Lecture 1: Introduction to Deep Learning

Overview

This class is about:
- deep learning
- application in computer vision and graphics
- applications in natural language processing
- deep reinforcement learning

It will include:
- 12 lectures
- 5 seminars
- 4 big assignments
- A big project work

2006

CAPTCHA

asirra

Computer vision = 60%

$0.6^{12} = 0.00217$

2014

Completed • Swag • 215 teams

Dogs vs. Cats

Wed 25 Sep 2013 – Sat 1 Feb 2014 (8 months ago)

Dashboard ▼

Private Leaderboard - Dogs vs. Cats

This competition has completed. This leaderboard reflects the final standings.

See someon

#	Δ1w	Team Name * in the money	Score	Entries	Last Submission UTC (Best – Las
1	—	Pierre Sermanet *	0.98914	5	Sat, 01 Feb 2014 21:43:19 (-
2	↑26	orchid *	0.98309	17	Sat, 01 Feb 2014 23:52:30
3	—	Owen	0.98171	15	Sat, 01 Feb 2014 17:04:40 (-
4	new	Paul Covington	0.98171	3	Sat, 01 Feb 2014 23:05:20
5	↓3	Maxim Milakov	0.98137	24	Sat, 01 Feb 2014 18:20:58

$$0.989^{12} = 0.875$$

2014

Microsoft Research

Search Mi

Our research Connections Careers About us

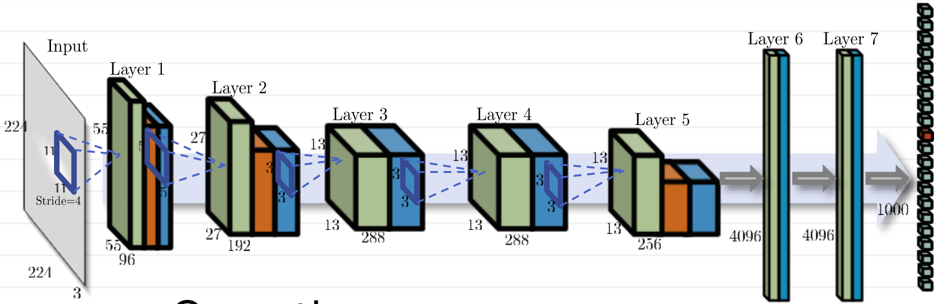
All Downloads Events Groups News People Projects Publications

ASIRRA

asirra

After 8 years of operation, Asirra is shutting down effective October 1, 2014. Thank you to all of our users!

The winner: convolutional networks



Operations:

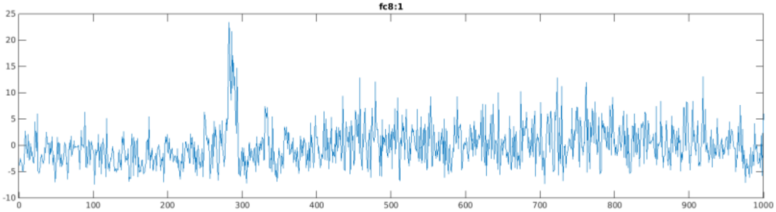
generalized convolutions

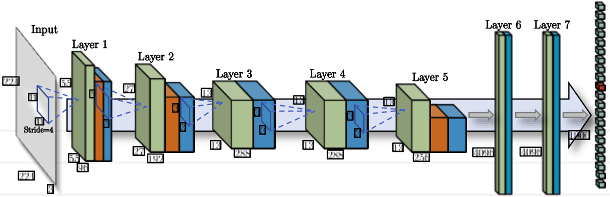
pooling (image resizing)

elementwise non-linearity

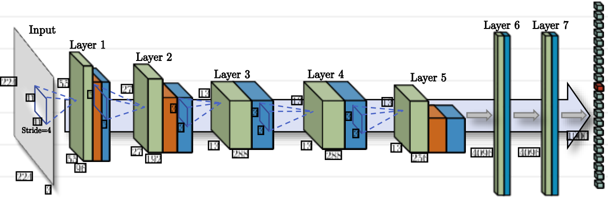
matrix multiplication

Representations

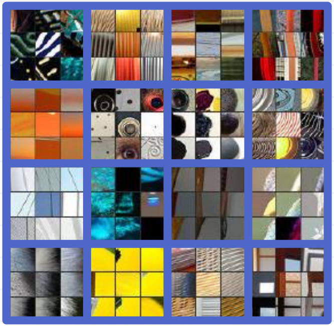




Left-to-right = "smarter"



Layer 1

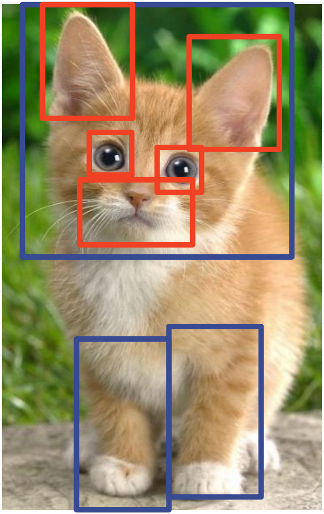


Layer 2

Layer 5

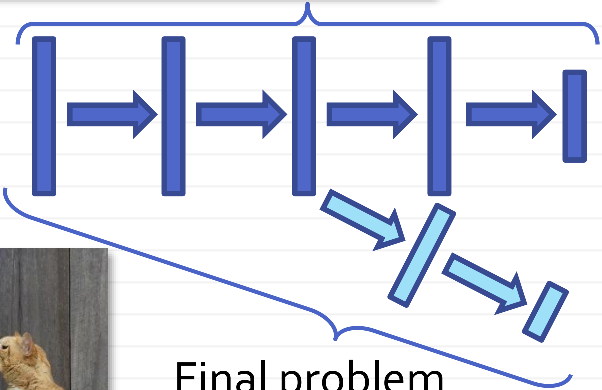
[Zeiler Fergus 14]

High level vision is part-based



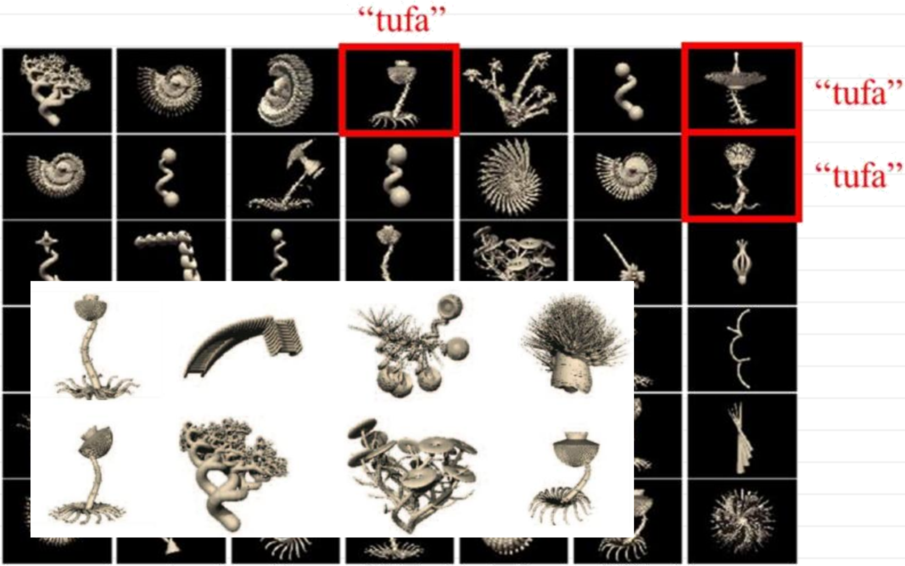
Transfer learning

Task where we have a lot of data



Final problem

Gazoob world



[Tenenbaum et al. Science 2011]

Learning intermediate representations

- The essence of modern "deep learning"
- Is essential for intelligence
- Can be done via supervised, unsupervised and other types of learning
- Has been done all along before "deep learning" revolution

Supervised learning

$$\{x_1, x_2, \ldots, x_M\} \subset R^N$$

$$\{y_1, y_2, \ldots, y_M\} \subset \mathcal{L}$$

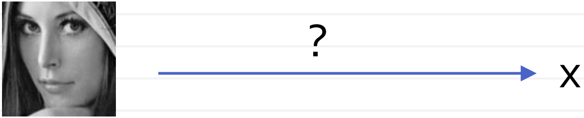
$$f \quad : \quad R^N \to \mathcal{L} \qquad\qquad \mathcal{L} = \{-1, 1\}$$

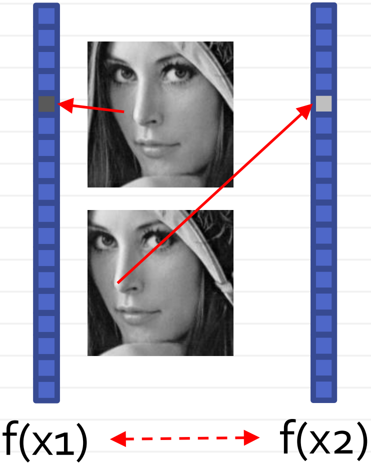
Goal: "recover" *f*.

E.g. linear classifier: $f(x) = \operatorname{sgn}\, w^T x$

Example:  vs. 

Face detection challenge

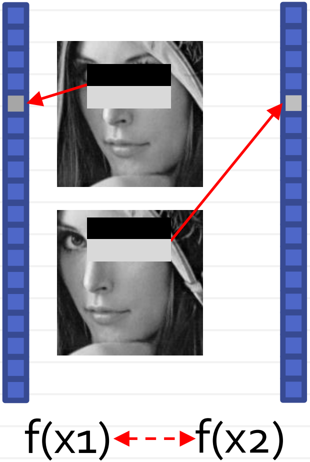




Natural feature mapping:

- Highly non-smooth w.r.t. jitter
- Require lots of training samples

Haar features



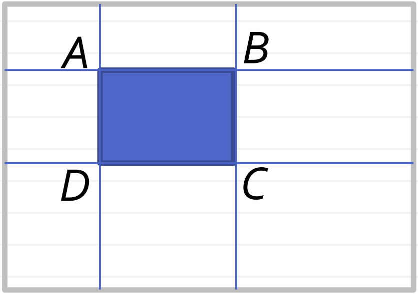
Viola-Jones features:

- Smoother w.r.t. jitter
- Less training examples needed
- *(also fast to compute)*

[Viola Jones, CVPR'01]

Haar features

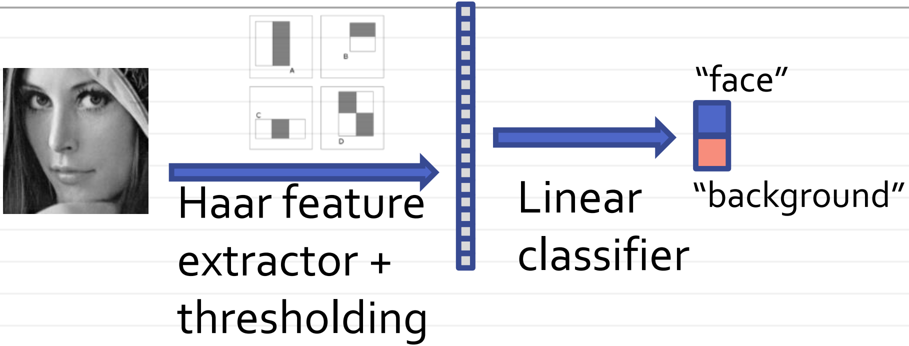
$$F(A) = \iint_{\preceq A} f(x,y)\,dx\,dy$$



$$\iint_{ABCD} f(x,y)\,dx\,dy = \quad F(C) + F(A) - F(B) - F(D)$$

[Viola Jones, CVPR'01]

Viola-Jones detector



- **Non-shallow**, learnable representation (AdaBoost greedy algorithm)
- Cascaded detector for speed
- Arguably, most impactful paper in CV history

[Viola Jones, CVPR'01]

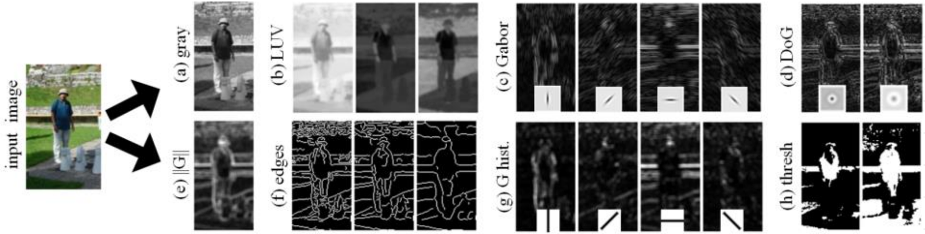
From face detection to pedestrian detection

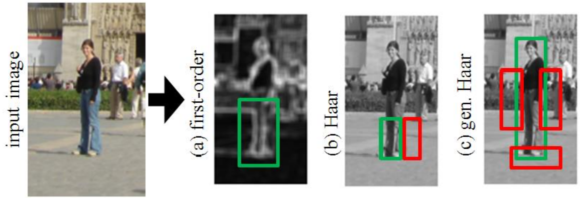
vs.

Good industy-grade performance by Viola-Jones (for frontal faces)

Viola-Jones detector not good enough

Improving pedestrian detection

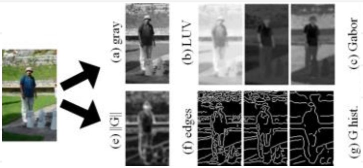




[Dollar et al. BMVC09]

Improved pedestrian detector

"channel features"



Haar feature

classifier

"pedestrian"

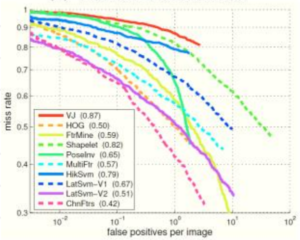
"background"

Hand-crafted

Trained usi

Deep

(c) Caltech Pedestrians



[Dollar, Tu, Perona, Belongie. *Integral Channel Features.* BMVC09]

Then, what is "deep learning"?

- Previous CV systems were "deep", they used multiple layers of representation with success

- The main "novelty" in modern age *deep learning*: **end-to-end joint learning** of multiple (10+) layers

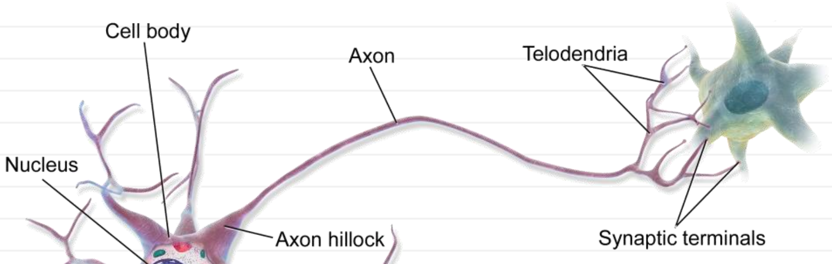
Then, what is “deep learning”?

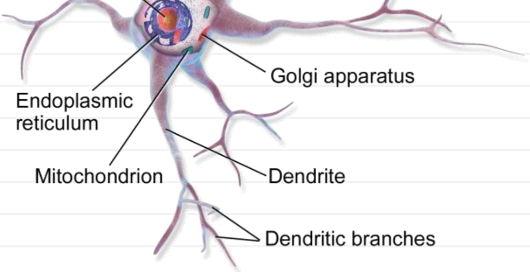
End-to-end joint learning of all layers:

- multiple assemblable blocks
- each block is piecewise-differentiable
- gradient-based optimization
- gradients computed by *backpropagation*

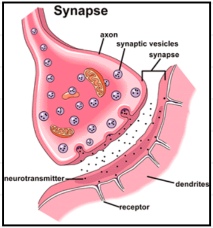
Deep learning “revolution” (2012? – now): rapid engineering improvements following these principles

Neuron model

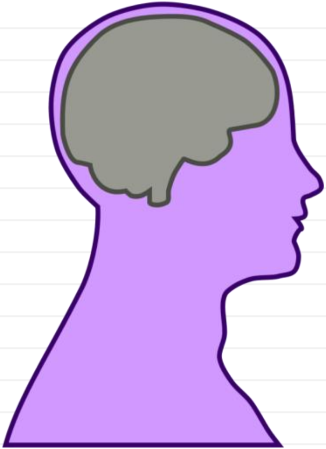




$$x_k = H\left(\sum_i w_{ik} x_i - \tau\right)$$



Brain statistics

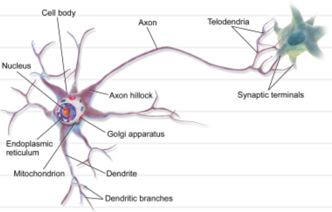


Human brain:

- 100 billion neurons
- average neuron is connected to 1000-10000 other neurons
- 100 trillion synapses
- 10-25% is in visual cortex

Perceptron

[Rosenblatt 1957]: an "artificial neuron"



$$y = H(w^T x)$$

loop over examples

$y = H(w^T x_i);$

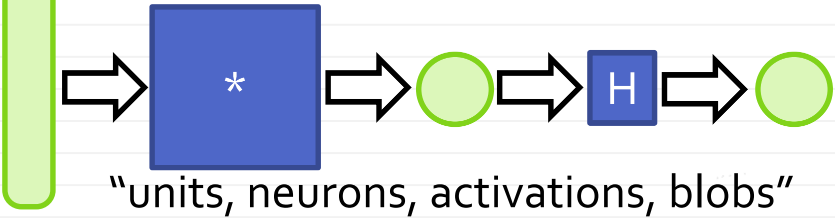
$w = w + 1/2\, x_i * (y_i - y);$

end

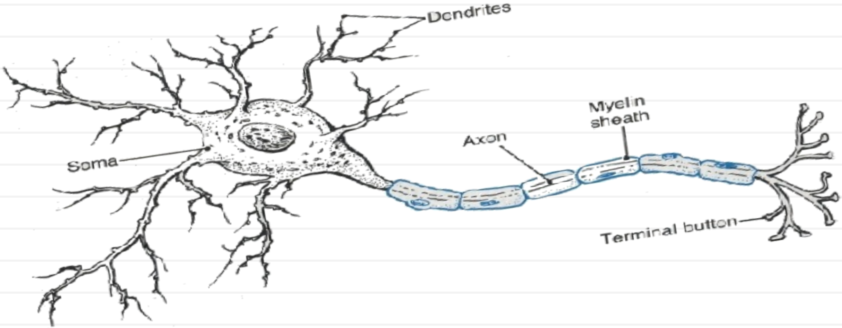
Converges to linear separator of the training data if it exists.

Terminology and graphical language

“operations, layers, transforms”



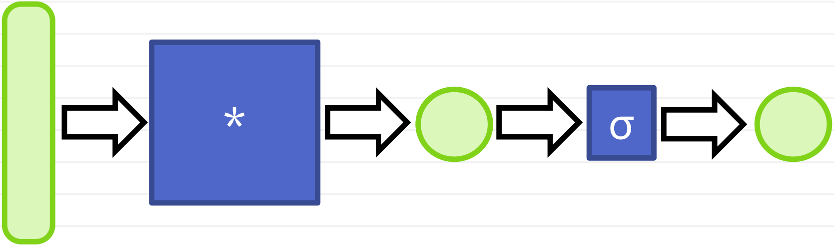
“units, neurons, activations, blobs”



Logistic regression

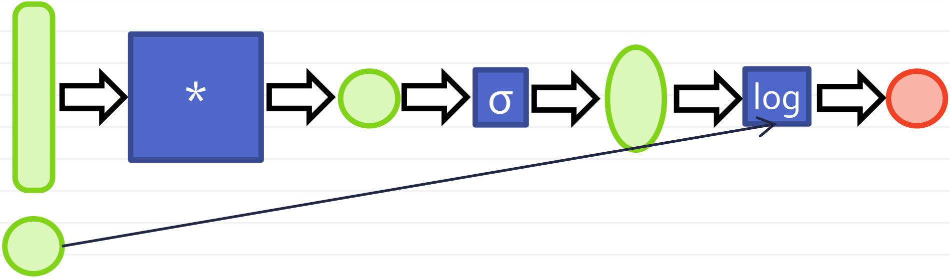
$$P(y(x) = y_i | w) = \frac{1}{1 + e^{-y_i w^T x_i}} = \sigma(y_i w^T x)$$

Same diagram/network:



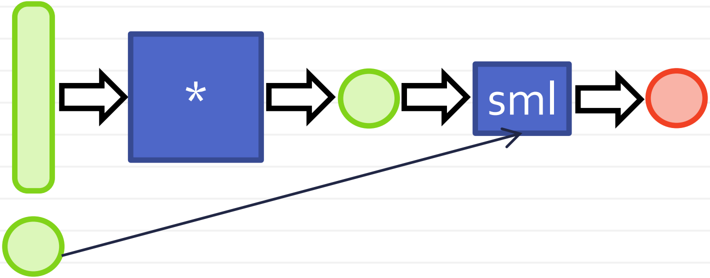
Training logistic regression

$$E(w) = -\sum_{i=1}^{N} \log P(y(x) = y_i | w) = \sum_{i=1}^{N} \log(1 + e^{-y_i w^T x_i})$$



Logistic regression: simplifying training

$$E(w) = -\sum_{i=1}^{N} \log P(y(x) = y_i | w) = \sum_{i=1}^{N} \log(1 + e^{-y_i w^T x_i})$$



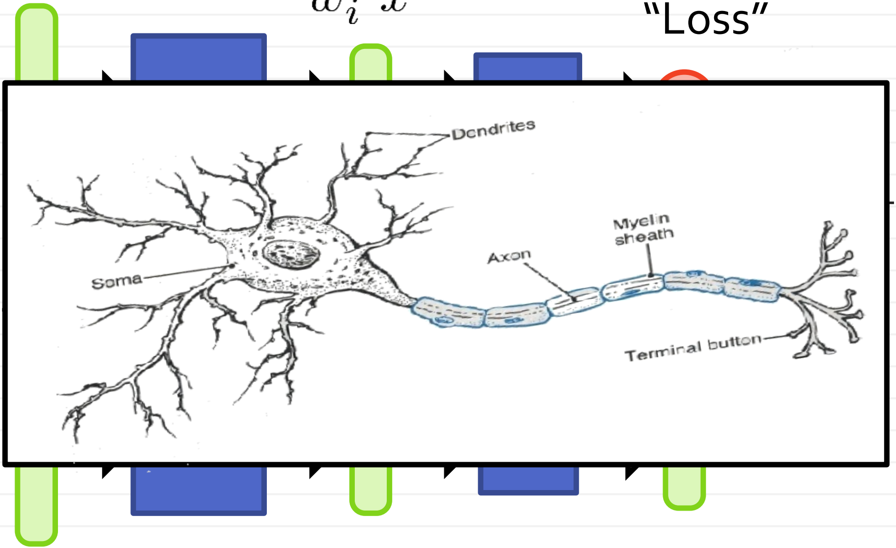
Softmax loss = log loss over softmax/logistic

Multinomial logistic regression

Training:

$w_i^T x$

"Loss"



Biological neuron layers

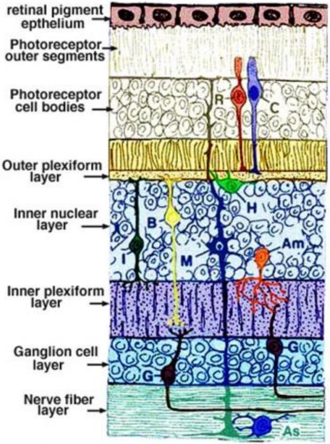


Fig. 5. Scheme of the layers of the developing retina around 5 months' gestation (Modified from Odgen,1989).

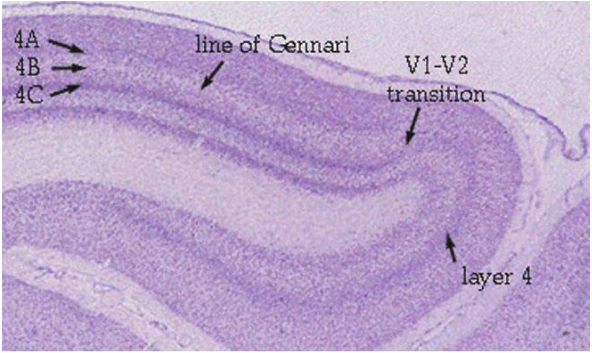
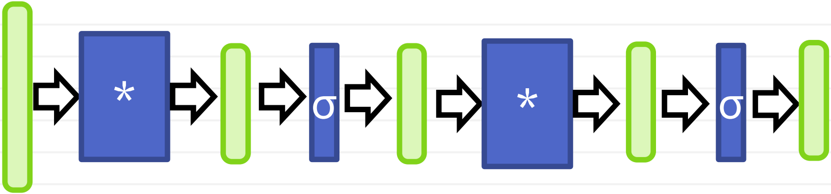


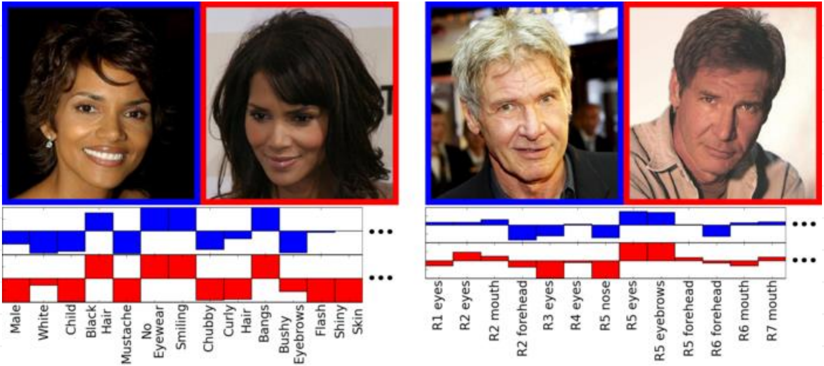
Figure 9. Nissl stained section of the visual cortex to show the border between area 17 (V1) and area 18 (V2).

Multi-layer perceptron idea



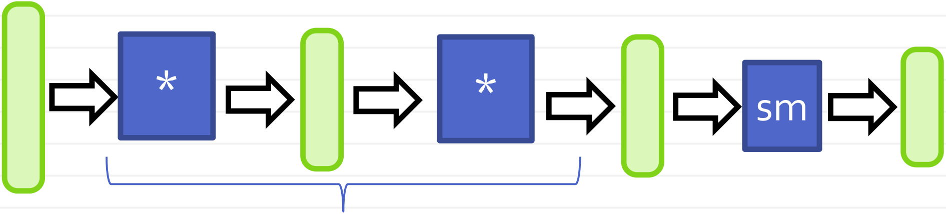
- First layer: parallel logistic regression
- Each predicts presence of some feature in the input
- Second layer is a logistic regression that "weighs" the input of the first layer

Classifier output as features



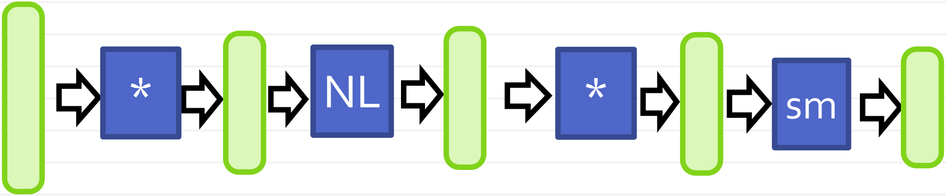
[Kumar et al. **Attribute and Simile Classifiers for Face Verification. ICCV 2009**]

Artificial multilayer networks



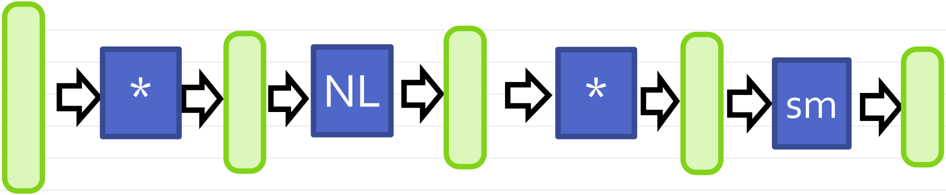
still single matrix multiplication

To get more powerful model need non-linearity:



Adding non-linearities

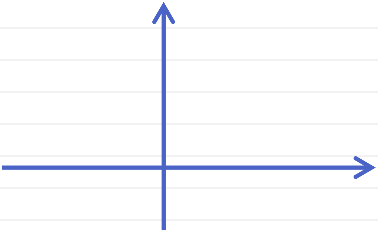
To get more powerful model need non-linearity:



Possible *elementwise* non-linearities:

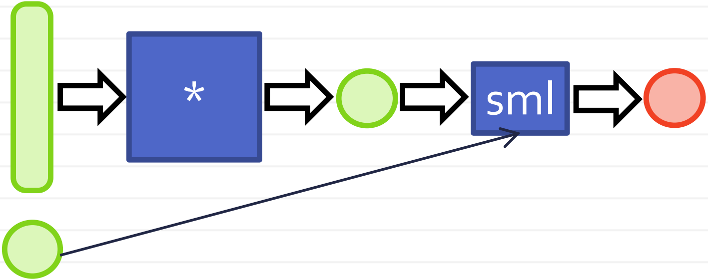
- Heaviside
- Sigmoid(logistic)/tanh
- More recently:

  $ReLu(x) = max(0,x)$



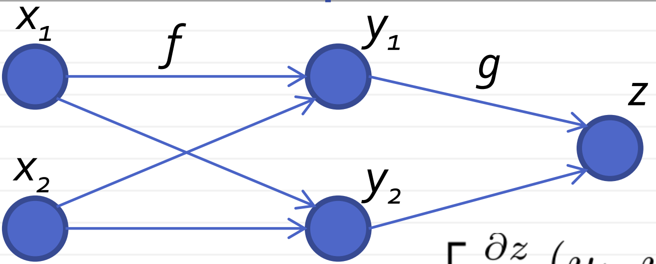
Training logistic regression

$$E(w) = -\sum_{i=1}^{N} \log P(y(x) = y_i | w) = \sum_{i=1}^{N} \log(1 + e^{-y_i w^T x_i})$$



$$\frac{dE}{dw} |_{x_i} = \left(\sigma(y_i w^T x_i) - 1\right) y_i x_i$$

Recap: chainrule



$$\frac{dz}{dy} = \begin{bmatrix} \frac{\partial z}{\partial y_1}(y_1, y_2) \\ \\ \frac{\partial z}{\partial y_2}(y_1, y_2) \end{bmatrix}$$

$$\frac{\partial z}{\partial x_1} = \frac{\partial z}{\partial y_1} \cdot \frac{\partial y_1}{\partial x_1} + \frac{\partial z}{\partial y_2} \cdot \frac{\partial y_2}{\partial x_1}$$

Recap: chainrule

$$\frac{\partial z}{\partial x_1} = \frac{\partial z}{\partial y_1} \cdot \frac{\partial y_1}{\partial x_1} + \frac{\partial z}{\partial y_2} \cdot \frac{\partial y_2}{\partial x_1}$$

$$\frac{\partial z}{\partial x_2} = \frac{\partial z}{\partial y_1} \cdot \frac{\partial y_1}{\partial x_2} + \frac{\partial z}{\partial y_2} \cdot \frac{\partial y_2}{\partial x_2}$$

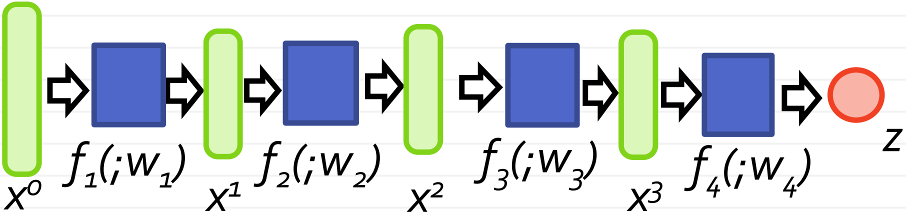
$$\frac{dz}{dx} = \begin{bmatrix} \frac{\partial z}{\partial x_1} \\ \\ \frac{\partial z}{\partial x_2} \end{bmatrix} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_2}{\partial x_1} \\ \\ \frac{\partial y_1}{\partial x_2} & \frac{\partial y_2}{\partial x_2} \end{bmatrix} \begin{bmatrix} \frac{\partial z}{\partial y_1} \\ \\ \frac{\partial z}{\partial y_2} \end{bmatrix}$$

Recap: chainrule

$$\frac{dz}{dx} = \begin{bmatrix} \frac{\partial z}{\partial x_1} \\ \\ \frac{\partial z}{\partial x_2} \end{bmatrix} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_2}{\partial x_1} \\ \\ \frac{\partial y_1}{\partial x_2} & \frac{\partial y_2}{\partial x_2} \end{bmatrix} \begin{bmatrix} \frac{\partial z}{\partial y_1} \\ \\ \frac{\partial z}{\partial y_2} \end{bmatrix}$$

$$\frac{dz}{dx} = \left(\frac{dy}{dx}\right)^T \frac{dz}{dy}$$

Computing deeper derivatives



$$z = f_4(f_3(f_2(f_1(x;\, w_1);\, w_2);\, w_3);\, w_4)$$

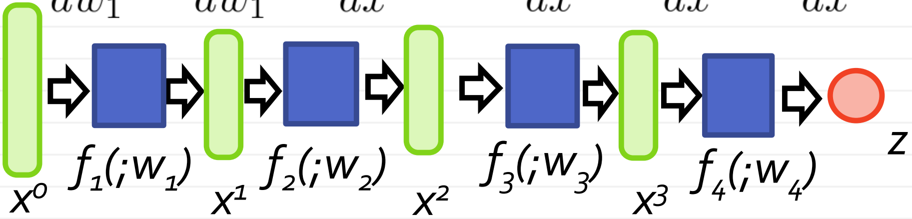
Sequential computation: *backpropagation*

$\frac{dz}{dx^3}, \frac{dz}{dw_4}$ can be computed

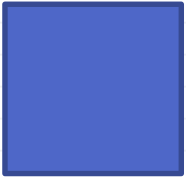
$$\frac{dz}{dw_3} = \frac{dx^3}{dw_3}^T \cdot \frac{dz}{dx^3} \qquad \frac{dz}{dx^2} = \frac{dx^3}{dx^2}^T \cdot \frac{dz}{dx^3}$$

$$\frac{dz}{dw_2} = \frac{dx^2}{dw_2}^T \cdot \frac{dz}{dx^2} \qquad \frac{dz}{dx^1} = \frac{dx^2}{dx^1}^T \cdot \frac{dz}{dx^2}$$

$$\frac{dz}{dw_1} = \frac{dx^1}{dw_1}^T \cdot \frac{dz}{dx^1} \qquad \frac{dz}{dx^0} = \frac{dx^1}{dx^0}^T \cdot \frac{dz}{dx^1}$$



Layer abstraction



Each layer is defined by:

- forward performance: $y = f(x; w)$
- backward performance:

$$z(x) = z(f(x;\ w))$$

$$\frac{dz}{dx} = \frac{dy}{dx}^T \cdot \frac{dz}{dy} \qquad\qquad \frac{dz}{dw} = \frac{dy}{dw}^T \cdot \frac{dz}{dy}$$

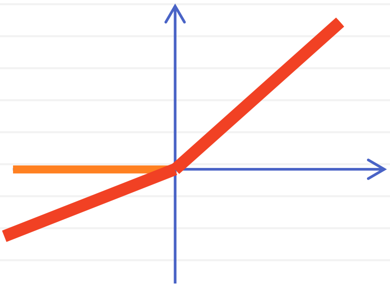
OOP pseudocode of deep learning

```
abstract class Layer {
        params w,dzdw;
        virtual y = forward(x);
        virtual dzdx = backward(dzdy,x,y);
        // should compute dzdw as well

        void update (tau) {
                w = w+tau*dzdw;
        }
};
```

Efficient implementations have to use vector/matrix instructions and work efficiently for minibatches!

Example: “leaky ReLu”



$$f(x;\alpha) = \max(x, \alpha x)$$

$$\frac{dz}{dx} =$$

$$\frac{dz}{d\alpha} =$$

Cornell University Library

arXiv.org > cs > arXiv:1502.01852

Computer Science > Computer Vision and Pattern Recognition

Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification

Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun

(Submitted on 6 Feb 2015)

Computing the partial derivatives

$$z(x) = z(f(x;\, w))$$

$$\frac{dz}{dx} = \frac{dy}{dx}^T \cdot \frac{dz}{dy} \qquad \frac{dz}{dw} = \frac{dy}{dw}^T \cdot \frac{dz}{dy}$$

Options for partial derivatives:

- Finite differences (bad idea)
- Derive gradients analytically (good idea)

Debugging is hard

Gradient checking is a good idea!

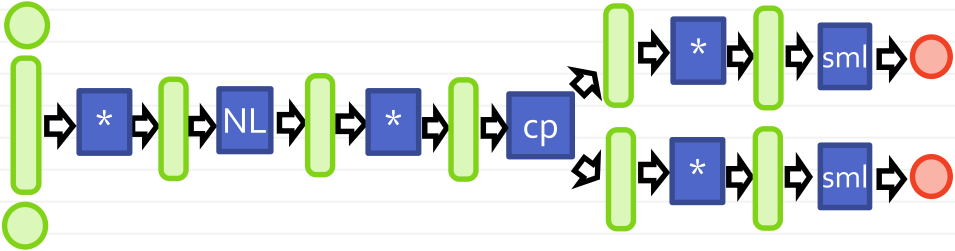
Recap

Deep learning:

- Define each layer
- Assemble a chain of layers
- Loop over minibatches
- For each minibatch find the stochastic gradient and update the parameters (use momentum, etc.)

In fact, chain can easily be replaced with DAG

Example: multitask learning



Typical usecase:

- Two related tasks
- Limited labeled data for the main task
- Lots of labeled data for auxiliary task

Zoo of layers

Multiplicative layer
Convolutional layer

Copy layer
Split layer
Cat layer
Merge layer

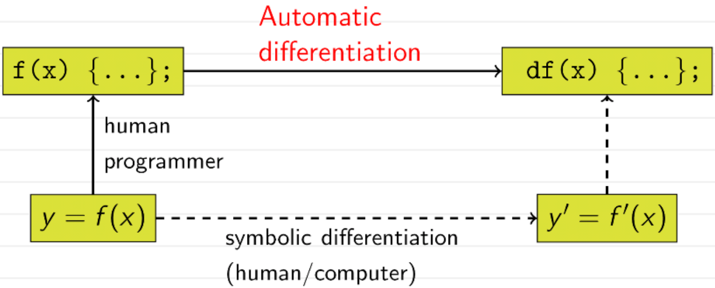
ReLu layer
Sigmoid layer
Softmax layer
Normalization layer
Max-pooling layer

Log-loss layer
Softmax loss layer
Hinge loss layer
L2-loss layer
Contrastive loss layer

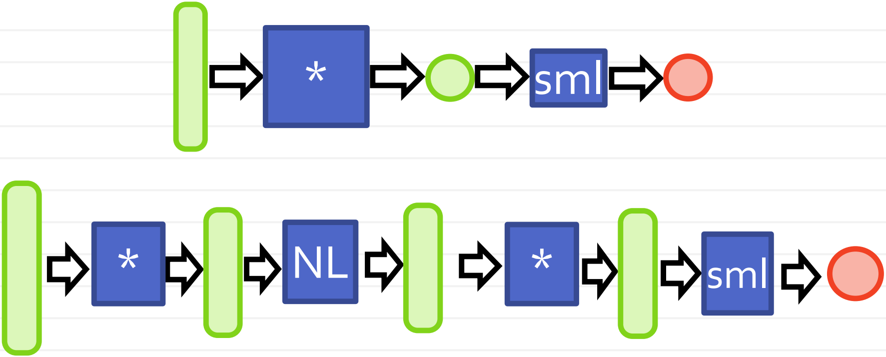
Data providers

Deep learning/symbolic comp. packages

- All packages facilitate stacking layers and defining new layers
- Differ on languages/levels of granularity
- Some allow **symbolic differentiation**
- Some allow **automatic differentiation**



Back to regularization



- Overfitting is severe for deep models (why?)
- The progress on deep learning was "delayed" till huge amount of data

Recap: regularization

4 strategies to avoid overfitting (aka *regularize learning*):

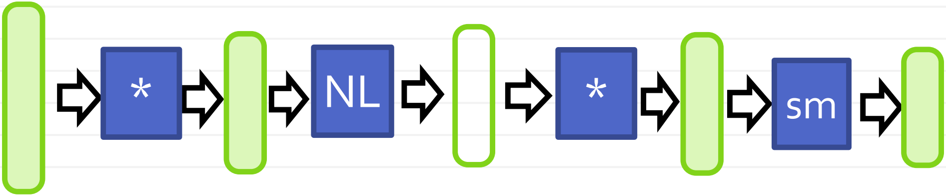
- Pick a “simpler” model (e.g. conv nets)
- Stop optimization early (always keep checking progress on)
- Impose smoothness (weight decay)
- Bag multiple models

Bagging multiple NN

- Different local minima help
- Diversifying architectures helps even more
- Unit weights are often prefered to tuned weights
- (Almost) all classification competitions are won by ensembles of deep models

Dropout regularization

Regularization with a special type of noise:



- At training time, define which units are active at random (*mask*) and which ones are dropped. Divide active unit values by the drop-out probablity

[Srivastava et al. 2011]

How to implement dropout

Define it as a layer!

Forward propagation (train-time only):

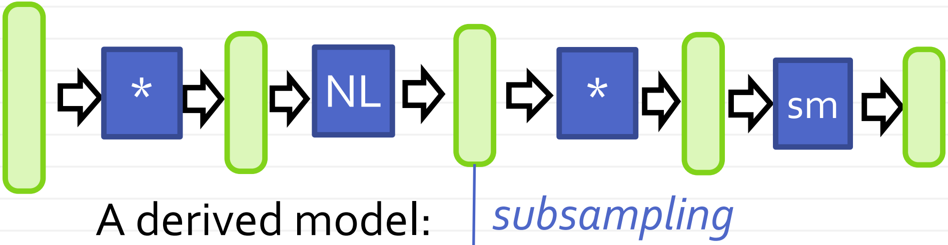
$$n \sim \mathrm{Bernouli}(p) \qquad\qquad y = \frac{1}{p} x \odot n$$

Backward propagation:

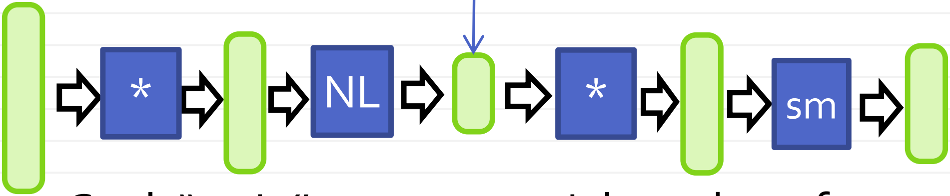
$$\frac{dz}{dx} = \frac{1}{p} \frac{dz}{dy} \odot n$$

Dropout idea: ensemble interpretation

Pseudo-ensemble training:



A derived model:

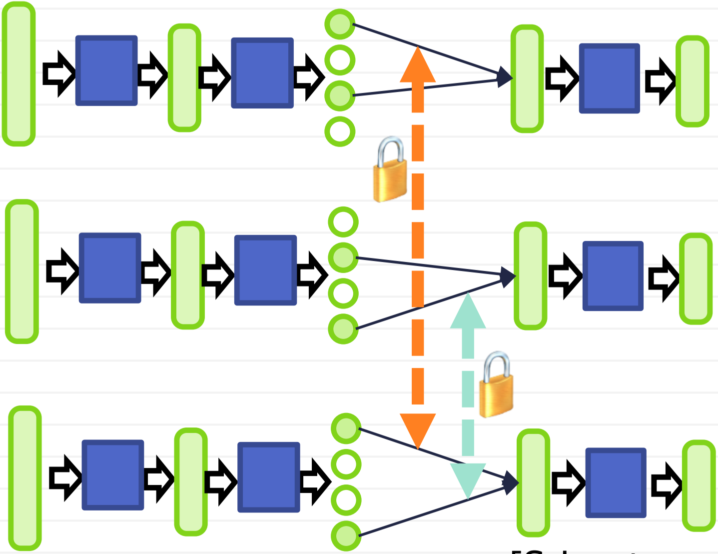


Goal: “train” an exponential number of such reduced models

[Srivastava et al. 2011]

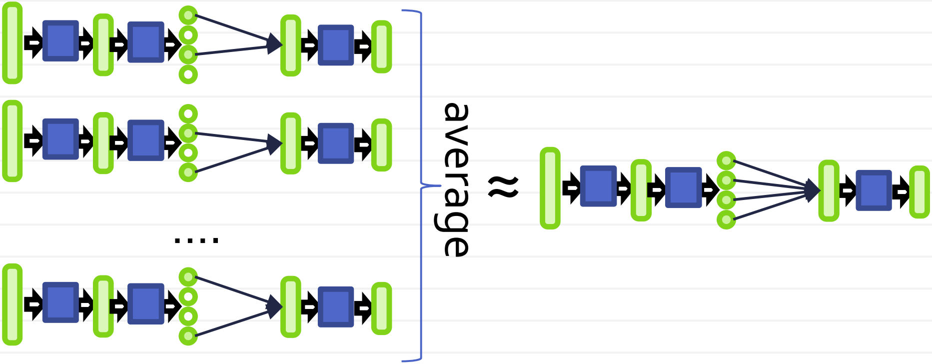
Dropout idea: ensemble interpretation

Training a very big ensemble of models:



[Srivastava et al. 2011]

Dropout idea: ensemble interpretation



- Approximation is not exact
- ...but works well in practice

[Srivastava et al. 2011]

Deep learning: recap

End-to-end joint learning of all layers:

- multiple assembleable blocks
- each block is piecewise-differentiable
- gradient-based optimization
- gradients computed by backpropagation

Big gains in many domains using **supervised learning**