Supervised and Unsupervised Learning with Energy-Based Models

Yann LeCun,

Computational and Biological Learning Lab

The Courant Institute of Mathematical Sciences

New York University

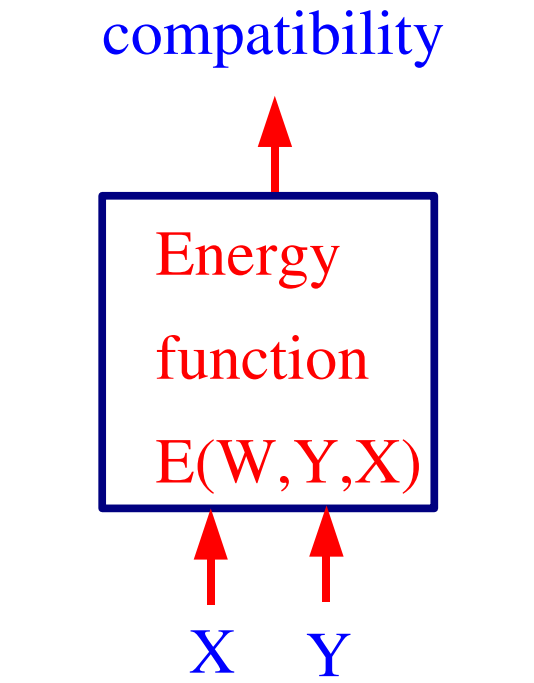
Collaborators: Marc'Aurelio Ranzato, Sumit Chopra,

Raia Hadsell, Fu Jie Huang,

http://yann.lecun.com

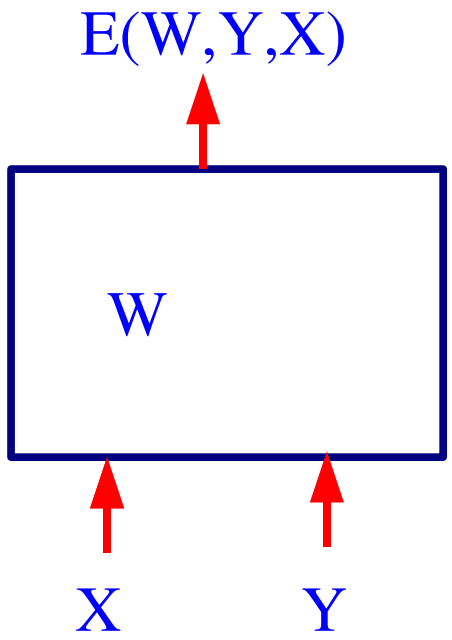
http://www.cs.nyu.edu/~yann

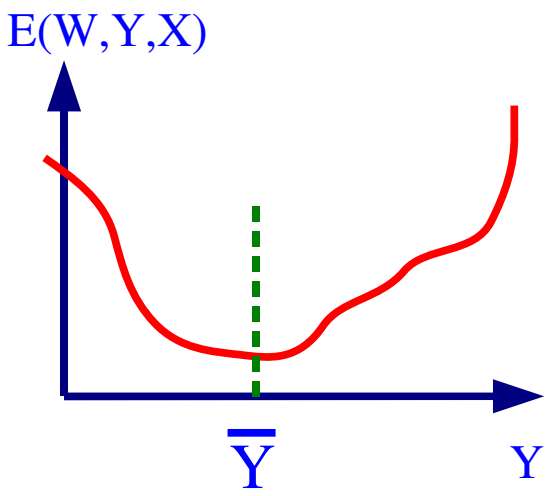
Energy-Based Models



- **A model measures the compatibility between a set of observed variables X, and a set of variables to be predicted Y.**
- **The compatibility is measured by an energy function (or contrast function) E(W,Y,X).**
  - Low energy = Y is compatible with X
  - High energy = Y is incompatible with X
- **W is a parameter vector**
  - W is to be learned from data

Energy-Based Models: Inference





- **Inference (decision making): for a given X, pick the value of Y within a set {Y} that minimizes the energy E(W,Y,X):**

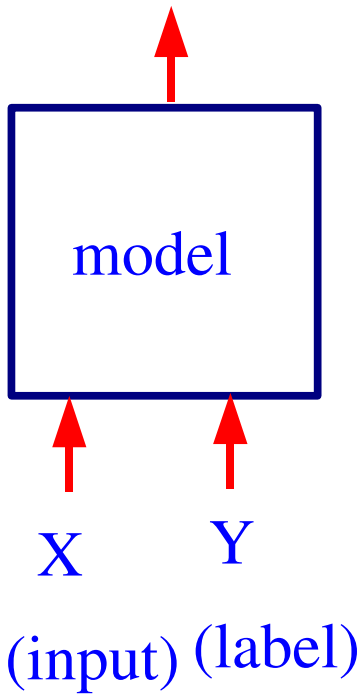
$$\check{Y} = \text{argmin}_{y \in \{Y\}} E(W, y, X)$$

- **Inference (probabilistic): Interpret the energy a negative log-probability:**

$$P(Y|X) = \frac{\exp\left(-\beta E(W, Y, X)\right)}{\sum_{y \in \{Y\}} \exp\left(-\beta E(W, y, X)\right)}$$

- **Three questions:**
  - **Architecture**: What do we put in the box?
  - **Inference algorithm:** How do we find the best Y?
  - **Learning:** how do we estimate W from data?

A Model is Designed and trained to Answer Questions



- **Example:** X is an image from a camera; Y is a discrete variable e.g. Y in {animal, human, plane, truck, car}.
- 1. **Best Guess for Y:** which category best describes X?
- 2. **Ranking on Y:** Is X more a car than an airplane?
- 3. **Distribution on Y:** give an estimate of P(animal | X)

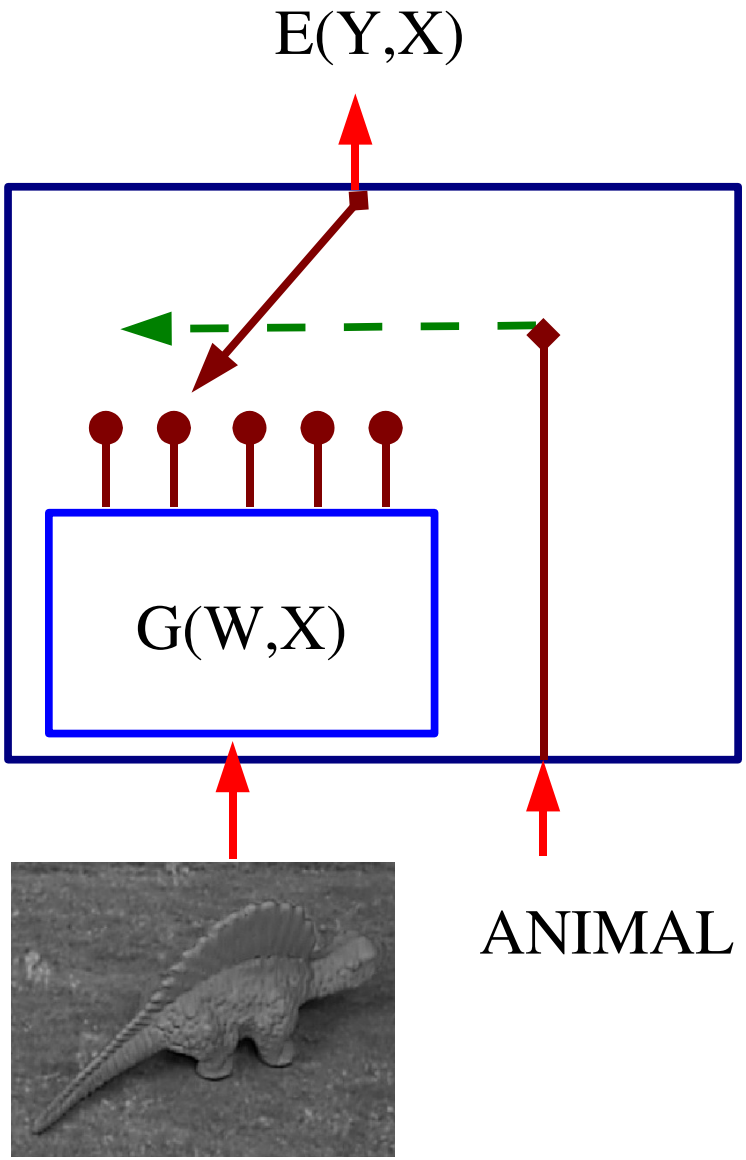
For each type of question:

- a different inference algorithm is required

- a different learning strategy is required

Never try to answer a more complex question than necessary

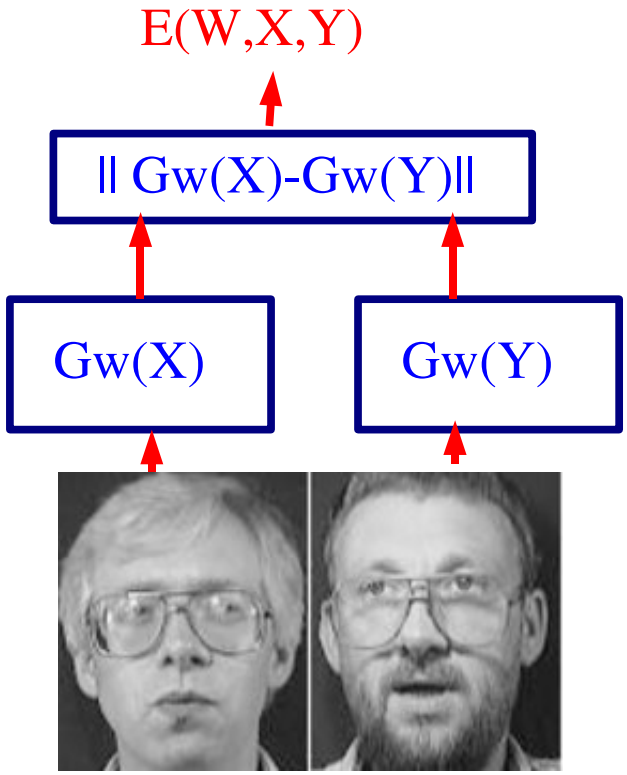
Examples of EBMs: traditional classification



- **X is an image**
- **Y is a discrete variable in the set:**

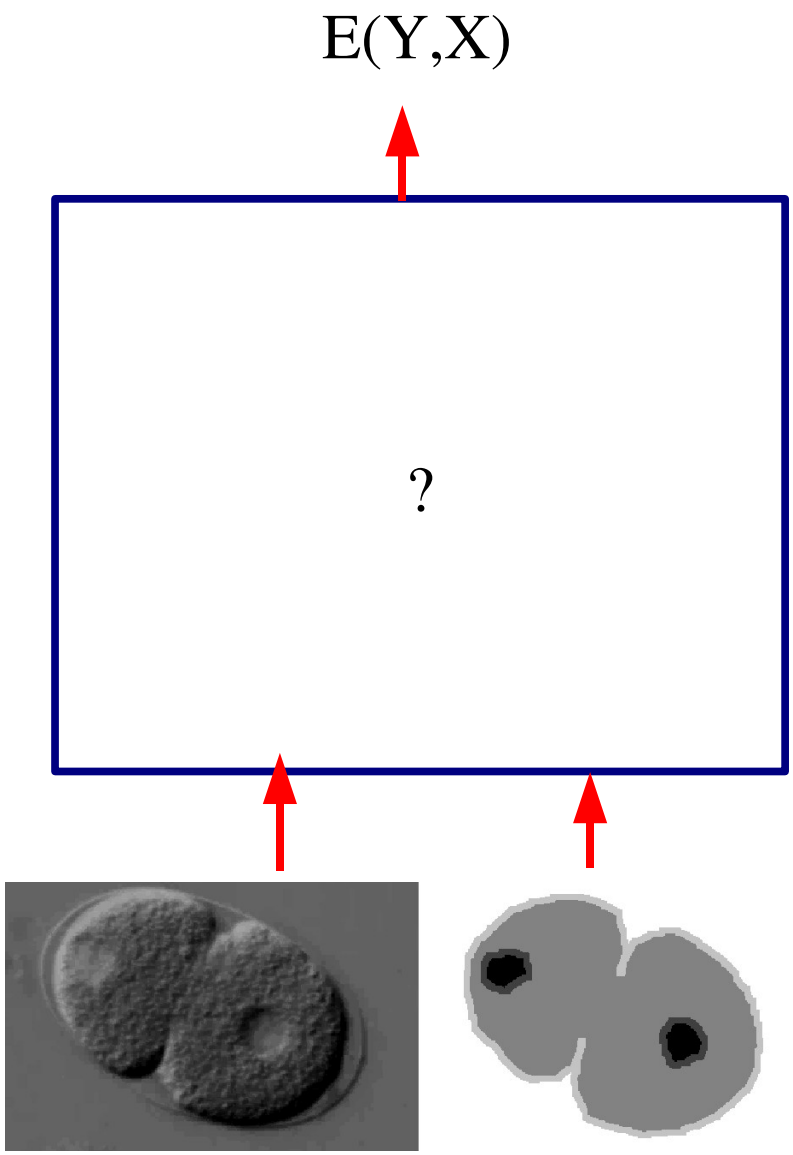
  {human, animal, airplane, car, truck, none of the above}
- **Example of architecture:**
  - A discriminative function G(X) gives a vector of scores (one component for each possible value of Y)
  - A switch picks the lowest value
- **Inference:**
  - exhaustive search over Y

Examples of EBMs: Face Recognition



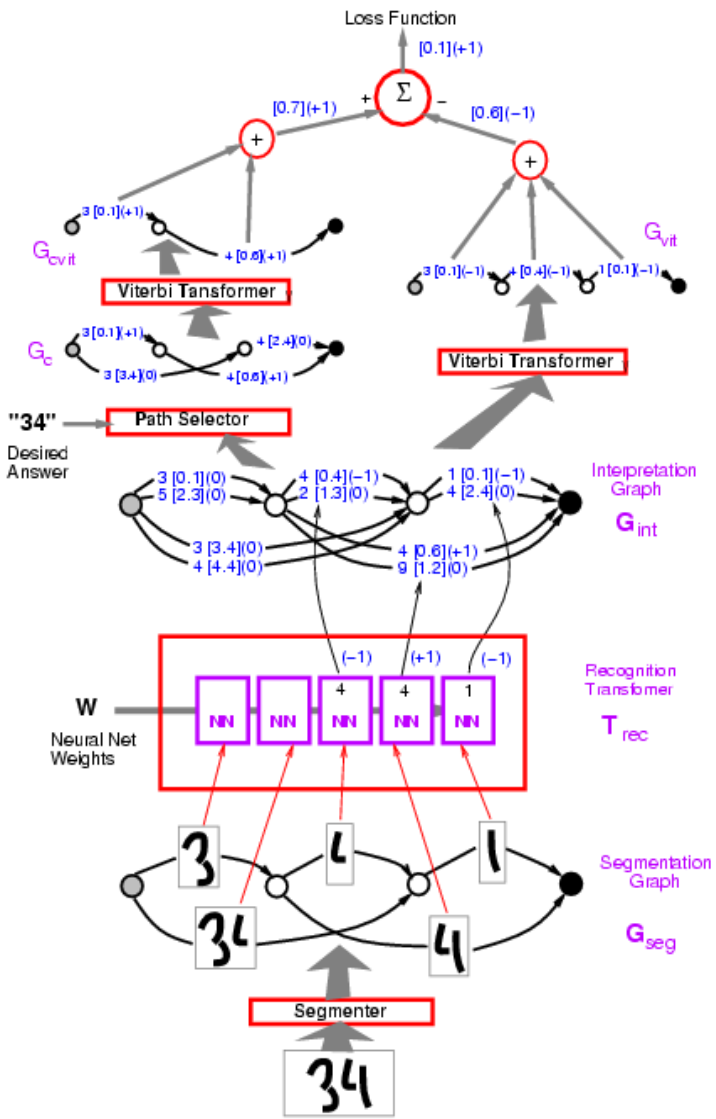
- **X and Y are images**
- **Y is a discrete variable with many possible values**
  - All the people in our gallery
- **Example of architecture:**
  - A function G(X) maps input images into a low-dimensional space in which the Euclidean distance measures dissemblance.
- **Inference:**
  - exhaustive search over Y?
  - Probabilistic inference is next to impossible

Examples of EBMs: Image Segmentation / Object Detection



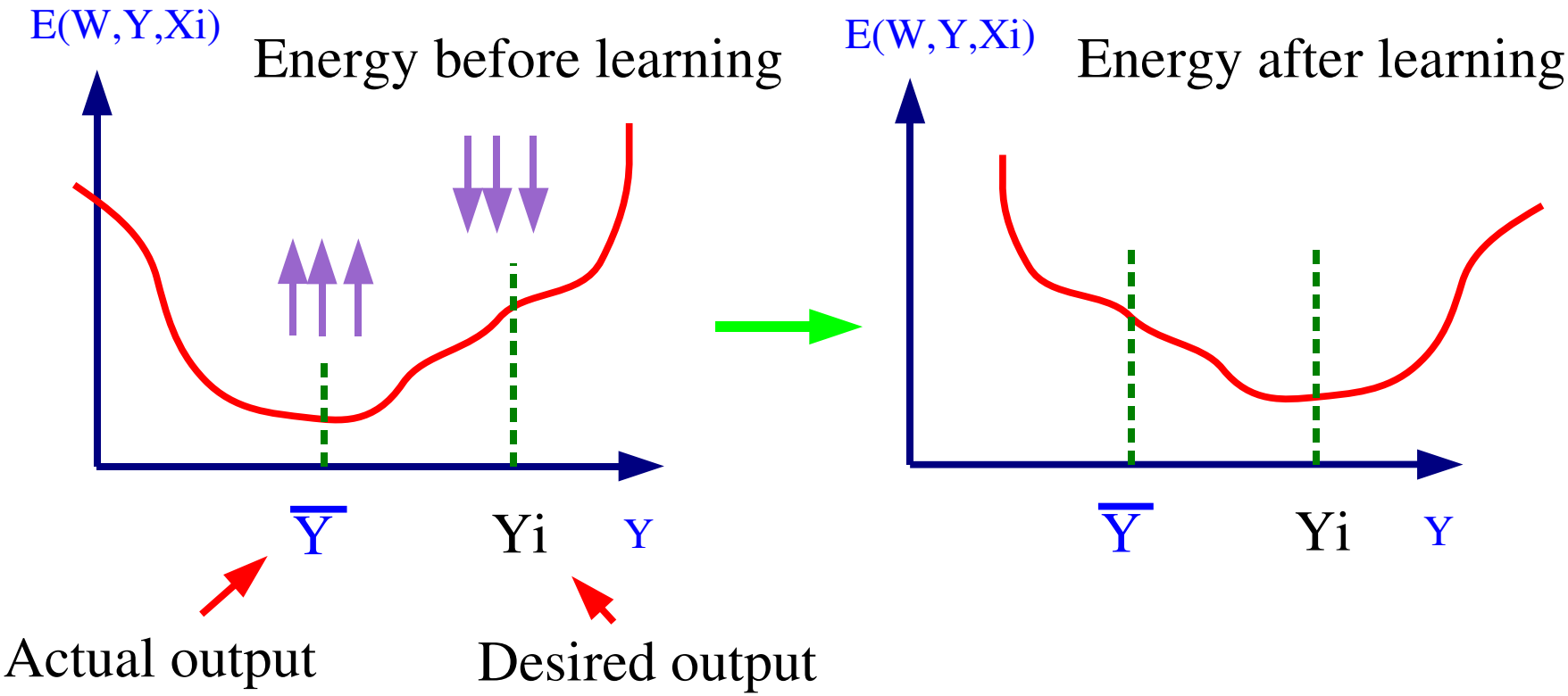
- **X is an image**
- **Y is a segmentation image**
  - Y is a discrete variable with an intractable number of possible values.
  - Y must satisfy certain topolgical constraints (structured output)
- **Architecture**
  - Some sort of factor graph (graphical model)
- **Inference:**
  - Relaxation
  - Belief propagation
  - Probabilistic inference is generally impossible (we can't normalize)

Examples of EBMs: Sequence Labeling



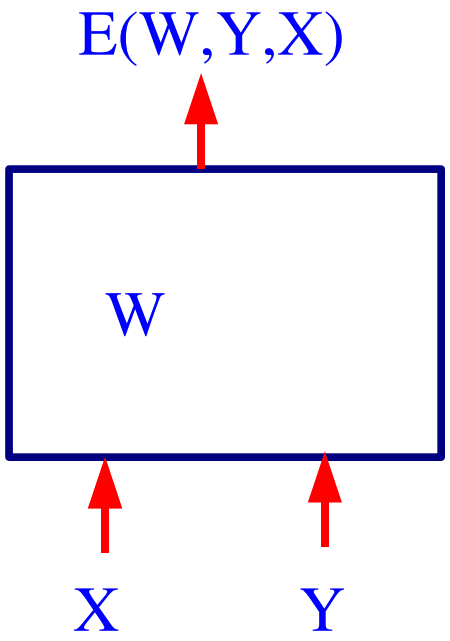
- **X is a sequence**
  - A handwritten word image or pen trajectory, a sequence of acoustic vectors (speech), a text sentence, a DNA sequence.....
- **Y is a sequence of labels**
  - Text transcription, parts of speech tags, gene annotations....
  - Y must satisfy certain grammatical constraints (structured output)
- **Architecture**
  - Some sort of factor graph.
- **Inference:**
  - Viterbi, Forward algorithm
  - Probabilistic inference is quasi impossible (we can't easily normalize)

Training an EBM



- **Given a set of training samples (Xi, Yi), find a W that makes E(W,Yi,Xi) smaller than E(W,y,Xi) for all y different from Yi.**
- **Question: how do we design a loss function L, such that minimizing L with respect to W will make the energy surface have the correct shape?**

Architecture + Inference Algo + Loss Function = Model



- **1. Design an architecture:** a particular form for E(W,Y,X).
- **2. Pick an inference algorithm for Y:** MAP or conditional distribution, belief prop, min cut, variational methods, gradient descent, MCMC, HMC.....
- **3. Pick a loss function:** in such a way that minimizing it with respect to W over a training set will make the inference algorithm find the correct Y for a given X.
- **4. Pick an optimization method.**

- **PROBLEM: What loss functions will make the machine approach the desired behavior?**

An Example of Good Loss Function:

- **Push down on the energy of the correct answer, and push up on the energy of the incorrect answer that has the lowest energy (most offending incorrect answer)**

$$\bar{Y}^i = \text{argmin}_{y \neq Y^i} E(W, y, X^i)$$

$$L(W, Y^i, X^i) = F1[E(W, Y^i, X^i)] - F2[E(W, \bar{Y}^i, X^i)]$$

- **F1 and F2 must be increasing functions of their argument.**

Examples of Loss Functions

BAD

- Energy Loss: $L_{\text{energy}}(W, Y^i, X^i) = \check{E}(W, Y^i, X^i)$.
  Only works if the architecture is such that decreasing $\check{E}(W, Y^i, X^i)$ will automatically increase $\check{E}(W, Y, X^i)$ for $y \neq Y^i$.

BAD

- Generalized Perceptron Loss [LeCun 1998][Collins 2002]:
  $L_{\text{ptron}}(W, Y^i, X^i) = \check{E}(W, Y^i, X^i) - \min_{Y \in \{Y\}} \check{E}(W, Y, X^i)$
  Does not work because the margin is zero. This reduces to the traditional linear perceptron loss when $\check{E}(W, Y, X) = -YW.X$.

GOOD

- Generalized Margin Loss: [LeCun et al. 2005]
  $L_{\text{gmargin}}(W, Y^i, X^i) = Q[\check{E}(W, Y^i, X^i), \check{E}(W, \bar{Y}, X^i)]$
  Where Q is an increasing function of $\check{E}(W, Y^i, X^i)$ and a decreasing function of $\check{E}(W, \bar{Y}, X^i)$.

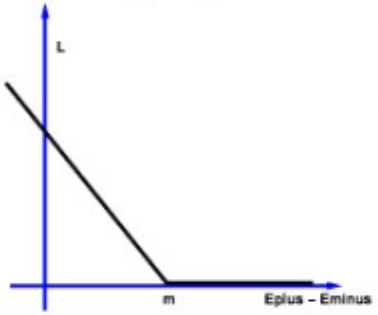
OK, but hard

- Negative Log Likelihood Loss: [Bengio 1992][LeCun 1998][Lafferty 2001]
  $L_{\text{nll}}(W, Y^i, X^i) = \check{E}(W, Y^i, X^i) - F_\beta(W, X^i)$
  with: $F_\beta(W, X^i) = -\frac{1}{\beta} \log \left(\int_{Y \in \{Y\}} \exp[-\beta \check{E}(W, Y, X^i)] \right)$

Standard loss for probabilistic model (e.g. CRF)

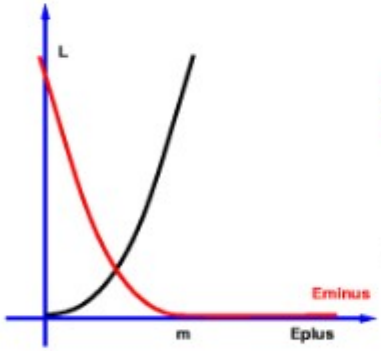
Special Cases of the Generalized Margin Loss

$$L_{\text{gmargin}}(W, Y^i, X^i) = Q[\check{E}(W, Y^i, X^i), \check{E}(W, \bar{Y}, X^i)]$$



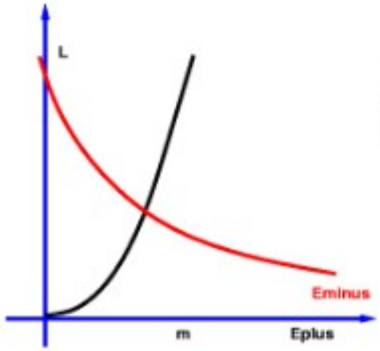
Hinge Loss [Taskar, Guestrin, Koller 2003],[Altun, Johnson, Hofmann, 2003]:

$$L_{\text{hinge}}(W, Y^i, X^i) = \max(0, m + \check{E}(W, Y^i, X^i) - \check{E}(W, \bar{Y}, X^i))$$



Square-Square Loss:

$$L_{\text{sqsq}}(W, Y^i, X^i) = \check{E}(W, Y^i, X^i)^2 + (\min(0, m - \check{E}(W, \bar{Y}, X^i)))^2$$



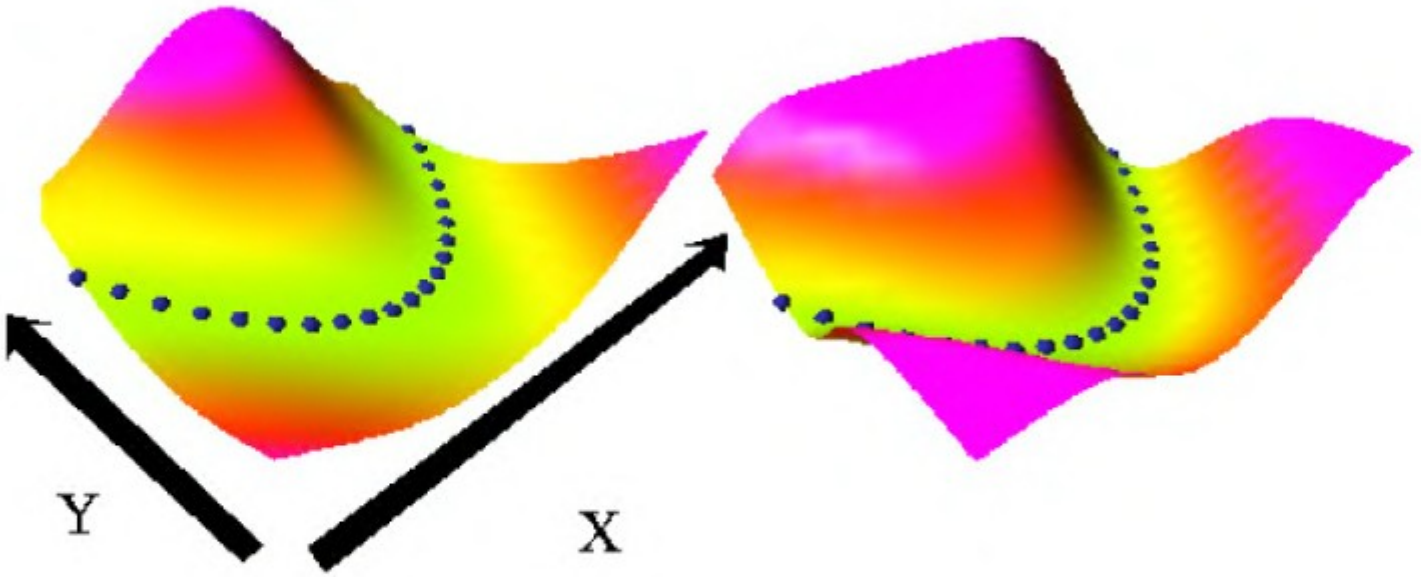
Square-Exp Loss: [Osadchy, Miller, LeCun, NIPS 2004]

$$L_{\text{sqexp}}(W, Y^i, X^i) = \check{E}(W, Y^i, X^i)^2 + K \exp(-\beta \check{E}(W, \bar{Y}, X^i))$$

What's so bad about probabilistic models?

- Why bother with a normalization since we don't use it for decision making?
- Why insist that P(Y|X) have a specific shape, when we only care about the position of its minimum?
- When Y is high-dimensional (or simply conbinatorial), normalizing becomes intractable (e.g. Language modeling, image restoration, large DoF robot control...).
- A tiny number of models are pre-normalized (Gaussian, exponential family)
- A very small number are easily normalizable
- A large number have intractable normalization
- A huuuge number can't be normalized at all (examples will be shown).
- Normalization forces us to take into account areas of the space that we don't actually care about because our inference algorithm never takes us there.
- **If we only care about making the right decisions, maximizing the likelihood solves a much more complex problem than we have to.**

EBM Energy Surfaces

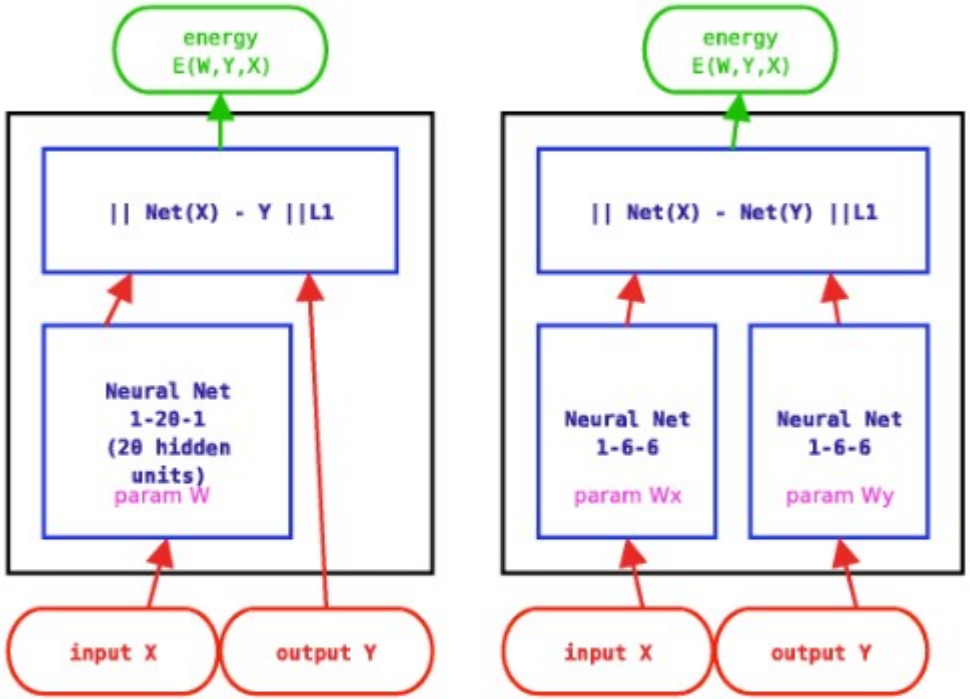


Examples: An EBM that computes $Y = X^2$.

On the left: $E(Y, X)$ is quadratic in Y. It corresponds to a Gaussian model of $P(Y|X)$.

On the right: $E(Y, X)$ is saturated. Although it gives the same answers as the EBM on the left, it has no probabilistic equivalent because the partition function $\int_y \exp(-E(Y, X))$ does not converge.

EBM Demos

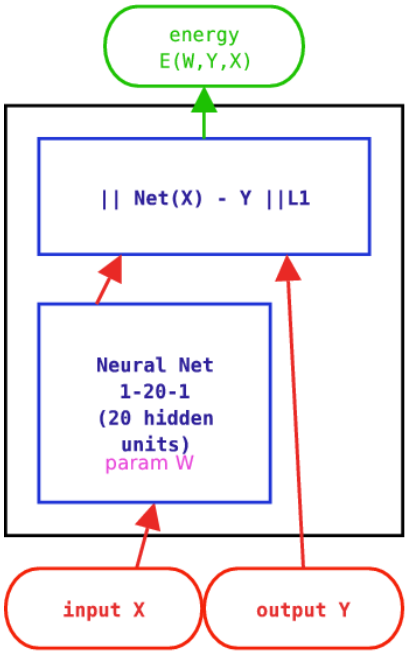


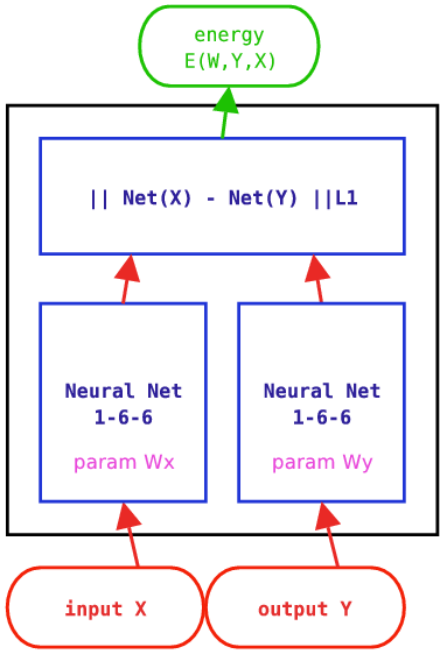
- Demo 1: $Y = X^2$, Architecture A, Square Energy Loss. It works because $E(Y, X)$ is a fixed quadratic function of Y.
- Demo 2: $Y = X^2$, Architecture B, Square Energy. It collapses.
- Demo 3: $Y = X^2$, Architecture B, Square-Square Margin Loss
- Demo 4: $Y = X^2$, Architecture B, Negative Log Likelihood Loss. Few iterations, but each iteration is expensive

Initially, the forbidden sphere around Y^i is 0.2, then 0.1.
Demo 5: eye pattern, Architecture B, Negative Log Likelihood Loss.

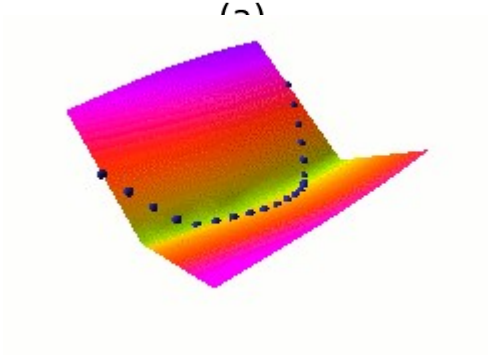
EBM Demos: energy loss

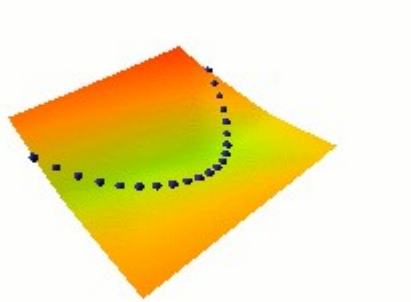
Loss: "Energy Loss": L(W,Y,X) = E(W,Y,X)





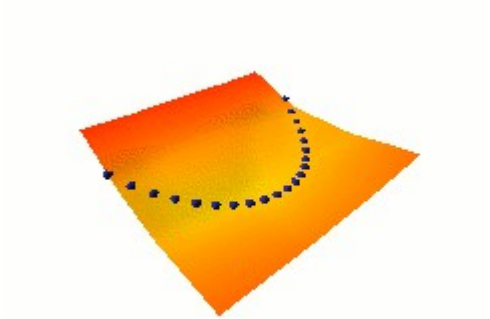
COLLAPSE!!!

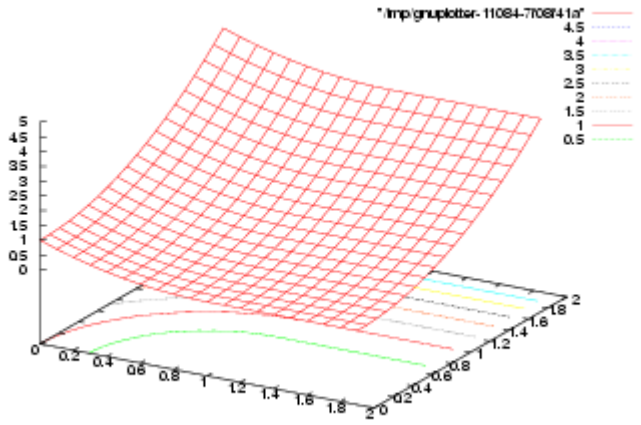


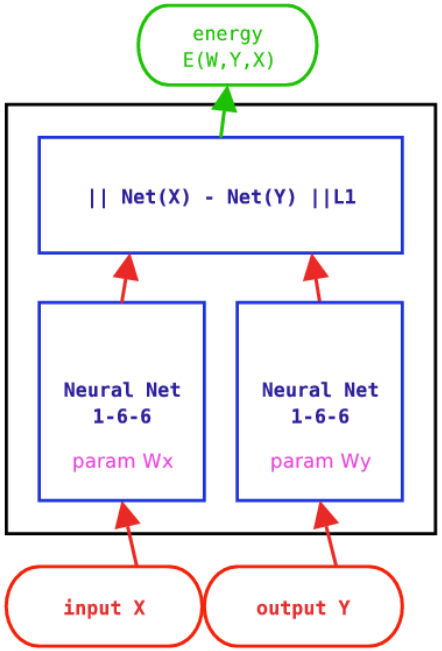


EBM Demos: good losses

Square-square loss

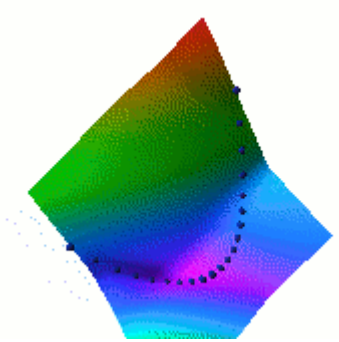


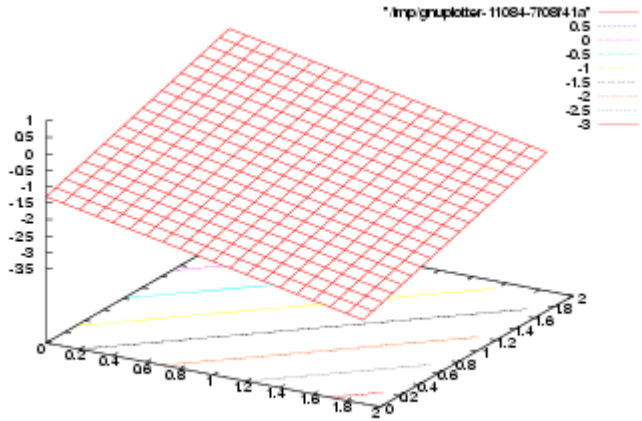




(b)

Neg-Log-Likelihood loss





$$L(W,X,Y) = E(W,Y,X)^2 - \max\left(0, m - E(W,\bar{Y},X)^2\right)$$

$$\bar{Y} = \text{argmin}_{y\in\{Y\},y\neq Y} E(W,y,X)$$

$$L(W,X,Y) = E(W,Y,X) + \frac{1}{\beta}\log\left[\sum_{y\in\{Y\}}\exp\left(-\beta E(W,y,X)\right)\right]$$

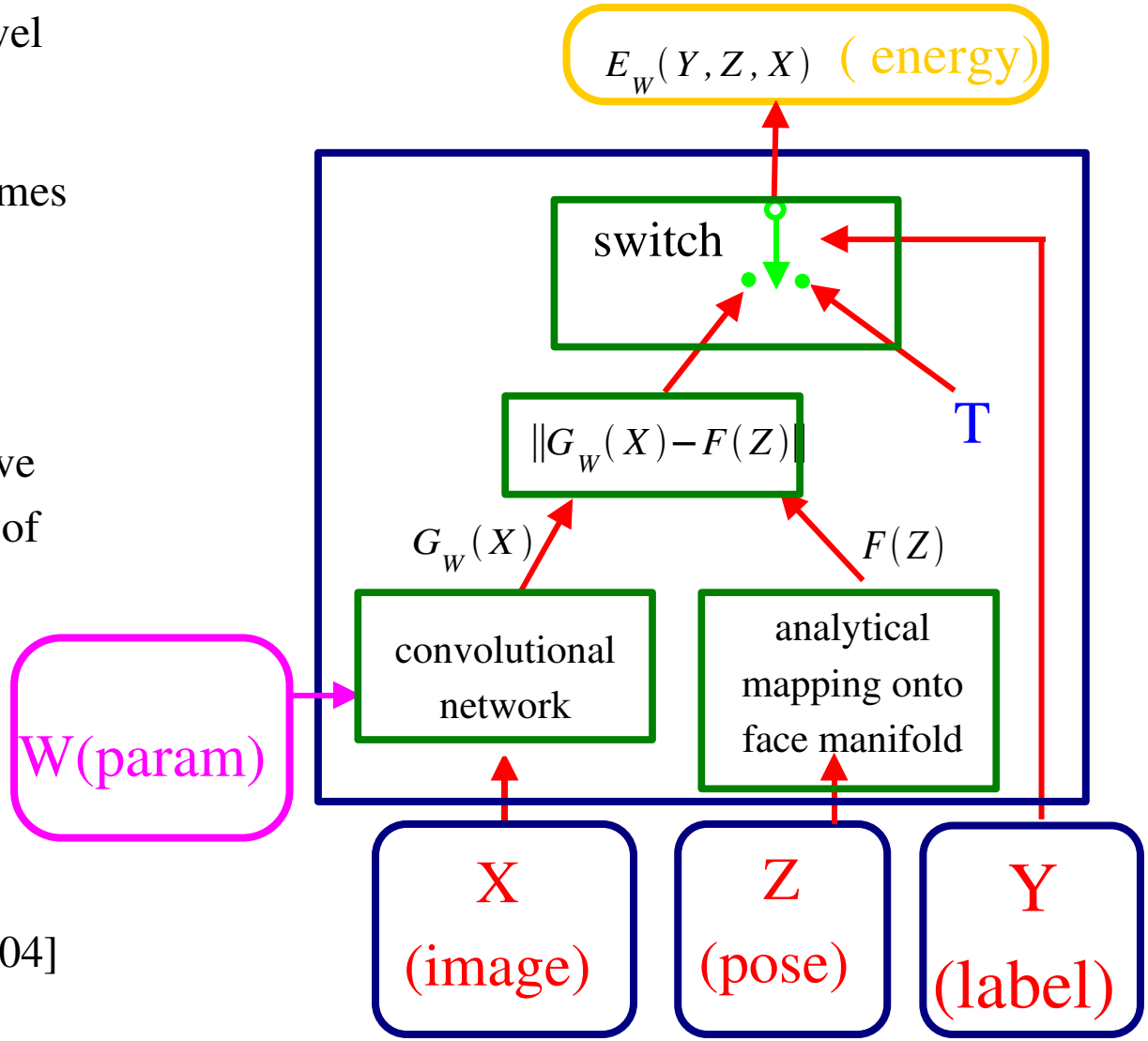
EBM

- Unlike traditional classifiers, EBMs can represent multiple alternative outputs
- The normalization in probabilistic models is often an unnecessary aggravation, particularly if the ultimate goal of the system is to make decisions.
- EBMs with appropriate loss function avoid the necessity to compute the partition function and its derivatives (which may be intractable)
- EBMs give us complete freedom in the choice of the architecture that models the joint "incompatibility" (energy) between the variables.
- We can use architectures that are not normally allowed in the probabilistic framework (like neural nets).
- **The inference algorithm that finds the most offending (lowest energy) incorrect answer does not need to be exact:** our model may give low energy to far-away regions of the landscape. But if our inference algorithm never finds those regions, **they do not affect us.** But they do affect normalized probabilistic models

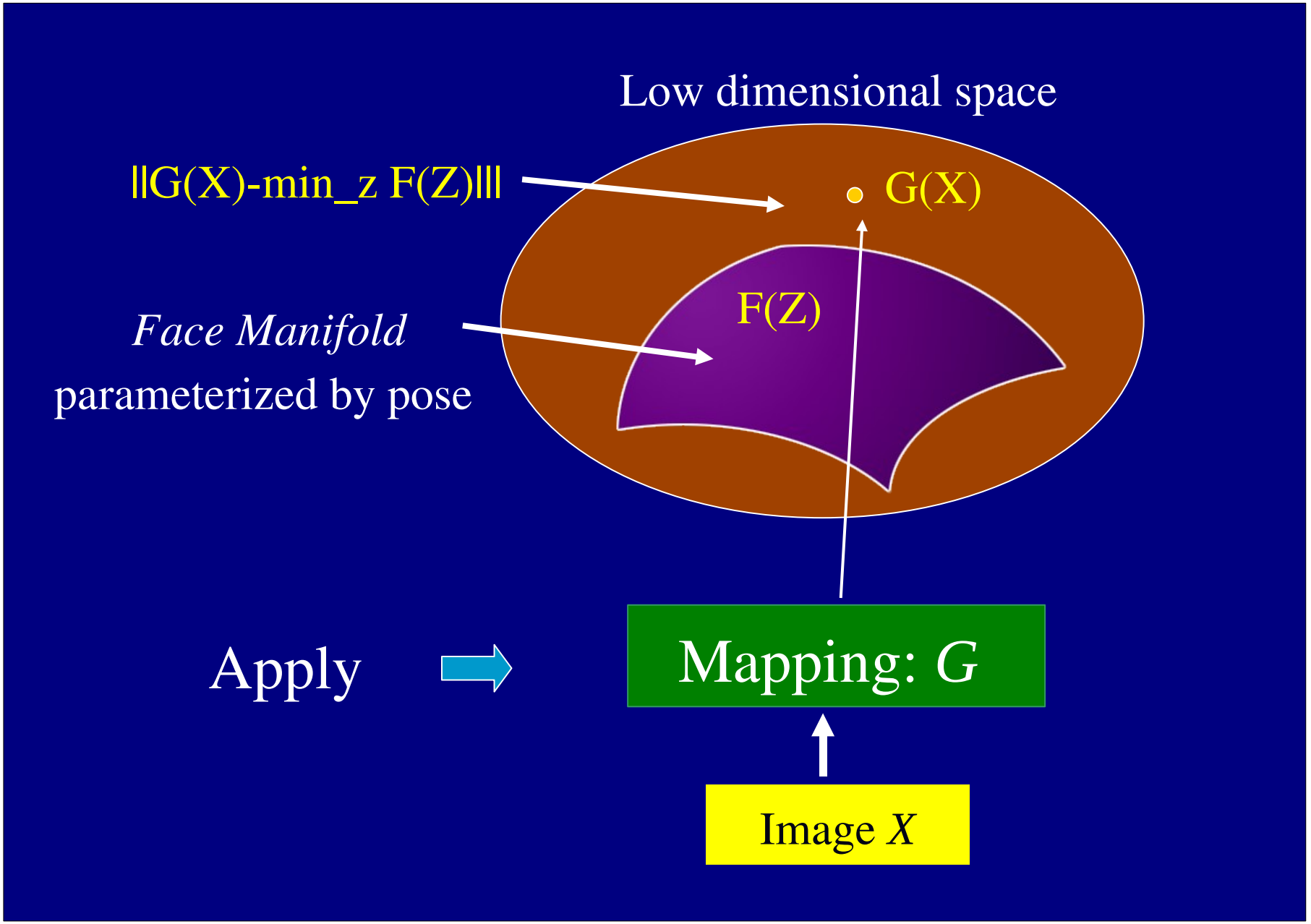
Face Detection and Pose Estimation with a Convolutional EBM

- **Training:** 52,850, 32x32 grey-level images of faces, 52,850 non-faces.
- Each training image was used 5 times with random variation in scale, in-plane rotation, brightness and contrast.
- **2nd phase:** half of the initial negative set was replaced by false positives of the initial version of the detector .

[Osadchy, Miller, LeCun, NIPS 2004]



Face Manifold



Loss Function

loss for face sample with known pose

loss for non-face sample

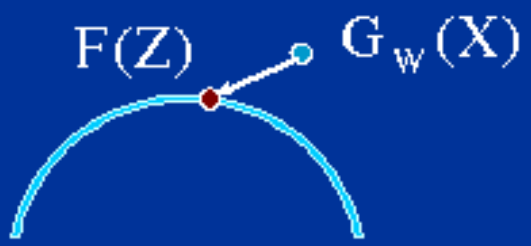
Minimize: $L(W) \quad \frac{1}{|S_1|} \sum_{i \in S_1} L_1(W, Z^i, X^i) + \frac{1}{|S_0|} \sum_{i \in S_0} L_0(W, X^i)$

training faces

training non-faces

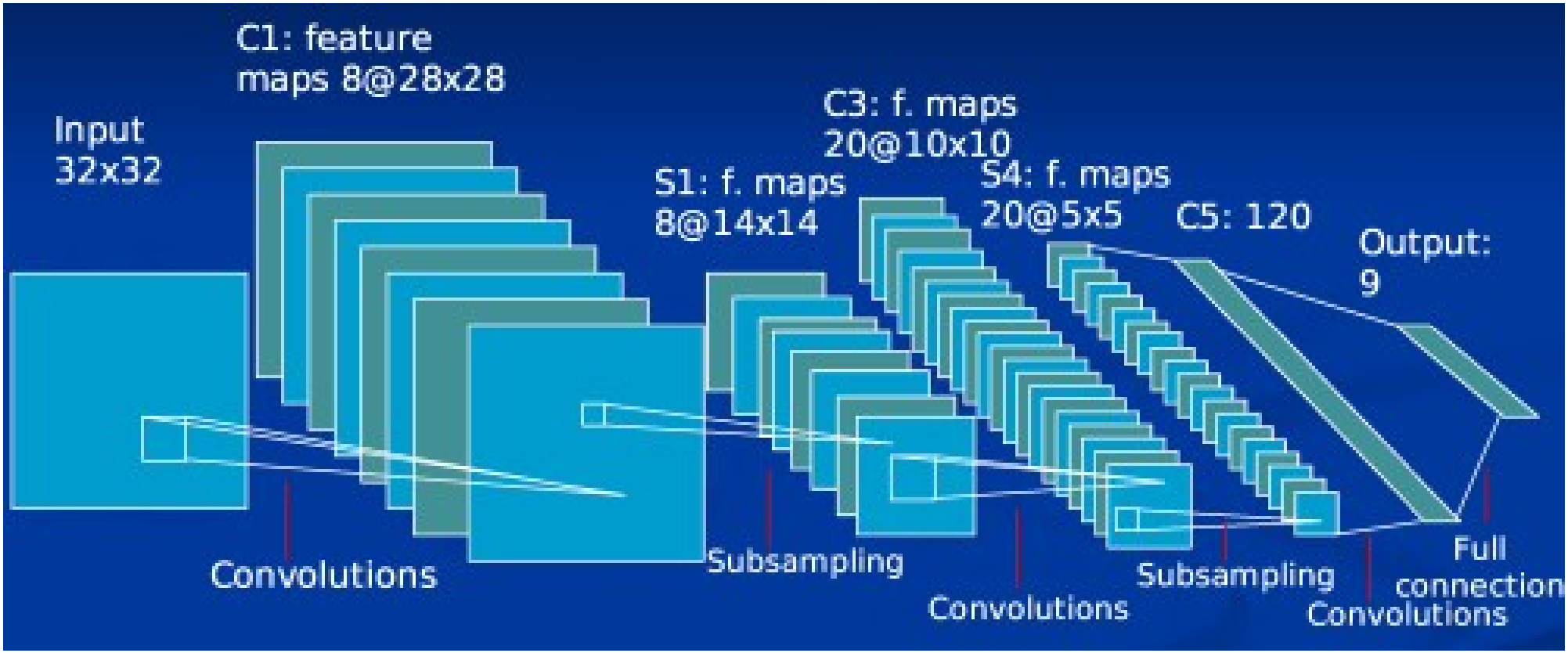
$$L_1(W, 1, Z, X) = E_W(1, Z, X)^2$$

$$L_0(W, 0, X) = K \exp\left(-E(1, \bar{Z}, X)\right)$$



Convolutional Network Architecture

[LeCun et al. 1988, 1989, 1998, 2005]



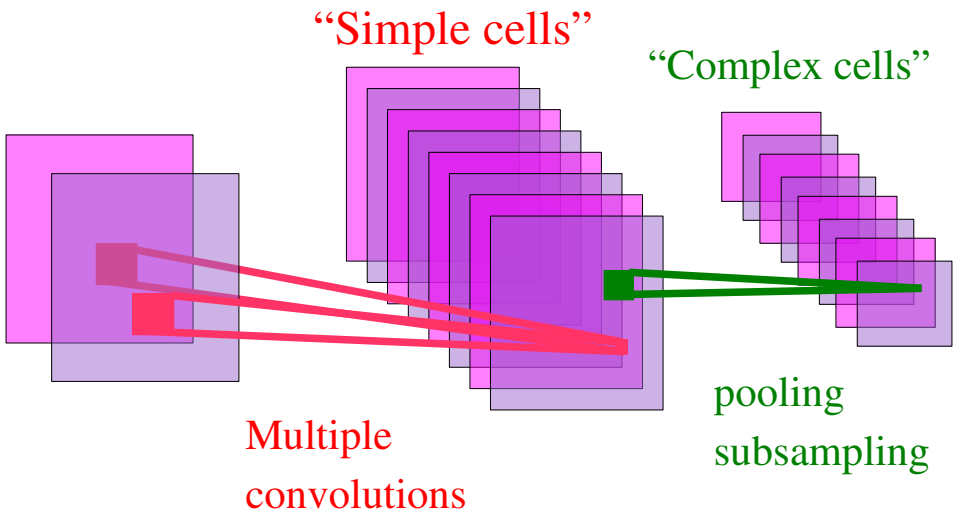
Hierarchy of local filters (convolution kernels),

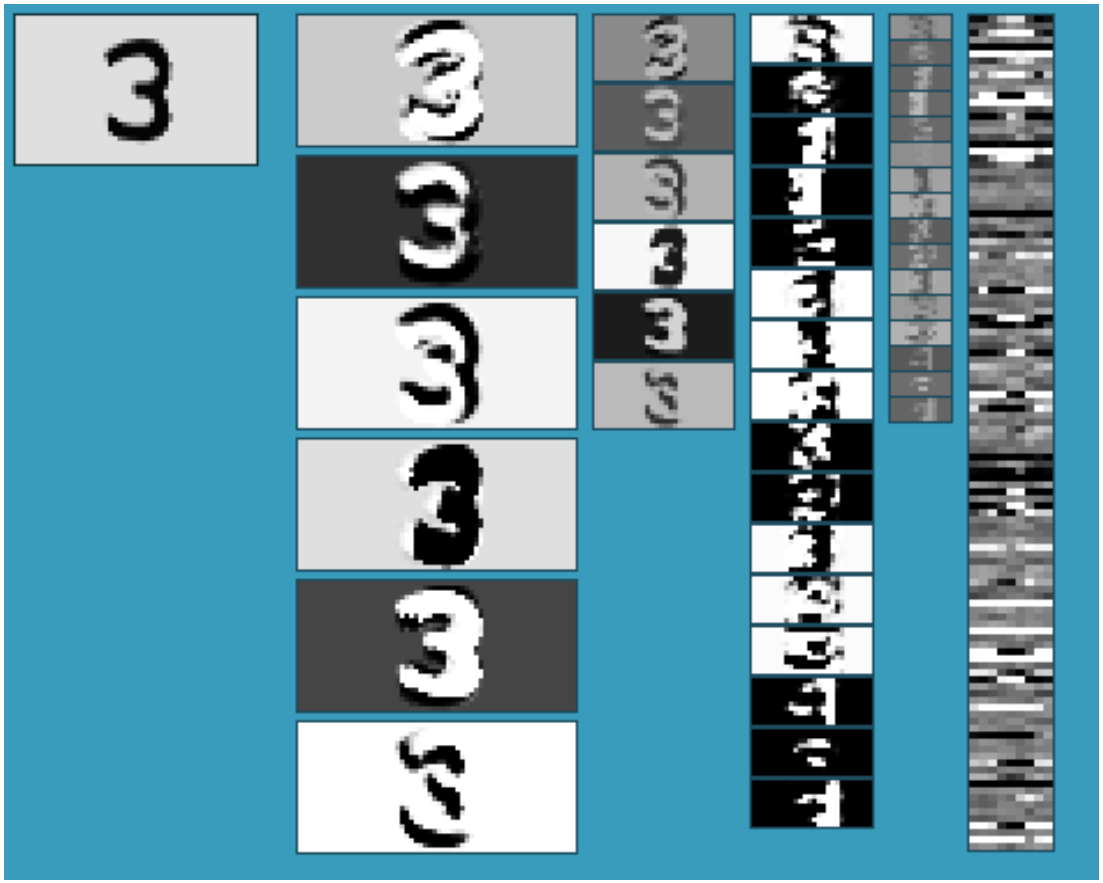
sigmoid pointwise non-linearities, and spatial subsampling

All the filter coefficients are learned with gradient descent (back-prop)

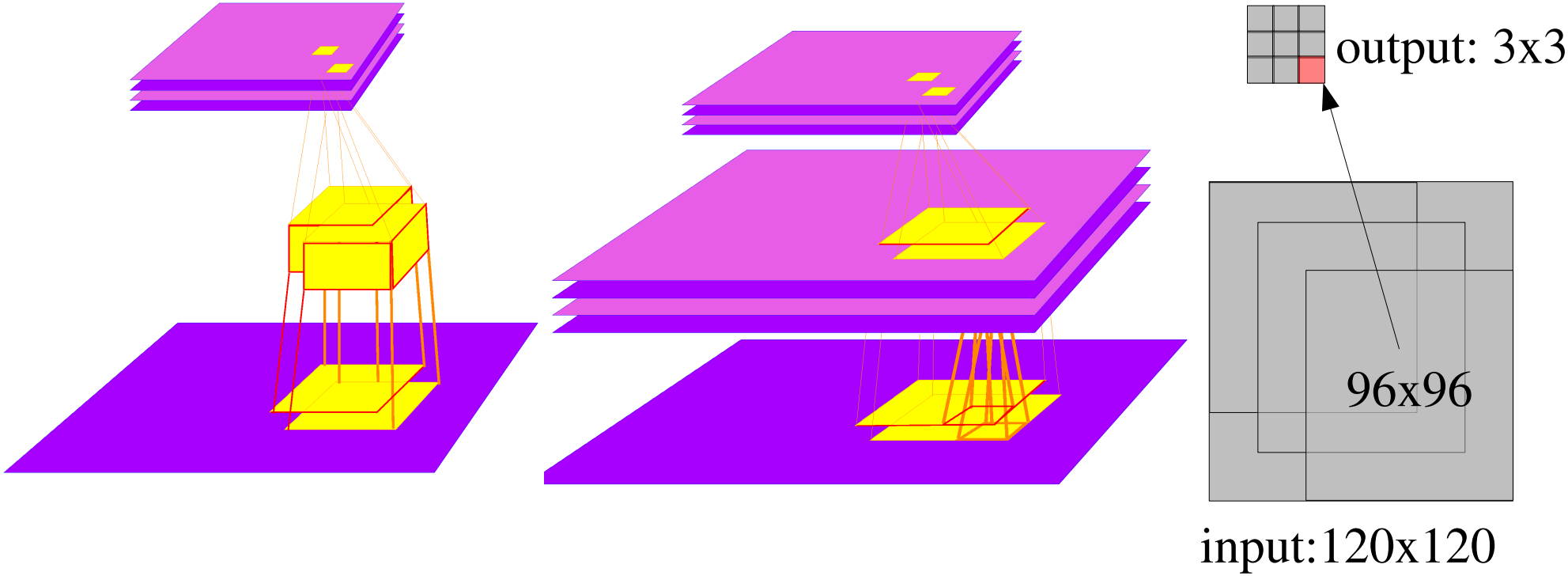
Alternated Convolutions and Pooling/Subsampling

- Local features are extracted everywhere.
- pooling/subsampling layer builds robustness to variations in feature locations.
- Long history in neuroscience and computer vision:
  - Hubel/Wiesel 1962,
  - Fukushima 1971-82,
  - LeCun 1988-06
  - Poggio, Riesenhuber, Serre 02-06
  - Ullman 2002-06
  - Triggs, Lowe,....





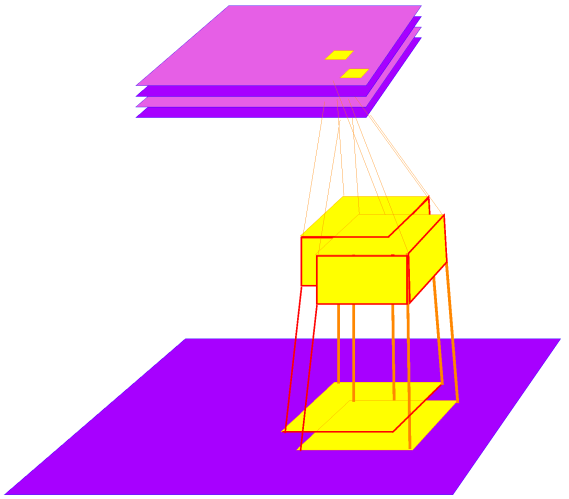
Building a Detector/Recognizer: Replicated Conv. Nets

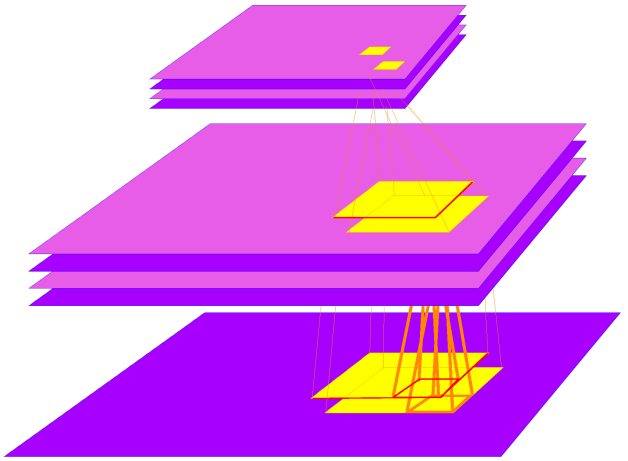


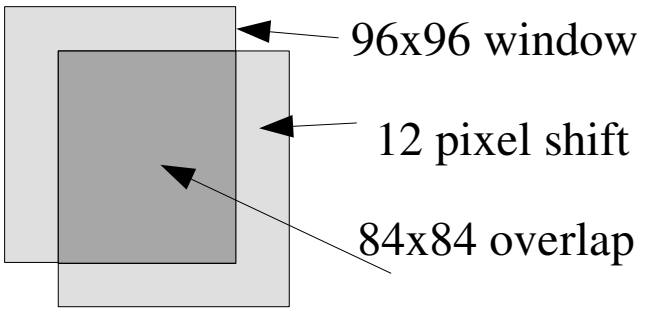
- Traditional Detectors/Classifiers must be applied to every location on a large input image, at multiple scales.
- Convolutional nets can replicated over large images very cheaply.
- The network is applied to multiple scales spaced by 1.5.

Building a Detector/Recognizer: Replicated Convolutional Nets

- Computational cost for replicated convolutional net:
  - 96x96 -> 4.6 million multiply-accumulate operations
  - 120x120 -> 8.3 million multiply-accumulate operations
  - 240x240 -> 47.5 million multiply-accumulate operations
  - 480x480 -> 232 million multiply-accumulate operations
- Computational cost for a non-convolutional detector of the same size, applied every 12 pixels:
  - 96x96 -> 4.6 million multiply-accumulate operations
  - 120x120 -> 42.0 million multiply-accumulate operations
  - 240x240 -> 788.0 million multiply-accumulate operations
  - 480x480 -> 5,083 million multiply-accumulate operations

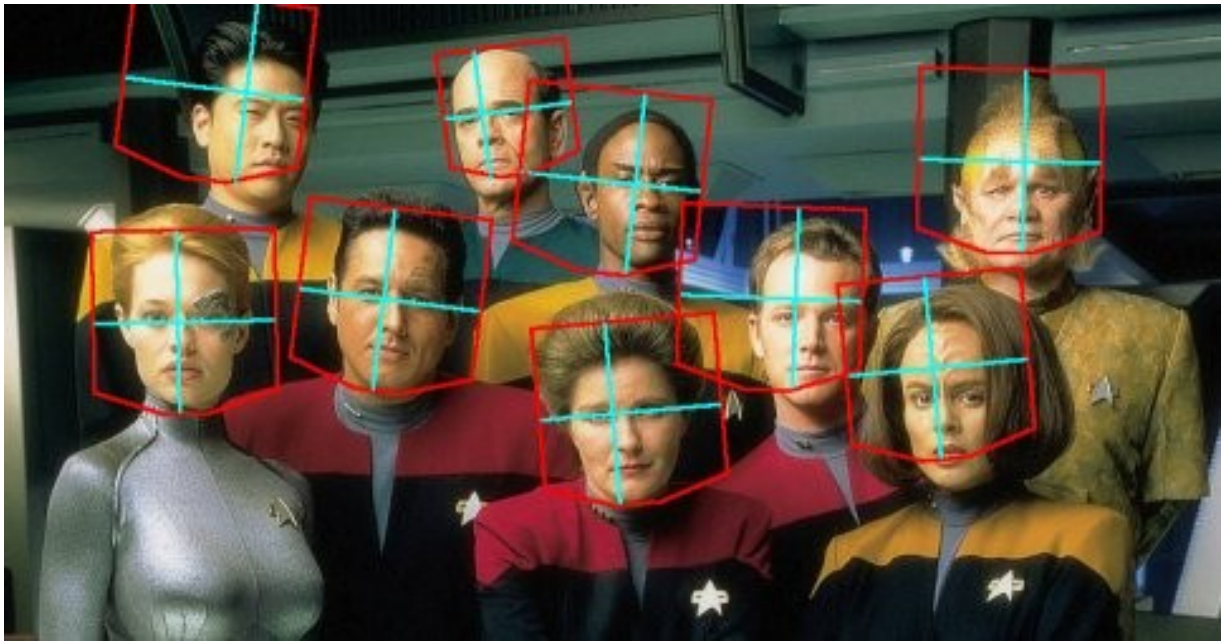


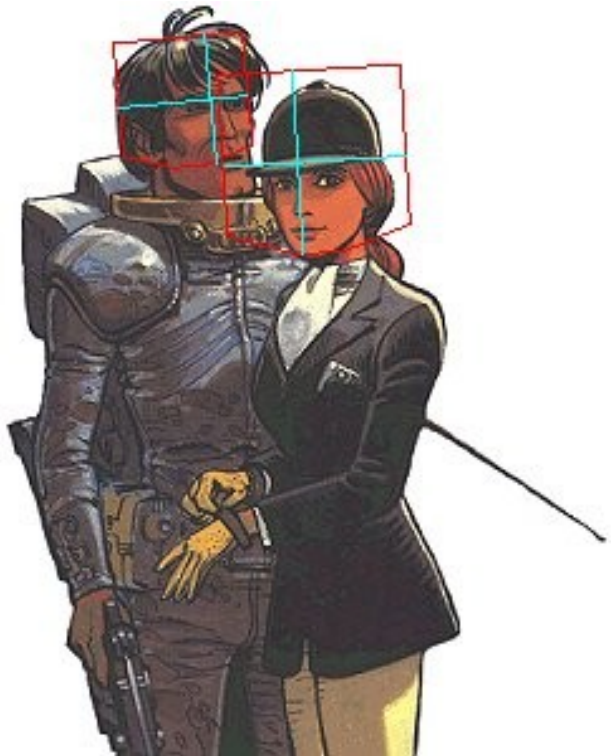




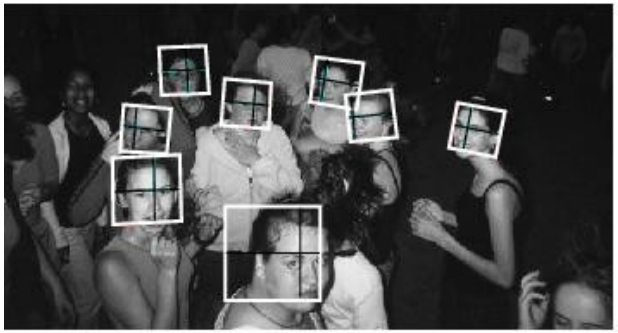
Face Detection: Results

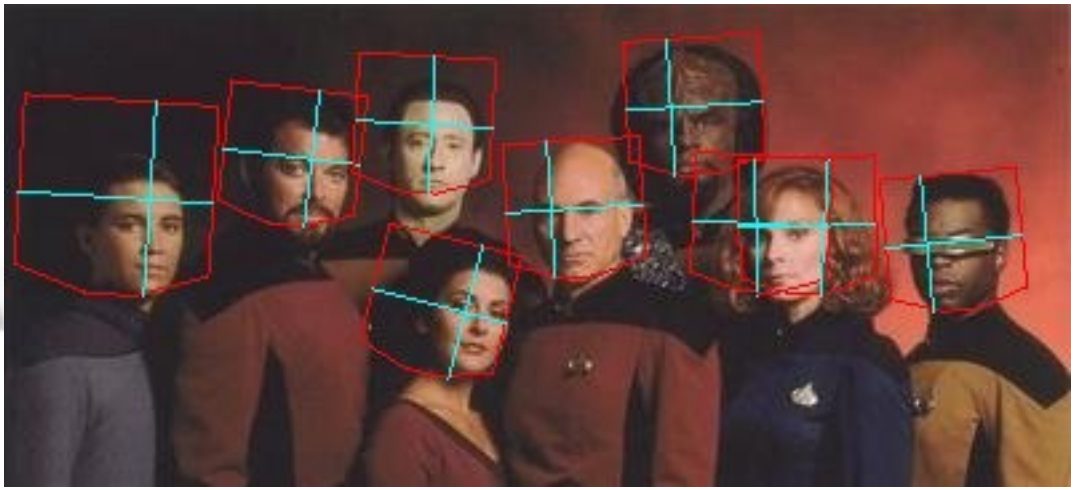
Data Set->	TILTED		PROFILE		MIT+CMU	
False positives per image->	4.42	26.9	0.47	3.36	0.5	1.28
Our Detector	90%	97%	67%	83%	83%	88%
Jones & Viola (tilted)	90%	95%	x		x	
Jones & Viola (profile)	x		70%	83%	x	

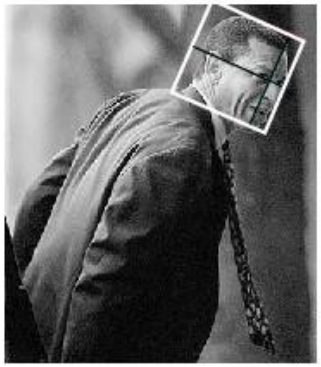


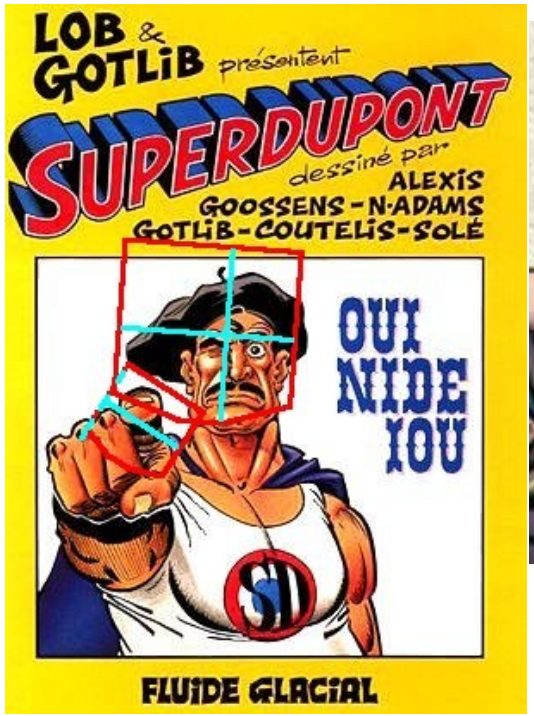


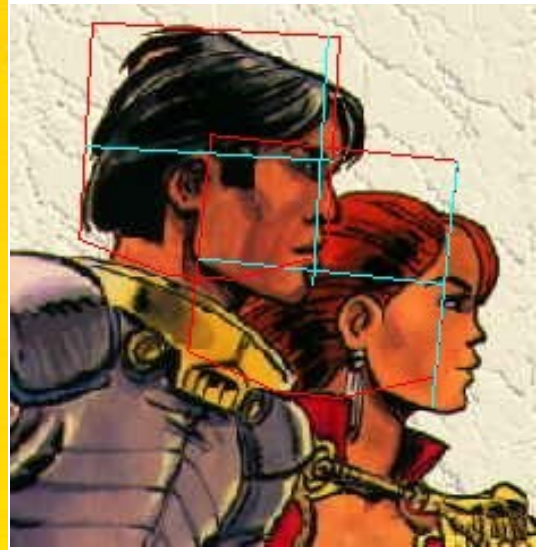
Face Detection and Pose Estimation: Results











Face Detection with a Convolutional Net

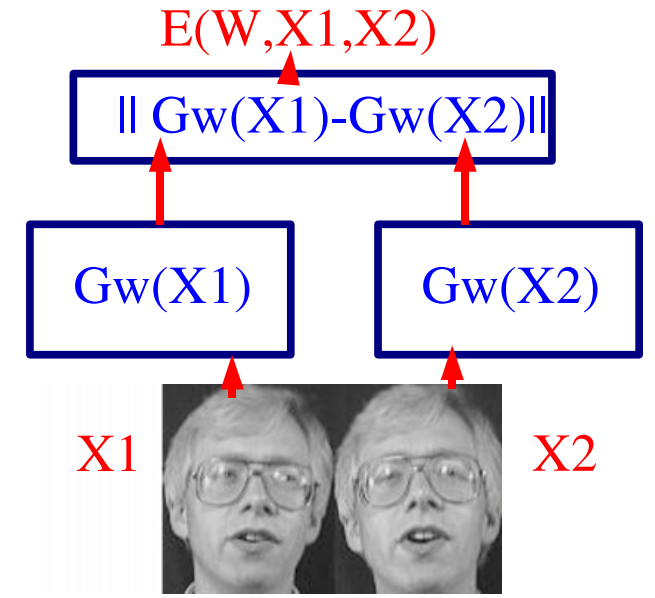
How do we Handle Lots of Classes?

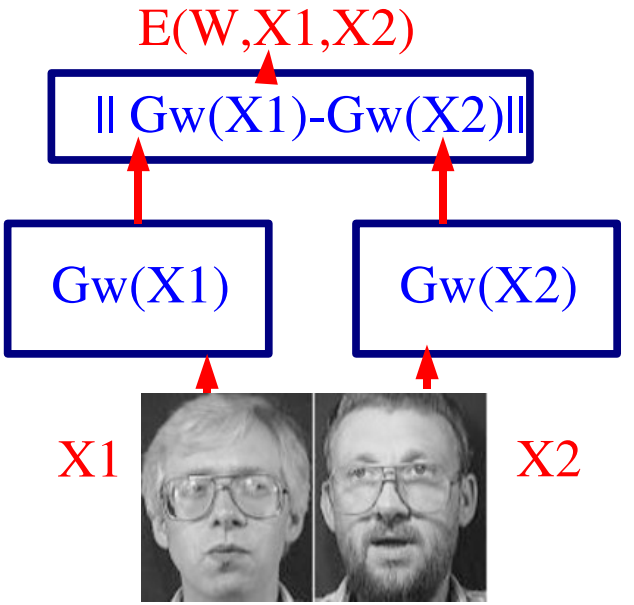
- **Example: face recognition**
  - We do not have pictures of every person
- **We must be able to learn something without seeing all the classes**
- **Solution: learn a similarity metric**
- **Map images to a low dimensional space in which**
  - Two images of the same person are mapped to nearby points
  - Two images of different persons are mapped to distant points

Comparing Objects: Learning an Invariant Dissimilarity Metric

[Chopra, Hadsell, LeCun CVPR 2005]

- Training a **parameterized, invariant dissimilarity metric** may be a solution to the **many-category problem**.
- Find a mapping **Gw(X)** such that the Euclidean distance **||Gw(X1)- Gw(X2)||** reflects the "semantic" distance between X1 and X2.
- Once trained, a trainable dissimilarity metric can be used to classify **new categories using a very small number of training samples** (used as prototypes).
- This is an example where probabilistic models are too constraining, because we would have to limit ourselves to models that can be normalized over the space of input pairs.
- With EBMs, we can put what we want in the box (e.g. A convolutional net).
- **Siamese Architecture**
- **Application:** face verification/recognition





Face Verification datasets: AT&T/ORL

- The AT&T/ORL dataset
- Total subjects: **40**. Images per subject: **10**. Total images: **400**.
- Images had a moderate degree of variation in pose, lighting, expression and head position.
- Images from **35** subjects were used for training. Images from **5** remaining subjects for testing.
- Training set was taken from: **3500** genuine and **119000** impostor pairs.
- Test set was taken from: **500** genuine and **2000** impostor pairs.
- http://www.uk.research.att.com/facedatabase.html

AT&T/ORL Dataset

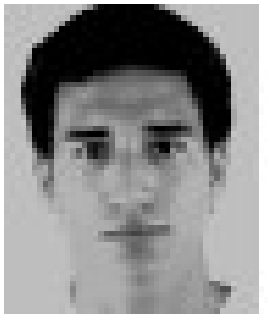
Internal state for genuine and impostor pairs

Classification Examples

- **Example: Correctly classified genuine pairs**

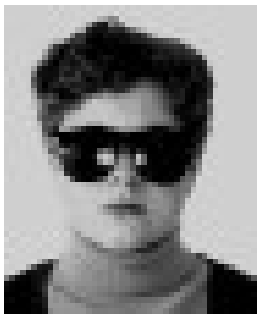

energy: 0.3159


energy: 0.0043




energy: 0.0046

- **Example: Correctly classified impostor pairs**

energy: 20.1259


energy: 32.7897


energy: 5.7186

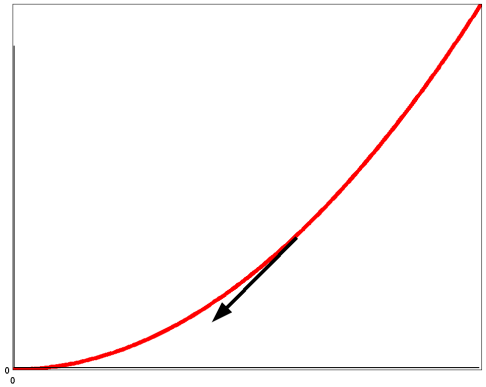
- **Example: Mis-classified pairs**


energy: 10.3209

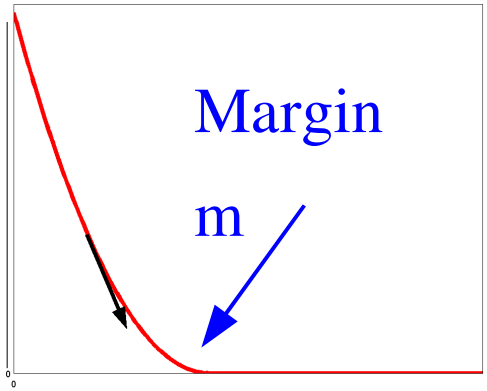

energy: 2.8243

A similar idea for Learning a Manifold with Invariance Properties

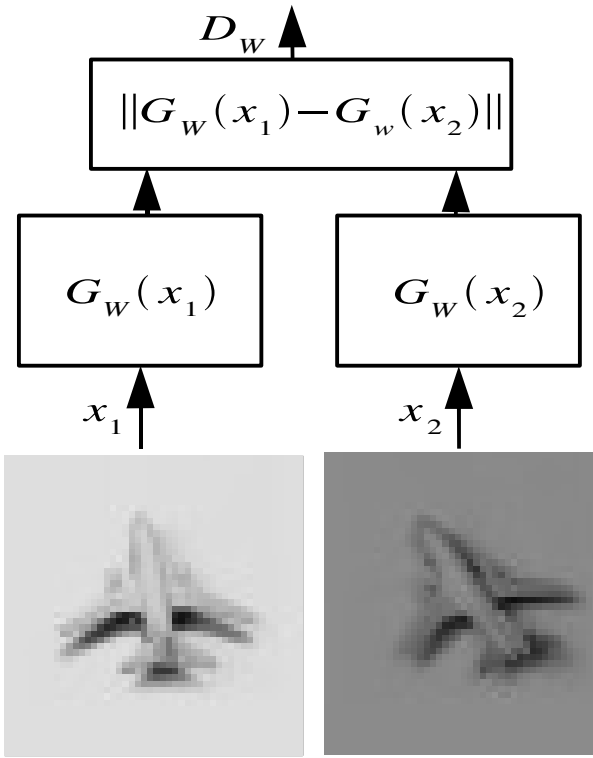
$$L_{similar} = \frac{1}{2} D_w^2$$

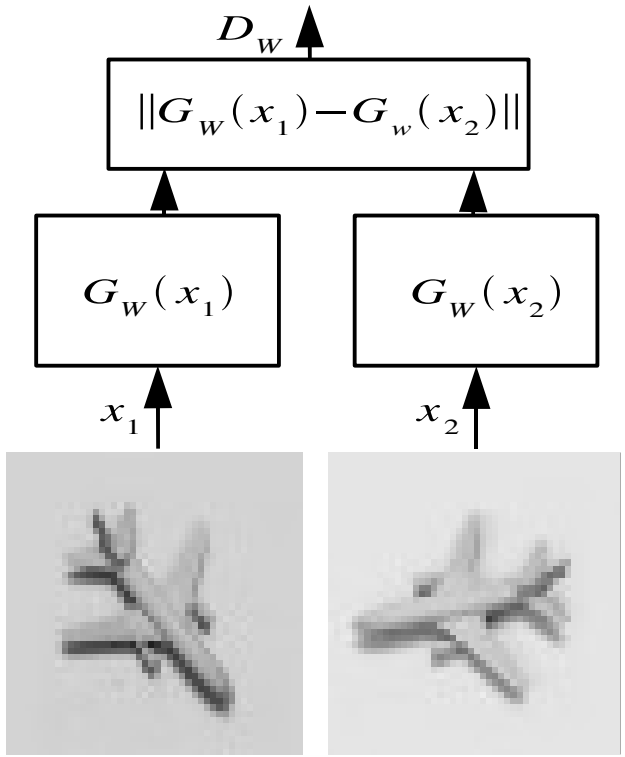


$$L_{dissimilar} = \frac{1}{2} \{ max(0, m - D_W) \}^2$$

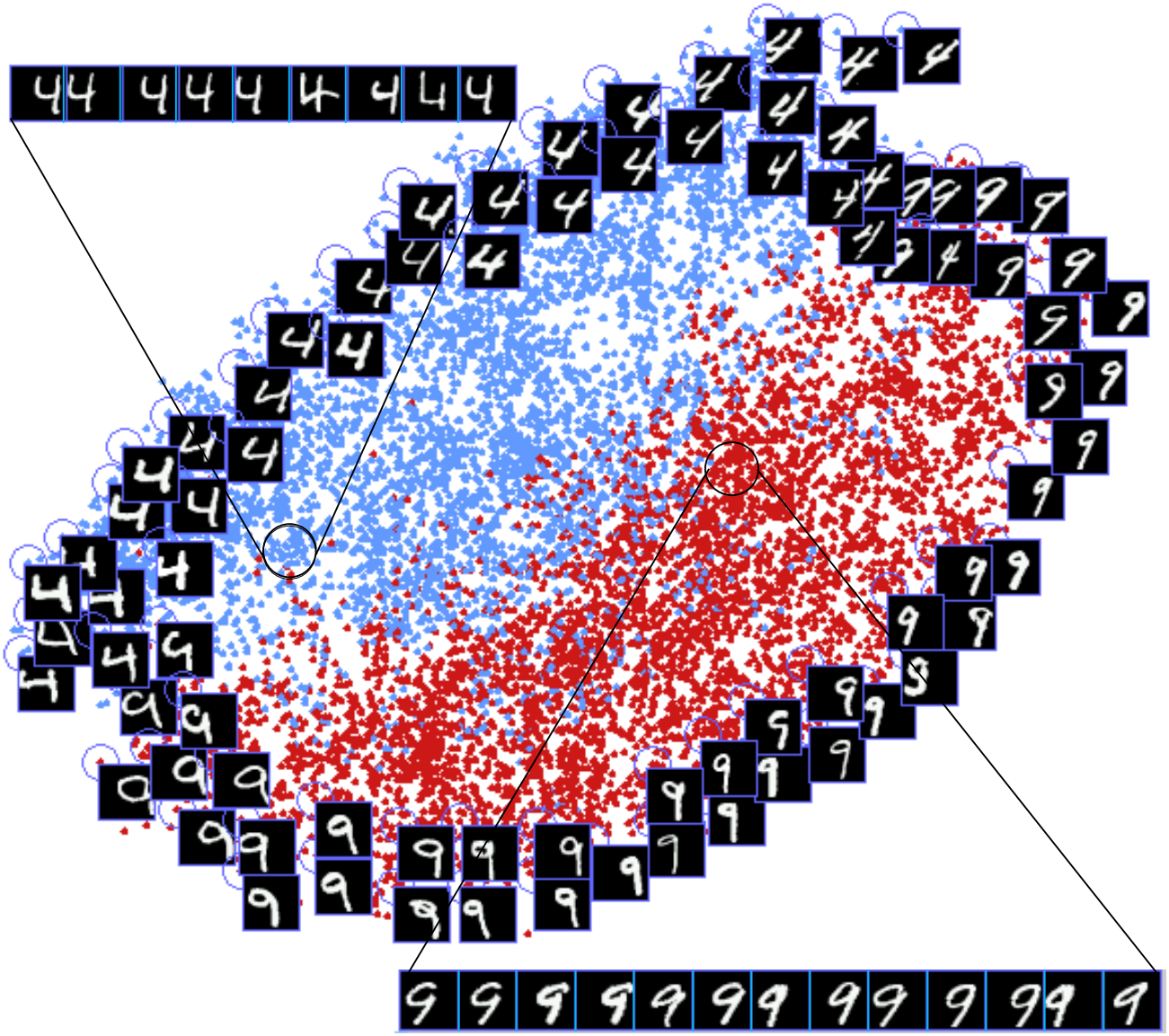


- **Loss function:**
  - Pay quadratically for making outputs of neighbors far apart
  - Pay quadratically for making outputs of non-neighbors smaller than a **margin** m



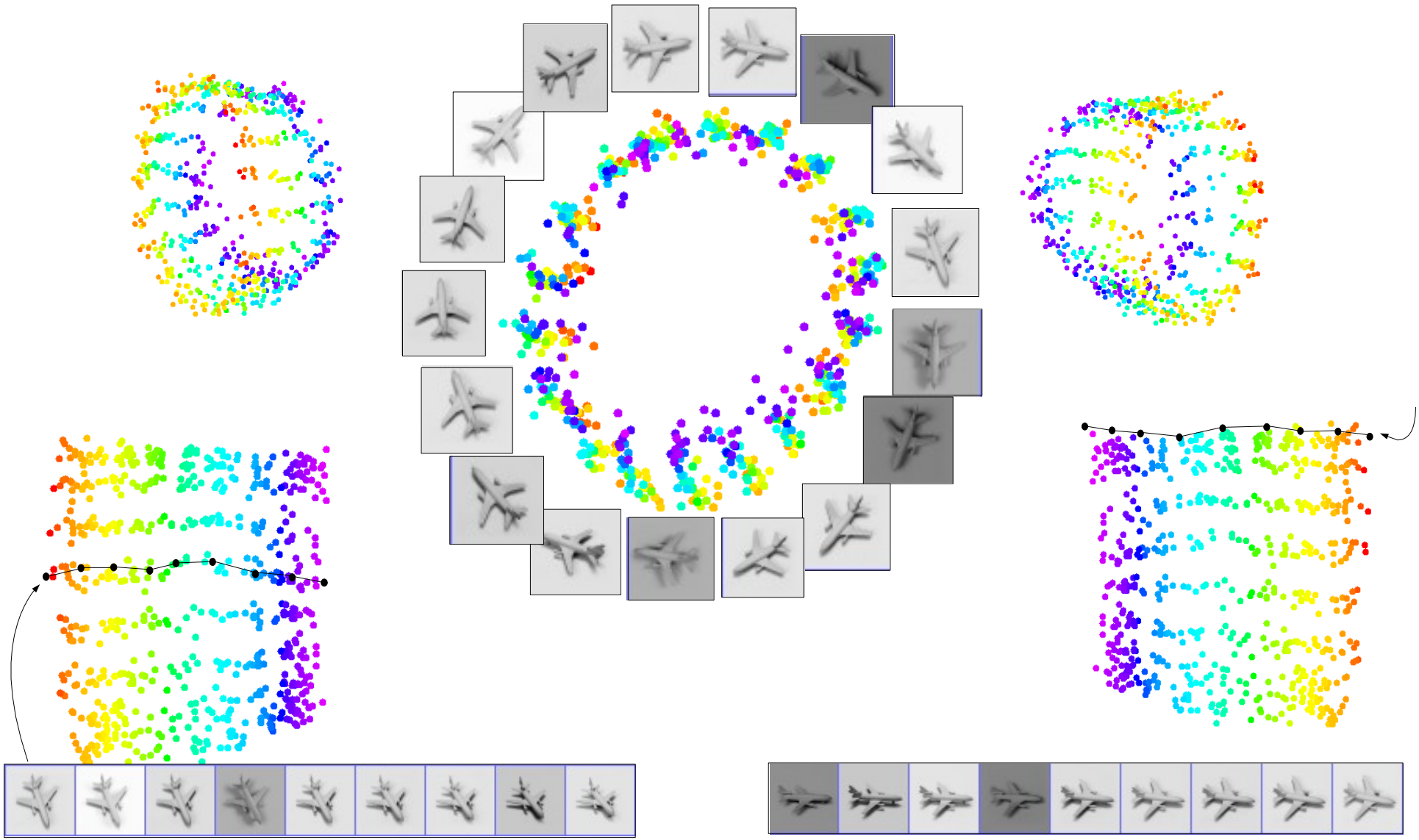


A Manifold with Invariance to Shifts



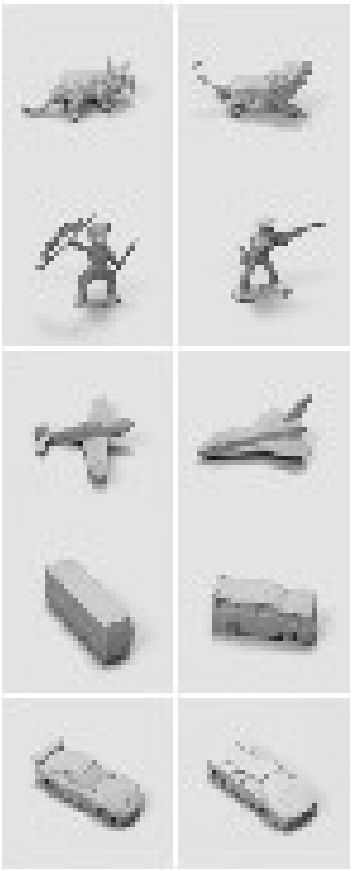
- Training set: 3000 “4” and 3000 “9” from MNIST. **Each digit is shifted horizontally by -6, -3, 3, and 6 pixels**
- Neighborhood graph: 5 nearest neighbors in Euclidean distance, **and shifted versions of self and nearest neighbors**
- Output Dimension: 2
- Test set (shown) 1000 “4” and 1000 “9”

Automatic Discovery of the Viewpoint Manifold with Invariant to Illumination



Generic Object Detection and Recognition with Invariance to Pose, Illumination and Clutter

- **Computer Vision and Biological Vision are getting back together again after a long divorce (Hinton, LeCun, Poggio, Perona, Ullman, Lowe, Triggs, S. Geman, Itti, Olshausen, Simoncelli, ...).**
- **What happened? (1) Machine Learning, (2) Moore's Law.**
- **Generic Object Recognition** is the problem of detecting and classifying objects into generic categories such as "cars", "trucks", "airplanes", "animals", or "human figures"
- **Appearances are highly variable within a category** because of shape variation, position in the visual field, scale, viewpoint, illumination, albedo, texture, background clutter, and occlusions.
- **Learning invariant representations is key.**
- **Understanding the neural mechanism behind invariant recognition is one of the main goals of Visual Neuroscience.**

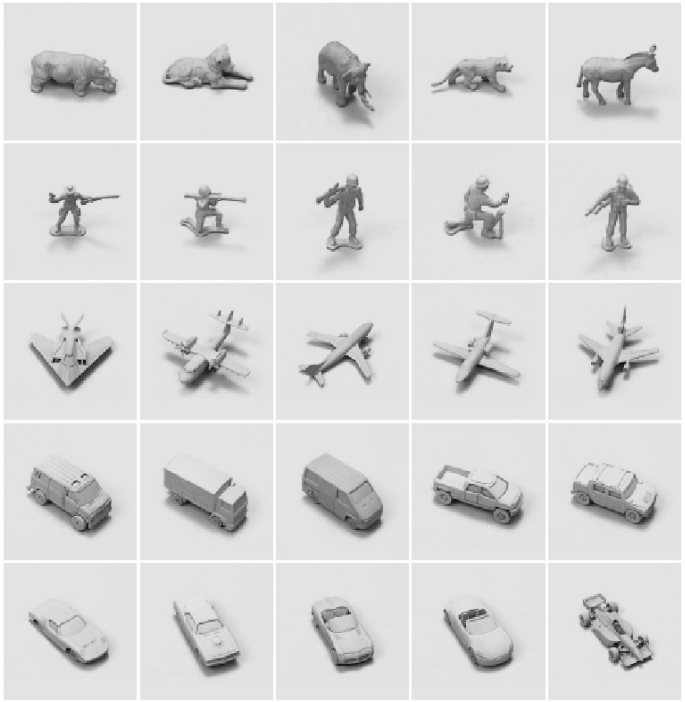


Why do we need "Deep" Architectures?

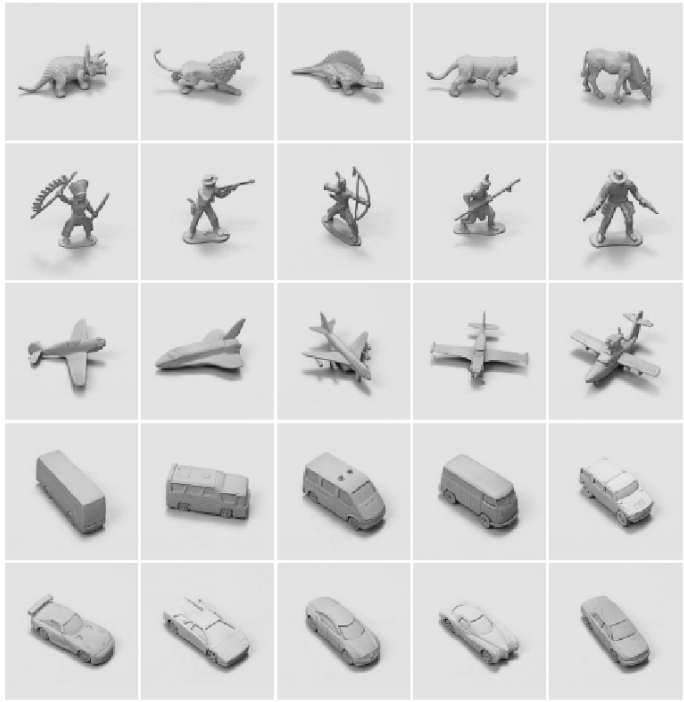
- **Conjecture: we won't solve the perception problem without solving the problem of learning in deep architectures [Hinton]**
  - Neural nets with lots of layers
  - Deep belief networks
  - Factor graphs with a "Markov" structure
- **We will not solve the perception problem with kernel machines**
  - Kernel machines are glorified template matchers
  - You can't handle complicated invariances with templates (you would need too many templates)
- **Many interesting functions are "deep"**
  - Any function can be approximated with 2 layers (linear combination of non-linear functions)
  - But many interesting functions a more efficiently represented with multiple layers
  - Stupid examples: binary addition

Generic Object Detection and Recognition with Invariance to Pose and Illumination

- **50** toys belonging to 5 categories: **animal, human figure, airplane, truck, car**
- **10** instance per category: 5 instances used for training, 5 instances for testing
- **Raw dataset: 972** stereo pair of each object instance. **48,600** image pairs total.
- **For each instance:**
  - **18 azimuths**
    - 0 to 350 degrees every 20 degrees
  - **9 elevations**
    - 30 to 70 degrees from horizontal every 5 degrees
  - **6 illuminations**
    - on/off combinations of 4 lights
  - **2 cameras (stereo)**
    - 7.5 cm apart
    - 40 cm from the object

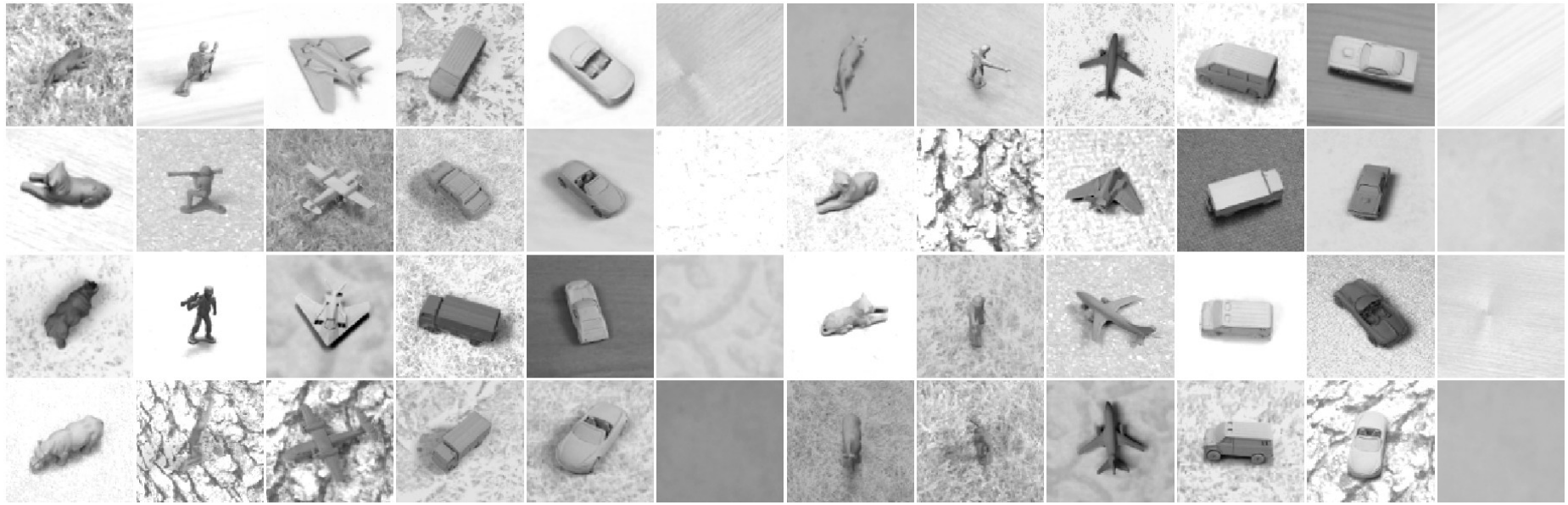


Training instances

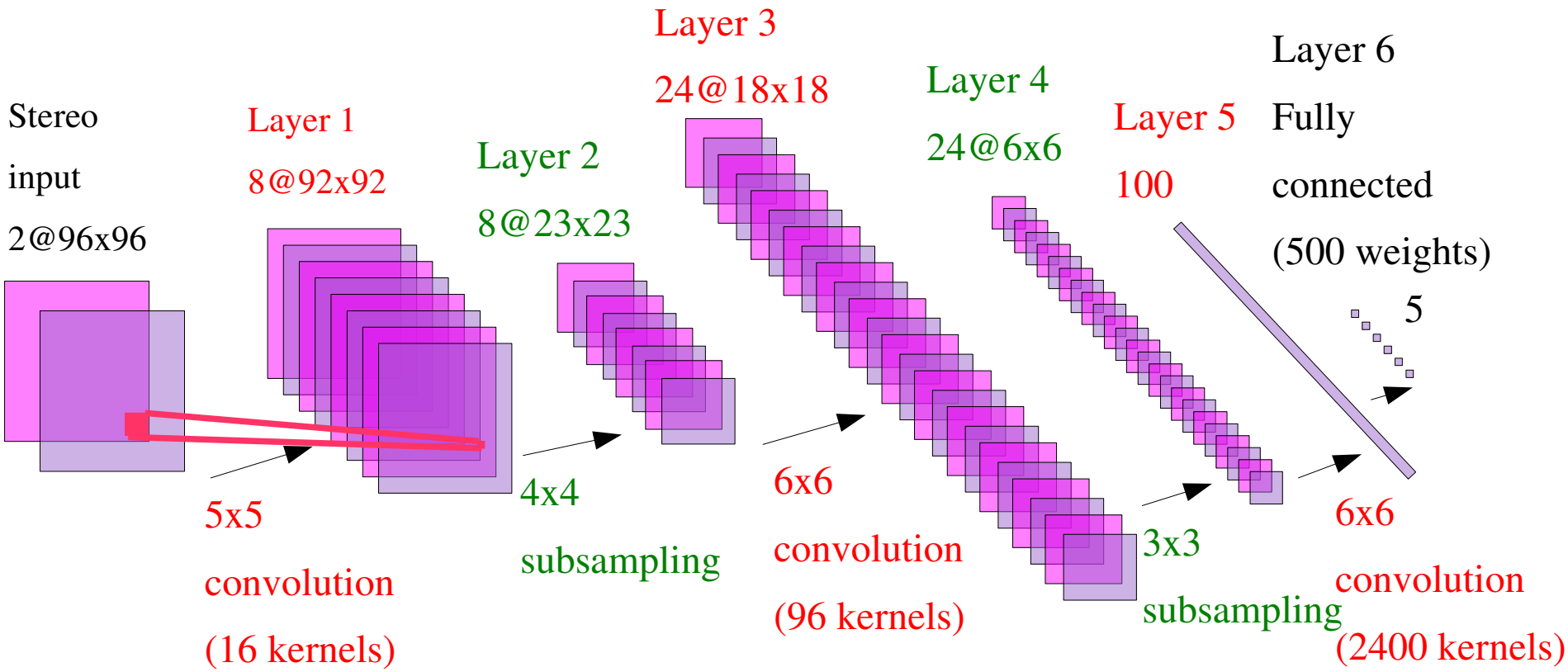


Test instances

Textured and Cluttered Datasets

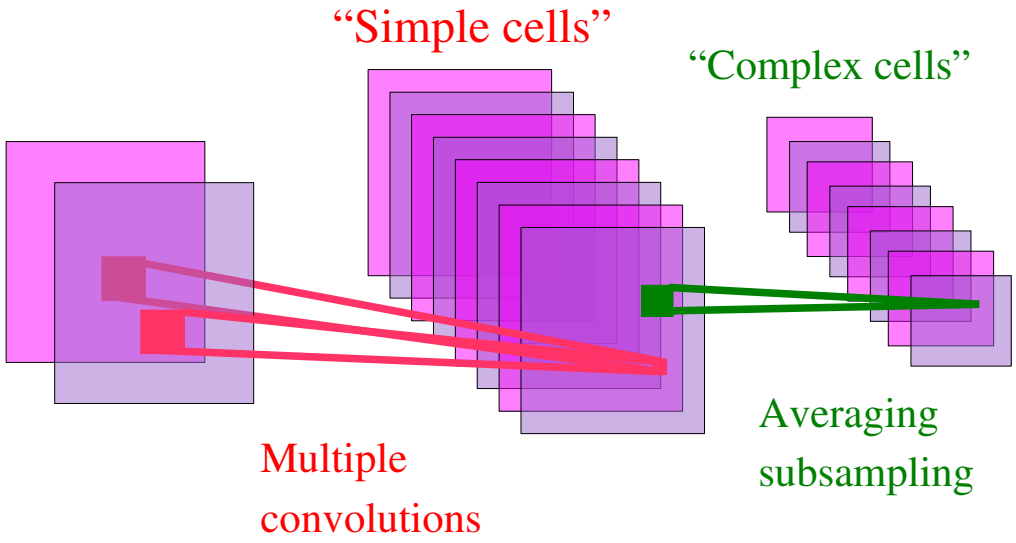


Convolutional Network

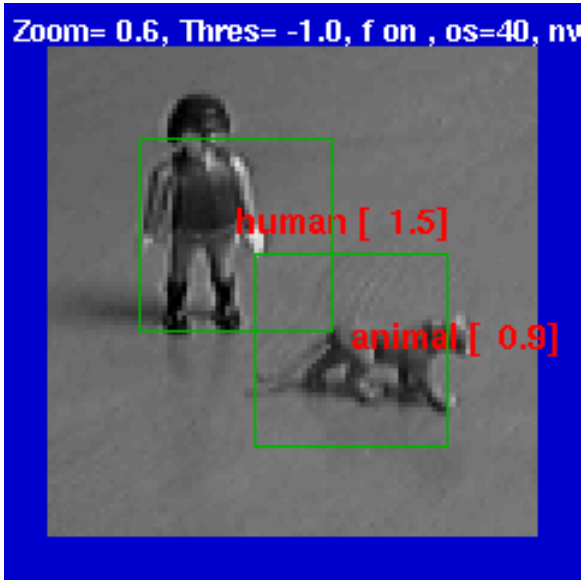


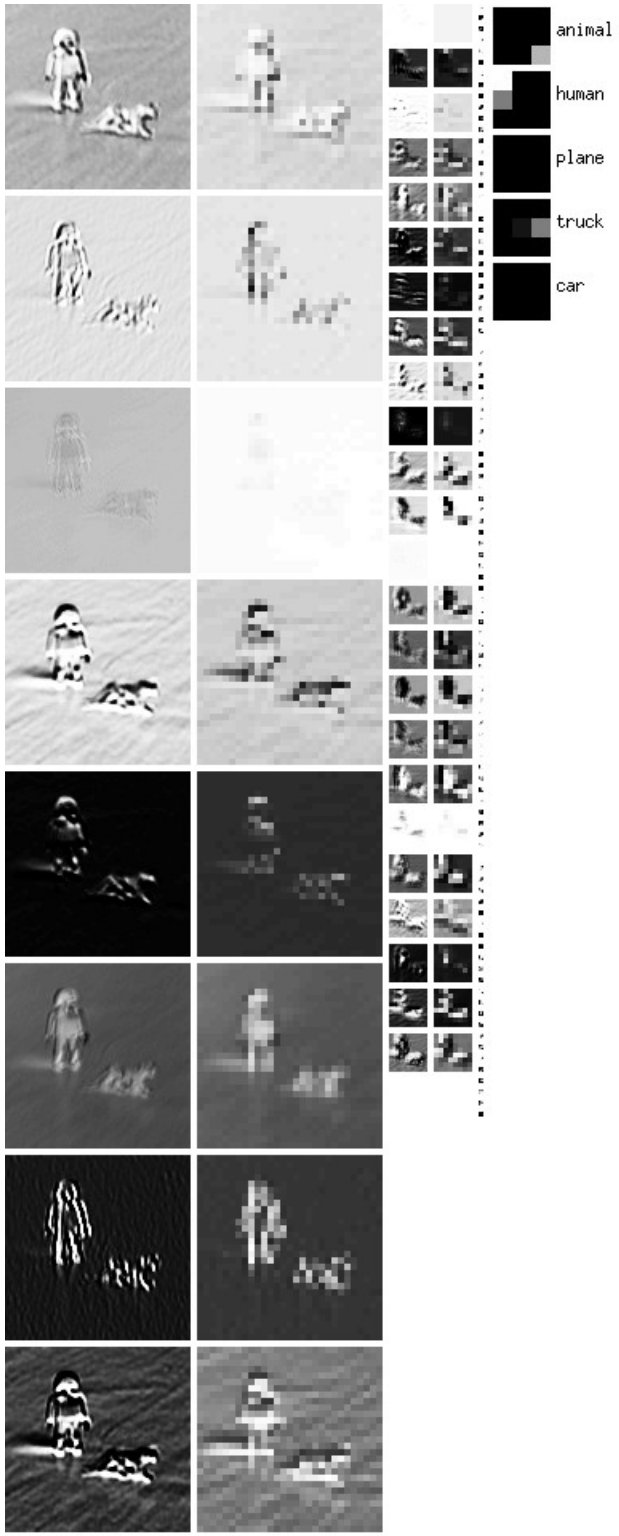
- **90,857 free parameters, 3,901,162 connections.**
- The architecture alternates convolutional layers (feature detectors) and subsampling layers (local feature pooling for invariance to small distortions).
- **The entire network is trained end-to-end** (all the layers are trained simultaneously).
- A gradient-based algorithm is used to minimize a supervised loss function.

Alternated Convolutions and Subsampling



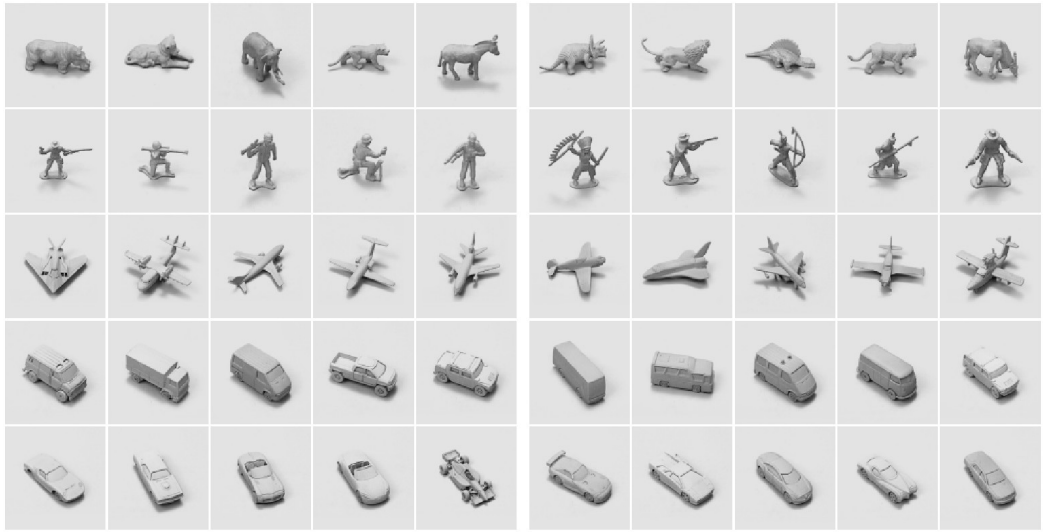
- **Local features are extracted everywhere.**
- **averaging/subsampling layer builds robustness to variations in feature locations.**
- **Hubel/Wiesel'62, Fukushima'71, LeCun'89, Riesenhuber & Poggio'02, Ullman'02,....**





Normalized-Uniform Set: Error Rates

- **Linear Classifier on raw stereo images:** **30.2% error.**
- **K-Nearest-Neighbors on raw stereo images:** **18.4% error.**
- **K-Nearest-Neighbors on PCA-95:** **16.6% error.**
- **Pairwise SVM on 96x96 stereo images:** **11.6% error**
- **Pairwise SVM on 95 Principal Components:** **13.3% error.**
- **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	σ=2,000 C=40					dim=80 σ=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

Jittered-Cluttered Dataset

- **Jittered-Cluttered Dataset:**
- **291,600** tereo pairs for training, **58,320** for testing
- Objects are jittered: position, scale, in-plane rotation, contrast, brightness, backgrounds, distractor objects,...
- Input dimension: 98x98x2 (approx 18,000)

Experiment 2: Jittered-Cluttered Dataset

- 291,600 training samples, 58,320 test samples
- 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	σ=10^4 C=40				dim=100 σ=5 C=1

OUCH!

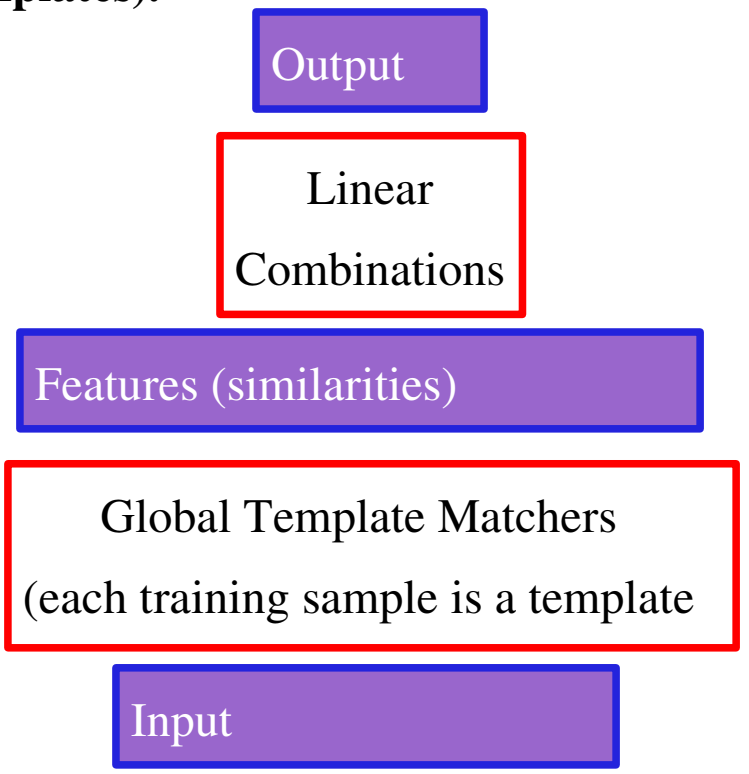
The convex loss, VC bounds and representers theorems don't seem to help

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

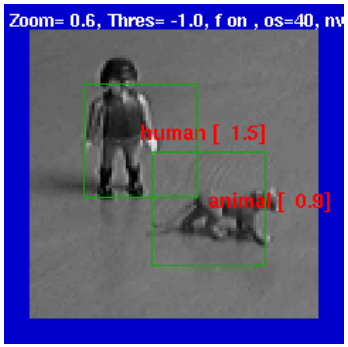
What's wrong with K-NN and SVMs?

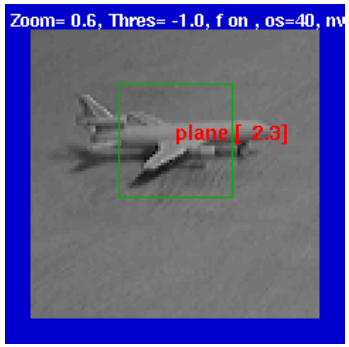
- **K-NN and SVM with Gaussian kernels are based on matching global templates**
- **Both are "shallow" architectures**
- **There is now way to learn invariant recognition tasks with such naïve architectures (unless we use an impractically large number of templates).**

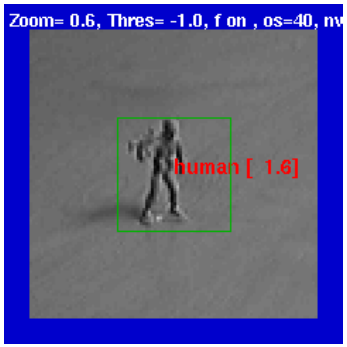
- **The number of necessary templates grows exponentially with the number of dimensions of variations.**
- **Global templates are in trouble when the variations include: category, instance shape, configuration (for articulated object), position, azimuth, elevation, scale, illumination, texture, albedo, in-plane rotation, background luminance, background texture, background clutter, ...**

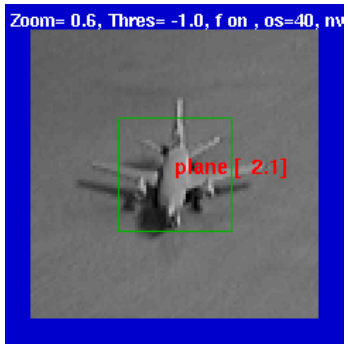


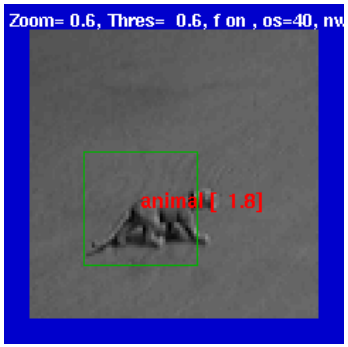
Examples (Monocular Mode)

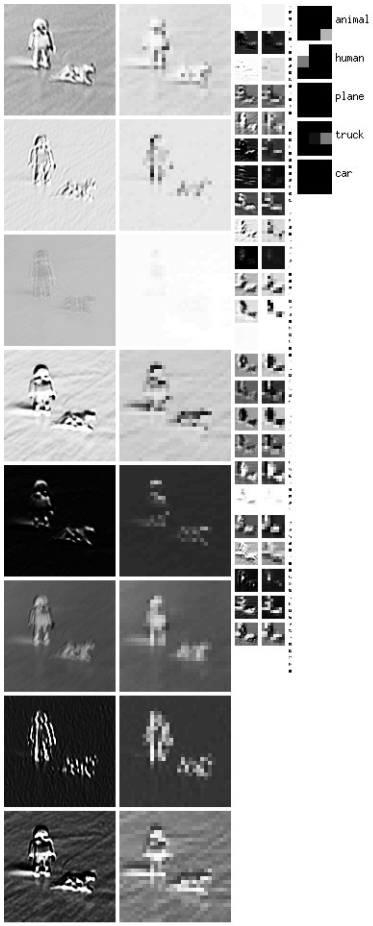


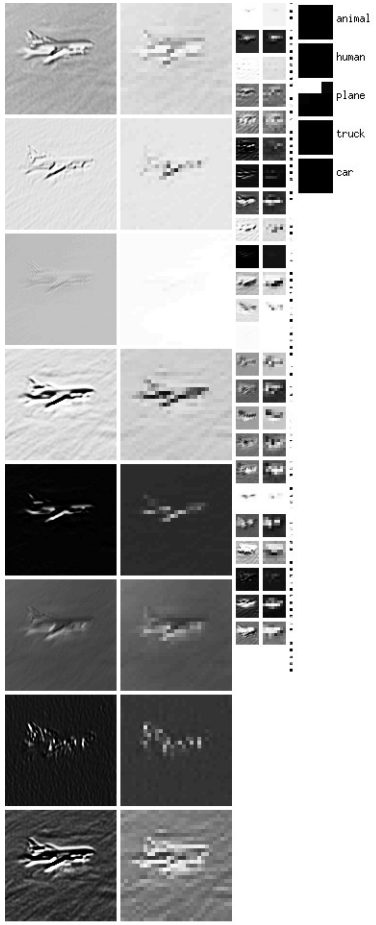


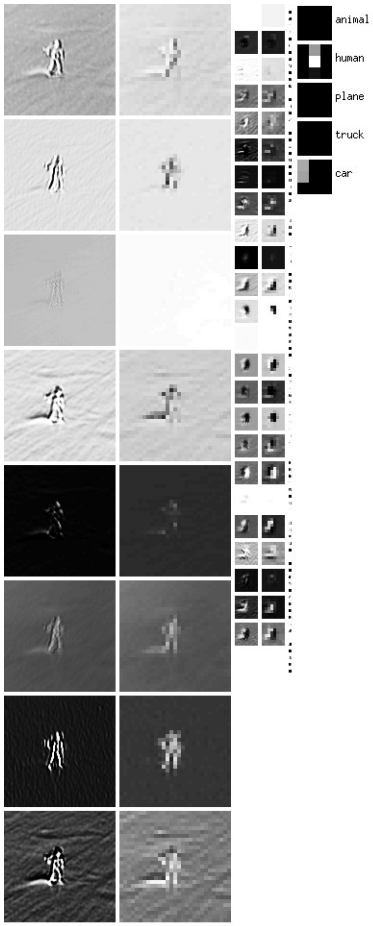


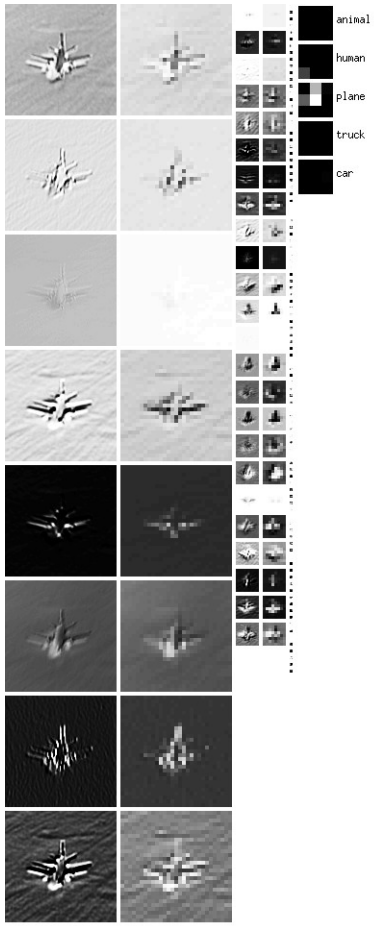


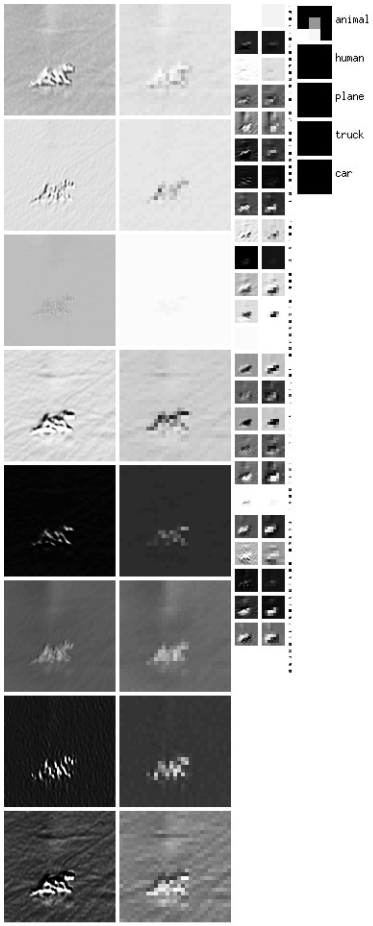








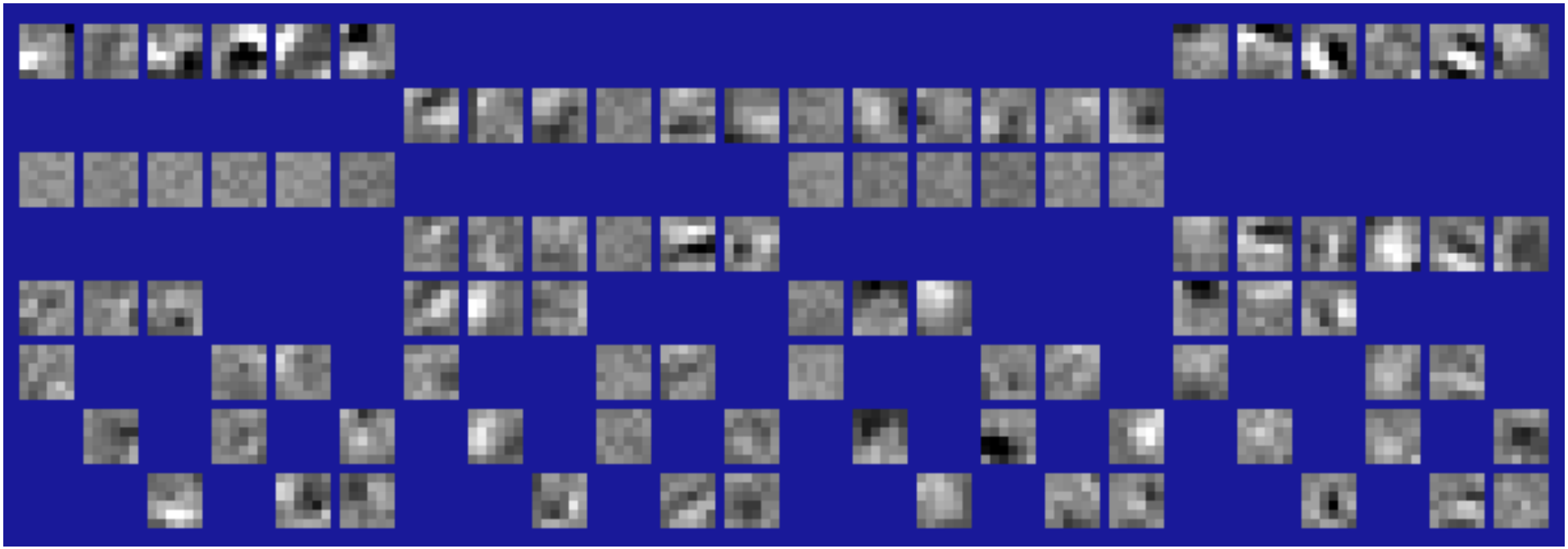




Learned Features

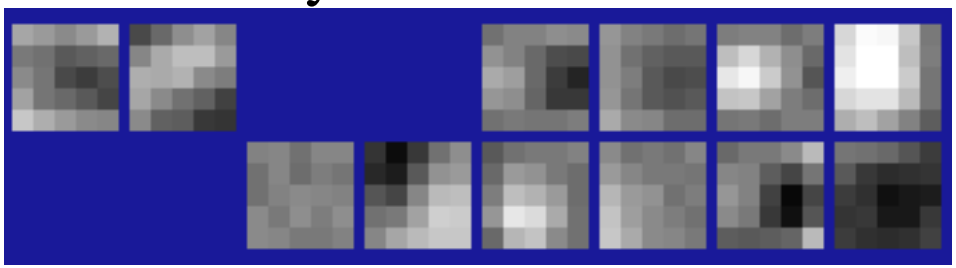
Layer 3

Layer 2

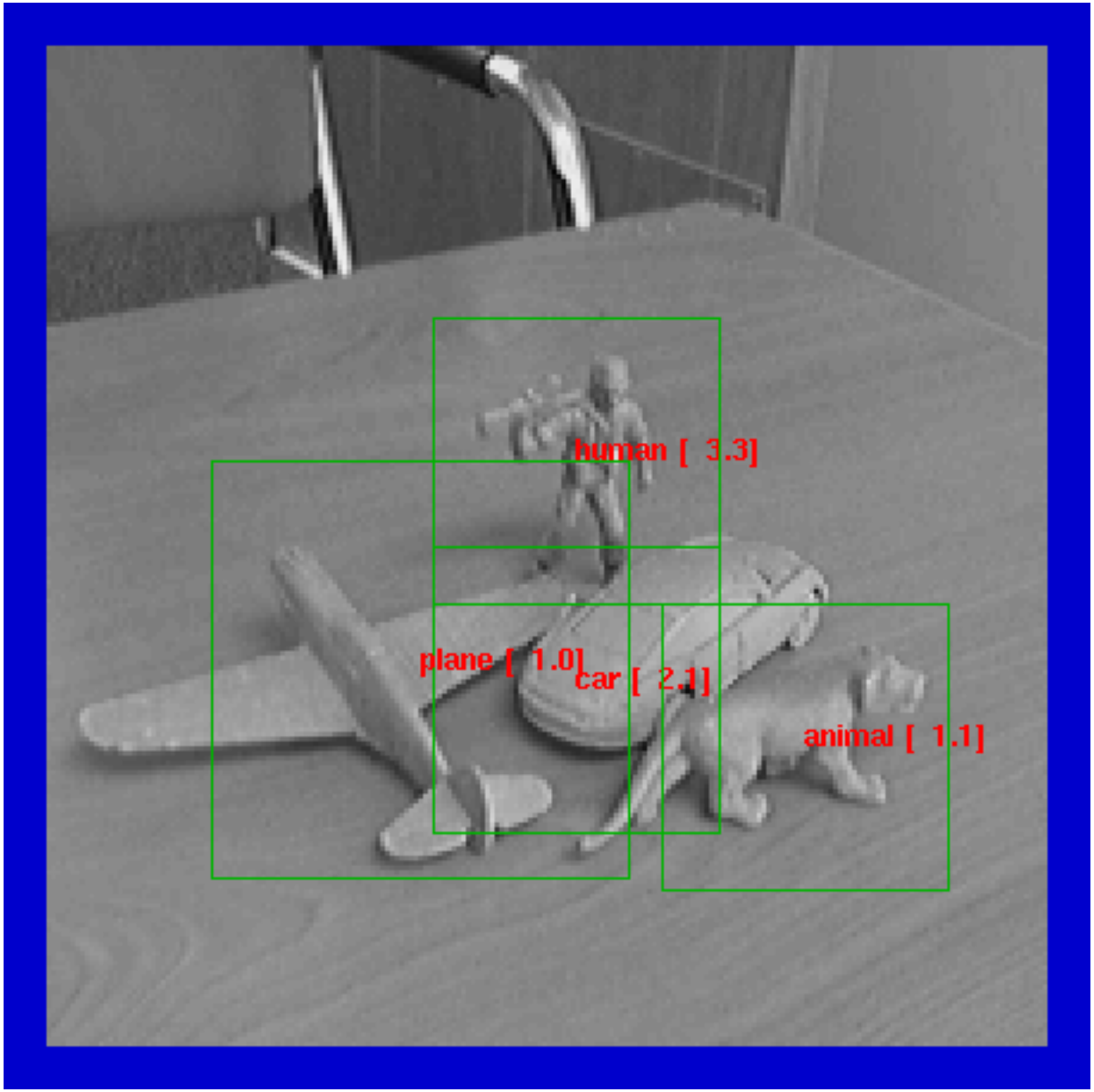


Layer 1

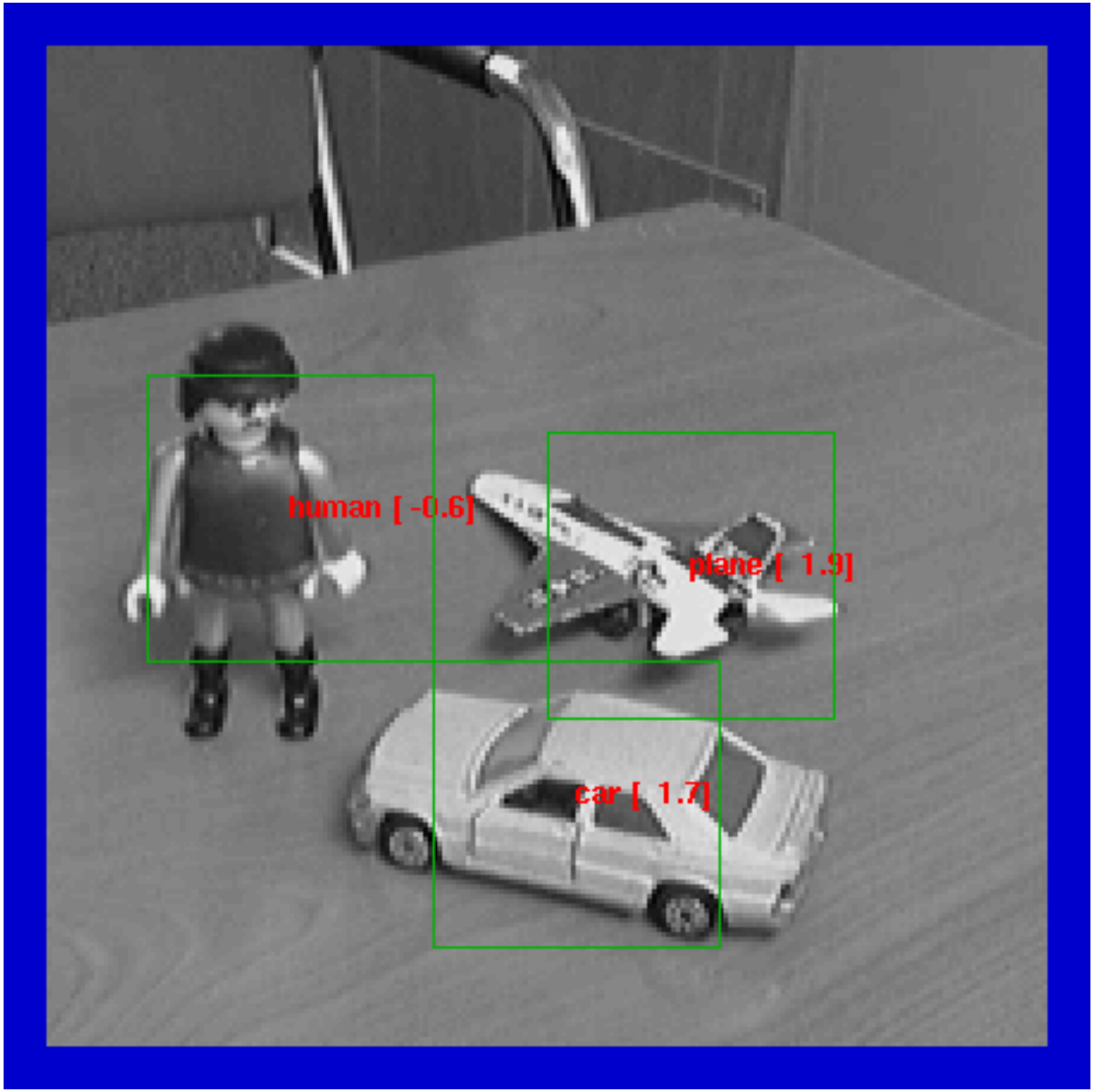
Input



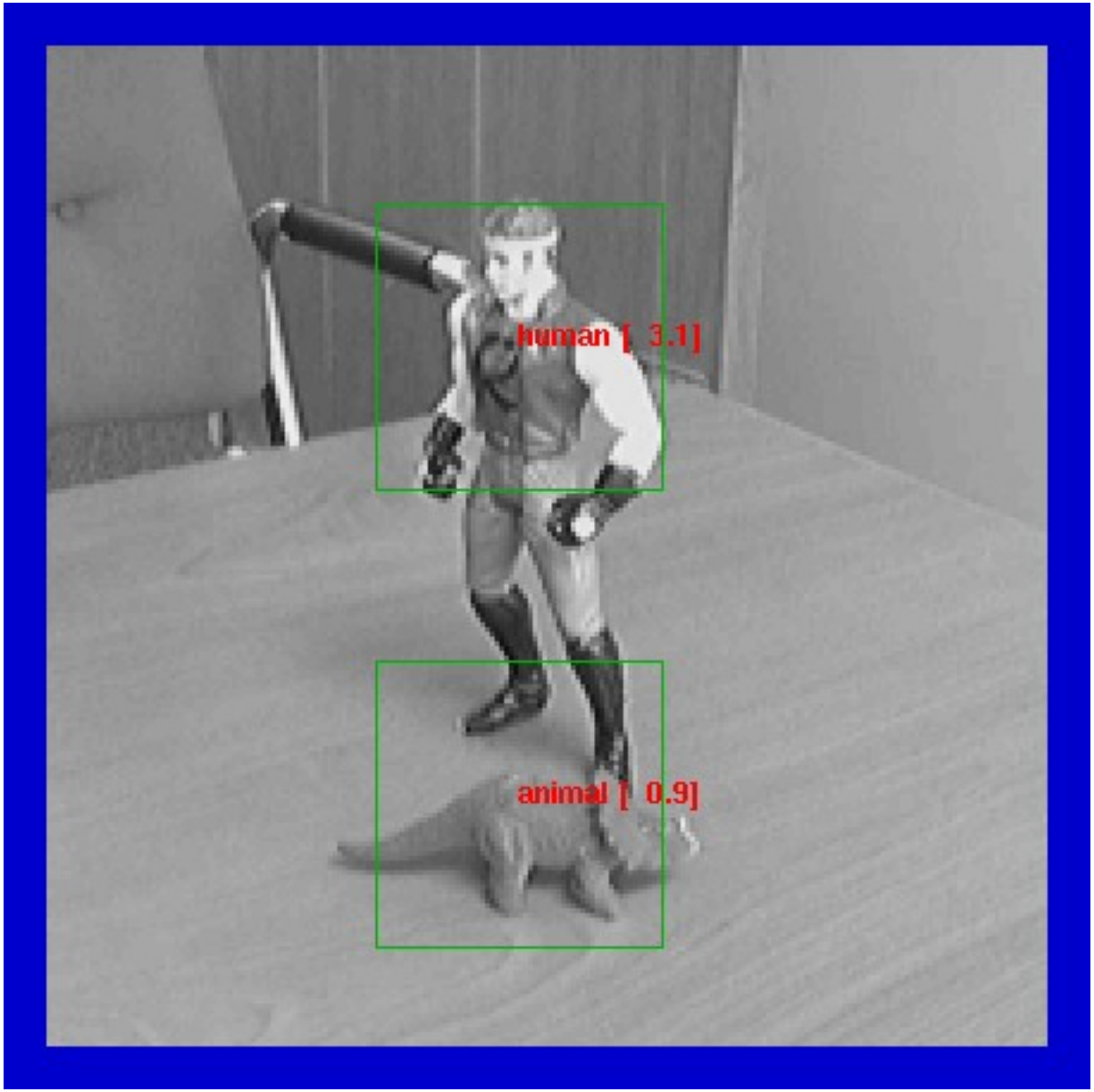
Examples (Monocular Mode)



Examples (Monocular Mode)



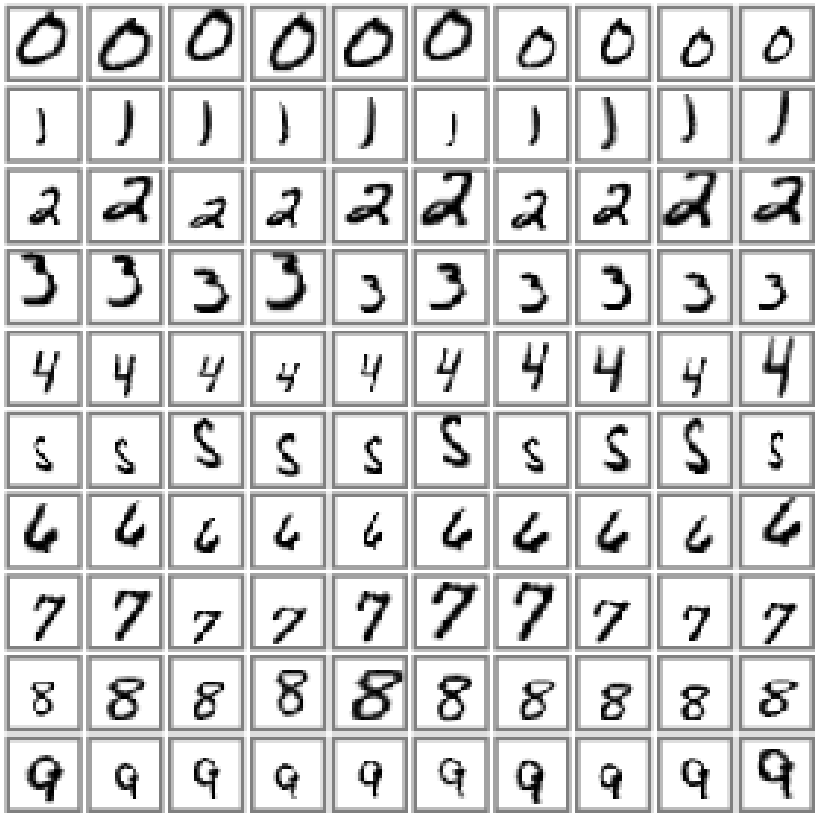
Examples (Monocular Mode)



Supervised Learning in "Deep" Architectures

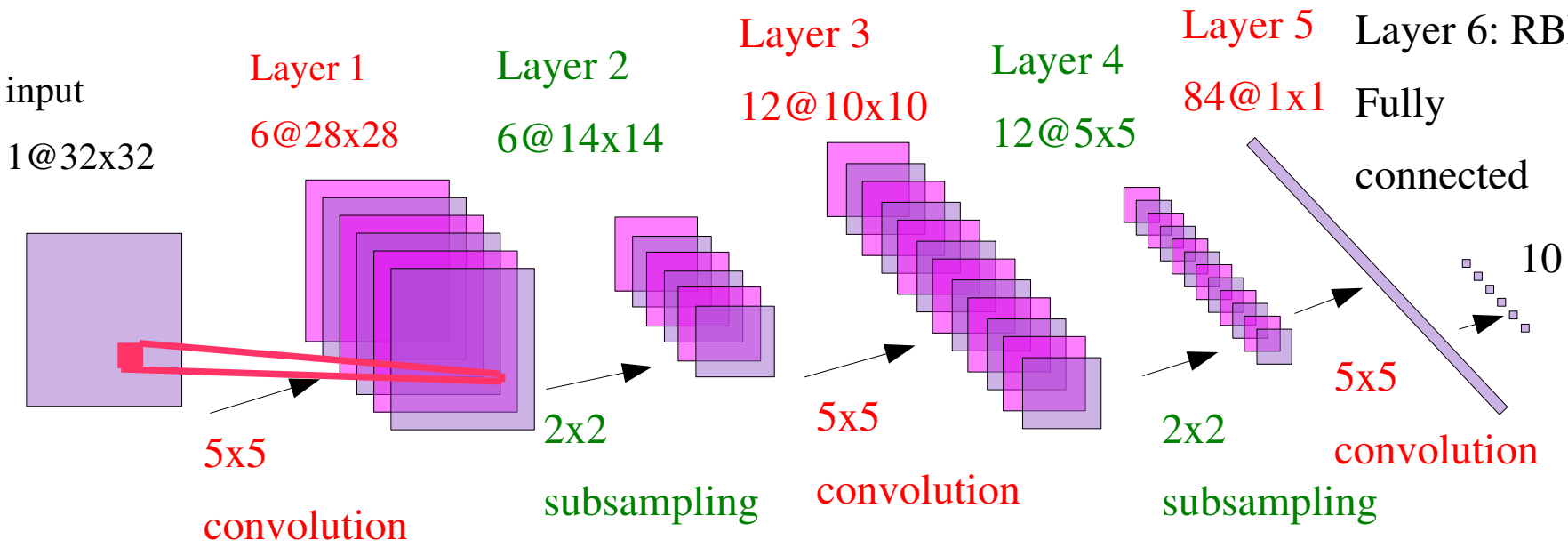
- **Backprop can train "deep" architectures reasonably well**
  - It works better if the architecture has some structure (e.g. A convolutional net)
- **Deep architectures with some structure (e.g. Convolutional nets) beat shallow ones (e.g. Kernel machines) on image classification tasks:**
  - Handwriting recognition
  - Face detection
  - Generic object recognition
- **Deep architectures are inherently more efficient for representing complex functions.**
- **Have we solved the problem of training deep architectures?**
  - Can we do backprop with lots of layers?
  - Can we train deep belief networks?
- **NO!**

MNIST Dataset



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

Handwritten Digit Recognition with a Convolutional Network



- **60,000 free parameters, 400,000 connections.**
- The architecture alternates convolutional layers (feature detectors) and subsampling layers (local feature pooling for invariance to small distortions).
- Handwritten Digit Dataset MNIST: 60,000 training samples, 10,000 test samples
- **The entire network is trained end-to-end** (all the layers are trained simultaneously).
- Test Error Rate: 0.8%

Results on MNIST Handwritten Digits (P=60,000)

	CLASSIFIER	DEFORMATION	PREPROCESSING	ERROR	Reference
	linear classifier (1-layer NN)		none	12.00	LeCun et al. 1998
	linear classifier (1-layer NN)		deskewing	8.40	LeCun et al. 1998
	pairwise linear classifier		deskewing	7.60	LeCun et al. 1998
	K-nearest-neighbors, (L2)		none	3.09	K. Wilder, U. Chicago
	K-nearest-neighbors, (L2)		deskewing	2.40	LeCun et al. 1998
	K-nearest-neighbors, (L2)		deskew, clean, blur	1.80	K. Wilder, U. Chicago
Best	K-NN L3, 2 pixel jitter		deskew, clean, blur	1.22	K. Wilder, U. Chicago
Hand-crafted	K-NN, shape context matching		shape context feature	0.63	Belongie PAMI 02
	40 PCA + quadratic classifier		none	3.30	LeCun et al. 1998
	1000 RBF + linear classifier		none	3.60	LeCun et al. 1998
	K-NN, Tangent Distance		subsamp 16x16 pixels	1.10	LeCun et al. 1998
	SVM, Gaussian Kernel		none	1.40	Many
	SVM deg 4 polynomial		deskewing	1.10	Cortes/Vapnik
	Reduced Set SVM deg 5 poly		deskewing	1.00	Scholkopf
	Virtual SVM deg-9 poly	Affine	none	0.80	Scholkopf
Best	V-SVM, 2-pixel jittered		none	0.68	DeCoste/Scholkopf, MLJ'02
Kernel-based	V-SVM, 2-pixel jittered		deskewing	0.56	DeCoste/Scholkopf, MLJ'02
	2-layer NN, 300 HU, MSE		none	4.70	LeCun et al. 1998
	2-layer NN, 300 HU, MSE,	Affine	none	3.60	LeCun et al. 1998
	2-layer NN, 300 HU		deskewing	1.60	LeCun et al. 1998
	3-layer NN, 500+150 HU		none	2.95	LeCun et al. 1998
Best fully-c	3-layer NN, 500+150 HU	Affine	none	2.45	LeCun et al. 1998
Neural Net	3-layer NN, 500+300 HU, CE, reg		none	1.53	Hinton, in press, 2005
	2-layer NN, 800 HU, CE		none	1.60	Simard et al., ICDAR 2003
	2-layer NN, 800 HU, CE	Affine	none	1.10	Simard et al., ICDAR 2003
	2-layer NN, 800 HU, MSE	Elastic	none	0.90	Simard et al., ICDAR 2003
Best know-	2-layer NN, 800 HU, CE	Elastic	none	0.70	Simard et al., ICDAR 2003
Ledge-free	Stacked RBM + backprop		none	0.95	Hinton, in press, 2005
	Convolutional net LeNet-1		subsamp 16x16 pixels	1.70	LeCun et al. 1998
	Convolutional net LeNet-4		none	1.10	LeCun et al. 1998
	Convolutional net LeNet-5,		none	0.95	LeCun et al. 1998
	Convolutional net LeNet-5,	Affine	none	0.80	LeCun et al. 1998
	Boosted LeNet-4	Affine	none	0.70	LeCun et al. 1998
	Convolutional net, CE	Affine	none	0.60	Simard et al., ICDAR 2003
Best overall	Convolutional net, CE	Elastic	none	0.40	Simard et al., ICDAR 2003

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
Convolutional nets			
Convolutional net LeNet-5,		0.80	LeCun 2005 Unpublished
Convolutional net LeNet-6,		0.70	LeCun 2006 Unpublished
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 et 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

Convolutional Nets are the best known method for handwriting recognition

Problems with Supervised Learning in Deep Architectures

- **vanishing gradient, symmetry breaking**
  - The first layers have a hard time learning useful things
  - How to break the symmetry so that different units do different things
- **Idea [Hinton]:**
  - 1 – Initialize the first (few) layers with unsupervised training
  - 2 – Refine the whole network with backprop
- **Problem: How do we train a layer in unsupervised mode?**
  - Auto-encoder: only works when the first layer is smaller than the input
  - What if the first layer is larger than the input?
  - Reconstruction is trivial!
- **Solution: sparse over-complete representations**
  - Keep the number of bits in the first layer low
  - Hinton uses a Restricted Boltzmann Machine in which the first layer uses stochastic binary units

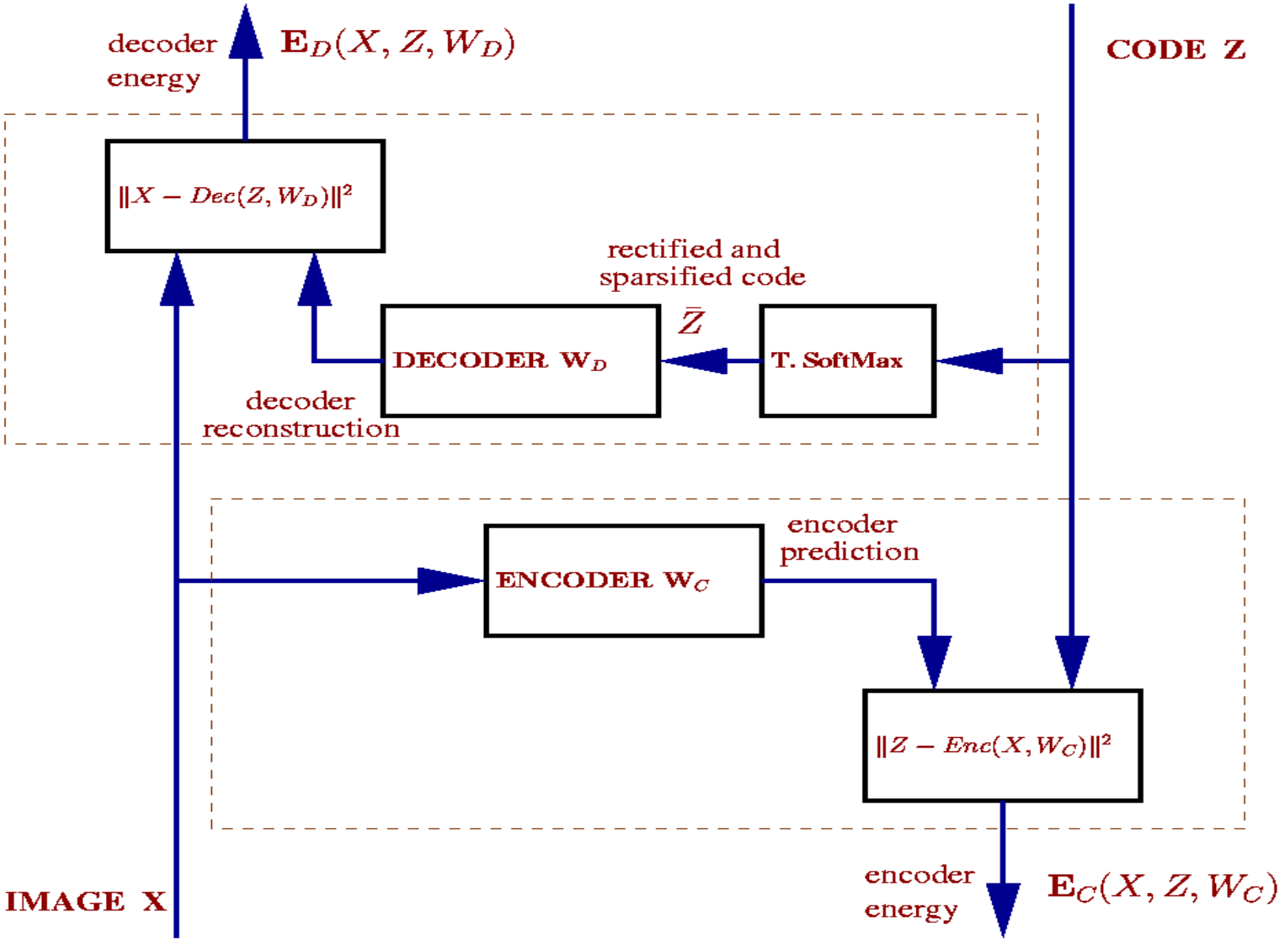
Best Results on MNIST (from raw images: no preprocessing)

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, in press, 2005
Convolutional nets			
Convolutional net LeNet-5,		0.80	LeCun 2005 Unpublished
Convolutional net LeNet-6,		0.70	LeCun 2006 Unpublished
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 et 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

Unsupervised Learning of Sparse Over-Complete Features

- **Classification is easier with over-complete feature sets**
- **Existing Unsupervised Feature Learning (non sparse/overcomplete):**
  - PCA, ICA, Auto-Encoder, Kernel-PCA
- **Sparse/Overcomplete Methods**
  - Non-Negative Matrix Factorization
  - Sparse-Overcomplete basis functions (Olshausen and Field 1997)
  - Product of Experts (Teh, Welling, Osindero, Hinton 2003)

Symmetric Product of Experts



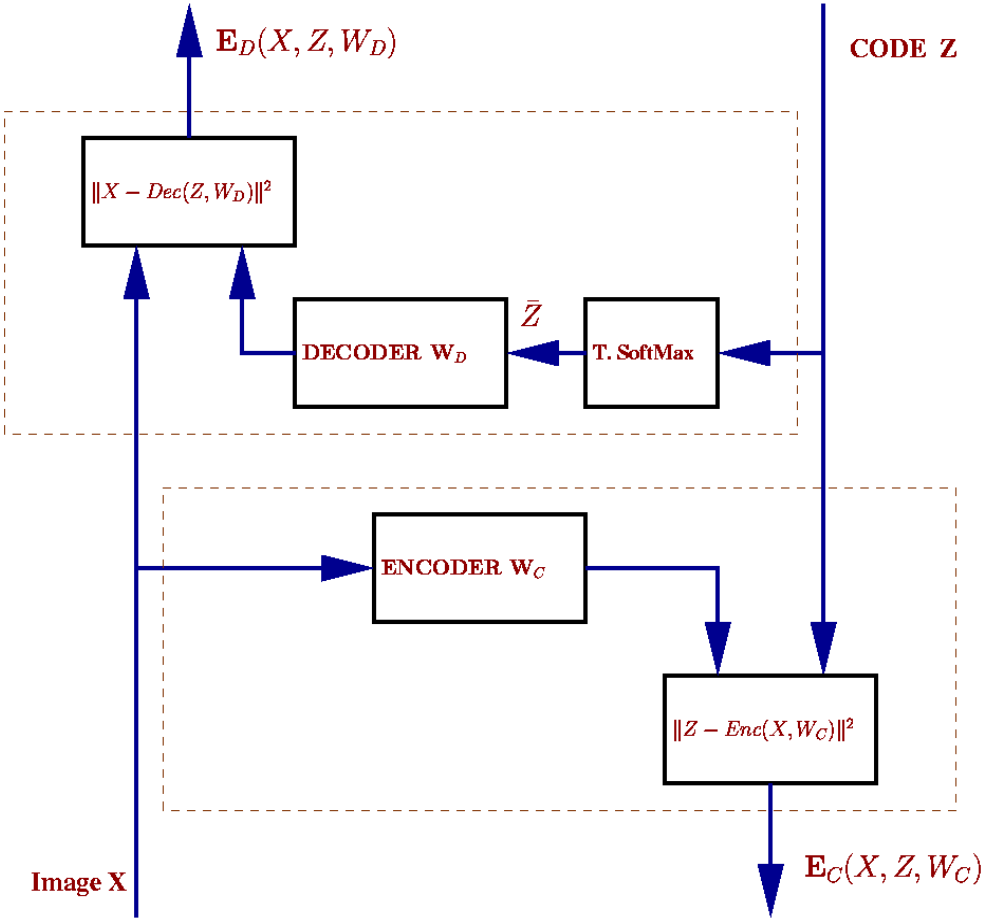
Symmetric Product of Experts

$$P(Z|X, W_c, W_d) \propto \exp(-\beta E(X, Z, W_c, W_d))$$

$$E(X, Z, W_c, W_d) = E_C(X, Z, W_c) + E_D(X, Z, W_d)$$

$$E_C(X, Z, W_c) = \frac{1}{2}\left\|Z - W_c X\right\|^2 = \frac{1}{2}\sum (z_i - W_c^i X)^2$$

$$E_D(X, Z, W_d) = \frac{1}{2}\left\|X - W_d \bar{Z}\right\|^2 = \frac{1}{2}\sum (x_i - W_d^i \bar{Z})^2$$



Inference & Learning

- *Inference*

$$\tilde{Z} = argmin_Z \; E(X, Z, W) = argmin_Z \left[E_C(X, Z, W) + E_D(X, Z, W) \right]$$

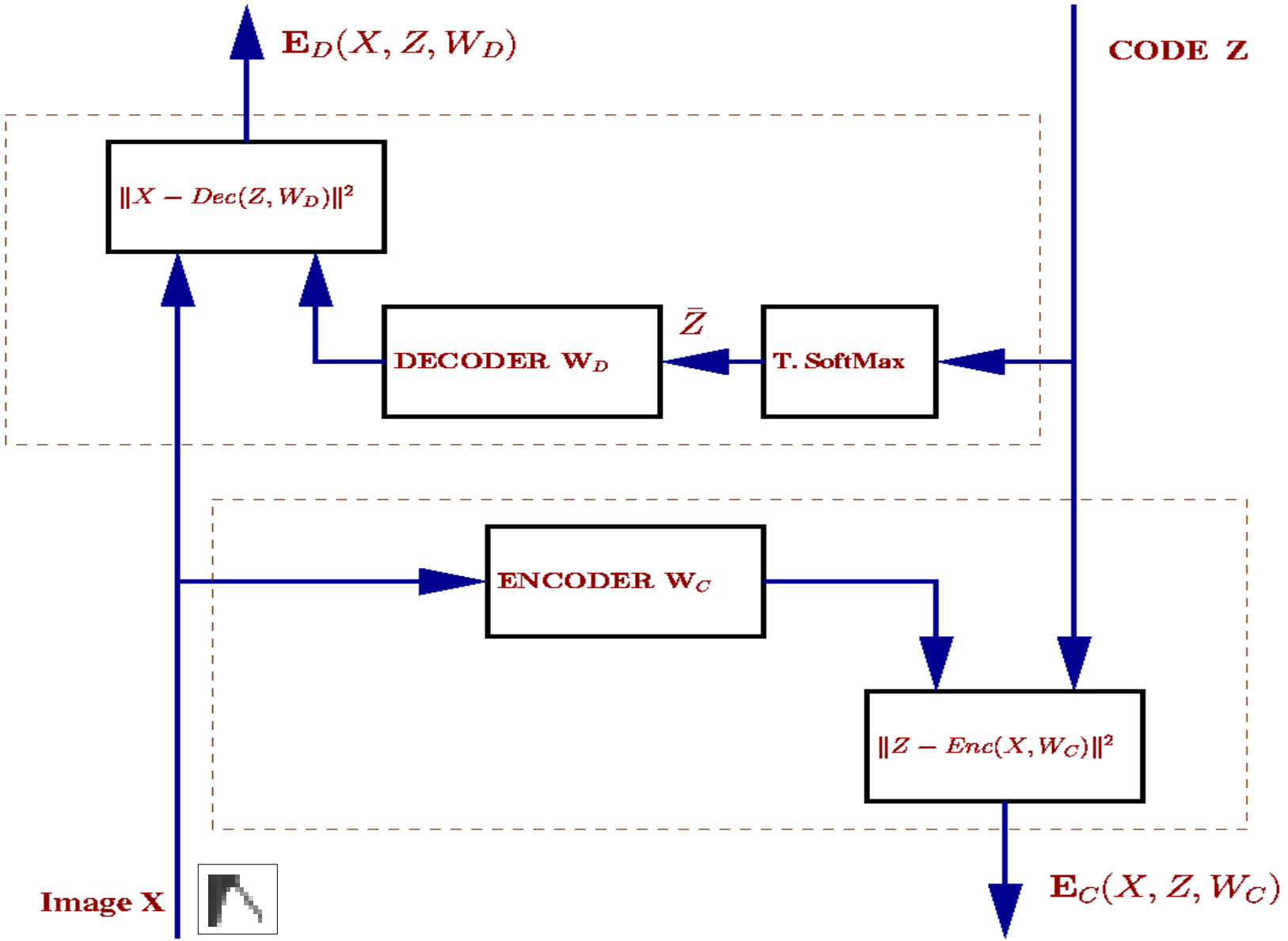
  - let Z(0) be the encoder prediction
  - find code which minimizes total energy
  - gradient descent optimization

- *Learning*

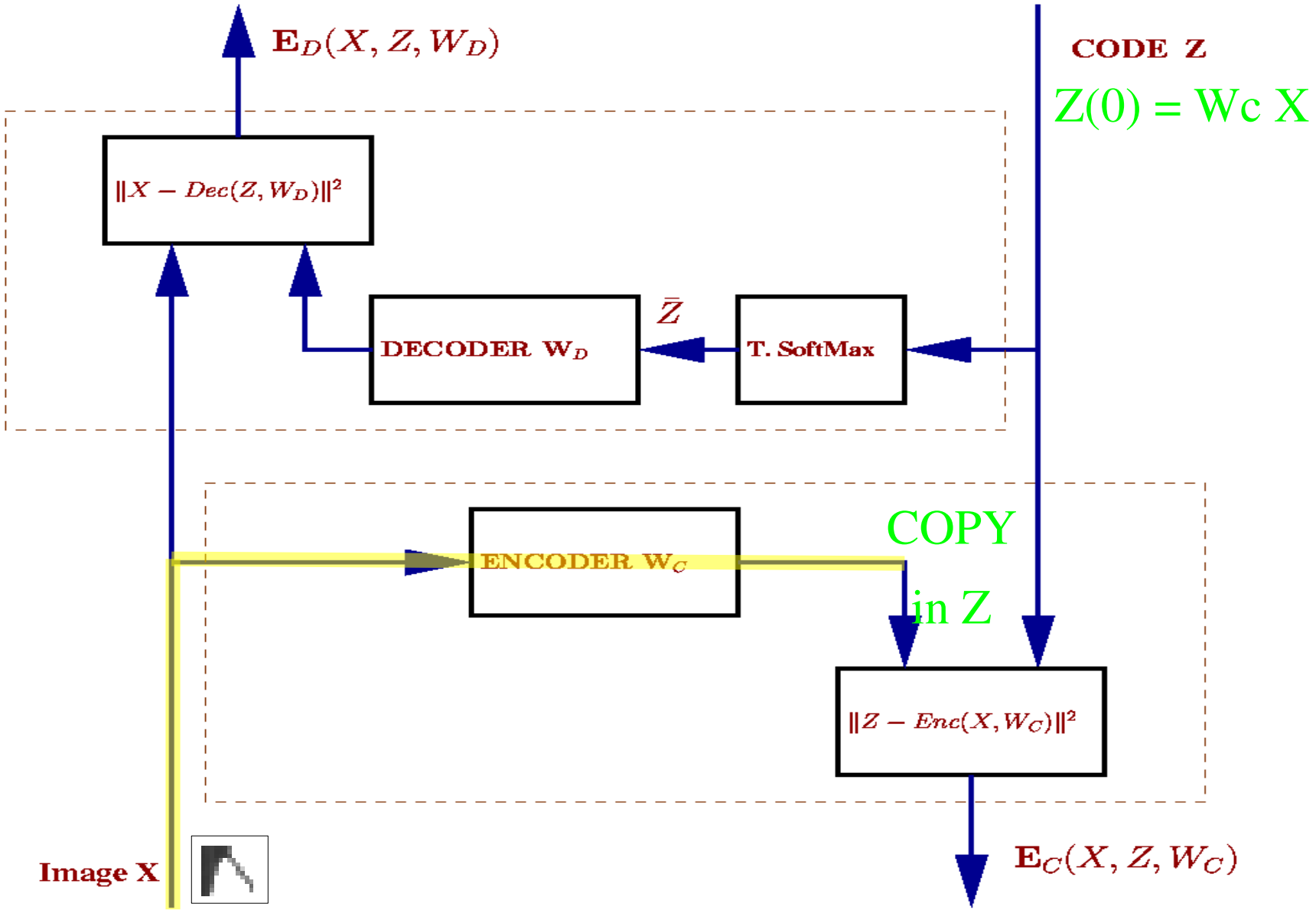
$$W \leftarrow W - \partial E(X, \tilde{Z}, W) / \partial W$$

  - using the optimal code, minimize E w.r.t. the weights W
  - gradient descent optimization

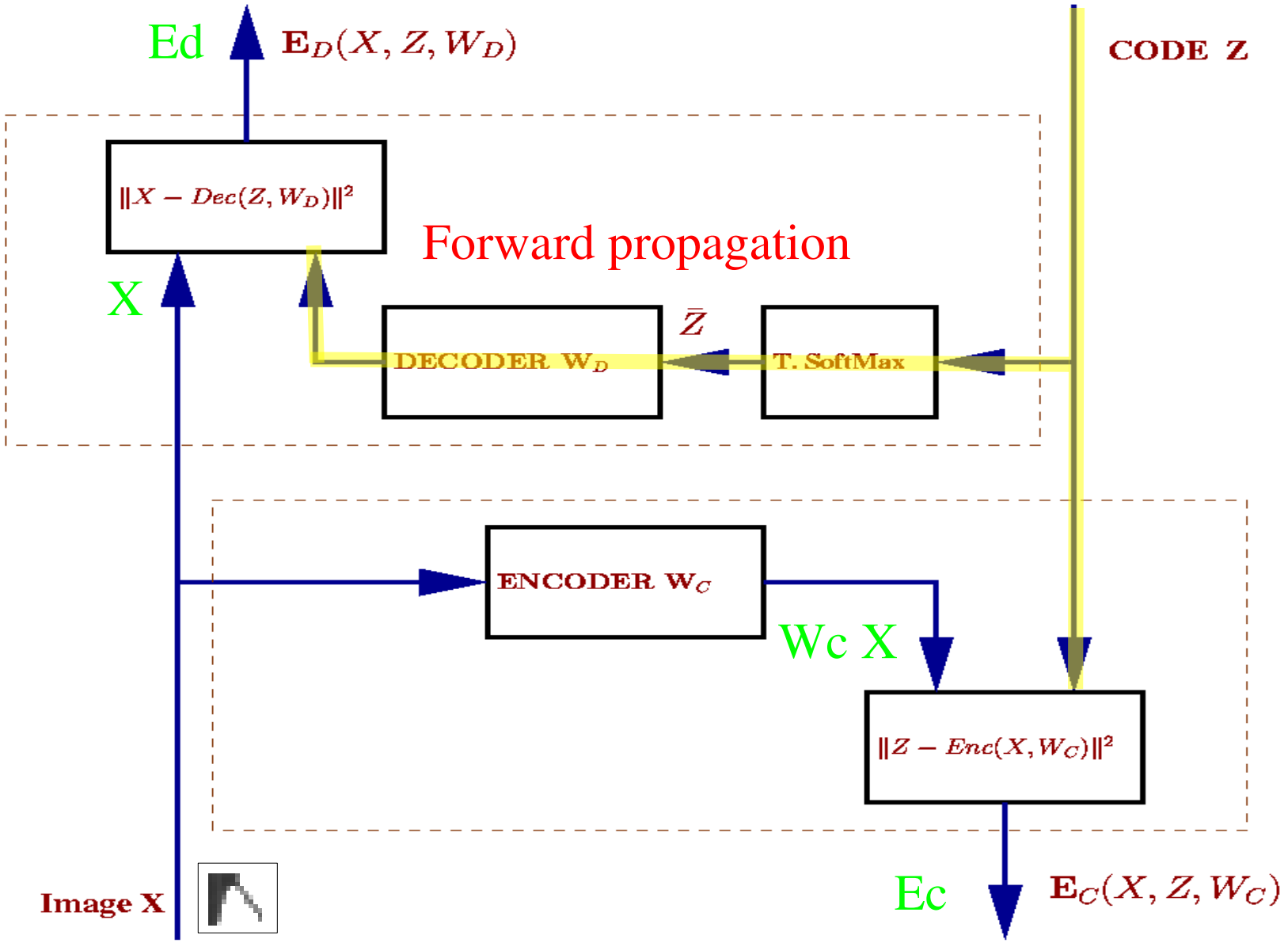
Inference & Learning



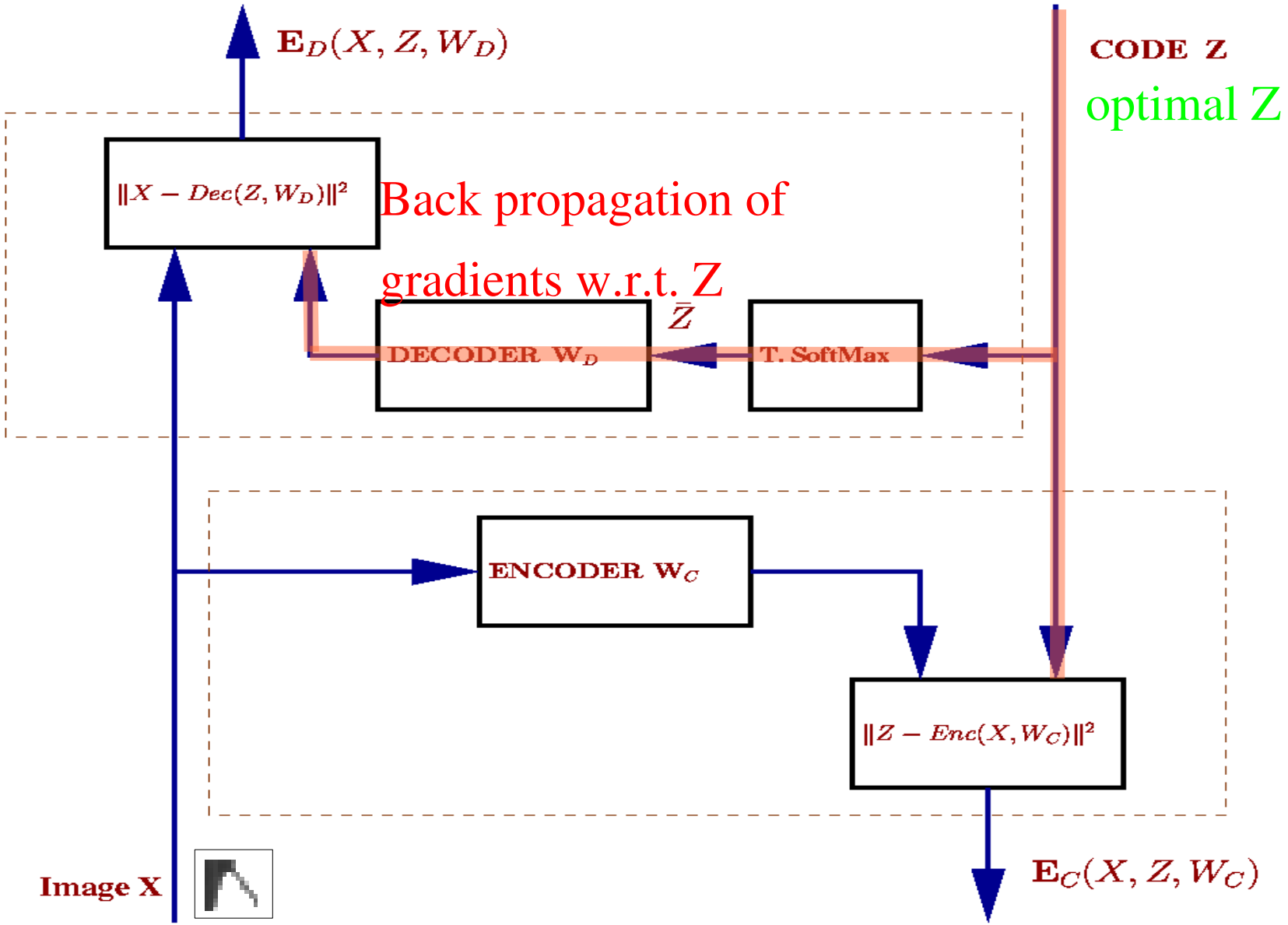
Inference - step 1



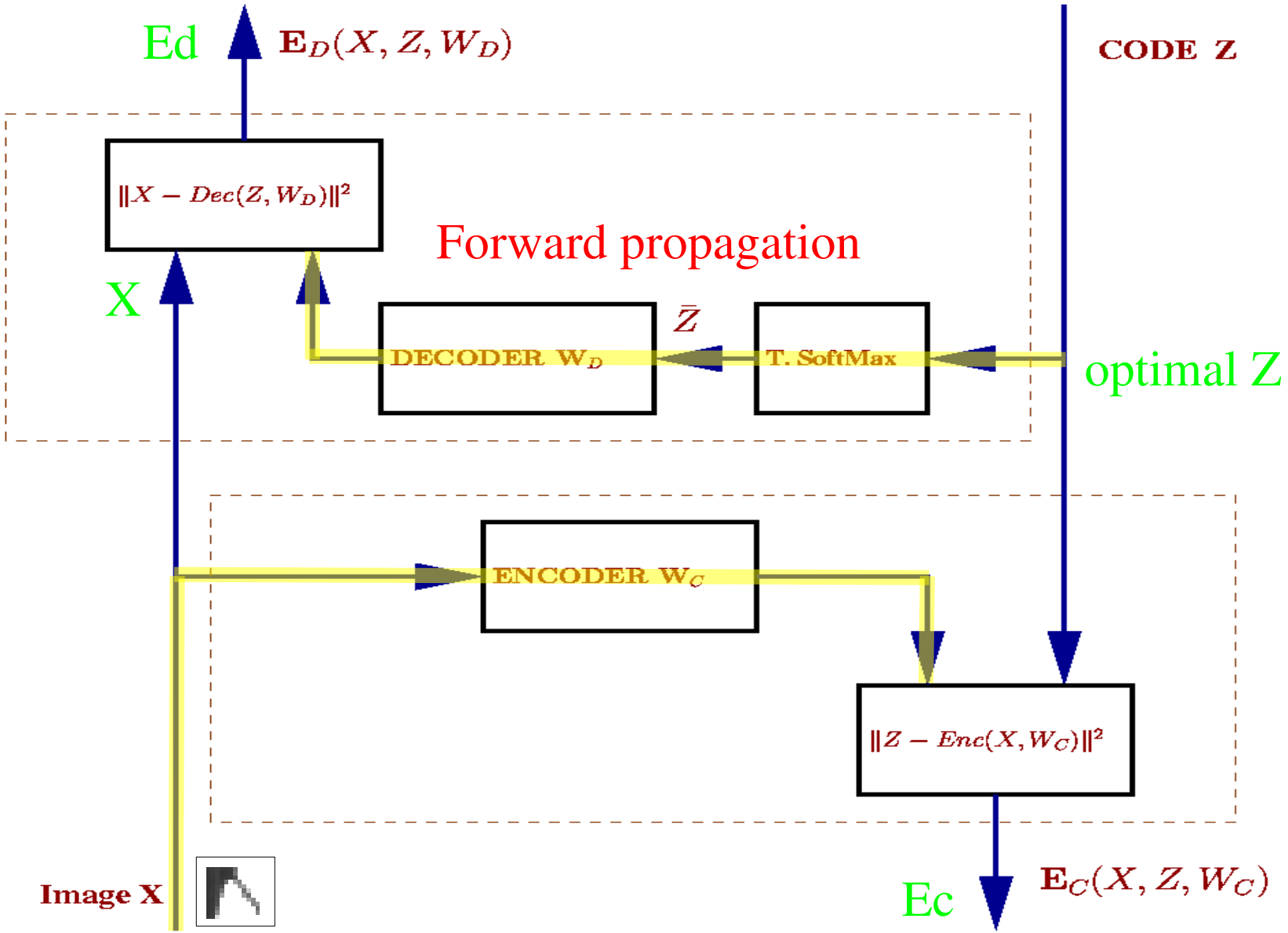
Inference - step 1



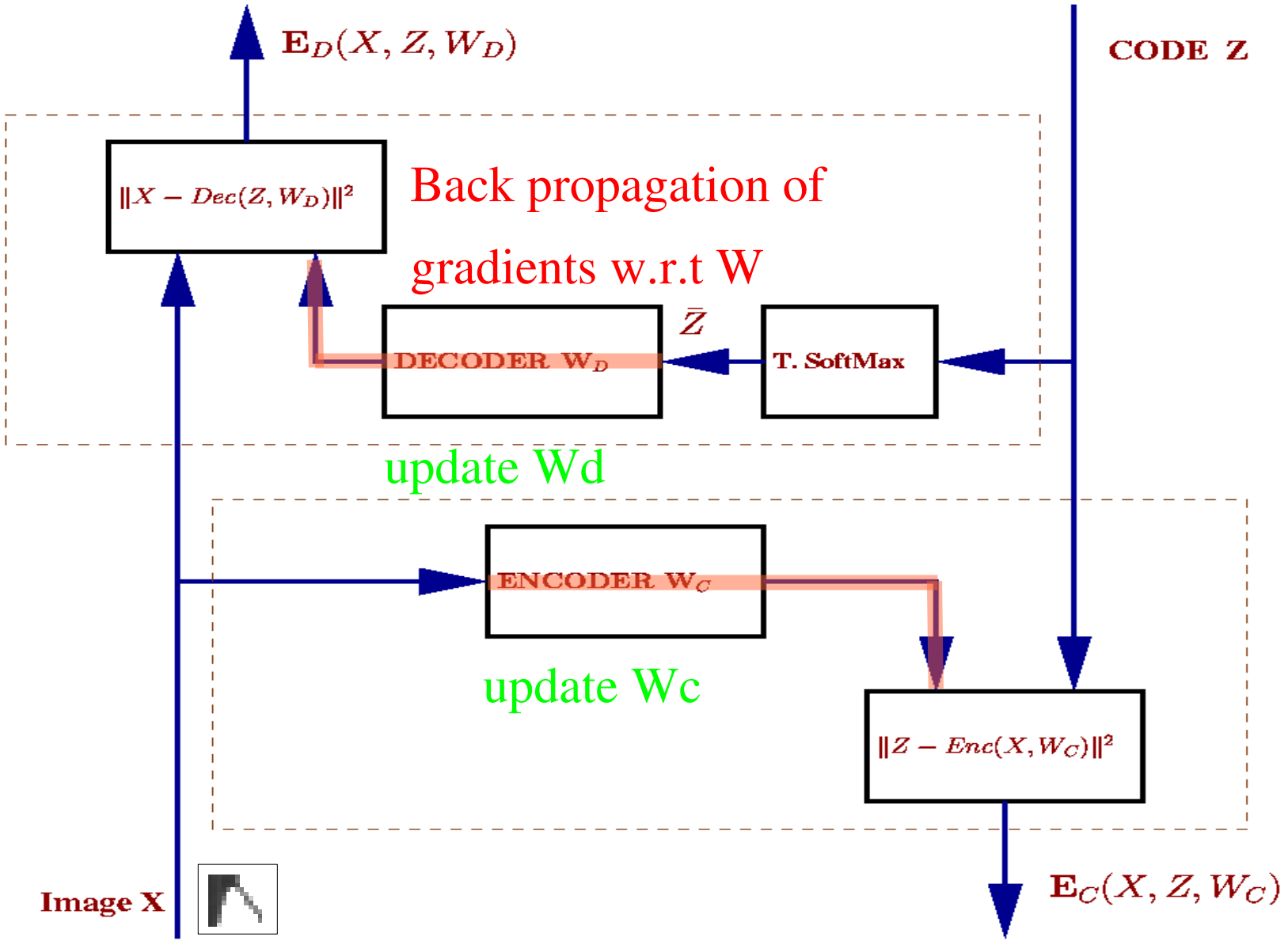
Inference - step 1



Learning - step 2



Learning - step 2



Sparsifying Logistic

$$\bar{z}_i(t) = \eta e^{\beta z_i(t)} / \xi_i(t), \quad i \in [1..m]$$

$$\xi_i(t) = \eta e^{\beta z_i(t)} + (1-\eta)\xi_i(t-1)$$

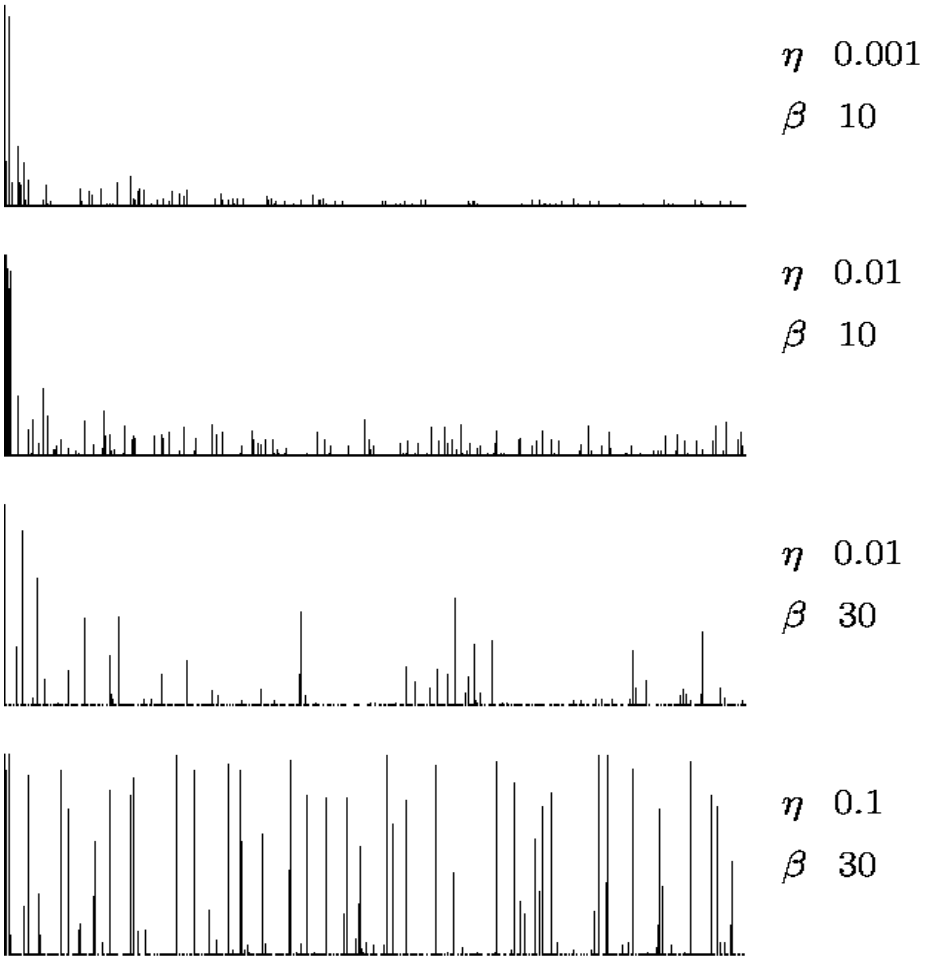
- temporal vs. spatial sparsity

  => no normalization

- ξ is treated as a learned parameter

  => TSM is a sigmoid function with a special bias $\bar{z}_i(t) = \frac{1}{1 + B e^{-\beta z_i(t)}}$

- ξ is saturated during training to allow units to have different sparseness



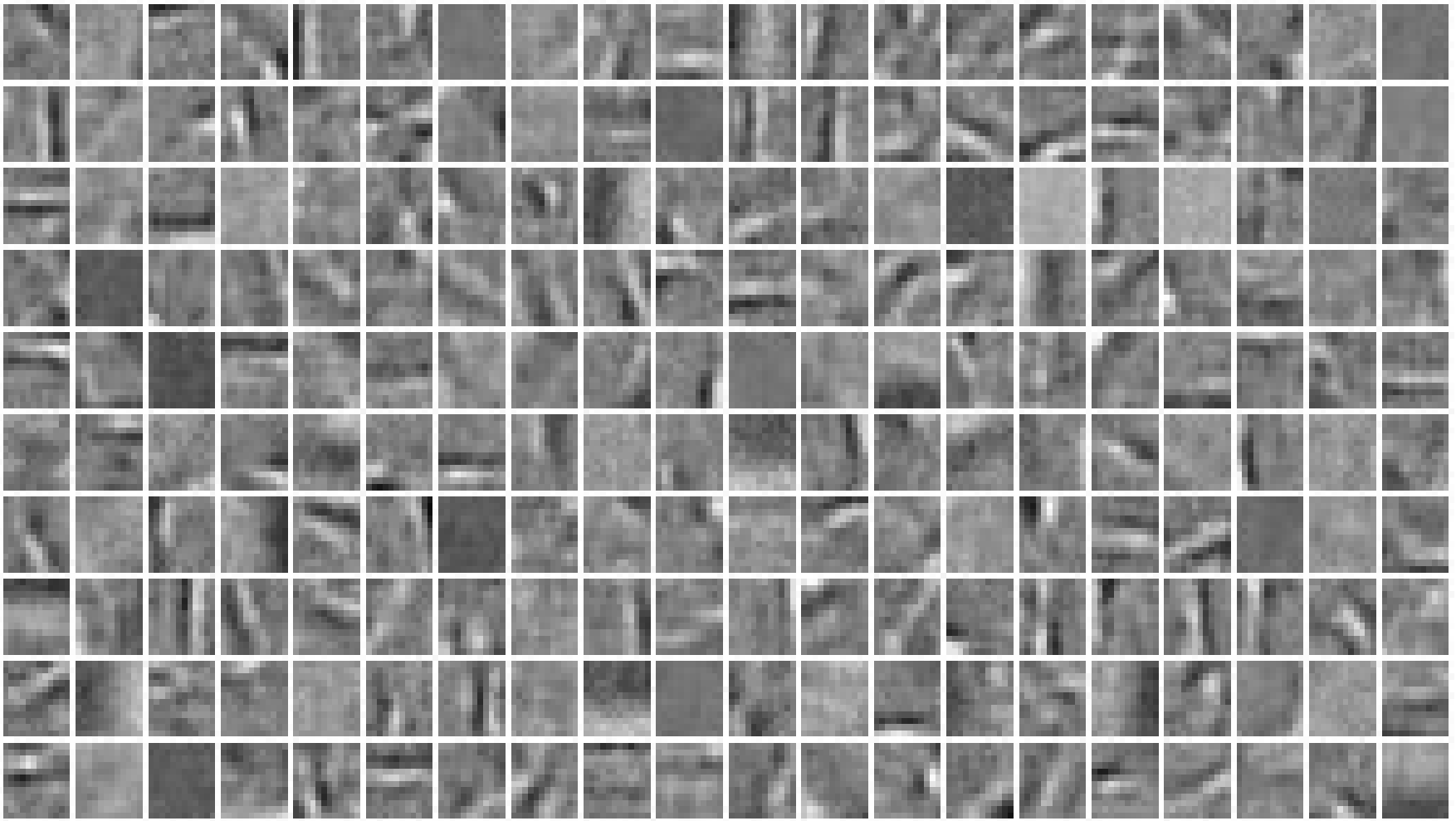
input uniformly distributed in [-1,1]

Natural image patches - Berkeley

Berkeley data set

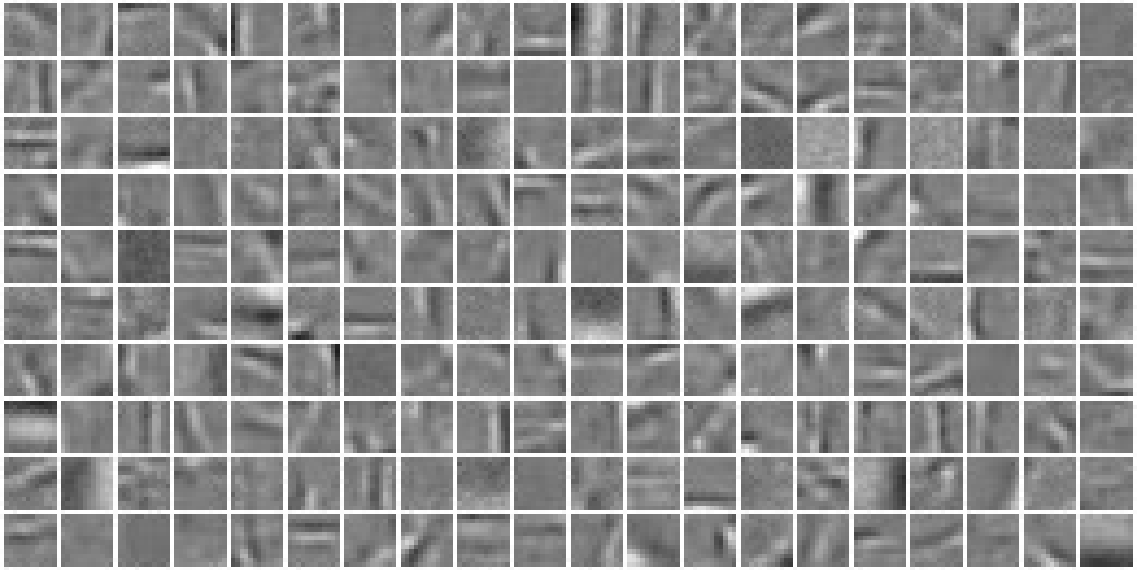
- 100,000 12x12 patches
- 200 units in the code
- η β 0.02
- 1
- learning rate 0.001
- L1, L2 regularizer 0.001
- fast convergence: < 30min.

Natural image patches - Berkeley



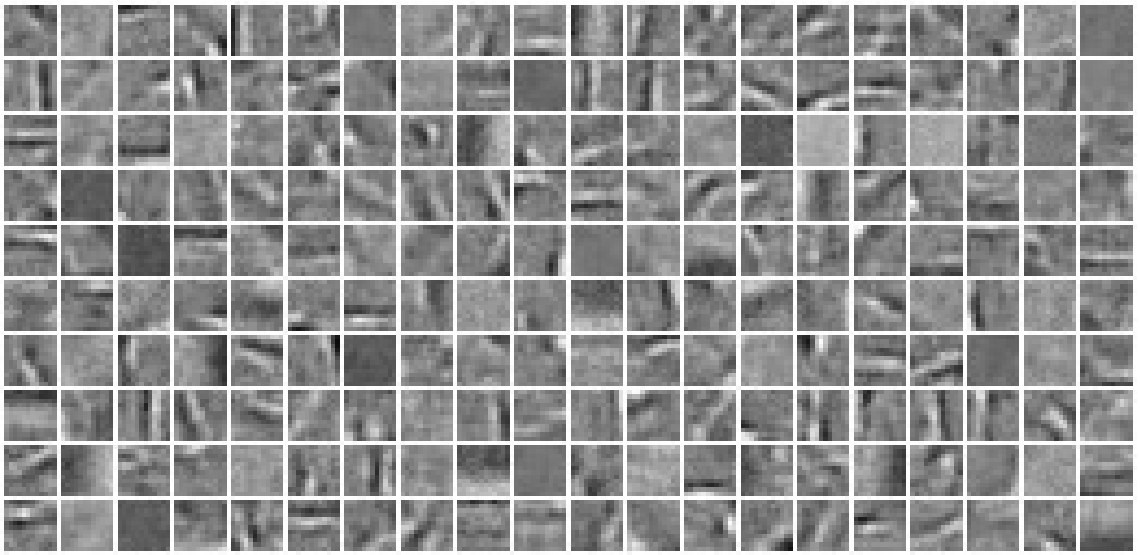
200 decoder filters (reshaped columns of matrix $\mathbf{W_d}$)

Natural image patches - Berkeley



Encoder *direct* filters

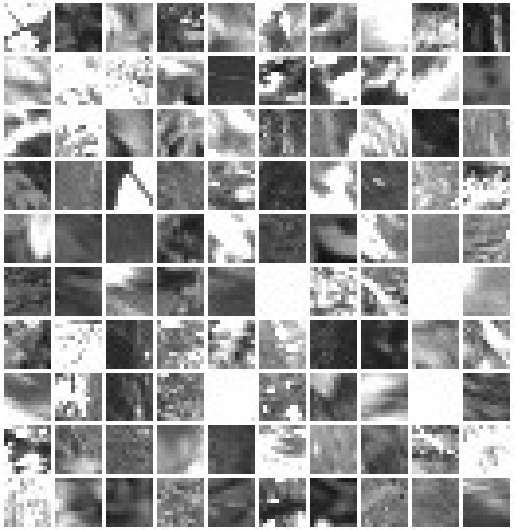
(rows of $\mathbf{W_c}$)



Decoder *reverse* filters

(cols. of $\mathbf{W_d}$)

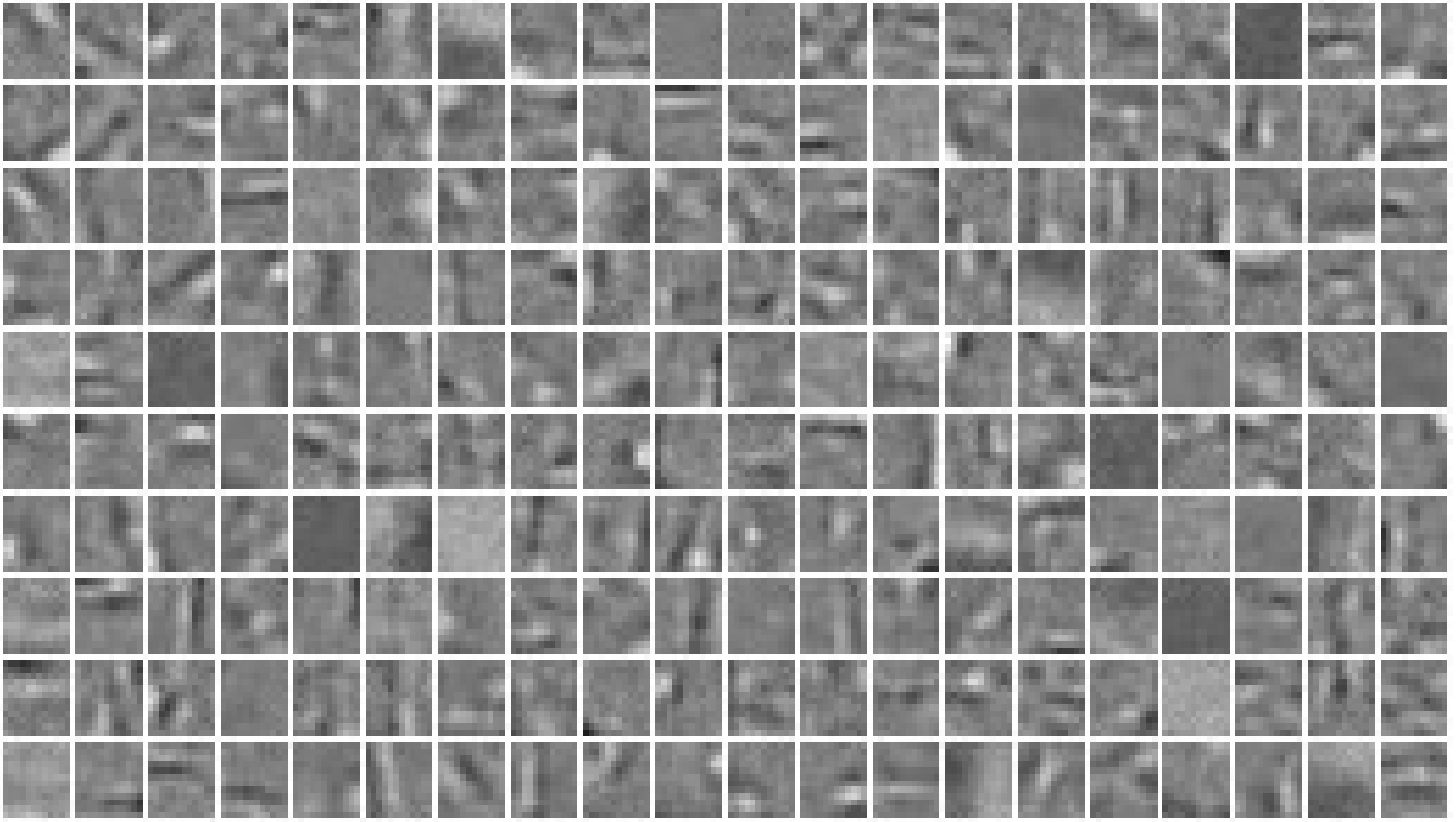
Natural image patches - Forest



Forest data set

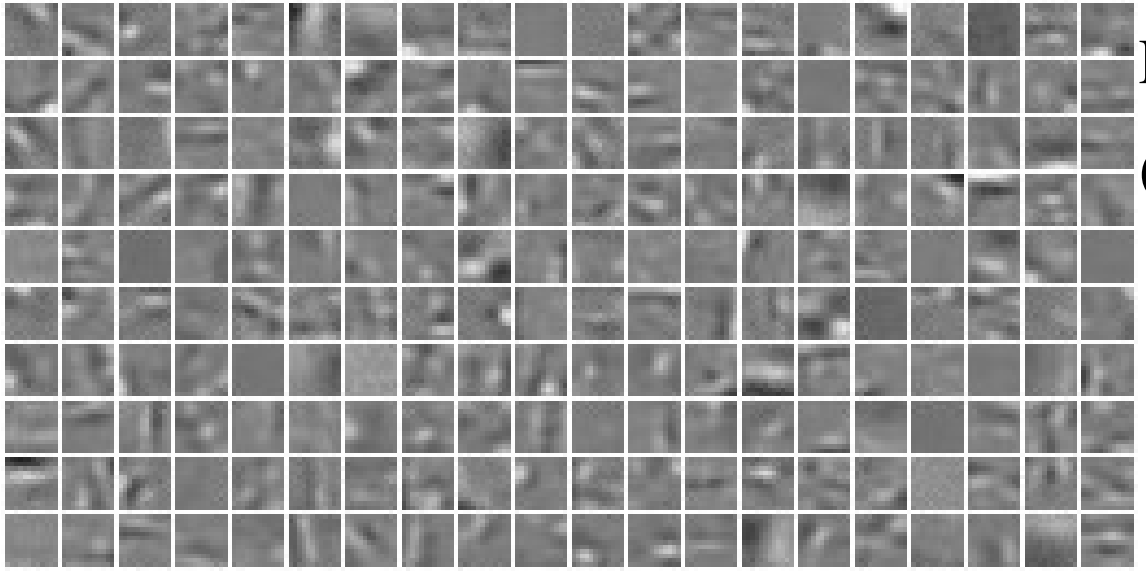
- 100,000 12x12 patches
- 200 units in the code
- η
- β 0.02
- 1
- learning rate 0.001
- L1, L2 regularizer 0.001
- fast convergence: < 30min.

Natural image patches - Forest



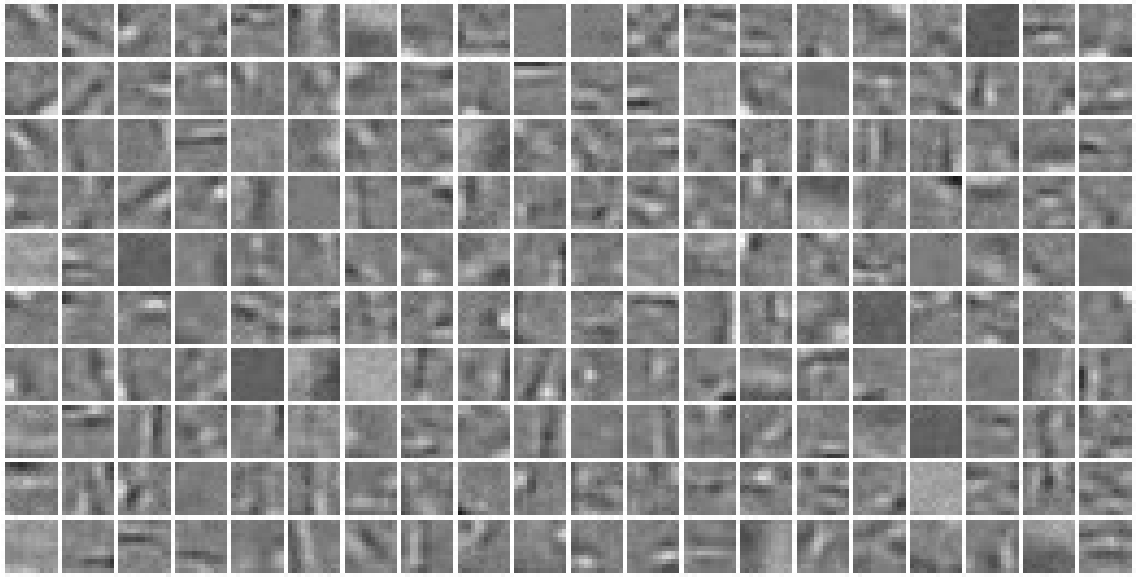
200 decoder filters (reshaped columns of matrix $\mathbf{W_d}$)

Natural image patches - Forest



Encoder *direct* filters

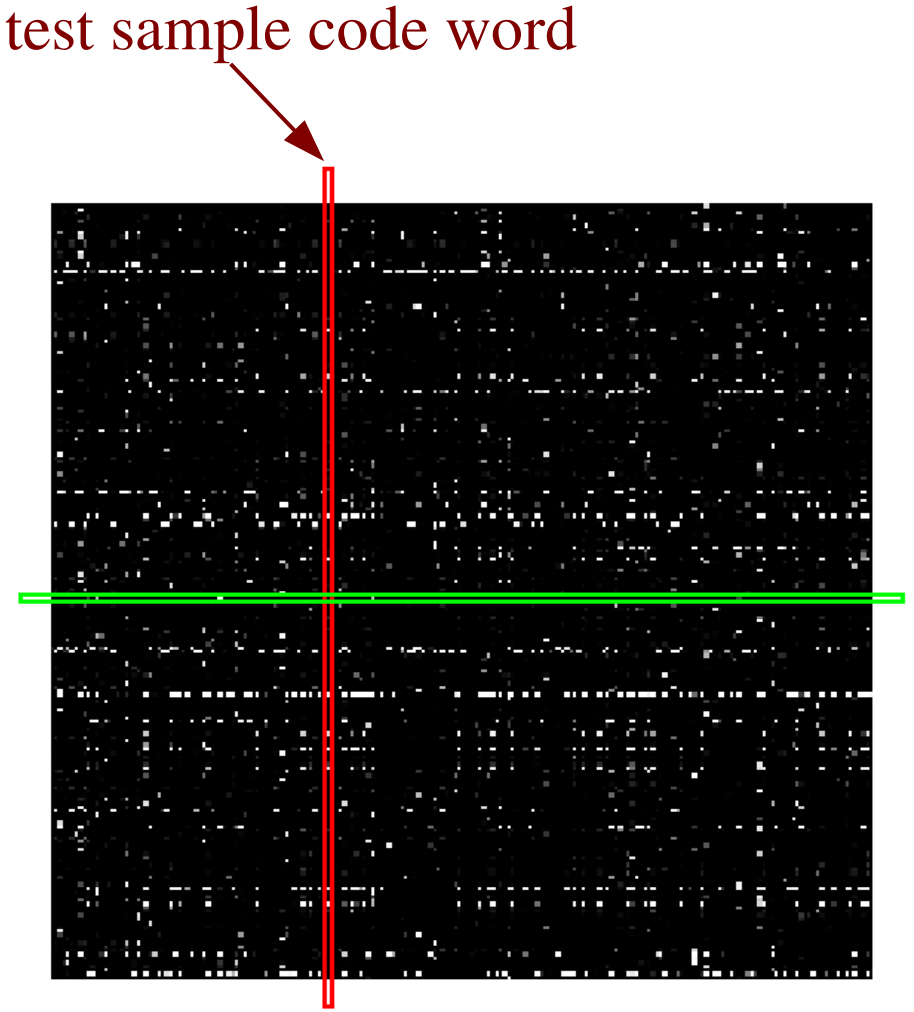
(rows of $\mathbf{W_c}$)



Decoder *reverse* filters

(cols. of $\mathbf{W_d}$)

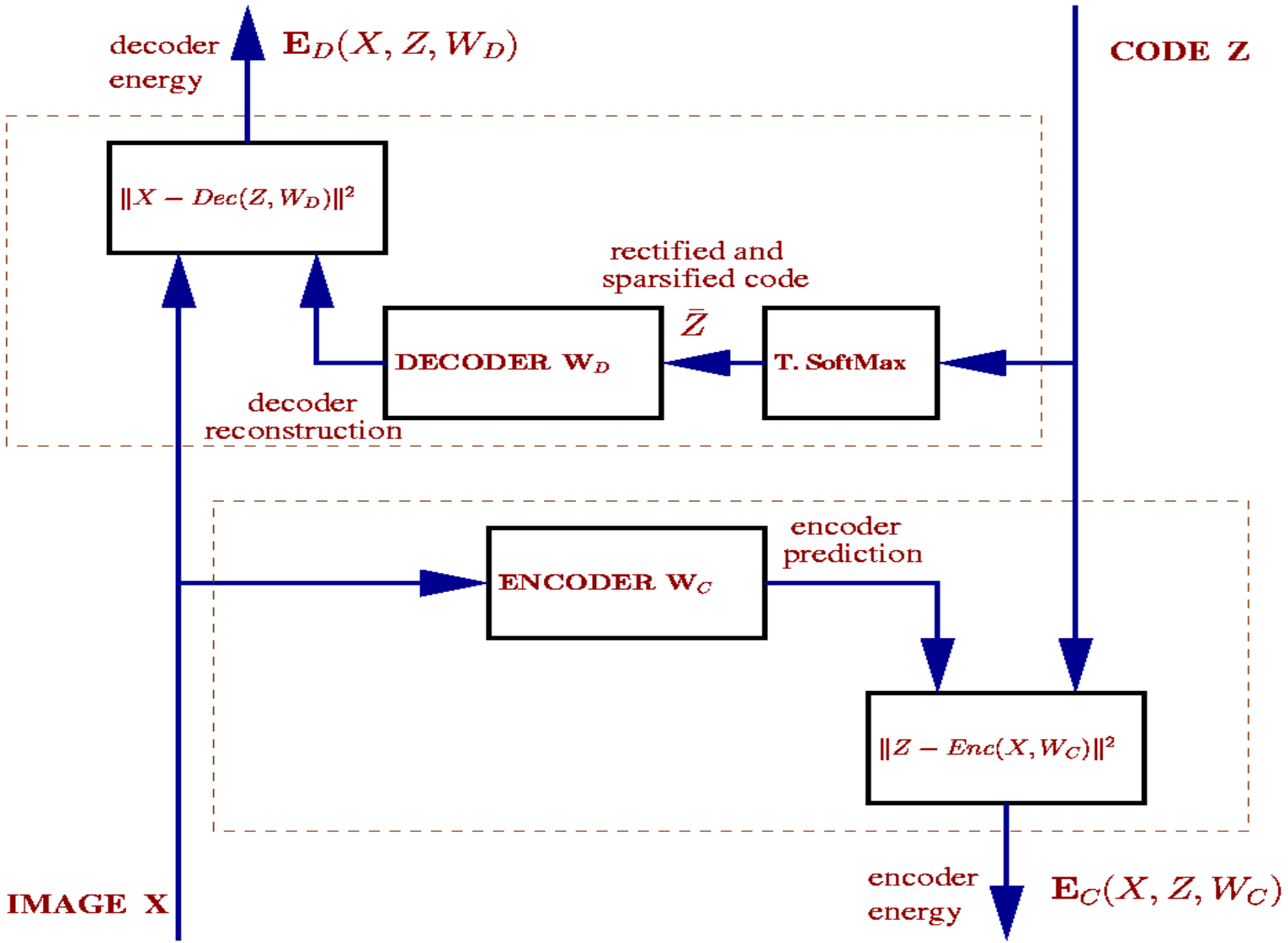
Natural image patches - Forest



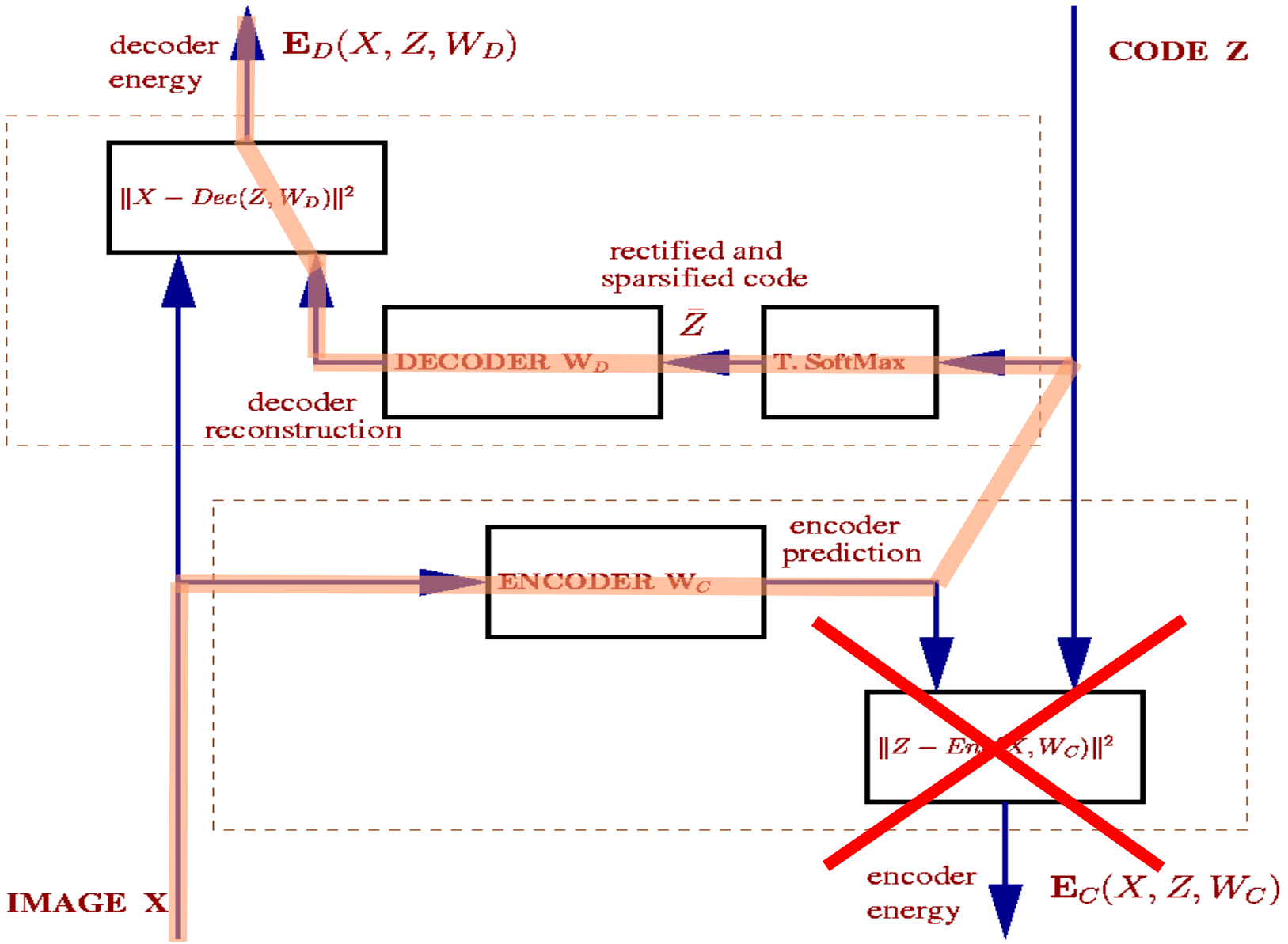
- codes are:
  - sparse
  - almost binary
  - quite decorrelated
- in testing codes are produced by propagating the input patch through encoder and TSM
- β controls sparsity
- controls the "bit content" in each code unit

code words from 200 randomly selected test patches

What about an autoencoder?

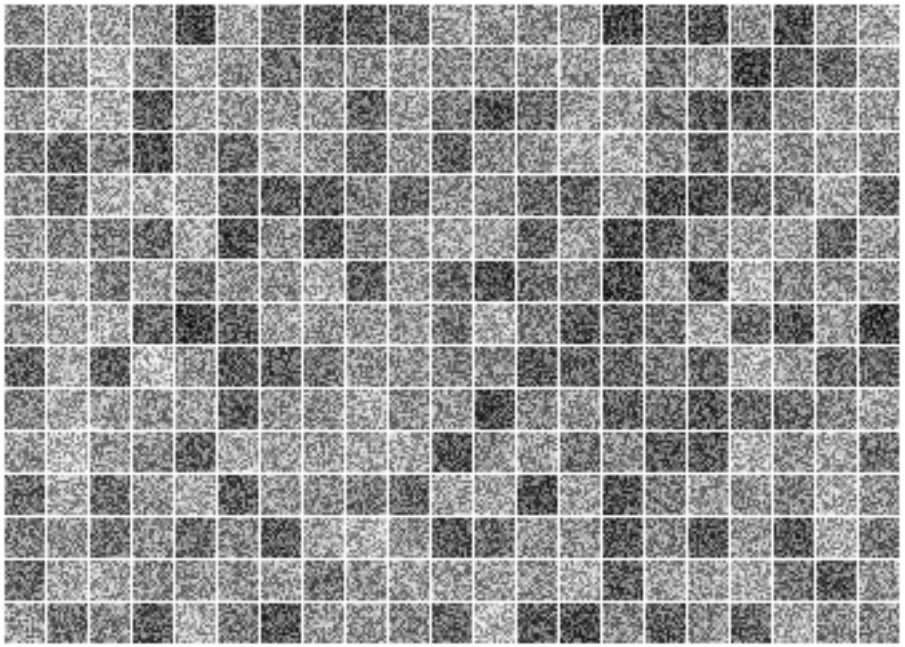


What about an autoencoder?

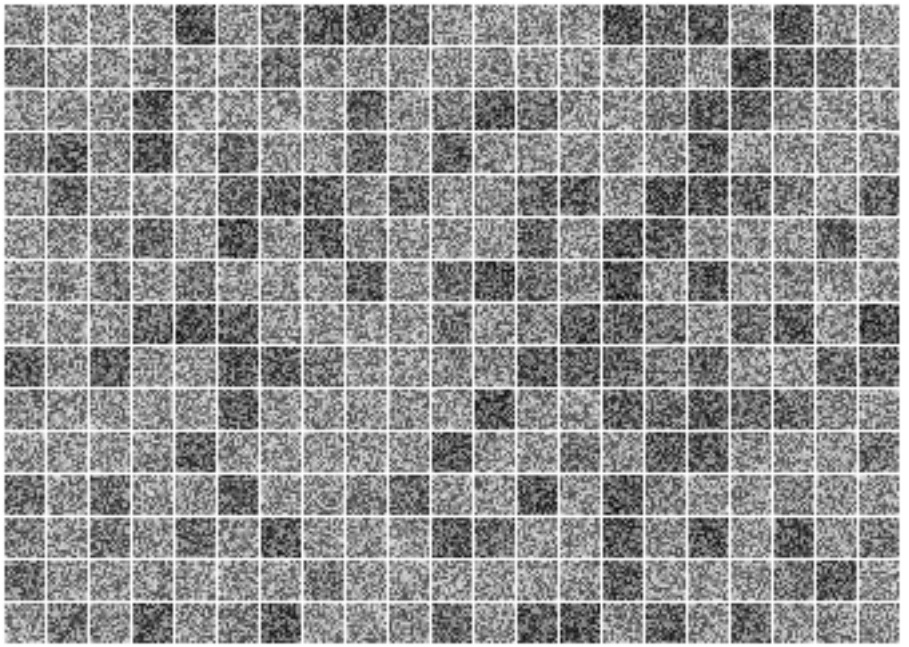


What about an autoencoder?

encoder filters



decoder filters

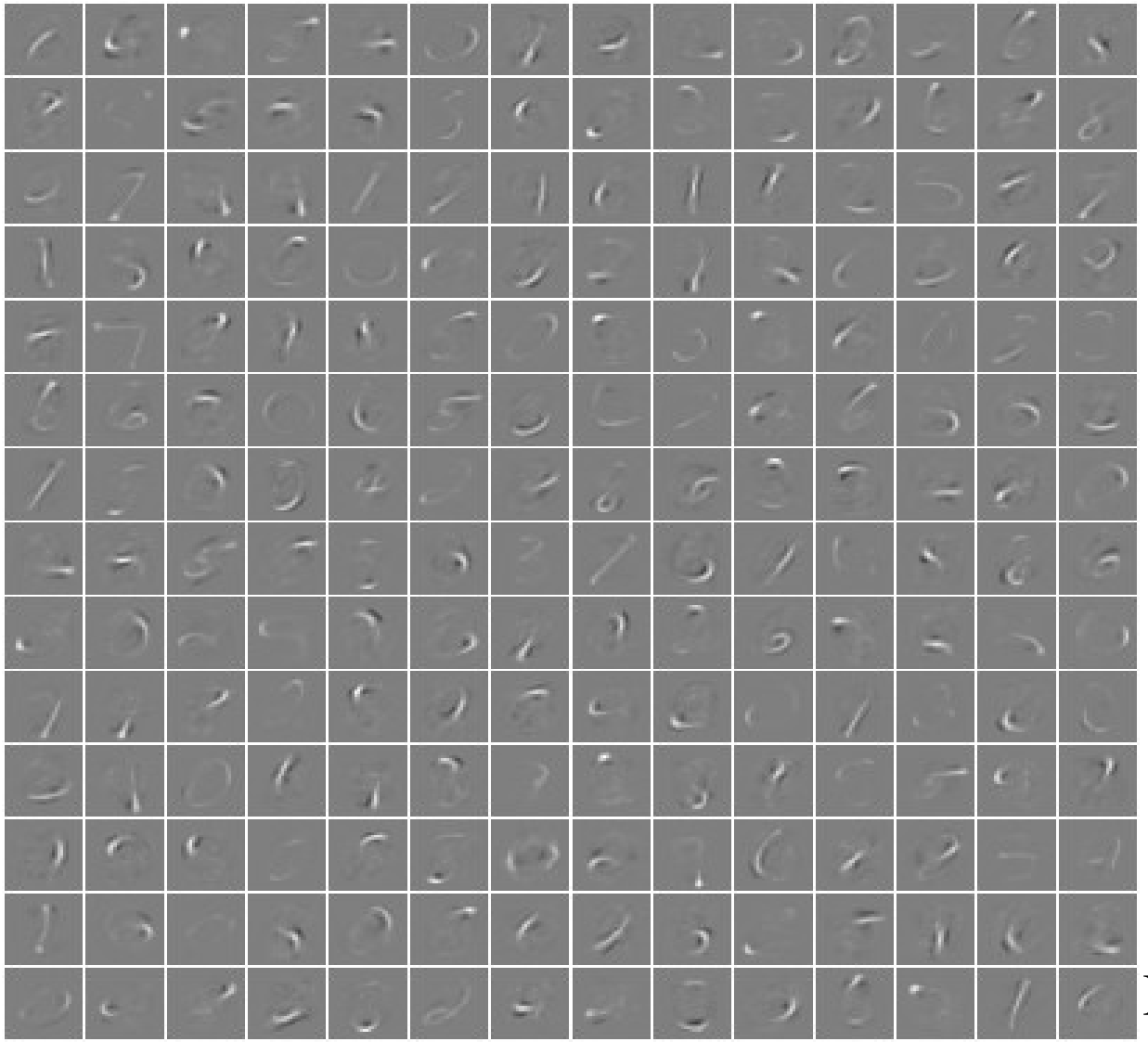


- filters are random
- convergence only for large η and small β

η 0.1

β 0.5

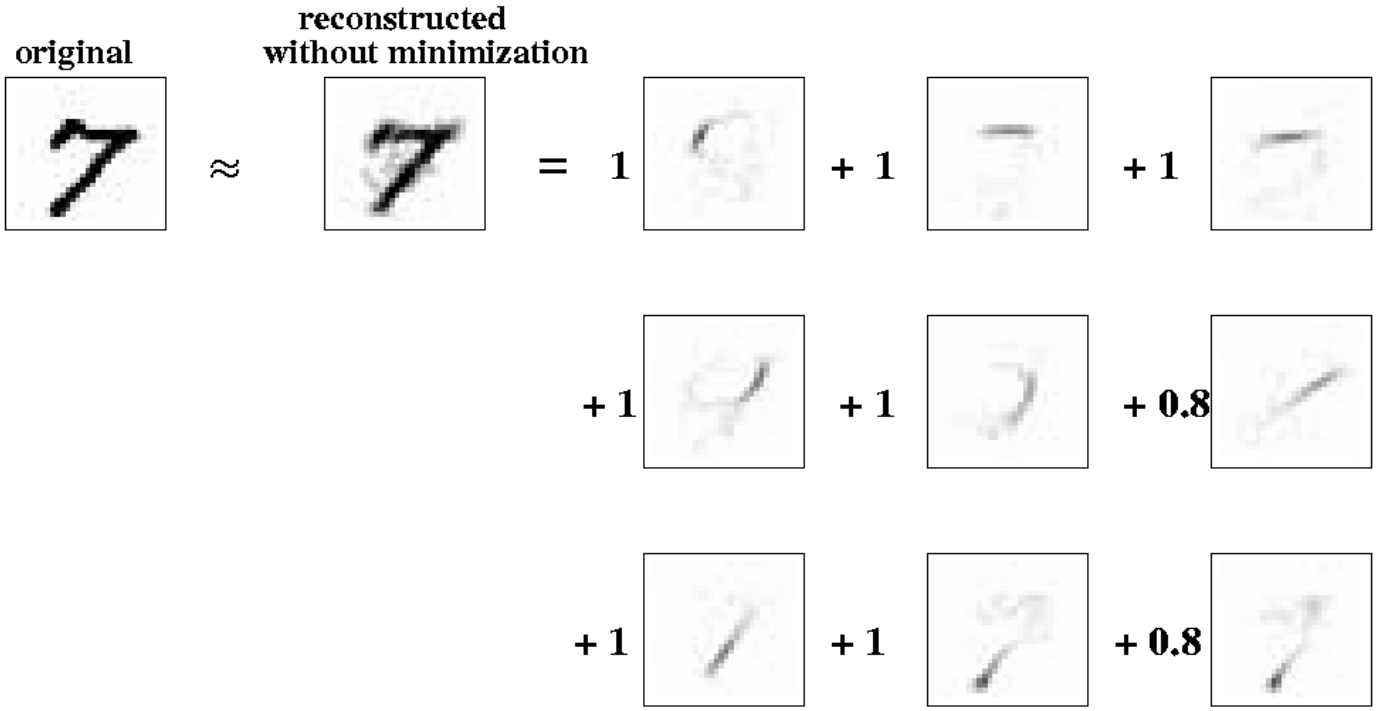
Handwritten digits - MNIST

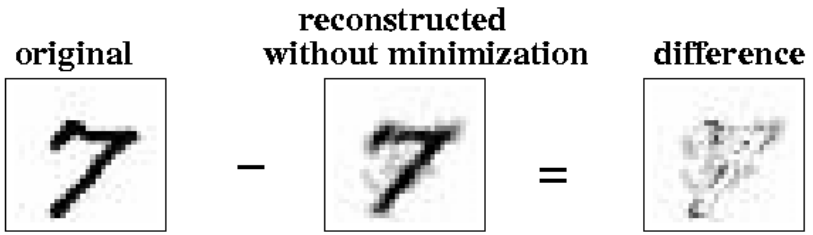


- 60,000 28x28 images
- 196 units in the code
- η 0.01
- β_1
- learning rate 0.001
- L1, L2 regularizer 0.005

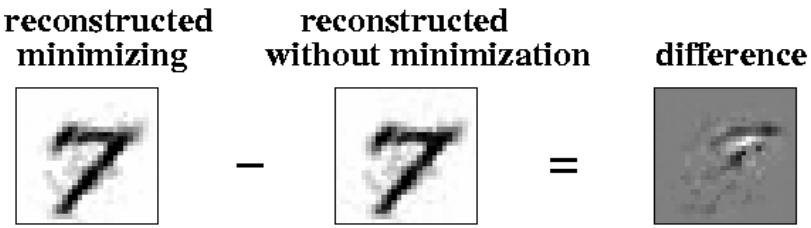
Encoder *direct* filters

Handwritten digits - MNIST





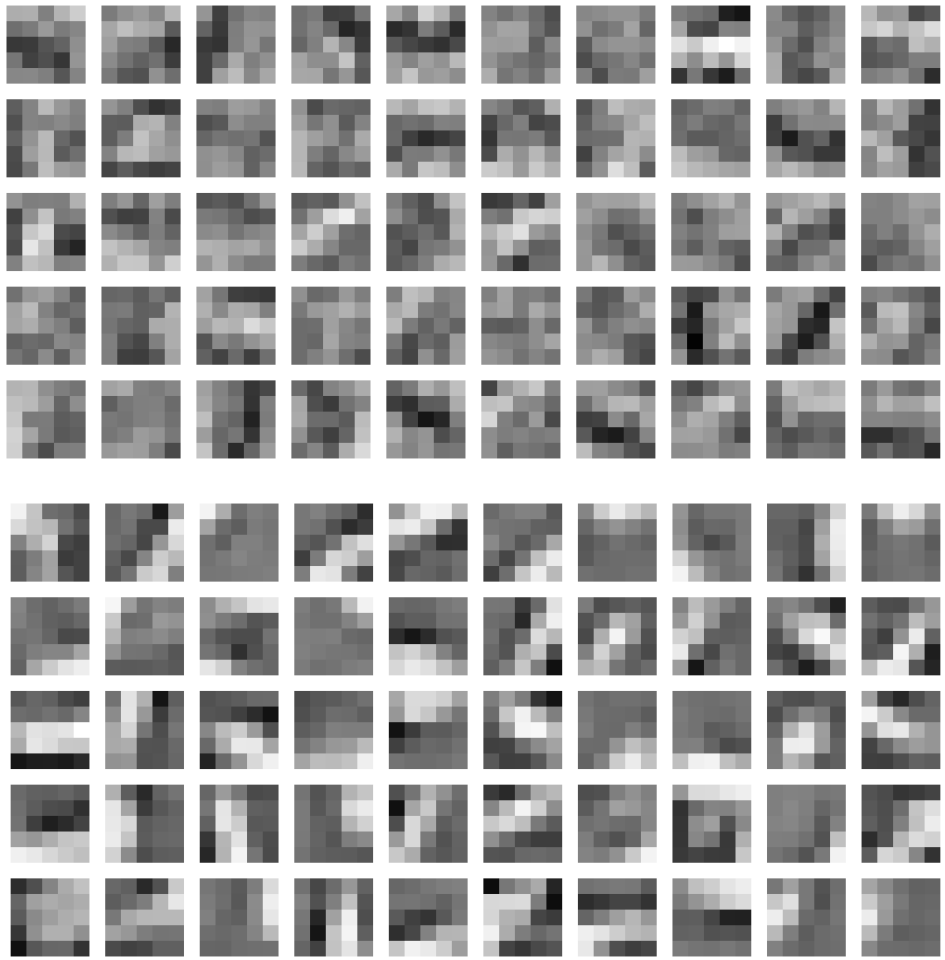
forward propagation through encoder and decoder



after training there is no need to minimize in code space

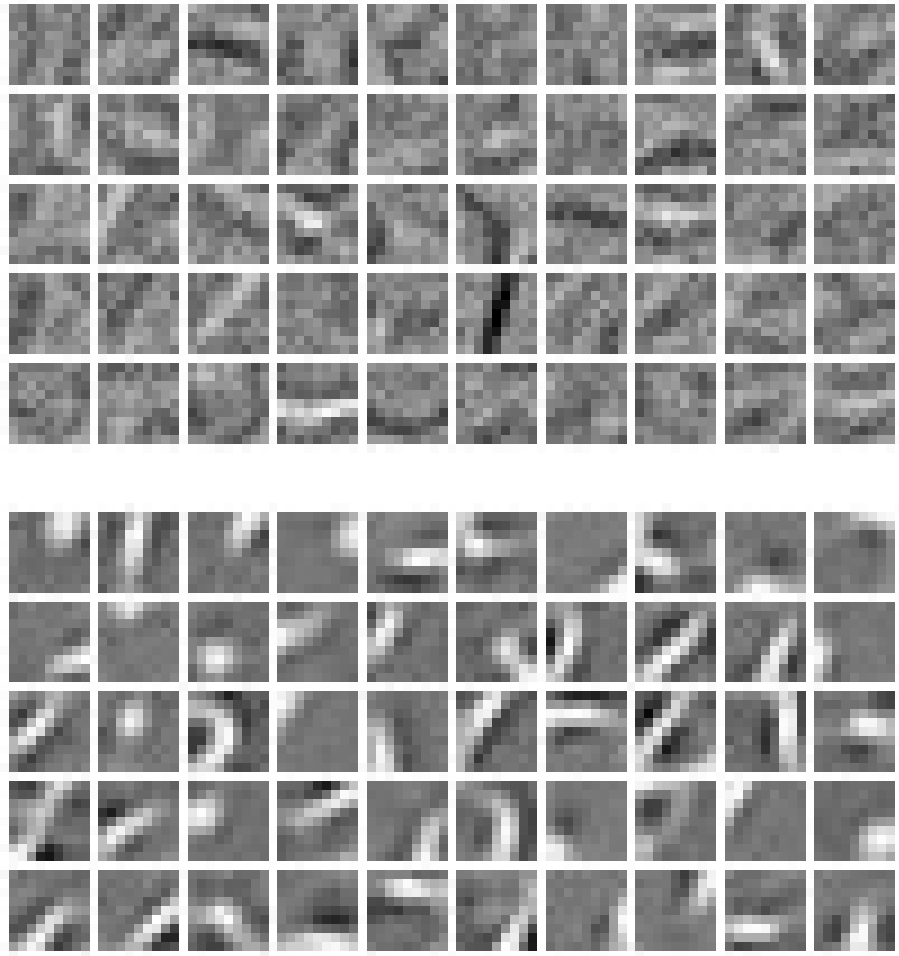
Initializing a Convolutional Net with SPoE

- **Architecture: LeNet-6**
  - 1->50->50->200->10
- **Baseline: random initialization**
  - 0.7% error on test set
- **First Layer Initialized with SpoE**
  - 0.6% error on test set
- **Training with elastically-distorted samples:**
  - 0.38% error on test set



Initializing a Convolutional Net with SPoE

- **Architecture: LeNet-6**
  - 1->50->50->200->10
  - 9x9 kernels instead of 5x5
- **Baseline: random initialization**
- **First Layer Initialized with SpoE**



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

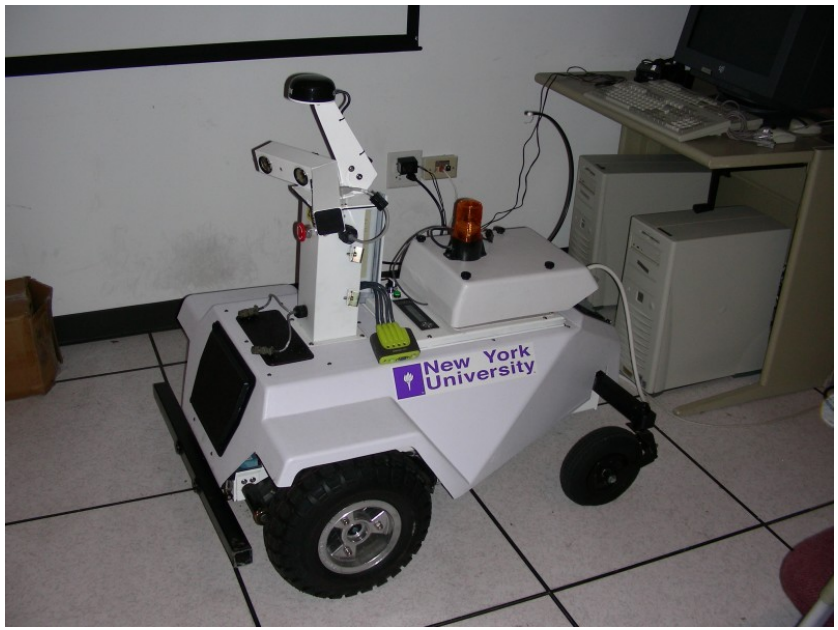
CLASSIFIER	DEFORMATION	ERROR	Reference
Knowledge-free methods			
2-layer NN, 800 HU, CE		1.60	Simard et al., ICDAR 2003
3-layer NN, 500+300 HU, CE, reg		1.53	Hinton, in press, 2005
SVM, Gaussian Kernel		1.40	Cortes 92 + Many others
Unsupervised Stacked RBM + backprop		0.95	Hinton, in press, 2005
Convolutional nets			
Convolutional net LeNet-5,		0.80	LeCun 2005 Unpublished
Convolutional net LeNet-6,		0.70	LeCun 2006 Unpublished
Conv. net LeNet-6- + unsup learning		0.60	LeCun 2006 Unpublished
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 et 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	0.38	LeCun 2006 Unpublished

Conclusion

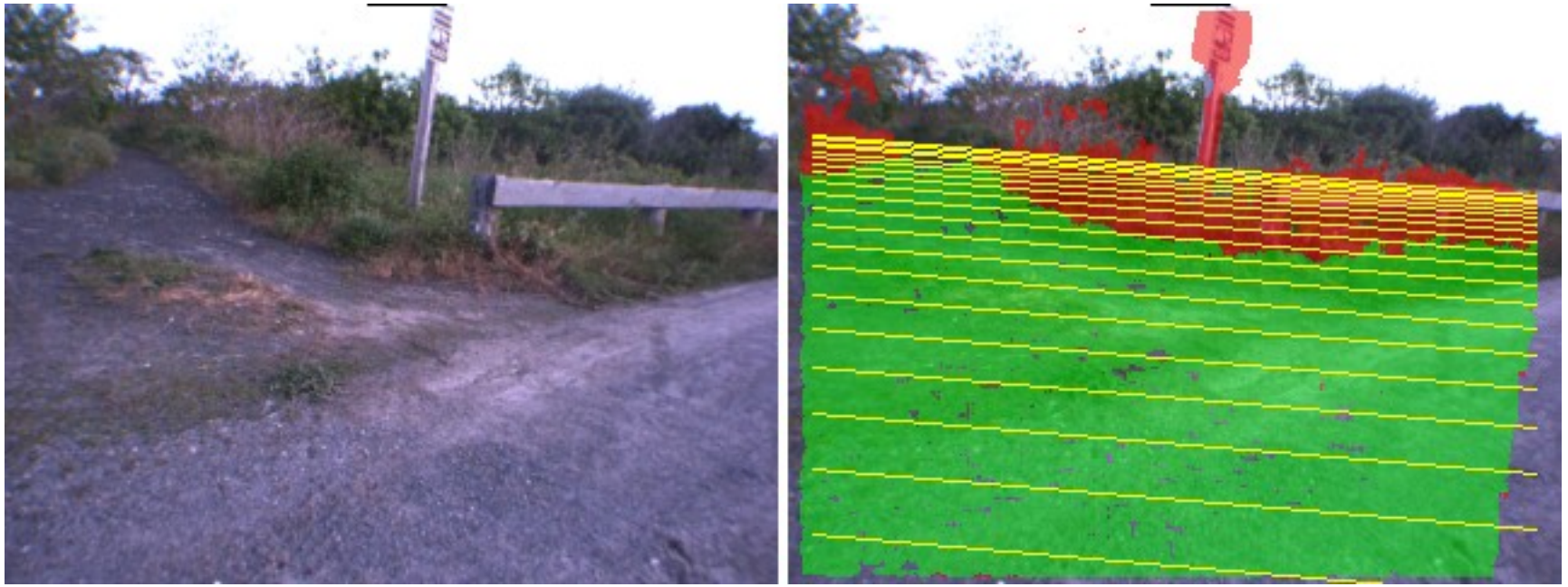
- Deep architectures are better than shallow ones
- We haven't solved the deep learning problem yet
- Larger networks are better
- Initializing the first layer(s) with unsupervised learning helps
- WANTED: a learning algorithm for deep architectures that seamlessly blends supervised and unsupervised learning

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.

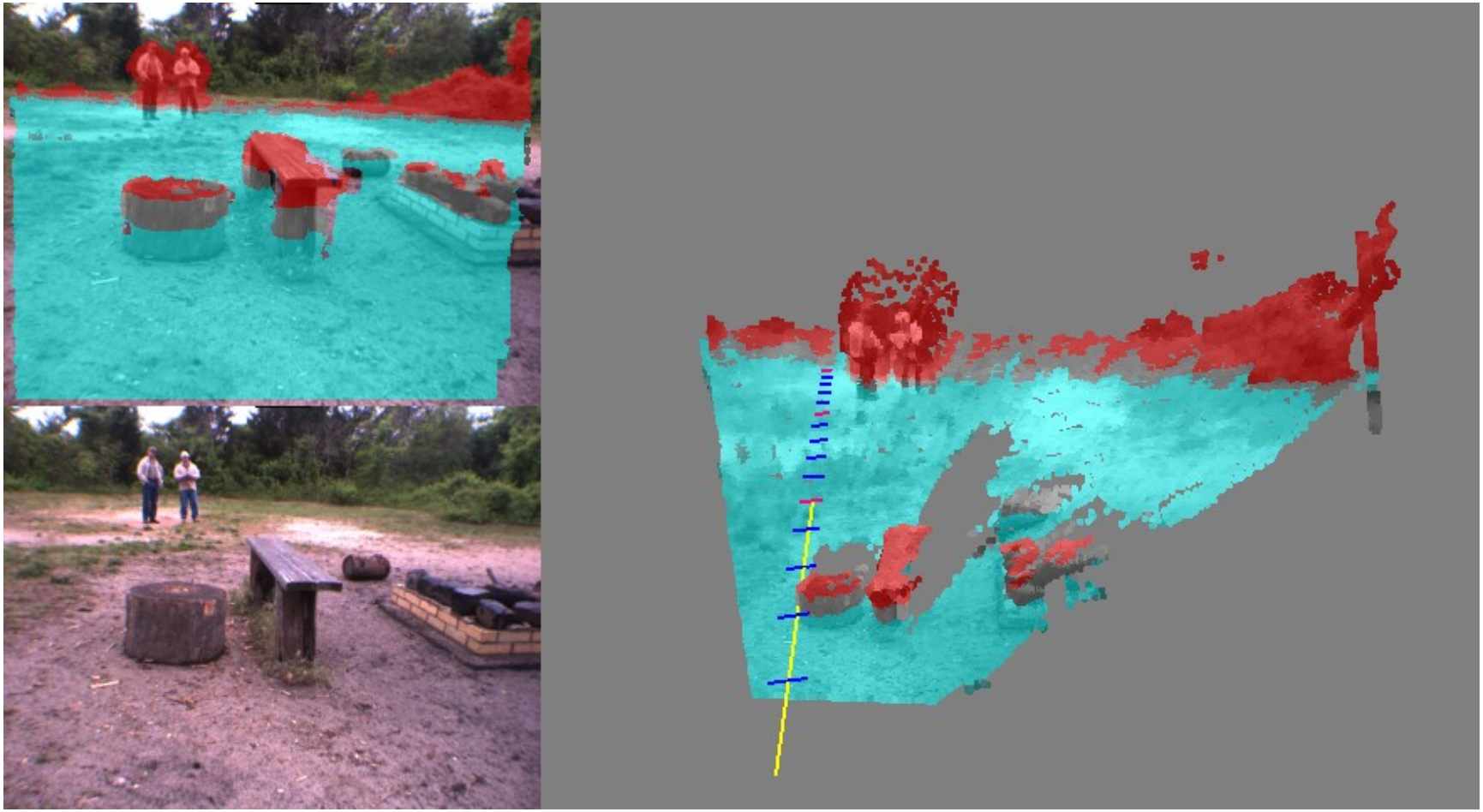


LAGR: Learning Applied to Ground Robotics



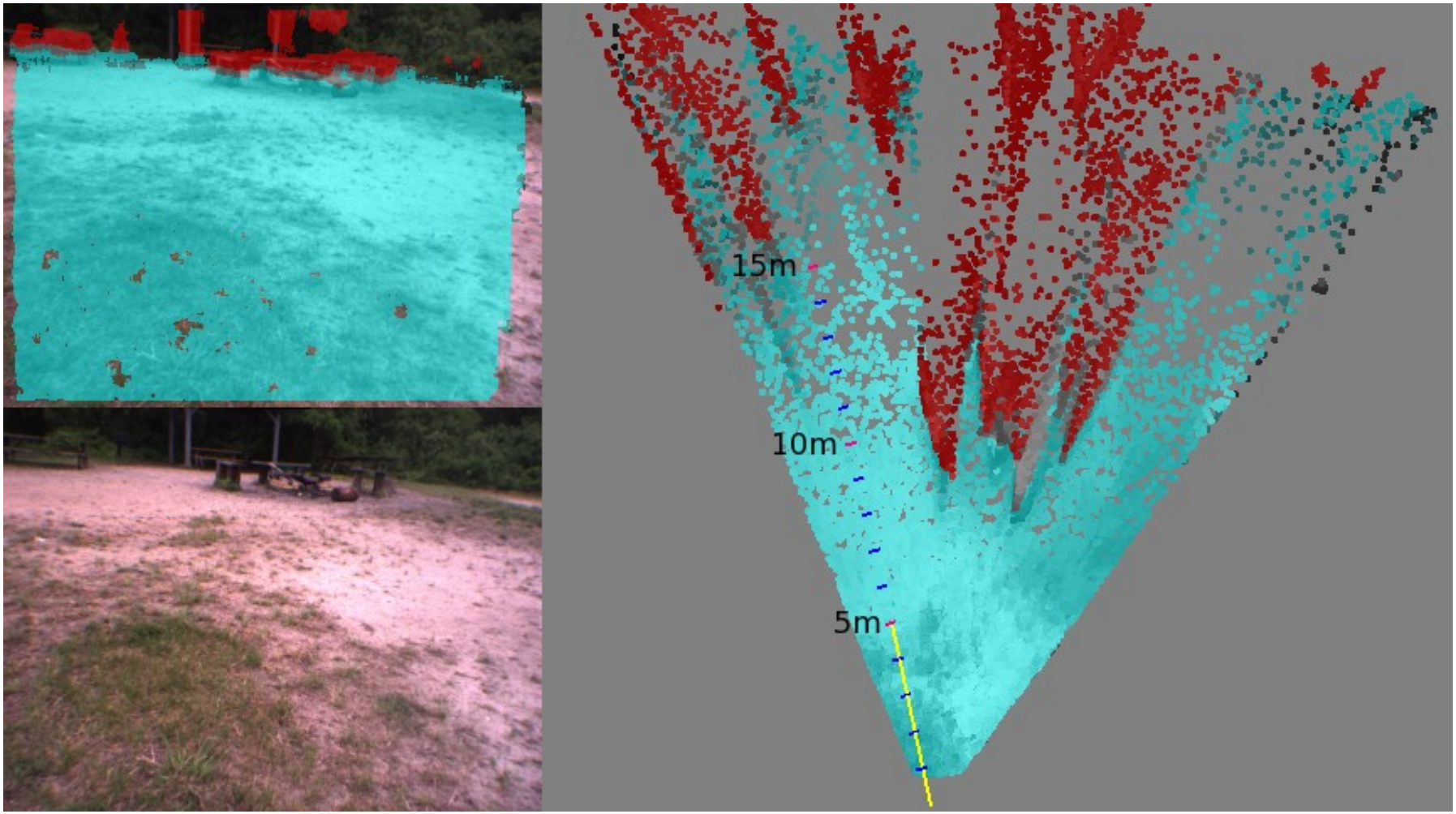
- Using stereo vision to estimate the distance of each pixel
- Estimating the location of the ground-plane using robust fitting
- Identifying what sticks out of the ground
- Building a map of the nearby environment
- PROBLEM: stereo vision is limited to a range of about 10 meters

LAGR: Stereo-Based Obstacle Detection



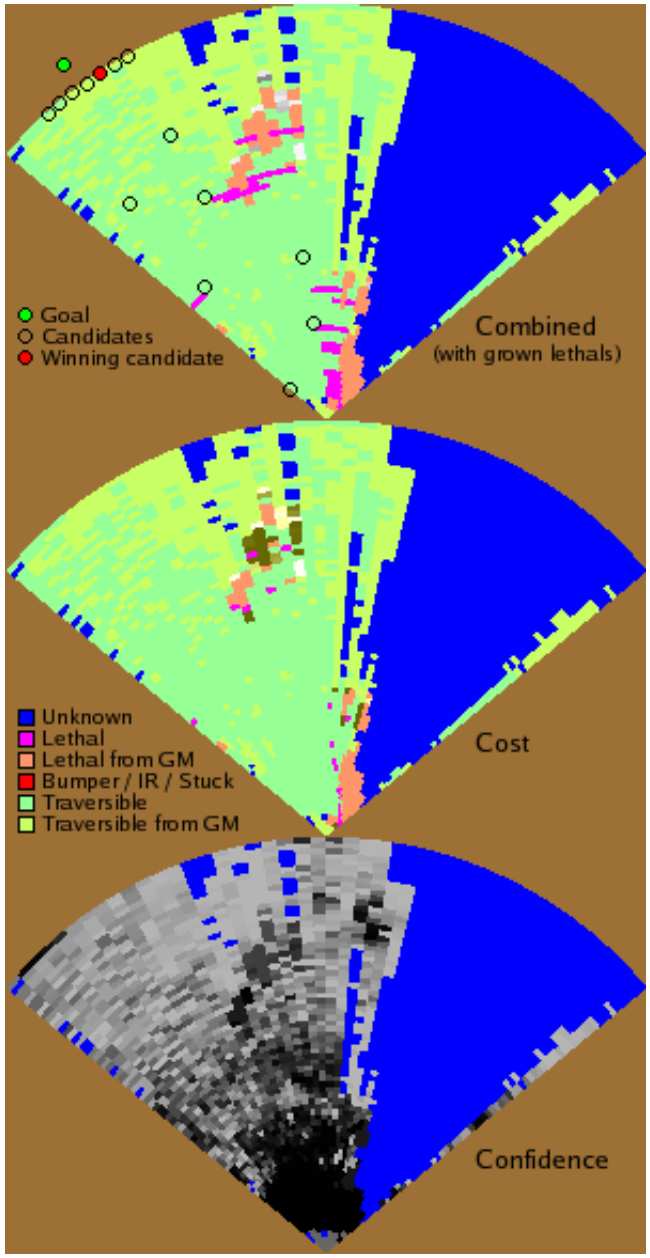
- Obstacle maps can be built from stereo.
- They are accurate to about 8-10 meters.

LAGR: Near-Sightedness of Stereo



- Stereo-based maps get "smeared out" and sparse above 10 meters

LAGR: Trusting our Eyes



- The path is obvious, even though the absolute distances to the obstacles may be known only approximately.
- This cannot be used to build a map, but we can certainly use it to decide in which direction to go next.
- SOLUTION: a polar map. Directions are accurate, independently of distance.

LAGR

- Long-Range Obstacle Detection with On-Line Learning.
- Distance-invariant image pyramid (object size independent of distance)
- Trainable obstacle detector: trained on short-range image bands using labels provided by stereo