# Neural Networks

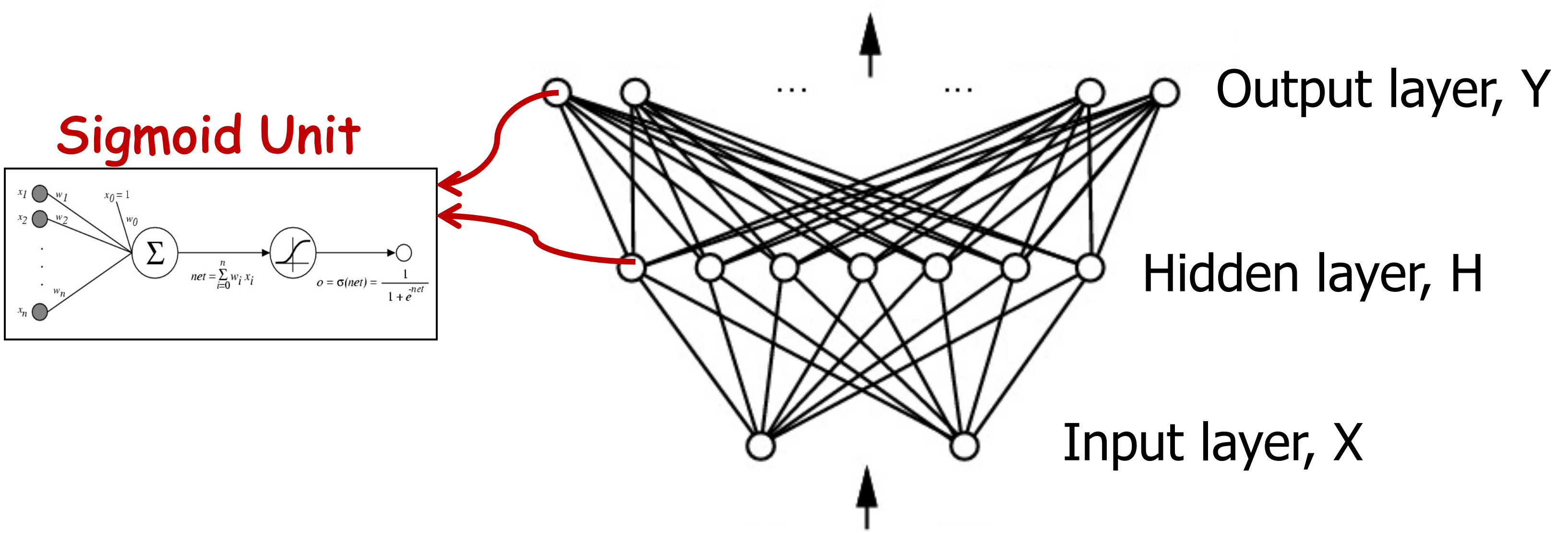
Aarti Singh

Machine Learning 10-315
Feb 16, 2022

# Neural Networks to learn f: X → Y

- f can be a **non-linear** function
- X (vector of) continuous and/or discrete variables
- Y (**vector** of) continuous and/or discrete variables

- Neural networks - Represent f by *network* of sigmoid (more recently ReLU – next lecture) units :



# Training Neural Networks – l2 loss

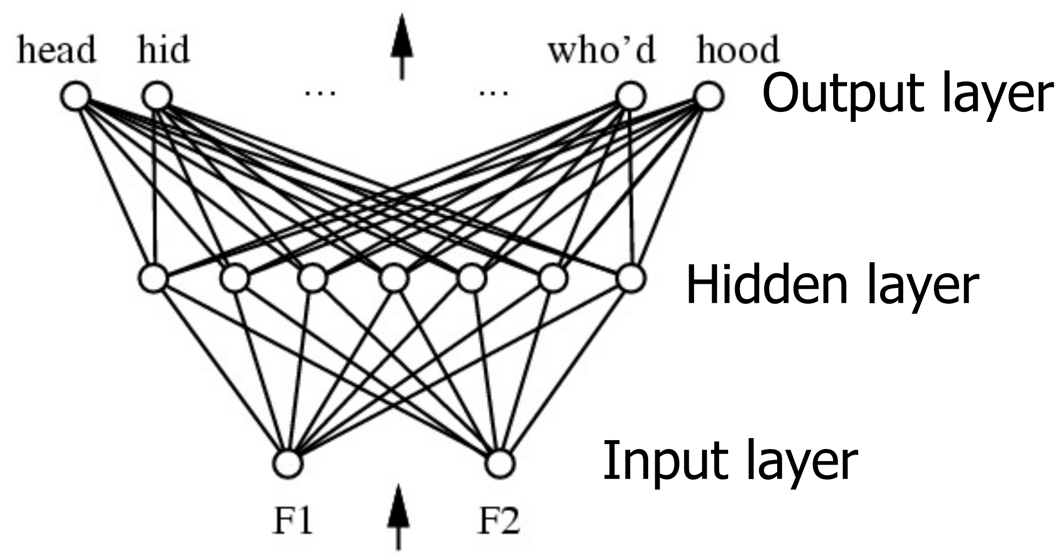
Train weights of all units to minimize sum of squared errors of predicted network outputs

$$W \leftarrow \arg\min_{W} \underbrace{\sum_{l} (y^l - \hat{f}(x^l))^2}_{E[W]}$$

Output of learned neural network

E[ *W*]

- Objective E[ *W*] is no longer convex in *W*.
- Still use Gradient descent to minimize E[ *W*].
- Training is slow with lot of data and lot of weights!



# Stochastic gradient descent

Stochastic gradient descent (SGD): Simplify computation by using a single data point at each iteration (instead of sum over all data points)

$$W \leftarrow \arg\min_{W} \underbrace{\sum_{l} (y^l - \hat{f}(x^l))^2}_{E[W]}$$

At each iteration of gradient descent

- Approximate E[W] $\approx (y^l - \hat{f}(x^l))^2$
- Stochastic Gradient =

Cycle through all points, then restart OR choose random data point at each iteration

# Backpropagation

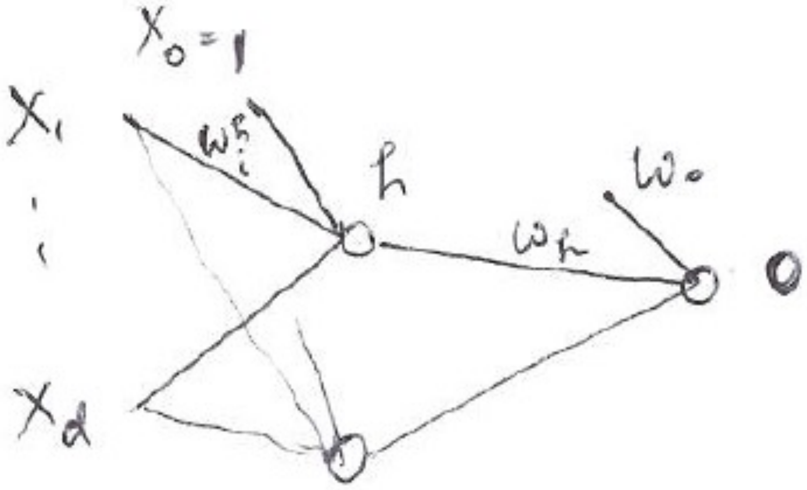
Backpropagation: Efficient implementation of (Stochastic) Gradient descent for Neural networks

chain rule for gradients

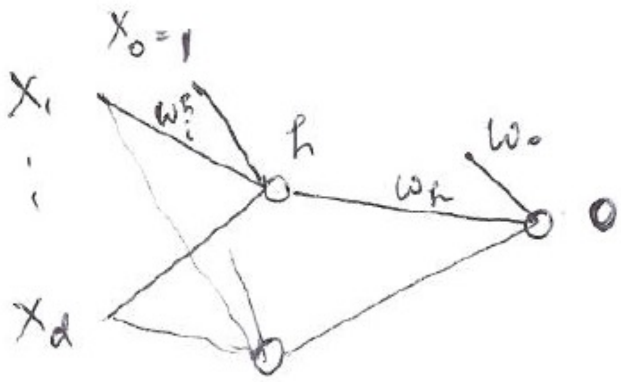
+

layer-wise computation

(going backward from output to input)



# Gradient Descent for 1 hidden layer 1 output NN




$$o = \sigma\left(w_0 + \sum_h w_h o_h\right)$$

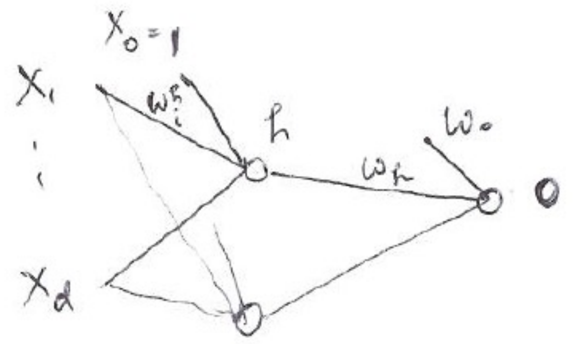
$$o_h = \sigma\left(w_0^h + \sum_i w_i^h x_i\right)$$

Gradient of the output with respect to one **final** layer weight $w_h$

$$\frac{\partial o}{\partial w_h} = o(1-o)o_h$$

# Backpropagation Algorithm
## using Stochastic gradient descent

1 final output unit



Initialize all weights to small random numbers.
Until satisfied, Do

- For each training example, Do

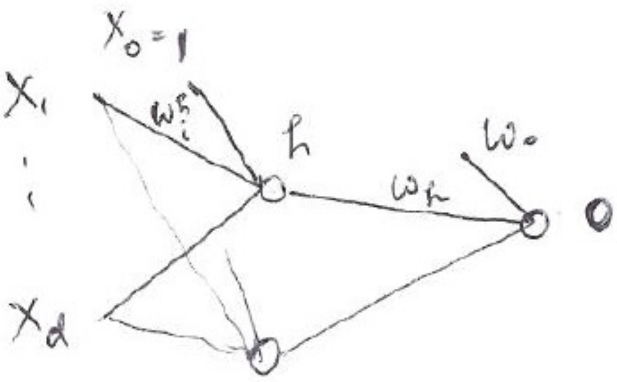
  1. Input the training example to the network and compute the network outputs → Using Forward propagation
  2. $$\delta \leftarrow o(1-o)(y-o)$$
  3. 
  4. Update each network weight $w_h$

$$w_h \leftarrow w_h + \Delta w_h$$

where

$$\Delta w_h = \eta \delta o_h$$

# Gradient Descent for 1 hidden layer 1 output NN




$$o = \sigma\left(w_0 + \sum_h w_h o_h\right)$$

$$o_h = \sigma\left(w_0^h + \sum_i w_i^h x_i\right)$$

Gradient of the output with respect to one **final** layer weight $w_h$

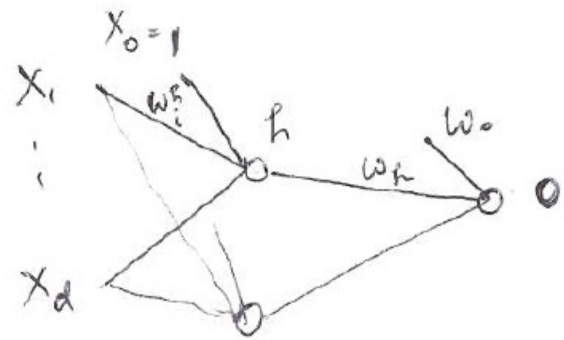
$$\frac{\partial o}{\partial w_h} = o(1-o)o_h$$

Gradient of the output with respect to one **hidden** layer weight $w_i^h$

$$\frac{\partial o}{\partial w_i^h} = \frac{\partial o}{\partial o_h} \cdot \frac{\partial o_h}{\partial w_i^h} = o(1-o)w_h \cdot o_h(1-o_h)x_i$$

# Backpropagation Algorithm
## using Stochastic gradient descent

1 final output unit



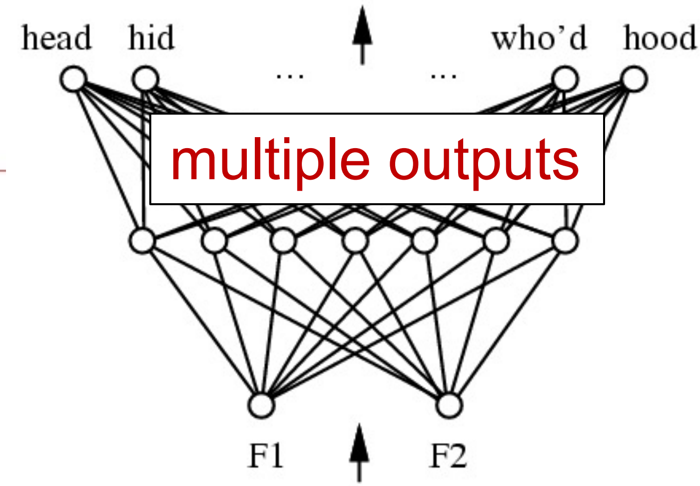

Initialize all weights to small random numbers.
Until satisfied, Do

- For each training example, Do
  1. Input the training example to the network and compute the network outputs → Using Forward propagation
  2. $$\delta \leftarrow o(1-o)(y-o)$$
  3. For each hidden unit $h$
  $$\delta_h \leftarrow o_h(1-o_h) w_h \delta$$
  4. Update each network weight $w_{i,j}$
  $$w_{i,j} \leftarrow w_{i,j} + \Delta w_{i,j}$$
  where
  $$\Delta w_{i,j} = \eta \delta_j o_i$$

$w_{ij}$ = wt from i to j

Note: if i is input variable, $o_i = x_i$

# Backpropagation Algorithm using Stochastic gradient descent




Initialize all weights to small random numbers.
Until satisfied, Do

- For each training example, Do
  1. Input the training example to the network and compute the network outputs → Using Forward propagation
  2. For each output unit $k$

$$\delta_k \leftarrow o_k(1 - o_k)(y_k - o_k)$$

  3. For each hidden unit $h$

$$\delta_h \leftarrow o_h(1 - o_h) \sum_{k \in outputs} w_{h,k} \delta_k$$

  4. Update each network weight $w_{i,j}$

$$w_{i,j} \leftarrow w_{i,j} + \Delta w_{i,j}$$

where

$$\Delta w_{i,j} = \eta \delta_j o_i$$

$y_k$ = label of current training example for output unit k

$o_k$ or $o_h$ = unit output (obtained by forward propagation)

$w_{ij}$ = wt from i to j

Note: if i is input variable, $o_i = x_i$

# More on Backpropagation

- Gradient descent over entire *network* weight vector
- Will find a local, not necessarily global error minimum
  - In practice, often works well (can run multiple times)
- Minimizes error over *training* examples
  - Will it generalize well to subsequent examples?
- Training can take thousands of iterations $\rightarrow$ slow!
- Using network after training is very fast

Objective/Error no longer convex in weights

# HW2

- Cross-entropy error metric for multi-class classification

$$-\sum_k y_k \, log \, \widehat{y_k}$$ loss for single data point

One-hot encoding – encode label as a vector $[y_1, y_2, \ldots y_K]$

where $y_k = 1$ if label is k and 0 otherwise

Interpret vector as probability distribution

# HW2

- Classification – cross-entropy error metric for multi-class classification

$$-\sum_k y_k \log \widehat{y_k}$$

Entropy of a random variable X:

$E_{X\sim p}[-\log p(X)]$ small p(X) => more information

-log p(X) = number of bits needed to encode an outcome X when we know true distribution p

Cross-entropy = expected number of bits needed to encode a random draw of X when using distribution q

$E_{X\sim p}[-\log q(X)]$ Minimized when q=p

# HW2

- Classification – cross-entropy error metric for multi-class classification

$$-\sum_k y_k \, log \, \widehat{y_k}$$

Cross-entropy = expected number of bits needed to encode a random draw of X when using distribution q

$$E_{X\sim p}[-\log q(X)]$$

Interpret one-hot-encoding $y$ and $\hat{y}$ as distributions.

# HW2

- Can implement backpropagation with matrix-vector products – uses matrix-vector calculus heavily

Caution: Denominator vs Numerator layout

https://en.wikipedia.org/wiki/Matrix_calculus

# Poll

Which of the following classifiers are discriminative?

A. Gaussian Naïve Bayes

B. Logistic Regression

C. Neural Networks

Which of these classifiers can exactly represent an XOR function?

# Deep Networks

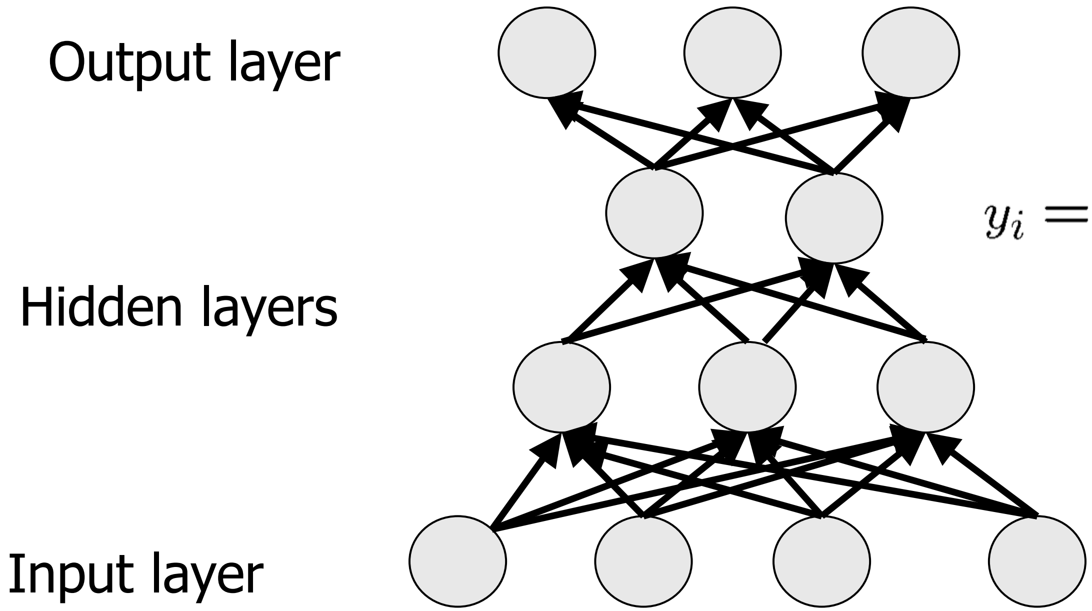
Aarti Singh

Machine Learning 10-315
Feb 16, 2022

Slides Courtesy: Barnabas Poczos, Ruslan Salakhutdinov, Joshua Bengio, Geoffrey Hinton, Yann LeCun, Pat Virtue

# Deep architectures

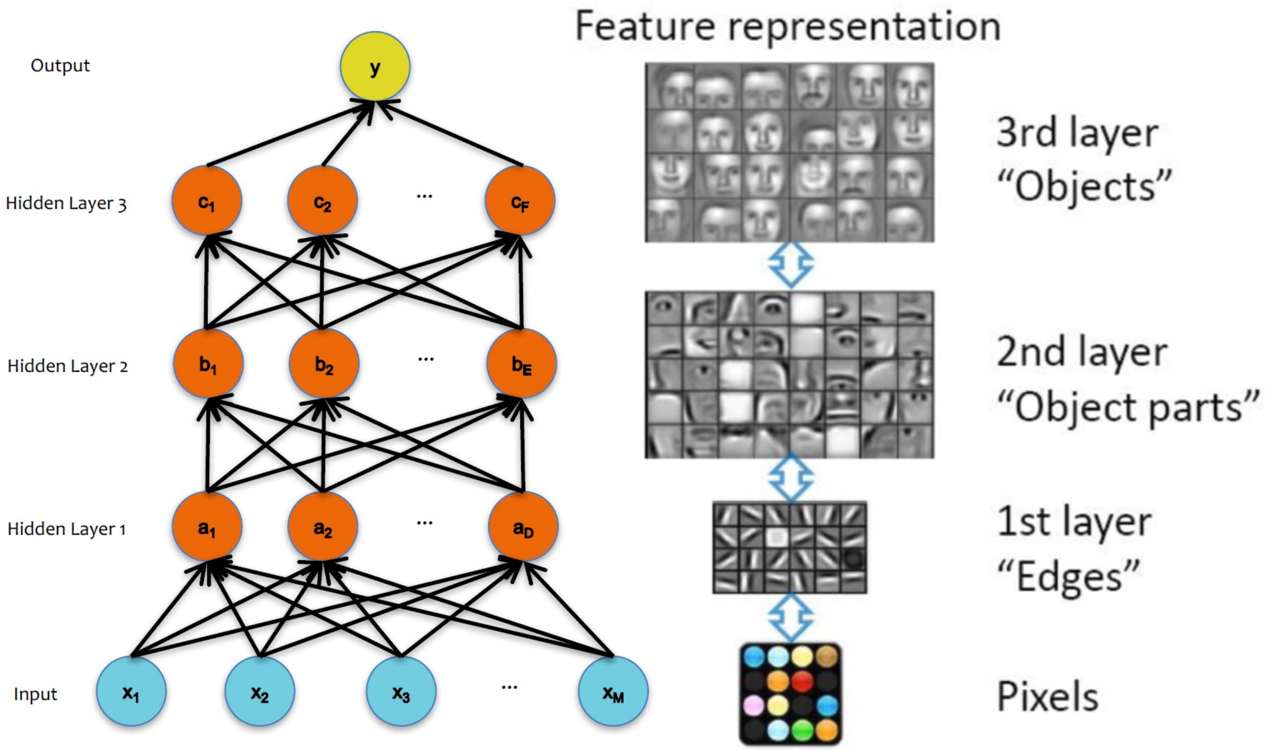
**Defintion:** Deep architectures are composed of *multiple levels* of non-linear operations, such as neural nets with many hidden layers.




$$y_i = \sigma(\sum_j W_{ij} x_j + W_{i0})$$

# Goal of Deep architectures

**Goal:** Deep learning methods aim at learning *feature hierarchies*




Example from Honglak Lee (NIPS 2010)

where features from higher levels of the hierarchy are formed by lower level features.

- Neurobiological motivation: The mammal brain is organized in a deep architecture (Serre, Kreiman, Kouh, Cadieu, Knoblich, & Poggio, 2007) (E.g. visual system has 5 to 10 levels)

# Deep Learning History

- **Inspired** by the architectural depth of the brain, researchers wanted for decades to train deep multi-layer neural networks.
- **No** very **success**ful attempts were reported before 2006 …

  Researchers reported positive experimental results with typically two or three levels (i.e. one or two hidden layers), but training deeper networks consistently yielded poorer results.

- **SVM**: Vapnik and his co-workers developed the Support Vector Machine (1993). It is a shallow architecture.
- **Digression**: In the 1990's, many researchers abandoned neural networks with multiple adaptive hidden layers because SVMs worked better, and there was no successful attempts to train deep networks.
- **GPUs + Large datasets -> Breakthrough in 2006**

# Breakthrough

**Deep Belief Networks (DBN)**

Hinton, G. E, Osindero, S., and Teh, Y. W. (2006).
A fast learning algorithm for deep belief nets.
Neural Computation, 18:1527-1554.

**Autoencoders**

Bengio, Y., Lamblin, P., Popovici, P., Larochelle, H. (2007).
Greedy Layer-Wise Training of Deep Networks,
Advances in Neural Information Processing Systems 19

**Convolutional neural networks running on GPUs** (2012)
Alex Krizhevsky, Ilya Sutskever, Geoffrey Hinton, Advances in Neural Information Processing Systems 2012

# Deep Convolutional Networks

# Convolutional Neural Networks

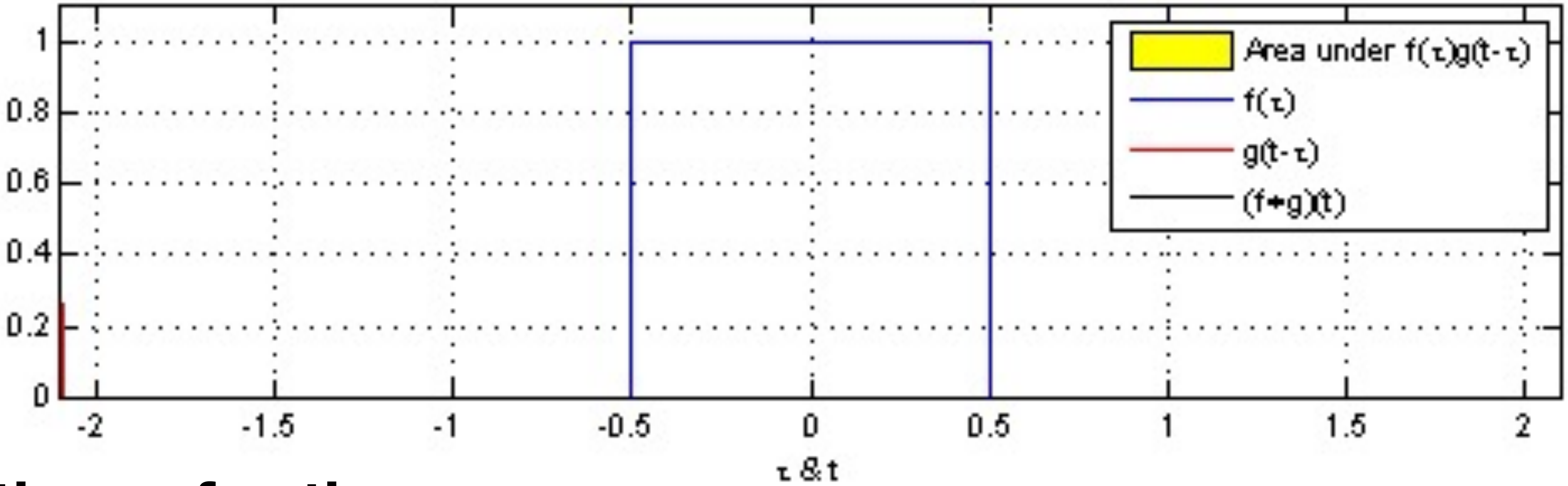
Compared to standard feedforward neural networks with similarly-sized layers,

- CNNs have much fewer connections and shared parameters
- and so they are easier to train,
- while their performance is likely to be only slightly worse, particularly for images as inputs.

**LeNet 5**

Y. LeCun, L. Bottou, Y. Bengio and P. Haffner: **Gradient-Based Learning Applied to Document Recognition**, *Proceedings of the IEEE, 86(11):2278-2324, November* ***1998***

# Convolution




**Continuous functions:**

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)\, g(t-\tau)\, d\tau - = \int_{-\infty}^{\infty} f(t-\tau)\, g(\tau)\, d\tau.$$

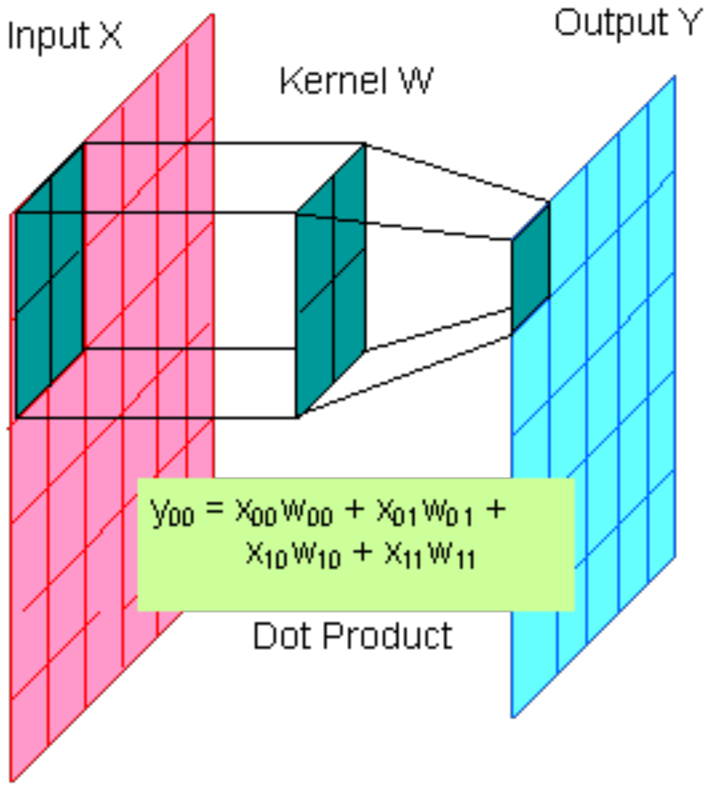
**Discrete functions:**

$$(f * g)[n] = \sum_{m=-\infty}^{\infty} f[m]\, g[n-m] = \sum_{m=-\infty}^{\infty} f[n-m]\, g[m]$$

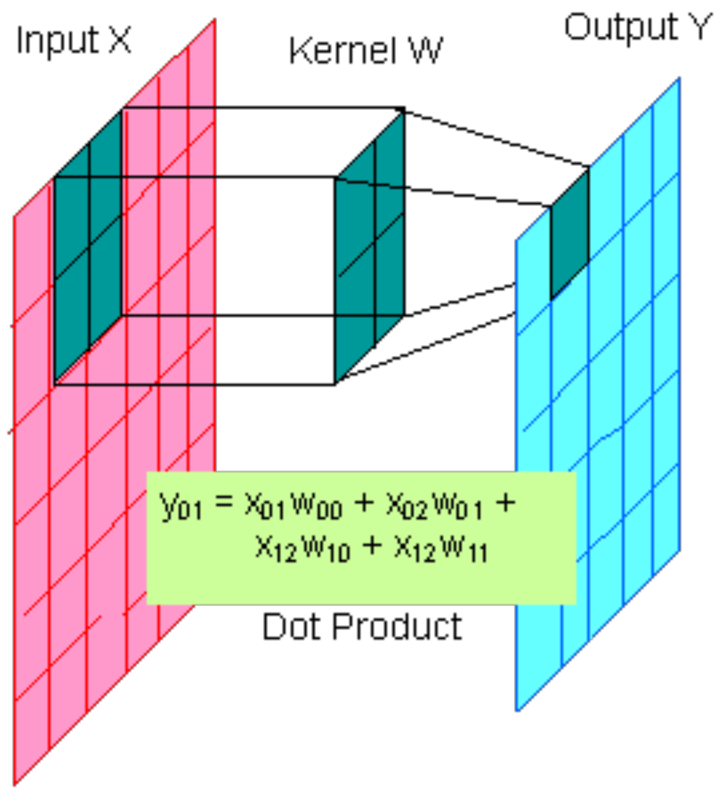
**If discrete g has support on {-M,…,M} :**

$$(f * g)[n] = \sum_{m=-M}^{M} f[n-m] g[m]$$

# 2-Dimensional Convolution



# 2-Dimensional Convolution



# 2-Dimensional Convolution

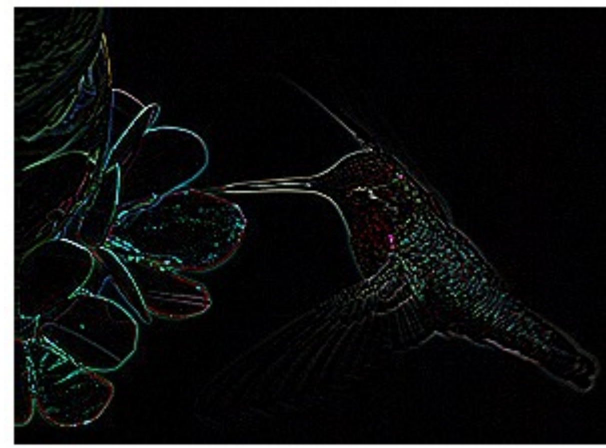
$$f[x,y] * g[x,y] = \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} f[n_1,n_2] \cdot g[x-n_1, y-n_2]$$

https://graphics.stanford.edu/courses/cs178/applets/convolution.html

Original

Filter (=kernel)

| | | | | |
|---|---|---|---|---|
| 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 0.00 | 0.00 | -2.00 | 0.00 | 0.00 |
| 0.00 | -2.00 | 8.00 | -2.00 | 0.00 |
| 0.00 | 0.00 | -2.00 | 0.00 | 0.00 |
| 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |



| | | | | |
|---|---|---|---|---|
| 0.04 | 0.04 | 0.04 | 0.04 | 0.04 |
| 0.04 | 0.04 | 0.04 | 0.04 | 0.04 |
| 0.04 | 0.04 | 0.04 | 0.04 | 0.04 |
| 0.04 | 0.04 | 0.04 | 0.04 | 0.04 |
| 0.04 | 0.04 | 0.04 | 0.04 | 0.04 |