Multilayer Neural Networks

(are no longer old-fashioned!)

MLSS TUEBINGEN 2013

LEON BOTTOU

Success stories

Record performance

- MNIST (1988, 2003, 2012)
- ImageNet (2012)
- …

Real applications

- Check reading (AT&T Bell Labs, 1995 – 2005)
- Optical character recognition (Microsoft OCR, 2000)
- Cancer detection from medical images (NEC, 2010)
- Object recognition (Google and Baidu’s photo taggers, 2013)
- Speech recognition (Microsoft, Google, IBM switched in 2012)
- Natural Language Processing (NEC 2010)
- …

Lecture 1

Neural information processing

- Origins
- Rumelhart’s propositional network
- Network construction kit
- Convolutional networks

Lecture 2

Training multilayer networks

- Optimization basics
- Initialization
- Stochastic gradient descent
- Improved algorithms

Lecture 3

Deep networks for complex tasks

- Introduction
- Structured problems
- Auxiliary tasks
- Circuit algebra

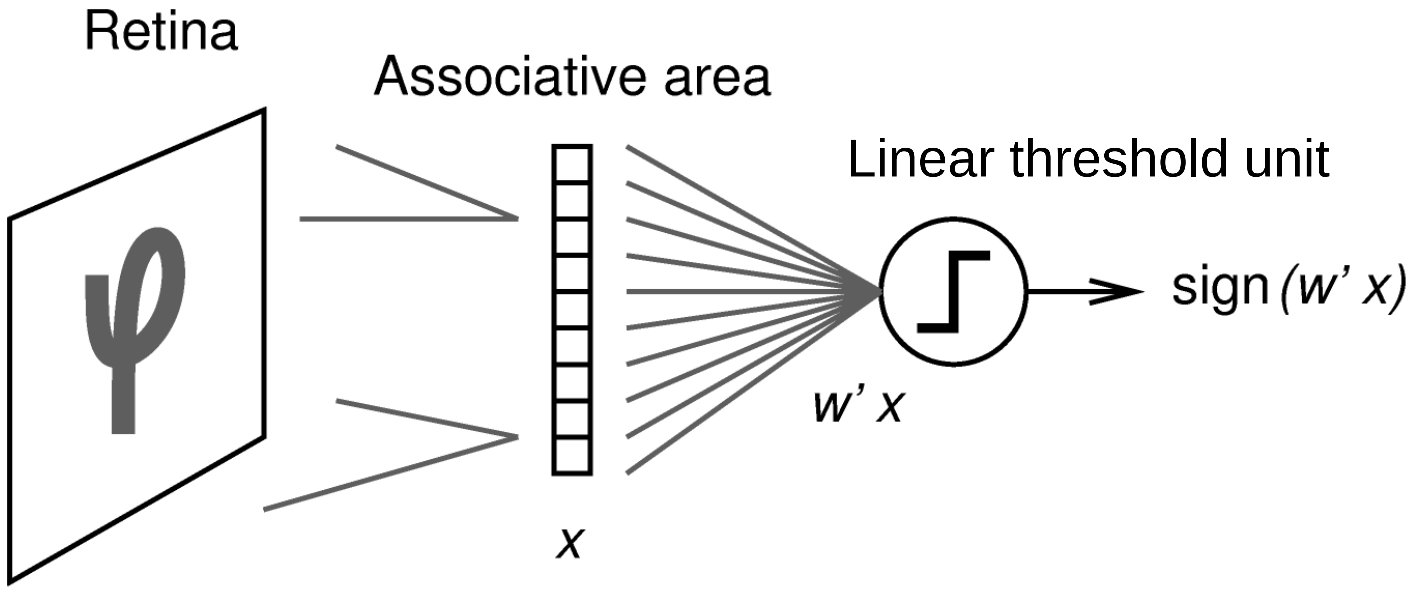
Neural Information Processing

Origins

The perceptron

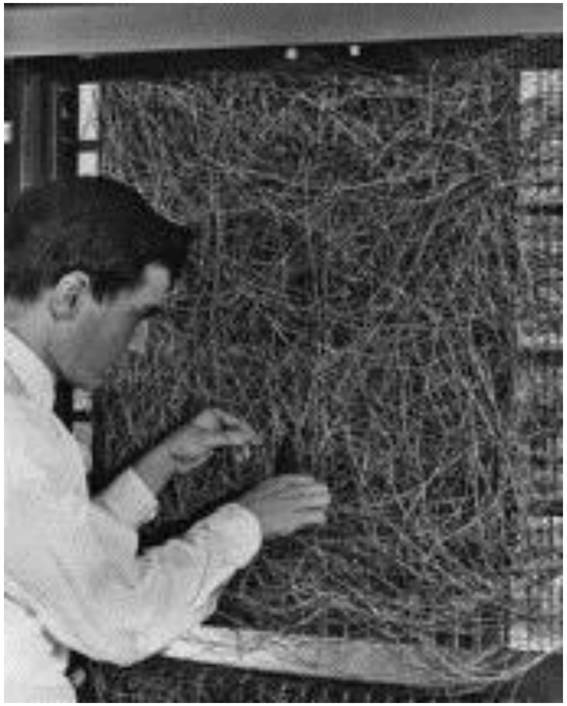
Rosenblatt 1957

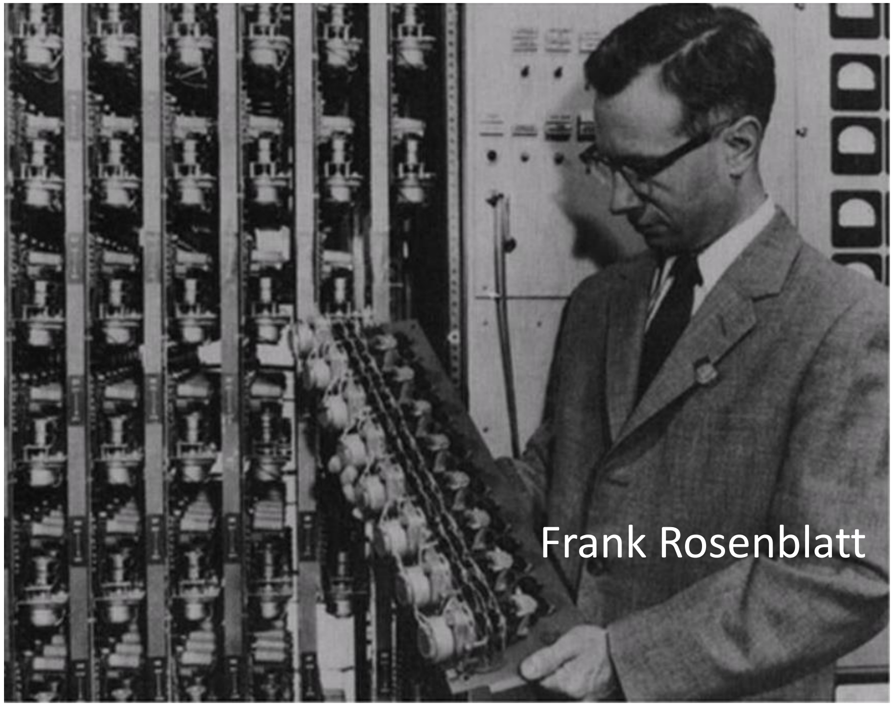
The perceptron



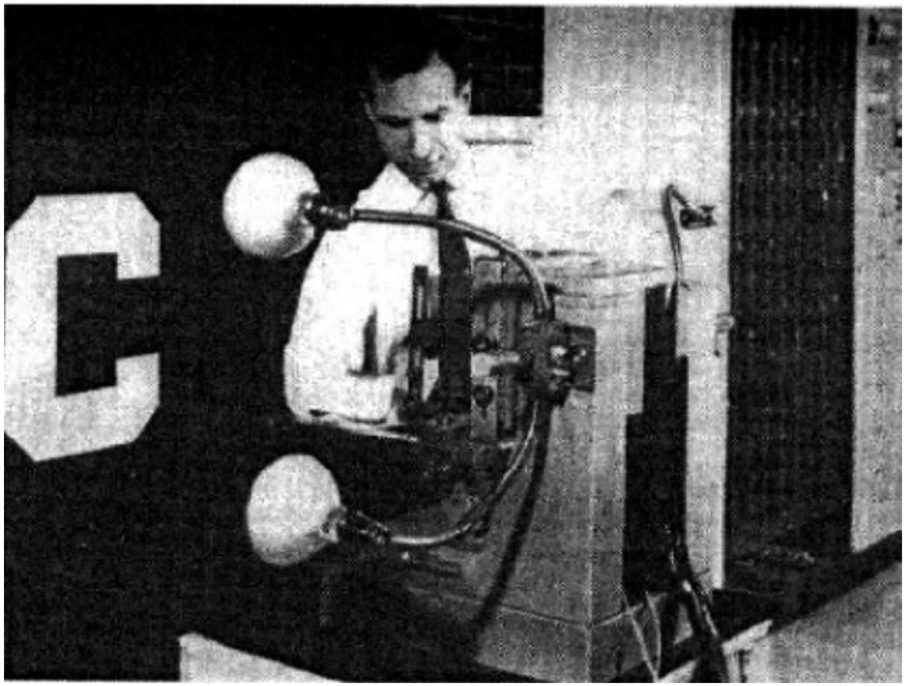
Supervised learning of the weights w using the Perceptron algorithm.

The perceptron is a machine



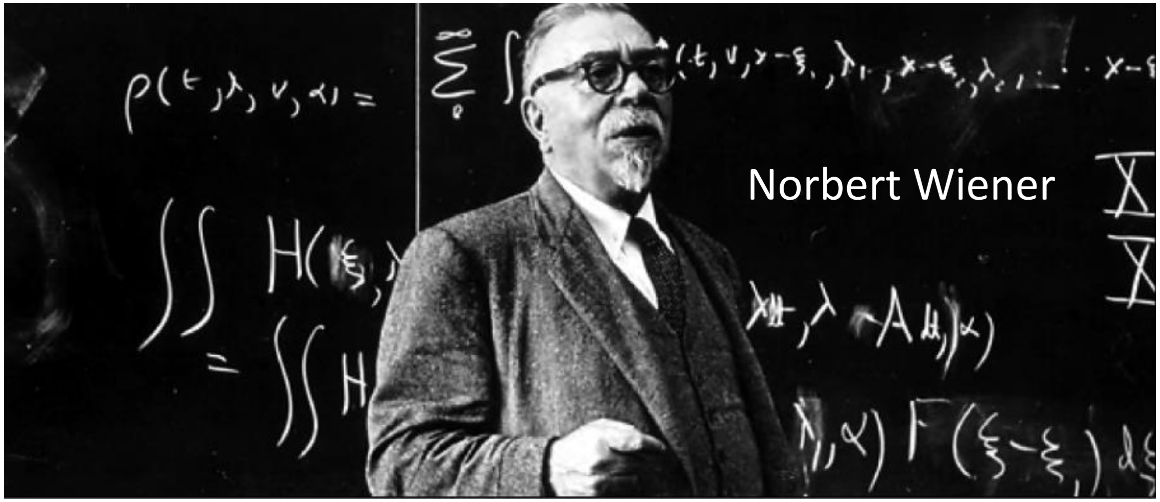


The perceptron



- The perceptron does things that vintage computers could not match.
- Alternative computer architecture? Analog computer?

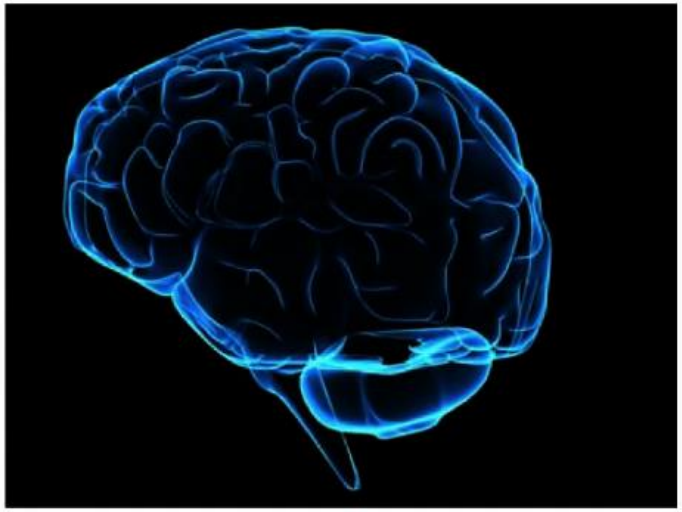
Cybernetics (1948)



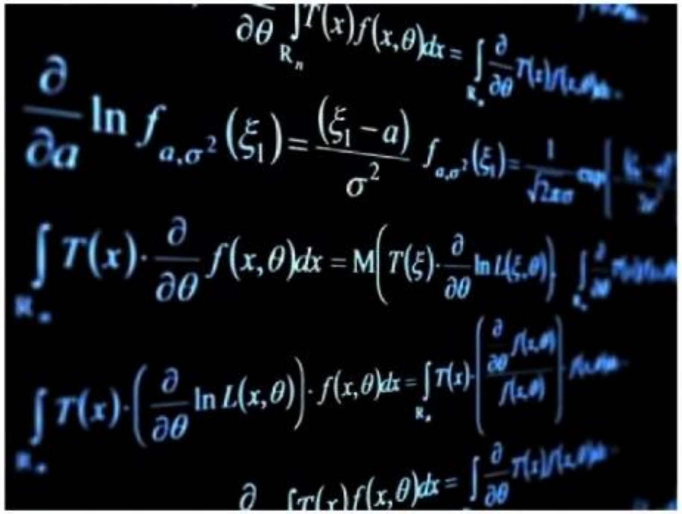
- Mature communication technologies, nascent computing technologies
- Redefining the man-machine boundary

How to design computers?

Biological computer



Mathematical computer

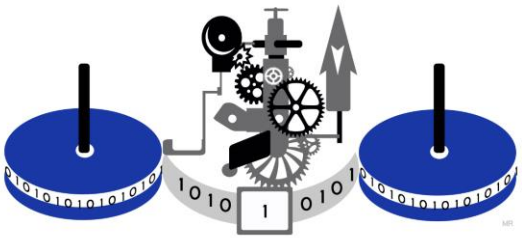


- Which model to emulate : brain or mathematical logic ?
- Mathematical logic won.

Computing with symbols

General computing machines

- Turing machine
- von Neumann machine



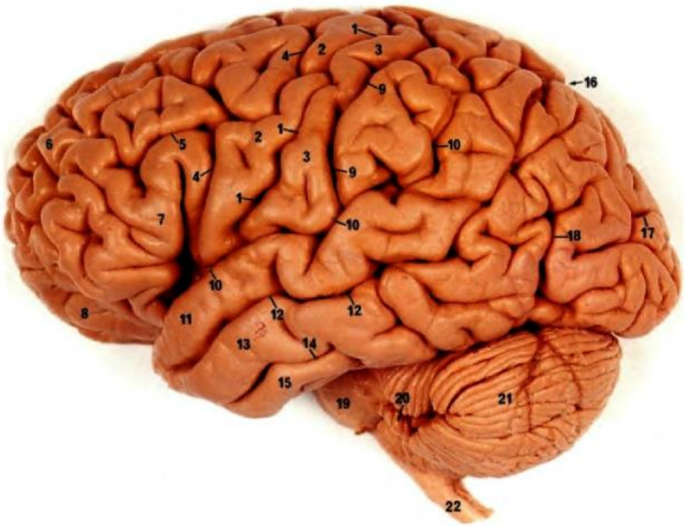
Engineering

- Programming
  (reducing a complex task into
  a collection of simple tasks.)
- Computer language
- Debugging
- Operating systems
- Libraries

Computing with the brain

An engineering perspective

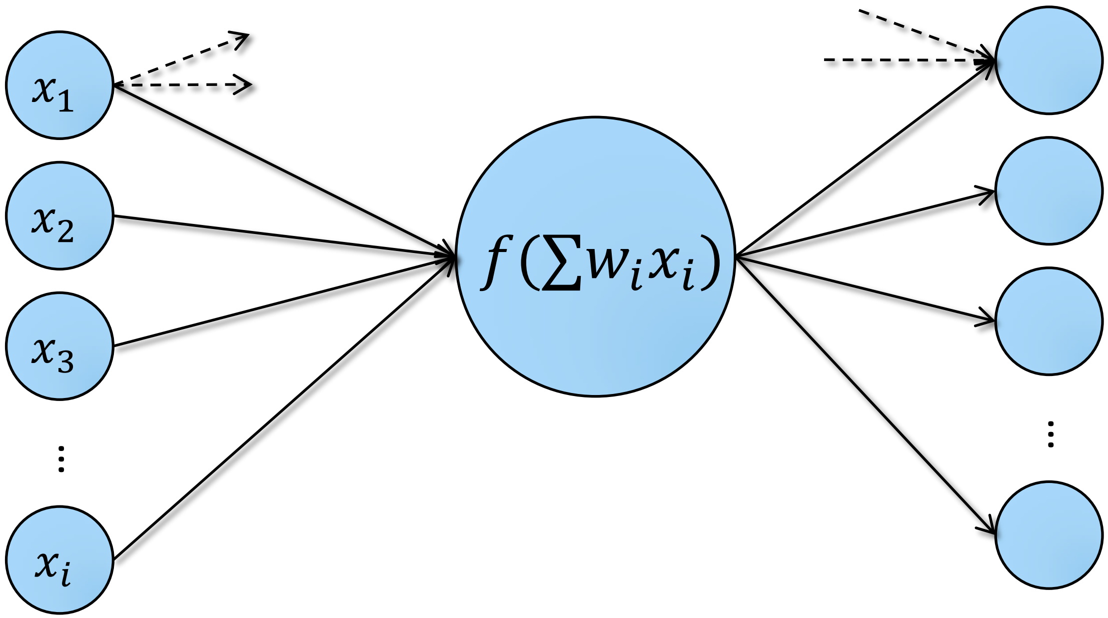
- Compact
- Energy efficient (20 watts)
- 10^{12} Glial cells (power, cooling, support)
- 10^{11} Neurons (soma + wires)
- 10^{14} Connections (synapses)
- Volume = 50% glial cells + 50% wires.



General computing machine?

- Slow for mathematical logic, arithmetic, etc.
- Very fast for vision, speech, language, social interactions, etc.
- Evolution: vision - language – logic.

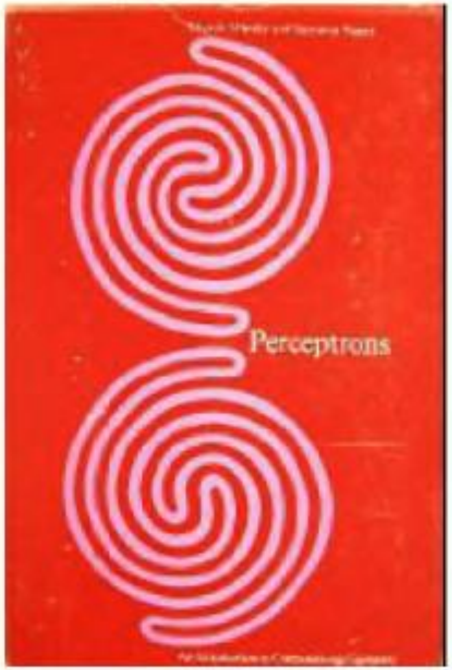
McCulloch & Pitts (1943)



- A simplified neuron model: the Linear Threshold Unit.

Perceptrons (1968)

- Linear threshold units as Boolean gates.
- Circuit theory is poorly known.
- Learning deep circuits means
  solving the credit assignment problem.
- Linearly separable problems are few.
- Elementary problems need complex circuits.
  (parity, connexity, invariances.)
- But have simple algorithmic solutions.
  (programming versus learning.)



→ Abandon perceptrons and other analog computers.
→ Develop symbolic computers and symbolic AI techniques.

Perceptrons revisited

- Linear threshold units as Boolean gates
- Circuit theory is poorly known.
- Learning deep circuits means solving the credit assignment problem.
- Linearly separable problems are few.
- Elementary problems need complex circuits. (parity, connexity, invariances.)
- But have simple algorithmic solutions. (programming versus learning.)

Still true.

Easier than expected but still puzzling.

Low VCdim is good!

Humans do not always do this well either

Learning is necessary when specs are not available.

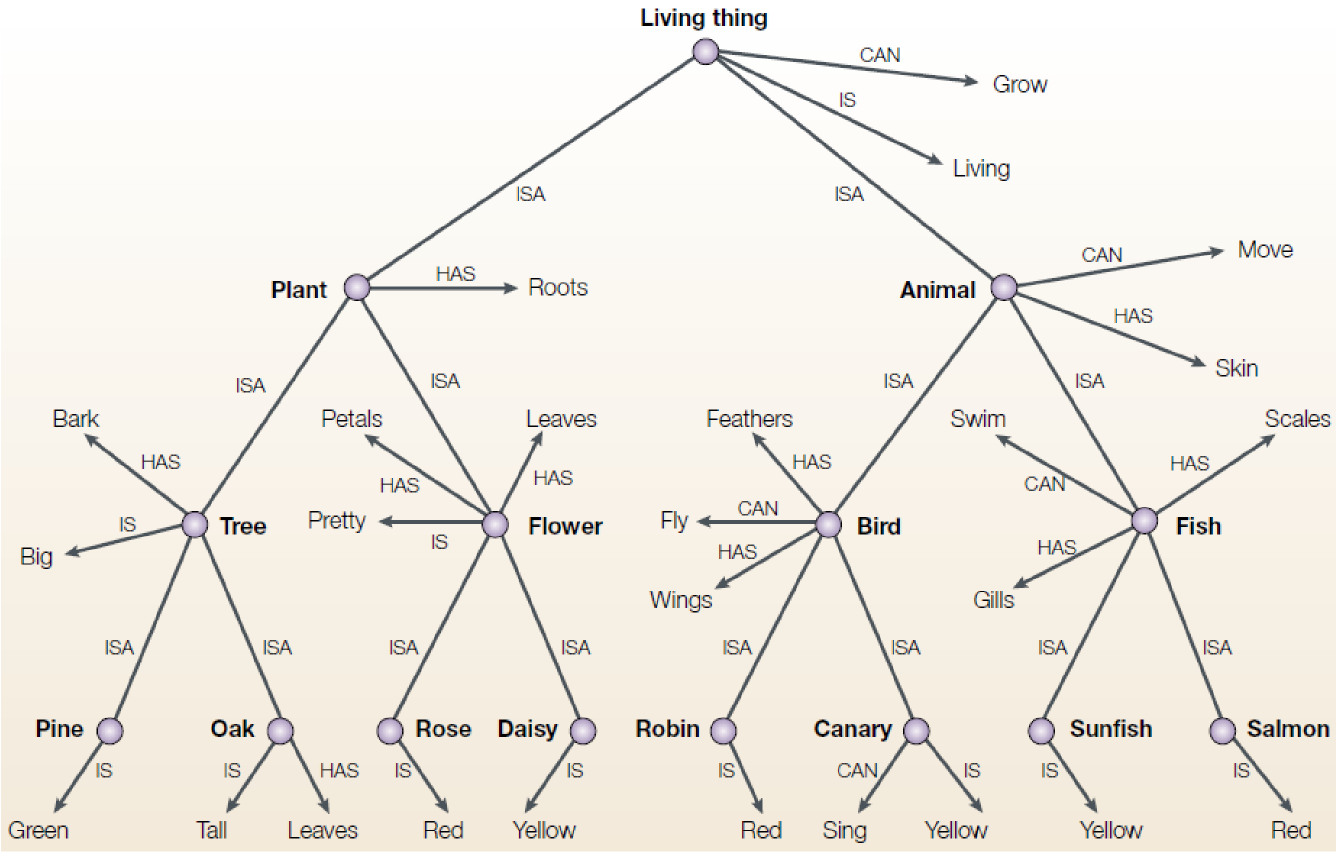
How to reduce complex learning problems into simple ones?

Neural Information Processing

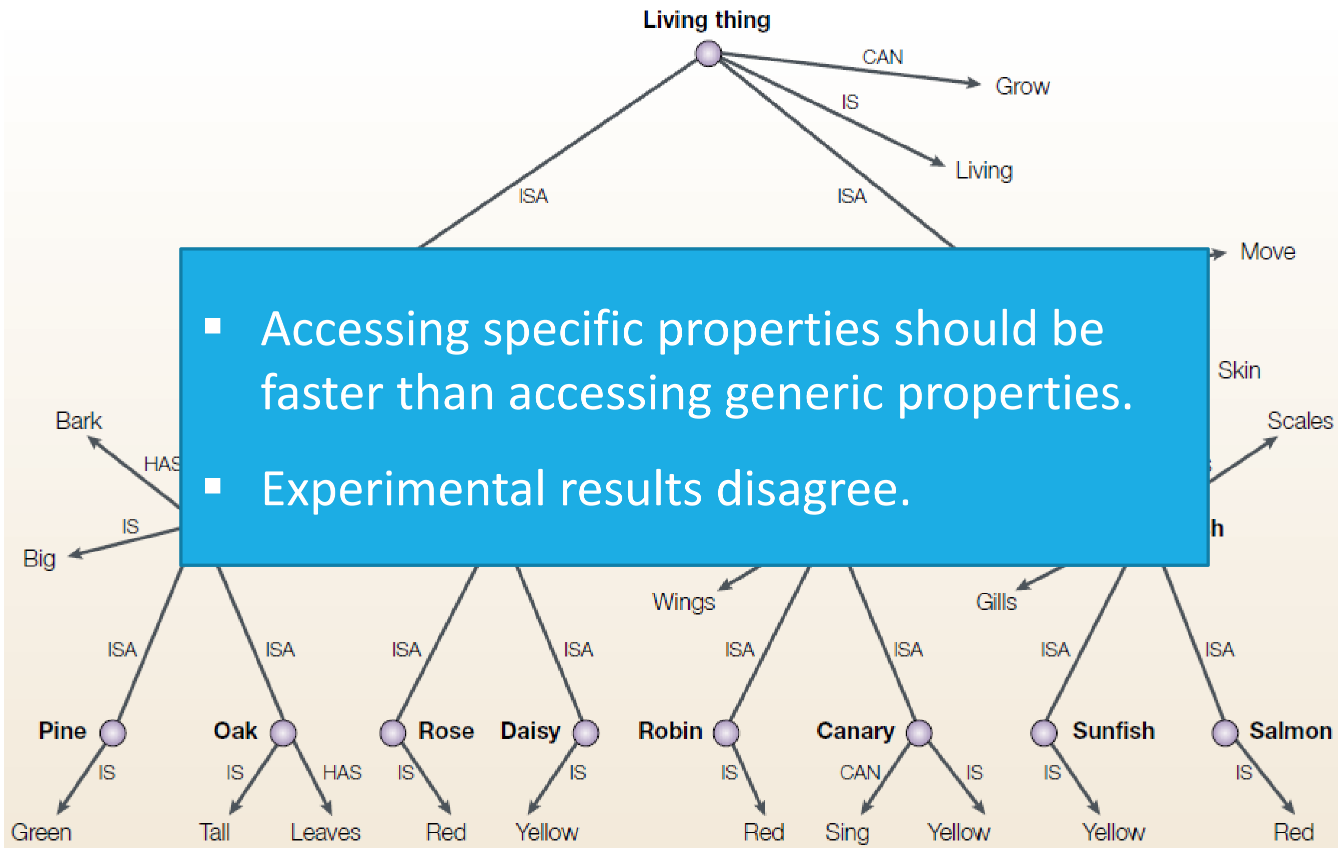
Rumelhart’s propositional network

(see McClelland and Roger, 2003)

Quillian's hierarchical propositional model (1968)



Quillian's hierarchical propositional model (1968)



- Accessing specific properties should be faster than accessing generic properties.
- Experimental results disagree.

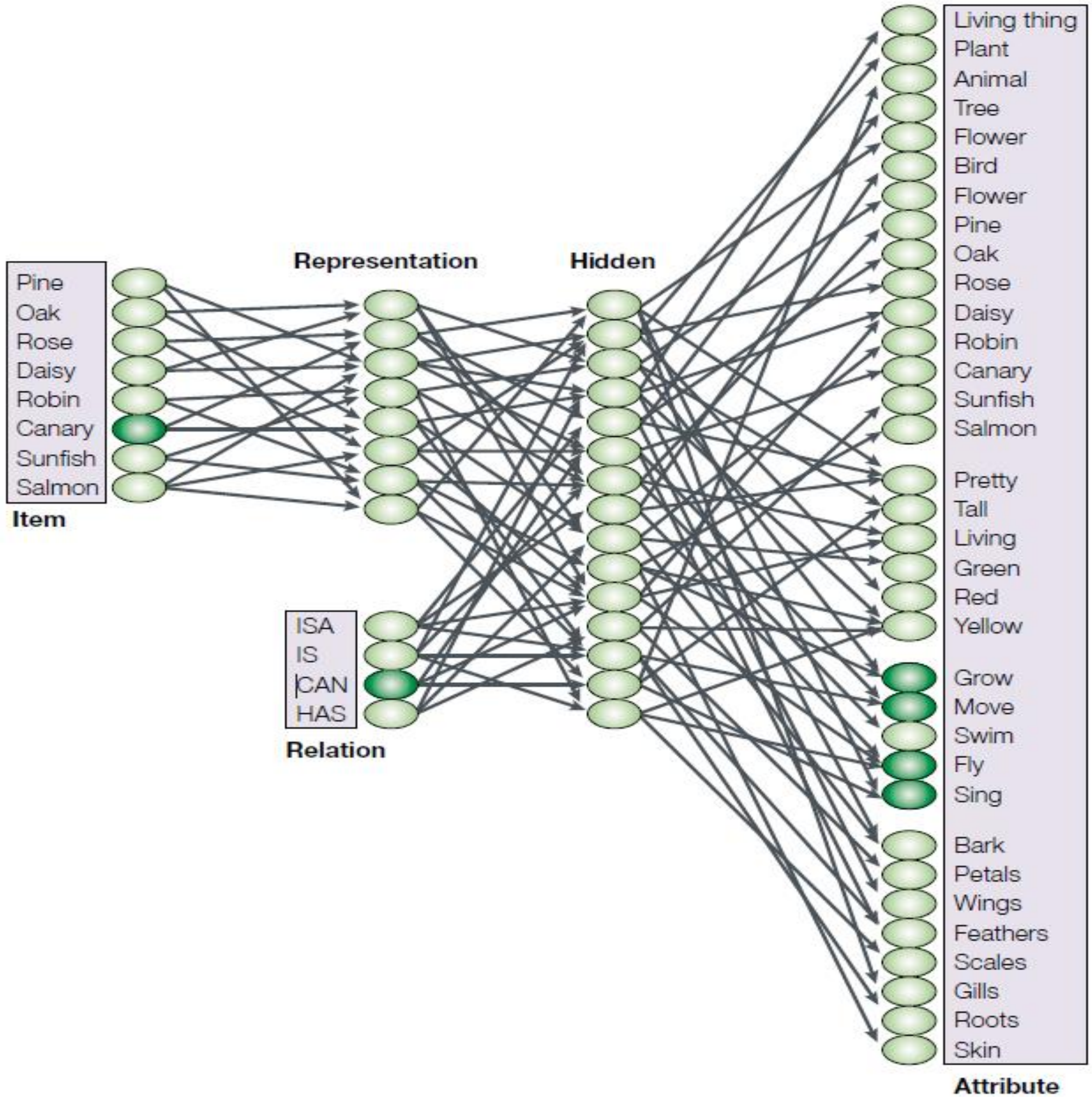
Connectionism

Connectionism

- From psychological ideas of the XIXth and XXth centuries.
- Some see connectionism as a regression (Fodor, Pinker, …)

Parallel Distributed Processing (PDP) Research Group (1980s)

- Neural representations are distributed.
- Neural computation is parallel.
- Processing units, connectivity, propagation rule, learning rule.
- Geoff Hinton “*I want to know how the brain works.*”


Representation
Hidden
Pine
Oak
Rose
Daisy
Robin
Canary
Sunfish
Salmon
Item
ISA
IS
CAN
HAS
Relation
Living thing
Plant
Animal
Tree
Flower
Bird
Flower
Pine
Oak
Rose
Daisy
Robin
Canary
Sunfish
Salmon
Pretty
Tall
Living
Green
Red
Yellow
Grow
Move
Swim
Fly
Sing
Bark
Petals
Wings
Feathers
Scales
Gills
Roots
Skin
Attribute

Training the network

- Replace threshold unit by sigmoid unit

$$s = f\left(\sum x_i\, w_i\right)$$

- Collect training examples

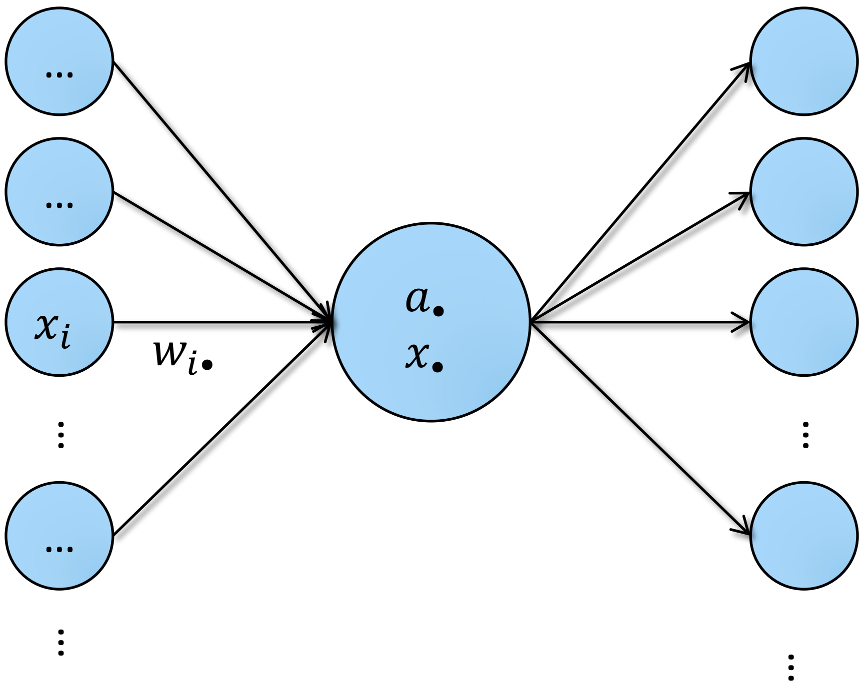
$$\{\ \dots \big(Item(k), Relation(k), DesiredOutput(k)\big)\ \dots\ \}$$

- Form the mean squared error

$$E = \sum_k \left(DesiredOutput(k) - Output(k)\right)^2$$

- Initialize with random weights and optimize by gradient descent (!)

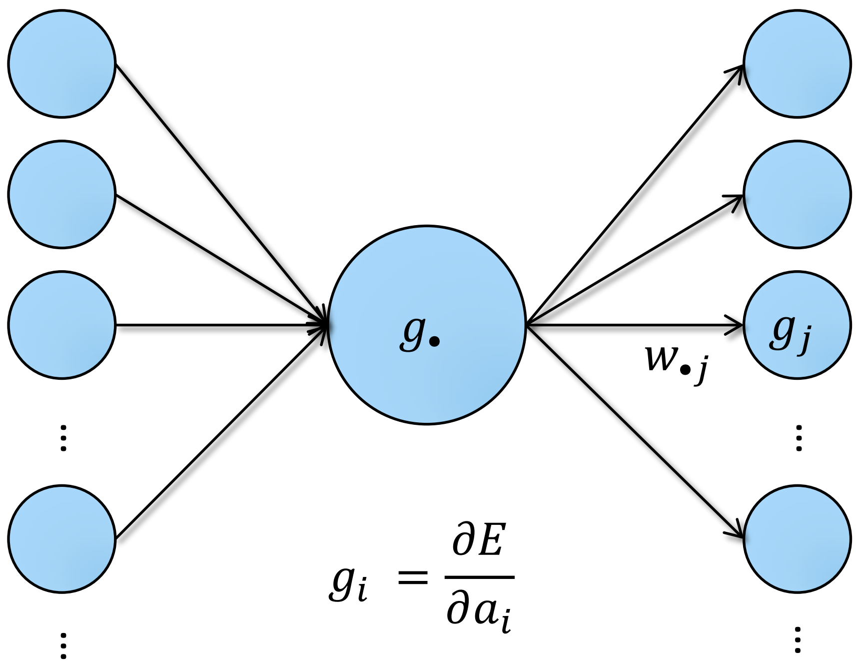
Propagation



$$a_\bullet = \sum_{i \in Pre(\bullet)} w_{i\bullet}\, x_i$$

$$x_\bullet = f(a_\bullet)$$

Back-Propagation



Chain rule

$$g_\bullet = f'(a_\bullet) \sum_{j \in Post(\bullet)} w_{\bullet j} \, g_j$$

$$\frac{\partial E}{\partial w_{\bullet j}} = x_\bullet \, g_j$$

Training algorithm (batch)

Repeat

- Clear gradient accumulators $\Delta_{ij} \leftarrow 0$
- For each example k
  - Set inputs as implied by $Item(k)$ and $Relation(k)$
  - Compute all $a_i(k)$ and $x_i(k)$ by propagation
  - For all output units j, compute

    $$g_j(k) = f'\big(a_j(k)\big)\left(x_j(k) - DesiredOutput_j(k)\right)$$
  - Compute all $g_j(k)$ by back-propagation
  - Accumulate $\Delta_{ij} \leftarrow \Delta_{ij} + x_i(k) g_j(k)$
- Perform a gradient update $w_{ij} = w_{ij} - \eta\, \Delta_{ij}$

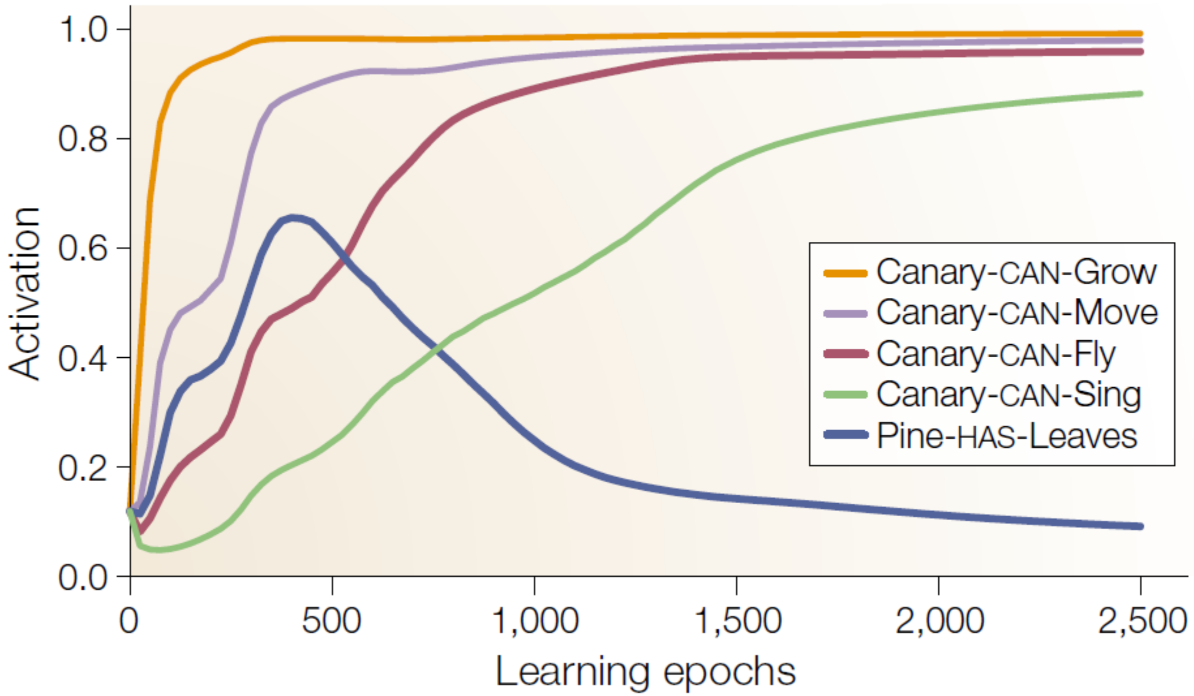
Training algorithm (stochastic)

Repeat

- For each example k
  - Set inputs as implied by $Item(k)$ and $Relation(k)$
  - Compute all $a_i(k)$ and $x_i(k)$ by propagation
  - For all output units j, compute

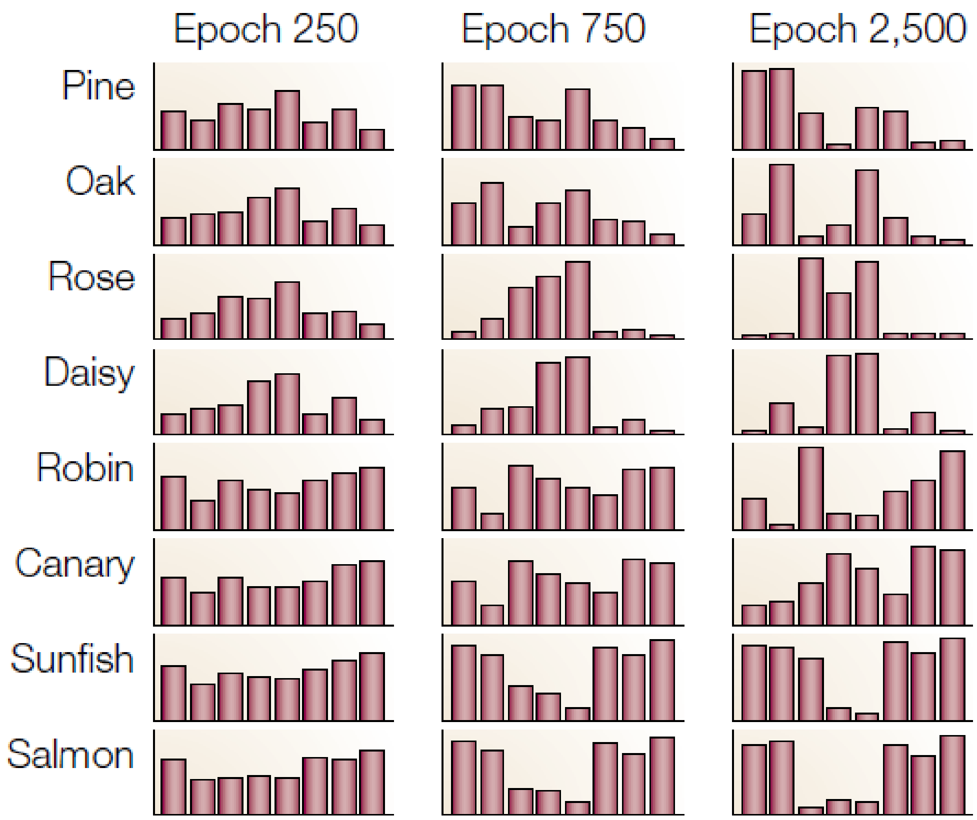
    $$g_j(k) = f'\left(a_j(k)\right)\left(x_j(k) - DesiredOutput_j(k)\right)$$
  - Compute all $g_j(k)$ by back-propagation
  - Set $\Delta_{ij} \leftarrow x_i(k) g_j(k)$
  - Perform a gradient update $w_{ij} = w_{ij} - \eta\, \Delta_{ij}$

Outputs

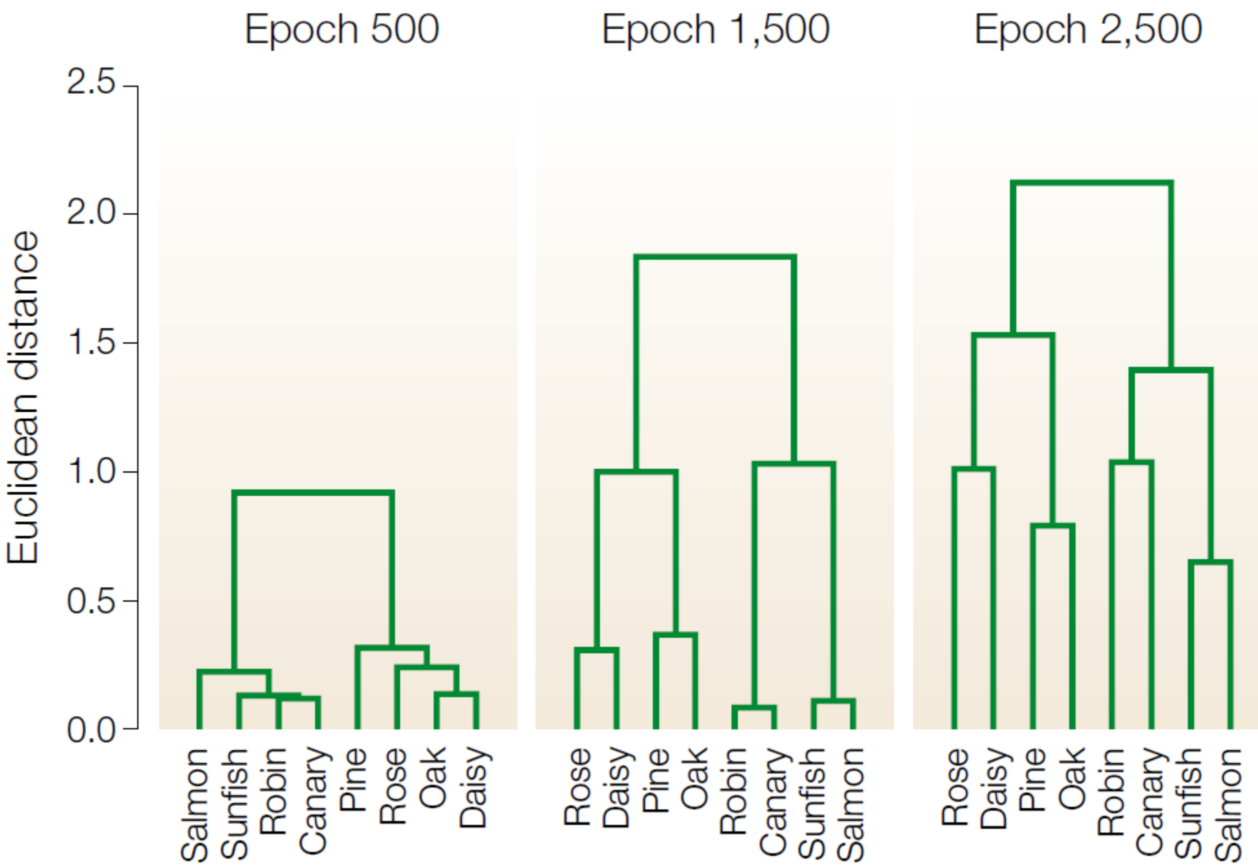


1.0
0.8
0.6
0.4
0.2
0.0
Activation
0
500
1,000
1,500
2,000
2,500
Learning epochs
Canary-CAN-Grow
Canary-CAN-Move
Canary-CAN-Fly
Canary-CAN-Sing
Pine-HAS-Leaves

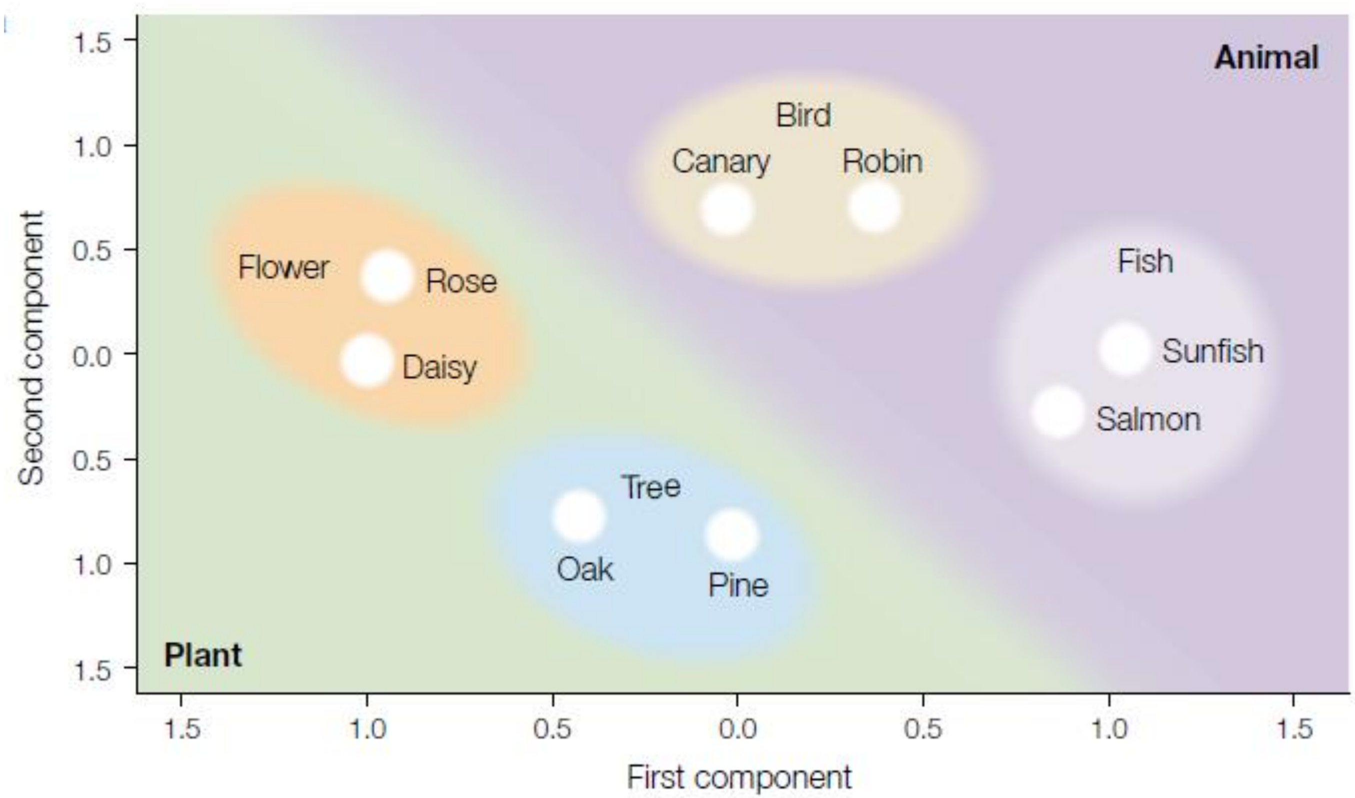
Representations



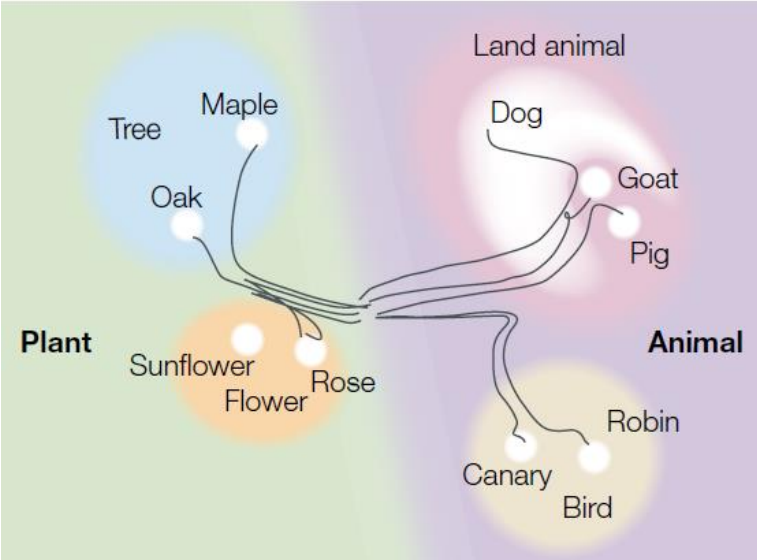
Representations

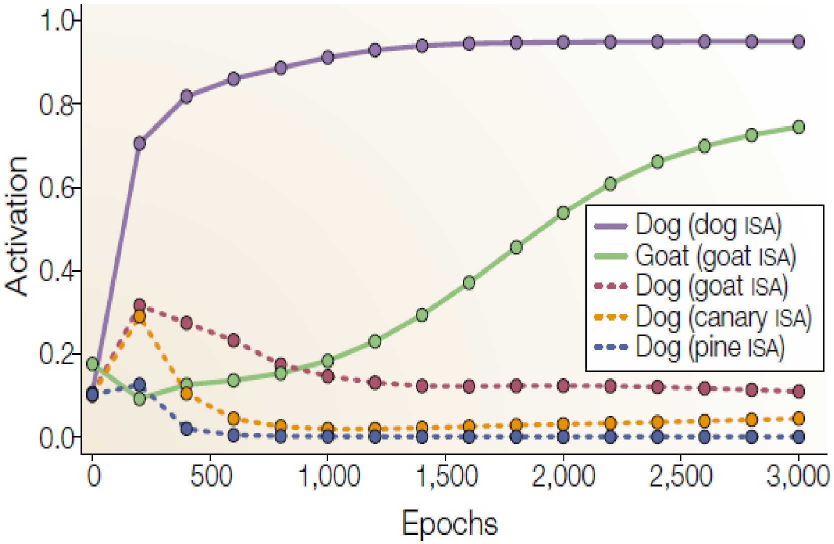


ISA in representation space

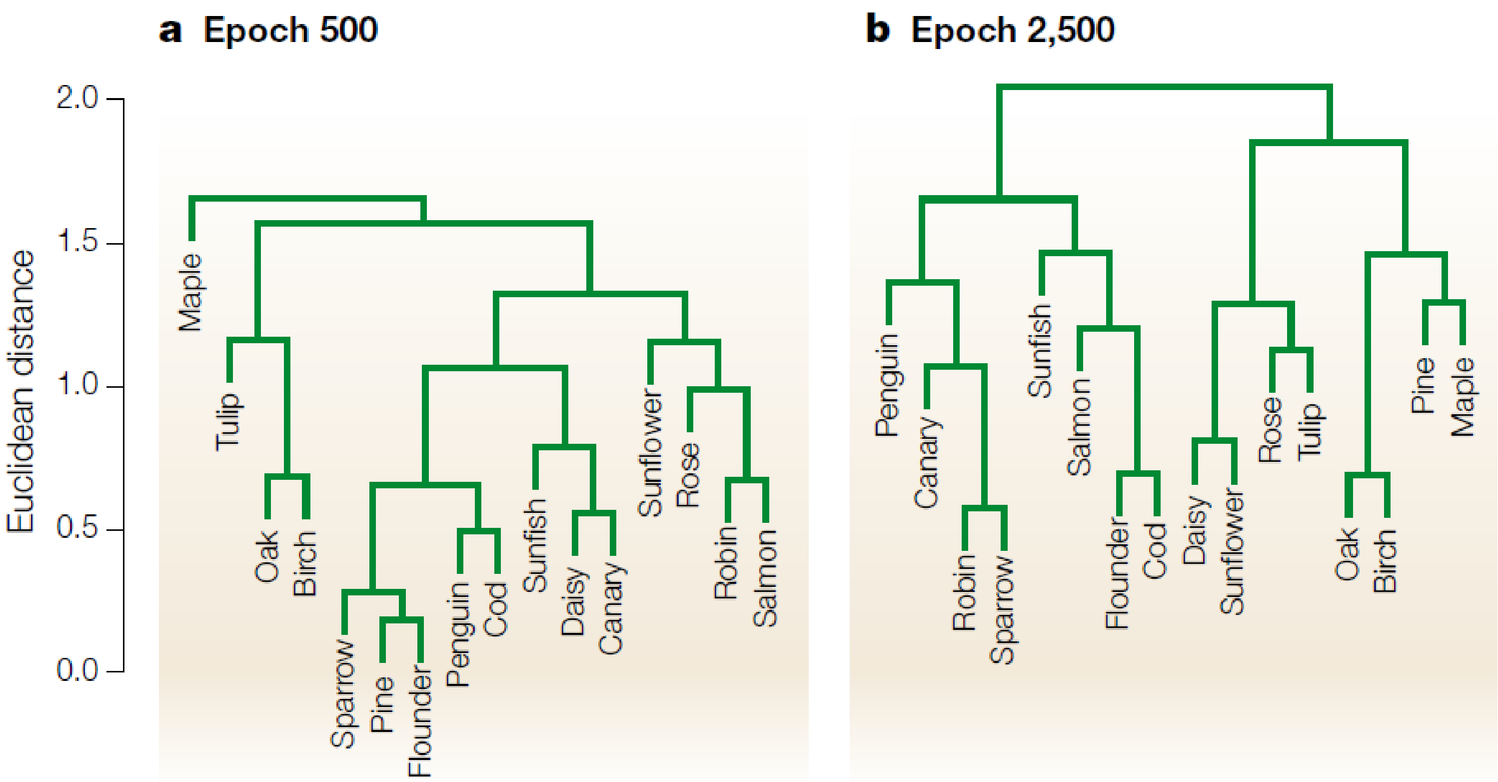


Playing with item frequency





Adding relations

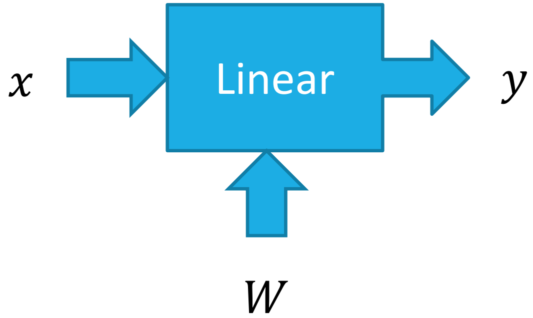


Neural Information Processing

Network construction kit

(B. & Gallinari, 1991)

Linear brick



Propagation

$$y = Wx$$

Back-propagation

$$\frac{\partial E}{\partial x} = \frac{\partial E}{\partial y} W$$

$$\frac{\partial E}{\partial W} = x \frac{\partial E}{\partial y}$$

Transfer function brick

Propagation

$$y_s = f(x_s)$$

Back-propagation

$$\left[\frac{\partial E}{\partial x}\right]_s = \left[\frac{\partial E}{\partial y}\right]_s f'(x_s)$$

Transfer functions

	Propagation	Back-propagation
Sigmoid	$y_s = \frac{1}{1+e^{-x_s}}$	$\left[\frac{\partial E}{\partial x}\right]_s = \left[\frac{\partial E}{\partial y}\right]_s \frac{1}{(1+e^{x_s})(1+e^{-x_s})}$
Tanh	$y_s = \tanh(x_s)$	$\left[\frac{\partial E}{\partial x}\right]_s = \left[\frac{\partial E}{\partial y}\right]_s \frac{1}{\cosh^2 x_s}$
ReLu	$y_s = \max(0, x_s)$	$\left[\frac{\partial E}{\partial x}\right]_s = \left[\frac{\partial E}{\partial y}\right]_s \mathbb{I}\{x_s > 0\}$
Ramp	$y_s = \min(-1, \max(1, x_s))$	$\left[\frac{\partial E}{\partial x}\right]_s = \left[\frac{\partial E}{\partial y}\right]_s \mathbb{I}\{-1 < x_s < 1\}$

Square loss brick

Propagation

$$E = y = \frac{1}{2}(x - d)^2$$

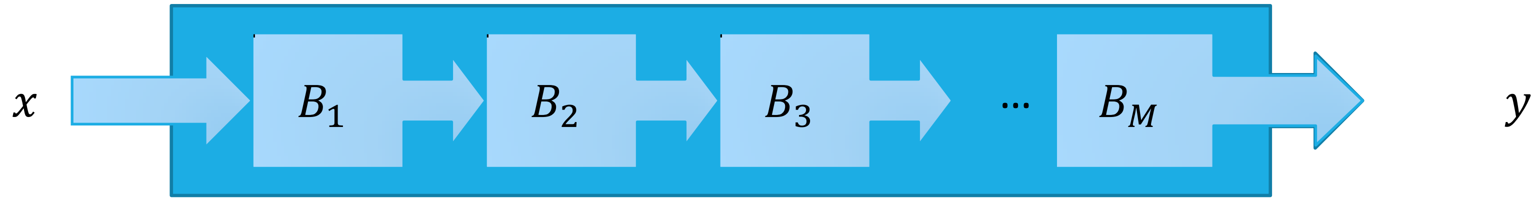
Back-propagation

$$\frac{\partial E}{\partial x} = (x - d)^T \frac{\partial E}{\partial y} = (x - d)^T$$

Loss bricks

		Propagation	Back-propagation
Square		$y = \frac{1}{2}(x-d)^2$	$\frac{\partial E}{\partial x} = (x-d)^T \frac{\partial E}{\partial y}$
Log	$c = \pm 1$	$y = \log(1 + e^{-cx})$	$\frac{\partial E}{\partial x} = \frac{-c}{1+e^{cx}} \frac{\partial E}{\partial y}$
Hinge	$c = \pm 1$	$y = \max(0, m - cx)$	$\frac{\partial E}{\partial x} = -c \ \mathbb{I}\{cx < m\} \frac{\partial E}{\partial y}$
LogSoftMax	$c = 1 \dots k$	$y = \log(\sum_k e^{x_k}) - x_c$	$\left[\frac{\partial E}{\partial x}\right]_s = (e^{x_s}/\sum_k e^{x_k} - \delta_{sc}) \frac{\partial E}{\partial y}$
MaxMargin	$c = 1 \dots k$	$y = \left[\max_{k \neq c}\{x_k + m\} - x_c\right]_+$	$\left[\frac{\partial E}{\partial x}\right]_s = (\delta_{sk^*} - \delta_{sc}) \ \mathbb{I}\{E > 0\} \frac{\partial E}{\partial y}$

Sequential brick



Propagation

- Apply propagation rule to $B_1, B_2, B_3, \ldots, B_M$.

Back-propagation

- Apply back-propagation rule to $B_M, \ldots, B_3, B_2, B_1$.

Benefits

Implementation

- Flexible modular framework
- Many toolkits (Lush, Torch, …)

Testing

- Each brick can be tested separately (finite differences)

Possibilities

- RBF brick, Vector Quantization brick, and more.

Torch code sample

Defining a network

(see http://code.cogbits.com/wiki.)

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

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

criterion = nn.ClassNLLCriterion()
```

Torch code sample

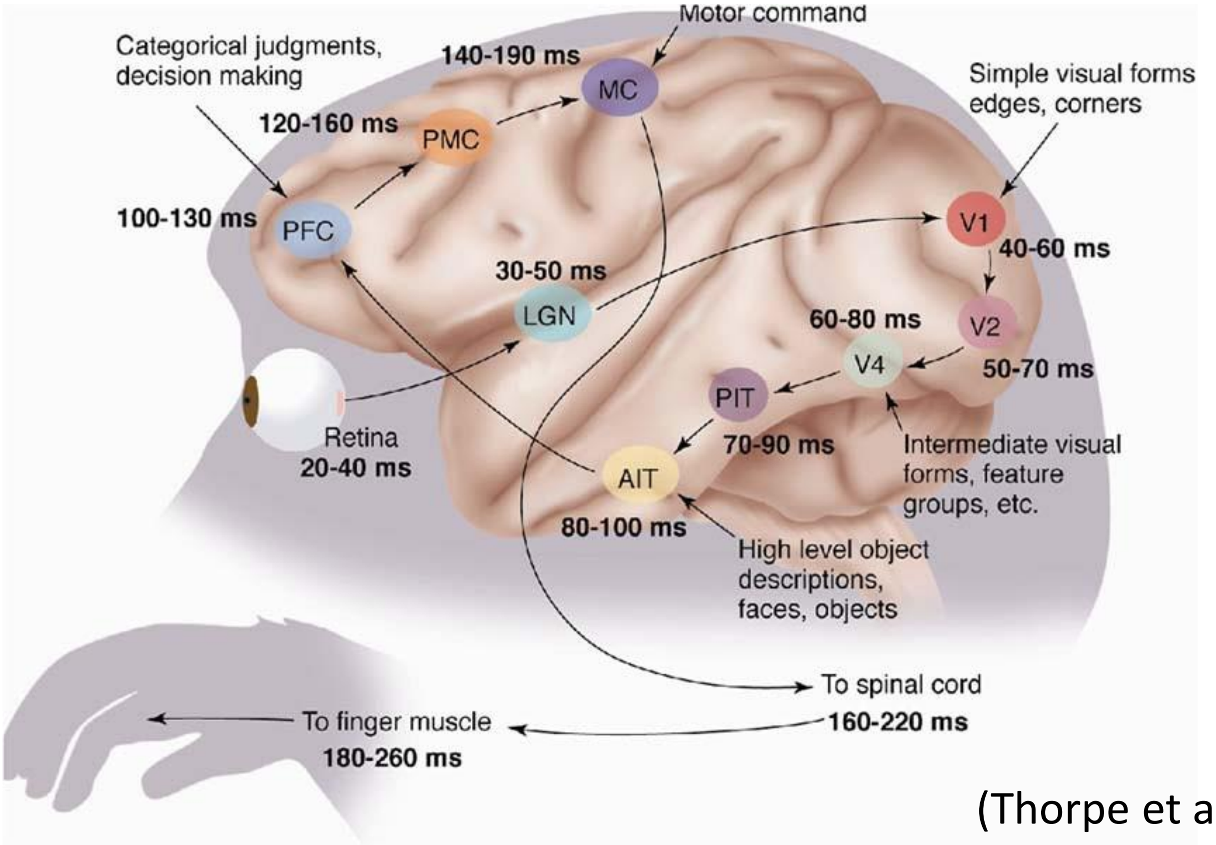
Training the network

```
for t = 1,trainData:size(),batchSize do
   inputs,outputs = getNextBatch()
   -- define closure that computes the gradient
   local feval = function(x)
         parameters:copy(x)
         gradParameters:zero()
         local f = 0
         for i = 1,#inputs do
            local output = model:forward(inputs[i])
            local err = criterion:forward(output,targets[i])
            f = f + err
            local df_do = criterion:backward(output,targets[i])
            model:backward(inputs[i], df_do)
         end
         gradParameters:div(#inputs)
         f = f/#inputs
         return f,gradParameters
      end
   -- perform sgd step on minibatch
   optim.sgd(feval,parameters,optimState)
end
```

Neural Information Processing

Convolutional networks (CNNs)

Vision is fast



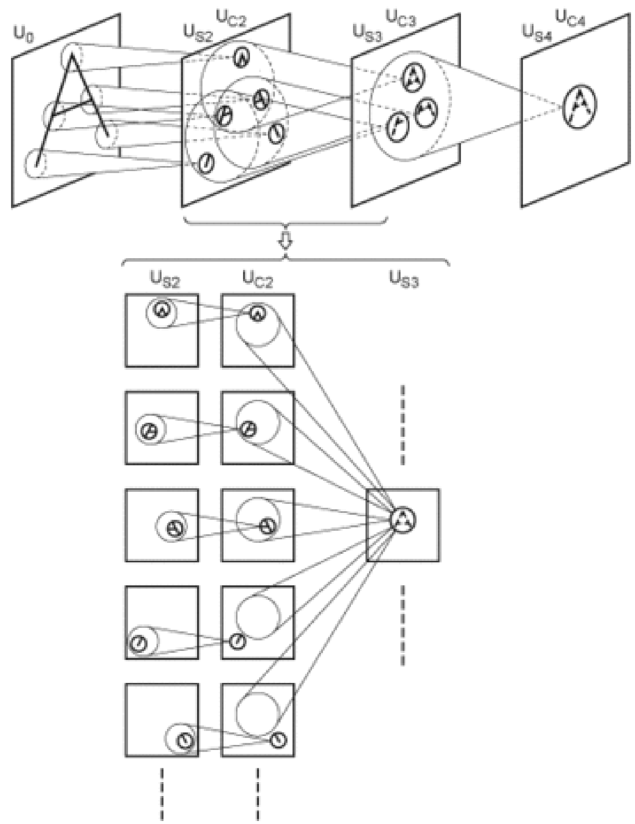
(Thorpe et al., 1995-…)

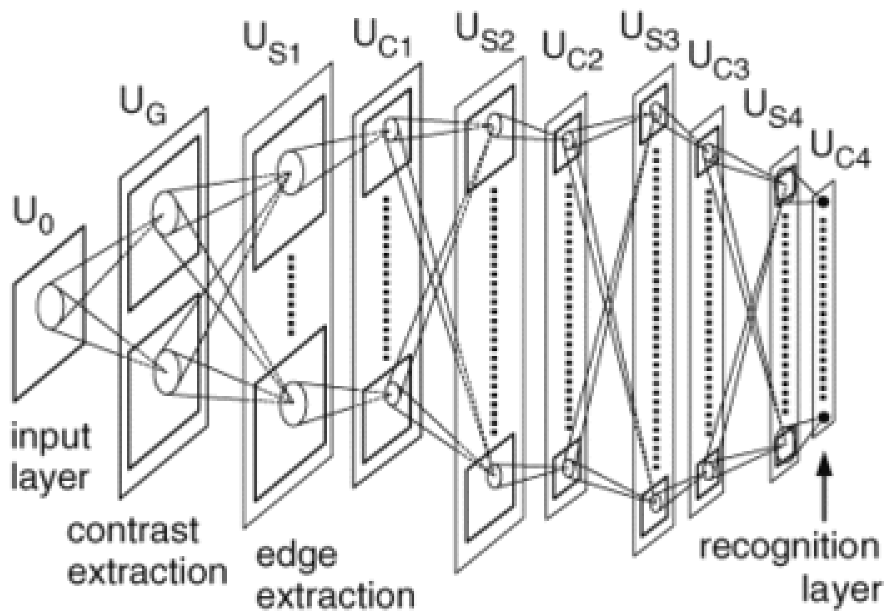
Hubel & Wiesel (1962)

Insights about early image processing in the brain.

- Simple cells detect local features
- Complex cells pool local features in a retinotopic neighborhood

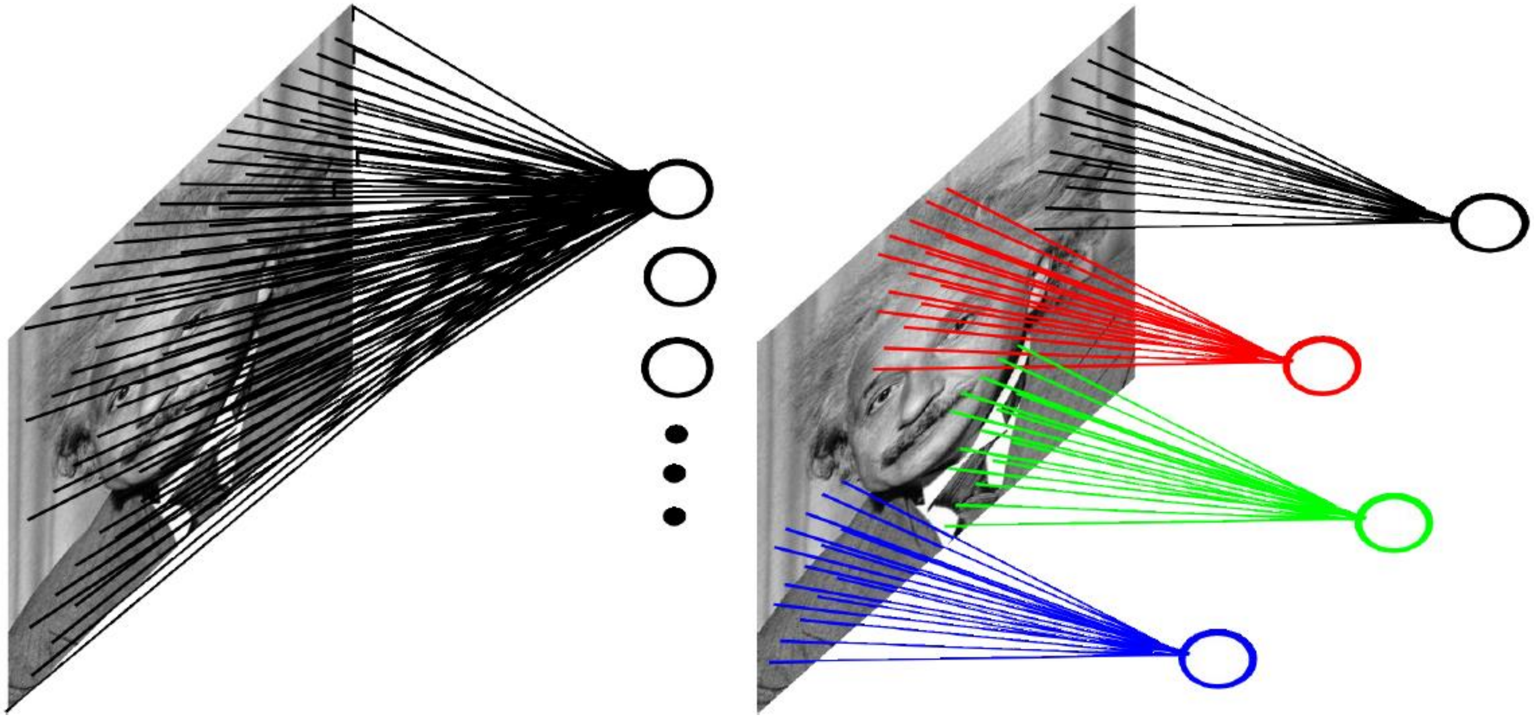
The Neocognitron



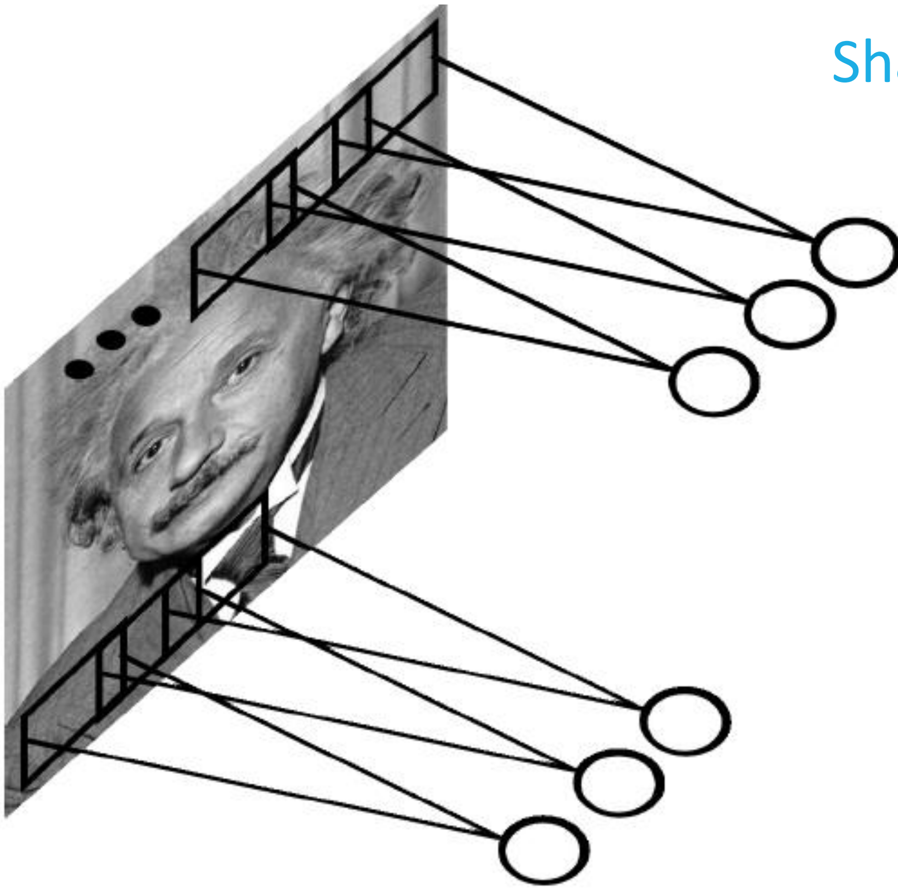


(Fukushima 1974-1982)

Local connections

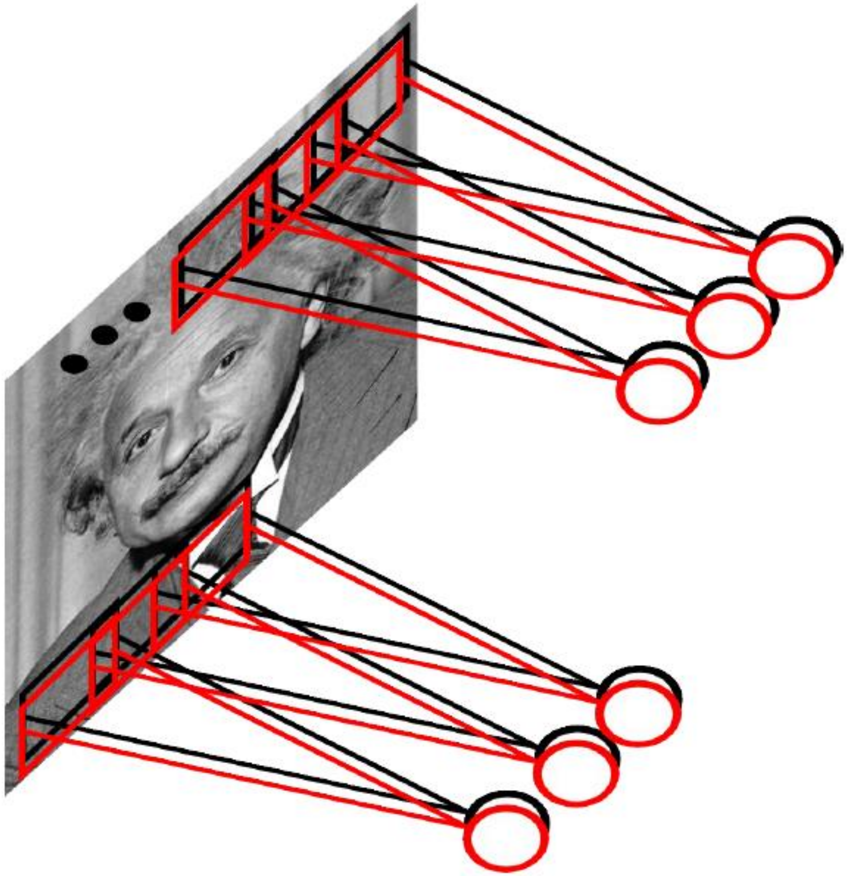


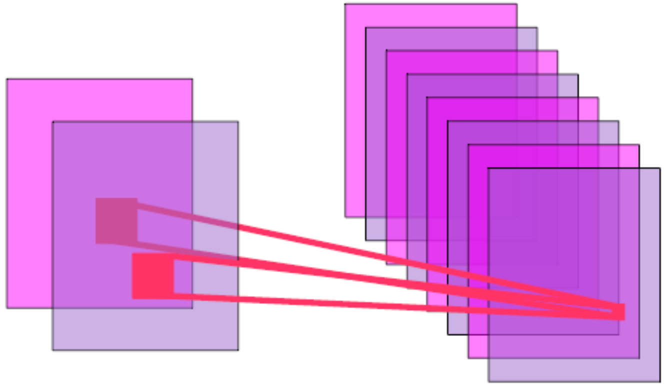
Convolution



Shared weights

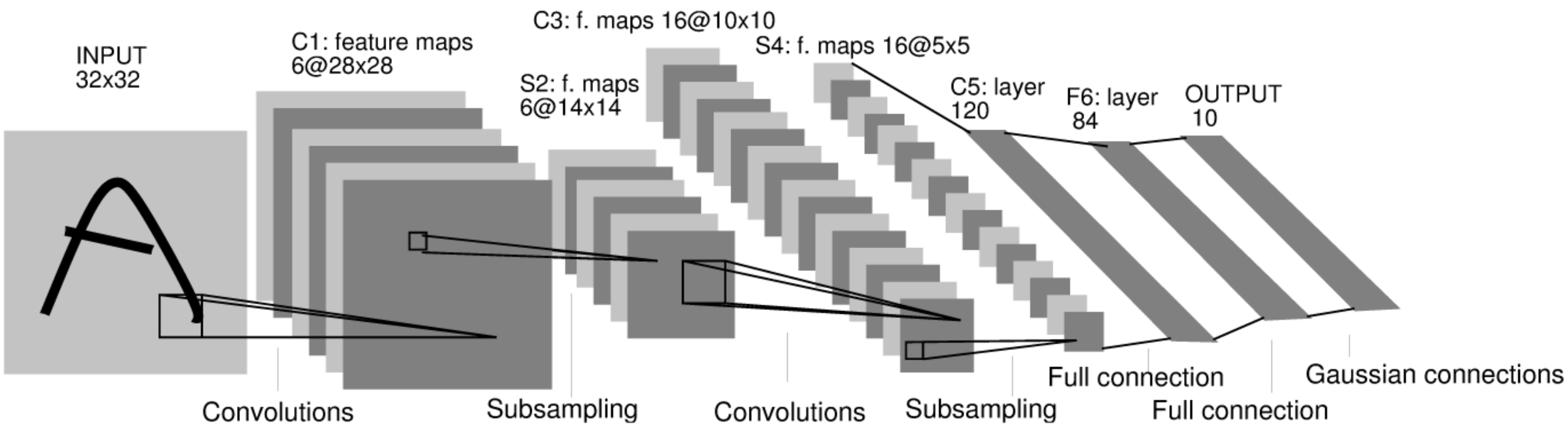
Multiple convolutions



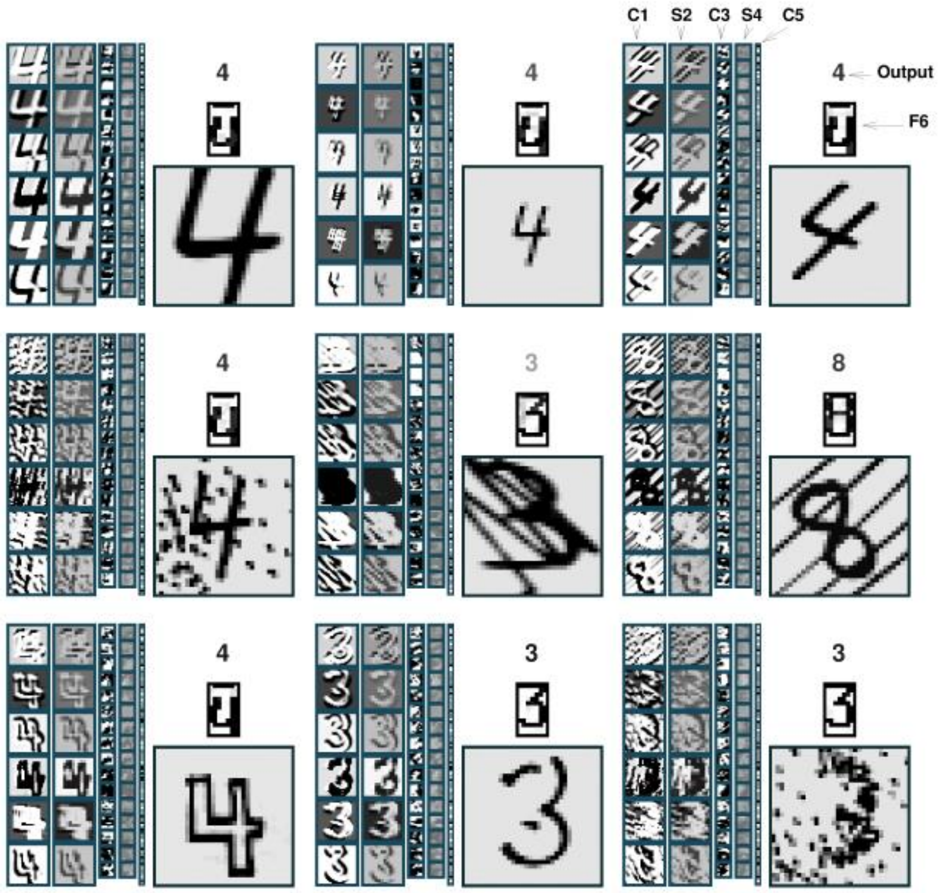


CNNs in the 1990s

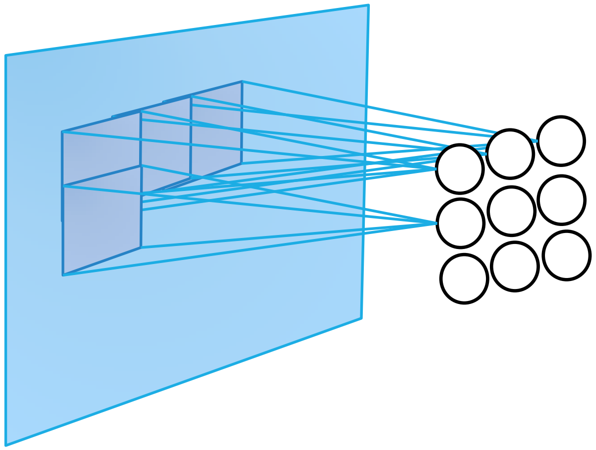
- 1989 Isolated handwritten character recognition (AT&T Bell Labs)
- 1991 Face recognition. Sonar image analysis. (Neuristique)
- 1993 Vehicle recognition. (Onera)
- 1994 Zip code recognition (AT&T Bell Labs)
- 1996 Check reading (AT&T Bell Labs)



Convnets in the 1990s



Pooling



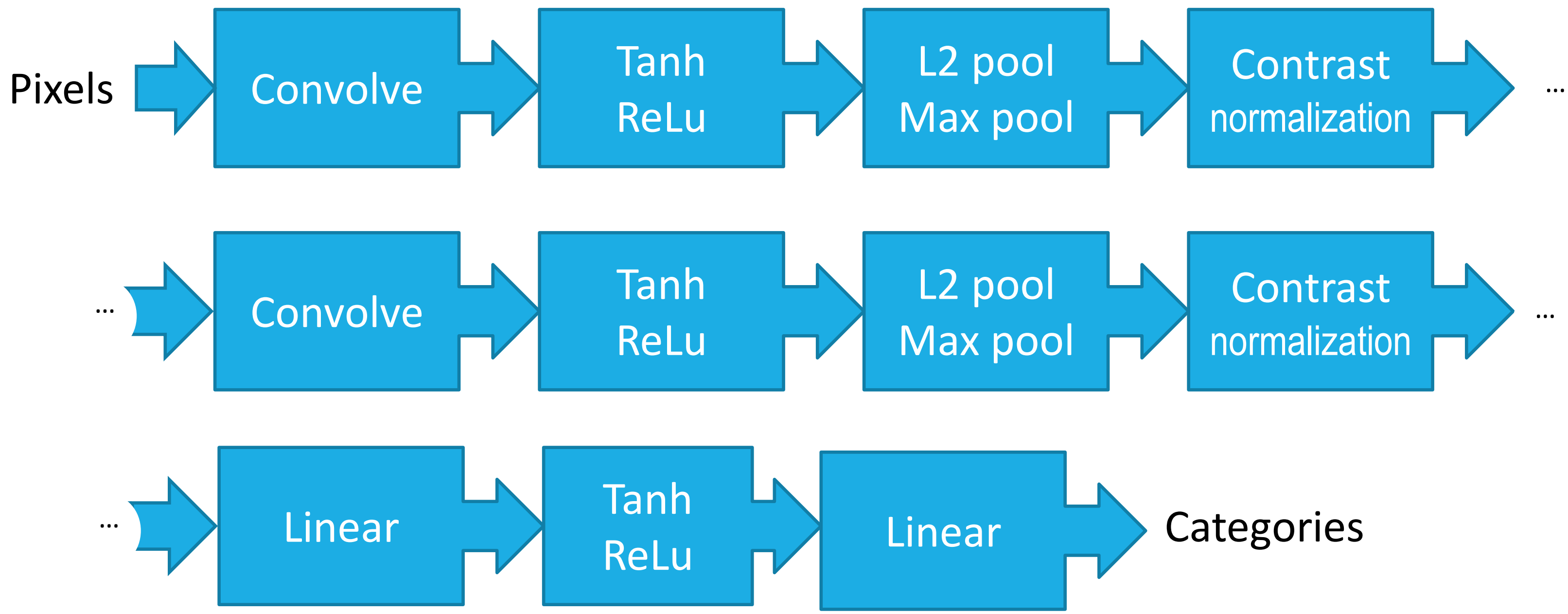
Name	Pooling formula
Average pool	$\frac{1}{s^2}\sum x_i$
Max pool	$\max\{x_i\}$
L2 pool	$\sqrt{\frac{1}{s^2}\sum x_i^2}$
L_p pool	$\left(\frac{1}{s^2}\sum \lvert x_i \rvert^p\right)^{\frac{1}{p}}$

Contrast Normalization

Contrast normalization

- Subtracting a low-pass smoothed version of the layer
- Just another convolution in fact (with fixed coefficients)
- Lots of variants (per feature map, across feature maps, …)
- Divisive normalization

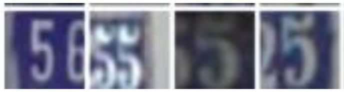
CNNs in the 2010s



Torch code sample

Defining a convolutional network

(see http://code.cogbits.com/wiki.)



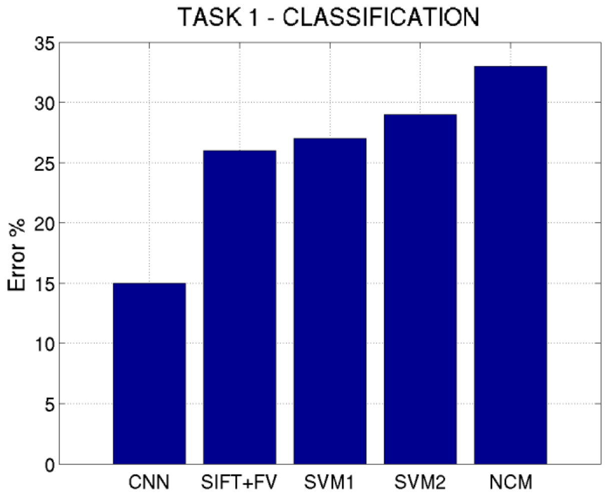
```
nstates = {16,256,128}; fanin = {1,4}; filtsize = 5; poolsize = 2
normkernel = image.gaussian1D(7)

-- Container:
model = nn.Sequential()
-- stage 1 : filter bank -> squashing -> L2 pooling -> normalization
model:add(nn.SpatialConvolutionMap(nn.tables.random(nfeats, nstates[1], fanin[1]), filtsize, filtsize))
model:add(nn.Tanh())
model:add(nn.SpatialLPPooling(nstates[1],2,poolsize,poolsize,poolsize,poolsize))
model:add(nn.SpatialSubtractiveNormalization(16, normkernel))
-- stage 2 : filter bank -> squashing -> L2 pooling -> normalization
model:add(nn.SpatialConvolutionMap(nn.tables.random(nstates[1], nstates[2], fanin[2]), filtsize, filtsize))
model:add(nn.Tanh())
model:add(nn.SpatialLPPooling(nstates[2],2,poolsize,poolsize,poolsize,poolsize))
model:add(nn.SpatialSubtractiveNormalization(nstates[2], normkernel))
-- stage 3 : standard 2-layer neural network
model:add(nn.Reshape(nstates[2]*filtsize*filtsize))
model:add(nn.Linear(nstates[2]*filtsize*filtsize, nstates[3]))
model:add(nn.Tanh())
model:add(nn.Linear(nstates[3], noutputs))
```

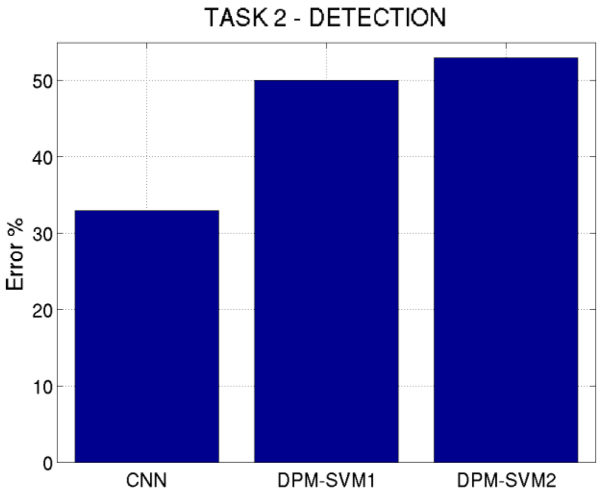
Convnets in the 2000s

- OCR in natural images [2011]. Streetview house numbers (NYU)
- Traffic sign recognition [2011]. GTRSB competition (IDSIA, NYU)
- Pedestrian detection [2013]. INRIA datasets (NYU)
- Volumetric brain segmentation [2009]. Connectomics (MIT)
- Human action recognition [2002,2011]. Smartcatch (NEC), Hollywood II (SF)
- Object recognition [2004,2012]. Norb (NEC), ImageNet (UofT)
- Scene parsing [2010-2012]. Stanford bldg, Barcelona (NEC, NYU)
- Medical image analysis [2008]. Cancer detection (NEC)

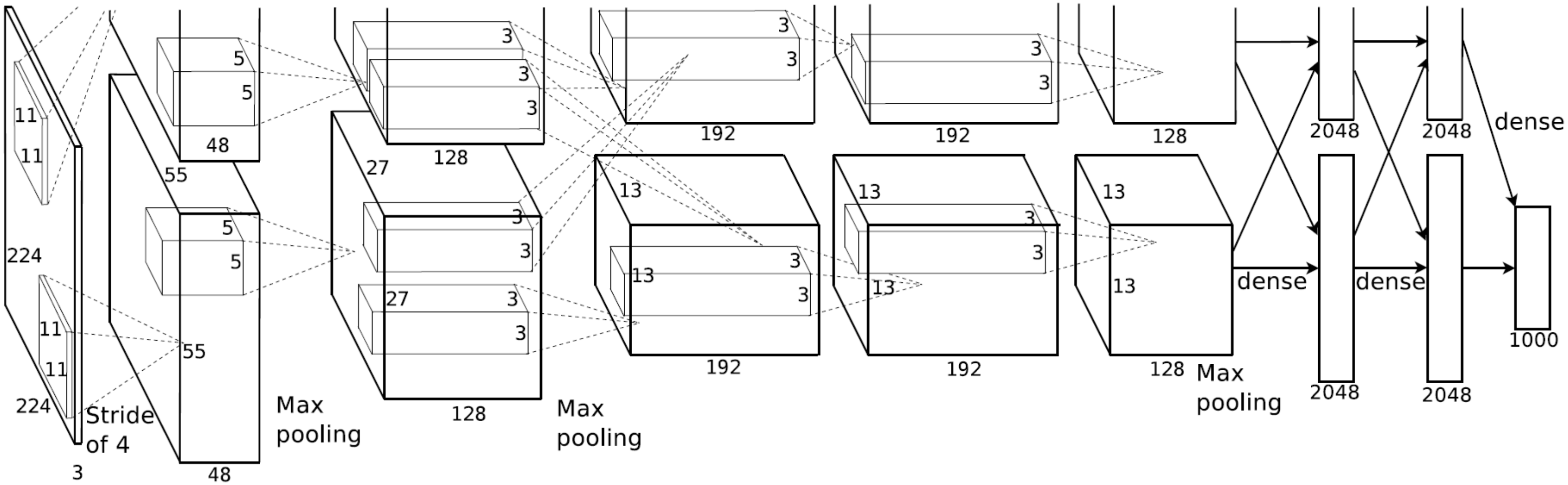
ImageNet 2012 competition

- Object recognition. 1000 categories. 1.2M examples





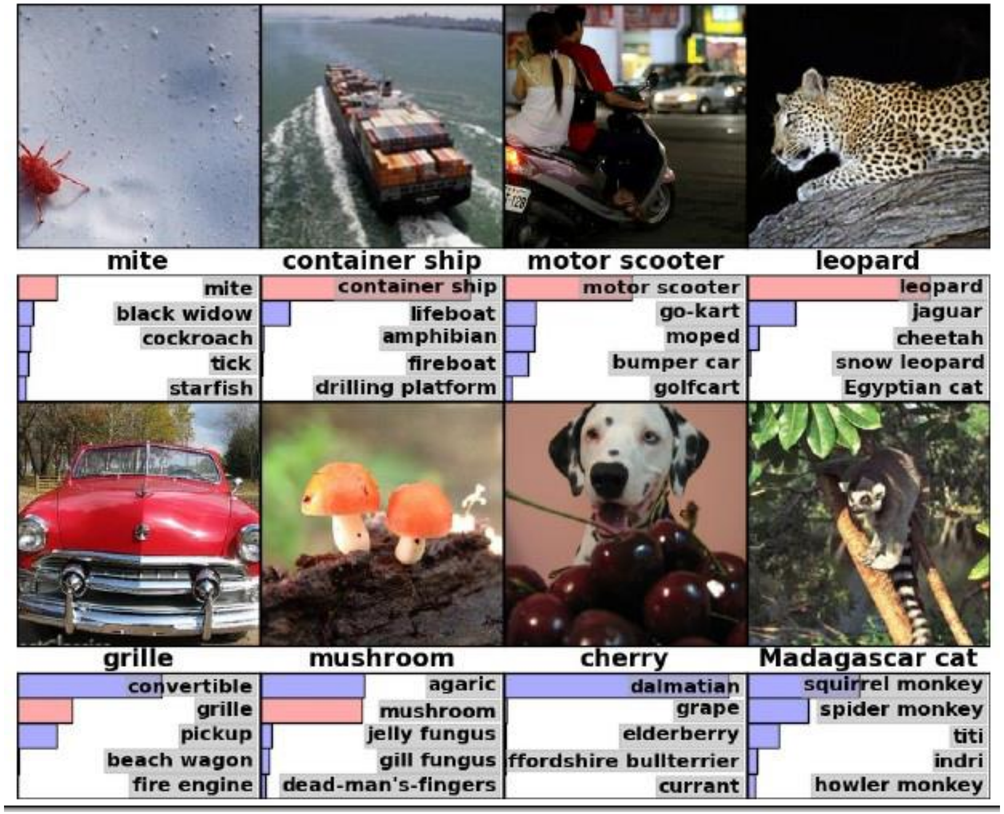
ImageNet CNN



- Structure (conv-relu-maxpool-norm)^3-linear-relu-linear-relu-linear
- Very good implementation, running on two GPUs.
- ReLU transfer function. Dropout trick.
- Also trains on full ImageNet (15M images, 15000 classes)

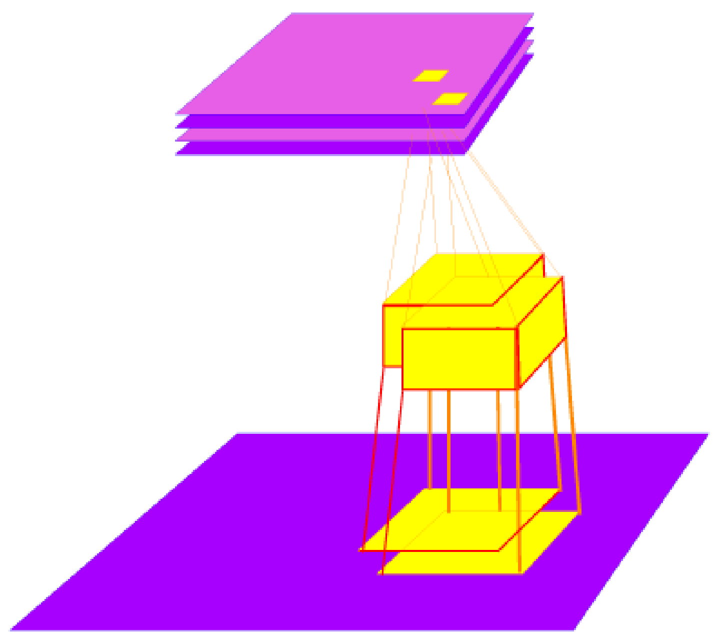
(Kirzhevsky, Sutskever, Hinton, 2012)

ImageNet CNN

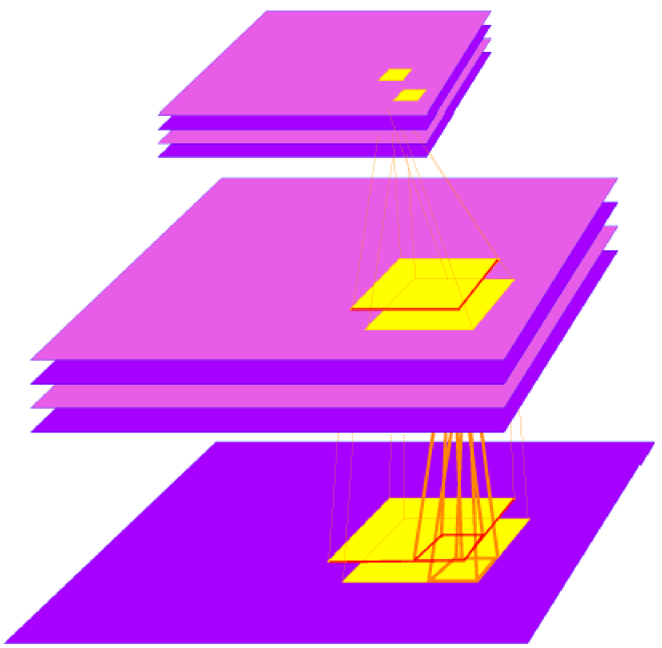


Replicated CNNs

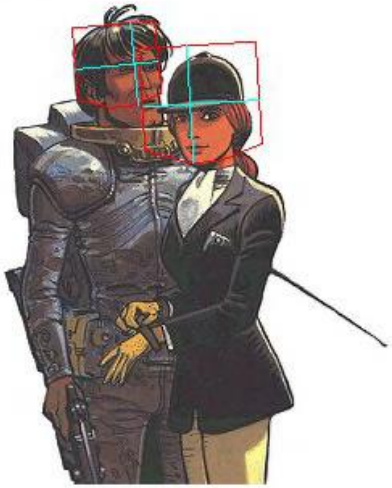
Wrong way



Right way



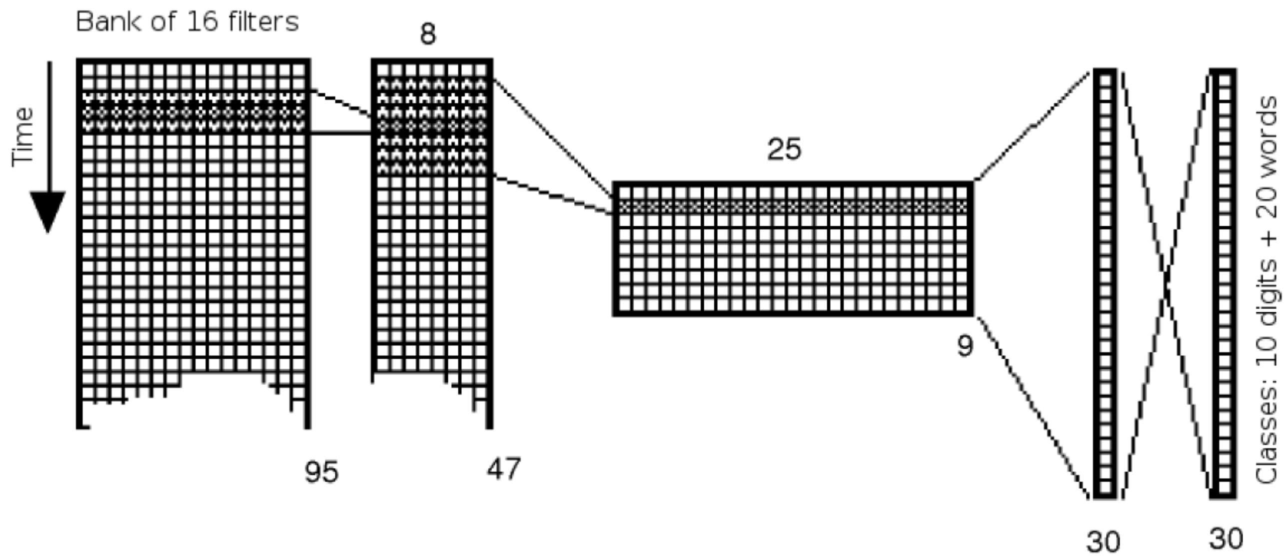




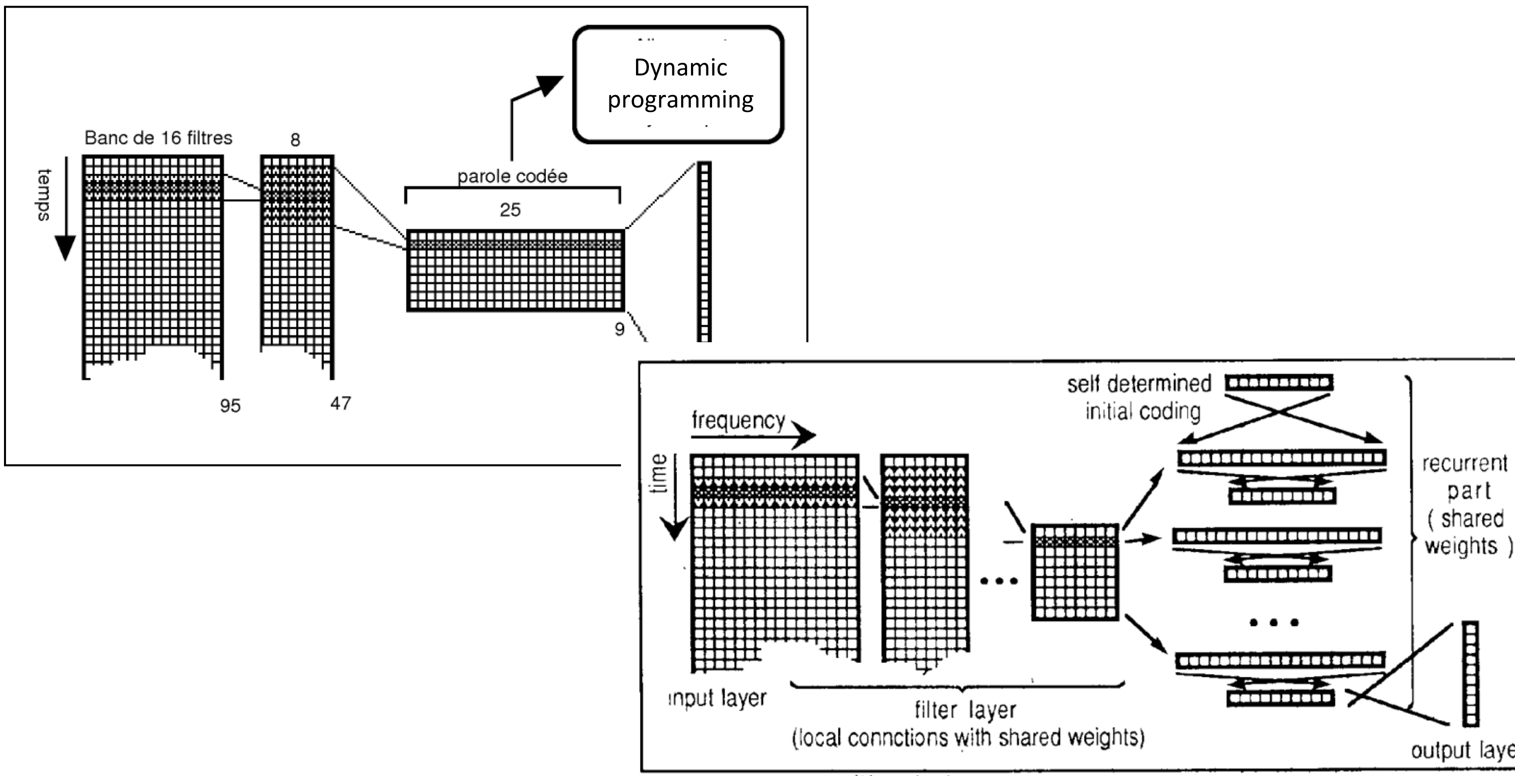
CNNs for speech recognition

Time delay neural networks

- 1988: speaker independent phoneme recognition (Hinton&Lang, Waibel)
- 1989: speaker independent word recognition (B.)
- 1991: continuous speech recognition (Driancourt & B.)



CNN for speech recognition



CNN for speech recognition

In the 1990s

- CNN are competitive with Gaussian Hidden Markov Models.
- But not sufficiently better to justify a switch.

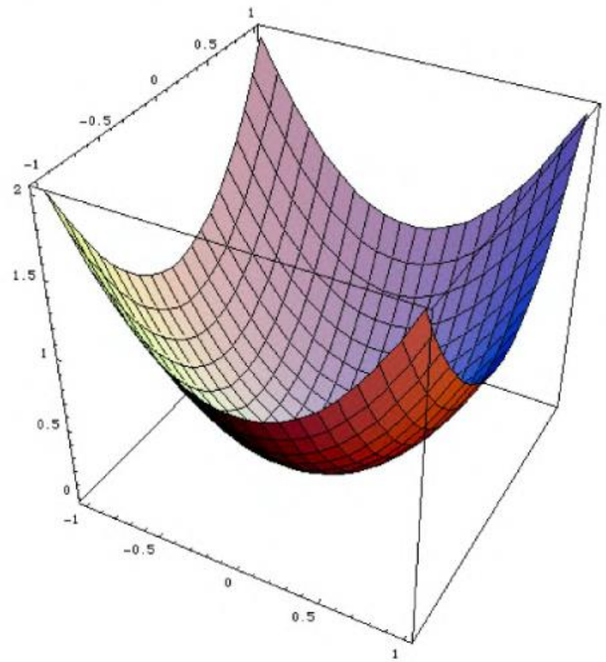
In the 2010s

- More data. More compute power. More results.
- Major speech recognition systems (MS, IBM, Google) have switched to neural network acoustic models around 2011-2012.

Training multilayer networks

Optimization basics

Convex



Definition

$\forall\, x, y,\ \forall\, 0 \leq \lambda \leq 1,$

$f(\lambda x + (1-\lambda)y) \leq \lambda f(x) + (1-\lambda) f(y)$

Property

Any local minimum is a global minimum.

Conclusion

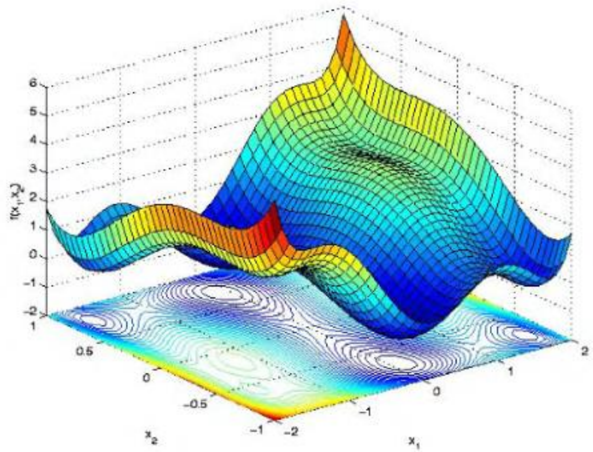
Optimization algorithms are easy to use.
They always return the same solution.

Example: Linear model with convex loss function.

– Curve fitting with mean squared error.

– Linear classification with log-loss or hinge loss.

Non-convex



Landscape

– local minima, saddle points.
– plateaux, ravines, etc.

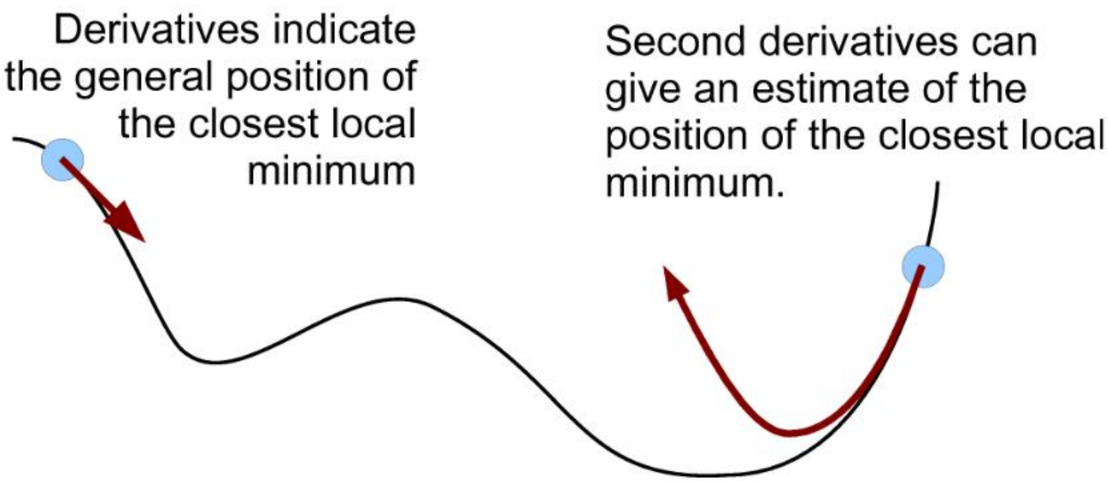
Optimization algorithms

– Usually find local minima.
– Good and bad local minima.
– Result depend on subtle details.

Examples

– Multilayer networks.
– Clustering algorithms.
– Learning features.
– Mixture models.
– Hidden Markov Models.
– Selecting features (some).

Derivatives



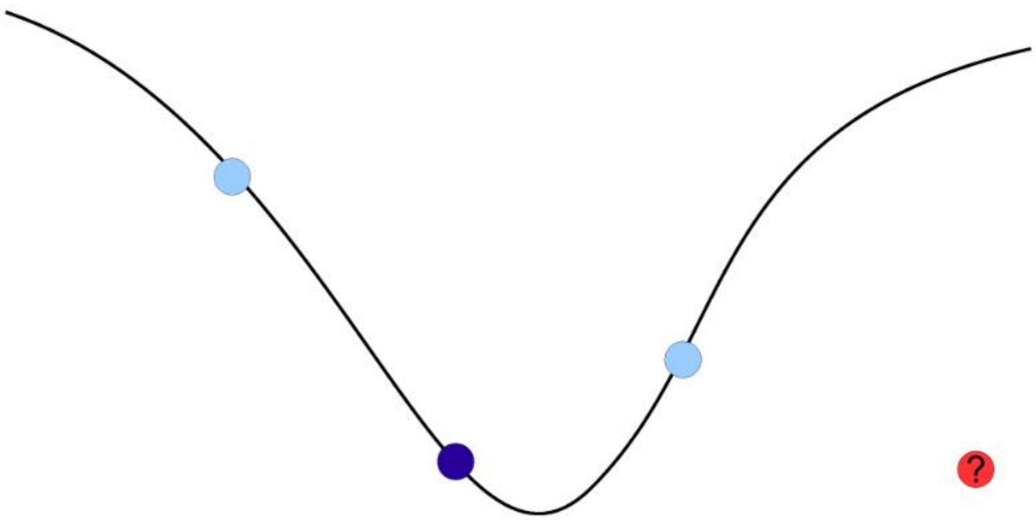
No such local cues without derivatives

– Derivatives may not exist.

– Derivatives may be too costly to compute.

Line search

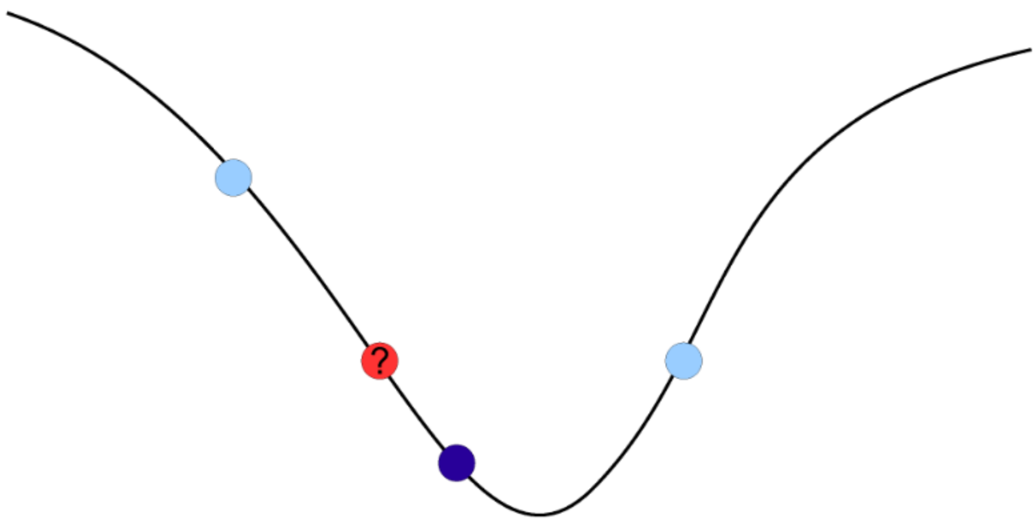
Bracketing a minimum



Three points $a < b < c$ such that $f(b) < f(a)$ and $f(b) < f(c)$.

Line search

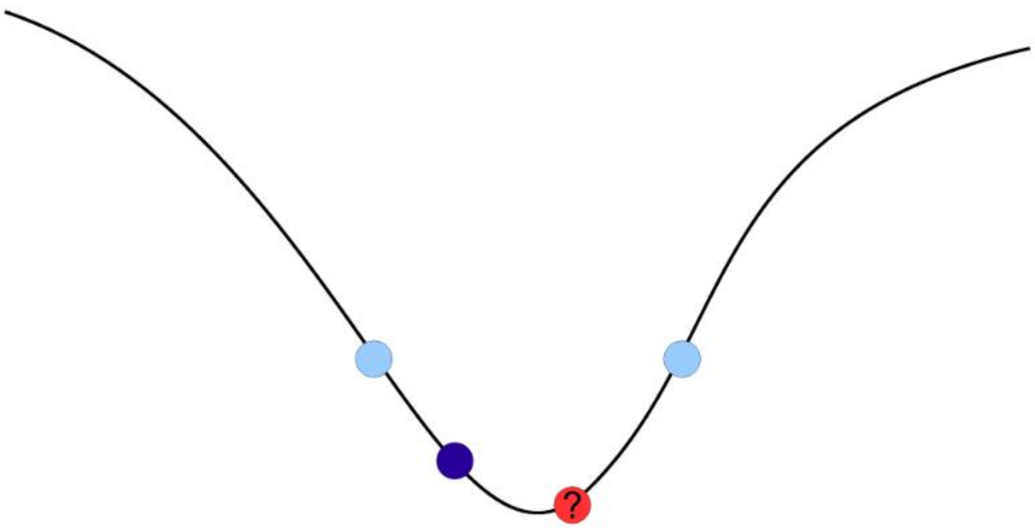
Refining the bracket



Split the largest half and compute $f(x)$.

Line search

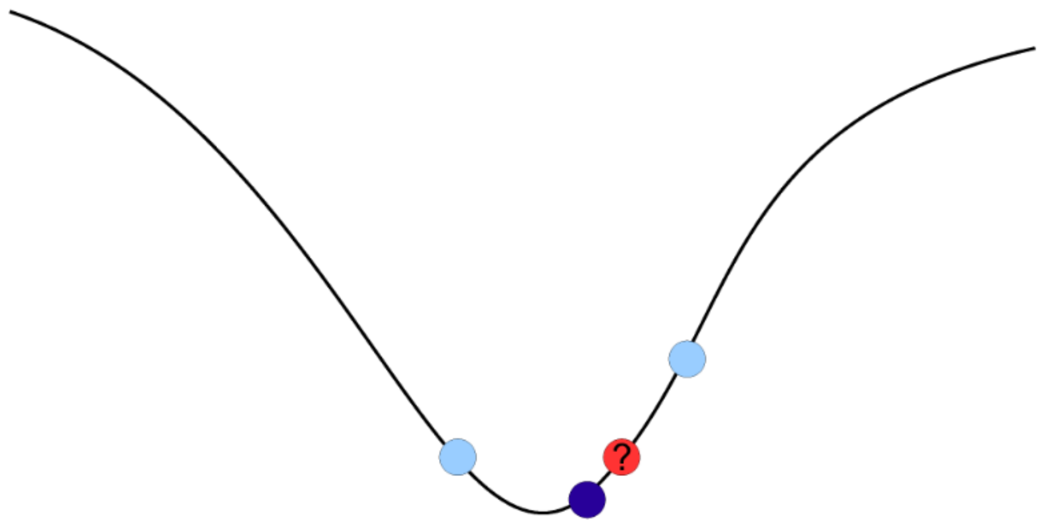
Refining the bracket (2)



– Redefine $a < b < c$. Here $a \leftarrow x$.
– Split the largest half and compute $f(x)$.

Line search

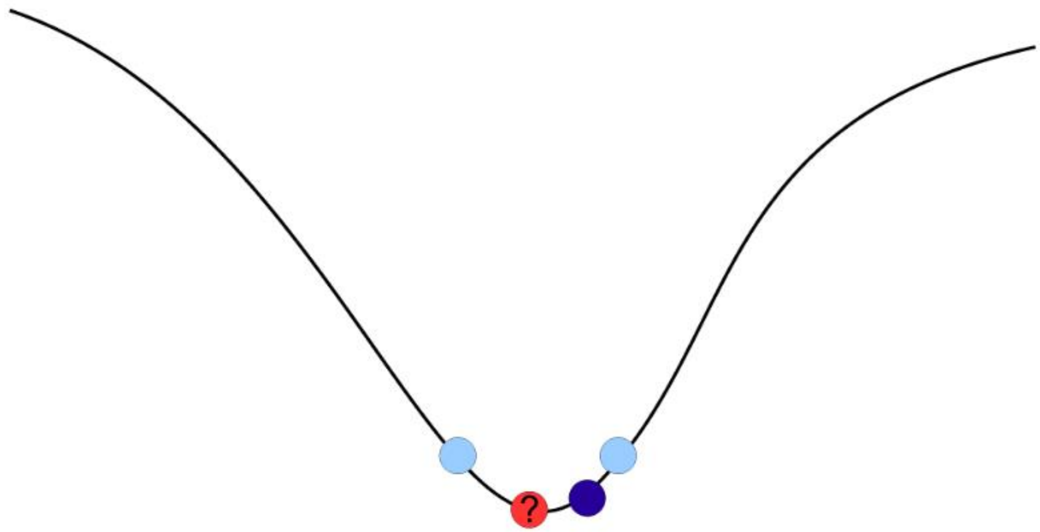
Refining the bracket (3)



- Redefine $a < b < c$. Here $a \leftarrow b$, $b \leftarrow x$.
- Split the largest half and compute $f(x)$.

Line search

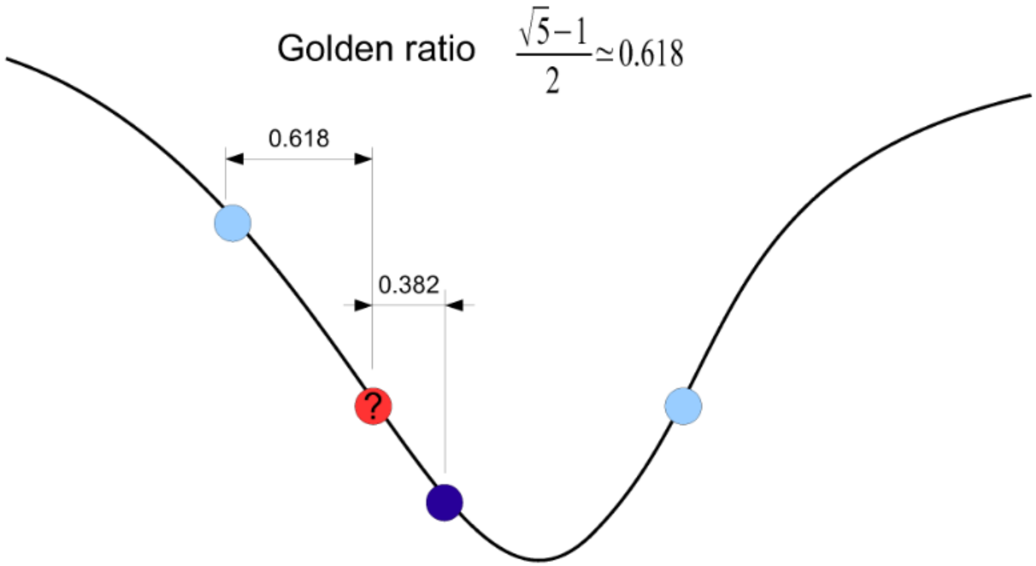
Refining the bracket (4)



- Redefine $a < b < c$. Here $c \leftarrow x$.
- Split the largest half and compute $f(x)$.

Line search

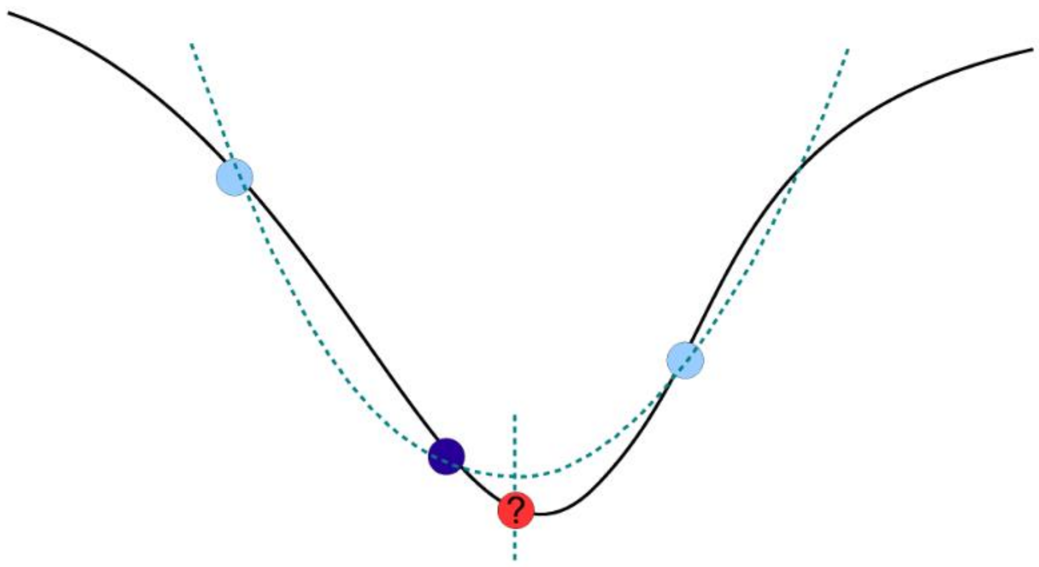
Golden ratio algorithm



– Optimal improvement by splitting at the *golden ratio*.

Line search

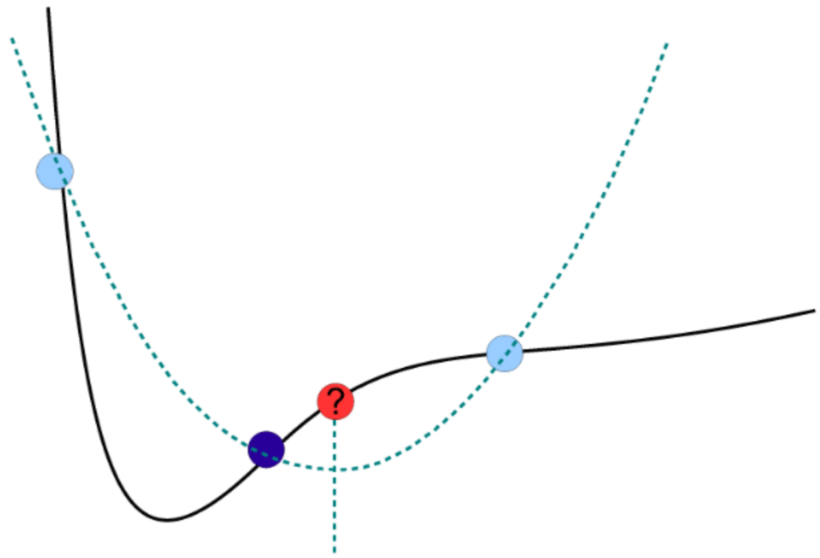
Parabolic interpolation



– Fitting a parabola can give much better guess.

Line search

Parabolic interpolation



– Fitting a parabola sometimes gives much better guess.

Line search

Brent algorithm

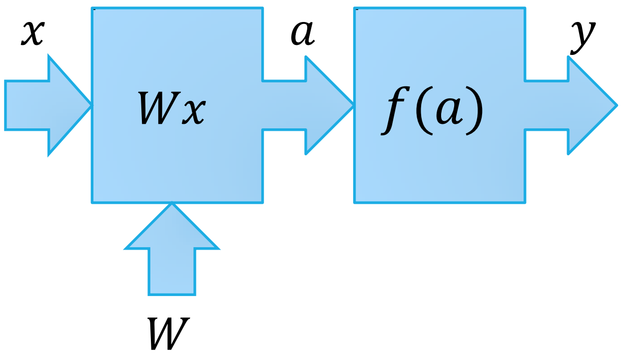
Brent Algorithm for line search

– Alternate golden section and parabolic interpolation.
– No more than twice slower than golden section.
– No more than twice slower than parabolic section.
– In practice, almost as good as the best of the two.

Variants with derivatives

– Improvements if we can compute $f(x)$ and $f'(x)$ together.
– Improvements if we can compute $f(x)$, $f'(x)$, $f''(x)$ together.

Rescaling weights



Propagation

- $y = f(Wx)$

Back-propagation

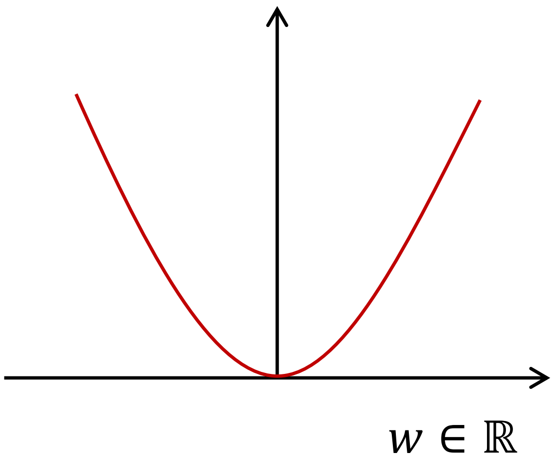
- $g_a = f'(a)\, g_y$
- $g_x = g_a\, W$
- $\Delta W = -\eta\, x\, g_a$

Consider the change $f_{new}(a) = f(2a)$ and $W_{new} = W/2$.

- This leaves $y(x)$ unchanged.
- What can you say about ΔW_{new} ?

Parabola

$$E(w) = \frac{c}{2} w^2$$



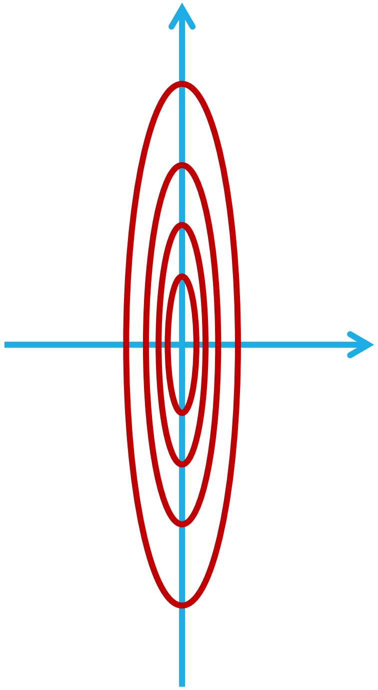
Gradient descent

- $w_{t+1} = w_t - \eta \frac{dE}{dw}(w_t)$

Questions

- How does η affect the convergence?
- What's the best value of η ?

More dimensions



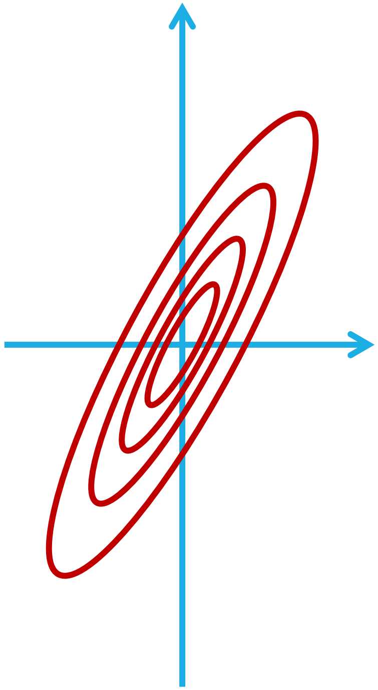
Two dimensions.

Two different curvatures.

Same questions

- How does η affect the convergence?
- What's the best value of η ?

More dimensions



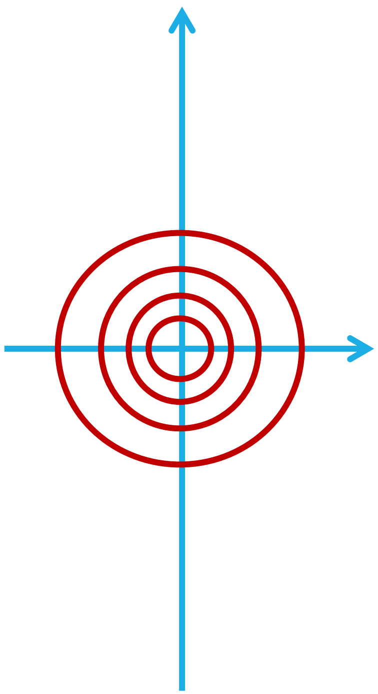
Gradient descent

- $E(w) = \frac{1}{2}\ w^T H\, w$
- $w_{t+1} = w_t - \eta\ \frac{dE}{dw}(w_t)$

Questions

- How does η affect the convergence?
- What's the best value of η ?

Second order rescaling



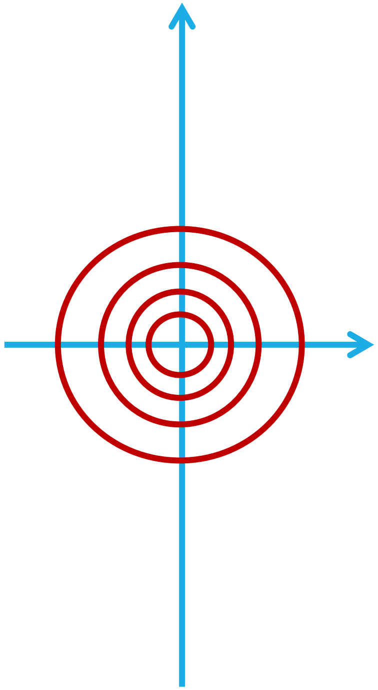
Rescale w

- $w_{new} \leftarrow H^{\frac{1}{2}}\, w$
- $E = \frac{1}{2}\, w^T H\, w = \frac{1}{2}\, w_{new}^T\, w_{new}$

Questions

- Write gradient descent in w_{new} space.
- Write the equivalent w update.

Second order rescaling



Rescale w

- $w_{new} \leftarrow H^{\frac{1}{2}}\, w$
- $E = \frac{1}{2}\, w^T H\, w = \frac{1}{2}\, w_{new}^T\, w_{new}$

Gradient descent in w_{new} space

- $\Delta w_{new} = -\eta\, \frac{dE}{dw_{new}} = -\eta\, H^{-\frac{1}{2}}\, \frac{dE}{dw}$
- $\Delta w = -\eta\, H^{-1}\, \frac{dE}{dw}$

Practical issues

- Objective function is not quadratic.
  - Local quadratic approximation is reasonable.
  - Hessian H changes with w.
  - When objective is non-convex, H can have negative eigenvalues
- Estimate the Hessian H on the fly.
- The Hessian is often too large to store or invert.

Standard solutions

Idea: estimate a compact approximation of H^{-1}
using the observed gradients $g(w_t), g(w_{t-1}), \ldots, g(w_{t-k}), \ldots$

Example: use line search and ensure conjugate search directions.

- Let $d_{t-1}, d_{t-2}, \ldots, d_{t-k}$ be the last k search directions.
  We want to choose $d_t = g(w_t) + \sum \lambda_i \, d_{t-i}$ such that $d_t^T \, H \, d_{t-i} = 0$

Very good algorithm have been developed.

- Conjugate gradient $(k = 1)$
- LBFGS ($k > 1$)

Attention

Three reasons to remain suspicious.

1. Our cost function is a sum of a large number of similar terms.
   This specific form can be used to speedup optimization.
   $$E(w) = \frac{1}{N} \sum_{i=1\ldots N} \ell(\, F(x_i), d_i)$$

2. Our problem is such that a random subset of terms is informative.
   Otherwise we cannot expect that our model will generalize!

3. Quickly achieving a good test set performance.
   $\neq$ quickly achieving a good training set performance

Simple things we can do

Precondition the inputs

- Normalize in similar ranges

Use different η in different layers, on the basis of

- Average size of the gradient
- Average size of the weights

Training multilayer networks

Initialization

Random weight initialization

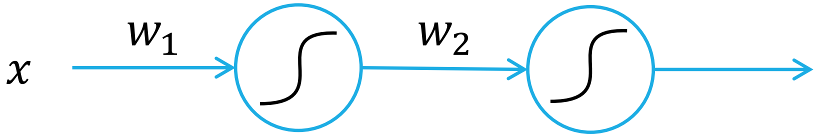
Why can we optimize such a complex non-convex function?

- We are not really optimizing.
- The problem is simpler than it looks.

Performance with random weights?

- The case of the two layer network with threshold units.
- The case of convolutional networks.

The simplest two-layer net

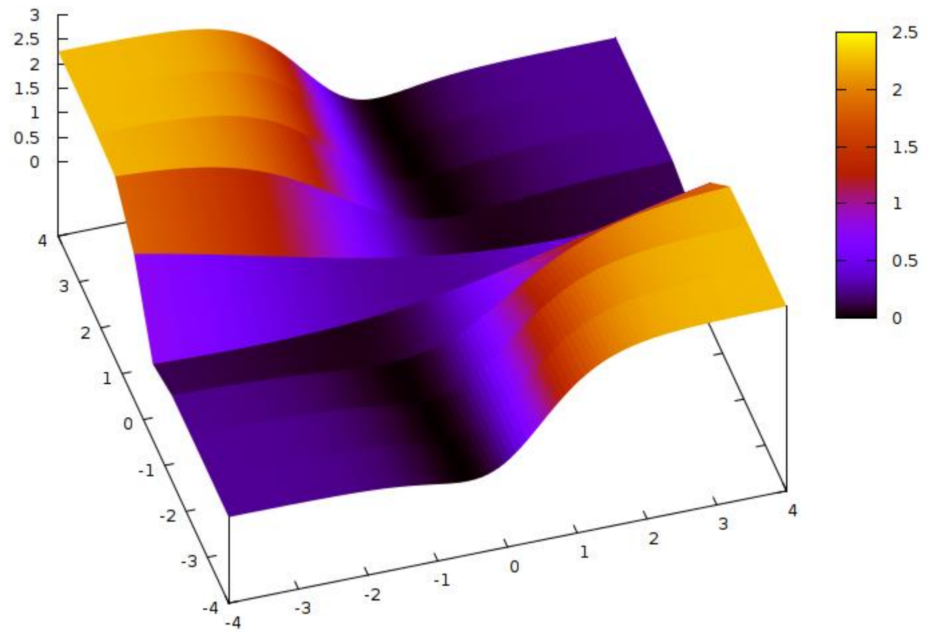


$y = \tanh(w_2 \tanh(w_1 x))$

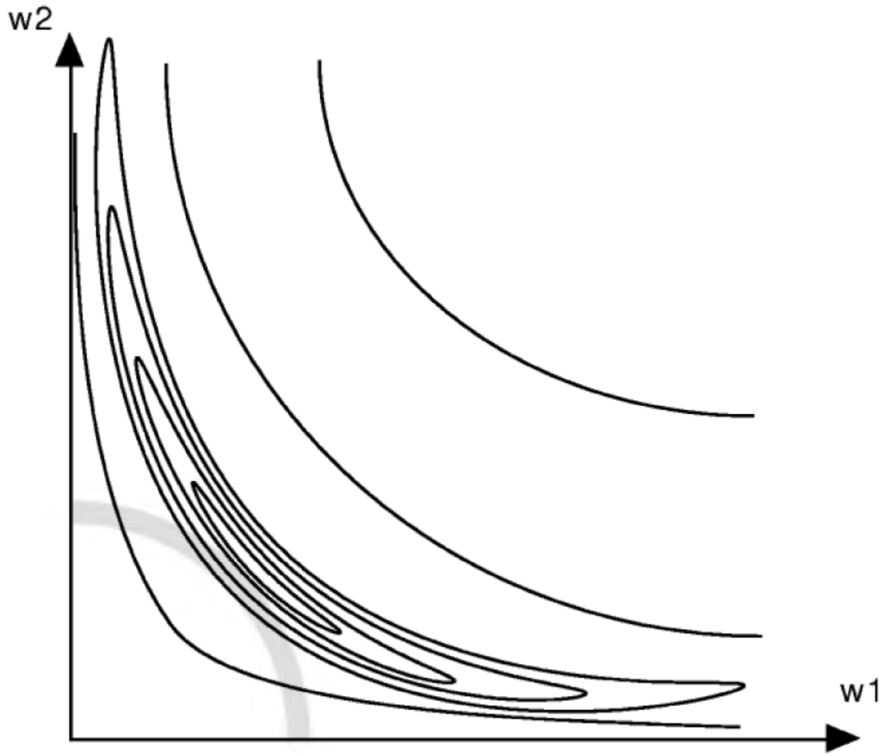
- Train on examples $(\frac{1}{2}, \frac{1}{2})$ and $(-\frac{1}{2}, -\frac{1}{2})$ with mean squared loss.
- $E = \left(\frac{1}{2} - \tanh\left(w_2 \tanh\left(\frac{w_1}{2}\right)\right)\right)^2$

How does this cost function look like?

The simplest two-layer net



The simplest two layer net



Initialization

The main rule for random weight initialization

- Do not pick initial weights that kill the gradient!

The role of the transfer function

- The distribution of the inputs to the transfer function
  - should target the linear part.
  - should have a chance to exploit the nonlinearity.
  - Exercises: Tanh, Sigmoid, ReLU.

Training multilayer networks

Stochastic gradient descent

Optimization vs. learning

Empirical cost
– Usually $f(w) = \frac{1}{n}\sum_{i=1}^{n} L(x_i, y_i, w)$
– The number n of training examples can be large (billions?)

Redundant examples
– Examples are redundant (otherwise there is nothing to learn.)
– Doubling the number of examples brings a little more information.
– Do we need it during the first optimization iterations?

Examples on-the-fly
– All examples may not be available simultaneously.
– Sometimes they come on the fly (e.g. web click stream.)
– In quantities that are too large to store or retrieve (e.g. click stream.)

Offline vs. online

Minimize $C(w) = \frac{\lambda}{2}\|w\|^2 + \frac{1}{n}\sum_{i=1}^{n} L(x_i, y_i, w)$.

Offline: process all examples together

– Example: minimization by gradient descent

$$\text{Repeat: } w \leftarrow w - \gamma \left(\lambda w + \frac{1}{n}\sum_{i=1}^{n} \frac{\partial L}{\partial w}(x_i, y_i, w) \right)$$

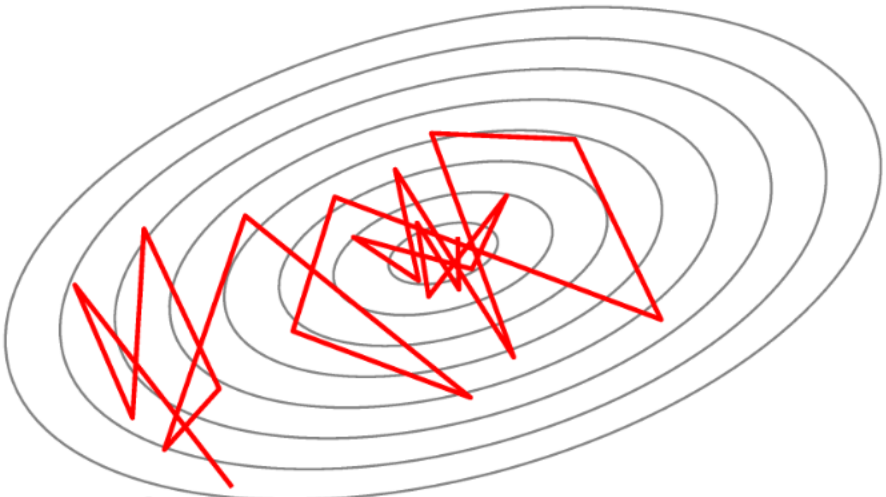
Offline: process examples one by one

– Example: minimization by stochastic gradient descent

Repeat: (a) Pick random example x_t, y_t

(b) $w \leftarrow w - \gamma_t \left(\lambda w + \frac{\partial L}{\partial w}(x_t, y_t, w) \right)$

Stochastic Gradient Descent



- Very noisy estimates of the gradient.
- Gain γ_t controls the size of the cloud.
- Decreasing gains $\gamma_t = \gamma_0(1 + \lambda\gamma_0 t)^{-1}$.
- Why is it attractive?

Stochastic Gradient Descent

Redundant examples

– Increase the computing cost of offline learning.

– Do not change the computing cost of online learning.

Imagine the dataset contains 10 copies of the same 100 examples.

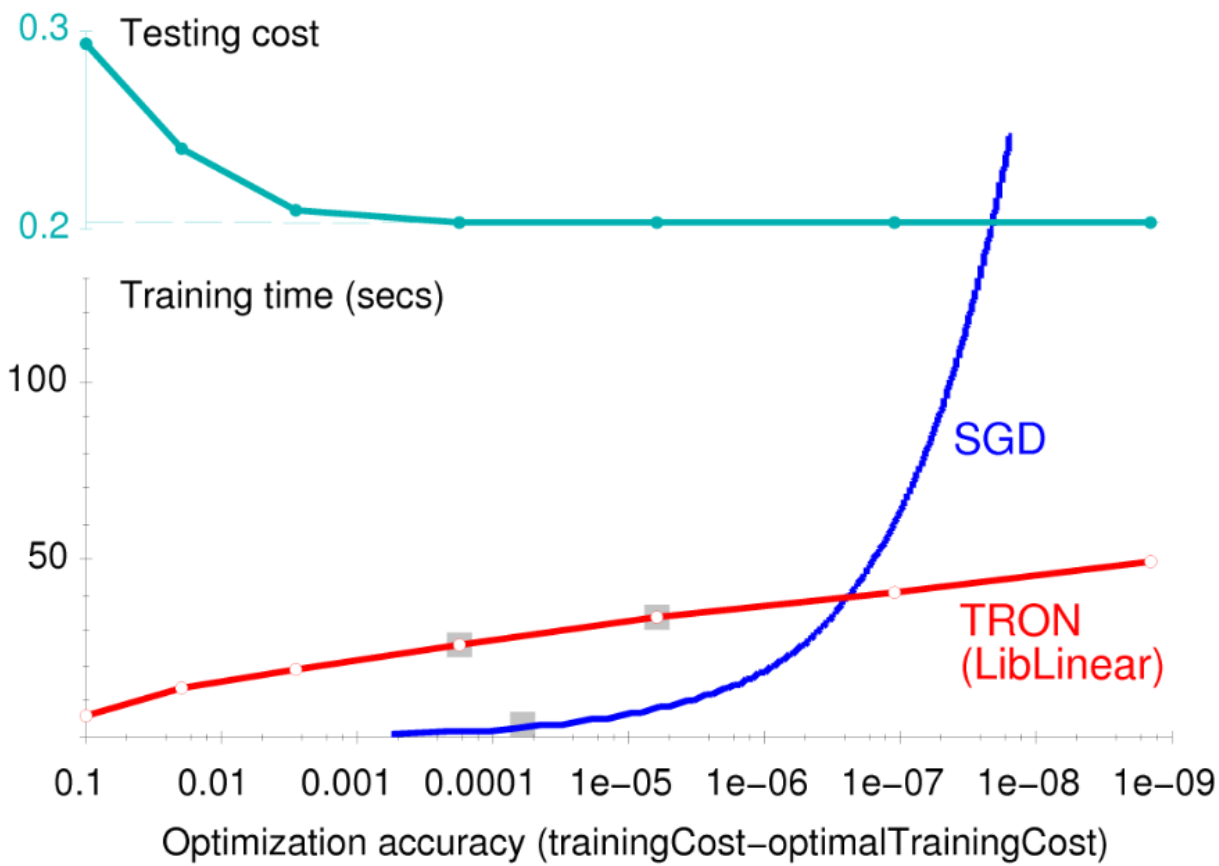
- Offline Gradient Descent

  Computation is 10 times larger than necessary.

- Stochastic Gradient Descent

  No difference regardless of the number of copies.

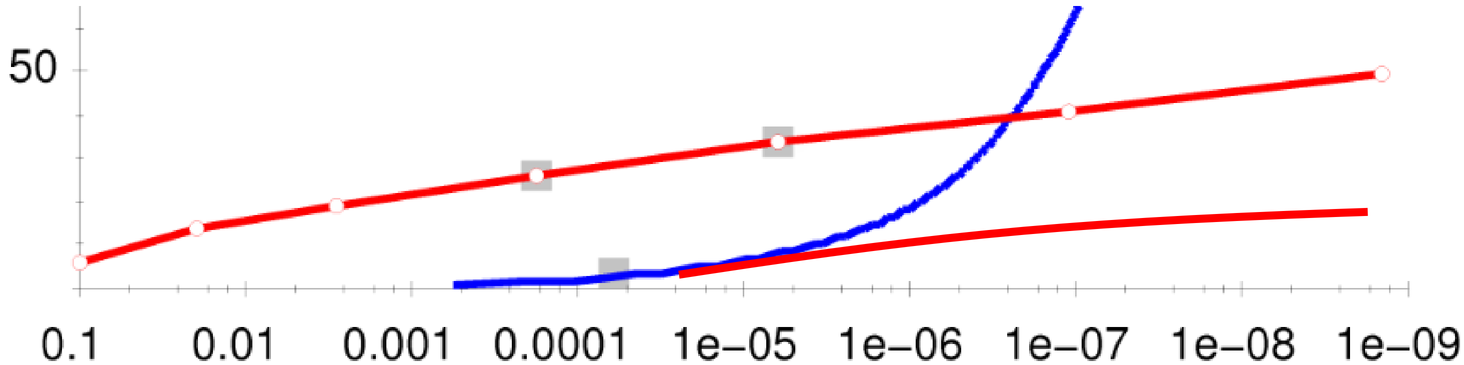
Practical illustration



Subtleties

How to quickly achieve a good training set performance?

- Initialize super-linear algorithm with SGD!



- Question : when does this help the testing set performance?

Training multilayer networks

Improved algorithms

Overview

Lots of improved algorithms in the recent literature

- Momentum and acceleration
- Mini-batch techniques
- Parallel training

Questions to ask ourselves…

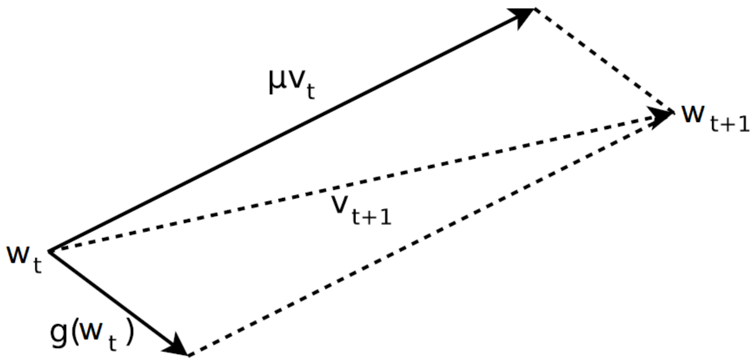
- Do they quickly achieve good test errors or good training errors?
- In most papers, the experiments target the test, and the theory targets the training.
- This does not mean that the proposed method is useless.
  It means that the theoretical argument is oversold.

Momentum and acceleration

MOMENTUM

$$v_{t+1} = \mu\, v_t - \eta \,\mathrm{grad}\, E(w_t)$$

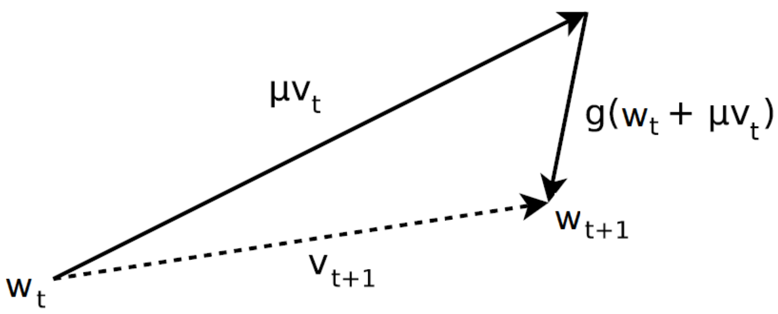
$$w_{t+1} = w_t + v_{t+1}$$



NESTEROV ACCELERATION

$$v_{t+1} = \mu\, v_t - \eta \,\mathrm{grad}\, E(w_t + \mu v_t)$$

$$w_{t+1} = w_t + v_{t+1}$$



(Sutskever et al., ICML 2013)

Mini-batches

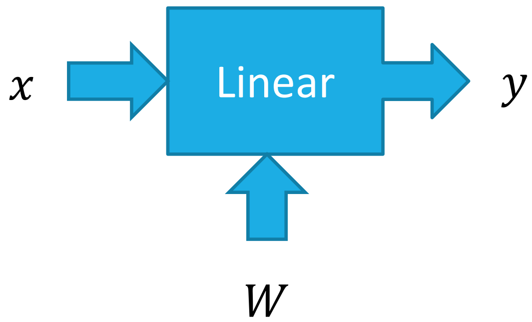
Stochastic gradient descent

- Use noisy gradient based on a single example.

Mini-batch stochastic gradient descent

- Use noisy gradient based on a small batch of examples.
- Theoretical results are unimpressive for first order gradient descent.
- However:
  1. Mini-batches are well suited to modern hardware
  2. Mini-batches provide an opportunity to use second order info.

Modern hardware



Single example formulas

- $y = Wx$ (GEMV)
- $g_x = g_y\, W$ (GEMV)
- $\Delta W = x\, g_y$ (GER)

Multiple example formulas

- $Y = WX$ (GEMM)
- $G_X = G_Y W$ (GEMM)
- $\Delta W = X\, G_Y$ (GEMM)

Successive LBFGS

```
for t = 1,2,3, …
```

- **pick** examples for mini-batch t
- **initialize** net with weights w_t
- **optimize** with LBFGS and obtain w_{t+1}

- This does not work with convex models (why?)
- But this works quite well with multilayer networks (why?)

Martens HF training

```
pick a first mini-batch (mini-batch 0)
for t = 1,2,3, …
```

- **pick** examples for mini-batch t
- **compute** $g_t = grad\, E_t(w_t)$ on mini-batch t
- **minimize** $d^T Hd + \lambda d^2 + g_t d$ by CG

  where the product Hd is evaluated directly using gradients measured on mini-batch 0.
- **update** $w_{t+1} = w_t + d$

- Lots of refinements are necessary to make this work well.

(Martens, 2010, 2012)

Parallel training of neural nets

An active topic of research.

- No clear winner yet.

Baseline: lock-free stochastic gradient

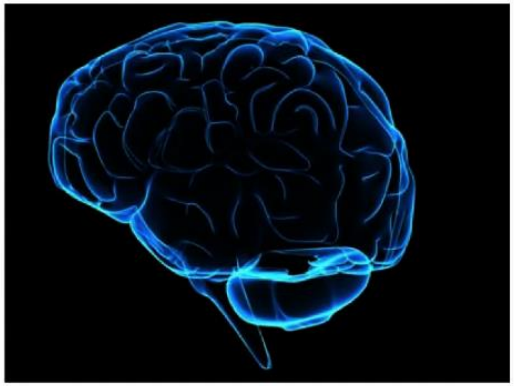
- Assume shared memory
- Each processor access the weights through the shared memory
- Each processor runs SGD on different examples
- Read and writes to the weight memory are unsynchronized.
- Synchronization issues are just another kind noise…

Deep networks for complex tasks

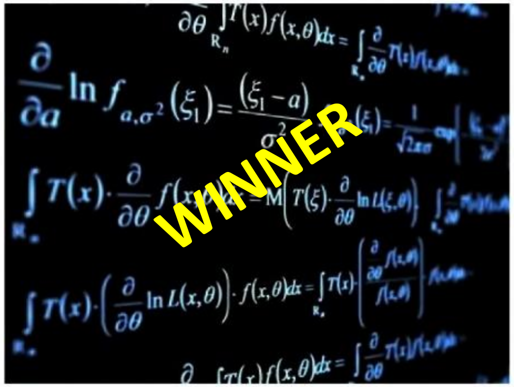
Introduction

How to design computers?

Biological computer



Mathematical computer



Why do computers emulate mathematical logic?

- Complex tasks are reduced to combinations of simple tasks.
- New ways to solve simple tasks immediately benefit everything.

Remember the perceptron

Reducing complex tasks to combinations of simple tasks

- An engineering necessity.
- Simple learning tasks
  - classification, regression, clustering, multi-armed bandits.
    (and many other eight-pages papers)
- Complex learning tasks
  - reading checks (segmentation, recognition, interpretation)
  - parsing visual scenes (finding objects and their relations)
  - composing personalized web pages (dealing with feedback)
  - natural language understanding (hard to define…)
  - strong AI (let’s dream…)

Bayesian inference

The appeal of Bayesian inference

- A language to describe complex models with simpler ones?
- Generic algorithms

Things that Bayesian inference does not do well

- Computationally costly algorithms lead to dirty approximations
- Causation versus correlation (a different kind of reasoning)
- Perception

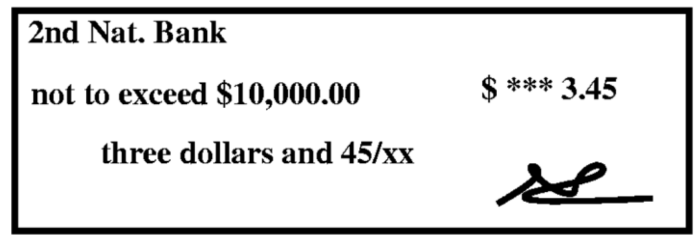
Deep networks for complex tasks

Structured problems

Engineering learning systems

Reading check amounts

- Input $x \in \mathcal{X}$: scanned check image
- Output $y \in \mathcal{Y}$: positive number



Direct approach

- Collect examples $\{(x_1, y_1), (x_2, y_2), \ldots\}$ and train from scratch.
- Possible (we did not really try)
- Requires excessive numbers of labeled examples
- Requires excessive computation time.

Engineering learning systems

Identify sub-tasks

- Locate amount fields
- Segment amount fields into isolated characters
- Recognize isolated characters
- Translate character string into amount

Define a model for each sub-task

- Fairly complex recognition models (e.g. CNN)
- Highly engineered location and segmentation models

Collect data and train

Interactions

- Locate amount fields
- Segment amount fields into isolated characters
- Recognize isolated characters
- Translate character string into amount

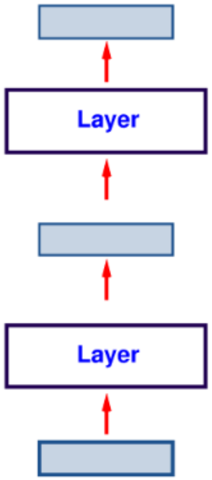
Training strategies

- Independent training
  - train each sub-model separately.
- Sequential training (better)
  - pre-train with independent training.
  - label outputs of sub-model n and train sub-model $n + 1$.
- Global training (best)
  - pre-train with sequential training.
  - simultaneously train all sub-models with examples from $\mathcal{X} \times \mathcal{Y}$.

Problem: tracking multiple hypothesis, backtracking, etc.

Graph transformer networks

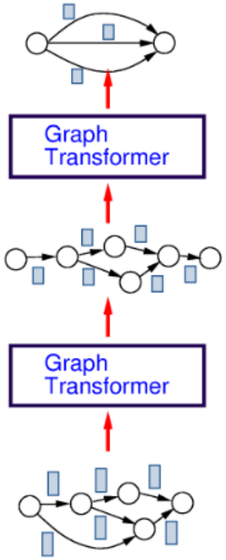
MULTILAYER NET



Intermediate representations are fixed size vectors.

Each vector represents a decision made by upstream modules and passed to downstream modules.

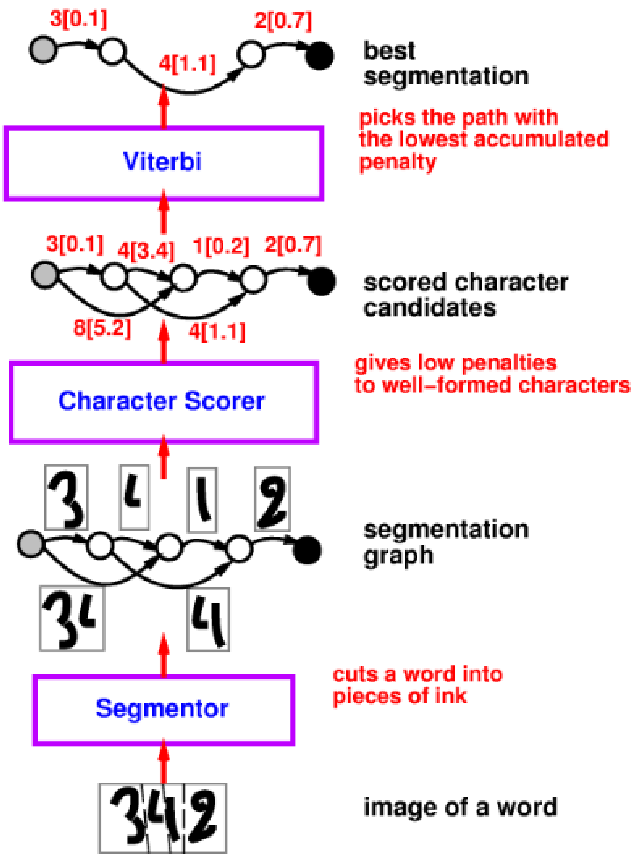
GRAPH TRANSFORMER NET



Intermediate representations are graphs.

Each path in a graph represents a combination of hypotheses made by upstream modules and passed to downstream modules.

A word reader



Normalization and discrimination

GENERATIVE TRAINING

Estimate $P(x, y)$

- Define model $p_w(x, y)$

$$\forall w \quad \sum_{x,y} p_w(x, y) = 1$$

- Optimize likelihood

$$\max \sum_i \log p_w(x_i, y_i)$$

DISCRIMINANT TRAINING

Estimate $P(y|x)$

- Define model $p_w(x, y)$

$$\forall\, w, x \quad \sum_y p_w(x, y) = 1$$

- Optimize likelihood

$$\max \sum_i \log p_w(x_i, y_i)$$

Spot the difference!

Probabilistic models

Generative Hidden Markov Model

$$p_w(x,y) = P(x,y|w) = \sum_{s[t]:y} \prod_t P(s_t|s_{t-1},w)\, P(x_t|s_t,w)$$

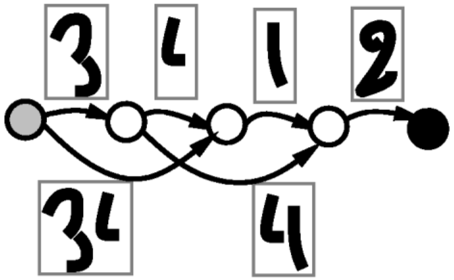
Probabilistic construction ensures normalization.

Discriminant Hidden Markov Model

$$p_w(x,y) = P(y|x,w) = \sum_{s[t]:y} \prod_t P(s_t|s_{t-1},x_t,w)$$

Output of the local classifier must be normalized.

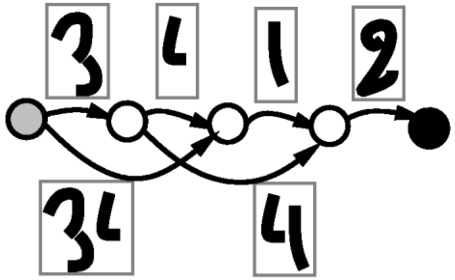
This is a very bad idea.



Denormalized models

Build models using measures instead of probabilities

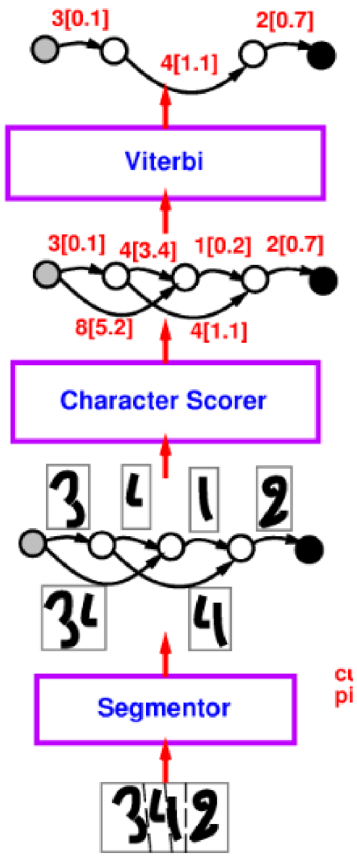
- Measures add and multiply like probabilities
- Measures are positive but not constrained to sum to one.



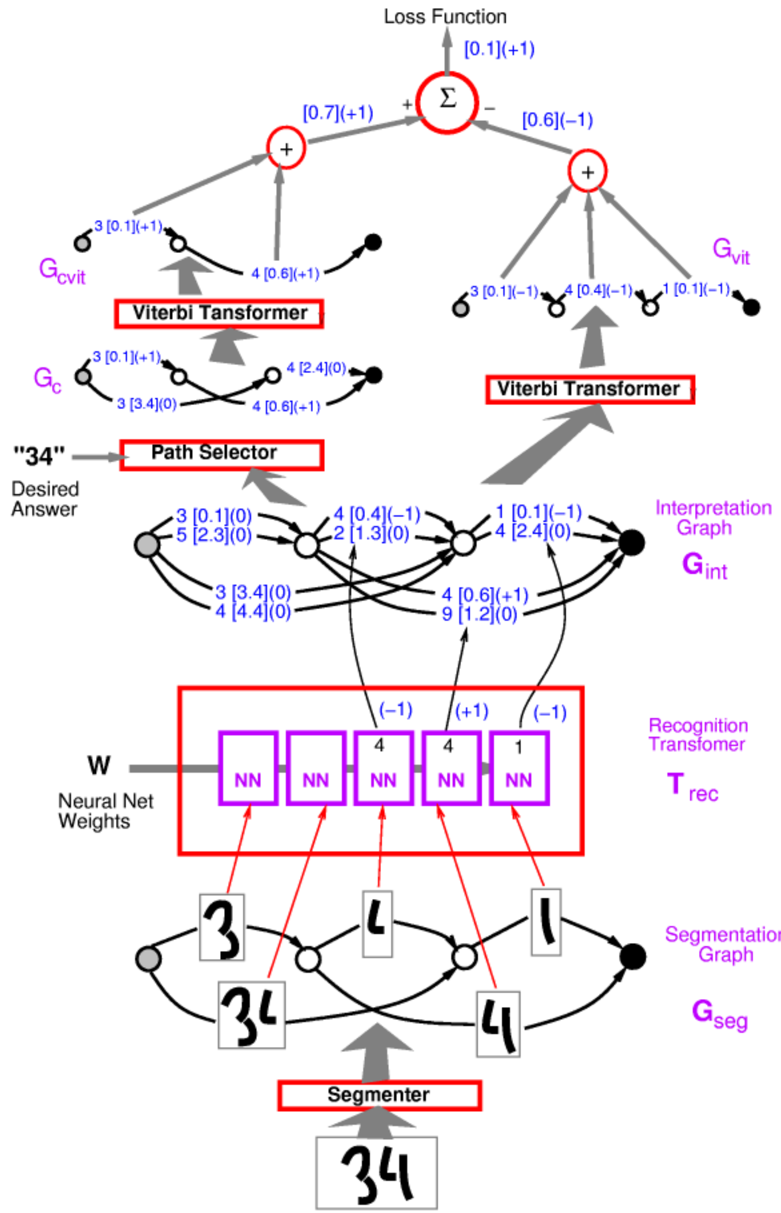
Score of a path = product of arc scores

Score of a subgraph = sum of path scores

- Train by maximizing $\sum_i \log \frac{p_W(x_i, y_i)}{\sum_y p_W(x_i, y)}$
- Same as CRF cost function.
- Semi-ring variations : $(\mathbb{R}_+, +, \times)$ $(\mathbb{R}, \oplus, +)$ $(\mathbb{R}, \max, +)$ …



3[0.1]
4[1.1]
2[0.7]
Viterbi
3[0.1]
4[3.4]
1[0.2]
2[0.7]
8[5.2]
4[1.1]
Character Scorer
Segmentor
cu
pi



Loss Function
[0.1](+1)
[0.7](+1)
[0.6](−1)
Σ
Viterbi Tansformer
Viterbi Transformer
Path Selector
"34"
Desired Answer
Interpretation Graph
Recognition Transfomer
W
Neural Net Weights
NN
Segmentation Graph
Segmenter

GTN and CRF

Graph Transformer Network

- CRF cost function
- Hierarchical coarse-to-fine model
- Cheap inference

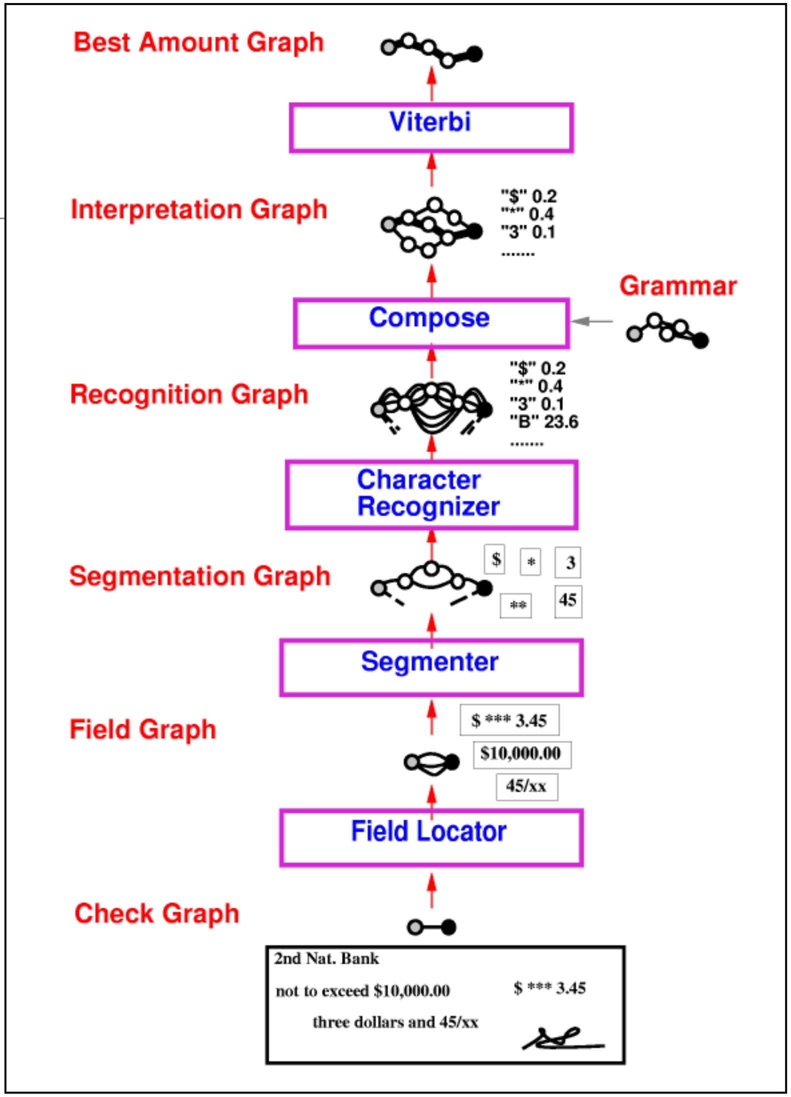
Check reader

AT&T Bell Labs, 1995-1996.

Industrially deployed in 1996.

Has processed
15% of all the US checks
for nearly fifteen years.

(B. et.al., CVPR 1997)



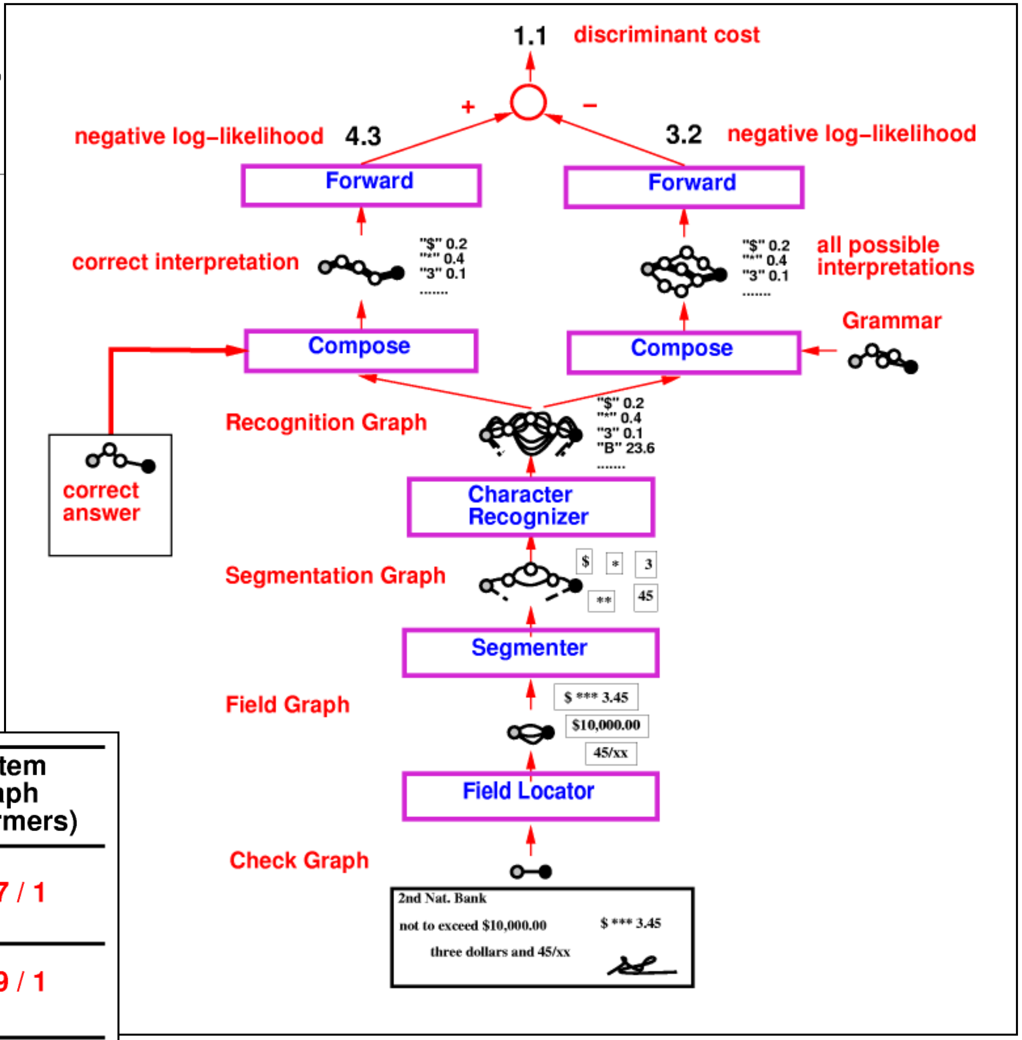
Check reader

Recognizer

Lenet5 (CNN)

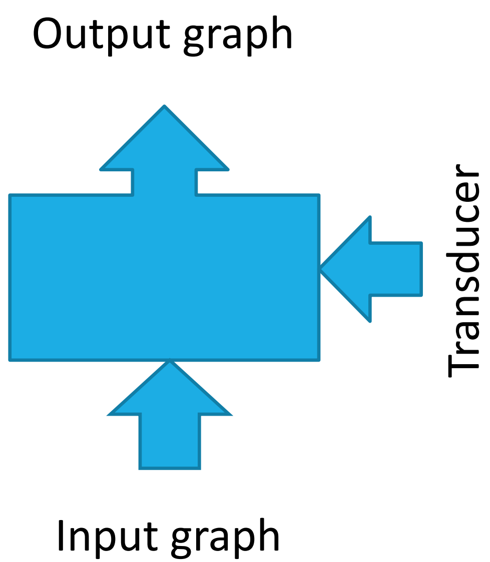
Pre-trained on 500K
isolated characters
95 categories (why?)

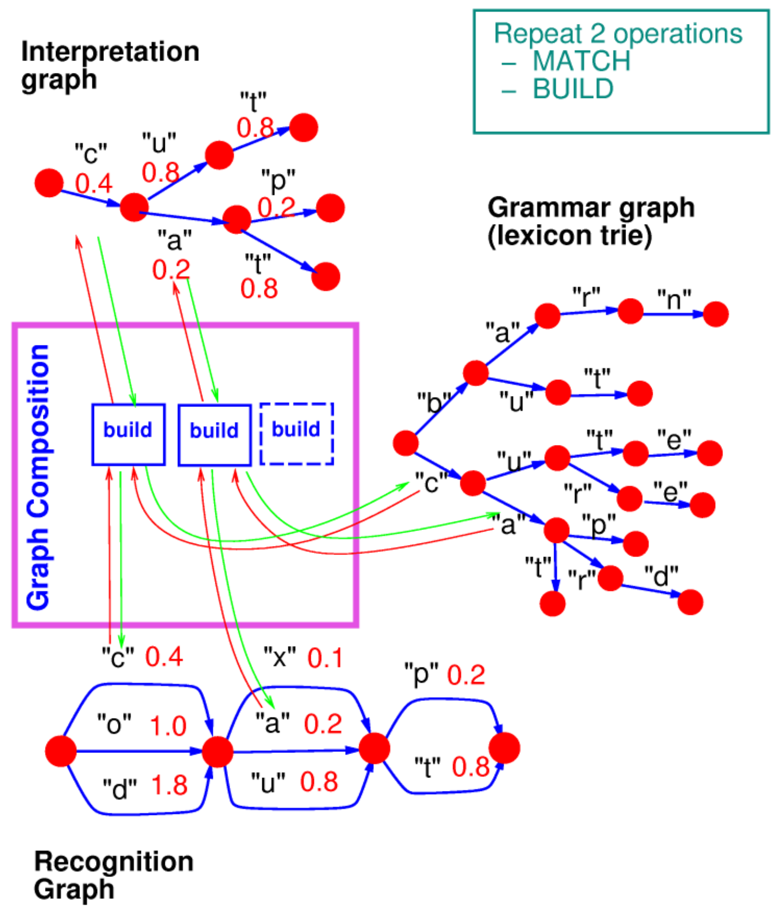
Global training architecture →



	old system (was state of the art)	new system (with graph transformers)
654 machine printed checks	68 / 31 / 1	82 / 17 / 1
realistic mixture of 1986 checks	45 / 54 / 1	50 / 49 / 1

Graph transduction brick





Deep networks for complex tasks

Auxiliary tasks

Retargeting learned features

ImageNet features for Caltech256

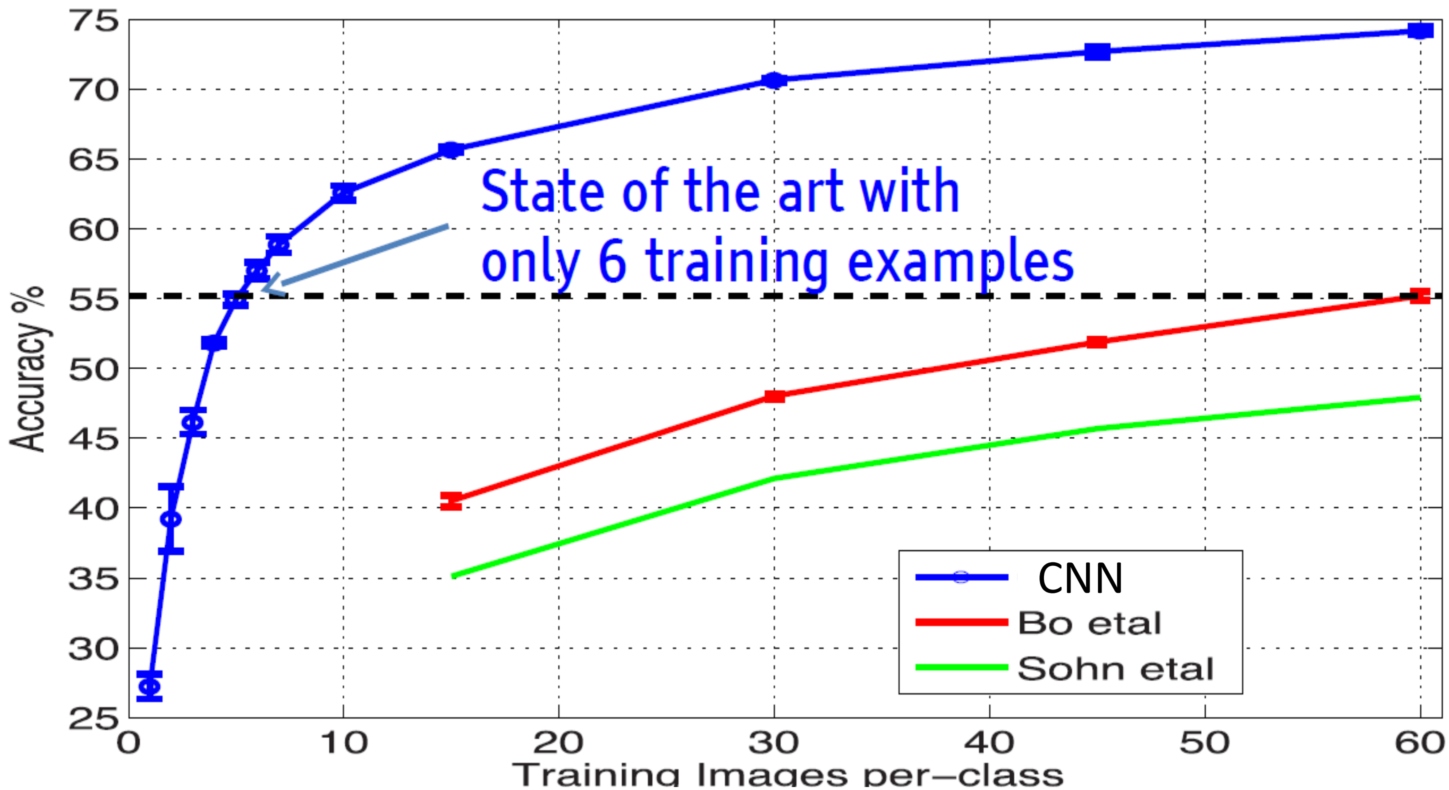
- Train CNN on ImageNet
- Chop the last layer (ImageNet categories)
- Append a new last layer (Caltech 256 categories)
- Train network on Caltech256 data.

Question

- Should we keep the weights fixed in the ImageNet-trained layers.

(LeCun, Ranzato, 2012)

Retargeting learned features



(LeCun, Ranzato, 2012)

Auxiliary tasks

The price of labels

- Labeled examples for interesting tasks are typically scarce.
- Abundant labeled examples exists for uninteresting tasks.

Auxiliary task

- In the vicinity of an interesting task (with scarce labels)
  there are uninteresting tasks (with cheap labels)
  that can be put to good use.

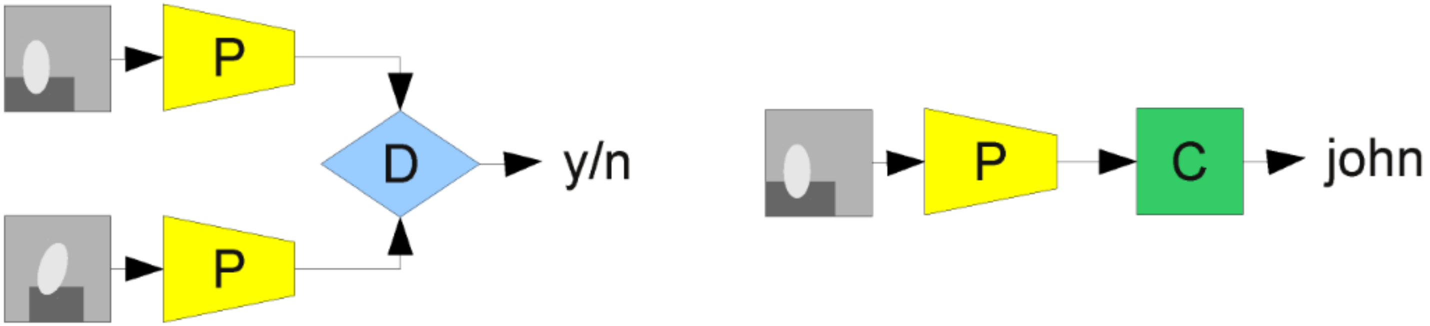
Example: face recognition

Interesting task. Recognizing the faces of one million persons.

- How many labeled images per person can we obtain?

Auxiliary task. Are two face images representing the same person?

- Abundant (but noisy) examples.
  - Two faces in the same picture are different persons (with exceptions)
  - Two faces in successive frames are often the same person (with exceptions)



(Matt Miller, NEC, 2006)

Example: NLP tagging

Interesting task. Standard NLP tagging tasks.

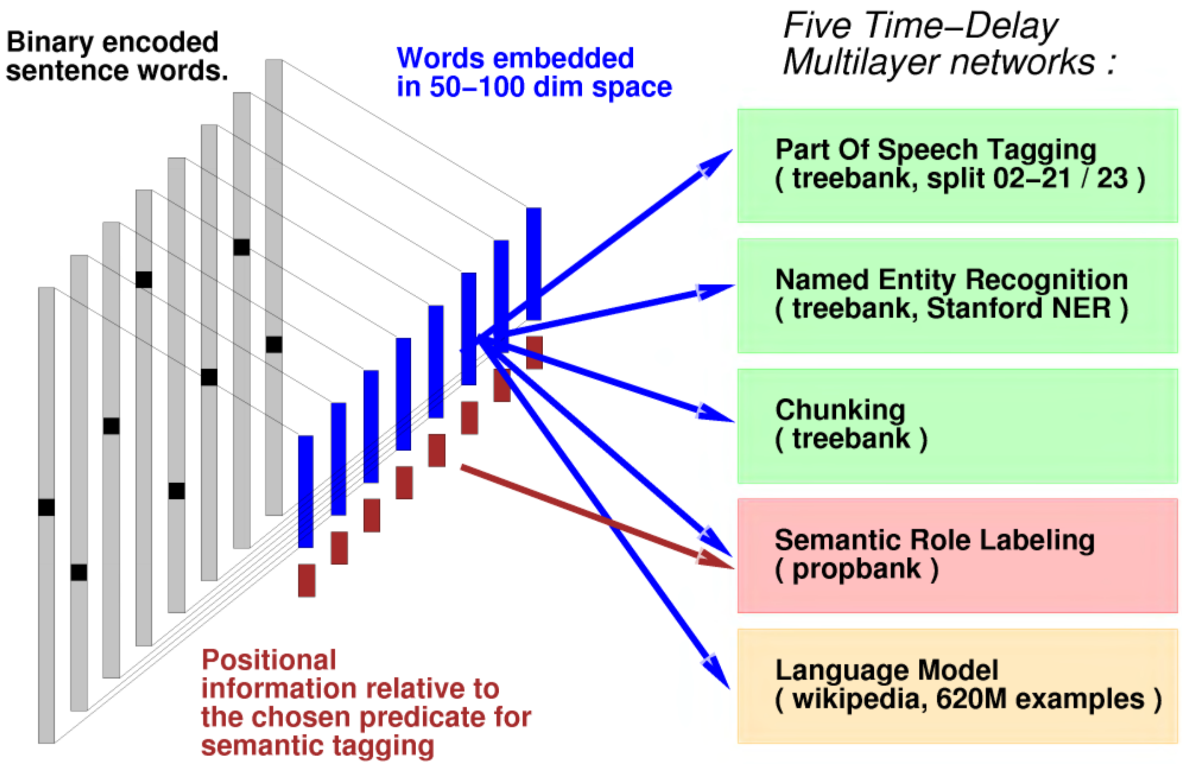
- Labeled data: Treebank, Propbank (1M words)

Auxiliary task. Word compatibility language model

- Positive examples: Wikipedia sentences segments. (600M words)
- Negative examples built by randomly replacing the central word.
- Ranking loss: score of positive > score of negative

(Collobert et.al., 2008-2010)

Example: NLP tagging



Example: NLP tagging

FRANCE	JESUS	XBOX	REDDISH	SCRATCHED	MEGABITS
454	1973	6909	11724	29869	87025
AUSTRIA	GOD	AMIGA	GREENISH	NAILED	OCTETS
BELGIUM	SATI	PLAYSTATION	BLUISH	SMASHED	MB/S
GERMANY	CHRIST	MSX	PINKISH	PUNCHED	BIT/S
ITALY	SATAN	IPOD	PURPLISH	POPPED	BAUD
GREECE	KALI	SEGA	BROWNISH	CRIMPED	CARATS
SWEDEN	INDRA	PSNUMBER	GREYISH	SCRAPED	KBIT/S
NORWAY	VISHNU	HD	GRAYISH	SCREWED	MEGAHERTZ
EUROPE	ANANDA	DREAMCAST	WHITISH	SECTIONED	MEGAPIXELS
HUNGARY	PARVATI	GEFORCE	SILVERY	SLASHED	GBIT/S
SWITZERLAND	GRACE	CAPCOM	YELLOWISH	RIPPED	AMPERES

Example: NLP tagging

- Tagging speed above 10000 words per second

```
$ echo "I've made up this sentence to demonstrate Senna." | ./senna-linux64
           I       PRP        S-NP          0             -      S-A0      S-A0
         've       VBP        B-VP          0             -         0         0
        made       VBN        E-VP          0          made       B-V         0
          up        RP       S-PRT          0             -       E-V         0
        this        DT        B-NP          0             -      B-A1         0
    sentence        NN        E-NP          0             -      E-A1         0
          to        TO        B-VP          0             -  B-AM-PNC         0
 demonstrate        VB        E-VP          0   demonstrate  I-AM-PNC       S-V
       Senna       NNP        S-NP      S-PER             -  E-AM-PNC      S-A1
           .         .           0          0             -         0         0
```

http://ronan.collobert.com/senna

Example: NLP tagging

Task		Benchmark	SENNA
Part of Speech (POS)	(Accuracy)	97.24 %	97.29 %
Chunking (CHUNK)	(F1)	94.29 %	94.32 %
Named Entity Recognition (NER)	(F1)	89.31 %	89.59 %
Parse Tree level 0 (PT0)	(F1)	91.94 %	92.25 %
Semantic Role Labeling (SRL)	(F1)	77.92 %	75.49 %

POS System	RAM (MB)	Time (s)
Toutanova et al. (2003)	800	64
Shen et al. (2007)	2200	833
SENNA	32	4

SRL System	RAM (MB)	Time (s)
Koomen et al. (2005)	3400	6253
SENNA	124	51

Unsupervised auxiliary tasks

Deep learning with unsupervised layer-wise training.

- Sequentially pre-train successive layers using unsupervised techniques.
  e.g., noisy auto-encoders (feedforward), RBM ($\neq$)
- Fine tune using multilayer supervised training.

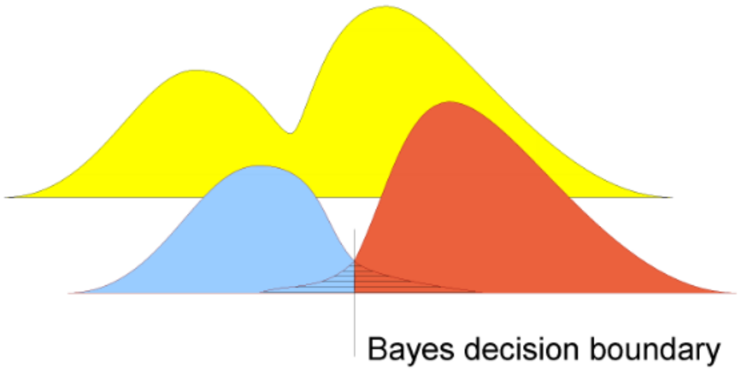
Remark

- This is less popular than it used to be two years ago.
  (fully supervised technique seem to work as well.)

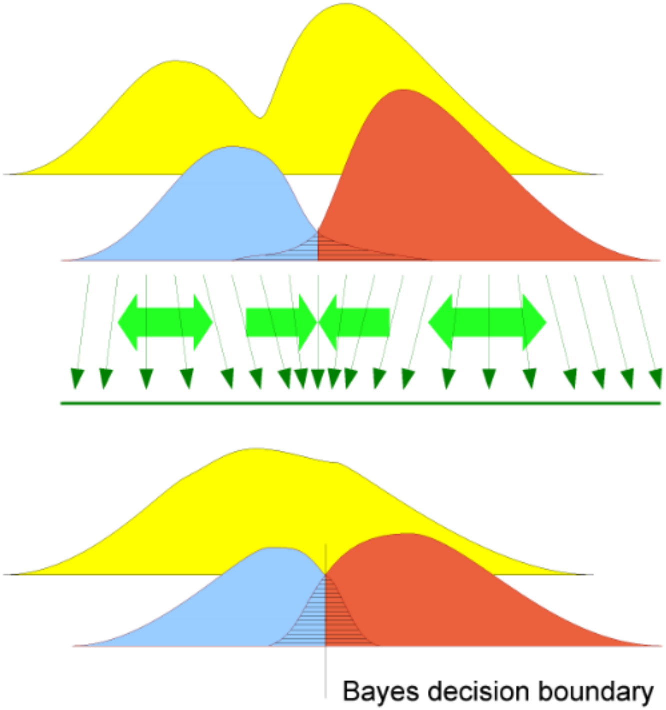
Unsupervised learning?

What is a cluster?

– Assumption: the shape of the density reveals the underlying categories.



Unsupervised learning?



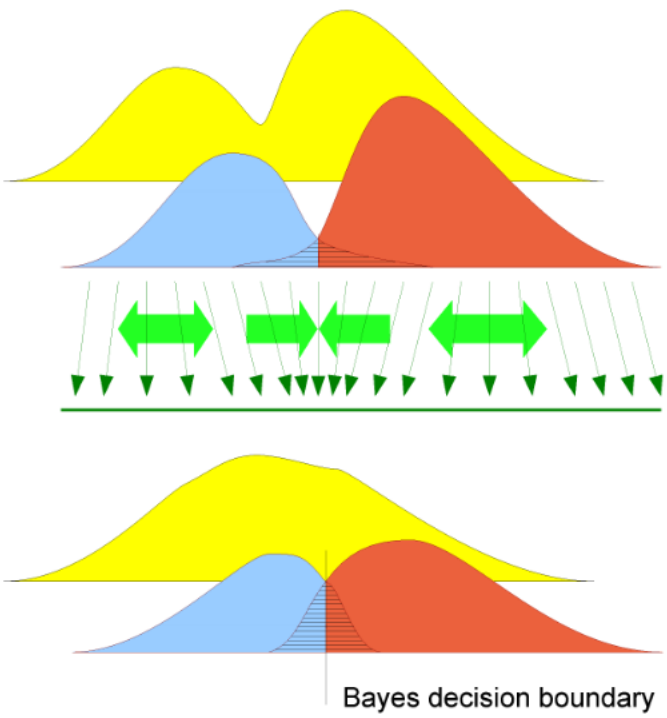
Input space transforms

– Categories are invariant.

– Bayes rate is invariant.

– Clustering is not invariant.

Unsupervised learning?



Clustering revisited

– Clustering is the expression of the prior knowledge encoded by our choice of input representation.

Unsupervised learning

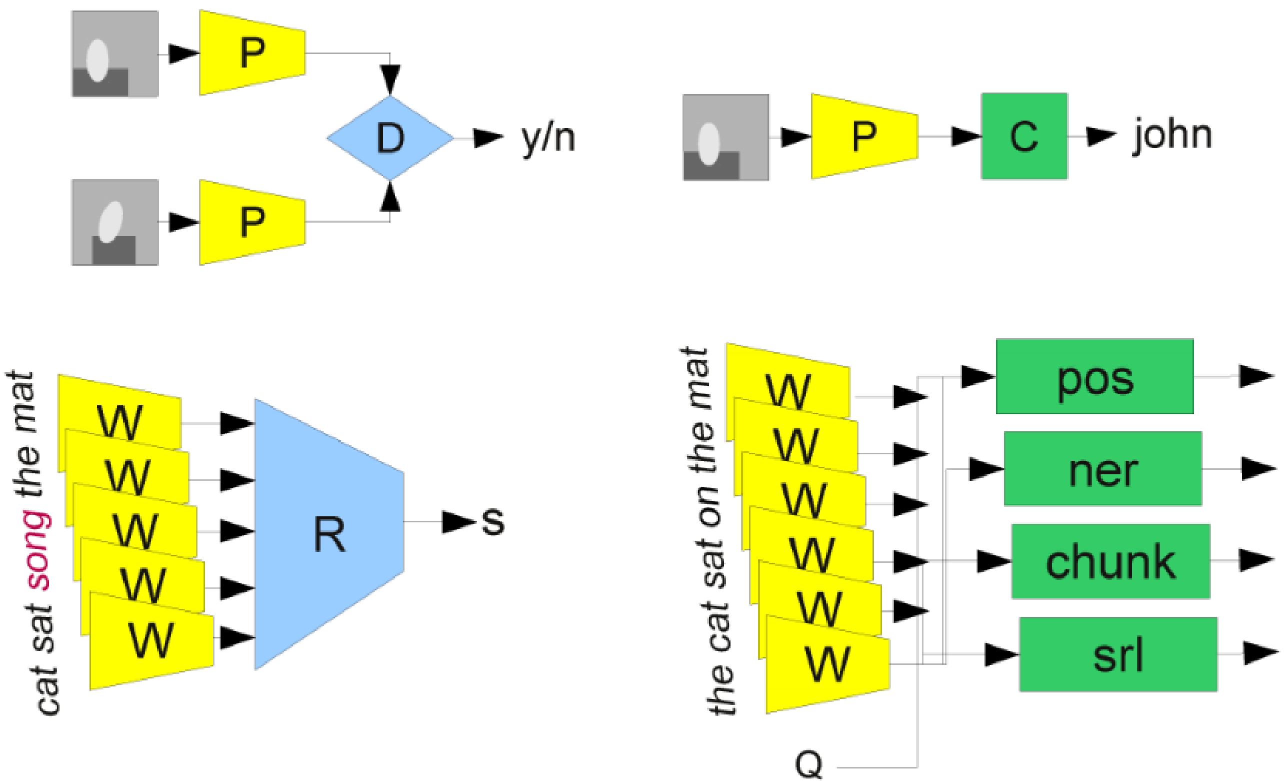
– Comparable to using really cheap labels:
"x_1 and x_2 are close".
"x_1 and x_3 are not close".

Deep networks for complex tasks

Circuits algebra

(B., ArXiV, 2011)

Transfer learning by rewiring



Bayesian perspective

Elementary modules are parametrized by distributions.

Step 1 - Training the auxillary task

– Posteriors do not necessarily factorize according to modular structure

$\rightarrow$ *Projecting posteriors on the factorized space.*

Step 2 - Training the main task

– The auxilliary task posterior becomes the main task prior.

$\rightarrow$ *Improved generalization (e.g. using the PAC-Bayes theorem.)*

– But this does not say which transfer strategies will work best.

Circuit algebra

Rewiring as Algebraic Operation

– Rewiring simultaneous operates in two spaces:
  - ◇ Composition of statistical models.
  - ◇ Composition of model realizations.

$\longrightarrow$ *Transporting the functions and their parametrization*

– Inherited structure in the parameter spaces

– Inherited structure in the "space" of questions of interest

Algebraic structure is an expression of the semantics

– Circuit algebra $\Longleftrightarrow$ Semantic Equation Models (Pearl, 2000).

– Causal semantics rather than probabilistic semantics.

Enriching the semantics

Algebraic structure is an expression of the semantics

– Enriching the algebraic structure $\Longleftrightarrow$ Enriching the semantics.

Making the structure recursive

– A time-honored way to generate rich algebraic structures.

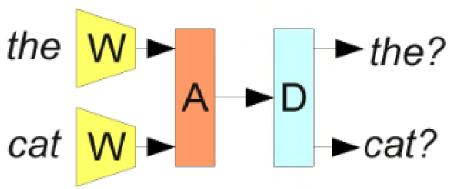
Recursive Auto-Associative Memory

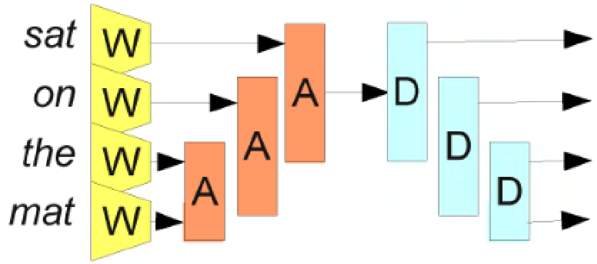
Elements

– A representation space $\mathcal{R}$.

– Association module $A : \mathcal{R} \times \mathcal{R} \longrightarrow \mathcal{R}$.

– Dissociation module: $D : \mathcal{R} \longrightarrow \mathcal{R} \times \mathcal{R}$.





– Desired invariance: $D(A(x, y)) = (x, y)$.

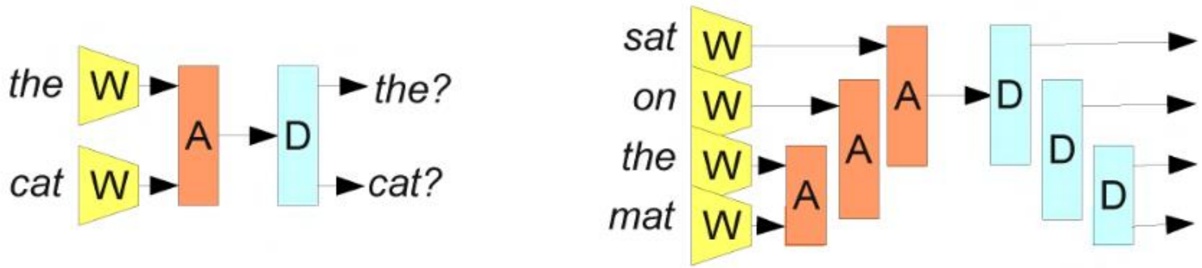
(Pollack, 1988) (Hinton, 1990)

Infinite depth structures

Algebraic structure matters more than representation space $\mathcal{R}$.

– RAAMs can represent infinite depth predicates.

– Same as `cons`, `car`, `cdr`.



Approximate invariance

– Consider a numerical representation space, i.e. $\mathcal{R} = \mathbb{R}^{100}$.

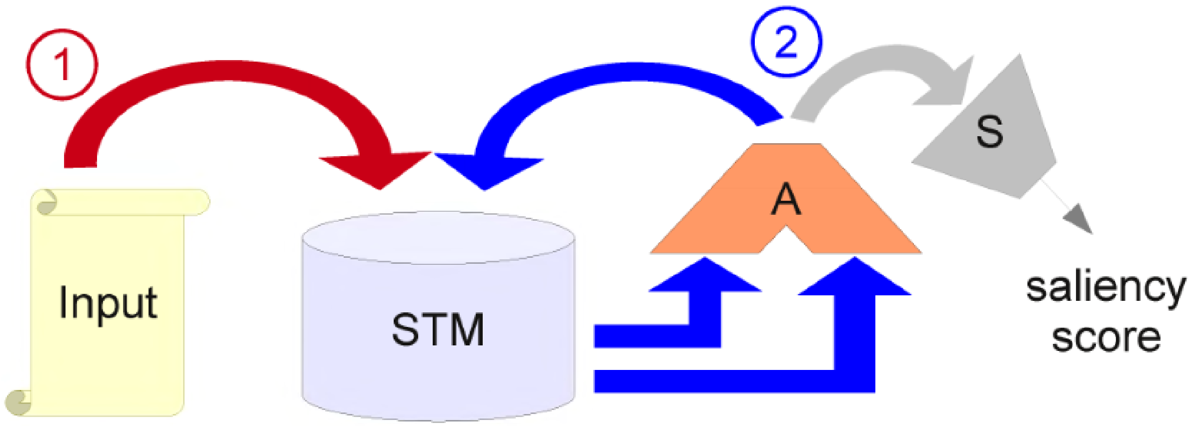
– Numerical accuracy will eventually degrade reconstruction.

– If the embeddings in the representation space make sense the dissociation module then reconstruct approximate sentences.

Universal parser

Elements

– Saliency module $S : \mathcal{R} \longrightarrow \mathbb{R}$
– Short term memory.



– Parsing text and images e.g. (Socher, 2010).
– Parsing anything in fact.
– Related to "chunking" (Miller, 1956).

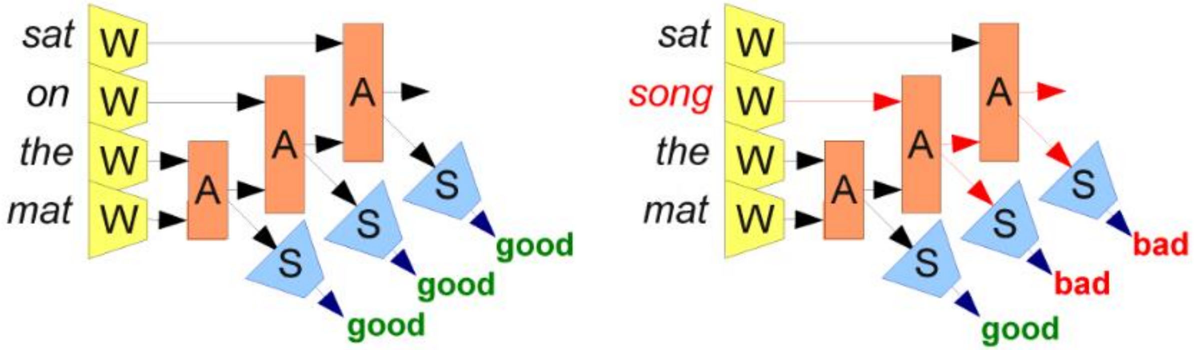
Training strategies

Supervised

- (Socher, 2010, …)

Unsupervised

- In the spirit of the NLP system of (Collobert, Weston, etal., 2008)



Learned representations

– Wikipedia dataset.
– Vocabulary restricted to 1000 words.
– All pairs of the 500 most frequent words were mapped into $\mathcal{R}$.
– Examples of nearest neighbors:

last year	red house	the city	two men
first year	french house	the town	three men
same year	rock house	the church	four men
first day	red court	the village	two children
third year	german house	the state	two women
first season	black house	the country	three children

Still lots of problems…
- Does not scale well.
- Does not induce good parse trees.

(Etter, 2008)

Conclusion

Exploitation

Lots of neural net applications in the coming years

- Learning perceptual tasks with neural nets works quite well

- Data and compute power are here.

Exploration

The statistical machine learning research program

- Discussing the models
  e.g., their approximation properties.
- Discussing the loss functions
  e.g., asymptotic consistency.
- Discussing learning algorithms
  e.g., optimization, large scale.
- Discussing generalization
  e.g., capacity control.

Exploration

The statistical machine learning research program

- Discussing the model
  e.g., their appro… …perties.
- Discussing …
  e.g., a… …ncy.
- Discu… …g algorithms
  e.g., op… …tion, large scale.
- Discussing generalization
  e.g., capacity control.

DONE
(essentially)

Exploration (my two cents)

A new object of study

- A collection of statistical models
- With different input and output spaces
- Endowed with an algebraic structure
  connecting the models and their realizations
  describing how to transfer knowledge across models.

Unstructured training data

- Each example can pertain to a different model.
- The algebraic structure is the glue.
- The glue is connected to reasoning.