

Visual Perception with Deep Learning

Yann LeCun

The Courant Institute of Mathematical Sciences

New York University

joint work with: Marc'Aurelio Ranzato,

Y-Lan Boureau, Koray Kavackuoglu,

Fu-Jie Huang, Raia Hadsell

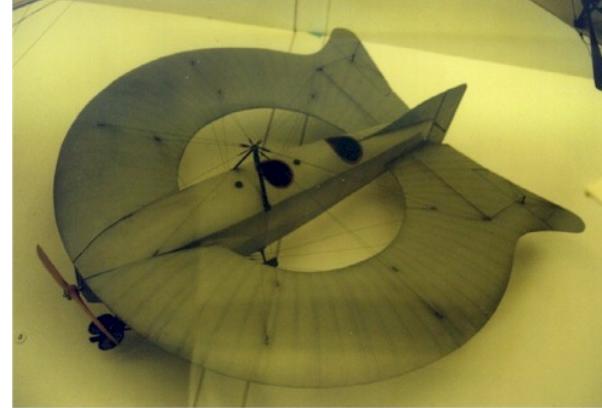
Challenges of Visual Neuroscience (and Computer Vision)

➊ How do we learn “invariant representations”?

- ▶ From the image of an airplane, how do we extract a representation that is invariant to pose, illumination, background, clutter, object instance....
- ▶ How can a human (or a machine) learn those representations by just looking at the world?

➋ How can we learn visual categories from just a few examples?

- ▶ I don't need to see many airplanes before I can recognize every airplane (even really weird ones)



Challenges of Visual Neuroscience (and Computer Vision)

- ➊ **The recognition of everyday objects is a very fast process.**

- ▶ Experiments by Simon Thorpe and others have shown that the recognition of common objects is essentially “feed forward.”
- ▶ Not all of vision is feed forward (what would all those feed-back connection be there for?).

- ➋ **How much of the visual system is the result of learning?**

- ▶ How much prior structure must be built into the visual system to enable it to learn to see?
- ▶ Are V1/V2/V4 neurons learned or hard-wired?

- ➌ **If the visual system is learned, what is the learning algorithm?**

- ▶ What learning algorithm can train neural network as “deep” as the visual system (10 layers?).

The visual system is “deep” and learned

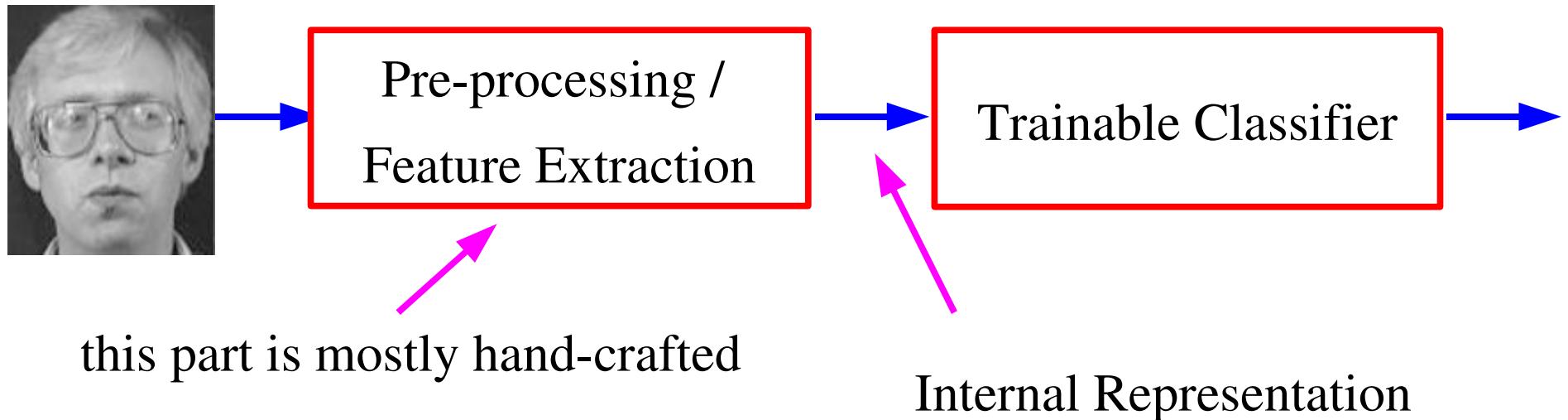
- ➊ The primate's visual system is “deep”

- ▶ It has 10-20 layers of neurons from the retina to the infero-temporal cortex (where object categories are encoded).
- ▶ How does it train itself by just looking at the world?.

- ➋ Is there a magic bullet for visual learning?

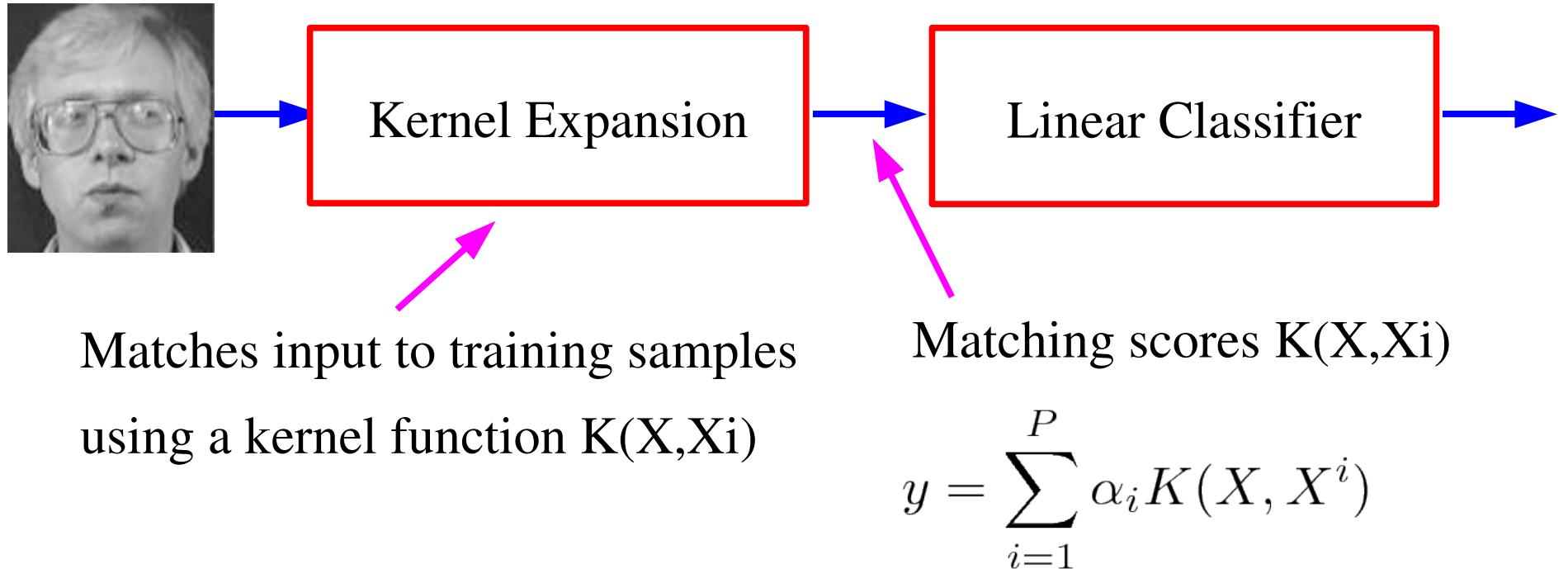
- ▶ The neo-cortex is pretty much the same all over
- ▶ The “learning algorithm” it implements is not specific to a modality (what works for vision works for audition)
- ▶ There is evidence that everything is learned, down to low-level feature detectors in V1
- ▶ Is there a **universal learning algorithm/architecture** which, given a small amount of appropriate prior structure, can produce an intelligent vision system?
- ▶ Or do we have to keep accumulating a large repertoire of pre-engineered “modules” to solve every specific problem an intelligent vision system must solve?

The Traditional “Shallow” Architecture for Recognition



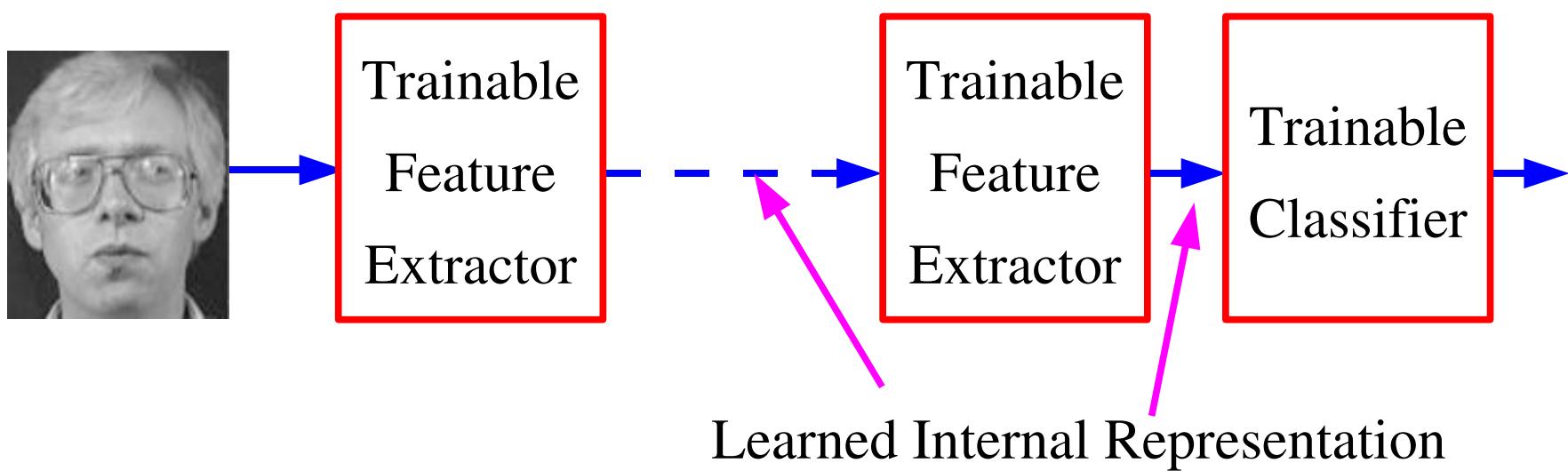
- ➊ The raw input is pre-processed through a hand-crafted feature extractor
- ➋ The trainable classifier is often generic (task independent)
- ➌ The most common Machine Learning architecture: the Kernel Machine
 - ▶ kernel machines are shallow

Kernel Machine (the most popular ML architecture)



- Kernel methods are very efficient for many applications **BUT**
- A kernel machine is a glorified template matcher
- How can we ever expect this to solve complex AI-type problems such as invariant visual object recognition?

“Deep” Learning: Learning Internal Representations



- ➊ Deep Learning: learning a hierarchy of internal representations
- ➋ From low-level features to mid-level invariant representations, to object identities

Do we really need deep architectures?

- ➊ We can approximate any function as close as we want with shallow architecture. Why would we need deep ones?

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

- ▶ kernel machines and 2-layer neural net are “universal”.

- ➋ Deep learning machines

$$y = F(W^K \cdot F(W^{K-1} \cdot F(\dots F(W^0 \cdot X) \dots)))$$

- ➌ 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”

- ➍ We need an efficient parameterization of the class of functions that we need to build intelligent machines (the “AI-set”)

Why would Deep Architectures be more efficient?

➊ A deep architecture trades space for time

- ▶ more layers (more sequential computation),
- ▶ but less hardware (less parallel computation).
- ▶ Depth-Breadth tradeoff

➋ Example1: N-bit parity

- ▶ requires $N-1$ XOR gates in a tree of depth $\log(N)$.
- ▶ requires an exponential number of gates if we restrict ourselves to 2 layers (DNF formula with exponential number of minterms).

➌ Example2: circuit for addition of 2 N-bit binary numbers

- ▶ Requires $O(N)$ gates, and $O(N)$ layers using N one-bit adders with ripple carry propagation.
- ▶ Requires lots of gates (some polynomial in N) if we restrict ourselves to two layers (e.g. Disjunctive Normal Form).
- ▶ Bad news: almost all boolean functions have a DNF formula with an exponential number of minterms $O(2^N)$

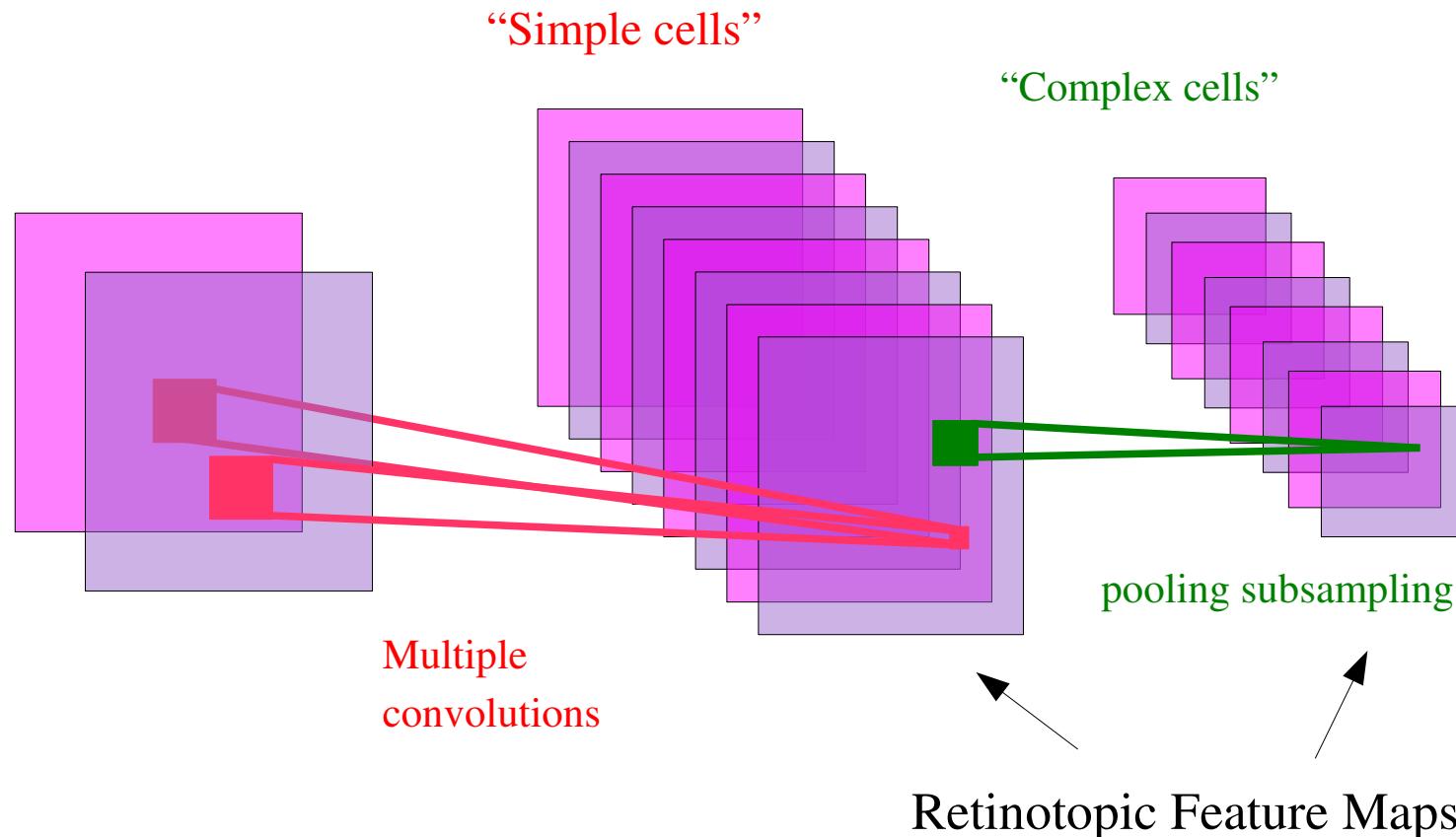
Strategies (after Hinton 2007)

- ➊ **Defeatism:** since no good parameterization of the AI-set is available, let's parameterize a much smaller set for each specific task through careful engineering (preprocessing, kernel....).
- ➋ **Denial:** kernel machines can approximate anything we want, and the VC-bounds guarantee generalization. Why would we need anything else?
 - ▶ unfortunately, kernel machines with common kernels can only represent a tiny subset of functions efficiently
- ➌ **Optimism:** Let's look for learning models that can be applied to the largest possible subset of the AI-set, while requiring the smallest amount of task-specific knowledge for each task.
 - ▶ There is a parameterization of the AI-set with neurons.
 - ▶ Is there an efficient parameterization of the AI-set with computer technology?
- ➍ Today, the ML community oscillates between defeatism and denial.

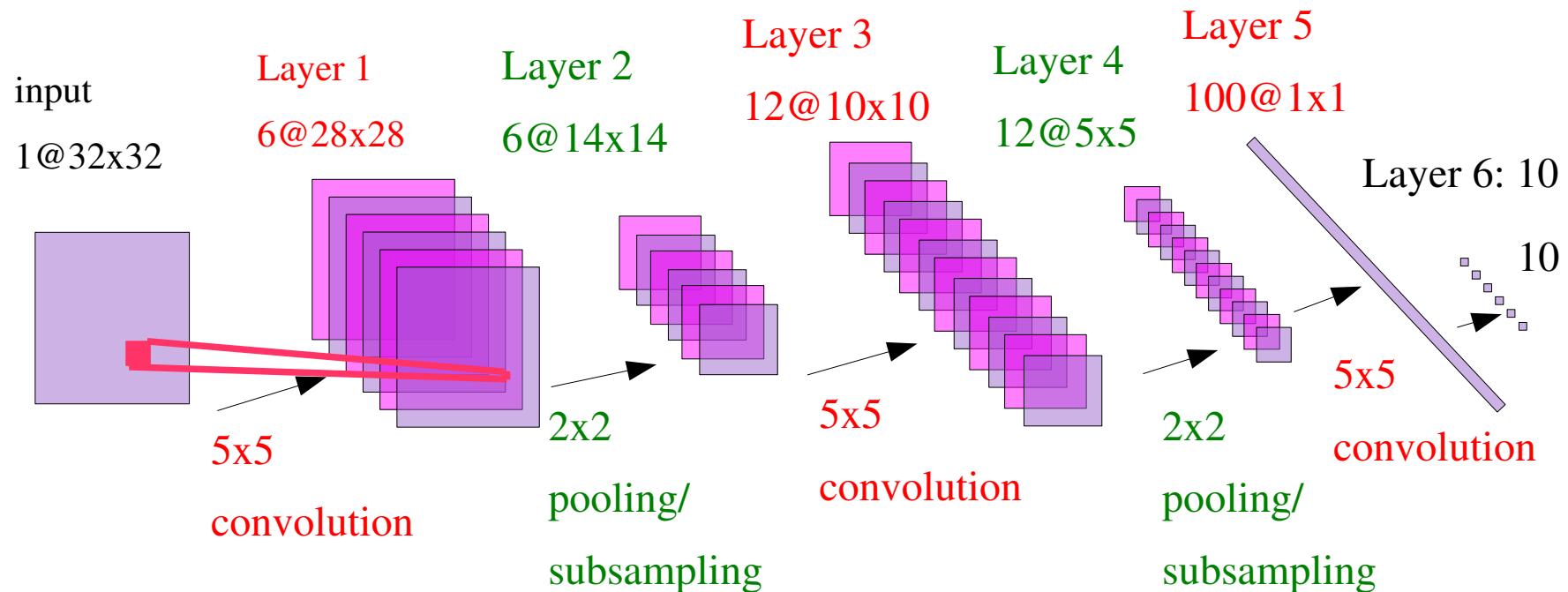
Supervised Learning of a Deep Feature Hierarchy

- [Hubel & Wiesel 1962]:

- ▶ simple cells detect local features
- ▶ complex cells “pool” the outputs of simple cells within a retinotopic neighborhood.



Convolutional Net Architecture



- ➊ **Convolutional net for handwriting recognition** (400,000 synapses)
- ➋ **Convolutional layers** (simple cells): all units in a feature plane share the same weights
- ➌ **Pooling/subsampling layers** (complex cells): for invariance to small distortions.
- ➍ **Supervised gradient-descent learning using back-propagation**
- ➎ **The entire network is trained end-to-end. All the layers are trained simultaneously.**

Deep Architectures for Vision: Convolutional Network

Building a complete artificial vision system:

- ▶ Stack multiple stages of simple cells / complex cells layers
 - ▶ Higher stages compute more global, more invariant features
 - ▶ Stick a classification layer on top

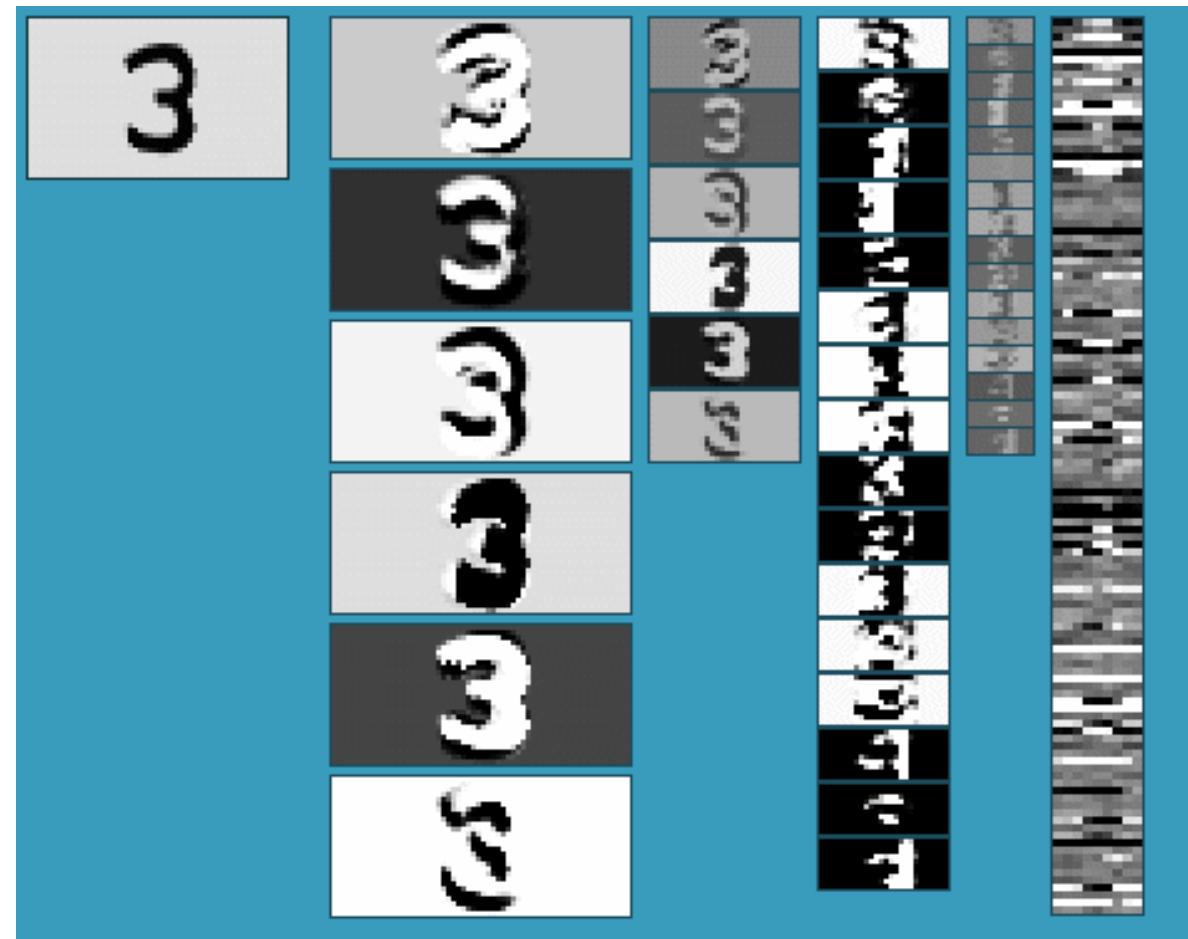
[Fukushima 1971-1982]
• neocognitron

[LeCun 1988-2007]
• convolutional net

[Poggio 2002-2006]
• HMAX

[Ullman 2002-2006]
• fragment hierarchy

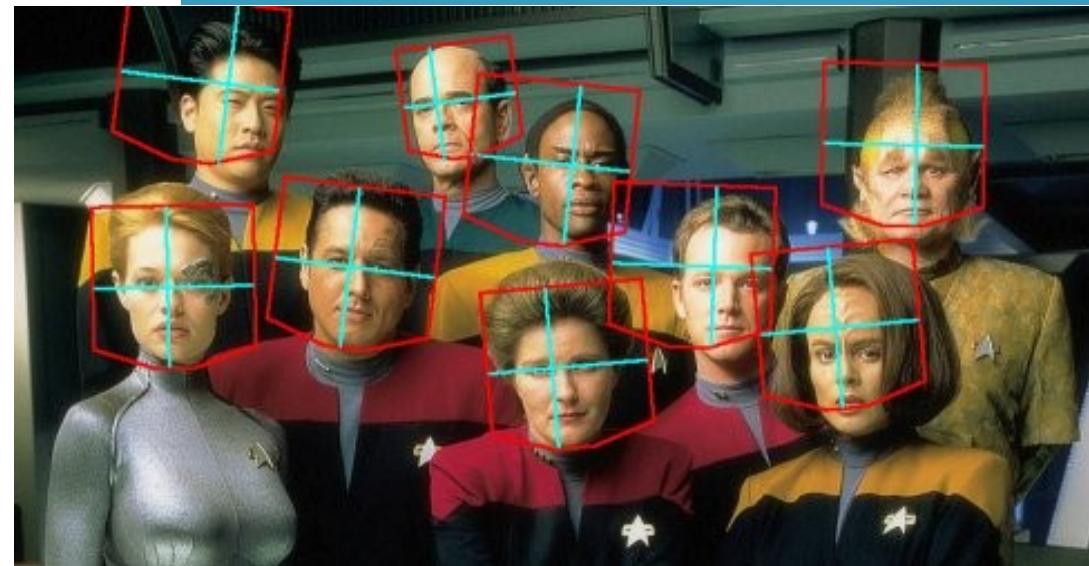
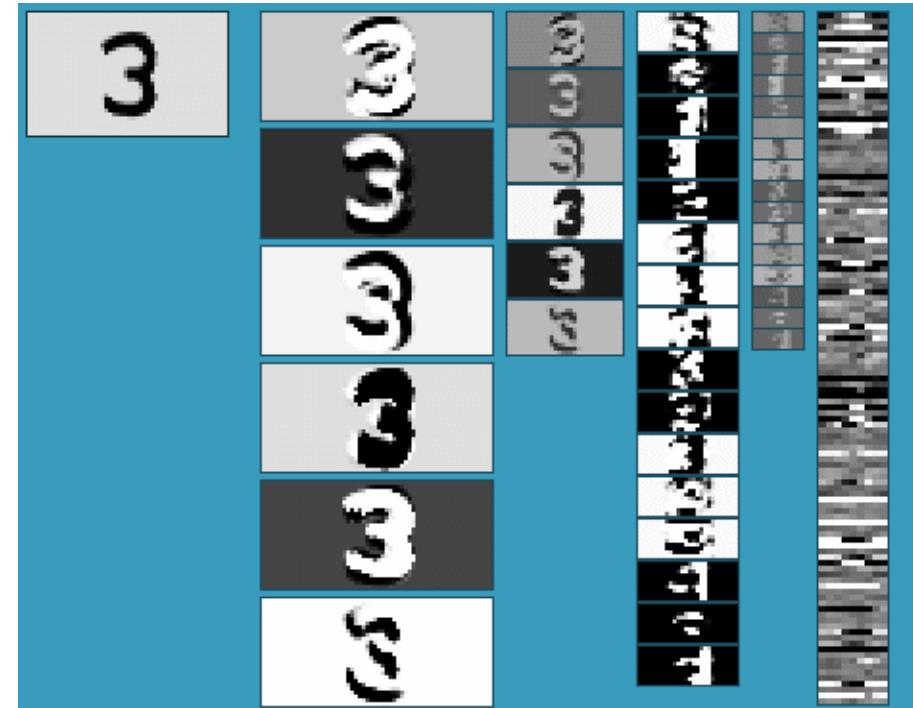
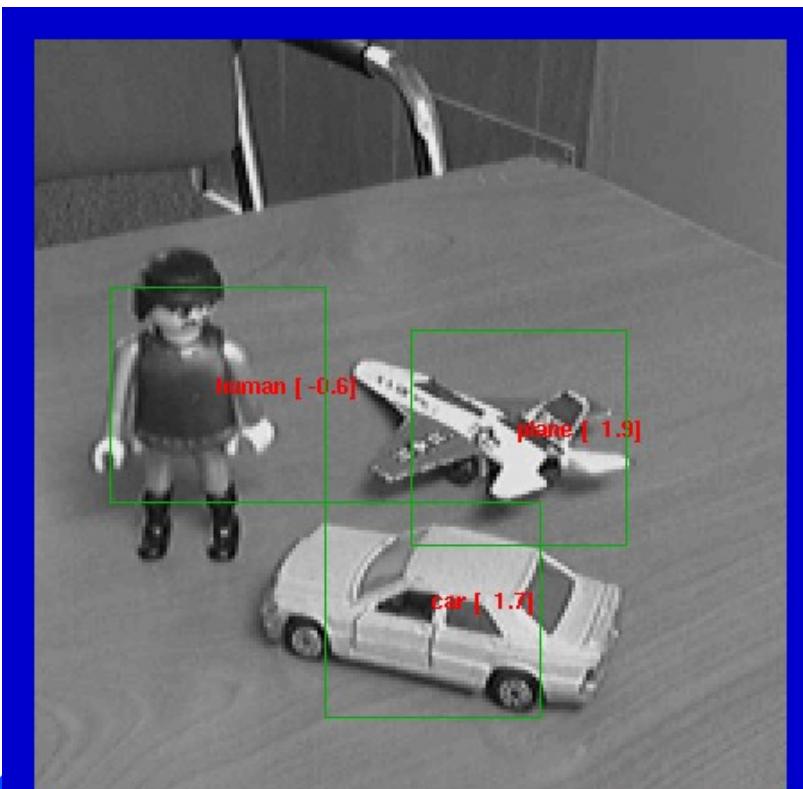
[Lowe 2006]
• HMAX



Supervised Convolutional Nets learn well with lots of data

- Supervised Convolutional nets work very well for:

- ▶ handwriting recognition(winner on MNIST)
- ▶ face detection
- ▶ object recognition with few classes and lots of training samples

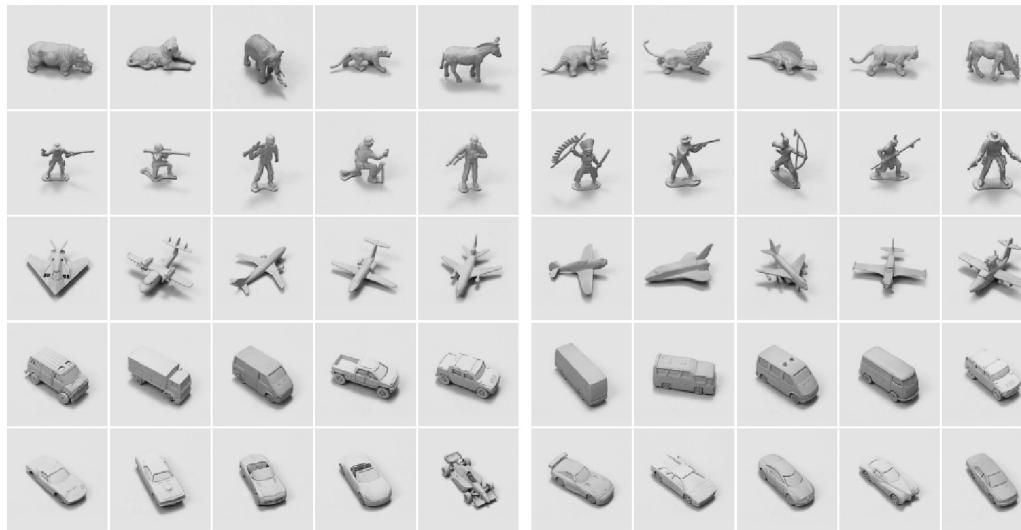


Learning Deep Feature Hierarchies

- ➊ The visual system is deep, and is learned
 - ▶ How do we learn deep hierarchies of invariant features?
- ➋ On recognition tasks with lots of training samples, deep supervised architecture outperform shallow architectures in speed and accuracy
- ➌ Handwriting Recognition:
 - ▶ raw MNIST: 0.62% for convolutional nets [Ranzato 07]
 - ▶ raw MNIST: 1.40% for SVMs [Cortes 92]
 - ▶ distorted MNIST: 0.40% for conv nets [Simard 03, Ranzato 06]
 - ▶ distorted MNIST: 0.67% for SVMs [Bordes 07]
- ➍ Object Recognition
 - ▶ small NORB: 6.0% for conv nets [Huang 05]
 - ▶ small NORB: 11.6% for SVM [Huang 05]
 - ▶ big NORB: 7.8% for conv nets [Huang 06]
 - ▶ big NORB: 43.3% for SVM [Huang 06]

Normalized-Uniform Set: Error Rates

- ➊ Linear Classifier on raw stereo images: **30.2% error.**
- ➋ K-Nearest-Neighbors on raw stereo images: **18.4% error.**
- ➌ K-Nearest-Neighbors on PCA-95: **16.6% error.**
- ➍ Pairwise SVM on 96x96 stereo images: **11.6% error**
- ➎ Pairwise SVM on 95 Principal Components: **13.3% error.**
- ➏ Convolutional Net on 96x96 stereo images: **5.8% error.**



Training instances Test instances

Normalized-Uniform Set: Learning Times

	SVM	Conv Net				SVM/Conv
test error	11.6%	10.4%	6.2%	5.8%	6.2%	5.9%
train time (min*GHz)	480	64	384	640	3,200	50+
test time per sample (sec*GHz)	0.95			0.03		0.04+
#SV	28%					28%
parameters	$\sigma=2,000$ $C=40$					dim=80 $\sigma=5$ $C=0.01$



SVM: using a parallel implementation by
Graf, Durdanovic, and Cosatto (NEC Labs)

Chop off the
last layer of the
convolutional net
and train an SVM on it

Experiment 2: Jittered-Cluttered Dataset



- ➊ **291,600 training samples, 58,320 test samples**
- ➋ **SVM with Gaussian kernel** **43.3% error**
- ⌂ **Convolutional Net with binocular input:** **7.8% error**
- ⌃ **Convolutional Net + SVM on top:** **5.9% error**
- ⌄ **Convolutional Net with monocular input:** **20.8% error**
- ⌅ **Smaller mono net (DEMO):** **26.0% error**
- ⌆ **Dataset available from <http://www.cs.nyu.edu/~yann>**

Jittered-Cluttered Dataset

	SVM	Conv Net		SVM/Conv	
test error	43.3%	16.38%	7.5%	7.2%	5.9%
train time (min*GHz)	10,944	420	2,100	5,880	330+
test time per sample (sec*GHz)	2.2	0.04			0.06+
#SV	5%				2%
parameters	$\sigma=10^4$ $C=40$				dim=100 $\sigma=5$ $C=1$

OUCH!

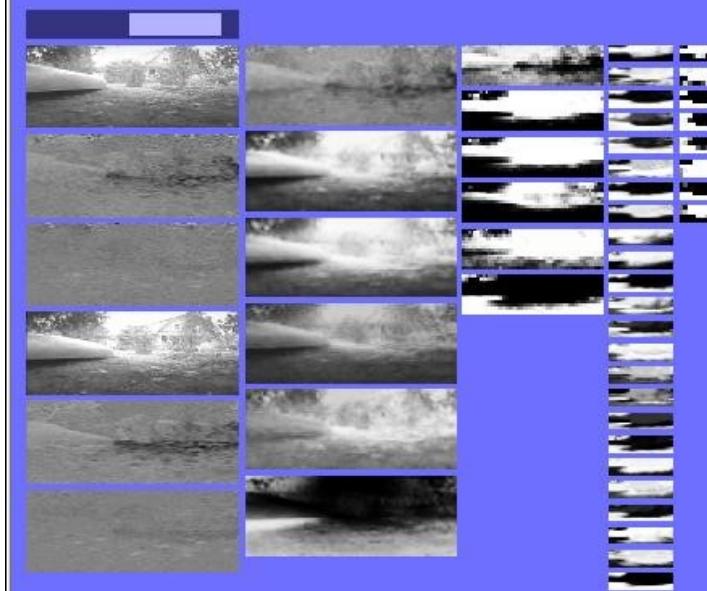
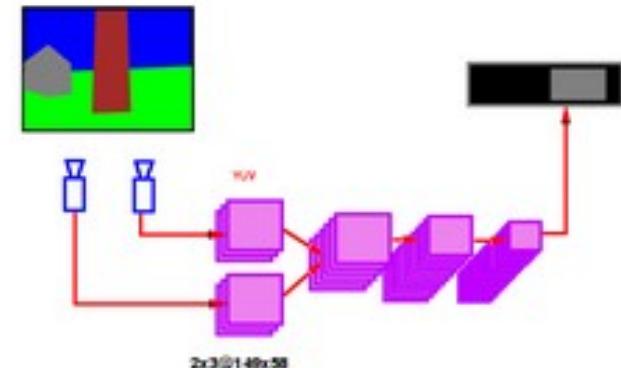
The convex loss, VC bounds
and representer theorems
don't seem to help

Chop off the last layer,
and train an SVM on it
it works!

Visual Navigation for a Mobile Robot

[LeCun et al. NIPS 2005]

- Mobile robot with two cameras
- The convolutional net is trained to emulate a human driver from recorded sequences of video + human-provided steering angles.
- The network maps stereo images to steering angles for obstacle avoidance

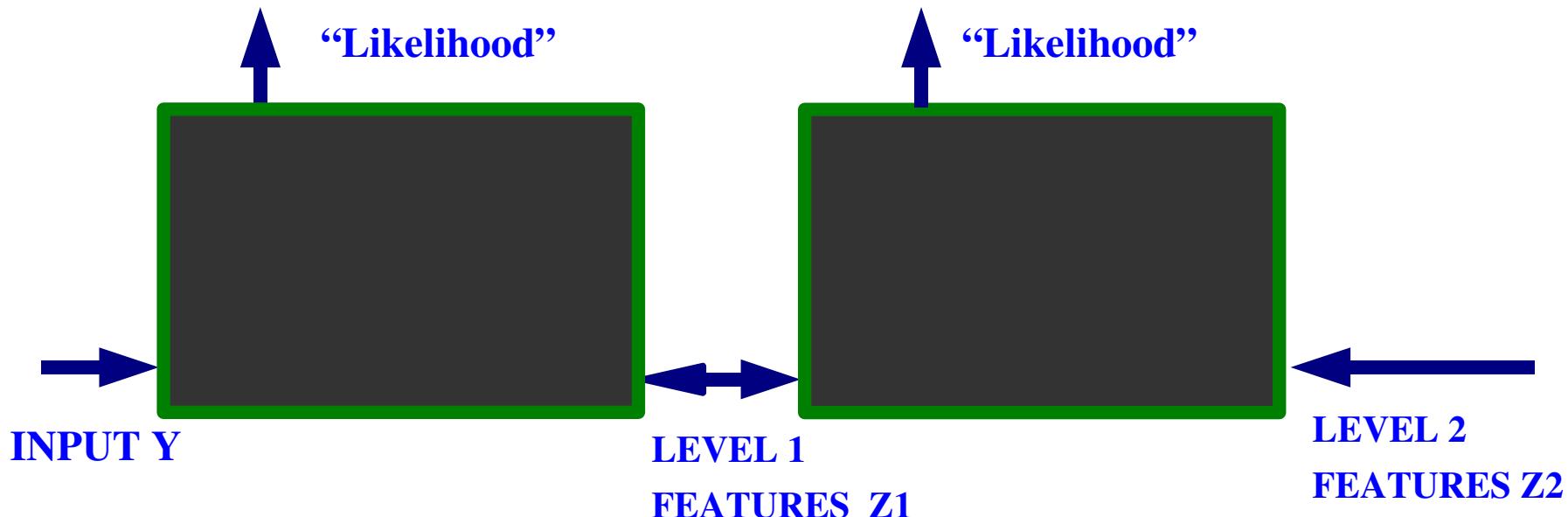


Learning Deep Feature Hierarchies (unsupervised)

- ➊ On recognition tasks **with few labeled samples**, deep supervised architectures don't do so well
 - ▶ a purely supervised convolutional net gets only 20% correct on Caltech-101 with 30 training samples/class
- ➋ We need **unsupervised learning methods** that can learn invariant feature hierarchies
- ➌ This talk will present methods to learn hierarchies of **sparse and invariant features**
- ➍ Sparse features are good for two reasons:
 - ▶ they are easier to deal with for a classifier
 - ▶ we will show that using sparsity constraints is a way to upper bound the partition function.

The Basic Idea for Training Deep Feature Hierarchies

- Stage k measures the “compatibility” between features at level $k-1$ (Z_{k-1}) and features at level k (Z_k).
 - compatibility = $-\log$ likelihood = energy = $E(Z_{k-1}, Z_k, W_k)$
- Inference: Find the Z 's that minimize the total energy.
- The stages are trained one after the other
 - the input to stage $k+1$ is the feature vector of stage k .



Unsupervised Feature Learning as Density Estimation

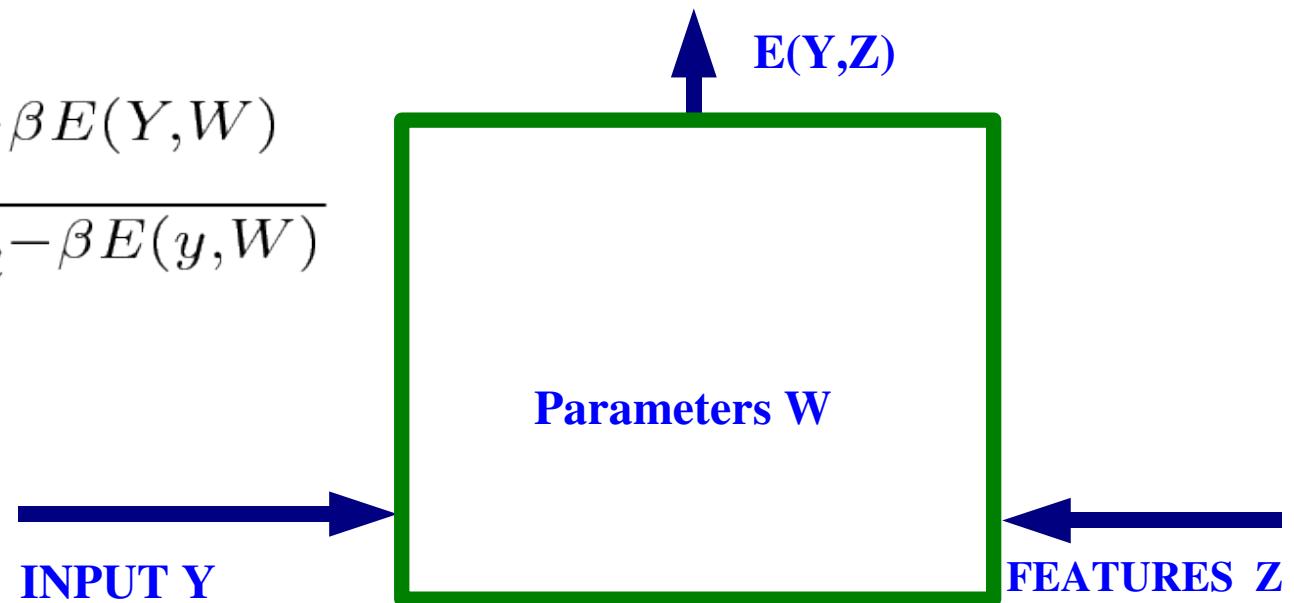
Energy function: $E(Y,W) = \text{MIN}_Z E(Y,Z,W)$

- ▶ Y: input
- ▶ Z: “feature” vector, representation, latent variables
- ▶ W: parameters of the model (to be learned)
- ▶ Maximum A Posteriori approximation for Z

Density function $P(Y|W)$

- ▶ Learn W so as to maximize the likelihood of the training data under the model

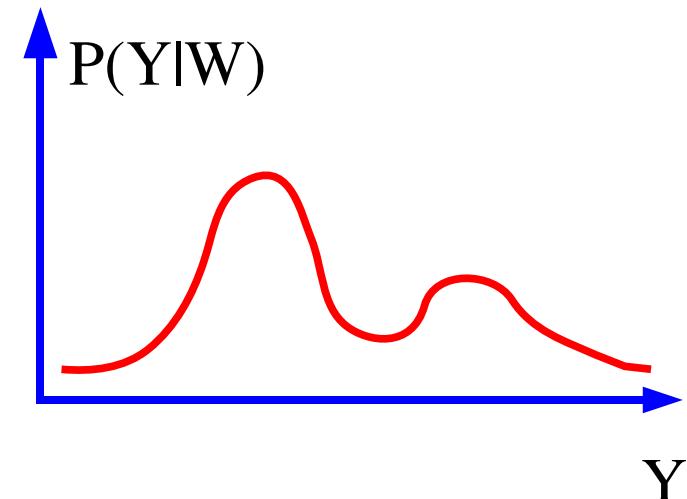
$$P(Y|W) = \frac{e^{-\beta E(Y,W)}}{\int_y e^{-\beta E(y,W)}}$$



What is Unsupervised Learning?

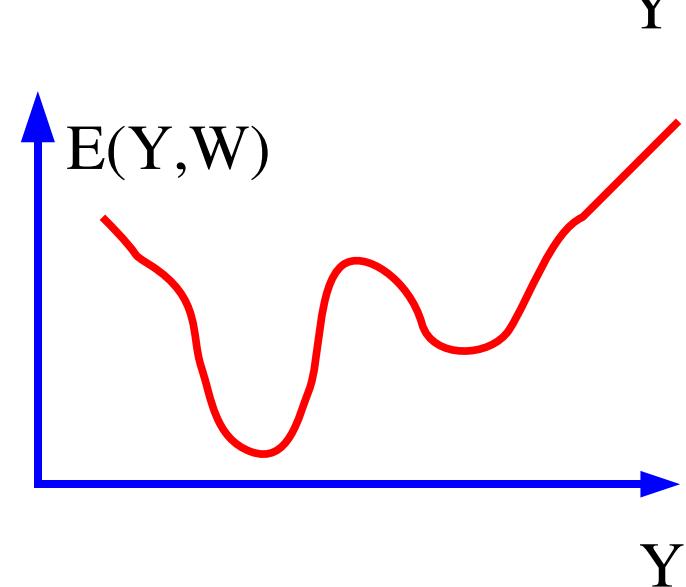
Probabilistic View:

- ▶ Produce a probability density function that:
- ▶ has high value in regions of high sample density
- ▶ has low value everywhere else (integral = 1).



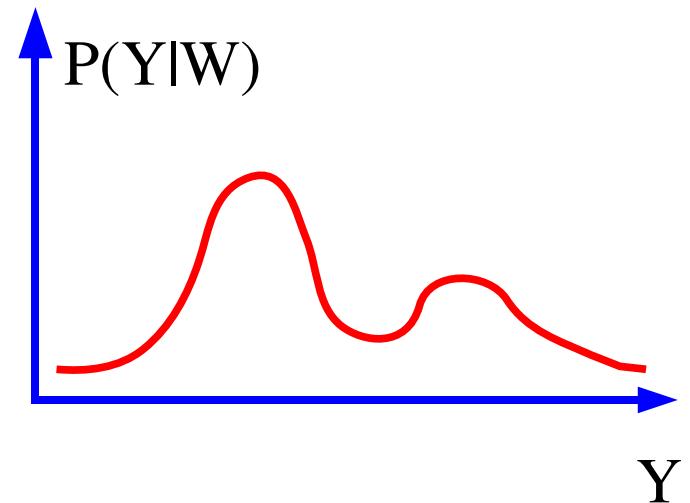
Energy-Based View:

- ▶ produce an energy function $E(Y, W)$ that:
- ▶ has low value in regions of high sample density
- ▶ has high(er) value everywhere else

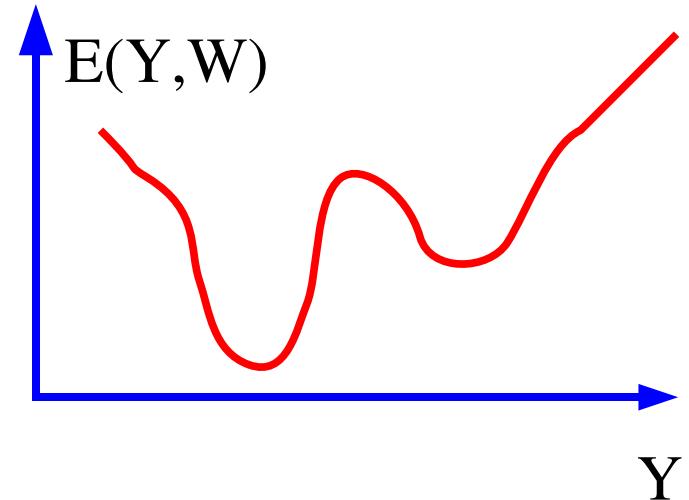


What is Unsupervised Learning?

$$P(Y|W) = \frac{e^{-\beta E(Y,W)}}{\int_y e^{-\beta E(y,W)}}$$



$$E(Y, W) \propto -\log P(Y|W)$$



Training a Probabilistic Unsupervised Model

Maximizing $P(Y|W)$ on training samples

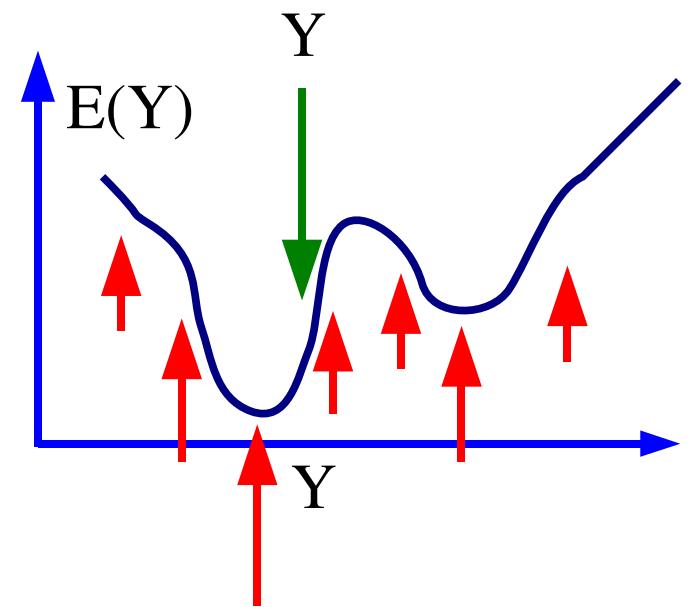
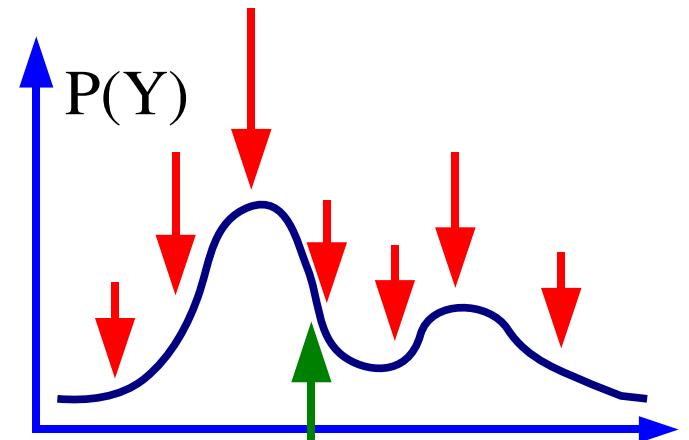
$$P(Y|W) = \frac{e^{-\beta E(Y,W)}}{\int_y e^{-\beta E(y,W)}}$$

make this big
make this small

Minimizing $-\log P(Y,W)$ on training samples

$$L(Y, W) = E(Y, W) + \frac{1}{\beta} \log \int_y e^{-\beta E(y,W)}$$

make this small
make this big



Training a Probabilistic Unsupervised Model

- Gradient of the negative log-likelihood loss for one sample Y :

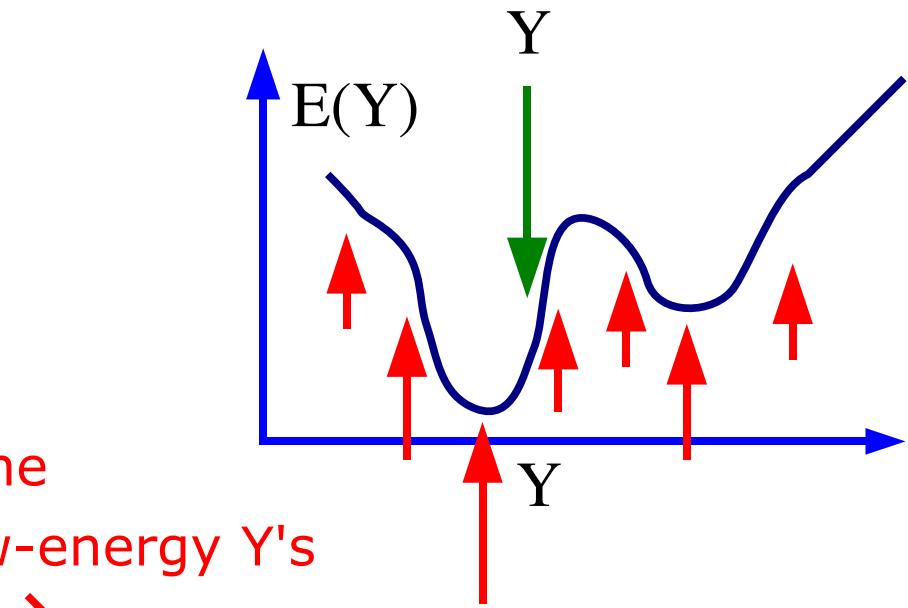
$$\frac{\partial L(Y, W)}{\partial W} = \frac{\partial E(Y, W)}{\partial W} - \int_y P(y|W) \frac{\partial E(y, W)}{\partial W}$$

- Gradient descent:

$$W \leftarrow W - \eta \frac{\partial L(Y, W)}{\partial W}$$

Pushes down on the
energy of the samples

Pulls up on the
energy of low-energy Y 's



$$W \leftarrow W - \eta \frac{\partial E(Y, W)}{\partial W} + \eta \int_y P(y|W) \frac{\partial E(y, W)}{\partial W}$$

The Normalization problem

➊ The “Intractable Partition Function Problem” a.k.a. the Normalization Problem

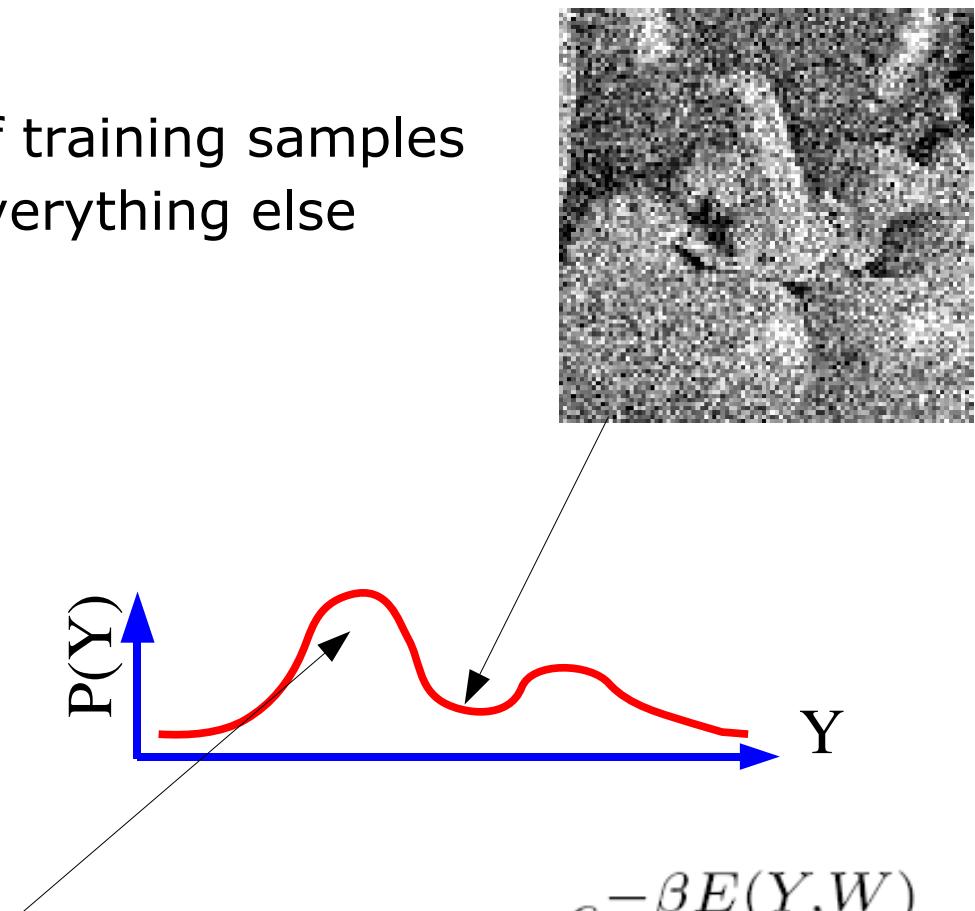
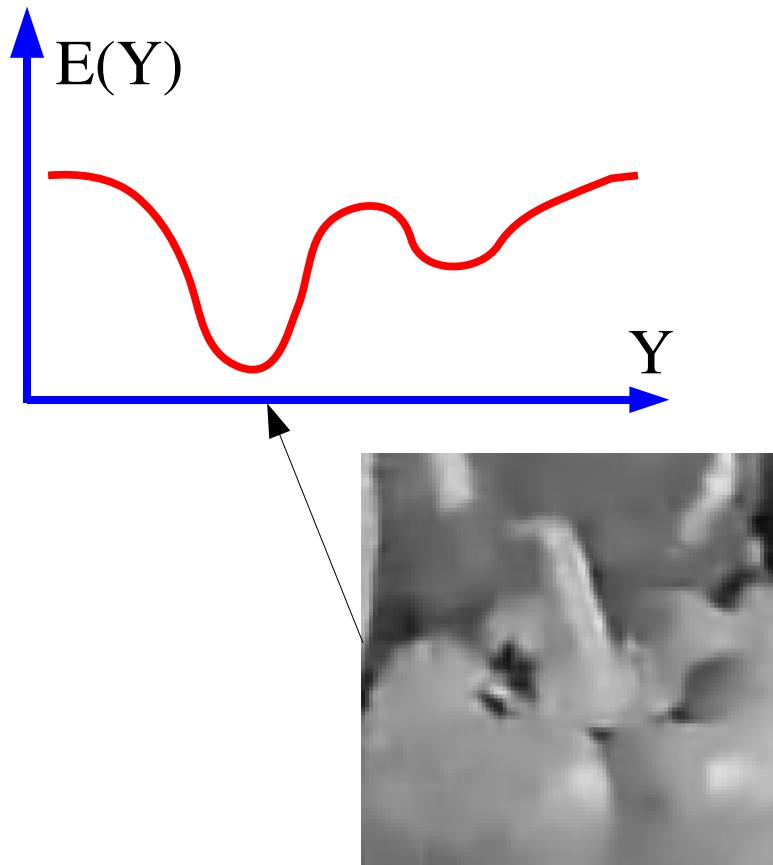
- ▶ Give high probability (or low energy) to training samples
- ▶ Give low probability (or high energy) to everything else
- ▶ There are too many “everything else”!
- ▶ The normalization constant of probabilistic models is a sum over too many terms.
- ▶ **Making the energies of everthing else large is very hard**

The Intractable Partition Function Problem

Example: Image Denoising

Learning:

- ▶ push down on the energy of training samples
- ▶ push up on the energy of everything else

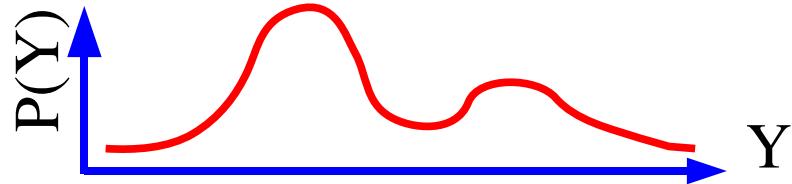


$$P(Y, W) = \frac{e^{-\beta E(Y, W)}}{\int_y e^{-\beta E(y, W)}}$$

The Two Biggest Challenges in Machine Learning

➊ 1. The “Intractable Partition Function Problem”

- ▶ Complicated probability distributions in high dimensional spaces are difficult to normalize



$$P(Y, W) = \frac{e^{-\beta E(Y, W)}}{\int_y e^{-\beta E(y, W)}}$$

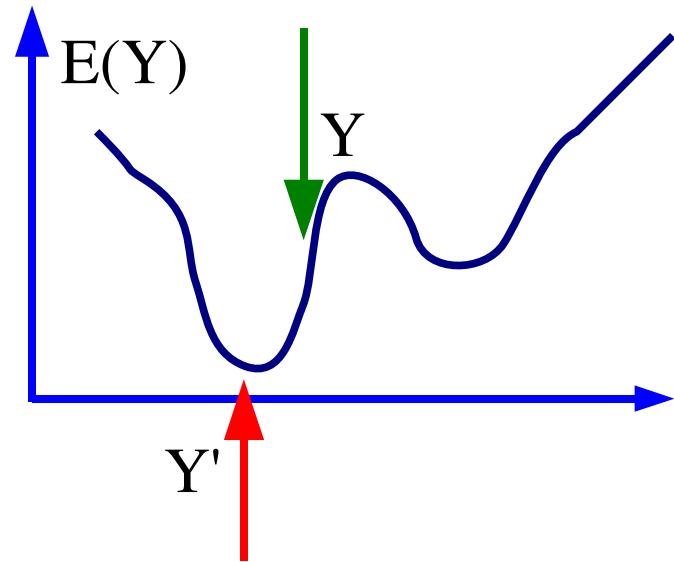
- ▶ Example: what is the PDF of natural images?
- ▶ Question: how do we get around this problem?

➋ 2. The “Deep Learning Problem”

- ▶ Complex tasks (vision, audition, natural language understanding....) require appropriate internal representations.
- ▶ With most current approaches to learning, the internal representation (and the similarity metric on it) are assumed to be given (or hand-engineered).
- ▶ Question: how do we learn internal representations?

Contrastive Divergence Trick [Hinton 2000]

- push down on the energy of the training sample \mathbf{Y}
- Pick a sample of low energy \mathbf{Y}' near the training sample, and pull up its energy
 - this digs a trench in the energy surface around the training samples



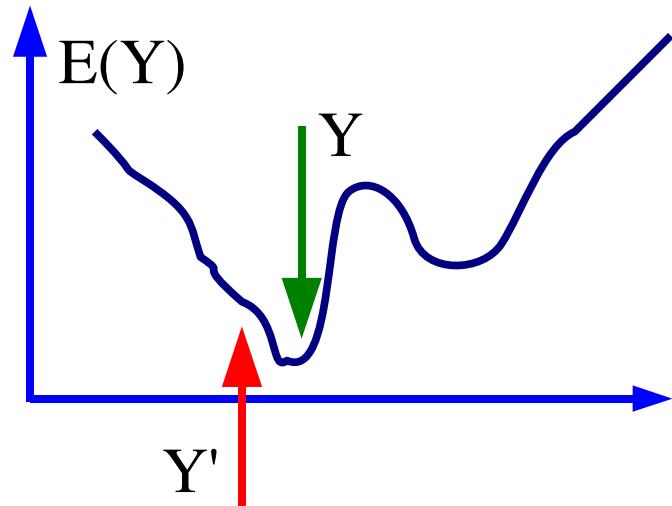
$$W \leftarrow W - \eta \frac{\partial E(Y, W)}{\partial W} + \eta \frac{\partial E(Y', W)}{\partial W}$$

Pushes down on the energy
of the training sample \mathbf{Y}

Pulls up on the energy \mathbf{Y}'

Contrastive Divergence Trick [Hinton 2000]

- push down on the energy of the training sample \mathbf{Y}
- Pick a sample of low energy \mathbf{Y}' near the training sample, and pull up its energy
 - this digs a trench in the energy surface around the training samples



$$W \leftarrow W - \eta \frac{\partial E(Y, W)}{\partial W} + \eta \frac{\partial E(Y', W)}{\partial W}$$

Pushes down on the energy
of the training sample \mathbf{Y}

Pulls up on the energy \mathbf{Y}'

Problems in CD in High Dimension

- If the energy surface is highly flexible and high-dimensional, there are simply too many points whose energy needs to be pulled up.
- It becomes very difficult to make it non-flat.
- We need a more “wholesale” way of making the energy surface non-flat.

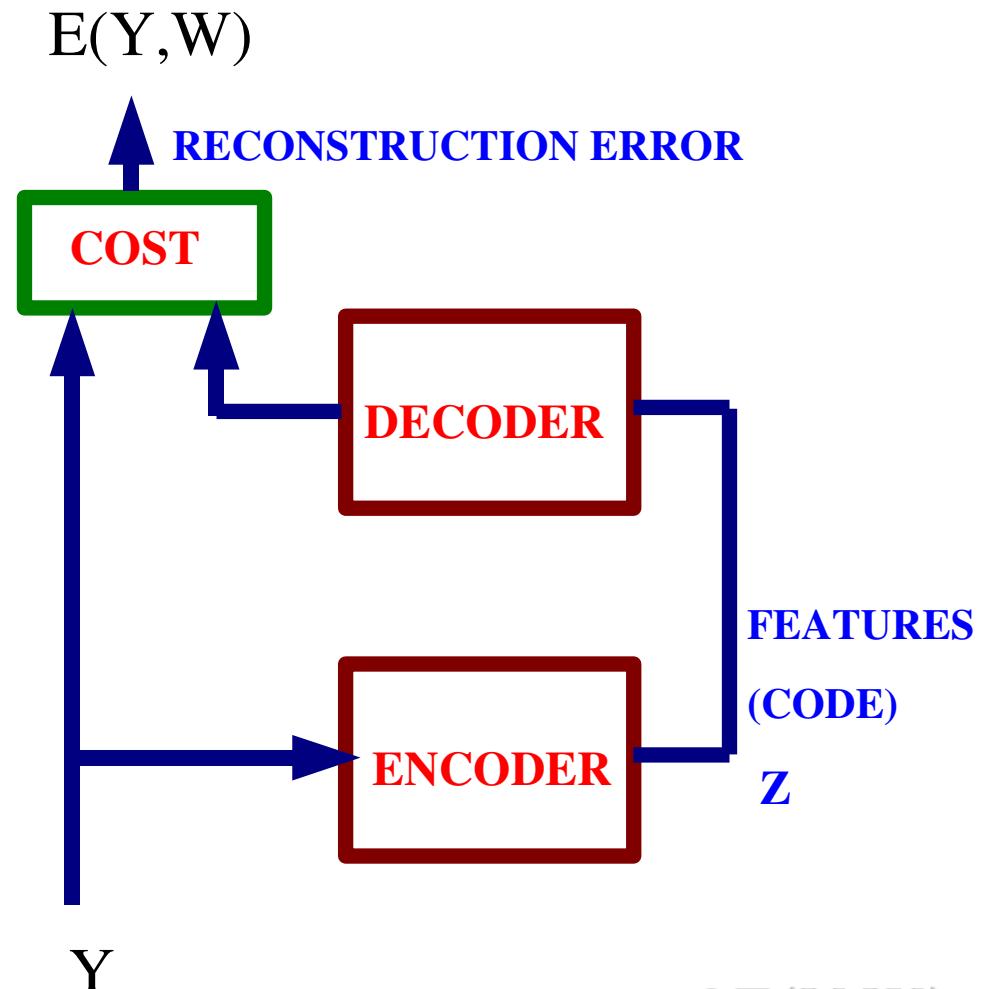
$$W \leftarrow W - \eta \frac{\partial E(Y, W)}{\partial W} + \eta \frac{\partial E(Y', W)}{\partial W}$$

Pushes down on the energy
of the training sample Y

Pulls up on the energy Y'

Unsupervised Feature Learning: Encoder/Decoder Architecture

- Learns a probability density function of the training data
- Generates Features in the process
- The feature space is akin to an embedding of the manifold containing regions of high-density of data.
- Learning Algorithms:
 - ▶ contrastive divergence
 - ▶ constraints on the information content of the features



$$P(Y, W) = \frac{e^{-\beta E(Y, W)}}{\int_y e^{-\beta E(y, W)}}$$

The Deep Encoder/Decoder Architecture

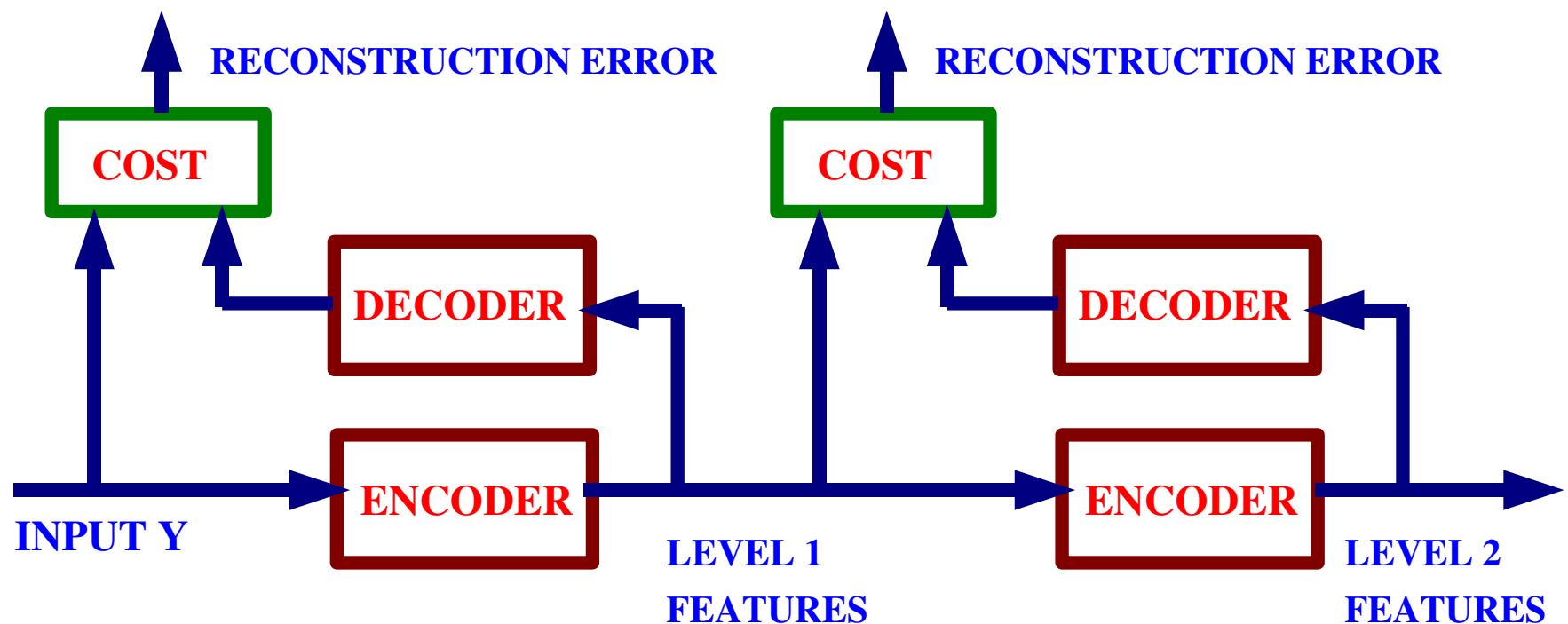
- Each stage is composed of

[Hinton 2005, Bengio 2006, LeCun 2006]

- an encoder that produces a feature vector from the input
- a decoder that reconstruct the input from the feature vector
 - (Restricted Boltzmann Machines are a special case)

- Each stage is trained one after the other

- the input to stage $k+1$ is the feature vector of stage k .



General Encoder/Decoder Architecture

- **Decoder:**

- ▶ Linear

- **Optional encoders of different types:**

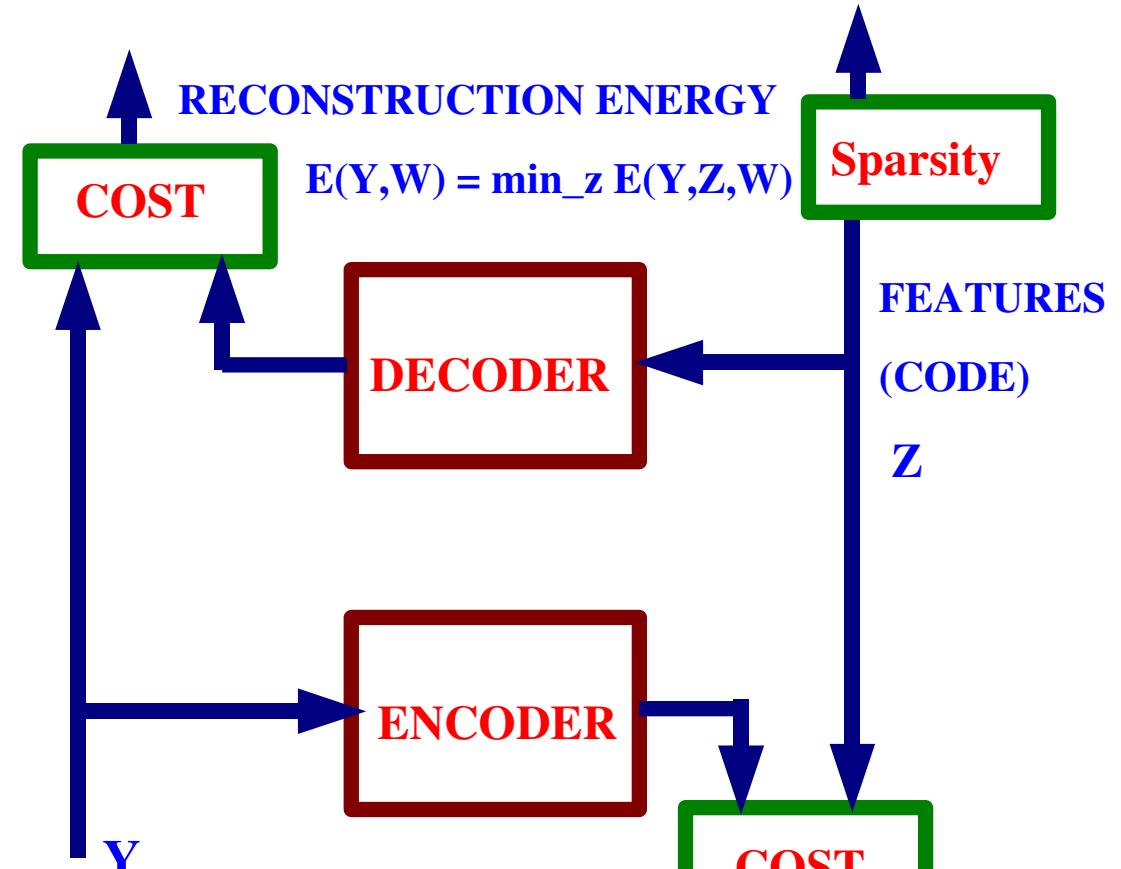
- ▶ None
 - ▶ Linear
 - ▶ Linear-Sigmoid-Scaling
 - ▶ Linear-Sigmoid-Linear

- **Optional sparsity penalty**

- ▶ None, L1, Log Student-T

- **Feature Vector Z**

- ▶ continuous
 - ▶ binary stochastic
 - ▶ discrete (e.g. 1-of-N)



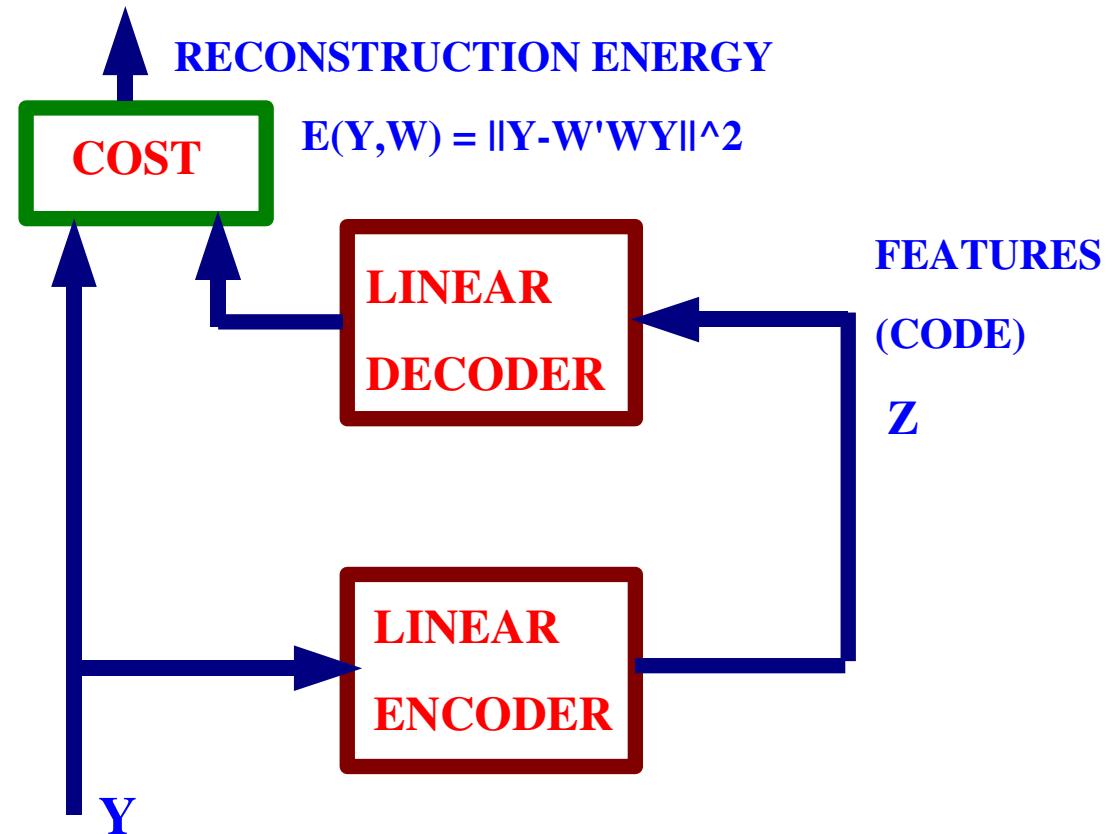
$$\bar{Z}_Y = \operatorname{argmin}_Z E(Y, Z, W)$$

$$E(Y, W) = \min_Z E(Y, Z, W)$$

Encoder/Decoder Architecture: PCA

PCA:

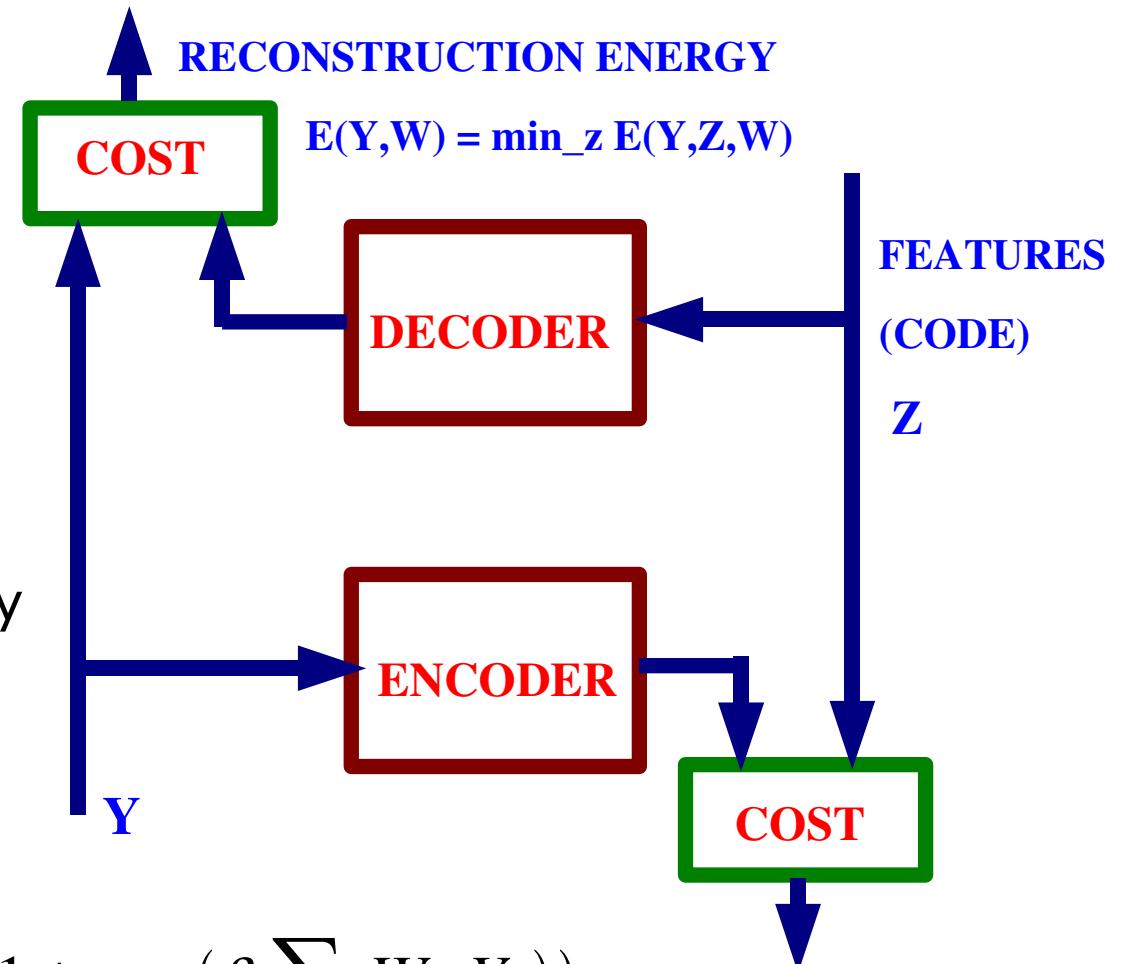
- ▶ linear encoder and decoder
- ▶ no sparsity
- ▶ low-dimensional code Z
- ▶ $E(Y) = \|Y - W'WY\|^2$



Encoder/Decoder Architecture: RBM

• Restricted Boltzmann Machines:

- ▶ [Hinton et al. 2005]
- ▶ symmetric encoder/decoder
- ▶ $E(Y, Z, W) = -Y'WZ$
- ▶ Z : binary stochastic vector
- ▶ Learning: contrastive Divergence
- ▶ It seems that the energy surface becomes non flat because Z is binary and noisy (not just because of contrastive divergence).



• Sampling is expensive

$$P(Z_j=1|Y, W) = 1/(1 + \exp(\beta \sum_i W_{ij} Y_i))$$

$$P(Y_i=1|Z, W) = 1/(1 + \exp(\beta \sum_j W_{ij} Z_j))$$

$$E(Y, W) = -1/\beta \log \sum_z \exp(-\beta E(Y, Z, W))$$

Unsupervised Feature Learning: Linear Decoder Architecture

• K-Means:

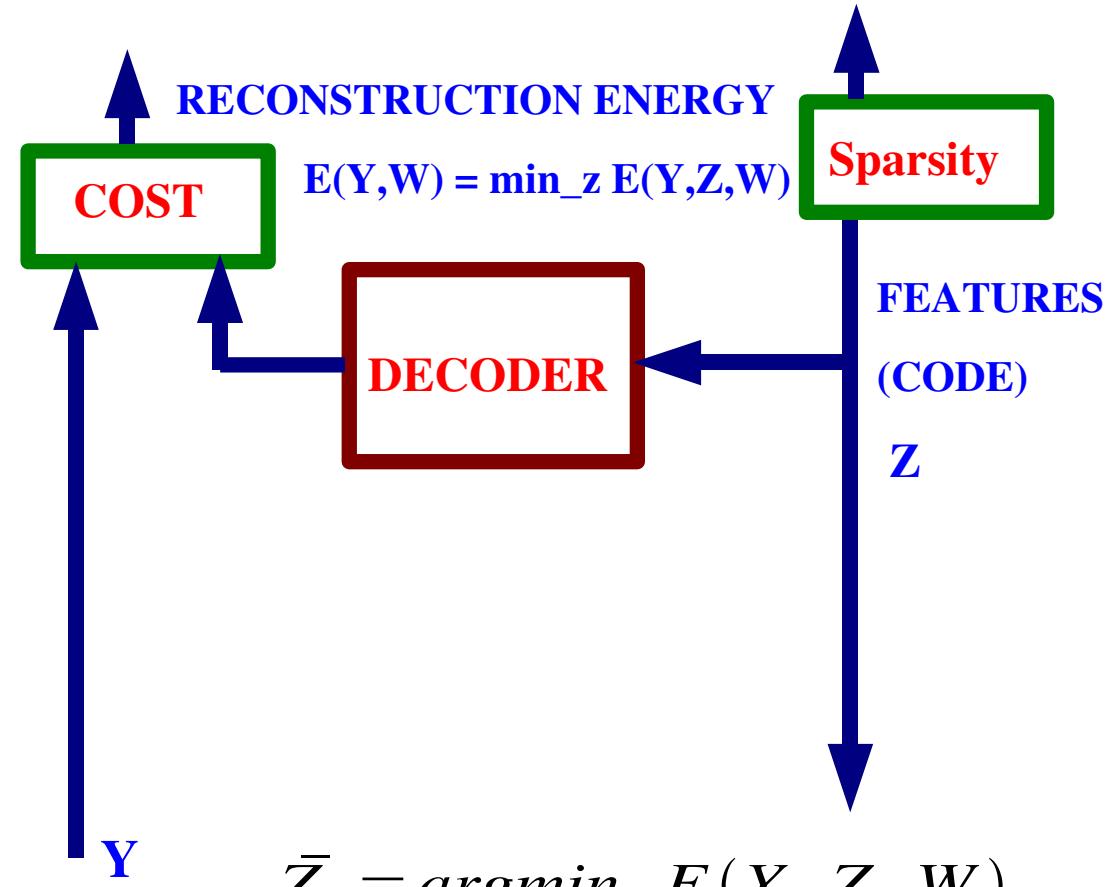
- ▶ no encoder, linear decoder
- ▶ Z is a one-of- N binary code
- ▶ $E(Y) = ||Y-ZW||^2$

• Sparse Overcomplete Bases:

- ▶ [Olshausen & Field]
- ▶ no encoder
- ▶ linear decoder
- ▶ log Student-T sparsity

• Basis Pursuit

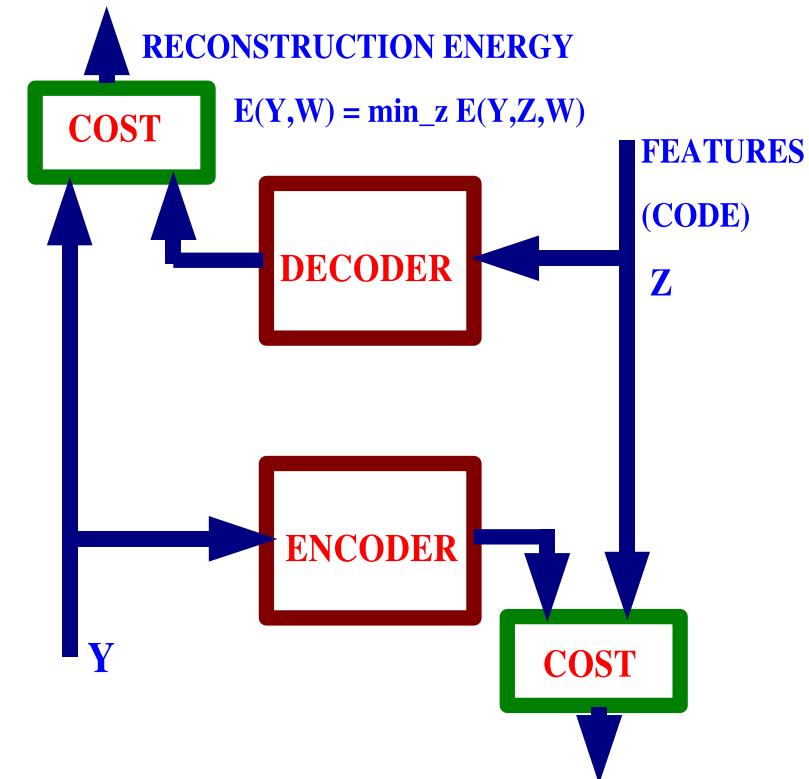
- ▶ [Chen & Donoho]
- ▶ no encoder
- ▶ linear decoder
- ▶ L1 sparsity



• Problem: computing Z from Y involves running a minimization algorithm

The Main New Idea in This Talk!

- ➊ Contrastive divergence doesn't work too well when the dimension of the input is very large (> a few hundred)
 - ▶ because the space of "everything else" is too large
- ➋ We need a more efficient way to ensure that the energy surface takes the right shape
 - ▶ with a groove around the manifold containing the training samples
- ➌ Main Idea: Restrict the information content in the feature vector Z
 - ▶ by making it sparse
 - ▶ by making it low dimensional
 - ▶ by making it binary
 - ▶ by making it noisy



$$\bar{Z}_Y = \operatorname{argmin}_Z E(Y, Z, W)$$

$$E(Y, W) = \min_Z E(Y, Z, W)$$

[Ranzato et al. AI-Stats 2007]

The Main Insight [Ranzato et al. 2007]

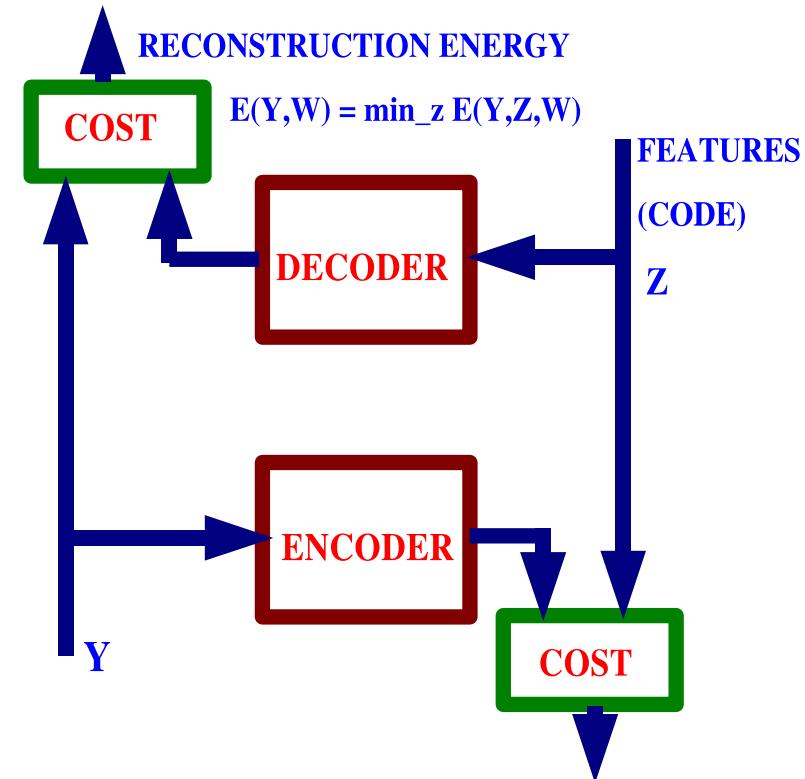
- If the information content of the feature vector is limited (e.g. by imposing sparsity constraints), the energy MUST be large in most of the space.

▶ pulling down on the energy of the training samples will necessarily make a groove

- The volume of the space over which the energy is low is limited by the entropy of the feature vector

▶ Input vectors are reconstructed from feature vectors.

▶ If few feature configurations are possible, few input vectors can be reconstructed properly



$$\bar{Z}_Y = \operatorname{argmin}_Z E(Y, Z, W)$$

$$E(Y, W) = \min_Z E(Y, Z, W)$$

Why sparsity puts an upper bound on the partition function

- Imagine the code has no restriction on it

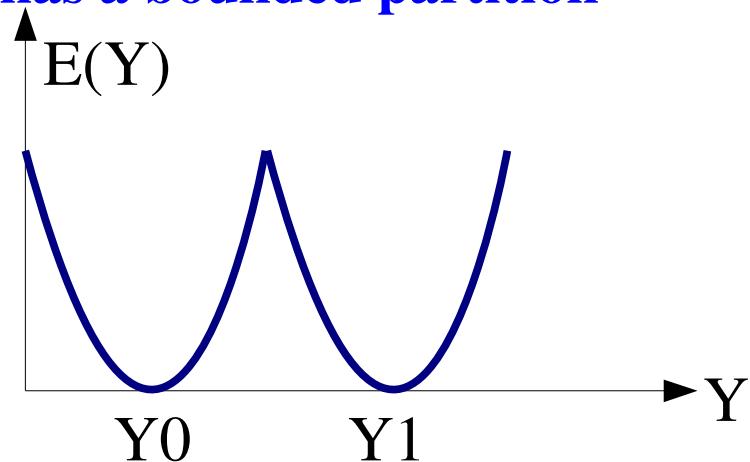
- The energy (or reconstruction error) can be zero everywhere, because every Y can be perfectly reconstructed. The energy is flat, and the partition function is unbounded

- Now imagine that the code is binary ($Z=0$ or $Z=1$), and that the reconstruction cost is quadratic $E(Y) = \|Y - \text{Dec}(Z)\|^2$

- Only two input vectors can be perfectly reconstructed:
 - $Y_0 = \text{Dec}(0)$ and $Y_1 = \text{Dec}(1)$.
 - All other vectors have a higher reconstruction error

- The corresponding probabilistic model has a bounded partition function:

$$P(Y) = \frac{e^{-E(Y)}}{\int_y e^{-E(y)}}$$

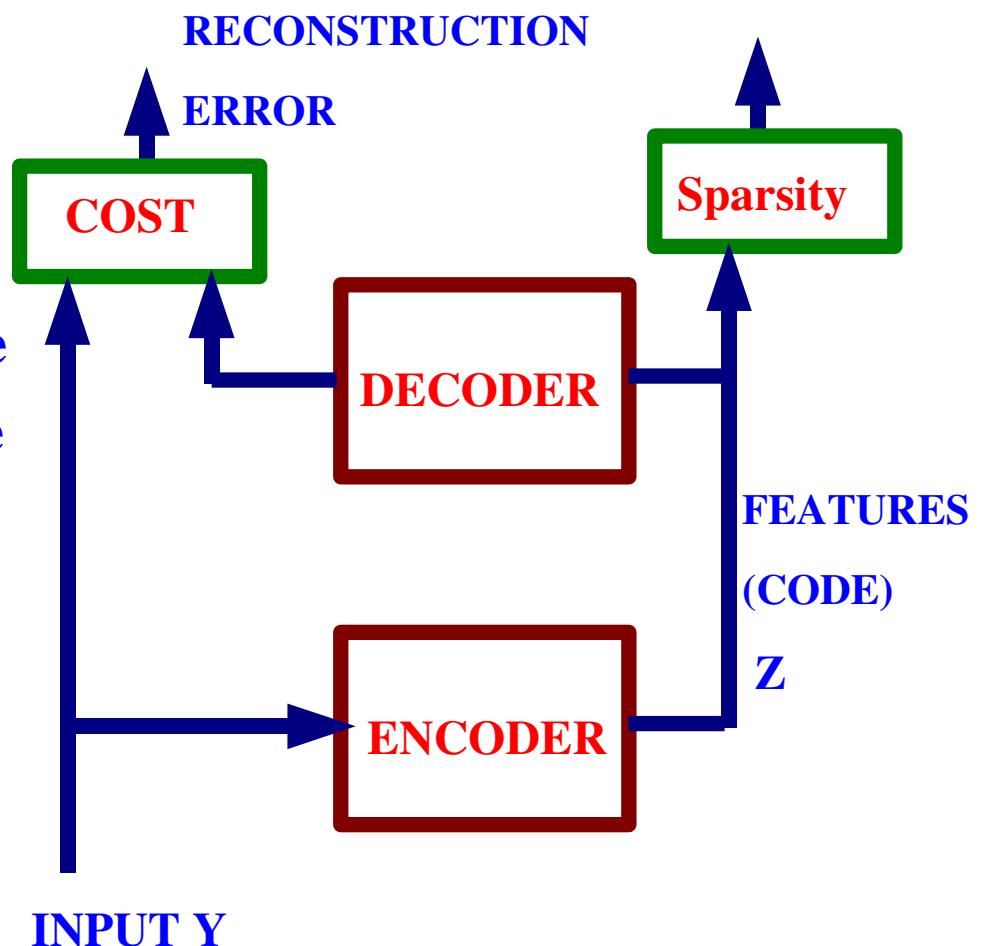


Restricting the Information Content of the Code

- ➊ Restricting the information content of the code alleviates the need to push up of the energy of everything.
- ➋ Hence, we can happily use a simple loss function that simply pulls down on the energy of the training samples.
- ➌ We do not need a contrastive term that pulls up on the energy everywhere else.

Learning Sparse Features

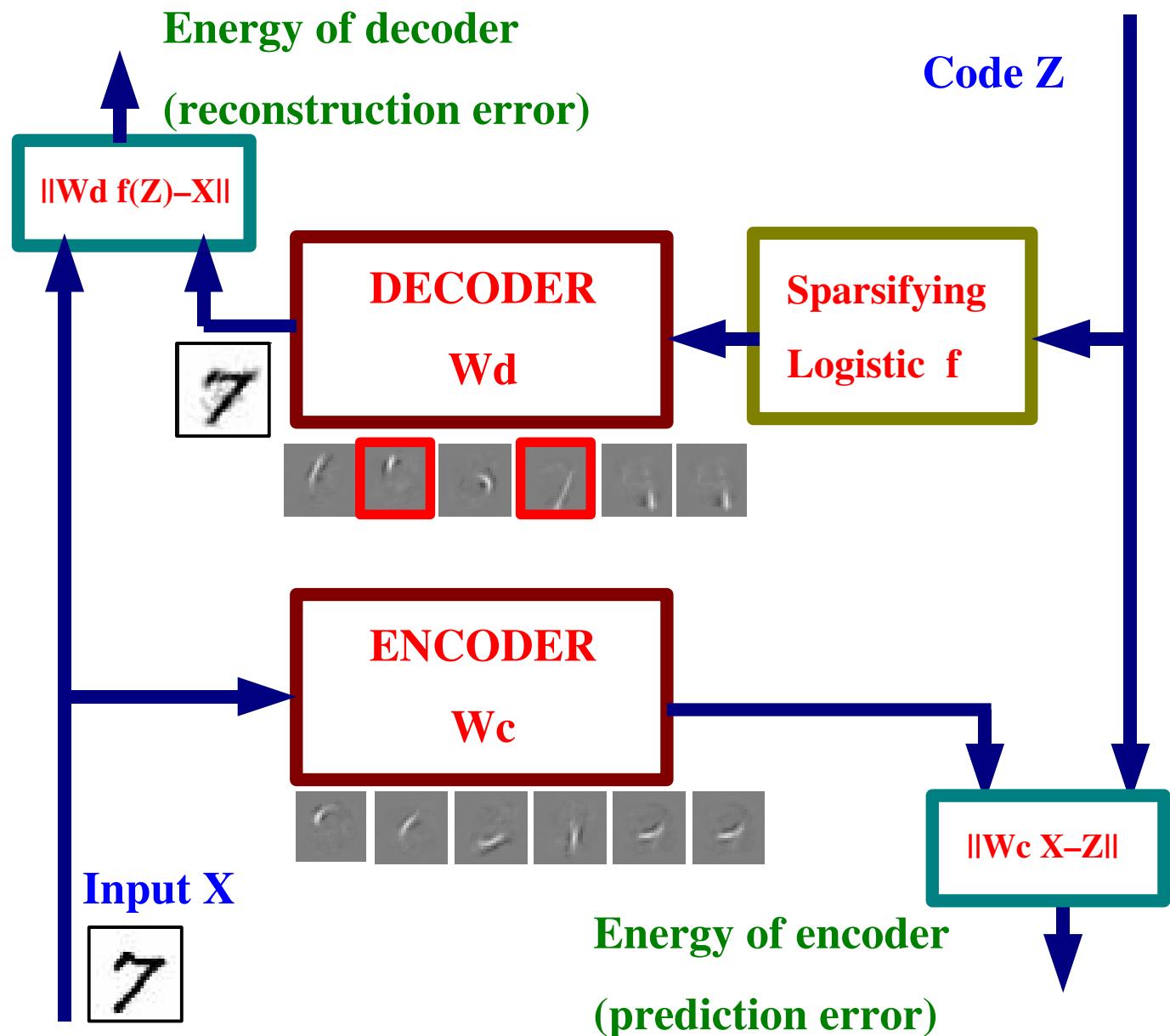
- If the feature vector is larger than the input, the system can learn the identity function in a trivial way
- To prevent this, we force the feature vector to be sparse
- By sparsifying the feature vector, we limit its information content, and we prevent system from being able to reconstruct everything perfectly
 - ▶ technically, code sparsification puts an upper bound on the partition function of $P(Y)$ under the model [Ranzato AI-Stats 07]



Sparsifying with a high-threshold logistic function

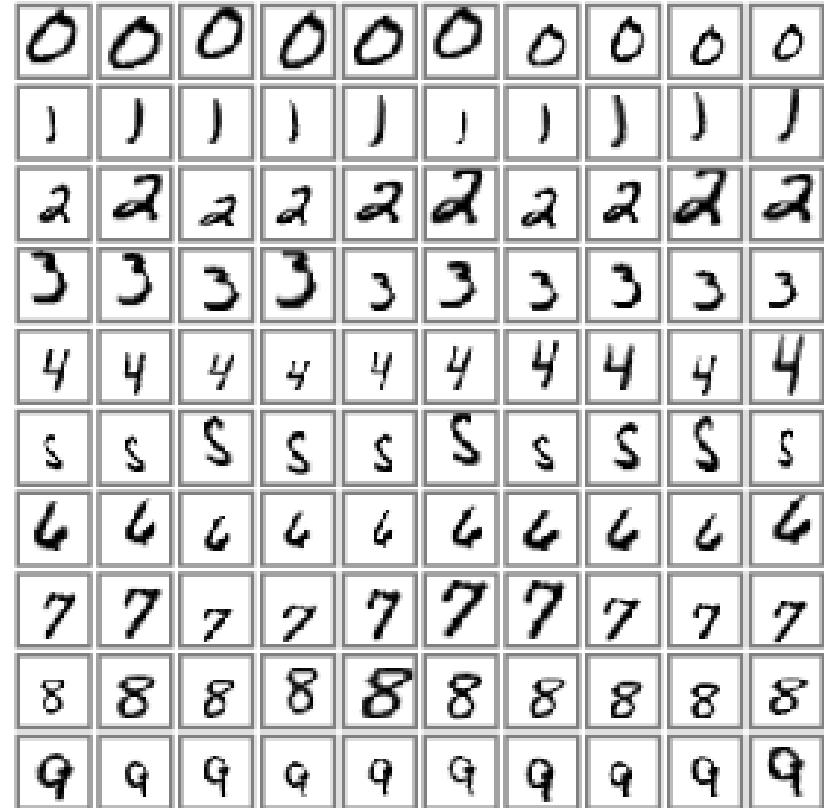
Algorithm:

- 1. find the code Z that minimizes the reconstruction error AND is close to the encoder output
- 2. Update the weights of the decoder to decrease the reconstruction error
- 3. Update the weights of the encoder to decrease the prediction error



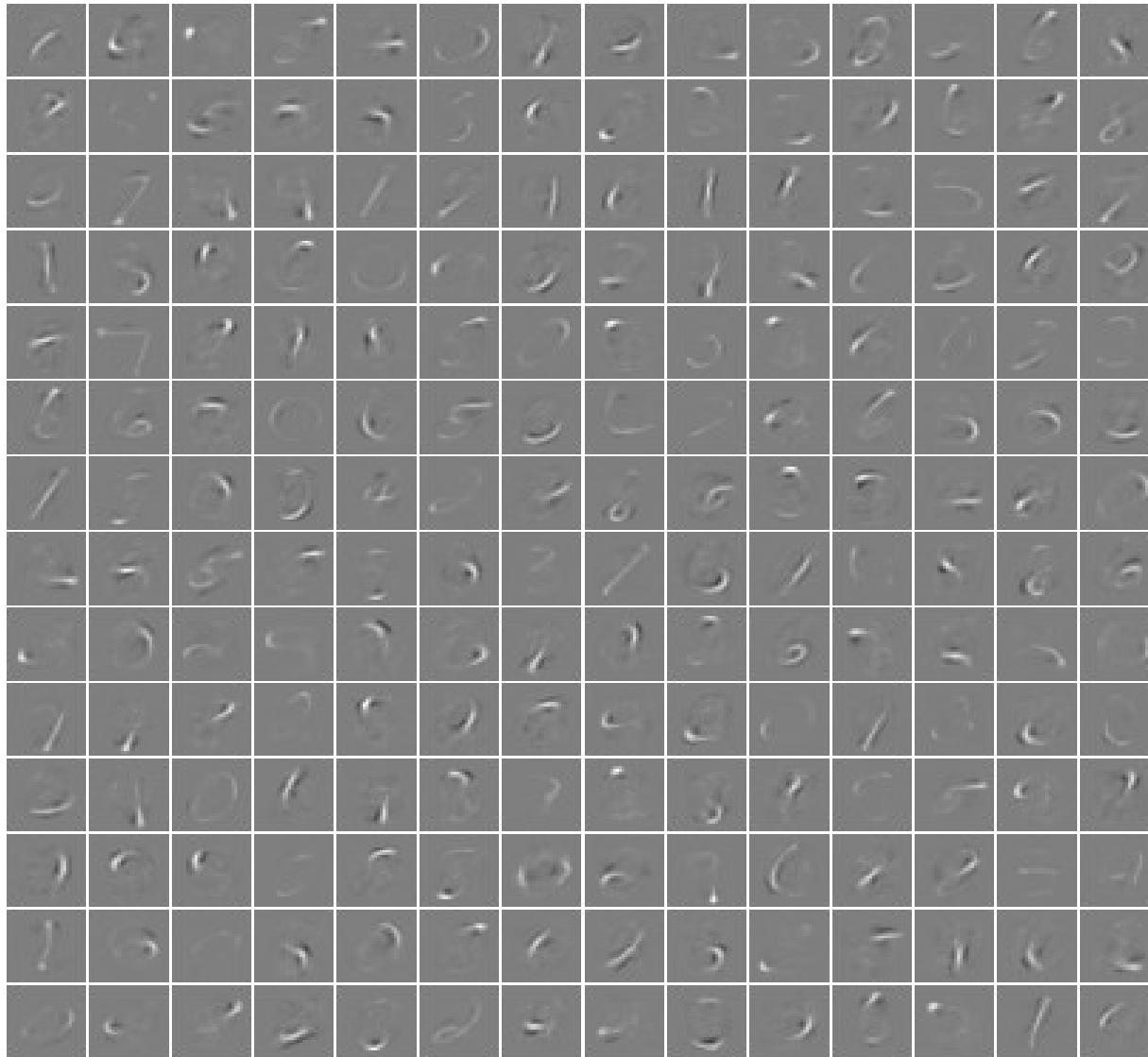
MNIST Dataset

3 6 8 1 7 9 6 6 4 1
6 7 5 7 8 6 3 4 8 5
2 1 7 9 7 1 2 8 4 5
4 8 1 9 0 1 8 8 9 4
7 6 1 8 6 4 1 5 6 0
7 5 9 2 6 5 8 1 9 7
2 2 2 2 3 4 4 8 0
0 2 3 8 0 7 3 8 5 7
0 1 4 6 4 6 0 2 4 3
7 1 2 8 7 6 9 8 6 1



- Handwritten Digit Dataset MNIST: 60,000 training samples, 10,000 test samples

Training on handwritten digits



- ◆ 60,000 28x28 images
- ◆ 196 units in the code
- ◆ η 0.01
- ◆ β 1
- ◆ learning rate 0.001
- ◆ L1, L2 regularizer 0.005

Encoder *direct* filters

Handwritten digits - MNIST

$$\begin{array}{c} \text{original} \\ \boxed{7} \end{array} \approx \begin{array}{c} \text{reconstructed} \\ \text{without minimization} \\ \boxed{7} \end{array} = 1 \begin{array}{c} \boxed{9} \end{array} + 1 \begin{array}{c} \boxed{5} \end{array} + 1 \begin{array}{c} \boxed{7} \end{array}$$

$$+ 1 \begin{array}{c} \boxed{8} \end{array} + 1 \begin{array}{c} \boxed{7} \end{array} + 0.8 \begin{array}{c} \boxed{7} \end{array}$$

$$+ 1 \begin{array}{c} \boxed{7} \end{array} + 1 \begin{array}{c} \boxed{7} \end{array} + 0.8 \begin{array}{c} \boxed{7} \end{array}$$

$$\begin{array}{c} \text{original} \\ \boxed{7} \end{array} - \begin{array}{c} \text{reconstructed} \\ \text{without minimization} \\ \boxed{7} \end{array} = \begin{array}{c} \text{difference} \\ \boxed{9} \end{array}$$

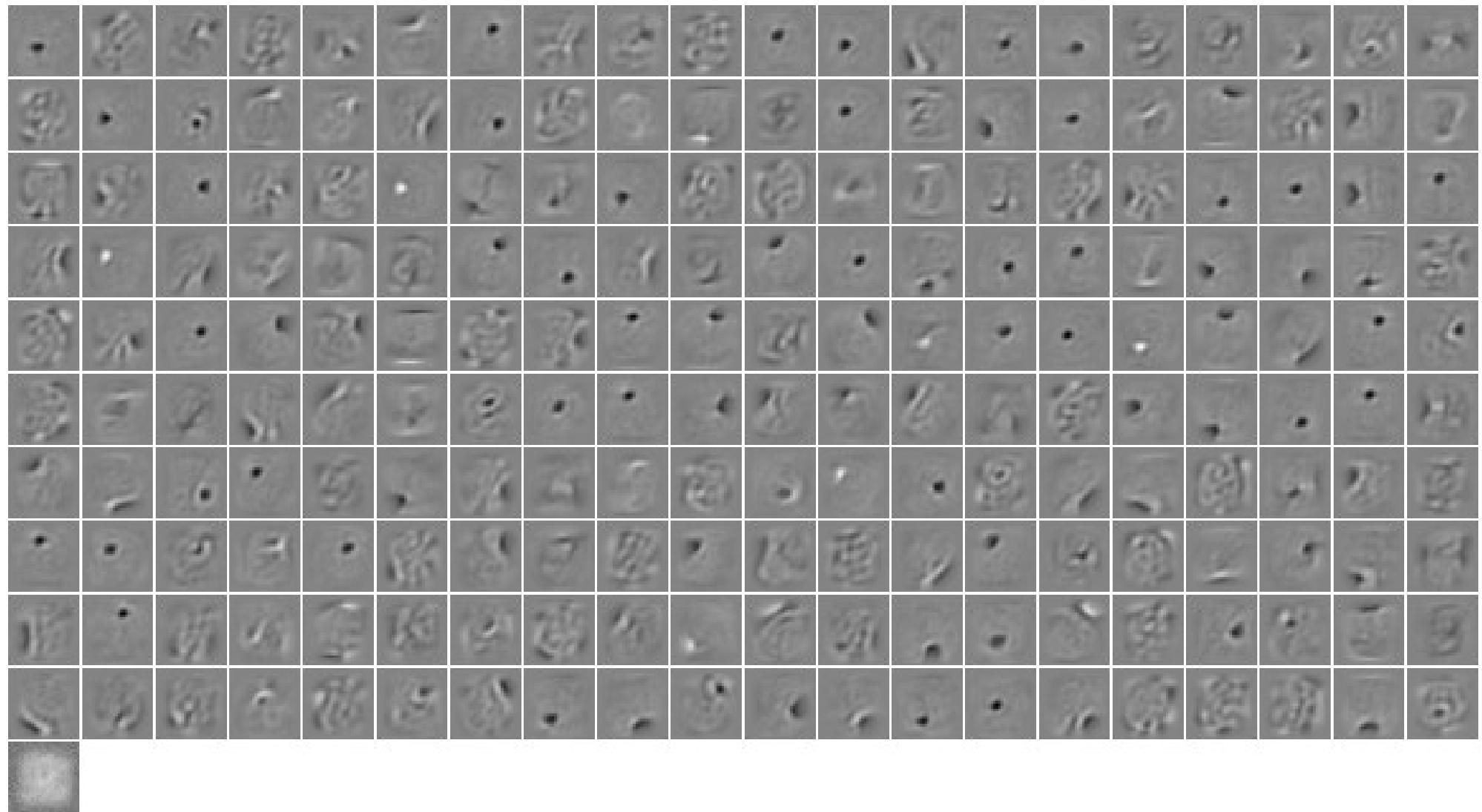
forward propagation through
encoder and decoder

$$\begin{array}{c} \text{reconstructed} \\ \text{minimizing} \\ \boxed{7} \end{array} - \begin{array}{c} \text{reconstructed} \\ \text{without minimization} \\ \boxed{7} \end{array} = \begin{array}{c} \text{difference} \\ \boxed{7} \end{array}$$

after training there is no need to
minimize in code space

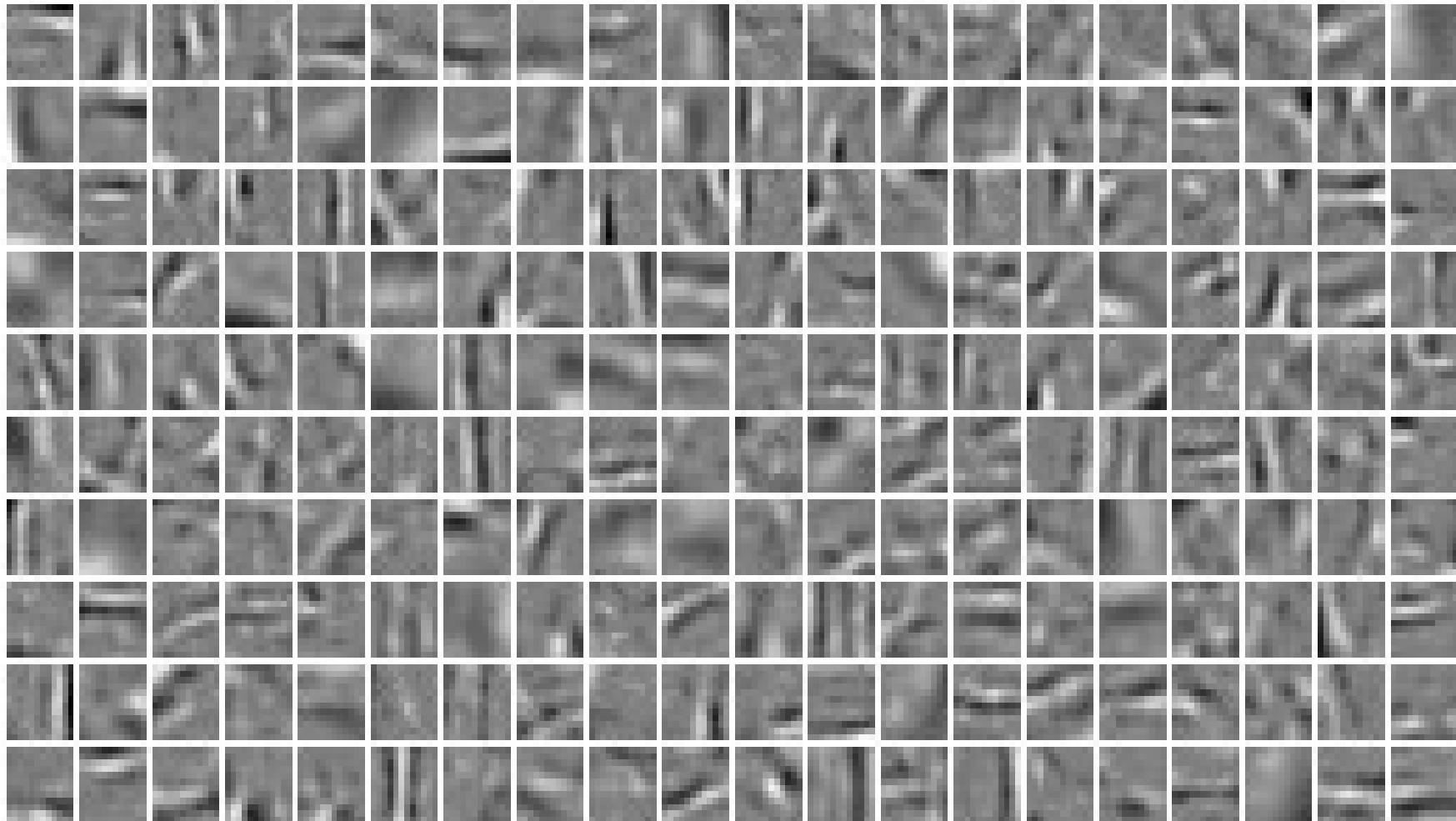
RBM: filters trained on MNIST

• “bubble” detectors

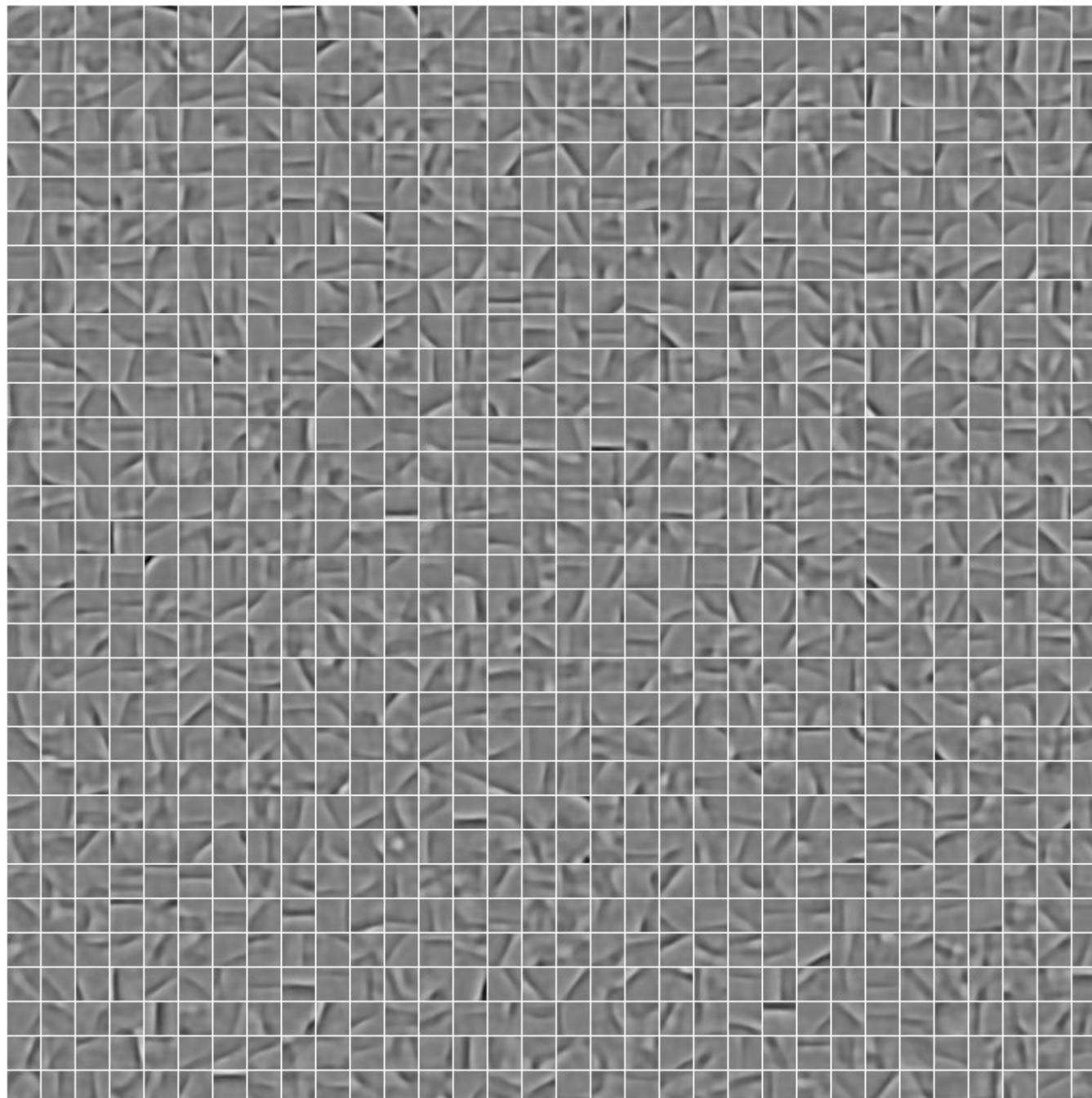


Sparse Encoder/Decoder Architecture on Natural Image Patches

- Orientation-sensitive feature detectors



RBM: filters trained on natural images WITH SPARSITY

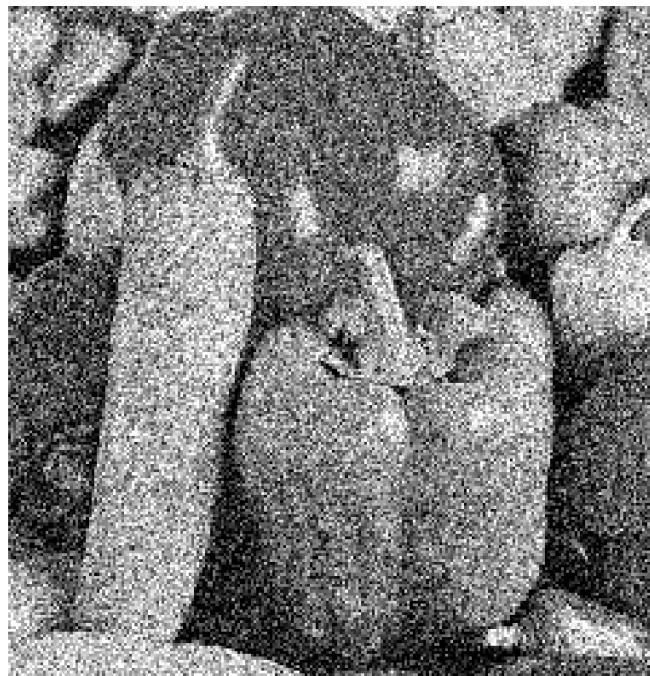


Denoising



original image

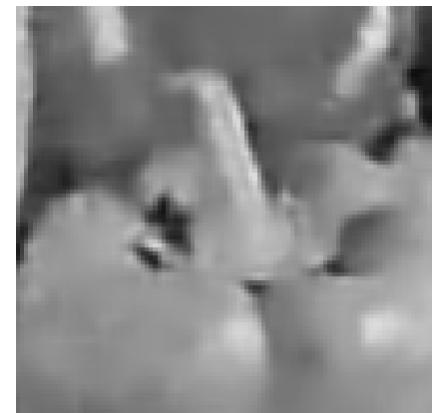
ZOOM ->



noisy image: PSNR 14.15dB
(std. dev. noise 50)



denoised image
PSNR 26.50dB



Denoising

<i>s.d. / PSNR</i>	<i>Lena</i>				<i>Barbara</i>				<i>Boat</i>				<i>House</i>				<i>Peppers</i>			
50 / 14.15	27.86 28.61 27.79 26.49				23.46 25.48 25.47 23.15				26.02 26.38 25.95 24.53				27.85 28.26 27.95 26.74				26.35 25.90 26.13 24.52			
75 / 10.63	25.97 26.84 25.80 24.13				22.46 23.65 23.01 21.36				24.31 24.79 23.98 22.48				25.77 26.41 25.22 24.13				24.56 24.00 23.69 21.68			
100 / 8.13	24.49 25.64 24.46 21.87				21.77 22.61 21.89 19.77				23.09 23.75 22.81 20.80				24.20 25.11 23.71 21.66				23.04 22.66 21.75 19.60			

Comparison between:

- our method [first column]
- Portilla et al. IEEE Trans. Image Processing (2003) [second column]
- Elad and Aharon CVPR 2006 [third column]
- Roth and Black CVPR 2005 [fourth column]

Unsupervised Training of Convolutional Filters

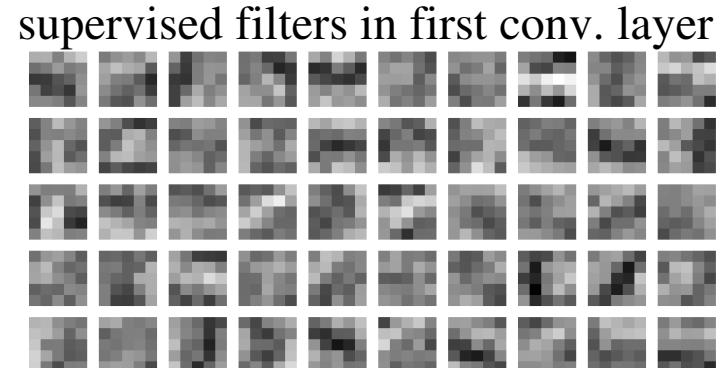
CLASSIFICATION EXPERIMENTS

IDEA: improving supervised learning by pre-training
with the unsupervised method (*)

sparse representations & lenet6 (1->50->50->200->10)

- The **baseline**: *lenet6* initialized randomly

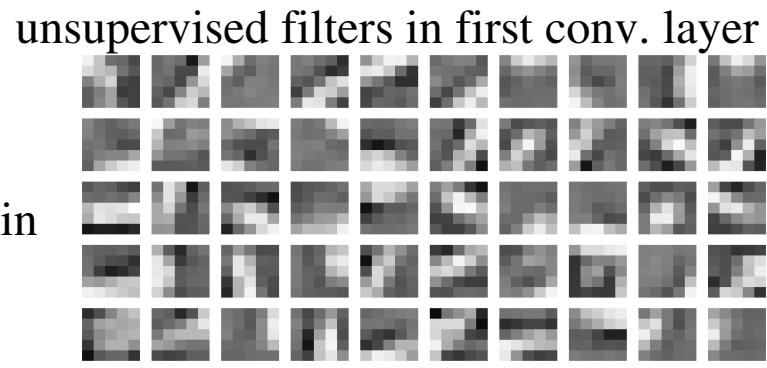
Test error rate: **0.70%**. Training error rate: 0.01%.



- *Experiment 1*

- Train on 5x5 patches to find 50 features
- Use the scaled filters in the encoder to initialize the kernels in the first convolutional layer

Test error rate: **0.60%**. Training error rate: 0.00%.



- *Experiment 2*

- Same as experiment 1, but training set augmented by elastically distorted digits (random initialization gives test error rate equal to **0.49%**).

Test error rate: **0.39%**. Training error rate: 0.23%.

Best Results on MNIST (from raw images: no preprocessing)

CLASSIFIER	DEFORMATION	ERROR	Reference
Knowledge-free methods			
2-layer NN, 800 HU, CE		1.60	Simard et al., ICDAR 2003
3-layer NN, 500+300 HU, CE, reg		1.53	Hinton, in press, 2005
SVM, Gaussian Kernel		1.40	Cortes 92 + Many others
Unsupervised Stacked RBM + backprop		0.95	Hinton, Neur Comp 2006
Convolutional nets			
Convolutional net LeNet-5,		0.80	Ranzato et al. NIPS 2006
Convolutional net LeNet-6,		0.70	Ranzato et al. NIPS 2006
Conv. net LeNet-6- + unsup learning		0.60	Ranzato et al. NIPS 2006
Training set augmented with Affine Distortions			
2-layer NN, 800 HU, CE	Affine	1.10	Simard et al., ICDAR 2003
Virtual SVM deg-9 poly	Affine	0.80	Scholkopf
Convolutional net, CE	Affine	0.60	Simard et al., ICDAR 2003
Training set augmented with Elastic Distortions			
2-layer NN, 800 HU, CE	Elastic	0.70	Simard et al., ICDAR 2003
Convolutional net, CE	Elastic	0.40	Simard et al., ICDAR 2003
Conv. net LeNet-6- + unsup learning	Elastic	0.39	Ranzato et al. NIPS 2006

Sparse Features with “Predictable Basis Pursuit”

- Linear Decoder with **normalized basis functions**

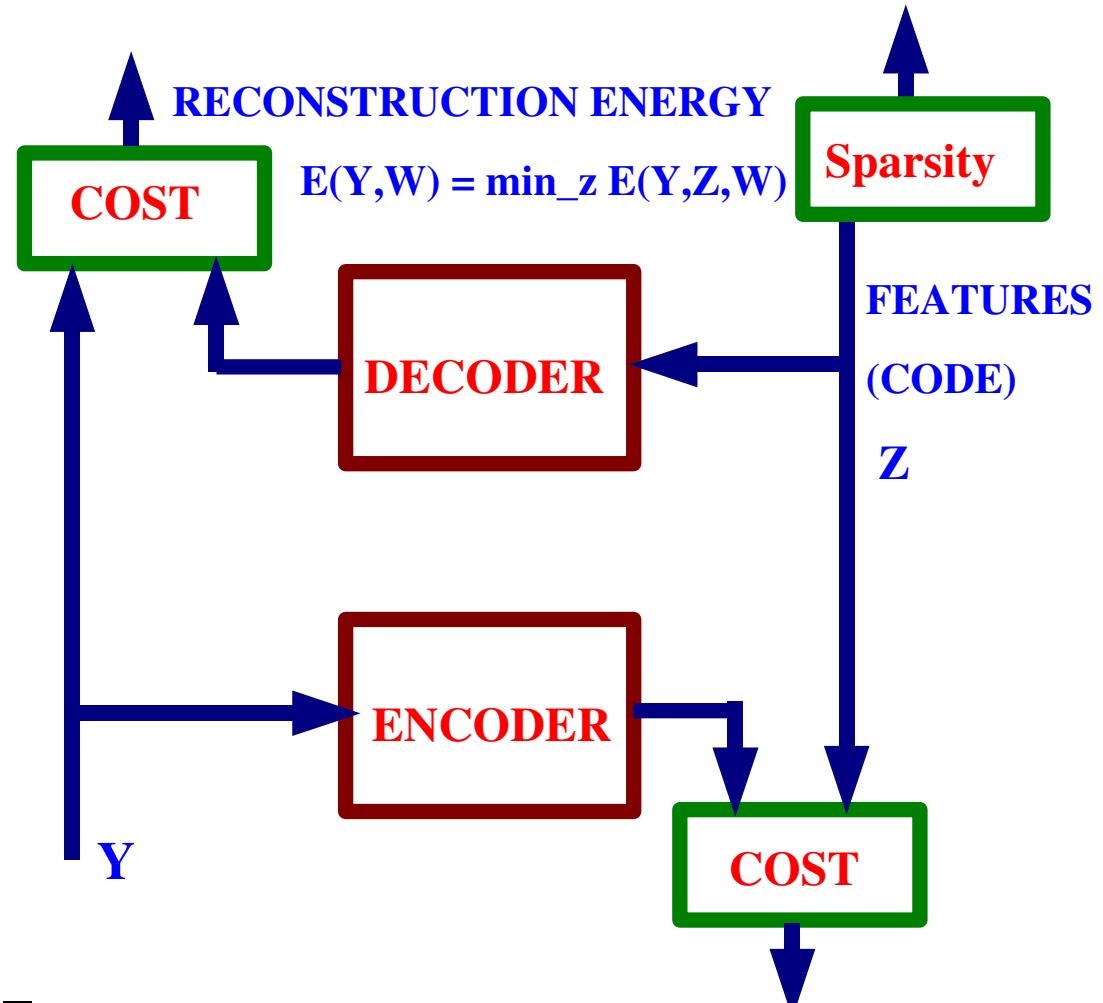
- L1 Sparsity penalty

- Encoder of different types

- ▶ Linear
- ▶ Linear-Sigmoid-Scaling
- ▶ Linear-Sigmoid-Linear

- The decoder+sparsity is identical to Chen & Donoho's basis pursuit

- But the encoder learns to “predicts” the optimal feature codes



$$\bar{Z}_Y = \operatorname{argmin}_Z E(Y, Z, W)$$

$$E(Y, W) = \min_Z E(Y, Z, W)$$

Encoder/Decoder: Predictable Basis Pursuit

Decoder:

- Linear $\|y - \Phi z\|_2^2 + \alpha_z \|z\|_1$

Encoders of different types:

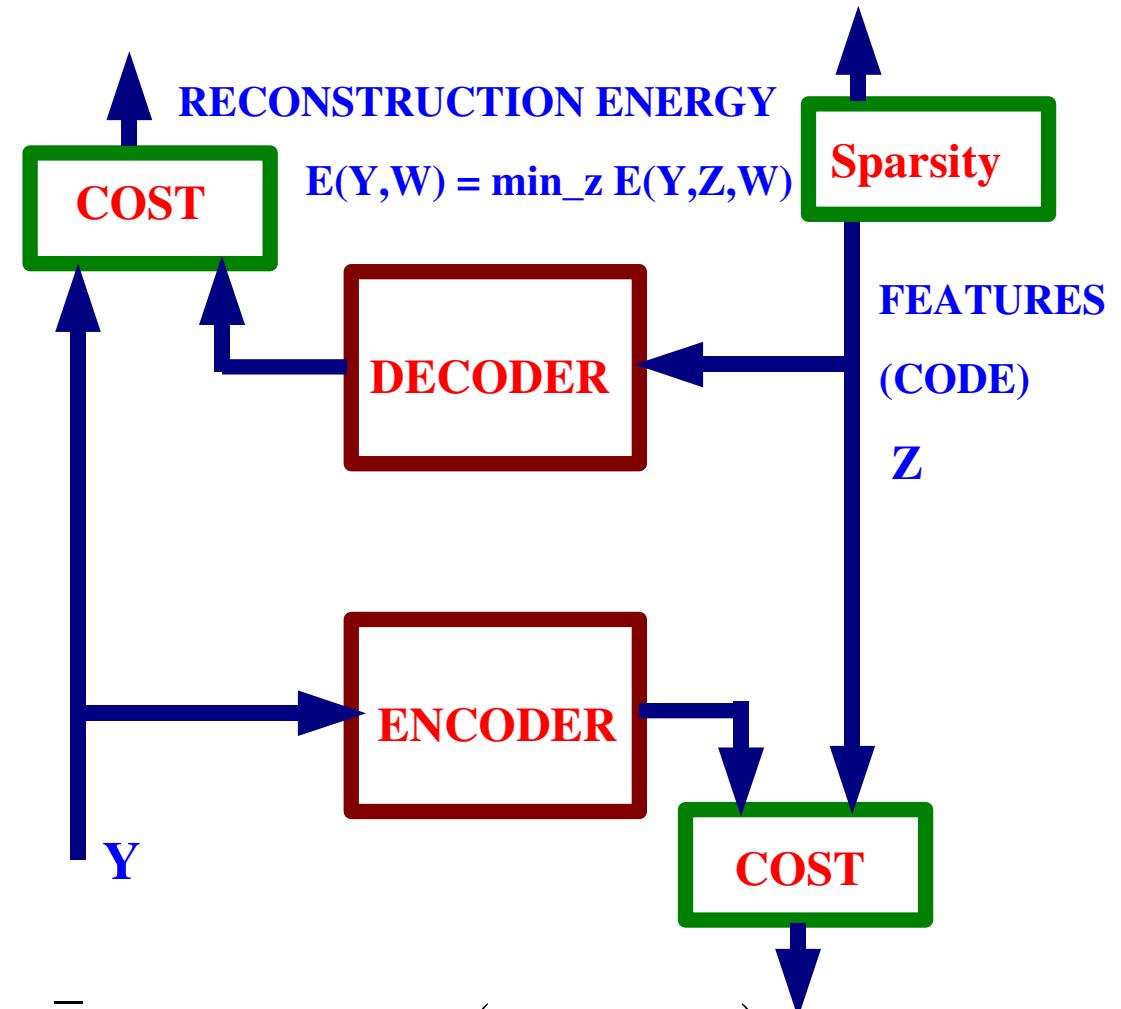
- None
- Linear
- Linear-Sigmoid-Scaling
- Linear-Sigmoid-Linear

Sparsity penalty

- L1

Main Idea:

- find basis functions such that the coefficients that reconstruct any vector can be predicted by the encoder.



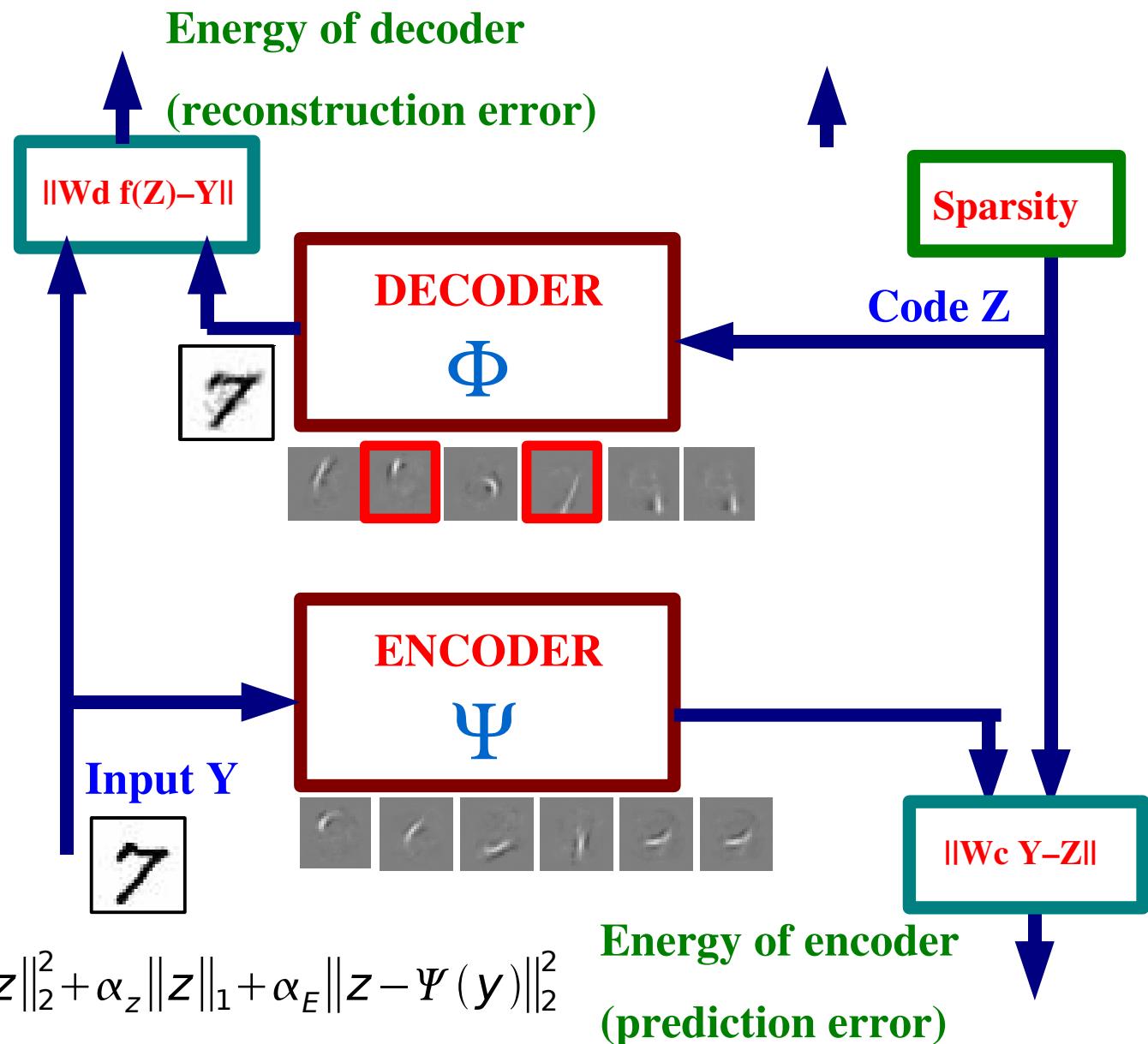
$$\bar{Z}_Y = \operatorname{argmin}_Z E(Y, Z, W)$$

$$E(Y, W) = \min_Z E(Y, Z, W)$$

Training The Predictable Basis Pursuit Model

Algorithm:

- 1. find the code Z that minimizes the reconstruction error AND is close to the encoder output
- 2. Update the weights of the decoder to decrease the reconstruction error
- 3. Update the weights of the encoder to decrease the prediction error



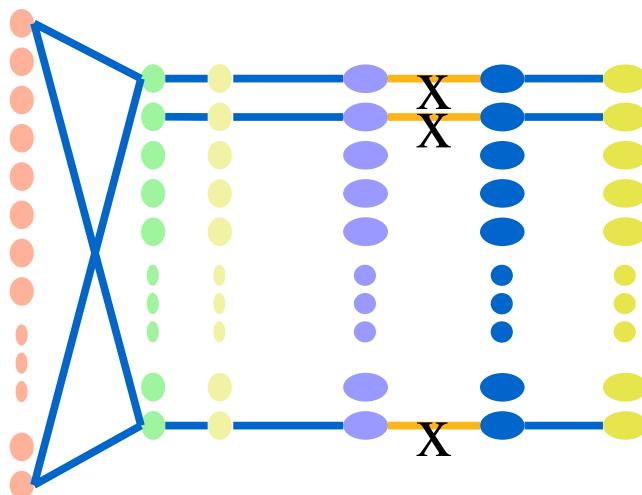
$$L(y, z, \Phi, \Psi) = \|y - \Phi z\|_2^2 + \alpha_z \|z\|_1 + \alpha_E \|z - \Psi(y)\|_2^2$$

Energy of encoder
(prediction error)

Encoder Architectures

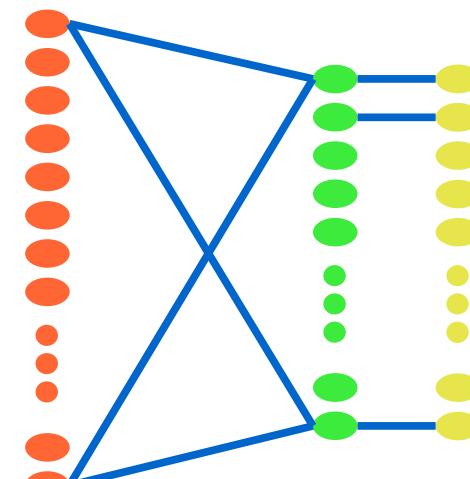
- L: Linear
- FD: Linear + Sigmoid + Gain + Bias
- FL: Linear + Sigmoid + Linear

Non-Linear – Individual gains (FD)

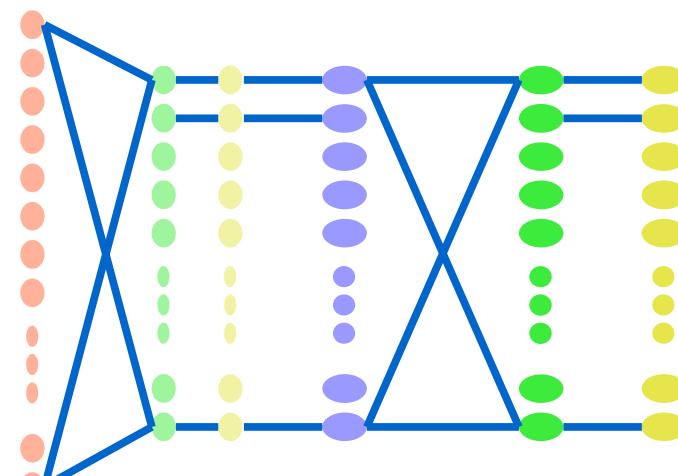


$$Z' = \sigma(MY + b_1) \times \text{diag}(g) + b_2$$

Linear (L)

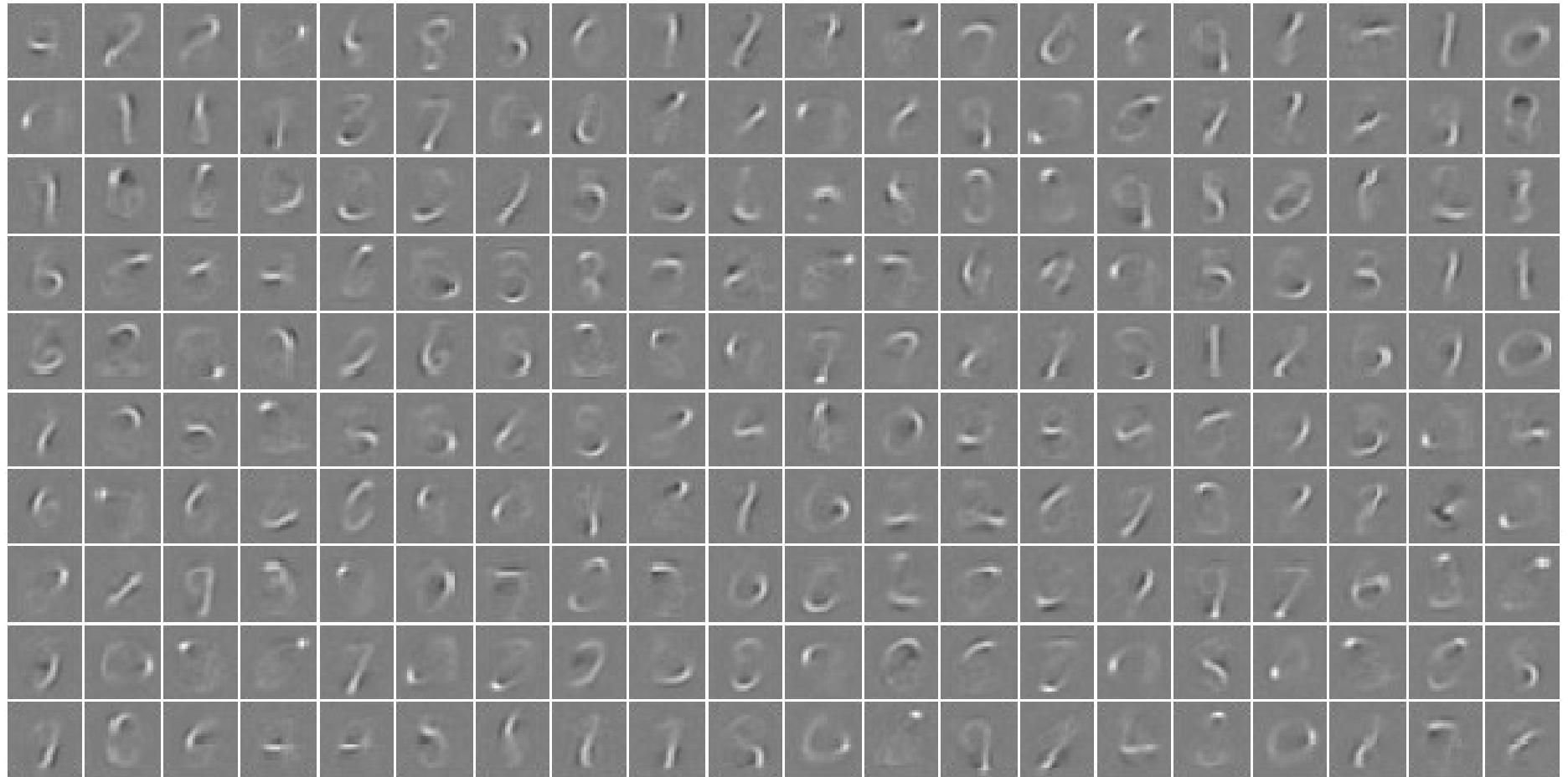


Non-Linear – 2 Layer NN (FL)



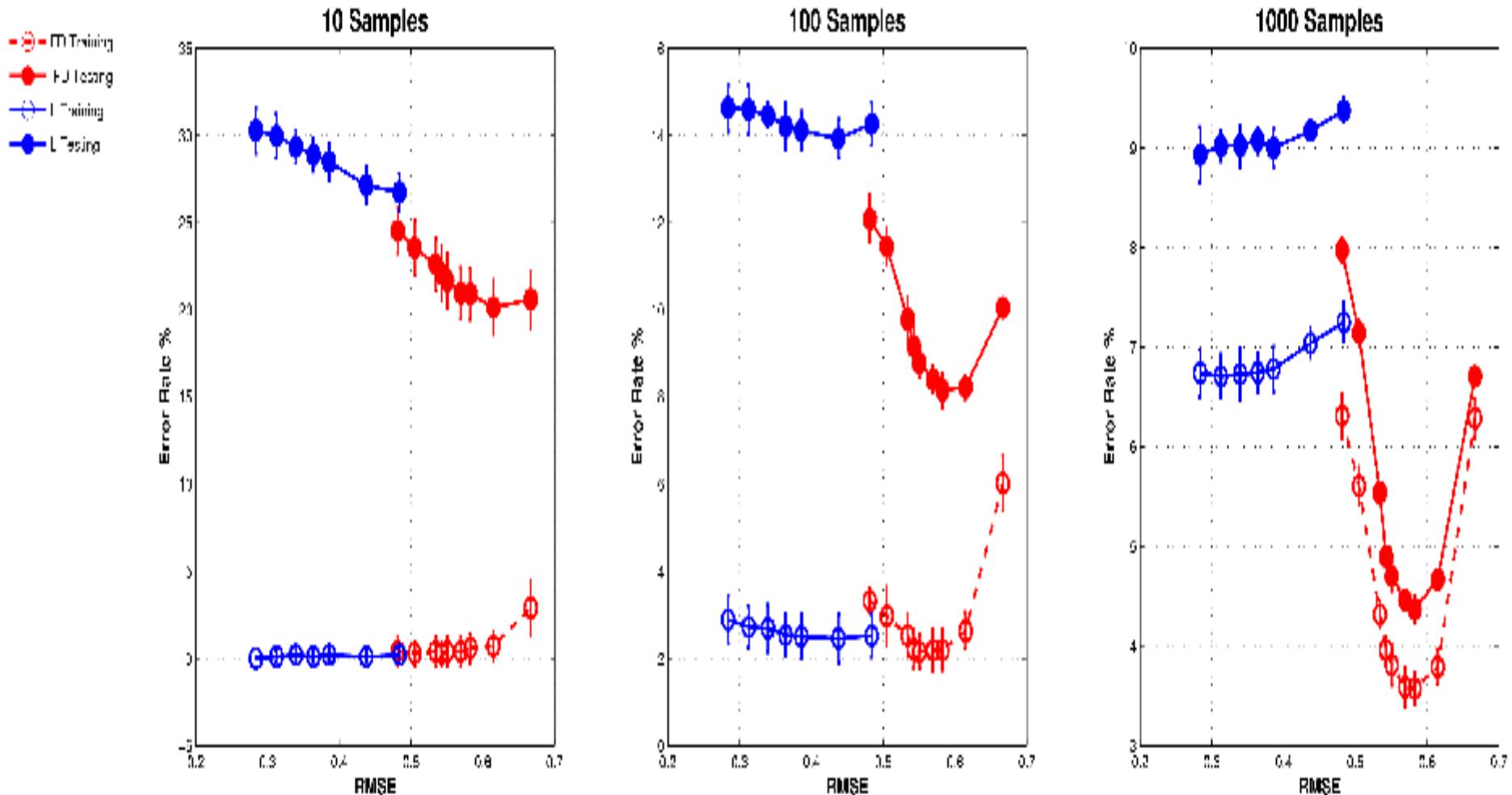
$$Z' = M_2 \sigma(M_1 Y + b_1) + b_2$$

Decoder Basis Functions on MNIST



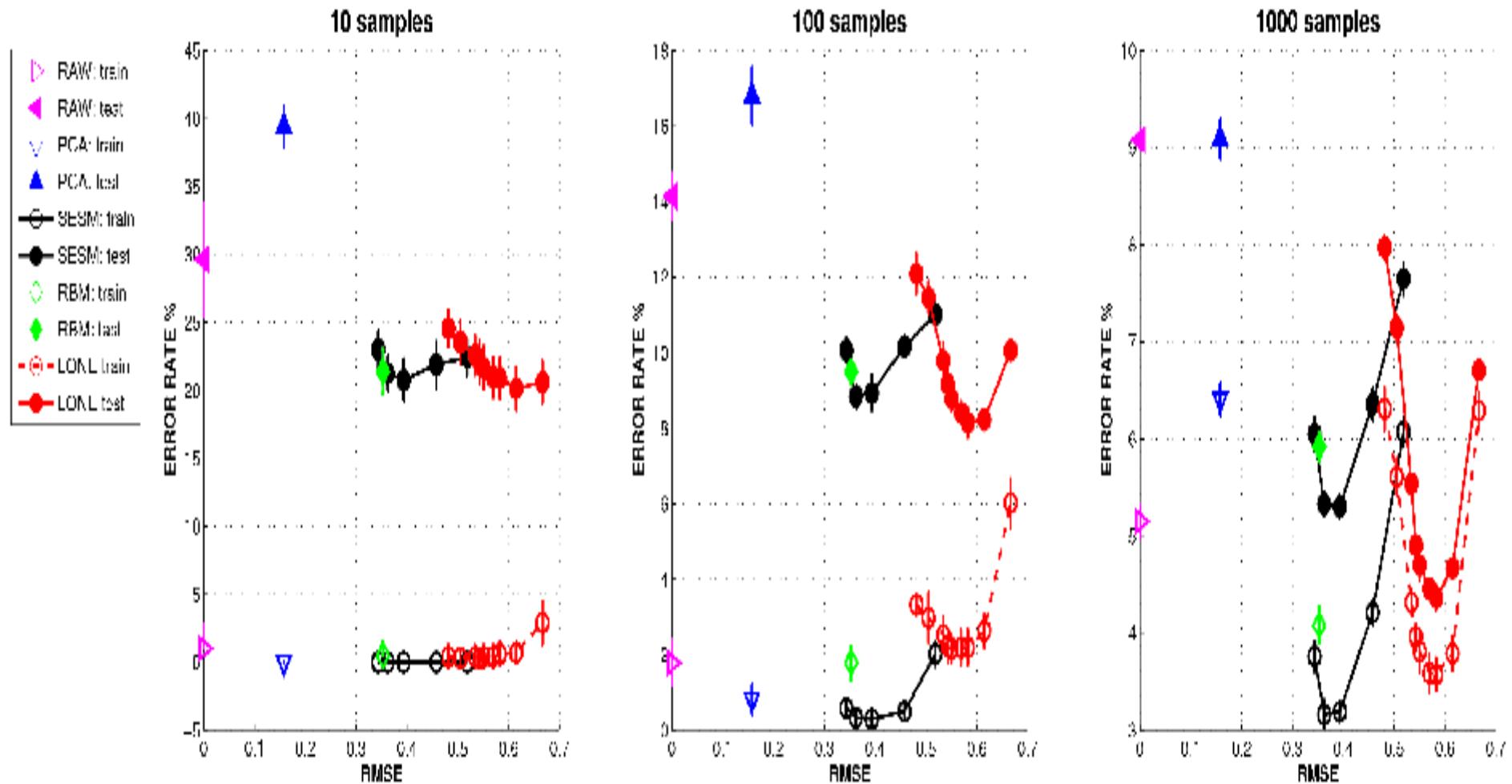
Classification Error Rate on MNIST

Supervised Linear Classifier trained on 200 trained sparse features

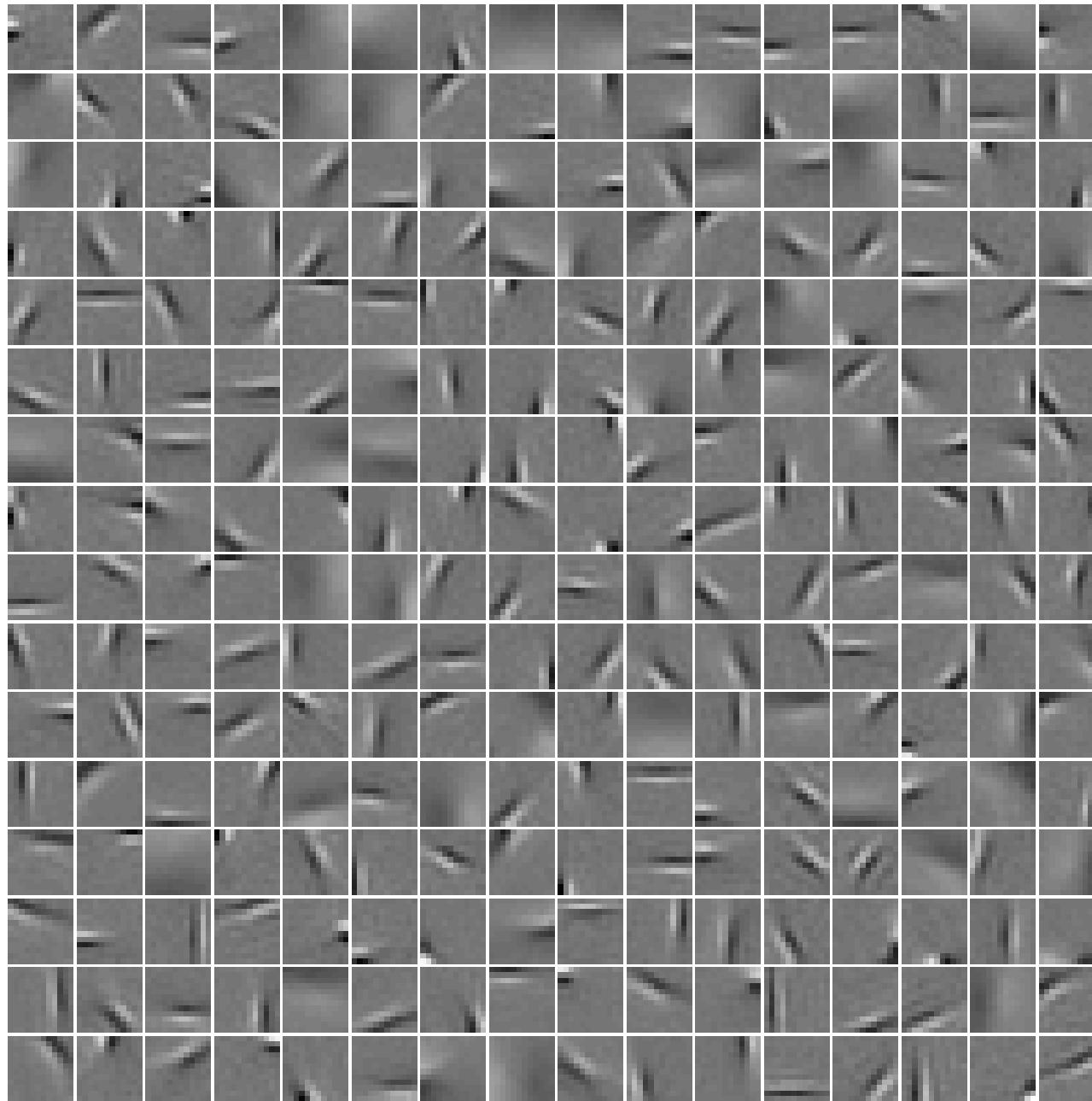


Classification Error Rate on MNIST

Supervised Linear Classifier trained on 200 trained sparse features

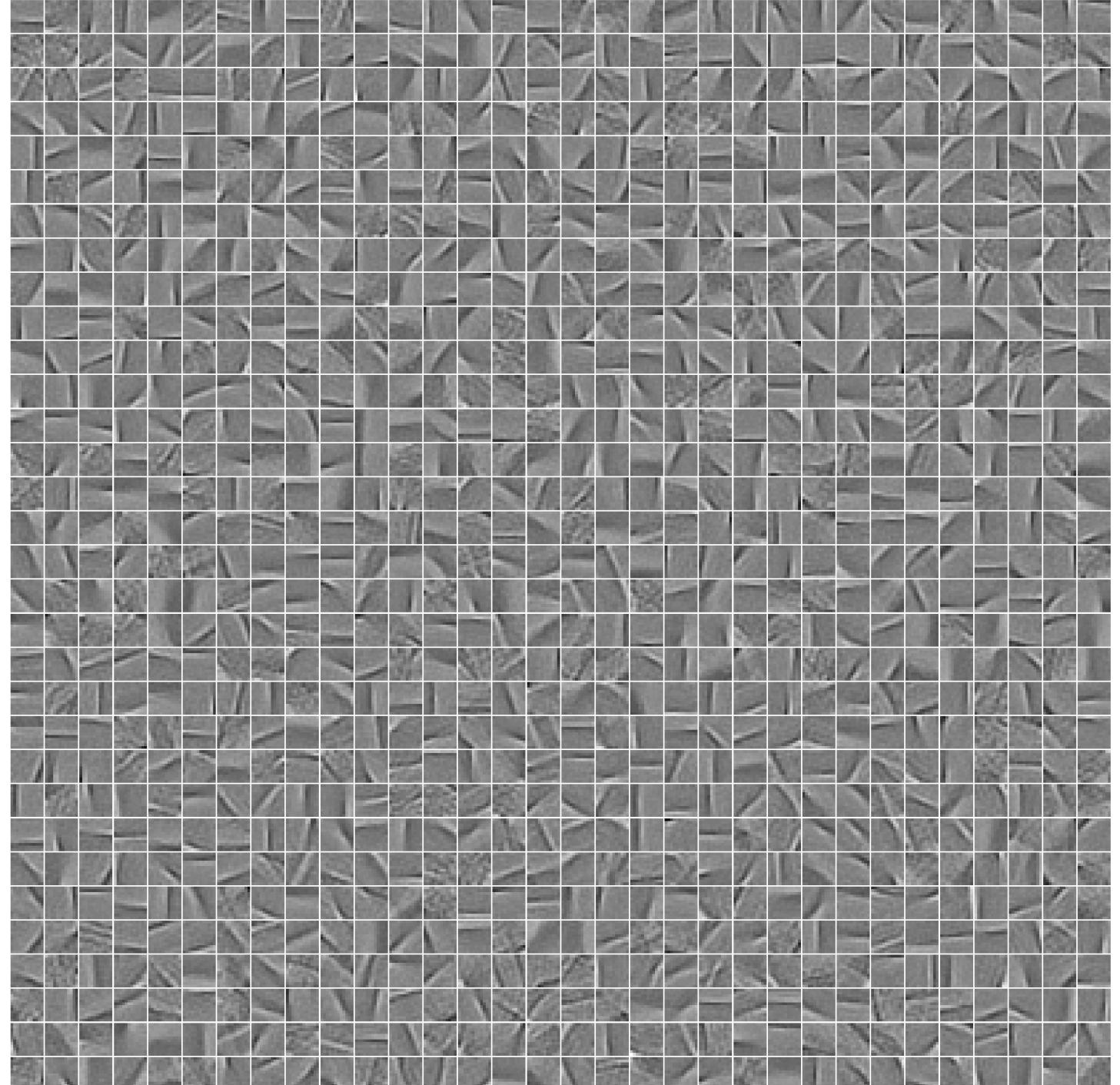


Learned Features: like V1 receptive fields

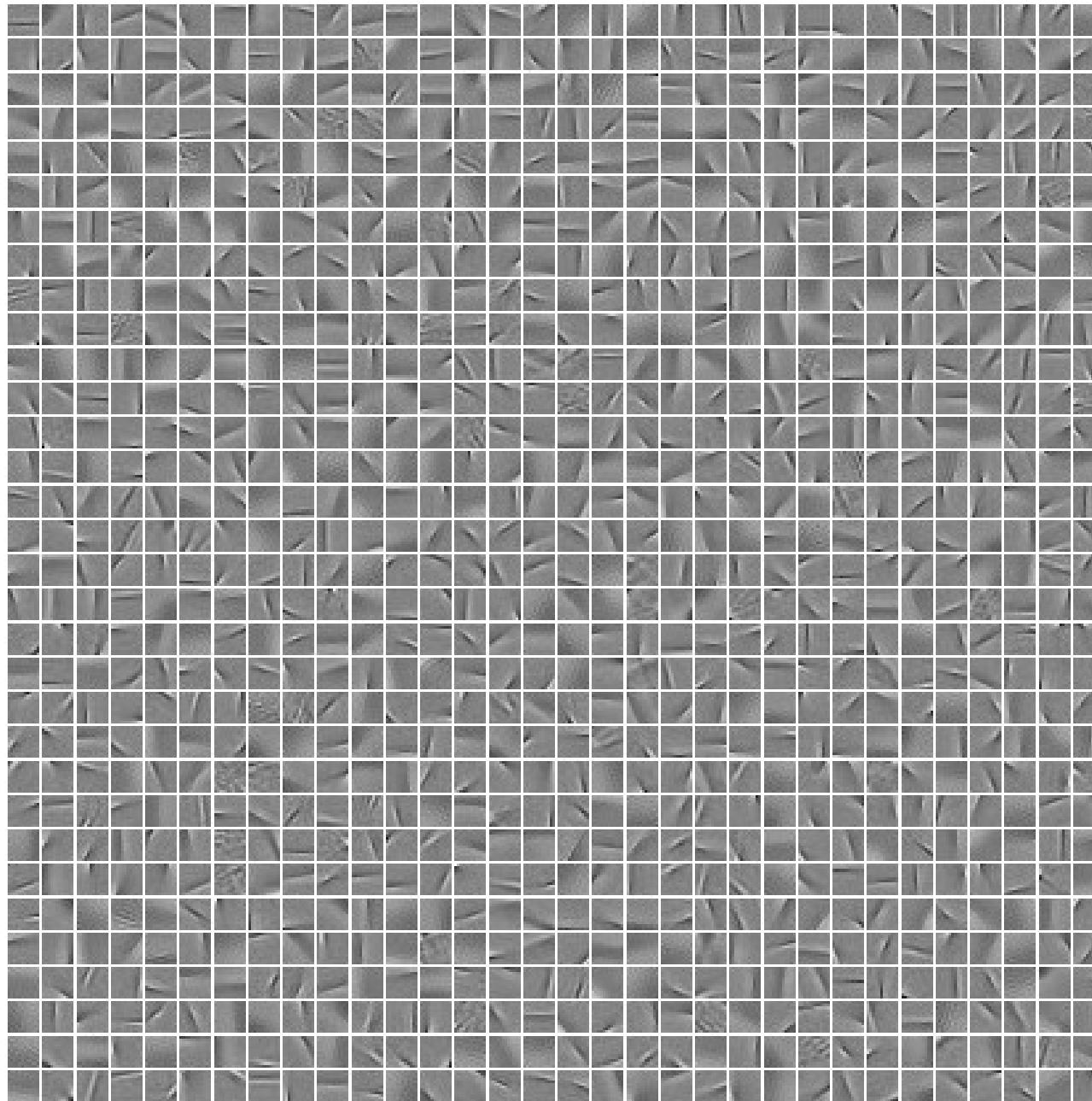


Training on Natural Image Patches

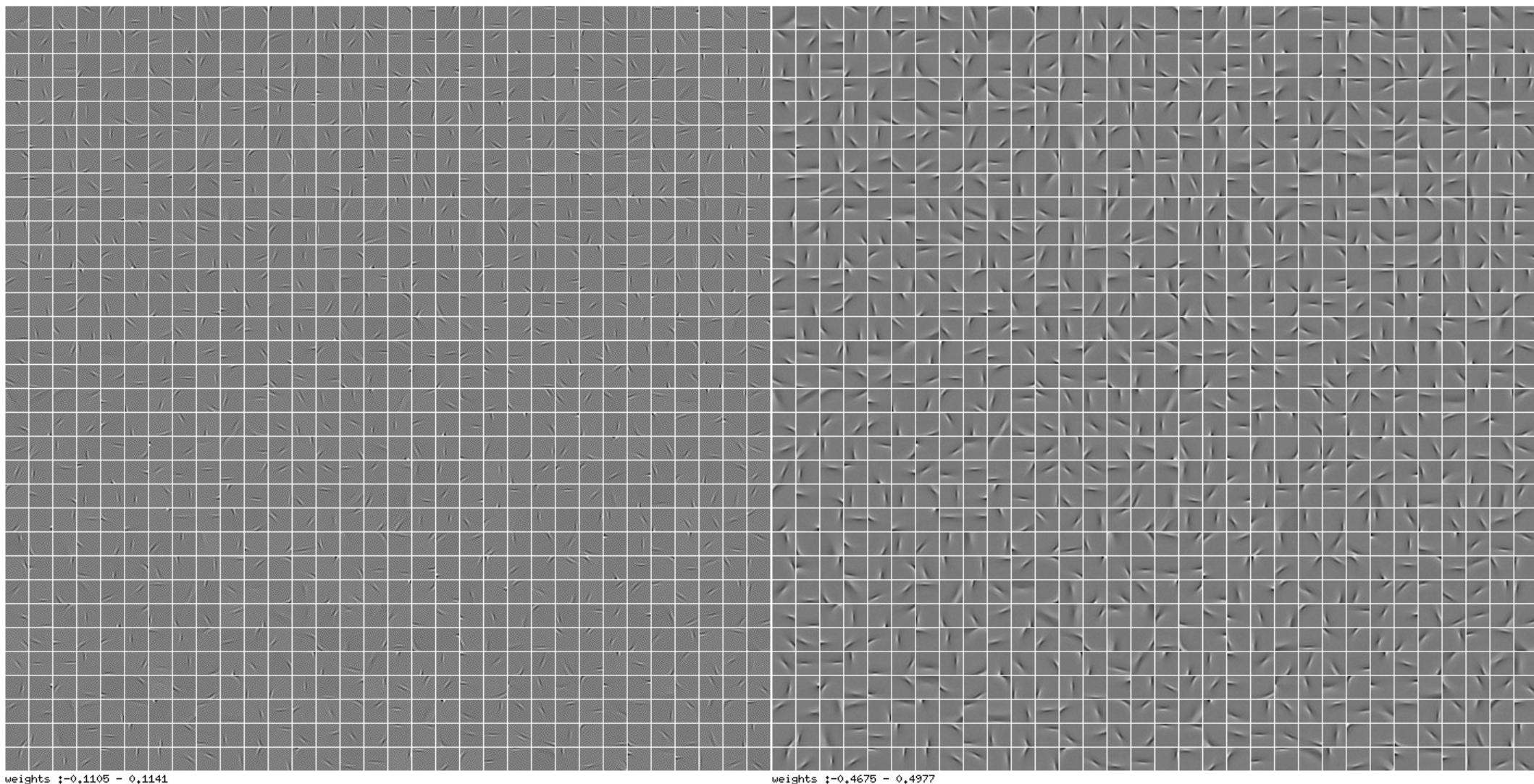
- 12x12 filters
- 1024 filters



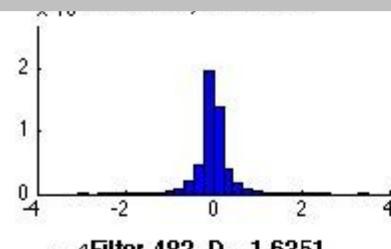
Learned Features: like V1 receptive fields



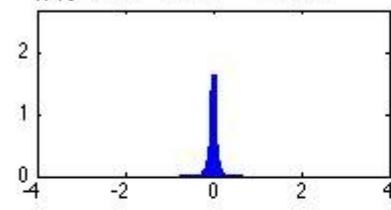
Learned Features: like V1 receptive fields



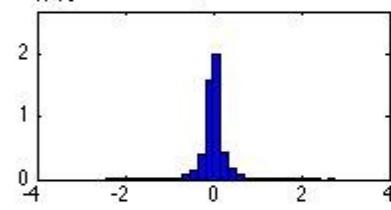
Learned Features: like V1 receptive fields



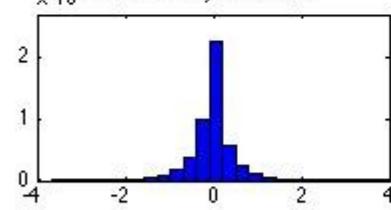
$\times 10^4$ Filter 482, D = 1.6351



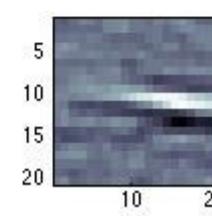
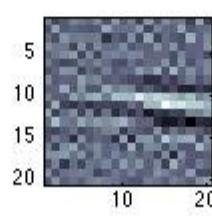
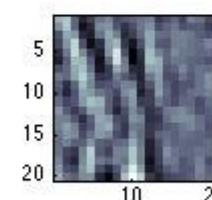
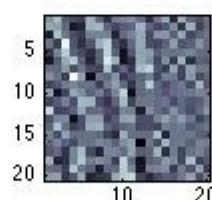
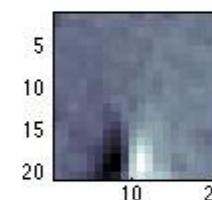
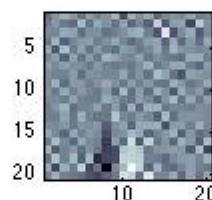
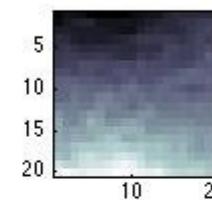
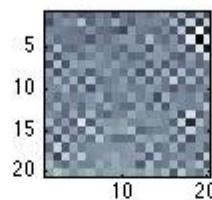
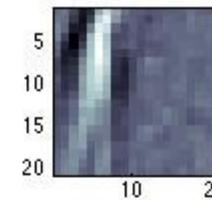
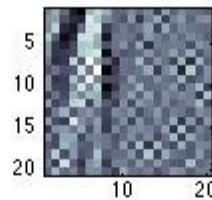
$\times 10^4$ Filter 294, D = 0.9921



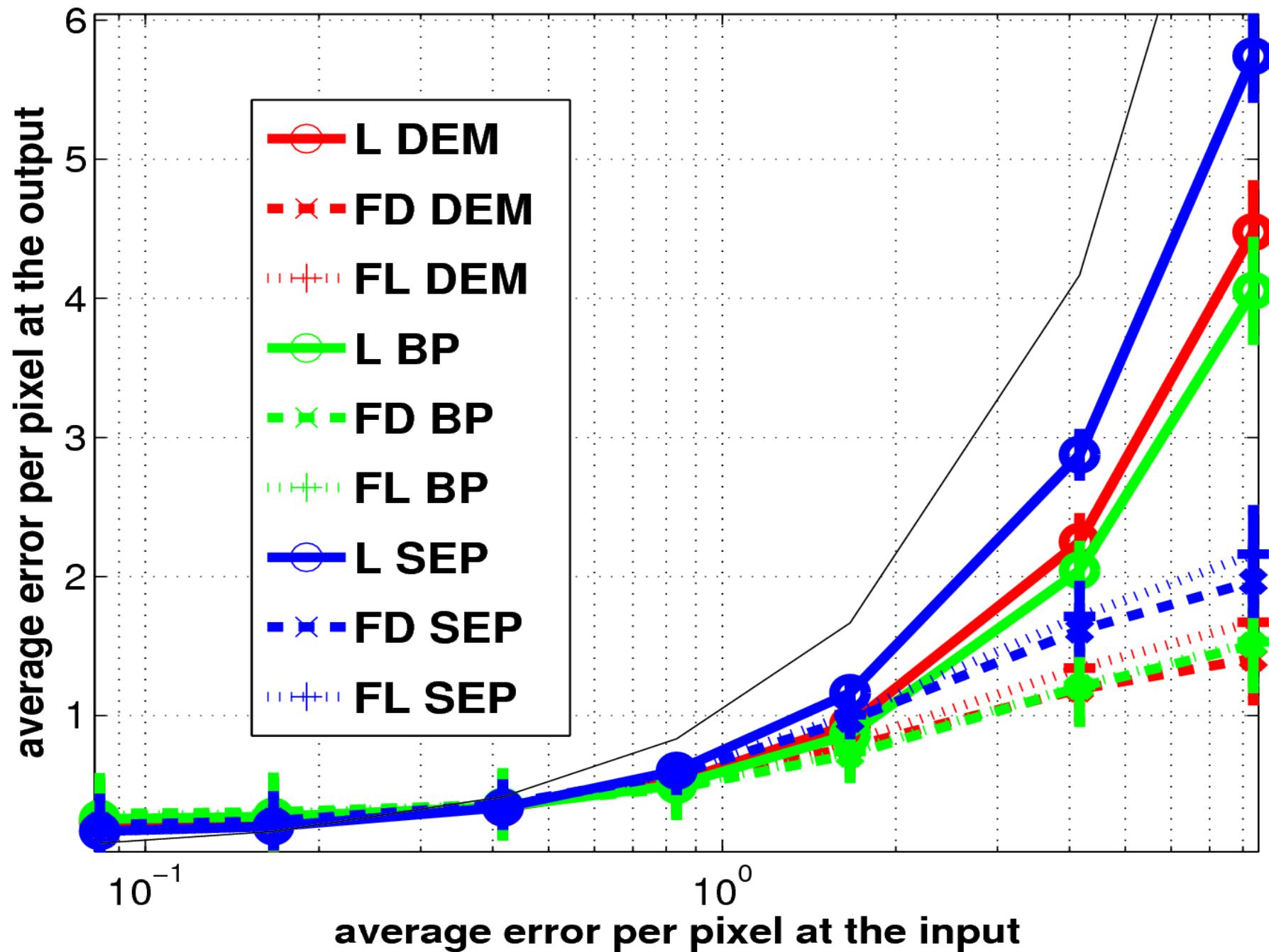
$\times 10^4$ Filter 181, D = 0.655



$\times 10^4$ Filter 22, D = 1.1796

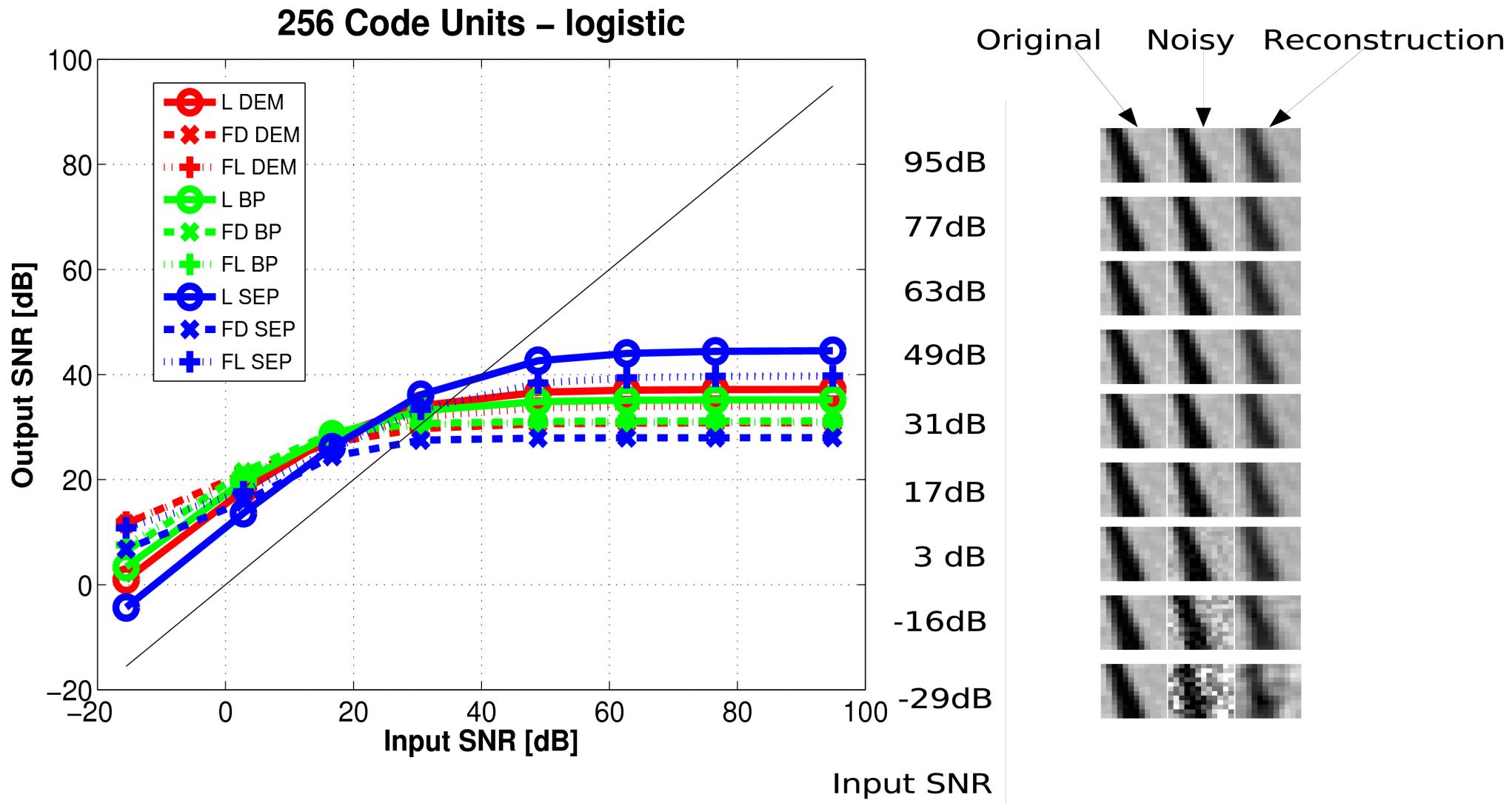


Noise out versus Noise in (1024 code units)



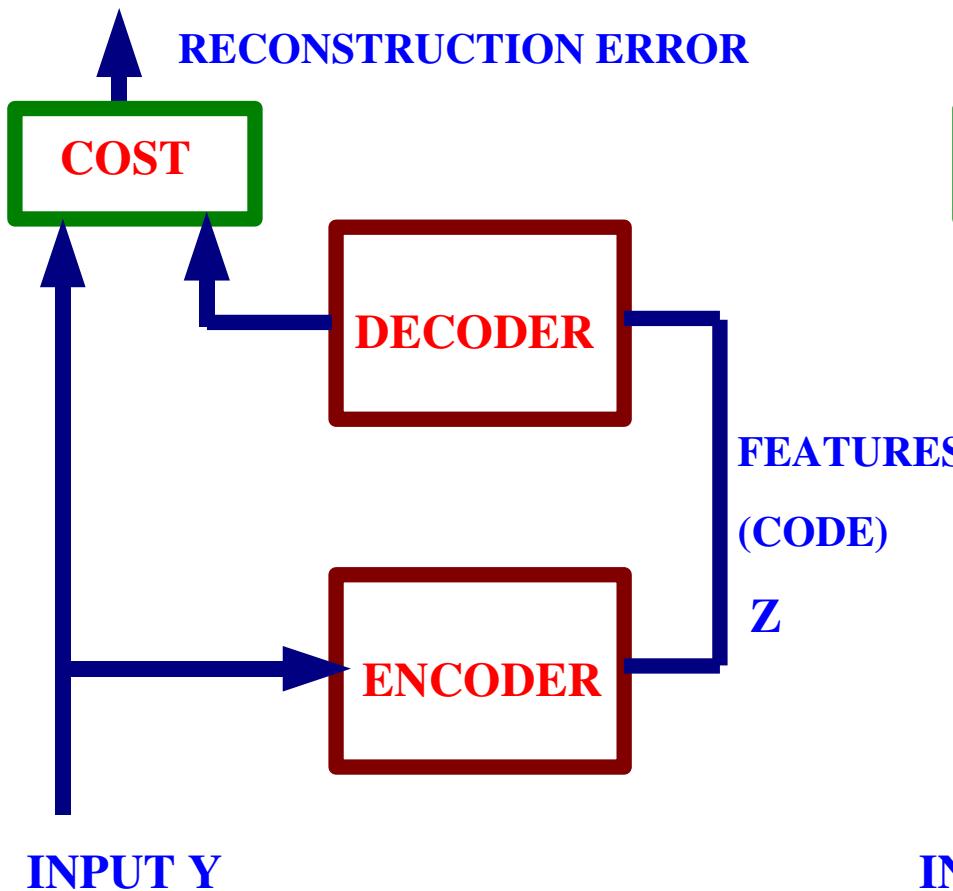
Denoising

■ PBP trained on natural image patches

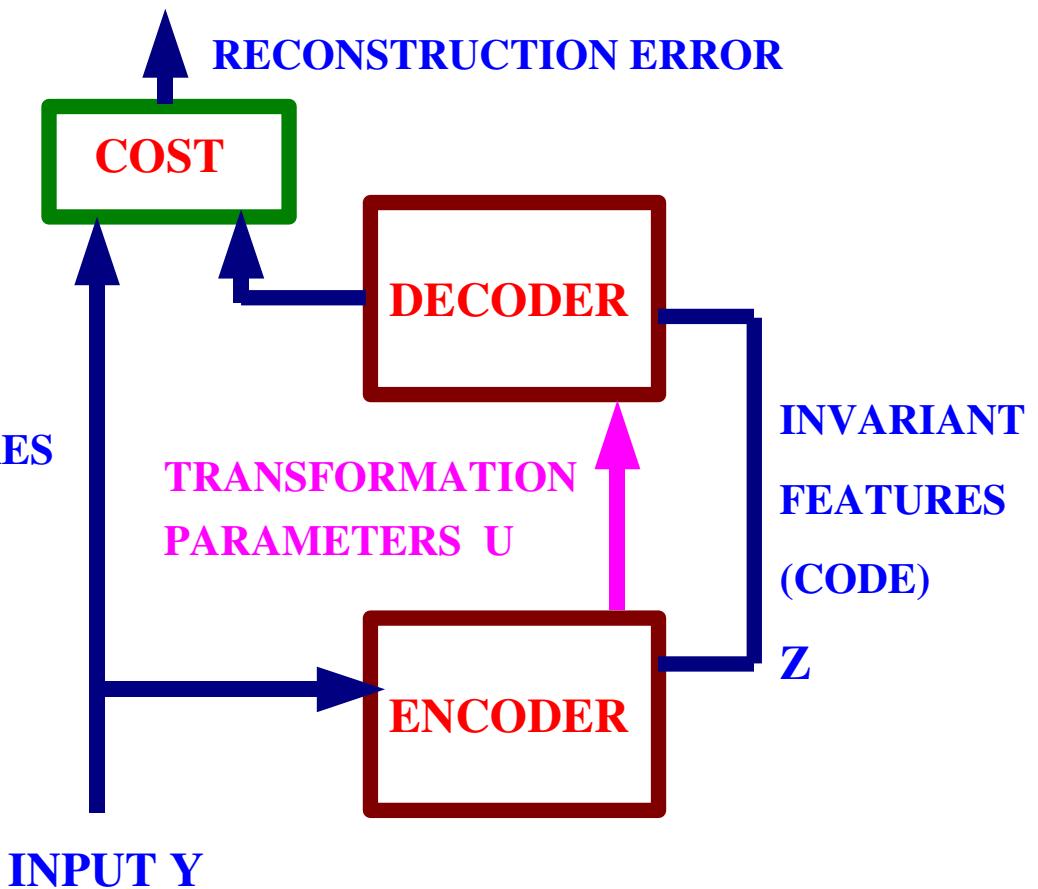


Learning Invariant Features

- Separating the “what” from the “where”



Standard Feature Extractor

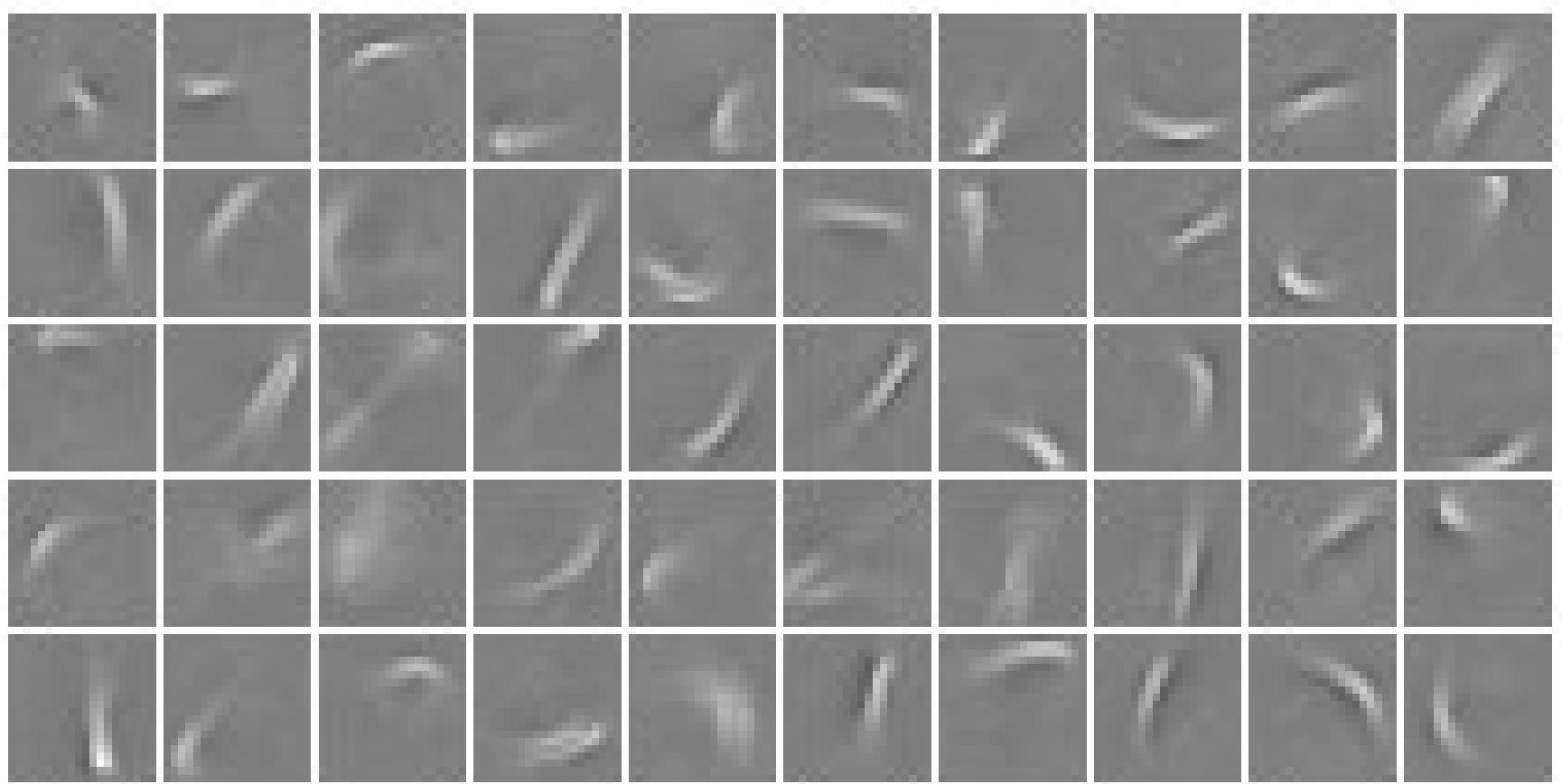


Invariant Feature Extractor

Shift Invariant Global Features on MNIST

Learning 50 Shift Invariant Global Features on MNIST:

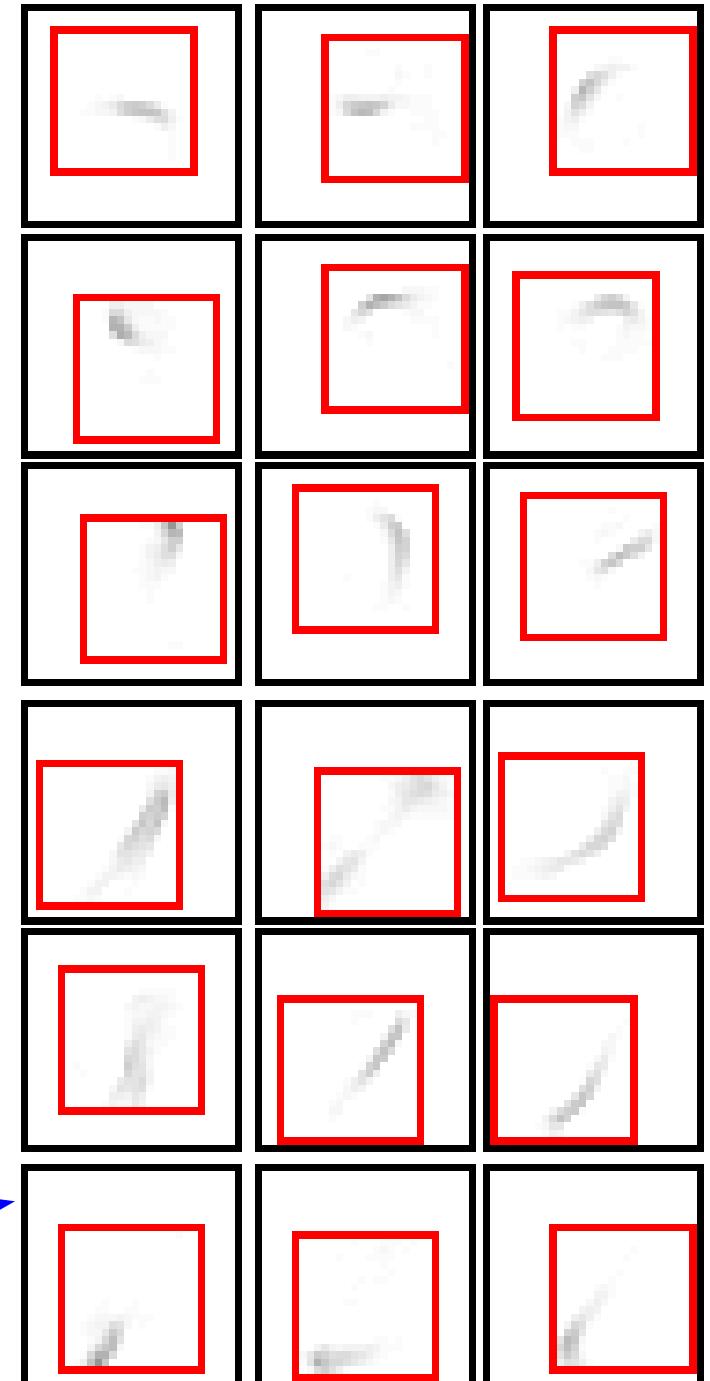
- ▶ 50 filters of size 20x20 movable in a 28x28 frame (81 positions)
- ▶ movable strokes!



Example of Reconstruction

- Any character can be reconstructed as a linear combination of a small number of basis functions.

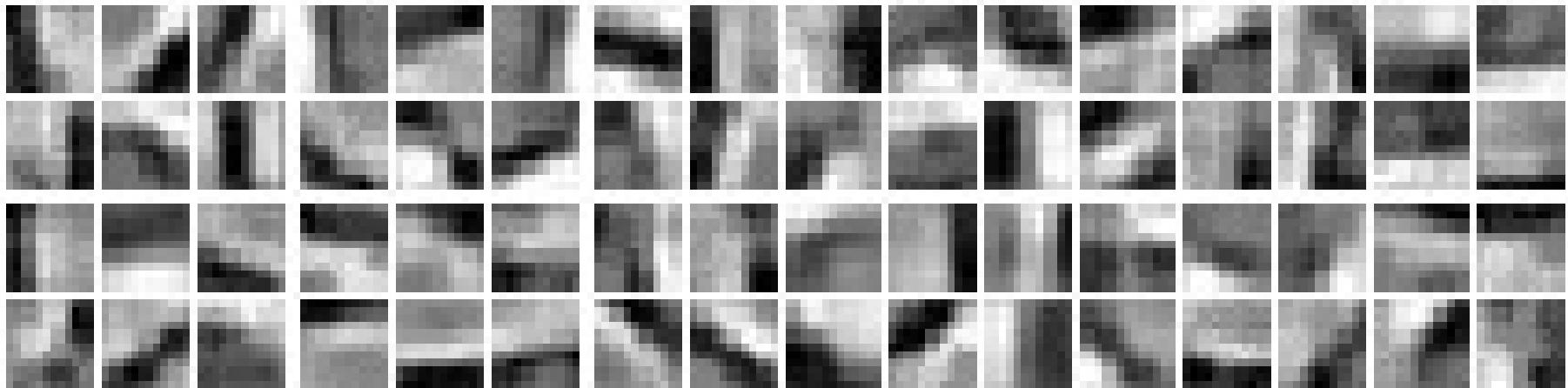
$$\Sigma \text{ ORIGINAL DIGIT} \approx \text{RECONSTRUCTION} = \text{ACTIVATED DECODER BASIS FUNCTIONS (in feed-back layer)}$$



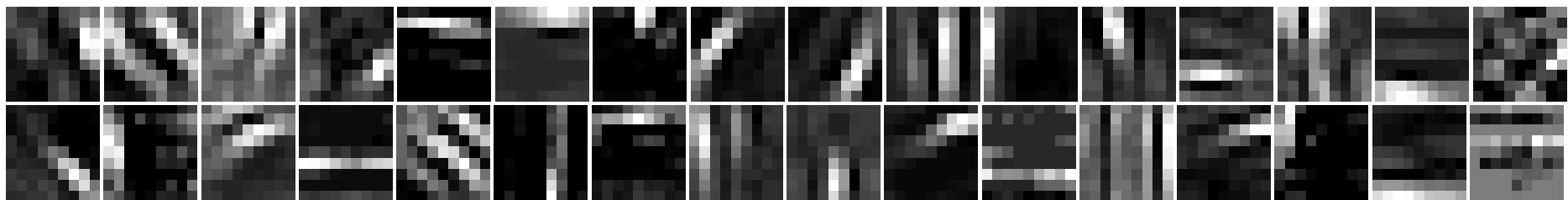
red squares: decoder bases

Sparse Enc/Dec on Object Images

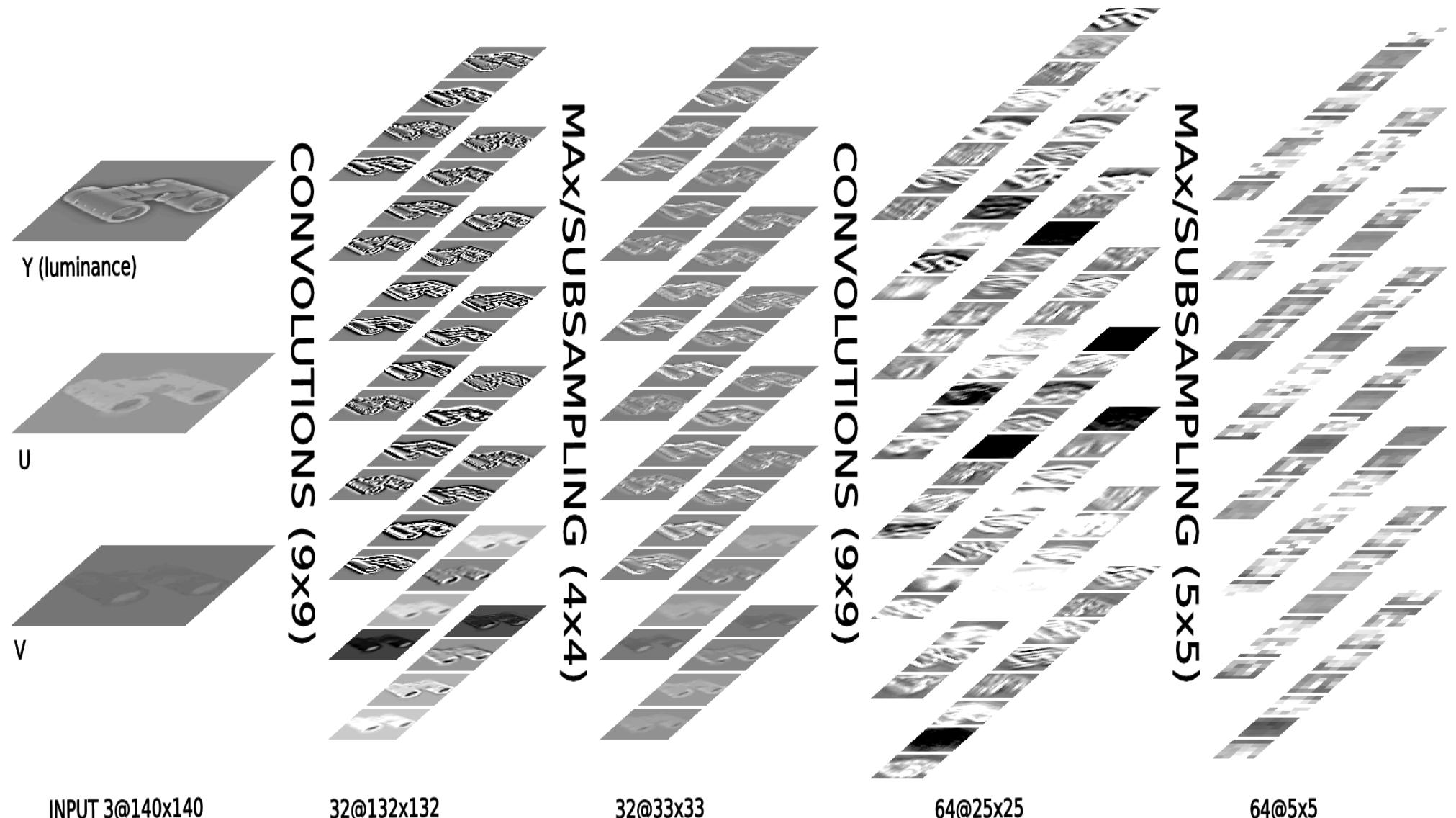
- 9x9 filters at the first level



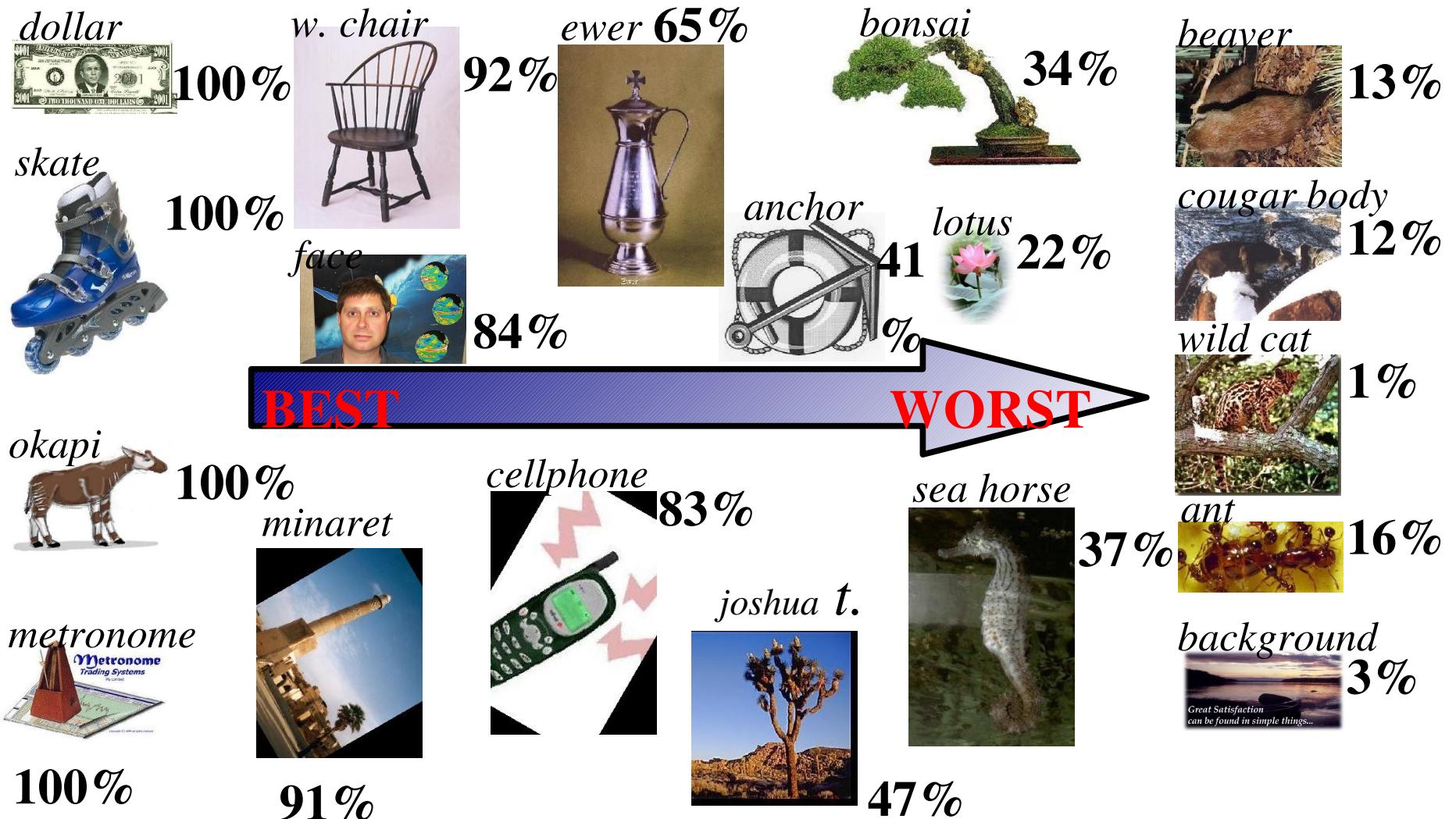
- 9x9 filters at the second level (like V4?)



Feature Hierarchy for Object Recognition



Recognition Rate on Caltech 101



LAGR: Learning Applied to Ground Robotics

- ➊ Getting a robot to drive autonomously in unknown terrain solely from vision (camera input).
- ➋ Our team (NYU/Net-Scale Technologies Inc.) is one of 8 participants funded by DARPA
- ➌ All teams received identical robots and can only modify the software (not the hardware)
- ➍ The robot is given the GPS coordinates of a goal, and must drive to the goal as fast as possible. The terrain is unknown in advance. The robot is run 3 times through the same course.

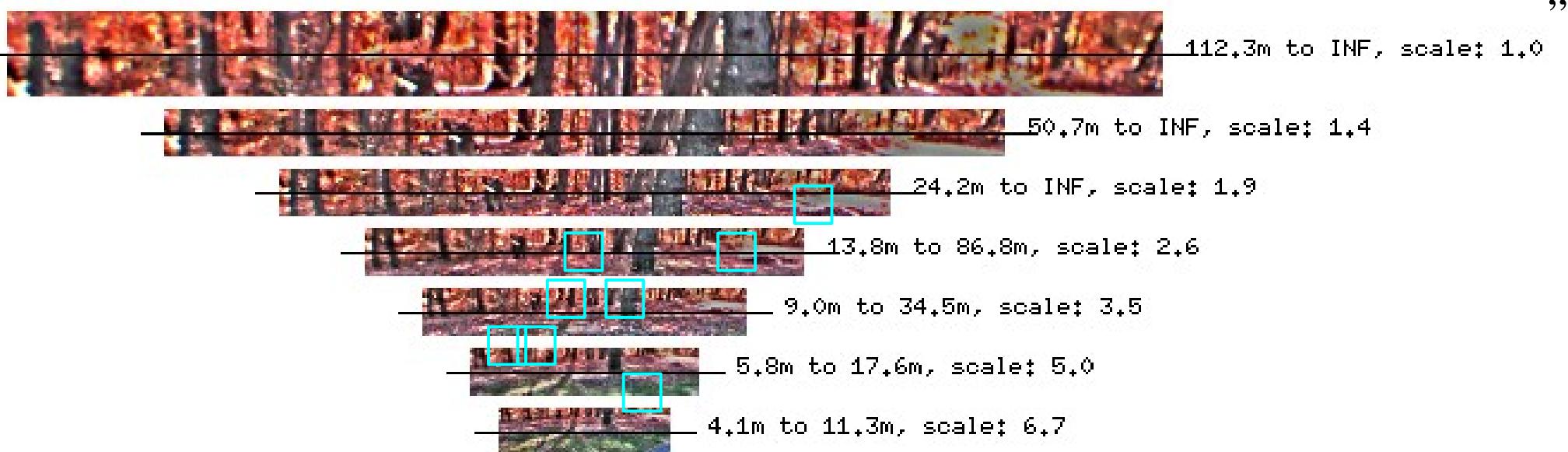


Long Range Vision: Distance Normalization



Pre-processing (125 ms)

- Ground plane estimation
- Horizon leveling
- Conversion to YUV + local contrast normalization
- Scale invariant pyramid of



Convolutional Net Architecture

- Operates on 12x25 YUV windows from the pyramid

100 features per

3x12x25 input window

Logistic regression 100 features -> 5 classes

100x1x1 input window

Convolutions with 6x5 kernels

20x6x5 input window

Pooling/subsampling with 1x4 kernels

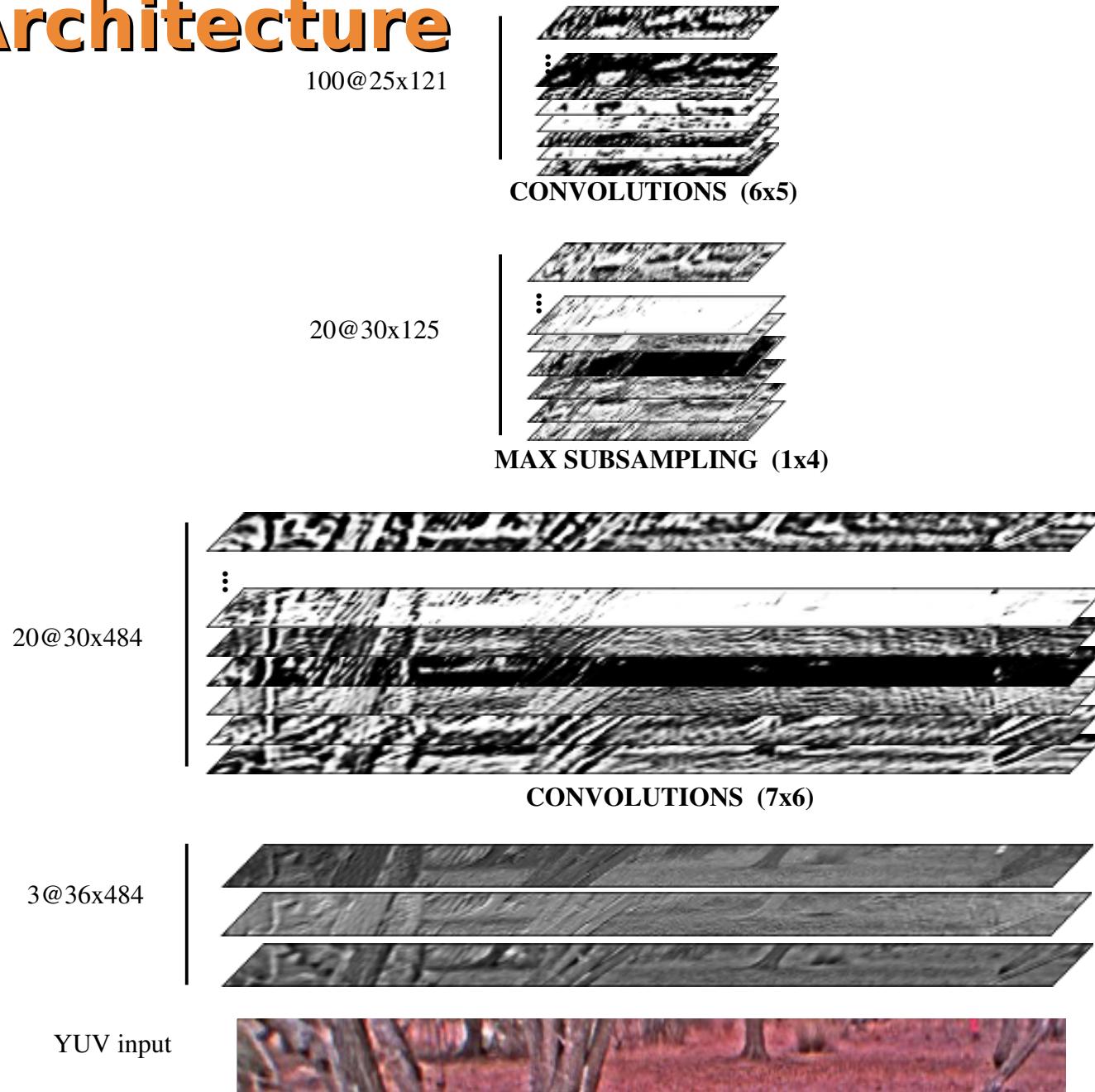
20x6x20 input window

Convolutions with 7x6 kernels

3x12x25 input window

YUV image band
20-36 pixels tall,
36-500 pixels wide

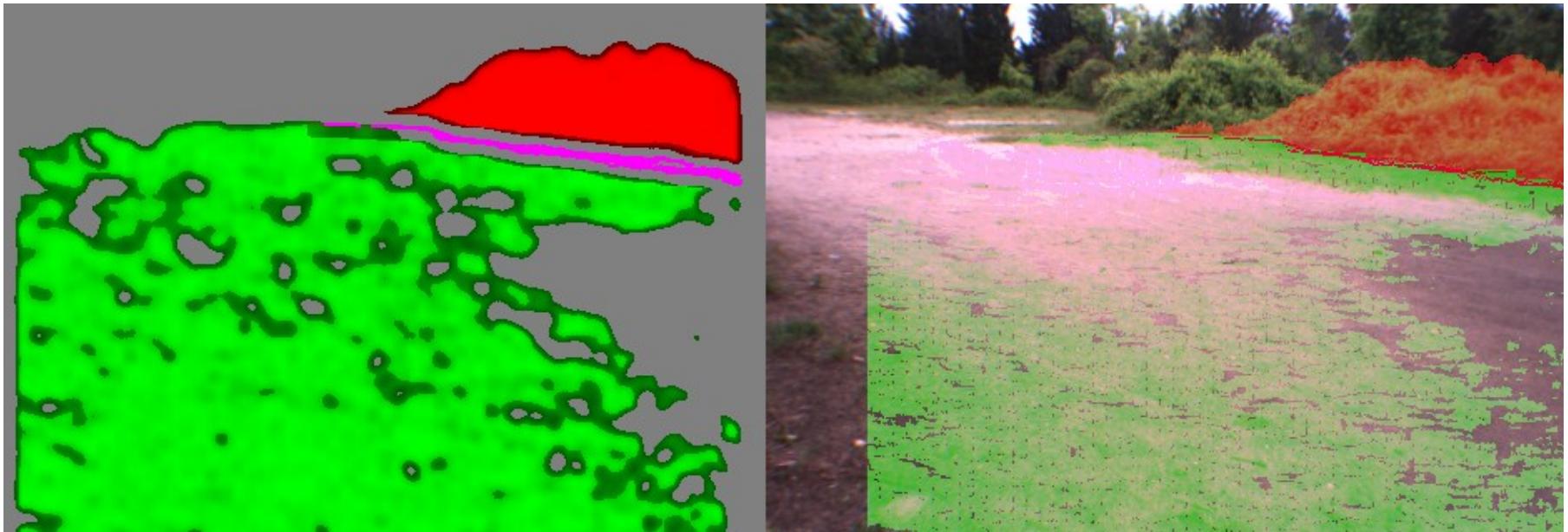
Convolutional Net Architecture



Long Range Vision: 5 categories

Online Learning (52 ms)

- Label windows using stereo information – 5 classes



super-ground



ground



footline



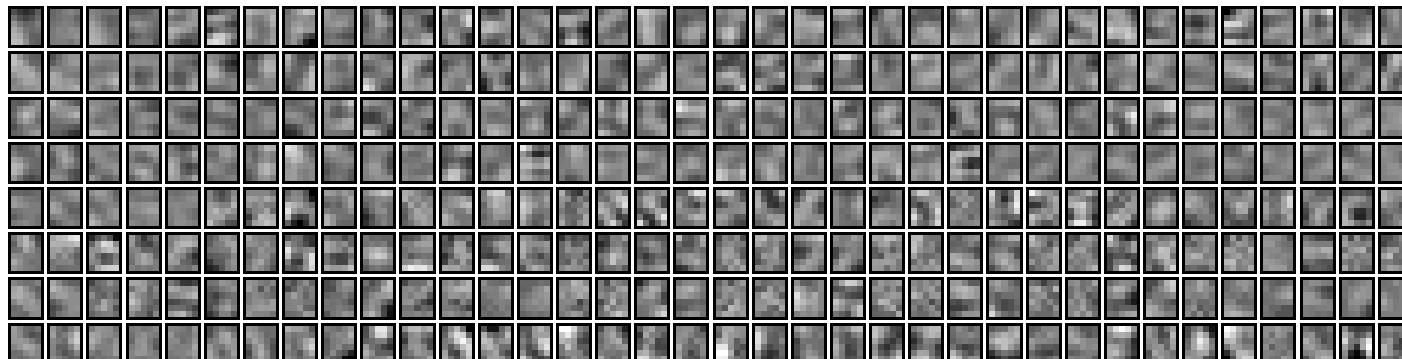
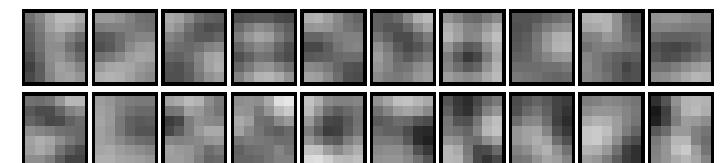
obstacle



super-obstacle

Trainable Feature Extraction

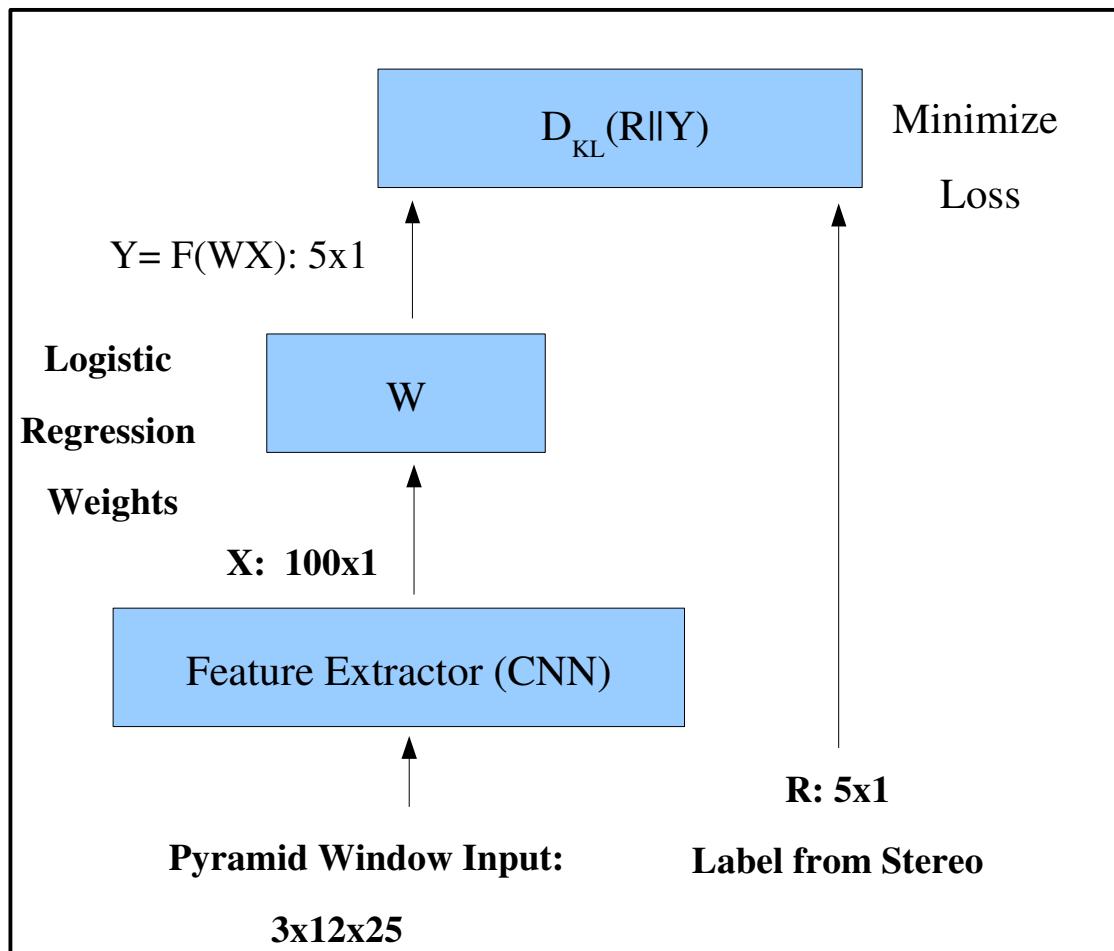
- “Deep belief net” approach to unsupervised feature learning
- Two stages are trained in sequence
 - each stage has a layer of convolutional filters and a layer of horizontal feature pooling.
 - Naturally shift invariant in the horizontal direction
- Filters of the convolutional net are trained so that the input can be reconstructed from the features
 - 20 filters at the first stage (layers 1 and 2)
 - 300 filters at the second stage (layers 3 and 4)
- Scale invariance comes from pyramid.
 - for near-to-far generalization



Long Range Vision: the Classifier

Online Learning (52 ms)

- Train a logistic regression on every frame, with cross entropy loss function

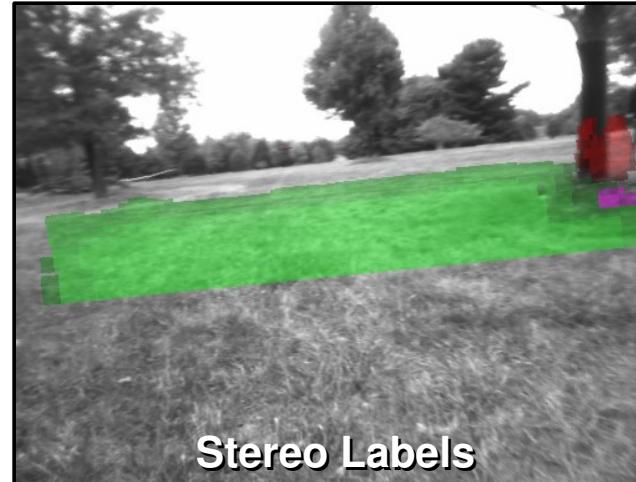


- 5 categories are learned
- 750 samples of each class are kept in a ring buffer: short term memory.
- Learning “snaps” to new environment in about 10 frames
- Weights are trained with stochastic gradient descent
- Regularization by decay to default weights

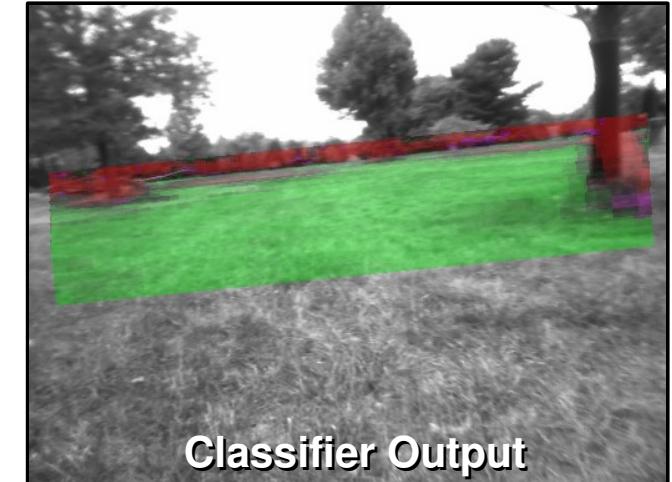
Long Range Vision Results



Input image



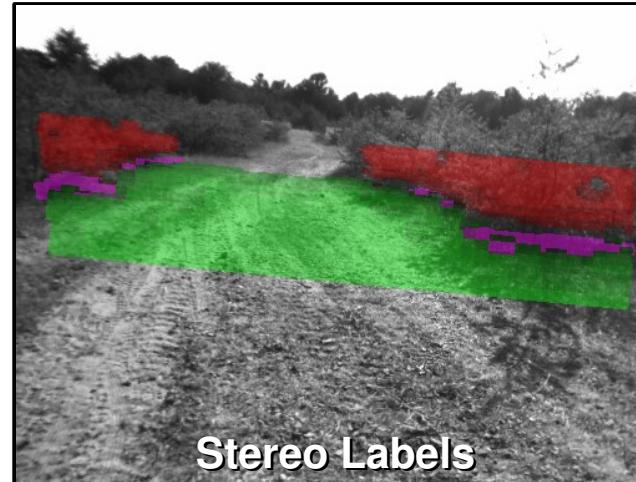
Stereo Labels



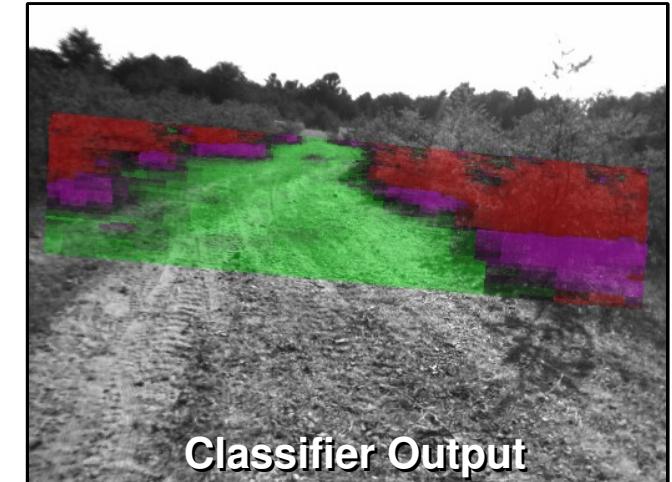
Classifier Output



Input image



Stereo Labels

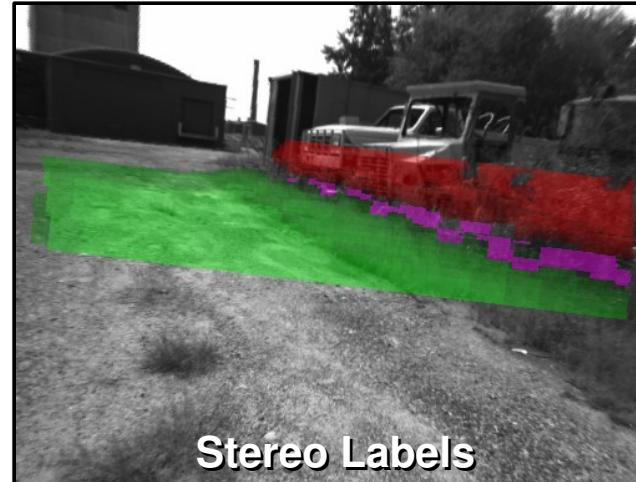


Classifier Output

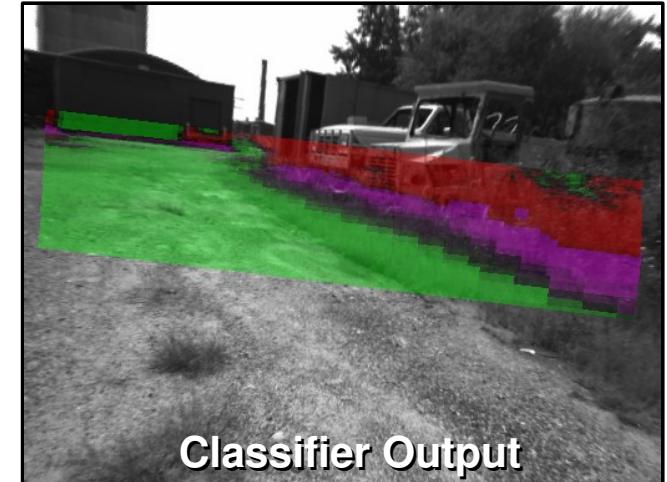
Long Range Vision Results



Input image



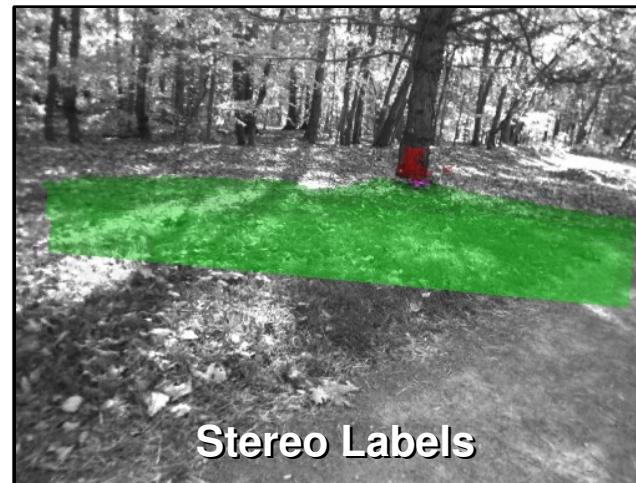
Stereo Labels



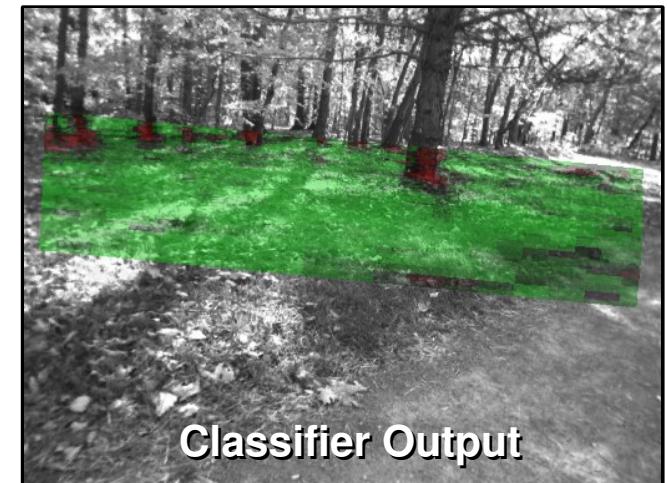
Classifier Output



Input image



Stereo Labels

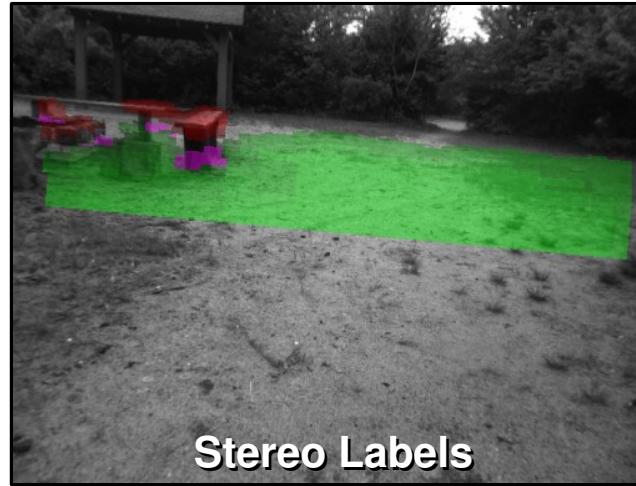


Classifier Output

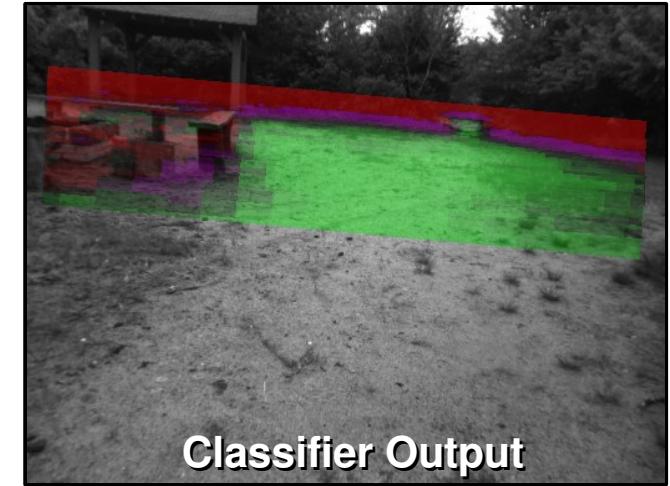
Long Range Vision Results



Input image



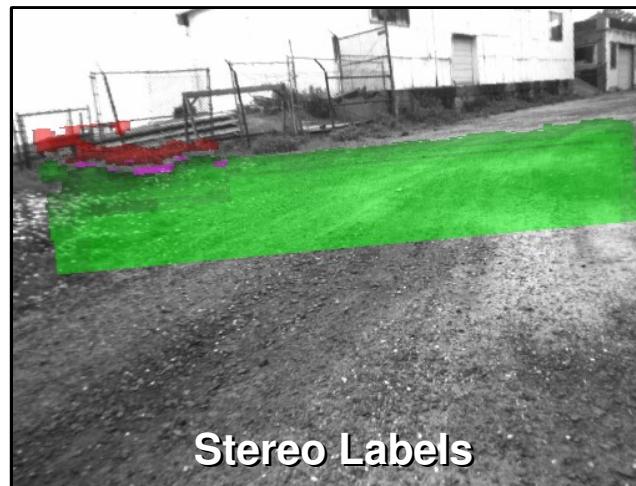
Stereo Labels



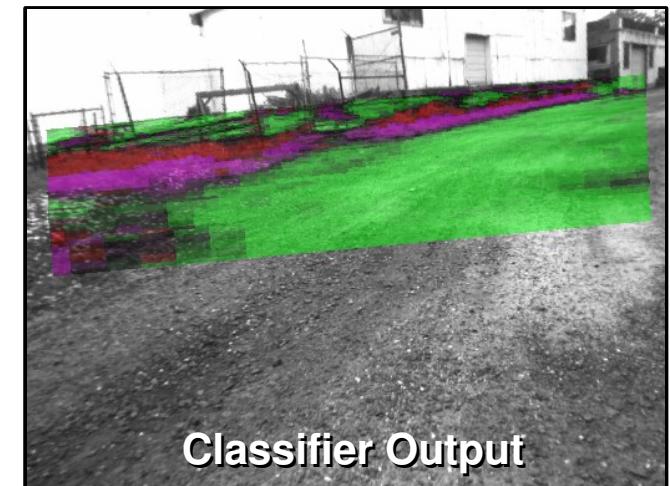
Classifier Output



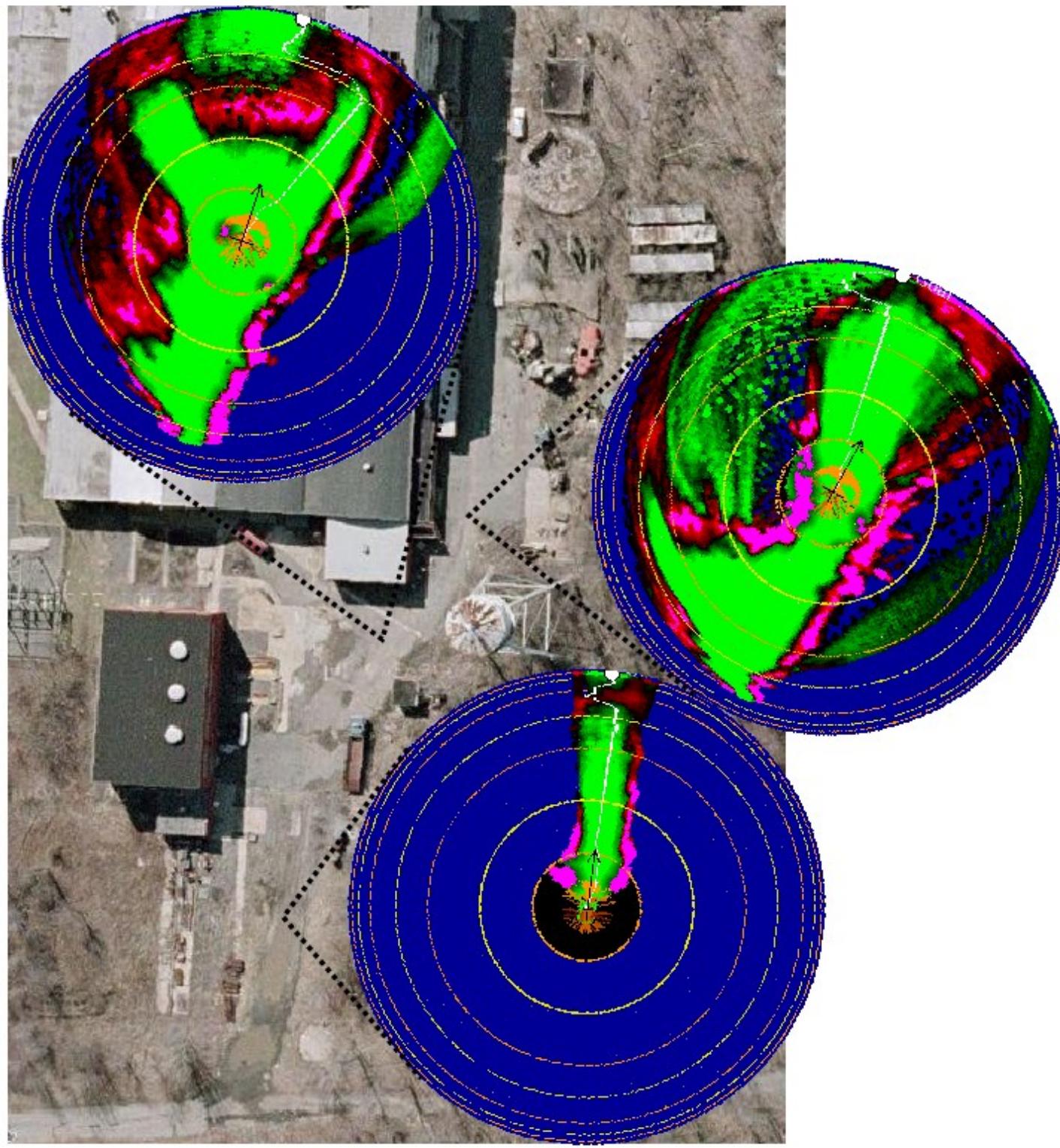
Input image

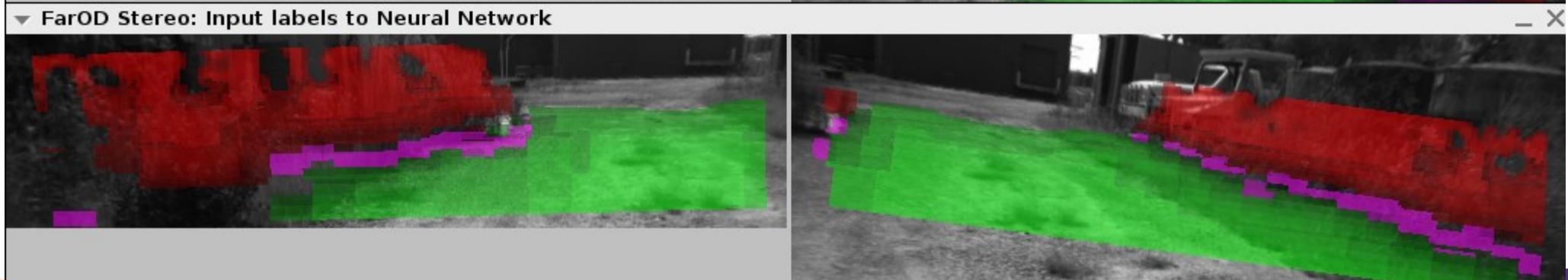
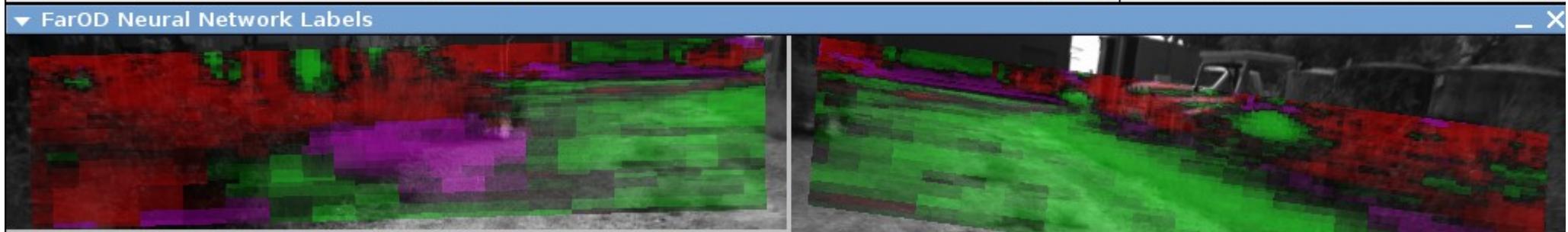
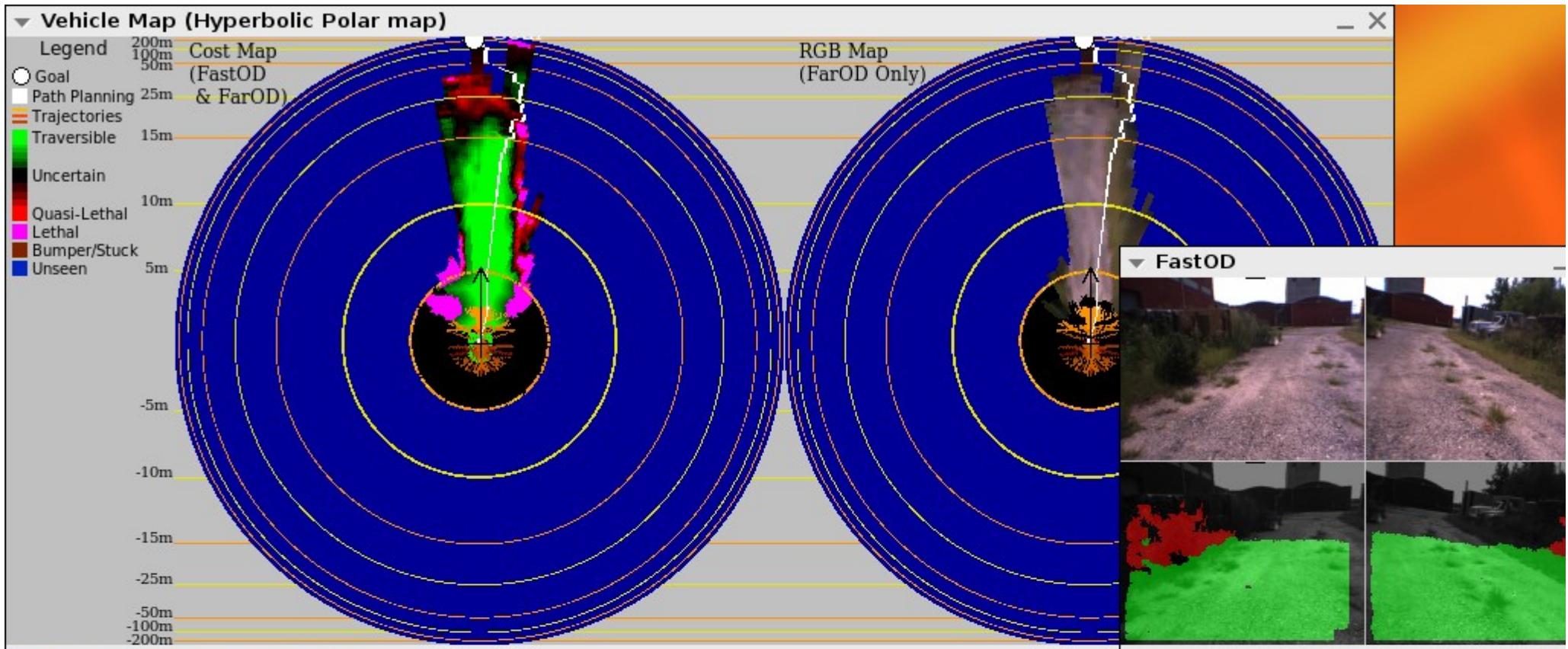


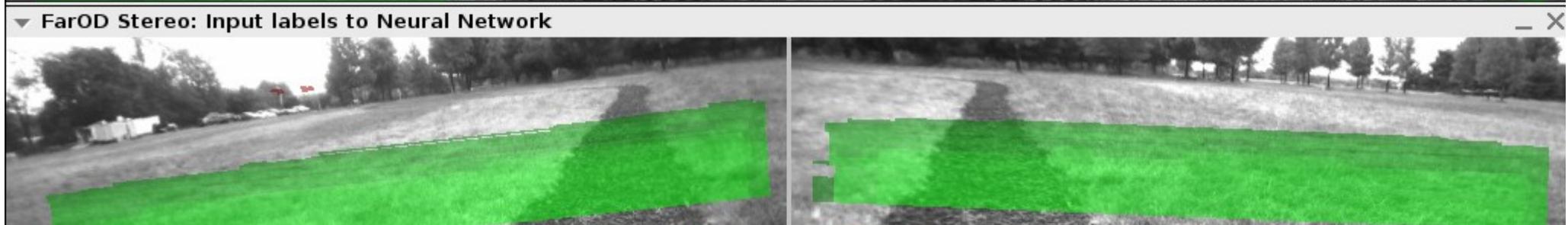
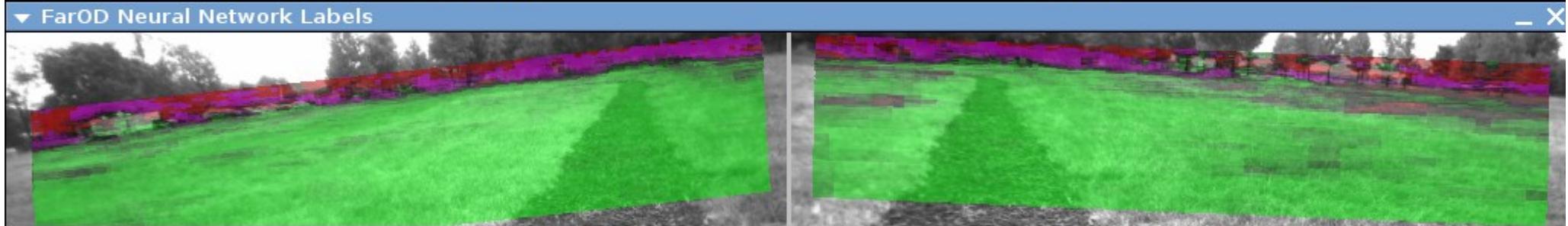
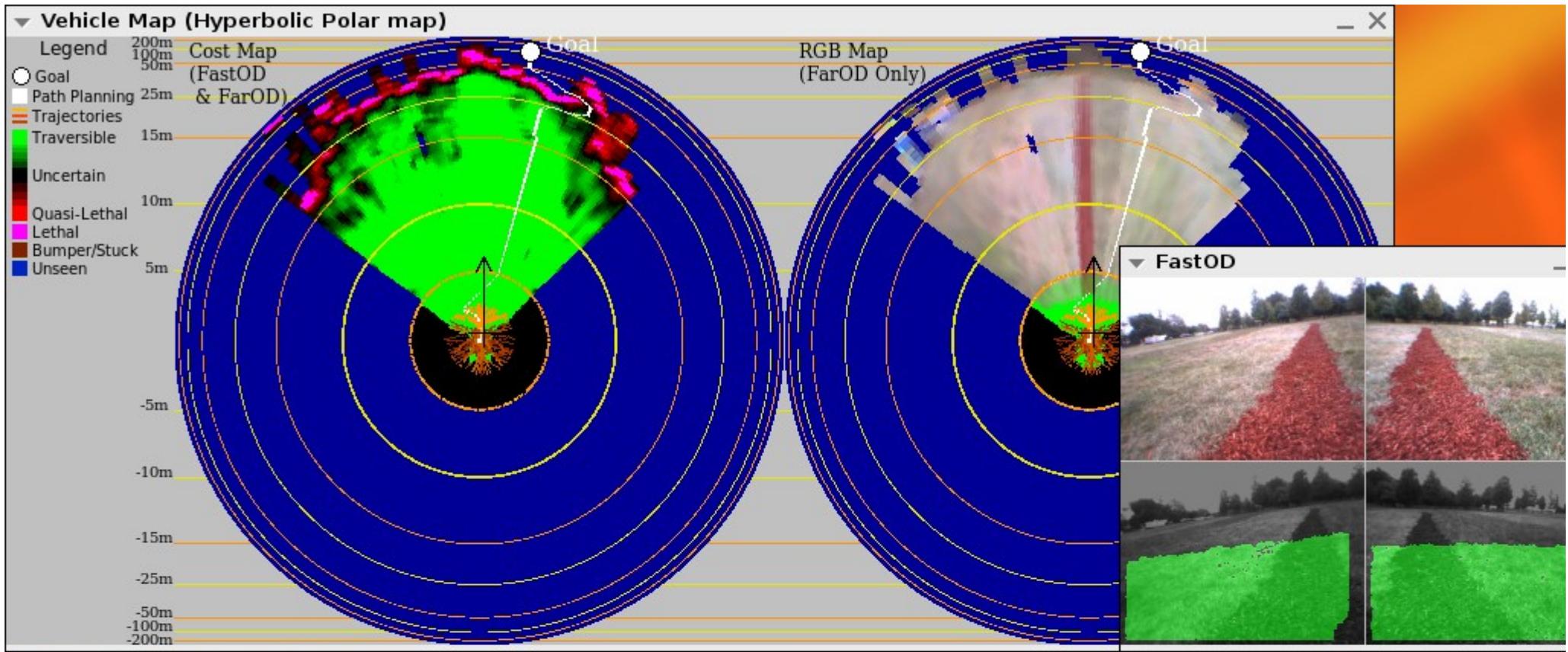
Stereo Labels

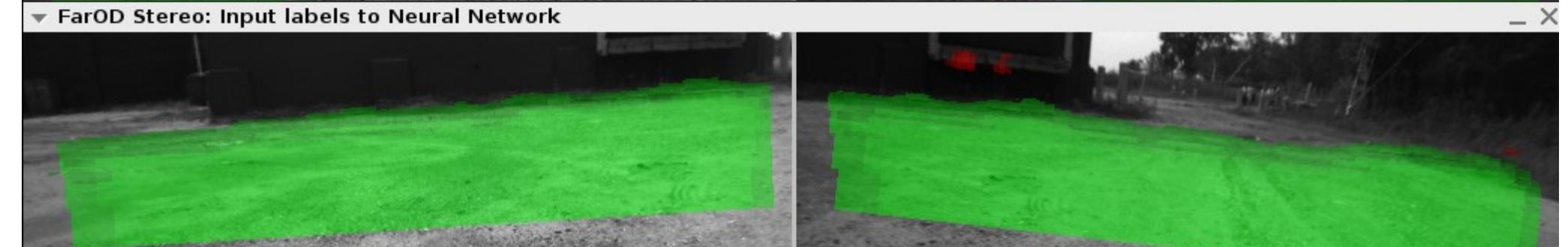
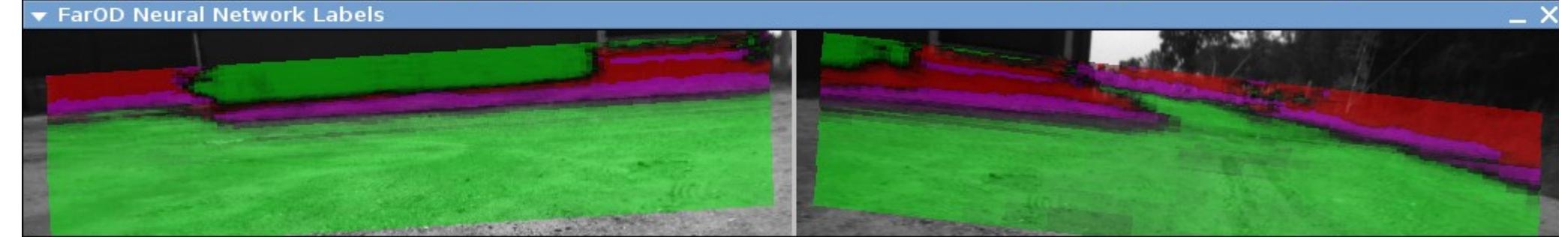
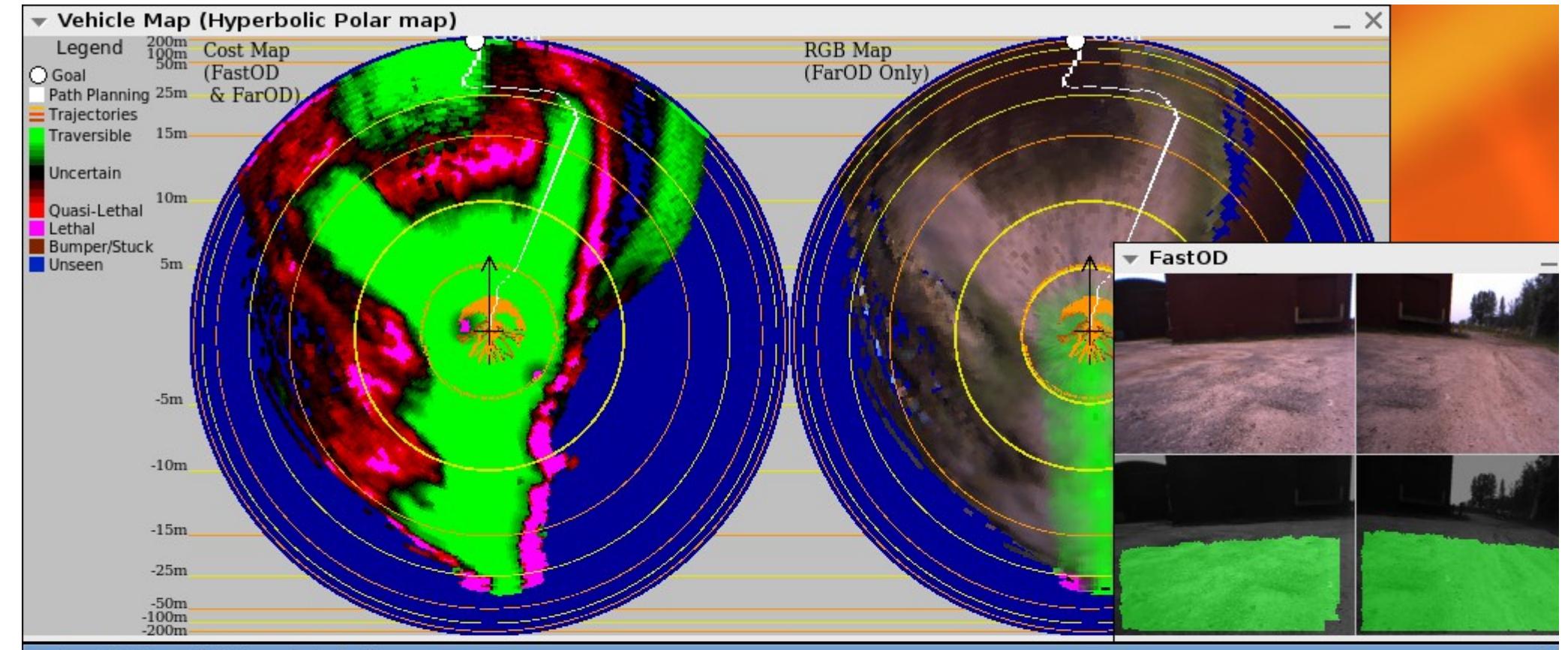


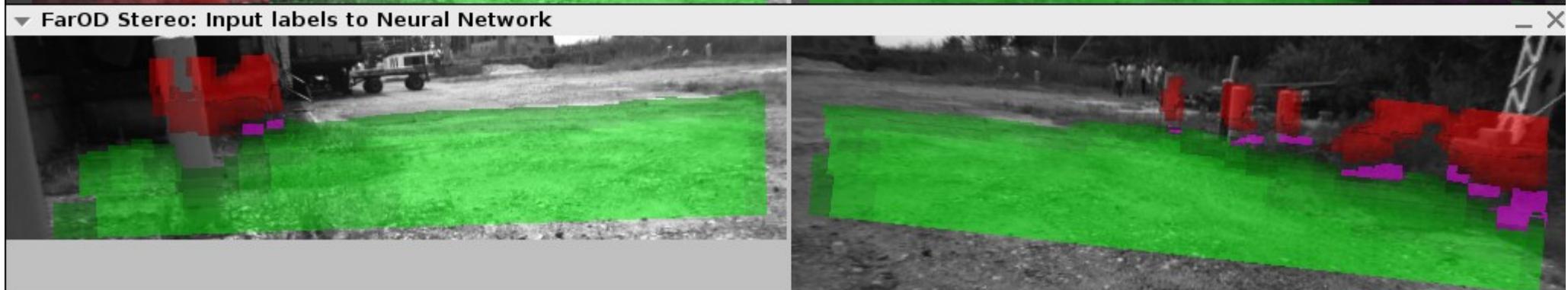
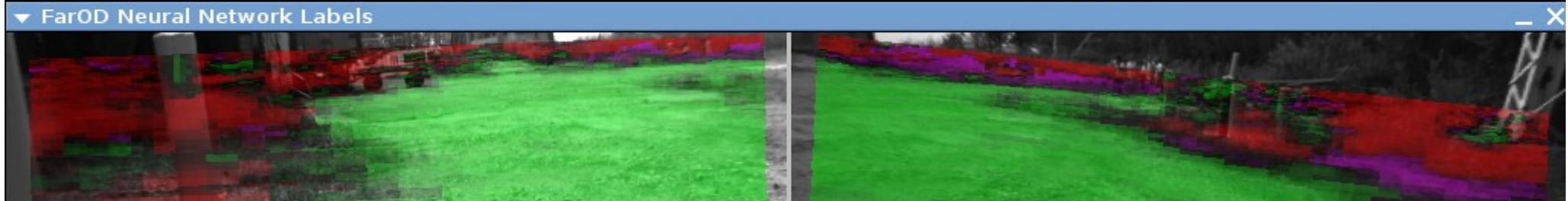
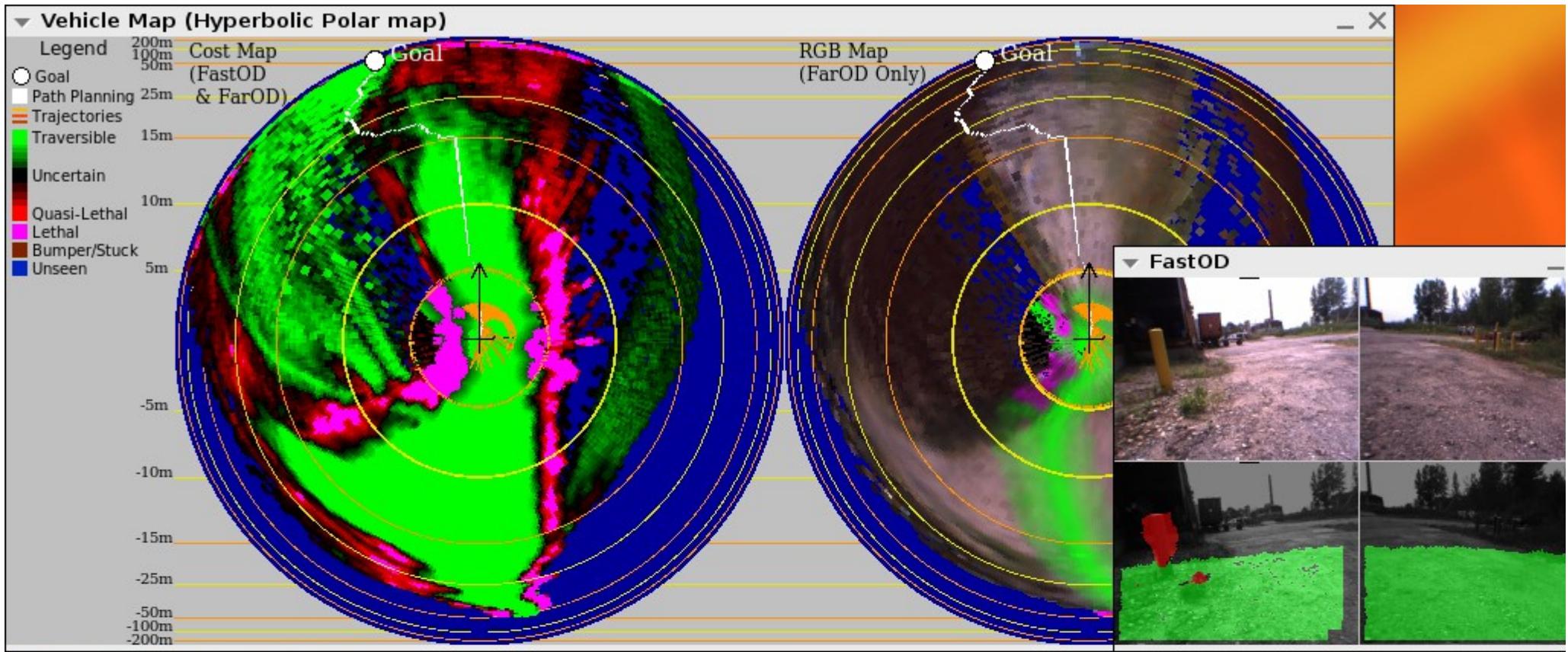
Classifier Output

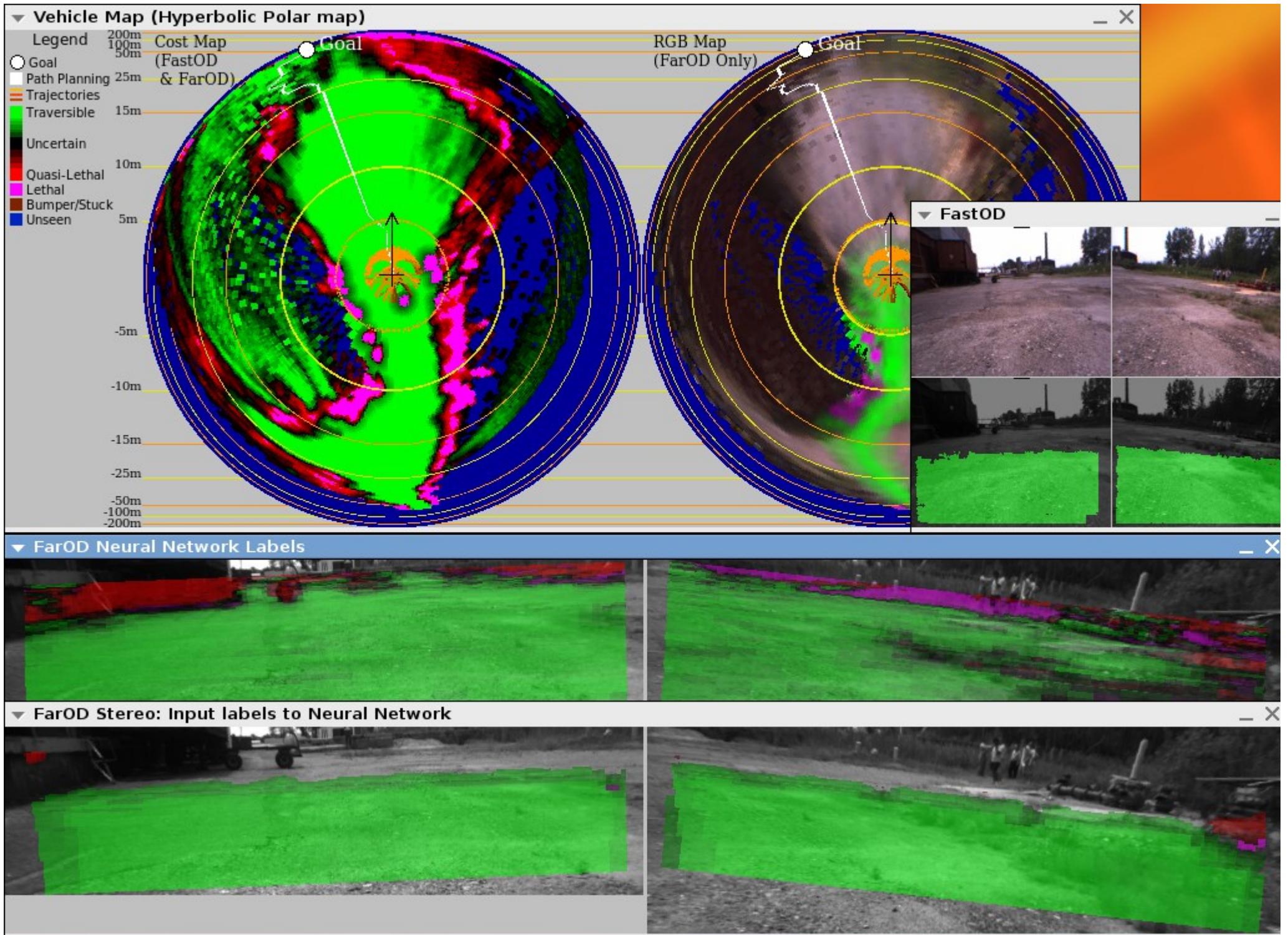




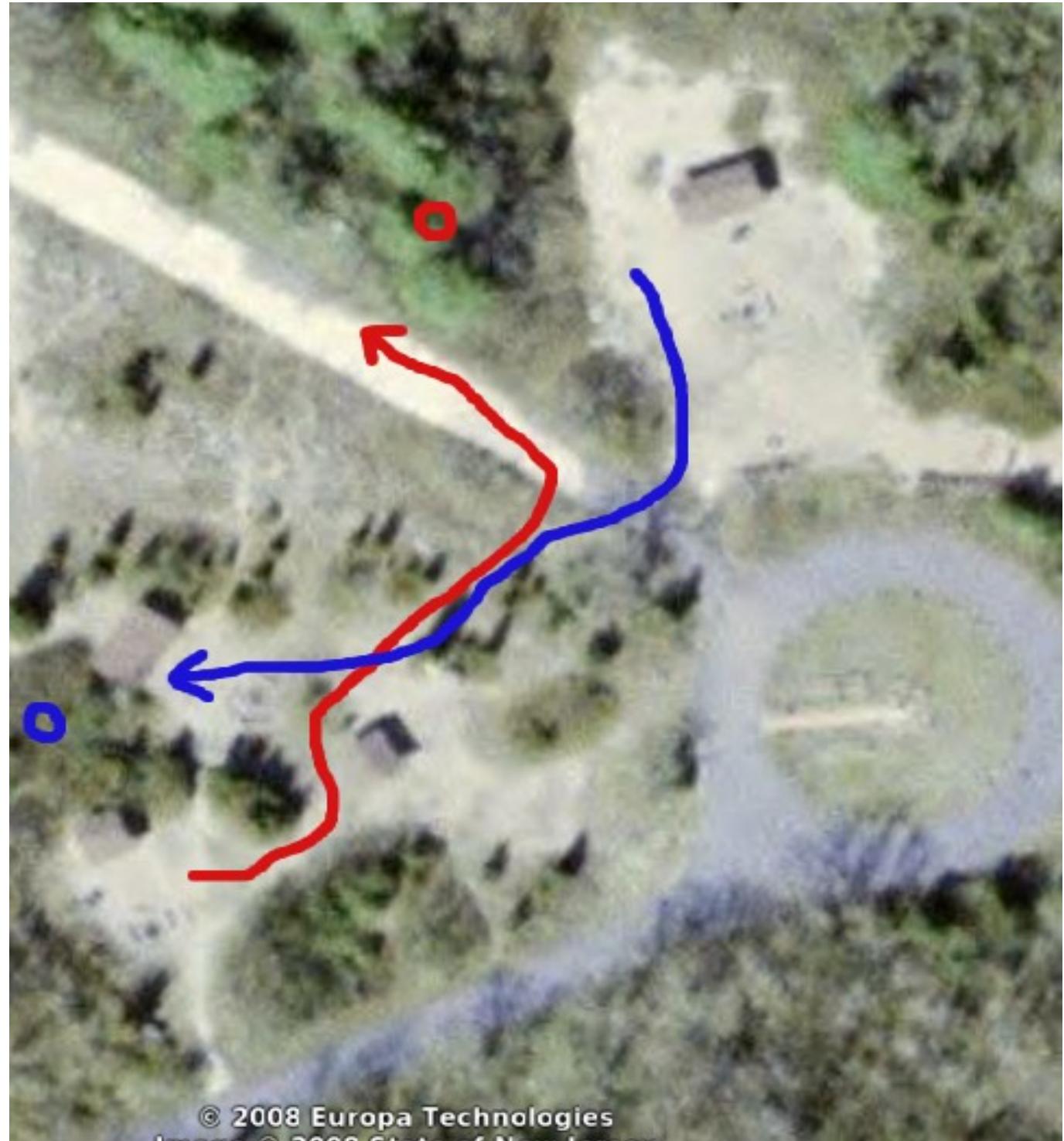








Videos



Application of Stacked Auto-Encoders to Text Retrieval

■ [Ranzato et al. ICML 08]: semi-supervised stacked auto-encoders

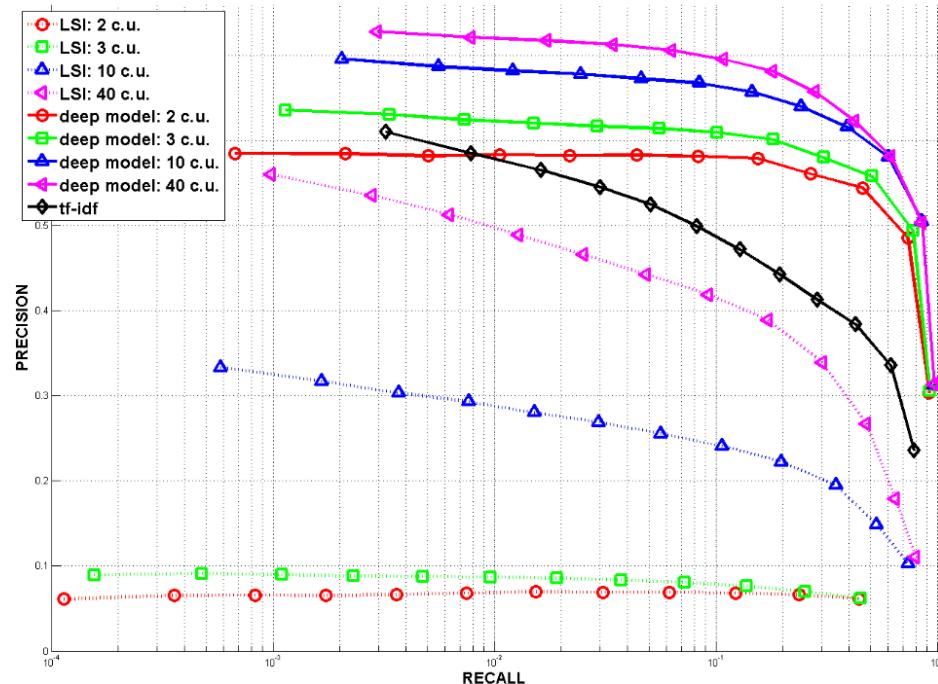


Figure 4. Precision-recall curves of the Reuters dataset comparing a linear model (LSI) to the non-linear deep model with the same number of code units (c.u.). Retrieval is done using the k most similar documents according to cosine similarity, with $k \in [1 \dots 2047]$.

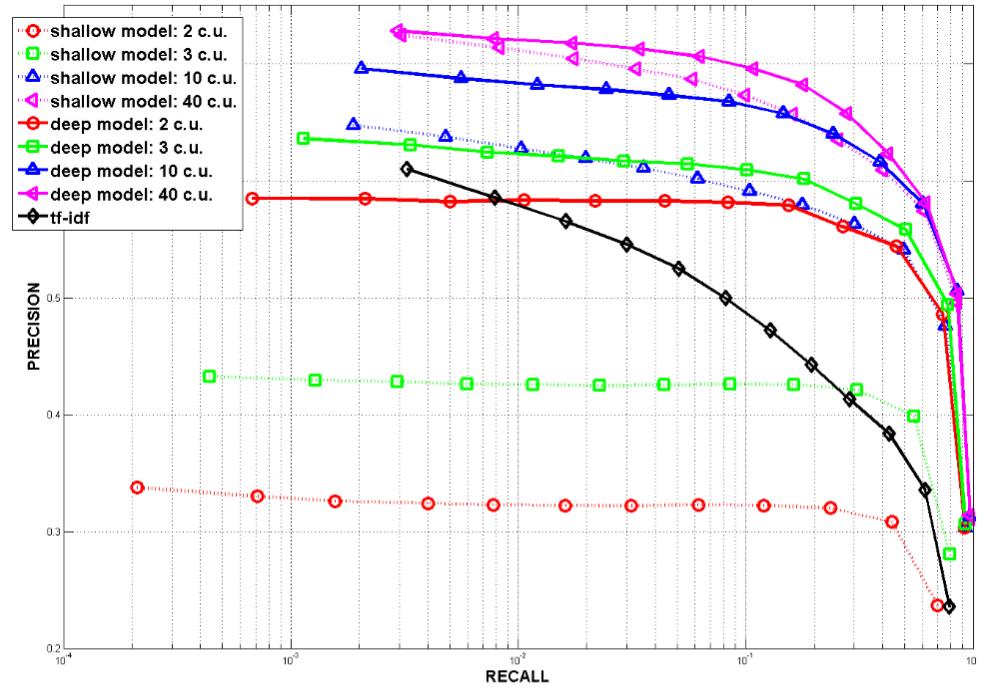


Figure 5. Precision-recall curves of the Reuters dataset comparing the model trained with only one layer (shallow architecture) to a deep model with the same number of code units. The deep model outperforms the shallow one overall when the features are extremely compact.

Application of Stacked Auto-Encoders to Text Retrieval

■ Ranzato et al. ICML 08

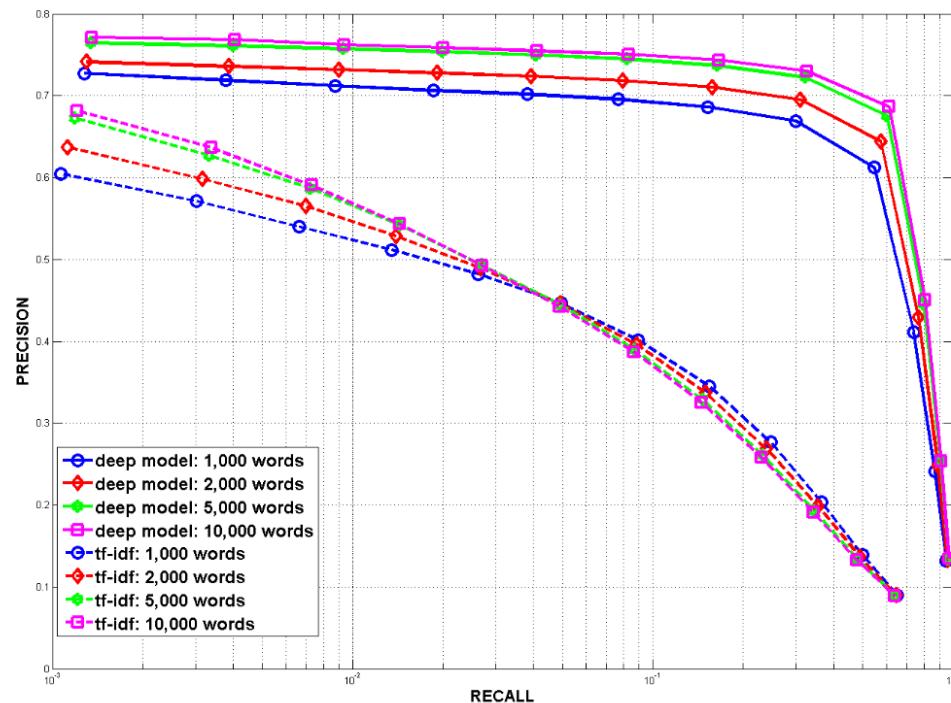


Figure 6. Precision-recall curves of the 20 Newsgroups dataset comparing the performance of the model (1 layer) trained on documents with various number of words in the dictionary (from 1000 to 10000).

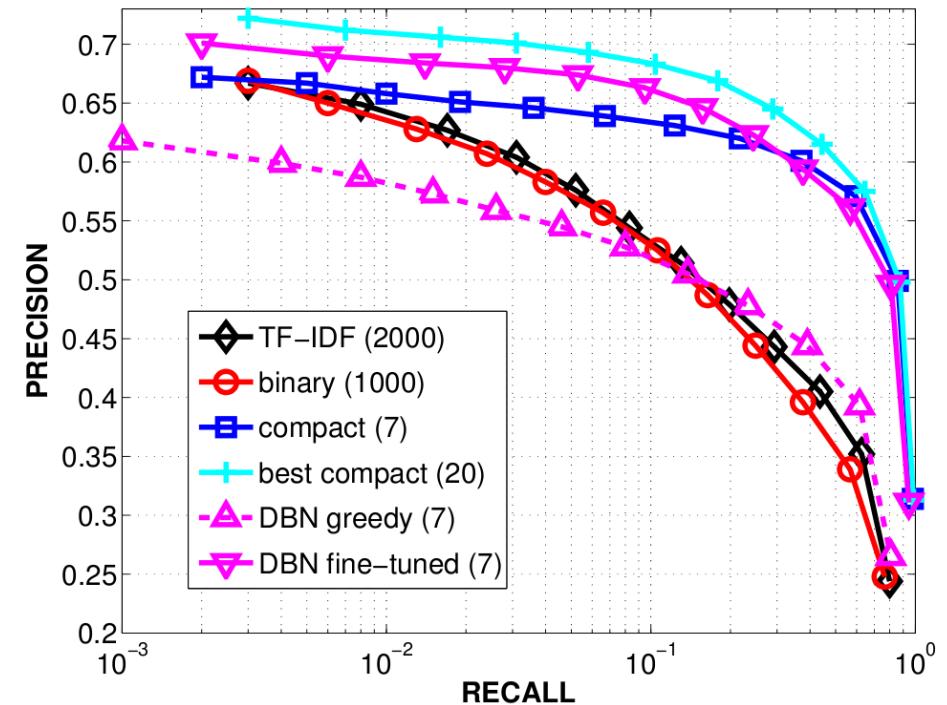


Figure 7. Precision-recall curves using very compact representations and high dimensional binary representations. Compact representations can achieve better performance using less memory and CPU time.

Application of Stacked Auto-Encoders to Text Retrieval

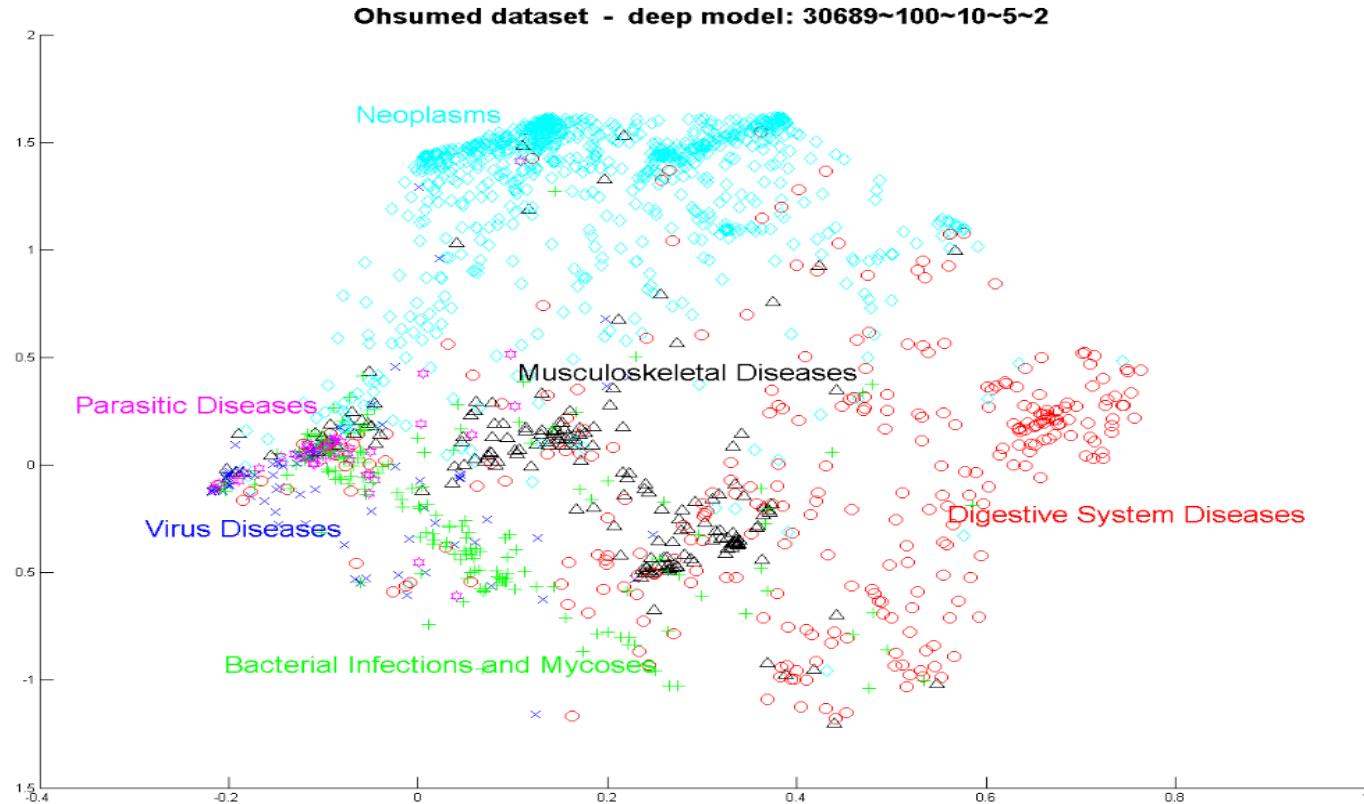


Figure 8. The two-dimensional codes produced by the deep model trained on the Ohsmed dataset (shown only the 6 most numerous classes). The codes have been computed by propagating test documents through the 4-layer network.



The End

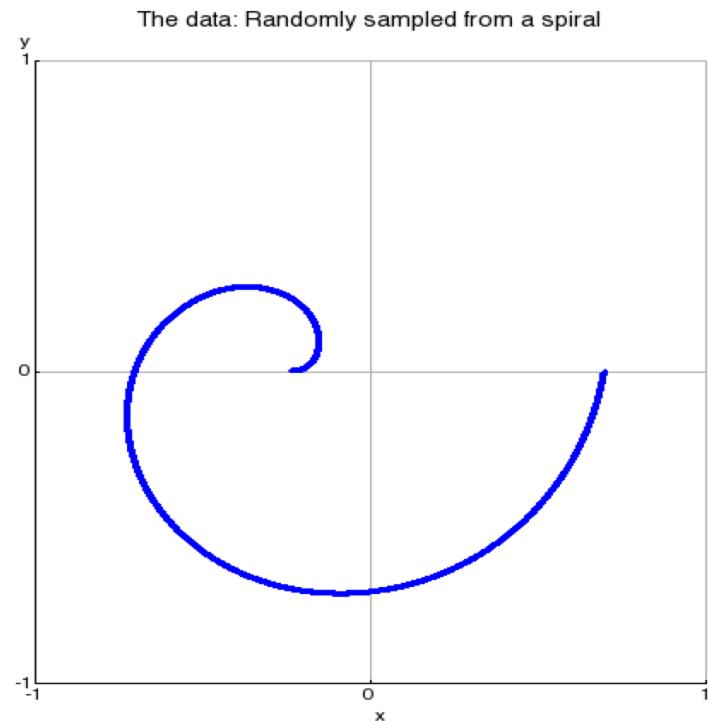
Example: A Toy Problem. The Spiral

- ◆ Dataset

- ◆ 10,000 random points along a spiral in a 2D plane
- ◆ The spiral fits in a square with opposite corners $(-1,1)$, $(1,-1)$
- ◆ The spiral is designed so that no function can predict a single value of f from

- ◆ Goal

- ◆ Learn an energy surface with low energies along the spiral and high energy everywhere else



PCA (Principal Component Analysis)

- Can be seen as encoder-decoder architecture that minimizes mean square reconstruction error (energy loss)
- Optimal code is constrained to be equal to the value predicted by encoder
- Flat surfaces are avoided by using a code of low dimension

$$Enc(Y) = WY$$

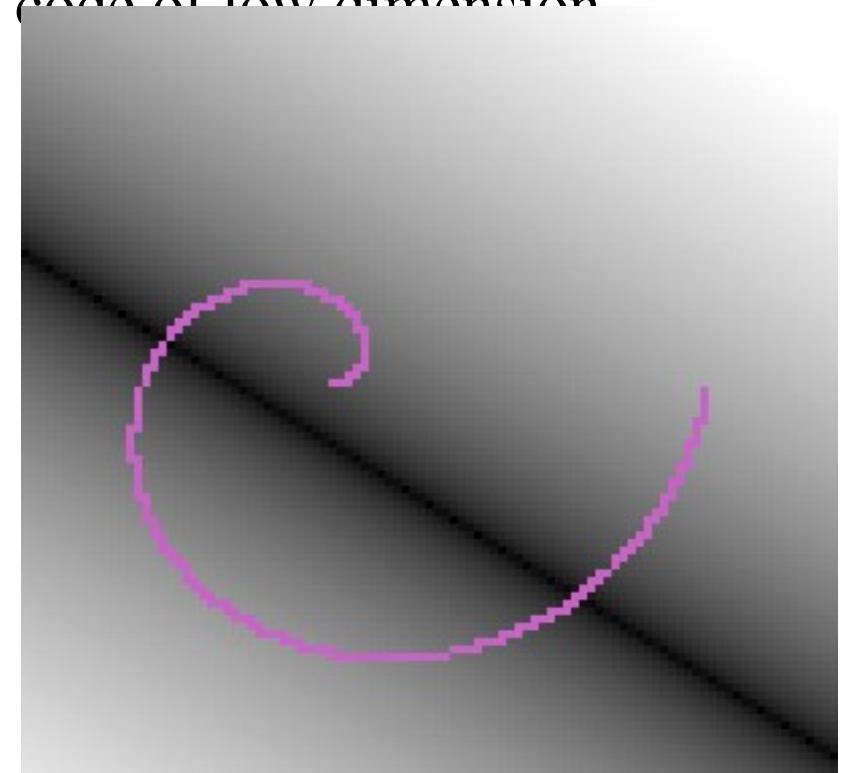
$$Dec(Z) = W^T Y, \text{ where } W \in \Re^{N \times M}$$

$$C_e(Y, Z) = \|WY - Z\|^2$$

$$C_d(Y, Z) = \|W^T Z - Y\|^2$$

- For large value of γ energy reduces to

$$E(Y, W) = \|W^T WY - Y\|^2$$

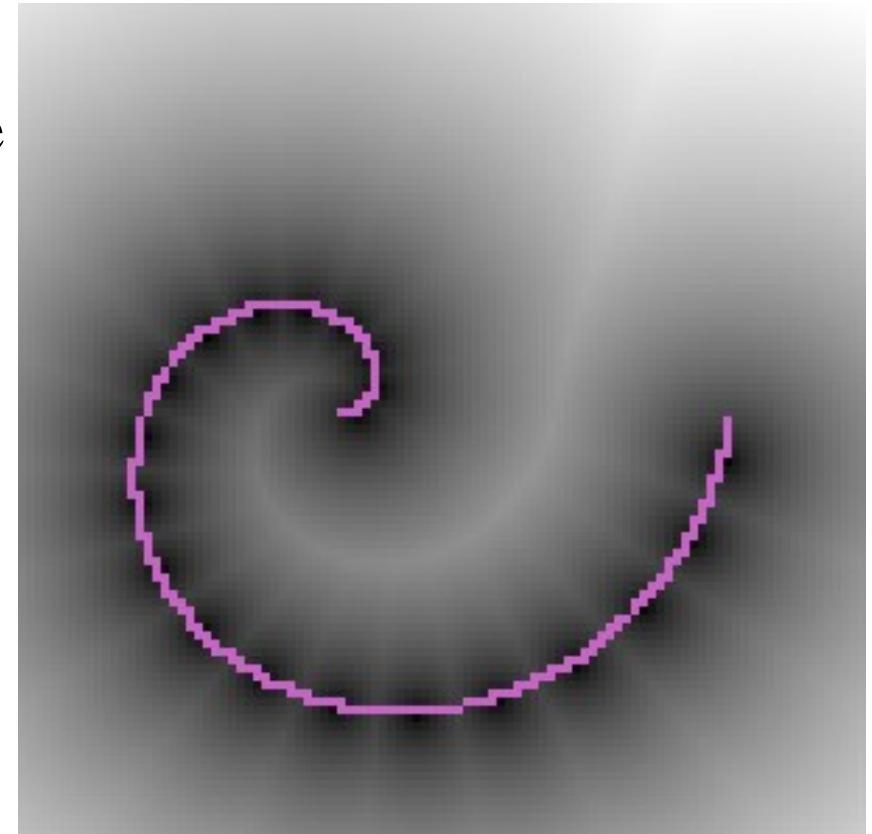


K-Means

- In this architecture the code Z is a binary vector of size N (N being the number of prototypes)
- For any sample Y , only one component is active (equal to 1) and all the others are inactive (equal to 0).
- This is a one-of- N sparse binary code
- The energy is:

$$E(Y, Z) = \sum_i Z_i \|Y - W_i\|^2$$

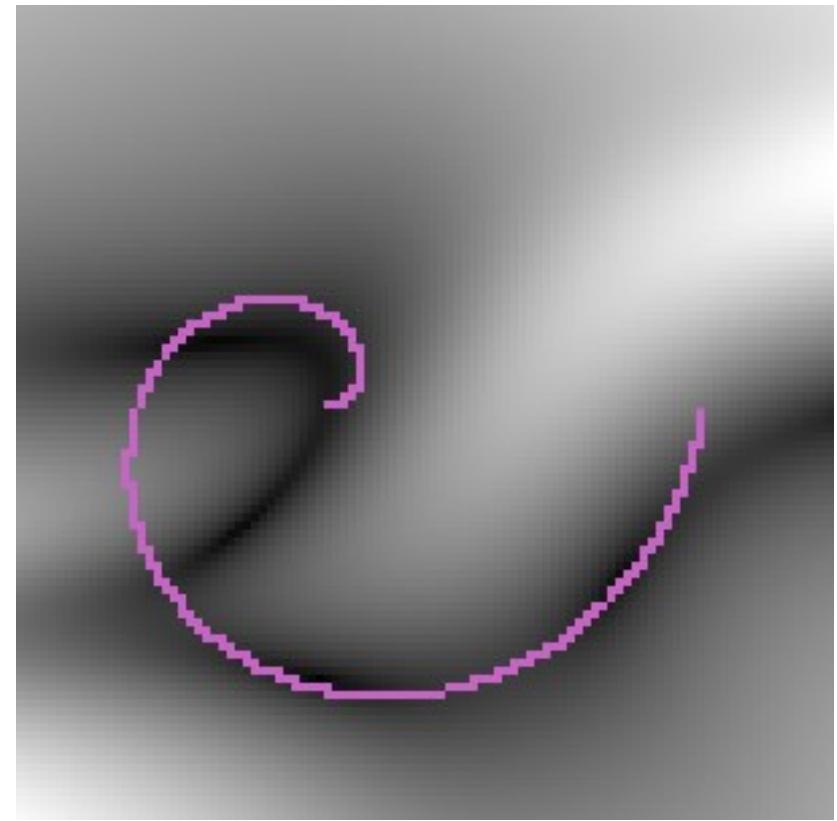
W_i is the i^{th} prototype



- Inference involves finding Z that minimizes the energy

Auto-encoder neural net (2-100-1-100-2 architecture)

- ◆ A neural network autoencoder that learns a low dimensional representation.
- ◆ Architecture used
 - ◆ Input layer: 2 units
 - ◆ First hidden layer: 100 units
 - ◆ Second hidden layer (the code): 1 unit
 - ◆ Third hidden layer: 100 units
 - ◆ Output layer: 2 units
- ◆ Similar to PCA but non-linear
- ◆ Energy is
$$E(Y) = |Dec(Enc(Y)) - Y|^2$$

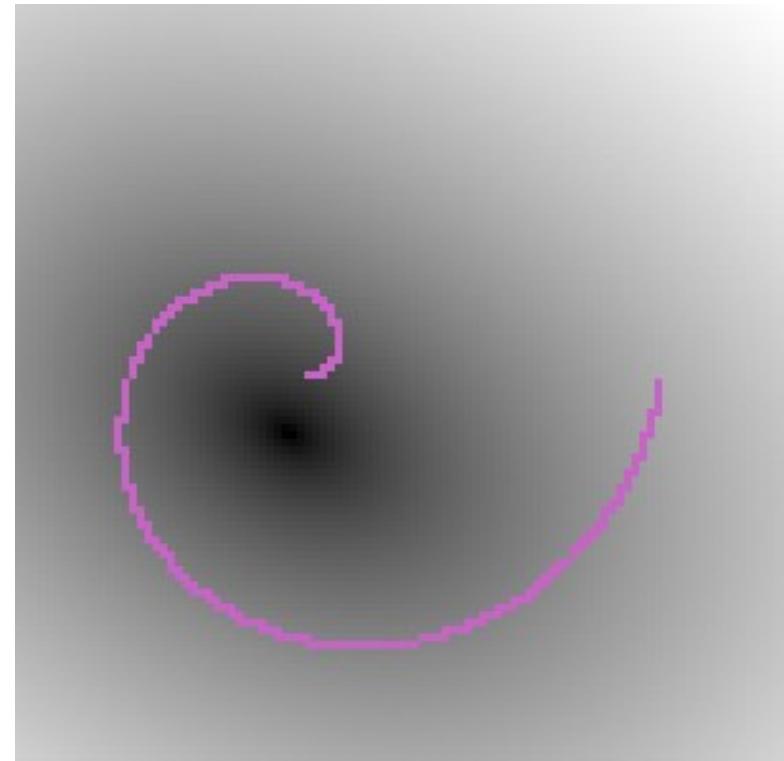


Wide auto-encoder neural net with energy loss

- ◆ The energy loss simply pulls down on the energy of the training samples (no contrastive term).
- ◆ Because the dimension of the code is larger than the input, nothing prevents the architecture from learning the identity function, which gives a very flat energy surface (a collapse): everything is perfectly reconstructed.
- ◆ Simplest example: a multi layer neural network with identical input and output layers and a large hidden layer.
- ◆ **Architecture used**
 - ◆ Input layer: 2 units
 - ◆ Hidden layer (the code): 20 units
 - ◆ Output layer: 2 units
- ◆ Energy loss leads to a collapse
- ◆ Tried a number of loss functions

Wide auto-encoder with negative-log-likelihood loss

- ♦ Negative Log Likelihood Loss
 - ♦ Pull down on the energy of training (observed) samples
 - ♦ Pull up on the energies of all the other (unobserved) samples
 - ♦ Approximate the log of partition function through dense sampling.
 - ♦ Energy surface is very “stiff” because of small number of parameters.
 - ♦ Hence the energy surface is not perfect.

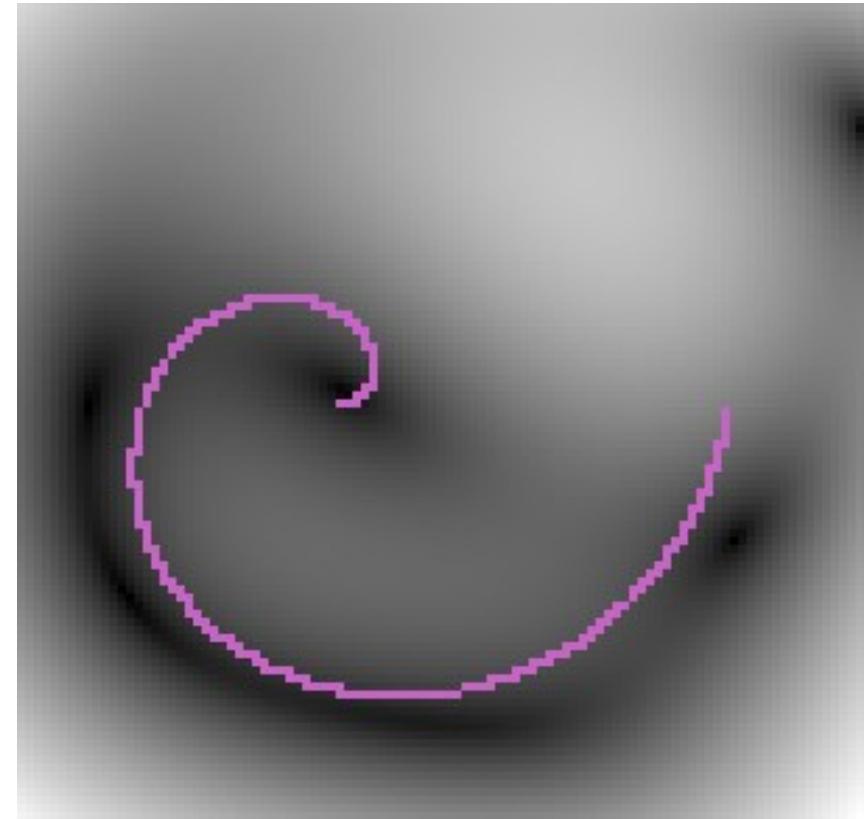


Wide auto-encoder with energy-based contrastive loss

- ♦ Linear-Linear Contrastive Loss

- ♦ Avoid the cost associated with minimizing negative log likelihood
- ♦ Idea is to pull up on unobserved points in the vicinity of training samples
- ♦ We use Langevin dynamics to generate such points

$$\bar{Y} \leftarrow \bar{Y} - \eta \frac{\delta E(\bar{Y})}{\delta Y} + \epsilon$$



- ♦ The loss function is

$$L(Y, W) = \alpha E(Y, W) + \max(0, m - E(\bar{Y}, W))$$

Wide auto-encoder with sparse code

- ♦ Sparse Codes
 - ♦ Limiting the information content of the code prevents flat energy surfaces, without the need to explicitly push up the bad points
 - ♦ Idea is to make the high dimensional code sparse by forcing each variable to be zero most of the time

