Energy-Based
Unsupervised Learning
Learning an energy function (or contrast function) that takes
- Low values on the data manifold
- Higher values everywhere else
The energy surface is a “contrast function” that takes low values on the data manifold, and higher values everywhere else

- Special case: energy = negative log density
- Example: the samples live in the manifold

\[ E = (Y_1)^2 \]
The energy can be interpreted as an unnormalized negative log density

Gibbs distribution: Probability proportional to \( \exp(-\text{energy}) \)
   - Beta parameter is akin to an inverse temperature

Don't compute probabilities unless you absolutely have to
   - Because the denominator is often intractable

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

\[E(Y, W) \propto - \log P(Y|W)\]
Learning the Energy Function

- Parameterized energy function $E(Y,W)$
  - Make the energy low on the samples
  - Make the energy higher everywhere else
  - Making the energy low on the samples is easy
  - But how do we make it higher everywhere else?
Seven Strategies to Shape the Energy Function

1. build the machine so that the volume of low energy stuff is constant
   - PCA, K-means, GMM, square ICA

2. push down of the energy of data points, push up everywhere else
   - Max likelihood (needs tractable partition function)

3. push down of the energy of data points, push up on chosen locations
   - contrastive divergence, Ratio Matching, Noise Contrastive Estimation, Minimum Probability Flow

4. minimize the gradient and maximize the curvature around data points
   - score matching

5. train a dynamical system so that the dynamics goes to the manifold
   - denoising auto-encoder

6. use a regularizer that limits the volume of space that has low energy
   - Sparse coding, sparse auto-encoder, PSD

7. if $E(Y) = ||Y - G(Y)||^2$, make $G(Y)$ as "constant" as possible.
   - Contracting auto-encoder, saturating auto-encoder
1. build the machine so that the volume of low energy stuff is constant
   - PCA, K-means, GMM, square ICA...

PCA

\[ E(Y) = \| W^T W Y - Y \|^2 \]

K-Means,
Z constrained to 1-of-K code

\[ E(Y) = \min_z \sum_i \| Y - W_i Z_i \|^2 \]
#2: push down of the energy of data points, push up everywhere else

Max likelihood (requires a tractable partition function)

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

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

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

$$L(Y, W) = E(Y, W) + \frac{1}{\beta} \log \int_y e^{-\beta E(y,W)}$$
Gradient of the negative log-likelihood loss for one sample \( Y \):

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

Gradient descent:

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

- Pushes down on the energy of the samples
- Pulls up on the energy of low-energy \( Y \)'s
contrastive divergence, Ratio Matching, Noise Contrastive Estimation, Minimum Probability Flow

**Contrastive divergence: basic idea**
- Pick a training sample, lower the energy at that point
- From the sample, move down in the energy surface with noise
- Stop after a while
- Push up on the energy of the point where we stopped
- This creates grooves in the energy surface around data manifolds
- CD can be applied to any energy function (not just RBMs)

**Persistent CD: use a bunch of “particles” and remember their positions**
- Make them roll down the energy surface with noise
- Push up on the energy wherever they are
- Faster than CD

**RBM**

\[
E(Y, Z) = -Z^T W Y \\
E(Y) = -\log \sum_z e^{Z^T W Y}
\]
#6. use a regularizer that limits the volume of space that has low energy

Sparse coding, sparse auto-encoder, Predictive Sparse Decomposition
Energy Functions of Various Methods

- **PCA** (1 code unit)
- **autoencoder** (1 code unit)
- **sparse coding** (20 code units)
- **K-Means** (20 code units)

<table>
<thead>
<tr>
<th></th>
<th>encoder</th>
<th>decoder</th>
<th>energy</th>
<th>loss</th>
<th>pull-up</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>W'Y</strong></td>
<td><strong>WZ</strong></td>
<td><strong>∥Y - WZ∥^2</strong></td>
<td><strong>F(Y)</strong></td>
<td><strong>dimens.</strong></td>
<td></td>
</tr>
<tr>
<td><strong>σ(WeY)</strong></td>
<td><strong>WdZ</strong></td>
<td><strong>∥Y - WZ∥^2</strong></td>
<td><strong>F(Y)</strong></td>
<td><strong>dimens.</strong></td>
<td></td>
</tr>
<tr>
<td><strong>σ(WeZ)</strong></td>
<td><strong>WdZ</strong></td>
<td><strong>∥Y - WZ∥^2</strong></td>
<td><strong>F(Y)</strong></td>
<td><strong>sparsity</strong></td>
<td></td>
</tr>
<tr>
<td>–</td>
<td><strong>WZ</strong></td>
<td><strong>∥Y - WZ∥^2</strong></td>
<td><strong>F(Y)</strong></td>
<td><strong>1-of-N code</strong></td>
<td></td>
</tr>
</tbody>
</table>

2 dimensional toy dataset: spiral

Visualizing energy surface

(black = low, white = high)
Sparse Modeling, Sparse Auto-Encoders, Predictive Sparse Decomposition LISTA
How to Speed Up Inference in a Generative Model?

- Factor Graph with an asymmetric factor
  - Inference $Z \rightarrow Y$ is easy
    - Run $Z$ through deterministic decoder, and sample $Y$
  - Inference $Y \rightarrow Z$ is hard, particularly if Decoder function is many-to-one
    - MAP: minimize sum of two factors with respect to $Z$
      - $Z^* = \text{argmin}_z \text{ Distance}[\text{Decoder}(Z), Y] + \text{FactorB}(Z)$
  - Examples: K-Means (1 of K), Sparse Coding (sparse), Factor Analysis

![Diagram](image)
Sparse linear reconstruction

Energy = reconstruction_error + code_prediction_error + code_sparsity

\[ E(Y^i, Z) = \| Y^i - W_d Z \|^2 + \lambda \sum_j |z_j| \]

Inference is slow

\[ Y \rightarrow \hat{Z} = \text{argmin}_Z E(Y, Z) \]
Examples: most ICA models, Product of Experts
Train a “simple” feed-forward function to predict the result of a complex optimization on the data points of interest.

1. Find optimal $Z_i$ for all $Y_i$; 2. Train Encoder to predict $Z_i$ from $Y_i$
Why Limit the Information Content of the Code?

- Training sample
- Input vector which is NOT a training sample
- Feature vector
Why Limit the Information Content of the Code?

- Training sample
- Input vector which is NOT a training sample
- Feature vector

*Training based on minimizing the reconstruction error over the training set*
Why Limit the Information Content of the Code?

- Training sample
- Input vector which is NOT a training sample
- Feature vector

BAD: machine does not learn structure from training data!!
It just copies the data.
Why Limit the Information Content of the Code?

- Training sample
- Input vector which is NOT a training sample
- Feature vector

IDEA: reduce number of available codes.
Why Limit the Information Content of the Code?

- Training sample
- Input vector which is NOT a training sample
- Feature vector

IDEA: reduce number of available codes.
Why Limit the Information Content of the Code?

- Training sample
- Input vector which is NOT a training sample
- Feature vector

**IDEA: reduce number of available codes.**
Predictive Sparse Decomposition (PSD): sparse auto-encoder

[Kavukcuoglu, Ranzato, LeCun, 2008 → arXiv:1010.3467],

Prediction the optimal code with a trained encoder

Energy = reconstruction_error + code_prediction_error + code_sparsity

\[ E(Y^i, Z) = \| Y^i - W_d Z \|^2 + \| Z - g_e(W_e, Y^i) \|^2 + \lambda \sum_j |z_j| \]

\[ g_e(W_e, Y^i) = shrinkage(W_e Y^i) \]
Basis functions (and encoder matrix) are digit parts
Training on natural images patches.

- 12X12
- 256 basis functions
Learned Features on natural patches: V1-like receptive fields
Better Idea: Give the “right” structure to the encoder

ISTA/FISTA: iterative algorithm that converges to optimal sparse code

[Gregor & LeCun, ICML 2010], [Bronstein et al. ICML 2012], [Rolfe & LeCun ICLR 2013]

\[
Z(t + 1) = \text{Shrinkage}_{\lambda/L} \left[ Z(t) - \frac{1}{L} W_d^T (W_d Z(t) - Y) \right]
\]

\[
Z(t + 1) = \text{Shrinkage}_{\lambda/L} \left[ W_e^T Y + S Z(t) \right]; \quad W_e = \frac{1}{L} W_d; \quad S = I - \frac{1}{L} W_d^T W_d
\]
LISTA: Train $W_e$ and $S$ matrices to give a good approximation quickly

Think of the FISTA flow graph as a recurrent neural net where $W_e$ and $S$ are trainable parameters.

Time-Unfold the flow graph for $K$ iterations.

Learn the $W_e$ and $S$ matrices with “backprop-through-time”.

Get the best approximate solution within $K$ iterations.
Learning ISTA (LISTA) vs ISTA/FISTA

The graph plots the reconstruction error against the number of LISTA or FISTA iterations. The x-axis represents the number of iterations, while the y-axis shows the reconstruction error. Different markers represent different iteration counts:

- FISTA (4x)
- FISTA (1x)
- LISTA (4x)
- LISTA (1x)

The x-axis ranges from 0 to 7 iterations, and the y-axis ranges from 0 to 10 reconstruction error units.
LISTA with partial mutual inhibition matrix

Proportion of S matrix elements that are non zero

Smallest elements removed
Learning Coordinate Descent (LcoD): faster than LISTA

![Graph showing comparison between CoD and LCoD]

- CoD (4x)
- CoD (1x)
- LCoD (4x)
- LCoD (1x)

Error vs. Number of LISTA or FISTA iterations
Y LeCun

Discriminative Recurrent Sparse Auto-Encoder (DrSAE)

Rectified linear units
Classification loss: cross-entropy
Reconstruction loss: squared error
Sparsity penalty: L1 norm of last hidden layer
Rows of \( W_d \) and columns of \( W_e \) constrained in unit sphere

[Rolfe & LeCun ICLR 2013]
DrSAE Discovers manifold structure of handwritten digits

Image = prototype + sparse sum of “parts” (to move around the manifold)
Replace the dot products with dictionary element by convolutions.
- Input $Y$ is a full image
- Each code component $Z_k$ is a feature map (an image)
- Each dictionary element is a convolution kernel

Regular sparse coding

$$E(Y, Z) = ||Y - \sum_{k} W_k Z_k||^2 + \alpha \sum_{k} |Z_k|$$

Convolutional S.C.

$$E(Y, Z) = ||Y - \sum_{k} W_k \ast Z_k||^2 + \alpha \sum_{k} |Z_k|$$

```
Y = \sum_{k} W_k \ast Z_k
```

“deconvolutional networks” [Zeiler, Taylor, Fergus CVPR 2010]
Convolutional PSD: Encoder with a soft $sh()$ Function

**Convolutional Formulation**
- Extend sparse coding from **PATCH** to **IMAGE**

$$
\mathcal{L}(x, z, D) = \frac{1}{2} \| x - \sum_{k=1}^{K} D_k \ast z_k \|_2^2 + \sum_{k=1}^{K} \| z_k - f(W^k \ast x) \|_2^2 + |z|_1
$$

- **PATCH** based learning
- **CONVOLUTIONAL** learning
Filters and Basis Functions obtained with 1, 2, 4, 8, 16, 32, and 64 filters.
Phase 1: train first layer using PSD

$\|Y^i - \hat{Y}\|^2$  $W_d Z$  $\|Z - \tilde{Z}\|^2$

$g_e(W_e, Y^i)$  $\|Z - \tilde{Z}\|^2$

$\lambda \sum_j |z_j|$
Phase 1: train first layer using PSD

Phase 2: use encoder + absolute value as feature extractor

\[ \text{features} = \left| z_j \right| \]
Using PSD to Train a Hierarchy of Features

- Phase 1: train first layer using PSD
- Phase 2: use encoder + absolute value as feature extractor
- Phase 3: train the second layer using PSD
Using PSD to Train a Hierarchy of Features

- **Phase 1:** train first layer using PSD
- **Phase 2:** use encoder + absolute value as feature extractor
- **Phase 3:** train the second layer using PSD
- **Phase 4:** use encoder + absolute value as 2\textsuperscript{nd} feature extractor
Phase 1: train first layer using PSD
Phase 2: use encoder + absolute value as feature extractor
Phase 3: train the second layer using PSD
Phase 4: use encoder + absolute value as 2nd feature extractor
Phase 5: train a supervised classifier on top
Phase 6 (optional): train the entire system with supervised back-propagation
Pedestrian Detection: INRIA Dataset. Miss rate vs false positives

[Kavukcuoglu et al. NIPS 2010] [Sermanet et al. ArXiv 2012]
Unsupervised Learning: Invariant Features
Unsupervised PSD ignores the spatial pooling step. Could we devise a similar method that learns the pooling layer as well? 

Idea [Hyvarinen & Hoyer 2001]: group sparsity on pools of features

- Minimum number of pools must be non-zero
- Number of features that are on within a pool doesn't matter
- Pools tend to regroup similar features

\[ E(Y, Z) = \| Y - W_d Z \|^2 + \| Z - g_e(W_e, Y) \|^2 + \sum_j \sqrt{\sum_{k \in P_j} Z_k^2} \]
Learning Invariant Features with L2 Group Sparsity

- Idea: features are pooled in group.
  - Sparsity: sum over groups of L2 norm of activity in group.
- [Hyvärinen Hoyer 2001]: “subspace ICA”
  - decoder only, square
- [Welling, Hinton, Osindero NIPS 2002]: pooled product of experts
  - encoder only, overcomplete, log student-T penalty on L2 pooling
- [Kavukcuoglu, Ranzato, Fergus LeCun, CVPR 2010]: Invariant PSD
  - encoder-decoder (like PSD), overcomplete, L2 pooling
- [Le et al. NIPS 2011]: Reconstruction ICA
  - Same as [Kavukcuoglu 2010] with linear encoder and tied decoder
  - Locally-connect non shared (tiled) encoder-decoder

INPUT

Y

Encoder only (PoE, ICA),
Decoder Only or
Encoder-Decoder (iPSD, RICA)

SIMPLE FEATURES

Z

L2 norm within each pool

\( \lambda \sum \sqrt{\sum Z_k^2} \)

INvariant FEATURES
The filters arrange themselves spontaneously so that similar filters enter the same pool.

The pooling units can be seen as complex cells.

Outputs of pooling units are invariant to local transformations of the input.

For some it's translations, for others rotations, or other transformations.
Training on 115x115 images. Kernels are 15x15 (not shared across space!)

- [Gregor & LeCun 2010]
- Local receptive fields
- No shared weights
- 4x overcomplete
- L2 pooling
- Group sparsity over pools

Image-level training, local filters but no weight sharing

Encoder

Decoder

Input

Reconstructed Input

(Inferred) Code

Predicted Code
Image-level training, local filters but no weight sharing

Training on 115x115 images. Kernels are 15x15 (not shared across space!)
119x119 Image Input
100x100 Code
20x20 Receptive field size
sigma=5

Michael C. Crair, et. al. The Journal of Neurophysiology
Vol. 77 No. 6 June 1997, pp. 3381-3385 (Cat)

K Obermayer and GG Blasdel, Journal of Neuroscience, Vol 13, 4114-4129 (Monkey)
Image-level training, local filters but no weight sharing

Color indicates orientation (by fitting Gabors)
Replace the L1 sparsity term by a lateral inhibition matrix

Easy way to impose some structure on the sparsity

\[
\min_{W,Z} \sum_{x \in X} \|Wz - x\|^2 + |z^T S z|
\]

[Gregor, Szlam, LeCun NIPS 2011]
Invariant Features via Lateral Inhibition: Structured Sparsity

- Each edge in the tree indicates a zero in the S matrix (no mutual inhibition)
- \( S_{ij} \) is larger if two neurons are far away in the tree
Non-zero values in $S$ form a ring in a 2D topology

- Input patches are high-pass filtered
Object is cross-product of object type and instantiation parameters

Mapping units [Hinton 1981], capsules [Hinton 2011]

Object type

[Karol Gregor et al.] Object size
What-Where Auto-Encoder Architecture

Decoder

\[ S^t \]
\[ S^{t-1} \]
\[ S^{t-2} \]

Predicted input

\[ \tilde{W}^1 \]
\[ \tilde{W}^1 \]
\[ \tilde{W}^1 \]
\[ \tilde{W}^2 \]

\[ C_1^t \]
\[ C_1^{t-1} \]
\[ C_1^{t-2} \]
\[ C_2^t \]

Inferred code

\[ f \circ \tilde{W}^1 \]
\[ f \circ \tilde{W}^1 \]
\[ f \circ \tilde{W}^1 \]
\[ \tilde{W}^2 \]

\[ C_1^t \]
\[ C_1^{t-1} \]
\[ C_1^{t-2} \]
\[ C_2^t \]

Predicted code

\[ f \circ \tilde{W}^1 \]
\[ f \circ \tilde{W}^1 \]
\[ f \circ \tilde{W}^1 \]
\[ \tilde{W}^2 \]

\[ C_1^t \]
\[ C_1^{t-1} \]
\[ C_1^{t-2} \]
\[ C_2^t \]

Input

Encoder

\[ S^t \]
\[ S^{t-1} \]
\[ S^{t-2} \]

\[ f \circ \tilde{W}^1 \]
\[ f \circ \tilde{W}^1 \]
\[ f \circ \tilde{W}^1 \]
\[ \tilde{W}^2 \]

\[ C_1^t \]
\[ C_1^{t-1} \]
\[ C_1^{t-2} \]
\[ C_2^t \]
Low-Level Filters Connected to Each Complex Cell

C1
(where)

C2
(what)
Generating images

Input