(Convolutional) Neural Network Basics

Connelly Barnes

CS 6501

Slides from E. Shelhamer, J. Donahue, Y. Jia, and R. Girshick, DeepLearning.net (Theano team)

Overview

- Perceptron
- Feedforward artificial neural networks
- Backpropagation
- Stochastic gradient descent
- Convolutional neural networks
  - Max/mean pooling
  - Softmax

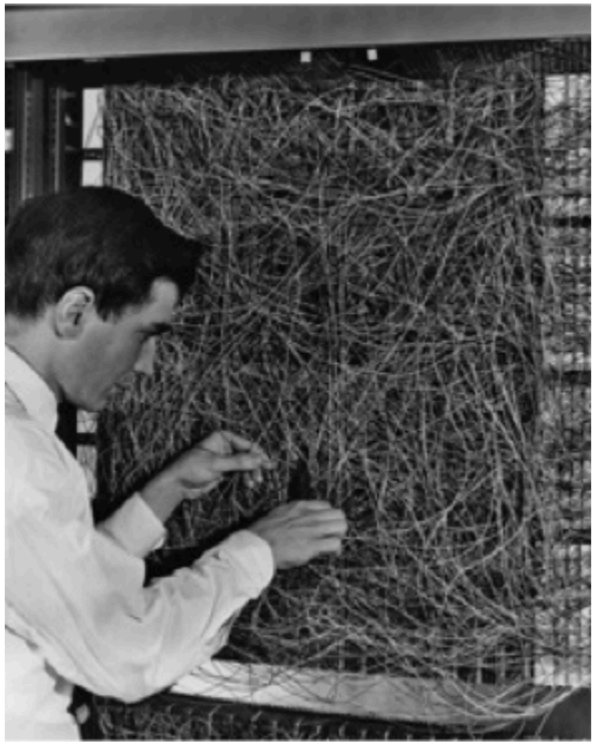
History

1960s: Automatic differentiation, first papers on backpropagation

1958 Rosenblatt proposed perceptrons
1980 Neocognitron (Fukushima, 1980)
1982 Hopfield network, SOM (Kohonen, 1982), Neural PCA (Oja, 1982)
1985 Boltzmann machines (Ackley et al., 1985)
1986 Multilayer perceptrons and $\boxed{\text{backpropagation}}$ (Rumelhart et al., 1986)
1988 RBF networks (Broomhead&Lowe, 1988)
1989 Autoencoders (Baldi&Hornik, 1989), $\boxed{\text{Convolutional network}}$ (LeCun, 1989)
1992 Sigmoid belief network (Neal, 1992)
1993 Sparse coding (Field, 1993)

2000s Sparse, Probabilistic, and Layer-wise models (Hinton, Bengio, Ng)

Is deep learning 2, 20, or 50 years old? What's changed?



Rosenblatt's Perceptron

“Neural” Networks

These models are not how the brain works.

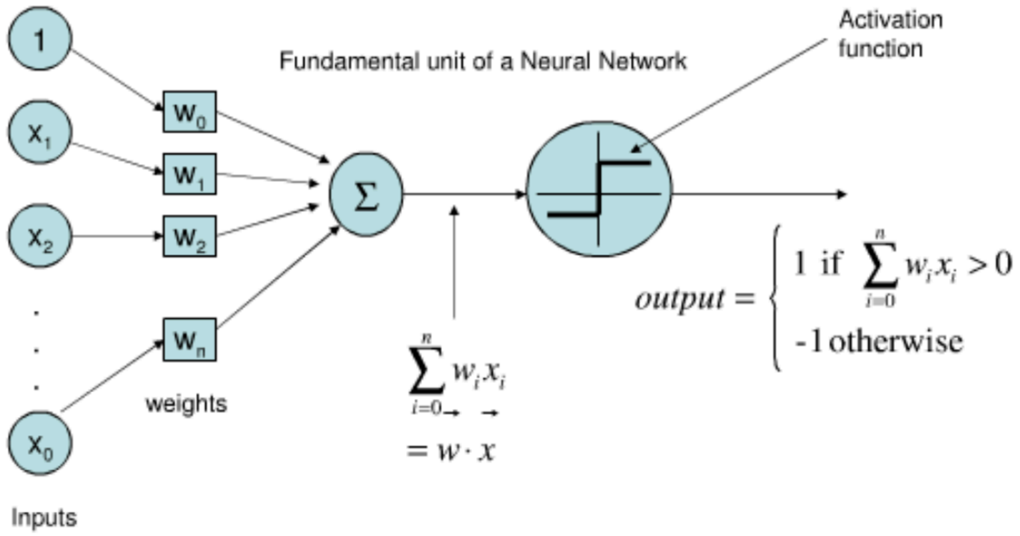
We don’t know how the brain works!

- This isn’t a problem (except for neuroscientists).
- Planes don’t flap and boats don’t swim.

Be wary of “Neural Realism,” or “it just works because it’s like the brain.”

- *network*, not neural network
- *unit*, and not neuron

Perceptron and MLP



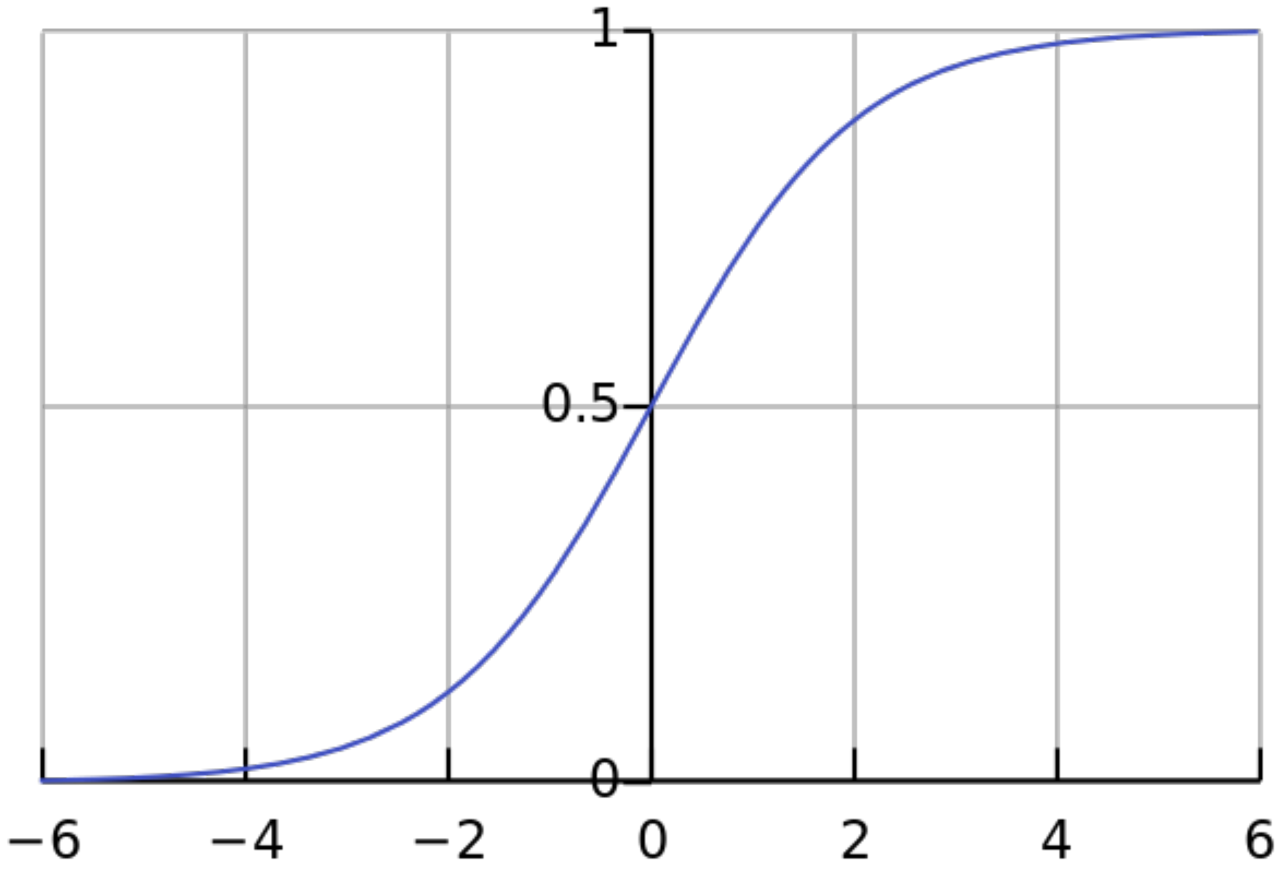
Fundamental unit of a Neural Network
Activation function
1
x_1
x_2
x_0
w_0
w_1
w_2
w_n
weights
Inputs
Σ
$\sum_{i=0}^{n} w_i x_i = \vec{w} \cdot \vec{x}$
$output = \begin{cases} 1 \text{ if } \sum_{i=0}^{n} w_i x_i > 0 \\ -1 \text{ otherwise} \end{cases}$

Multi-Layer Perceptron: Just stack these.

Perceptron: sum and threshold by "step" activation function.

Activation Function: Sigmoid Function

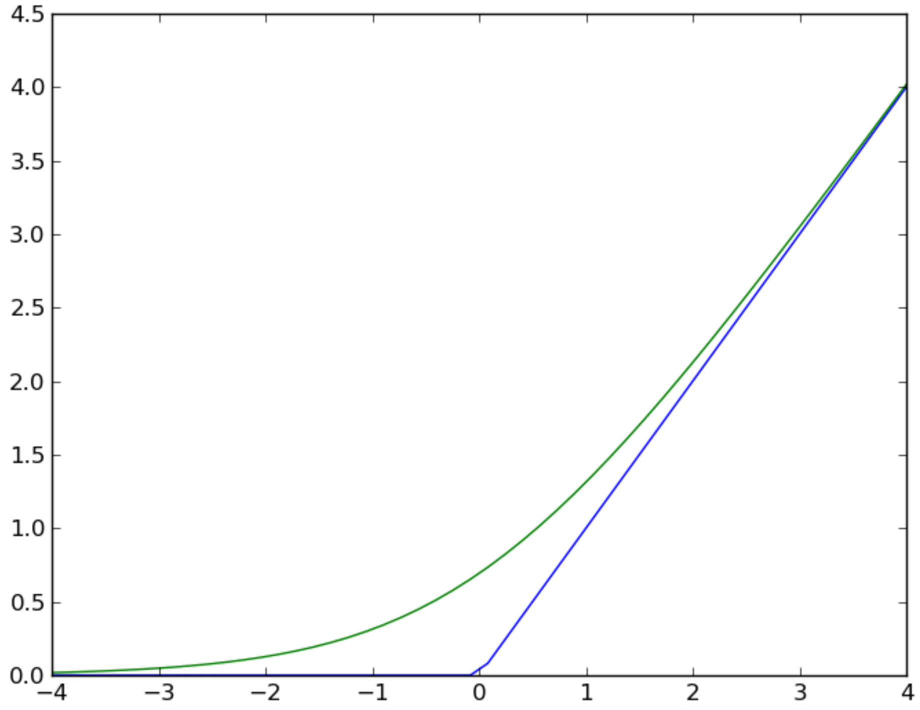
$$s(x) = \frac{1}{1+e^{-x}} \qquad \frac{\partial s}{\partial x} = s(x)\left[1 - s(x)\right]$$



Activation Function: ReLu (REctified Linear Unit)

$$r(x) = \max(0, x)$$

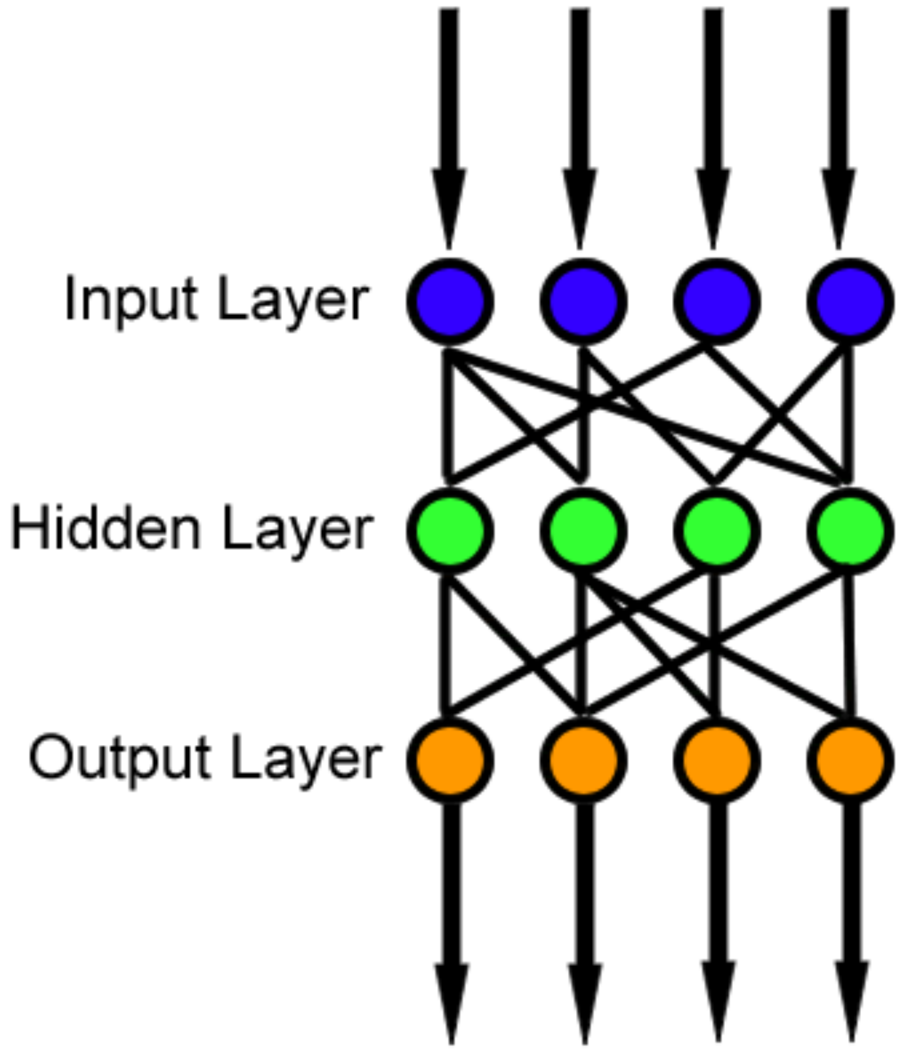
$$\frac{\partial r}{\partial x} = \begin{cases} 0, & x < 0 \\ 1, & x > 0 \end{cases}$$



Activation Function: ReLu

- Gradients do not get too big or small during backpropagation (see vanishing gradient problem)
- Efficient to compute
- Possibly more biologically plausible
- Typically sparse activation (fewer neurons activate)

Multilayer Feedforward Neural Network



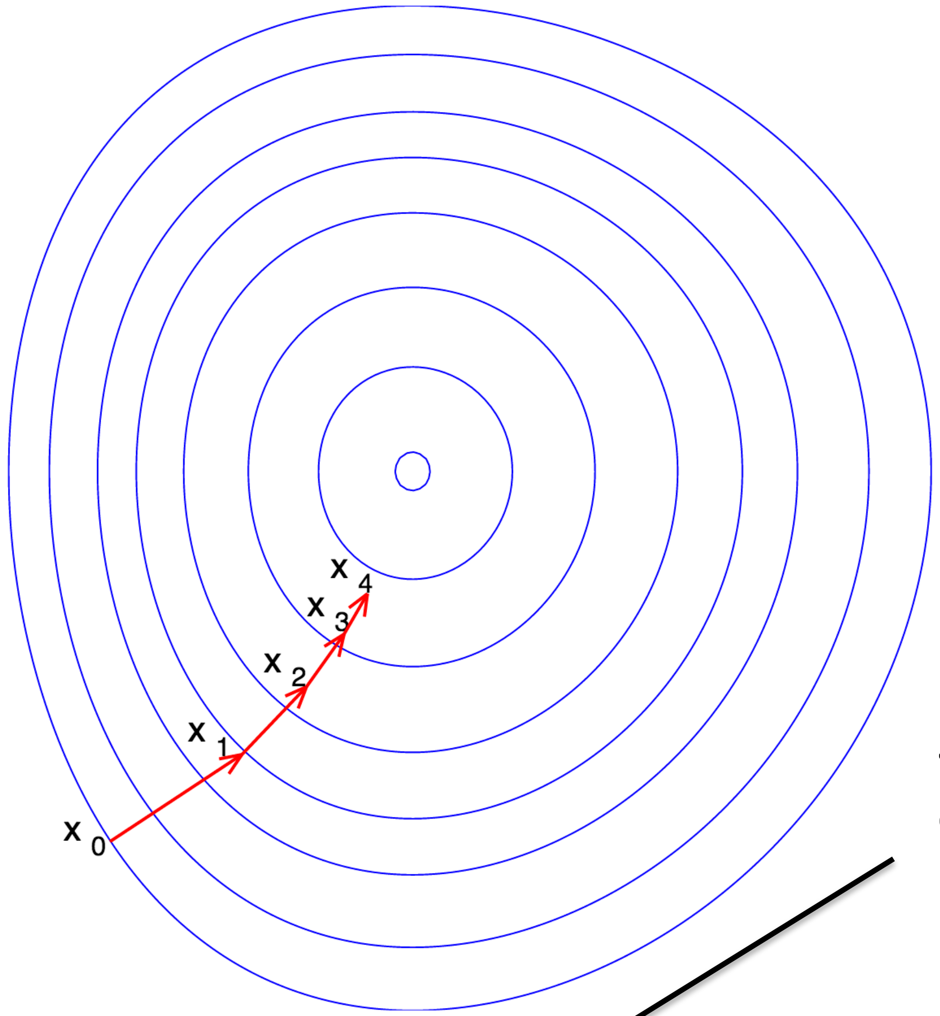
Matrix notation:

$$\mathbf{L}_i = f(\mathbf{W}_i \mathbf{L}_{i-1} + \mathbf{b}_i)$$

Backpropagation

- A special case of reverse mode automatic differentiation (1960s)
- Backpropagation tutorial

Gradient Descent



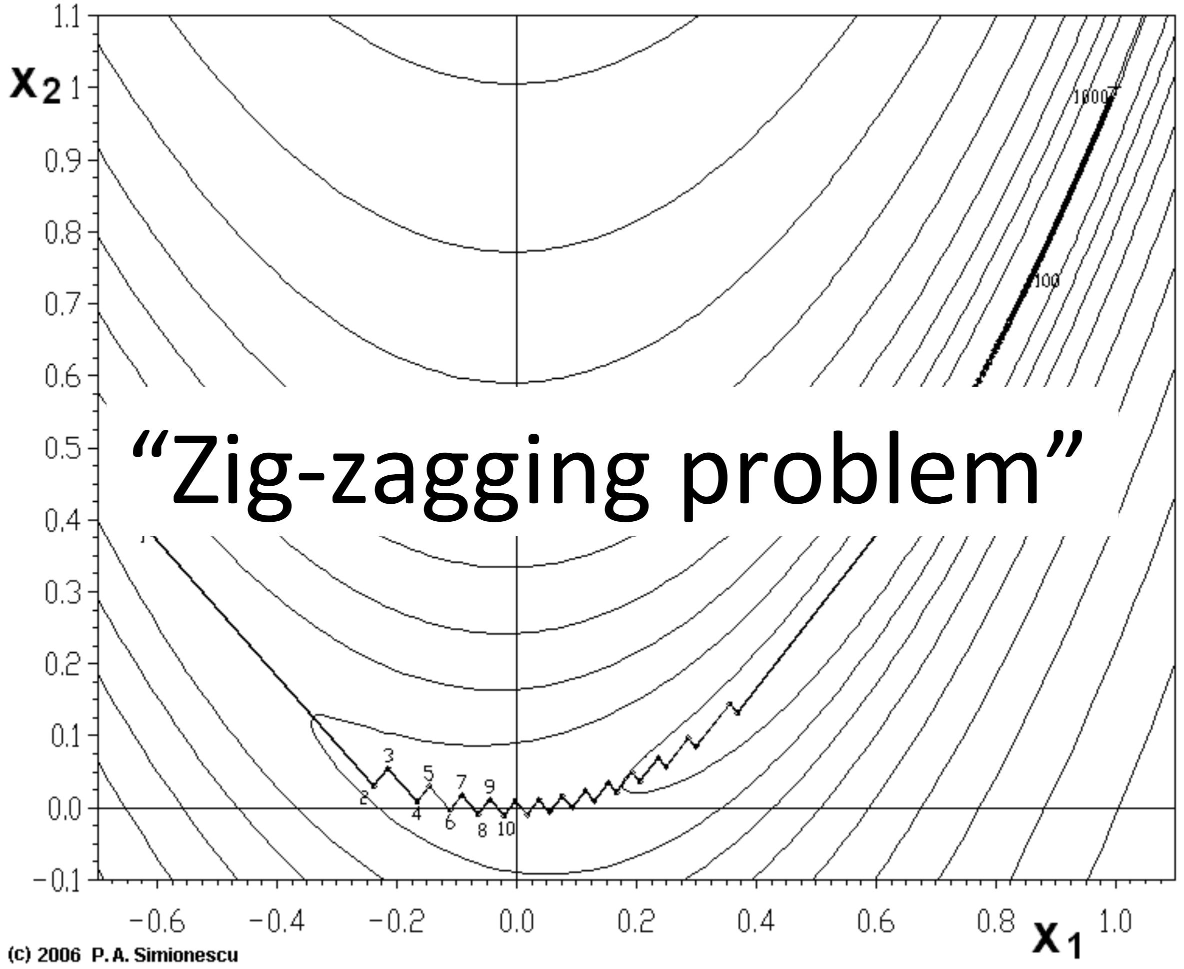
Step size
or
Learning rate

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \gamma_n \nabla F(\mathbf{x}_n)$$

Gradient Descent

- Discussion Questions:
  - What energy function shapes would be easy to minimize with gradient descent? Hard?
  - Should the learning rate be constant? Or change?

Gradient Descent with Energy Functions that have Narrow Valleys



“Zig-zagging problem”

Source: "Banana-SteepDesc" by P.A. Simionescu – Wikipedia English

Gradient Descent with Momentum

$$\mathbf{x}_{n+1} = \mathbf{x}_n + \Delta_n$$

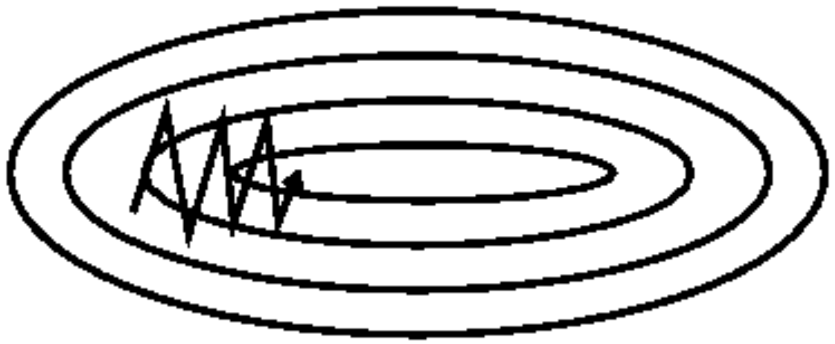
$$\Delta_n = \gamma_n \nabla F(\mathbf{x}_n) + m\Delta_{n-1}$$

Momentum

Could use small value e.g. *m*=0.5 at first

Could use larger value e.g. *m*=0.9 near end of training when there are more oscillations.

Gradient Descent with Momentum


Without Momentum

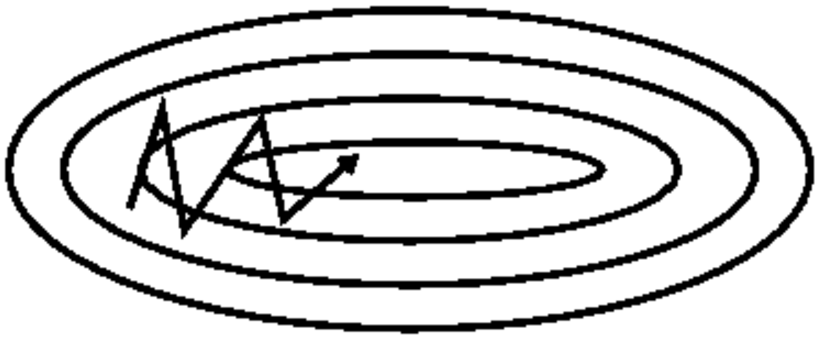

With Momentum

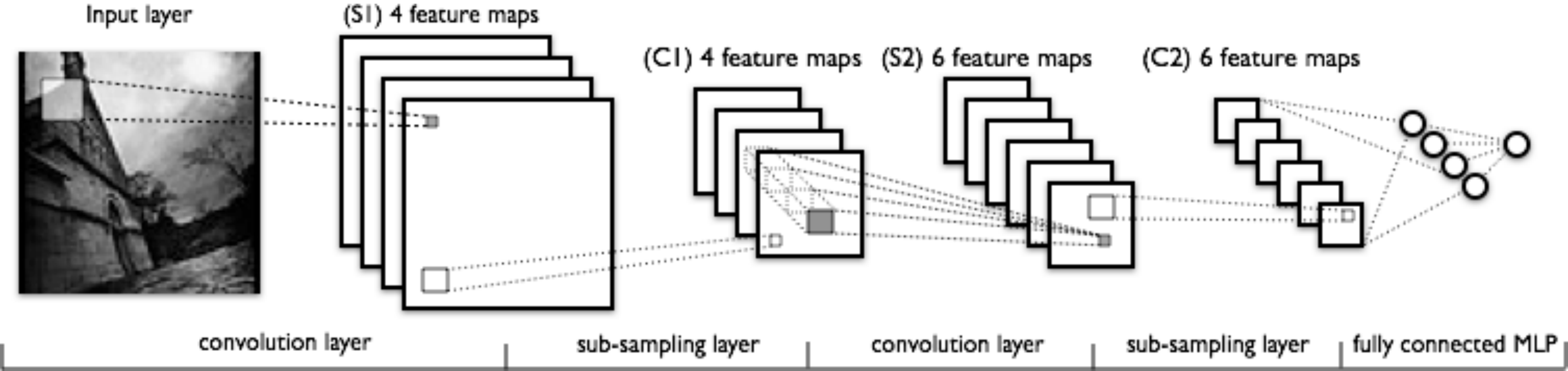
Figure from Genevieve B. Orr, Willamette.edu

Stochastic gradient descent

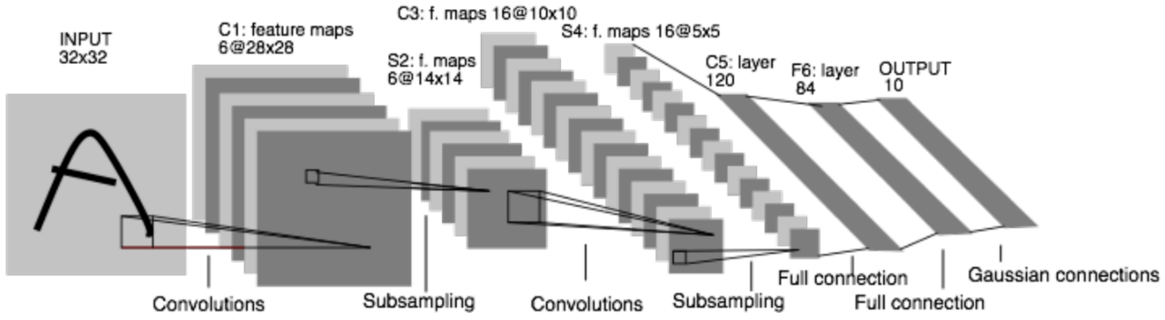
- Stochastic gradient descent (Wikipedia)
- Gradient of sum of n terms where n is large
- Sample rather than computing the full sum
  - Sample size *s* is “mini-batch size”
  - Could be 1 (very noisy gradient estimate)
  - Could be 100 (collect photos 100 at a time to find each noisy “next” estimate for the gradient)
- Use gradient descent

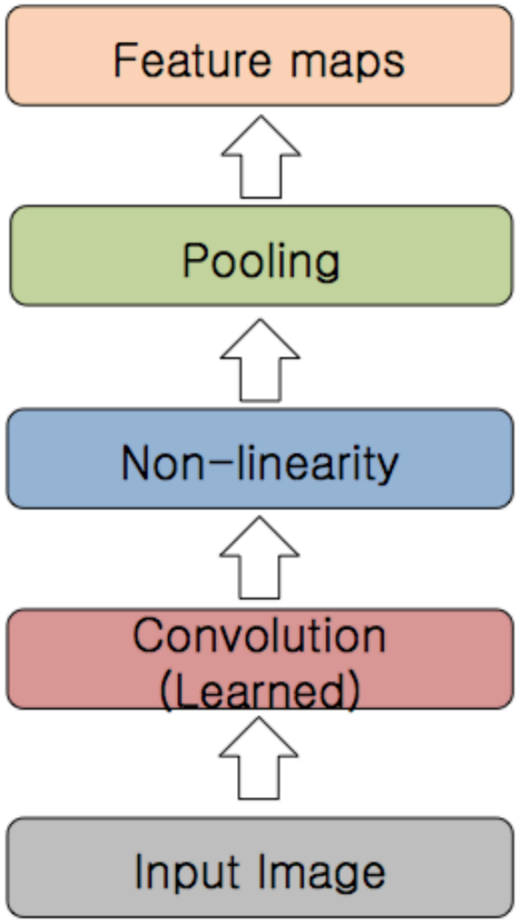
Convolutional Neural Networks

- Similar to multilayer neural network, but weight matrices now have a special structure (Toeplitz or block Toeplitz) due to convolutions.
- The convolutions typically sum over all color channels.



Convolutional Nets

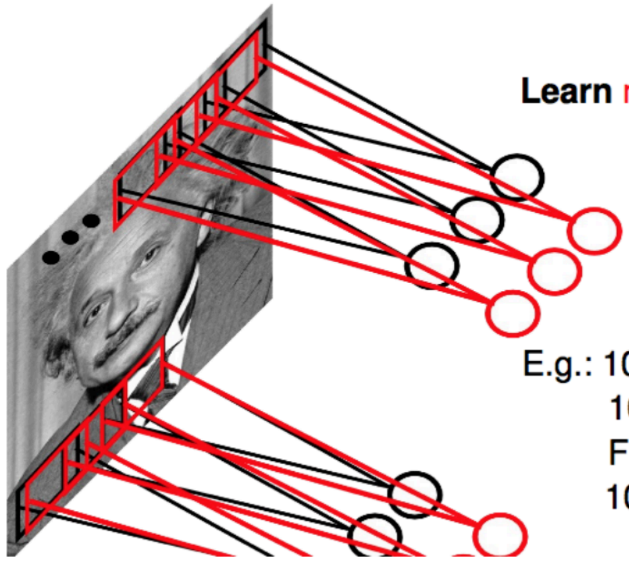




Model structure adapted for vision:

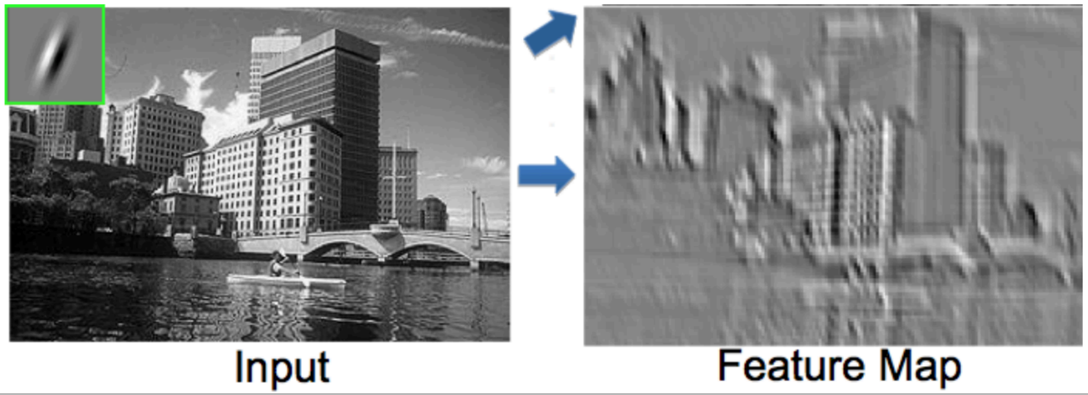
- feature maps keep spatial structure
- pooling / subsampling increases “field of view”
- parameters are shared across space / translation invariance

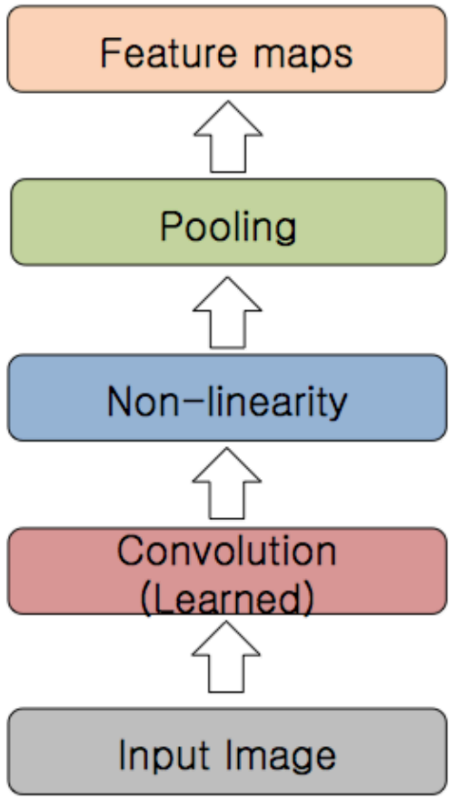
Convolutional / Filtering



Learn multiple filters.

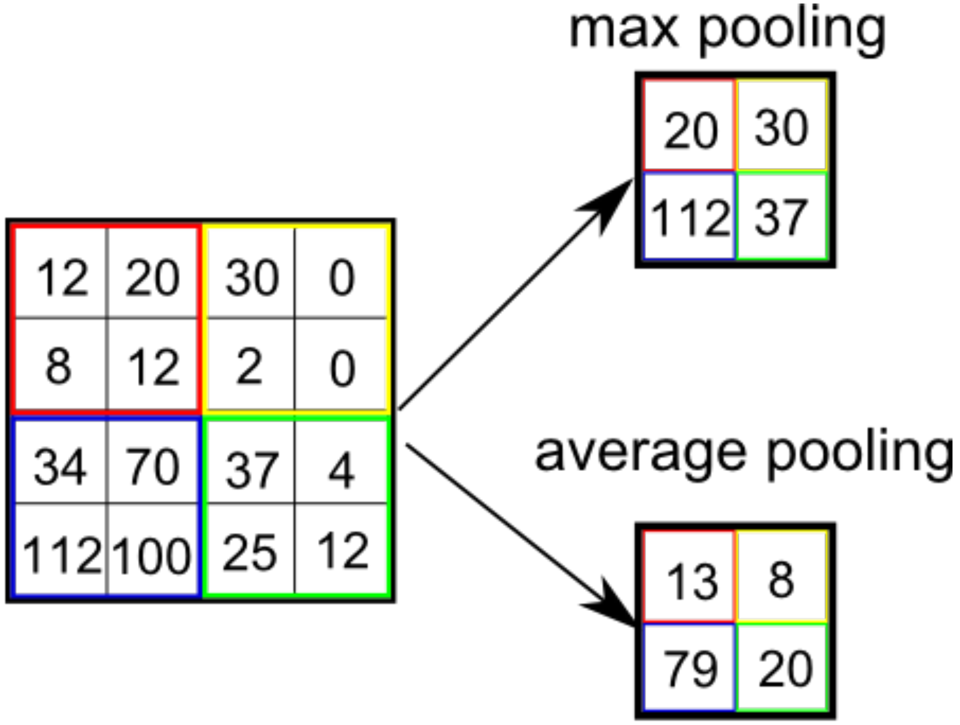
E.g.: 1000x1000 image
100 Filters
Filter size: 10x10
10K parameters





Max/average pooling

- "Downsampling" using max() operator
- Downsampling factor *f* could differ from neighborhood size *N* that is pooled over.



Max/average pooling

- For max pooling, backpropagation just propagates error back to to whichever neuron had the maximum value.
- For average pooling, backpropagation splits error equally among all the input neurons.

Softmax

- Often used in final output layer to convert neuron outputs into a class probability scores that sum to 1.
- For example, might want to convert the final network output to:
  - P(dog) = 0.2   (Probabilities in range [0, 1])
  - P(cat) = 0.8
  - (Sum of all probabilities is 1).

Softmax

- Softmax takes a vector **z** and outputs a vector of the same length.

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad \text{for } j = 1, \ldots, K.$$

$$\frac{\partial}{\partial q_k}\sigma(\mathbf{q}, i) = \cdots = \sigma(\mathbf{q}, i)(\delta_{ik} - \sigma(\mathbf{q}, k))$$