UNIVERSITÀ DEGLI STUDI DI MILANO
DIPARTIMENTO DI INFORMATICA

Introduction to Deep Learning

Deep Learning in Biometrics

Ruggero Donida Labati

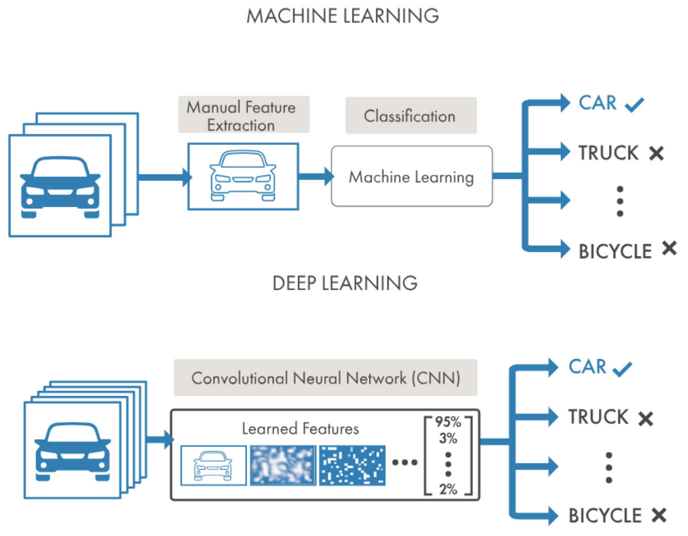
Academic year 2018/2019

Summary

- Introduction
- Convolutionary Neural Networks
- Autoencoders
- Training Deep Networks
- Deep Belief Networks
- Validation
- Transfer Learning
- Applications

Introduction

Machine Learning V.S. Deep Learning



Traditional Recognition Approach

Features are not learned

Input data (pixels) → feature representation (hand-crafted) → Learning Algorithm (e.g., SVM)

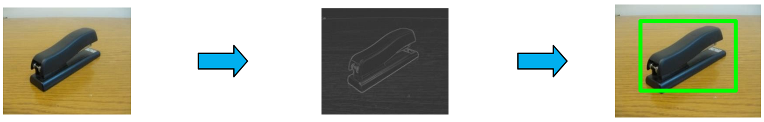
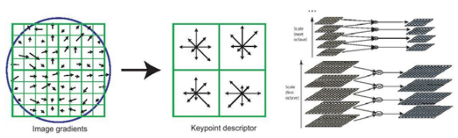


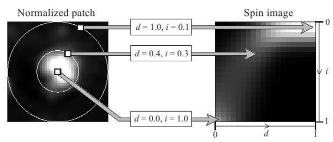
Image → Low-level vision features (edges, SIFT, HOG, etc.) → Object detection / classification

James Hays

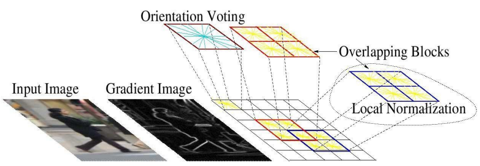
Feature extraction



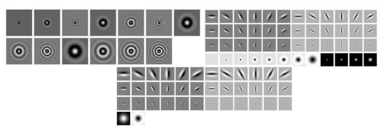
SIFT



Spin image



HoG



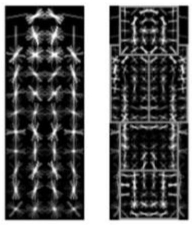
Textons

and many others:

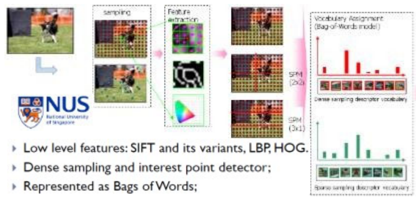
SURF, MSER, LBP, Color-SIFT, Color histogram, GLOH, …..

Motivation

- Features are key to recent progress in recognition
- Multitude of hand-designed features currently in use
- Where next? Better classifiers? building better features?



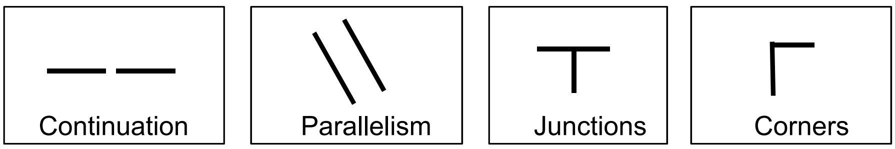
Felzenszwalb, Girshick, McAllester and Ramanan, PAMI 2007



Yan & Huang
(Winner of PASCAL 2010 classification competition)

Mid level representations

- Mid-level cues



"Tokens" from Vision by D.Marr:

- Object parts:

- Difficult to hand-engineer→ What about learning them?

Learning Feature Hierarchy (1/2)

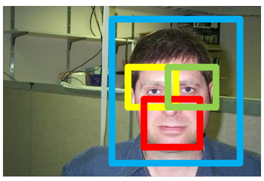
- Learn hierarchy
- All the way from pixels → classifier
- One layer extracts features from output of previous layer

- Train all layers jointly

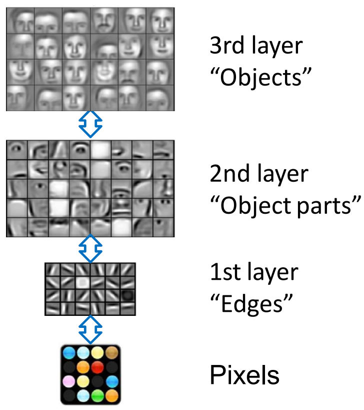
Learning Feature Hierarchy (2/2)

Learn **useful higher-level features** from images

Input data



Feature representation

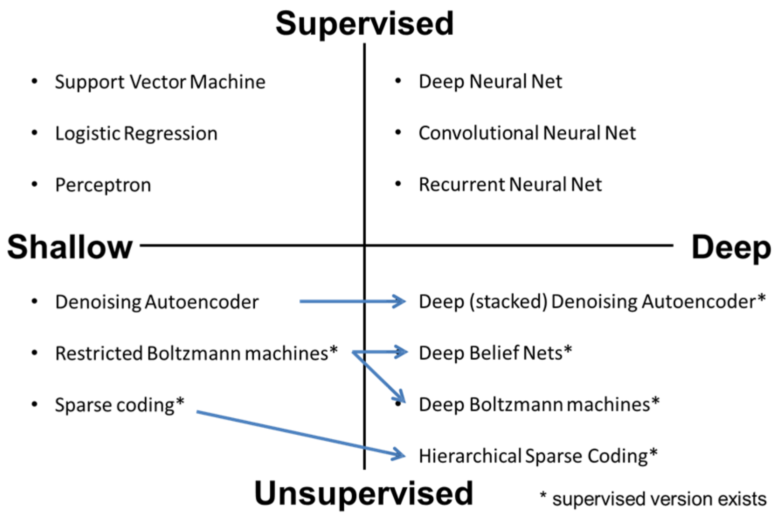


Lee et al., ICML 2009;
CACM 2011

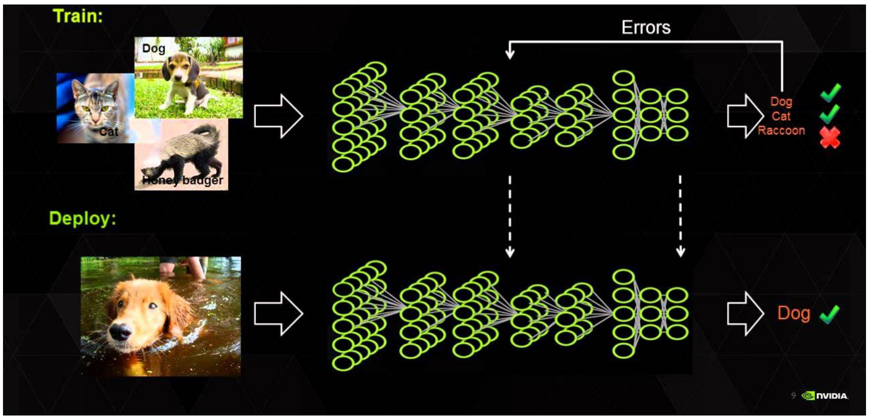
Approaches to Learn Features

- Supervised Learning
  - End-to-end learning of deep architectures (e.g., deep neural networks) with back-propagation
  - Works well when the amounts of labels is large
  - Structure of the model is important (e.g. convolutional structure)
- Unsupervised Learning
  - Learn statistical structure or dependencies of the data from unlabeled data
  - Layer-wise training
  - Useful when the amount of labels is not large

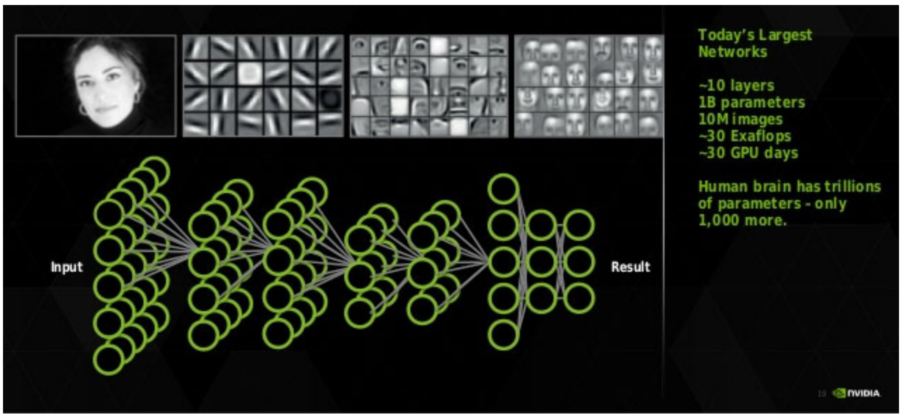
Taxonomy



Deep Learning approach

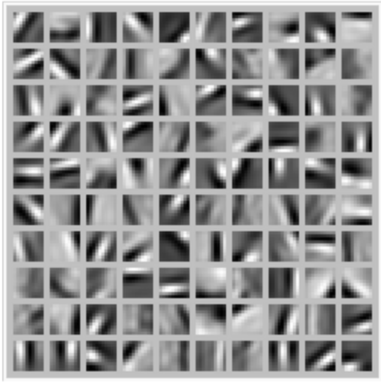


What makes Deep Learning deep?



Deep Learning tasks

- Usually best when input space is locally structured – spatial or temporal: images, language, etc. vs arbitrary input features
- Images Example: view of a learned vision feature layer (Basis)
- Each square in the figure shows the input image that maximally activates one of the 100 unit



UNIVERSITÀ DEGLI STUDI DI MILANO
DIPARTIMENTO DI INFORMATICA

Why Deep Learning

- Biological Plausibility – e.g. Visual Cortex
- Hastad proof - Problems which can be represented with a polynomial number of nodes with *k* layers, may require an exponential number of nodes with *k*-1 layers (e.g. parity)
- Highly varying functions can be efficiently represented with deep architectures
  - Less weights/parameters to update than a less efficient shallow representation
- Sub-features created in deep architecture can potentially be shared between multiple tasks
  - Type of Transfer/Multi-task learning

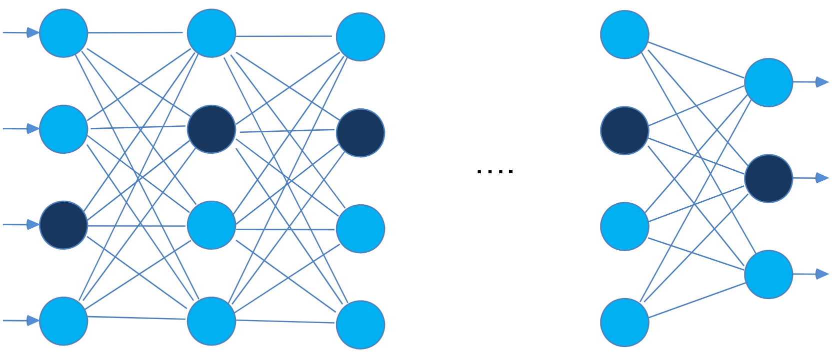
UNIVERSITÀ DEGLI STUDI DI MILANO
DIPARTIMENTO DI INFORMATICA

Early Work

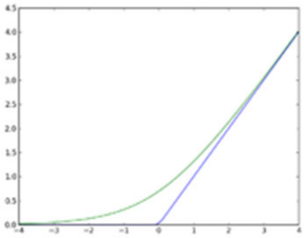
- Fukushima (1980) – Neo-Cognitron
- LeCun (1998) – Convolutional Neural Networks (CNN)
  - Similarities to Neo-Cognitron
- Many layered MLP with backpropagation
  - Tried early but without much success
    - Very slow
    - Diffusion of gradient
  - Very recent work has shown significant accuracy improvements by "patiently" training deeper MLPs with BP using fast machines (GPUs) – may be best most general approach

Backpropagation Training Problems

- Error attenuation, long fruitless training
- Recent – Long patient training with GPUs and special hardware
- Becoming more popular, Rectified Linear Units



Rectified Linear Units



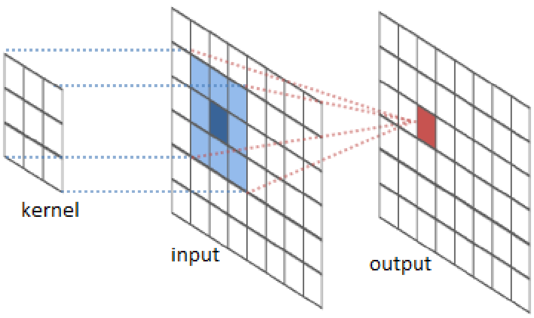
- More efficient gradient propagation, derivative is 0 or constant, just fold into learning rate
- More efficient computation.
- Sparse activation: For example, in a randomly initialized networks, only about 50% of hidden units are activated (having a non-zero output)

Convolutionary Neural Networks

Convolutionary Neural Networks (CNN)

- Networks built specifically for problems with low dimensional (e.g. 2-*d*) local structure
  - Character recognition – where neighboring pixels will have high correlations and local features (edges, corners, etc.), while distant pixels (features) are un-correlated
  - Natural images have the property of being stationary, meaning that the statistics of one part of the image are the same as any other part
  - While standard NN nodes take input from all nodes in the previous layer, CNNs enforce that a node receives only a small set of features which are spatially or temporally close to each other called *receptive fields* from one layer to the next (e.g. 3x3, 5x5), thus enforcing ability to handle local 2-D structure.
    - Can find edges, corners, endpoints, etc.
    - Good for problems with local 2-D structure, but lousy for general learning with abstract features having no prescribed ordering

Convolution



$$[f * g](t) \equiv \int_0^t f(\tau)\, g(t-\tau)\, d\tau,$$

CNN - Schema

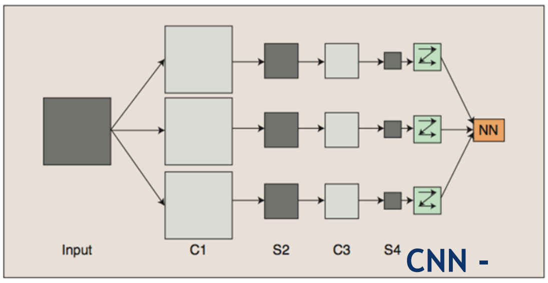
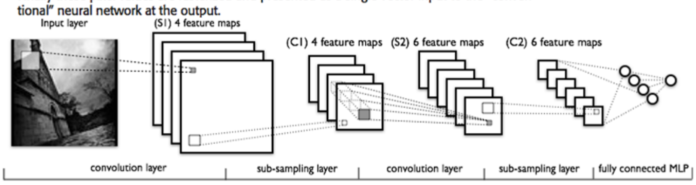


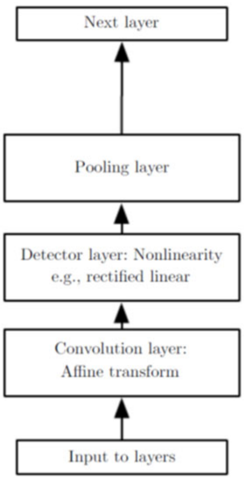
FIGURE 2 Conceptual example of convolutional neural network. The input image is convolved with three trainable filters and biases as in Figure 1 to produce three feature maps at the C1 level. Each group of four pixels in the feature maps are added, weighted, combined with a bias, and passed through a sigmoid function to produce the three feature maps at S2. These are again filtered to produce the C3 level. The hierarchy then produces S4 in a manner analogous to S2. Finally these pixel values are rasterized and presented as a single vector input to the "conventional" neural network at the output.



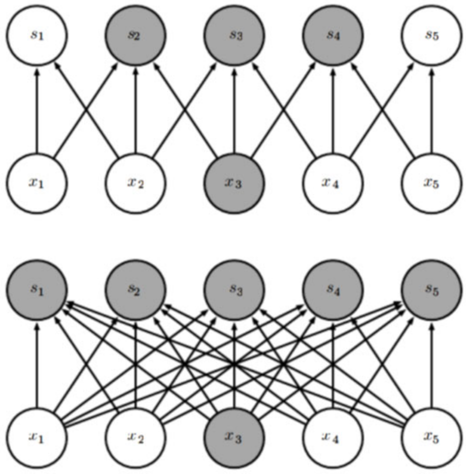
C layers are convolutions, S layers pool/sample

Often starts with fairly raw features at initial input and lets CNN discover improved feature layer for final supervised learner – eg. MLP/BP

CNN - Terminology



CNN - Sparse connectivity



CNN – Translation Invariance

- The 2-*d* planes of nodes (or their outputs) at subsequent layers in a CNN are called *feature maps*
- To deal with translation invariance, each node in a feature map has the same weights (based on the feature it is looking for), and each node connects to a different overlapping receptive field of the previous layer
- Thus each *feature map* searches the full previous layer to see if and how often its feature occurs (precise position not critical)
  - The output will be high at each node in the map corresponding to a receptive field where the feature occurs
  - Later layers could concern themselves with higher order combinations of features and rough relative positions
  - Each calculation of a node's net value, Σxw in the feature map, is called a convolution, based on the similarity to standard overlapping convolutions

CNN - Structure

- Each node (e.g. convolution) is calculated for each receptive field in the previous layer
  - During training the corresponding weights are always tied to be the same
  - Thus a relatively small number of unique weight parameters to learn, although they are replicated many times in the feature map
  - Each node output in CNN is sigmoid($\Sigma xw + b$) (just like BP)
  - Multiple feature maps in each layer
  - Each feature map should learn a different translation invariant feature
- Convolution layer causes total number of features to increase

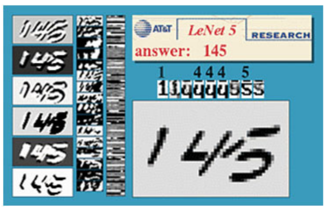
Sub-Sampling (Pooling)

- Convolution and sub-sampling layers are interleaved
- Sub-sampling (Pooling) allows number of features to be diminished, non-overlapped
  - Reduces spatial resolution and thus naturally decreases importance of exactly where a feature was found, just keeping the rough location
  - Averaging or Max-Pooling (Just as long as the feature is there, take the max, as exact position is not that critical)
  - 2x2 pooling would do 4:1 compression, 3x3 9:1, etc.
  - Pooling smooths the data and makes the data invariant to small translational changes
  - Since after first layer, there are always multiple feature maps to connect to the next layer, it is a pre-made human decision as to which previous maps the current map receives inputs from

CNN - Training

- Trained with BP but with weight tying in each feature map
  - Randomized initial weights through entire network
  - Just average the weight updates over the tied weights in feature map layers
- While all weights are trained, the structure of the CNN is currently usually hand crafted with trial and error.
  - Number of total layers, number of receptive fields, size of receptive fields, size of sub-sampling (pooling) fields, which fields of the previous layer to connect to
  - Typically decrease size of feature maps and increase number of feature maps for later layers

CNN - LeNet 5



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X				X	X	X			X	X	X	X		X	X
1	X	X				X	X	X			X	X	X	X		X
2	X	X	X				X	X	X			X		X	X	X
3		X	X	X			X	X	X	X			X		X	X
4			X	X	X			X	X	X	X		X	X		X
5				X	X	X			X	X	X	X		X	X	X

TABLE I
EACH COLUMN INDICATES WHICH FEATURE MAP IN S2 ARE COMBINED BY THE UNITS IN A PARTICULAR FEATURE MAP OF C3.

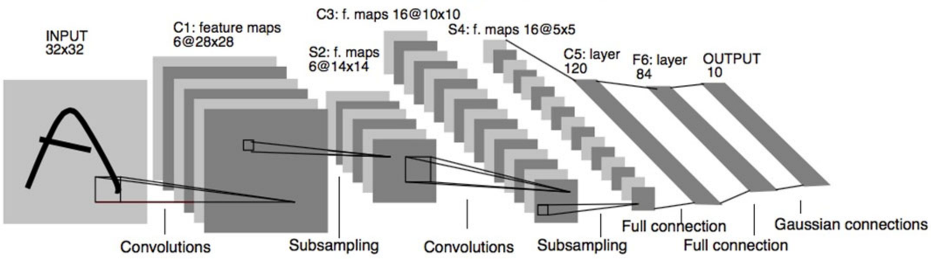


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

Training Deep Networks

Training deep networks

- Build a feature space
  - Note that this is what we do with SVM kernels, or trained hidden layers in BP, etc., but now we will build the feature space using deep architectures
  - Unsupervised training between layers can decompose the problem into distributed sub-problems (with higher levels of abstraction) to be further decomposed at subsequent layers

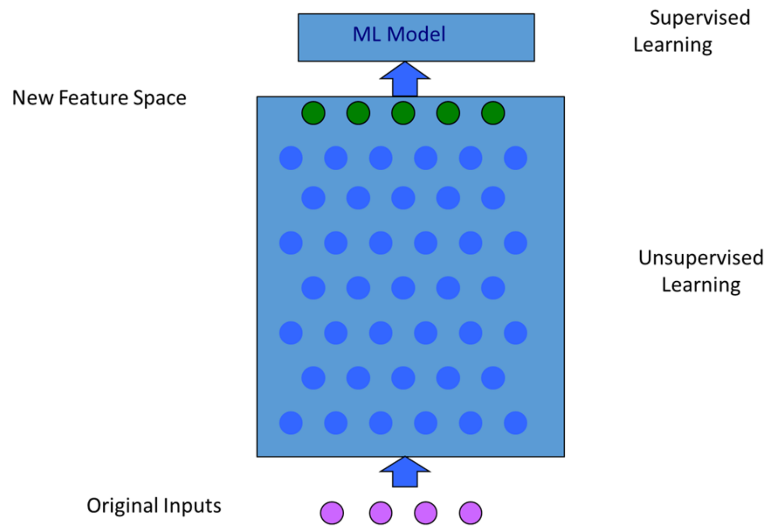
Problems in training deep networks

- Difficulties of supervised training of deep networks
  - Early layers of MLP do not get trained well
  - Diffusion of Gradient – error attenuates as it propagates to earlier layers
  - Leads to very slow training
  - Exacerbated since top couple layers can usually learn any task "pretty well" and thus the error to earlier layers drops quickly as the top layers "mostly" solve the task– lower layers never get the opportunity to use their capacity to improve results, they just do a random feature map
  - Need a way for early layers to do effective work
  - Instability of gradient in deep networks: Vanishing or exploding gradient
    - Product of many terms, which unless "balanced" just right, is unstable
    - Either early or late layers stuck while "opposite" layers are learning
- Often not enough labeled data available while there may be lots of unlabeled data
  - Can we use unsupervised/semi-supervised approaches to take advantage of the unlabeled data
- Deep networks tend to have more sensitive training issues problems than shallow networks during supervised training

Greedy Layer-Wise Training

- One answer is greedy layer-wise training
  1. Train first layer using your data without the labels (unsupervised)
  2. Then freeze the first layer parameters and start training the second layer using the output of the first layer as the unsupervised input to the second layer
  3. Repeat this for as many layers as desired
  4. Use the outputs of the final layer as inputs to a supervised layer/model and train the last supervised layer(s) (leave early weights frozen)
  5. Unfreeze all weights and fine tune the full network by training with a supervised approach, given the *pre-training* weight settings

Deep Net with Greedy Layer Wise Training



Adapted from © Tony Martinez

UNIVERSITÀ DEGLI STUDI DI MILANO
DIPARTIMENTO DI INFORMATICA

Greedy Layer-Wise Training

- Greedy layer-wise training avoids many of the problems of trying to train a deep net in a supervised fashion
  - Each layer gets full learning focus in its turn since it is the only current "top" layer
  - Can take advantage of unlabeled data
  - When you 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. This helps with problems of
    - Ineffective early layer learning
    - Deep network local minima

Adapted from © Tony Martinez

UNIVERSITÀ DEGLI STUDI DI MILANO
DIPARTIMENTO DI INFORMATICA

Autoencoders

Unsupervised Learning (1/2)

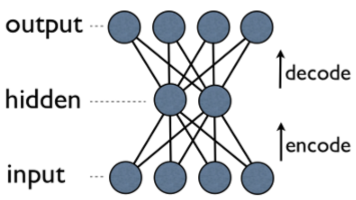
- Model distribution of input data
- Can use unlabeled data (unlimited)
- Can be refined with standard supervised techniques (e.g. backprop)
- Useful when the amount of labels is small

Unsupervised Learning (2/2)

- Main idea: model distribution of input data
  - Reconstruction error + regularizer (sparsity, denoising, etc.)
  - Log-likelihood of data
- Models
  - Basic: PCA, KMeans
  - Denoising autoencoders
  - Sparse autoencoders
  - Restricted Boltzmann machines
  - Sparse coding
  - Independent Component Analysis
  - ...

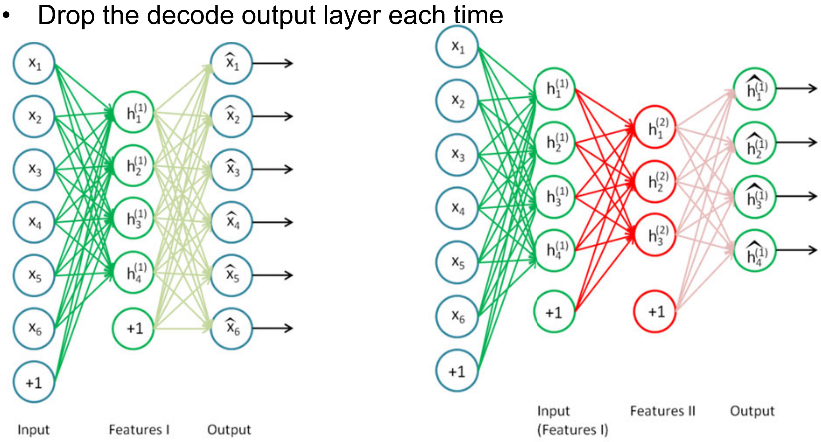
Autoencoders

- Try to discover generic features of the data
  - Learn identity function by learning important sub-features (not by just passing through data)
  - Compression, etc.
  - Can use just new features in the new training set or concatenate both



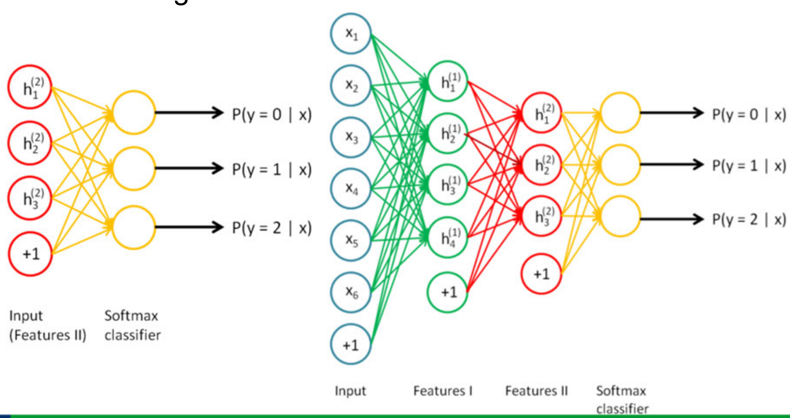
Stacked Auto-Encoders (1/2)

- Bengio (2007) – After Deep Belief Networks (2006)
- 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 (2/2)

- Do supervised training on the last layer using final features
- Then do supervised training on the entire network to fine-tune all weights



Sparse Encoders

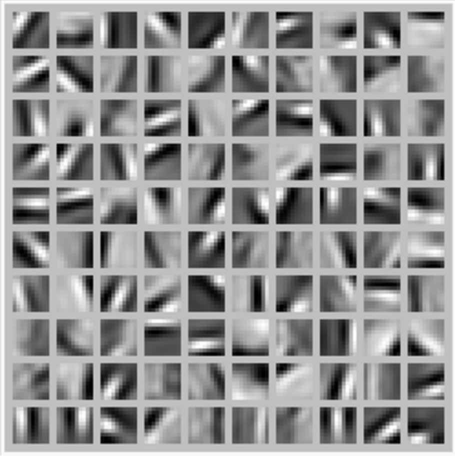
- Auto encoders will often do a dimensionality reduction
  - PCA-like or non-linear dimensionality reduction
- This leads to a "dense" representation which is nice in terms of parsimony
  - All features typically have non-zero values for any input and the combination of values contains the compressed information
- A *sparse* representation uses more features where at any given time many/most of the features will have a 0 value
  - Thus there is an implicit compression each time but with varying nodes
  - 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

Implementing a sparse Auto-Encoder

- 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-Encoder
  - 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

Sparse Representation

- Easier to learn if sparse



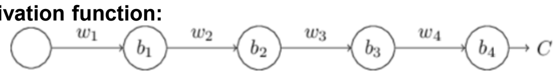
Properties of Stacked Auto-Encoders

- Concatenation approach (i.e. using both hidden features and original features in final (or other) layers) can be better if not doing fine tuning. If fine tuning, the pure replacement approach can work well.
- Always fine tune if there is a sufficient amount of labeled data
- For real valued inputs, MLP training is like regression and thus could use linear output node activations, still sigmoid at hidden
- Stacked Auto-Encoders empirically not quite as accurate as DBNs (Deep Belief Networks)

Deep Belief Networks

Vanishing Gradient and Backpropagation

- **One of the reasons that made the training of deep neural networks a difficult task is related to the vanishing gradient, which results from gradient based training techniques and the backpropagation algorithm**
- **Considering a very simple deep network, with only one neuron per layer, a cost *C* and a sigmoid activation function:**



- **A small change in b_1 sets off a series of cascading changes in the network**

$\Delta a_1 \approx \frac{d\sigma(\omega_1 a_0 + b_1)}{db_1}\Delta b_1$ or $\Delta a_1 = \sigma'(z_1)\Delta b_1$ The chance in a_1, then causes a change in the weighted input z_2, which would be given by $\Delta z_2 \approx \sigma'(z_1)\omega_2\Delta b_1$. Basically, a term $\sigma'(z_j)$ and ω_j is picked up in every neuron

The resulting change in cost (divided by Δb_1), produced by the change Δb_1 is given by:

$$\frac{dC}{db_1} = \sigma'(z_1) \times \omega_2 \times \sigma'(z_2) \times \omega_3 \times \sigma'(z_3) \times \omega_4 \times \sigma'(z_4) \times \frac{dC}{da_4}$$

- **For a sigmoid, $\sigma'(0) = \frac{1}{4}$, in this sense, the terms $|\omega_j \sigma'(z_j)| < \frac{1}{4}$, contributing to a vanishing gradient**

Boltzmann Machines (BM)

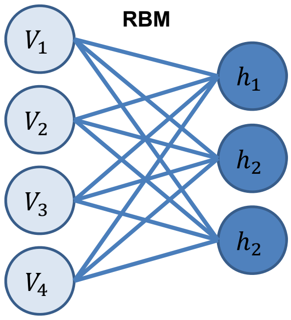
- The vanishing gradient results in very slow training for the front layers of the network
- One of the solutions to this issue was proposed by Hinton (2006). He proposed to use a Restricted Boltzmann Machine (RBM) to model each new layer of higher level features

- RBMs are energy based models, they associate a scalar energy to each configuration of the variables of interest
- Energy based probabilistic models define a probability distribution as:

$$p(x) = \frac{e^{-E(x)}}{Z} \text{ where } Z = \sum_x e^{-E(x)}$$

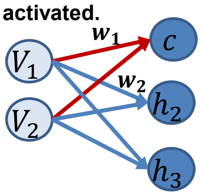
- An energy-based model can be learnt by performing (stochastic) gradient descent on the empirical negative log-likelihood of the training data, where the log-likelihood and the loss function are:

$$L(\theta, D) = \frac{1}{N} \sum_{x^i \epsilon D} log\ p(x^i) \quad \text{and} \quad l(\theta, D) = -L(\theta, D)$$

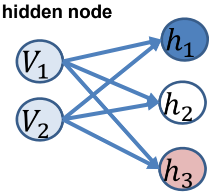


RBM Training Summary

1. Forward Pass: Inputs are combined with an individual weights and a bias. Some hidden nodes are activated.

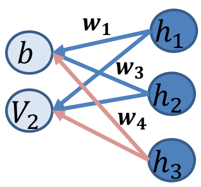


Input being passed to first hidden node



h_1 activates in this example

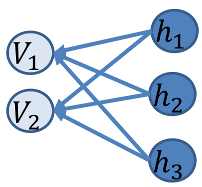
2. Backward Pass: Activations are combined with an individual weight and a bias. Results are passed to the visible layer.



Activations are passed to visible layer for reconstruction

3. Divergence calculation: Input x and samples $\tilde{x}$ are compared in visible layer. Parameters are updated and steps are repeated

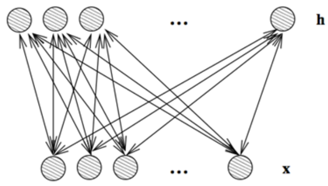
$$-\frac{dlogp(x)}{d\theta}$$



Update W, b, c

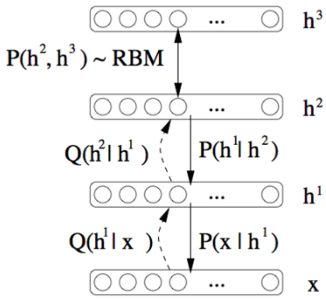
Deep Belief Networks (DBN)

- Geoff Hinton (2006)
- Uses Greedy layer-wise training but each layer is an RBM (Restricted Boltzmann Machine)
- RBM is a constrained Boltzmann machine with
  - No lateral connections between
  - hidden (**h**) and visible (**x**) nodes
  - Symmetric weights
  - Does not use annealing/temperature, but that is all right since each RBM not seeking a global minima, but rather an incremental transformation of the feature space
  - Typically uses probabilistic logistic node, but other activations possible



Deep Belief Network Training

- Same greedy layer-wise approach
- First train lowest RBM ($h^0 – h^1$) using RBM update algorithm (note h^0 is x)
- Freeze weights and train subsequent RBM layers
- Then connect final outputs to a supervised model and train that model
- Finally, unfreeze all weights, and train full DBN with supervised model to fine-tune weights



DBN Execution

- After all layers have learned then the output of the last layer can be input to a supervised learning model
- Note that at this point we could potentially throw away all the downward weights in the network as they will not actually be used during the normal feedforward execution process (as we did with the Stacked Auto Encoder)
  - Note that except for the downward bias weights **b** they are the same symmetric weights anyways
  - If we are relaxing *M* times in the top layer then we would still need the downward weights for that layer
  - Also if we are generating **x** values we would need all of them
- The final weight tuning is usually done with backpropagation, which only updates the feedforward weights, ignoring any downward weights

Transfer Learning

Pretrained models

- Since modern ConvNets take 2-3 weeks to train across multiple GPUs on ImageNet, it is common to see people release their final ConvNet checkpoints for the benefit of others who can use the networks for fine-tuning.
- For example, the Caffe library has a Model Zoo where people share their network weights.

ConvNets as feature extractors

- Take a ConvNet pretrained on ImageNet, remove the last fully-connected layer (this layer's outputs are the 1000 class scores for a different task like ImageNet), then treat the rest of the ConvNet as a fixed feature extractor for the new dataset. In an AlexNet, this would compute a 4096-D vector for every image that contains the activations of the hidden layer immediately before the classifier. We call these features CNN codes. It is important for performance that these codes are ReLUd (i.e. thresholded at zero) if they were also thresholded during the training of the ConvNet on ImageNet (as is usually the case). Once you extract the 4096-D codes for all images, train a linear classifier (e.g. Linear SVM or Softmax classifier) for the new dataset.

Fine-tuning the ConvNet

- This strategy is to not only replace and retrain the classifier on top of the ConvNet on the new dataset, but to also fine-tune the weights of the pretrained network by continuing the backpropagation. It is possible to fine-tune all the layers of the ConvNet, or it's possible to keep some of the earlier layers fixed (due to overfitting concerns) and only fine-tune some higher-level portion of the network. This is motivated by the observation that the earlier features of a ConvNet contain more generic features (e.g. edge detectors or color blob detectors) that should be useful to many tasks, but later layers of the ConvNet becomes progressively more specific to the details of the classes contained in the original dataset. In case of ImageNet for example, which contains many dog breeds, a significant portion of the representational power of the ConvNet may be devoted to features that are specific to differentiating between dog breeds.

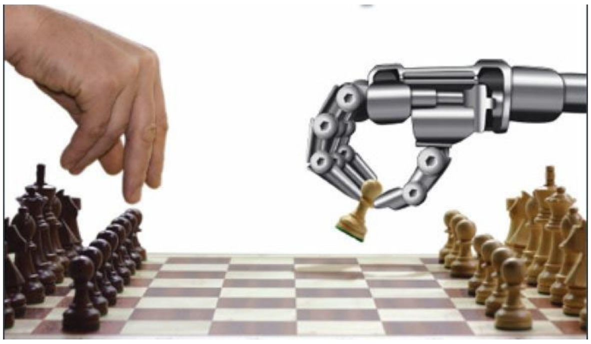
Applications

UNIVERSITÀ DEGLI STUDI DI MILANO
DIPARTIMENTO DI INFORMATICA

Face recognition: recognition performance

LFW dataset (13323 images, 5749 celebrities)

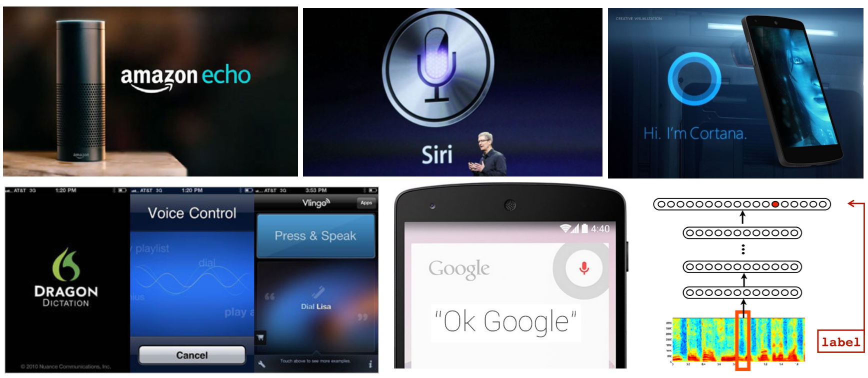
- deep learning strategies surpassed humans
- ≈ 99 % of accuracy for verification
- > 95 % of accuracy for closed-set identification



UNIVERSITÀ DEGLI STUDI DI MILANO
DIPARTIMENTO DI INFORMATICA

Deep learning for voice signlas

- Speaker recognition
- Speach recognition



UNIVERSITÀ DEGLI STUDI DI MILANO
DIPARTIMENTO DI INFORMATICA

Who is using deep learning?

NVIDIA Deep Learning Course

UNIVERSITÀ DEGLI STUDI DI MILANO
DIPARTIMENTO DI INFORMATICA

MATLAB

- https://it.mathworks.com/campaigns/products/offer/deep-learning-with-matlab.html
- https://it.mathworks.com/videos/series/introduction-to-deep-learning.html

UNIVERSITÀ DEGLI STUDI DI MILANO
DIPARTIMENTO DI INFORMATICA

MatConvNet: CNNs for MATLAB

- MatConvNet is a MATLAB toolbox implementing Convolutional Neural Networks (CNNs) for computer vision applications
- http://www.vlfeat.org/matconvnet/

UNIVERSITÀ DEGLI STUDI DI MILANO
DIPARTIMENTO DI INFORMATICA

Caffe

- Caffe is a deep learning framework made with expression, speed, and modularity in mind. It is developed by Berkeley AI Research (BAIR) and by community contributors.
- http://caffe.berkeleyvision.org/

UNIVERSITÀ DEGLI STUDI DI MILANO
DIPARTIMENTO DI INFORMATICA

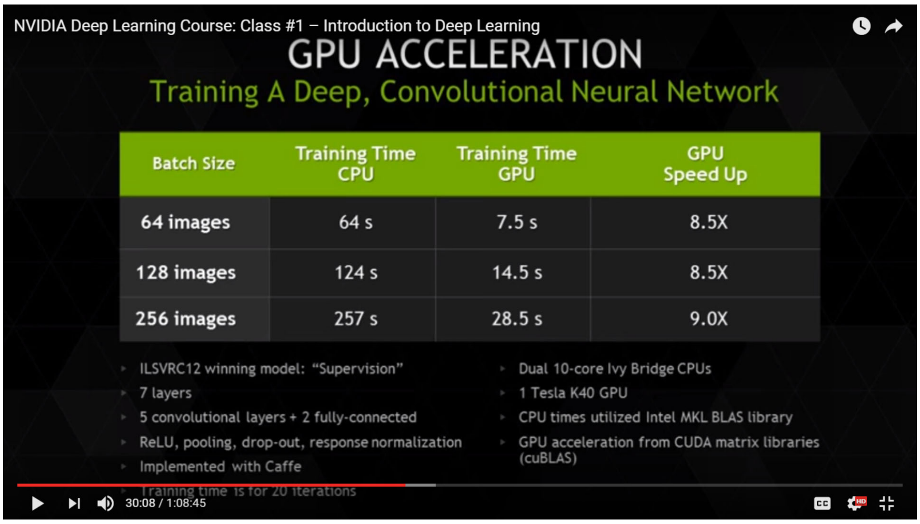
Tehano

- Theano is a Python library that allows you to define, optimize, and evaluate mathematical expressions involving multi-dimensional arrays efficiently.
- http://deeplearning.net/software/theano/

TensorFlow

- TensorFlow™ is an open source software library for high performance numerical computation. Its flexible architecture allows easy deployment of computation across a variety of platforms (CPUs, GPUs, TPUs), and from desktops to clusters of servers to mobile and edge devices.
- Originally developed by researchers and engineers from the Google Brain team within Google's AI organization, it comes with strong support for machine learning and deep learning and the flexible numerical computation core is used across many other scientific domains.
- https://www.tensorflow.org/

GPU



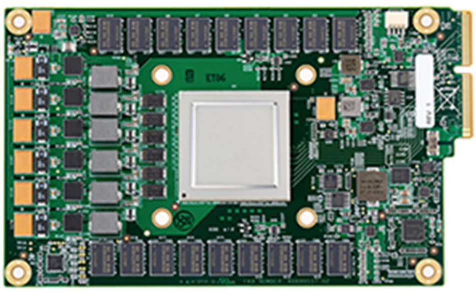
Batch Size	Training Time CPU	Training Time GPU	GPU Speed Up
64 images	64 s	7.5 s	8.5X
128 images	124 s	14.5 s	8.5X
256 images	257 s	28.5 s	9.0X

NVIDIA Deep Learning Course

UNIVERSITÀ DEGLI STUDI DI MILANO
DIPARTIMENTO DI INFORMATICA

TPU

- A tensor processing unit (TPU) is an AI accelerator application-specific integrated circuit (ASIC) developed by Google specifically for neural network machine



UNIVERSITÀ DEGLI STUDI DI MILANO
DIPARTIMENTO DI INFORMATICA

Intel Movidius

- What can you do with a Raspberry PI and a Movidius?

- https://developer.movidius.com/