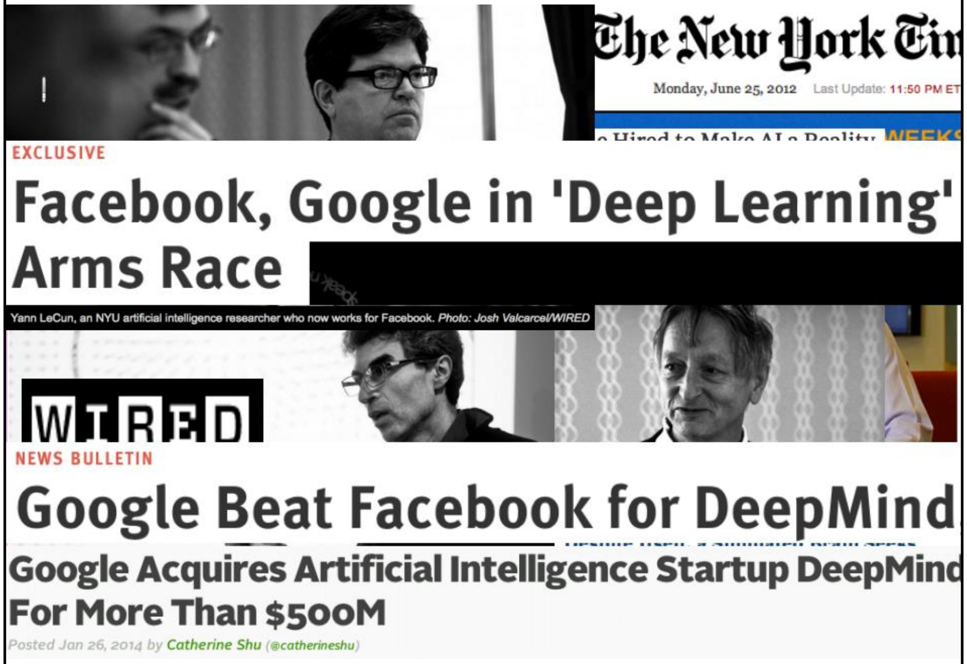
Deep Learning

Yoonjung Choi
yjchoi@cs.pitt.edu



The New York Times
Monday, June 25, 2012 Last Update: 11:50 PM ET

EXCLUSIVE
Facebook, Google in 'Deep Learning' Arms Race

Yann LeCun, an NYU artificial intelligence researcher who now works for Facebook. Photo: Josh Valcarcel/WIRED

WIRED
NEWS BULLETIN
Google Beat Facebook for DeepMind

Google Acquires Artificial Intelligence Startup DeepMind For More Than $500M
Posted Jan 26, 2014 by Catherine Shu (@catherineshu)

Contents

Contents

Introduction

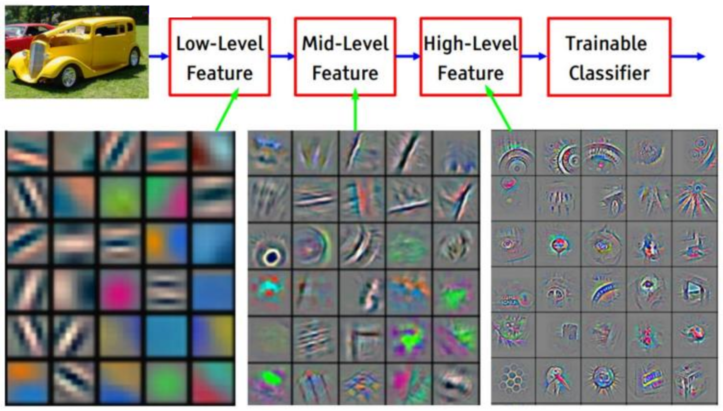
- Learning:
  - Mathematical and computational principles allowing one to learn from examples in order to acquire knowledge

Introduction

- Learning:
  - Mathematical and computational principles allowing one to learn from examples in order to acquire knowledge
- Deep learning
  - Machine learning algorithms inspired by brains, based on learning multiple levels of representation / abstraction

Introduction

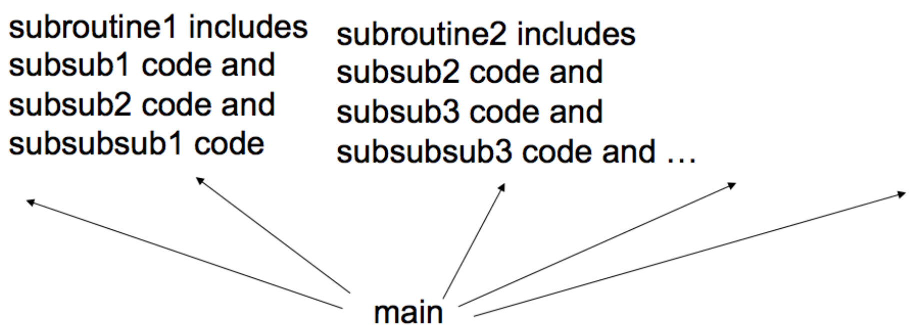
- It's deep if it has more than one state of non-linear feature transformation.



Learning Multiple Levels

- There is theoretical and empirical evidence in favor of multiple levels of representation.
- Biologically inspired learning
  - Brain has a deep architecture.
  - Cortex seems to have a generic learning algorithm.
  - Humans first learn simpler concepts and compose them.

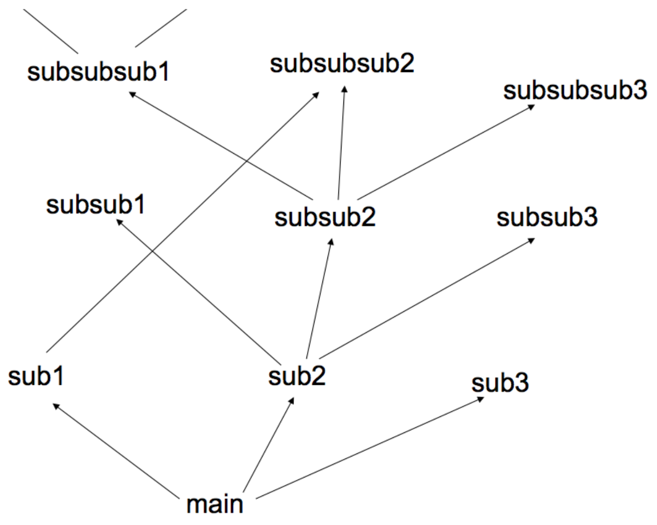
“Shallow” Computer Program



subroutine1 includes subsub1 code and subsub2 code and subsubsub1 code

subroutine2 includes subsub2 code and subsub3 code and subsubsub3 code and …

“Deep” Computer Program



Deep Learning

- A model (e.g., neural network) with many layers, trained in a layer-wise way.
- Multiple layers work to build an improved feature space
  - The first layer learns 1st order features.
  - The 2nd layer learns higher order features
  - Layers often learn in an unsupervised mode and discover general features of the input space.
  - Then the final layer features are fed into supervised layer.

Deep Learning Task

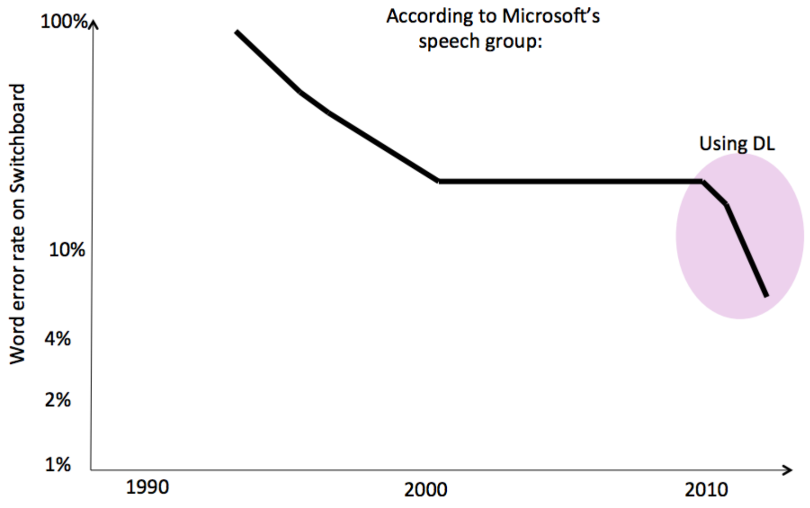
- Usually best when input space is locally structured; spatial or temporal
  - e.g., images, language, speech

Impact

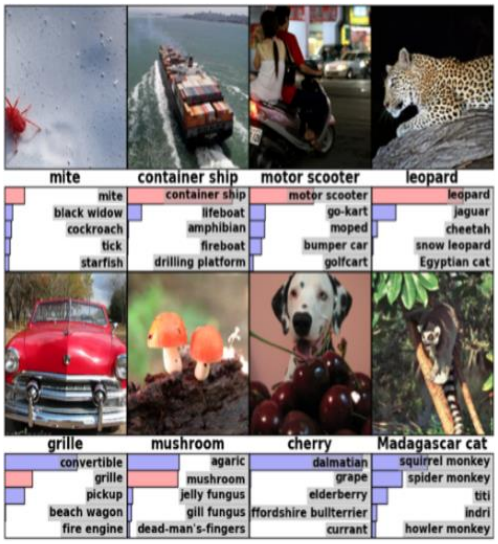
- Deep learning has revolutionized
  - Speech recognition
  - Object recognition

- More coming, including other areas of computer vision, NLP, dialogue, reinforcement learning, and so on.

Impact of Deep Learning



Object Recognition Breakthrough



- ImageNet
  - Achieves state-of-the-art on many object recognition tasks.

See deeplearning.cs.toronto.edu

Contents

Motivation

- Deep Architectures can be representationally efficient.
  - Fewer computational units for same function
- It can learn a distributed feature representation.

Distributed Feature Representation

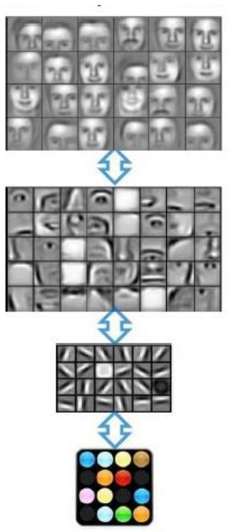
- One-hot representation is common in NLP:
  - “dog” = [1,0,0,…,0]
  - “cat” = [0,1,0,…,0]
  - “the” = [0,0,0,…,1]
- Word clustering has proven effective in many task:
  - “dog” = [1,0,0,0]
  - “cat” = [1,0,0,0]
  - “the” = [0,1,0,0]
- Distributed represented is a multi-clustering, modeling factors like POS & semantics:
  - “dog” = [1, 0, 0.9, 0.0]
  - “cat” = [1, 0, 0.5, 0.2]
  - “the” = [0, 1, 0.0, 0.0]

Motivation

- Deep Architectures can be representationally efficient.
  - Fewer computational units for same function
- It can learn a distributed feature representation.
- It can learn a hierarchical feature representation.

Hierarchical Feature Representation

- Hierarchical features effective captures part-and-whole relationships and naturally addresses multi-task problems.
- It is easier to monitor what is being learnt and to guide the machine to better subspaces.
- A good lower level representation can be used for many distinct tasks.



Motivation

- Deep Architectures can be representationally efficient.
  - Fewer computational units for same function
- It can learn a distributed feature representation.
- It can learn a hierarchical feature representation.
- It can exploit unlabeled data.

Contents

Definition of "Depth"

- It depends on elementary computational elements:
  - weighted sum, product, single neuron, kernel, etc.
- 1-Layer: Linear Classifier
  - Logistic Regression, Maximum Entropy Classifier
  - Perceptron, Linear SVM
- 2-Layers: Universal approximator
  - Multi-layer Perceptron, SVMs with kernels
  - Decision trees
- 3 or more Layers: compact universal approximator
  - Deep learning
  - Boosted decision trees

Neural Networks

- Goals: Learn function $f{:}x \rightarrow y$ that predicts correctly on new inputs x
- Step1: Choose a function model family
  - E.g., logistic regression, perceptron, SVM, etc.
- Step2: Optimize parameters w on the Training Data
  - E.g., minimize loss function

1-Layer Net (Logistic Regression)

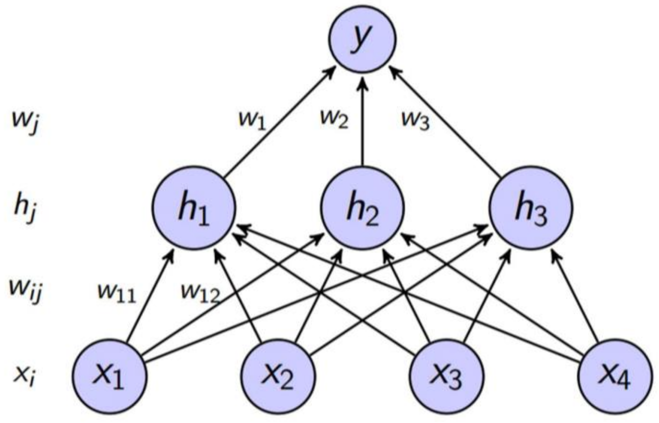
- Function model:

$$f(x) = \sigma(w^T \cdot x + b)$$

$$\sigma(z) = 1/(1 + \exp(-z))$$

- Training 1-Layer Nets
  - Easiest method: gradient descent
  - Stochastic gradient descent

2-Layer Nets (MLP)



$$f(x) = \sigma(\textstyle\sum_j w_j \cdot h_j) = \sigma(\sum_j w_j \cdot \sigma(\sum_i w_{ij} x_i))$$

2-Layer Nets (MLP)

- Training 2-Layer Nets: Backpropagation
  - Minimize error of calculated output.
  - Firstly, run sample through network to get result f(x)
  - “Errors” are propagated back and weights fixed according to their responsibility

2-Layer Nets (MLP)

- Problem with backpropagation
  - It requires labeled training data
    - Almost data is unlabeled.
  - The learning time does not scale well
    - It is very slow in networks with multiple hidden layers.
  - It can get stuck in poor local optima

Contents

Deep Network

- Deep Architecture – multiple layers
- Unsupervised training between layers can decompose the problem into distributed sub-problems (with higher levels of abstraction)

Training Deep Network

- Difficulties of supervised training
  - Early layers of MLP do not get trained well.
    - Error attenuates as it propagates to earlier layers.
    - Leads to very slow training.
    - Exacerbated since top layers can usually learn task "pretty well" and thus the error to earlier layers drops quickly.
  - Often not enough labeled data available.
  - Deep networks tend to have more local minima problems than shallow networks.

Greedy Layer-Wise Training

1. Train first layer using data without the labels (unsupervised).
2. Freeze the first layer parameters and start training the second layer using the output of the first layer.
3. Repeat this for as many layers as desired.
4. Use the outputs of the final layers as inputs to a supervised layer/model and train the last supervised layer.
5. Unfreeze all weights and find tune the full network by training with a supervised approach, given the pre-processed weight settings.

Greedy Layer-Wise Training

- It can avoid many problems:
  - Each layer gets full learning focus in its turn.
  - Can take advantage of the unlabeled data.
  - When finally tune the entire network with supervised training, the network weights have already been adjusted so that you are in a good error basin and just need fine tuning.

Contents

Deep Belief Nets (DBN)

- Goal: Discover useful latent features *h* from data *x*

- One possibility: Directed Graphical Models
  - *p(h1)* and *p(h2)* are a priori independent, but dependent given x:
    $\mathrm{p}(h1, h2|x) \neq p(h1|x) \cdot p(h2|x)$
  - Thus, posterior *p(h|e)*, which is needed for features or deep learning, is not easy to compute.

Undirected Graphical Model

- Boltzmann Machines
  - Defined Energy of the network and probability of a unit's state.

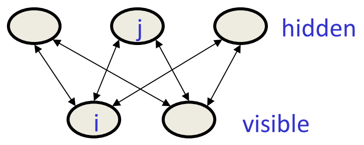
$$p(x, h) = \frac{1}{Z_\theta} \exp\left(E_\theta(x, h)\right)$$

$$E_\theta(x, h) = -\frac{1}{2}x^T Ux - \frac{1}{2}h^T Vh - x^T Wh - b^T w - d^T h$$

  - Posterior p(h|x) is also intractable

Restricted Boltzmann Machine

- Restricted Boltzmann Machine (RBM)
  - The building block of a DBN
    - 2-layer graphical model



  - Boltzmann Machine with only h-x interactions

$$E_\theta(x,h) = -x^T W h - b^T x - d^T h$$

  - Conditional distribution over hidden units factorizes
    - Computing posteriors p(h|x) or features (E[p(h|x)) is tractable.

Restricted Boltzmann Machine

- Training RBMs
  - Gradient of the Log-likelihood

$$\nabla_w \log P_w\left(x = x^{(m)}\right) = \nabla_{w_{ij}} \log \sum_h P_w\left(x = x^{(m)}, h\right)$$

$$= \nabla_{w_{ij}} \log \sum_h \frac{1}{Z_w} \exp\left(-E_w\left(x^{(m)}, h\right)\right)$$

$$= -\nabla_{w_{ij}} \log Z_w + \nabla_{w_{ij}} \log \sum_h \frac{1}{Z_w} \exp\left(-E_w\left(x^{(m)}, h\right)\right)$$

$$= \frac{1}{Z_w} \sum_{h,x} e^{\left(-E_w(x,h)\right)} \nabla_{w_{ij}} E_w(x,h) - \frac{1}{\sum_h e^{\left(-E_w\left(x^{(m)},h\right)\right)}} \sum_h e^{\left(-E_w\left(x^{(m)},h\right)\right)} \nabla_{w_{ij}} E_w\left(x^{(m)}, h\right)$$

$$= \sum_{h,x} P_w(x,h)\left[\nabla_{w_{ij}} E_w(x,h)\right] - \sum_h P_w\left(x^{(m)}, h\right) \nabla_{w_{ij}} E_w\left(x^{(m)}, h\right)$$

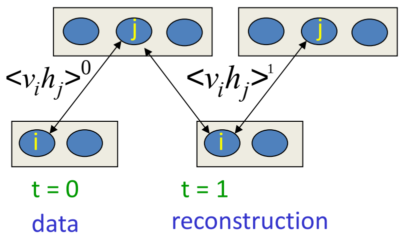
$$= -E_{p(x,h)}\left[x_i \cdot h_j\right] + E_{p\left(h|x=x^{(m)}\right)}\left[x_i^{(m)} \cdot h_j\right]$$

Restricted Boltzmann Machine

- Training RBMs (cont')
  - In the Gradient of the Log-likelihood, the first term is expensive.
  - Gibbs Sampling (sample x then h iteratively) works but re-running for each gradient step is slow.

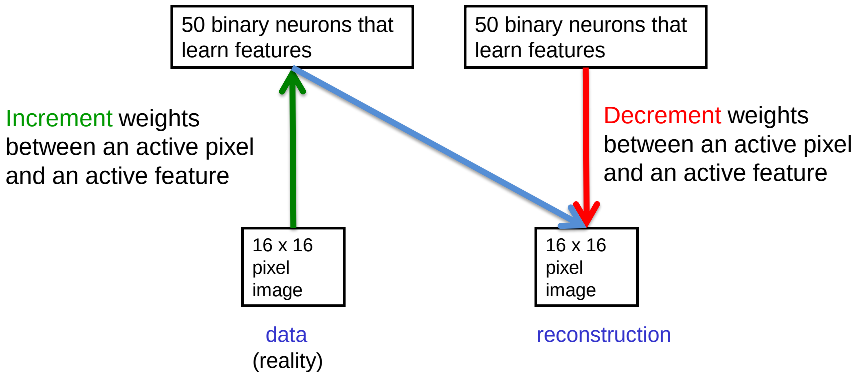
Restricted Boltzmann Machine

- Training RBMs (cont')
  - Contrastive Divergence
    - Start with a training vector on the visible units.
    - Update all the hidden units in parallel.
    - Update the all the visible units in parallel to get a "reconstruction".
    - Update the hidden units again.



$$\Delta w_{ij} = \varepsilon(<v_i h_j>^0 - <v_i h_j>^1)$$

Example: Handwritten 2s


50 binary neurons that learn features
50 binary neurons that learn features
Increment weights between an active pixel and an active feature
Decrement weights between an active pixel and an active feature
16 x 16 pixel image
16 x 16 pixel image
data
(reality)
reconstruction

Restricted Boltzmann Machine

- Training RBMs (cont')
  - Contrastive Divergence vs. Gradient
    - Gradient: pull down energy surface at the examples and <u>pull it up everywhere else</u>, with more emphasis where model puts more probability mass
    - Contrastive divergence: pull down energy surface at the examples and <u>pull it up in their neighborhood</u>, with more emphasis where model puts more probability mass

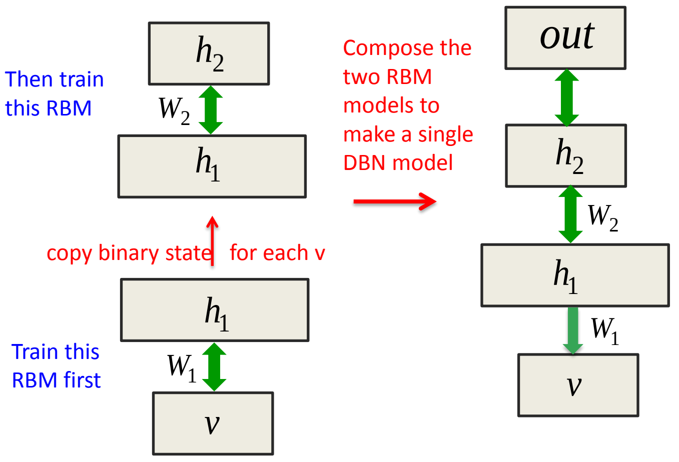
Restricted Boltzmann Machine

- Training RBMs (cont')
  - In the Gradient of the Log-likelihood, the first term is expensive.
  - Gibbs Sampling (sample x then h iteratively) works but re-running for each gradient step is slow.
  - Contrastive Divergence is a faster but biased method that initialized with training data.

Deep Belief Nets (DBN)

- DBN stacks RBMs layer-by-layer to get deep architecture.
- Layer-wise pre-training is critical
  - Firstly, train RBM to learn 1st layer of features h from input x
  - Then, treat h as input and learn a 2nd layer of features
  - Each added layer improves the variational lower bound on the log probability of training data

Deep Belief Nets (DBN)



Deep Belief Nets (DBN)

- Why greedy learning works?
  - Each time we learn a new layer, the inference at the layer below becomes incorrect, but the variational bound on the log probability of the data improves.
  - Since the bound starts as an equality, learning a new layer never decreases the log probability of the data, provided we start the learning from the tied weights
  - We have a guarantee we can loosen the restrictions and still feel confident.
    - Allow layers to vary in size.
    - Do not start the learning at each layer from the weights in the layer below.

Deep Belief Nets (DBN)

- Further fine-tuning can be obtained with the Wake-Sleep algorithm
  - Do stochastic bottom-up pass

    (adjust weights to reconstruct layer below)
  - Do a few iterations of Gibbs sampling at top-level RBM
  - Do stochastic top-down pass

    (adjust weights to reconstruct layer above)

Summary of DBNs

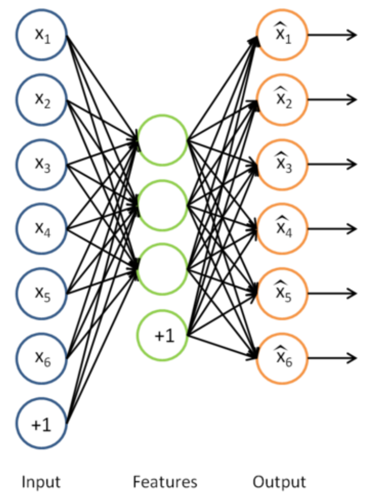
- Layer-wise pre-training is the innovation that enable training deep architectures.
- Pre-training focuses on optimizing likelihood on the data, not the target label.
- Undirected graphical model like RBM is used since a posteriori is computationally tractable.
- Learning RBM still require approximates inference since partition function is expensive.

Contents

Auto-Encoders

- The type of unsupervised learning which tries to discover generic features of the data
  - Learn identify function by learning important sub-features
  - Compression
  - Can use just new features in the new training set or concatenate both.

Auto-Encoders



Auto-Encoders

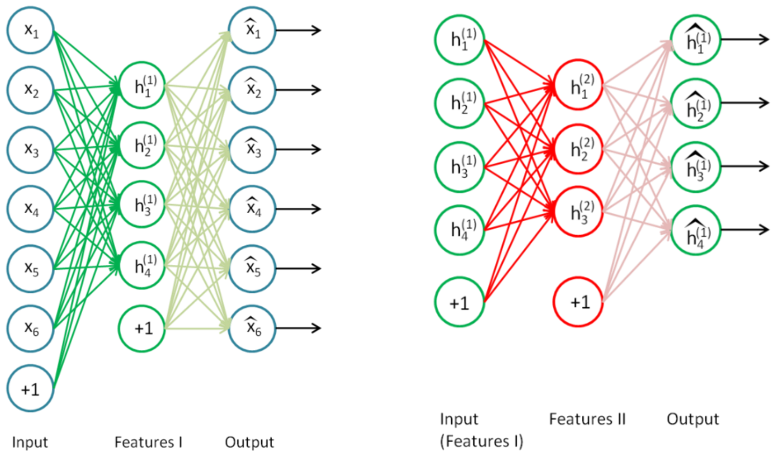
- Auto-Encoders are simpler non-probabilistic alternative to RBMs.
- Define encoder and decoder and pass the data through:

  Encoder: $h = f_\theta(x), e.g., h = \sigma(Wx + b)$
  Decoder: $x = g_\theta(h), e.g., x = \sigma(W'h + d)$
- Linear encoder/decoder with squared reconstruction error learns same subspace of PCA.
- Sigmoid encoder/decoder gives same form *p(h|x)*, *p(x|h)* as RBMs.

Stacked Auto-Encoders

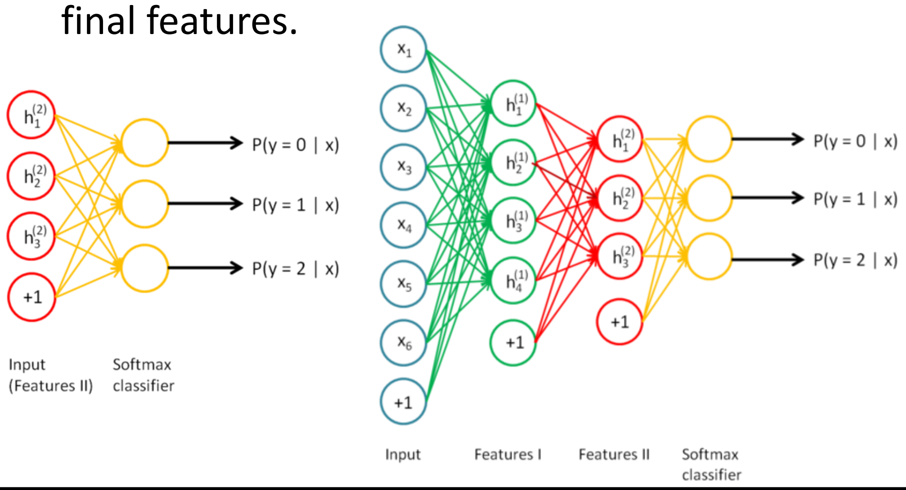
- Auto-encoders can be stacked in the same way RBMs are stacked to give Deep Architectures.
- Stack many (sparse) auto-encoders in succession and train them using greedy layer-wise training
- Drop the decode output layer each time

Stacked Auto-Encoders



Stacked Auto-Encoders

- Do supervised training on the last layer using final features.



Auto-Encoders

- Auto encoders will often do a dimensionality reduction
  - PCA-like or non-linear dimensionality reduction
- This leads to a "dense" representation which is nice in terms of parsimony
  - All features typically have non-zero values for any input and the combination of values contains the compressed information.
- However, this distributed and entangled representation can often make it more difficult for successive layers to pick out the salient features.

Sparse Auto-Encoders

- A *sparse* representation uses more features where at any given time a significant number of the features will have a 0 value
  - This leads to more localist variable length encodings where a particular node (or small group of nodes) with value 1 signifies the presence of a feature (small set of bases)
  - A type of simplicity bottleneck (regularizer)
  - This is easier for subsequent layers to use for learning

Sparse Auto-Encoders

- Implementation
  - Use more hidden nodes in the encoder
  - Use regularization techniques which encourage sparseness (e.g. a significant portion of nodes have 0 output for any given input)
    - Penalty in the learning function for non-zero nodes
    - Weight decay, etc.
  - De-noising Auto-Encoders

De-noising Auto-Encoders

- Stochastically corrupt training instance each time, but still train auto-encoder to decode the uncorrupted instance, forcing it to learn conditional dependencies within the instance.
- Better empirical results, handles missing values well.

Summary of Stacked Auto-Encoders

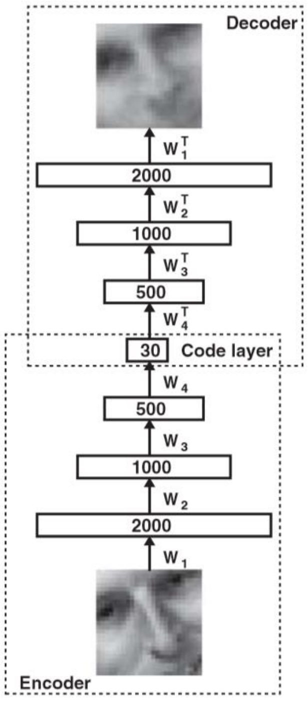
- Auto-encoders are computationally cheaper alternatives to RBMs.
- Auto-encoders learn to “compress” and “re-construct” input data. Low reconstruction error corresponds to an encoding that captures the main variations in data.
- Many variants of encoders are out there, and some provide effective ways to incorporate expertise domain knowledge.

Contents

Applications

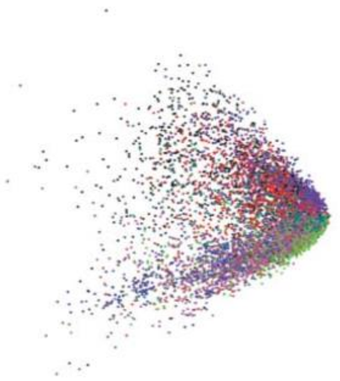
- Dimensionality reduction
  - Use a stacked RBM as deep auto-encoder
  1. Train RBM with images as input & output
  2. Limit one layer to few dimensions

→ Information has to pass through middle layer

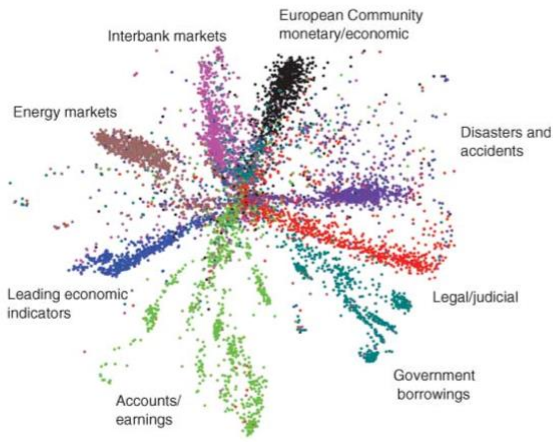


Applications

- Dimensionality reduction (cont’)



PCA



Deep RBN

Applications

- Classification
  - Unlabeled data is readily available
  - Example: Images from the web
  1. Download 10,000,000 images
  2. Train a 9-layer DNN
  3. Concepts are formed by DNN

→ 70% better than previous state of the art (by Le et al.)

Thank you ☺

Reference

- Geoffrey E. Hinton. Learning multiple layers of representation. 2007.
- Yoshua Bengio. Learning Deep Architectures for AI.
- Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation Learning: A Review and New Perspectives. 2012.
- Quoc V. Le, Marc'Aurelio Ranzato, Rajat Monga, Matthieu Devin, Kai Chen, Greg S. Corrado, Jeffrey Dean, and Andrew Y. Ng. Building High-level Features Using Large Scale Unsupervised Learning. ICML 2012.
- Slides of Geoffrey E. Hinton, Tutorial Deep Belief Nets, 2009
- Slides of Yoshua Bengio, Deep Learning for AI, 2014
- Slides of Kevin Duh, Deep Learning: An Introduction from the NLP Perspective, 2012.