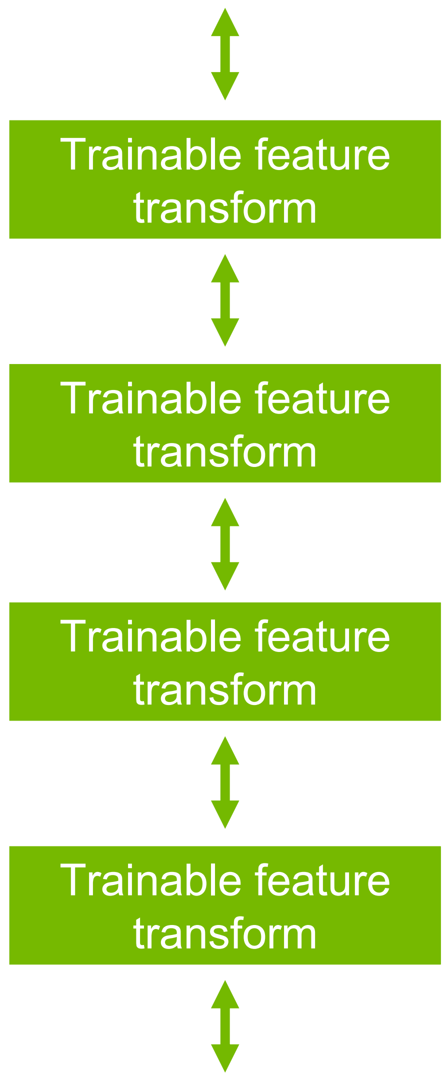
Trainable feature hierarchy

- 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...

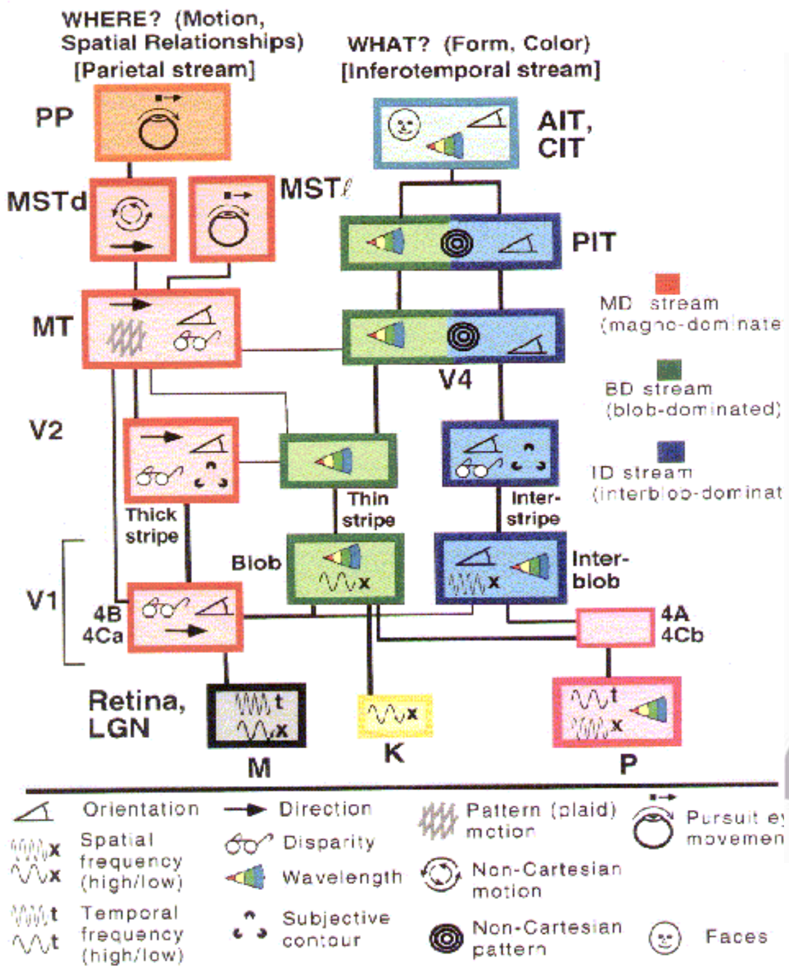
- How do we learn representations of the perceptual world?
  - How can a perceptual system build itself by looking at the world?
  - How much prior structure is necessary
- ML/AI: how do we learn features or feature hierarchies?
  - What is the fundamental principle? What is the learning algorithm? What 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

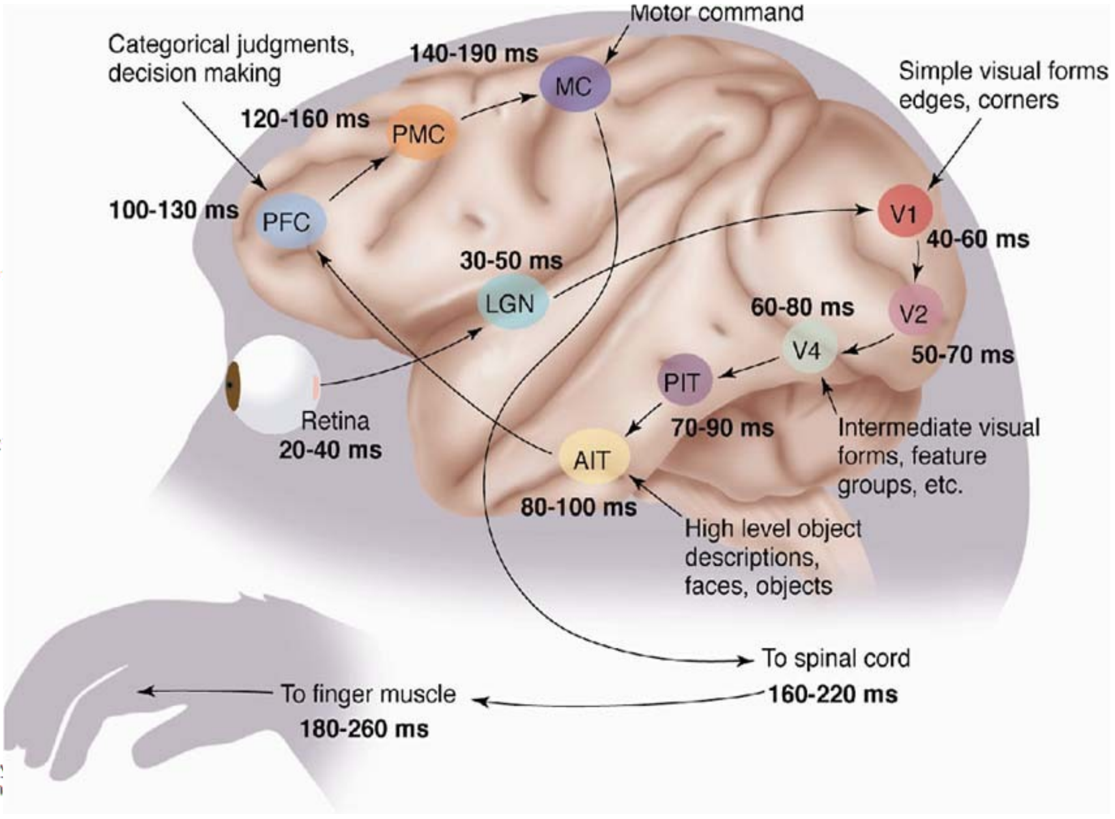

Trainable feature transform
Trainable feature transform
Trainable feature transform
Trainable feature transform

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





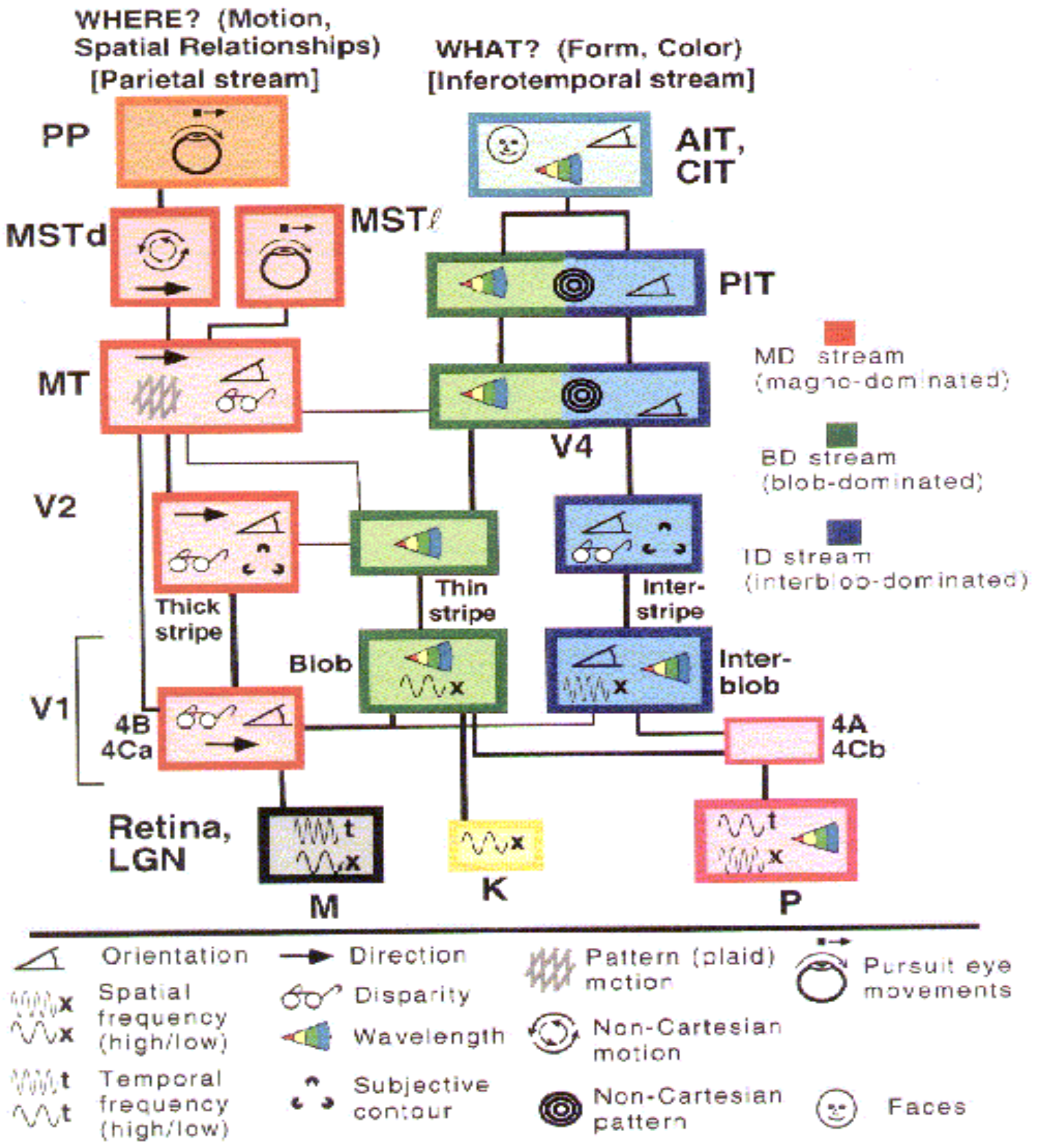
[picture from Simon Thorpe]

[Gallant & Van Essen]

Architecture of the mammalian visual cortex

- Ventral pathway = “what”
- dorsal pathway = “where”
- It's hierarchical
- There is feedback
- There is motion processing
- Learning is mostly unsupervised
- It does recognition, localization, navigation, grasping.....

[Gallant & Van Essen]



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

- It's nice 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
- QUESTION: What is the equivalent of aerodynamics for understanding intelligence?

L'Avion III de Clément Ader, 1897

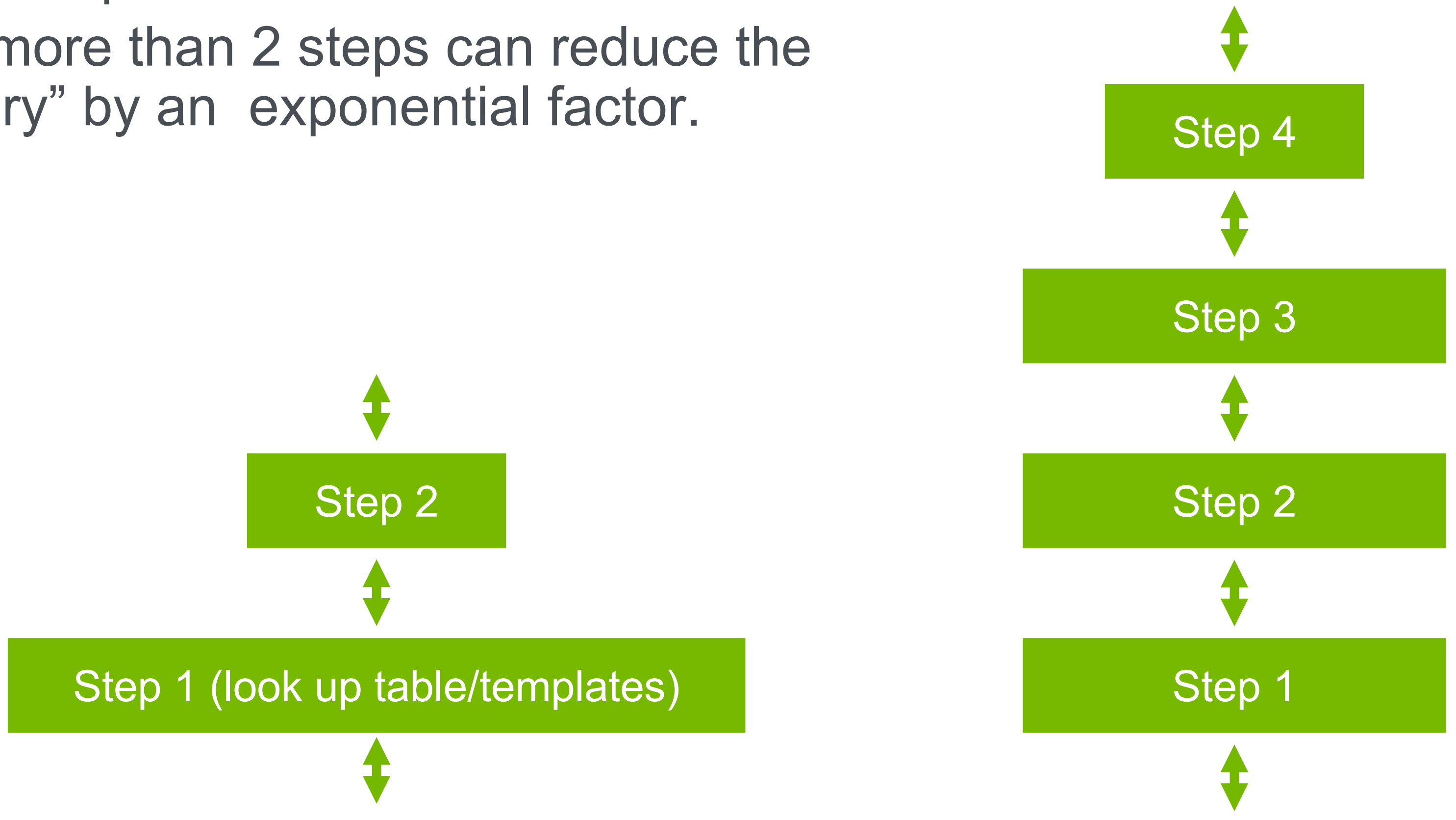
(Musée du CNAM, Paris)

His Eole took off from the ground in 1890, 13 years before the Wright Brothers, but you probably never heard of it.

Shallow vs Deep == lookup table vs multi-step algorithm

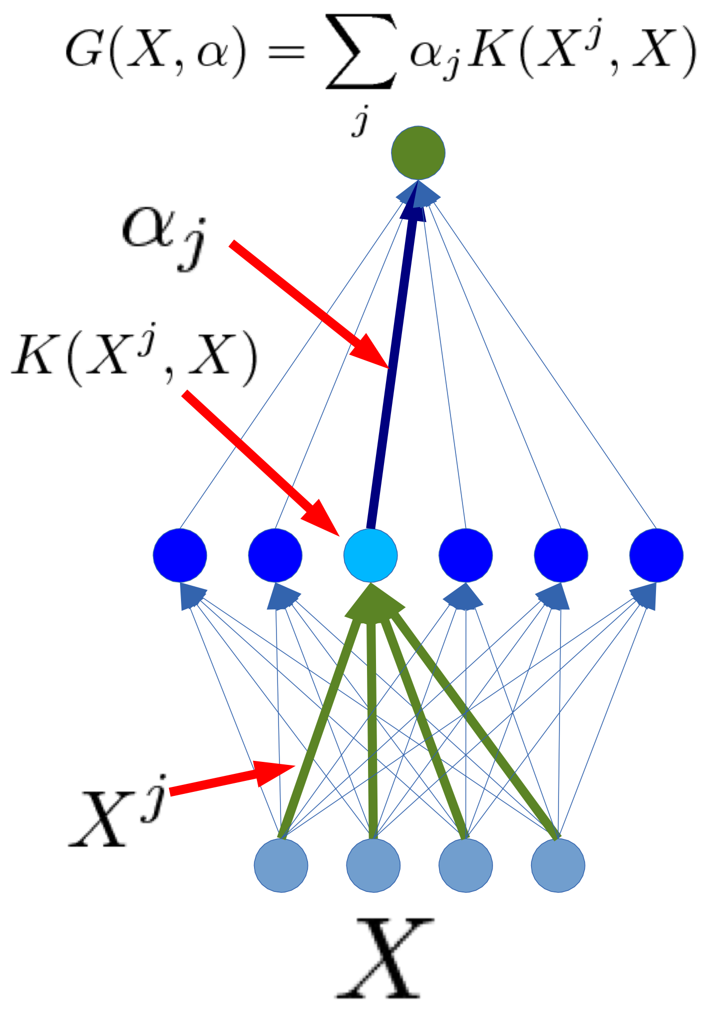
“Shallow & wide” vs “deep and narrow” == “more memory” vs “more time”

- Look-up table vs algorithm
- Few functions can be computed in two steps without an exponentially large lookup table
- Using more than 2 steps can reduce the “memory” by an exponential factor.



Which models are deep?

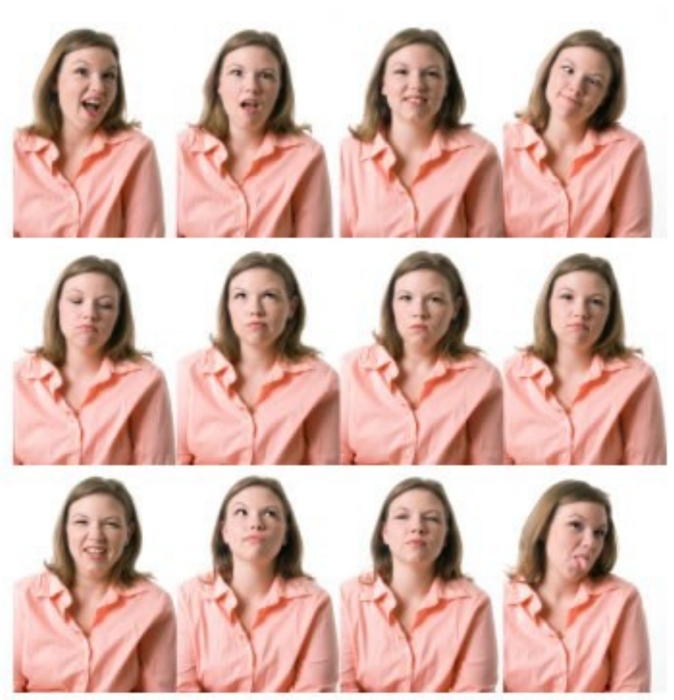
- 2-layer models are not deep (even if you train the first layer)
  - Because there is no feature hierarchy
- Neural nets with 1 hidden layer are not deep
- SVMs and Kernel methods are not deep
  - Layer1: kernels; layer2: linear
  - The first layer is “trained” in with the simplest unsupervised method ever devised: using the samples as templates for the kernel functions.
  - “glorified template matching”
- Classification trees are not deep
  - No hierarchy of features. All decisions are made in the input space

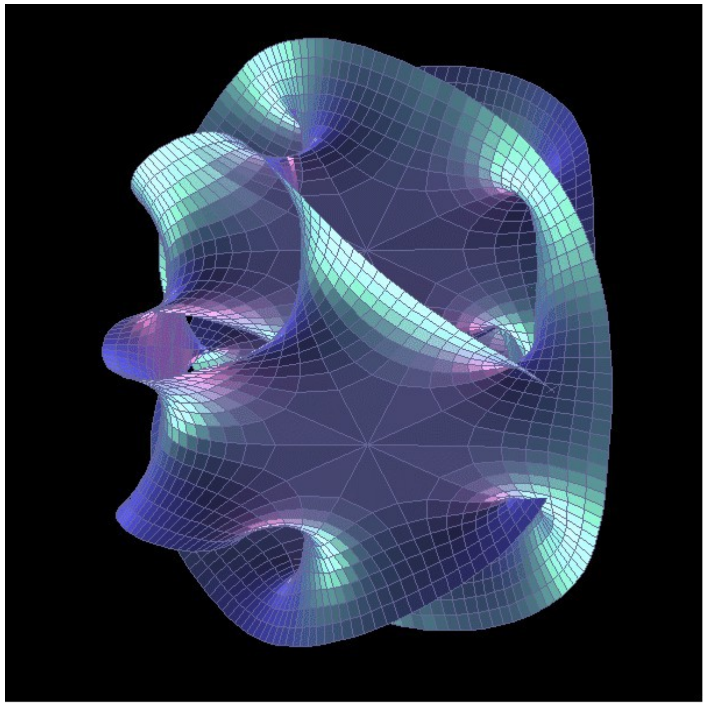


What are good features?

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

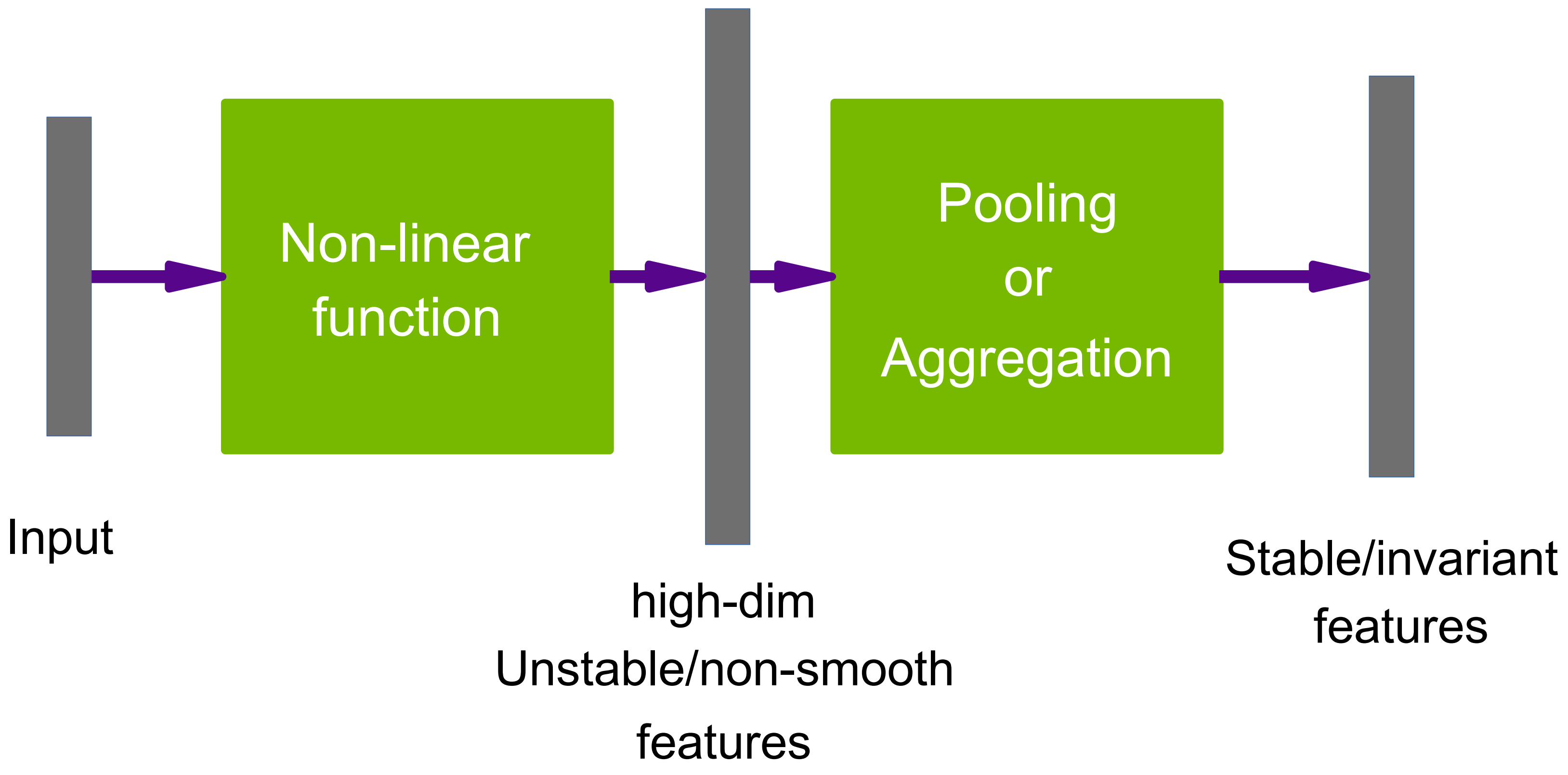
- Learning representations of data:
  - Discovering & disentangling the independent explanatory factors
- The manifold hypothesis:
  - Natural data lives in a low-dimensional (non-linear) manifold
  - Because variables in natural data are mutually dependent





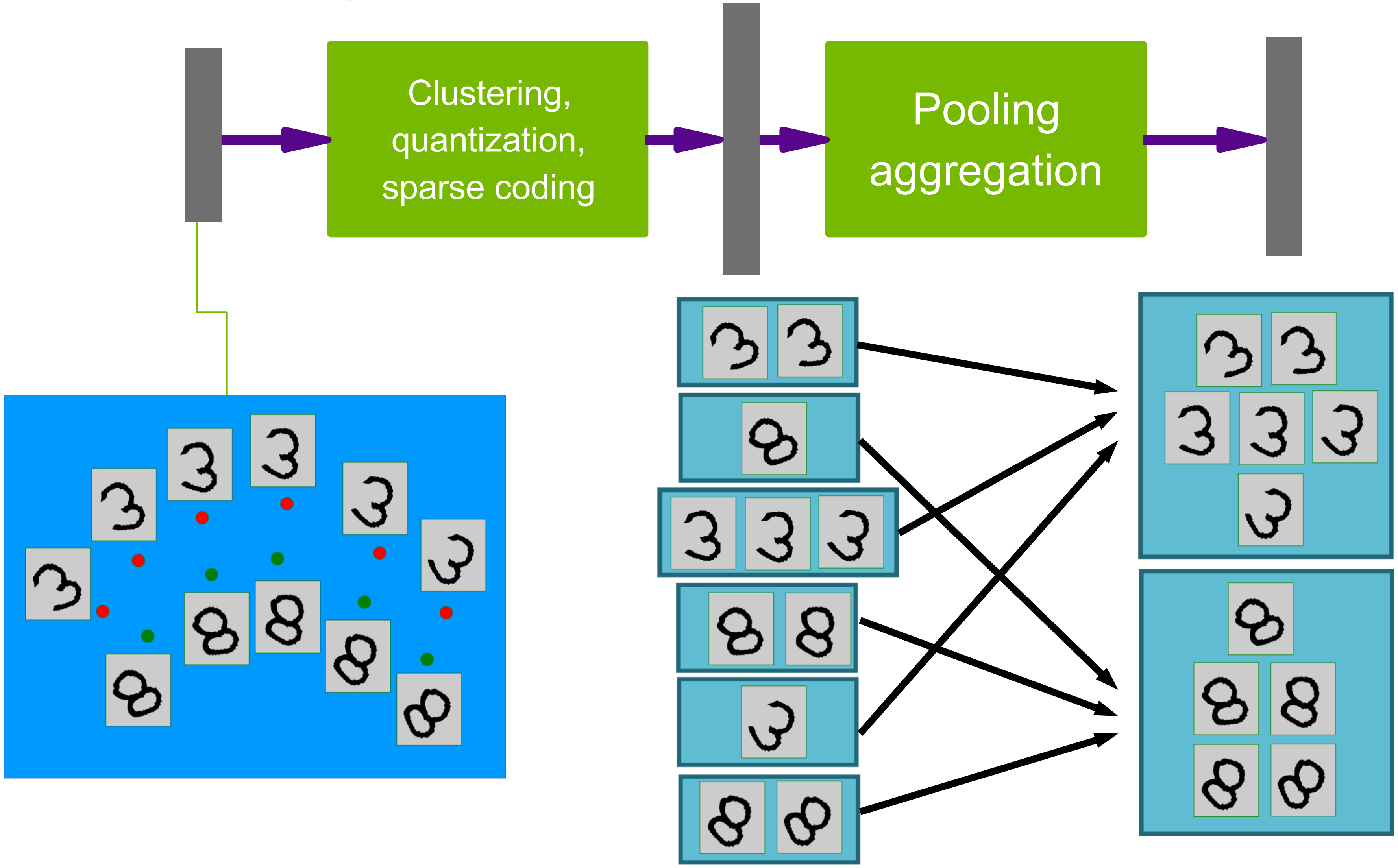
Basic idea for invariant feature learning

- 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. Like pooling.

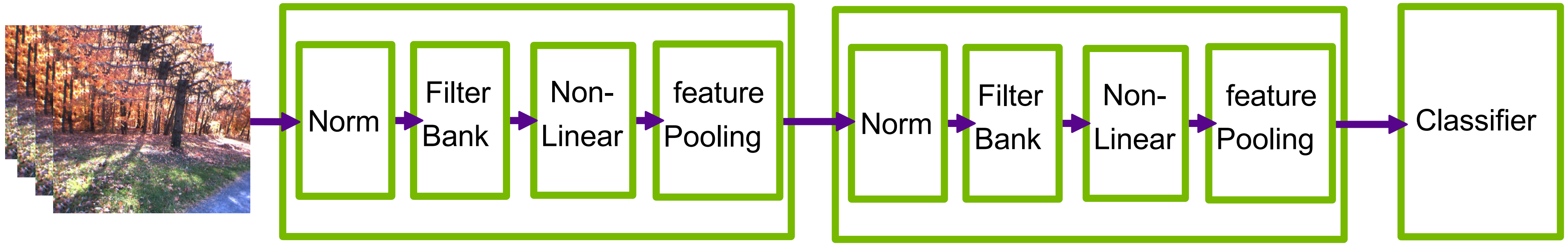

Non-linear function
Pooling or Aggregation
Input
high-dim Unstable/non-smooth features
Stable/invariant features

Sparse non-linear expansion → pooling

Use clustering to break things apart, pool together similar things

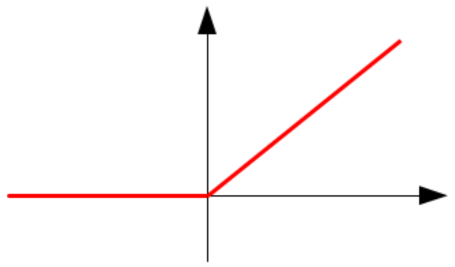


Overall architecture: multiple stages of Normalization → filter bank → non-linearity → pooling



- Normalization: variations on whitening
  - 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,..

    ReLU (x)=max (x , 0)



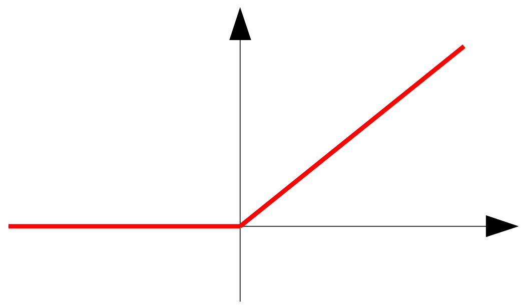
- Pooling: aggregation over space or feature type
  - Max, Lp norm, log prob.

$$X_i;\quad L_p : \sqrt[p]{X_i^p};\quad PROB : \frac{1}{b}\log\left(\sum_i e^{bX_i}\right)$$

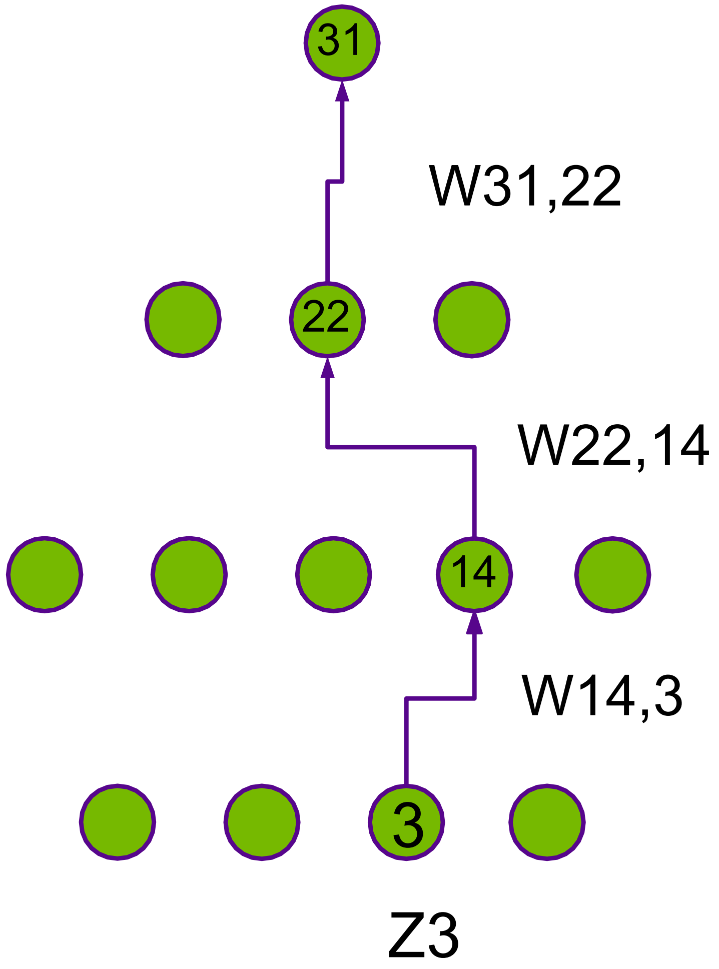
Deep nets with ReLUs and max pooling

- Stack of linear transforms interspersed with Max operators
- Point-wise ReLUs:

$$ReLU(x) = max(x, 0)$$

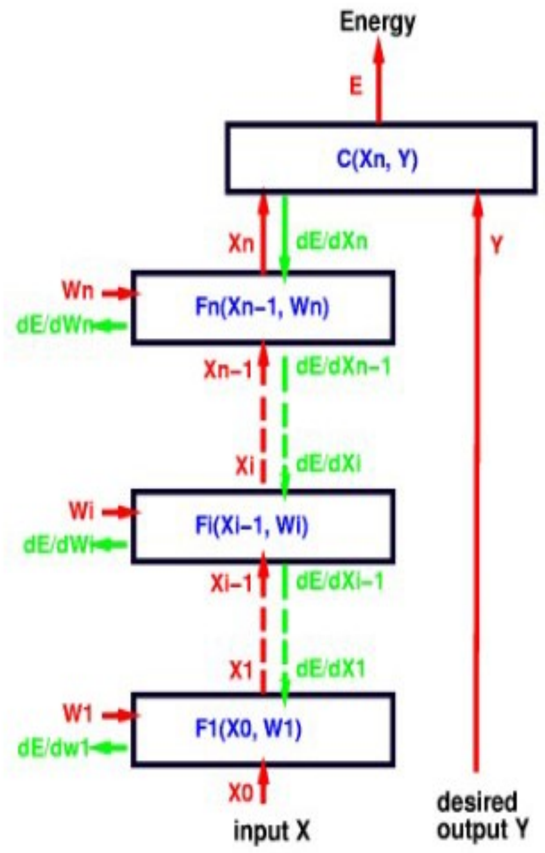


- Max Pooling
  - "switches" from one layer to the next



Supervised training: stochastic (sub) gradient optimization

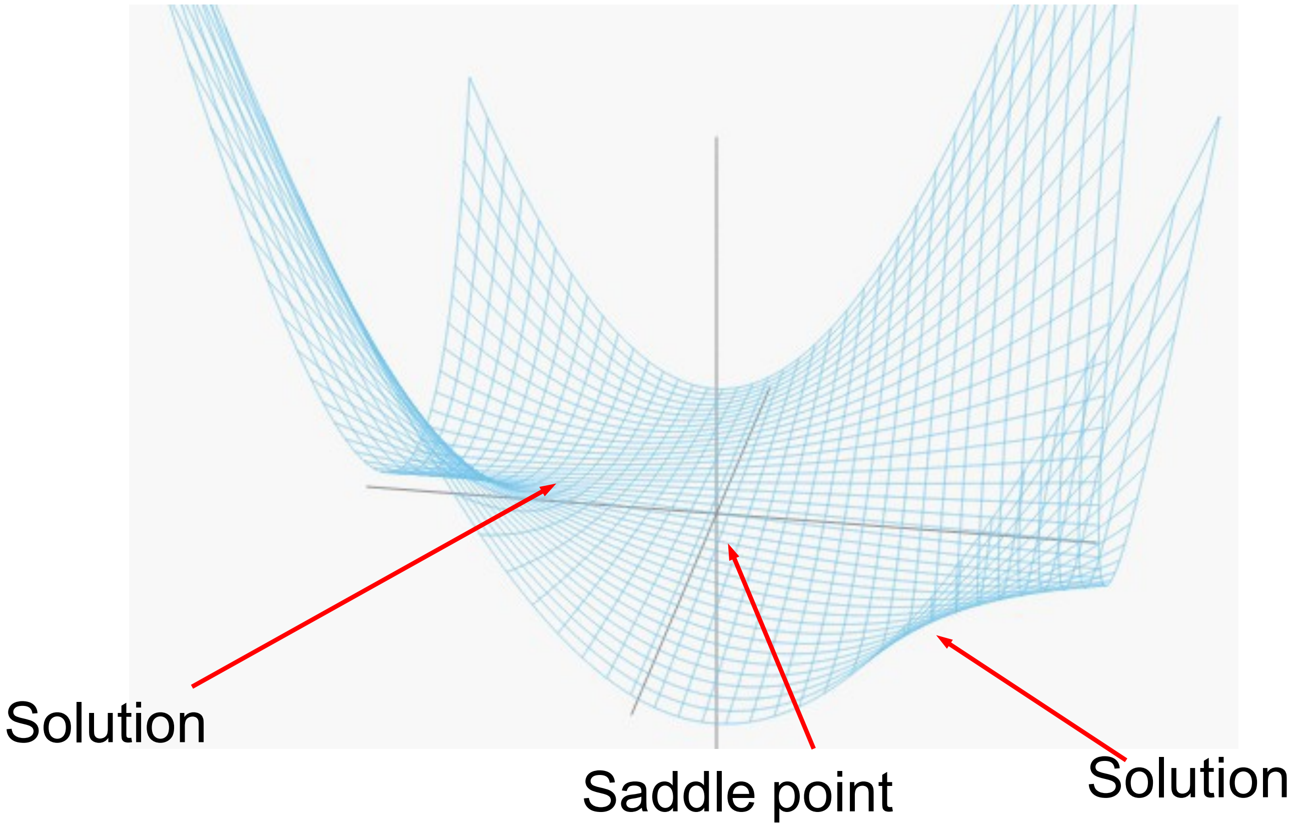
– To compute all the derivatives, we use a backward sweep called the back-propogation algorithm that uses the recurrence equation for $\frac{\partial E}{\partial X_i}$

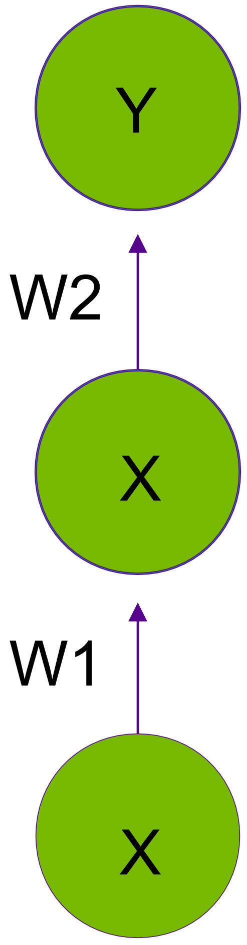


- $\frac{\partial E}{\partial X_n} = \frac{\partial C(X_n,Y)}{\partial X_n}$
- $\frac{\partial E}{\partial X_{n-1}} = \frac{\partial E}{\partial X_n} \frac{\partial F_n(X_{n-1},W_n)}{\partial X_{n-1}}$
- $\frac{\partial E}{\partial W_n} = \frac{\partial E}{\partial X_n} \frac{\partial F_n(X_{n-1},W_n)}{\partial W_n}$
- $\frac{\partial E}{\partial X_{n-2}} = \frac{\partial E}{\partial X_{n-1}} \frac{\partial F_{n-1}(X_{n-2},W_{n-1})}{\partial X_{n-2}}$
- $\frac{\partial E}{\partial W_{n-1}} = \frac{\partial E}{\partial X_{n-1}} \frac{\partial F_{n-1}(X_{n-2},W_{n-1})}{\partial W_{n-1}}$
- ....etc, until we reach the first module.
- we now have all the $\frac{\partial E}{\partial W_i}$ for $i \in [1, n]$.

Loss function for a simple network

- 1-1-1 network
  - Y = W1*W2*X
- trained to compute the identity function with quadratic loss
  Single sample X=1, Y=1 L(W) = (1-W1*W2)^2





Deep nets with ReLUs

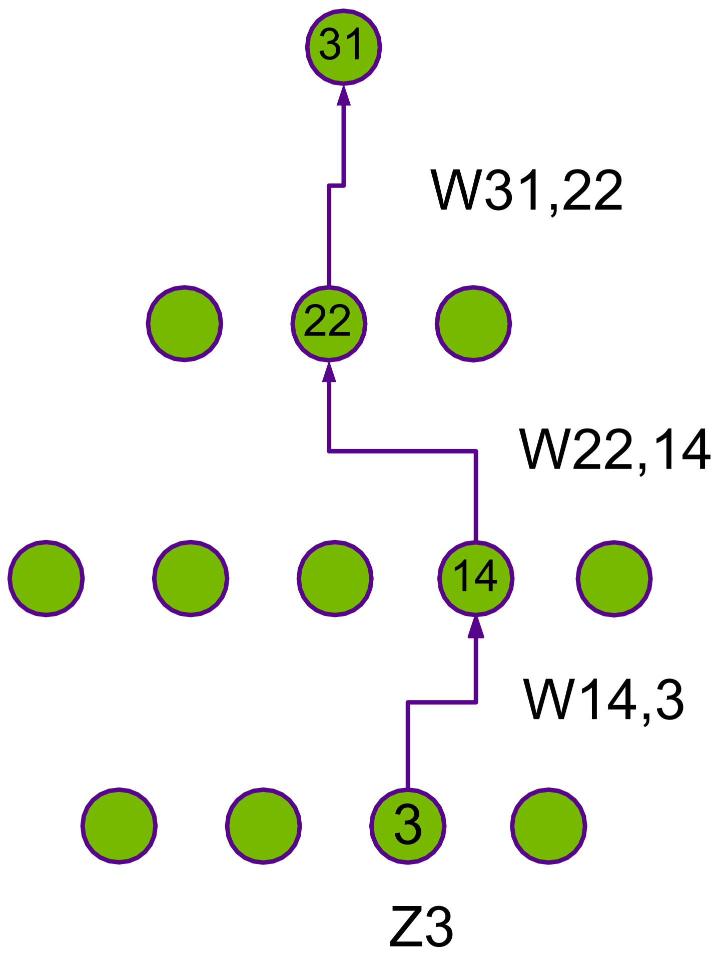
- Single output:

$$\hat{Y} = \sum_{P} \delta_P(W, X)\left(\prod_{(ij)\in P} W_{ij}\right) X_{P_{start}}$$

- Wij: weight from j to i
- P: path in network from input to output
  - P=(3,(14,3),(22,14),(31,22))
- di: 1 if ReLU i is linear, 0 if saturated.
- Xpstart: input unit for path P.

$$\hat{Y} = \sum_{P} \delta_P(W, X)\left(\prod_{(ij)\in P} W_{ij}\right) X_{P_{start}}$$

- Dp(W,X): 1 if path P is “active”, 0 if inactive
- Input-output function is piece-wise linear
- Polynomial in W with random coefficients



Deep convolutional nets (and other deep neural nets)

– Training sample: (Xi,Yi) k=1 to K

– Objective function (with margin-type loss = ReLU)

$$L(W)=\sum_k ReLU(1-Y^k \sum_P \delta_P(W,X^k)(\prod_{(ij)\in P} W_{ij}) X^k_{P_{start}})$$

$$L(W)=\sum_k \sum_P (X^k_{P_{start}} Y^k)\delta_P(W,X^k)(\prod_{(ij)\in P} W_{ij})$$

$$L(W)=\sum_P [\sum_k (X^k_{P_{start}} Y^k)\delta_P(W,X^k)](\prod_{(ij)\in P} W_{ij})$$

$$L(W)=\sum_P C_p(X,Y,W)(\prod_{(ij)\in P} W_{ij})$$

– Polynomial in W of degree l (number of adaptive layers)

– Continuous, piece-wise polynomial with “switched” and partially random coefficients

  – Coefficients are switched in an out depending on W

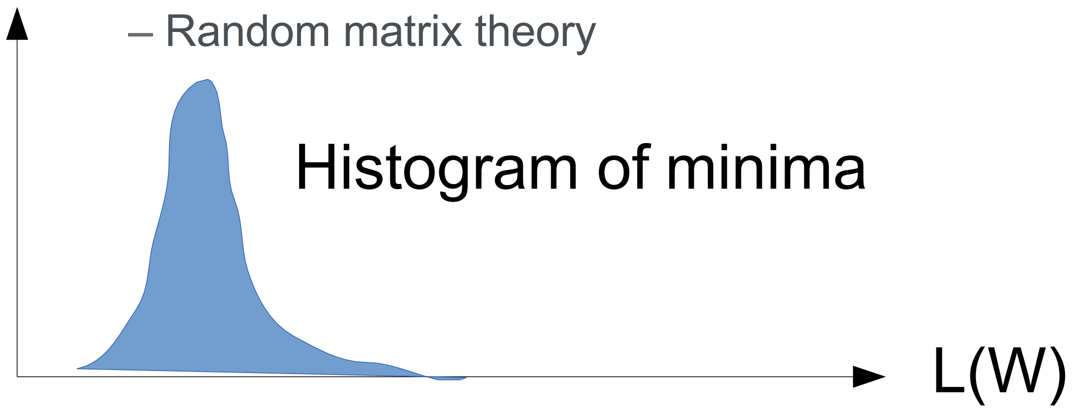
Deep nets with ReLUs:

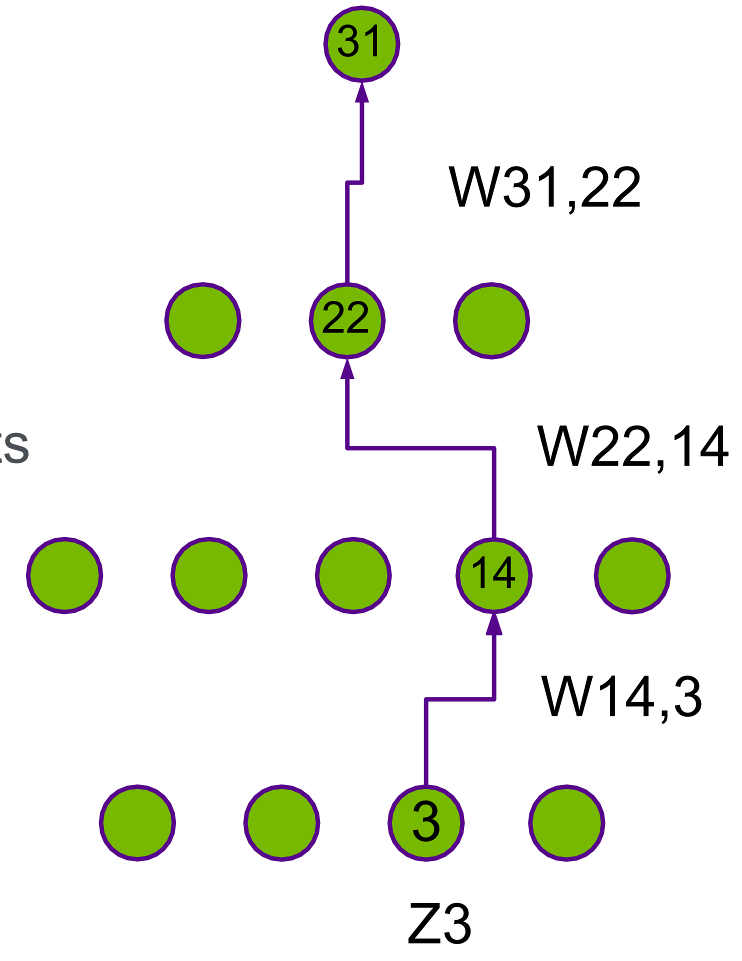
Objective function is piecewise polynomial

- If we use a hinge loss, delta now depends on label Yk:

$$L(W) = \sum_P C_p(X, Y, W) \left(\prod_{(ij) \in P} W_{ij} \right)$$

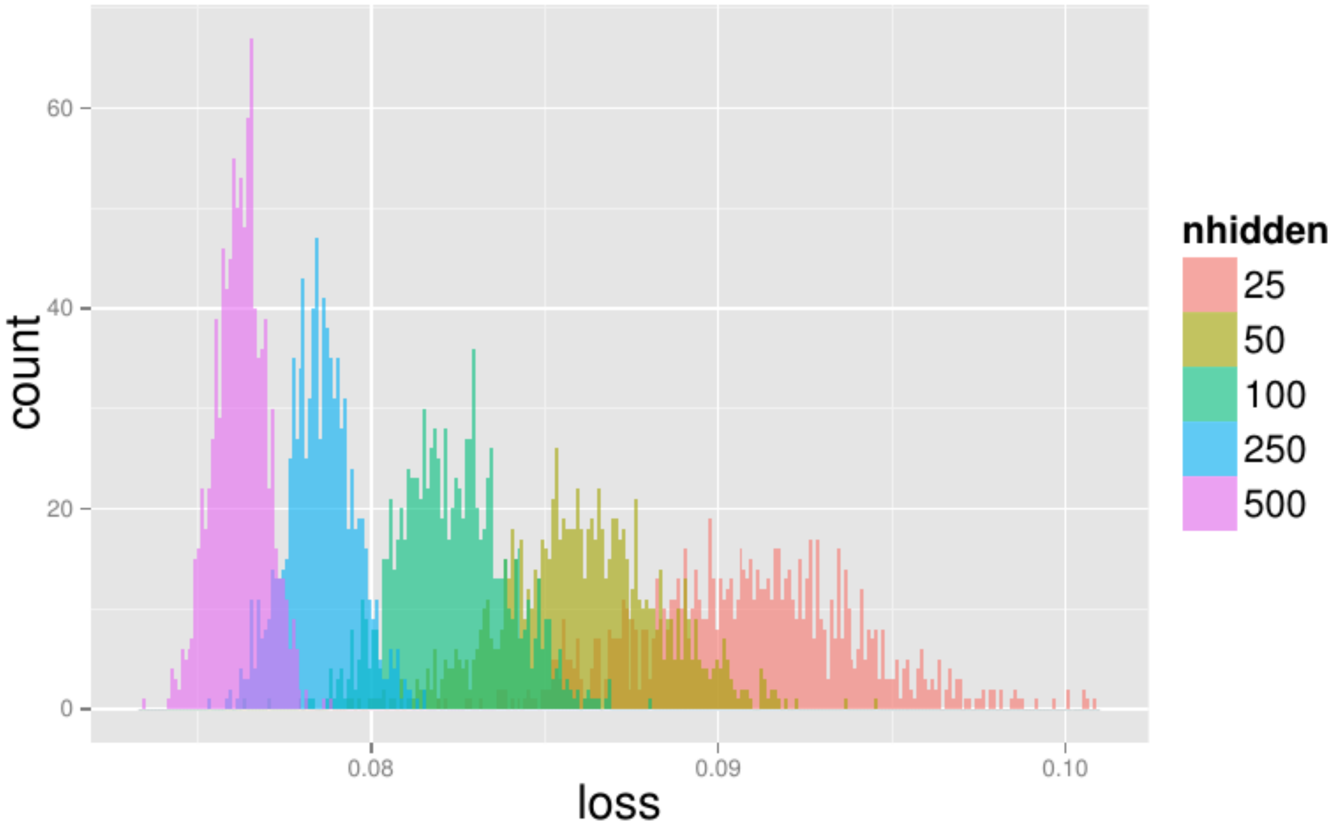
- Piecewise polynomial in W with random coefficients
- A lot is known about the distribution of critical points of polynomials on the sphere with random (Gaussian) coefficients [Ben Arous et al.]
  - High-order spherical spin glasses
  - Random matrix theory





Deep nets with ReLUs: objective function is piecewise polynomial

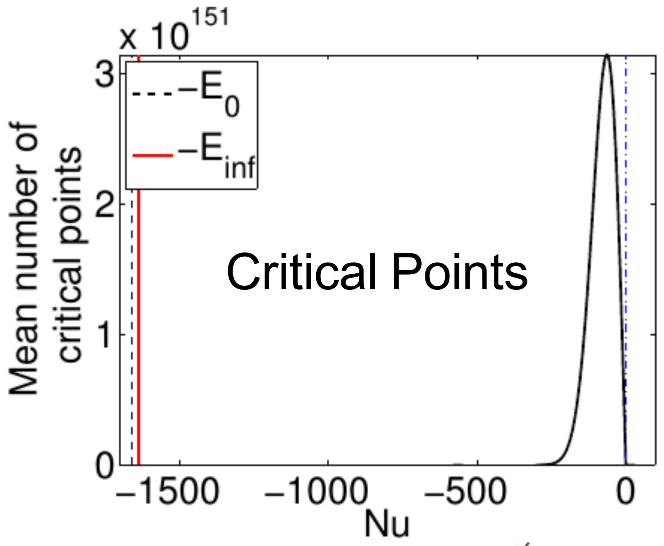
– Train 2-layer nets on scaled-down MNIST (10x10) from multiple initial conditions. Measure loss on test set.

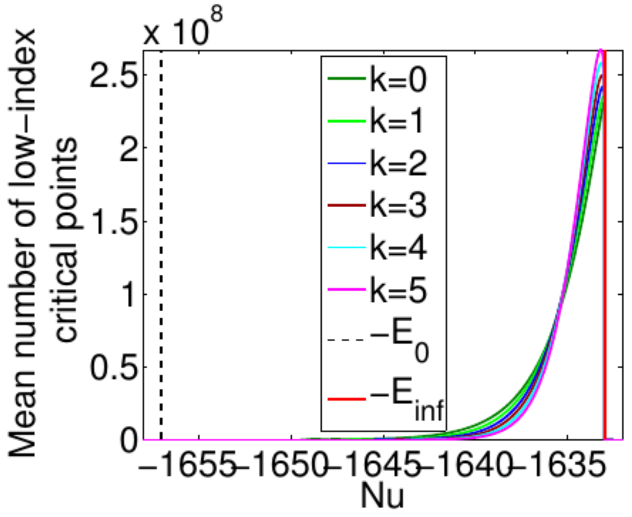


[Choromanska, Henaff, Mathieu, Ben Arous, LeCun 2015]

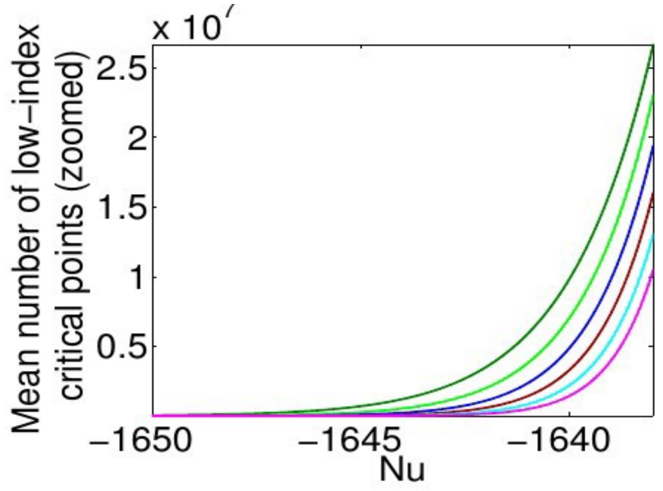
Spherical spin glass theory

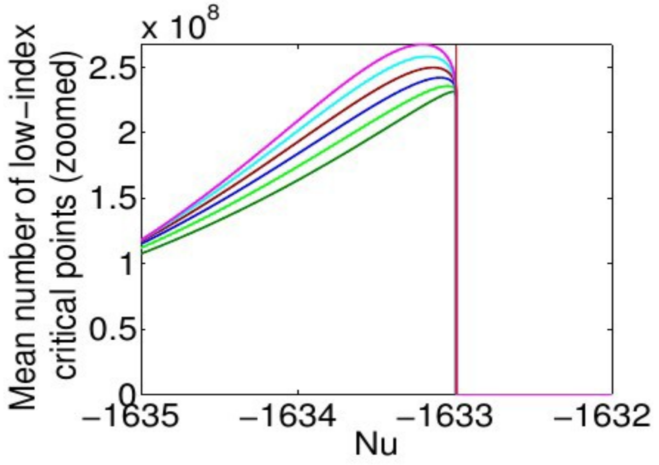
- Distribution of critical points (saddle points, minima, maxima)
  - K=number of negative eigenvalues of Hessian (K=0 → minimum)





Zoomed:


x 10^7
Mean number of low−index critical points (zoomed)
Nu


x 10^8
Mean number of low−index critical points (zoomed)
Nu