Deep Learning

MLSS 2014
Reykjavik, Iceland

Yoshua Bengio
April 26-27, 2014
Outline of the Tutorial

1. Representation Learning, motivations

2. Algorithms
   - Feedforward deep networks
   - Convolutional nets
   - Recurrent and recursive nets
   - Regularization and optimization
   - RBMs, DBMs and DBNs
   - Regularized auto-encoders

3. Practical Considerations and Applications

4. Challenges & Ongoing Work

Ultimate Goals

- AI
- Needs **knowledge**
- Needs **learning**
  (involves priors + *optimization/search*)
- Needs **generalization**
  (guessing where probability mass concentrates)
- Needs ways to fight the curse of dimensionality
  (exponentially many configurations of the variables to consider)
- Needs disentangling the underlying explanatory factors
  (making sense of the data)
Part 1

Intro & Motivations for Representation Learning and Deep Learning
Representation Learning

- Good **features** essential for successful ML: 90% of effort

  - Handcrafting features vs learning them

- Good representation?
- guesses
  - the features / factors / causes
Google Image Search: Different object types represented in the same space

Google:
S. Bengio, J. Weston & N. Usunier

Learn \( \Phi_1(\cdot) \) and \( \Phi_w(\cdot) \) to optimize precision@k.
Following up on (Bengio et al. NIPS’2000)
Neural word embeddings – visualization
Analogical Representations for Free (Mikolov et al, ICLR 2013)

• Semantic relations appear as linear relationships in the space of learned representations
• King – Queen ≈ Man – Woman
• Paris – France + Italy ≈ Rome
Deep Representation Learning

Learn multiple levels of representation of increasing complexity/abstraction

- theory: exponential gain
- brains are deep
- cognition is compositional
- Better mixing (Bengio et al, ICML 2013)

- They work! SOTA on industrial-scale AI tasks
  (object recognition, speech recognition, language modeling, music modeling)
Learning multiple levels of representation

Successive model layers learn deeper intermediate representations

Prior: underlying factors & concepts compactly expressed w/ multiple levels of abstraction

(Lee, Largman, Pham & Ng, NIPS 2009)
(Lee, Grosse, Ranganath & Ng, ICML 2009)
subroutine1 includes subsub1 code and subsub2 code and subsubsub1 code

subroutine2 includes subsub2 code and subsub3 code and subsubsub3 code and ...

main

“Shallow” computer program
“Deep” computer program
Deep Architectures are More Expressive

Theoretical arguments:

2 layers of

- Logic gates
- Formal neurons
- RBF units

= universal approximator

RBM & auto-encoders = universal approximator

Theorems on advantage of depth:
(Hastad et al 86 & 91, Bengio et al 2007, Bengio & Delalleau 2011, Braverman 2011)

Some functions compactly represented with k layers may require exponential size with 2 layers
Breakthrough in 2006

- Ability to train deep architectures by using layer-wise unsupervised learning, whereas previous purely supervised attempts had failed

- Unsupervised feature learners:
  - RBMs
  - Auto-encoder variants
  - Sparse coding variants
Stacking Single-Layer Learners

• One of the big ideas from 2006: layer-wise unsupervised feature learning

Stacking Restricted Boltzmann Machines (RBM) \(\Rightarrow\) Deep Belief Network (DBN)

Stacking regularized auto-encoders \(\Rightarrow\) deep neural nets
Deep Supervised Neural Nets

• Now can train them even without unsupervised pre-training:
  better initialization and non-linearities (rectifiers, maxout), generalize well with large labeled sets and regularizers (dropout)

• Unsupervised pre-training:
  rare classes, transfer, smaller labeled sets, or as extra regularizer.
Deep Learning in the News

WIRED

Researcher Dreams Up Machines That Learn Without Humans
06.27.13

The New York Times
Scientists See Promise in Deep-Learning Programs
John Markoff
November 23, 2012

THE GLOBE AND MAIL
Google taps U of T professor to teach context to computers
03.11.13

Despite Itself, a Simulated Brain Seeks Cats
By JOHN MARKOFF 12 minutes ago
A Google research team, led by Andrew Y. Ng, above, and Jeff Dean, created a neural network of 16,000 processors that reflected human obsession with Internet felines.
Deep Learning

With massive amounts of computational power, machines can now recognize objects and translate speech in real time. Artificial intelligence is finally getting smart.
Deep Learning

When the number of levels can be data-selected, this is a deep architecture
A Good Old Deep Architecture: MLPs

Output layer
Here predicting a supervised target

Hidden layers
These learn more abstract representations as you head up

Input layer
This has raw sensory inputs (roughly)
A (Vanilla) Modern Deep Architecture

Optional Output layer
Here predicting or conditioning on a supervised target

Hidden layers
These learn more abstract representations as you head up

Input layer
Inputs can be reconstructed, filled-in or sampled
To generalize locally, need representative examples for all relevant variations!

Classical solution: hope for a smooth enough target function, or make it smooth by handcrafting good features / kernel
Easy Learning

learned function: prediction = f(x)

true unknown function

* = example (x,y)

learned function: prediction = f(x)
Local Smoothness Prior: Locally Capture the Variations

* = training example

true function: unknown

learnt = interpolated

prediction

f(x)

test point x
However, Real Data Are near Highly Curved Sub-Manifolds
Not Dimensionality so much as Number of Variations

(Bengio, Dellalleau & Le Roux 2007)

- **Theorem:** Gaussian kernel machines need at least $k$ examples to learn a function that has $2k$ zero-crossings along some line

- **Theorem:** For a Gaussian kernel machine to learn some maximally varying functions over $d$ inputs requires $O(2^d)$ examples
Geometrical view on machine learning

- Generalization: guessing where *probability* mass concentrates
- Challenge: the curse of dimensionality (exponentially many configurations of the variables to consider)
For AI Tasks: Manifold structure

- examples **concentrate** near a lower dimensional “manifold
- **Evidence:** most input configurations are unlikely
Putting Probability Mass where Structure is Plausible

- Empirical distribution: mass at training examples
- Smoothness: spread mass around
- Insufficient
- Guess some ‘structure’ and generalize accordingly
Is there any hope to generalize non-locally?

Yes! Need good priors!
Bypassing the curse

We need to build compositionality into our ML models

Just as human languages exploit compositionality to give representations and meanings to complex ideas

Exploiting compositionality gives an exponential gain in representational power

Distributed representations / embeddings: feature learning

Deep architecture: multiple levels of feature learning

Prior: compositionality is useful to describe the world around us efficiently
Six Good Reasons to Explore Representation Learning
#1 Learning features, not just handcrafting them

Most ML systems use very carefully hand-designed features and representations

Many practitioners are very experienced – and good – at such feature design (or kernel design)

“Machine learning” often reduces to linear models (including CRFs) and nearest-neighbor-like features/models (including n-grams, kernel SVMs, etc.)

Hand-crafting features is time-consuming, brittle, incomplete
#2 Distributed Representations

Many neurons active simultaneously in the brain: around 1%

The input is represented by the activation of a set of features that are not mutually exclusive. Can be exponentially more efficient than local representations
Non-distributed representations

- Clustering, Nearest-Neighbors, RBF SVMs, local non-parametric density estimation & prediction, decision trees, etc.

- Parameters for each distinguishable region

- \# of distinguishable regions is linear in \# of parameters

→ No non-trivial generalization to regions without examples
#2 The need for distributed representations

- Factor models, PCA, RBMs, Neural Nets, Sparse Coding, Deep Learning, etc.

- Each parameter influences many regions, not just local neighbors

- # of distinguishable regions grows almost exponentially with # of parameters

- GENERALIZE NON-LOCALLY TO NEVER-SEEN REGIONS

The need for distributed representations also underscores the importance of multiclustering and non-mutually exclusive features/attributes. This approach ensures that the model can generalize non-locally to never-seen regions, thereby enhancing its versatility and adaptability.
The need for distributed representations

Learning a set of features that are not mutually exclusive can be exponentially more statistically efficient than having nearest-neighbor-like or clustering-like models.
Today, most practical ML applications require (lots of) labeled training data

But almost all data is unlabeled, e.g. text, images on the web

The brain needs to learn about $10^{14}$ synaptic strengths

... in about $10^9$ seconds

Labels cannot possibly provide enough information

Most information acquired in an *unsupervised* fashion
#3 How do humans generalize from very few examples?

- They **transfer** knowledge from previous learning:
  - Representations
  - Explanatory factors

- Previous learning from: unlabeled data
  + labels for other tasks

- **Prior:** shared underlying explanatory factors, in particular between $P(x)$ and $P(Y|x)$
#3 Sharing Statistical Strength by Semi-Supervised Learning

- Hypothesis: $P(x)$ shares structure with $P(y|x)$
Why Semi-Supervised Learning Works

• The labeled examples (circles) help to identify the class of each cluster of unlabeled examples.

• The unlabeled examples (colored dots) help to identify the shape of each cluster.

few labeled examples + "happiness" = many unlabeled examples
Depth

Deep Architecture in the Brain

Area V1

Area V2

Retina

Higher level visual abstractions

Primitive shape detectors

Edge detectors

pixels
#4 Levels of Representation

very high level representation:

![CAT](image1)

... etc ...

slightly higher level representation

raw input vector representation:

\[ x = \begin{bmatrix} 23 & 19 & 20 \\ x_1 & x_2 & x_3 \end{bmatrix} \]

![Image of a cat jumping](image2)
Humans organize their ideas and concepts hierarchically
Humans first learn simpler concepts and then compose them to represent more abstract ones
Engineers break-up solutions into multiple levels of abstraction and processing

It would be good to automatically learn / discover these concepts (knowledge engineering failed because of superficial introspection?)
Learning multiple levels of representation

There is theoretical and empirical evidence in favor of multiple levels of representation

Exponential gain for some families of functions

Biologically inspired learning

Brain has a deep architecture

Cortex seems to have a generic learning algorithm

Humans first learn simpler concepts and then compose them into more complex ones
#4 Sharing Components in a Deep Architecture

Polynomial expressed with shared components: advantage of depth may grow exponentially

\[(x_1x_2)(x_2x_3) + (x_1x_2)(x_3x_4) + (x_2x_3)^2 + (x_2x_3)(x_3x_4)\]

Sum-product network

Theorems in
(Bengio & Delalleau, ALT 2011; Delalleau & Bengio NIPS 2011)
#4 Handling the compositionality of human language and thought

- Human languages, ideas, and artifacts are composed from simpler components

- **Recursion**: the same operator (same parameters) is applied repeatedly on different states/components of the computation

- Result after unfolding = deep computation / representation

(Bottou 2011, Socher et al 2011)
#5 Multi-Task Learning

- Generalizing better to new tasks (tens of thousands!) is crucial to approach AI

- Deep architectures learn good intermediate representations that can be shared across tasks
  (Collobert & Weston ICML 2008, Bengio et al AISTATS 2011)

- Good representations that disentangle underlying factors of variation make sense for many tasks because each task concerns a subset of the factors

  Prior: shared underlying explanatory factors between tasks
Combining Multiple Sources of Evidence with Shared Representations

- Traditional ML: data = matrix
- Relational learning: multiple sources, different tuples of variables
- Share representations of same types across data sources
- Shared learned representations help propagate information among data sources: e.g., WordNet, XWN, Wikipedia, FreeBase, ImageNet... (Bordes et al AISTATS 2012, ML J. 2013)
- FACTS = DATA
- Deduction = Generalization
Different object types represented in same space

Google:
S. Bengio, J. Weston & N. Usunier

Learn $\Phi(x)$ and $\Phi_w(x)$ to optimize precision@k.
Invariance and Disentangling

- Invariant features
- Which invariances?
- Alternative: learning to disentangle factors
- Good disentangling ⇒ avoid the curse of dimensionality
Emergence of Disentangling

- **(Goodfellow et al. 2009)**: sparse auto-encoders trained on images
  - some higher-level features more invariant to geometric factors of variation

- **(Glorot et al. 2011)**: sparse rectified denoising auto-encoders trained on bags of words for sentiment analysis
  - different features specialize on different aspects (domain, sentiment)

WHY?
#6 Sparse Representations

- Just add a sparsifying penalty on learned representation (prefer 0s in the representation)
- Information disentangling (compare to dense compression)
- More likely to be linearly separable (high-dimensional space)
- Locally low-dimensional representation = local chart
- Hi-dim. sparse = efficient variable size representation = data structure

Prior: only few concepts and attributes relevant per example
Deep Sparse Rectifier Neural Networks
(Glorot, Bordes and Bengio AISTATS 2011), following up on (Nair & Hinton 2010) softplus RBMs

**Neuroscience motivations**
Leaky integrate-and-fire model

**Machine learning motivations**
- Sparse representations
- Sparse gradients
- Trains deep nets even w/o pretraining

**Rectifier**
\[ f(x) = \max(0, x) \]

**Outstanding results** by Krizhevsky et al 2012
killing the state-of-the-art on ImageNet 1000:

<table>
<thead>
<tr>
<th>2^nd best</th>
<th>1^st choice</th>
<th>Top-5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Previous SOTA</td>
<td>45% err</td>
<td>26% err</td>
</tr>
<tr>
<td>Krizhevsky et al</td>
<td>37% err</td>
<td>15% err</td>
</tr>
</tbody>
</table>
Temporal Coherence and Scales

- Hints from nature about different explanatory factors:
  - Rapidly changing factors (often noise)
  - Slowly changing (generally more abstract)
  - Different factors at different time scales

- Exploit those hints to disentangle better!
Bypassing the curse by sharing statistical strength

• Besides very fast GPU-enabled predictors, the main advantage of representation learning is statistical: potential to learn from less labeled examples because of sharing of statistical strength:
  • Re-use, combination and composition of learned functions/factors
  • Unsupervised pre-training and semi-supervised training
  • Multi-task learning
  • Multi-data sharing, learning about symbolic objects and their relations
Why now?

Despite prior investigation and understanding of many of the algorithmic techniques ...

Before 2006 training deep architectures was unsuccessful (except for convolutional neural nets when used by people who speak French)

What has changed?

• New methods for unsupervised pre-training have been developed (variants of Restricted Boltzmann Machines = RBMs, regularized auto-encoders, sparse coding, etc.)
• New methods to successfully train deep supervised nets even without unsupervised pre-training
• Successful real-world applications, winning challenges and beating SOTAs in various areas, large-scale industrial apps
Major Breakthrough in 2006

- Ability to train deep architectures by using layer-wise unsupervised learning, whereas previous purely supervised attempts had failed

- Unsupervised feature learners:
  - RBMs
  - Auto-encoder variants
  - Sparse coding variants
More about depth
Architecture Depth

Element set

- *
- sin
- +
- -

Inputs: x, a, b

Output:

* sin +

Element set

- neuron
- neuron
- ...
- neuron

Output:

neuron neuron neuron neuron

Inputs:
Deep Architectures are More Expressive

Theoretical arguments:

2 layers of

Logic gates
Formal neurons
RBF units

= universal approximator

RBM & auto-encoders = universal approximator

Theorems on advantage of depth:

Some functions compactly represented with k layers may require exponential size with 2 layers
“Deep” computer program
subroutine1 includes subsub1 code and subsub2 code and subsubsub1 code

subroutine2 includes subsub2 code and subsub3 code and subsubsub3 code and ...

main

“Shallow” computer program
“Deep” circuit
Falsely reassuring theorems: one can approximate any reasonable (smooth, boolean, etc.) function with a 2-layer architecture
Good work -- but I think we might need a little more detail right here.
Part 2

Representation Learning Algorithms
A neural network = running several logistic regressions at the same time

If we feed a vector of inputs through a bunch of logistic regression functions, then we get a vector of outputs

But we don’t have to decide ahead of time what variables these logistic regressions are trying to predict!
A neural network = running several logistic regressions at the same time

... which we can feed into another logistic regression function

and it is the training criterion that will decide what those intermediate binary target variables should be, so as to make a good job of predicting the targets for the next layer, etc.
A neural network = running several logistic regressions at the same time

- Before we know it, we have a multilayer neural network....
Back-Prop

- Compute gradient of example-wise loss wrt parameters

- Simply applying the derivative chain rule wisely

\[ z = f(y) \quad y = g(x) \quad \frac{\partial z}{\partial x} = \frac{\partial z}{\partial y} \frac{\partial y}{\partial x} \]

- If computing the loss(example, parameters) is \(O(n)\) computation, then so is computing the gradient
Simple Chain Rule

\[ \Delta z = \frac{\partial z}{\partial y} \Delta y \]
\[ \Delta y = \frac{\partial y}{\partial x} \Delta x \]
\[ \Delta z = \frac{\partial z}{\partial y} \frac{\partial y}{\partial x} \Delta x \]
\[ \frac{\partial z}{\partial x} = \frac{\partial z}{\partial y} \frac{\partial y}{\partial x} \]
Multiple Paths Chain Rule

\[ \frac{\partial z}{\partial x} = \frac{\partial z}{\partial y_1} \frac{\partial y_1}{\partial x} + \frac{\partial z}{\partial y_2} \frac{\partial y_2}{\partial x} \]
Multiple Paths Chain Rule - General

\[
\frac{\partial z}{\partial x} = \sum_{i=1}^{n} \frac{\partial z}{\partial y_i} \frac{\partial y_i}{\partial x}
\]
Chain Rule in Flow Graph

Flow graph: any directed acyclic graph
node = computation result
arc = computation dependency

\[ \{y_1, y_2, \ldots, y_n\} = \text{successors of } x \]

\[ \frac{\partial z}{\partial x} = \sum_{i=1}^{n} \frac{\partial z}{\partial y_i} \frac{\partial y_i}{\partial x} \]
Back-Prop in Multi-Layer Net

\[
NLL = -\log P(Y = y|x)
\]

\[
P(Y = y|x) = \text{softmax}(Wb)
\]

\[
h = \tanh(Vx)
\]
Back-Prop in General Flow Graph

Single scalar output $z$

1. Fprop: visit nodes in topo-sort order
   - Compute value of node given predecessors
2. Bprop:
   - initialize output gradient $= 1$
   - visit nodes in reverse order:
     Compute gradient wrt each node using gradient wrt successors

\[
\{y_1, y_2, \ldots, y_n\} = \text{successors of } x
\]

\[
\frac{\partial z}{\partial x} = \sum_{i=1}^{n} \frac{\partial z}{\partial y_i} \frac{\partial y_i}{\partial x}
\]
Back-Prop in Recurrent & Recursive Nets

- Replicate a parameterized function over different time steps or nodes of a DAG
- Output state at one time-step/node is used as input for another time-step/node
Backpropagation Through Structure

• Inference $\rightarrow$ discrete choices
  • (e.g., shortest path in HMM, best output configuration in CRF)
• E.g. Max over configurations or sum weighted by posterior
• The loss to be optimized depends on these choices
• The inference operations are flow graph nodes
• If continuous, can perform stochastic gradient descent
  • $\text{Max}(a,b)$ is continuous.
Automatic Differentiation

- The gradient computation can be automatically inferred from the symbolic expression of the fprop.
- Each node type needs to know how to compute its output and how to compute the gradient wrt its inputs given the gradient wrt its output.
- Easy and fast prototyping
Deep Supervised Neural Nets

• We can now train them even without unsupervised pre-training, thanks to better initialization and non-linearities (rectifiers, maxout) and they can generalize well with large labeled sets and dropout.

• Unsupervised pre-training still useful for rare classes, transfer, smaller labeled sets, or as an extra regularizer.

- **Dropouts** trick: during training multiply neuron output by random bit (p=0.5), during test by 0.5
- Used in deep supervised networks
- Similar to denoising auto-encoder, but corrupting every layer
- Works better with some non-linearities (rectifiers, maxout) (Goodfellow et al. ICML 2013)
- Equivalent to averaging over exponentially many architectures
  - Used by Krizhevsky et al to break through ImageNet SOTA
  - Also improves SOTA on CIFAR-10 (18→16% err)
  - Knowledge-free MNIST with DBMs (.95→.79% err)
  - TIMIT phoneme classification (22.7→19.7% err)
Dropout Regularizer: Super-Efficient Bagging
Temporal & Spatial Inputs: Convolutional & Recurrent Nets

- Local connectivity across time/space
- Sharing weights across time/space (translation equivariance)
- Pooling (translation invariance, cross-channel pooling for learned invariances)

Recurrent nets (RNNs) can summarize information from the past
Bidirectional RNNs also summarize information from the future
Convolution = sparse connectivity + parameter sharing

\[ s[t] = (x \ast w)(t) = \sum_{a=-\infty}^{\infty} x[a] w[t - a] \]
Pooling Layers

• Aggregate to achieve local invariance

Max-pooling

Effect of translation

• Subsampling to reduce temporal/spatial scale and computation
Multiple Convolutions: Feature Maps
Alternating convolutions & pooling

- Inspired by visual cortex, idea from Fukushima’s Neocognitron, combined with back-prop and developed by **LeCun** since 1989
- Increasing number of features, decreasing spatial resolution
- Top layers are fully connected

Krizhevsky, Sutskever & Hinton 2012 breakthrough in object recognition
Distributed Representations & Neural Nets:

How to do unsupervised training?
Unsupervised and Transfer Learning Challenge + Transfer Learning Challenge: Deep Learning 1st Place

ICML’2011 workshop on Unsup. & Transfer Learning

NIPS’2011 Transfer Learning Challenge
Paper: ICML’2012

Unsupervised and Transfer Learning Challenge: Deep Learning 1st Place

ICML’2011 workshop on Unsup. & Transfer Learning

NIPS’2011 Transfer Learning Challenge
Paper: ICML’2012
PCA

= Linear Manifold
= Linear Auto-Encoder
= Linear Gaussian Factors

input $x$, 0-mean
features = code = $h(x) = W x$
reconstruction($x$) = $W^T h(x) = W^T W x$
$W$ = principal eigen-basis of $\text{Cov}(X)$

Probabilistic interpretations:
1. Gaussian with full covariance $W^T W + \lambda I$
2. Latent marginally iid Gaussian factors $h$ with
   $x = W^T h + \text{noise}$
Directed Factor Models:
\[ P(x,h) = P(h)P(x|h) \]

- \( P(h) \) factorizes into \( P(h_1) \ P(h_2) \ldots \)
- Different priors:
  - PCA: \( P(h_i) \) is Gaussian
  - ICA: \( P(h_i) \) is non-parametric
  - **Sparse coding**: \( P(h_i) \) is concentrated near 0
- Likelihood is typically Gaussian \( x \mid h \) with mean given by \( W^T h \)
- **Inference** procedures (predicting \( h \), given \( x \)) differ
- **Sparse \( h \)**: \( x \) is explained by the weighted addition of selected filters \( h_i \)

\[ x = 0.9x \begin{array}{l} h_1 \end{array} + 0.8x \begin{array}{l} h_3 \end{array} + 0.7x \begin{array}{l} h_5 \end{array} \]
Sparse autoencoder illustration for images

Natural Images

Learned bases:

Test example

\[
[h_1, \ldots, h_{64}] = [0, 0, \ldots, 0, 0.8, 0, \ldots, 0, 0.3, 0, \ldots, 0, 0.5, 0]
\]

(feature representation)
Stacking Single-Layer Learners

- PCA is great but can’t be stacked into deeper more abstract representations (linear x linear = linear)
- One of the big ideas from Hinton et al. 2006: layer-wise unsupervised feature learning

Stacking Restricted Boltzmann Machines (RBM) \(\rightarrow\) Deep Belief Network (DBN)
Effective deep learning first became possible with unsupervised pre-training

[Erhan et al., JMLR 2010]

(with RBMs and Denoising Auto-Encoders)
Optimizing Deep Non-Linear Composition of Functions Seems Hard

- Failure of training deep supervised nets before 2006
- Regularization effect + optimization effect of unsupervised pre-training
- Is optimization difficulty due to
  - ill-conditioning?
  - local minima?
  - something else?
- The jury is still out, but we now have success stories of training deep supervised nets without unsupervised pre-training
Initial Examples Matter More (critical period?)

Variance of the output

- 1-layer network without pretraining
- 1-layer network with RBM pre-training

Fraction at which we vary the examples

Mean of the variance and the std to the mean

0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9

0 1 2 3 4 5 6 7 8
Order & Selection of Examples Matters

(Bengio, Louradour, Collobert & Weston, ICML’2009)

- Curriculum learning
  - (Bengio et al 2009, Krueger & Dayan 2009)
  - Start with easier examples
  - Faster convergence to a better local minimum in deep architectures
Curriculum Learning

Guided learning helps training humans and animals

Start from simpler examples / easier tasks  (Piaget 1952, Skinner 1958)
Continuation Methods

Target objective

Heavily smoothed objective = surrogate criterion

Final solution

Track local minima

Easy to find minimum
Layer-wise Unsupervised Learning

input ...
Layer-Wise Unsupervised Pre-training

features

input
Layer-Wise Unsupervised Pre-training

reconstruction of input

features

input

\[ \text{input} = ? \]
Layer-Wise Unsupervised Pre-training
Layer-Wise Unsupervised Pre-training

More abstract features → … → … → …
features

input
Layer-wise Unsupervised Learning

reconstruction of features

More abstract features

features

input
Layer-Wise Unsupervised Pre-training

More abstract features
features
input
Layer-wise Unsupervised Learning

Even more abstract features

More abstract features

features

input
Supervised Fine-Tuning

- Additional hypothesis: features good for $P(x)$ good for $P(y|x)$
Greedy Layerwise Supervised Training

Generally worse than unsupervised pre-training but better than ordinary training of a deep neural network (Bengio et al. NIPS’2006). Has been used successfully on large labeled datasets, where unsupervised pre-training did not make as much of an impact.
Supervised Fine-Tuning is Important

- Greedy layer-wise unsupervised pre-training phase with RBMs or auto-encoders on MNIST
- Supervised phase with or without unsupervised updates, with or without fine-tuning of hidden layers
- Can train all RBMs at the same time, same results
Understanding the difficulty of training deep feedforward supervised neural networks

(Glorot & Bengio, AISTATS 2010)

Study the activations and gradients
  • wrt depth
  • as training progresses
  • for different initializations $\rightarrow$ big difference
  • for different non-linearities $\rightarrow$ big difference

First demonstration that deep supervised nets can be successfully trained almost as well as with unsupervised pre-training, by setting up the optimization problem appropriately...
Restricted Boltzmann Machines
Undirected Models: the Restricted Boltzmann Machine

[Hinton et al 2006]

• Probabilistic model of the joint distribution of the observed variables (inputs alone or inputs and targets) $x$

• Latent (hidden) variables $h$ model high-order dependencies

• Inference is easy, $P(h|x)$ factorizes into product of $P(h_i | x)$


• See Hinton (2010)
  “A practical guide to training Restricted Boltzmann Machines”
Boltzmann Machines & MRFs

- Boltzmann machines:
  
  $P(x) = \frac{1}{Z} e^{-\text{Energy}(x)} = \frac{1}{Z} e^{c^T x + x^T W x} = \frac{1}{Z} e^{\sum_i c_i x_i + \sum_{i,j} x_i W_{ij} x_j}$

- Markov Random Fields:

  $P(x) = \frac{1}{Z} e^{\sum_i w_i f_i(x)}$

- More interesting with latent variables!

Undirected graphical models

Soft constraint / probabilistic statement
Restricted Boltzmann Machine (RBM)

\[ P(x, h) = \frac{1}{Z} e^{b^T h + c^T x + h^T W x} = \frac{1}{Z} e^{\sum_i b_i h_i + \sum_j c_j x_j + \sum_{i,j} h_i W_{ij} x_j} \]

- A popular building block for deep architectures
- **Bipartite** undirected graphical model
Block Gibbs Sampling in RBMs

\[ P(h \mid x) \text{ and } P(x \mid h) \text{ factorize} \]
\[ P(h \mid x) = \prod_i P(h_i \mid x) \]

\[ P(x, h) = \frac{1}{\mathcal{Z}} e^{b^T h + c^T x + h^T W x} \]

- Easy inference
- Efficient block Gibbs sampling: \( x \rightarrow h \rightarrow x \rightarrow h \ldots \)
RBM with (image, label) visible units

(Larochelle & Bengio 2008)
RBMs are Universal Approximators

(Le Roux & Bengio 2008)

- Adding one hidden unit (with proper choice of parameters) guarantees increasing likelihood
- With enough hidden units, can perfectly model any discrete distribution
- RBMs with variable # of hidden units = non-parametric
RBM Conditionals Factorize

\[ P(h|x) = \frac{\exp(b'x + c'h + h'Wx)}{\sum_{h} \exp(b'x + c'h + \tilde{h}'Wx)} \]

\[ = \frac{\prod_i \exp(c_i h_i + h_i W_i x)}{\prod_i \sum_{\tilde{h}_i} \exp(c_i \tilde{h}_i + \tilde{h}_i W_i x)} \]

\[ = \prod_i \frac{\exp(h_i(c_i + W_i x))}{\sum_{\tilde{h}_i} \exp(h_i(c_i + W_i x))} \]

\[ = \prod_i P(h_i|x). \]
RBM Energy Gives Binomial Neurons

With $h_i \in \{0, 1\}$, recall $\text{Energy}(x, h) = -b^T x - c^T h - h^T W x$

$$P(h_i = 1|x) = \frac{e^{c_i + W_i x + \text{other terms}}}{e^{c_i + W_i x + \text{other terms}} + e^{0c_i + 0W_i x + \text{other terms}}}$$

$$= \frac{e^{c_i + W_i x}}{e^{c_i + W_i x} + 1}$$

$$= \frac{1}{1 + e^{c_i - W_i x}}$$

$$= \text{sigm}(c_i + W_i x).$$

since $\text{sigm}(a) = \frac{1}{1+e^{-a}}$. 
RBM Free Energy

\[ P(x, h) = \frac{e^{-\text{Energy}(x,h)}}{Z} \]

- Free Energy = equivalent energy when marginalizing
  \[ P(x) = \sum_h \frac{e^{-\text{Energy}(x,h)}}{Z} = \frac{e^{-\text{FreeEnergy}(x)}}{Z} \]
- Can be computed exactly and efficiently in RBMs
  \[ \text{FreeEnergy}(x) = -b'x - \sum_i \log \sum_{h_i} e^{h_i(c_i + W_i x)} \]
- Marginal likelihood \( P(x) \) tractable up to partition function \( Z \)
Energy-Based Models Gradient

\[ P(x) = \frac{e^{-\text{Energy}(x)}}{Z} \quad Z = \sum_x e^{-\text{Energy}(x)} \]

\[ \frac{\partial \log P(x)}{\partial \theta} = -\frac{\partial \text{Energy}(x)}{\partial \theta} - \frac{\partial \log Z}{\partial \theta} \]

\[ \frac{\partial \log Z}{\partial \theta} = \frac{\partial \log \sum_x e^{-\text{Energy}(x)}}{\partial \theta} \]

\[ = \frac{1}{Z} \frac{\partial \sum_x e^{-\text{Energy}(x)}}{\partial \theta} \]

\[ = -\frac{1}{Z} \sum_x e^{-\text{Energy}(x)} \frac{\partial \text{Energy}(x)}{\partial \theta} \]

\[ = -\sum_x P(x) \frac{\partial \text{Energy}(x)}{\partial \theta} \]
Boltzmann Machine Gradient

\[ P(x) = \frac{1}{Z} \sum_h e^{-\text{Energy}(x,h)} = \frac{1}{Z} e^{-\text{FreeEnergy}(x)} \]

- Gradient has two components:

\[
\frac{\partial \log P(x)}{\partial \theta} = -\sum_h P(h|x) \frac{\partial \text{Energy}(x,h)}{\partial \theta} + \sum_{\tilde{x}} P(\tilde{x}) \frac{\partial \text{FreeEnergy}(\tilde{x})}{\partial \theta} + \sum_{\tilde{x},\tilde{h}} P(\tilde{x},\tilde{h}) \frac{\partial \text{Energy}(\tilde{x},\tilde{h})}{\partial \theta}
\]

- In RBMs, easy to sample or sum over \( h|x \)
- Difficult part: sampling from \( P(x) \), typically with a Markov chain
Positive & Negative Samples

- Observed (+) examples push the energy down
- Generated / dream / fantasy (-) samples / particles push the energy up

Equilibrium: $E[\text{gradient}] = 0$
Training RBMs

Contrastive Divergence: start negative Gibbs chain at observed $x$, run $k$ Gibbs steps

SML/Persistent CD: run negative Gibbs chain in background while weights slowly change

Fast PCD: two sets of weights, one with a large learning rate only used for negative phase, quickly exploring modes

Herding: Deterministic near-chaos dynamical system defines both learning and sampling

Tempered MCMC: use higher temperature to escape modes
Contrastive Divergence

Contrastive Divergence (CD-k): start negative phase block Gibbs chain at observed \( x \), run \( k \) Gibbs steps (Hinton 2002)

\[ h^+ \sim P(h \mid x^+) \]

\[ h^- \sim P(h \mid x^-) \]

Free Energy

push down

push up

Observed \( x^+ \) positive phase

\( k = 2 \) steps

Sampled \( x^- \) negative phase
Persistent CD (PCD) / Stochastic Max. Likelihood (SML)

Run negative Gibbs chain in background while weights slowly change (Younes 1999, Tieleman 2008):

- Guarantees (Younes 1999; Yuille 2005)
- If learning rate decreases in $1/t$,
  chain mixes before parameters change too much,
  chain stays converged when parameters change

$$h^+ \sim P(h \mid x^+)$$

Observed $x^+$ (positive phase)
Obstacle: Vicious Circle Between Learning and MCMC Sampling

- Early during training, density smeared out, mode bumps overlap

- Later on, hard to cross empty voids between modes

Are we doomed if we rely on MCMC during training? Will we be able to train really large & complex models?
Some RBM Variants

- Different energy functions and allowed values for the hidden and visible units:
  - Hinton et al 2006: binary-binary RBMs
  - Welling NIPS’2004: exponential family units
  - Ranzato & Hinton CVPR’2010: Gaussian RBM weaknesses (no conditional covariance), propose mcRBM
  - Ranzato et al NIPS’2010: mPoT, similar energy function
  - Courville et al ICML’2011: spike-and-slab RBM
Convolutionally Trained
Spike & Slab RBMs Samples
ssRBM is not Cheating

Samples from $\mu$-ssRBM:

Nearest examples in CIFAR:
(least square dist.)
Auto-Encoders & Variants: Learning a computational graph
Computational Graphs

• Operations for particular task

• Neural nets’ structure = computational graph for $P(y \mid x)$

• Graphical model’s structure ≠ computational graph for inference

• Recurrent nets & graphical models

→ family of computational graphs sharing parameters

• Could we have a parametrized family of computational graphs defining “the model”?
Simple Auto-Encoders

- MLP whose target output = input
- Reconstruction=decoder(encoder(input)), e.g.
  \[ h = \tanh(b + Wx) \]
  \[ \text{reconstruction} = \tanh(c + W^T h) \]
  \[ \text{Loss } L(x, \text{reconstruction}) = ||\text{reconstruction} - x||^2 \]

- With bottleneck, code = new coordinate system
- Encoder and decoder can have 1 or more layers
- Training deep auto-encoders notoriously difficult
Link Between Contrastive Divergence and Auto-Encoder Reconstruction Error Gradient

• (Bengio & Delalleau 2009):
  • CD-2k estimates the log-likelihood gradient from 2k diminishing terms of an expansion that mimics the Gibbs steps
  • reconstruction error gradient looks only at the first step, i.e., is a kind of mean-field approximation of CD-0.5

\[
\frac{\partial \log P(x_1)}{\partial \theta} = \sum_{s=1}^{t-1} \left( E \left[ \frac{\partial \log P(x_s|h_s)}{\partial \theta} \bigg| x_1 \right] + E \left[ \frac{\partial \log P(h_s|x_{s+1})}{\partial \theta} \bigg| x_1 \right] \right)
\]

\[+ E \left[ \frac{\partial \log P(x_t)}{\partial \theta} \bigg| x_1 \right] \]
I finally understand what auto-encoders do!

• Try to carve holes in $||r(x)-x||^2$ or $-\log P(x \mid h(x))$ at examples

\[ \begin{align*}
\end{align*} \]

• Vector $r(x)-x$ points in direction of increasing prob., i.e. estimate score $= \frac{d \log p(x)}{d x}$: learn score vector field = local mean
• Generalize (valleys) in between above holes to form manifolds
  • $d r(x) / dx$ estimates the local covariance and is linked to the Hessian $d^2 \log p(x) / d x^2$
• A Markov Chain associated with AEs estimates the data-generating distribution (Bengio et al, arxiv 1305.663, 2013)
Auto-encoders can be stacked successfully (Bengio et al NIPS’2006) to form highly non-linear representations, which with fine-tuning overperformed purely supervised MLPs
(Auto-Encoder) Reconstruction Loss

• Discrete inputs: cross-entropy for binary inputs
  • $- \sum_i x_i \log r_i(x) + (1-x_i) \log(1-r_i(x))$ (with $0<r_i(x)<1$)
  or log-likelihood reconstruction criterion, e.g., for a multinomial (one-hot) input
  • $- \sum_i x_i \log r_i(x)$ (where $\sum_i r_i(x)=1$, summing over subset of inputs associated with this multinomial variable)

• In general: consider what are appropriate loss functions to predict each of the input variables,
  typically, reconstruction neg. log-likelihood $-\log P(x|h(x))$
Denoising Auto-Encoder
(Vincent et al 2008)

- Corrupt the input during training only
- Train to reconstruct the uncorrupted input

Hidden code (representation)

Corrupted input
Raw input
reconstruction

\( \text{KL(reconstruction | raw input)} \)

- Encoder & decoder: any parametrization
- As good or better than RBMs for unsupervised pre-training
Denoising Auto-Encoder

• Learns a vector field pointing towards higher probability direction (Alain & Bengio 2013)
  \[ r(x) - x \propto d\log p(x)/dx \]

• Some DAEs correspond to a kind of Gaussian RBM with regularized Score Matching (Vincent 2011)
  [equivalent when noise \( \to 0 \)]

• Compared to RBM:
  No partition function issue, + can measure training criterion

prior: examples concentrate near a lower dimensional “manifold”
Auto-Encoders Learn Salient Variations, like a non-linear PCA

- Minimizing reconstruction error forces to keep variations along manifold.
- Regularizer wants to throw away all variations.
- With both: keep ONLY sensitivity to variations ON the manifold.
Regularized Auto-Encoders Learn a Vector Field or a Markov Chain Transition Distribution

- (Bengio, Vincent & Courville, TPAMI 2013) review paper
- (Alain & Bengio ICLR 2013; Bengio et al, arxiv 2013)
**Contractive Auto-Encoders**

(Rifai, Vincent, Muller, Glorot, Bengio ICML 2011; Rifai, Mesnil, Vincent, Bengio, Dauphin, Glorot ECML 2011; Rifai, Dauphin, Vincent, Bengio, Muller NIPS 2011)

reconstruction\( (x) = g(h(x)) = \text{decoder}(\text{encoder}(x)) \)

Training criterion:
\[
J_{CAE}(\theta) = \sum_{x \in D_n} \lambda \sum_{i,j} \left( \frac{\partial h_j(x)}{\partial x_i} \right)^2 + L(x, \text{reconstruction}(x))
\]

wants contraction in all directions

cannot afford contraction in manifold directions

If \( h_j = \text{sigmoid}(b_j + W_j x) \)

\[
\left( \frac{\partial h_j(x)}{\partial x_i} \right)^2 = h_j^2 (1-h_j)^2 W_{ji}^2
\]
Contractive Auto-Encoders

(Rifai, Vincent, Muller, Glorot, Bengio ICML 2011; Rifai, Mesnil, Vincent, Bengio, Dauphin, Glorot ECML 2011; Rifai, Dauphin, Vincent, Bengio, Muller NIPS 2011)

Most hidden units saturate (near 0 or 1, derivative near 0): few responsive units represent the active subspace (local chart)

Each region/chart = subset of active hidden units
Neighboring region: one of the units becomes active/inactive

SHARED SET OF FILTERS ACROSS REGIONS, EACH USING A SUBSET
Jacobian’s spectrum is peaked = local low-dimensional representation / relevant factors

Inactive hidden unit = 0 singular value
Input Point

Tangents

\[ \begin{array}{c}
\text{0} \\
+ 0.5 \times \\
\end{array} \begin{array}{c}
\text{0} \\
\text{0} \\
\text{0} \\
\text{0} \\
\text{0} \\
\text{0} \\
\text{0} \\
\end{array} = \begin{array}{c}
\text{0} \\
\text{0} \\
\end{array} \]

MNIST
Input Point

MNIST Tangents
Distributed vs Local
(CIFAR-10 unsupervised)

Input Point  

Tangents

Local PCA (no sharing across regions)

Contractive Auto-Encoder
Denoising auto-encoders are also contractive!

• Taylor-expand Gaussian corruption noise in reconstruction error:

\[
E[\ell(x, r(x + \epsilon))] \approx E \left[ \left( x - \left( r(x) + \frac{\partial r(x)}{\partial x} \epsilon \right) \right)^T \left( x - \left( r(x) + \frac{\partial r(x)}{\partial x} \epsilon \right) \right) \right]
\]

\[
= E[\|x - r(x)\|^2] + \sigma^2 E \left[ \left\| \frac{\partial r(x)}{\partial x} \right\|_F^2 \right]
\]

• Yields a contractive penalty in the reconstruction function (instead of encoder) proportional to amount of corruption noise
Learned Tangent Prop: the Manifold Tangent Classifier

Makes classifier $f(x)$ insensitive to variations on manifold at $x$

Tangent plane characterized by $dh(x)/dx$

(Rifai et al NIPS’2012)
Manifold Tangent Classifier Results

- Leading singular vectors on MNIST, CIFAR-10, RCV1:

- Knowledge-free MNIST: 0.81% error

<table>
<thead>
<tr>
<th></th>
<th>K-NN</th>
<th>NN</th>
<th>SVM</th>
<th>DBN</th>
<th>CAE</th>
<th>DBM</th>
<th>CNN</th>
<th>MTC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3.09%</td>
<td>1.60%</td>
<td>1.40%</td>
<td>1.17%</td>
<td>1.04%</td>
<td>0.95%</td>
<td>0.95%</td>
<td>0.81%</td>
</tr>
</tbody>
</table>

- Semi-sup.

<table>
<thead>
<tr>
<th></th>
<th>NN</th>
<th>SVM</th>
<th>CNN</th>
<th>TSVM</th>
<th>DBN-rNCA</th>
<th>EmbedNN</th>
<th>CAE</th>
<th>MTC</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>25.81</td>
<td>23.44</td>
<td>22.98</td>
<td>16.81</td>
<td>-</td>
<td>16.86</td>
<td>13.47</td>
<td>12.03</td>
</tr>
<tr>
<td>600</td>
<td>11.44</td>
<td>8.85</td>
<td>7.68</td>
<td>6.16</td>
<td>8.7</td>
<td>5.97</td>
<td>6.3</td>
<td>5.13</td>
</tr>
<tr>
<td>1000</td>
<td>10.7</td>
<td>7.77</td>
<td>6.45</td>
<td>5.38</td>
<td>-</td>
<td>5.73</td>
<td>4.77</td>
<td>3.64</td>
</tr>
<tr>
<td>3000</td>
<td>6.04</td>
<td>4.21</td>
<td>3.35</td>
<td>3.45</td>
<td>3.3</td>
<td>3.59</td>
<td>3.22</td>
<td>2.57</td>
</tr>
</tbody>
</table>

- Forest (500k examples)

<table>
<thead>
<tr>
<th></th>
<th>SVM</th>
<th>Distributed SVM</th>
<th>MTC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>4.11%</td>
<td>3.46%</td>
<td>3.13%</td>
</tr>
</tbody>
</table>
Predictive Sparse Decomposition

- Introduce an auxiliary variable in auto-encoder
- Approximate the inference of sparse coding by a parametric encoder:

  Predictive Sparse Decomposition
  (Kavukcuoglu et al 2008)

- Very successful applications in machine vision with convolutional architectures
Predictive Sparse Decomposition

- Stacked to form deep architectures
- Alternating convolution, rectification, pooling
- Tiling: no sharing across overlapping filters
- Group sparsity penalty yields topographic maps
Deep Variants
Level-Local Learning is Important

- Initializing each layer of an unsupervised deep Boltzmann machine helps a lot
- Initializing each layer of a supervised neural network as an RBM, auto-encoder, denoising auto-encoder, etc can help a lot
- Helps most the layers further away from the target
- Not just an effect of the unsupervised prior
- Jointly training all the levels of a deep architecture is difficult because of the increased non-linearity / non-smoothness
- Initializing using a level-local learning algorithm is a useful trick
- Providing intermediate-level targets can help tremendously (Gulcehre & Bengio ICLR 2013)
Stack of RBMs / AEs → Deep MLP

- Encoder or $P(h|v)$ becomes MLP layer
Stack of RBMs / AEs \(\rightarrow\) Deep Auto-Encoder

(Hinton & Salakhutdinov 2006)

- Stack encoders / \(P(h|x)\) into deep encoder
- Stack decoders / \(P(x|h)\) into deep decoder
Stack of RBMs / AEs → Deep Recurrent Auto-Encoder

(Savard 2011) (Bengio & Laufer, arxiv 2013)

- Each hidden layer receives input from below and above
- Deterministic (mean-field) recurrent computation (Savard 2011)
- Stochastic (injecting noise) recurrent computation: Deep Generative Stochastic Networks (GSNs)
  (Bengio & Laufer arxiv 2013)
Stack of RBMs → Deep Belief Net

(Hinton et al 2006)

- Stack lower levels RBMs’ $P(x|h)$ along with top-level RBM
- $P(x, h_1, h_2, h_3) = P(h_2, h_3) P(h_1 | h_2) P(x | h_1)$
- Sample: Gibbs on top RBM, propagate down
Halve the RBM weights because each layer now has inputs from below and from above

Positive phase: (mean-field) variational inference = recurrent AE

Negative phase: Gibbs sampling (stochastic units)

train by SML/PCD
Stack of Auto-Encoders \rightarrow Deep Generative Auto-Encoder

(Rifai et al ICML 2012)

- MCMC on top-level auto-encoder
  - $h_{t+1} = \text{encode}(\text{decode}(h_t)) + \sigma \text{ noise}$
    where noise is $\text{Normal}(0, \frac{d}{dh} \text{encode}(\text{decode}(h_t)))$
- Then deterministically propagate down with decoders
Generative Stochastic Networks (GSN)
(Bengio, Yao, Alain & Vincent, arxiv 2013; Bengio & Laufer, arxiv 2013)

- Recurrent parametrized stochastic computational graph that defines a transition operator for a Markov chain whose asymptotic distribution is implicitly estimated by the model
- Noise injected in input and hidden layers
- Trained to max. reconstruction prob. of example at each step
- Example structure inspired from the DBM Gibbs chain:
Denoising Auto-Encoder Markov Chain

- $\mathcal{P}(X)$: true data-generating distribution
- $\mathcal{C}(\tilde{X}|X)$: corruption process
- $P_{\theta_n}(X|\tilde{X})$: denoising auto-encoder trained with $n$ examples $X, \tilde{X}$ from $\mathcal{C}(\tilde{X}|X)\mathcal{P}(X)$, probabilistically "inverts" corruption
- $T_n$: Markov chain over $X$ alternating $\tilde{X} \sim \mathcal{C}(\tilde{X}|X)$, $X \sim P_{\theta_n}(X|\tilde{X})$
Previous Theoretical Results on Probabilistic Interpretation of Auto-Encoders

(Vincent 2011, Alain & Bengio 2013)

- Continuous $X$
- Gaussian corruption
- Noise $\sigma \rightarrow 0$
- Squared reconstruction error $||r(X+\text{noise})-X||^2$

$(r(X)-X)/\sigma^2$ estimates the score $\frac{d \log p(X)}{dX}$
New Theoretical Results
(Bengio et al NIPS 2013)

- Denoising AE are consistent estimators of the data-generating distribution through their Markov chain, so long as they consistently estimate the conditional denoising distribution and the Markov chain converges.

\[
\text{Making } P_{\theta_n}(X|\tilde{X}) \text{ match } \mathcal{P}(X|\tilde{X}) \text{ makes } \pi_n(X) \text{ match } \mathcal{P}(X)
\]
Shallow Model: Generalizing the Denoising Auto-Encoder Probabilistic Interpretation

- Classical denoising auto-encoder architecture, single hidden layer with noise only injected in input
- Factored Bernouilli reconstruction prob. distr.
- $\tilde{X} = f_{\theta_1}(X, Z) = \text{parameter-less, salt-and-pepper noise on top of } X$
- Generalizes (Alain & Bengio ICLR 2013): not just continuous r.v., any training criterion (as log-likelihood), not just Gaussian but any corruption (no need to be tiny to correctly estimate distribution).
Generative Stochastic Networks

- Generalizes the denoising auto-encoder training scheme
  - Introduce latent variables in the Markov chain (over $X, H$)
  - Instead of a fixed corruption process, have a deterministic function with parameters $\theta_1$ and a noise source $Z$ as input

\[ H_{t+1} = f_{\theta_1}(X_t, Z_t, H_t) \]

\[ H_{t+1} \sim P_{\theta_1}(H|H_t, X_t) \]
\[ X_{t+1} \sim P_{\theta_2}(X|H_{t+1}) \]
Experiments: Shallow vs Deep

- Shallow (DAE), no recurrent path at higher levels, state=X only

- Deep GSN:
GSN Experiments: Consecutive Samples

Filling-in the LHS
A Proper Generative Model for Dependency Networks, MP-DBMs, and efficient deep NADE sampling

• Dependency nets (Heckerman et al 2000) estimate $P_{\theta_i}(X_i \mid X_{-i})$ not guaranteed to be conditionals of a unique joint
• Heckerman et al’s sampling iterates over $i$: not ergodic?
• Randomly choosing $i$: proper GSN
• Defines a unique joint distribution = stationary distr. of chain (which averages out over resampling orders)
• Generalized to estimators of $P(\text{subset}(X) \mid X \setminus \text{subset}(X))$ and justify efficient sampling schemes for MP-DBMs and deep NADE.
MP-DBM Results

• Single model of (X,Y) vs multiple stages of training DBM + fine-tuning
• SOTA on permutation-invariant MNIST (at time of submission):
  • 0.88% error
• Salakhutdinov & Hinton’s DBM: 0.95%
• NORB: 10.6% (vs 10.8% with S&H’s DBM)
• DBM (Gibbs) samples of trained MP-DBM are ugly, while GSN sampling works because it better corresponds to the training criterion:
Reparameterizing latent variables

• Insight from (Bengio et al 2013, arxiv 1306.1091 & 1308.3432) papers on GSNs and stochastic neurons:
  • Sampling from continuous latent variables (given some ancestors) can be rewritten as a deterministic function of other variables and of independent noise sources: \( h = f(x; \eta) \)
  • This enables rewriting the gradient log-likelihood as back-prop, averaged over samples of the noise sources

\[
P(y|x) = \int_h P(y|h, x)P(h|x)dh = \int_\eta P(y|f(x; \eta), x)P(\eta)d\eta
\]

\[
\frac{\partial P(y|x)}{\partial \theta} = \int_\eta \frac{\partial P(y|f(x; \eta), x)}{\partial \theta} P(\eta)d\eta
\]

• A deeper formal analysis of this approach:
  • Kingma & Welling 2014, arxiv 1402.0480; see also Wierstra et al 2014, arxiv 1401.4082.
Learned Approximate Inference

1. Construct a computational graph corresponding to inference
   - Variational mean-field (Goodfellow et al, ICLR 2013)
   - MAP (Kavukcuoglu et al 2008, Gregor & LeCun ICML 2010)

2. Optimize parameters wrt criterion of interest, possibly decoupling from the generative model’s parameters

Learning can compensate for the inadequacy of approximate inference, taking advantage of specifics of the data distribution
Auto-Encoding Variational Bayes
(Kingma & Welling, ICLR 2014; DeepMind 2014)

- Revisiting the wake-sleep algorithm
- Generative model = deep net with injected noise: decoder
- Learned approximate inference = deep net with injected noise: encoder
- Latent variables are continuous, allowing to back-prop through (trick from GSN paper, Bengio et al ICML 2014) and train encoder & decoder jointly
Applications
AI Tasks

- Perception
  - Vision
  - Speech
  - Multiple modalities
- Natural language understanding
- Reinforcement learning & control

- COMPLEX HIGHLY-STRUCTURED DISTRIBUTION
- LOTS OF DATA (maybe mostly unlabeled)
2012: Industrial-scale success in speech recognition

- Google uses DL in their android speech recognizer (both server-side and on some phones with enough memory)
- Microsoft uses DL in their speech recognizer
- Error reductions on the order of 30%, a major progress
The dramatic impact of Deep Learning on Speech Recognition (according to Microsoft)

<table>
<thead>
<tr>
<th>Year</th>
<th>Word Error Rate on Switchboard</th>
</tr>
</thead>
<tbody>
<tr>
<td>1990</td>
<td>179%</td>
</tr>
<tr>
<td>2000</td>
<td>10%</td>
</tr>
<tr>
<td>2010</td>
<td>2%</td>
</tr>
</tbody>
</table>

Using DL
Deep Networks for Speech Recognition: results from Google, IBM, Microsoft

<table>
<thead>
<tr>
<th>task</th>
<th>Hours of training data</th>
<th>Deep net+HMM</th>
<th>GMM+HMM same data</th>
<th>GMM+HMM more data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Switchboard</td>
<td>309</td>
<td>16.1</td>
<td>23.6</td>
<td>17.1 (2k hours)</td>
</tr>
<tr>
<td>English Broadcast news</td>
<td>50</td>
<td>17.5</td>
<td>18.8</td>
<td></td>
</tr>
<tr>
<td>Bing voice search</td>
<td>24</td>
<td>30.4</td>
<td>36.2</td>
<td></td>
</tr>
<tr>
<td>Google voice input</td>
<td>5870</td>
<td>12.3</td>
<td></td>
<td>16.0 (lots more)</td>
</tr>
<tr>
<td>Youtube</td>
<td>1400</td>
<td>47.6</td>
<td>52.3</td>
<td></td>
</tr>
</tbody>
</table>

(numbers taken from Geoff Hinton’s June 22, 2012 Google talk)
Some Applications of DL

- **Language Modeling** (Speech Recognition, Machine Translation)
- **Acoustic Modeling** (*speech recognition*, music modeling)
- **NLP syntactic/semantic tagging** (Part-Of-Speech, chunking, Named Entity Recognition, Semantic Role Labeling, Parsing)
- **NLP applications**: sentiment analysis, paraphrasing, question-answering, Word-Sense Disambiguation
- **Personalization/recommendation/fraud/ads**
- **Molecular properties**: QSAR, quantum calculations
Industrial-scale success in object recognition

- Krizhevsky, Sutskever & Hinton NIPS 2012

<table>
<thead>
<tr>
<th></th>
<th>1st choice</th>
<th>Top-5</th>
</tr>
</thead>
<tbody>
<tr>
<td>2nd best</td>
<td></td>
<td>27% err</td>
</tr>
<tr>
<td>Previous SOTA</td>
<td>45% err</td>
<td>26% err</td>
</tr>
<tr>
<td>Krizhevsky et al</td>
<td>37% err</td>
<td>15% err</td>
</tr>
</tbody>
</table>

- Google incorporates DL in Google+ photo search, “A step across the semantic gap” (Google Research blog, June 12, 2013)

- Baidu now offers similar services
Montreal Deep Nets Win Emotion Recognition in the Wild Challenge

Predict emotional expression from video (using images + audio)

<table>
<thead>
<tr>
<th>Team Name</th>
<th>Classification accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Audio baseline</td>
<td>22.4 %</td>
</tr>
<tr>
<td>Video baseline</td>
<td>22.7 %</td>
</tr>
<tr>
<td>Fusion</td>
<td>27.5 %</td>
</tr>
<tr>
<td>Nottingham</td>
<td>24.7 %</td>
</tr>
<tr>
<td>Oulu</td>
<td>21.5 %</td>
</tr>
<tr>
<td>KIT</td>
<td>29.8 %</td>
</tr>
<tr>
<td>UCSD</td>
<td>37.1 %</td>
</tr>
<tr>
<td>ICT@CAS</td>
<td>35.9 %</td>
</tr>
<tr>
<td>York</td>
<td>27.6 %</td>
</tr>
<tr>
<td>LNMIIT</td>
<td>20.5 %</td>
</tr>
<tr>
<td>Montreal</td>
<td>41.0 %</td>
</tr>
<tr>
<td>Ulm</td>
<td>27.2 %</td>
</tr>
</tbody>
</table>

Dec. 9, 2013
More Successful Applications

• Microsoft uses DL for speech rec. service (audio video indexing), based on Hinton/Toronto’s DBNs (Mohamed et al 2012)
• Google uses DL in its Google Goggles service, using Ng/Stanford DL systems, and in its Google+ photo search service, using deep convolutional nets
• NYT talks about these: http://www.nytimes.com/2012/06/26/technology/in-a-big-network-of-computers-evidence-of-machine-learning.html?_r=1
• Substantially beating SOTA in language modeling (perplexity from 140 to 102 on Broadcast News) for speech recognition (WSJ WER from 16.9% to 14.4%) (Mikolov et al 2011) and translation (+1.8 BLEU) (Schwenk 2012)
• SENNA: Unsup. pre-training + multi-task DL reaches SOTA on POS, NER, SRL, chunking, parsing, with >10x better speed & memory (Collobert et al 2011)
• Recursive nets surpass SOTA in paraphrasing (Socher et al 2011)
• Denoising AEs substantially beat SOTA in sentiment analysis (Glorot et al 2011)
• Contractive AEs SOTA in knowledge-free MNIST (.8% err) (Rifai et al NIPS 2011)
• Le Cun/NYU’s stacked PSDs most accurate & fastest in pedestrian detection and DL in top 2 winning entries of German road sign recognition competition
Already Many NLP Applications of DL

- Language Modeling (Speech Recognition, Machine Translation)
- Acoustic Modeling
- Part-Of-Speech Tagging
- Chunking
- Named Entity Recognition
- Semantic Role Labeling
- Parsing
- Sentiment Analysis
- Paraphrasing
- Question-Answering
- Word-Sense Disambiguation
Neural Language Model

- **Bengio et al NIPS’2000 and JMLR 2003** “A Neural Probabilistic Language Model”
  - Each word represented by a distributed continuous-valued code vector = embedding
  - Generalizes to sequences of words that are semantically similar to training sequences
Neural word embeddings - visualization
Analogical Representations for Free (Mikolov et al, ICLR 2013)

• Semantic relations appear as linear relationships in the space of learned representations
• King – Queen ≈ Man – Woman
• Paris – France + Italy ≈ Rome
Practical Considerations
Deep Learning Tricks of the Trade

• Y. Bengio (2013), “Practical Recommendations for Gradient-Based Training of Deep Architectures”
  • Unsupervised pre-training
  • Stochastic gradient descent and setting learning rates
  • Main hyper-parameters
    • Learning rate schedule
    • Early stopping
    • Minibatches
    • Parameter initialization
    • Number of hidden units
    • L1 and L2 weight decay
    • Sparsity regularization
  • Debugging
  • How to efficiently search for hyper-parameter configurations
Stochastic Gradient Descent (SGD)

- Gradient descent uses total gradient over all examples per update, SGD updates after only 1 or few examples:

\[
\theta^{(t)} \leftarrow \theta^{(t-1)} - \epsilon_t \frac{\partial L(z_t, \theta)}{\partial \theta}
\]

- L = loss function, \( z_t \) = current example, \( \theta \) = parameter vector, and \( \epsilon_t \) = learning rate.

- Ordinary gradient descent is a batch method, very slow, should never be used. 2nd order batch method are being explored as an alternative but SGD with selected learning schedule remains the method to beat.


Learning Rates

- Simplest recipe: keep it fixed and use the same for all parameters.
- Collobert scales them by the inverse of square root of the fan-in of each neuron.
- Better results can generally be obtained by allowing learning rates to decrease, typically in $O(1/t)$ because of theoretical convergence guarantees, e.g.,

\[
\epsilon_t = \frac{\epsilon_0 \tau}{\max(t, \tau)}
\]

with hyper-parameters $\epsilon_0$ and $\tau$.
- New papers on adaptive learning rates procedures (Schaul 2012, 2013), Adagrad (Duchi et al 2011), ADADELTA (Zeiler 2012)
Early Stopping

• Beautiful **FREE LUNCH** (no need to launch many different training runs for each value of hyper-parameter for #iterations)

• Monitor validation error during training (after visiting # of training examples = a multiple of validation set size)

• Keep track of parameters with best validation error and report them at the end

• If error does not improve enough (with some patience), stop.
Long-Term Dependencies

• In very deep networks such as recurrent networks (or possibly recursive ones), the gradient is a product of Jacobian matrices, each associated with a step in the forward computation. This can become very small or very large quickly [Bengio et al 1994], and the locality assumption of gradient descent breaks down.

\[
L = L(s_T(s_{T-1}(\ldots s_{t+1}(s_t, \ldots))))
\]

\[
\frac{\partial L}{\partial s_t} = \frac{\partial L}{\partial s_T} \frac{\partial s_T}{\partial s_{T-1}} \ldots \frac{\partial s_{t+1}}{\partial s_t}
\]

• Two kinds of problems:
  • sing. values of Jacobians > 1 → gradients explode
  • or sing. values < 1 → gradients shrink & vanish
The Optimization Challenge in Deep / Recurrent Nets

- Higher-level abstractions require highly non-linear transformations to be learned
- Sharp non-linearities are difficult to learn by gradient
- Composition of many non-linearities = sharp non-linearity
- Exploding or vanishing gradients
RNN Tricks
(Pascanu, Mikolov, Bengio, ICML 2013; Bengio, Boulanger & Pascanu, ICASSP 2013)

- Clipping gradients (avoid exploding gradients)
- Leaky integration (propagate long-term dependencies)
- Momentum (cheap 2\textsuperscript{nd} order)
- Initialization (start in right ballpark avoids exploding/vanishing)
- Sparse Gradients (symmetry breaking)
- Gradient propagation regularizer (avoid vanishing gradient)
- LSTM self-loops (avoid vanishing gradient)
Long-Term Dependencies and Clipping Trick

Trick first introduced by Mikolov is to clip gradients to a maximum NORM value.

Makes a big difference in Recurrent Nets (Pascanu et al ICML 2013)
Allows SGD to compete with HF optimization on difficult long-term dependencies tasks. Helped to beat SOTA in text compression, language modeling, speech recognition.
Combining clipping to avoid gradient explosion and Jacobian regularizer to avoid gradient vanishing

- (Pascanu, Mikolov & Bengio, ICML 2013)
Normalized Initialization to Achieve Unity-Like Jacobian

Assuming $f'(act=0) = 1$

To keep information flowing in both direction we would like to have the following properties.

- **Forward-propagation:**

  $$\forall (i, i'), Var[z^i] = Var[z^{i'}] \Leftrightarrow \forall i, n_i Var[W^i] = 1$$

- **Back-propagation:**

  $$\forall (i, i'), Var[\frac{\partial Cost}{\partial s^i}] = Var[\frac{\partial Cost}{\partial s^{i'}}] \Leftrightarrow \forall i, n_{i+1} Var[W^{i}] = 1$$

Possible compromise:

$$\forall i, Var[W^i] = \frac{2}{n_i + n_{i+1}} \quad (4)$$

This gives rise to proposed **normalized initialization** procedure:

$$W^i \sim U \left[ -\frac{\sqrt{6}}{\sqrt{n_i + n_{i+1}}}, \frac{\sqrt{6}}{\sqrt{n_i + n_{i+1}}} \right] \quad (5)$$
Normalized Initialization with Variance-Preserving Jacobians

Unsupervised pre-training: Automatically variance-preserving!
Orthogonal Initialization Works Even Better

- Auto-encoder pre-training tends to yield orthogonal W
- (Saxe, McClelland & Ganguli ICLR 2014) showed that very deep nets initialized with random orthogonal weights are much easier to train
- All singular values = 1
Handling Large Output Spaces

- Auto-encoders and RBMs reconstruct the input, which is sparse and high-dimensional; Language models have a huge output space (1 unit per word).

- (Dauphin et al, ICML 2011) Reconstruct the non-zeros in the input, and reconstruct as many randomly chosen zeros, + importance weights

- (Collobert & Weston, ICML 2008) sample a ranking loss

- Decompose output probabilities hierarchically (Morin & Bengio 2005; Blitzer et al 2005; Mnih & Hinton 2007,2009; Mikolov et al 2011)
Automatic Differentiation

- Makes it easier to quickly and safely try new models.
- Theano Library (python) does it symbolically. Other neural network packages (Torch, Lush) can compute gradients for any given run-time value.

(Bergstra et al SciPy’2010)
Random Sampling of Hyperparameters
(Bergstra & Bengio 2012)

- Common approach: manual + grid search
- Grid search over hyperparameters: simple & wasteful
- Random search: simple & efficient
  - Independently sample each HP, e.g. l.rate~exp(U[log(.1),log(.0001)])
  - Each training trial is iid
  - If a HP is irrelevant grid search is wasteful
  - More convenient: ok to early-stop, continue further, etc.
Sequential Model-Based Optimization of Hyper-Parameters

- (Hutter et al JAIR 2009; Bergstra et al NIPS 2011; Thornton et al arXiv 2012; Snoek et al NIPS 2012)
- Iterate
- Estimate $P(\text{valid. err} \mid \text{hyper-params config } x, D)$
- choose optimistic $x$, e.g. $\max_x P(\text{valid. err} < \text{current min. err} \mid x)$
- train with config $x$, observe valid. err. $v$, $D \leftarrow D \cup \{(x,v)\}$
Part 4

Challenges & Questions
Why is Unsupervised Pre-Training Sometimes Working So Well?

• Regularization hypothesis:
  • Unsupervised component forces model close to $P(x)$
  • Representations good for $P(x)$ are good for $P(y|x)$

• Optimization hypothesis:
  • Unsupervised initialization near better local minimum of $P(y|x)$
  • Can reach lower local minimum otherwise not achievable by random initialization
  • Easier to train each layer using a layer-local criterion

(Erhan et al JMLR 2010)
Learning Trajectories in Function Space

- Each point a model in function space
- Color = epoch
- Top: trajectories w/o pre-training
- Each trajectory converges in different local min.
- No overlap of regions with and w/o pre-training
Learning Trajectories in Function Space

- Each trajectory converges in different local min.
- With ISOMAP, try to preserve geometry: pretrained nets converge near each other (less variance)
- Good answers = worse than a needle in a haystack (learning dynamics)
Deep Learning Challenges
(Bengio, arxiv 1305.0445 Deep learning of representations: looking forward)

- Computational Scaling
- Optimization & Underfitting
- Intractable Marginalization, Approximate Inference & Sampling
- Disentangling Factors of Variation
- Reasoning & One-Shot Learning of Facts
Deep Learning Challenges
(Bengio, arxiv 1305.0445 Deep learning of representations: looking forward)

• Computational Scaling
• Optimization & Underfitting
• Intractable Marginalization, Approximate Inference & Sampling
• Disentangling Factors of Variation
• Reasoning & One-Shot Learning of Facts
Challenge: Computational Scaling

• Recent breakthroughs in speech, object recognition and NLP hinged on faster computing, GPUs, and large datasets

• A 100-fold speedup is possible without waiting another 10 yrs?
  • Challenge of distributed training
  • Challenge of conditional computation
Conditional Computation: only visit a small fraction of parameters / example

- Deep nets vs decision trees
- Hard mixtures of experts (Collobert, Bengio & Bengio 2002)
- Conditional computation for deep nets: sparse distributed gaters selecting combinatorial subsets of a deep net
- Challenges:
  - Credit assignment for hard decisions
  - Gated architectures exploration
- Noisy rectifiers work well
Distributed Training

- Minibatches
- Large minibatches + 2\textsuperscript{nd} order & natural gradient methods
  - Bottleneck: sharing weights/updates among nodes, to avoid node-models to move too far from each other
- Ideas forward:
  - Low-resolution sharing only where needed
  - Specialized conditional computation (each computer specializes in updates to some cluster of gated experts, and prefers examples which trigger these experts)
Deep Learning Challenges
(Bengio, arxiv 1305.0445 Deep learning of representations: looking forward)

- Computational Scaling
- Optimization & Underfitting
- Intractable Marginalization, Approximate Inference & Sampling
- Disentangling Factors of Variation
- Reasoning & One-Shot Learning of Facts
Optimization & Underfitting

- On large datasets, major obstacle is underfitting
- **Marginal utility** of wider MLPs decreases quickly below memorization baseline

- Current limitations: local minima, ill-conditioning or else?
Guided Training, Intermediate Concepts

• In (Gulcehre & Bengio ICLR’2013) we set up a task that seems almost impossible to learn by shallow nets, deep nets, SVMs, trees, boosting etc

• Breaking the problem in two sub-problems and pre-training each module separately, then fine-tuning, nails it

• *Need prior knowledge to decompose the task*

• *Guided pre-training* allows to find much better solutions, escape effective local minima
Deep Learning Challenges

(Bengio, arxiv 1305.0445 Deep learning of representations: looking forward)

- Computational Scaling
- Optimization & Underfitting
- Intractable Marginalization, Approximate Inference & Sampling
- Disentangling Factors of Variation
- Reasoning & One-Shot Learning of Facts
Why Unsupervised Learning?

• Recent progress mostly in supervised DL
• ∃ real challenges for unsupervised DL
• Potential benefits:
  • Exploit tons of unlabeled data
  • Answer new questions about the variables observed
  • Regularizer – transfer learning – domain adaptation
  • Easier optimization (local training signal)
  • Structured outputs
Basic Challenge with Probabilistic Models: marginalization

- Joint and marginal likelihoods involve intractable sums over configurations of random variables (inputs x, latent h, outputs y) e.g.

\[
P(x) = \sum_h P(x, h)
\]

\[
P(x, h) = e^{-\text{energy}(x, h)}/Z
\]

\[
Z = \sum_{x, h} e^{-\text{energy}(x, h)}
\]

- MCMC methods can be used for these sums, by sampling from a chain of x’s (or of (x, h) pairs) approximately from \( P(x, h) \)
Two Fundamental Problems with Probabilistic Models with Many Random Variables

1. MCMC mixing between modes (manifold hypothesis)
2. Many non-negligible modes (both in posterior & joint distributions)
For gradient & inference: More difficult to mix with better trained models

- Early during training, density smeared out, mode bumps overlap
- Later on, hard to cross empty voids between modes

Training updates

vicious circle

Mixing

Are we doomed if we rely on MCMC during training? Will we be able to train really large & complex models?
Poor Mixing: Depth to the Rescue

(Bengio et al ICML 2013)

- Sampling from DBNs and stacked Contractive Auto-Encoders:
  1. MCMC sampling from top layer model
  2. Propagate top-level representations to input-level repr.
- Deeper nets visit more modes (classes) faster
Space-Filling in Representation-Space

- Deeper representations $\rightarrow$ abstractions $\rightarrow$ disentangling
- Manifolds are expanded and flattened

- Linear interpolation at layer 1
- Linear interpolation at layer 2
- Linear interpolation in pixel space
Many Modes Challenge: Instead of learning $P(x)$ directly, learn Markov chain operator $P(x_t | x_{t-1})$

- $P(x)$ may have many modes, making the normalization constant intractable, and MCMC approximations poor
- Partition fn of $P(x_t | x_{t-1})$ much simpler because most of the time a local move, might even be well approximated by unimodal
Deep Learning Challenges
(Bengio, arxiv 1305.0445 Deep learning of representations: looking forward)

- Computational Scaling
- Optimization & Underfitting
- Intractable Marginalization, Approximate Inference & Sampling
- Disentangling Factors of Variation
- Reasoning & One-Shot Learning of Facts
Disentangling the Underlying Factors

• How could a learner disentangle the unknown underlying factors of variation?
  • Statistical structure present in the data
  • Hints = priors

• Good disentangling $\rightarrow$
  avoid the curse of dimensionality
Broad Priors as Hints to Disentangle the Factors of Variation

- **Multiple factors**: distributed representations
- Multiple levels of abstraction: *depth*
- *Semi-supervised* learning: Y is one of the factors explaining X
- *Multi-task* learning: different tasks share some factors
- *Manifold* hypothesis: probability mass concentration
- Natural *clustering*: class = manifold, well-separated manifolds
- Temporal and spatial *coherence*
- *Sparsity*: most factors irrelevant for particular X
- *Simplicity* of factor dependencies (in the right representation)
Learning Multiple Levels of Abstraction

- The big payoff of deep learning is to allow learning higher levels of abstraction.

- Higher-level abstractions disentangle the factors of variation, which allows much easier generalization and transfer.
Conclusions

• Deep Learning has matured
  • Int. Conf. on Learning Representation 2013 a huge success!
• Industrial applications (Google, Microsoft, Baidu, Facebook, ...)

• Room for improvement:
  • Scaling computation
  • Optimization
  • Bypass intractable marginalizations
  • More disentangled abstractions
  • Reason from incrementally added facts
If Time Permits...
Issue: **underfitting** due to combinatorially many poor **effective** local minima

where the optimizer gets stuck

**Culture vs Effective Local Minima**

Bengio 2013 (also arXiv 2012)
Parallelized exploration in brain space

- Each brain explores a potential solution
- Instead of exchanging synaptic configurations, exchange ideas through language
Memes

Genetic Algorithms
Population of candidate solutions
Recombination mechanism

Evolution of ideas
Brains
Culture and language
Hypothesis 1

• When the brain of a single biological agent learns, it performs an approximate optimization with respect to some endogenous objective.

Hypothesis 2

• When the brain of a single biological agent learns, it relies on approximate local descent in order to gradually improve itself.
Theoretical and experimental results on deep learning suggest:

**Hypothesis 3**

- Higher-level abstractions in brains are represented by deeper computations (going through more areas or more computational steps in sequence over the same areas).

**Hypothesis 4**

- Learning of a single human learner is limited by *effective* local minima.
Hypothesis 5

• A single human learner is unlikely to discover high-level abstractions by chance because these are represented by a deep sub-network in the brain.

Hypothesis 6

• A human brain can learn high-level abstractions if guided by the signals produced by other humans, which act as hints or indirect supervision for these high-level abstractions.

Supporting evidence: (Gulcehre & Bengio ICLR 2013)
How is one brain transferring abstractions to another brain?
How do we escape local minima?

• linguistic inputs = extra examples, summarize knowledge
• criterion landscape easier to optimize (e.g. curriculum learning)
• turn difficult unsupervised learning into easy supervised learning of intermediate abstractions
Hypothesis 7

• Language and meme recombination provide an efficient evolutionary operator, allowing rapid search in the space of memes, that helps humans build up better high-level internal representations of their world.

More than random search: potential exponential speed-up by divide-and-conquer combinatorial advantage: can combine solutions to independently solved sub-problems.
From where do new ideas emerge?

• Seconds: *inference* (novel explanations for current $x$)

• Minutes, hours: *learning* (local descent, like current DL)

• Years, centuries: *cultural evolution* (global optimization, recombination of ideas from other humans)
Related Tutorials

• Deep Learning tutorials (python): [http://deeplearning.net/tutorials](http://deeplearning.net/tutorials)
• Stanford deep learning tutorials with simple programming assignments and reading list [http://deeplearning.stanford.edu/wiki/](http://deeplearning.stanford.edu/wiki/)
• More reading: Paper references in separate pdf, on my web page
Software

• Theano (Python CPU/GPU) mathematical and deep learning library [http://deeplearning.net/software/theano](http://deeplearning.net/software/theano)
  • Can do automatic, symbolic differentiation
• Senna: POS, Chunking, NER, SRL
  • by Collobert et al. [http://ronan.collobert.com/senna/](http://ronan.collobert.com/senna/)
  • State-of-the-art performance on many tasks
  • 3500 lines of C, extremely fast and using very little memory
• Torch ML Library (C++ + Lua) [http://www.torch.ch/](http://www.torch.ch/)
• Recursive Neural Net and RAE models for paraphrase detection, sentiment analysis, relation classification [www.socher.org](http://www.socher.org)
Software: what’s next

• Off-the-shelf SVM packages are useful to researchers from a wide variety of fields (no need to understand RKHS).

• To make deep learning more accessible: release off-the-shelf learning packages that handle hyper-parameter optimization, exploiting multi-core or cluster at disposal of user.
  • Spearmint (Snoek)
  • HyperOpt (Bergstra)