

# **(Unsupervised) Deep Learning. Learning Feature Hierarchies**

**Yann LeCun,**

**The Courant Institute of Mathematical Sciences**

**New York University**

**collaborators: Marc'Aurelio Ranzato,**

**Koray Kavakcuoglu, Y-Lan Boureau**

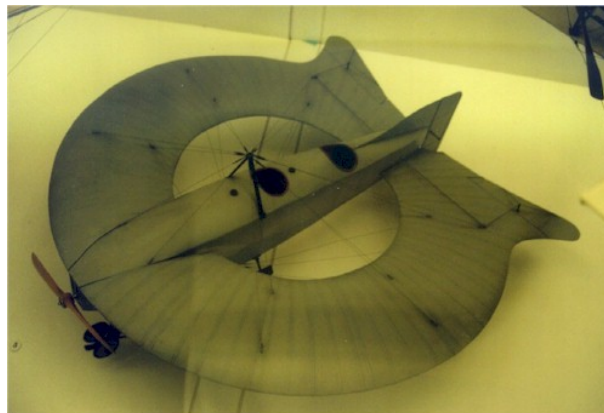
# Challenges of Visual Neuroscience (and Computer Vision)

## 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?

## 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)

## ● 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?).

## ● 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?

## ● 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?).

# Learning Deep Feature Hierarchies

## • The visual system is deep, and is learned

- ▶ How do we learn deep hierarchies of invariant features?

## • On recognition tasks **with lots of training samples**, deep supervised architecture outperform shallow architectures in speed and accuracy

## • 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]

## • 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]

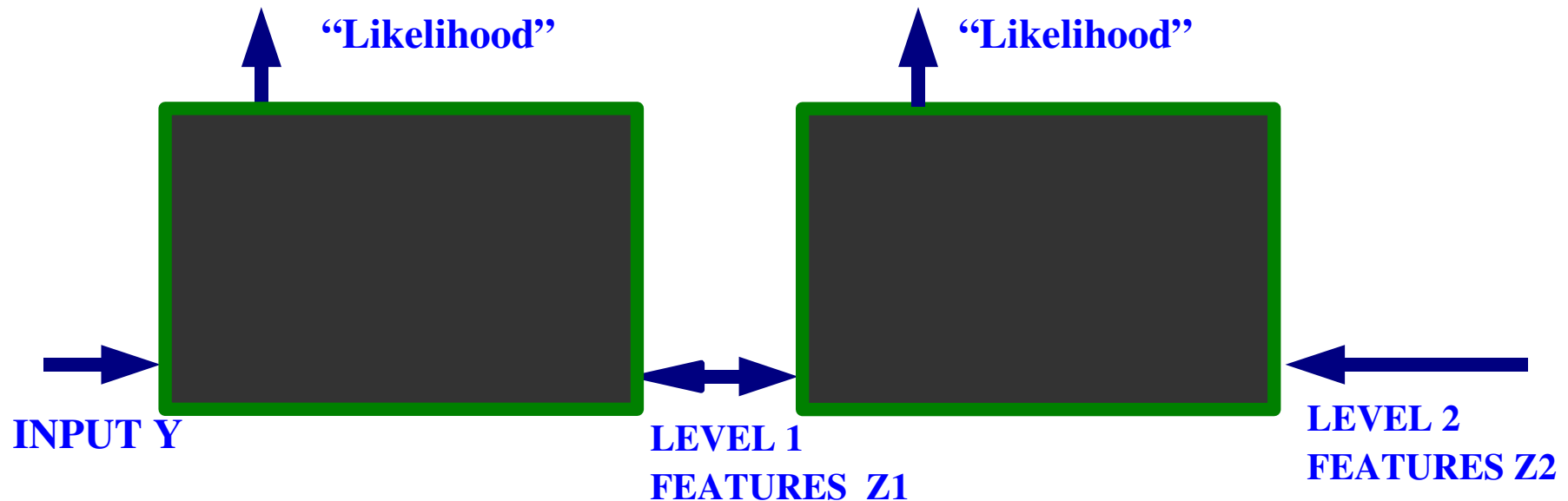


# Learning Deep Feature Hierarchies (unsupervised)

- 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
- We need **unsupervised** learning methods that can learn invariant feature hierarchies
- This talk will present methods to learn hierarchies of **sparse and invariant features**
- 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

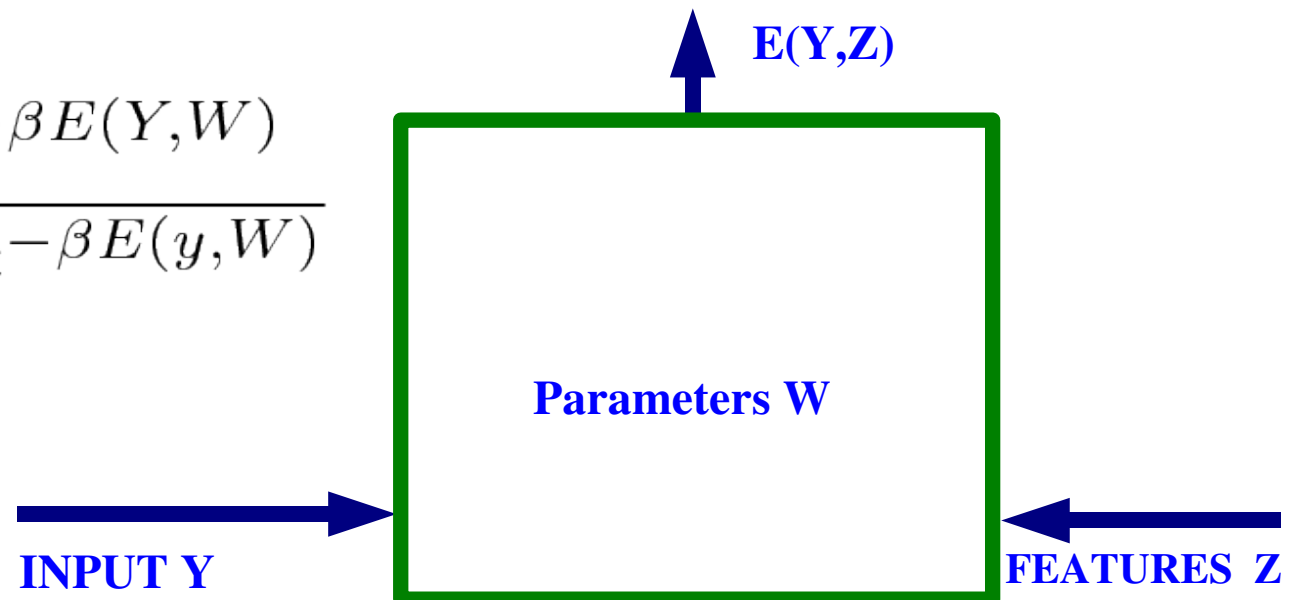
## Energy function: $E(Y,W) = \text{MIN}_Z E(Y,Z,W)$

- ▶ 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

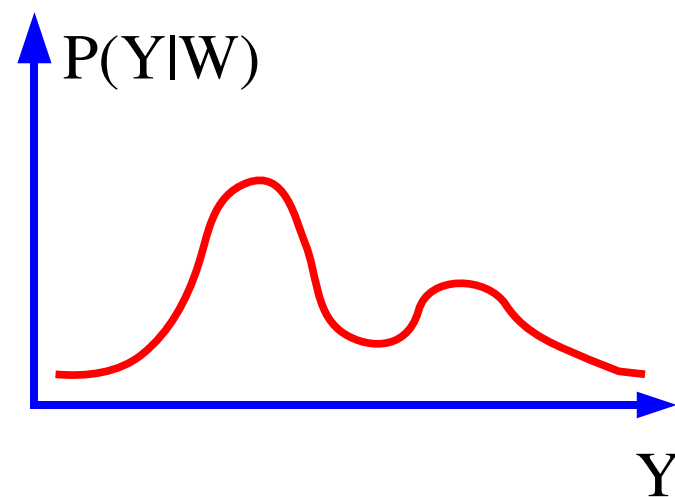
$$P(Y|W) = \frac{e^{-\beta E(Y,W)}}{\int_y e^{-\beta E(y,W)}}$$



# What is Unsupervised Learning?

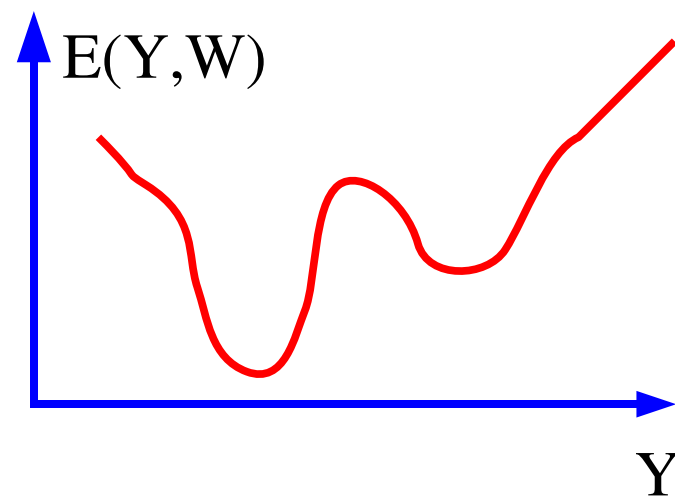
## Probabilistic View:

- ▶ 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:

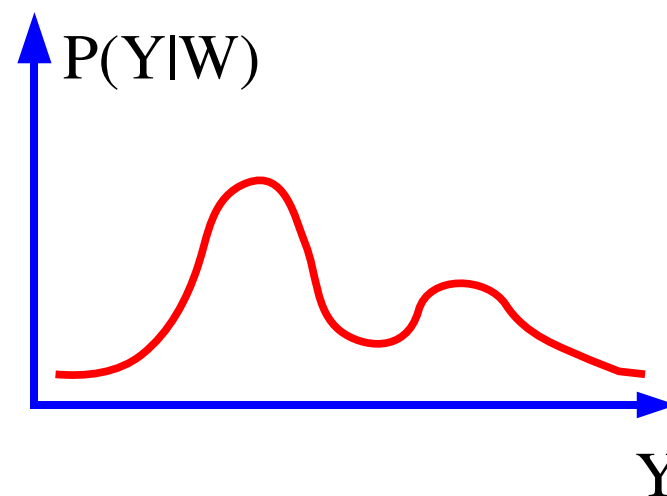
- ▶ produce an energy function  $E(Y, W)$  that:
- ▶ has low value in regions of high sample density
- ▶ has high(er) value everywhere else



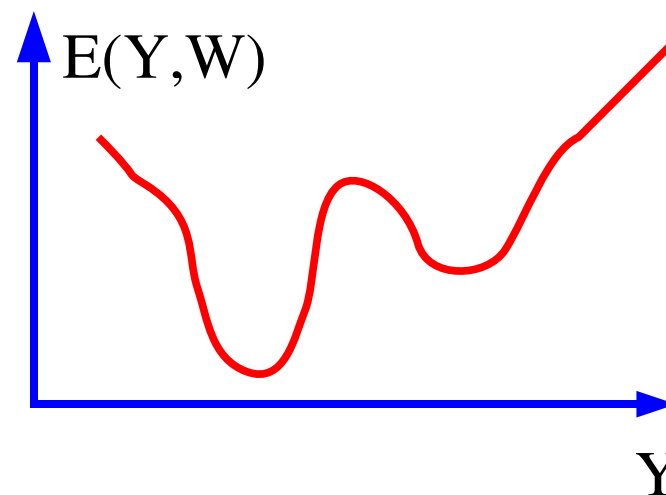


# What is Unsupervised Learning?

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



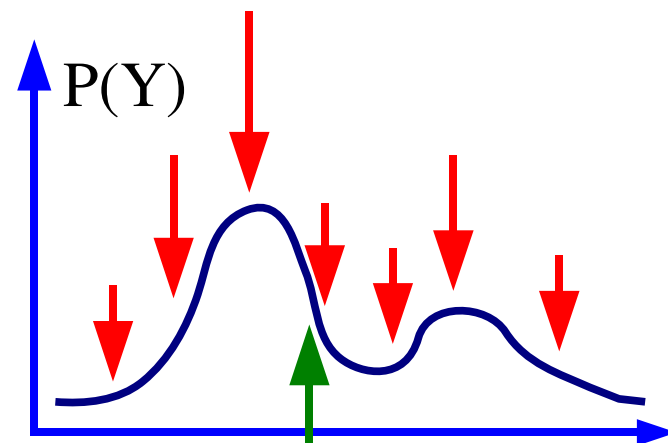
$$E(Y, W) \propto -\log P(Y|W)$$



# Training a Probabilistic Unsupervised Model

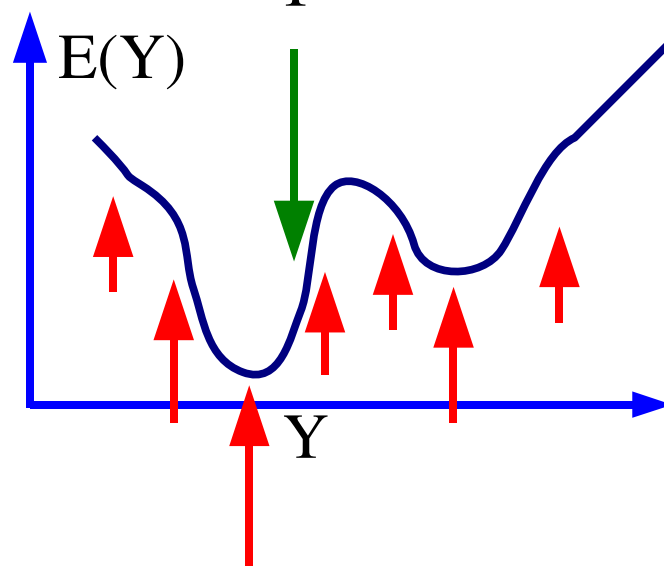
Maximizing  $P(Y|W)$  on training samples

$$P(Y|W) = \frac{e^{-\beta E(Y,W)}}{\int_y e^{-\beta E(y,W)}} \begin{matrix} \text{make this big} \\ \text{make this small} \end{matrix}$$



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

$$L(Y, W) = E(Y, W) + \frac{1}{\beta} \log \int_y e^{-\beta E(y,W)} \begin{matrix} \text{make this small} \\ \text{make this big} \end{matrix}$$



# Training a Probabilistic Unsupervised Model

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

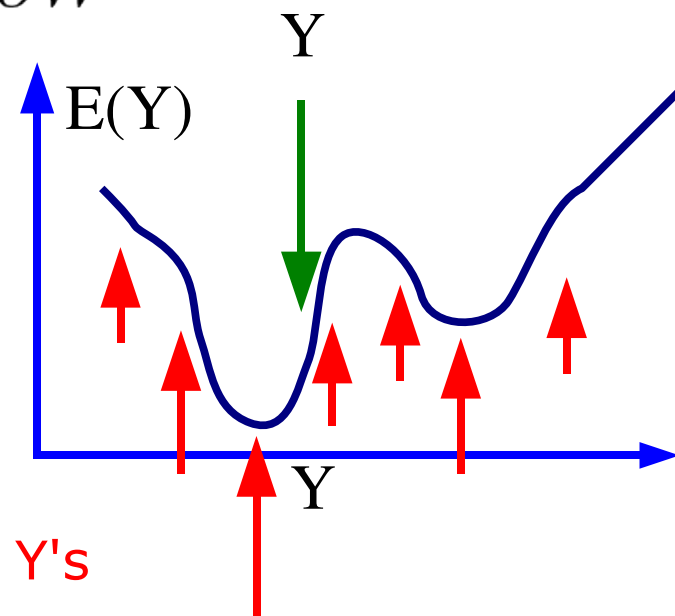
$$\frac{\partial L(Y, W)}{\partial W} = \frac{\partial E(Y, W)}{\partial W} - \int_y P(y|W) \frac{\partial E(y, W)}{\partial W}$$

- 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

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

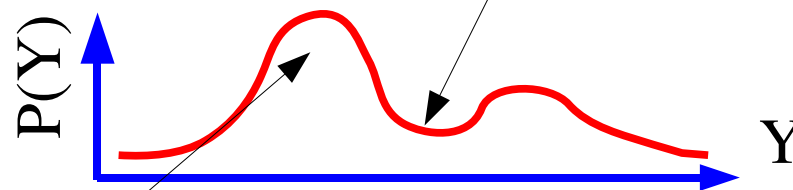
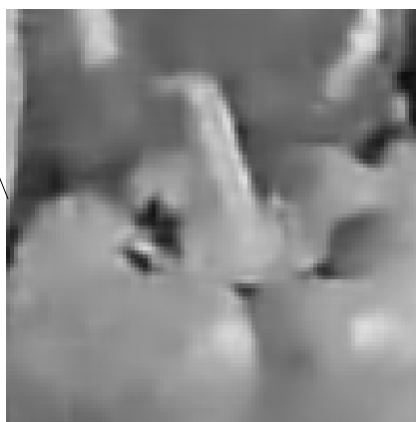
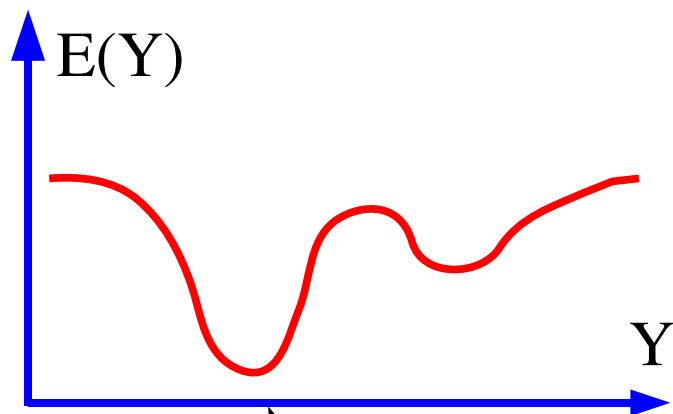
- ▶ Give high probability (or low energy) to training samples
- ▶ Give low probability (or high energy) to everything else
- ▶ There are too many “everything else”!
- ▶ The normalization constant of probabilistic models is a sum over too many terms.
- ▶ **Making the energies of everything else large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

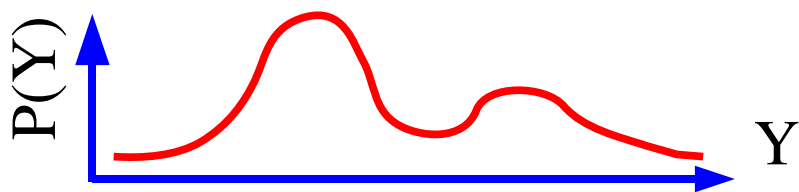


$$P(Y, W) = \frac{e^{-\beta E(Y, W)}}{\int_y e^{-\beta E(y, W)}$$

# The Two Biggest Challenges in Machine Learning

## 1. The “Intractable Partition Function Problem”

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



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

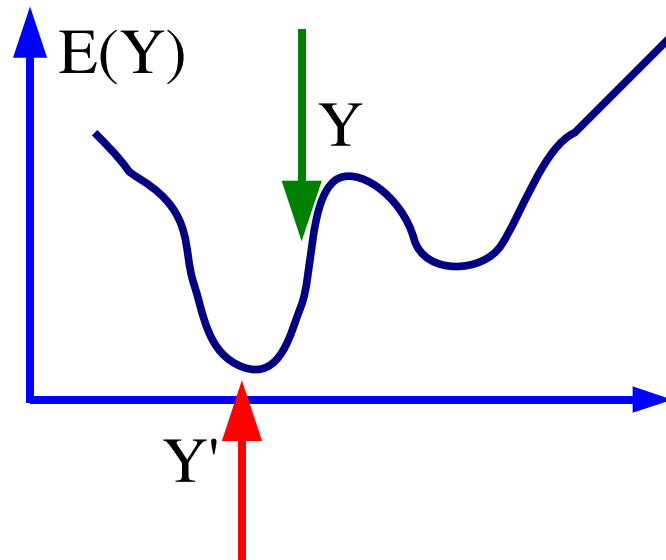
- ▶ Example: what is the PDF of natural images?
- ▶ Question: how do we get around this problem?

## 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 learning internal representations?

# Contrastive Divergence Trick [Hinton 2000]

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



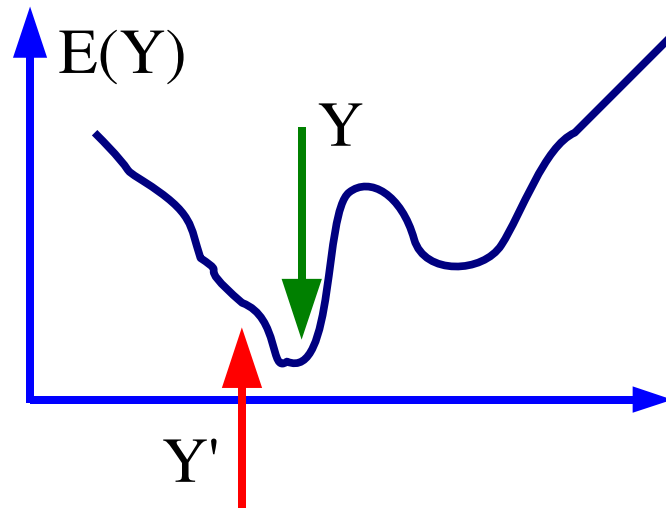
$$W \leftarrow W - \eta \frac{\partial E(Y, W)}{\partial W} + \eta \frac{\partial E(Y', W)}{\partial W}$$

Pushes down on the energy of the training sample  $Y$

Pulls up on the energy  $Y'$

# Contrastive Divergence Trick [Hinton 2000]

- **push down** on the energy of the training sample **Y**
- Pick a sample of low energy **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 Y

Pulls up on the energy 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.

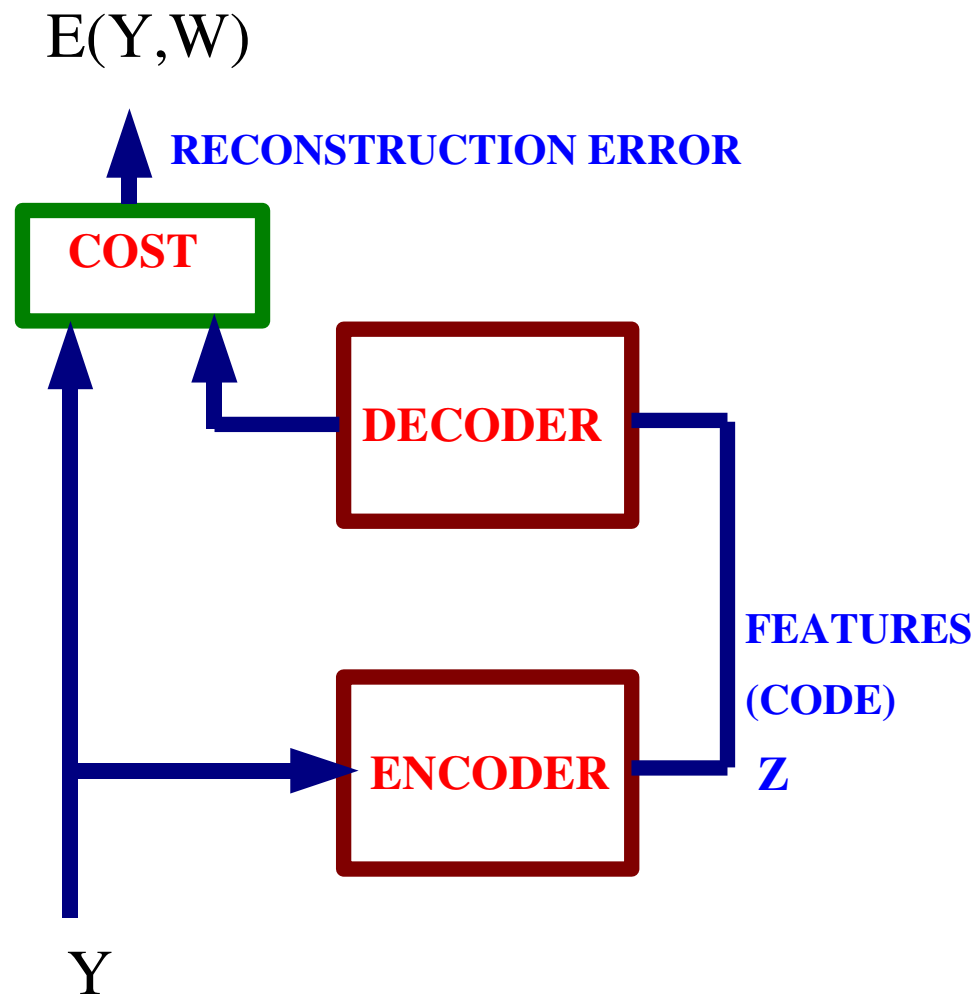
$$W \leftarrow W - \eta \frac{\partial E(Y, W)}{\partial W} + \eta \frac{\partial E(Y', W)}{\partial W}$$

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

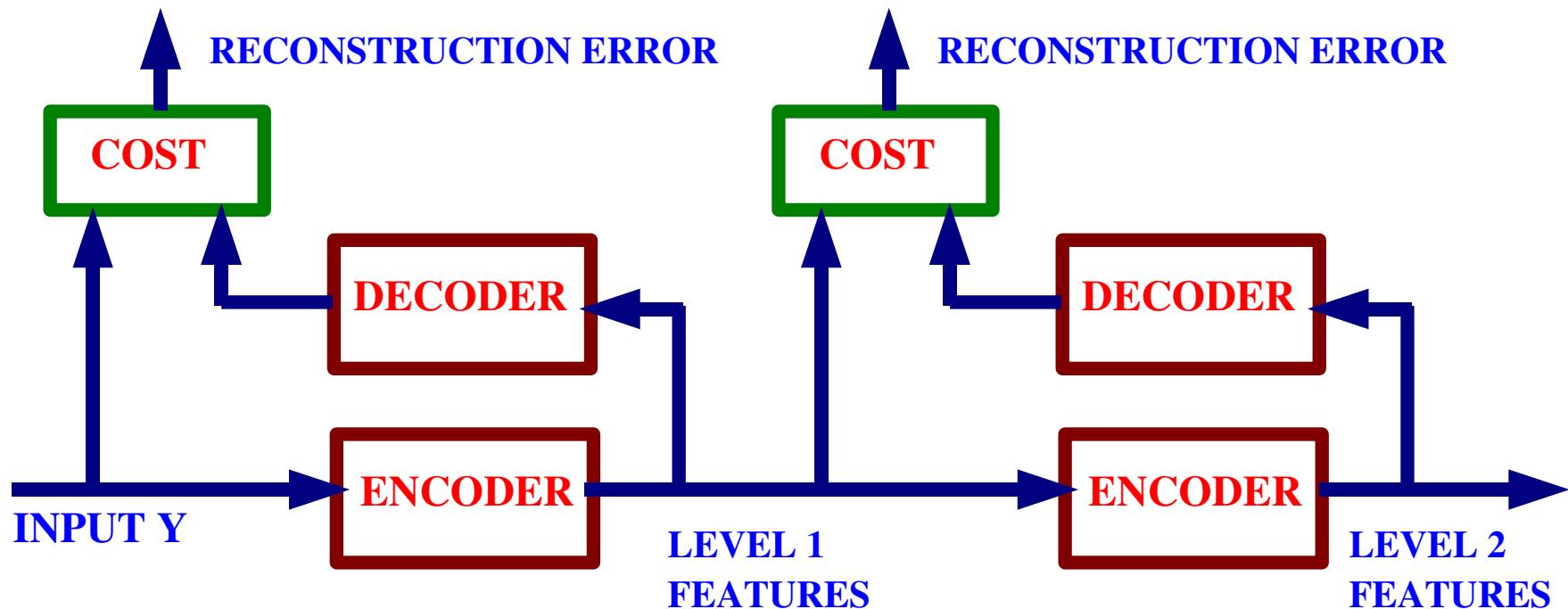
[Hinton 2005, Bengio 2006, LeCun 2006]

## Each stage is composed of

- ▶ 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:

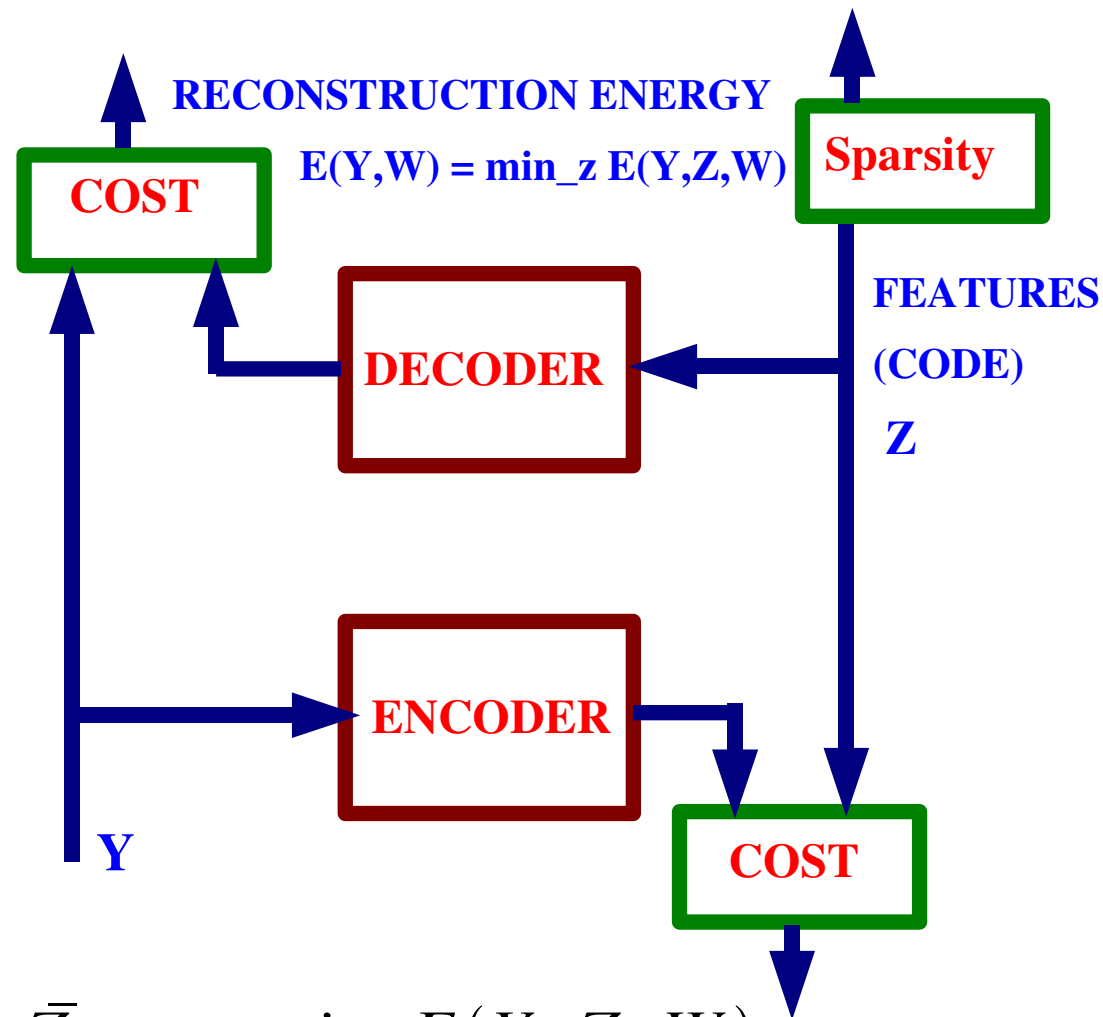
- ▶ None
- ▶ Linear
- ▶ Linear-Sigmoid-Scaling
- ▶ Linear-Sigmoid-Linear

## Optional sparsity penalty

- ▶ None, L1, Log Student-T

## Feature Vector $Z$

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



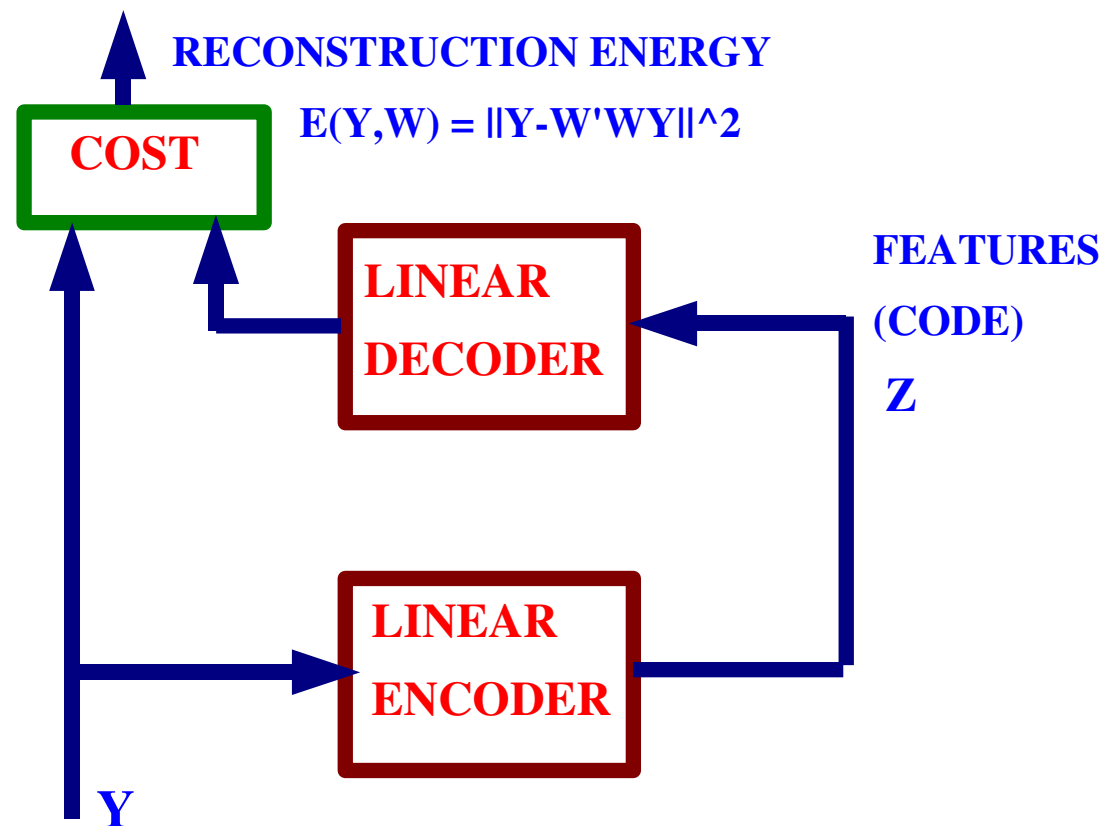
$$\bar{Z}_Y = \operatorname{argmin}_Z E(Y, Z, W)$$

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

# Encoder/Decoder Architecture: PCA

## PCA:

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

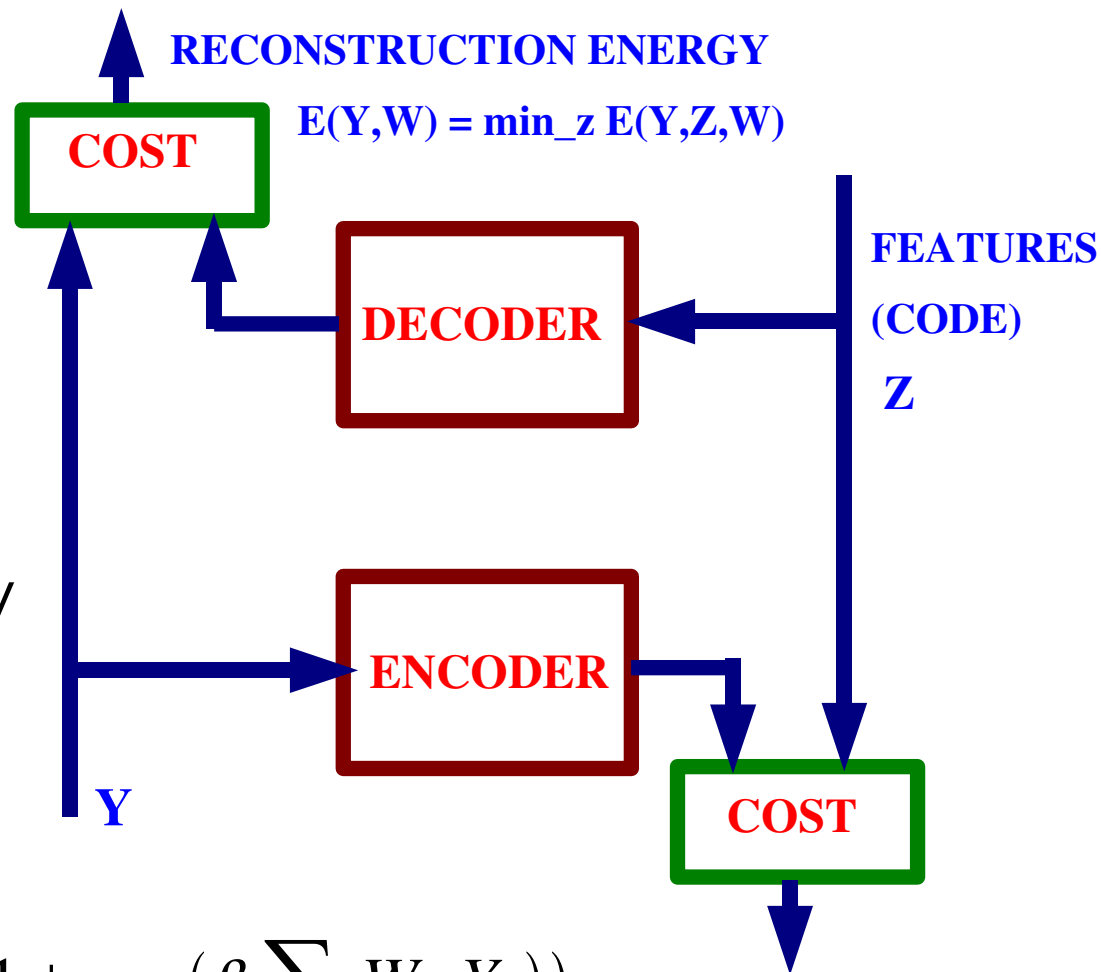


# Encoder/Decoder Architecture: RBM

## 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



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

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

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

# Restricted Boltzmann Machine [Hinton]

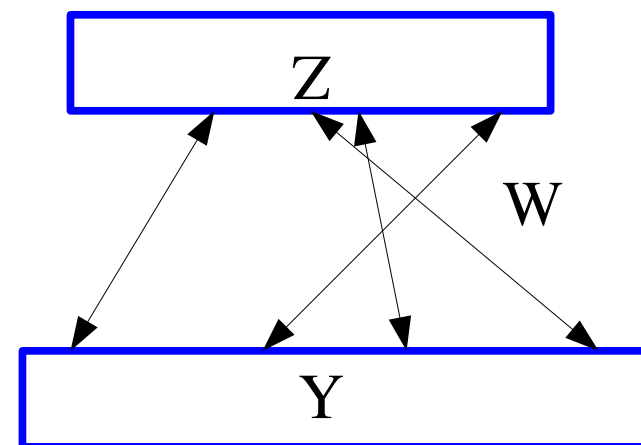
$$E(Y, Z, W) = \sum_{ij} -Y_i W_{ij} Z_j$$

$$E(Y, W) = -\lambda / \beta \log \sum_Z \exp(-\beta E(Y, Z, W))$$

• **W** is a symmetric matrix

• **Z** is a binary (stochastic) vector

▶ the distribution on Z can be approximated by sampling

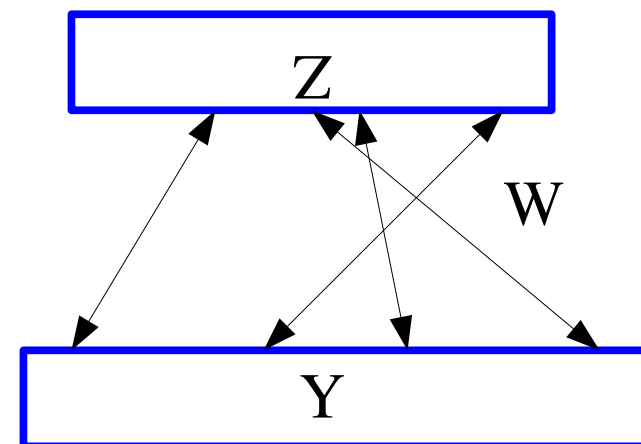


$$P(Z_j = \lambda / Y, W) = \lambda / (\lambda + \exp(\beta \sum_i W_{ij} Y_i))$$

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

# Restricted Boltzmann Machine: Learning Procedure

- 1. Clamp  $Y$  with observed data vector
- 2. Sample  $Z$  from  $P(Z/Y, W)$
- 3. Sample  $\bar{Y}$  from  $P(Y/Z, W)$
- 4. Sample  $\bar{Z}$  from  $P(Z/\bar{Y}, W)$
- update  $W$  with



$$W_{ij} \leftarrow W_{ij} + \eta (Y_i Z_j - \bar{Y}_i \bar{Z}_j)$$

The learning rule minimizes the loss:

$$L(W, Y) = E(Y, Z, W) - E(\bar{Y}, \bar{Z}, W)$$

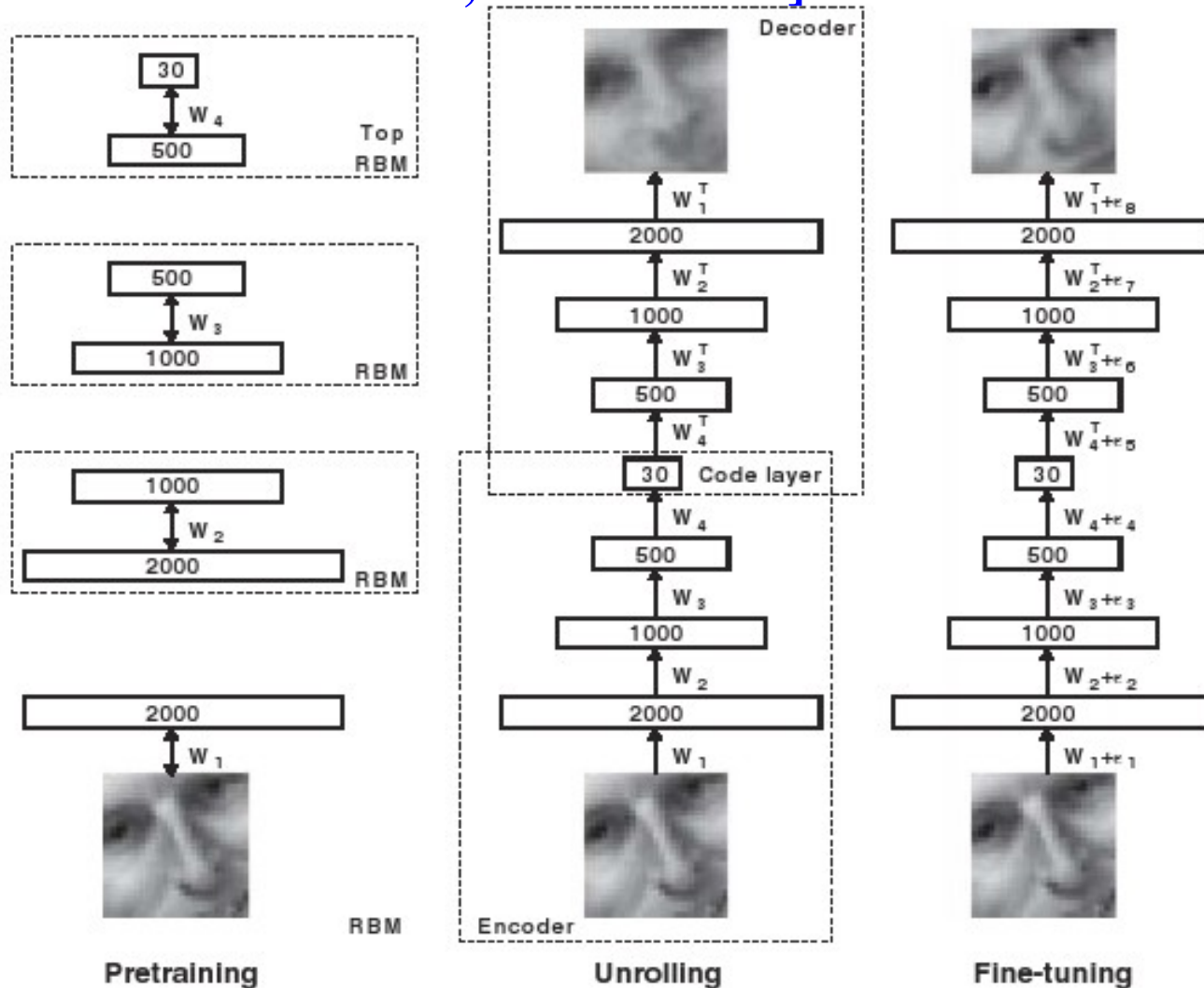
which can be seen a sampled approximation of

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



# Non-Linear Dimensionality Reduction

● [Hinton and Salakhutdinov, Science 2006]

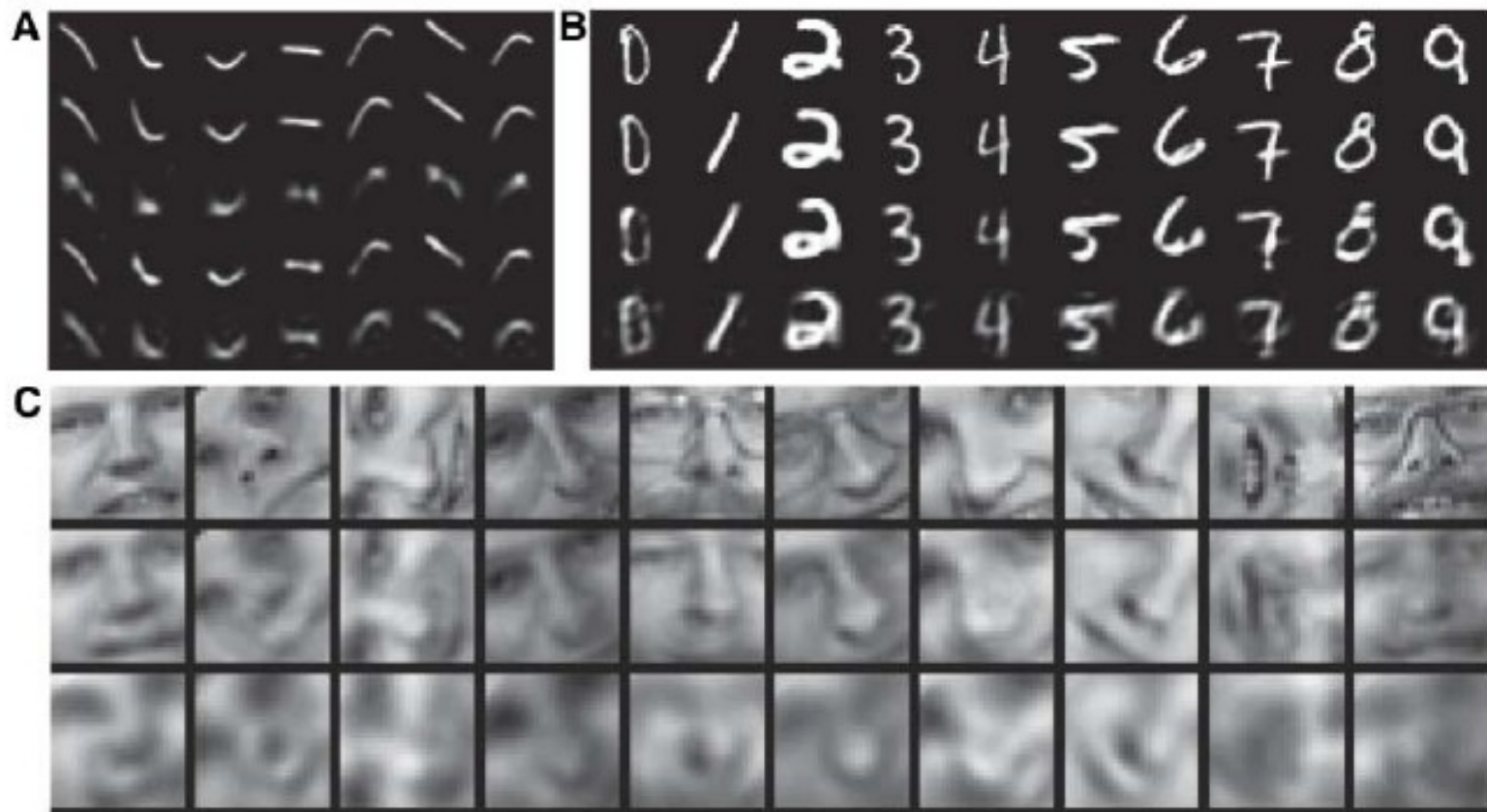


**Fig. 1.** Pretraining consists of learning a stack of restricted Boltzmann machines (RBMs), each having only one layer of feature detectors. The learned feature activations of one RBM are used as the "data" for training the next RBM in the stack. After the pretraining, the RBMs are "unrolled" to create a deep autoencoder, which is then fine-tuned using backpropagation of error derivatives.

# Non-Linear Dimensionality Reduction

● [Hinton and Salakhutdinov, Science 2006]

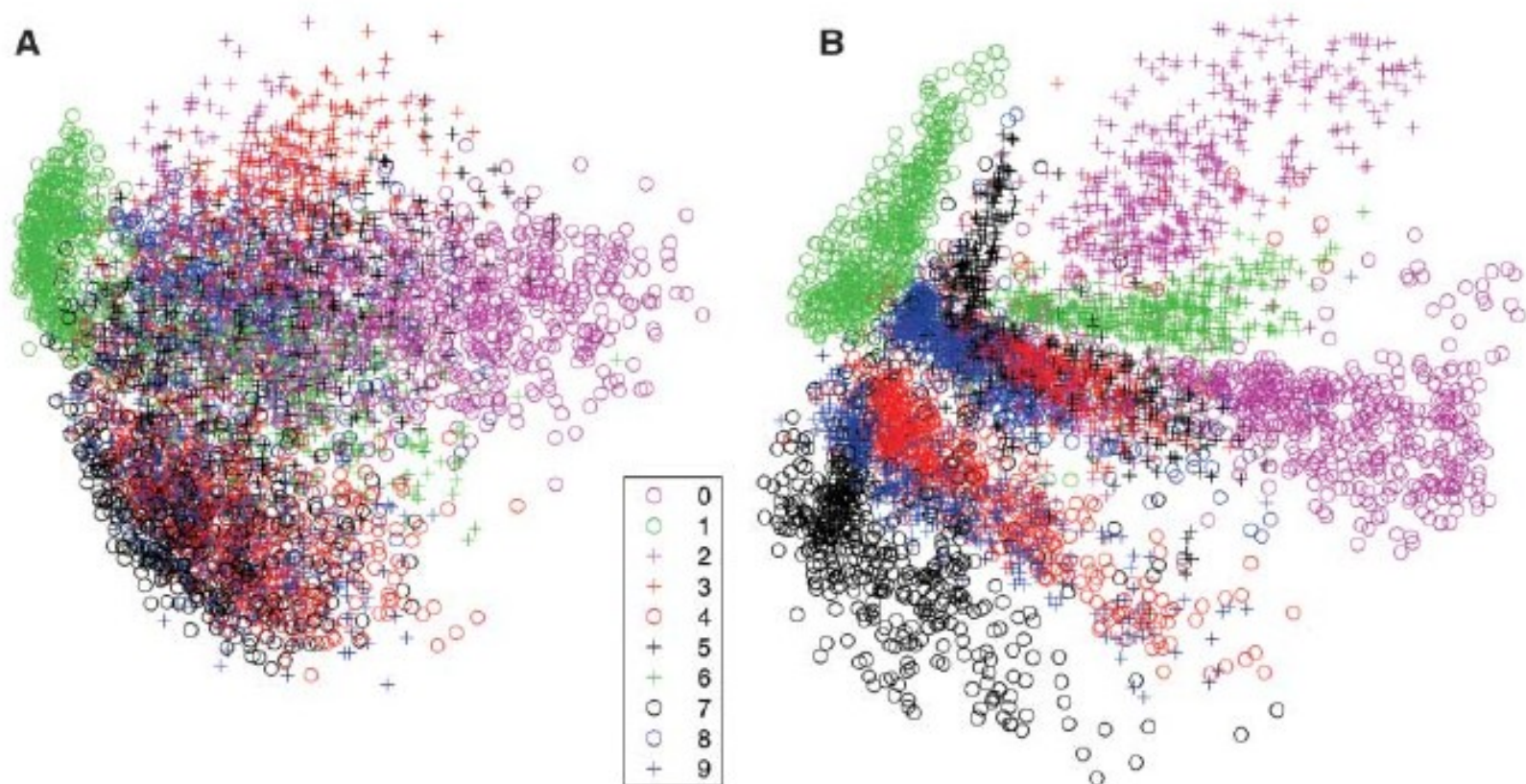
**Fig. 2.** (A) Top to bottom: Random samples of curves from the test data set; reconstructions produced by the six-dimensional deep autoencoder; reconstructions by "logistic PCA" ( $\beta$ ) using six components; reconstructions by logistic PCA and standard PCA using 18 components. The average squared error per image for the last four rows is 1.44, 7.64, 2.45, 5.90. (B) Top to bottom: A random test image from each class; reconstructions by the 30-dimensional autoencoder; reconstructions by 30-dimensional logistic PCA and standard PCA. The average squared errors for the last three rows are 3.00, 8.01, and 13.87. (C) Top to bottom: Random samples from the test data set; reconstructions by the 30-dimensional autoencoder; reconstructions by 30-dimensional PCA. The average squared errors are 126 and 135.



# Non-Linear Dimensionality Reduction

■ [Hinton and Salakhutdinov, Science 2006]

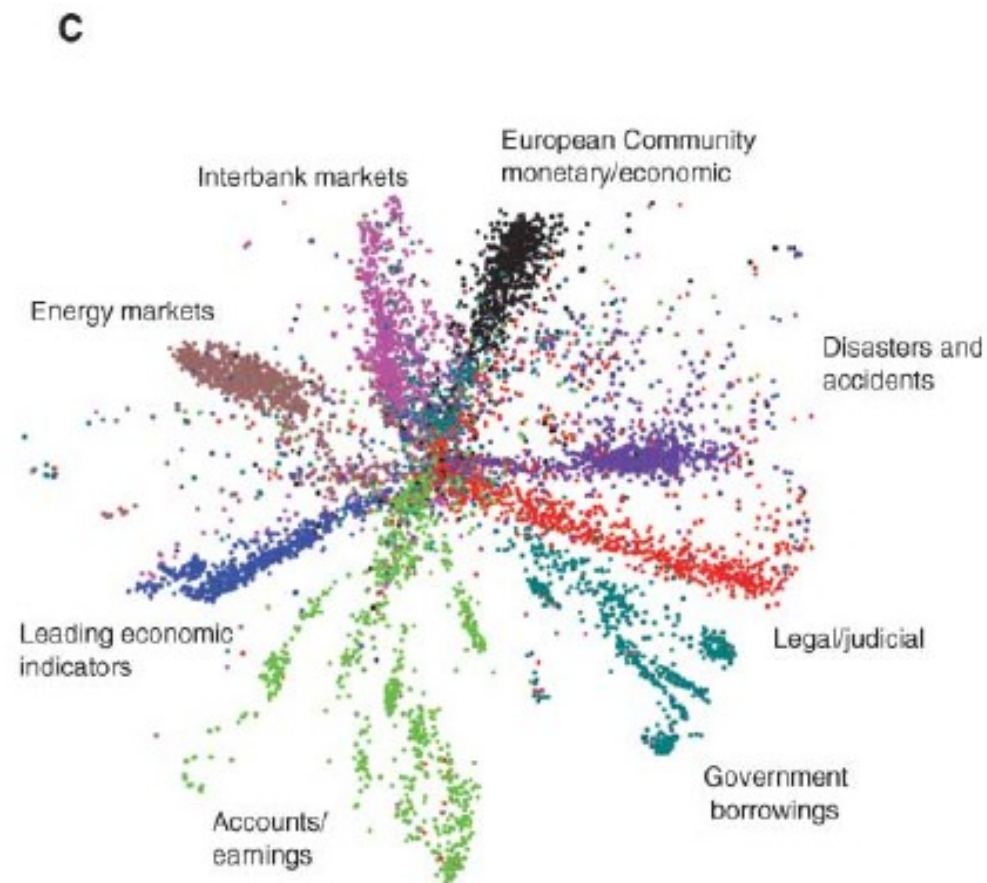
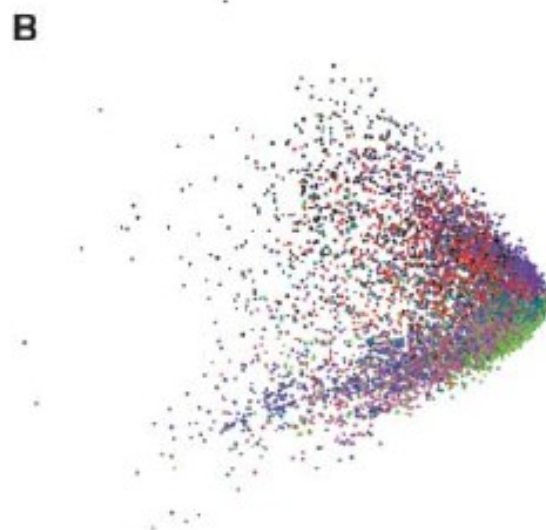
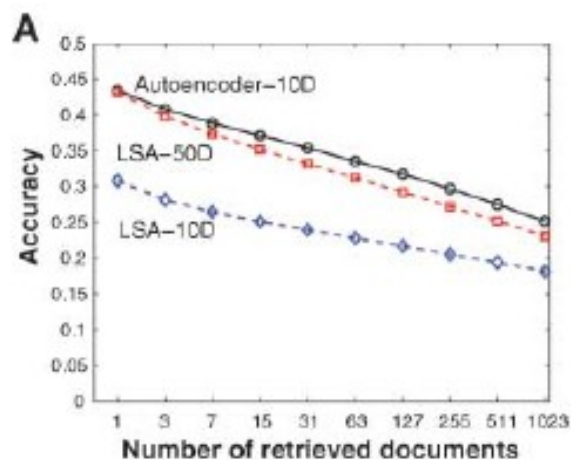
**Fig. 3.** (A) The two-dimensional codes for 500 digits of each class produced by taking the first two principal components of all 60,000 training images. (B) The two-dimensional codes found by a 784-1000-500-250-2 autoencoder. For an alternative visualization, see (8).



# Non-Linear Dimensionality Reduction

[Hinton and Salakhutdinov, Science 2006]

**Fig. 4.** (A) The fraction of retrieved documents in the same class as the query when a query document from the test set is used to retrieve other test set documents, averaged over all 402,207 possible queries. (B) The codes produced by two-dimensional LSA. (C) The codes produced by a 2000-500-250-125-2 autoencoder.



# Unsupervised Feature Learning: Linear Decoder Architecture

## ● K-Means:

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

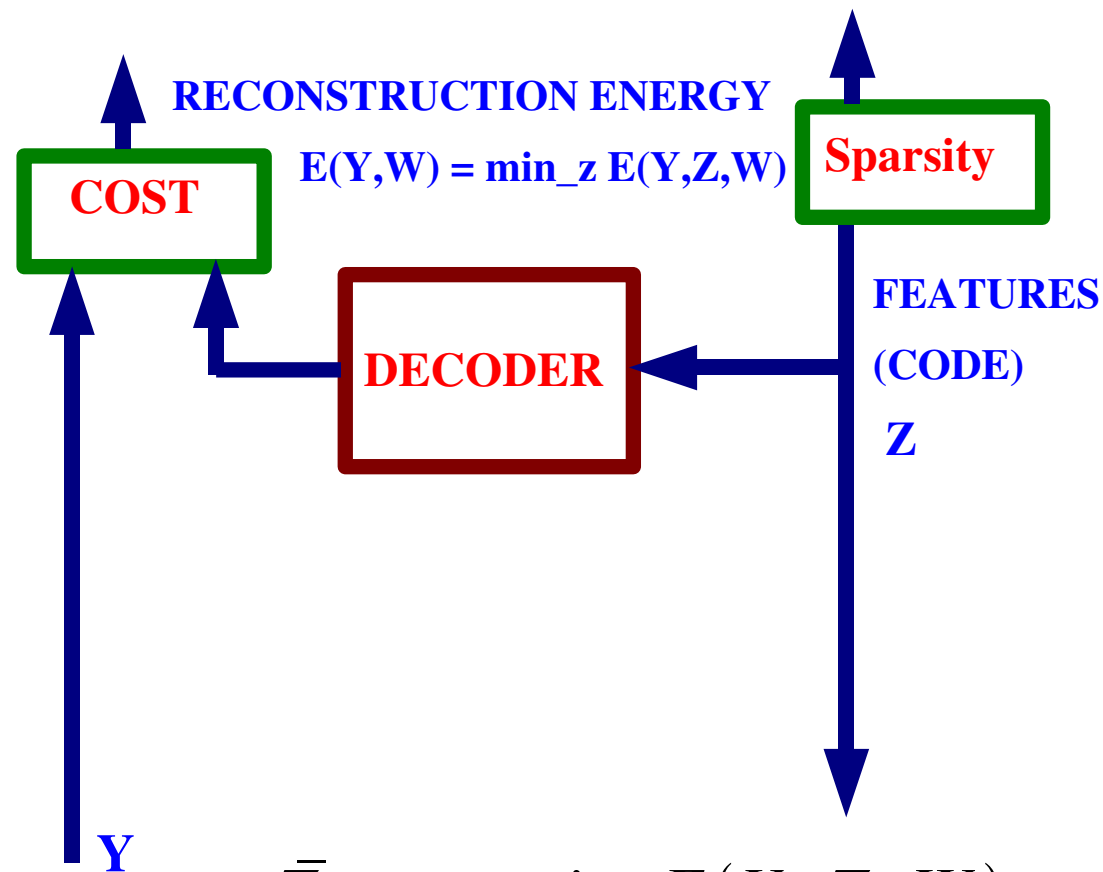
## ● Sparse Overcomplete Bases:

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

## ● Learned Basis Pursuit

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

● **Problem: computing Z from Y involves running a minimization algorithm**

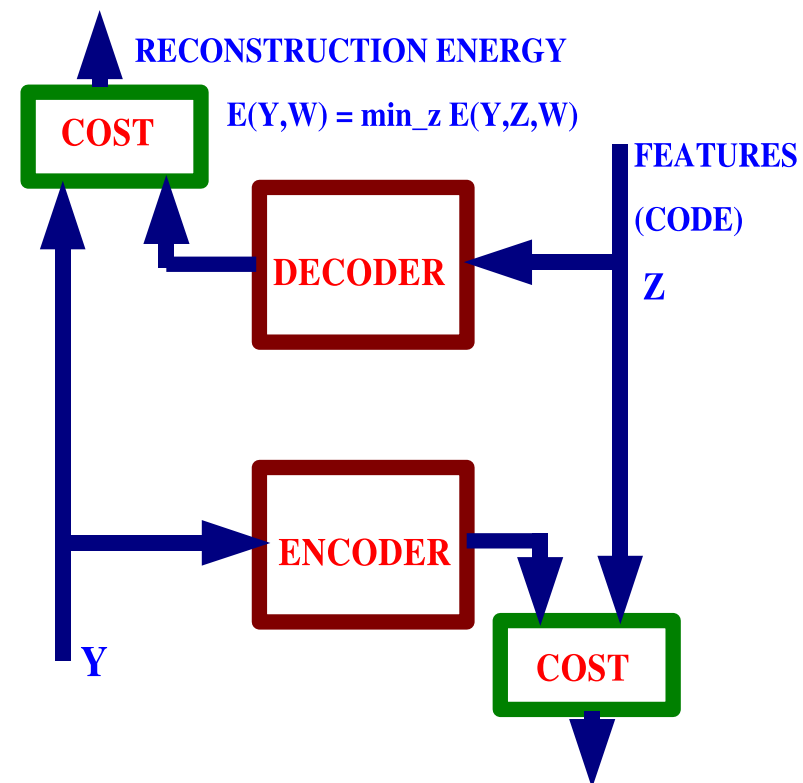


$$\bar{Z}_Y = \operatorname{argmin}_Z E(Y, Z, W)$$

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

# The Main New Idea in This Talk!

- 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
- 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
- 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



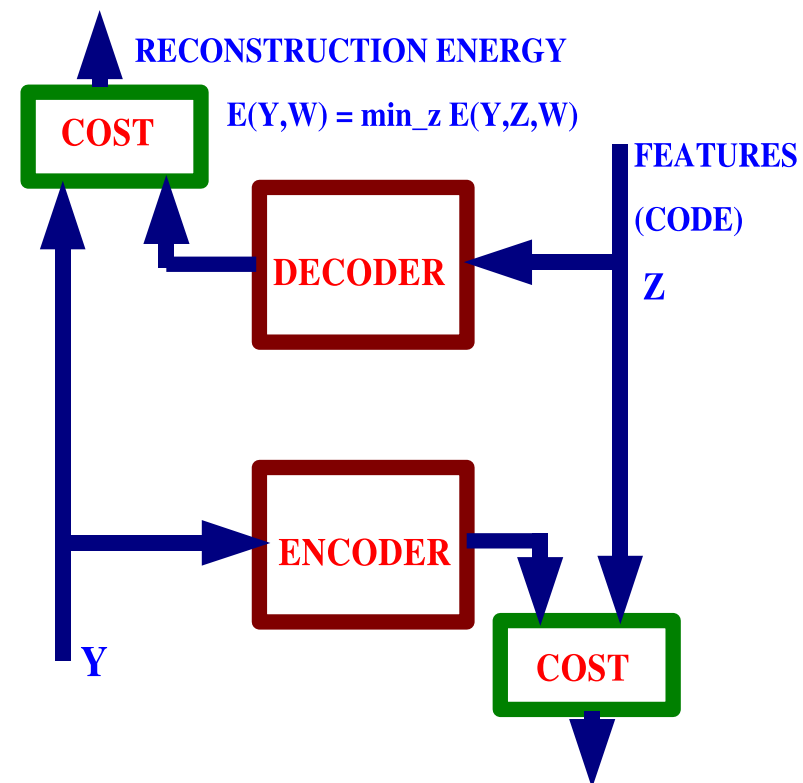
$$\bar{Z}_Y = \operatorname{argmin}_z E(Y, Z, W)$$

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

[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

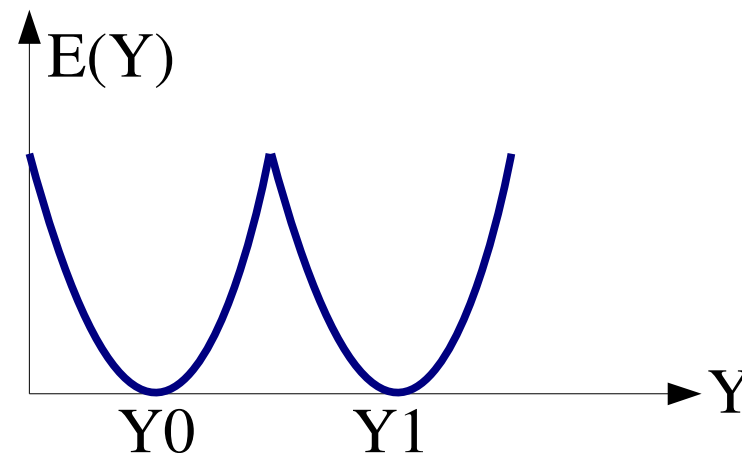
- ▶ The energy (or reconstruction error) can be zero everywhere, because every  $Y$  can be perfectly reconstructed. The energy is flat, and the partition function is unbounded

## 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

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

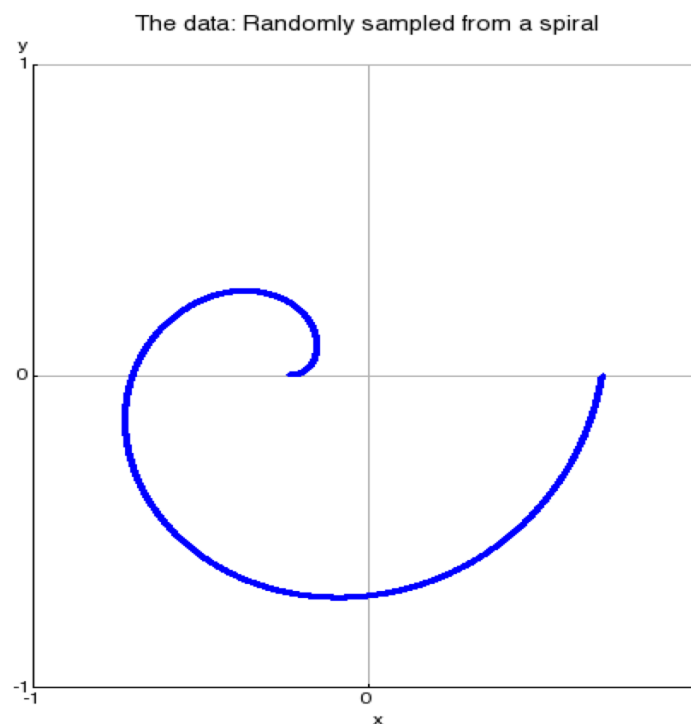
## Example: A Toy Problem. The Spiral

### 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  $y$  from  $x$

### 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

$$Enc(Y) = WY$$

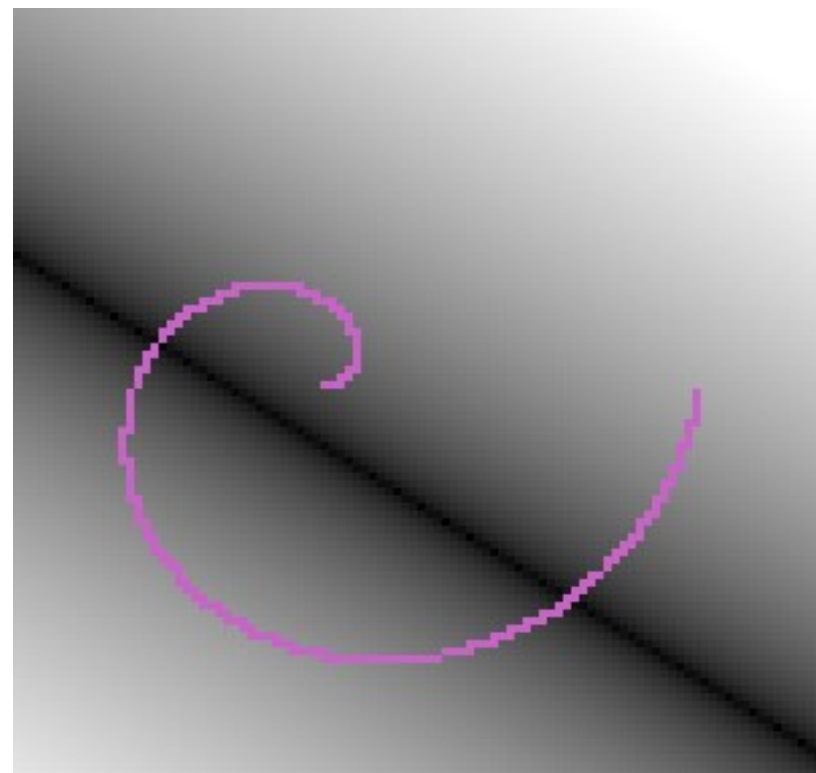
$$Dec(Z) = W^T Z, \text{ where } W \in \mathbb{R}^{N \times M}$$

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

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

- ◆ For large value of  $\gamma$  energy reduces to

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



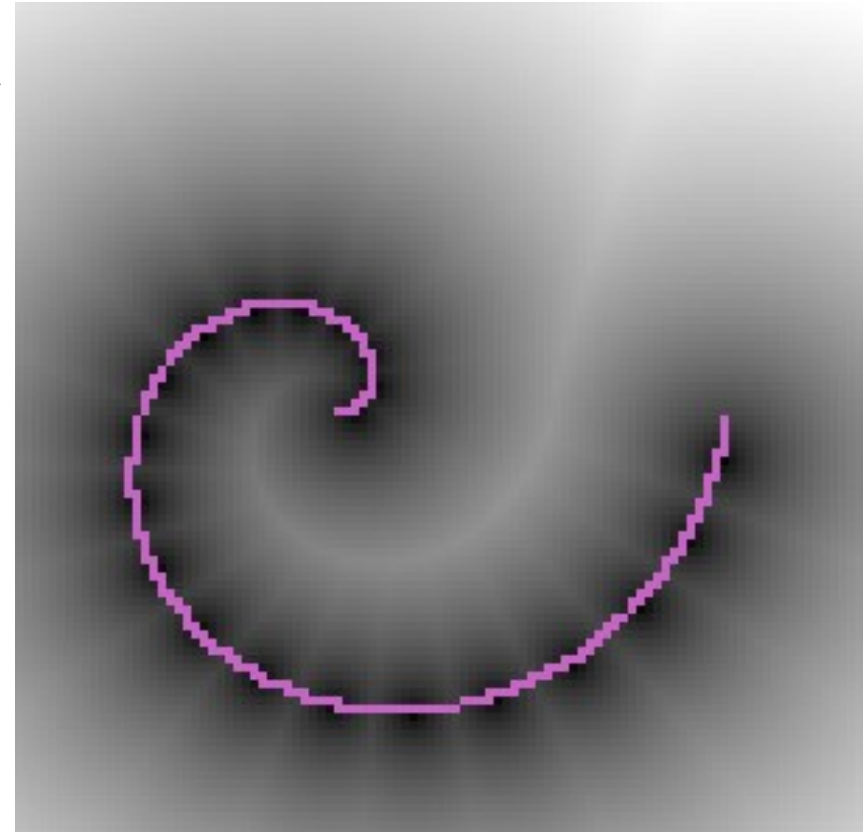
# K-Means

- ◆ 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:

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

$W_i$  is the  $i^{\text{th}}$  prototype

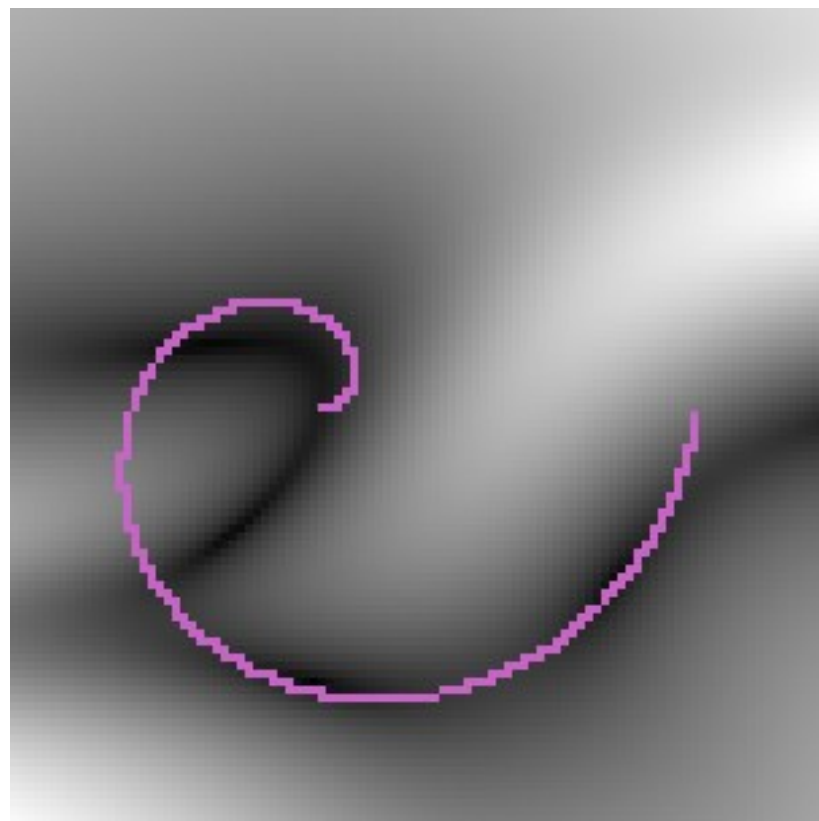
- ◆ Inference involves finding  $Z$  that minimizes the energy



## 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

$$E(Y) = |Dec(Enc(Y)) - Y|^2$$



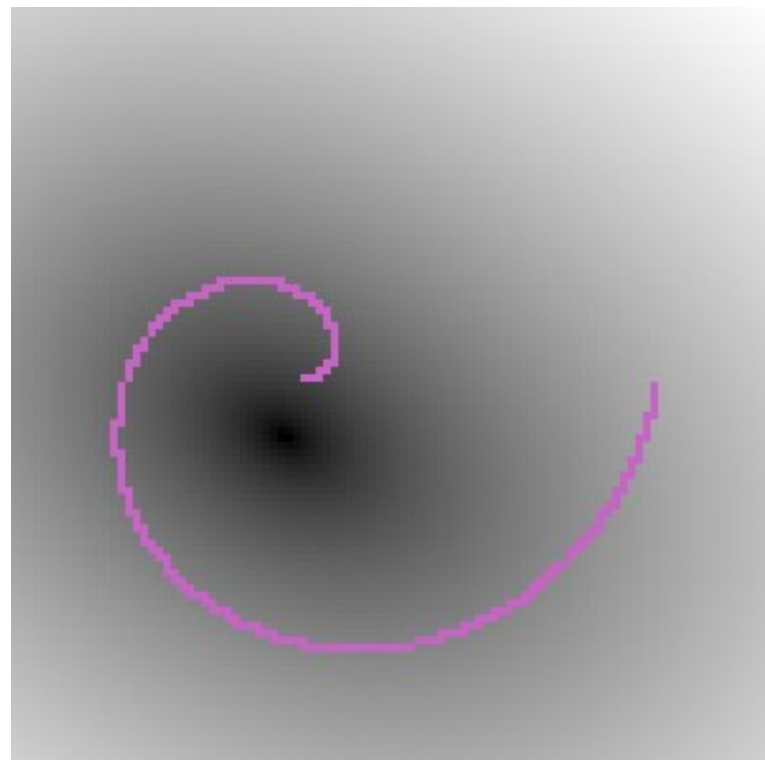
# Wide auto-encoder neural net with energy loss

- ◆ 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

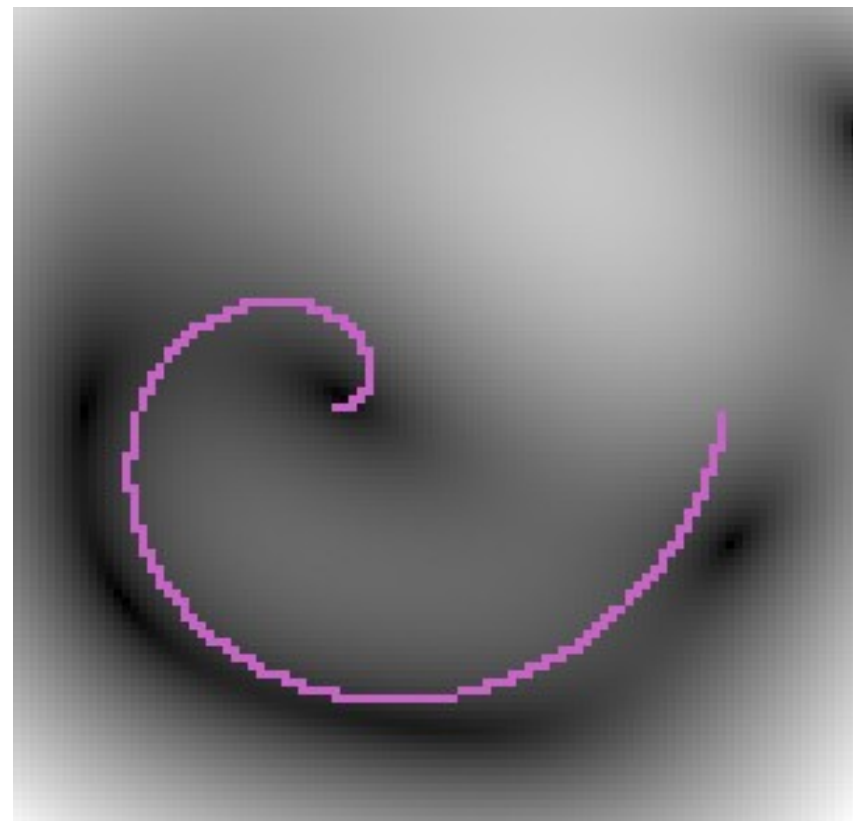
## ◆ Linear-Linear Contrastive Loss

- ◆ 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

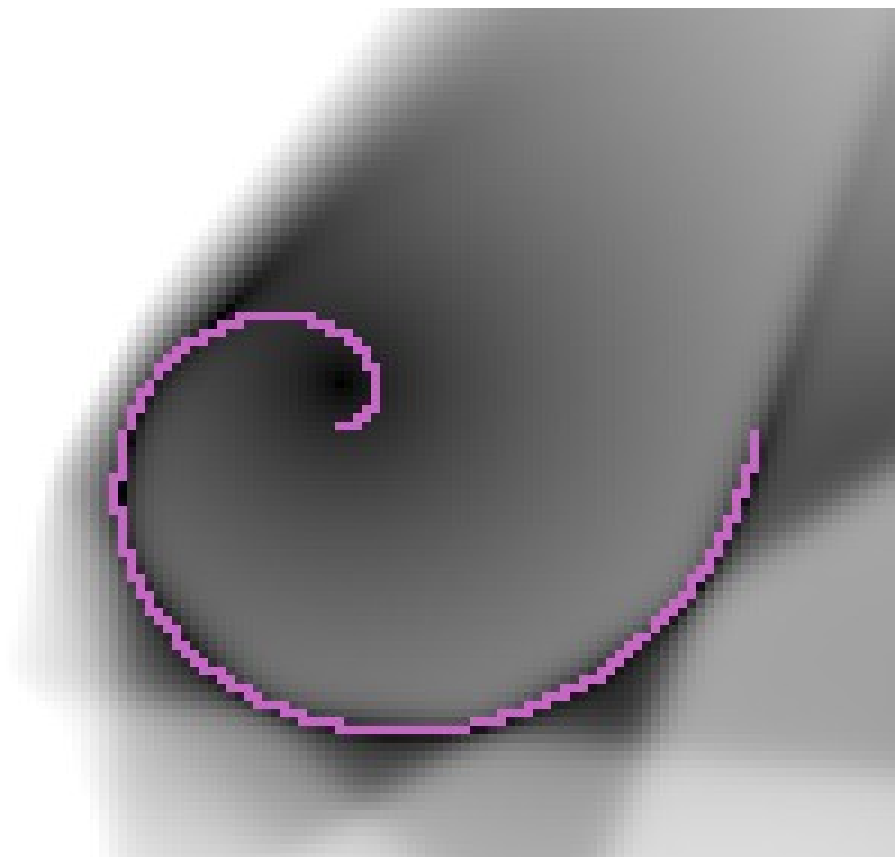
$$L(Y, W) = \alpha E(Y, W) + \max(0, m - E(\bar{Y}, W))$$





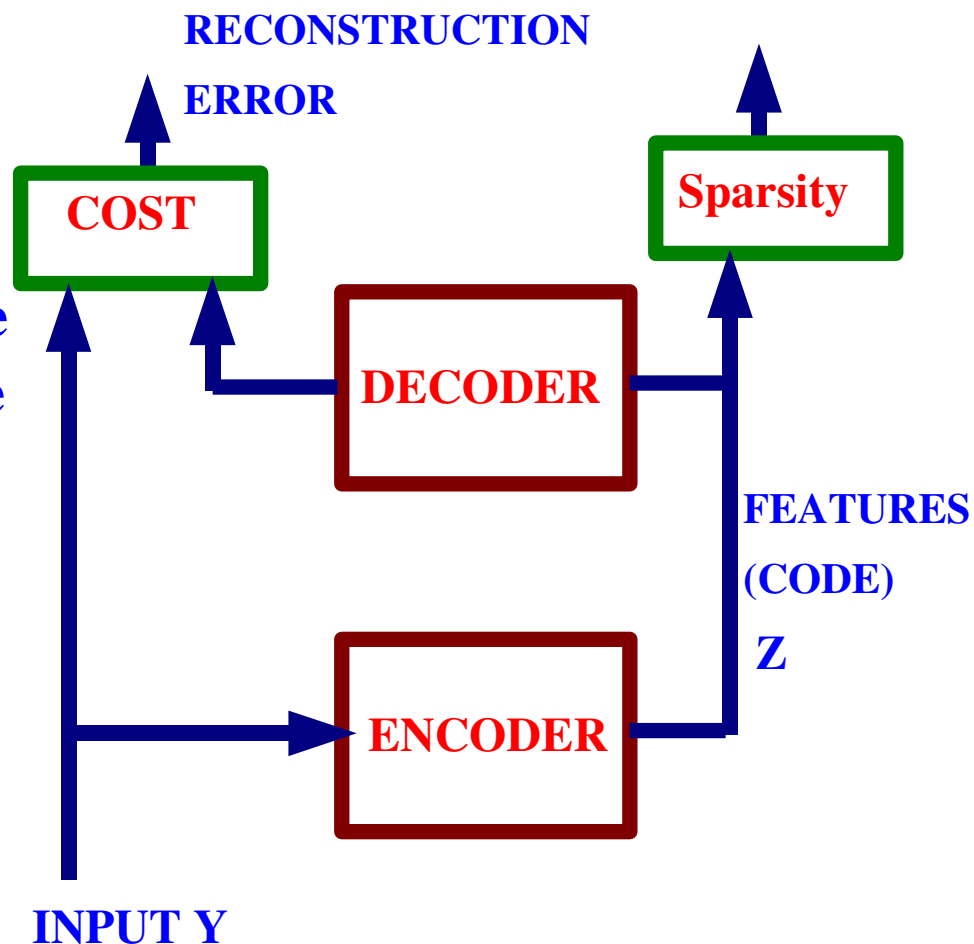
# Wide auto-encoder with sparse code

- ◆ 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



# Learning Sparse Features

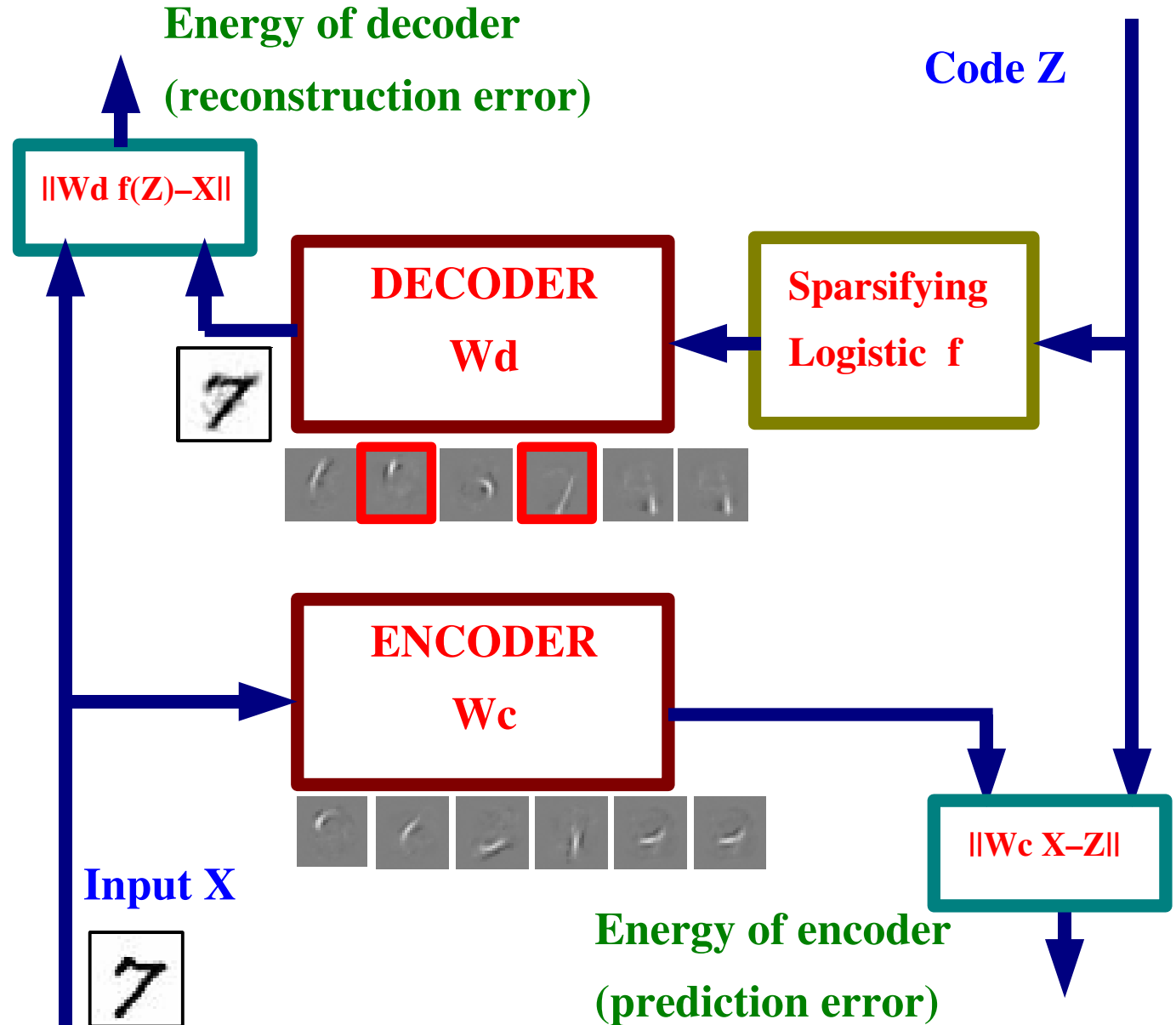
- If the feature vector is larger than the input, the system can learn the identity function in a trivial way
- To prevent this, we force the feature vector to be sparse
- By sparsifying the feature vector, we limit its information content, and we prevent system from being able to reconstruct everything 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



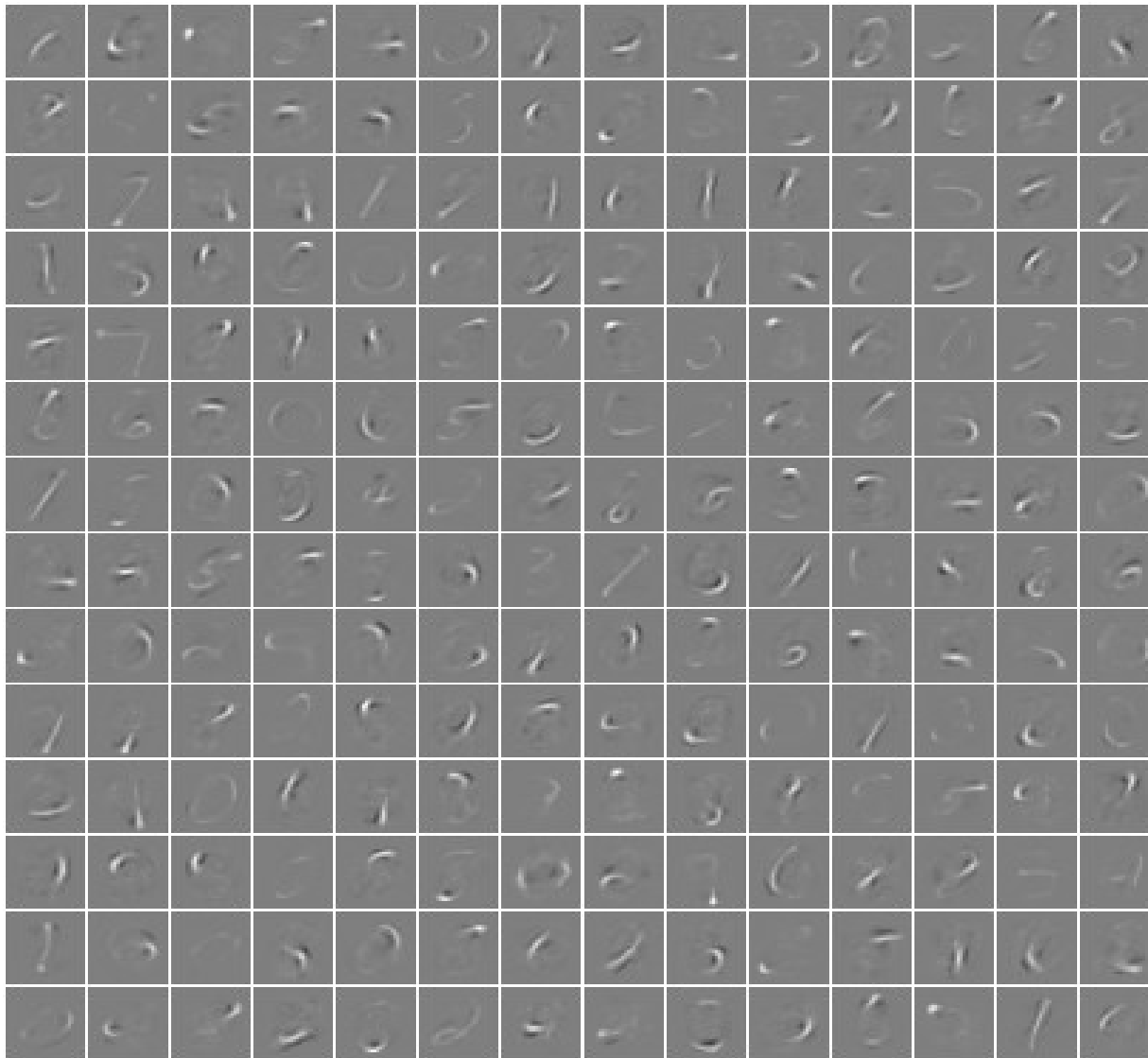
# MNIST Dataset

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 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

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

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

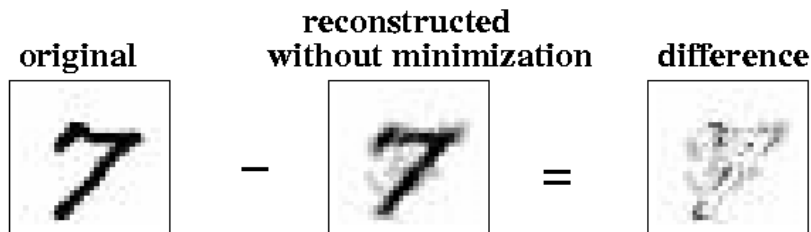
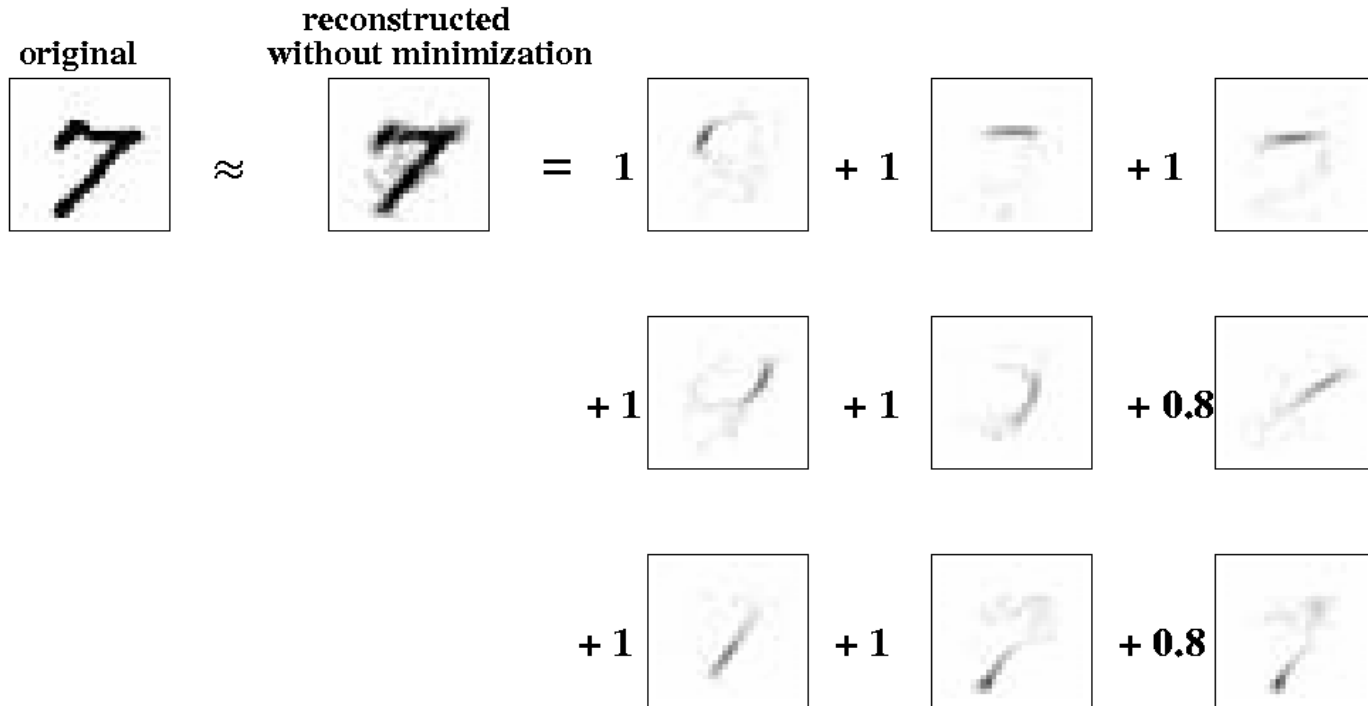
# Training on handwritten digits



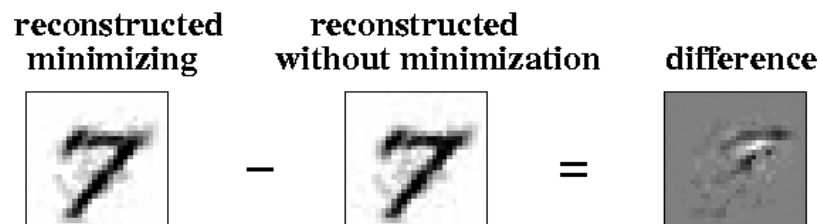
- ◆ 60,000 28x28 images
- ◆ 196 units in the code
- ◆  $\eta$  0.01
- ◆ 1
- ◆  $\beta$  learning rate 0.001
- ◆ L1, L2 regularizer 0.005

Encoder *direct* filters

# Handwritten digits - MNIST



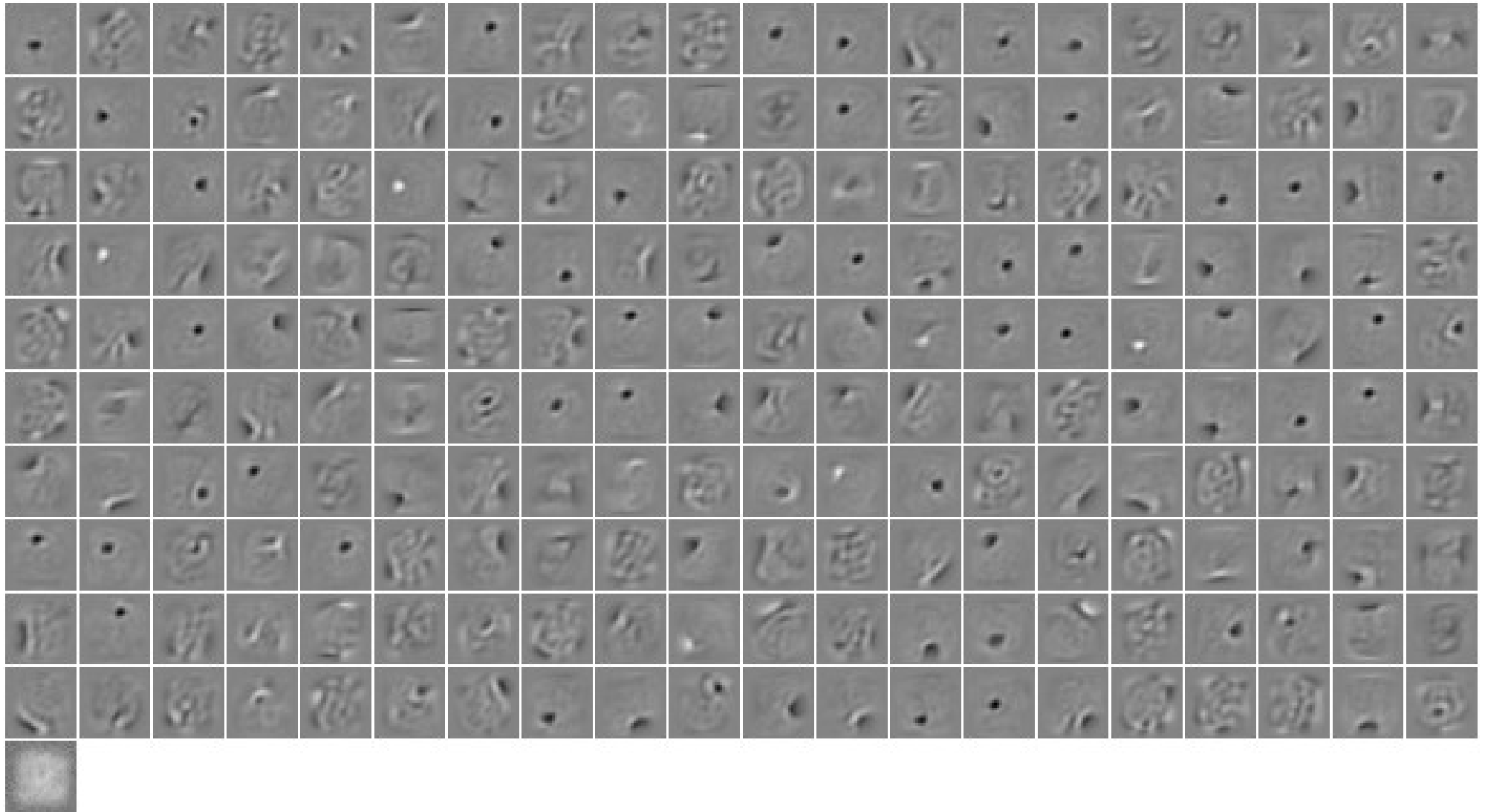
forward propagation through encoder and decoder



after training there is no need to minimize in code space

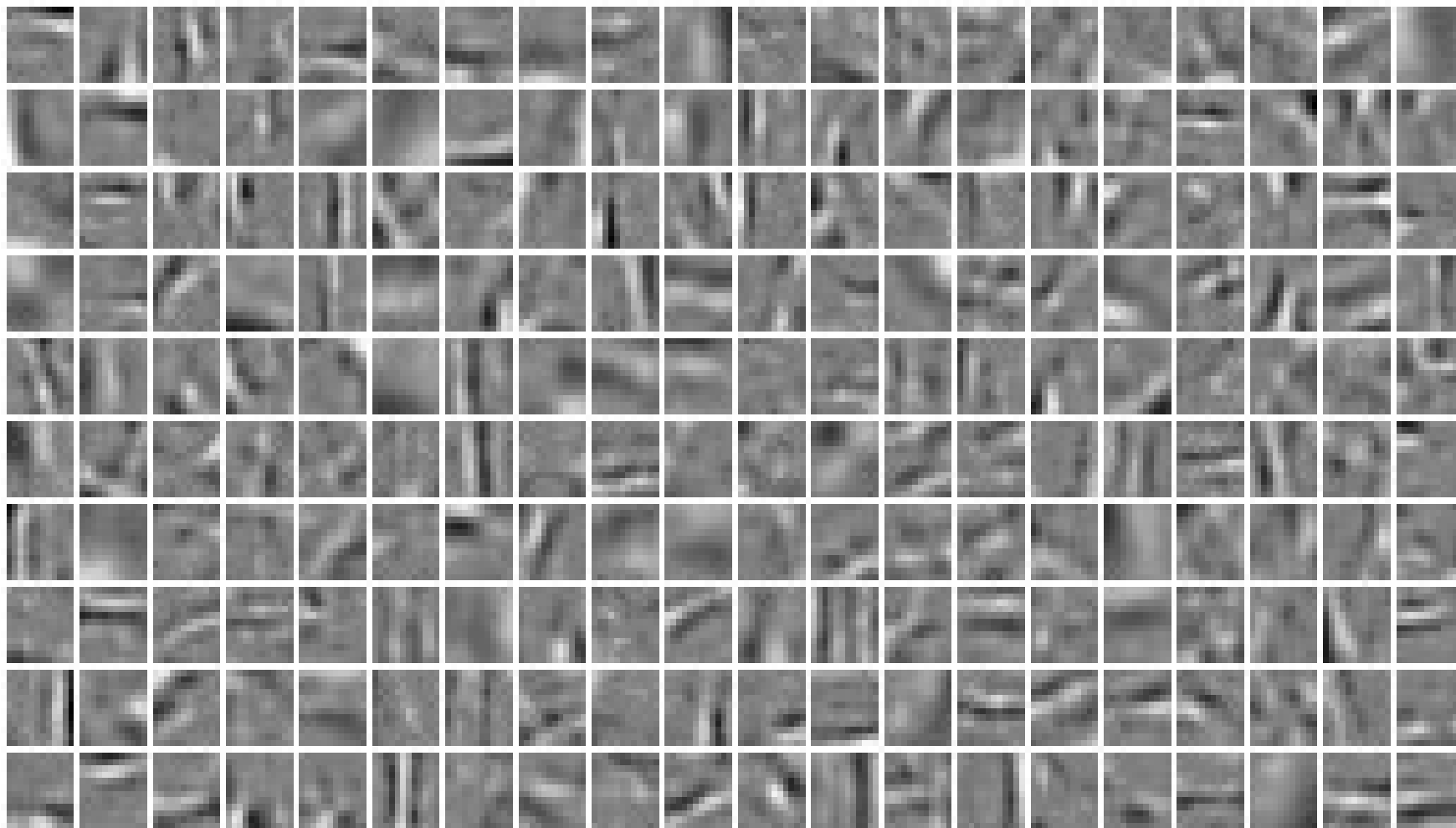
# RBM: filters trained on MNIST

## ● “bubble” detectors



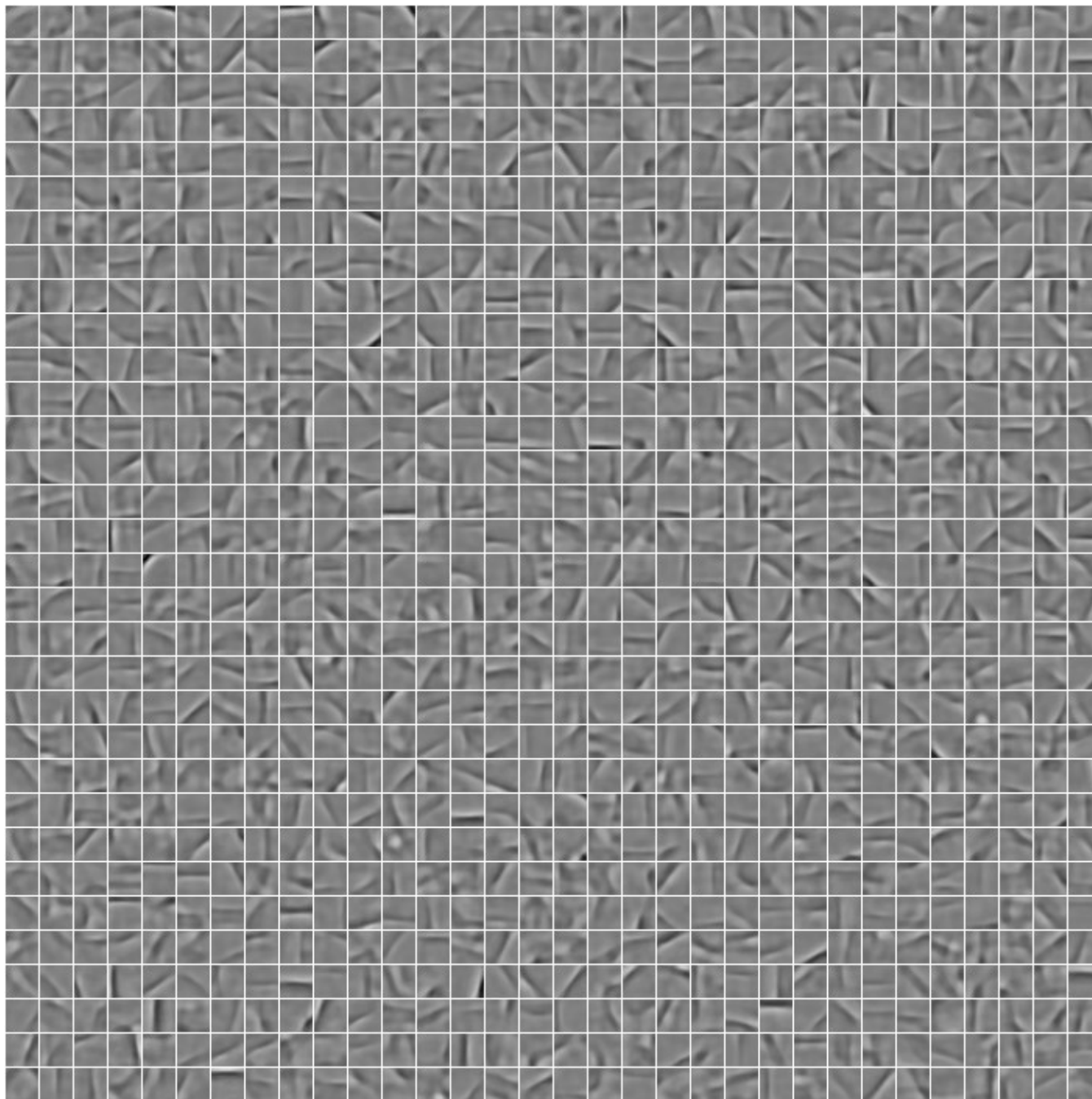
# Sparse Encoder/Decoder Architecture on Natural Image Patches

## Orientation-sensitive feature detectors

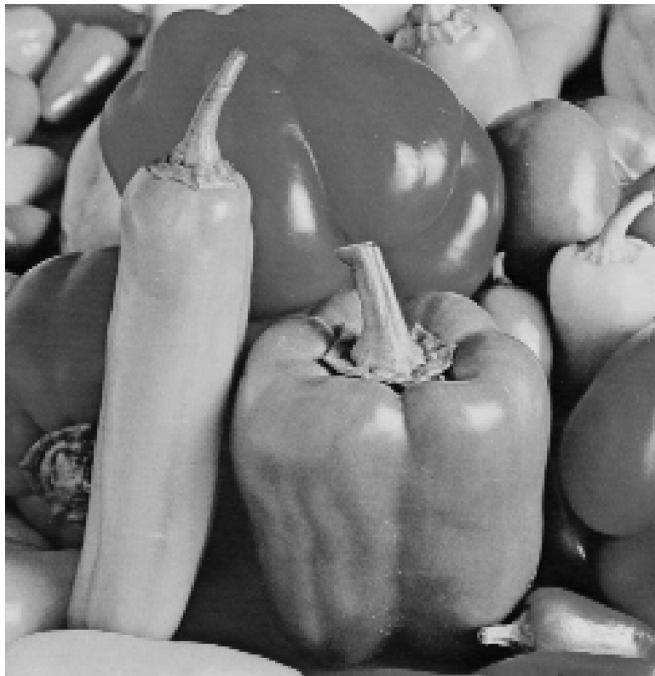




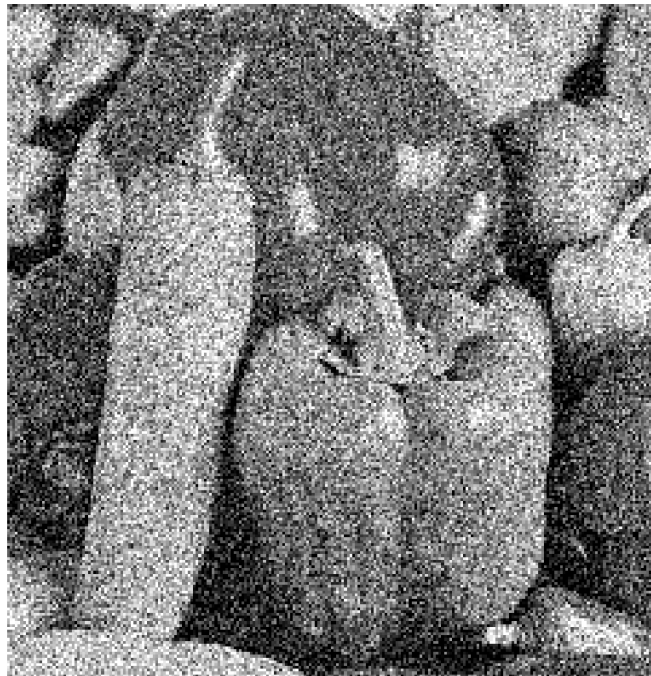
# RBM: filters trained on natural images WITH SPARSITY



# Denoising



original image



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



denoised image  
PSNR 26.50dB

ZOOM ->



# 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	<b>28.61</b>	27.79	26.49	23.46	<b>25.48</b>	25.47	23.15	26.02	<b>26.38</b>	25.95	24.53	27.85	<b>28.26</b>	27.95	26.74	<b>26.35</b>	25.90	26.13	24.52
75 / 10.63	25.97	<b>26.84</b>	25.80	24.13	22.46	<b>23.65</b>	23.01	21.36	24.31	<b>24.79</b>	23.98	22.48	25.77	<b>26.41</b>	25.22	24.13	<b>24.56</b>	24.00	23.69	21.68
100 / 8.13	24.49	<b>25.64</b>	24.46	21.87	21.77	<b>22.61</b>	21.89	19.77	23.09	<b>23.75</b>	22.81	20.80	24.20	<b>25.11</b>	23.71	21.66	<b>23.04</b>	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

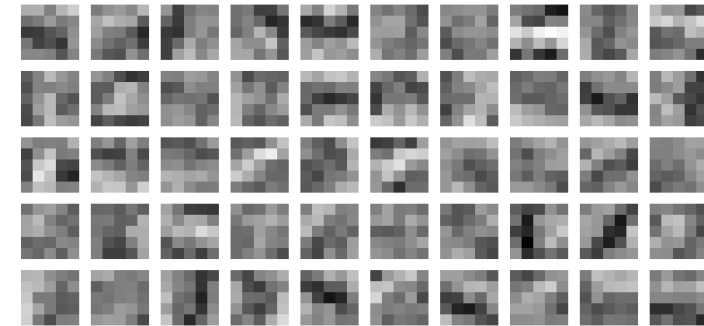
**IDEA:** improving supervised learning by pre-training with the unsupervised method (\*)

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

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

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

supervised filters in first conv. layer

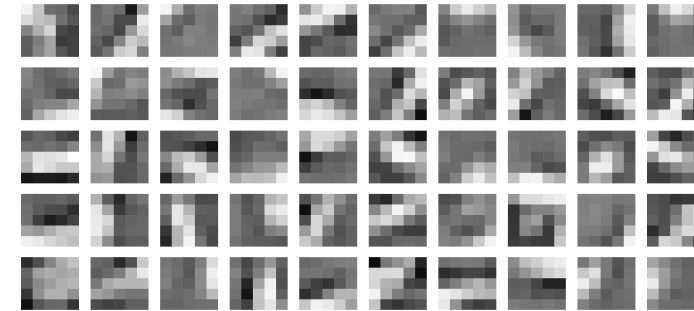


### • *Experiment 1*

- 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%.

unsupervised filters in first conv. layer



### • *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%.

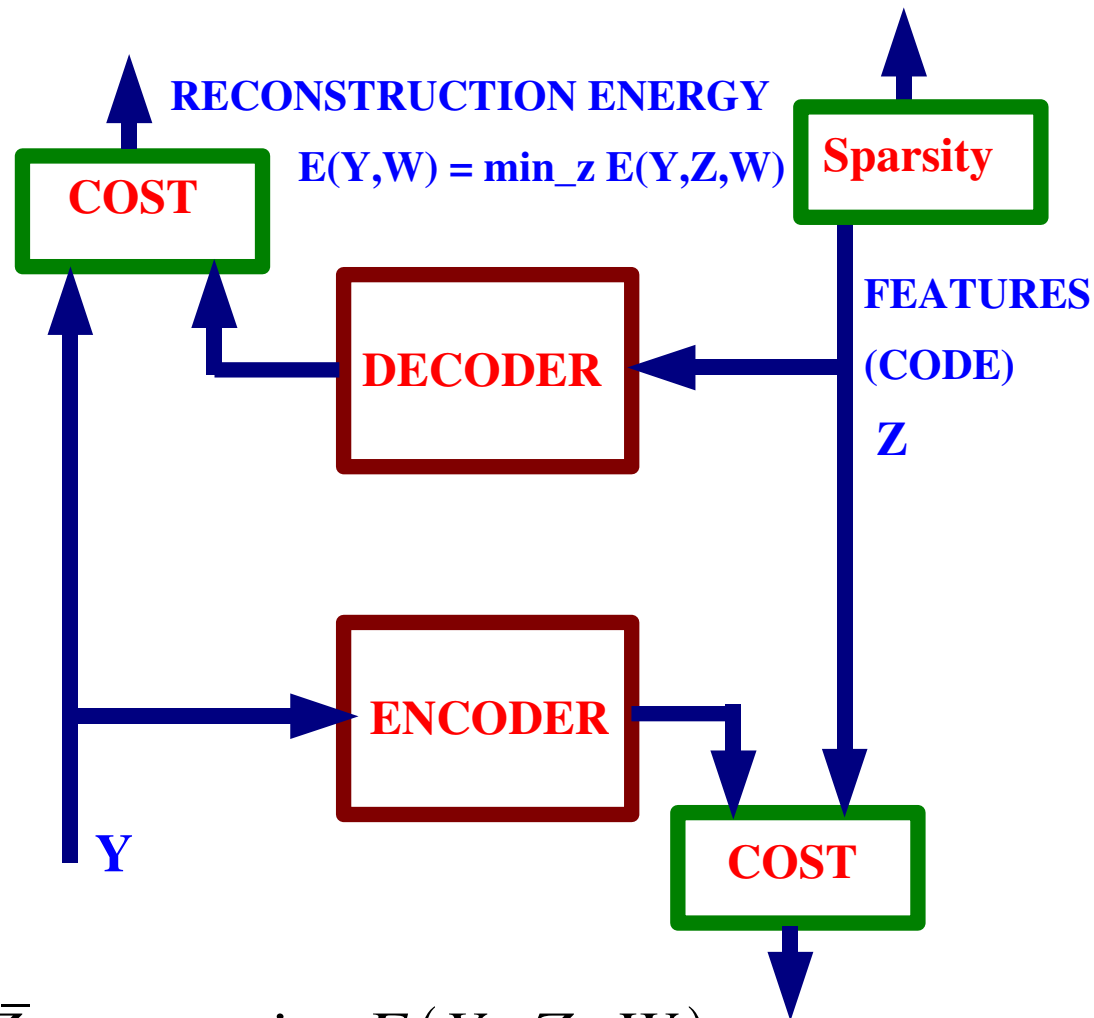
(\*)[Hinton, Osindero, Teh "A fast learning algorithm for deep belief nets" Neural Computaton 2006]

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

CLASSIFIER	DEFORMATION	ERROR	Reference
<b>Knowledge-free methods</b>			
2-layer NN, 100 HU, CE		1.70	Simard et al., ICDAR 2003
3-layer NN, 500+300 HU, CE, reg		1.03	Hinton, in press, 2000
SVM, Gaussian Kernel		1.80	Cortes 92 + Many others
Unsupervised Stacked RBM + backprop		0.90	Hinton, Neur Comp 2006
<b>Convolutional nets</b>			
Convolutional net LeNet-0,		0.80	Ranzato et al. NIPS 2006
Convolutional net LeNet-1,		0.70	Ranzato et al. NIPS 2006
Conv. net LeNet-1- + unsup learning		0.60	Ranzato et al. NIPS 2006
<b>Training set augmented with Affine Distortions</b>			
2-layer NN, 100 HU, CE	Affine	1.10	Simard et al., ICDAR 2003
Virtual SVM deg-9 poly	Affine	0.80	Scholkopf
Convolutional net, CE	Affine	0.70	Simard et al., ICDAR 2003
<b>Training set augmented with Elastic Distortions</b>			
2-layer NN, 100 HU, CE	Elastic	0.70	Simard et al., ICDAR 2003
Convolutional net, CE	Elastic	0.80	Simard et al., ICDAR 2003
Conv. net LeNet-1- + 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 “predict” 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:

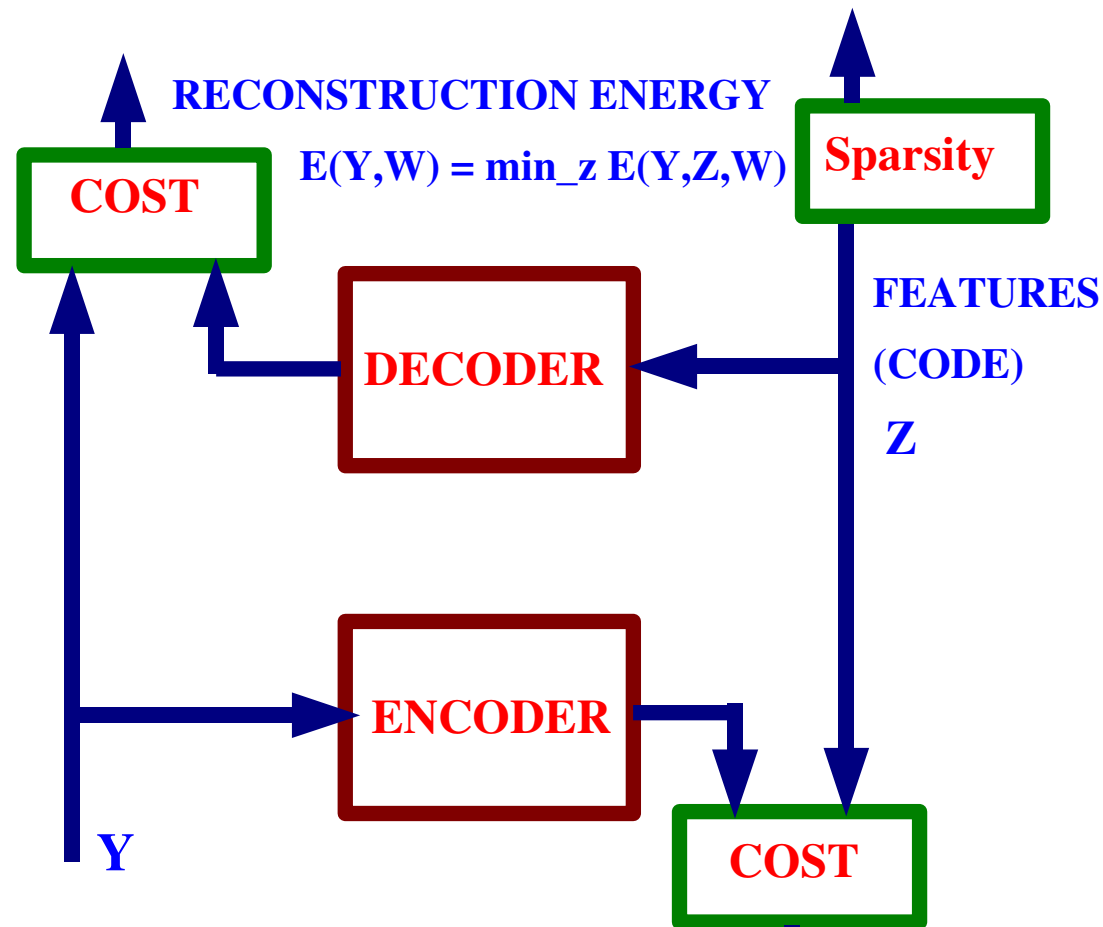
- ▶ None
- ▶ Linear
- ▶ Linear-Sigmoid-Scaling
- ▶ Linear-Sigmoid-Linear

## Sparsity penalty

- ▶ L1

## Main Idea:

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



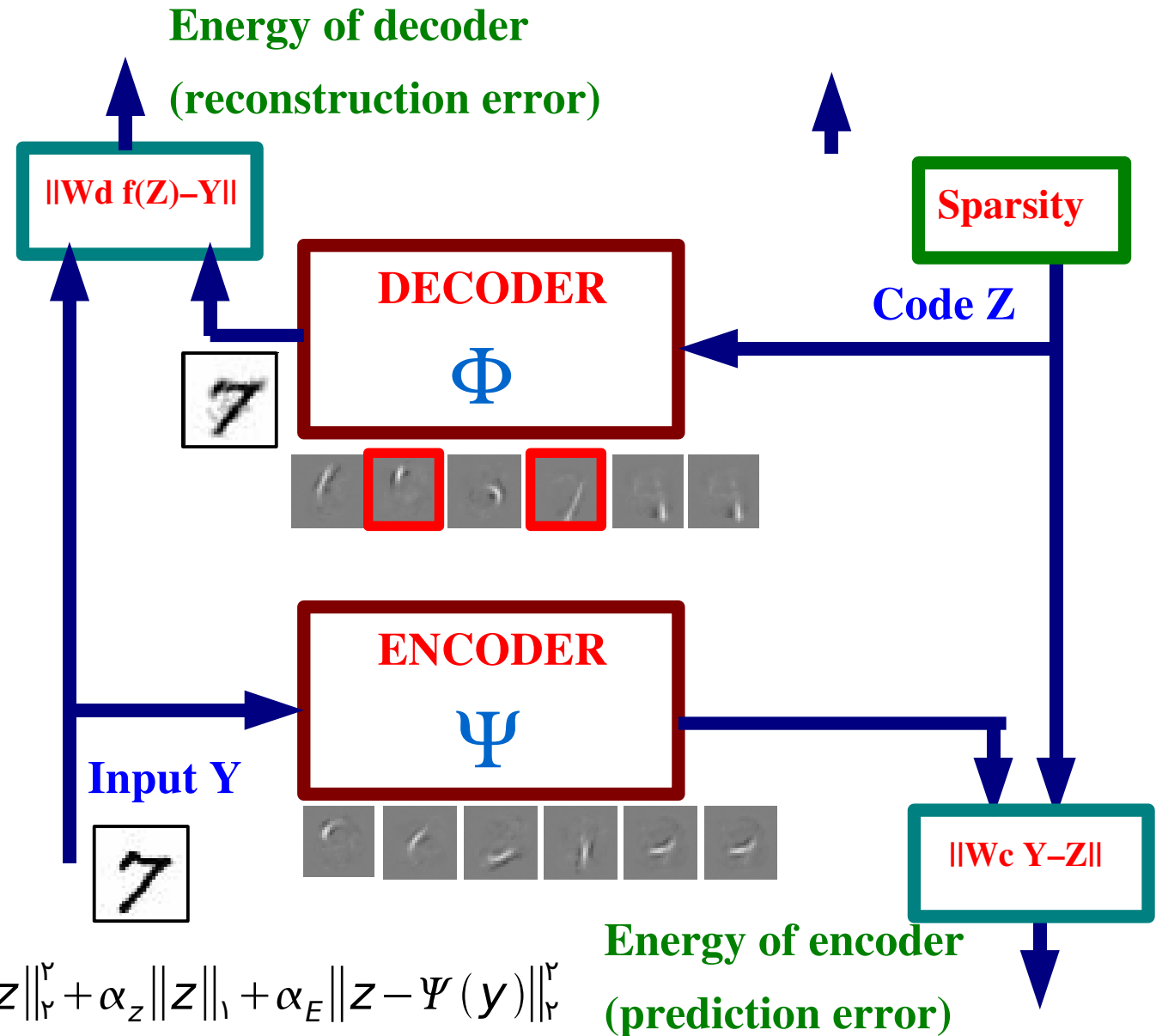
$$\bar{Z}_Y = \operatorname{argmin}_z E(Y, Z, W)$$

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

# 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



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

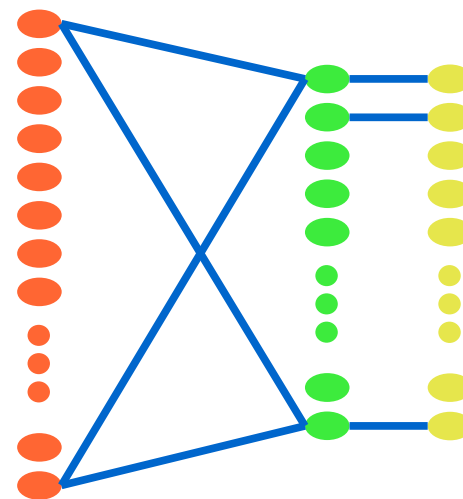
Energy of encoder  
(prediction error)



# Encoder Architectures

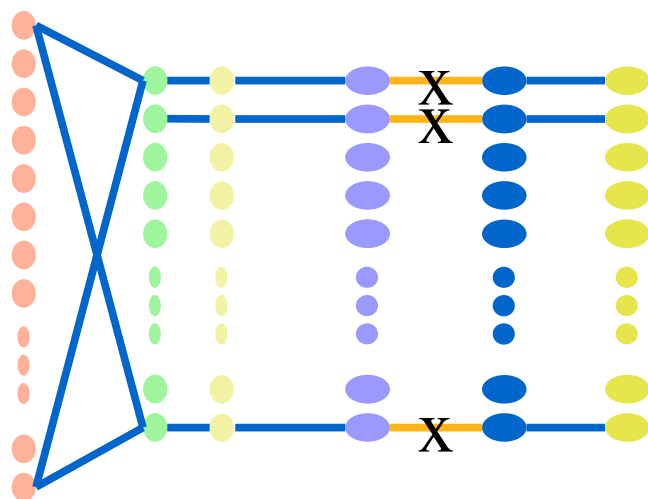
- **L: Linear**
- **FD: Linear + Sigmoid + Gain + Bias**
- **FL: Linear + Sigmoid + Linear**

Linear (L)



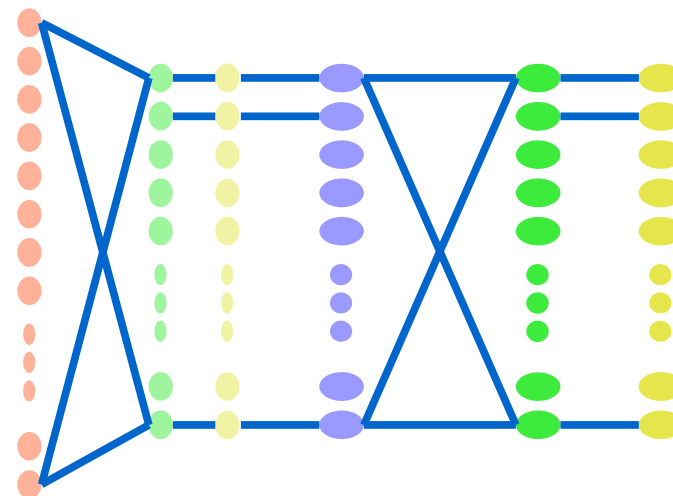
$$Z' = MY + b$$

Non-Linear – Individual gains (FD)



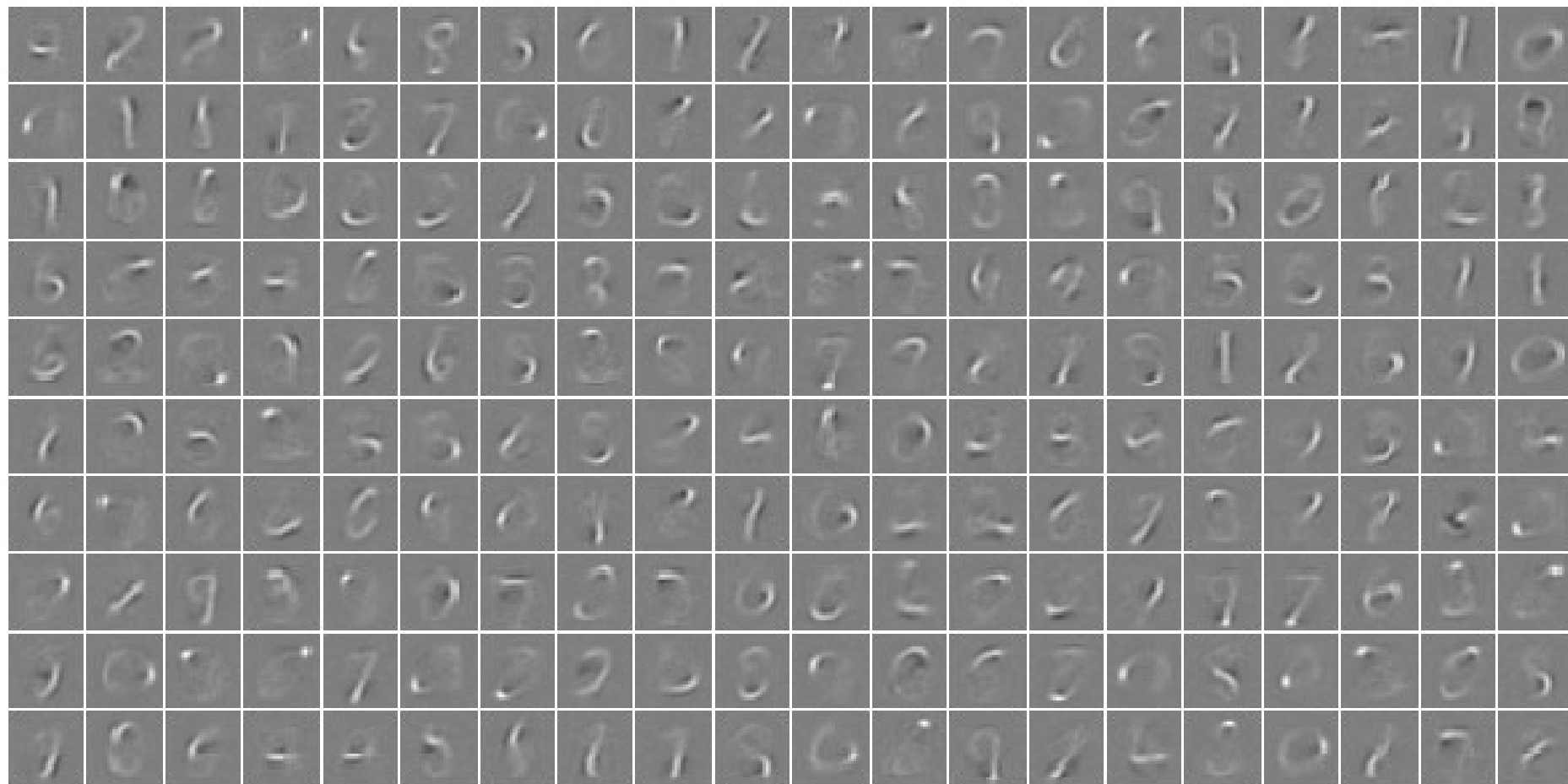
$$Z' = \sigma(MY + b_1) \times \text{diag}(g) + b_2$$

Non-Linear – 2 Layer NN (FL)



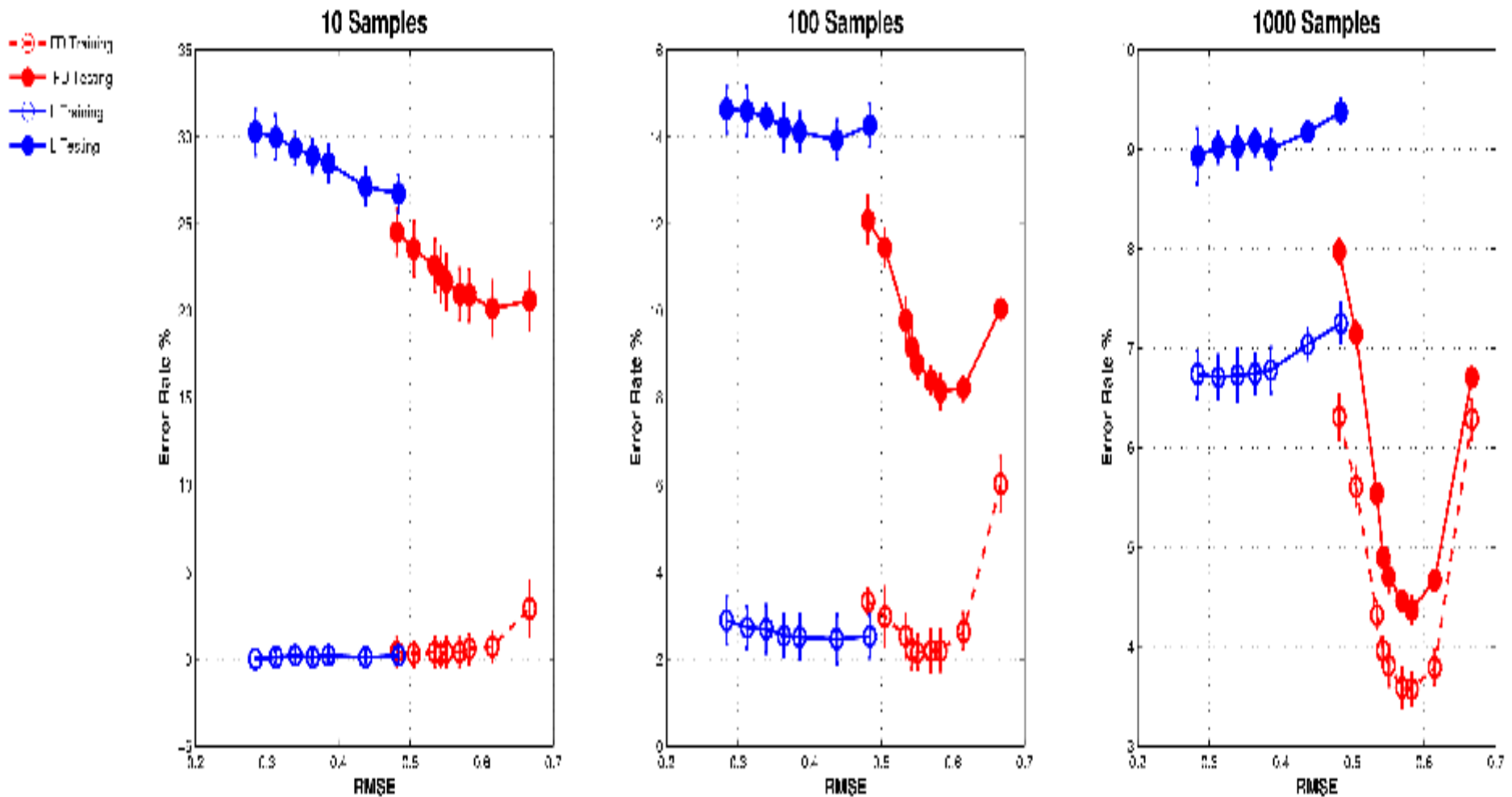
$$Z' = M_2 \sigma(M_1 Y + b_1) + b_2$$

# Decoder Basis Functions on MNIST



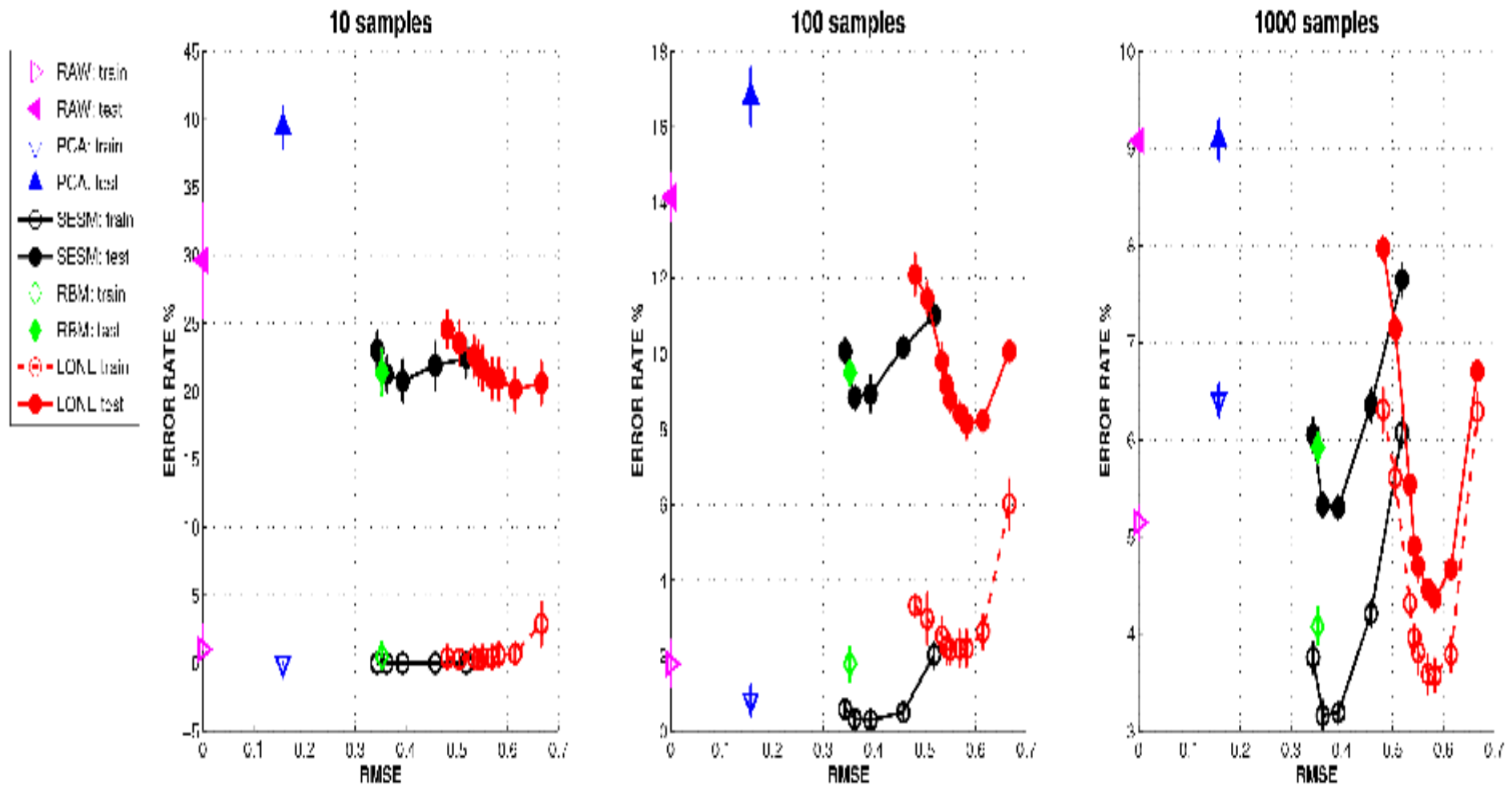
# Classification Error Rate on MNIST

Supervised Linear Classifier trained on 200 trained sparse features

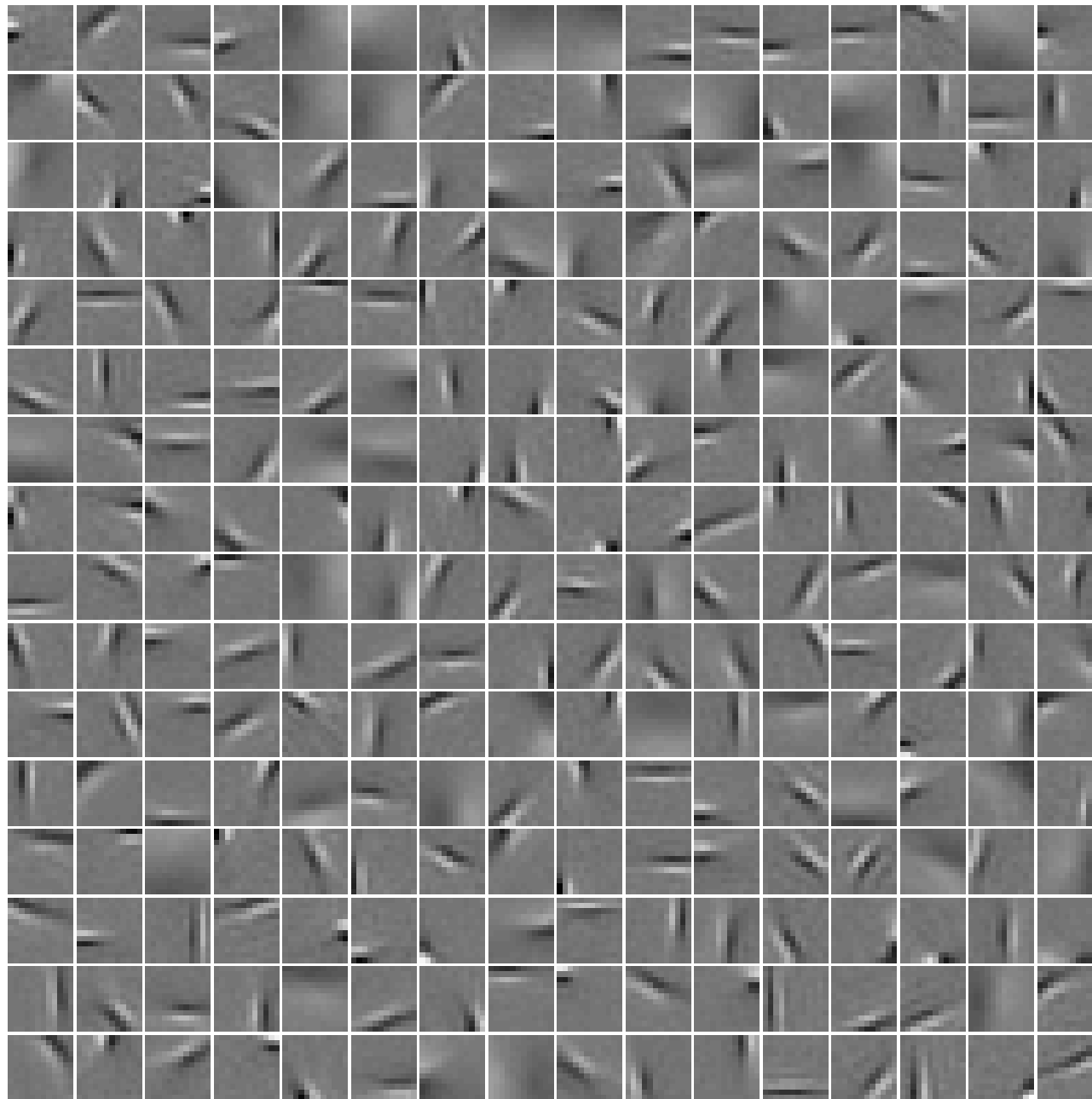


# Classification Error Rate on MNIST

Supervised Linear Classifier trained on 200 trained sparse features

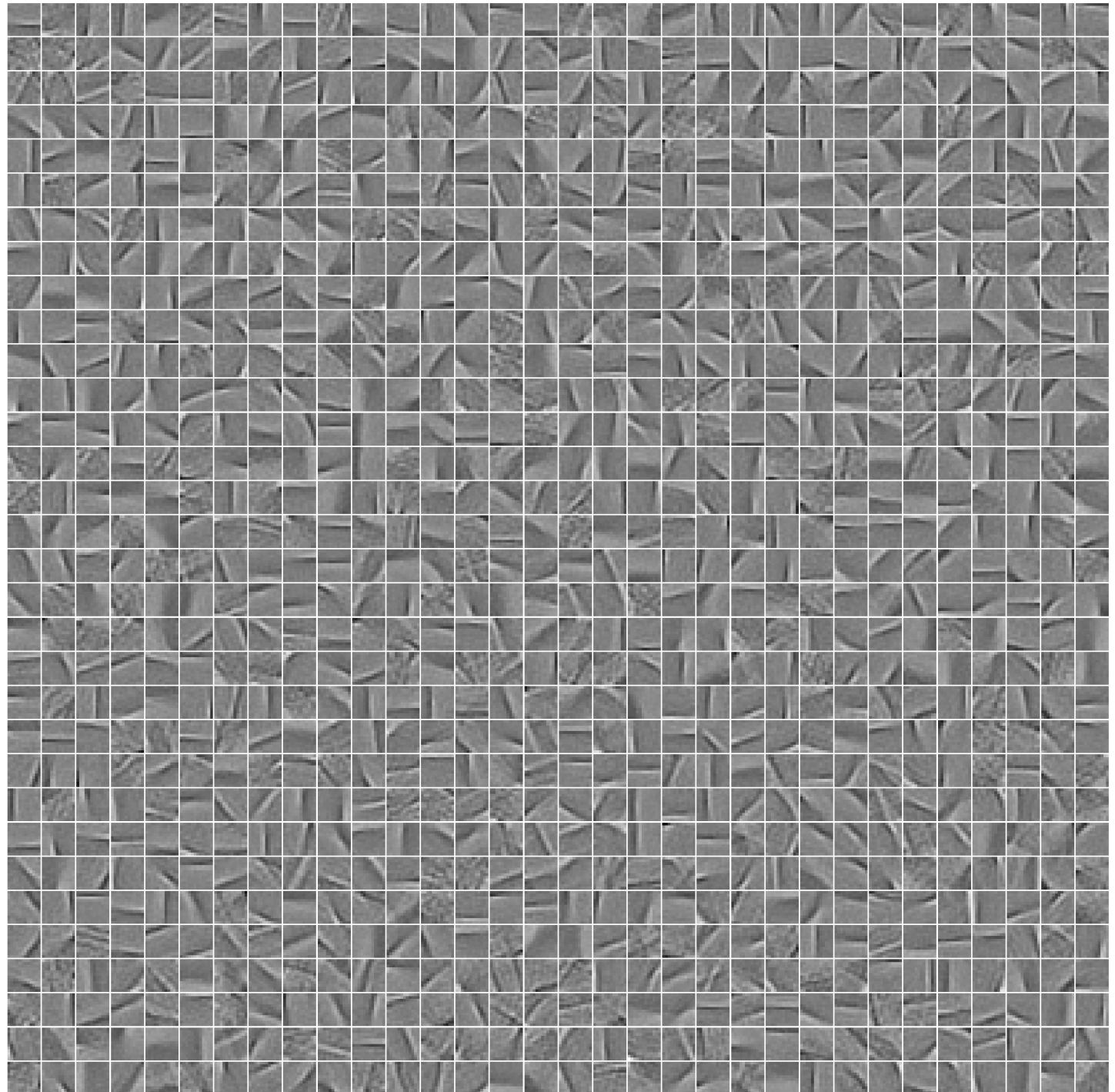


# Learned Features: like V1 receptive fields

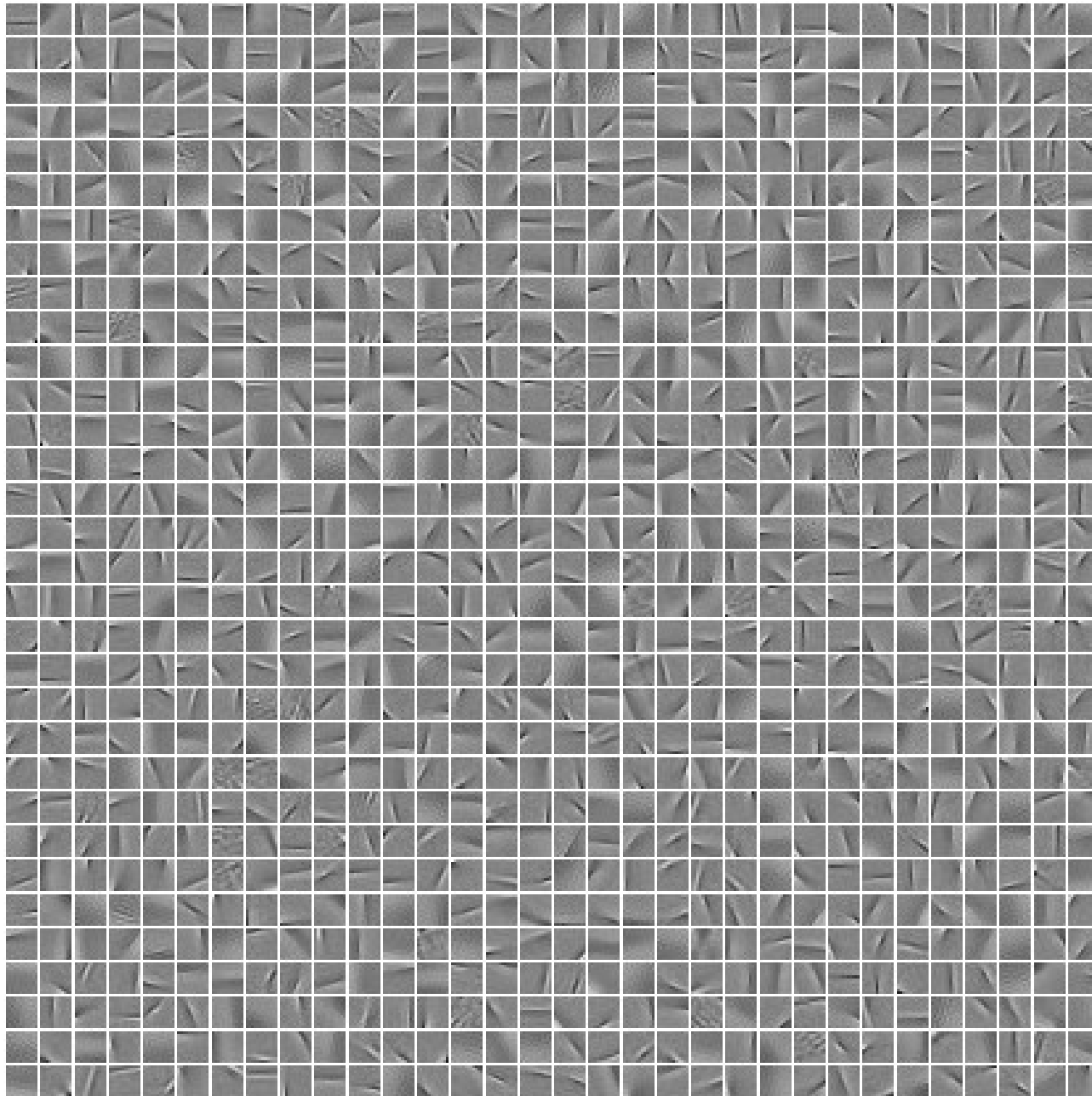


# Training on Natural Image Patches

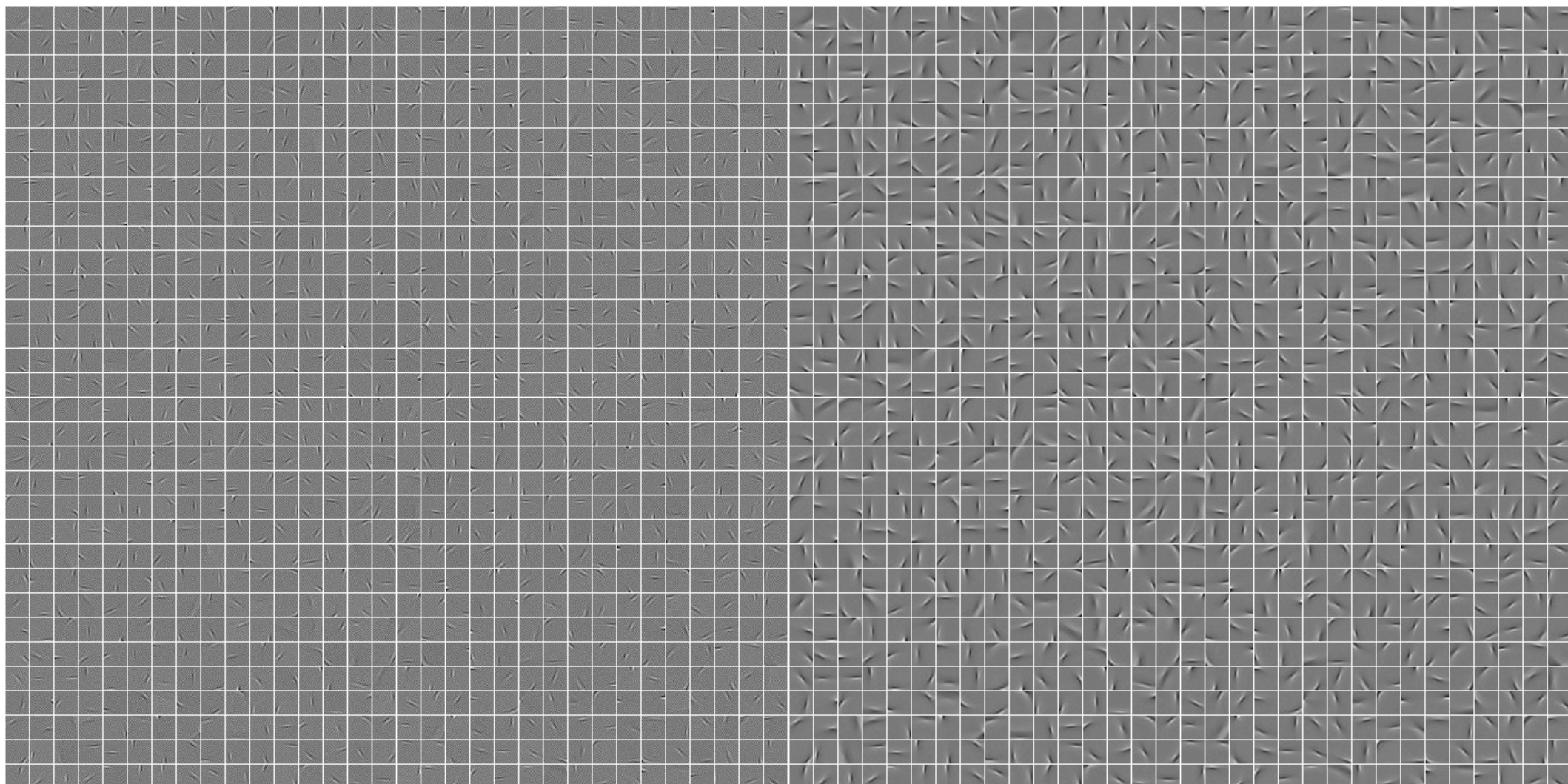
- 12x12 filters
- 1024 filters



# Learned Features: like V1 receptive fields



# Learned Features: like V1 receptive fields

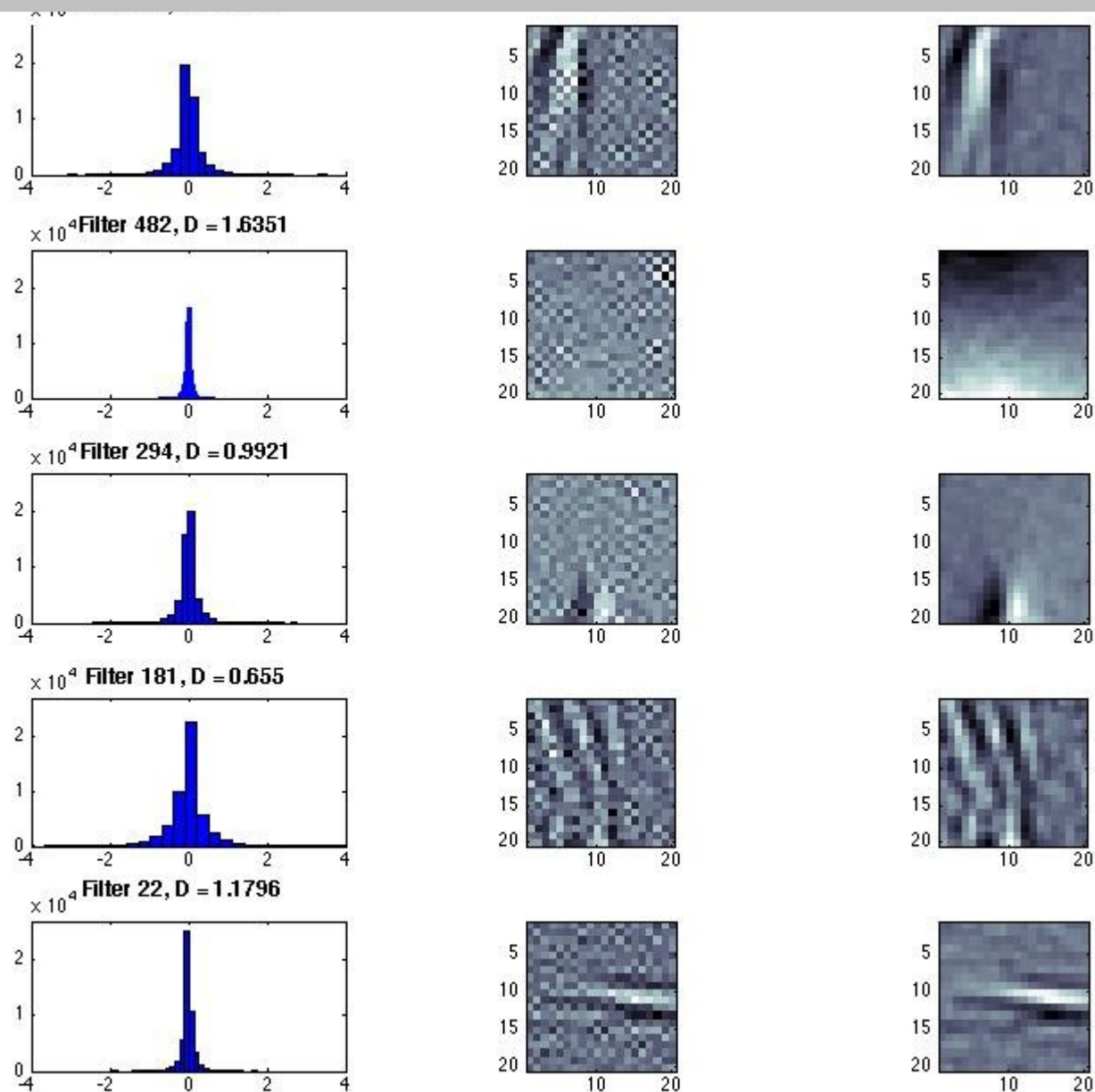


weights :-0,1105 - 0,1141

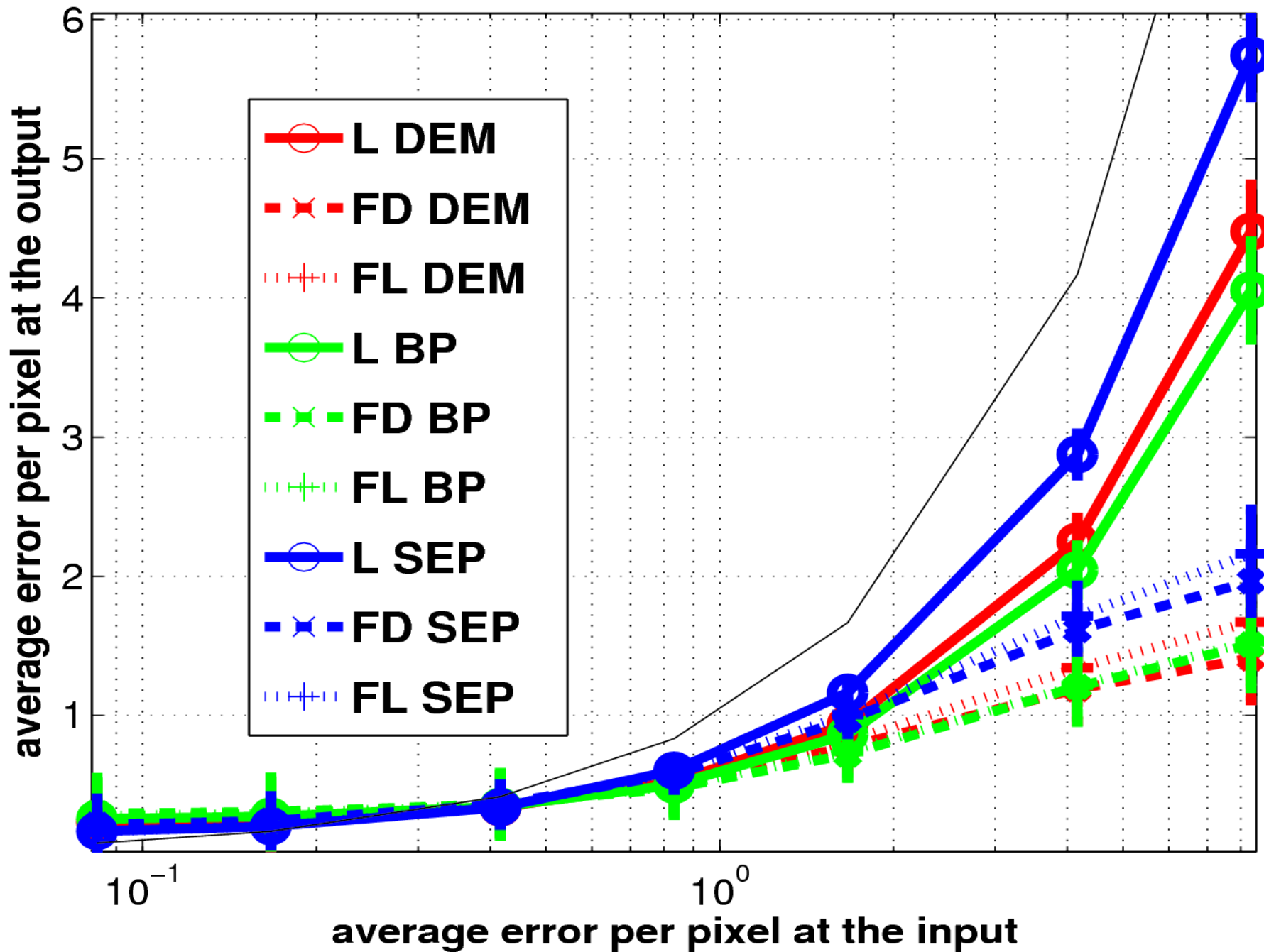
weights :-0,4675 - 0,4977



# Learned Features: like V1 receptive fields



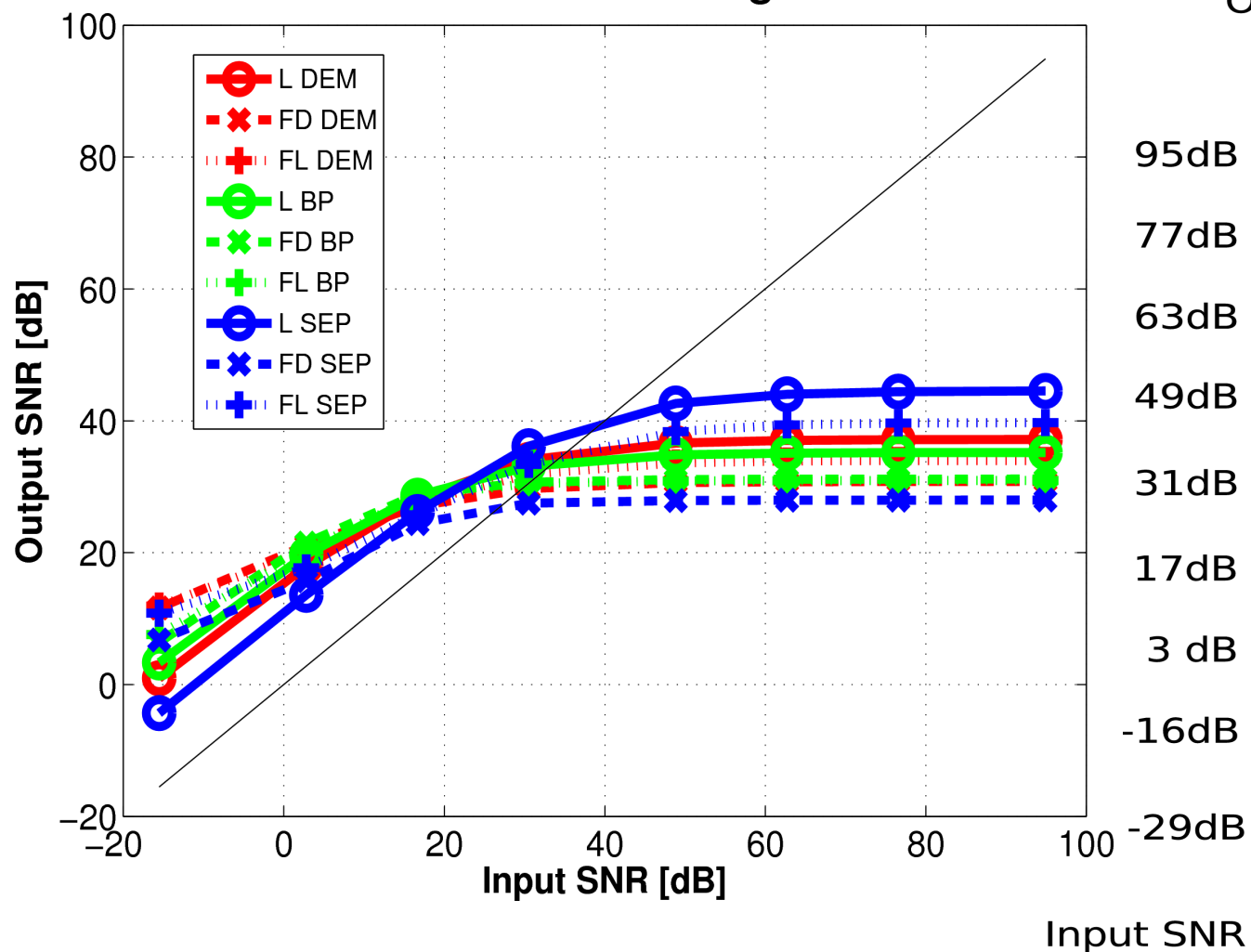
# Noise out versus Noise in (1024 code units)



# Denoising

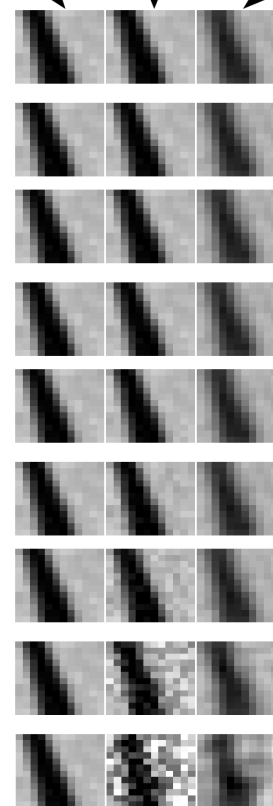
## PBP trained on natural image patches

256 Code Units – logistic



Original Noisy Reconstruction

95dB  
77dB  
63dB  
49dB  
31dB  
17dB  
3 dB  
-16dB  
-29dB



Input SNR



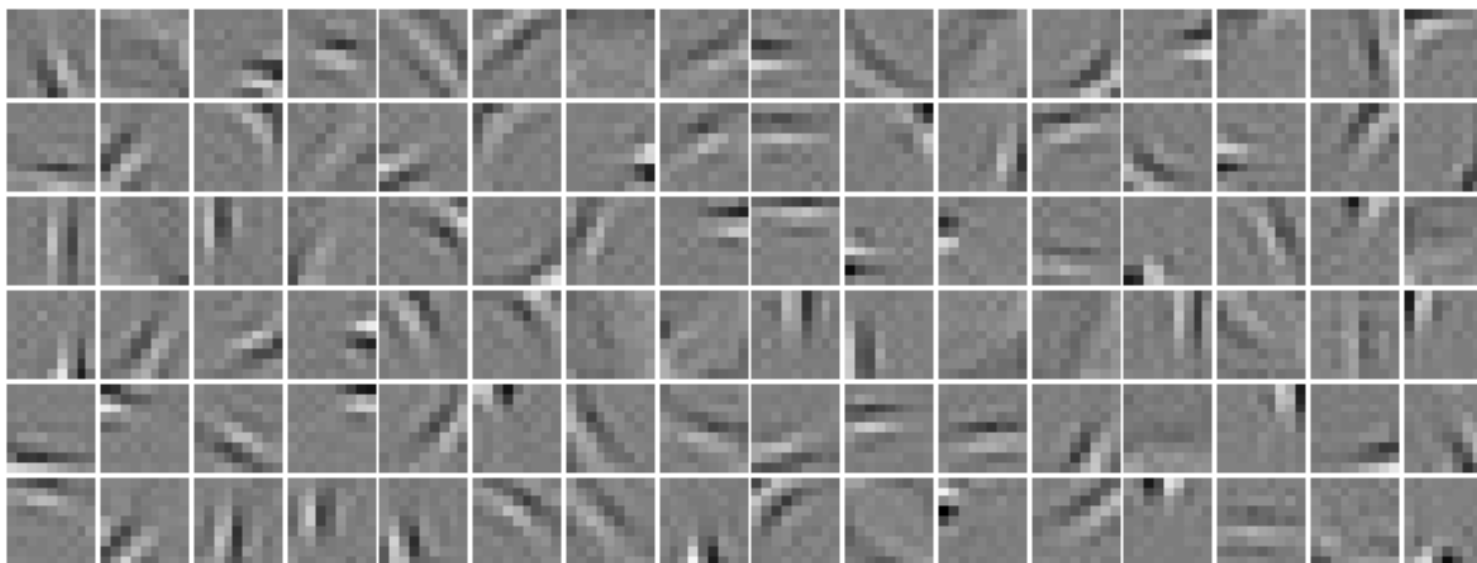
# Using PBP Features for Recognition

## 96 filters on 9x9 patches trained with PBP

- ▶ with Linear-Sigmoid-Gain Encoder

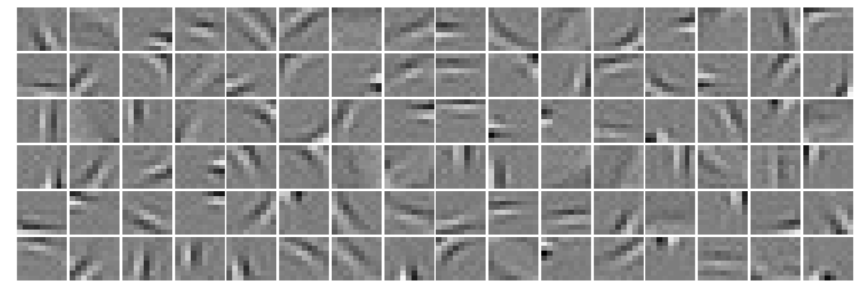
## Recognition:

- ▶ Normalized\_Image -> Learned\_Filters -> Rectification -> Local\_Normalization -> Spatial\_Pooling -> PCA -> Linear\_Classifier
- ▶ What is the effect of rectification and normalization?



weights  $\pm 0.9275 - 0.8688$

# Caltech-101 Recognition Rate



weights  $\pm 0.9275 - 0.8688$

## ● [96\_Filters->Rectification]->Pooling->PCA->Linear\_Classifier

- ▶ [Filters->Sigmoid] 16%
- ▶ [Filters->Absolute\_Value] 51%
- ▶ [Local\_Norm->Filters->Absolute\_Value] 56%
- ▶ [Local\_Norm->Filters->Absolute\_Value->Local\_Norm] 58%

## ● Multi-Scale Filters->Rectification->Pooling->PCA->Linear\_Classifier

- ▶ LN->Gabor\_Filters->Rectif->LN (Pinto&diCarlo 08) 59%

## ● Unsupervised Convolutional Net

- ▶ Filt->Sigm->Pooling->Filt->Sigm->Pooling->Classifier 54%

## ● Supervised Convolutional Net

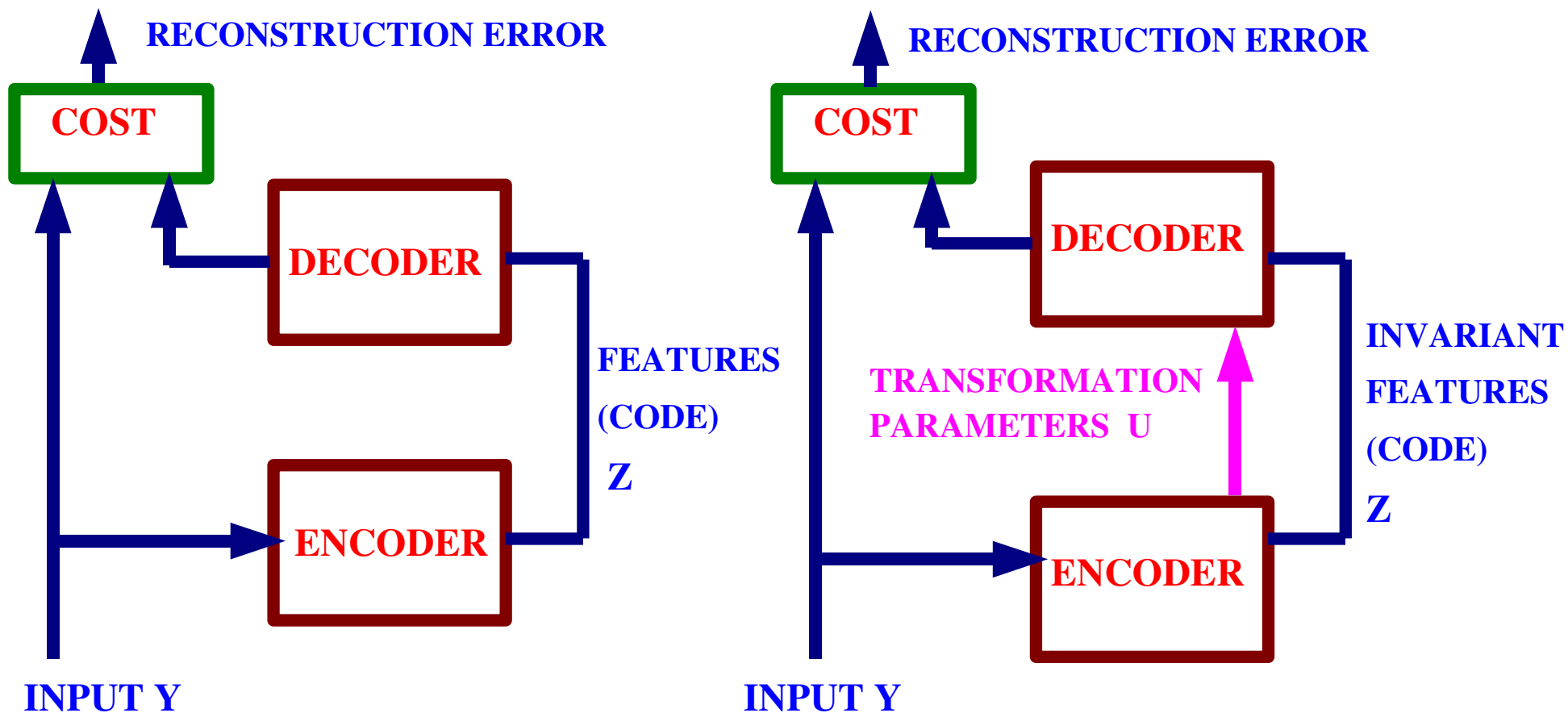
- ▶ Filt->Sigm->Pooling->Filt->Sigm->Pooling->Classifier 20%

## ● HMAX (Serre -> Mutch&Lowe 06)

- ▶ Filt->Sigm->Pooling->Filt->Sigm->Pooling->Classifier 56%

# Learning Invariant Features

- Separating the “what” from the “where”



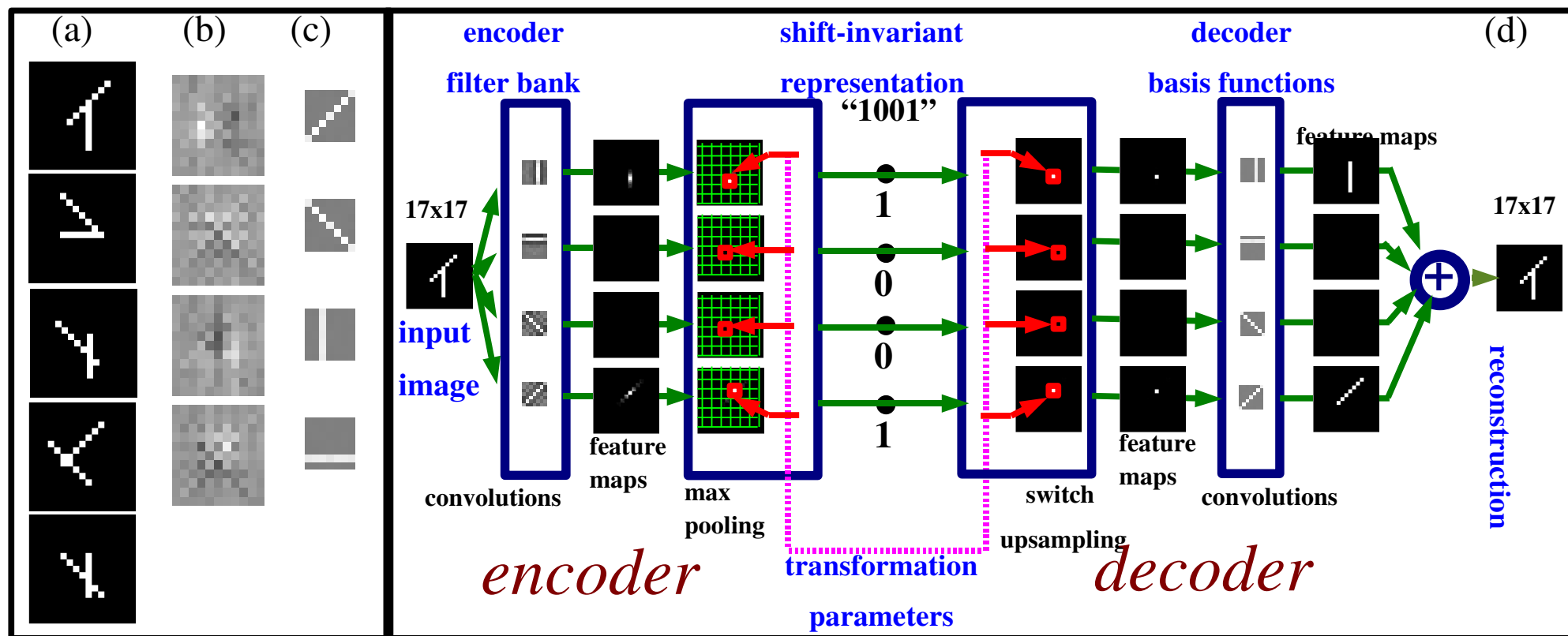
*Standard Feature Extractor*

*Invariant Feature Extractor*

# Learning Invariant Feature Hierarchies

## Learning Shift Invariant Features

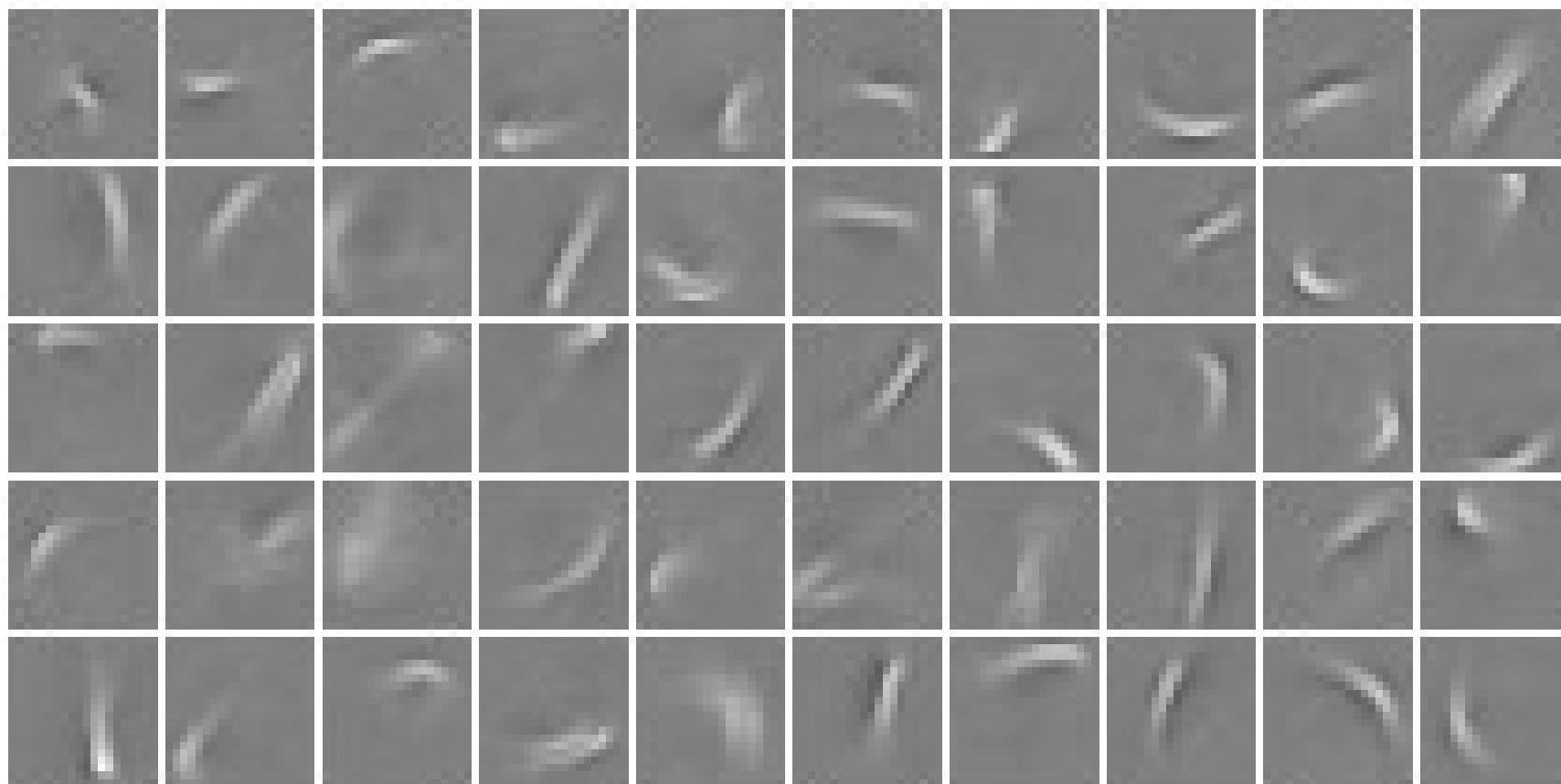
image  $\rightarrow$  filters  $\rightarrow$  pooling  $\rightarrow$  switches  $\rightarrow$  bases  $\rightarrow$  reconstruction



# Shift Invariant Global Features on MNIST

## ■ Learning 50 Shift Invariant Global Features on MNIST:

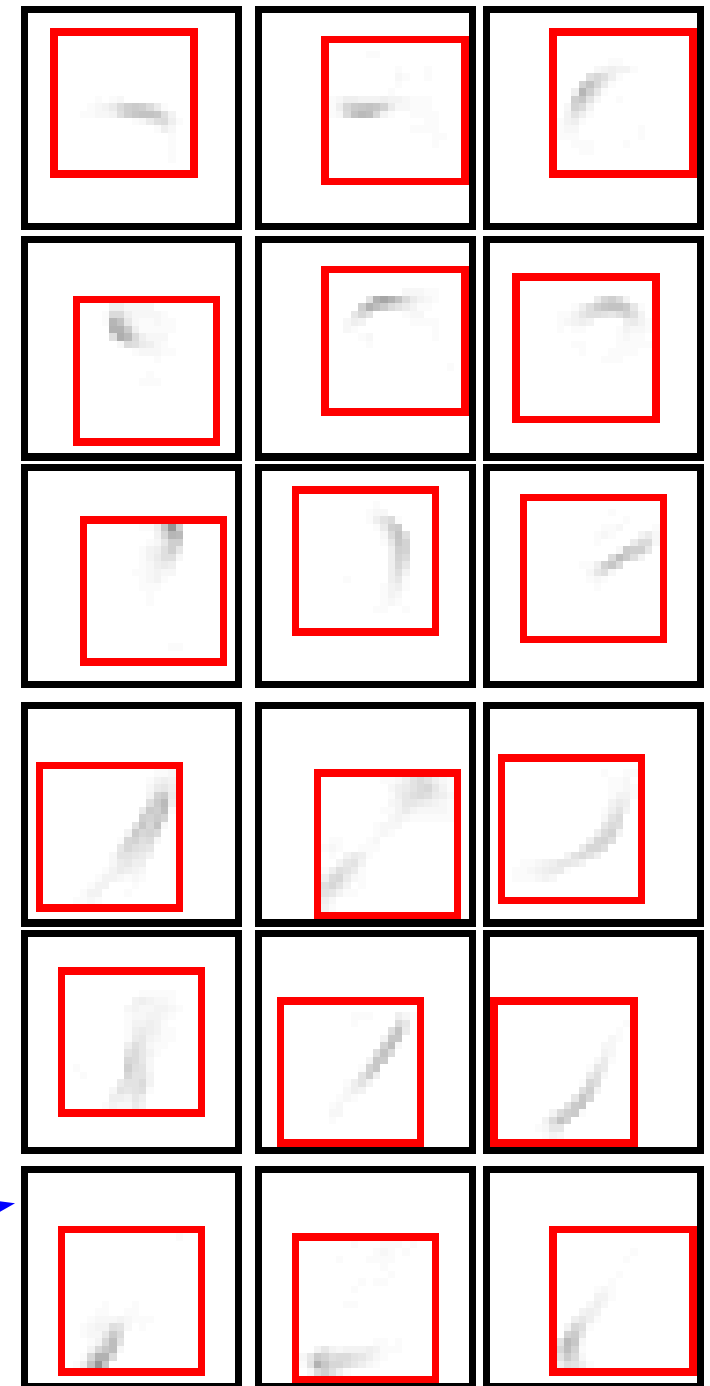
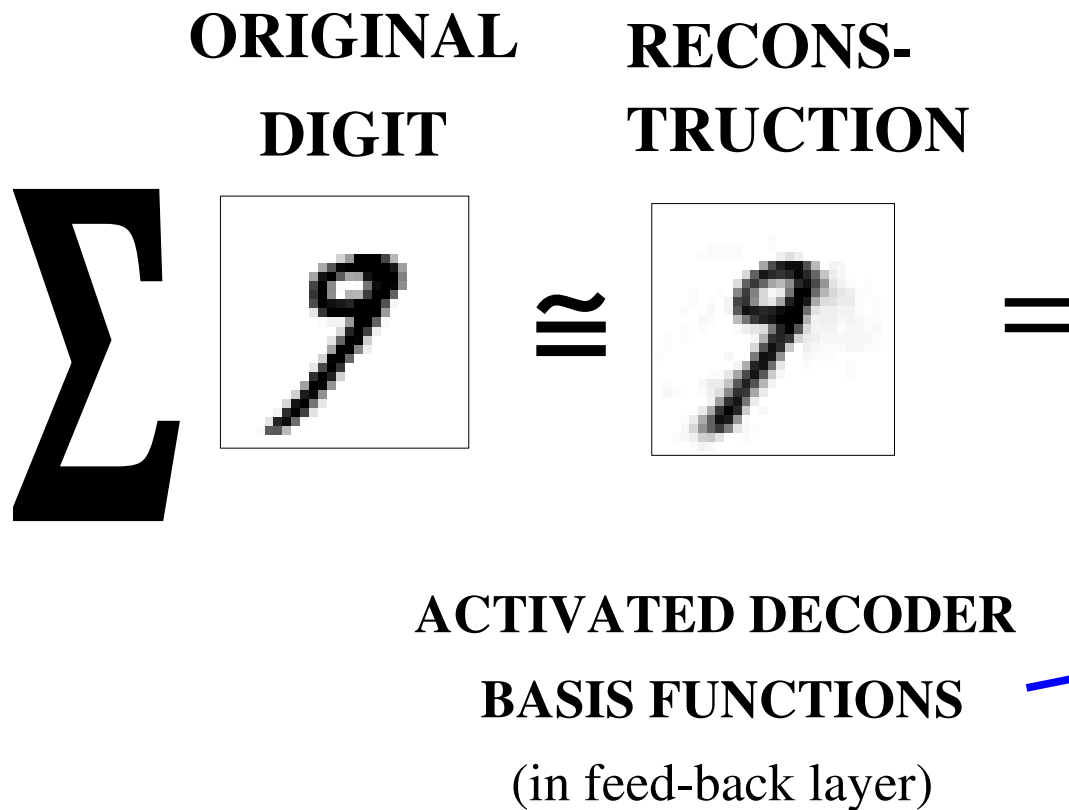
- ▶ 50 filters of size 20x20 movable in a 28x28 frame (81 positions)
- ▶ movable strokes!





# Example of Reconstruction

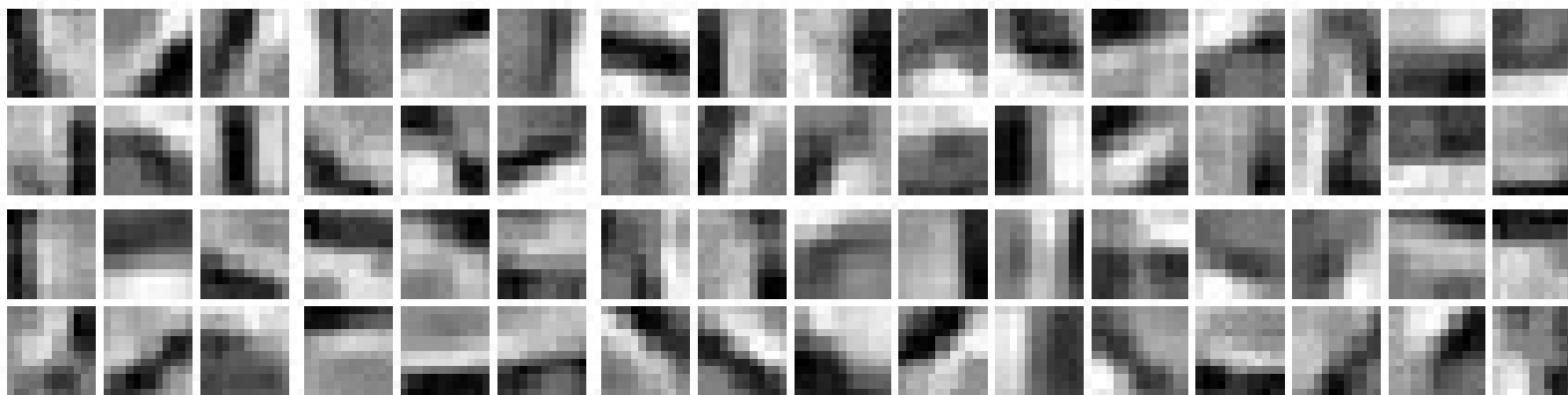
- Any character can be reconstructed as a linear combination of a small number of basis functions.



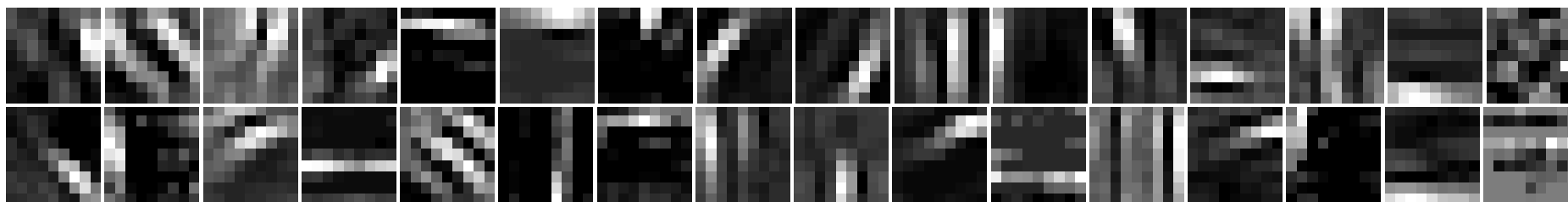
red squares: decoder bases

# Sparse Enc/Dec on Object Images

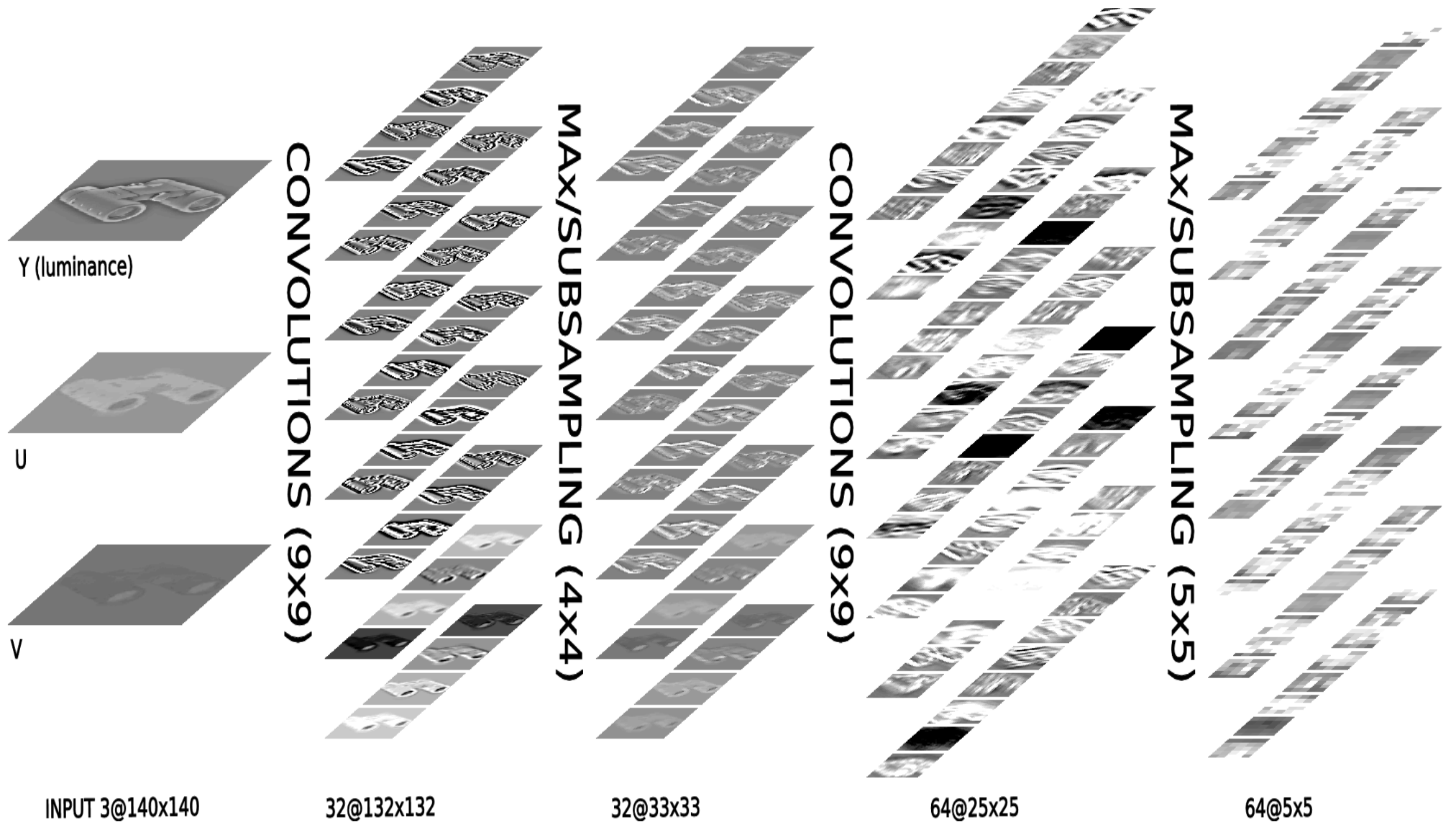
## 9x9 filters at the first level



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

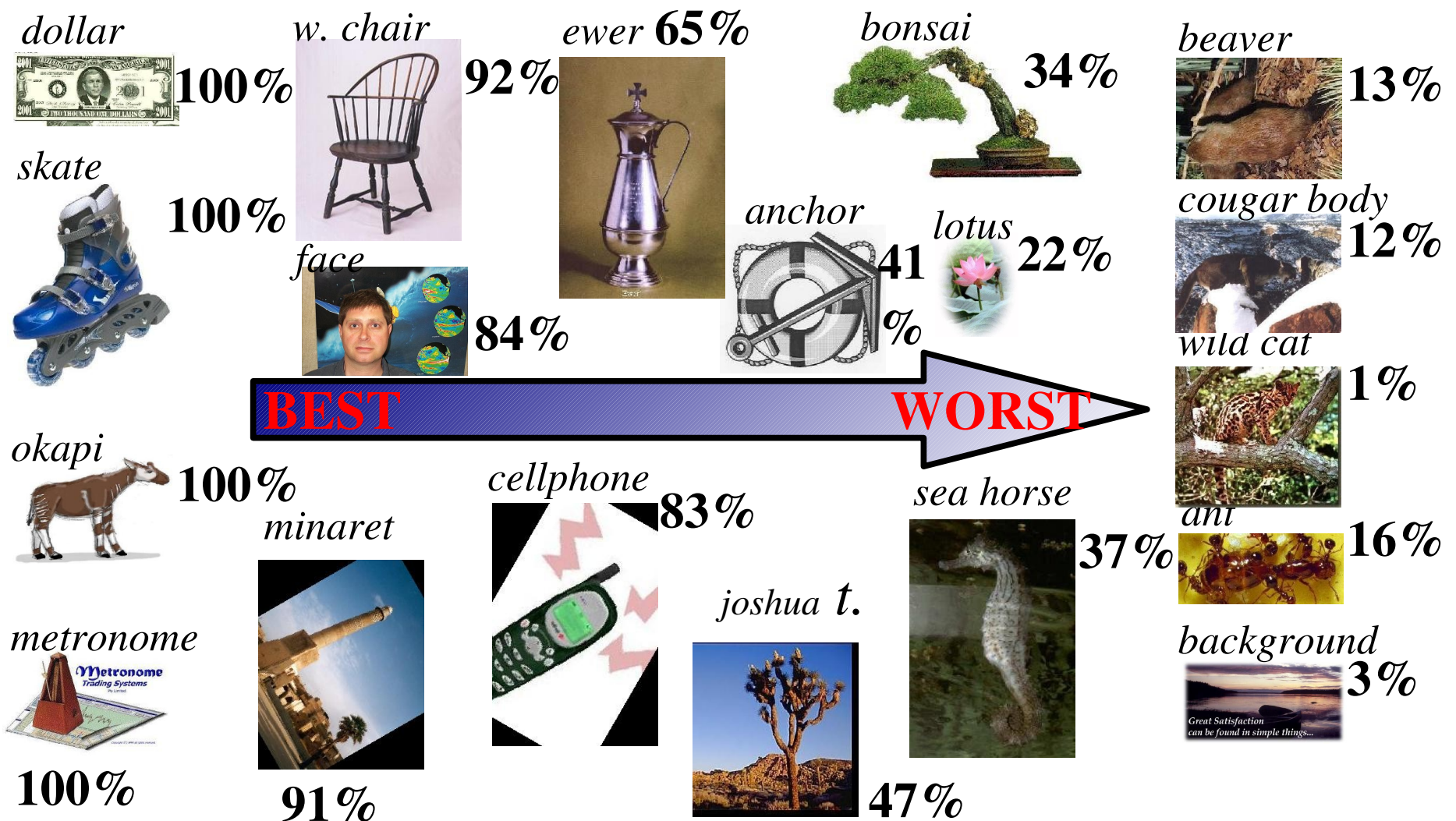


# Feature Hierarchy for Object Recognition



# Recognition Rate on Caltech 101

■ 54% on Caltech-101 (only 20% with purely supervised backprop)

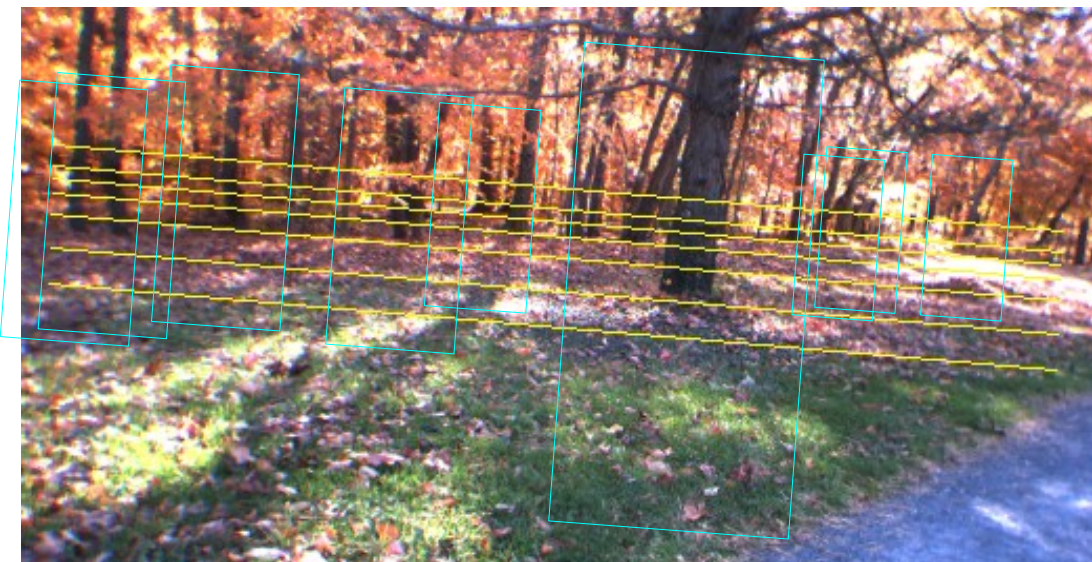


# LAGR: Learning Applied to Ground Robotics

- Getting a robot to drive autonomously in unknown terrain solely from vision (camera input).
- Our team (NYU/Net-Scale Technologies Inc.) is one of 8 participants funded by DARPA
- All teams received identical robots and can only modify the software (not the hardware)
- 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.

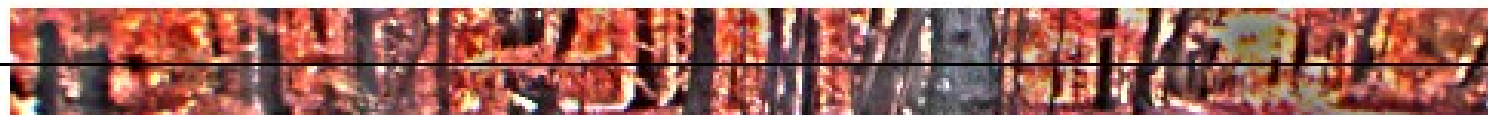


# Long Range Vision: Distance Normalization



## Pre-processing (125 ms)

- Ground plane estimation
- Horizon leveling
- Conversion to YUV + local contrast normalization
- Scale invariant pyramid of distance-normalized image “bands”



112.3m to INF, scale: 1.0



50.7m to INF, scale: 1.4



24.2m to INF, scale: 1.9



13.8m to 86.8m, scale: 2.6



9.0m to 34.5m, scale: 3.5



5.8m to 17.6m, scale: 5.0



4.1m to 11.3m, scale: 6.7

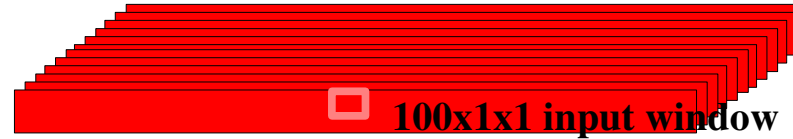
# Convolutional Net Architecture

- Operates on 12x25 YUV windows from the pyramid

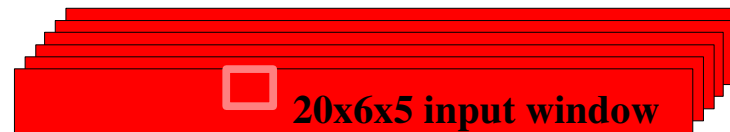


Logistic regression 100 features -> 5 classes

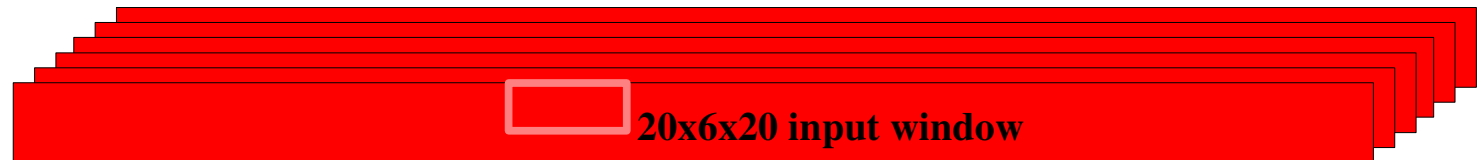
100 features per  
3x12x25 input window



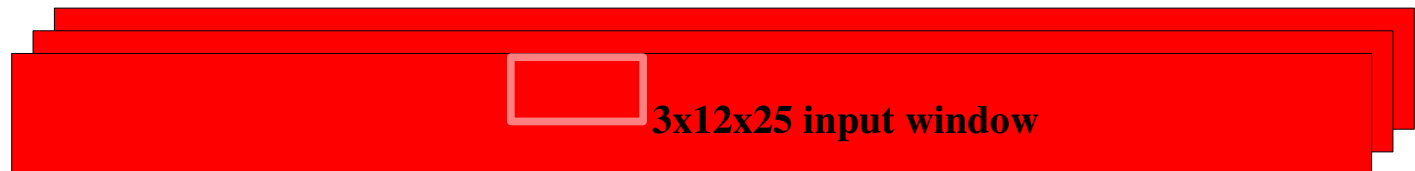
Convolutions with 6x5 kernels



Pooling/subsampling with 1x4 kernels



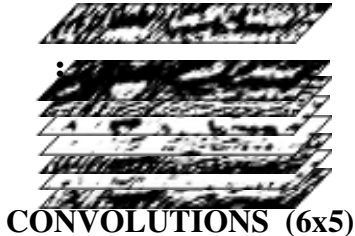
Convolutions with 7x6 kernels



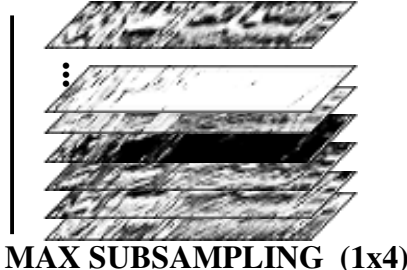
YUV image band  
20-36 pixels tall,  
36-500 pixels wide

# Convolutional Net Architecture

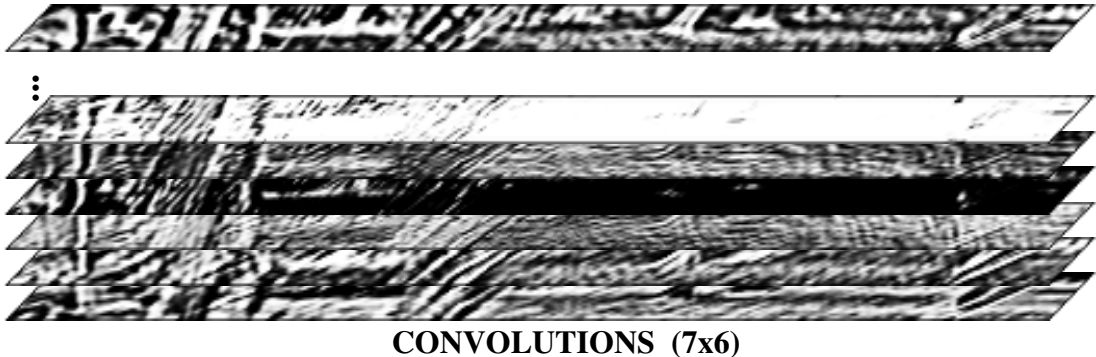
100@25x121



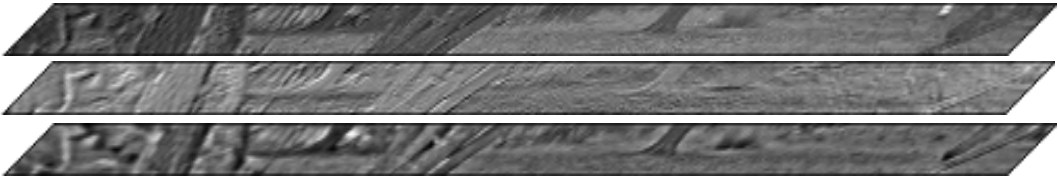
20@30x125



20@30x484



3@36x484



YUV input

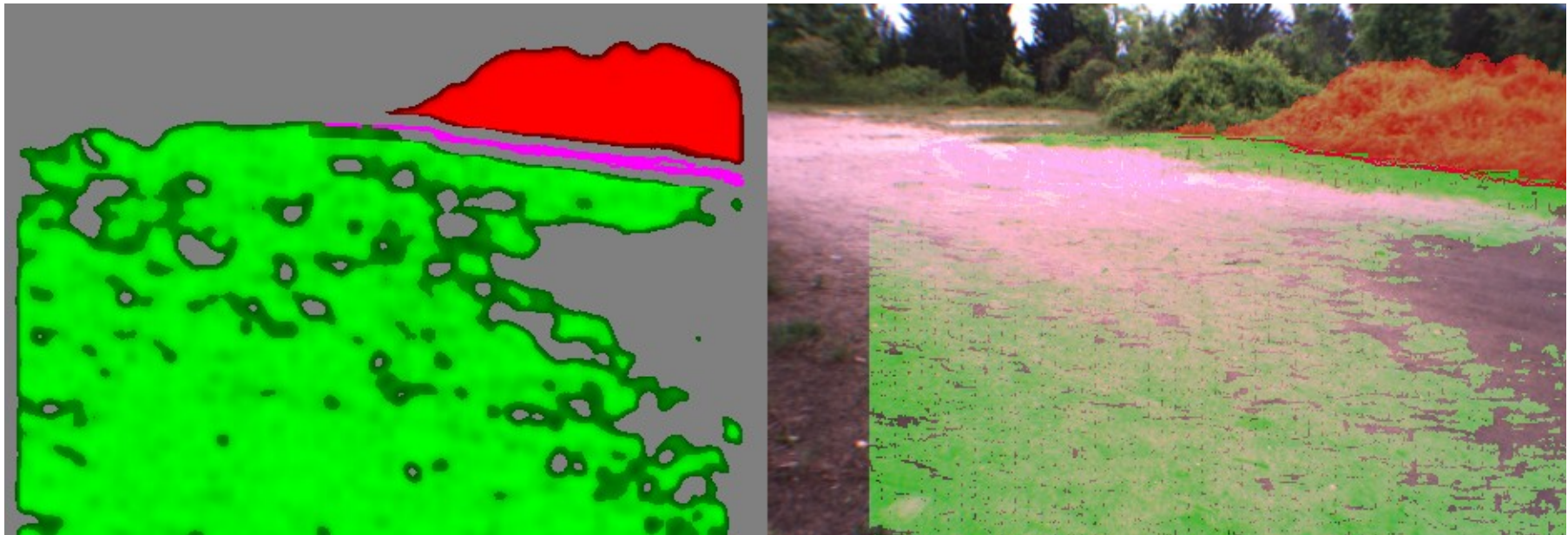




# Long Range Vision: 5 categories

## Online Learning (52 ms)

- Label windows using stereo information – 5 classes



**super-ground**



**ground**



**footline**



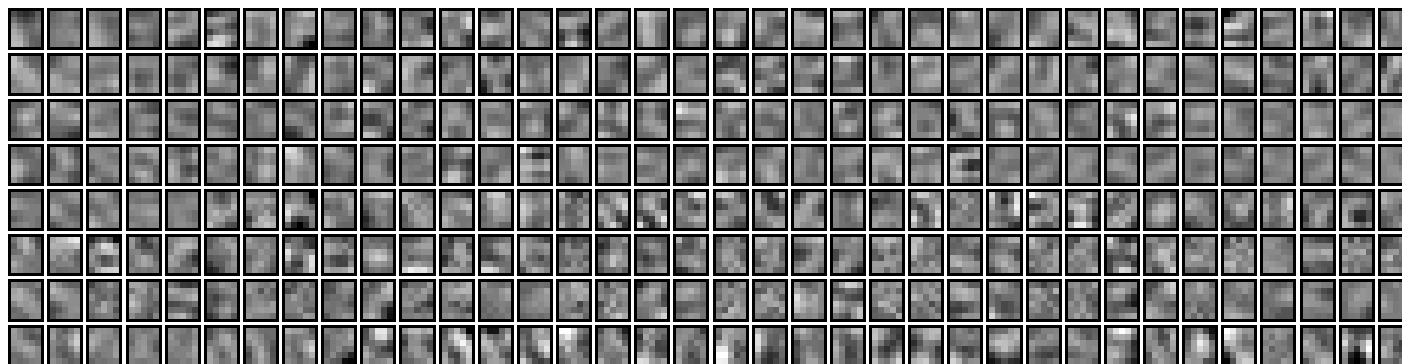
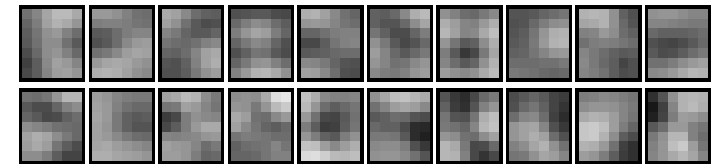
**obstacle**



**super-obstacle**

# Trainable Feature Extraction

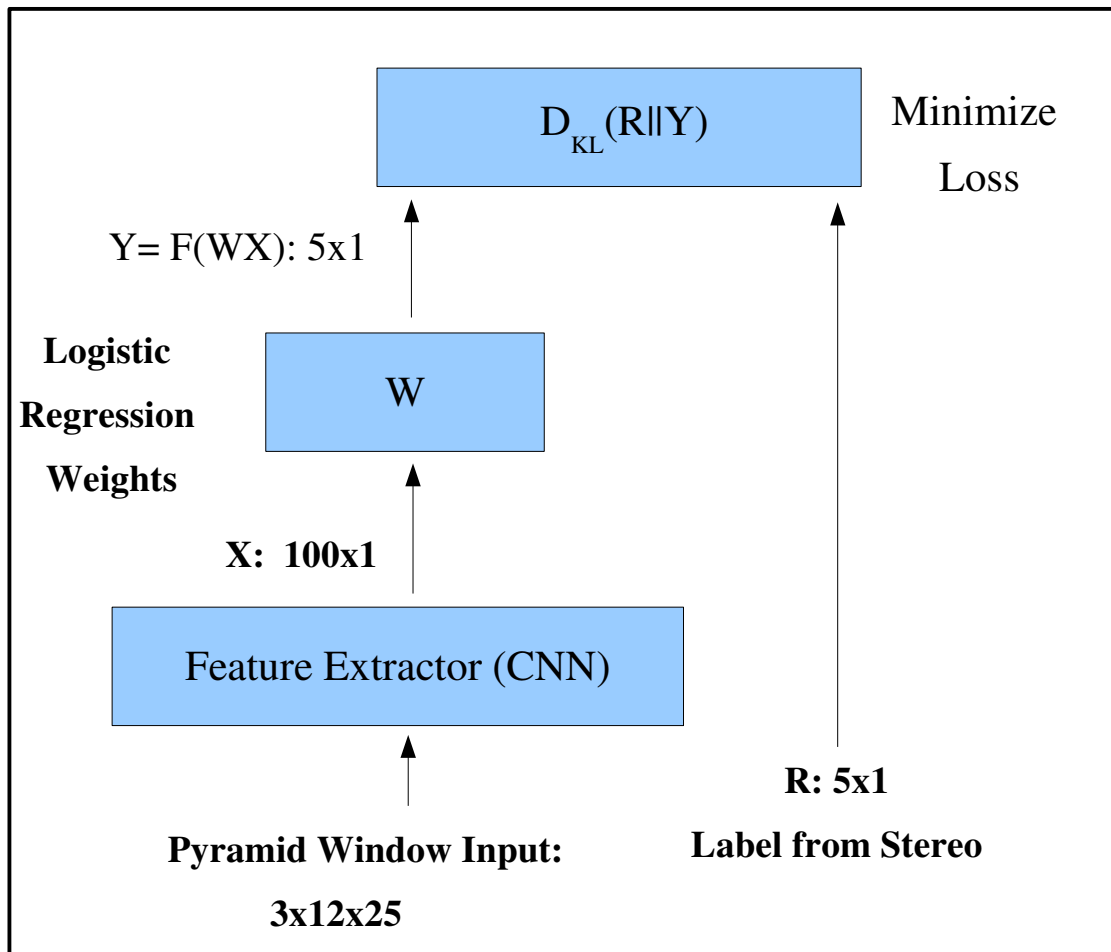
- “Deep belief net” approach to unsupervised feature learning
- Two stages are trained in sequence
  - each stage has a layer of convolutional filters and a layer of horizontal feature pooling.
  - Naturally shift invariant in the horizontal direction
- Filters of the convolutional net are trained so 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

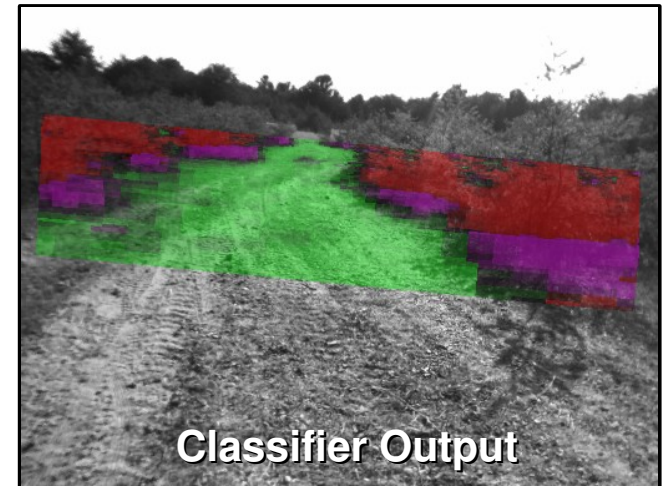
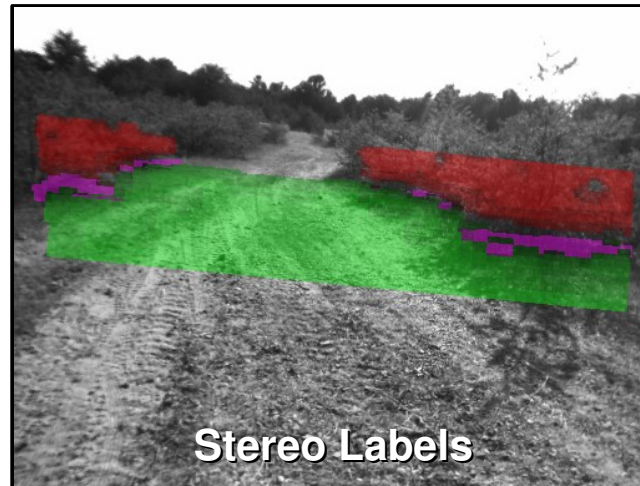
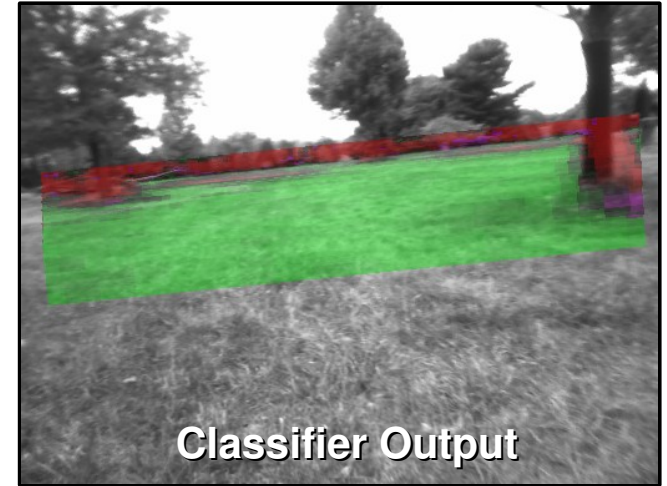
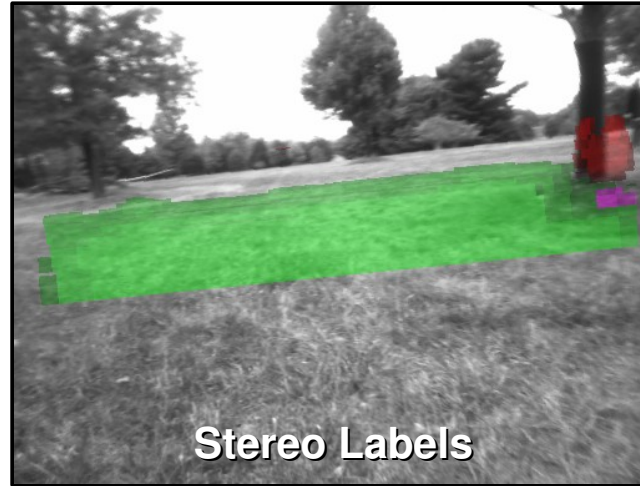
## Online Learning (52 ms)

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

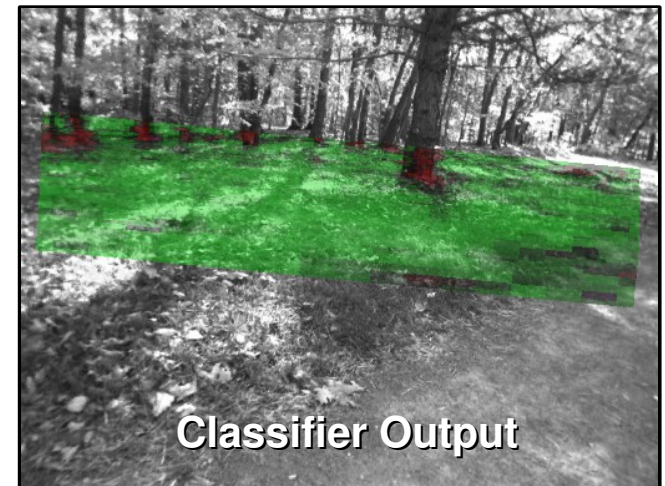
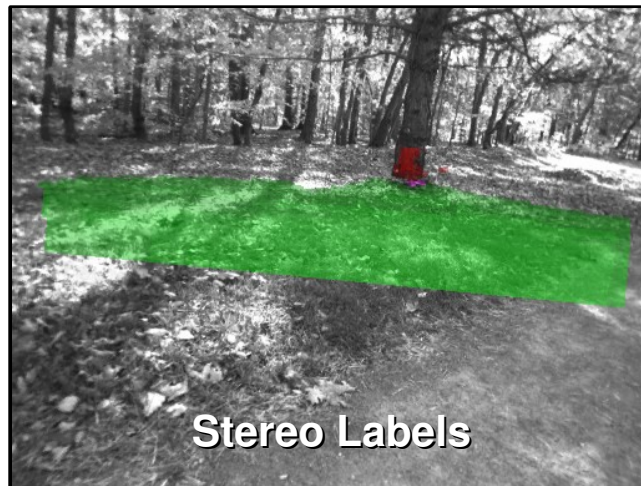
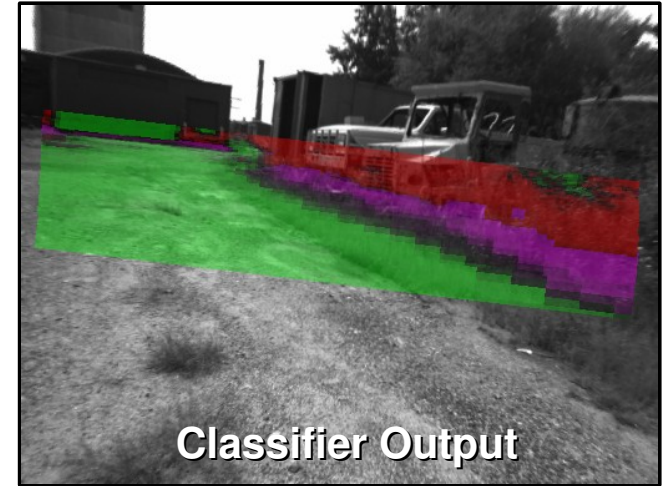
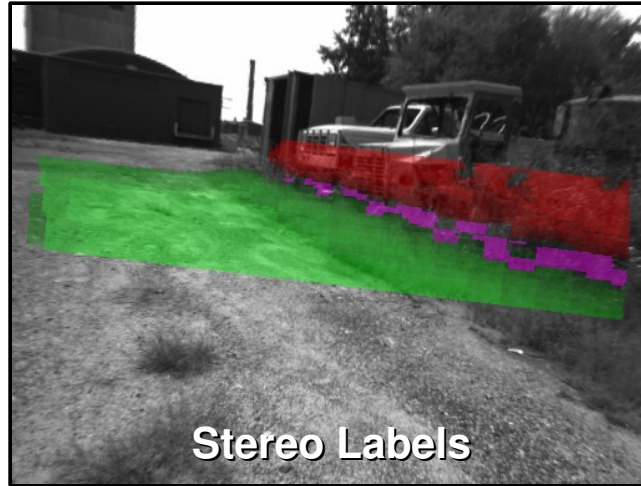


- 5 categories are learned
- 750 samples of each class are kept in a ring buffer: short term memory.
- Learning “snaps” to new environment in about 10 frames
- Weights are trained with stochastic gradient descent
- Regularization by decay to default weights

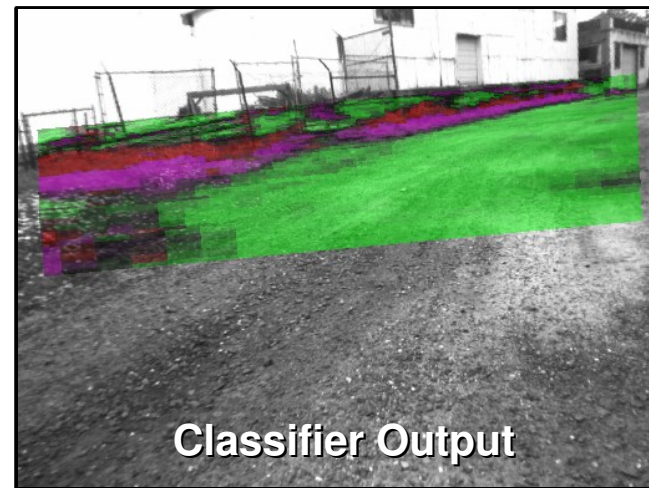
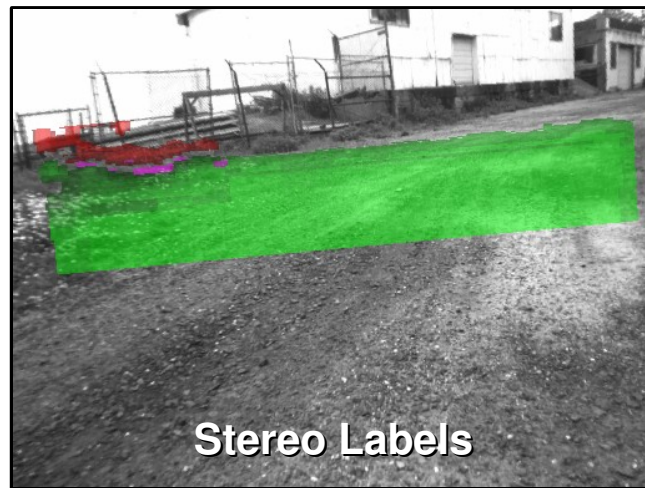
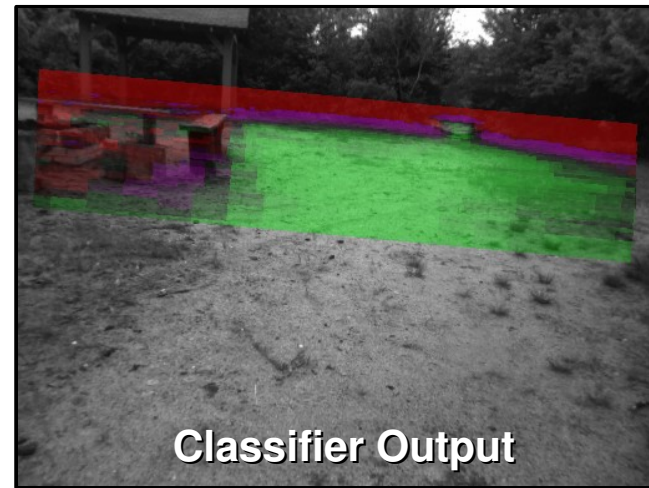
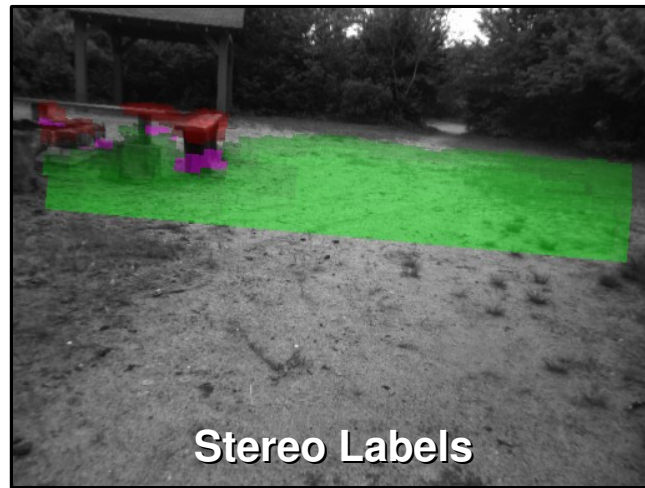
# Long Range Vision Results

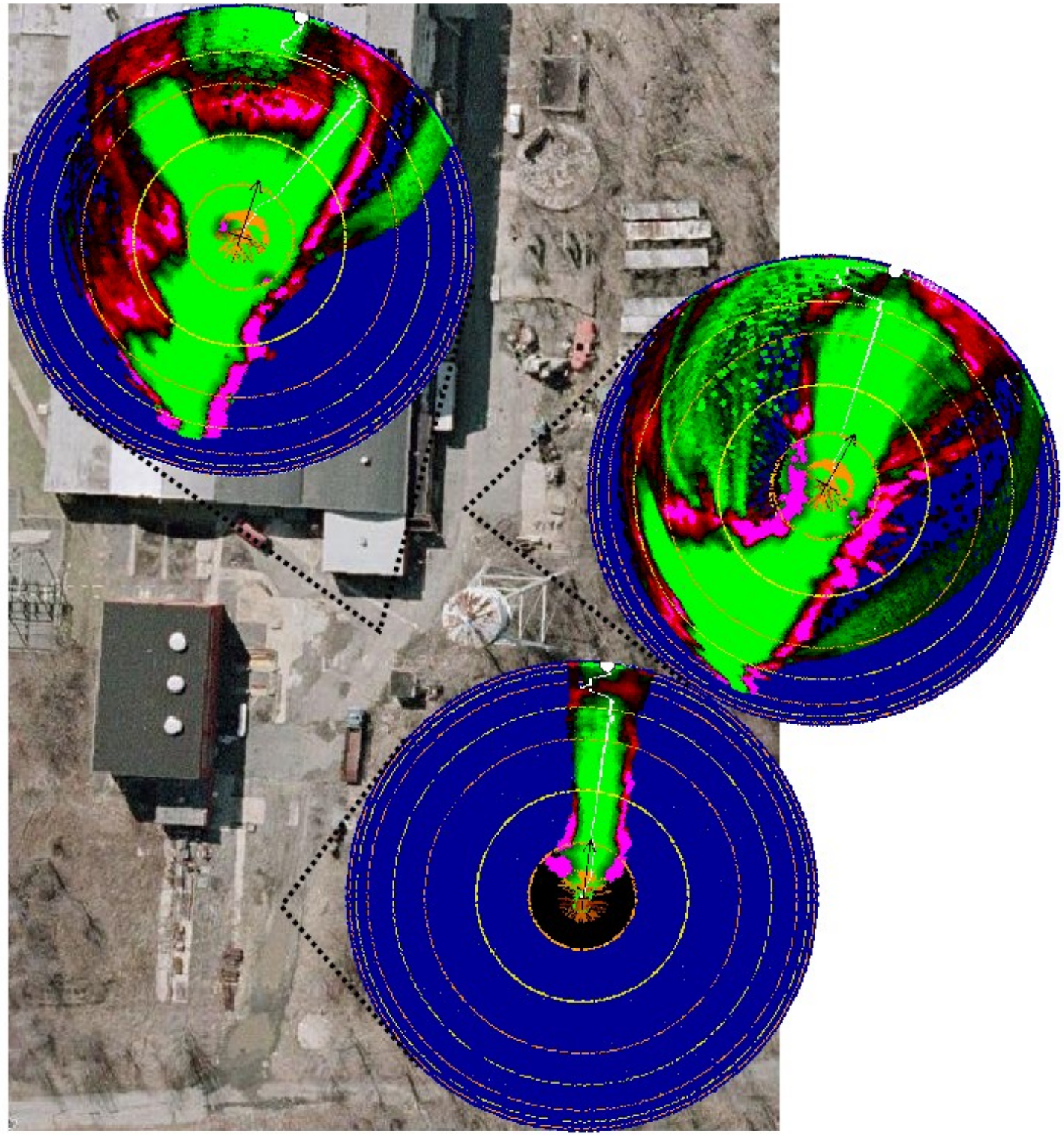


# Long Range Vision Results



# Long Range Vision Results



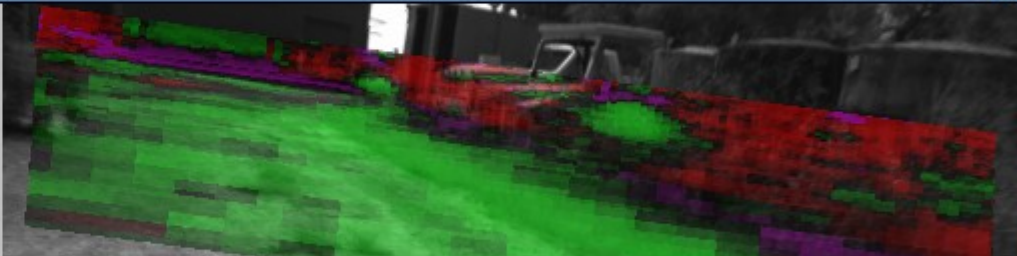
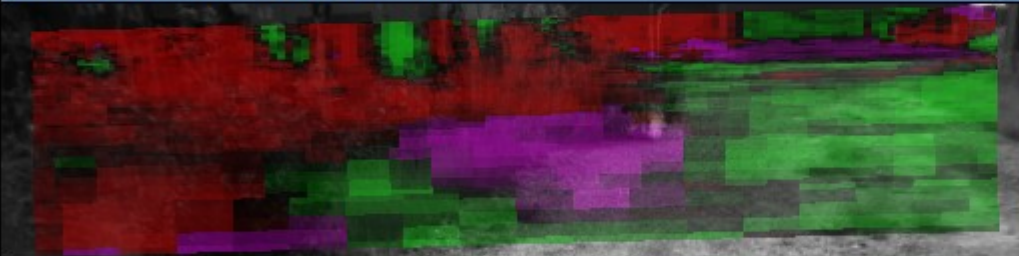


### Vehicle Map (Hyperbolic Polar map)

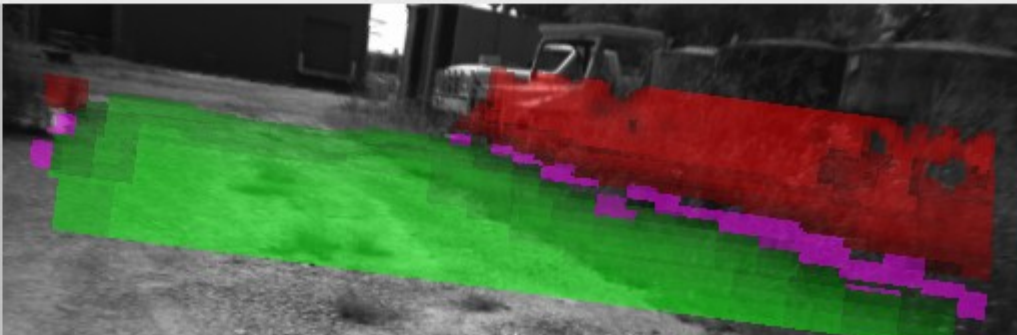
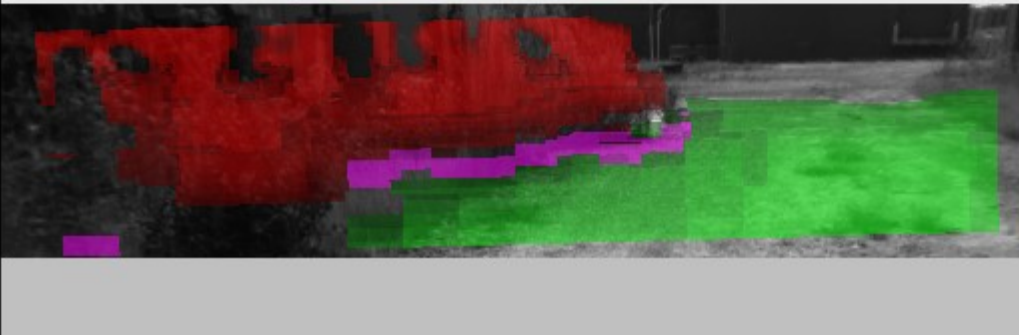
- Legend
  - Goal
  - Path Planning
  - ▬ Trajectories
  - ▬ Traversable
  - ▬ Uncertain
  - ▬ Quasi-Lethal
  - ▬ Lethal
  - ▬ Bumper/Stuck
  - ▬ Unseen
- 200m  
100m  
50m  
25m  
15m  
10m  
5m  
-5m  
-10m  
-15m  
-25m  
-50m  
-100m  
-200m



### FarOD Neural Network Labels



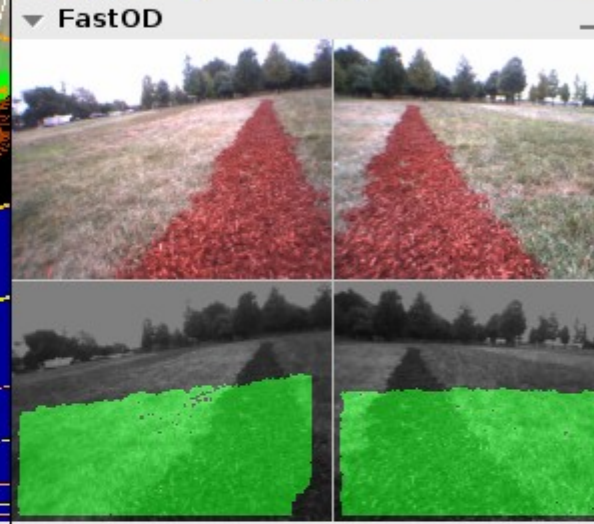
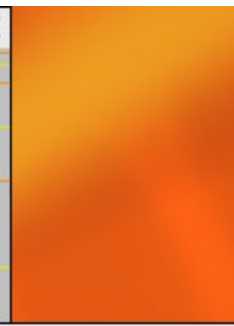
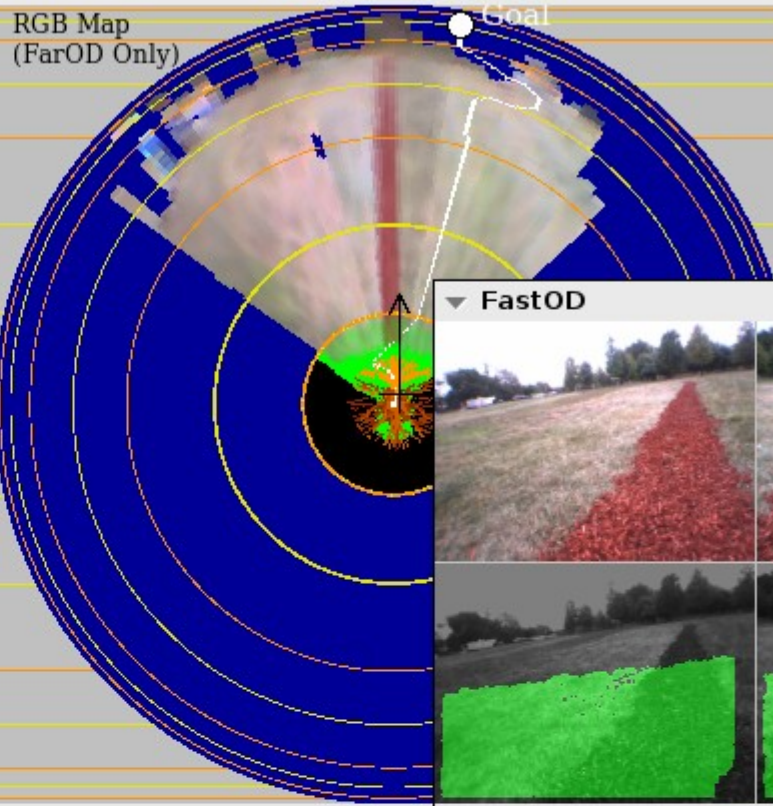
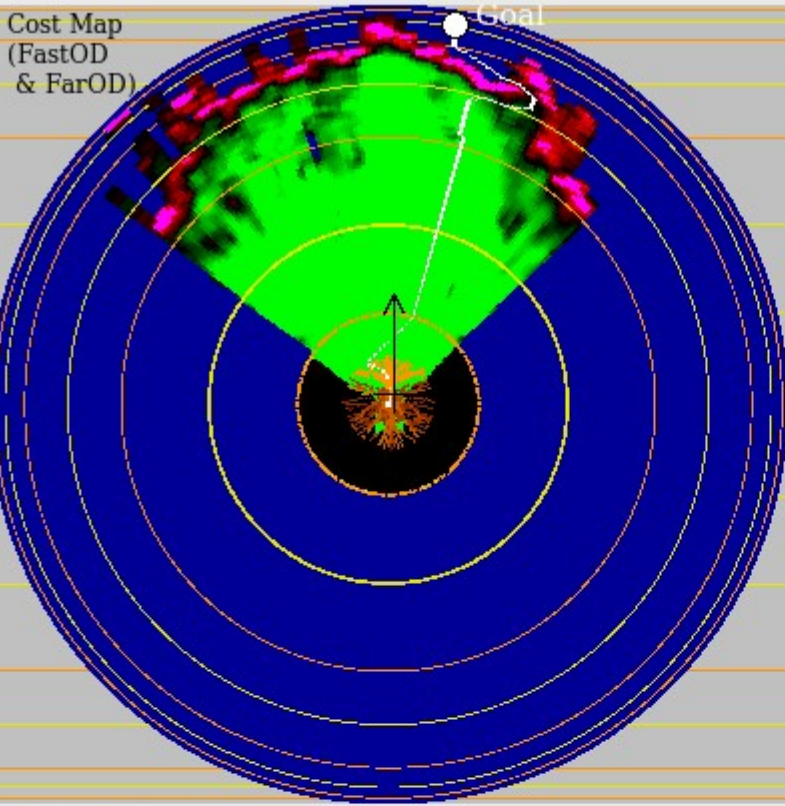
### FarOD Stereo: Input labels to Neural Network



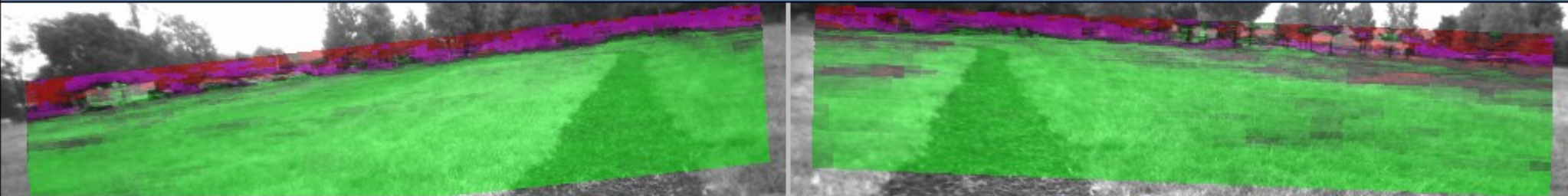


### Vehicle Map (Hyperbolic Polar map)

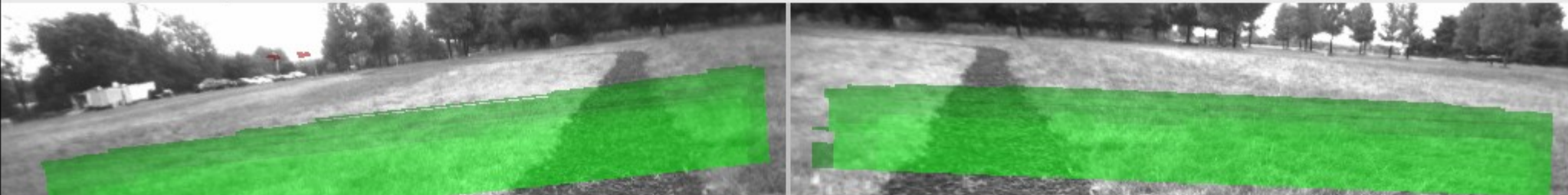
- Legend
- 200m
- 100m
- 50m
- Goal
- Path Planning
- Trajectories
- Traversable
- Uncertain
- Quasi-Lethal
- Lethal
- Bumper/Stuck
- Unseen
- 25m
- 15m
- 10m
- 5m
- 5m
- 10m
- 15m
- 25m
- 50m
- 100m
- 200m



### FarOD Neural Network Labels

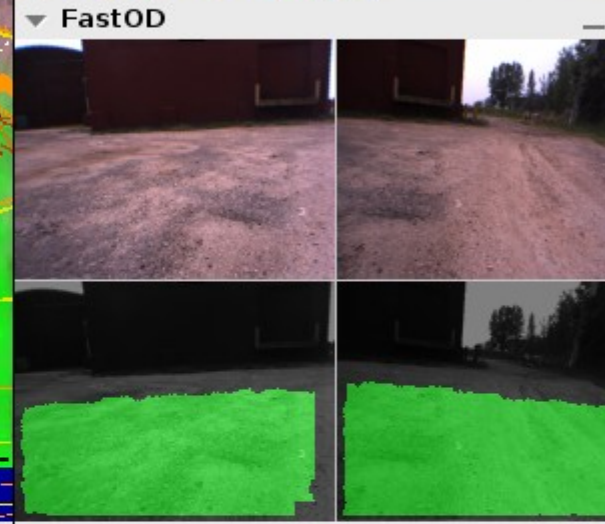
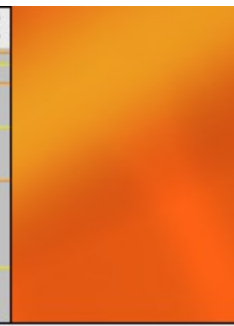
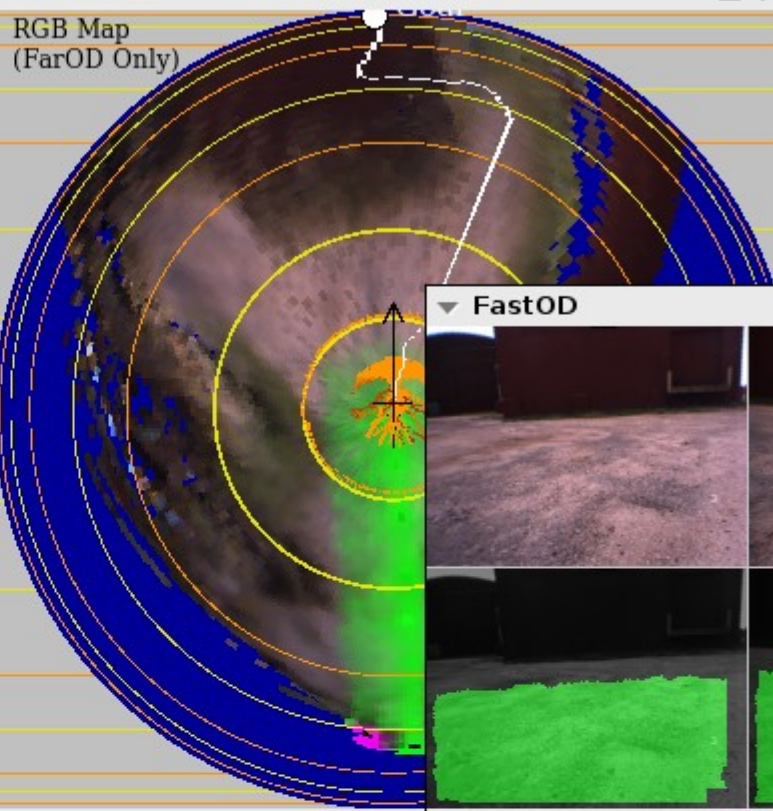
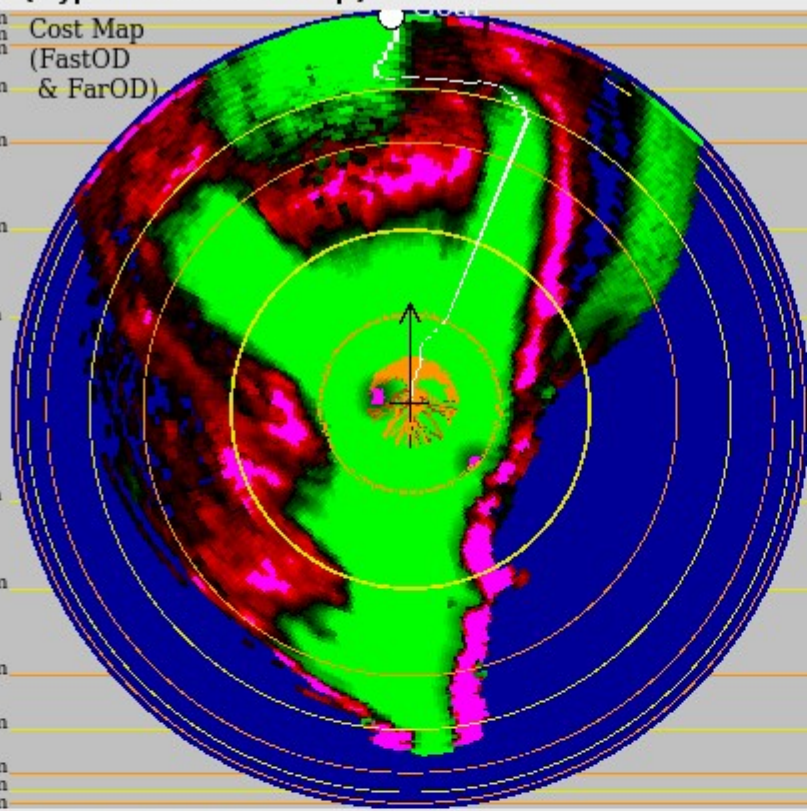


### FarOD Stereo: Input labels to Neural Network

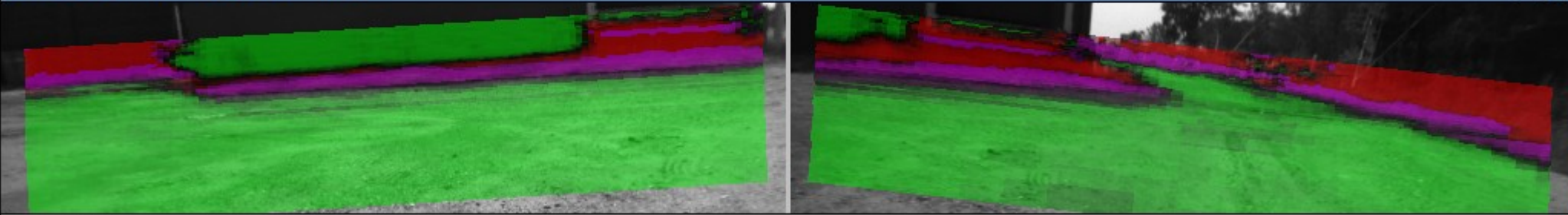


### Vehicle Map (Hyperbolic Polar map)

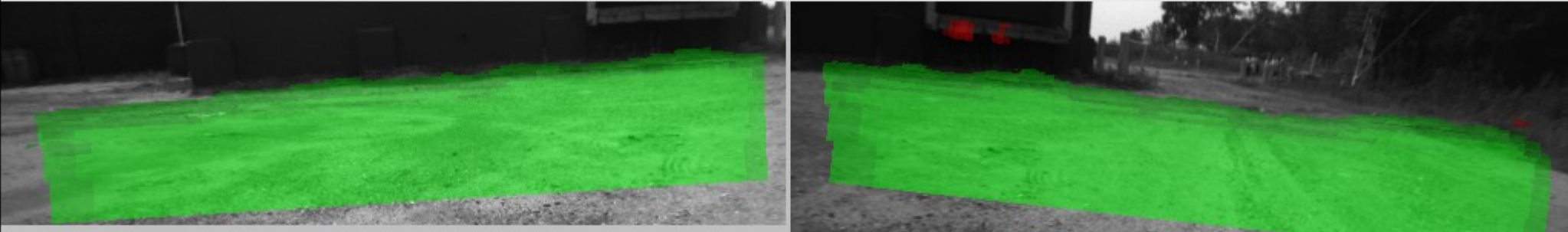
- Legend
- 200m
- 100m
- 50m
- Cost Map (FastOD & FarOD)
- Goal
- Path Planning
- Trajectories
- Traversable
- Uncertain
- Quasi-Lethal
- Lethal
- Bumper/Stuck
- Unseen
- 5m
- 10m
- 15m
- 25m
- 50m
- 100m
- 200m



### FarOD Neural Network Labels

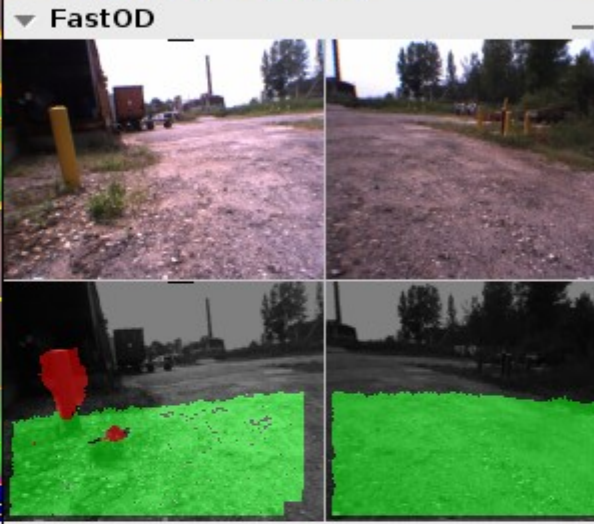
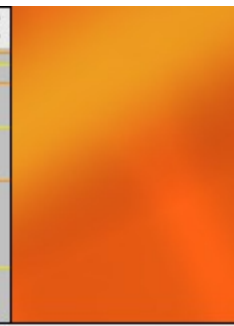
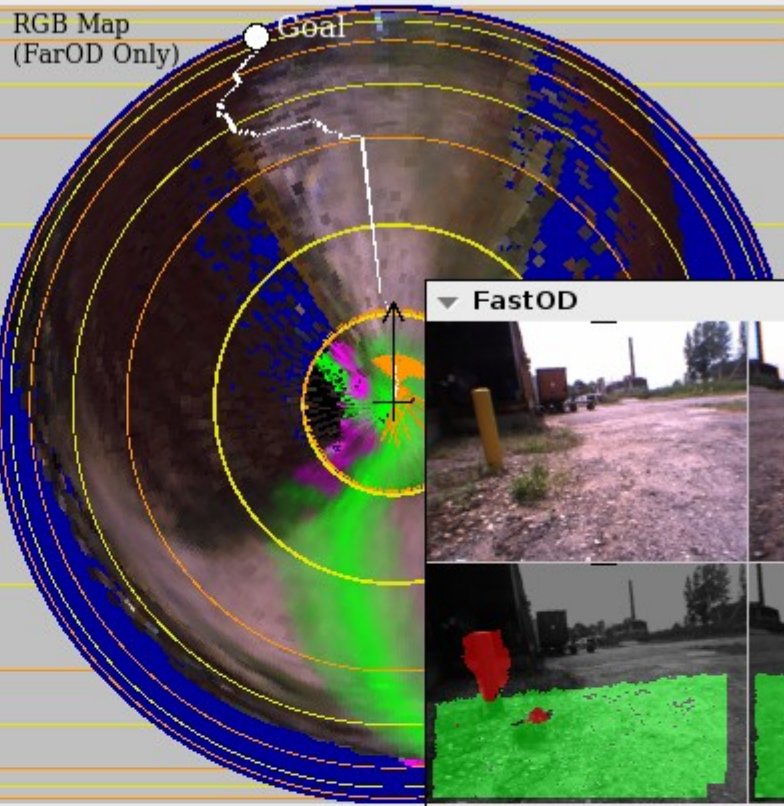
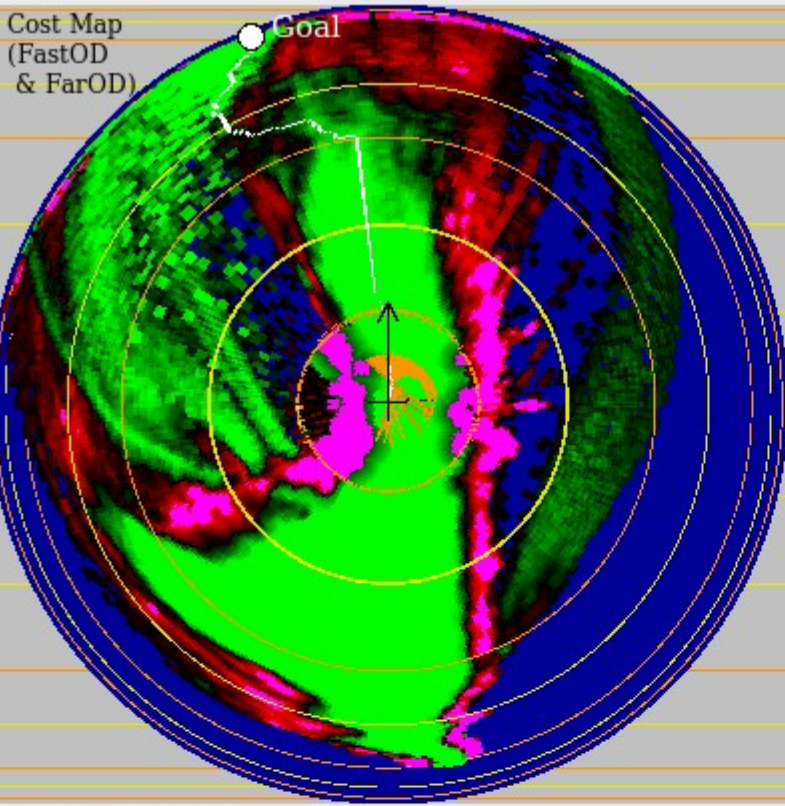


### FarOD Stereo: Input labels to Neural Network

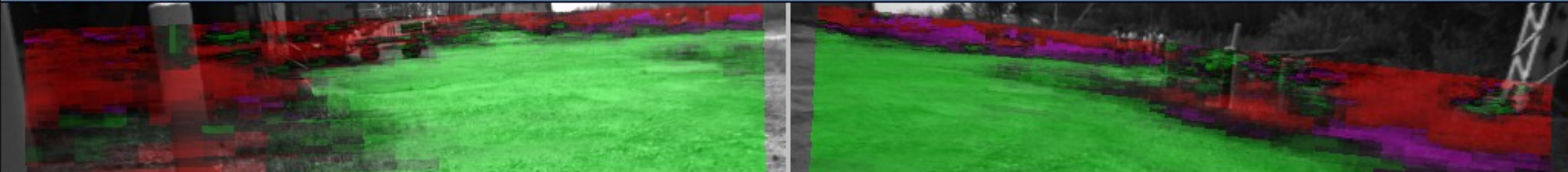


### Vehicle Map (Hyperbolic Polar map)

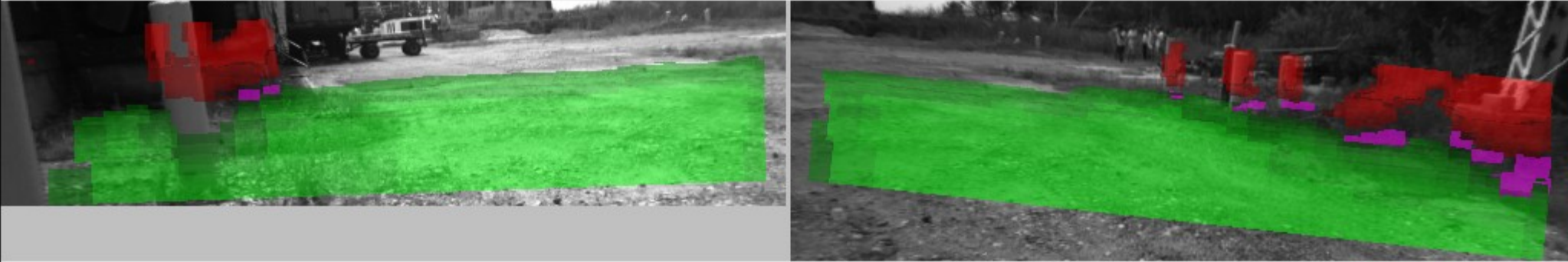
- Legend
- 200m
- 100m
- 50m
- Goal
- Path Planning
- Trajectories
- Traversable
- Uncertain
- Quasi-Lethal
- Lethal
- Bumper/Stuck
- Unseen
- 25m
- 15m
- 10m
- 5m
- 5m
- 10m
- 15m
- 25m
- 50m
- 100m
- 200m



### FarOD Neural Network Labels

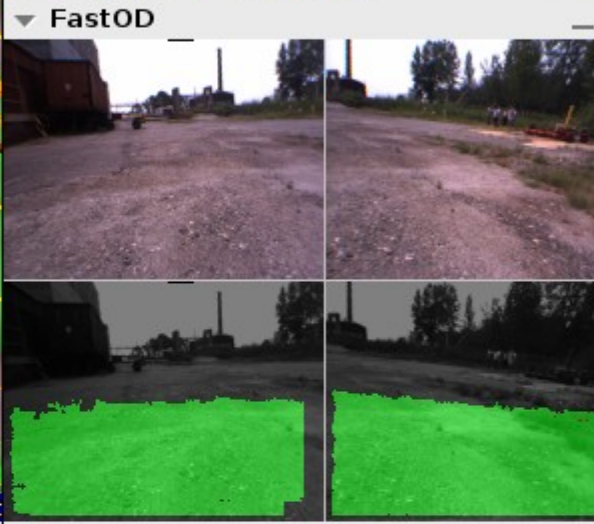
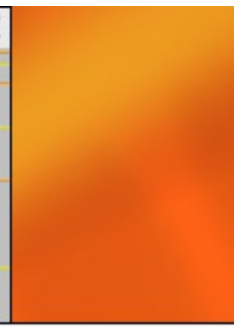
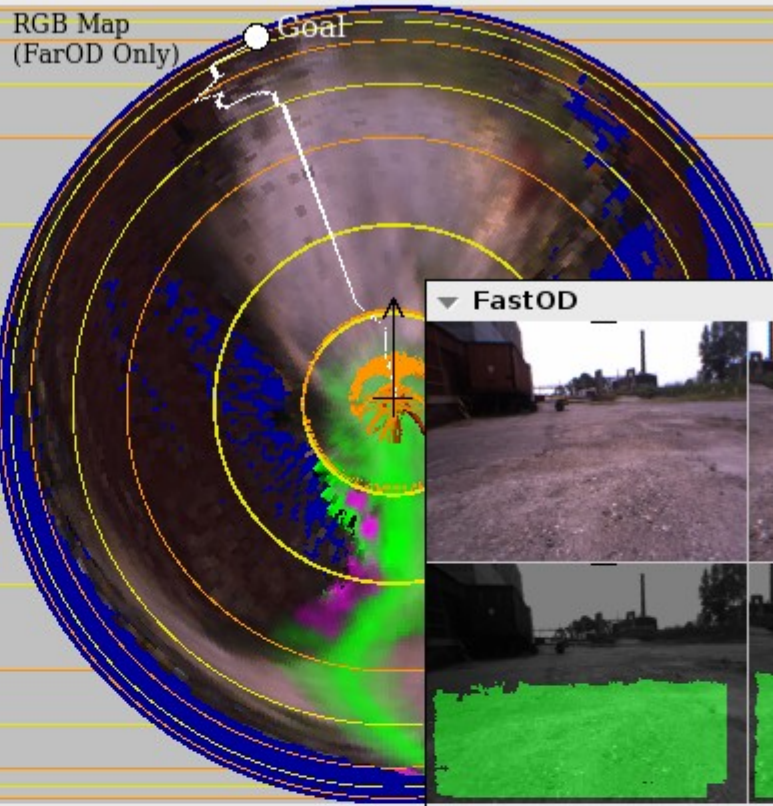
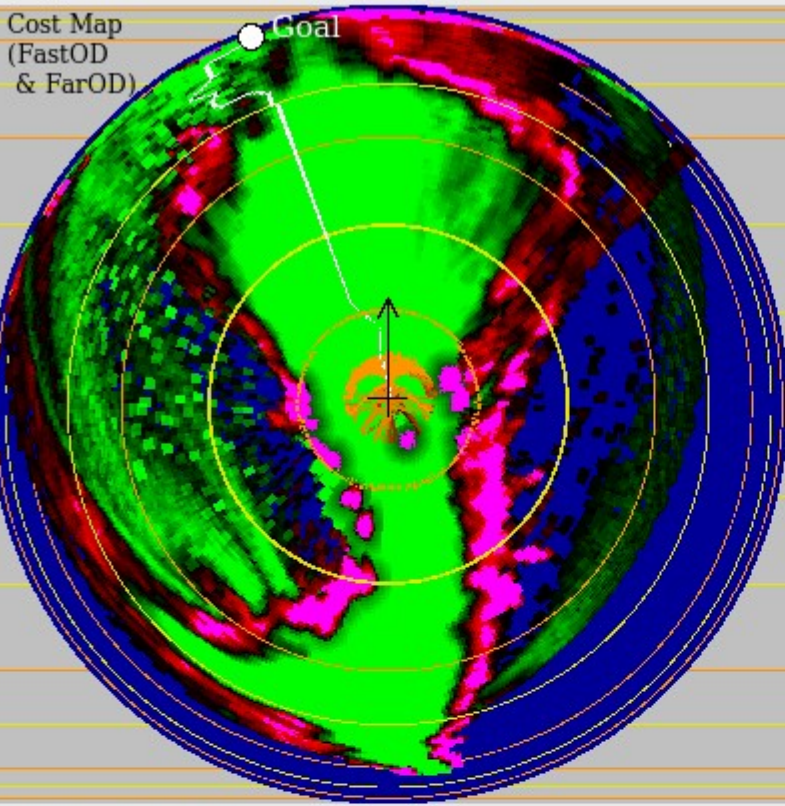


### FarOD Stereo: Input labels to Neural Network

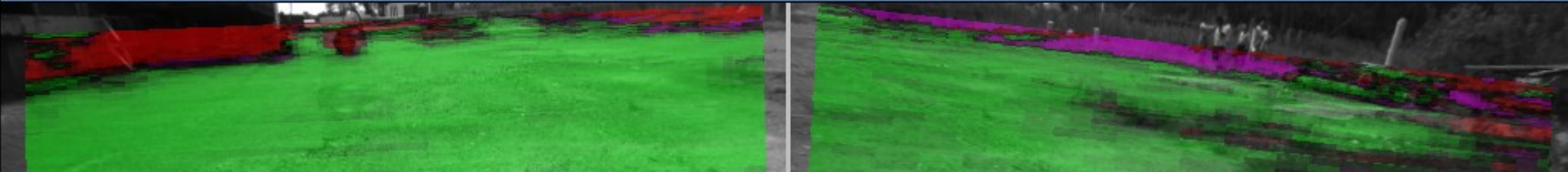


### Vehicle Map (Hyperbolic Polar map)

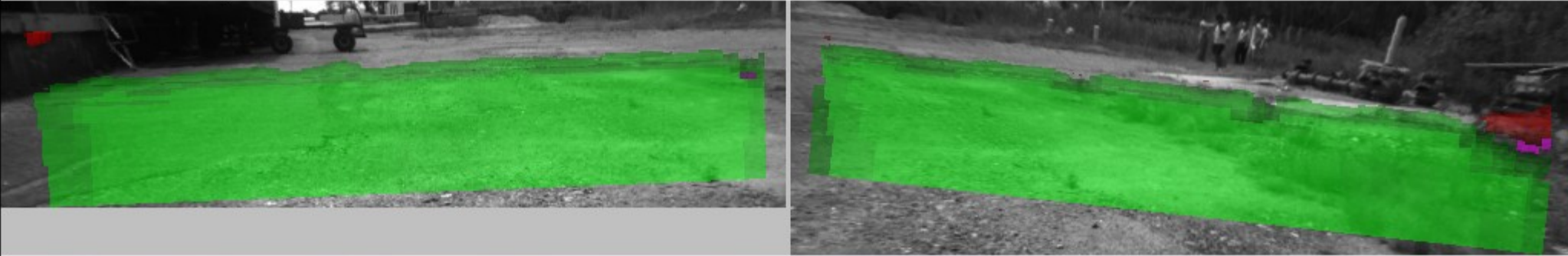
- Legend
- 200m
- 100m
- 50m
- Goal
- Path Planning
- Trajectories
- Traversable
- Uncertain
- Quasi-Lethal
- Lethal
- Bumper/Stuck
- Unseen
- 25m
- 15m
- 10m
- 5m
- 5m
- 10m
- 15m
- 25m
- 50m
- 100m
- 200m



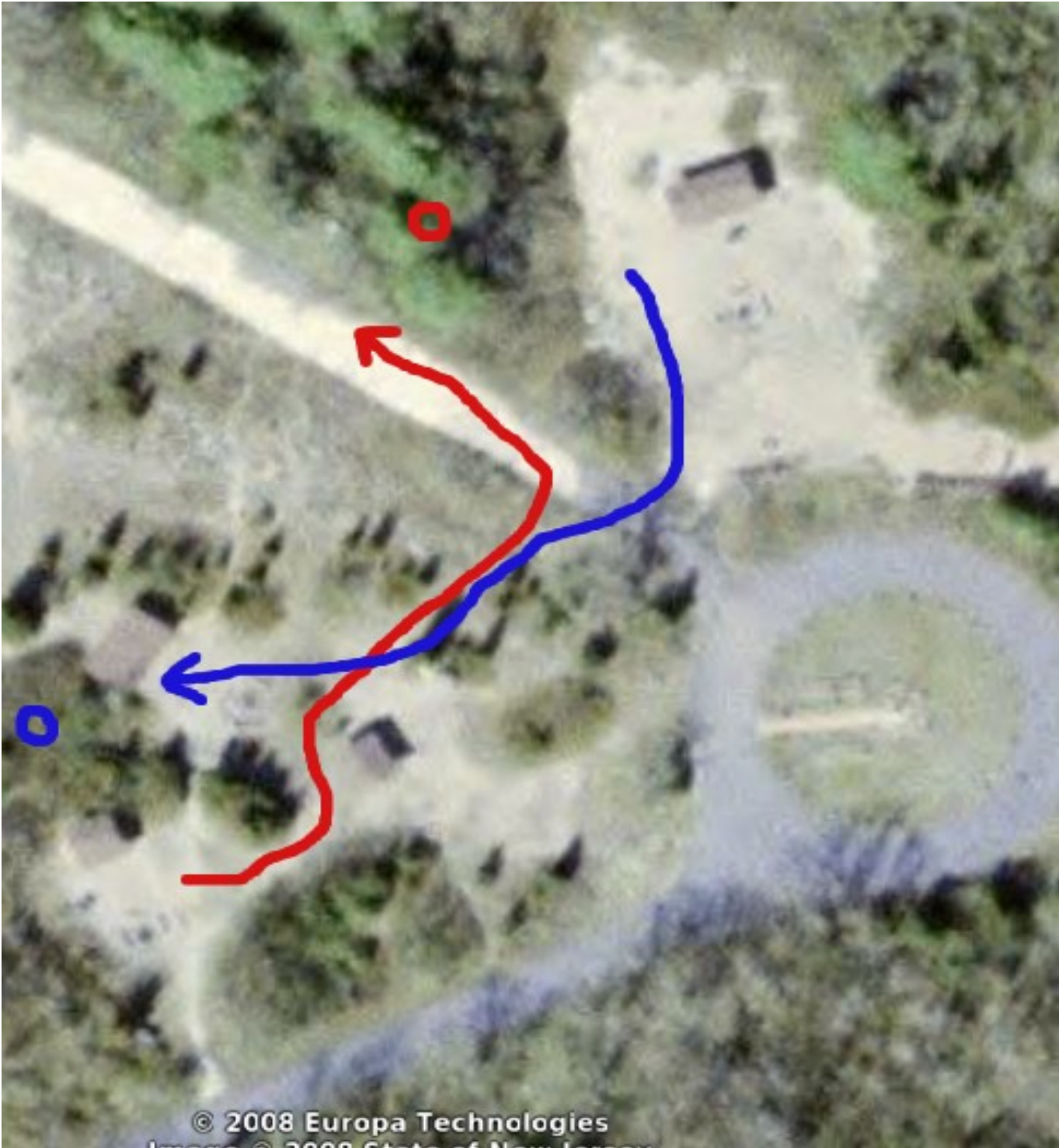
### FarOD Neural Network Labels



### FarOD Stereo: Input labels to Neural Network



# Videos



**The End**