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,

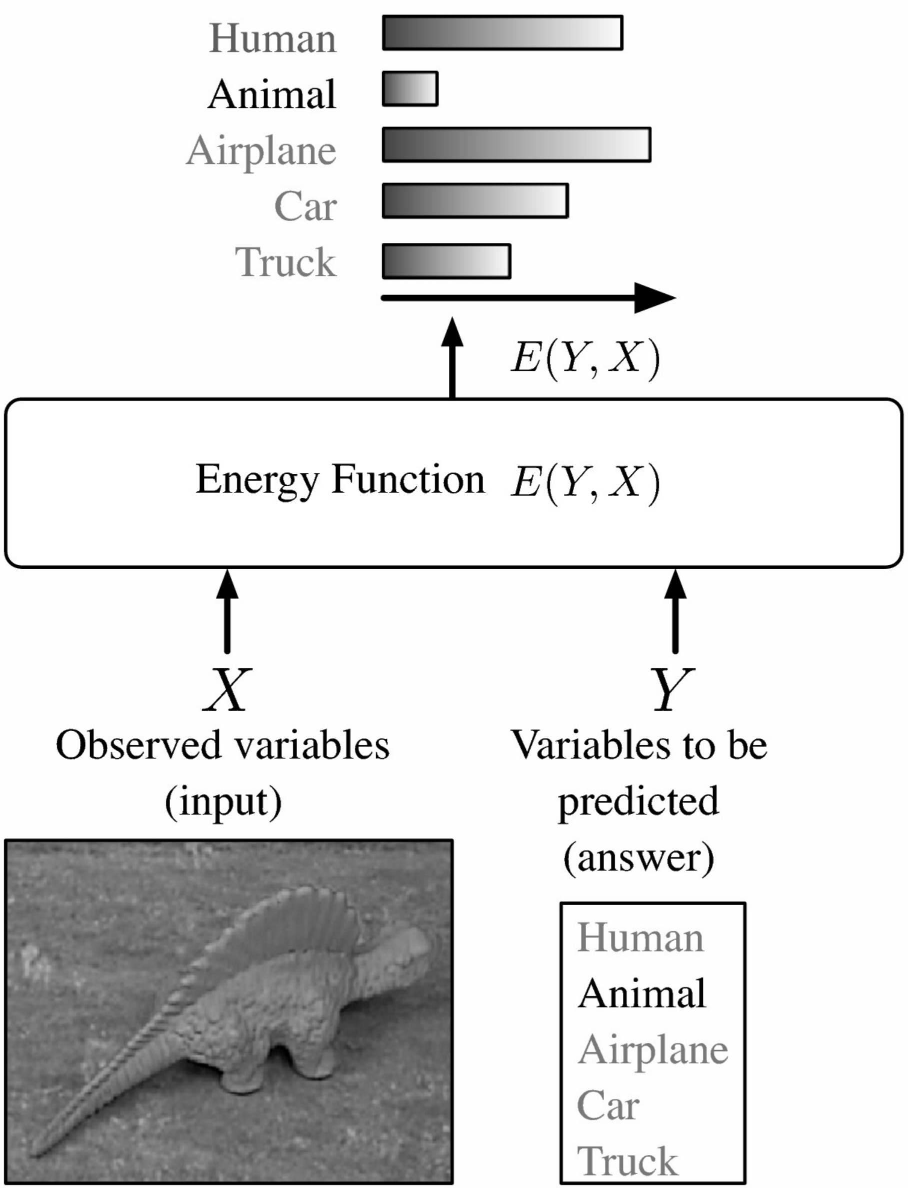
http://yann.lecun.com

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

Two Big Problems in Machine Learning

- **1. The "Intractable Partition Function Problem"**
  - Give high probability (or low energy) to good answers
  - Give low probability (or high energy) to bad answers
  - There are too many bad answers!
  - The normalization constant of probabilistic models is a sum over too many terms.
- **2. The "Deep Learning Problem"**
  - Training "Deep Belief Networks" is a necessary step towards solving the invariance problem in vision (and perception in general).
  - How do we train deep architectures with lots of non-linear stages?
- **This talks addresses those two problems:**
  - The partition function problem arises with probabilistic approaches. Non-probabilistic **Energy-Based Models** may allow us to get around it.
  - How far can we go with traditional deep learning methods (backprop)
  - How unsupervised feature learning can help guide deep learning.

Energy-Based Model for Decision-Making

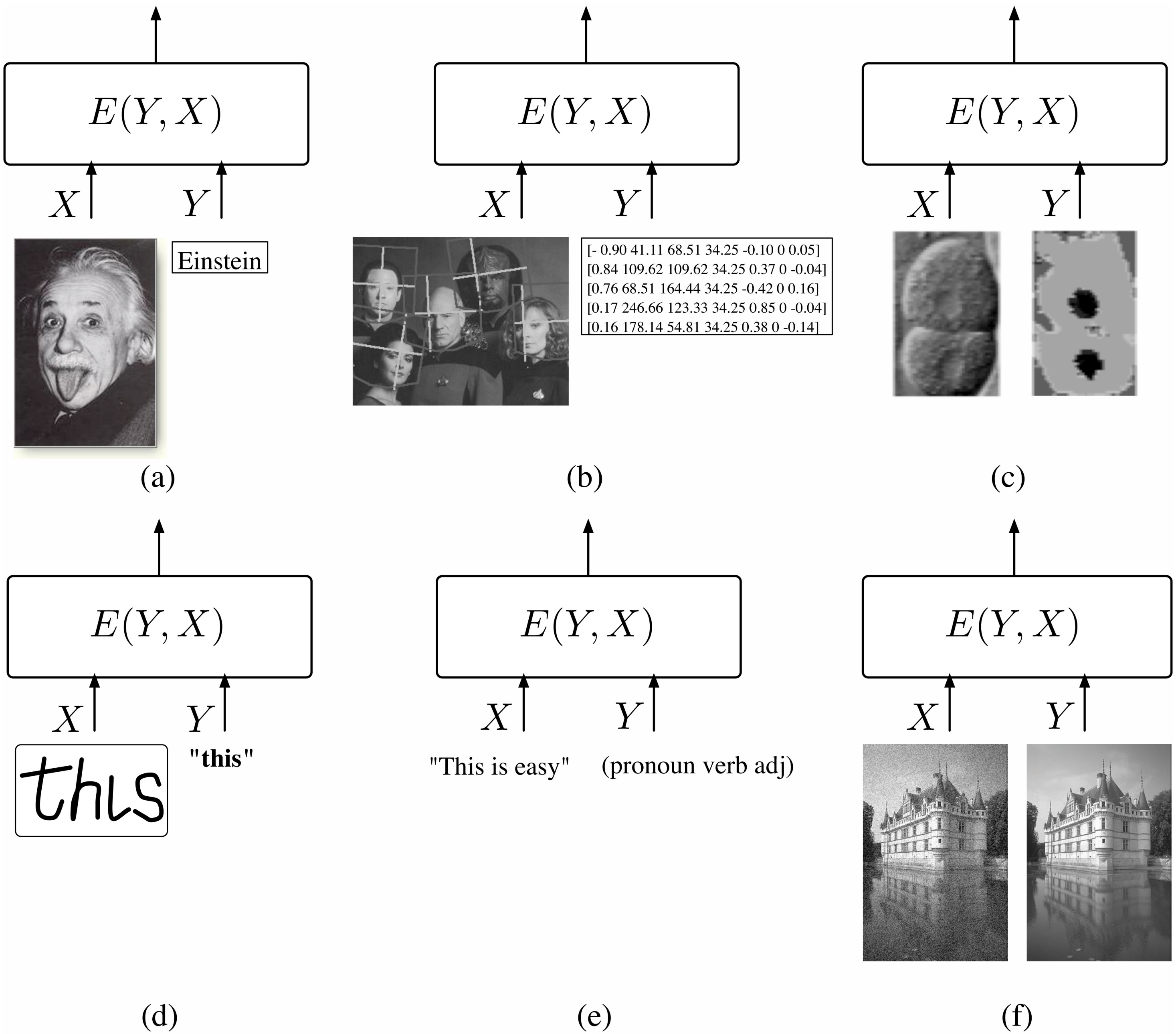


- **Model: Measures the compatibility between an observed variable X and a variable to be predicted Y through an energy function E(Y,X).**

$$Y^* = \text{argmin}_{Y \in \mathcal{Y}} E(Y, X).$$

- **Inference: Search for the Y that minimizes the energy within a set $\mathcal{Y}$**
- **If the set has low cardinality, we can use exhaustive search.**

Complex Tasks: Inference is non-trivial



- **When the cardinality or dimension of Y is large, exhaustive search is impractical.**
- **We need to use “smart” inference procedures: min-sum, Viterbi, min cut, gradient decent.....**

What Questions Can a Model Answer?

1. Classification & Decision Making:

- "which value of Y is most compatible with X?"
- Applications: Robot navigation,.....
- Training: give the lowest energy to the correct answer

2. Ranking:

- "Is Y1 or Y2 more compatible with X?"
- Applications: Data-mining....
- Training: produce energies that rank the answers correctly

3. Detection:

- "Is this value of Y compatible with X"?
- Application: face detection....
- Training: energies that increase as the image looks less like a face.

4. Conditional Density Estimation:

- "What is the conditional distribution P(Y|X)?"
- Application: feeding a decision-making system
- Training: differences of energies must be just so.

Decision-Making versus Probabilistic Modeling

- **Energies are uncalibrated**
  - The energies of two separately-trained systems cannot be combined
  - The energies are uncalibrated (measured in arbitrary untis)
- **How do we calibrate energies?**
  - We turn them into probabilities (positive numbers that sum to 1).
  - Simplest way: Gibbs distribution
  - Other ways can be reduced to Gibbs by a suitable redefinition of the energy.

$$P(Y|X) = \frac{e^{-\beta E(Y,X)}}{\int_{y \in \mathcal{Y}} e^{-\beta E(y,X)}},$$

Partition function

Inverse temperature

Architecture and Loss Function

- **Family of energy functions** $\mathcal{E} = \{E(W,Y,X) : \quad W \in \mathcal{W}\}.$
- **Training set** $\mathcal{S} = \{(X^i, Y^i) : i = 1 \ldots P\}$
- **Loss functional / Loss function** $\mathcal{L}(E,\mathcal{S})$ $\quad \mathcal{L}(W,\mathcal{S})$
  - Measures the quality of an energy function on training set
- **Training** $W^* = \min_{W \in \mathcal{W}} \mathcal{L}(W,\mathcal{S}).$
- **Form of the loss functional**
  - invariant under permutations and repetitions of the samples

$$\mathcal{L}(E,\mathcal{S}) = \frac{1}{P}\sum_{i=1}^{P} L(Y^i, E(W,\mathcal{Y},X^i)) + R(W).$$

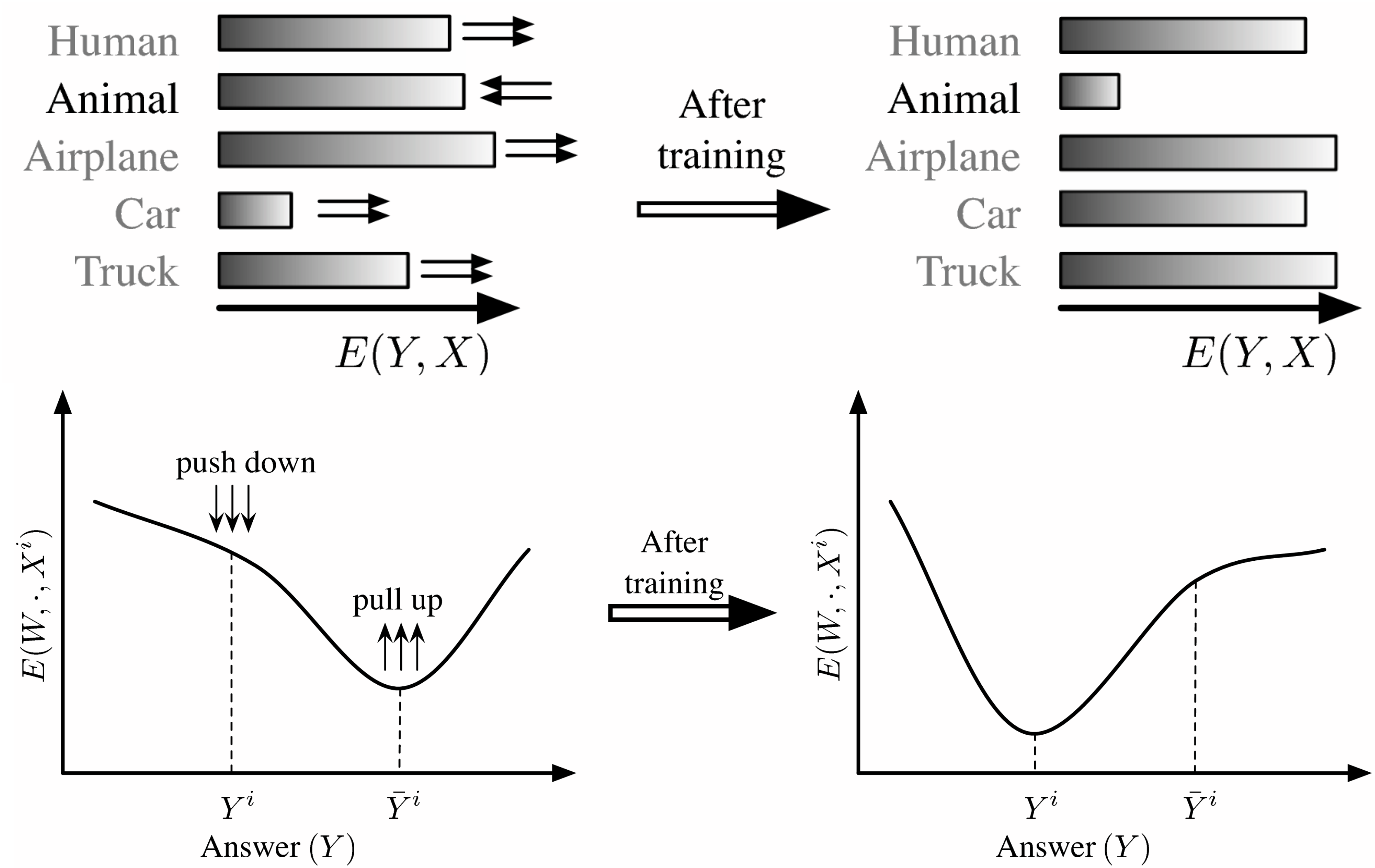
Per-sample loss

Desired answer

Energy surface for a given Xi as Y varies

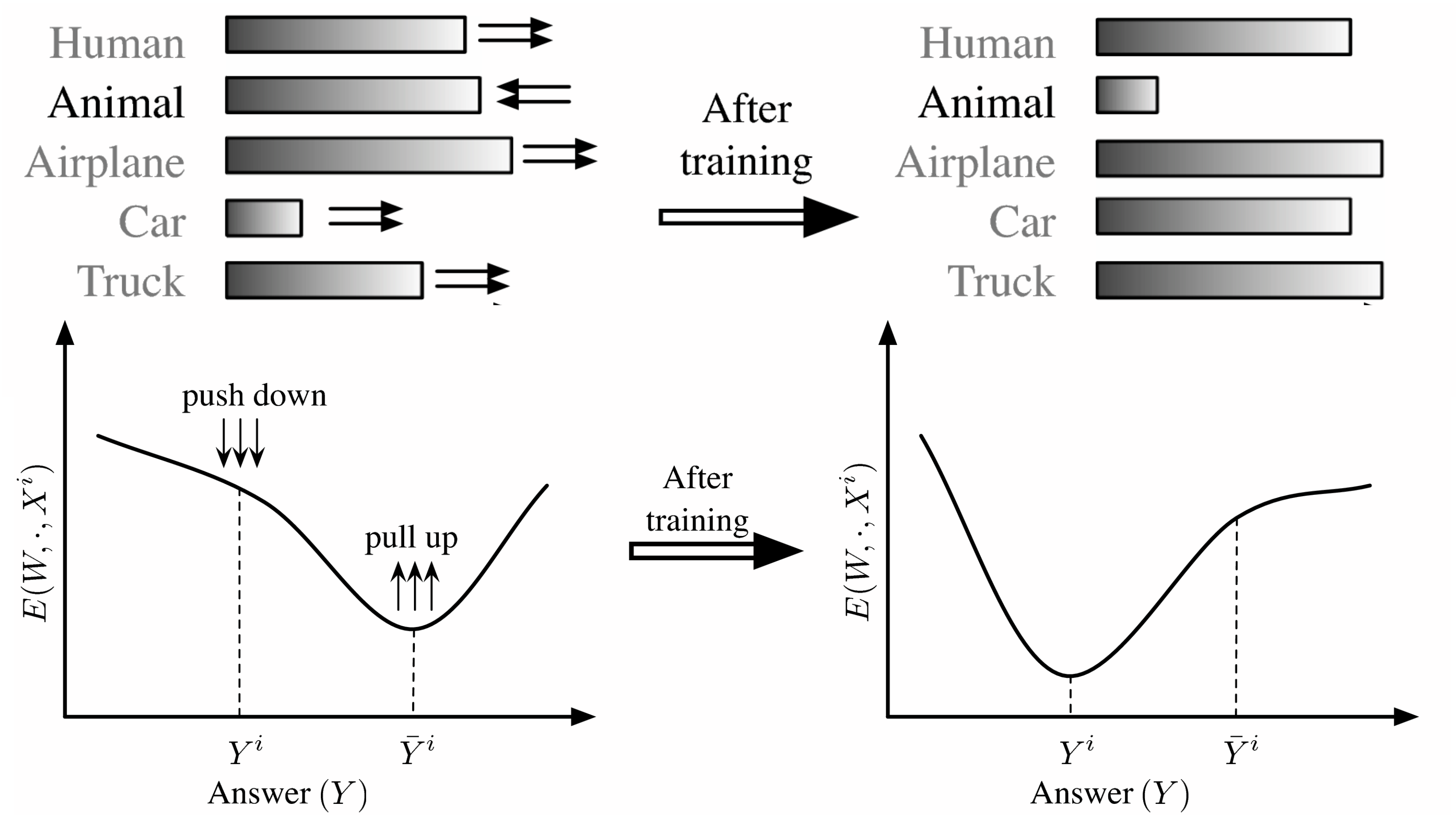
Regularizer

Designing a Loss Functional



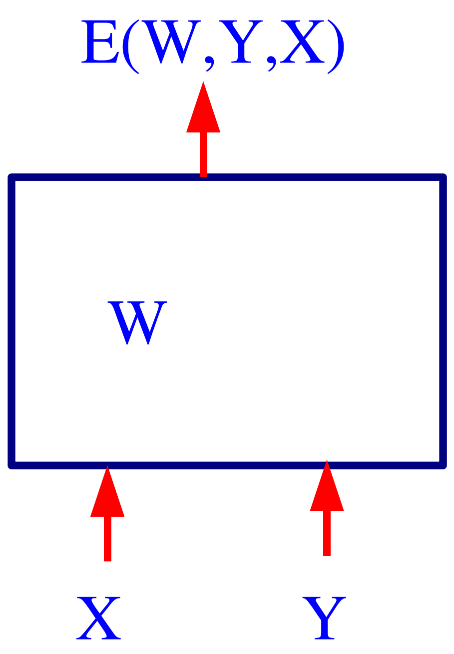
- **Correct answer has the lowest energy -> LOW LOSS**
- **Lowest energy is not for the correct answer -> HIGH LOSS**

Designing a Loss Functional



- **Push down on the energy of the correct answer**
- **Pull up on the energies of the incorrect answers, particularly if they are smaller than the correct one**

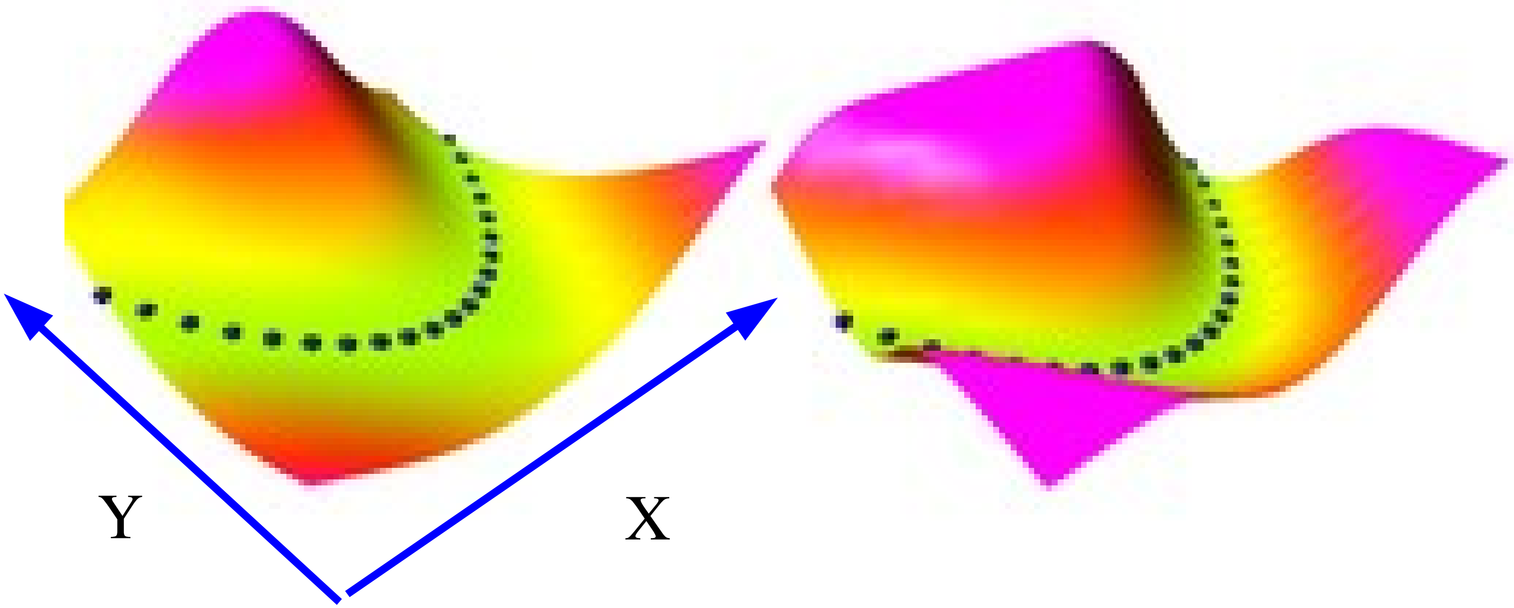
Architecture + Inference Algo + Loss Function = Model



- **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.**

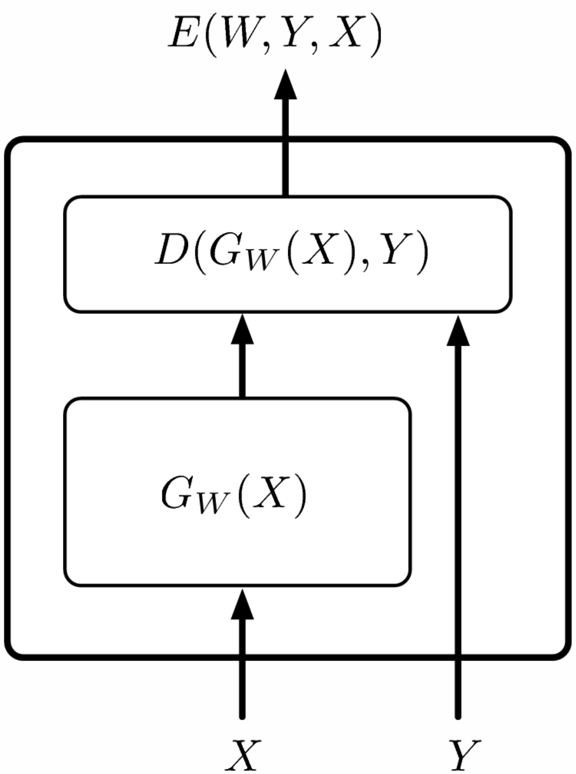
- **PROBLEM: What loss functions will make the machine approach the desired behavior?**

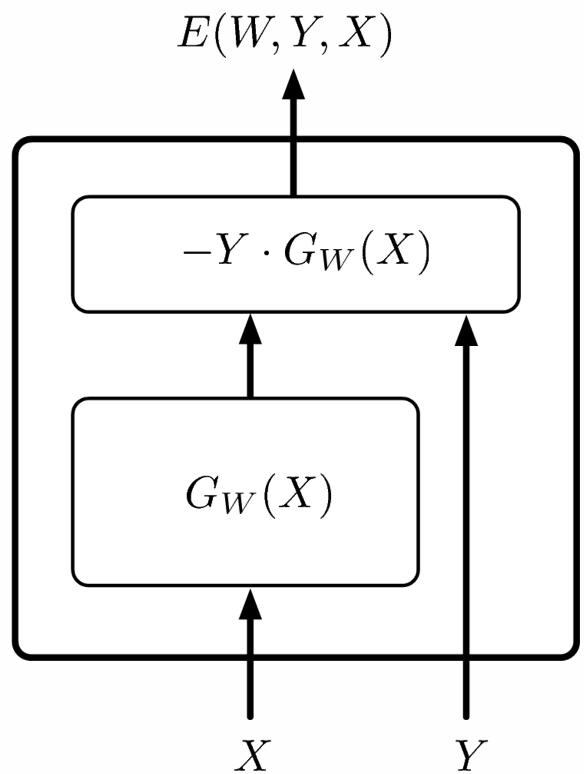
Several Energy Surfaces can give the same answers

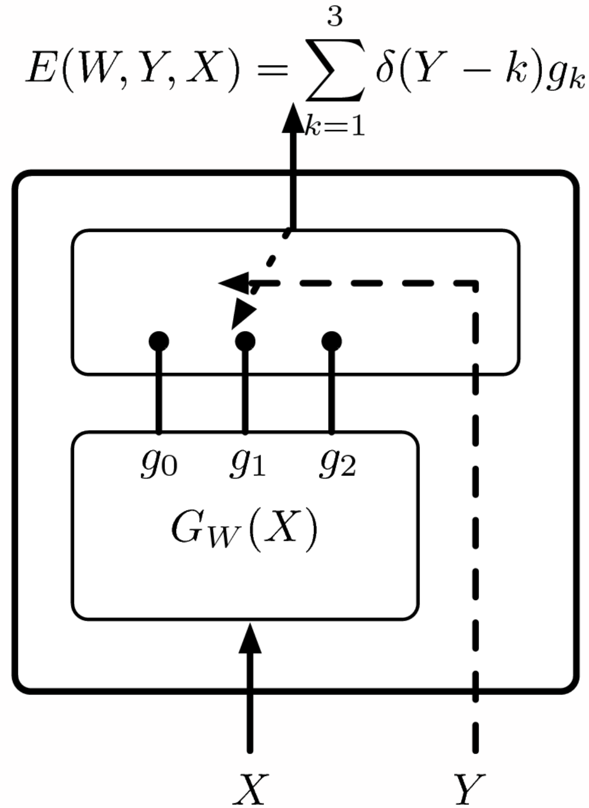


- Both surfaces compute Y=X^2
- MINy E(Y,X) = X^2
- Minimum-energy inference gives us the same answer

Simple Architectures







- **Regression**

$$E(W,Y,X) = \frac{1}{2}||G_W(X) - Y||^2.$$

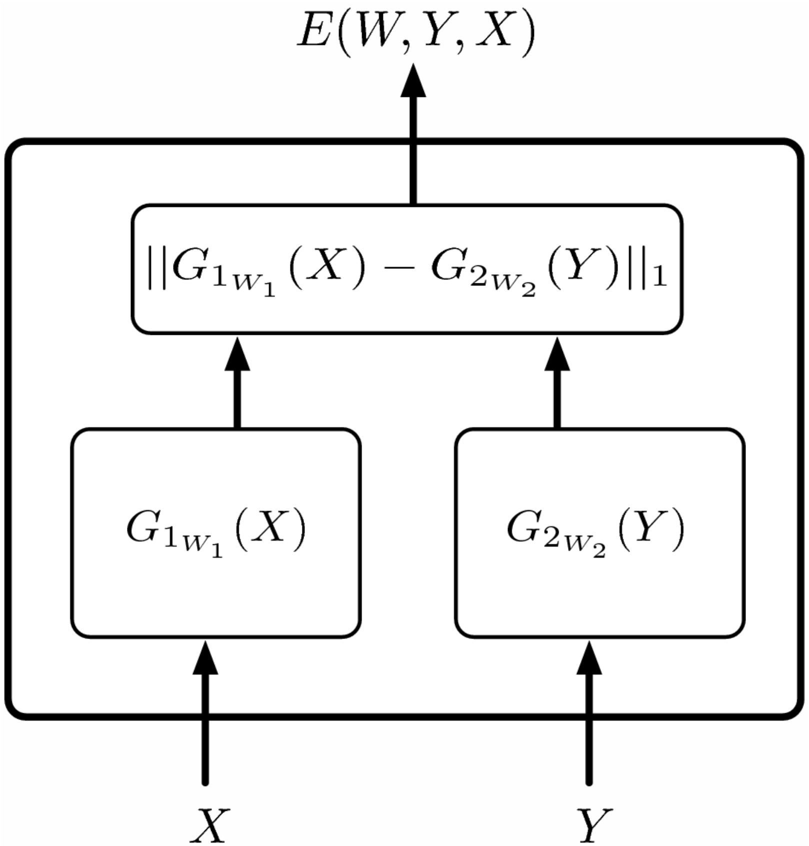
- **Binary Classification**

$$E(W,Y,X) = -YG_W(X),$$

- **Multi-class Classification**

Simple Architecture: Implicit Regression

$$E(W, X, Y) = ||G_{1_{W_1}}(X) - G_{2_{W_2}}(Y)||_1,$$



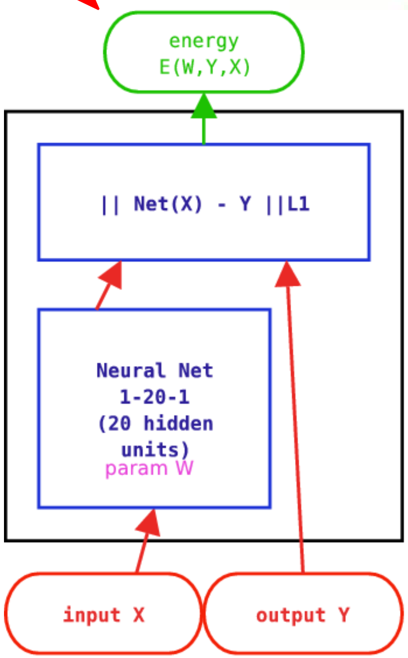
The Implicit Regression architecture

- allows multiple answers to have low energy.
- Encodes a constraint between X and Y rather than an explicit functional relationship
- This is useful for many applications
- Example: sentence completion: “The cat ate the {mouse,bird,homework,...}”
- [Bengio et al. 2003]
- But, inference may be difficult.

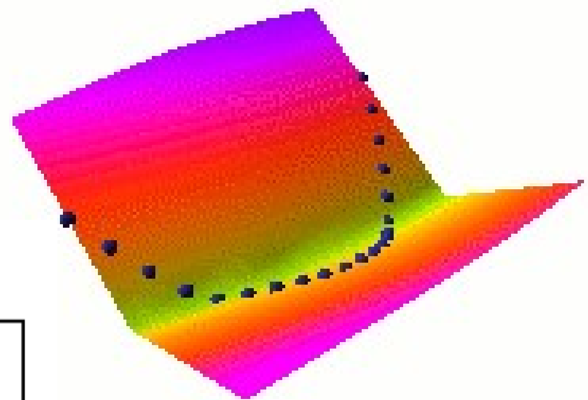
Examples of Loss Functions: Energy Loss

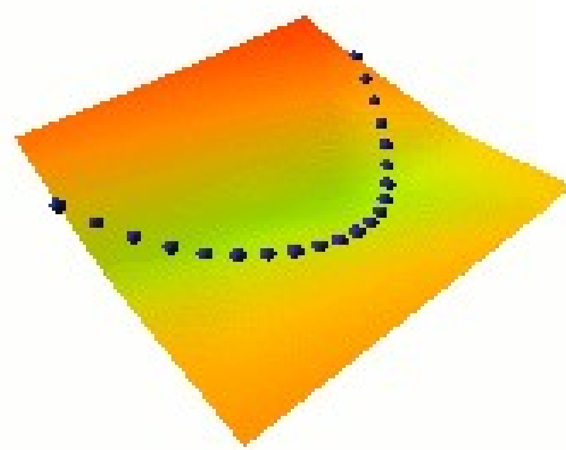
- **Energy Loss** $$L_{energy}(Y^i, E(W, \mathcal{Y}, X^i)) = E(W, Y^i, X^i).$$
  - Simply pushes down on the energy of the correct answer

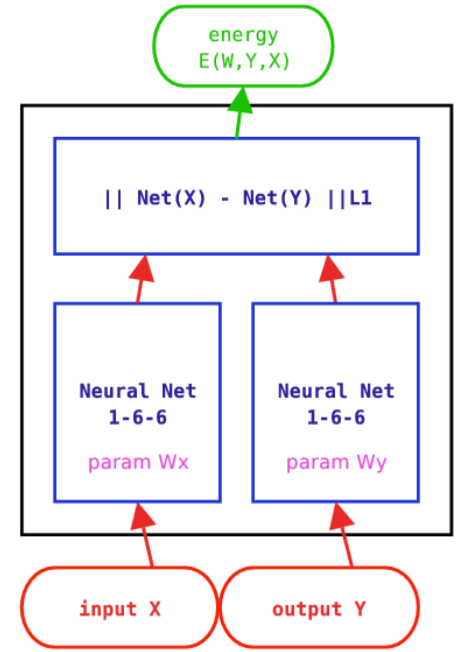
WORKS!!!



(a)







(b)

COLLAPSES!!!

Examples of Loss Functions:Perceptron Loss

$$L_{perceptron}(Y^i, E(W, \mathcal{Y}, X^i)) = E(W, Y^i, X^i) - \min_{Y \in \mathcal{Y}} E(W, Y, X^i).$$

- **Perceptron Loss [LeCun et al. 1998], [Collins 2002]**
  - Pushes down on the energy of the correct answer
  - Pulls up on the energy of the machine's answer
  - Always positive. Zero when answer is correct
  - No “margin”: technically does not prevent the energy surface from being almost flat.
  - Works pretty well in practice, particularly if the energy parameterization does not allow flat surfaces.

Perceptron Loss for Binary Classification

$$L_{perceptron}(Y^i, E(W, \mathcal{Y}, X^i)) = E(W, Y^i, X^i) - \min_{Y \in \mathcal{Y}} E(W, Y, X^i).$$

- **Energy:** $E(W, Y, X) = -Y G_W(X),$
- **Inference:** $Y^* = \text{argmin}_{Y \in \{-1,1\}} - Y G_W(X) = \text{sign}(G_W(X)).$
- **Loss:** $\mathcal{L}_{\text{perceptron}}(W, \mathcal{S}) = \frac{1}{P} \sum_{i=1}^{P} \left(\text{sign}(G_W(X^i)) - Y^i\right) G_W(X^i).$
- **Learning Rule:** $W \leftarrow W + \eta \left(Y^i - \text{sign}(G_W(X^i))\right) \frac{\partial G_W(X^i)}{\partial W},$
- **If Gw(X) is linear in W:** $E(W, Y, X) = -Y W^T \Phi(X)$

$$W \leftarrow W + \eta \left(Y^i - \text{sign}(W^T \Phi(X^i))\right) \Phi(X^i)$$

Examples of Loss Functions: Generalized Margin Losses

- **First, we need to define the Most Offending Incorrect Answer**

- **Most Offending Incorrect Answer: discrete case**

Definition 1 *Let Y be a discrete variable. Then for a training sample (X^i, Y^i), the* ***most offending incorrect answer*** *$\bar{Y}^i$ is the answer that has the lowest energy among all answers that are incorrect:*

$$\bar{Y}^i = \text{argmin}_{Y \in \mathcal{Y} and Y \neq Y^i} E(W, Y, X^i). \quad (8)$$

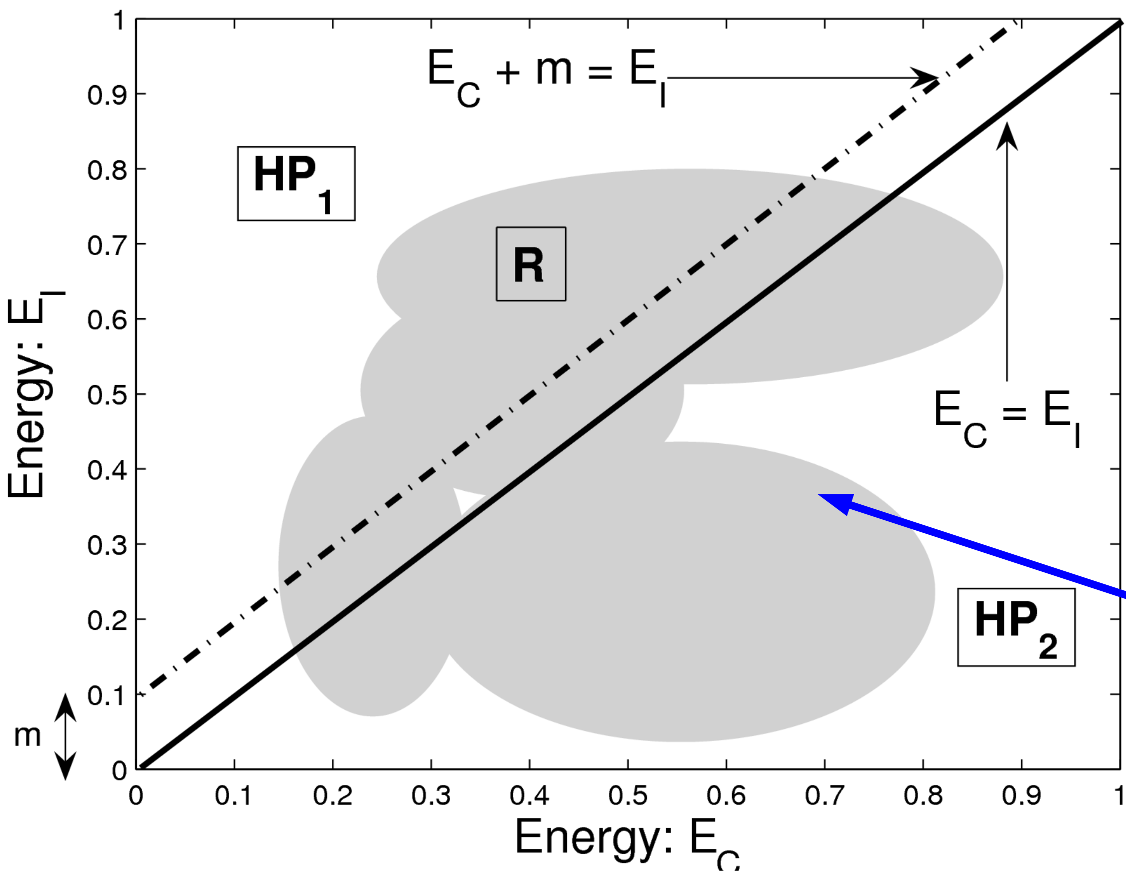
- **Most Offending Incorrect Answer: continuous case**

Definition 2 *Let Y be a continuous variable. Then for a training sample (X^i, Y^i), the* ***most offending incorrect answer*** *$\bar{Y}^i$ is the answer that has the lowest energy among all answers that are at least ϵ away from the correct answer:*

$$\bar{Y}^i = \text{argmin}_{Y \in \mathcal{Y}, \|Y - Y^i\| > \epsilon} E(W, Y, X^i). \quad (9)$$

Examples of Loss Functions: Generalized Margin Losses

$$L_{\text{margin}}(W, Y^i, X^i) = Q_m\left(E(W, Y^i, X^i), E(W, \bar{Y}^i, X^i)\right).$$



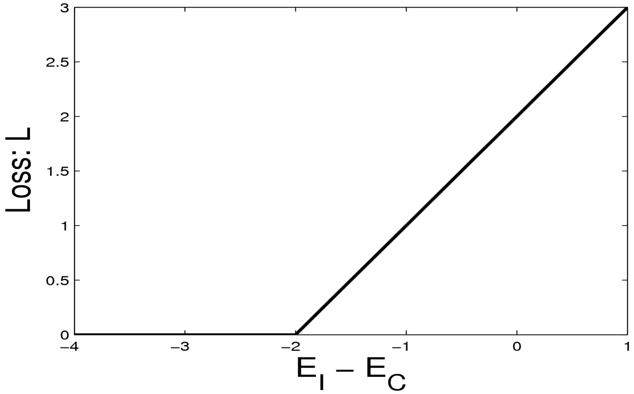
- **Generalized Margin Loss**
  - Qm increases with the energy of the correct answer
  - Qm decreases with the energy of the most offending incorrect answer
  - whenever it is less than the energy of the correct answer plus a margin m.

Examples of Generalized Margin Losses

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

Hinge Loss

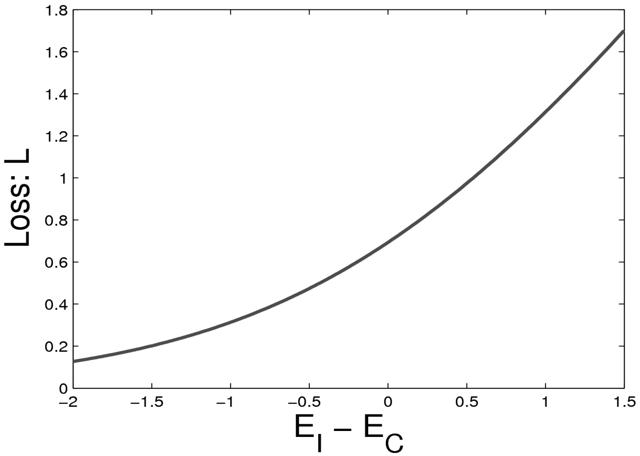
- [Altun et al. 2003], [Taskar et al. 2003]
- With the linearly-parameterized binary classifier architecture, we get linear SVM



$$L_{\log}(W, Y^i, X^i) = \log\left(1 + e^{E(W,Y^i,X^i) - E(W,\bar{Y}^i,X^i)}\right).$$

Log Loss

- "soft hinge" loss
- With the linearly-parameterized binary classifier architecture, we get linear Logistic Regression

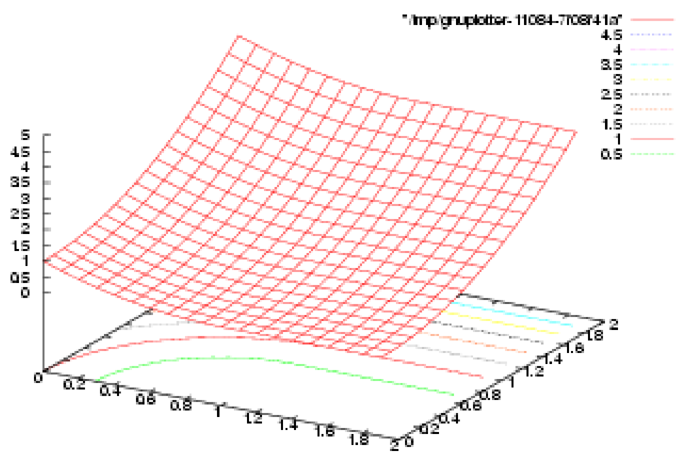


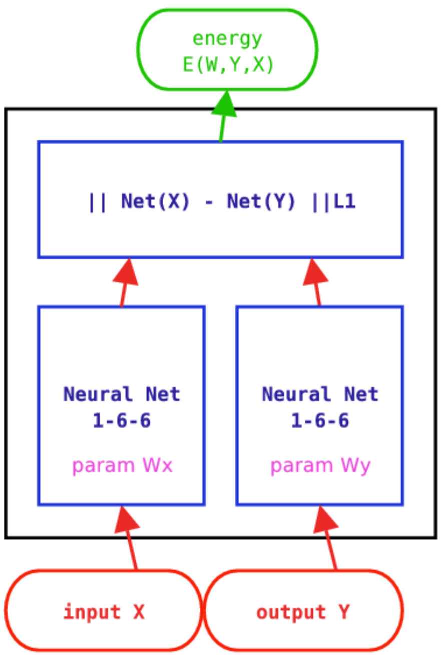
Examples of Margin Losses: Square-Square Loss

$$L_{\text{sq-sq}}(W, Y^i, X^i) = E(W, Y^i, X^i)^2 + \left(\max(0, m - E(W, \bar{Y}^i, X^i))\right)^2.$$

Square-Square Loss

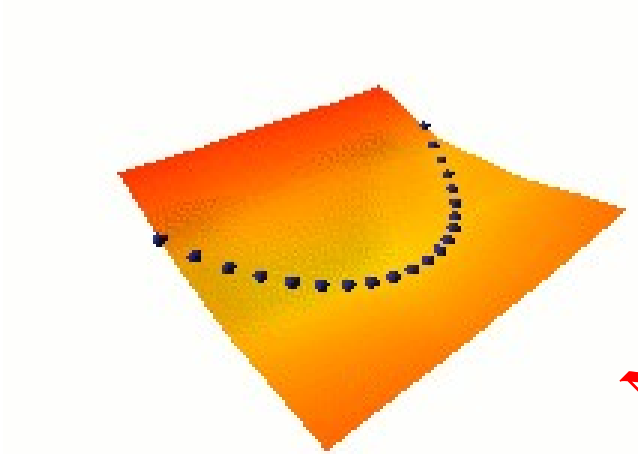
- [LeCun-Huang 2005]
- Appropriate for positive energy functions





(b)

Learning Y = X^2



NO COLLAPSE!!!

Other Margin-Like Losses

- **LVQ2 Loss** [Kohonen, Oja], Driancourt-Bottou 1991]

$$L_{\mathrm{lvq2}}(W, Y^i, X^i) = \min\left(1, \max\left(0, \frac{E(W, Y^i, X^i) - E(W, \bar{Y}^i, X^i)}{\delta E(W, \bar{Y}^i, X^i)}\right)\right),$$

- **Minimum Classification Error Loss** [Juang, Chou, Lee 1997]

$$L_{\mathrm{mce}}(W, Y^i, X^i) = \sigma\left(E(W, Y^i, X^i) - E(W, \bar{Y}^i, X^i)\right),$$

$$\sigma(x) = (1 + e^{-x})^{-1}$$

- **Square-Exponential Loss** [Osadchy, Miller, LeCun 2004]

$$L_{\mathrm{sq-exp}}(W, Y^i, X^i) = E(W, Y^i, X^i)^2 + \gamma e^{-E(W, \bar{Y}^i, X^i)},$$

Negative Log-Likelihood Loss

- **Conditional probability of the samples (assuming independence)**

$$P(Y^1,\ldots,Y^P|X^1,\ldots,X^P,W) = \prod_{i=1}^{P} P(Y^i|X^i,W).$$

$$-\log \prod_{i=1}^{P} P(Y^i|X^i,W) = \sum_{i=1}^{P} -\log P(Y^i|X^i,W).$$

- **Gibbs distribution:** $P(Y|X^i,W) = \frac{e^{-\beta E(W,Y,X^i)}}{\int_{y\in\mathcal{Y}} e^{-\beta E(W,y,X^i)}}.$

$$-\log \prod_{i=1}^{P} P(Y^i|X^i,W) = \sum_{i=1}^{P} \beta E(W,Y^i,X^i) + \log \int_{y\in\mathcal{Y}} e^{-\beta E(W,y,X^i)}.$$

- **We get the NLL loss by dividing by P and Beta:**

$$\mathcal{L}_{\text{nll}}(W,\mathcal{S}) = \frac{1}{P}\sum_{i=1}^{P}\left(E(W,Y^i,X^i) + \frac{1}{\beta}\log \int_{y\in\mathcal{Y}} e^{-\beta E(W,y,X^i)}\right).$$

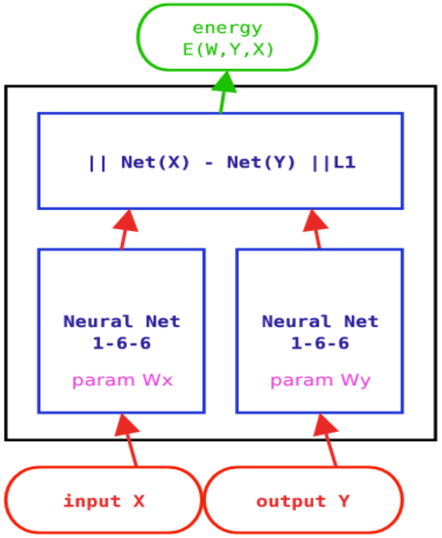
- **Reduces to the perceptron loss when Beta->infinity**

Negative Log-Likelihood Loss

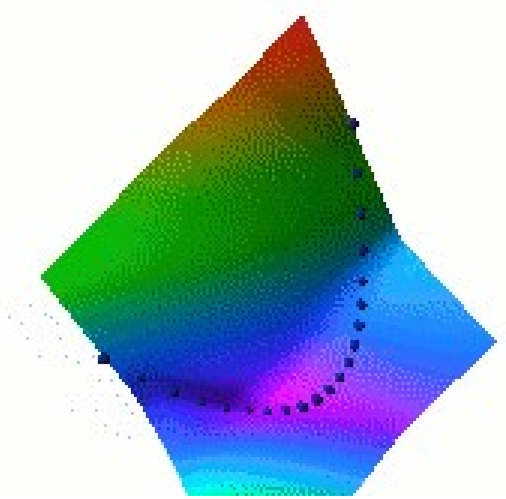
- **Pushes down on the energy of the correct answer**
- **Pulls up on the energies of all answers in proportion to their probability**

$$\mathcal{L}_{\text{nll}}(W, \mathcal{S}) = \frac{1}{P} \sum_{i=1}^{P} \left(E(W, Y^i, X^i) + \frac{1}{\beta} \log \int_{y \in \mathcal{Y}} e^{-\beta E(W, y, X^i)} \right).$$

$$\frac{\partial L_{\text{nll}}(W, Y^i, X^i)}{\partial W} = \frac{\partial E(W, Y^i, X^i)}{\partial W} - \int_{Y \in \mathcal{Y}} \frac{\partial E(W, Y, X^i)}{\partial W} P(Y | X^i, W),$$



(b)



Negative Log-Likelihood Loss: Binary Classification

- **Binary Classifier Architecture:**

$$\mathcal{L}_{\rm nll}(W,\mathcal{S}) = \frac{1}{P}\sum_{i=1}^{P}\left[-Y^iG_W(X^i) + \log\left(e^{Y^iG_W(X^i)} + e^{-Y^iG_W(X^i)}\right)\right].$$

$$\mathcal{L}_{\rm nll}(W,\mathcal{S}) = \frac{1}{P}\sum_{i=1}^{P}\log\left(1 + e^{-2Y^iG_W(X^i)}\right),$$

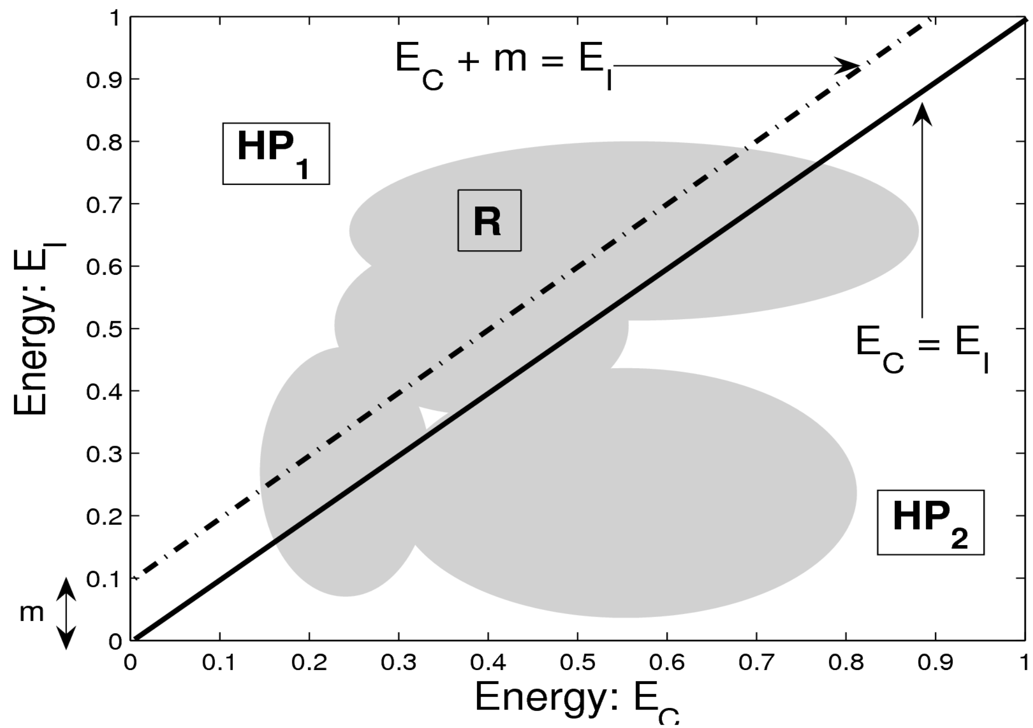
- **Linear Binary Classifier Architecture:**

$$\mathcal{L}_{\rm nll}(W,\mathcal{S}) = \frac{1}{P}\sum_{i=1}^{P}\log\left(1 + e^{-2Y^iW^T\Phi(X^i)}\right).$$

- **Learning Rule: logistic regression**

What Makes a "Good" Loss Function

- **Good loss functions make the machine produce the correct answer**
  - Avoid collapses and flat energy surfaces



Sufficient Condition on the Loss

Let (X^i, Y^i) be the i^{th} training example and m be a positive margin. Minimizing the loss function L will cause the machine to satisfy $E(W, Y^i, X^i) < E(W, Y, X^i) - m$ for all $Y \neq Y^i$, if there exists at least one point (e_1, e_2) with $e_1 + m < e_2$ such that for all points (e'_1, e'_2) with $e'_1 + m \geq e'_2$, we have

$$Q_{[E_y]}(e_1, e_2) < Q_{[E_y]}(e'_1, e'_2),$$

where $Q_{[E_y]}$ is given by

$$L(W, Y^i, X^i) = Q_{[E_y]}(E(W, Y^i, X^i), E(W, \bar{Y}^i, X^i)).$$

What Make a “Good” Loss Function

Good and bad loss functions

Loss (equation #)	Formula	Margin
energy loss	$E(W, Y^i, X^i)$	none
perceptron	$E(W, Y^i, X^i) - \min_{Y \in \mathcal{Y}} E(W, Y, X^i)$	0
hinge	$\max\left(0, m + E(W, Y^i, X^i) - E(W, \bar{Y}^i, X^i)\right)$	m
log	$\log\left(1 + e^{E(W,Y^i,X^i) - E(W,\bar{Y}^i,X^i)}\right)$	> 0
LVQ2	$\min\left(M, \max(0, E(W, Y^i, X^i) - E(W, \bar{Y}^i, X^i)\right)$	0
MCE	$\left(1 + e^{-\left(E(W,Y^i,X^i) - E(W,\bar{Y}^i,X^i)\right)}\right)^{-1}$	> 0
square-square	$E(W, Y^i, X^i)^2 - \left(\max(0, m - E(W, \bar{Y}^i, X^i))\right)^2$	m
square-exp	$E(W, Y^i, X^i)^2 + \beta e^{-E(W,\bar{Y}^i,X^i)}$	> 0
NLL/MMI	$E(W, Y^i, X^i) + \frac{1}{\beta} \log \int_{y \in \mathcal{Y}} e^{-\beta E(W,y,X^i)}$	> 0
MEE	$1 - e^{-\beta E(W,Y^i,X^i)} / \int_{y \in \mathcal{Y}} e^{-\beta E(W,y,X^i)}$	> 0

Advantages/Disadvantages of various losses

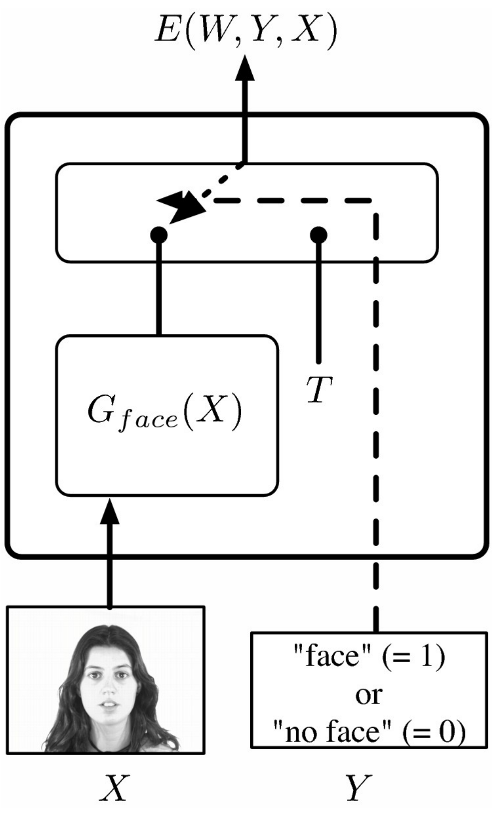
- **Loss functions differ in how they pick the point(s) whose energy is pulled up, and how much they pull them up**
- **Losses with a log partition function in the contrastive term pull up all the bad answers simultaneously.**
  - This may be good if the gradient of the contrastive term can be computed efficiently
  - This may be bad if it cannot, in which case we might as well use a loss with a single point in the contrastive term
- **Variational methods pull up many points, but not as many as with the full log partition function.**
- **Efficiency of a loss/architecture: how many energies are pulled up for a given amount of computation?**
  - The theory for this is to be developed

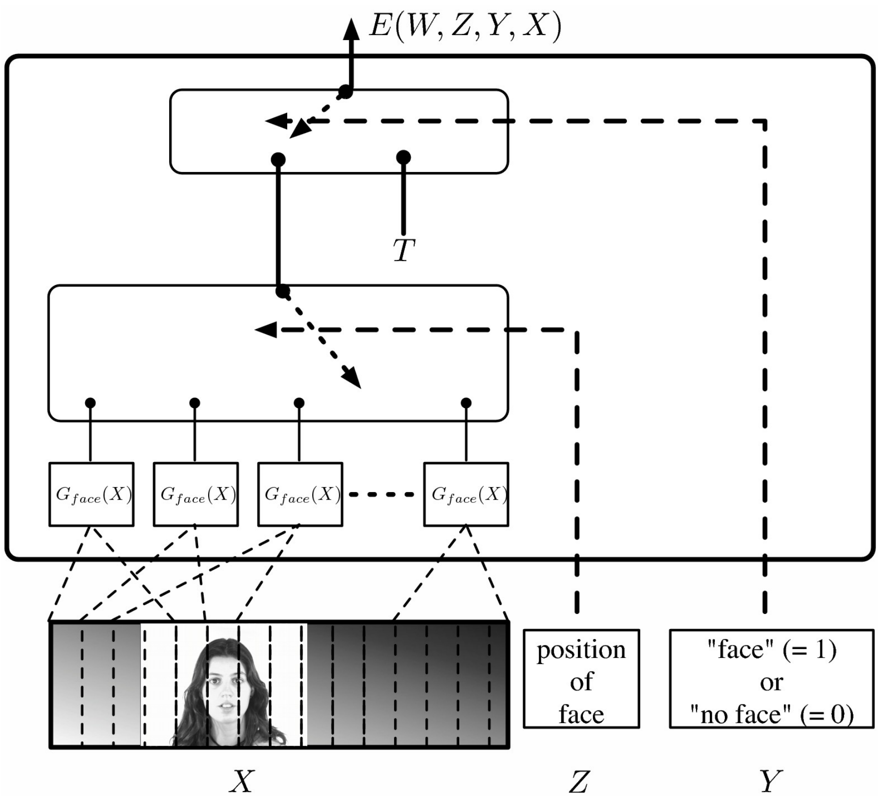
Latent Variable Models

- **The energy includes "hidden" variables Z whose value is never given to us**

$$E(Y, X) = \min_{Z \in \mathcal{Z}} E(Z, Y, X).$$

$$Y^* = \text{argmin}_{Y \in \mathcal{Y}, Z \in \mathcal{Z}} E(Z, Y, X).$$





What can the latent variables represent?

- **Variables that would make the task easier if they were known:**
  - **Face recognition**: the gender of the person, the orientation of the face.
  - **Object recognition**: the pose parameters of the object (location, orientation, scale), the lighting conditions.
  - **Parts of Speech Tagging**: the segmentation of the sentence into syntactic units, the parse tree.
  - **Speech Recognition**: the segmentation of the sentence into phonemes or phones.
  - **Handwriting Recognition**: the segmentation of the line into characters.
- **In general, we will search for the value of the latent variable that allows us to get an answer (Y) of smallest energy.**

Probabilistic Latent Variable Models

- **Marginalizing over latent variables instead of minimizing.**

$$P(Z,Y|X) = \frac{e^{-\beta E(Z,Y,X)}}{\int_{y\in\mathcal{Y},\ z\in\mathcal{Z}} e^{-\beta E(y,z,X)}}.$$

$$P(Y|X) = \frac{\int_{z\in\mathcal{Z}} e^{-\beta E(Z,Y,X)}}{\int_{y\in\mathcal{Y},\ z\in\mathcal{Z}} e^{-\beta E(y,z,X)}}.$$

- **Equivalent to traditional energy-based inference with a redefined energy function:**

$$Y^* = \text{argmin}_{Y\in\mathcal{Y}} - \frac{1}{\beta} \log \int_{z\in\mathcal{Z}} e^{-\beta E(z,Y,X)}.$$

- **Reduces to traditional minimization when Beta->infinity**

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

- 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

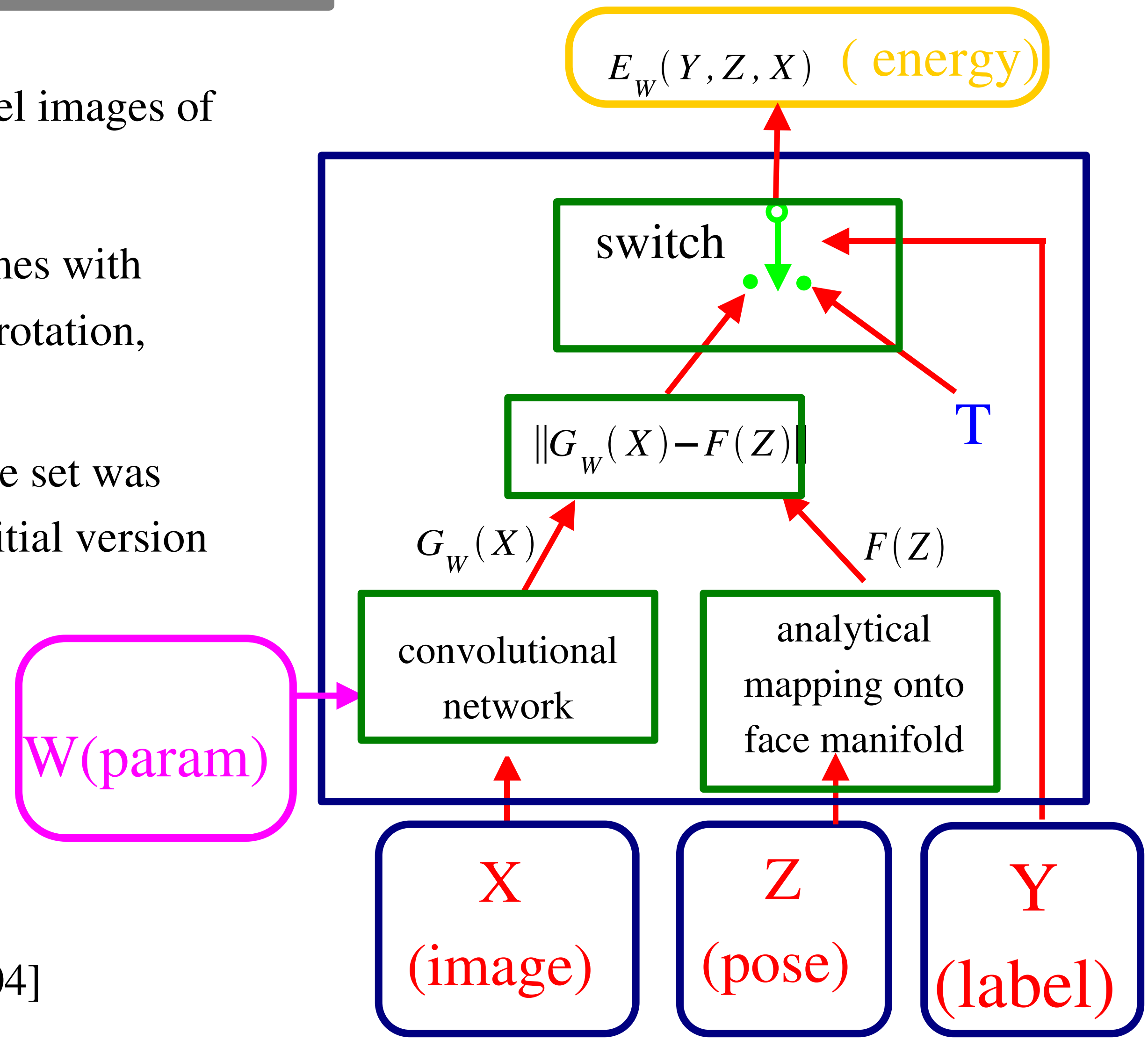
$$E^*(W,X) = \min_Z ||G_W(X) - F(Z)||$$

$$Z^* = \text{argmin}_Z ||G_W(X) - F(Z)||$$

- **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 .

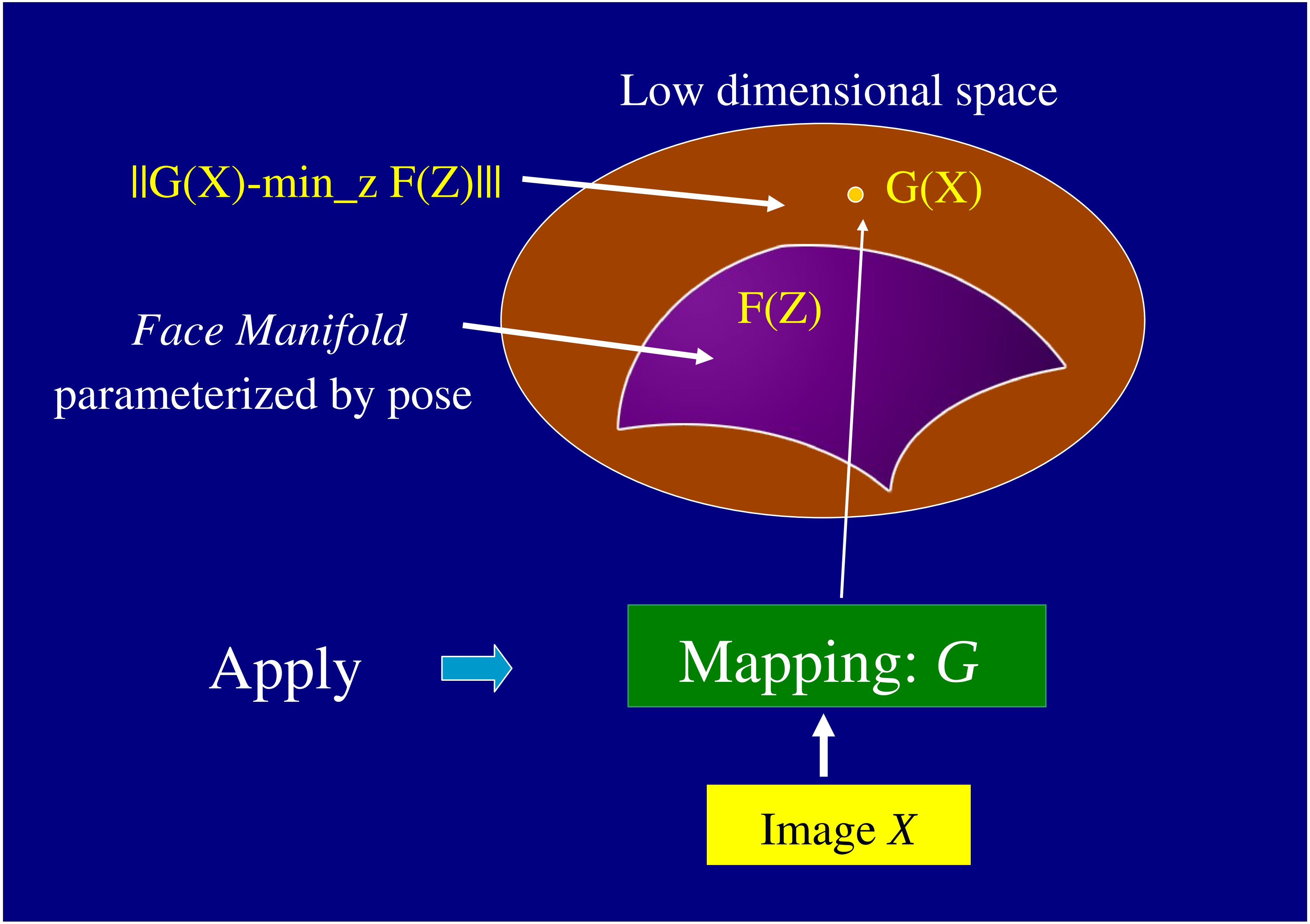
Small E*(W,X): face

Large E*(W,X): no face



[Osadchy, Miller, LeCun, NIPS 2004]

Face Manifold



Probabilistic Approach: Density model of joint P(face,pose)

Probability that image X is a face with pose Z

$$P(X,Z) = \frac{\exp(-E(W,Z,X))}{\int_{X,Z \in \text{images,poses}} \exp(-E(W,Z,X))}$$

Given a training set of faces annotated with pose, find the W that maximizes the likelihood of the data under the model:

$$P(\text{faces} + \text{pose}) = \prod_{X,Z \in \text{faces+pose}} \frac{\exp(-E(W,Z,X))}{\int_{X,Z \in \text{images,poses}} \exp(-E(W,Z,X))}$$

Equivalently, minimize the negative log likelihood:

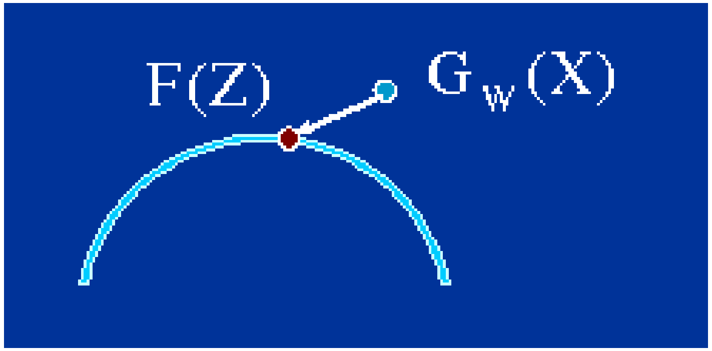
$$\mathcal{L}(W, \text{faces} + \text{pose}) = \sum_{X,Z \in \text{faces+pose}} E(W,Z,X) + \log \left[\int_{X,Z \in \text{images,poses}} \exp(-E(W,Z,X)) \right]$$

COMPLICATED

Energy-Based Contrastive Loss Function

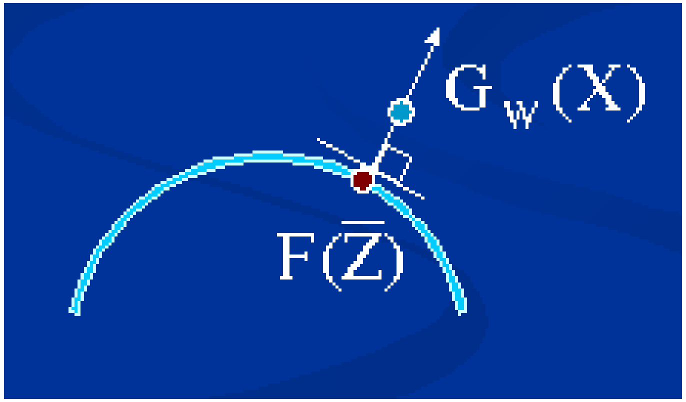
$$\mathcal{L}(W) = \frac{1}{|\mathrm{f+p}|} \sum_{X,Z \in \mathrm{faces+pose}} \left[L^+\left(E(W,Z,X)\right)\right] + L^-\left(\min_{X,Z \in \mathrm{bckgnd,poses}} E(W,Z,X)\right)$$

$$L^+\left(E(W,Z,X)\right) = E(W,Z,X)^2 = ||G_W(X) - F(Z)||^2$$



Attract the network output Gw(X) to the location of the desired pose F(Z) on the manifold

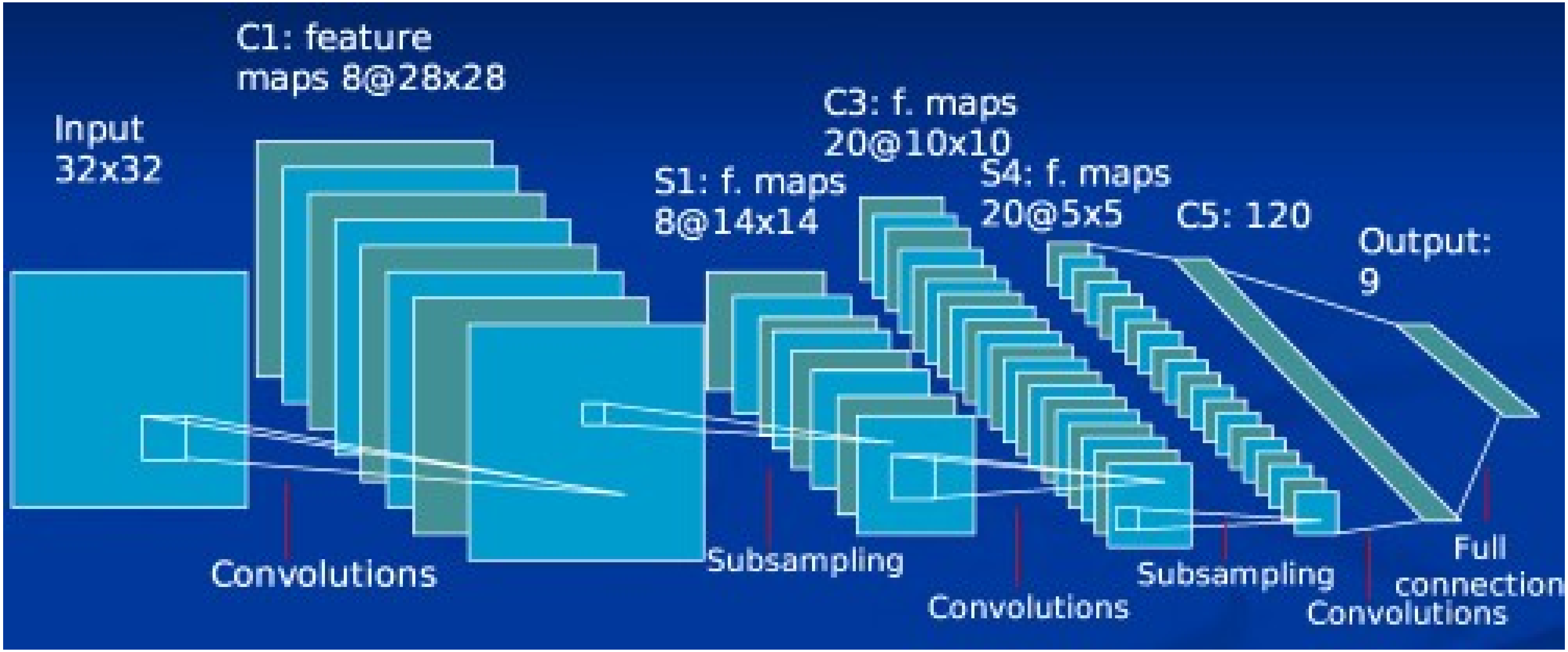
$$L^-\left(\min_{X,Z \in \mathrm{bckgnd,poses}} E(W,Z,X)\right) = K \exp\left(-\min_{X,Z \in \mathrm{bckgnd,poses}} ||G_W(X) - F(Z)||\right)$$



Repel the network output Gw(X) away from the face/pose manifold

Convolutional Network Architecture

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



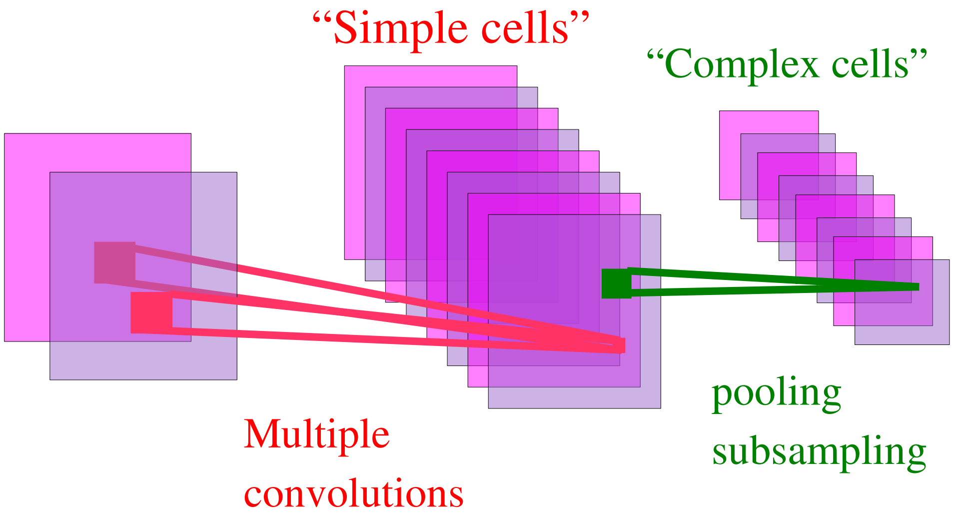
Hierarchy of local filters (convolution kernels),

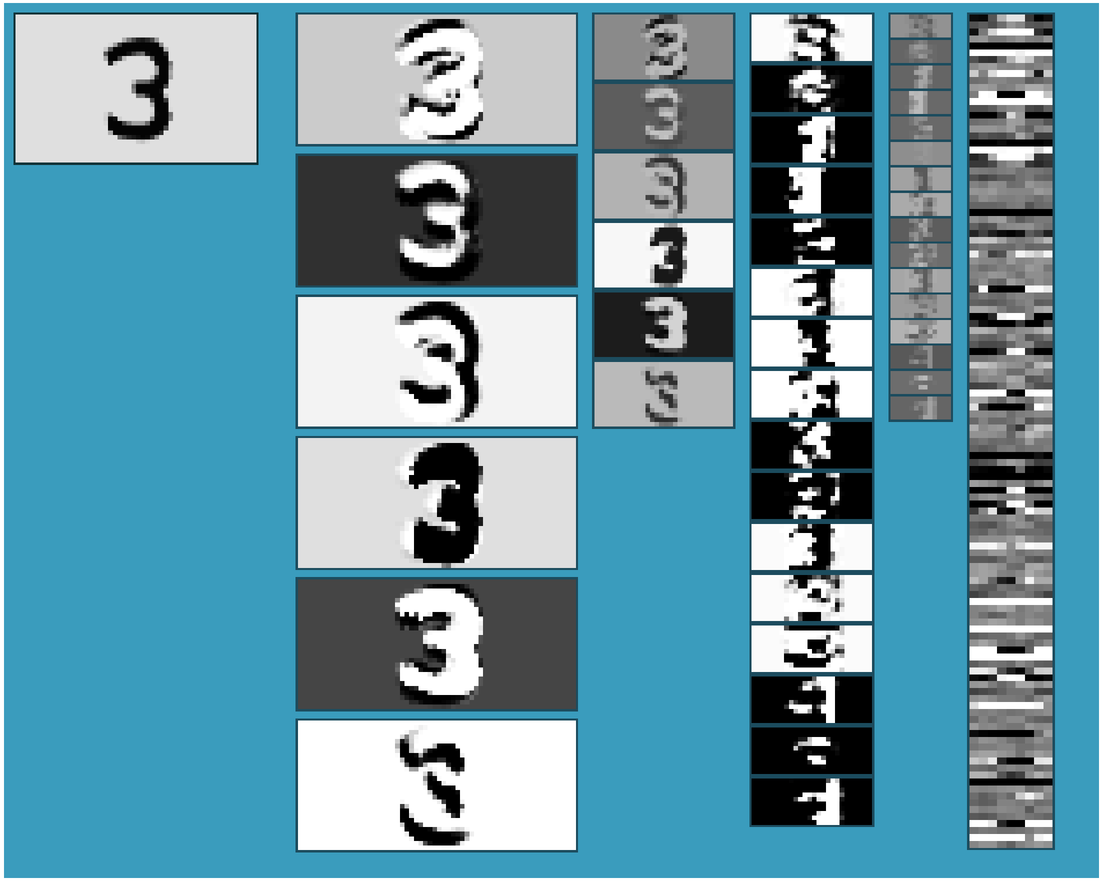
sigmoid pointwise non-linearities, and spatial subsampling

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

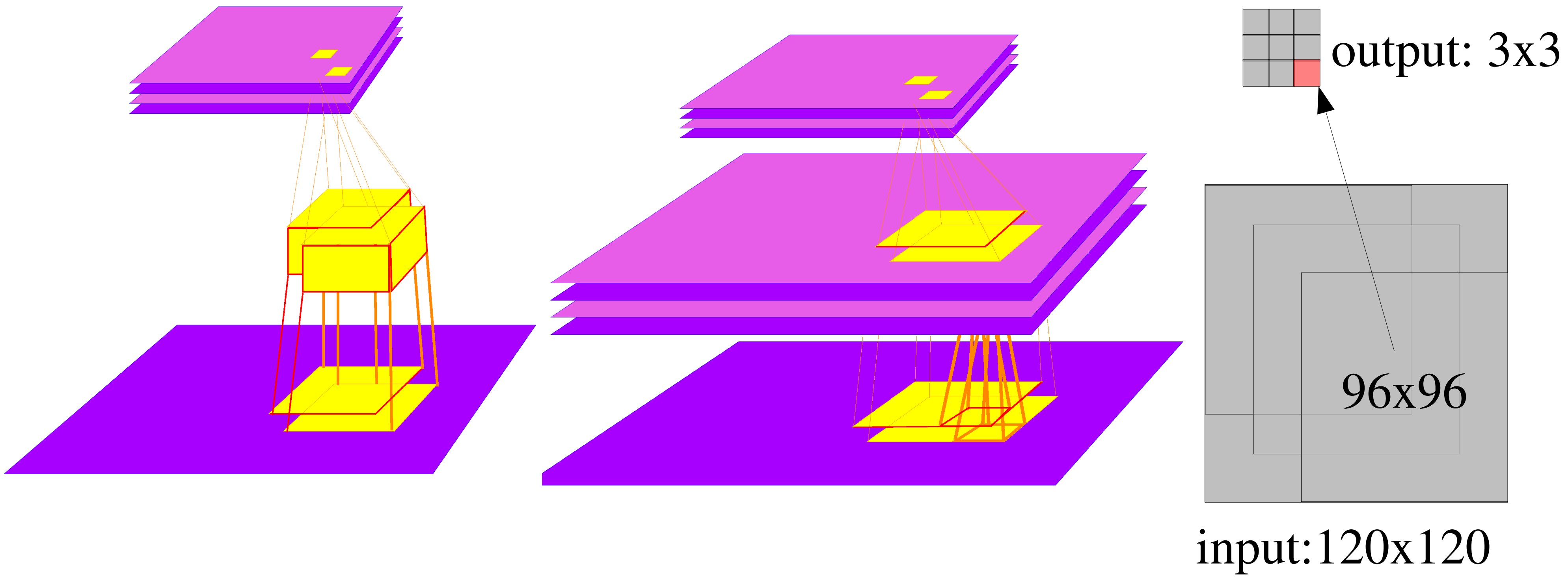
Alternated Convolutions and Pooling/Subsampling

- 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,....





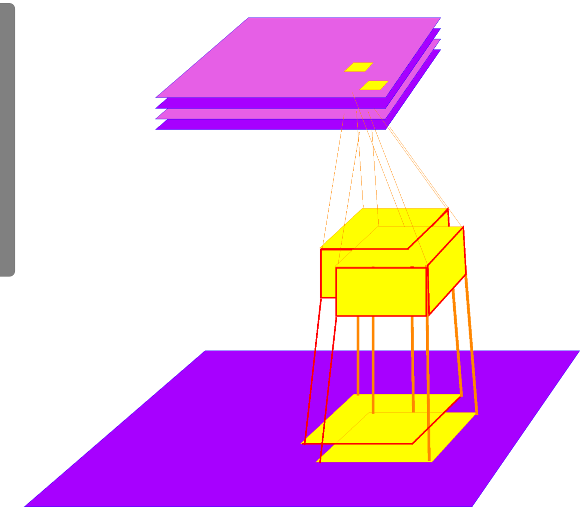
Building a Detector/Recognizer: Replicated Conv. Nets

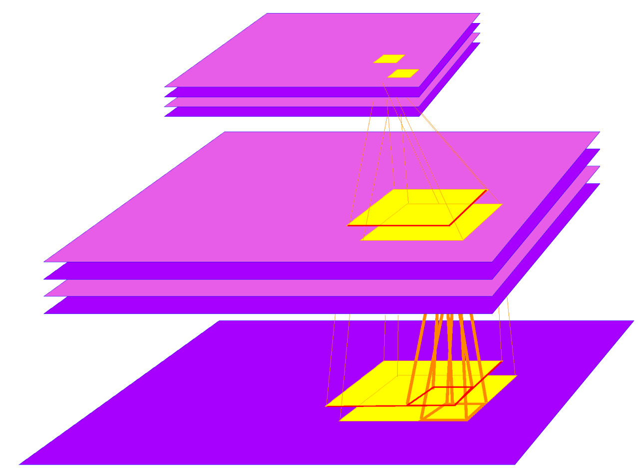


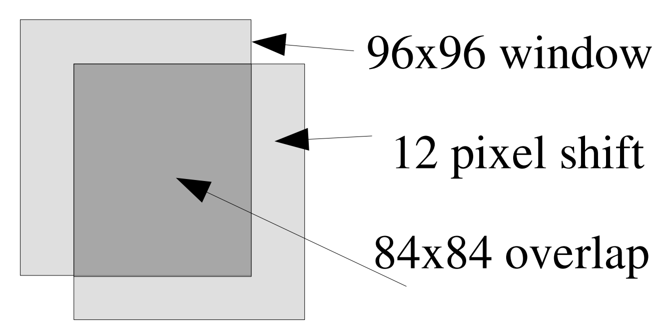
- Traditional Detectors/Classifiers must be applied to every location on a large input image, at multiple scales.
- Convolutional nets can replicated over large images very cheaply.
- The network is applied to multiple scales spaced by 1.5.

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

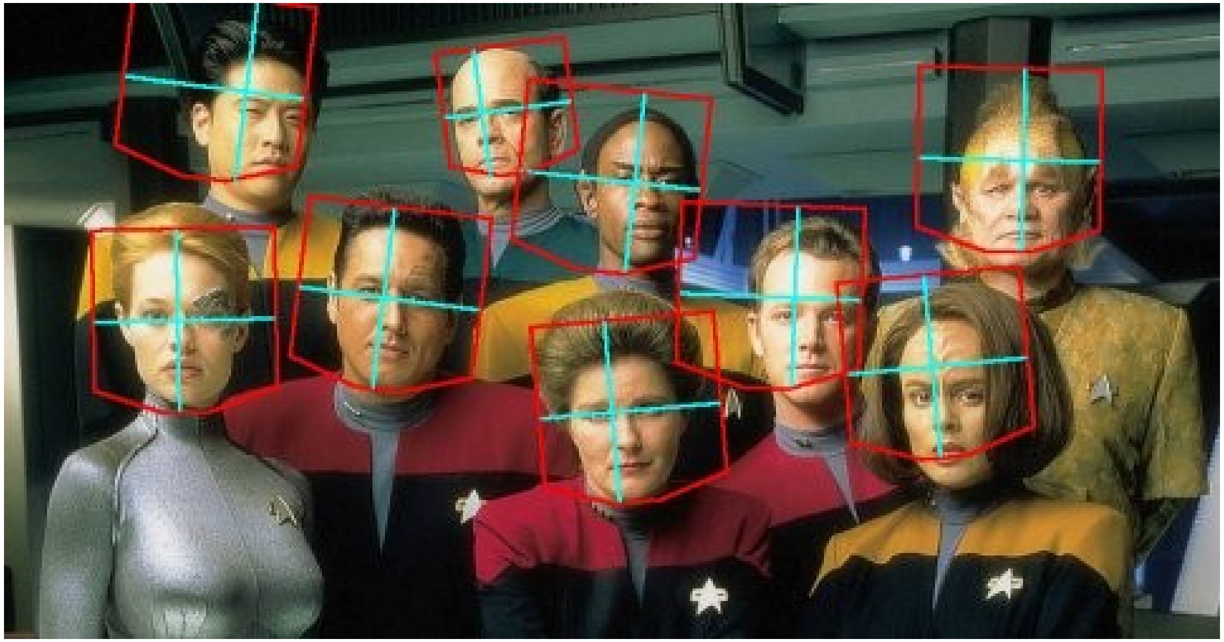


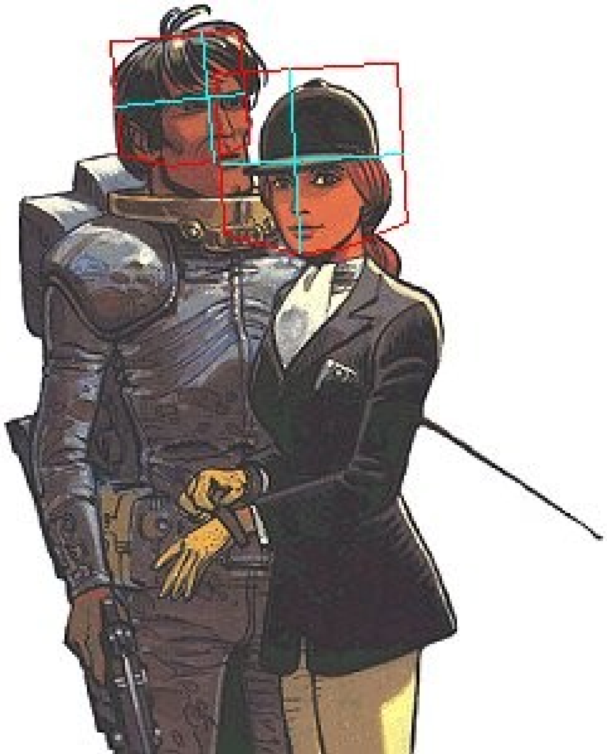




Face Detection: Results

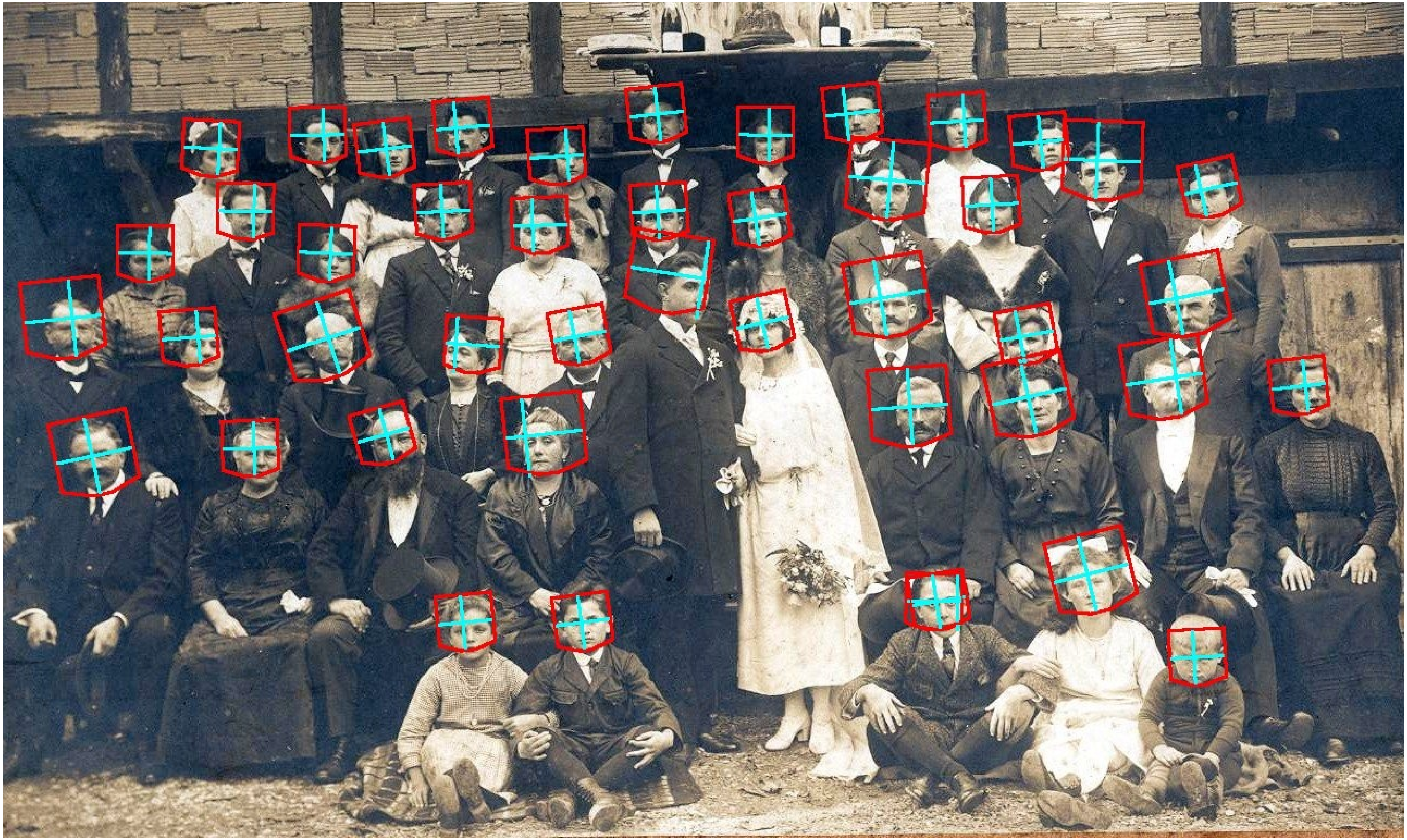
Data Set->	TILTED		PROFILE		MIT+CMU	
False positives per image->	4.42	26.9	0.47	3.36	0.5	1.28
Our Detector	90%	97%	67%	83%	83%	88%
Jones & Viola (tilted)	90%	95%	x		x	
Jones & Viola (profile)	x		70%	83%	x	





Face Detection and Pose Estimation: Results

Face Detection with a Convolutional Net



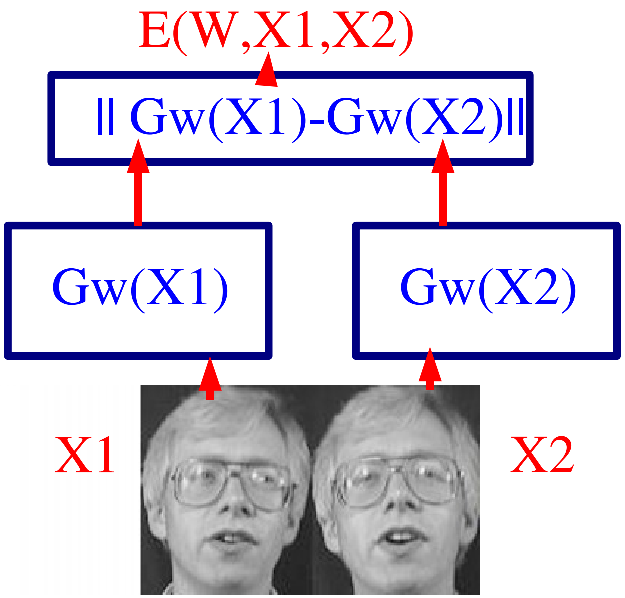
How do we Handle Lots of Classes?

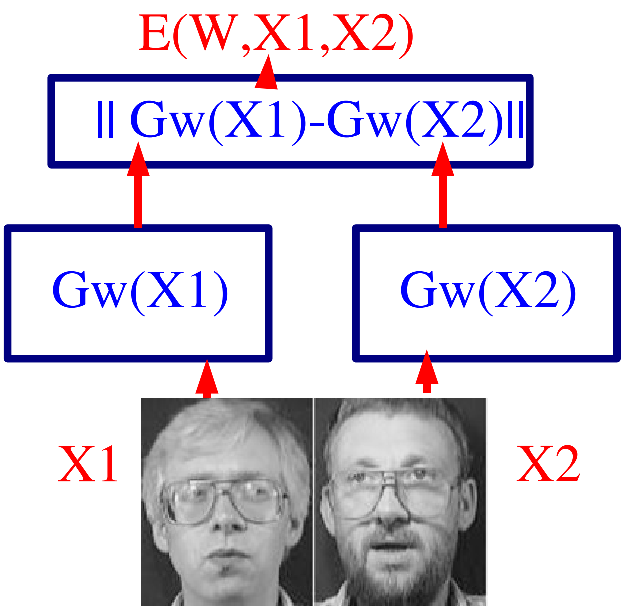
- **Example: face recognition**
  - We do not have pictures of every person
- **We must be able to learn something without seeing all the classes**
- **Solution: learn a similarity metric**
- **Map images to a low dimensional space in which**
  - Two images of the same person are mapped to nearby points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

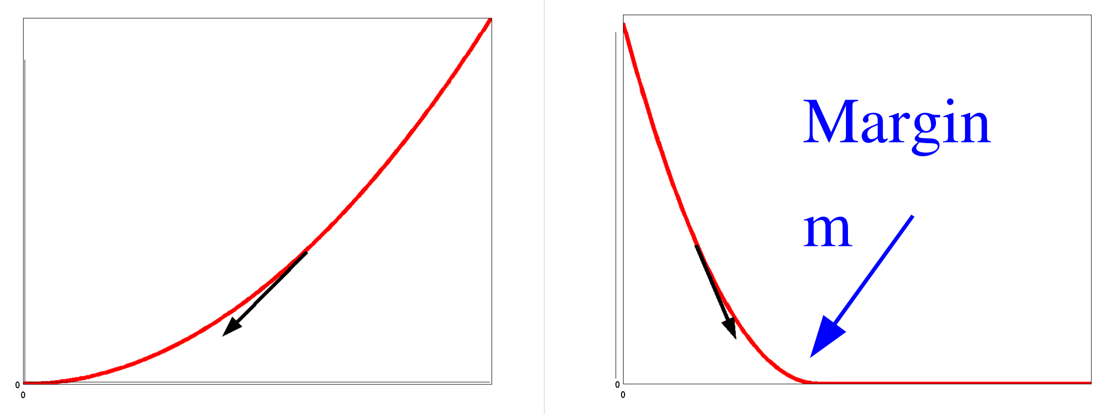
energy: 10.3209

energy: 2.8243

A similar idea for Learning a Manifold with Invariance Properties

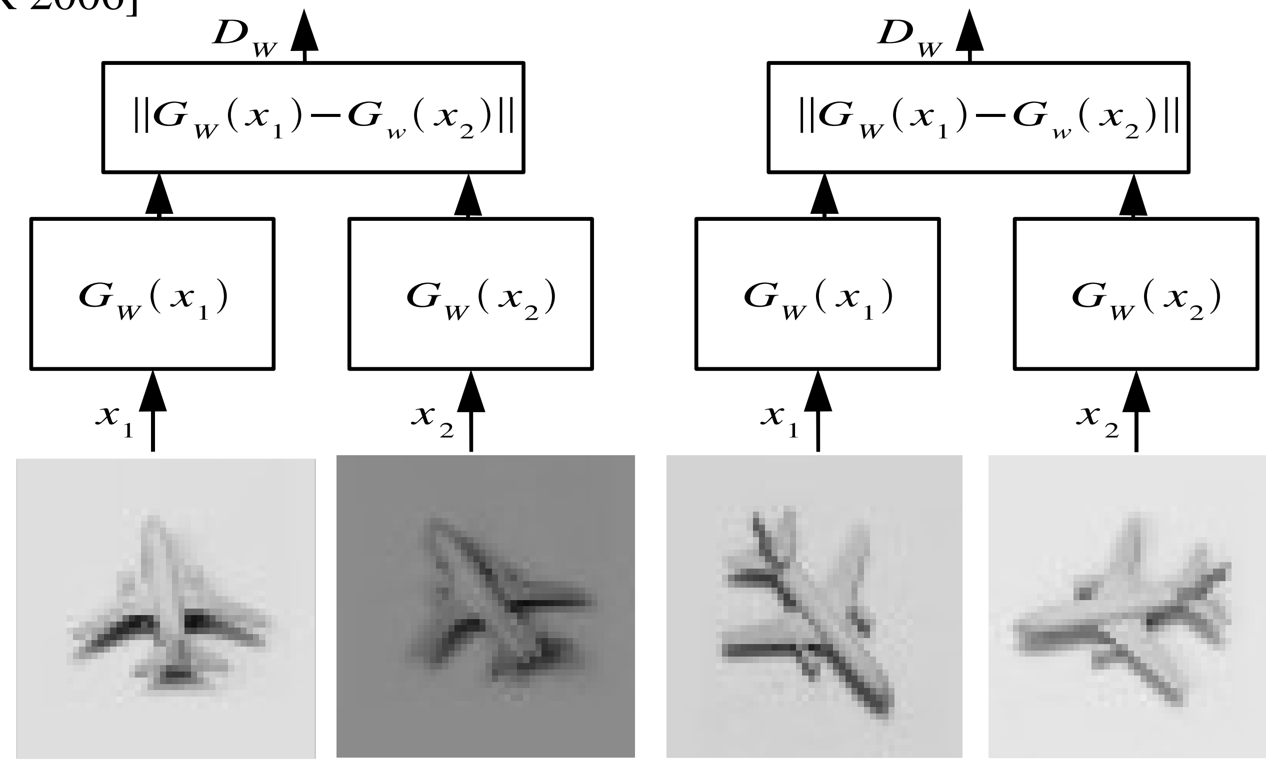
[Hadsell, Chopra, LeCun, CVPR 2006]

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

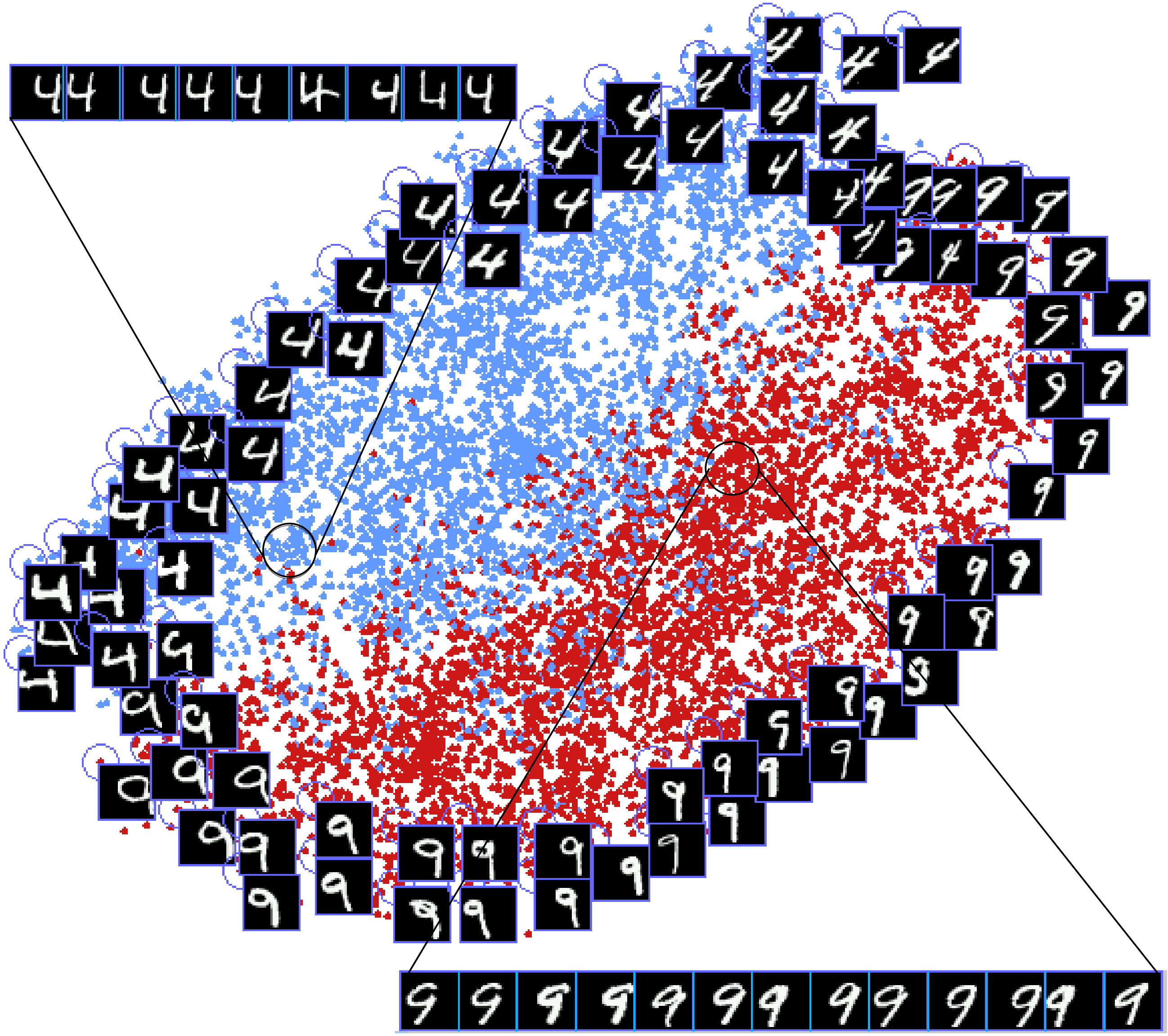


Loss function:

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

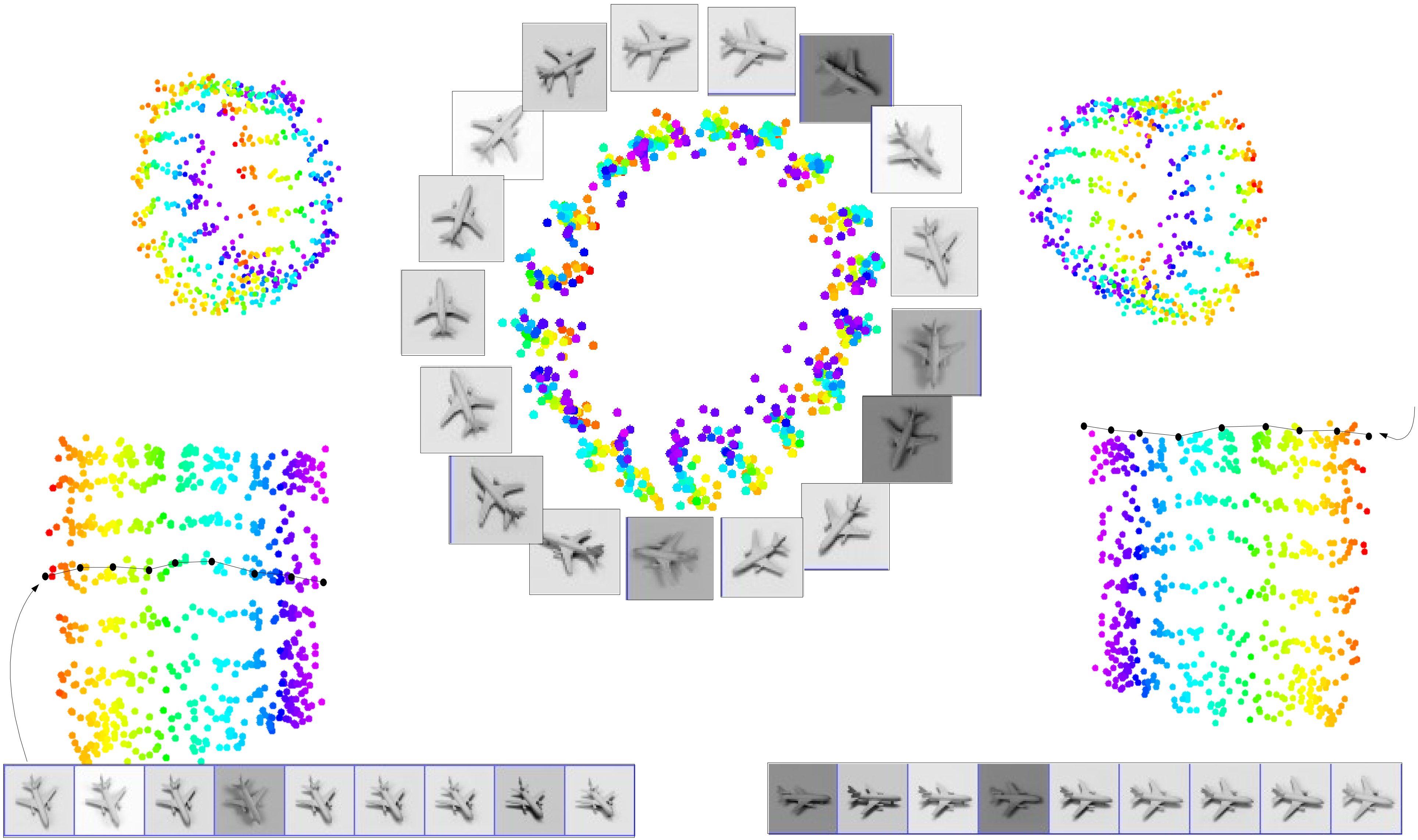


A Manifold with Invariance to Shifts



- 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

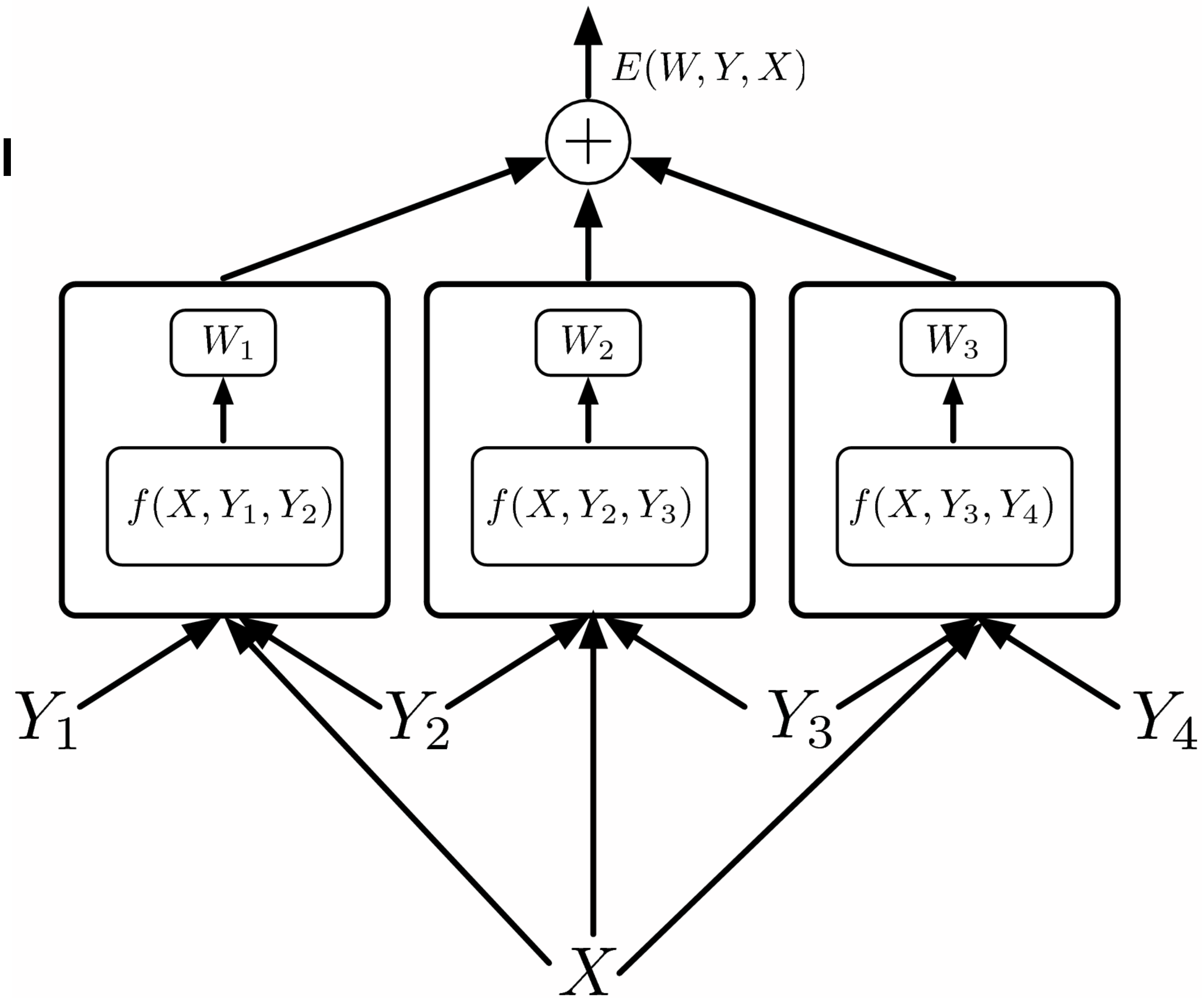


Non-Probabilistic Graphical Models: Energy-Based Factor Graphs

- Graphical models have brought us efficient inference algorithms, such as belief propagation and its numerous variations.
- Traditionally, graphical models are viewed as probabilistic models
- At first glance, is seems difficult to dissociate graphical models from the probabilistic view
- Energy-Based Factor Graphs are an extension of graphical models to non-probabilistic settings.
- An EBFG is an energy function that can be written as a sum of “factor” functions that take different subsets of variables as inputs.

Example of EBFG: Shallow Factors / Deep Graph

- **Linearly Parameterized Factors**
- **with the NLL Loss :**
  - Lafferty's **Conditional Random Field**
- **with Hinge Loss:**
  - Taskar's **Max Margin Markov Nets**
- **with Perceptron Loss**
  - Collins's sequence labeling model
- **With Log Loss:**
  - Altun/Hofmann sequence labeling model



Deep Factors / Deep Graph: ASR with TDNN/DTW

- Trainable Speech/Handwriting Recognition systems that integrate Neural Nets (or other "deep" classifiers) with dynamic time warping, Hidden Markov Models, or other graph-based hypothesis representations

- Training the feature extractor as part of the whole process.

- with the LVQ2 Loss :
  - Driancourt and Bottou's speech recognizer (1991)

- with NLL:
  - Bengio's speech recognizer (1992)
  - Haffner's speech recognizer (1993)

- With Minimum Empirical Error loss
  - Ljolje and Rabiner (1990)

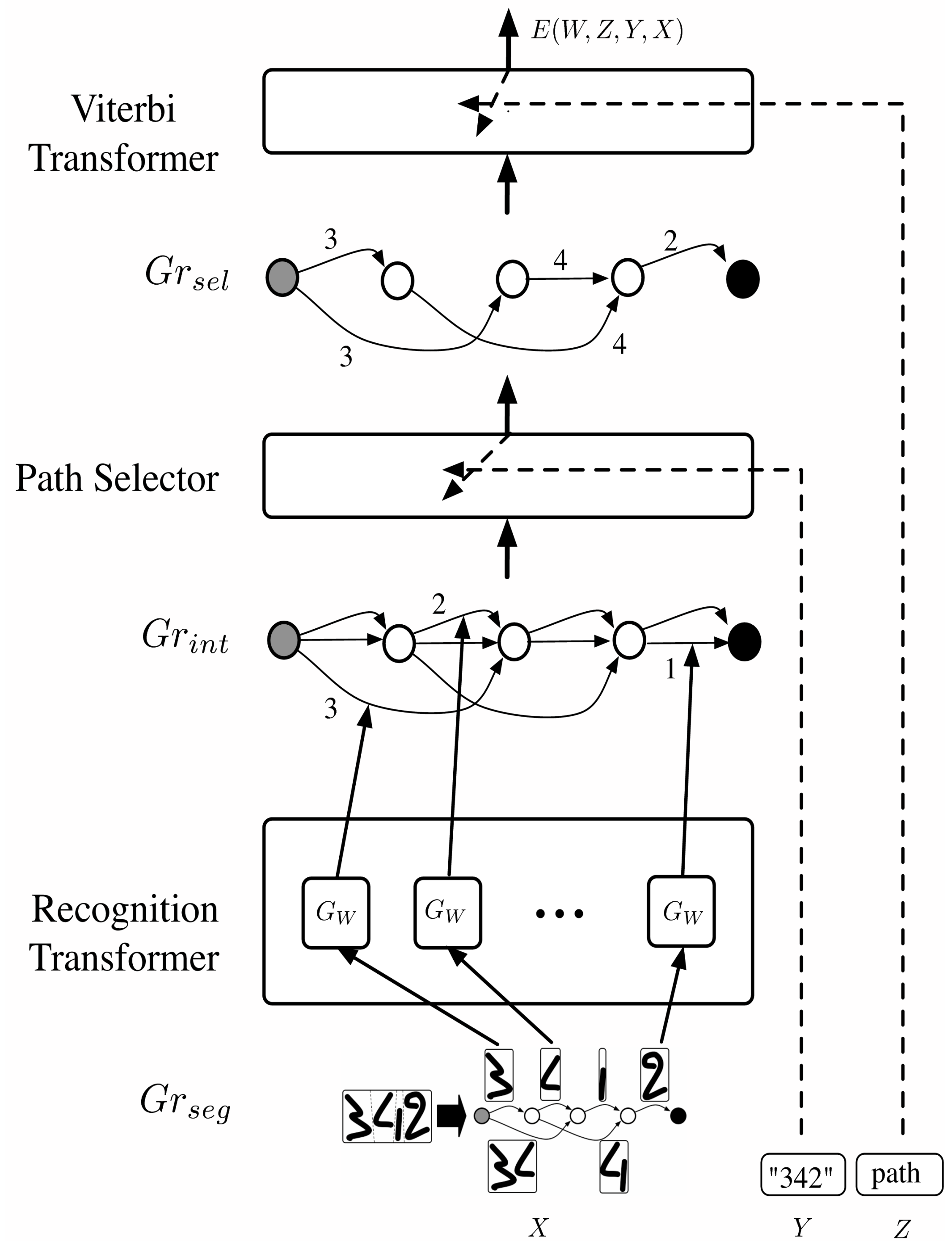
- with NLL:
  - Bengio (1992), Haffner (1993), Bourlard (1994)

- With MCE
  - Juang et al. (1997)

- Late normalization scheme (un-normalized HMM)
  - Bottou pointed out the **label bias problem** (1991)
  - Denker and Burges proposed a solution (1995)

Really Deep Factors / Really Deep Graph

- **Handwriting Recognition with Graph Transformer Networks**
- **Un-normalized hierarchical HMMs**
  - Trained with Perceptron loss [LeCun, Bottou, Bengio, Haffner 1998]
  - Trained with NLL loss [Bengio, LeCun 1994], [LeCun, Bottou, Bengio, Haffner 1998]
- **Answer = sequence of symbols**
- **Latent variable = segmentation**

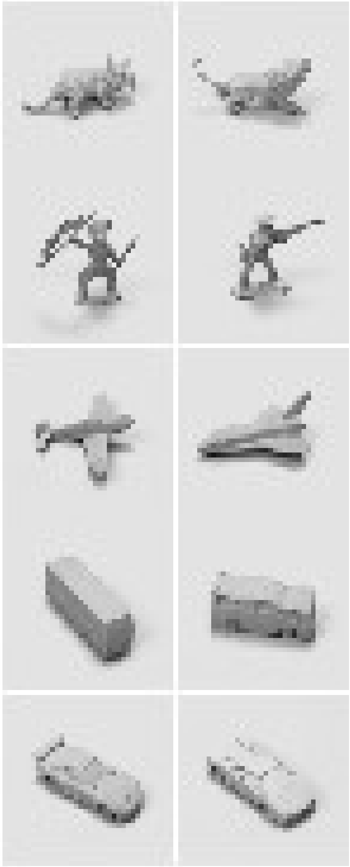


The “Deep Learning Problem”: Generic Object Detection and Recognition with Invariance to Pose, Illumination and Clutter

[Huang, LeCun, CVPR 2006, CVPR 2004]

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.**

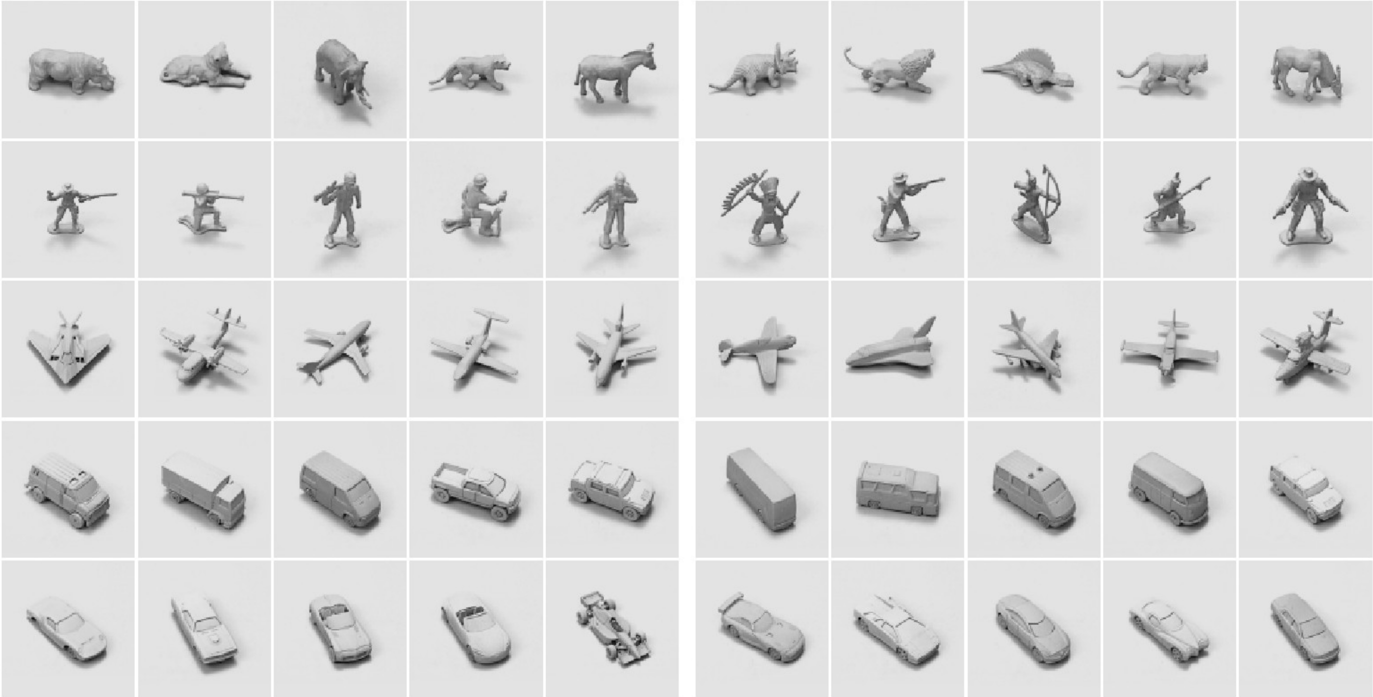


Why do we need "Deep" Architectures?

- **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

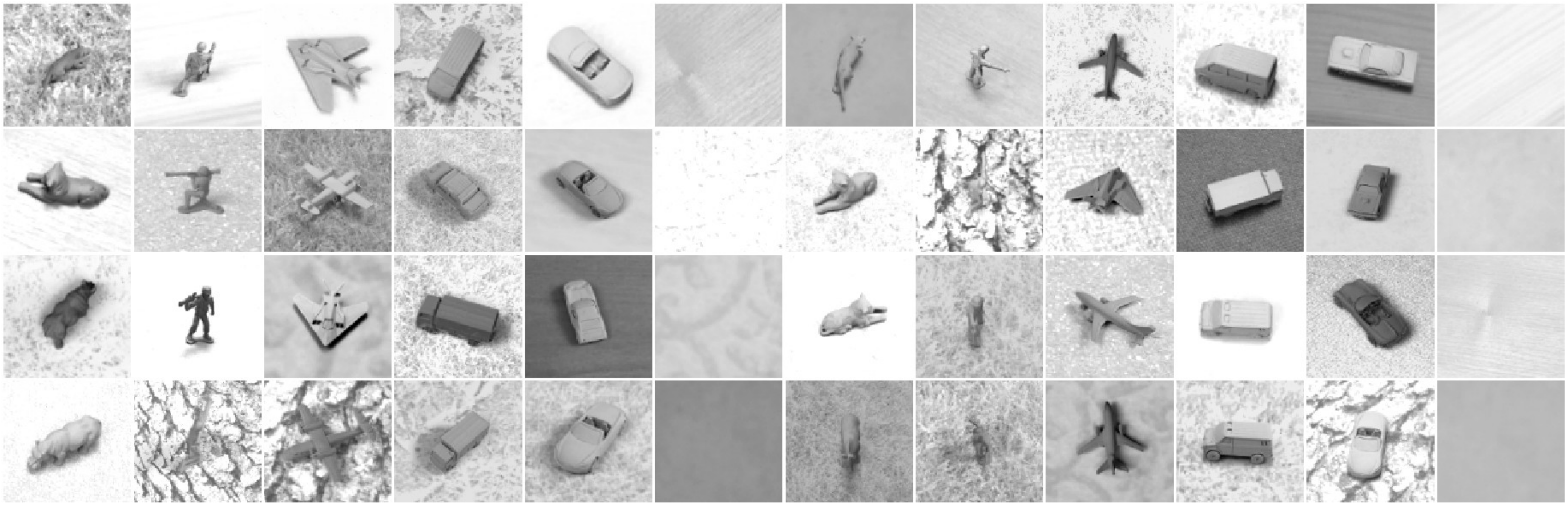
Generic Object Detection and Recognition with Invariance to Pose and Illumination

- **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

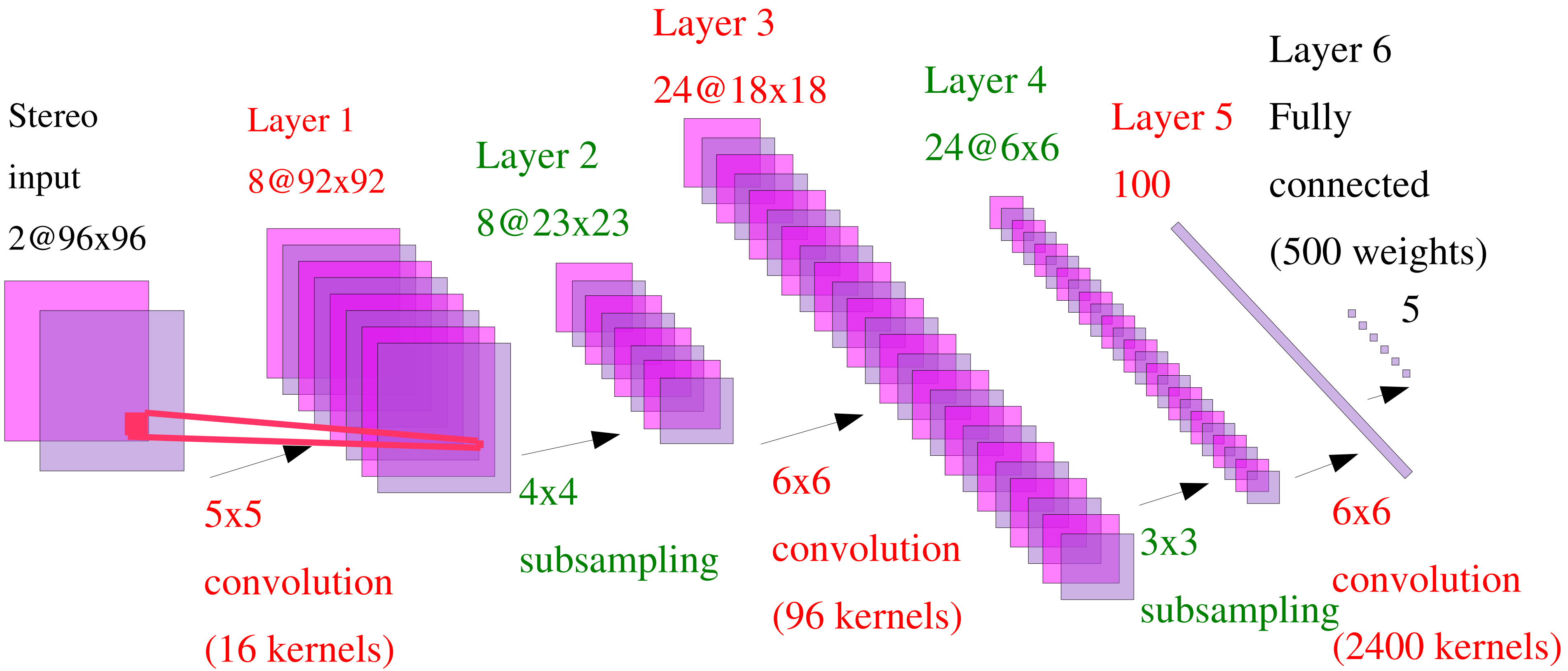


Training instances **Test instances**

Textured and Cluttered Datasets

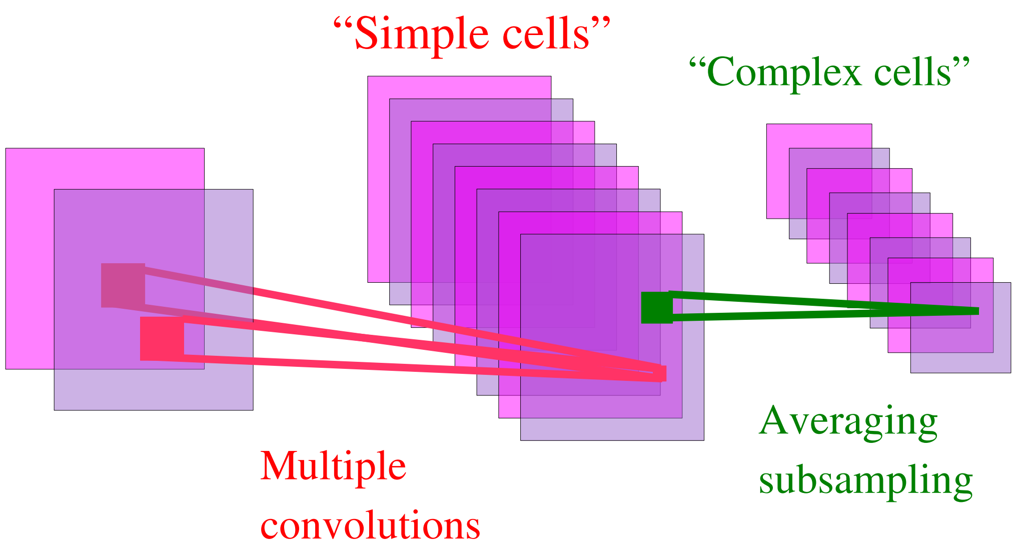


Convolutional Network

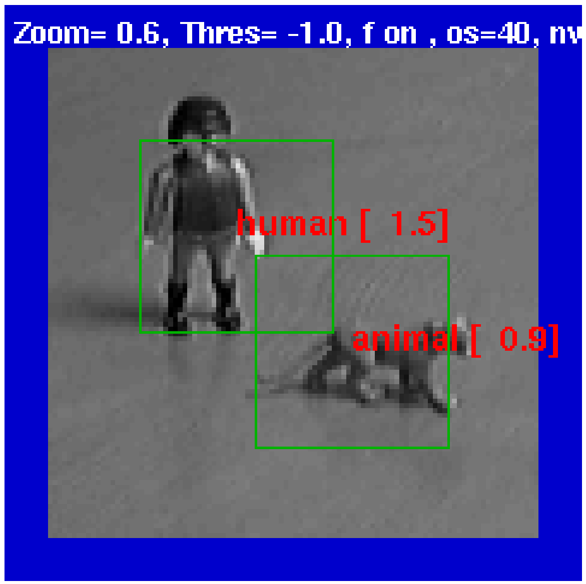


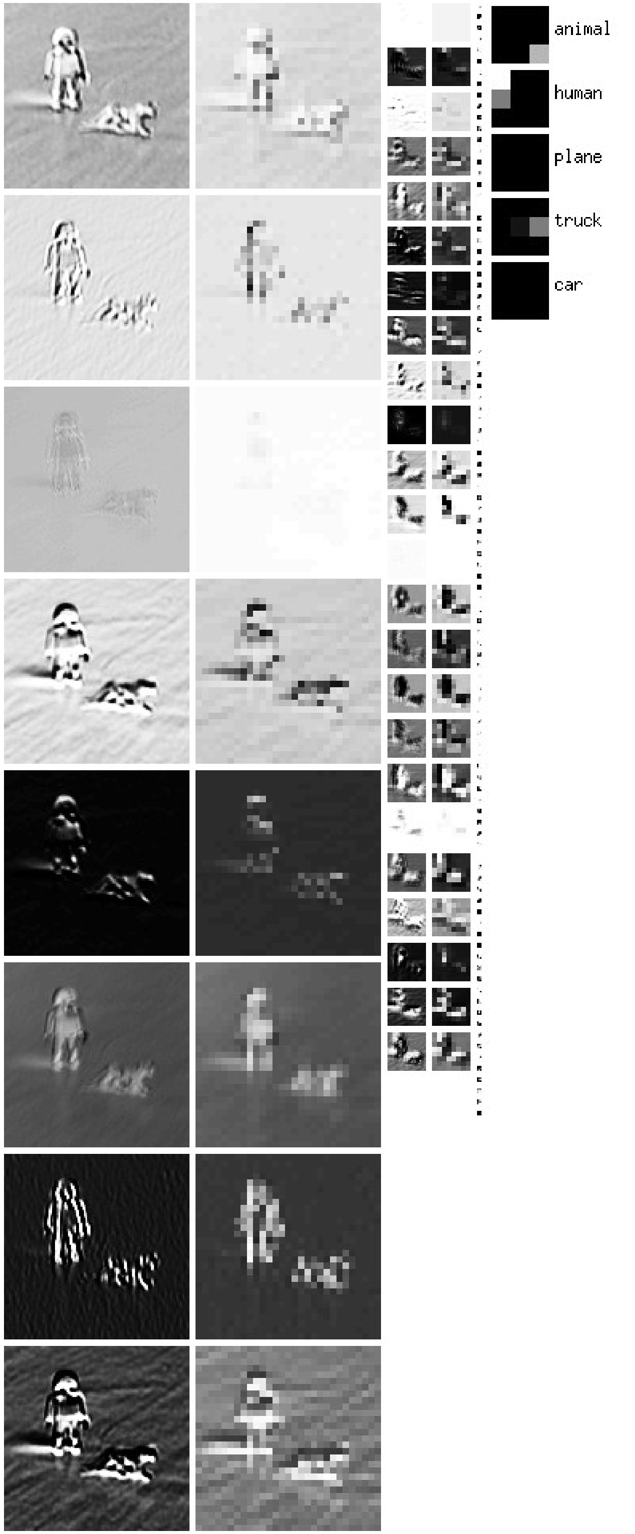
- **90,857 free parameters, 3,901,162 connections.**
- The architecture alternates convolutional layers (feature detectors) and subsampling layers (local feature pooling for invariance to small distortions).
- **The entire network is trained end-to-end** (all the layers are trained simultaneously).
-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

Alternated Convolutions and Subsampling



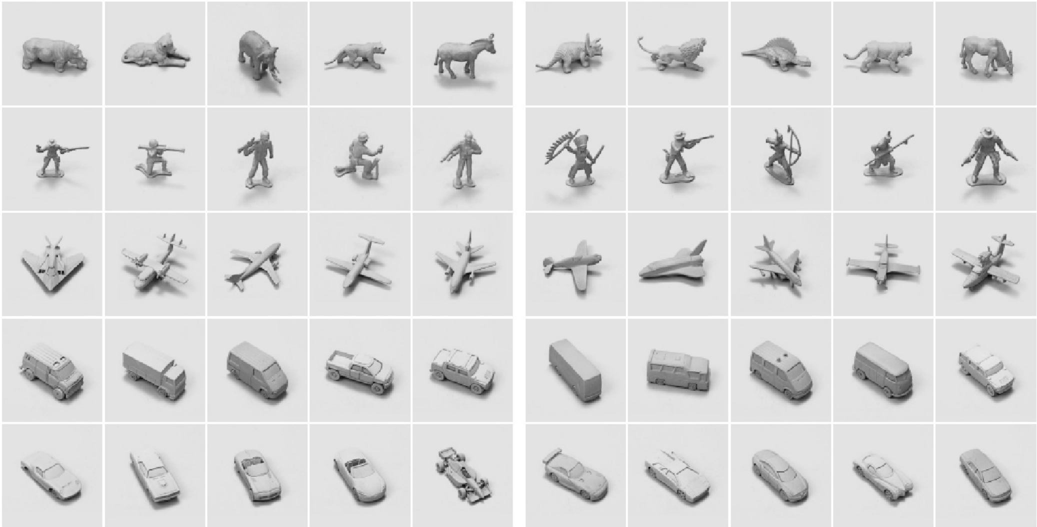
- Local features are extracted everywhere.
- averaging/subsampling layer builds robustness to variations in feature locations.
- Hubel/Wiesel'62, Fukushima'71, LeCun'89, Riesenhuber & Poggio'02, Ullman'02,....





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

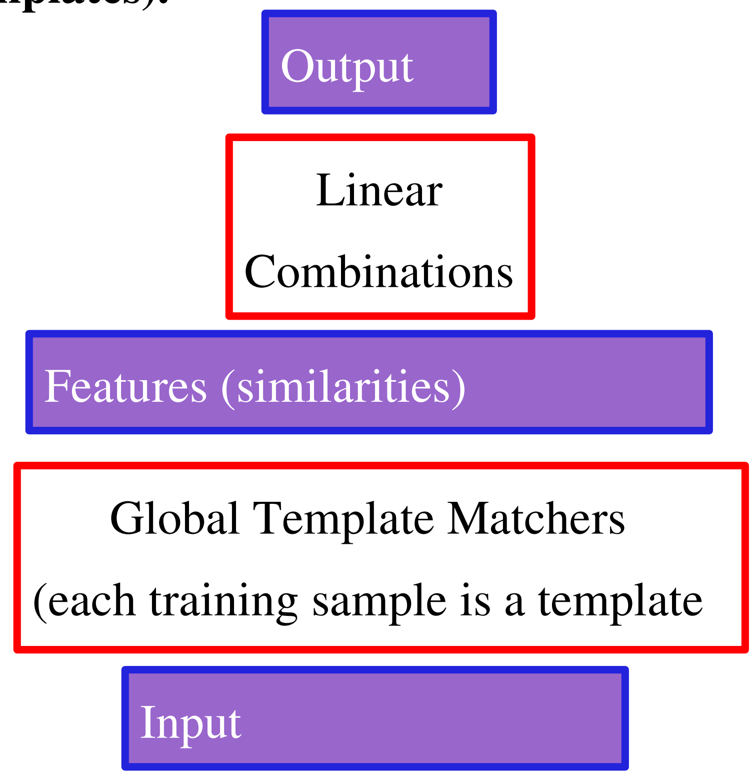
OUCH!

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

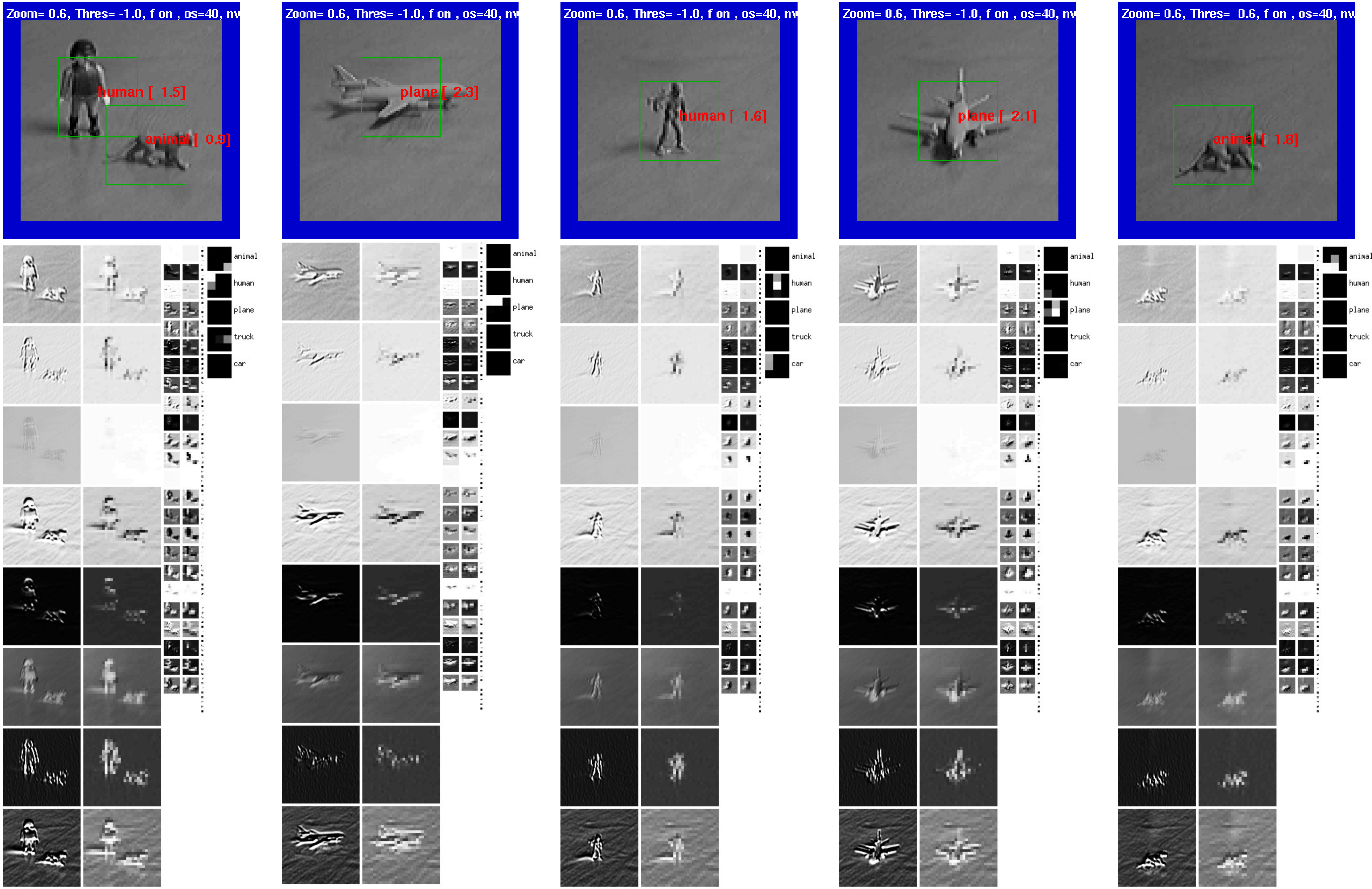
Chop off the last layer, and train an SVM on it it works!

What's wrong with K-NN and SVMs?

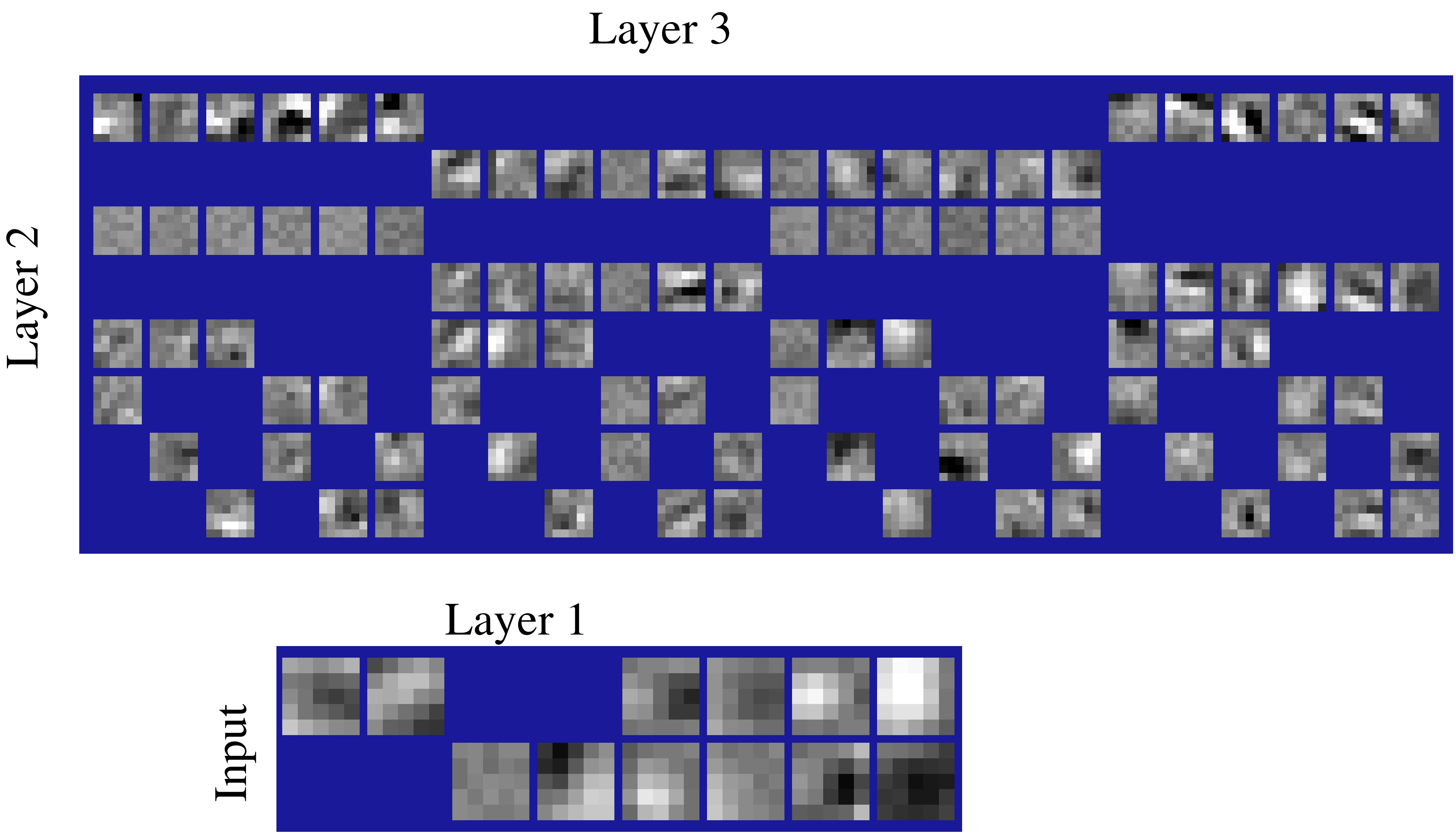
- **K-NN and SVM with Gaussian kernels are based on matching global templates**
- **Both are “shallow” architectures**
- **There is now way to learn invariant recognition tasks with such naïve architectures (unless we use an impractically large number of templates).**
  - **The number of necessary templates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



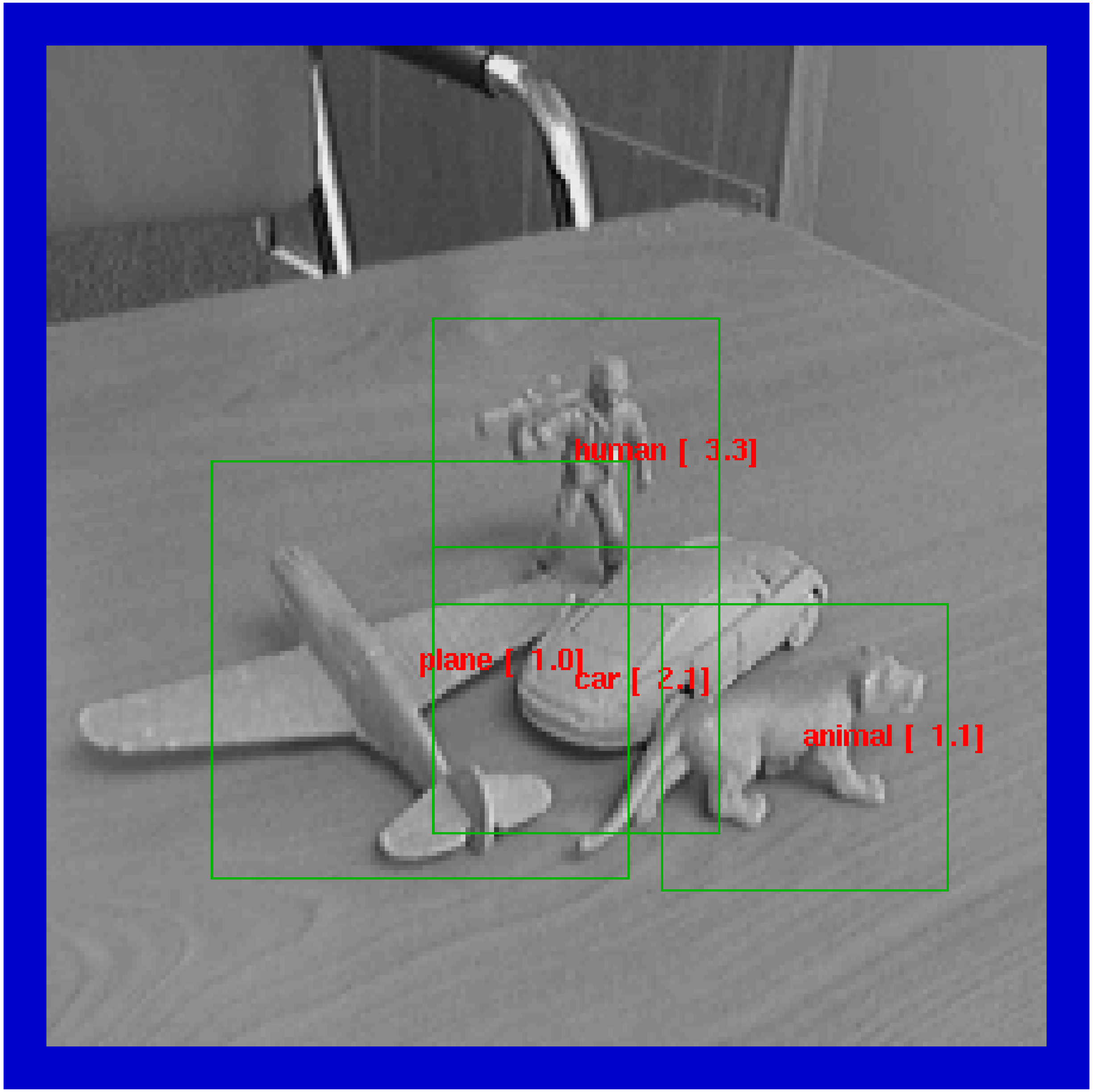
Examples (Monocular Mode)



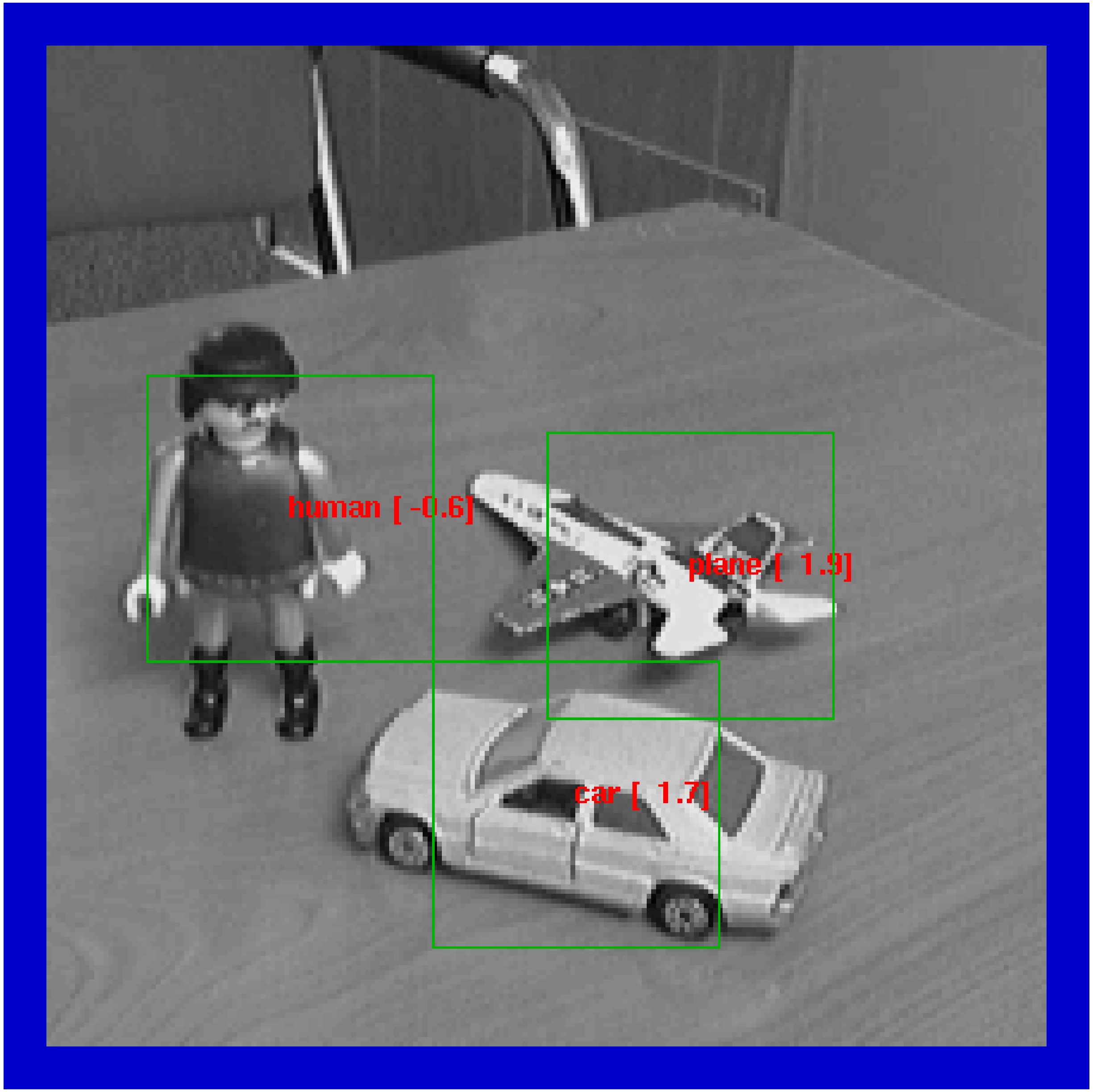
Learned Features



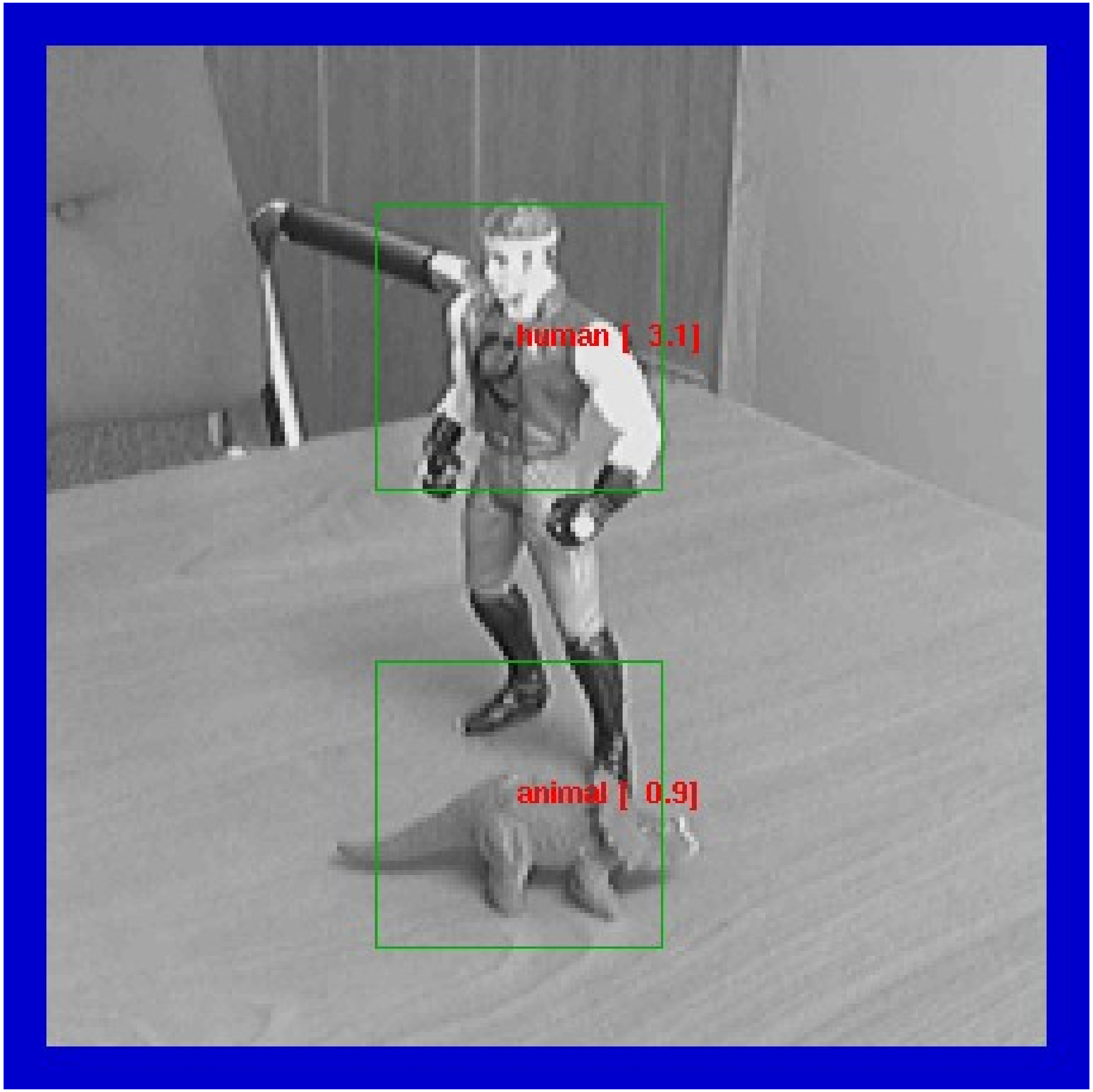
Examples (Monocular Mode)



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!**

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

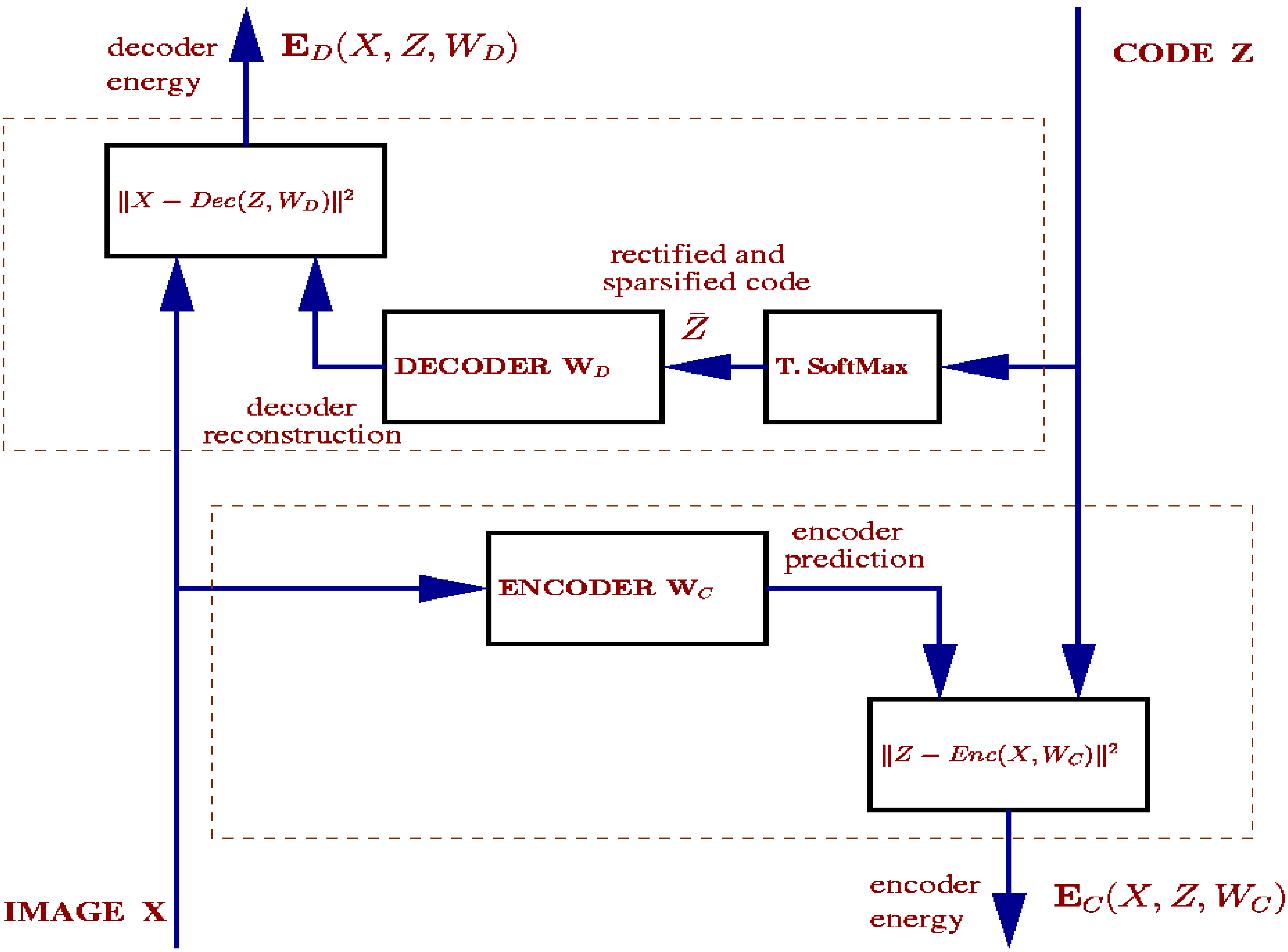
Unsupervised Learning of Sparse-Overcomplete Features

[Ranzato, Poultney, Chopra, LeCun, NIPS 2006]

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



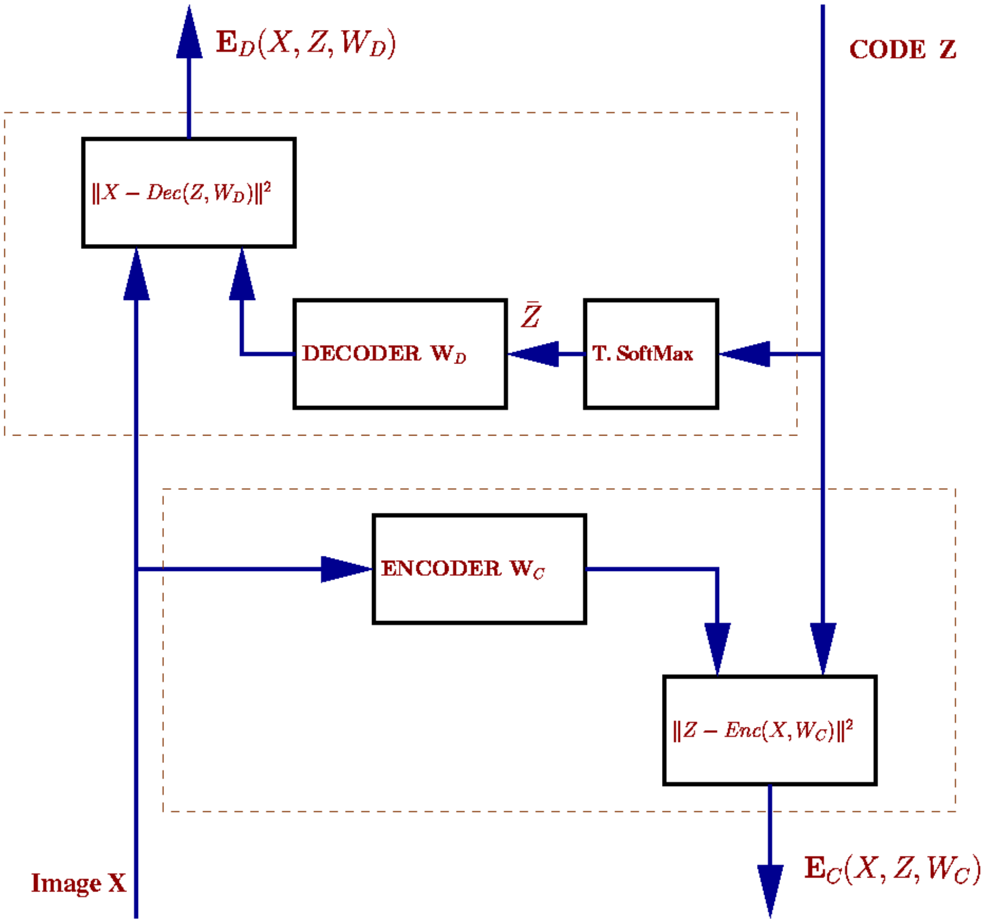
Symmetric Product of Experts

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

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

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

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



Inference & Learning

- *Inference*

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

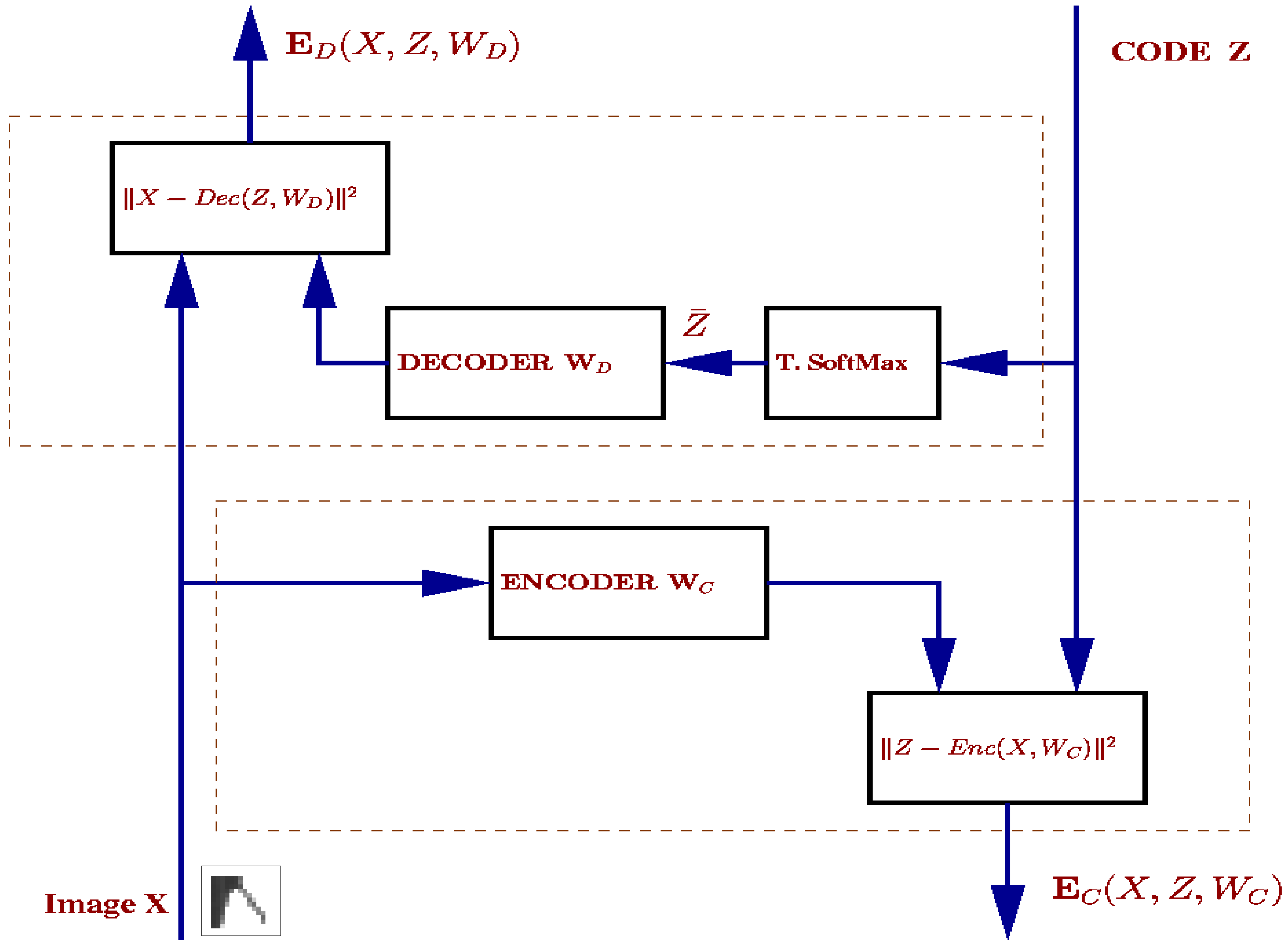
  - let Z(0) be the encoder prediction
  - find code which minimizes total energy
  - gradient descent optimization

- *Learning*

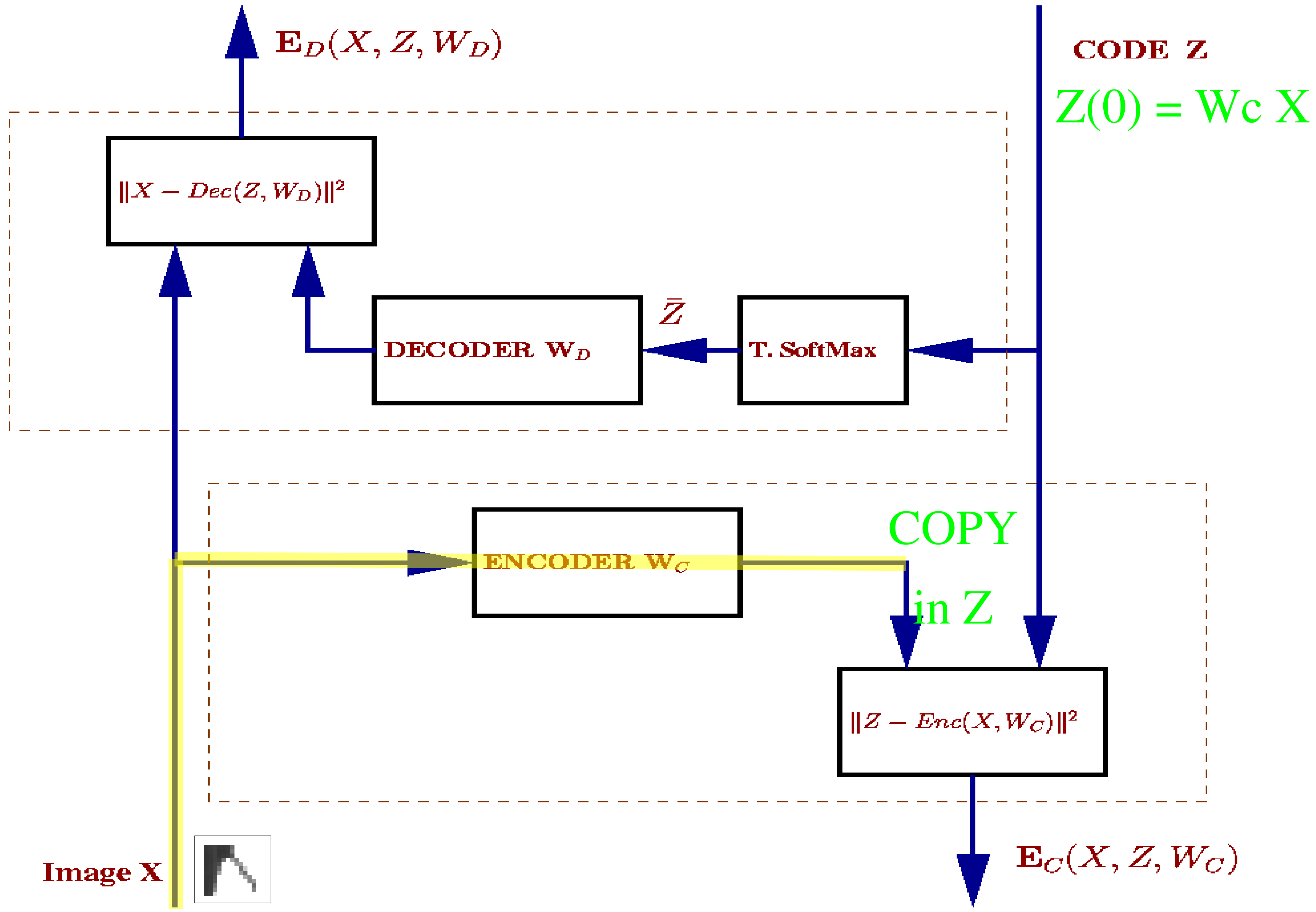
$$W \leftarrow W - \partial E(X, \tilde{Z}, W) / \partial W$$

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

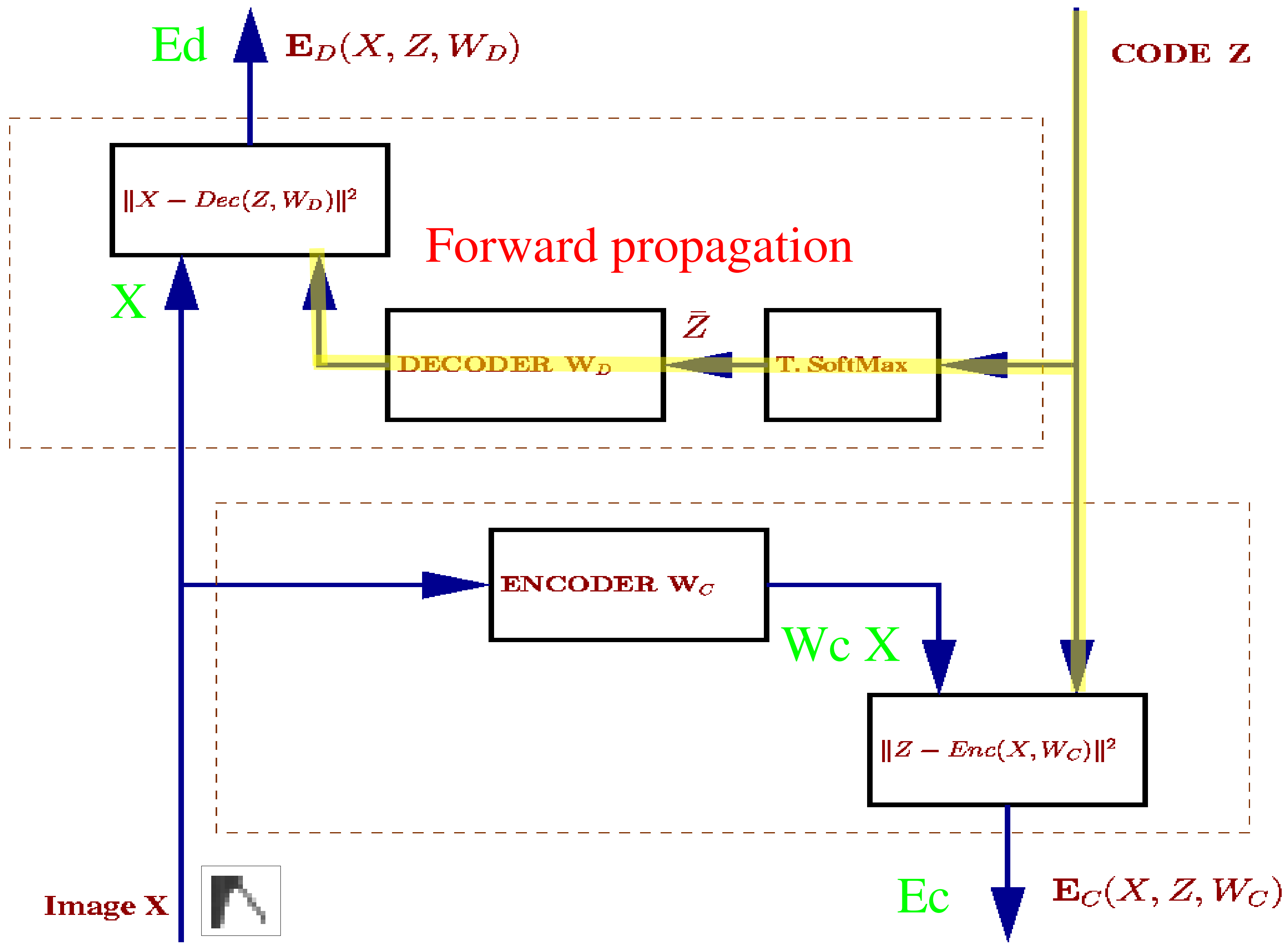
Inference & Learning



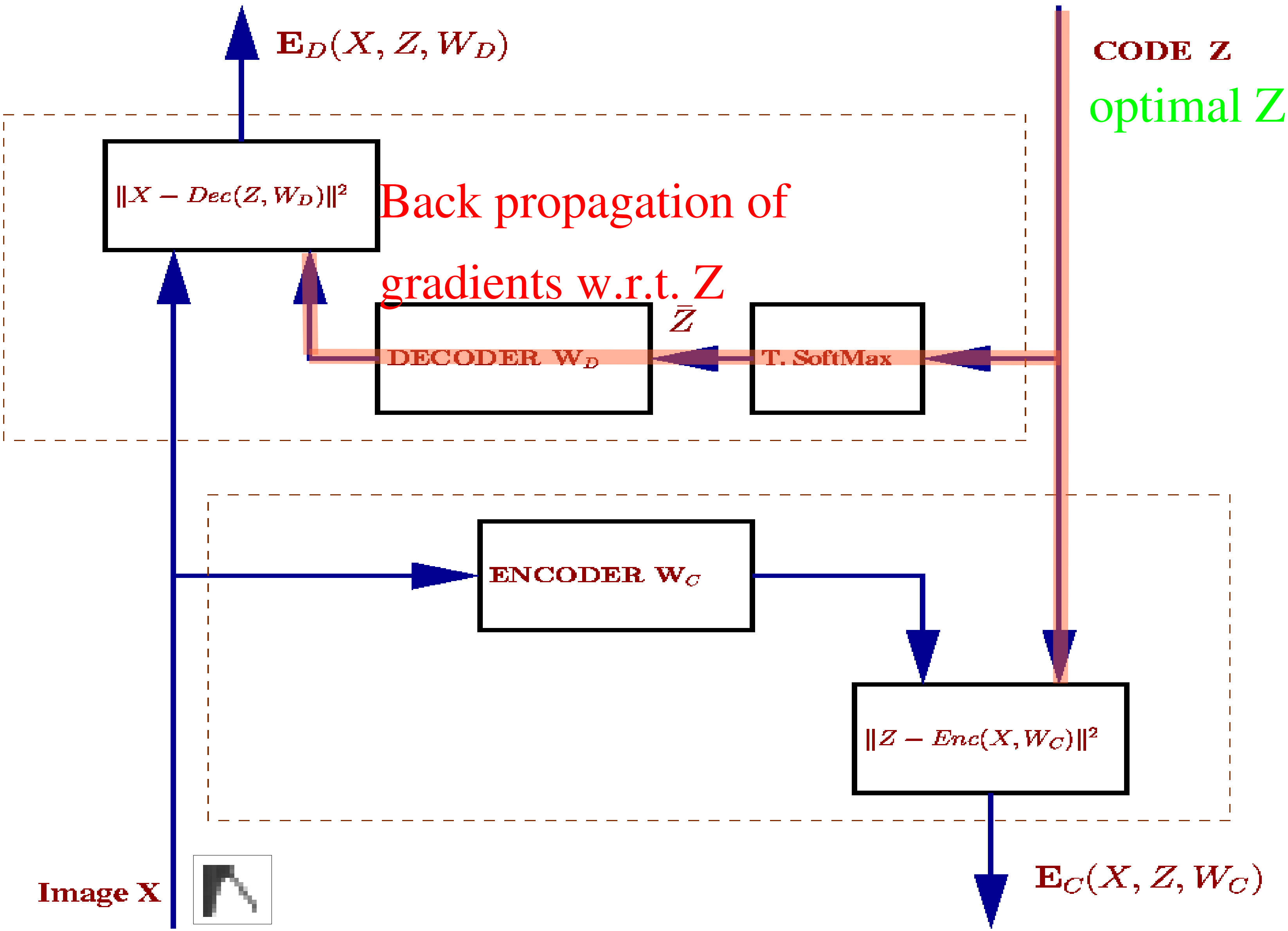
Inference - step 1



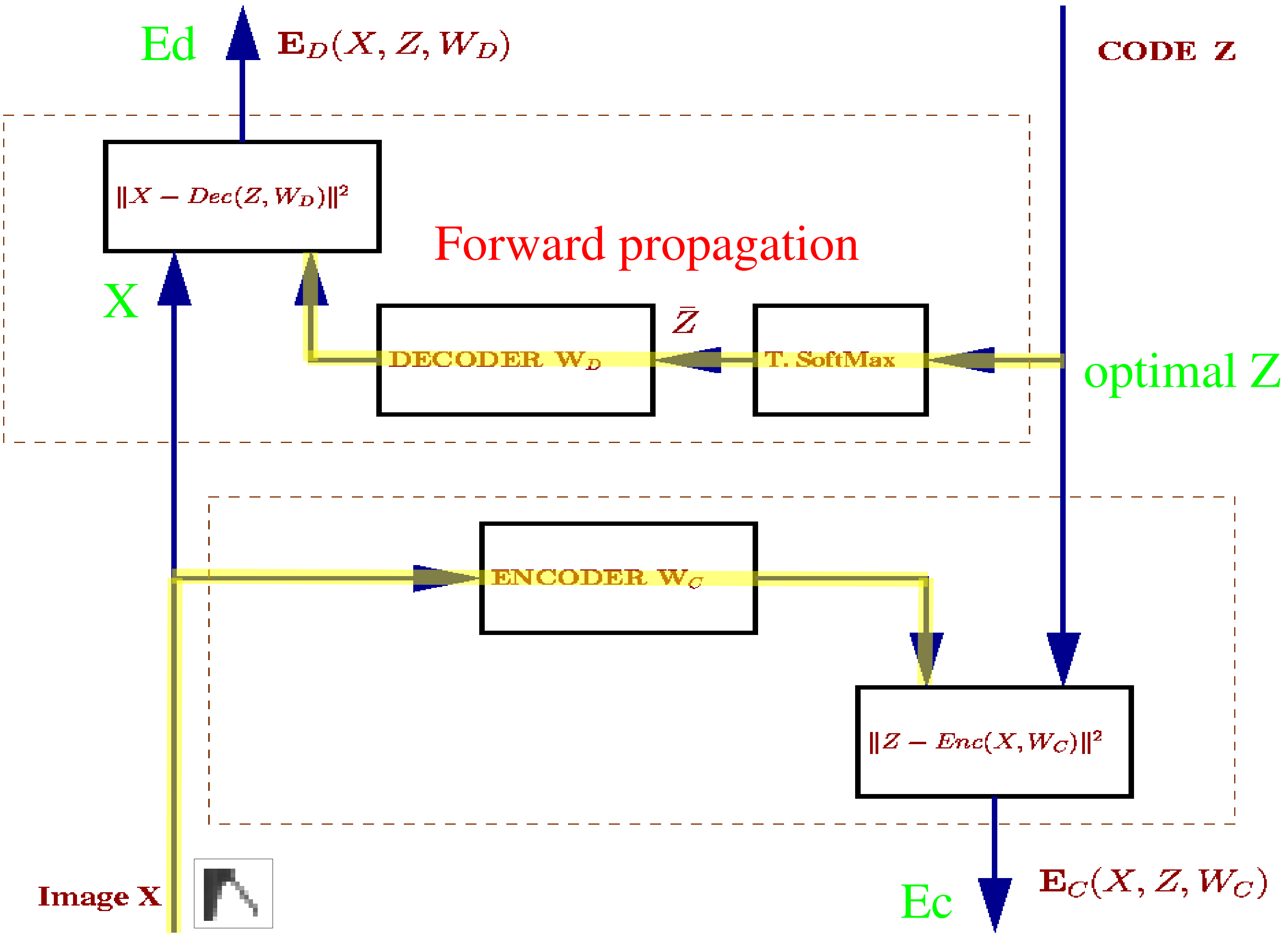
Inference - step 1



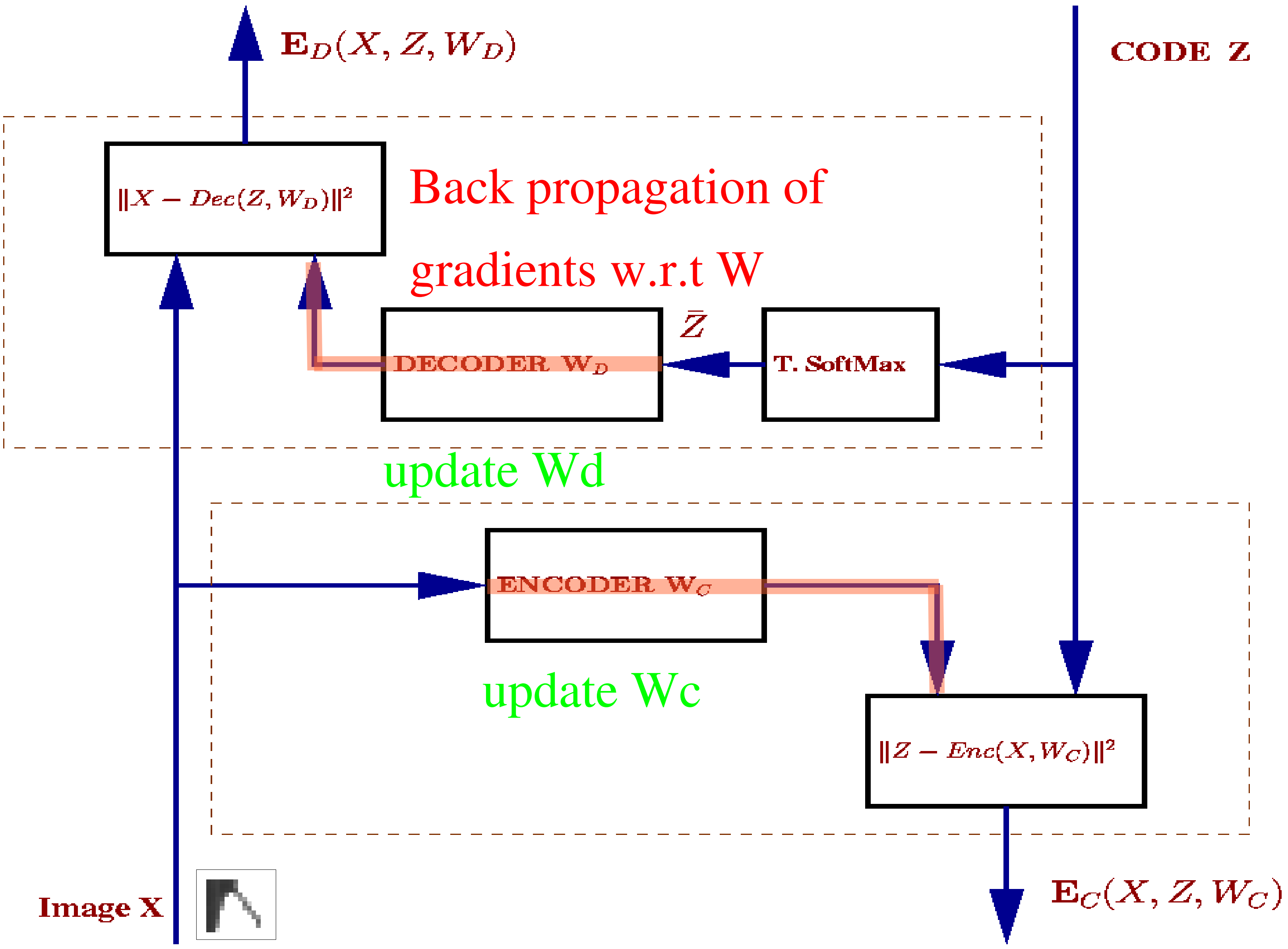
Inference - step 1



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

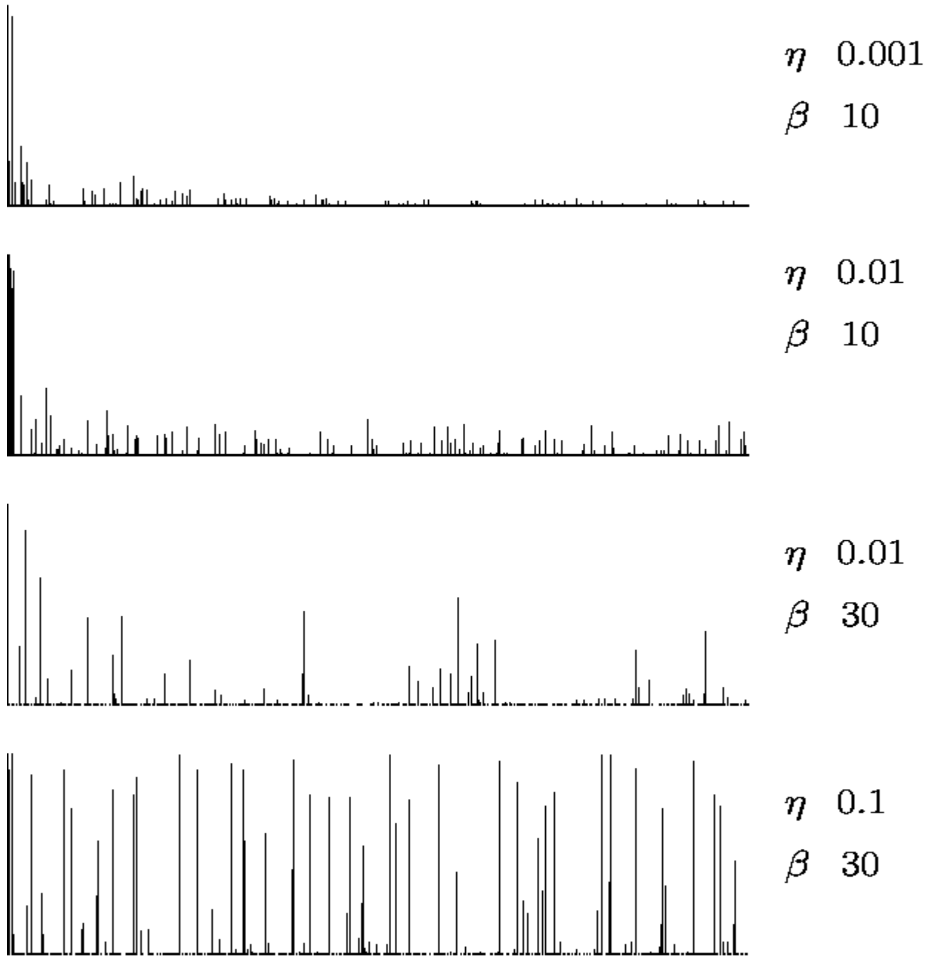
  => no normalization

- ξ is treated as a learned parameter

  => TSM is a sigmoid function with a special bias

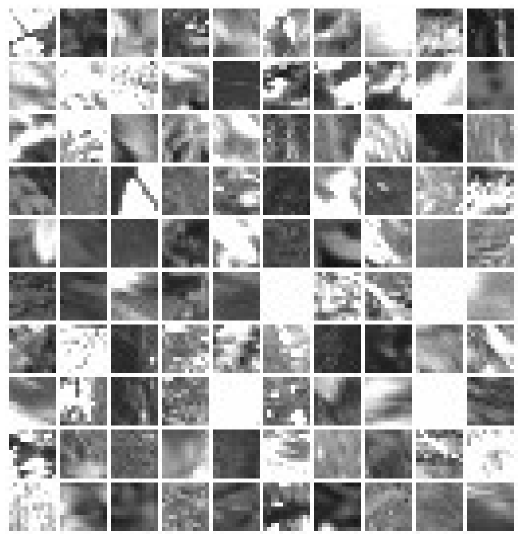
$$\bar{z}_i(t) = \frac{1}{1 + B e^{-\beta z_i(t)}}$$

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



input uniformly distributed in [-1,1]

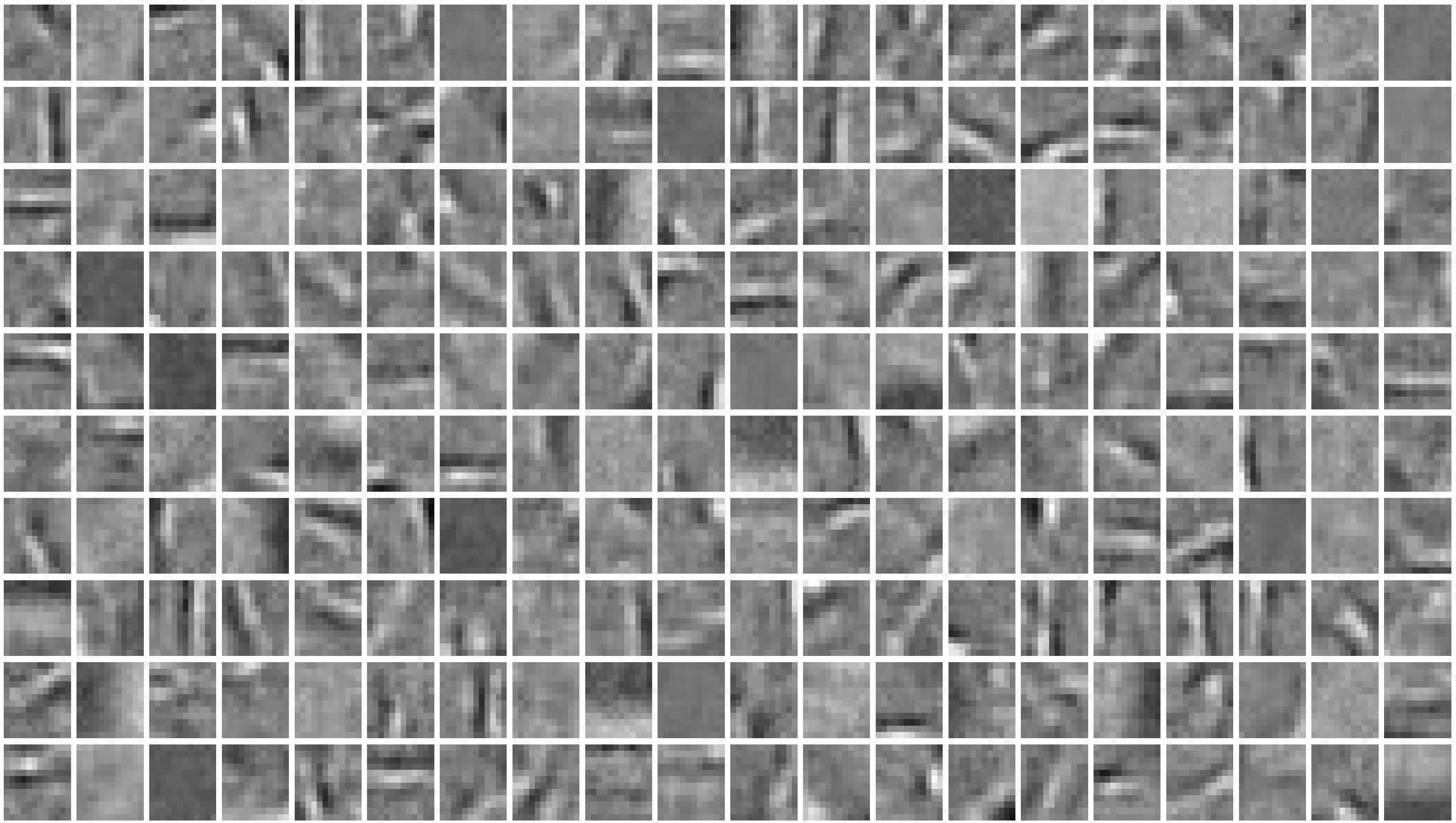
Natural image patches - Berkeley



Berkeley data set

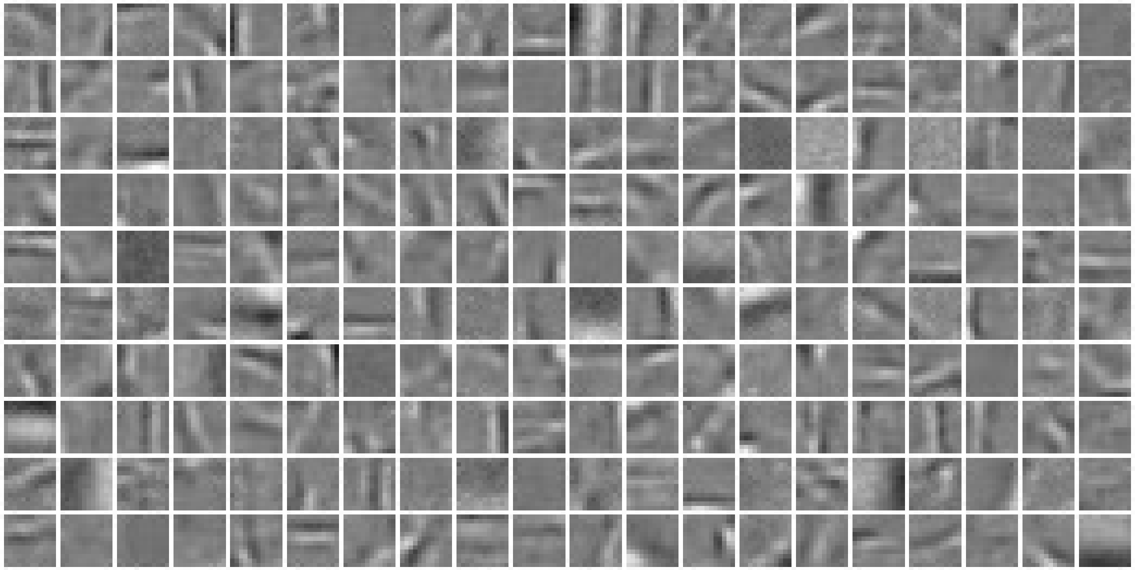
- 100,000 12x12 patches
- 200 units in the code
- η β 0.02
- 1
- learning rate 0.001
- L1, L2 regularizer 0.001
- fast convergence: < 30min.

Natural image patches - Berkeley



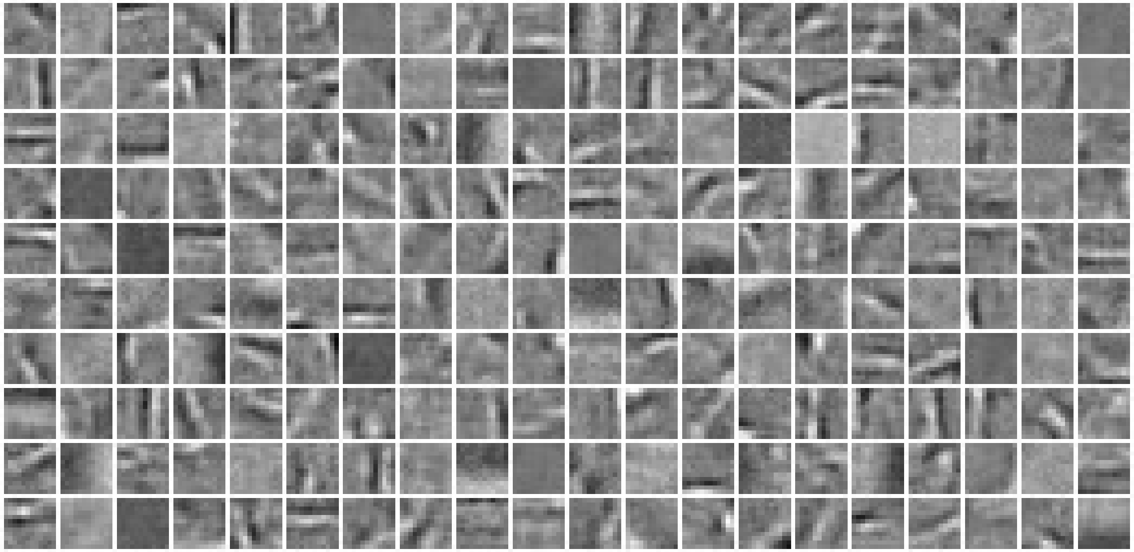
200 decoder filters (reshaped columns of matrix $\mathbf{W_d}$)

Natural image patches - Berkeley



Encoder *direct* filters

(rows of $\mathbf{W_c}$)

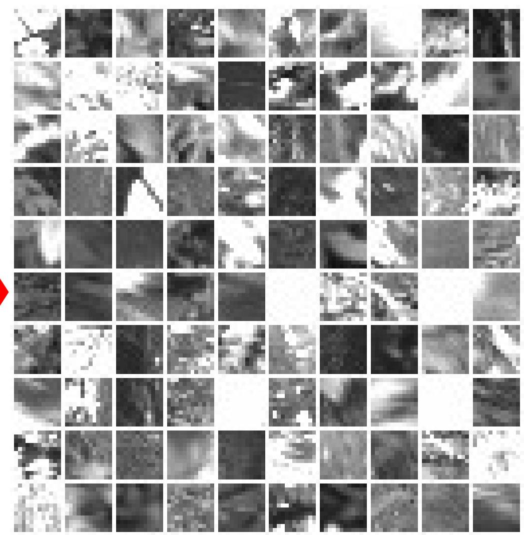


Decoder *reverse* filters

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

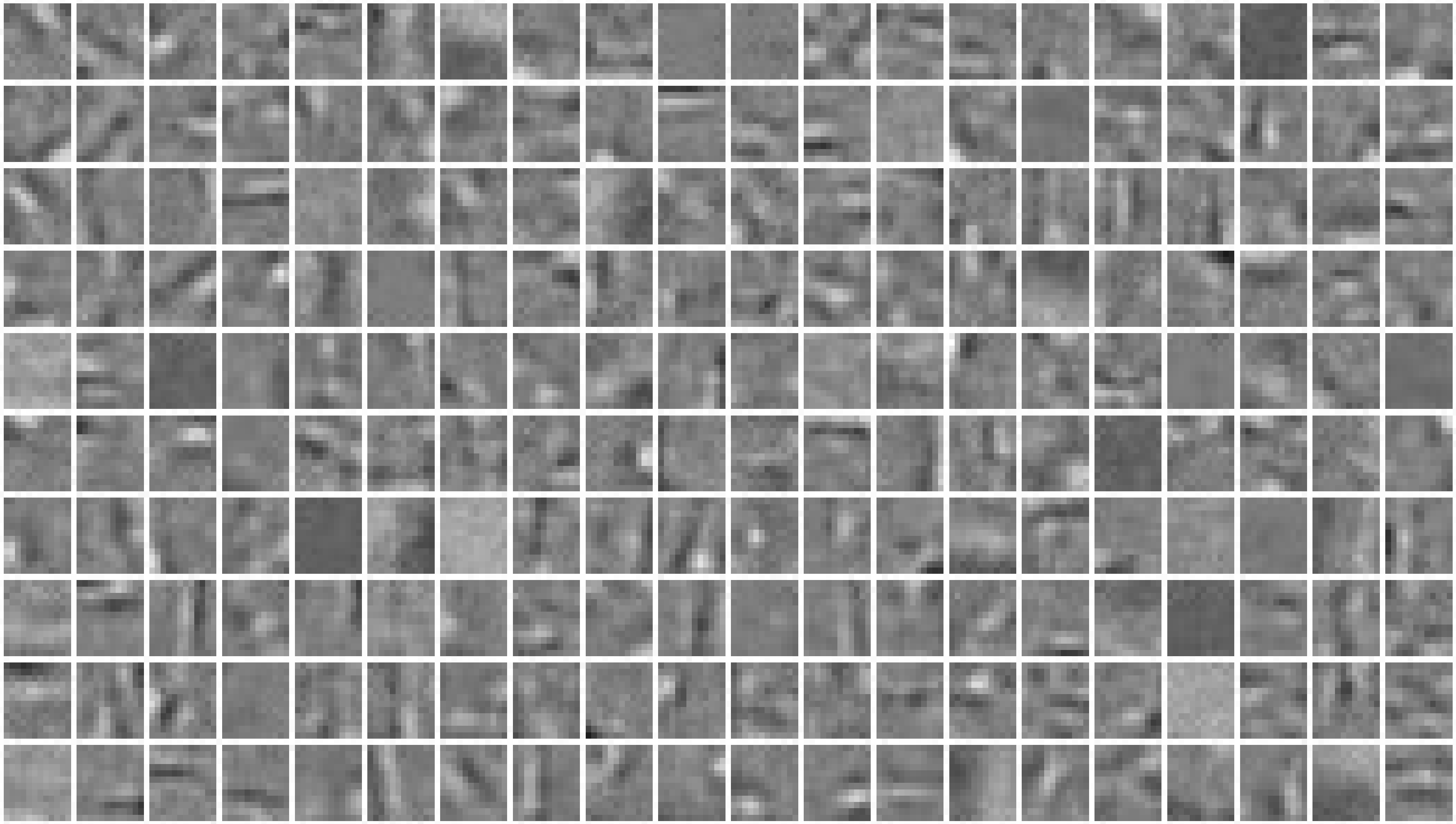
Natural image patches - Forest

Forest data set



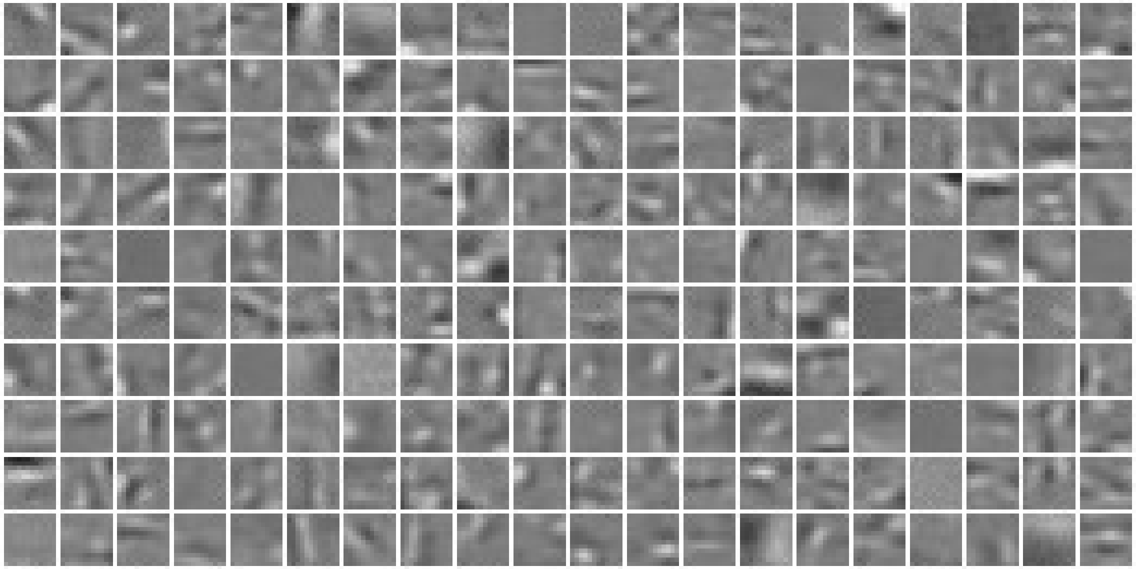
- 100,000 12x12 patches
- 200 units in the code
- η β 0.02
- 1
- learning rate 0.001
- L1, L2 regularizer 0.001
- fast convergence: < 30min.

Natural image patches - Forest



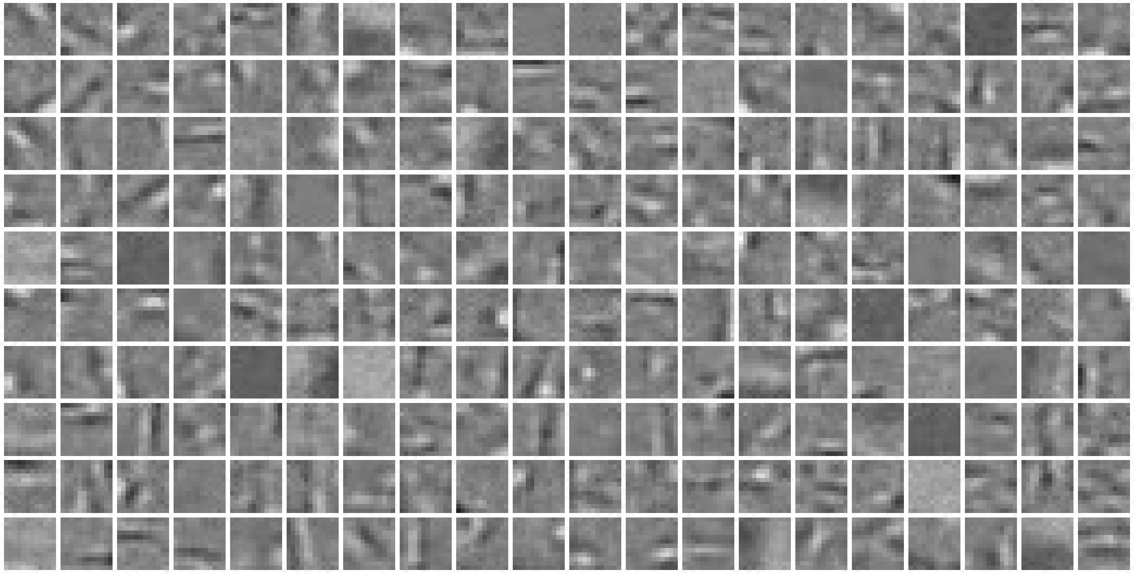
200 decoder filters (reshaped columns of matrix $\mathbf{W_d}$)

Natural image patches - Forest



Encoder *direct* filters

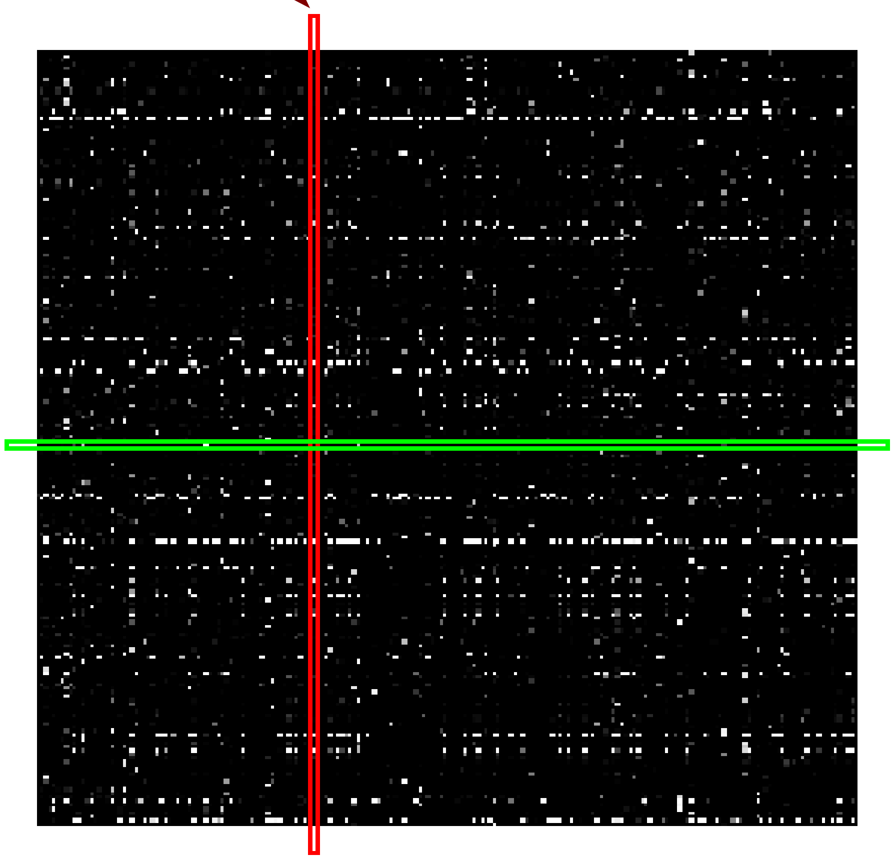
(rows of $\mathbf{W_c}$)



Decoder *reverse* filters

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

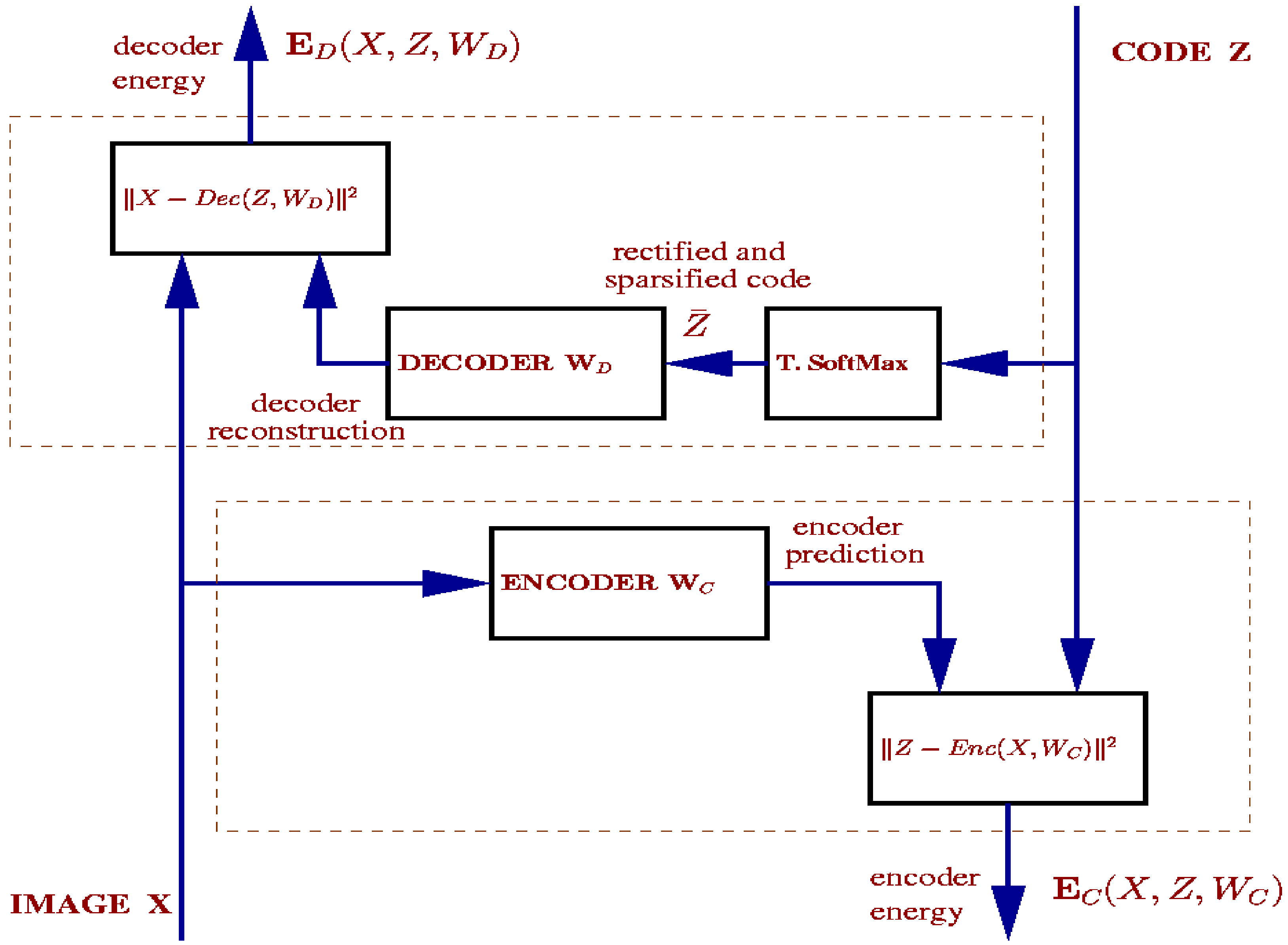
Natural image patches - Forest



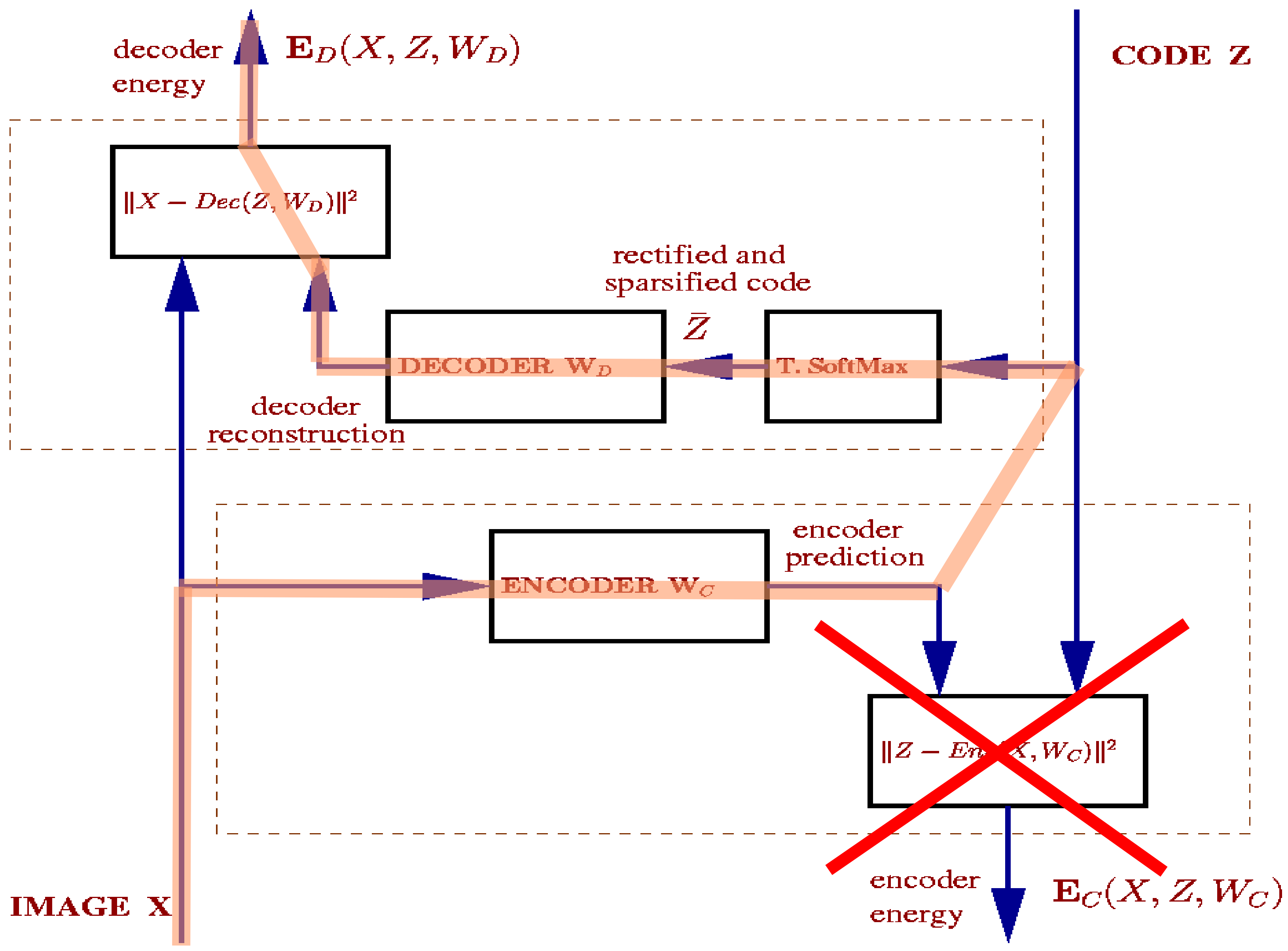
- codes are:
  - sparse
  - almost binary
  - quite decorrelated
- in testing codes are produced by propagating the input patch through encoder and TSM
- η controls sparsity
- β controls the “bit content” in each code unit

code words from 200 randomly selected test patches

What about an autoencoder?

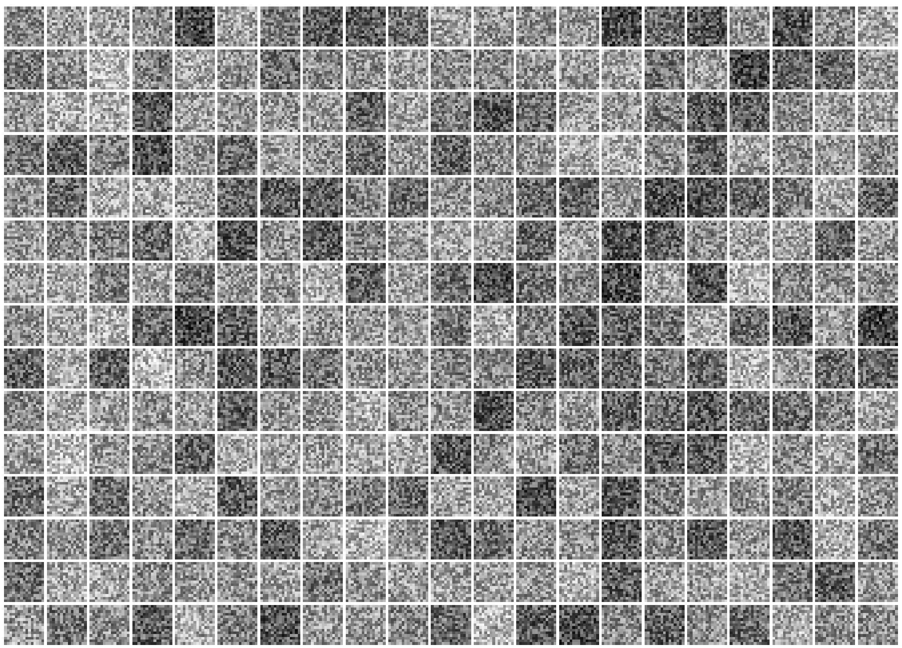


What about an autoencoder?

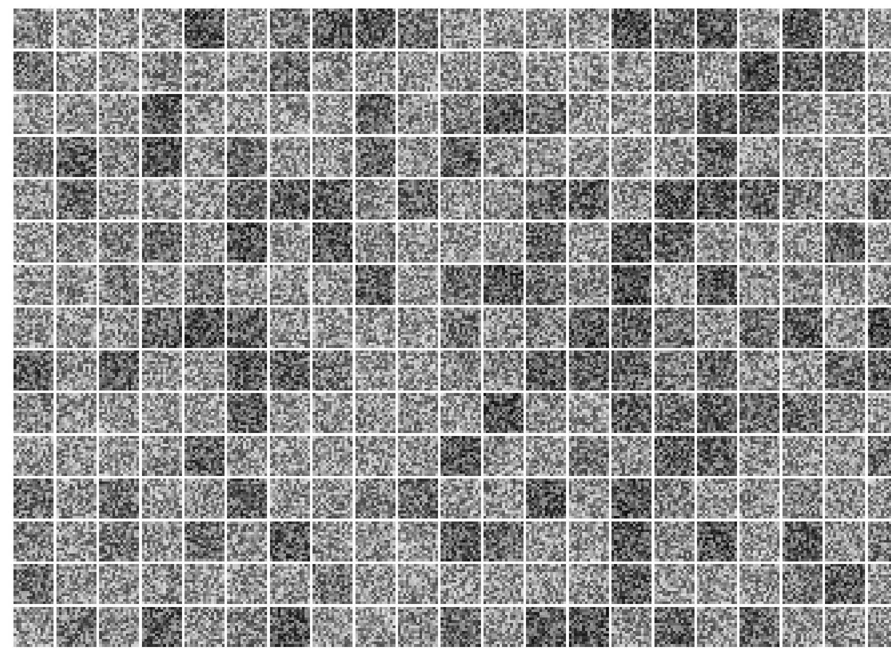


What about an autoencoder?

encoder filters



decoder filters

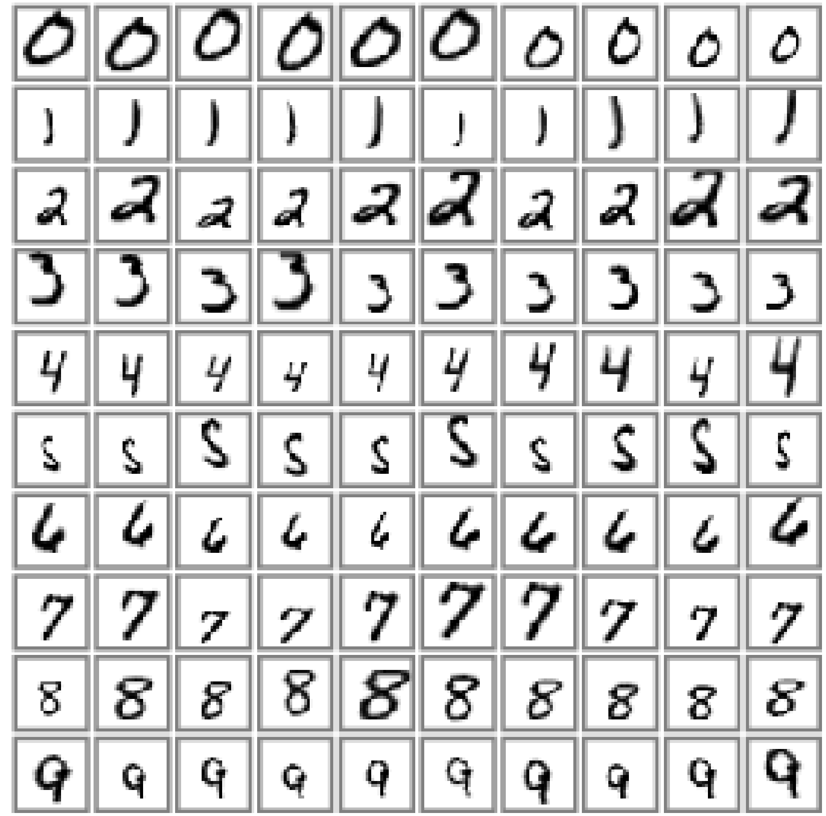


η 0.1

β 0.5

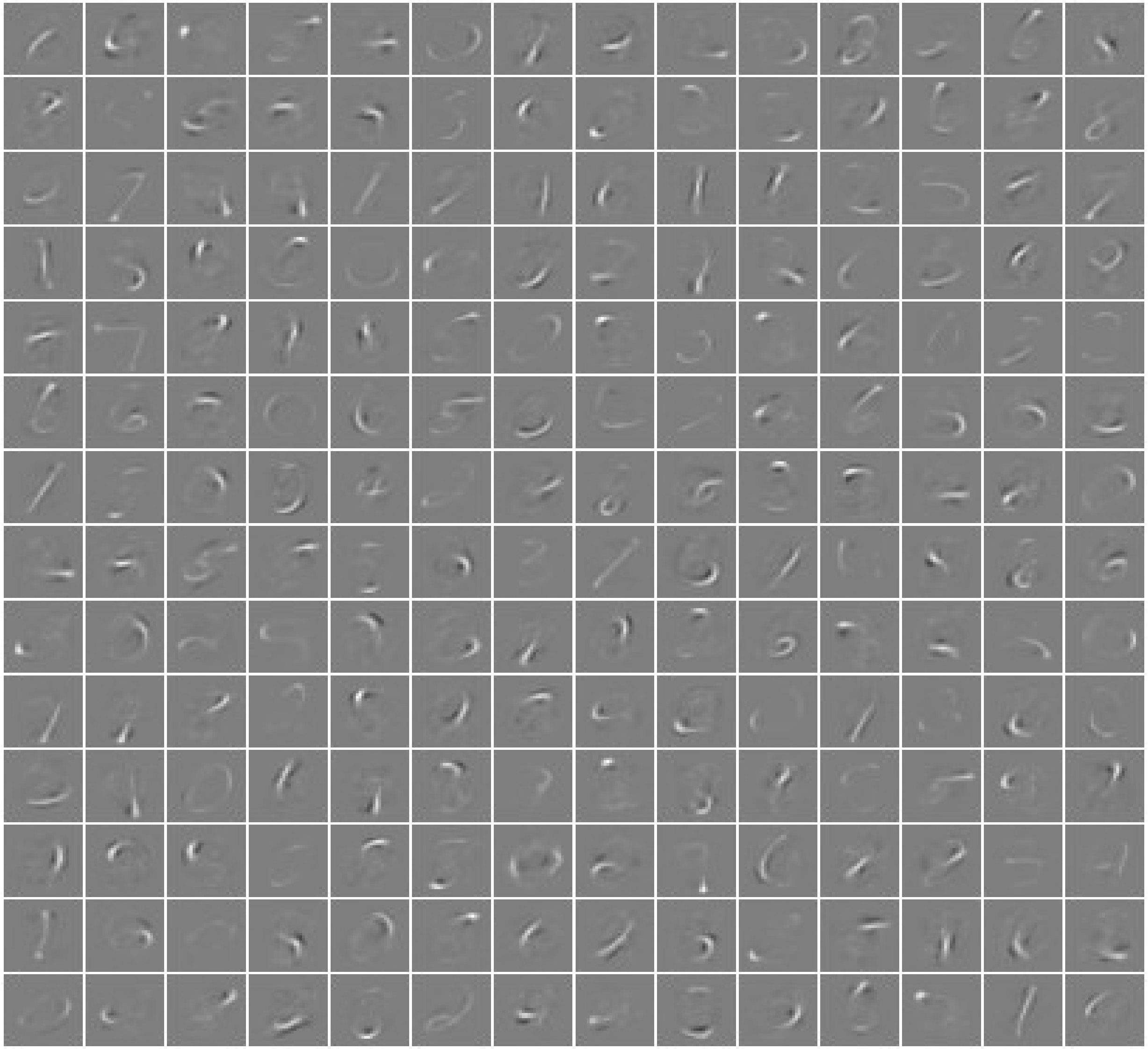
- filters are random
- convergence only for large η and small β

MNIST Dataset



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

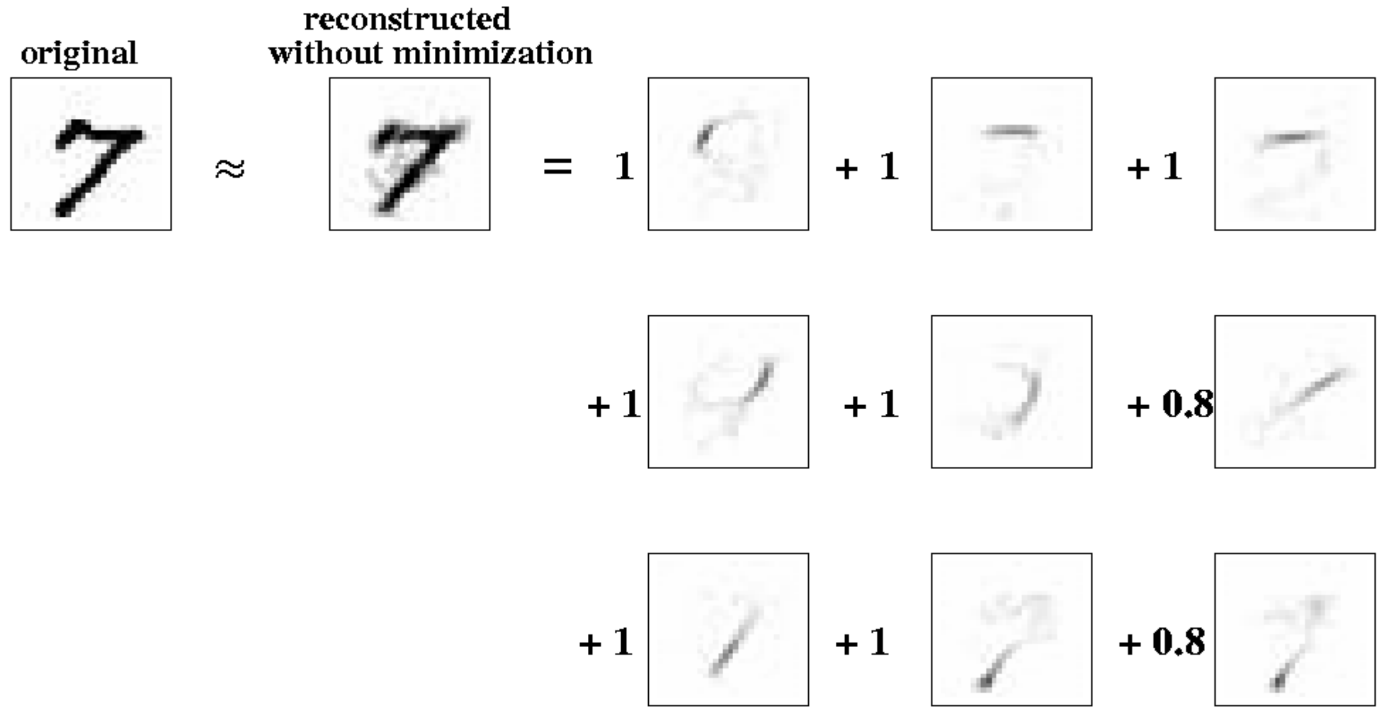
Handwritten digits - MNIST

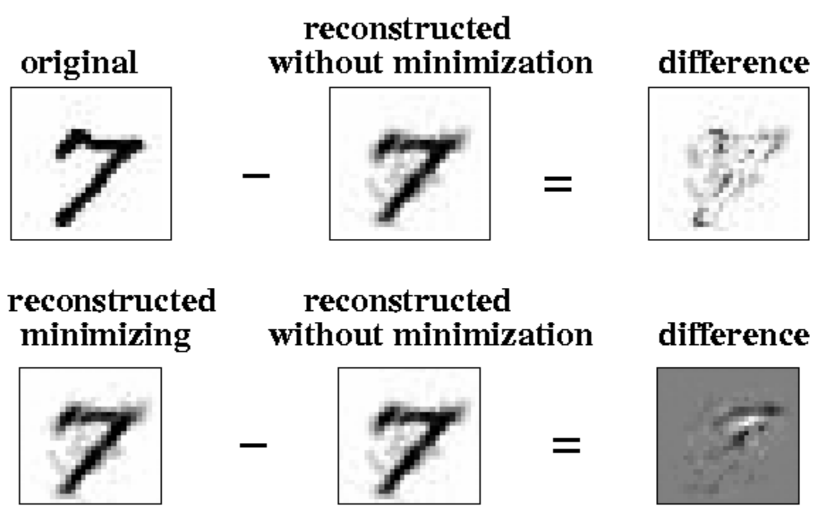


Encoder *direct* filters

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

Handwritten digits - MNIST



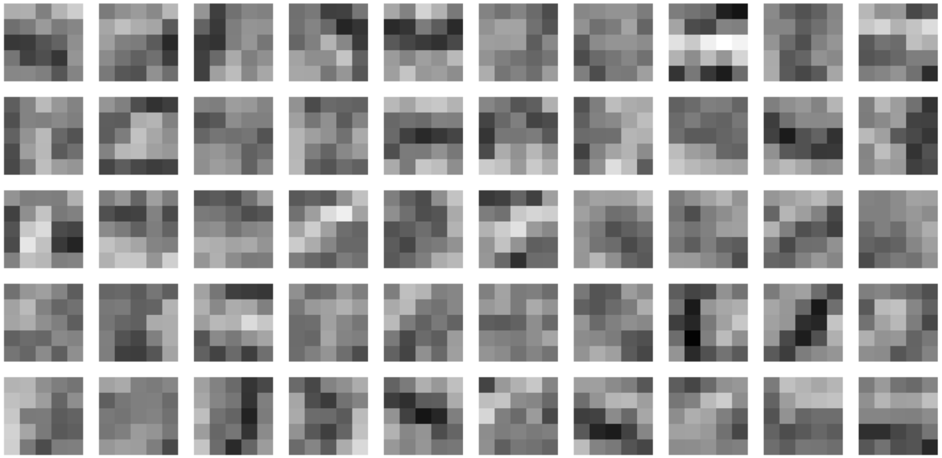


forward propagation through encoder and decoder

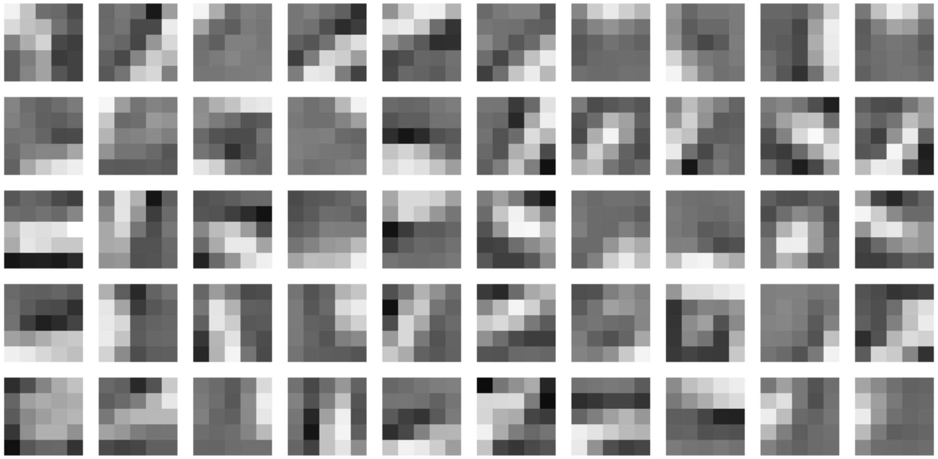
after training there is no need to minimize in code space

Initializing a Convolutional Net with SPoE

- **Architecture: LeNet-6**
  - 1->50->50->200->10
- **Baseline: random initialization**
  - 0.7% error on test set



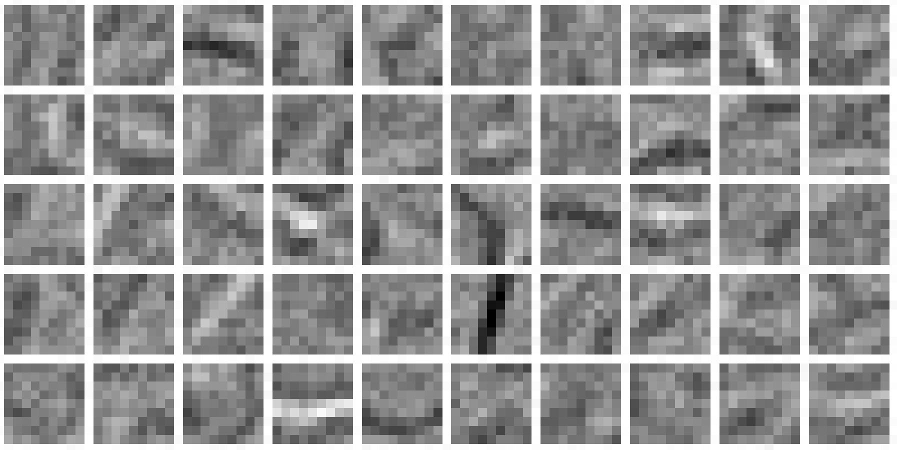
- **First Layer Initialized with SpoE**
  - 0.6% error on test set



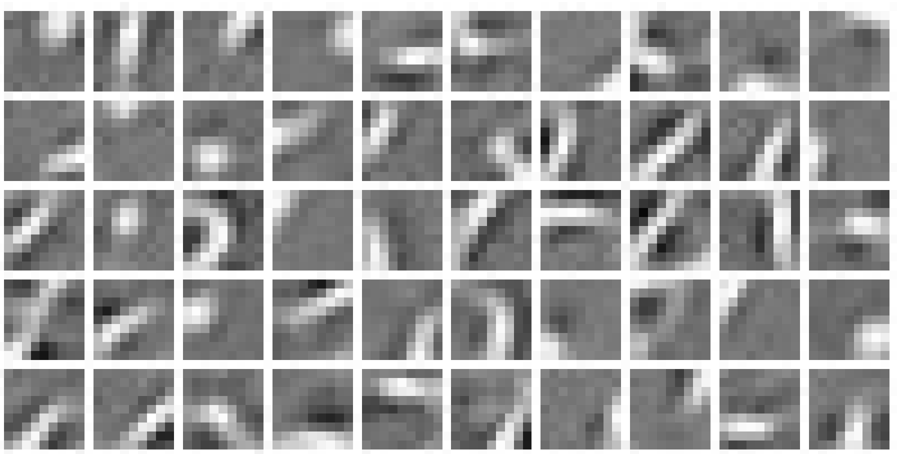
- **Training with elastically-distorted samples:**
  - 0.38% error on test set

Initializing a Convolutional Net with SPoE

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



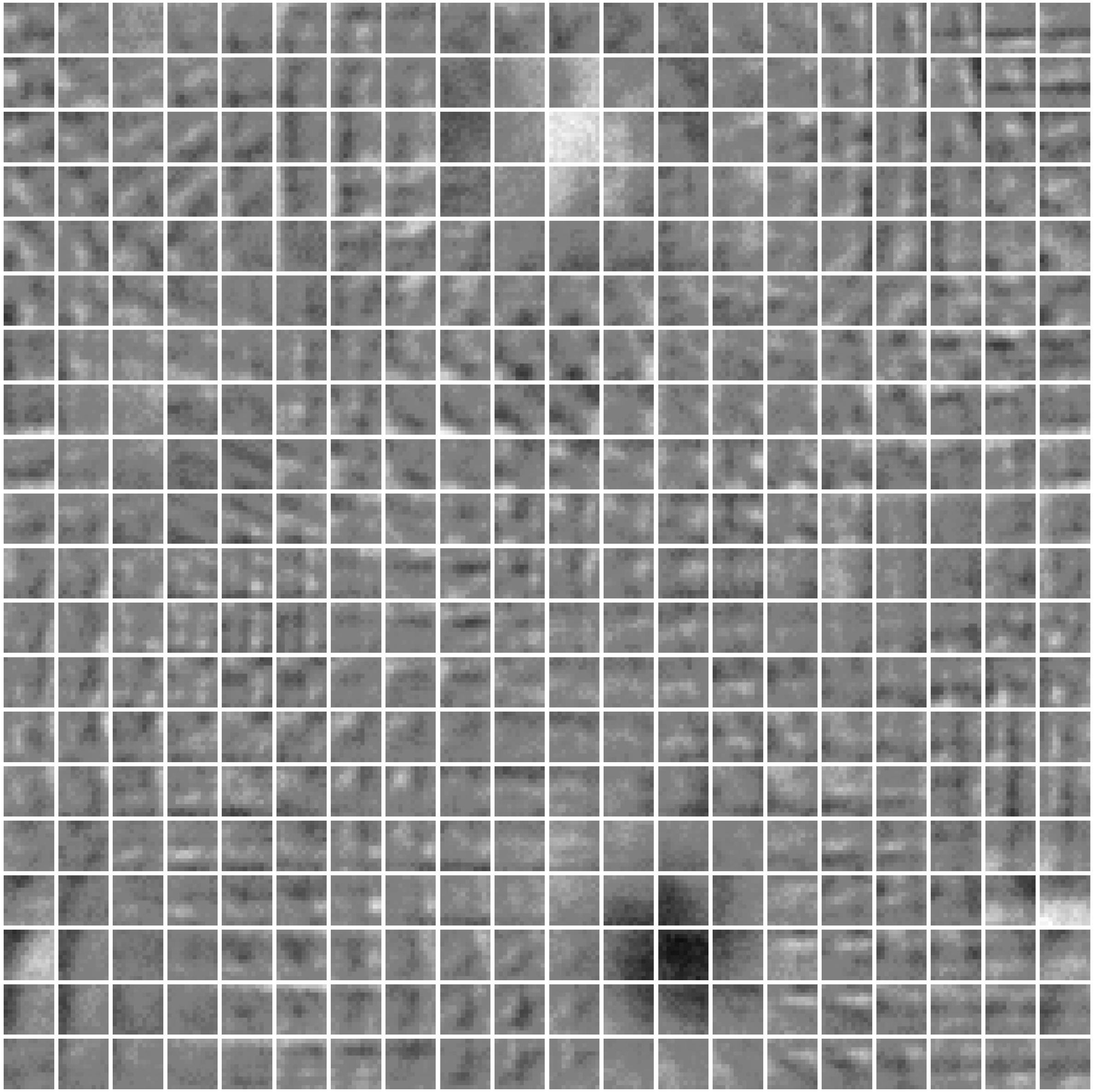
- **First Layer Initialized with SpoE**

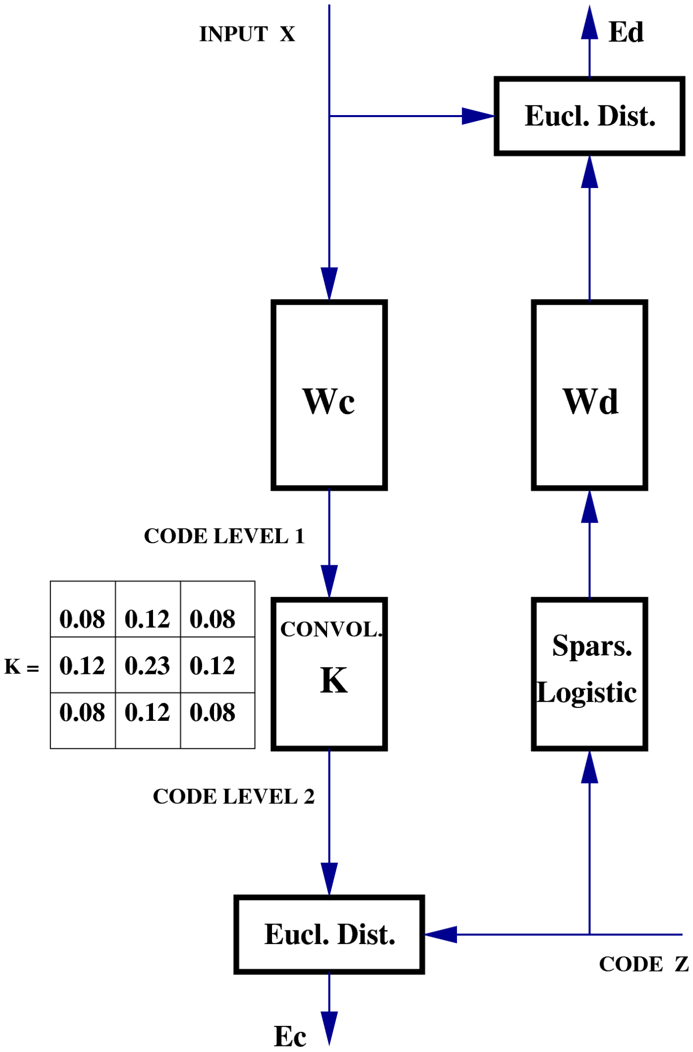


Best Results on MNIST (from raw images: no preprocessing)

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

Topographic maps





Lessons

- **Initializing the first layer(s) with unsupervised learning helps**
- **Why is there no partition function here?**
  - The partition function is bounded because of the information bottleneck in the code
  - There is only a few input configuration that can have low energy because there are only a few possible codes.

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