The Unreasonable Effectiveness of Deep Learning

Yann LeCun
Facebook AI Research & Center for Data Science, NYU
yann@cs.nyu.edu
http://yann.lecun.com
The traditional model of pattern recognition (since the late 50's)
- Fixed/engineered features (or fixed kernel) + trainable classifier

- Perceptron
"Classic" architecture for pattern recognition

- Speech recognition: 1990-2011
- Object Recognition: 2005-2012
- Handwriting recognition (long ago)
- Graphical model has latent variables (locations of parts)

Fixed unsupervised supervised fixed

MFCC SIFT, HoG K-Means Gaussians Cuboids Sparse Coding Pooling

Low-level Features Mid-level Features parts, phones, characters Object, Utterance, word

Graphical Model
“Deep” architecture for pattern recognition

- Speech, and Object recognition: since 2011/2012
- Handwriting recognition: since the early 1990s
- Convolutional Net with optional Graphical Model on top
- Trained purely supervised
- Graphical model has latent variables (locations of parts)
Future Systems: deep learning + structured prediction

Globally-trained deep architecture

- Handwriting recognition: since the mid 1990s
- Speech Recognition: since 2011
- All the modules are trained with a combination of unsupervised and supervised learning

End-to-end training == deep structured prediction

Unsup + supervised

Filters + ReLU

Poolings

Unsup + supervised

Filters + ReLU

Poolings

Unsup + supervised

Filters + ReLU

Graphical Model

Low-level Features

Mid-level Features

parts, phones, characters

Object, Utterance, word
Deep Learning = Learning Hierarchical Representations

It's deep if it has more than one stage of non-linear feature transformation

Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]
Trainable Feature Hierarchy

- Hierarchy of representations with increasing level of abstraction
- Each stage is a kind of trainable feature transform
- Image recognition
  - Pixel → edge → texton → motif → part → object
- Text
  - Character → word → word group → clause → sentence → story
- Speech
  - Sample → spectral band → sound → ... → phone → phoneme → word
How do we learn representations of the perceptual world?

- How can a perceptual system build itself by looking at the world?
- How much prior structure is necessary

**ML/AI:** how do we learn features or feature hierarchies?

- What is the fundamental principle? What is the learning algorithm? What is the architecture?

**Neuroscience:** how does the cortex learn perception?

- Does the cortex “run” a single, general learning algorithm? (or a small number of them)

**CogSci:** how does the mind learn abstract concepts on top of less abstract ones?

Deep Learning addresses the problem of learning hierarchical representations with a single algorithm or perhaps with a few algorithms.
The Mammalian Visual Cortex is Hierarchical

- The ventral (recognition) pathway in the visual cortex has multiple stages
- Retina - LGN - V1 - V2 - V4 - PIT - AIT ....
- Lots of intermediate representations

[picture from Simon Thorpe]

[picture from Gallant & Van Essen]
Ventral pathway = “what”
- dorsal pathway = “where”
- It's hierarchical
- There is feedback
- There is motion processing
- Learning is mostly unsupervised
- It does recognition, localization, navigation, grasping.....

[Gallant & Van Essen]
Let's be inspired by nature, but not too much

- It's nice to imitate Nature,
- But we also need to understand
  - How do we know which details are important?
  - Which details are merely the result of evolution, and the constraints of biochemistry?
- For airplanes, we developed aerodynamics and compressible fluid dynamics.
  - We figured that feathers and wing flapping weren't crucial.

**QUESTION:** What is the equivalent of aerodynamics for understanding intelligence?

L'Avion III de Clément Ader, 1897
(Musée du CNAM, Paris)
His “Eole” took off from the ground in 1890, 13 years before the Wright Brothers, but you probably never heard of it (unless you are French).
Shallow vs Deep == lookup table vs multi-step algorithm

“shallow & wide” vs “deep and narrow” == “more memory” vs “more time”

- Look-up table vs algorithm
- Few functions can be computed in two steps without an exponentially large lookup table
- Using more than 2 steps can reduce the “memory” by an exponential factor.
Which Models are Deep?

- **2-layer models are not deep (even if you train the first layer)**
  - Because there is no feature hierarchy

- **Neural nets with 1 hidden layer are not deep**
  - Layer1: kernels; layer2: linear
  - The first layer is “trained” in with the simplest unsupervised method ever devised: using the samples as templates for the kernel functions.
  - “glorified template matching”

- **SVMs and Kernel methods are not deep**

- **Classification trees are not deep**
  - No hierarchy of features. All decisions are made in the input space
What Are Good Feature?
**Basic Idea for Invariant Feature Learning**

- **Embed the input non-linearly into a high(er) dimensional space**
  - In the new space, things that were non separable may become separable

- **Pool regions of the new space together**
  - Bringing together things that are semantically similar. Like pooling.

```
Input ➔ Non-Linear Function ➔ Pooling Or Aggregation ➔ Stable/invariant features
```

- **Stable/invariant features**
  - Unstable/non-smooth features
  - high-dim

---

Y LeCun
Sparse Non-Linear Expansion $\rightarrow$ Pooling

- Use clustering to break things apart, pool together similar things
- Clustering, Quantization, Sparse Coding
- Pooling, Aggregation
Overall Architecture: multiple stages of Normalization $\rightarrow$ Filter Bank $\rightarrow$ Non-Linearity $\rightarrow$ Pooling

- **Normalization**: variation on whitening (optional)
  - Subtractive: average removal, high pass filtering
  - Divisive: local contrast normalization, variance normalization
- **Filter Bank**: dimension expansion, projection on overcomplete basis
- **Non-Linearity**: sparsification, saturation, lateral inhibition...
  - Rectification (ReLU), Component-wise shrinkage, tanh, ...

\[
ReLU(x) = \max(x, 0)
\]

- **Pooling**: aggregation over space or feature type
  - Max, Lp norm, log prob.

\[
\text{MAX} : \max_i(X_i); \quad L_p : \sqrt[p]{\sum X_i^p}; \quad \text{PROB} : \frac{1}{b} \log \left( \sum_i e^{bX_i} \right)
\]
Deep Nets with ReLUs and Max Pooling

- Stack of linear transforms interspersed with Max operators
- Point-wise ReLUs:

\[ \text{ReLU}(x) = \max(x, 0) \]

- Max Pooling
  - “switches” from one layer to the next
To compute all the derivatives, we use a backward sweep called the **back-propagation algorithm** that uses the recurrence equation for $\frac{\partial E}{\partial X_i}$

- $\frac{\partial E}{\partial X_n} = \frac{\partial C(X_n, Y)}{\partial X_n}$
- $\frac{\partial E}{\partial X_{n-1}} = \frac{\partial E}{\partial X_n} \frac{\partial F_n(X_{n-1}, W_n)}{\partial X_{n-1}}$
- $\frac{\partial E}{\partial W_n} = \frac{\partial E}{\partial X_n} \frac{\partial F_n(X_{n-1}, W_n)}{\partial W_n}$
- $\frac{\partial E}{\partial X_{n-2}} = \frac{\partial E}{\partial X_{n-1}} \frac{\partial F_{n-1}(X_{n-2}, W_{n-1})}{\partial X_{n-2}}$
- $\frac{\partial E}{\partial W_{n-1}} = \frac{\partial E}{\partial X_{n-1}} \frac{\partial F_{n-1}(X_{n-2}, W_{n-1})}{\partial W_{n-1}}$
- ...etc, until we reach the first module.
- We now have all the $\frac{\partial E}{\partial W_i}$ for $i \in [1, n]$. 
1-1-1 network
- \( Y = W_1 \times W_2 \times X \)

trained to compute the identity function with quadratic loss
- Single sample \( X = 1, \ Y = 1 \) \( L(W) = (1 - W_1 \times W_2)^2 \)

Solution

Saddle point

Solution
Single output:

\[ \hat{Y} = \sum_{P} \delta_{P}(W, X)(\prod_{(ij) \in P} W_{ij})X_{P_{start}} \]

- W_{ij}: weight from j to i
- P: path in network from input to output
  - P=(3, (14, 3), (22, 14), (31, 22))
- d_{i}: 1 if ReLU i is linear, 0 if saturated.
- X_{P_{start}}: input unit for path P.

\[ \hat{Y} = \sum_{P} \delta_{P}(W, X)(\prod_{(ij) \in P} W_{ij})X_{P_{start}} \]

- D_{P}(W, X): 1 if path P is “active”, 0 if inactive
- Input-output function is piece-wise linear
- Polynomial in W with random coefficients
Training sample: \((X_i, Y_i)\) \(k=1\) to \(K\)

Objective function (with margin-type loss = ReLU)

\[
L(W) = \sum_k \text{ReLU} \left( 1 - Y^k \sum_P \delta_P(W, X^k)(\prod_{(ij)\in P} W_{ij})X_{P_{start}}^k \right)
\]

\[
L(W) = \sum_k \sum_P (X_{P_{start}}^k Y^k)\delta_P(W, X^k)(\prod_{(ij)\in P} W_{ij})
\]

\[
L(W) = \sum_P \left[ \sum_k (X_{P_{start}}^k Y^k)\delta_P(W, X^k) \right](\prod_{(ij)\in P} W_{ij})
\]

\[
L(W) = \sum_P C_P(X, Y, W)(\prod_{(ij)\in P} W_{ij})
\]

Polynomial in \(W\) of degree \(l\) (number of adaptive layers)

Continuous, piece-wise polynomial with “switched” and partially random coefficients

Coefficients are switched in an out depending on \(W\)
Deep Nets with ReLUs: 
Objective Function is Piecewise Polynomial

If we use a hinge loss, delta now depends on label $Y_k$:

$$L(W) = \sum_{P} C_p(X, Y, W) \left( \prod_{(ij) \in P} W_{ij} \right)$$

Piecewise polynomial in $W$ with random coefficients

A lot is known about the distribution of critical points of polynomials on the sphere with random (Gaussian) coefficients [Ben Arous et al.]

- High-order spherical spin glasses
- Random matrix theory

![Histogram of minima]
Train 2-layer nets on scaled-down MNIST (10x10) from multiple initial conditions. Measure loss on test set.

[Choromanska, Henaff, Mathieu, Ben Arous, LeCun 2015]
Spherical Spin Glass theory

Distribution of critical points (saddle points, minima, maxima)

K = number of negative eigenvalues of Hessian (K=0 → minimum)

Zoomed:
Convolutional Networks
Convolutional Network

Filter Bank + non-linearity

Pooling

Filter Bank + non-linearity

Pooling

Filter Bank + non-linearity

[LeCun et al. NIPS 1989]
Early Hierarchical Feature Models for Vision

- [Hubel & Wiesel 1962]:
  - **simple cells** detect local features
  - **complex cells** “pool” the outputs of simple cells within a retinotopic neighborhood.

Cognitron & Neocognitron [Fukushima 1974-1982]
The Convolutional Net Model
(Multistage Hubel-Wiesel system)

Local Divisive Normalization
Convolutions w/ filter bank: 20x7x7 kernels
Pooling: 20x4x4 kernels
Convs: 100x7x7 kernels
Pooling: 20x4x4 kernels
Convs: 800x7x7 kernels
Linear Classifier

“Simple cells”
“Complex cells”

Multiple convolutions
pooling subsampling
Retinotopic Feature Maps

Training is supervised
With stochastic gradient descent

Input Image
1x500x500
Normalized Image
1x500x500
C1: 20x494x494
S2: 20x123x123
C3: 20x117x117
S4: 20x29x29
C5: 200x23x23
F6: Nx23x23

Categories / Positions

[LeCun et al. 89]
[LeCun et al. 98]
Convolutional Network (ConvNet)

- **Input**: 83x83
- **Layer 1**: 64x75x75
  - 9x9 convolution (64 kernels)
- **Layer 2**: 64@14x14
  - 10x10 pooling, 5x5 subsampling
- **Layer 3**: 256@6x6
  - 9x9 convolution (4096 kernels)
- **Layer 4**: 256@1x1
  - 6x6 pooling, 4x4 subsampling
- **Output**: 101

- **Non-Linearity**: half-wave rectification (ReLU), shrinkage function, sigmoid
- **Pooling**: max, average, L1, L2, log-sum-exp
Convolutional Network (vintage 1990)

filters → tanh → average-tanh → filters → tanh → average-tanh → filters → tanh

Curved manifold

Flatter manifold
Running on a 486 PC with an AT&T DSP32C add-on board (20 Mflops!)
“Mainstream” object recognition pipeline 2006-2012: somewhat similar to ConvNets

- Fixed Features + unsupervised mid-level features + simple classifier
  - SIFT + Vector Quantization + Pyramid pooling + SVM
    - [Lazebnik et al. CVPR 2006]
  - SIFT + Local Sparse Coding Macrofeatures + Pyramid pooling + SVM
    - [Boureau et al. ICCV 2011]
  - SIFT + Fisher Vectors + Deformable Parts Pooling + SVM
    - [Perronin et al. 2012]
Global (end-to-end) Training. Integrating Deep Learning with Structured Prediction. Energy-Based Models

Making every single module in the system trainable.

Every module is trained simultaneously so as to optimize a global loss function.

Includes the feature extractor, the recognizer, and the contextual post-processor (graphical model).

Problem: back-propagating gradients through the graphical model.
Highly popular methods in the Machine Learning and Natural Language Processing Communities have their roots in Speech and Handwriting Recognition

- Structured Perceptron, Conditional Random Fields, and related learning models for “structured prediction” are descendants of discriminative learning methods for speech recognition and word-level handwriting recognition methods from the early 90's

A Tutorial and Energy-Based Learning:
- [LeCun & al., 2006]

Discriminative Training for “Structured Output” models
- The whole literature on discriminative speech recognition [1987-]
- Graph Transformer Networks [LeCun & al. Proc IEEE 1998]
- Structured Perceptron [Collins 2001]
- Conditional Random Fields [Lafferty & al 2001]
Energy-Based Models for Decision Making

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

\[ Y^* = \arg\min_{Y \in \mathcal{Y}} E(Y, X). \]

**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.
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, belief propagation, gradient decent.....
Energies are uncalibrated
- The energies of two separately-trained systems cannot be combined
- The energies are uncalibrated (measured in arbitrary units)

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.

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

Partition function  Inverse temperature
Deep Learning systems can be assembled into factor graphs

- Energy function is a sum of factors
- Factors can embed whole deep learning systems
- $X$: observed variables (inputs)
- $Z$: never observed (latent variables)
- $Y$: observed on training set (output variables)

Inference is energy minimization (MAP) or free energy minimization (marginalization) over $Z$ and $Y$ given an $X$
Deep Learning systems can be assembled into factor graphs
- Energy function is a sum of factors
- Factors can embed whole deep learning systems
- X: observed variables (inputs)
- Z: never observed (latent variables)
- Y: observed on training set (output variables)

Inference is energy minimization (MAP) or free energy minimization (marginalization) over Z and Y given an X
- \( F(X,Y) = \text{MIN}_z E(X,Y,Z) \)
- \( F(X,Y) = -\log \sum_z \exp[-E(X,Y,Z)] \)
integrated segmentation and recognition of sequences. Each segmentation and recognition hypothesis is a path in a graph inference = finding the shortest path in the interpretation graph.

Un-normalized hierarchical HMMs a.k.a. Graph Transformer Networks

The energy includes “hidden” variables $Z$ whose value is never given to us

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

$$Y^* = \arg\min_{Y \in \mathcal{Y}, Z \in \mathcal{Z}} E(Z, Y, X).$$
The energy includes “hidden” variables $Z$ whose value is never given to us
- We can minimize the energy over those latent variables
- We can also “marginalize” the energy over the latent variables

Minimization over latent variables:

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

Marginalization over latent variables:

$$E(X, Y) = -\frac{1}{\beta} \log \int_{Z \in \mathcal{Z}} e^{-\beta E(Z, Y, X)}$$

Estimation this integral may require some approximations (sampling, variational methods,....)
Example 1: Integrated Training with Sequence Alignment

Spoken word recognition with trainable elastic templates and trainable feature extraction [Driancourt & Bottou 1991, Bottou 1991, Driancourt 1994]
Spoken word recognition with trainable elastic templates and trainable feature extraction [Driancourt & Bottou 1991, Bottou 1991, Driancourt 1994]
Elastic matching using dynamic time warping (Viterbi algorithm on a trellis).
The corresponding EBFG is implicit (it changes for every new sample).
The Oldest Example of Structured Prediction

Trainable Automatic Speech Recognition system with a convolutional net (TDNN) and dynamic time warping (DTW)

The feature extractor and the structured classifier are trained simultaneously in an integrated fashion.

with the LVQ2 Loss:
- Driancourt and Bottou's speech recognizer (1991)
- Bengio's speech recognizer (1992)
- Haffner's speech recognizer (1993)
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.
- **Object Recognition/Scene Parsing**: the segmentation of the image into components (objects, parts, ...)

In general, we will search for the value of the latent variable that allows us to get an answer ($Y$) of smallest energy.
Marginalizing over latent variables instead of minimizing:

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

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

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

\[
Y^* = \arg\min_{Y \in \mathcal{Y}} -\frac{1}{\beta} \log \int_{z \in \mathcal{Z}} e^{-\beta E(z, Y, X)}.
\]

Reduces to traditional minimization when \(\beta \to \infty\)
Training an EBM consists in shaping the energy function so that the energies of the correct answer is lower than the energies of all other answers.

- Training sample: $X =$ image of an animal, $Y =$ “animal”

$$E(\text{animal}, X) < E(\text{y, } X) \forall y \neq \text{animal}$$
Architecture and Loss Function

Family of energy functions
\[ \mathcal{E} = \{ E(W, Y, X) : W \in \mathcal{W} \}. \]

Training set
\[ \hat{\mathcal{S}} = \{(X^i, Y^i) : i = 1 \ldots P \} \]

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

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

Per-sample loss
Desired answer
Energy surface for a given \( X_i \) as \( Y \) varies
Regularizer
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

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?
Examples of Loss Functions: Energy Loss

**Energy Loss**

\[ L_{energy}(Y^i, E(W, Y, X^i)) = E(W, Y^i, X^i). \]

- Simply pushes down on the energy of the correct answer
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:

\[
P(Y | X^i, W) = \frac{\exp(-\beta E(W, Y, X^i))}{\int_{y \in \mathcal{Y}} \exp(-\beta E(W, y, X^i))}.
\]

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

We get the NLL loss by dividing by \( P \) and Beta:

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

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

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

\[
\frac{\partial \mathcal{L}_{\text{nll}}(W, Y^i, X^i)}{\partial W} = \frac{\partial E(W, Y^i, X^i)}{\partial W} - \int_{Y \in \mathcal{Y}} \frac{\partial E(W, Y, X^i)}{\partial W} P(Y|X^i, W),
\]
A probabilistic model is an EBM in which:

- The energy can be integrated over Y (the variable to be predicted)
- The loss function is the negative log-likelihood

Negative Log Likelihood Loss has been used for a long time in many communities for discriminative learning with structured outputs

- Speech recognition: many papers going back to the early 90's [Bengio 92], [Bourlard 94]. They call “Maximum Mutual Information”
- Handwriting recognition [Bengio LeCun 94], [LeCun et al. 98]
- Bio-informatics [Haussler]
- Conditional Random Fields [Lafferty et al. 2001]
- Lots more......
- In all the above cases, it was used with non-linearly parameterized energies.
Perceptron Loss \cite{LeCun1998}, \cite{Collins2002}

- 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.
- This is often called "\textbf{discriminative Viterbi training}" in the speech and handwriting literature

\[
L_{\text{perceptron}}(Y^i, E(W, Y, X^i)) = E(W, Y^i, X^i) - \min_{Y \in \mathcal{Y}} E(W, Y, X^i).
\]
Perceptron Loss for Binary Classification

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

**Energy:**

\[ E(W, Y, X) = -YG_W(X), \]

**Inference:**

\[ Y^* = \arg\min_{Y \in \{-1, 1\}} -YG_W(X) = \text{sign}(G_W(X)). \]

**Loss:**

\[ \mathcal{L}_{\text{perceptron}}(W, S) = \frac{1}{P} \sum_{i=1}^{P} (\text{sign}(G_W(X^i)) - Y^i) G_W(X^i). \]

**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 \( G_W(X) \) is linear in \( W \):

\[ E(W, Y, X) = -YW^T \Phi(X) \]

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

$$\bar{Y}^i = \arg\min_{Y \in \mathcal{Y} \text{ and } Y \neq Y^i} E(W, Y, X^i).$$

(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 $\epsilon$ away from the correct answer:

$$\bar{Y}^i = \arg\min_{Y \in \mathcal{Y}, \|Y - Y^i\| > \epsilon} E(W, Y, X^i).$$

(9)
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 SVMs

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

**Log Loss**
- "soft hinge" loss
- With the linearly-parameterized binary classifier architecture, we get linear Logistic Regression
Examples of Margin Losses: Square-Square Loss

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

Square-Square Loss

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

Learning $Y = X^2$
Other Margin-Like Losses

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

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

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

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

\[ \sigma(x) = \frac{1}{1 + e^{-x}} \]

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

\[ L_{\text{sq-exp}}(W, Y^i, X^i) = E(W, Y^i, X^i)^2 + \gamma e^{-E(W, Y^i, X^i)}, \]
What Make a “Good” Loss Function

Good and bad loss functions

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

Slightly more general form:

$$L(W, X^i, Y^i) = \sum_y H \left(E(W, Y^i, X^i) - E(W, y, X^i) + C(Y^i, y)\right)$$
Recognizing Words With Deep Learning And Structured Prediction
Making every single module in the system trainable.

Every module is trained simultaneously so as to optimize a global loss function.

Includes the feature extractor, the recognizer, and the contextual post-processor (graphical model).

Problem: back-propagating gradients through the graphical model.
“Shallow” Structured Prediction

Energy function is linear in the parameters

\[ E(X, Y, Z) = \sum_i W_i^T h_i(X, Y, Z) \]

with the NLL Loss:
- **Conditional Random Field**
  [Lafferty, McCallum, Pereira 2001]

with Hinge Loss:
- **Max Margin Markov Nets** and **Latent SVM**
  [Taskar, Altun, Hofmann...]

with Perceptron Loss
- **Structured Perceptron**
  [Collins...]

Input: \( X \)

Features:
- \( h(X, Y, Z) \)

Latent Vars:
- \( Z_1, Z_2, Z_3 \)

Outputs:
- \( Y_1, Y_2, Y_3, Y_4 \)

Params:
- \( W_1, W_2, W_3 \)
Deep Structured Prediction

Energy function is linear in the parameters

\[ E(X, Y, Z) = \sum_i g_i(X, Y, Z, W_i) \]

Graph Transformer Networks
- [LeCun, Bottou, Bengio, Haffner 97, 98]
- NLL loss
- Perceptron loss

Input:
- \( X \)

Latent Vars:
- \( Z_1 \)
- \( Z_2 \)
- \( Z_3 \)

Outputs:
- \( Y_1 \)
- \( Y_2 \)
- \( Y_3 \)
- \( Y_4 \)

ConvNet
- \( g(X, Y, Z, W) \)
- \( g(X, Y, Z, W) \)
- \( g(X, Y, Z, W) \)
Using Graphs instead of Vectors or Arrays.

Whereas traditional learning machines manipulate fixed-size vectors, Graph Transformer Networks manipulate graphs.
Handwriting Recognition with Graph Transformer Networks

Un-normalized hierarchical HMMs
- Trained with Perceptron loss [LeCun, Bottou, Bengio, Haffner 1998]

Answer = sequence of symbols
Latent variable = segmentation
Variables:
- $X$: input image
- $Z$: path in the interpretation graph/segmentation
- $Y$: sequence of labels on a path

Loss function: computing the energy of the desired answer:

$$E(W, Y, X)$$
Variables:
- X: input image
- Z: path in the interpretation graph/segmentation
- Y: sequence of labels on a path

Loss function: computing the contrastive term:

\[ E(W, \hat{Y}, X) \]
Example: Perceptron loss
– (no margin)
Structured prediction: when the output is structured: string, graph.....

Integrating deep learning and structured prediction is an old idea

In fact, it predates structured prediction [LeCun, Bottou, Bengio, Haffner 1998]

Globally-trained convolutional-net + graphical models for handwriting recognition

- trained discriminatively at the word level
- Loss identical to CRF and structured perceptron
- Compositional movable parts model
Pen-based handwriting recognition (for tablet computer)
  - [Bengio&LeCun 1995]
The composition of two graphs can be computed, the same way the dot product between two vectors can be computed.

General theory: semi-ring algebra on weighted finite-state transducers and acceptors.
Graph transformer network trained to read check amounts. 
Trained globally with Negative-Log-Likelihood loss. 
50% percent correct, 49% reject, 1% error (detectable later in the process. 
Fielded in 1996, used in many banks in the US and Europe. 
Processes an estimated 10% to 20% of all the checks written in the US.
# Loss Function to train Energy-Based Models

Good and bad loss functions

A tutorial on Energy-Based Learning [LeCun et al 2006]

<table>
<thead>
<tr>
<th>Loss (equation #)</th>
<th>Formula</th>
<th>Margin</th>
</tr>
</thead>
<tbody>
<tr>
<td>energy loss</td>
<td>$E(W, Y^i, X^i)$</td>
<td>none</td>
</tr>
<tr>
<td>perceptron</td>
<td>$E(W, Y^i, X^i) - \min_{Y \in \mathcal{Y}} E(W, Y, X^i)$</td>
<td>0</td>
</tr>
<tr>
<td>hinge</td>
<td>$\max\left(0, m + E(W, Y^i, X^i) - E(W, \bar{Y}^i, X^i)\right)$</td>
<td>$m$</td>
</tr>
<tr>
<td>log</td>
<td>$\log\left(1 + e^{E(W, Y^i, X^i) - E(W, \bar{Y}^i, X^i)}\right)$</td>
<td>&gt; 0</td>
</tr>
<tr>
<td>LVQ2</td>
<td>$\min\left(M, \max(0, E(W, Y^i, X^i) - E(W, \bar{Y}^i, X^i))\right)$</td>
<td>0</td>
</tr>
<tr>
<td>MCE</td>
<td>$\left(1 + e^{-\left(E(W, Y^i, X^i) - E(W, \bar{Y}^i, X^i)\right)}\right)^{-1}$</td>
<td>&gt; 0</td>
</tr>
<tr>
<td>square-square</td>
<td>$E(W, Y^i, X^i)^2 - \left(\max(0, m - E(W, \bar{Y}^i, X^i))\right)^2$</td>
<td>$m$</td>
</tr>
<tr>
<td>square-exp</td>
<td>$E(W, Y^i, X^i)^2 + \beta e^{-E(W, \bar{Y}^i, X^i)}$</td>
<td>&gt; 0</td>
</tr>
<tr>
<td>NLL/MMI</td>
<td>$E(W, Y^i, X^i) + \frac{1}{\beta} \log \int_{y \in \mathcal{Y}} e^{-\beta E(W, y, X^i)}$</td>
<td>&gt; 0</td>
</tr>
<tr>
<td>MEE</td>
<td>$1 - e^{-\beta E(W, Y^i, X^i)} / \int_{y \in \mathcal{Y}} e^{-\beta E(W, y, X^i)}$</td>
<td>&gt; 0</td>
</tr>
</tbody>
</table>
Deep Structured Predictions for Speech and Handwriting

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

Word-level global discriminative training with GMM:

- With Minimum Empirical Error loss
  - Ljolje and Rabiner (1990)
- With MCE
  - Juang et al. (1997)

Word-level global discriminative training with ConvNets:

- With the LVQ2 Loss:
  - Driancourt and Bottou's speech recognizer (1991)
- With Neg Log Likelihood (aka MMI):

CRF-like Late normalization

- un-normalized HMM
- Bottou pointed out the label bias problem (1991)
- Denker and Burges proposed a solution (1995)
- Implemented in (LeCun et al 1998)
Brute Force Approach To Multiple Object Recognition
Idea #1: Sliding Window ConvNet + Weighted FSM

- “Space Displacement Neural Net”.
- Convolutions are applied to a large image.
- Output and feature maps are extended/replicated accordingly.
Idea #1: Sliding Window ConvNet + Weighted FSM
Idea #1: Sliding Window ConvNet + Weighted FSM
Idea #1: Sliding Window ConvNet + Weighted FSM
Convolutional Networks
In
Visual Object Recognition
We knew ConvNet worked well with characters and small images.

Traffic Sign Recognition (GTSRB)
- German Traffic Sign Reco Bench
- 99.2% accuracy (IDSIA)

House Number Recognition (Google)
- Street View House Numbers
- 94.3% accuracy (NYU)
NORB Dataset (2004): 5 categories, multiple views and illuminations

- Training instances: 291,600 training samples
- Test instances: 58,320 test samples

Less than 6% error on test set with cluttered backgrounds

291,600 training samples,
58,320 test samples
mid 2000s: state of the art results on face detection

<table>
<thead>
<tr>
<th>Data Set</th>
<th>TILTED</th>
<th>PROFILE</th>
<th>MIT+CMU</th>
</tr>
</thead>
<tbody>
<tr>
<td>False positives per image</td>
<td>4.42</td>
<td>0.47</td>
<td>0.5</td>
</tr>
<tr>
<td>Our Detector</td>
<td>90%</td>
<td>67%</td>
<td>83%</td>
</tr>
<tr>
<td>Jones &amp; Viola (tilted)</td>
<td>90%</td>
<td>95%</td>
<td>x</td>
</tr>
<tr>
<td>Jones &amp; Viola (profile)</td>
<td>x</td>
<td>70%</td>
<td>x</td>
</tr>
</tbody>
</table>

[Vaillant et al. IEE 1994] [Osadchy et al. 2004] [Osadchy et al, JMLR 2007]
Simultaneous face detection and pose estimation

[Vaillant et al. IEE 1994] [Osadchy et al. 2004] [Osadchy et al, JMLR 2007]
Simultaneous face detection and pose estimation
In the mid 2000s, ConvNets were getting decent results on object classification.

**Dataset: “Caltech101”:**
- 101 categories
- 30 training samples per category

But the results were slightly worse than more “traditional” computer vision methods, because
- 1. the datasets were too small
- 2. the computers were too slow
Late 2000s: we could get decent results on object recognition

But we couldn't beat the state of the art because the datasets were too small.

Caltech101: 101 categories, 30 samples per category.

But we learned that rectification and max pooling are useful! [Jarrett et al. ICCV 2009]

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

<table>
<thead>
<tr>
<th>R/N/P</th>
<th>(R_{ab} - N - P_A)</th>
<th>(R_{ab} - 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%((\pm 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%((\pm 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: \([64.F_{CSG}^{9\times9} - R/N/P_{5\times5}] - [256.F_{CSG}^{9\times9} - R/N/P_{4\times4}] - \text{log_reg}\)

<table>
<thead>
<tr>
<th>R/N/P</th>
<th>(R_{ab} - N - P_A)</th>
<th>(R_{ab} - 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%((\pm 1.5))</td>
<td>37.6%((\pm 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></td>
</tr>
</tbody>
</table>

← like HMAX model

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

| \(U\) | 64.0% |

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

| \(UU\) | 52.8% |
Scene Parsing – Semantic Labeling

State of the art accuracy: ConvNet: 74.5%; [Munoz et al.]: 66.2%

[Farabet et al. ICML 2012, PAMI 2013]
Then, two things happened...

The ImageNet dataset [Fei-Fei et al. 2012]
- 1.5 million training samples
- 1000 categories

Fast Graphical Processing Units (GPU)
- Capable of 1 trillion operations/second
The ImageNet dataset
- 1.5 million training samples
- 1000 fine-grained categories (breeds of dogs,...)
Object Recognition [Krizhevsky, Sutskever, Hinton 2012]

Won the 2012 ImageNet LSVRC. 60 Million parameters, 832M MAC ops

<table>
<thead>
<tr>
<th>Layer Type</th>
<th>Parameters</th>
<th>MAC Ops</th>
</tr>
</thead>
<tbody>
<tr>
<td>FULL CONNECT</td>
<td>4M</td>
<td>4Mflop</td>
</tr>
<tr>
<td>FULL 4096/ReLU</td>
<td>16M</td>
<td>16M</td>
</tr>
<tr>
<td>FULL 4096/ReLU</td>
<td>37M</td>
<td>37M</td>
</tr>
<tr>
<td>MAX POOLING</td>
<td>442K</td>
<td>74M</td>
</tr>
<tr>
<td>CONV 3x3/ReLU 256fm</td>
<td>1.3M</td>
<td>224M</td>
</tr>
<tr>
<td>CONV 3x3/ReLU 384fm</td>
<td>884K</td>
<td>149M</td>
</tr>
<tr>
<td>MAX POOLING 2x2sub</td>
<td>307K</td>
<td>223M</td>
</tr>
<tr>
<td>LOCAL CONTRAST NORM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CONV 11x11/ReLU 256fm</td>
<td>35K</td>
<td>105M</td>
</tr>
<tr>
<td>MAX POOL 2x2sub</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LOCAL CONTRAST NORM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CONV 11x11/ReLU 96fm</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Object Recognition [Krizhevsky, Sutskever, Hinton 2012]

- **Method:** large convolutional net
  - 650K neurons, 832M synapses, 60M parameters
  - Trained with backprop on GPU
  - Trained “with all the tricks Yann came up with in the last 20 years, plus dropout” (Hinton, NIPS 2012)
  - Rectification, contrast normalization,…

- **Error rate:** 15% *(whenever correct class isn't in top 5)*
- **Previous state of the art:** 25% error

**A REVOLUTION IN COMPUTER VISION**

- Acquired by Google in Jan 2013
- Deployed in Google+ Photo Tagging in May 2013
Searched my personal collection for “bird”
NYU ConvNet Trained on ImageNet: OverFeat

- [Sermanet et al. arXiv:1312.6229]
- Trained on GPU using Torch7
- Uses a number of new tricks
- Classification 1000 categories:
  - 13.8% error (top 5) with an ensemble of 7 networks (Krizhevsky: 15%)
  - 15.4% error (top 5) with a single network (Krizhevsky: 18.2%)
- Classification+Localization
  - 30% error (Krizhevsky: 34%)
- Detection (200 categories)
  - 19% correct

Downloadable code (running, no training)
- Search for “overfeat NYU” on Google
- http://cilvr.nyu.edu → software
Layer 1: 3x96 kernels, RGB->96 feature maps, 7x7 Kernels, stride 2

Layer 2: 96x256 kernels, 7x7
Layer 1: 3x96 kernels, RGB->96 feature maps, 11x11 Kernels, stride 4
### ImageNet: Classification

- **Give the name of the dominant object in the image**
- **Top-5 error rates: if correct class is not in top 5, count as error**
- **Red: ConvNet, blue: no ConvNet**

#### 2012 Teams

<table>
<thead>
<tr>
<th>Team</th>
<th>%error</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supervision (Toronto)</td>
<td>15.3</td>
</tr>
<tr>
<td>ISI (Tokyo)</td>
<td>26.1</td>
</tr>
<tr>
<td>VGG (Oxford)</td>
<td>26.9</td>
</tr>
<tr>
<td>XRCE/INRIA</td>
<td>27.0</td>
</tr>
<tr>
<td>UvA (Amsterdam)</td>
<td>29.6</td>
</tr>
<tr>
<td>INRIA/LEAR</td>
<td>33.4</td>
</tr>
</tbody>
</table>

#### 2013 Teams

<table>
<thead>
<tr>
<th>Team</th>
<th>%error</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clarifai (NYU spinoff)</td>
<td>11.7</td>
</tr>
<tr>
<td>NUS (Singapore)</td>
<td>12.9</td>
</tr>
<tr>
<td>Zeiler-Fergus (NYU)</td>
<td>13.5</td>
</tr>
<tr>
<td>A. Howard</td>
<td>13.5</td>
</tr>
<tr>
<td>OverFeat (NYU)</td>
<td>14.1</td>
</tr>
<tr>
<td>UvA (Amsterdam)</td>
<td>14.2</td>
</tr>
<tr>
<td>Adobe</td>
<td>15.2</td>
</tr>
<tr>
<td>VGG (Oxford)</td>
<td>15.2</td>
</tr>
<tr>
<td>VGG (Oxford)</td>
<td>23.0</td>
</tr>
</tbody>
</table>

#### 2014 Teams

<table>
<thead>
<tr>
<th>Team</th>
<th>%error</th>
</tr>
</thead>
<tbody>
<tr>
<td>GoogLeNet</td>
<td>6.6</td>
</tr>
<tr>
<td>VGG (Oxford)</td>
<td>7.3</td>
</tr>
<tr>
<td>MSRA</td>
<td>8.0</td>
</tr>
<tr>
<td>A. Howard</td>
<td>8.1</td>
</tr>
<tr>
<td>DeeperVision</td>
<td>9.5</td>
</tr>
<tr>
<td>NUS-BST</td>
<td>9.7</td>
</tr>
<tr>
<td>TTIC-ECP</td>
<td>10.2</td>
</tr>
<tr>
<td>XYZ</td>
<td>11.2</td>
</tr>
<tr>
<td>UvA</td>
<td>12.1</td>
</tr>
</tbody>
</table>
Classification + Localization: Results

Top 5:
white wolf
white wolf
timber wolf
timber wolf
Arctic fox

Groundtruth:
white wolf
white wolf (2)
white wolf (3)
white wolf (4)
white wolf (5)
It's best to propose several categories for the same window
One of them might be right

2014 results: 25.3% (VGG Oxford), 26.4% (GoogLeNet)
Apply convnet with a sliding window over the image at multiple scales

Important note: it's very cheap to slide a convnet over an image

Just compute the convolutions over the whole image and replicate the fully-connected layers
Applying a ConvNet on Sliding Windows is Very Cheap!

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.

Simply apply the convolutions to the entire image and spatially replicate the fully-connected layers.
Apply convnet with a sliding window over the image at multiple scales

For each window, predict a class and bounding box parameters

Even if the object is not completely contained in the viewing window, the convnet can predict where it thinks the object is.
Classification + Localization:
sliding window + bounding box regression + bbox voting

1. Apply convnet with a sliding window over the image at multiple scales
2. For each window, predict a class and bounding box parameters
3. Compute an “average” bounding box, weighted by scores
Localization: Sliding Window + bbox vote + multiscale
Detection / Localization

OverFeat • Pierre Sermanet • New York University
Detection / Localization

OverFeat • Pierre Sermanet • New York University
Detection: Examples

- 200 broad categories
- There is a penalty for false positives
- Some examples are easy; some are impossible/ambiguous
- Some classes are well detected

Burritos?
Detection: Examples

- Groundtruth is sometimes ambiguous or incomplete
- Large overlap between objects stops non-max suppression from working

**Top predictions:**
- tv or monitor (confidence 11.5)
- person (confidence 4.5)
- miniskirt (confidence 3.1)

**Groundtruth:**
- tv or monitor
- tv or monitor (2)
- tv or monitor (3)
- person
- remote control
- remote control (2)
ImageNet: Detection (200 categories)

- Give a bounding box and a category for all objects in the image
- MAP = mean average precision
- Red: ConvNet, blue: no ConvNet

<table>
<thead>
<tr>
<th>2013 Teams</th>
<th>mAP</th>
</tr>
</thead>
<tbody>
<tr>
<td>UvA Amsterdam</td>
<td>22.6</td>
</tr>
<tr>
<td>NEC Labs-America</td>
<td>20.9</td>
</tr>
<tr>
<td>OverFeat NYU</td>
<td>19.4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Off cycle results</th>
<th>mAP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Berkeley RCNN</td>
<td>34.5</td>
</tr>
<tr>
<td>OverFeat NYU</td>
<td>24.3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>2014 Teams</th>
<th>mAP</th>
</tr>
</thead>
<tbody>
<tr>
<td>GoogLeNet</td>
<td>43.9</td>
</tr>
<tr>
<td>CUHK-DeepID2</td>
<td>40.7</td>
</tr>
<tr>
<td>DeepInsight</td>
<td>40.4</td>
</tr>
<tr>
<td>NUS</td>
<td>37.2</td>
</tr>
<tr>
<td>UvA Amsterdam</td>
<td>35.4</td>
</tr>
<tr>
<td>MSRA</td>
<td>35.1</td>
</tr>
<tr>
<td>Berkeley RCNN</td>
<td>34.5</td>
</tr>
</tbody>
</table>
Results: pre-trained on ImageNet1K, fine-tuned on ImageNet Detection
Detection Examples

Y LeCun
Detection Examples

/she/home/snwiz/data/imagenet12/original/det/ILSVRC2013_DET_test/ILSVRC2012_test_00090628.JPEG

dog conf 3.419652
sheep conf 1.616741
Groundtruth is sometimes ambiguous or incomplete.
Non-max suppression makes us miss many objects

- Person behind instrument
- A bit of contextual post-processing would fix many errors
Snake → Corkscrew
Detection: Bad Groundtruth

One of the labelers likes ticks.....
ConvNets
As Generic
Feature Extractors
- Kaggle competition: Dog vs Cats
- Won by Pierre Sermanet (NYU):
- ImageNet network (OverFeat) last layers retrained on cats and dogs

http://www.csc.kth.se/cvap/cvg/DL/ots/

<table>
<thead>
<tr>
<th>best non-CNN results</th>
<th>VOC07c</th>
<th>VOC12c</th>
<th>VOC12a</th>
<th>MIT67</th>
<th>SUN397</th>
<th>VOC07d</th>
<th>VOC10d</th>
<th>VOC11s</th>
<th>200Birds</th>
<th>102Flowers</th>
<th>H3Datt</th>
<th>UIUCatt</th>
<th>LFW</th>
<th>YTF</th>
<th>Paris6k</th>
<th>Oxford5k</th>
<th>Sculp6k</th>
<th>Holidays</th>
<th>UKB</th>
</tr>
</thead>
</table>

**VOC07c:** Pascal VOC 2007 (Object Image Classification)
**VOC12c:** Pascal VOC 2012 (Object Image Classification)
**VOC12a:** Pascal VOC 2012 (Action Image Classification)
**MIT67:** MIT 67 Indoor Scenes (Scene Image Classification)
**VOC07d:** PASCAL VOC 2007 (Object Detection)
**VOC10d:** PASCAL VOC 2010 (Object Detection)
**VOC12d:** PASCAL VOC 2012 (Object Detection)
**VOC11s:** PASCAL VOC 2011 (Object Category Segmentation)

**200Birds:** UCSD-Caltech 2011-200 Birds dataset (Fine-grained Recognition)
**102Flowers:** Oxford 102 Flowers (Fine-grained Recognition)
**H3Datt:** H3D poselets Human 9 Attributes (Attribute Detection)
**UIUCatt:** UIUC object attributes (Attribute Detection)
**LFW:** Labelled Faces in the Wild (Metric Learning)
**Oxford5k:** Oxford 5k Buildings Dataset (Instance Retrieval)
**Paris6k:** Paris 6k Buildings Dataset (Instance Retrieval)
**Sculp6k:** Oxford Sculptures Dataset (Instance Retrieval)
**Holidays:** INRIA Holidays Scenes Dataset (Instance Retrieval)
**UHK:** Uni. of Kentucky Retrieval Benchmark Dataset (Instance Retrieval)
### OverFeat Features + Classifier on various datasets

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Performance</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>[Sermanet et al 2014]: OverFeat (fine-tuned features for each task)</strong> (tasks are ordered by increasing difficulty)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><em>image classification</em></td>
<td>ImageNet LSVRC 2013</td>
<td>competitive</td>
</tr>
<tr>
<td></td>
<td>Dogs vs Cats Kaggle challenge 2014</td>
<td>state of the art</td>
</tr>
<tr>
<td><em>object localization</em></td>
<td>ImageNet LSVRC 2013</td>
<td>state of the art</td>
</tr>
<tr>
<td><em>object detection</em></td>
<td>ImageNet LSVRC 2013</td>
<td>state of the art</td>
</tr>
</tbody>
</table>

| **[Razavian et al, 2014]: public OverFeat library (no retraining) + SVM (simplest approach possible on purpose, no attempt at more complex classifiers)** (tasks are ordered by “distance” from classification task on which OverFeat was trained) |                              |             |
| *image classification*           | Pascal VOC 2007               | competitive | 73.9 % mAP     |
|                                  | MIT-67                        | competitive | 58.4 % mAP     |
| *scene recognition*              | Caltech-UCSD Birds 200-2011   | competitive | 53.3 % mAP     |
| *fine grained recognition*       | Oxford 102 Flowers            | competitive | 74.70 % mAP    |
|                                  | UIUC 64 object attributes     | state of the art | 89.0 % mAUC    |
|                                  | H3D Human Attributes          | state of the art | 70.78 % mAP    |
| *attribute detection*            | Oxford 5k buildings           | ?            | 0.52           |
|                                  | Paris 6k buildings             | ?            | 0.676          |
|                                  | Sculp6k                        | ?            | 0.269          |
|                                  | Holidays                       | competitive  | 0.646          |
|                                  | UKBench                        | relatively poor | 3.05        |

Image Similarity Matching With Siamese Networks Embedding, DrLIM
Dimensionality Reduction by Learning an Invariant Mapping

- **Step 1:** Construct neighborhood graph.
- **Step 2:** Choose a parameterized family of functions.
- **Step 3:** Optimize the parameters such that:
  - Outputs for similar samples are pulled closer.
  - Outputs for dissimilar samples are pushed away.

 joint work with Sumit Chopra: Hadsell et al. CVPR 06; Chopra et al., CVPR 05
Siamese Architecture

\[ E_w \quad \parallel G_\mathbf{w}(X_1) - G_\mathbf{w}(X_2) \parallel_2 \]

Siamese Architecture [Bromley, Sackinger, Shah, LeCun 1994]
Siamese Architecture and loss function

Loss function:
- Outputs corresponding to input samples that are neighbors in the neighborhood graph should be nearby.
- Outputs for input samples that are not neighbors should be far away from each other.

\[ D_w \| G_w(x_1) - G_w(x_2) \| \]

```
\begin{align*}
\text{Make this small} \\
G_w(x_1) \quad G_w(x_2) \\
\quad x_1 \quad x_2
\end{align*}
```

```
\begin{align*}
\text{Make this large} \\
G_w(x_1) \quad G_w(x_2) \\
\quad x_1 \quad x_2
\end{align*}
```

Similar images (neighbors in the neighborhood graph)

Dissimilar images (non-neighbors in the neighborhood graph)
Face Recognition: DeepFace (Facebook AI Research)

[Taigman et al. CVPR 2014]
- Alignment
- Convnet
Accurate Depth Estimation from Stereo
KITTI Dataset (RGB registered with LIDAR)

- Stereo Cameras + Velodyne LIDAR: aligned data.
- Collected from a car
- Lots of results with many methods from many teams around the world
- Stereo images sparsely labeled with depth values
- supervised learning
- supervised learning of a patch matcher (as a binary classifier)

<table>
<thead>
<tr>
<th>Left patch</th>
<th>Right patch</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="Image" /></td>
<td><img src="image2.png" alt="Image" /></td>
<td>Good match</td>
</tr>
<tr>
<td><img src="image3.png" alt="Image" /></td>
<td><img src="image4.png" alt="Image" /></td>
<td>Bad match</td>
</tr>
</tbody>
</table>
ConvNet for Stereo Matching

Using a ConvNet to learn a similarity measure between image patches

Left input image

Right input image

Output disparity map

90 m  20 m  1.7 m

Figure 2: Network architecture
ConvNet for Stereo Matching

Image

Winner-take-all output
(14.55% error >3 pix)

Result after cleanup
(2.61% error >3 pix)
Depth Estimation from Stereo Pairs

Using a ConvNet to learn a similarity measure between image patches/

Record holder on KITTI dataset (Sept 2014):

<table>
<thead>
<tr>
<th>Rank</th>
<th>Method</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Our method</td>
<td>2.61 %</td>
</tr>
<tr>
<td>2</td>
<td>SPS-StFl</td>
<td>2.83 %</td>
</tr>
<tr>
<td>3</td>
<td>VC-SF</td>
<td>3.05 %</td>
</tr>
<tr>
<td>4</td>
<td>PCBP-SS</td>
<td>3.40 %</td>
</tr>
<tr>
<td>5</td>
<td>DDS-SS</td>
<td>3.83 %</td>
</tr>
</tbody>
</table>

Original image

SADD

Census

ConvNet

Figure 2: Network architecture
KITTI Stereo Leaderboard (Sept 2014)

Metric: Percentage of pixels with more than 3 pixel error on disparity

Our ConvNet method is first. Doesn't use optical flow nor multiple images.

#2 uses optical flow and 2 frames

#3 uses multiple frames

#4 is at 3.39% vs our 2.61%

<table>
<thead>
<tr>
<th>Rank</th>
<th>Method</th>
<th>Setting</th>
<th>Code</th>
<th>Out-Noc</th>
<th>Out-All</th>
<th>Avg-Noc</th>
<th>Avg-All</th>
<th>Density</th>
<th>Runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>MC-CNN</td>
<td></td>
<td></td>
<td>2.61 %</td>
<td>3.84 %</td>
<td>0.8 px</td>
<td>1.0 px</td>
<td>100.00%</td>
<td>100 s</td>
</tr>
<tr>
<td>2</td>
<td>SPS-StFl</td>
<td></td>
<td></td>
<td>2.83 %</td>
<td>3.64 %</td>
<td>0.8 px</td>
<td>0.9 px</td>
<td>100.00%</td>
<td>35 s</td>
</tr>
<tr>
<td>3</td>
<td>VC-SF</td>
<td></td>
<td></td>
<td>3.05 %</td>
<td>3.31 %</td>
<td>0.8 px</td>
<td>0.8 px</td>
<td>100.00%</td>
<td>300 s</td>
</tr>
<tr>
<td>4</td>
<td>SPS-St</td>
<td></td>
<td></td>
<td>3.39 %</td>
<td>4.41 %</td>
<td>0.9 px</td>
<td>1.0 px</td>
<td>100.00%</td>
<td>5 s</td>
</tr>
<tr>
<td>5</td>
<td>PCBP-SS</td>
<td></td>
<td></td>
<td>3.40 %</td>
<td>4.72 %</td>
<td>0.8 px</td>
<td>1.0 px</td>
<td>100.00%</td>
<td>5 min</td>
</tr>
</tbody>
</table>


Depth Estimation from Stereo Pairs: Results

[Zbontar & LeCun Arxiv '14]

Presentation at ECCV workshop 2014/9/6
Body Pose Estimation
Pose Estimation and Attribute Recovery with ConvNets

Pose-Aligned Network for Deep Attribute Modeling
[Zhang et al. CVPR 2014] (Facebook AI Research)

(a) Highest scoring results for people wearing glasses.

(b) Highest scoring results for people wearing a hat.

Real-time hand pose recovery
[Tompson et al. Trans. on Graphics 14]

Body pose estimation [Tompson et al. ICLR, 2014]
Other Tasks for Which Deep Convolutional Nets are the Best

- Handwriting recognition MNIST (many), Arabic HWX (IDSIA)
- OCR in the Wild [2011]: StreetView House Numbers (NYU and others)
- Traffic sign recognition [2011] GTSRB competition (IDSIA, NYU)
- Asian handwriting recognition recognition [2013] ICDAR competition (IDSIA)
- Pedestrian Detection [2013]: INRIA datasets and others (NYU)
- Volumetric brain image segmentation [2009] connectomics (IDSIA, MIT)
- Object Recognition [2012] ImageNet competition (Toronto)
- Scene Parsing [2012] Stanford bgd, SiftFlow, Barcelona datasets (NYU)
- Scene parsing from depth images [2013] NYU RGB-D dataset (NYU)
- Speech Recognition [2012] Acoustic modeling (IBM and Google)
- Breast cancer cell mitosis detection [2011] MITOS (IDSIA)

The list of perceptual tasks for which ConvNets hold the record is growing.
Most of these tasks (but not all) use purely supervised convnets.
Deep Learning and Convolutional Networks in Speech, Audio, and Signals
Acoustic Modeling in Speech Recognition (Google)

A typical speech recognition architecture with DL-based acoustic modeling

- Features: log energy of a filter bank (e.g. 40 filters)
- Neural net acoustic modeling (convolutional or not)
- Input window: typically 10 to 40 acoustic frames
- Fully-connected neural net: 10 layers, 2000-4000 hidden units/layer
- But convolutional nets do better....
- Predicts phone state, typically 2000 to 8000 categories

Mohamed et al. “DBNs for phone recognition” NIPS Workshop 2009
Zeiler et al. “On rectified linear units for speech recognition” ICASSP 2013
Acoustic Model: ConvNet with 7 layers. 54.4 million parameters.
Classifies acoustic signal into 3000 context-dependent subphones categories
ReLU units + dropout for last layers
Trained on GPU. 4 days of training
Subphone-level classification error (sept 2013):
- Cantonese: phone: 20.4% error; subphone: 33.6% error (IBM DNN: 37.8%)

Subphone-level classification error (march 2013)
- Cantonese: subphone: 36.91%
- Vietnamese: subphone 48.54%
- Full system performance (token error rate on conversational speech):
  - 76.2% (52.9% substitution, 13.0% deletion, 10.2% insertion)
Training samples.

- 40 MEL-frequency Cepstral Coefficients
- Window: 40 frames, 10ms each
Convolution Kernels at Layer 1:
- 64 kernels of size 9x9
Convolutional Networks
In
Image Segmentation,
& Scene Labeling
**ConvNets for Image Segmentation**

- **Biological Image Segmentation**
  - [Ning et al. IEEE-TIP 2005]

- **Pixel labeling with large context using a convnet**

- **ConvNet takes a window of pixels and produces a label for the central pixel**

- **Cleanup using a kind of conditional random field (CRF)**
  - Similar to a field of expert
Semantic Labeling / Scene Parsing:
Labeling every pixel with the object it belongs to

- Would help identify obstacles, targets, landing sites, dangerous areas
- Would help line up depth map with edge maps
Scene Parsing/Labeling: ConvNet Architecture

- Each output sees a large input context:
  - **46x46** window at full rez; **92x92** at ½ rez; **184x184** at ¼ rez
  - [7x7conv]–>[2x2pool]–>[7x7conv]–>[2x2pool]–>[7x7conv]–>
  - Trained supervised on fully-labeled images
Method 1: majority over super-pixel regions

- **Input image**
- **Super-pixel boundaries**
- **Features from Convolutional net (d=768 per pixel)**
- **Convolutional classifier**
- **Categories aligned With region boundaries**
- **Multi-scale ConvNet**
- **“soft” categories scores**
- **Majority Vote Over Superpixels**

[Farabet et al. IEEE T. PAMI 2013]
### Scene Parsing/Labeling: Performance

**Stanford Background Dataset [Gould 1009]: 8 categories**

<table>
<thead>
<tr>
<th>Method</th>
<th>Pixel Acc.</th>
<th>Class Acc.</th>
<th>CT (sec.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gould <em>et al.</em> 2009 [14]</td>
<td>76.4%</td>
<td>-</td>
<td>10 to 600s</td>
</tr>
<tr>
<td>Munoz <em>et al.</em> 2010 [32]</td>
<td>76.9%</td>
<td>66.2%</td>
<td>12s</td>
</tr>
<tr>
<td>Tighe <em>et al.</em> 2010 [46]</td>
<td>77.5%</td>
<td>-</td>
<td>10 to 300s</td>
</tr>
<tr>
<td>Socher <em>et al.</em> 2011 [45]</td>
<td>78.1%</td>
<td>-</td>
<td>?</td>
</tr>
<tr>
<td>Kumar <em>et al.</em> 2010 [22]</td>
<td>79.4%</td>
<td>-</td>
<td>&lt; 600s</td>
</tr>
<tr>
<td>Lempitzky <em>et al.</em> 2011 [28]</td>
<td>81.9%</td>
<td>72.4%</td>
<td>&gt; 60s</td>
</tr>
<tr>
<td>singlescale convnet</td>
<td>66.0 %</td>
<td>56.5 %</td>
<td>0.35s</td>
</tr>
<tr>
<td>multiscale convnet</td>
<td>78.8 %</td>
<td>72.4%</td>
<td>0.6s</td>
</tr>
<tr>
<td><strong>multiscale net + superpixels</strong></td>
<td>80.4%</td>
<td>74.56%</td>
<td>0.7s</td>
</tr>
<tr>
<td>multiscale net + gPb + cover</td>
<td>80.4%</td>
<td>75.24%</td>
<td>61s</td>
</tr>
<tr>
<td>multiscale net + CRF on gPb</td>
<td>81.4%</td>
<td>76.0%</td>
<td>60.5s</td>
</tr>
</tbody>
</table>

[Farabet et al. IEEE T. PAMI 2013]
## Scene Parsing/Labeling: Performance

<table>
<thead>
<tr>
<th></th>
<th>Pixel Acc.</th>
<th>Class Acc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Liu et al. 2009 [31]</td>
<td>74.75%</td>
<td>-</td>
</tr>
<tr>
<td>Tighe et al. 2010 [44]</td>
<td>76.9%</td>
<td>29.4%</td>
</tr>
<tr>
<td>raw multiscale net(^1)</td>
<td>67.9%</td>
<td>45.9%</td>
</tr>
<tr>
<td>multiscale net + superpixels(^1)</td>
<td>71.9%</td>
<td>50.8%</td>
</tr>
<tr>
<td>multiscale net + cover(^1)</td>
<td>72.3%</td>
<td>50.8%</td>
</tr>
<tr>
<td>multiscale net + cover(^2)</td>
<td>78.5%</td>
<td>29.6%</td>
</tr>
</tbody>
</table>

- **SIFT Flow Dataset**
- [Liu 2009]: 33 categories

<table>
<thead>
<tr>
<th></th>
<th>Pixel Acc.</th>
<th>Class Acc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tighe et al. 2010 [44]</td>
<td>66.9%</td>
<td>7.6%</td>
</tr>
<tr>
<td>raw multiscale net(^1)</td>
<td>37.8%</td>
<td>12.1%</td>
</tr>
<tr>
<td>multiscale net + superpixels(^1)</td>
<td>44.1%</td>
<td>12.4%</td>
</tr>
<tr>
<td>multiscale net + cover(^1)</td>
<td>46.4%</td>
<td>12.5%</td>
</tr>
<tr>
<td>multiscale net + cover(^2)</td>
<td>67.8%</td>
<td>9.5%</td>
</tr>
</tbody>
</table>

- **Barcelona dataset**
- [Tighe 2010]: 170 categories.

[Farabeta et al. IEEE T. PAMI 2012]
Scene Parsing/Labeling: SIFT Flow dataset (33 categories)

Samples from the SIFT-Flow dataset (Liu)

[Farabet et al. ICML 2012, PAMI 2013]
Scene Parsing/Labeling: SIFT Flow dataset (33 categories)

[Farabet et al. ICML 2012, PAMI 2013]
Scene Parsing/Labeling

[Farabet et al. ICML 2012, PAMI 2013]
Scene Parsing/Labeling

[Farabet et al. ICML 2012, PAMI 2013]
Scene Parsing/Labeling

[Farabet et al. ICML 2012, PAMI 2013]
Scene Parsing/Labeling

[Farabet et al. ICML 2012, PAMI 2013]
Scene Parsing/Labeling

- No post-processing
- Frame-by-frame
- ConvNet runs at 50ms/frame on Virtex-6 FPGA hardware
  - But communicating the features over ethernet limits system performance
Temporal Consistency

- **Spatio-Temporal Super-Pixel segmentation**
  - [Couprie et al ICIP 2013]
  - [Couprie et al JMLR under review]
  - Majority vote over super-pixels

Independent segmentations $S'_1$, $S'_2$, and $S'_3$

Temporally consistent segmentations $S_1(=S'_1)$, $S_2$, and $S_3$
Causal method for temporal consistency

[Couprie, Farabet, Najman, LeCun ICLR 2013, ICIP 2013]
Captured with a Kinect on a steadycam
Results

Depth helps a bit
- Helps a lot for floor and props
- Helps surprisingly little for structures, and hurts for furniture

<table>
<thead>
<tr>
<th></th>
<th>Ground</th>
<th>Furniture</th>
<th>Props</th>
<th>Structure</th>
<th>Class Acc.</th>
<th>Pixel Acc.</th>
<th>Comput. time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Silberman et al. (2012)</td>
<td>68</td>
<td>70</td>
<td>42</td>
<td>59</td>
<td>59.6</td>
<td>58.6</td>
<td>&gt;3</td>
</tr>
<tr>
<td>Cadena and Kosecka (2013)</td>
<td>87.9</td>
<td>64.1</td>
<td>31.0</td>
<td>77.8</td>
<td>65.2</td>
<td>66.9</td>
<td>1.7</td>
</tr>
<tr>
<td>Multiscale convnet</td>
<td>68.1</td>
<td>51.1</td>
<td>29.9</td>
<td>87.8</td>
<td>59.2</td>
<td>63.0</td>
<td>0.7</td>
</tr>
<tr>
<td>Multiscale+depth convnet</td>
<td>87.3</td>
<td>45.3</td>
<td>35.5</td>
<td>86.1</td>
<td>63.5</td>
<td>64.5</td>
<td>0.7</td>
</tr>
</tbody>
</table>

Scene Parsing/Labeling on RGB+Depth Images

Ground truths

Our results

[Couprie, Farabet, Najman, LeCun ICLR 2013, ICIP 2013]
Scene Parsing/Labeling on RGB+Depth Images

[Ground truths]

[Our results]

[Couprie, Farabet, Najman, LeCun ICLR 2013, ICIP 2013]
Labeling Videos

Temporal consistency

(a) Output of the Multiscale convnet trained using depth information - frame by frame

(b) Results smoothed temporally using Couprie et al. (2013a)

[Couprie, Farabet, Najman, LeCun ICLR 2013]
[Couprie, Farabet, Najman, LeCun ICIP 2013]
[Couprie, Farabet, Najman, LeCun submitted to JMLR]
Semantic Segmentation on RGB+D Images and Videos

[Couprie, Farabet, Najman, LeCun ICLR 2013, ICIP 2013]
Self-Learning ConvNet for Scene Labeling
Vision-Based Navigation for Off-Road Robots
Getting a robot to drive autonomously in unknown terrain solely from vision (camera input).

Our team (NYU/Net-Scale Technologies Inc.) was one of 8 participants funded by DARPA.

All teams received identical robots and can only modify the software (not the hardware).

The robot is given the GPS coordinates of a goal, and must drive to the goal as fast as possible. The terrain is unknown in advance. The robot is run 3 times through the same course.

Long-Range Obstacle Detection with on-line, self-trained ConvNet

Uses temporal consistency!
Obstacle Detection at Short Range: Stereovision

Obstacles overlaid with camera image

Camera image  Detected obstacles (red)
Stereo is only good up to about 10 meters.
But not seeing past 10 meters is like driving in a fog or a snowstorm!
Long Range Vision with a Convolutional Net

Pre-processing (125 ms)

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

100 features per 3x12x25 input window

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

YUV input

100@25x121

CONVOLUTIONS (6x5)

20@30x125

MAX SUBSAMPLING (1x4)

20@30x484

CONVOLUTIONS (7x6)

3@36x484
Scene Labeling with ConvNet + online learning

**Image Labeling for Off-Road Robots [Hadsell JFR 2008]**
- ConvNet labels pixels as one of 3 categories
- Traversable/flat (green), non traversible (red), foot of obstacle (purple)
- Labels obtained from stereo vision and SLAM
Long Range Vision Results

Input image

Stereo Labels

Classifier Output

Input image

Stereo Labels

Classifier Output
Long Range Vision Results

- Input image
- Stereo Labels
- Classifier Output

- Input image
- Stereo Labels
- Classifier Output
Form Reading: AT&T 1994
Check reading: AT&T/NCR 1996 (read 10-20% of all US checks in 2000)
Handwriting recognition: Microsoft early 2000
Face and person detection: NEC 2005, France Telecom late 2000s.
Gender and age recognition: NEC 2010 (vending machines)
OCR in natural images: Google 2013 (StreetView house numbers)
Photo tagging: Google 2013
Image Search by Similarity: Baidu 2013
Since early 2014, the number of deployed applications of ConvNets has exploded
Many applications at Facebook, Google, Baidu, Microsoft, IBM, NEC, Yahoo.....
   Speech recognition, face recognition, image search, content filtering/ranking,....
Tens of thousands of servers run ConvNets continuously every day.
Software Platform for Deep Learning: Torch7

Torch7
- based on the LuaJIT language
- Simple and lightweight dynamic language (widely used for games)
- Multidimensional array library with CUDA and OpenMP backends
- FAST: Has a native just-in-time compiler
- Has an unbelievably nice foreign function interface to call C/C++ functions from Lua

Torch7 is an extension of Lua with
- Multidimensional array engine
- A machine learning library that implements multilayer nets, convolutional nets, unsupervised pre-training, etc
- Various libraries for data/image manipulation and computer vision
- Used at Facebook Ai Research, Google (Deep Mind, Brain), Intel, and many academic groups and startups

Single-line installation on Ubuntu and Mac OSX:
- http://torch.ch

Torch7 Cheat sheet (with links to libraries and tutorials):
  - https://github.com/torch/torch7/wiki/Cheatsheet
Example: building a Neural Net in Torch7

Net for SVHN digit recognition
10 categories
Input is 32x32 RGB (3 channels)
1500 hidden units

Creating a 2-layer net
Make a cascade module
Reshape input to vector
Add Linear module
Add tanh module
Add Linear Module
Add log softmax layer

Create loss function module

Noutputs = 10;
nfeats = 3; Width = 32; height = 32
ninputs = nfeats*width*height
nhiddens = 1500

-- Simple 2-layer neural network
model = nn.Sequential()
model:add(nn.Reshape(ninputs))
model:add(nn.Linear(ninputs,nhiddens))
model:add(nn.Tanh())
model:add(nn.Linear(nhiddens,noutputs))
model:add(nn.LogSoftMax())
criterion = nn.ClassNLLCriterion()

See Torch7 example at http://bit.ly/16tyLAx
NeuFlow architecture (NYU + Purdue)

- Collaboration NYU-Purdue: Eugenio Culurciello's e-Lab.
- Running on Picocomputing 8x10cm high-performance FPGA board
  - Virtex 6 LX240T: 680 MAC units, 20 neuflow tiles
- Full scene labeling at 20 frames/sec (50ms/frame) at 320x240

board with Virtex-6
NewFlow: Architecture

A Runtime Reconfigurable Dataflow Architecture

grid of passive processing tiles (PTs)
[x20 on a Virtex6 LX240T]

Multi-port memory controller (DMA)
[x12 on a V6 LX240T]

RISC CPU, to reconfigure tiles and data paths, at runtime

global network-on-chip to allow fast reconfiguration
NewFlow: Processing Tile Architecture

Term-by-term streaming operators (MUL, DIV, ADD, SUB, MAX)

- Configurable bank of FIFOs, for stream buffering, up to 10 kB per PT
- Configurable router, to stream data in and out of the tile, to neighbors or DMA ports
- Configurable piece-wise linear or quadratic mapper

- Full 1/2D parallel convolver with 100 MAC units

[Virtex6 LX240T]
NewFlow ASIC: 2.5x5 mm, 45nm, 0.6Watts, >300GOPS

Collaboration Purdue-NYU: Eugenio Culurciello's e-Lab

Suitable for vision-enabled embedded and mobile devices

(but the fabrication was botched...)

[Pham, Jelaca, Farabet, Martini, LeCun, Culurciello 2012]
### NewFlow: Performance

<table>
<thead>
<tr>
<th></th>
<th>Intel I7 4 cores</th>
<th>neuFlow Virtex4</th>
<th>neuFlow Virtex 6</th>
<th>nVidia GT335m</th>
<th>NeuFlow ASIC 45nm</th>
<th>nVidia GTX480*</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Peak GOP/sec</strong></td>
<td>40</td>
<td>40</td>
<td>160</td>
<td>182</td>
<td>320</td>
<td>1350</td>
</tr>
<tr>
<td><strong>Actual GOP/sec</strong></td>
<td>12</td>
<td>37</td>
<td>147</td>
<td>54</td>
<td>300</td>
<td>294</td>
</tr>
<tr>
<td><strong>FPS</strong></td>
<td>14</td>
<td>46</td>
<td>182</td>
<td>67</td>
<td>364</td>
<td>374</td>
</tr>
<tr>
<td><strong>Power (W)</strong></td>
<td>50</td>
<td>10</td>
<td>10</td>
<td>30</td>
<td>0.6</td>
<td>220</td>
</tr>
<tr>
<td><strong>Embed? (GOP/s/W)</strong></td>
<td>0.24</td>
<td>3.7</td>
<td>14.7</td>
<td>1.8</td>
<td>490</td>
<td>1.34</td>
</tr>
</tbody>
</table>

- NeuFlow Virtex6 can run the semantic labeling system at 50ms/frame
- * performance of Nvidia GPU is higher when using minibatch training
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

\[ Y_2 = (Y_1)^2 \]
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**

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

**K-Means,**

\[ E(Y) = \min_z \sum_i \| Y - W_i Z_i \|^2 \]

Z constrained to 1-of-K code
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

#### Visualizing energy surface
- 2 dimensional toy dataset: spiral
- (black = low, white = high)

<table>
<thead>
<tr>
<th>Method</th>
<th>Encoder $W'Y$</th>
<th>Decoder $WZ$</th>
<th>Energy Loss $|Y - WZ|^2$</th>
<th>Loss $F(Y)$</th>
<th>Pull-up Dimensionality</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>PCA</strong></td>
<td></td>
<td></td>
<td>$\sigma(W_e Y)$</td>
<td>$W_d Z$</td>
<td>dimens.</td>
</tr>
<tr>
<td><strong>autoencoder</strong></td>
<td>$W_e Y$</td>
<td>$W_d Z$</td>
<td>$|Y - WZ|^2$</td>
<td>$F(Y)$</td>
<td>dimens.</td>
</tr>
<tr>
<td><strong>sparse coding</strong></td>
<td>$\sigma(W_e Z)$</td>
<td>$W_d Z$</td>
<td>$|Y - WZ|^2$</td>
<td>$F(Y)$</td>
<td>sparsity</td>
</tr>
<tr>
<td><strong>K-Means</strong></td>
<td></td>
<td>$WZ$</td>
<td>$|Y - WZ|^2$</td>
<td>$F(Y)$</td>
<td>1-of-N code</td>
</tr>
</tbody>
</table>
Dictionary Learning With Fast Approximate Inference: Sparse Auto-Encoders
How to Speed Up Inference in a Generative Model?

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

\[ Y \rightarrow \hat{Z} = \text{argmin}_Z E(Y, Z) \]
#6. use a regularizer that limits the volume of space that has low energy

Sparse coding, sparse auto-encoder, Predictive Sparse Decomposition
Examples: most ICA models, Product of Experts

Encoder Architecture

Fast Feed-Forward Model

INPUT Y

Encoder

Distance

Factor A'

Factor B

LATENT VARIABLE Z
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.
Learning to Perform Approximate Inference: Predictive Sparse Decomposition Sparse Auto-Encoders
Sparse auto-encoder: Predictive Sparse Decomposition (PSD)

[Prediction the optimal code with a trained encoder]

Energy = reconstruction_error + code_prediction_error + code_sparsity

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

\[ g_e(W_e, Y^i) = \text{shrinkage}(W_e Y^i) \]
Regularized Encoder-Decoder Model (auto-Encoder) for Unsupervised Feature Learning

- **Encoder**: computes feature vector $Z$ from input $X$
- **Decoder**: reconstructs input $X$ from feature vector $Z$
- **Feature vector**: high dimensional and regularized (e.g. sparse)
- **Factor graph with energy function** $E(X,Z)$ with 3 terms:
  - Linear decoding function and reconstruction error
  - Non-Linear encoding function and prediction error term
  - Pooling function and regularization term (e.g. sparsity)

\[
E(Y,Z) = \| Y - W_d Z \|^2 + \| Z - g_e(W_e, Y) \|^2 + \sum_j \sqrt{\sum_{k \in P_j} Z_k^2}
\]

**INPUT**

- $Y$

**FEATURES**

- $Z$
- $\lambda \sum$
- $\sqrt{\sum Z_k^2}$
- L2 norm within each pool
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
Learning to Perform Approximate Inference
LISTA
ISTA/FISTA: iterative algorithm that converges to optimal sparse code

Better Idea: Give the “right” structure to the encoder

ISTA/FISTA reparameterized:

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

LISTA (Learned ISTA): learn the We and S matrices to get fast solutions

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

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

- Error vs Number of LISTA or FISTA iterations

- Graph comparing reconstruction error for different numbers of iterations for FISTA and LISTA.

- Legend:
  - FISTA (4x)
  - FISTA (1x)
  - LISTA (4x)
  - LISTA (1x)
LISTA with partial mutual inhibition matrix

Reconstruction Error vs. Proportion of S matrix elements that are non zero

- Dim reduction (4x)
- Elements removal (4x)
- Dim reduction (1x)
- Elements removal (1x)

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

The graph shows the reconstruction error as a function of the number of LISTA or FISTA iterations. The legend indicates different iterations:
- CoD (4x)
- CoD (1x)
- LCoD (4x)
- LCoD (1x)

The error decreases as the number of iterations increases, with LCoD (4x) showing the fastest convergence compared to the other methods.
Discriminative Recurrent Sparse Auto-Encoder (DrSAE)

\[ X \xrightarrow{W_e} (\cdot)^+ \xrightarrow{S} + \xrightarrow{(\cdot)^+} \]

- 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

\[
E(Y, Z) = \| Y - \sum_k W_k Z_k \|_2^2 + \alpha \sum_k |Z_k|
\]

Convolutional S.C.

\[
E(Y, Z) = \| Y - \sum_k W_k * Z_k \|_2^2 + \alpha \sum_k |Z_k|
\]

“deconvolutional networks” [Zeiler, Taylor, Fergus CVPR 2010]
Convolutional Formulation

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

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

- **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

\[ \|Y^i - \tilde{Y}\|^2 \]

\[ g_e(W_e, Y^i) \]

\[ W_d Z \]

\[ \|Z - \tilde{Z}\|^2 \]

\[ |z_j| \]

\[ \lambda \sum \]

\[ Y \]

\[ Z \]

\[ \text{FEATURES} \]
Phase 1: train first layer using PSD
Phase 2: use encoder + absolute value as 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
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

\[
g_e(W_e, Y^i) \rightarrow |z_j| \rightarrow g_e(W_e, Y^i) \rightarrow |z_j| \rightarrow \text{FEATURES}
\]
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 5: train a supervised classifier on top
- Phase 6 (optional): train the entire system with supervised back-propagation

\[
Y \xrightarrow{g_e(W_e,Y)} |z_j| \xrightarrow{g_e(W_e,Y')} |z_j| \xrightarrow{\text{classifier}}
\]
Unsupervised + Supervised
For
Pedestrian Detection
Pedestrian Detection, Face Detection

[Osadchy, Miller LeCun JMLR 2007], [Kavukcuoglu et al. NIPS 2010] [Sermanet et al. CVPR 2013]
Feature maps from all stages are pooled/subsampled and sent to the final classification layers

- Pooled low-level features: good for textures and local motifs
- High-level features: good for “gestalt” and global shape

ConvNet Architecture with Multi-Stage Features

<table>
<thead>
<tr>
<th>Task</th>
<th>Single-Stage features</th>
<th>Multi-Stage features</th>
<th>Improvement %</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pedestrians detection (INRIA)</td>
<td>14.26%</td>
<td>9.85%</td>
<td>31%</td>
</tr>
<tr>
<td>Traffic Signs classification (GTSRB) [33]</td>
<td>1.80%</td>
<td>0.83%</td>
<td>54%</td>
</tr>
<tr>
<td>House Numbers classification (SVHN) [32]</td>
<td>5.54%</td>
<td>5.36%</td>
<td>3.2%</td>
</tr>
</tbody>
</table>

[Sermanet, Chintala, LeCun CVPR 2013]
Pedestrian Detection: INRIA Dataset. Miss rate vs false positives

[Kavukcuoglu et al. NIPS 2010] [Sermanet et al. ArXiv 2012]
128 stage-1 filters on Y channel.

Unsupervised training with convolutional predictive sparse decomposition
Unsupervised pre-training with convolutional PSD

- Stage 2 filters.
- Unsupervised training with convolutional predictive sparse decomposition
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

\[
E(Y,Z) = \|Y - W_d Z\|^2 + \|Z - g_e(W_e, Y)\|^2 + \lambda \sum_j \sqrt{\sum_{k \in P_j} Z_k^2}
\]

\[
E(Y,Z) = \|Y^i - \tilde{Y}\|^2 + (W_d Z) + \|Z - \tilde{Z}\|^2 + \lambda \sum_j \sqrt{\sum_{k \in P_j} Z_k^2}
\]
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

**Diagram**

- **INPUT** $Y$
- **Encoder only (PoE, ICA), Decoder Only or Encoder-Decoder (iPSD, RICA)**
- **SIMPLE FEATURES** $Z$
- **L2 norm within each pool** $\lambda \sum \sqrt{\left(\sum Z^2_k\right)}$
- **INvariant FEATURES**
Groups are local in a 2D Topographic Map

- 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.
Image-level training, local filters but no weight sharing

- 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

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

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

Decoder

\[ \begin{align*}
S^t & \quad S^{t-1} & \quad S^{t-2} \\
\bigcirc & & \bigcirc & & \bigcirc \\
W^1 & & W^1 & & W^2 \\
C_1^t & & C_1^{t-1} & & C_1^{t-2} & & C_2^t \\
\end{align*} \]

Predicted input

Inferred code

Encoder

\[ \begin{align*}
C_1^t & \quad C_1^{t-1} & \quad C_1^{t-2} \\
\bigcirc & & \bigcirc & & \bigcirc \\
\tilde{W}^1 & & \tilde{W}^1 & & \tilde{W}^2 \\
f \circ \tilde{W}^1 & & f \circ \tilde{W}^1 & & f \circ \tilde{W}^1 \\
\end{align*} \]

Predicted code

Input
Low-Level Filters Connected to Each Complex Cell

C1
(where)

C2
(what)
Generating images

Input
Memory Networks
Memory Network [Weston, Chopra, Bordes 2014]

- **Add a short-term memory to a network**

  - **I**: (input feature map) – converts the incoming input to the internal feature representation.
  - **G**: (generalization) – updates old memories given the new input.
  - **O**: (output feature map) – produces a new output (in the feature representation space), given the new input and the current memory.
  - **R**: (response) – converts the output into the response format desired. For example, a textual response or an action.

### Results on Question Answering Task

<table>
<thead>
<tr>
<th>Method</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Fader et al., 2013)</td>
<td>0.54</td>
</tr>
<tr>
<td>(Bordes et al., 2014)</td>
<td>0.73</td>
</tr>
<tr>
<td>MemNN</td>
<td>0.71</td>
</tr>
<tr>
<td>MemNN (with BoW features)</td>
<td>0.79</td>
</tr>
</tbody>
</table>

**Fig. 2.** An example story with questions correctly answered by a MemNN. The MemNN was trained on the simulation described in Section 4.2 and had never seen many of these words before, e.g. Bilbo, Frodo and Gollum.
Future Challenges

- Integrated feed-forward and feedback
  - Deep Boltzmann machine do this, but there are issues of scalability.

- Integrating supervised and unsupervised learning in a single algorithm

- Integrating deep learning and structured prediction ("reasoning")
  - This has been around since the 1990's but needs to be revived

- Learning representations for complex reasoning
  - "recursive" networks [Pollack 90's] [Bottou 10] [Socher 11]

- Integrating Deep Learning with "memory"
  - LSTM [Hochreiter 97], MemNN [Weston 14], NTM [Graves 14]

- Representation learning in natural language processing
  - [Y. Bengio 01], [Collobert Weston 10], [Mnih Hinton 11] [Socher 12]

- Better theoretical understanding of deep learning and convolutional nets
  - e.g. Stephane Mallat's "scattering transform", work on the sparse representations from the applied math community....
Towards Practical AI: Challenges

- Applying deep learning to NLP (requires “structured prediction”)
- Video analysis/understanding (requires unsupervised learning)
- High-performance/low power embedded systems for ConvNets (FPGA/ASIC?)
- Very-large-scale deep learning (distributed optimization)
- Integrating reasoning with DL (“energy-based models”, recursive neural nets)

Then we can have
- Automatically-created high-performance data analytics systems
- Vector-space embedding of everything (language, users,...)
- Multimedia content understanding, search and indexing
- Multilingual speech dialog systems
- Driver-less cars
- Autonomous maintenance robots / personal care robots
Marrying feed-forward convolutional nets with generative “deconvolutional nets”
- Deconvolutional networks
  - [Zeiler-Graham-Fergus ICCV 2011]

Feed-forward/Feedback networks allow reconstruction, multimodal prediction, restoration, etc...
- Deep Boltzmann machines can do this, but there are scalability issues with training

Finding a single rule for supervised and unsupervised learning
- Deep Boltzmann machines can also do this, but there are scalability issues with training
The Graph of Deep Learning ↔ Sparse Modeling ↔ Neuroscience

- **Compr. Sensing** [Candès-Tao 04]
- **L2-L1 optim** [Nesterov, Nemirovski, Daubechies, Osher, ...]
- **Restricted Boltzmann Machine** [Hinton 05]
- **MCMC, HMC**
- **Cont. Div.** [Neal, Hinton]
- **Sparse Auto-Encoder** [LeCun 06; Ng 07]
- **Speech Recognition** [Goog, IBM, MSFT 12]
- **Object Recog** [Hinton 12]
- **Scene Labeling** [LeCun 12]
- **Connectomics** [Seung 10]
- **Neocognitron** [Fukushima 82]
- **Scattering Transform** [Mallat 10]
- **Normalization** [Simoncelli 94]
- **Visual Metamers** [Simoncelli 12]

**Backprop** [many 85]

**Sparse Modeling** [Olshausen-Field 97]

**Basis/Matching Pursuit** [Mallat 93; Donoho 94]

**Stochastic Optimization** [Nesterov, Bottou, Nemirovski, ...]

**Convolutional Net** [LeCun 89]

**Object Recog** [LeCun 10]