Learning Invariant Feature Hierarchies

Yann LeCun
The Courant Institute of Mathematical Sciences
Center For Neural Science
New York University

collaborators:
Y-Lan Boureau, Rob Fergus,
Karol Gregor, Kevin Jarrett,
Koray Kavukcuoglu, Marc'Aurelio Ranzato
Problem: supervised ConvNets don't work with few labeled samples

On recognition tasks with few labeled samples, deep supervised architectures don't do so well.

Example: Caltech-101 Object Recognition Dataset
- 101 categories of objects (gathered from the web)
- Only 30 training samples per category!

Recognition rates (OUCH!):
- Supervised ConvNet: \(29.0\%\)
- SIFT features + Pyramid Match Kernel SVM: \(64.6\%\)
  [Lazebnik et al. 2006]

When learning the features, there are simply too many parameters to learn in purely supervised mode (or so we thought).
Unlabeled data is usually available in large quantity.

A lot can be learned about the world by just looking at it.

Unsupervised learning captures underlying regularities about the data.

The best way to capture underlying regularities is to learn good representations of the data.

The main idea of Unsupervised Deep Learning:
- Learn each layer one at a time in unsupervised mode.
- Stick a supervised classifier on top.
- Optionally: refine the entire system in supervised mode.

Unsupervised Learning view as Energy-Based Learning.

[Hinton 05, Bengio 06, LeCun 06, Ng 07]
Energy-Based Framework for Unsupervised Learning

GOAL: make $F(Y; W)$ lower around areas of high data density
Energy-Based Framework for Unsupervised Learning

**GOAL:** make $F(Y,W)$ lower around areas of high data density

**ENERGY BEFORE TRAINING**
Energy-Based Framework for Unsupervised Learning

GOAL: make $F(Y; W)$ lower around areas of high data density

ENERGY AFTER TRAINING
GOAL: make $F(Y,W)$ lower around areas of high data density

Training the model by minimizing a loss functional $L[F(\cdot, W)]$
Energy-Based Framework for Unsupervised Learning

**GOAL:** make $F(Y; W)$ lower around areas of high data density

**Contrastive loss**
- Pushes down on the energy of data points
- Pushes on the energy of everything else

$$L(W) = L(F(Y; W), F(\bar{Y}; W))$$

$L(a,b)$: increasing function of $a$, decreasing function of $b.$

$Y$: data point from the training set

$\bar{Y}$: “fantasy” point outside of the region of high data density
Energy-Based Framework for Unsupervised Learning

Contrastive loss

\[ L(W) = L(F(Y; W), F(\bar{Y}; W)) \]
Energy-Based Framework for Unsupervised Learning

Contrastive loss

\[ L(W) = L(F(Y; W), F(\bar{Y}; W)) \]
Probabilistic View:
- Produce a probability density function that:
  - has high value in regions of high sample density
  - has low value everywhere else (integral = 1).

Energy-Based View:
- produce an energy function \( F(Y,W) \) that:
  - has low value in regions of high sample density
  - has high(er) value everywhere else
Energy <-> Probability

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

\[
E(Y, W) \propto -\log P(Y|W)
\]
Example: Image Patches

Learning:
- Make the energy of every “natural image” patch low
- Make the energy of everything else high!

\[ P(Y, W) = \frac{e^{-\beta E(Y, W)}}{\int_y e^{-\beta E(y, W)}} \]
Training an Energy-Based Model to Approximate a Density

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

$$L(Y,W) = E(Y,W) + \frac{1}{\beta} \log \int_y e^{-\beta E(y,W)}$$
Training an Energy-Based Model with Gradient Descent

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

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

Gradient descent:

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

Pushes down on the energy of the samples

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

$$W \leftarrow W - \eta \frac{\partial E(Y, W)}{\partial W} + \eta \int_y P(y|W) \frac{\partial E(y, W)}{\partial W}$$
Contrastive Divergence Trick [Hinton 2000]

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

Pick a sample of low energy $Y'$ near the training sample, and pull up its energy

this digs a trench in the energy surface around the training samples

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

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

Pulls up on the energy $Y'$
**Contrastive Divergence Trick [Hinton 2000]**

- **push down** on the energy of the training sample \( Y \)

- **Pick a sample of low energy** \( Y' \) **near the training sample, and pull up its energy**
  - this digs a trench in the energy surface around the training samples

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

- **Pushes down on the energy of the training sample** \( Y \)
- **Pulls up on the energy** \( Y' \)
Energy-Based Model Framework

Restrict information content of internal representation
- assume that input is reconstructed from code
- inference determines the value of Z and F(Y;W)
MAIN INSIGHT:

- Assume that the input is reconstructed from an internal code $Z$
- Assume that the energy measures the reconstruction error
- Restricting the information content of the code will automatically push up the energy outside of regions of high data density
How do we push up on the energy of everything else?

- **Solution 1: contrastive divergence [Hinton 2000]**
  - Move away from a training sample a bit
  - Push up on that

- **Solution 2: score matching [Hyvarinen]**
  - On the training samples: minimize the gradient of the energy, and maximize the trace of its Hessian.

- **Solution 3: denoising auto-encoder [Vincent & Bengio 2008]**
  - Train the inference dynamics to map noisy samples to clean samples (not really energy based, but simple and efficient)

- **Solution 4: MAIN INSIGHT! [Ranzato, ..., LeCun AI-Stat 2007]**
  - Restrict the information content of the code (features) Z
  - If the code Z can only take a few different configurations, only a correspondingly small number of Ys can be perfectly reconstructed
  - Idea: impose a sparsity prior on Z
  - This is reminiscent of sparse coding [Olshausen & Field 1997]
The Encoder/Decoder Architecture

Each stage is composed of [Hinton 05, Bengio 06, LeCun 06, Ng 07]

- an encoder that produces a feature vector from the input
- a decoder that reconstructs the input from the feature vector

PCA is a special case (linear encoder and decoder)
Deep Learning: Stack of Encoder/Decoders

- Train each stage one after the other
- 1. Train the first stage
Deep Learning: Stack of Encoder/Decoders

- Train each stage one after the other
- 2. Remove the decoder, and train the second Stage
Deep Learning: Stack of Encoder/Decoders

- Train each stage one after the other
- 3. Remove the 2nd stage decoder, and train a supervised classifier on top
- 4. Refine the entire system with supervised learning
  - e.g. using gradient descent / backprop
Training an Encoder/Decoder Module

Define the Energy $F(Y)$ as the reconstruction error
- Example: $F(Y) = || Y - \text{Decoder(Encoder(Y))} ||^2$

Probabilistic Training, given a training set ($Y_1, Y_2, ....$)
- Interpret the energy $F(Y)$ as a $-\log P(Y)$ (unnormalized)
- Train the encoder/decoder to maximize the prob of the data

Train the encoder/decoder so that:
- $F(Y)$ is small in regions of high data density (good reconstruction)
- $F(Y)$ is large in regions of low data density (bad reconstruction)
Encoder-Decoder: feature \( Z \) is a latent variable

**Energy:**

\[
E(Y, Z) = \text{Dist}[Y, \text{Dec}(Z)] + \text{Dist}[Z, \text{Enc}(Y)]
\]

**Inference through minimization or marginalization**

\[
F(Y) = \min_z E(Y, z) \quad \text{or} \quad F(Y) = -\frac{1}{\beta} \log \int_z e^{-\beta E(Y, z)}
\]
Restricted Boltzmann Machines

Y and Z are binary

Enc and Dec are linear

Distance is negative dot product

\[ E(Y, Z) = \text{Dist}[Y, \text{Dec}(Z)] + \text{Dist}[Z, \text{Enc}(Y)] \]

\[ \text{Enc}(Y) = -W.Y \quad \text{Dist}(Z, W.Y) = -\frac{1}{2}Z^T.W.Y \]

\[ \text{Dec}(Y) = -W^T.Z \quad \text{Dist}(Y, E^T.Z) = -\frac{1}{2}Y^T.W^T.Z \]

\[ E(Y, Z) = -Z^T.W.Y \quad F(Y) = -\log \sum_Z e^{Z^T.W.Y} \]
Fig. 1. Pretraining consists of learning a stack of restricted Boltzmann machines (RBMs), each having only one layer of feature detectors. The learned feature activations of one RBM are used as the “data” for training the next RBM in the stack. After the pretraining, the RBMs are “unrolled” to create a deep autoencoder, which is then fine-tuned using backpropagation of error derivatives.
Fig. 2. (A) Top to bottom: Random samples of curves from the test data set; reconstructions produced by the six-dimensional deep autoencoder; reconstructions by “logistic PCA” (θ) using six components; reconstructions by logistic PCA and standard PCA using 18 components. The average squared error per image for the last four rows is 1.44, 7.64, 2.45, 5.90. (B) Top to bottom: A random test image from each class; reconstructions by the 30-dimensional autoencoder; reconstructions by 30-dimensional logistic PCA and standard PCA. The average squared errors for the last three rows are 3.00, 8.01, and 13.87. (C) Top to bottom: Random samples from the test data set; reconstructions by the 30-dimensional autoencoder; reconstructions by 30-dimensional PCA. The average squared errors are 126 and 135.
Non-Linear Dimensionality Reduction: MNIST

[Hinton and Salakhutdinov, Science 2006]

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

[Hinton and Salakhutdinov, Science 2006]

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

- [Torralba, Fergus, Weiss, CVPR 2008]
- 12 closest neighbors under different distance metrics
LabelMe Retrieval Comparison of methods

Size of retrieval set

% of 50 true neighbors in retrieval set

- 32-LSH
- 32-boosting
- 32-RBM
- 1024-gist/kd
- 4096-gist/kd
- 16384-gist
Encoder-Decoder with Sparsity

Energy:

\[ E(Y, Z) = \text{Dist}[Y, \text{Dec}(Z)] + \text{Dist}[Z, \text{Enc}(Y)] + \text{Regularizer}(Z) \]

Inference through minimization or marginalization

\[ F(Y) = \min_{z} E(Y, z) \quad \text{or} \quad F(Y) = -\frac{1}{\beta} \log \int_{z} e^{-\beta E(Y, z)} \]
If the information content of the feature vector is limited (e.g. by imposing sparsity constraints), the energy MUST be large in most of the space.

- Pulling down on the energy of the training samples will necessarily make a groove

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

- Input vectors are reconstructed from feature vectors.
- If few feature configurations are possible, few input vectors can be reconstructed properly
Why Limit the Information Content of the Code?

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

INPUT SPACE

FEATURE SPACE
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.
We are going to impose a sparsity penalty on the code to restrict its information content.

We will allow the code to have higher dimension than the input.

Categories are more easily separable in high-dim sparse feature spaces.
- This is a trick that SVM use: they have one dimension per sample.

Sparse features are optimal when an active feature costs more than an inactive one (zero).
- e.g. neurons that spike consume more energy.
- The brain is about 2% active on average.
2 dimensional toy dataset
- Mixture of 3 Cauchy distrib.
- Visualizing energy surface
  (black = low, white = high)

[Ranzato 's PhD thesis 2009]

<table>
<thead>
<tr>
<th>Encoder</th>
<th>Decoder</th>
<th>Energy</th>
<th>Loss</th>
<th>Pull-up</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCA</td>
<td>autoencoder</td>
<td>sparse coding</td>
<td>K-Means</td>
<td></td>
</tr>
<tr>
<td>1 code unit</td>
<td>3 code units</td>
<td>3 code units</td>
<td>3 code units</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Encoder</th>
<th>Decoder</th>
<th>Energy</th>
<th>Loss</th>
<th>Pull-up</th>
</tr>
</thead>
<tbody>
<tr>
<td>$W'Y$</td>
<td>$WZ$</td>
<td>$|Y - WZ|^2$</td>
<td>$F(Y)$</td>
<td>dimens.</td>
</tr>
<tr>
<td>$\sigma(W_eY)$</td>
<td>$W_dZ$</td>
<td>$|Y - WZ|^2$</td>
<td>$F(Y) + \log \Gamma$</td>
<td>part. func.</td>
</tr>
<tr>
<td>$-\frac{-\frac{1}{2} |Y - WZ|^2 + \lambda</td>
<td>Z</td>
<td>}{-\frac{1}{2} |Y - WZ|^2}$</td>
<td>$-\frac{-\frac{1}{2} |Y - WZ|^2 + \lambda</td>
<td>Z</td>
</tr>
</tbody>
</table>
2 dimensional toy dataset
- spiral

Visualizing energy surface
(black = low, white = high)

<table>
<thead>
<tr>
<th>Method</th>
<th>Encoder</th>
<th>Decoder</th>
<th>Energy</th>
<th>Loss</th>
<th>Pull-up</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCA</td>
<td>$W_Y$</td>
<td>$W_Z$</td>
<td>$|Y - WZ|^2$</td>
<td>$F(Y)$</td>
<td>dimens.</td>
</tr>
<tr>
<td>autoencoder</td>
<td>$\sigma(W_eY)$</td>
<td>$W_dZ$</td>
<td>$|Y - WZ|^2$</td>
<td>$F(Y)$</td>
<td>dimens.</td>
</tr>
<tr>
<td>sparse coding</td>
<td>$\sigma(W_eZ)$</td>
<td>$W_dZ$</td>
<td>$|Y - WZ|^2$</td>
<td>$F(Y)$</td>
<td>sparsity</td>
</tr>
<tr>
<td>K-Means</td>
<td>-</td>
<td>$W_Z$</td>
<td>$|Y - WZ|^2$</td>
<td>$F(Y)$</td>
<td>1-of-N code</td>
</tr>
</tbody>
</table>
**Sparse Decomposition with Linear Reconstruction**

[Olshausen and Field 1997]

\[ \text{Energy}(\text{Input}, \text{Code}) = \| \text{Input} - \text{Decoder}(\text{Code}) \|^2 + \text{Sparsity}(\text{Code}) \]

\[ \text{Energy}(\text{Input}) = \min_{\text{Code}} [ \text{Energy}(\text{Input}, \text{Code}) ] \]

\[ \begin{align*}
L(W) &= \sum_i F(Y^i; W) = \sum_i \left( \min_{Z^i} E(Y^i, Z^i; W) \right)
\end{align*} \]

**Energy:** minimize to infer \( Z \)

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

\[ F(Y^i; W) = \min_z E(Y^i, z; W) \]

**Loss:** minimize to learn \( W \) (the columns of \( W \) are constrained to have norm 1)

\[ \begin{align*}
Yann LeCun
\end{align*} \]
Problem with Sparse Decomposition: It's slow

Inference: Optimal Code = $\text{Arg\_Min\_over\_Code}[\text{Energy}(\text{Input}, \text{Code})]

$$E(Y^i, Z^i; W) = \|Y^i - W_d Z^i\|^2 + \lambda \sum_j |z^i_j|$$

$$F(Y^i; W) = \min_z E(Y^i, z; W)$$

$$Z^i = \text{argmin}_z E(Y^i, z; W)$$

- For each new $Y$, an optimization algorithm must be run to find the corresponding optimal $Z$

- This would be very slow for large scale vision tasks

- Also, the optimal $Z$ are very unstable:
  - A small change in $Y$ can cause a large change in the optimal $Z$
**Solution: Predictive Sparse Decomposition (PSD)**

- **Prediction** the optimal code with a trained encoder

- **Energy** = reconstruction_error + code_prediction_error + code_sparsity

\[
E(Y^i, Z^i; W) = \|Y^i - W_d Z^i\|^2 + \|Z^i - f_e(Y^i)\|^2 + \lambda \sum_j |z^i_j|
\]

\[
f_e(Y^i) = D \tanh(W_e Y)
\]

[Kavukcuoglu, Ranzato, LeCun, 2009]
Inference by gradient descent starting from the encoder output

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

\[ Z^i = \text{argmin}_z E(Y^i, z; W) \]
Learning by minimizing the average energy of the training data with respect to \( W_d \) and \( W_e \).

**Loss function:**

\[
L(W) = \sum_i F(Y^i; W)
\]

\[
F(Y^i; W) = \min_z E(Y^i, z; W)
\]
PSD: Learning Algorithm

1. Initialize $Z = \text{Encoder}(Y)$
2. Find $Z$ that minimizes the energy function
3. Update the Decoder basis functions to reduce reconstruction error
4. Update Encoder parameters to reduce prediction error

Repeat with next training sample
• PSD trained on handwritten digits: decoder filters are “parts” (strokes).

• Any digit can be reconstructed as a linear combination of a small number of these “parts”.

Yann LeCun
Basis functions are like Gabor filters (like receptive fields in V1 neurons)

256 filters of size 12x12

Trained on natural image patches from the Berkeley dataset

Encoder is linear-tanh-diagonal
Supervised Linear Classifier trained on 200 trained sparse features

Red: linear-tanh-diagonal encoder; Blue: linear encoder
Learned Features on natural patches: V1-like receptive fields
Learned Features: V1-like receptive fields

- 12x12 filters
- 1024 filters
Learning the filters of a ConvNet-like architecture with PSD

1. Train filters on images patches with PSD
2. Plug the filters into a ConvNet architecture
3. Train a supervised classifier on top
“Modern” Object Recognition Architecture in Computer Vision

Example:
- Edges + Rectification + Histograms + SVM [Dalal & Triggs 2005]
- SIFT + classification

Fixed Features + “shallow” classifier
“State of the Art” architecture for object recognition

Example:
- SIFT features with Spatial Pyramid Match Kernel SVM [Lazebnik et al. 2006]

Fixed Features + unsupervised features + “shallow” classifier
Can't we get the same results with (deep) learning?

Stacking multiple stages of feature extraction/pooling.

Creates a hierarchy of features

ConvNets and SIFT+PMK-SVM architectures are conceptually similar

Can deep learning make a ConvNet match the performance of SIFT+PNK-SVM?
How well do PSD features work on Caltech-101?

Recognition Architecture

Filter Bank → Non-Linearity → Spatial Pooling → Classifier

SVM
Procedure for a single-stage system

1. Pre-process images
   - remove mean, high-pass filter, normalize contrast

2. Train encoder-decoder on 9x9 image patches

3. use the filters in a recognition architecture
   - Apply the filters to the whole image
   - Apply the tanh and D scaling
   - Add more non-linearities (rectification, normalization)
   - Add a spatial pooling layer

4. Train a supervised classifier on top
   - Multinomial Logistic Regression or Pyramid Match Kernel SVM
Using PSD Features for Recognition

- 64 filters on 9x9 patches trained with PSD
  - with Linear-Sigmoid-Diagonal Encoder

weights $\pm 0.2828 - 0.3043$
Feature Extraction

Convolution/sigmoid layer: filter bank? Learning, fixed Gabors?
Feature Extraction

Convolution/sigmoid layer: filter bank? Learning, fixed Gabors?

Pinto, Cox and DiCarlo, PloS 08
Feature Extraction

- **C** Convolution/sigmoid layer: filter bank? Learning, fixed Gabors?
- **Abs** Rectification layer: needed?

Pinto, Cox and DiCarlo, PloS 08
Feature Extraction

- **C** Convolution/sigmoid layer: filter bank? Learning, fixed Gabors?
- **Abs** Rectification layer: needed?

Pinto, Cox and DiCarlo, PloS 08
Feature Extraction

- **C** Convolution/sigmoid layer: filter bank? Learning, fixed Gabors?
- **Abs** Rectification layer: needed?

\[ C \quad Abs \quad \text{Local Contrast} \]

\[ \frac{x - \mu}{\max(t, \sigma)} \]

Pinto, Cox and DiCarlo, PloS 08
Feature Extraction

- **C** Convolution/sigmoid layer: filter bank? Learning, fixed Gabors?
- **Abs** Rectification layer: needed?
- **N** Normalization layer: needed?

**Local Contrast Normalization Layer**

\[
\frac{x - \mu}{\max(t, \sigma)}
\]
Feature Extraction

- **C** Convolution/sigmoid layer: filter bank? Learning, fixed Gabors?
- **Abs** Rectification layer: needed?
- **N** Normalization layer: needed?

Pinto, Cox and DiCarlo, PloS 08
Feature Extraction

- **C** Convolution/sigmoid layer: filter bank? Learning, fixed Gabors?
- **Abs** Rectification layer: needed?
- **N** Normalization layer: needed?

![Diagram](image)
Feature Extraction

- **C** Convolution/sigmoid layer: filter bank? Learning, fixed Gabors?
- **Abs** Rectification layer: needed?
- **N** Normalization layer: needed?
- **P** Pooling down-sampling layer: average or max?

```
Pooling Down-Sampling Layer
```
Feature Extraction

- **C** Convolution/sigmoid layer: filter bank? Learning, fixed Gabors?
- **Abs** Rectification layer: needed?
- **N** Normalization layer: needed?
- **P** Pooling down-sampling layer: average or max?
Feature Extraction

- **C** Convolution/sigmoid layer: filter bank? Learning, fixed Gabors?
- **Abs** Rectification layer: needed?
- **N** Normalization layer: needed?
- **P** Pooling down-sampling layer: average or max?

This is one stage of feature extraction.
Training Protocol

- **Training**
  - Logistic Regression on Random Features: \( R \)
  - Logistic Regression on PSD features: \( U \)
  - Refinement of whole net from random with backprop: \( R^+ \)
  - Refinement of whole net starting from PSD filters: \( U^+ \)

- **Classifier**
  - Multinomial Logistic Regression or Pyramid Match Kernel SVM
Using PSD Features for Recognition

\[
[64.F^{9 \times 9}_{\text{CSG}} - R/N/P^{5 \times 5}] - \log_{\text{reg}}
\]

<table>
<thead>
<tr>
<th>R/N/P</th>
<th>( R_{\text{abs}} - N - P_A )</th>
<th>( R_{\text{abs}} - P_A )</th>
<th>( N - P_M )</th>
<th>( N - P_A )</th>
<th>( P_A )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( U^+ )</td>
<td>54.2%</td>
<td>50.0%</td>
<td>44.3%</td>
<td>18.5%</td>
<td>14.5%</td>
</tr>
<tr>
<td>( R^+ )</td>
<td>54.8%</td>
<td>47.0%</td>
<td>38.0%</td>
<td>16.3%</td>
<td>14.3%</td>
</tr>
<tr>
<td>( U )</td>
<td>52.2%</td>
<td>43.3(±1.6)%</td>
<td>44.0%</td>
<td>17.2%</td>
<td>13.4%</td>
</tr>
<tr>
<td>( R )</td>
<td>53.3%</td>
<td>31.7%</td>
<td>32.1%</td>
<td>15.3%</td>
<td>12.1(±2.2)%</td>
</tr>
</tbody>
</table>

\[
[64.F^{9 \times 9}_{\text{CSG}} - R/N/P^{5 \times 5}] - \text{PMK}
\]

| U | 65.0\% |

\[
[96.F^{9 \times 9}_{\text{CSG}} - R/N/P^{5 \times 5}] - \text{PCA - lin_{svm}}
\]

| U | 58.0\% |

96.Gabors - PCA - lin_{svm} (Pinto and DiCarlo 2006)

| Gabors | 59.0\% |

SIFT - PMK (Lazebnik et al. CVPR 2006)

| Gabors | 64.6\% |
Using PSD Features for Recognition

- Rectification makes a **huge difference**:
  - 14.5% -> 50.0%, without normalization
  - 44.3% -> 54.2% with normalization

- **Normalization makes a difference**:
  - 50.0 -> 54.2

- Unsupervised pretraining makes small difference

- **PSD works just as well as SIFT**

- **Random filters work as well as anything!**
  - If rectification/normalization is present

- **PMK_SVM classifier works a lot better than multinomial log_reg on low-level features**
  - 52.2% -> 65.0%
Approximated Sparse Features Predicted by PSD give better recognition results than Optimal Sparse Features computed with Feature Sign!

- PSD features are more stable.

Feature Sign (FS) is an optimization method for computing sparse codes [Lee...Ng 2006]
Approximated Sparse Features Predicted by PSD give better recognition results than Optimal Sparse Features computed with Feature Sign!

Because PSD features are more stable. Feature obtained through sparse optimization can change a lot with small changes of the input.

How many features change sign in patches from successive video frames (a,b), versus patches from random frame pairs (c)
PSD features are much cheaper to compute

Computing PSD features is hundreds of times cheaper than Feature Sign.
How Many 9x9 PSD features do we need?

- Accuracy increases slowly past 64 filters.
1. Train stage-1 filters with PSD on patches from natural images
2. Compute stage-1 features on training set
3. Train state-2 filters with PSD on stage-1 feature patches
4. Compute stage-2 features on training set
5. Train linear classifier on stage-2 features
6. Refine entire network with supervised gradient descent

What are the effects of the non-linearities and unsupervised pretraining?
Multistage Hubel-Wiesel Architecture on Caltech-101

Y (luminance)

CONVOLUTIONS (9x9)

MAX/SUBSAMPLING (4x4)

CONVOLUTIONS (9x9)

MAX/SUBSAMPLING (5x5)

INPUT 3@140x140

32@132x132

32@33x33

64@25x25

64@5x5
Multistage Hubel-Wiesel Architecture

**Image Preprocessing:**
- High-pass filter, local contrast normalization (divisive)

**First Stage:**
- Filters: 64 9x9 kernels producing 64 feature maps
- Pooling: 10x10 averaging with 5x5 subsampling

**Second Stage:**
- Filters: 4096 9x9 kernels producing 256 feature maps
- Pooling: 6x6 averaging with 3x3 subsampling
- Features: 256 feature maps of size 4x4 (4096 features)

**Classifier Stage:**
- Multinomial logistic regression

**Number of parameters:**
- Roughly 750,000
### Multistage Hubel-Wiesel Architecture on Caltech-101

#### Single Stage System: \([64.F_{\text{CSG}}^{9\times9} - R/N/P_{5\times5}] - \text{log_reg}\)

<table>
<thead>
<tr>
<th>R/N/P</th>
<th>(R_{\text{abs}} - N - P_A)</th>
<th>(R_{\text{abs}} - P_A)</th>
<th>(N - P_M)</th>
<th>(N - P_A)</th>
<th>(P_A)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(U^+)</td>
<td>54.2%</td>
<td>50.0%</td>
<td>44.3%</td>
<td>18.5%</td>
<td>14.5%</td>
</tr>
<tr>
<td>(R^+)</td>
<td>54.8%</td>
<td>47.0%</td>
<td>38.0%</td>
<td>16.3%</td>
<td>14.3%</td>
</tr>
<tr>
<td>(U)</td>
<td>52.2%</td>
<td>43.3% (±1.6)</td>
<td>44.0%</td>
<td>17.2%</td>
<td>13.4%</td>
</tr>
<tr>
<td>(R)</td>
<td>53.3%</td>
<td>31.7%</td>
<td>32.1%</td>
<td>15.3%</td>
<td>12.1% (±2.2)</td>
</tr>
<tr>
<td>(G)</td>
<td>52.3%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Two Stage System: \([64.F_{\text{CSG}}^{9\times9} - R/N/P_{5\times5}] - [256.F_{\text{CSG}}^{9\times9} - R/N/P_{4\times4}] - \text{log_reg}\)

<table>
<thead>
<tr>
<th>R/N/P</th>
<th>(R_{\text{abs}} - N - P_A)</th>
<th>(R_{\text{abs}} - P_A)</th>
<th>(N - P_M)</th>
<th>(N - P_A)</th>
<th>(P_A)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(U^+U^+)</td>
<td>65.5%</td>
<td>60.5%</td>
<td>61.0%</td>
<td>34.0%</td>
<td>32.0%</td>
</tr>
<tr>
<td>(R^+R^+)</td>
<td>64.7%</td>
<td>59.5%</td>
<td>60.0%</td>
<td>31.0%</td>
<td>29.7%</td>
</tr>
<tr>
<td>(UU)</td>
<td>63.7%</td>
<td>46.7%</td>
<td>56.0%</td>
<td>23.1%</td>
<td>9.1%</td>
</tr>
<tr>
<td>(RR)</td>
<td>62.9%</td>
<td>33.7% (±1.5)</td>
<td>37.6% (±1.9)</td>
<td>19.6%</td>
<td>8.8%</td>
</tr>
<tr>
<td>(GT)</td>
<td>55.8%</td>
<td></td>
<td></td>
<td></td>
<td>like HMAX model</td>
</tr>
</tbody>
</table>

#### Single Stage: \([64.F_{\text{CSG}}^{9\times9} - R/N/P_{5\times5}] - \text{PMK-SVM}\)

| \(U\) | 64.0% |

#### Two Stages: \([64.F_{\text{CSG}}^{9\times9} - R/N/P_{5\times5}] - [256.F_{\text{CSG}}^{9\times9} - R/N] - \text{PMK-SVM}\)

| \(UU\) | 52.8% |
Two-Stage Result Analysis

- Second Stage + logistic regression = PMK_SVM
- Unsupervised pre-training doesn't help much :-(
- Random filters work amazingly well with normalization
- Supervised global refinement helps a bit
- The best system is really cheap
- Either use rectification and average pooling or no rectification and max pooling.
Multistage Hubel-Wiesel Architecture: Filters

- **Stage 1**
  - After PSD
    - Weights: -0.2232 - 0.2075
  - Stage 2
    - After supervised refinement
      - Weights: -0.2828 - 0.3043

- **Stage 2**
  - Weights: -0.0778 - 0.064
  - Weights: -0.0929 - 0.0784
MNIST dataset

- 10 classes and up to 60,000 training samples per class
Architecture

U⁺U⁺: 0.53% error (this is a record on the undistorted MNIST!)

Comparison: $RR$ versus $UU$ and $R⁺R⁺$
Why Random Filters Work?
Small NORB dataset

- 5 classes and up to 24,300 training samples per class
NORB Generic Object Recognition Dataset

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
Small NORB dataset

Architecture

Two Stages

Error Rate (log scale) VS. Number Training Samples (log scale)
Unsupervised PSD ignores the spatial pooling step.

Could we devise a similar method that learns the pooling layer as well?

Idea [Hyvarinen & Hoyer 2001]: 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

\[
\sum_j \left( \sqrt{\sum_{i \in P_j} Z_i^2} \right)
\]
Using an idea from Hyvarinen: topographic square pooling (subspace ICA)

1. Apply filters on a patch (with suitable non-linearity)
2. Arrange filter outputs on a 2D plane
3. Square filter outputs
4. Minimize sqrt of sum of blocks of squared filter outputs

Overall Sparsity term: \( \sum_{i=1}^{K} \sqrt{v_i^2} \)

\( v_1^2 = \sum_{j \in P_1} (w_jz_j)^2 \)

\( v_k^2 = \sum_{j \in P_k} (w_jz_j)^2 \)

Units in the code Z

Define pools and enforce sparsity across pools
The filters arrange themselves spontaneously so that similar filters enter the same pool.

The pooling units can be seen as complex cells

They are invariant to local transformations of the input

For some it's translations, for others rotations, or other transformations.
Pinwheels?
Invariance Properties Compared to SIFT

- Measure distance between feature vectors (128 dimensions) of 16x16 patches from natural images
  - Left: normalized distance as a function of translation
  - Right: normalized distance as a function of translation when one patch is rotated 25 degrees.

- Topographic PSD features are more invariant than SIFT
**Learning Invariant Features**

- **Recognition Architecture**
  - $\rightarrow$ HPF/LCN $\rightarrow$ filters $\rightarrow$ tanh $\rightarrow$ sqr $\rightarrow$ pooling $\rightarrow$ sqrt $\rightarrow$ Classifier
  - Block pooling plays the same role as rectification
Recognition Accuracy on Caltech 101

- A/B Comparison with SIFT (128x34x34 descriptors)
- 32x16 topographic map with 16x16 filters
- Pooling performed over 6x6 with 2x2 subsampling
- 128 dimensional feature vector per 16x16 patch
- Feature vector computed every 4x4 pixels (128x34x34 feature maps)
- Resulting feature maps are spatially smoothed

<table>
<thead>
<tr>
<th>Method</th>
<th>Av. Accuracy/Class (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>local norm$<em>{5 \times 5}$ + boxcar$</em>{5 \times 5}$ + PCA$_{3060}$ + linear SVM</td>
<td></td>
</tr>
<tr>
<td>IPSD (24x24)</td>
<td>50.9</td>
</tr>
<tr>
<td>SIFT (24x24) (non rot. inv.)</td>
<td>51.2</td>
</tr>
<tr>
<td>SIFT (24x24) (rot. inv.)</td>
<td>45.2</td>
</tr>
<tr>
<td>Serre et al. features [25]</td>
<td>47.1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method</th>
<th>Av. Accuracy/Class (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>local norm$_{9 \times 9}$ + Spatial Pyramid Match Kernel SVM</td>
<td></td>
</tr>
<tr>
<td>SIFT [11]</td>
<td>64.6</td>
</tr>
<tr>
<td>IPSD (34x34)</td>
<td>59.6</td>
</tr>
<tr>
<td>IPSD (56x56)</td>
<td>62.6</td>
</tr>
<tr>
<td>IPSD (120x120)</td>
<td>65.5</td>
</tr>
</tbody>
</table>
Recognition Accuracy on Tiny Images & MNIST

- A/B Comparison with SIFT (128x5x5 descriptors)
- 32x16 topographic map with 16x16 filters.

<table>
<thead>
<tr>
<th>Performance on Tiny Images Dataset</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Method</strong></td>
</tr>
<tr>
<td>IPSD (5x5)</td>
</tr>
<tr>
<td>SIFT (5x5) (non rot. inv.)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Performance on MNIST Dataset</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Method</strong></td>
</tr>
<tr>
<td>IPSD (5x5)</td>
</tr>
<tr>
<td>SIFT (5x5) (non rot. inv.)</td>
</tr>
</tbody>
</table>
The End