Chapter 6

Multilayer Neural Networks

About Final Exam

Format

Open-book
开卷考试：
所携带的纸质材料仅局限于 **教科书** 与 **3页A4纸的笔记（手写、打印 均可）**，涉及其他内容的纸质材料（如作业等），均不可带入考场

Bring Calculators by yourselves
自带计算器

No electronic devices such as laptops, cell phones and pads
不可使用笔记本、手机等电子设备

Time

June 21 （Friday）Afternoon，14:00 – 16:00
6月21日（周五）下午，14:00 – 16:00

About Course Experiments

Time

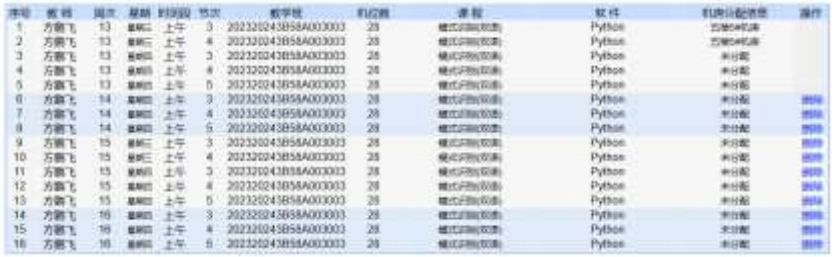
Week 13 – Week 16
第13周 至 第16周

Notes

Better to bring your own laptop
建议自带笔记本

Check the big screen in floor 3 of the Jinzhi Building before the experiment session

在每次实验课程，在金智楼三楼的大屏幕确认具体实验教室

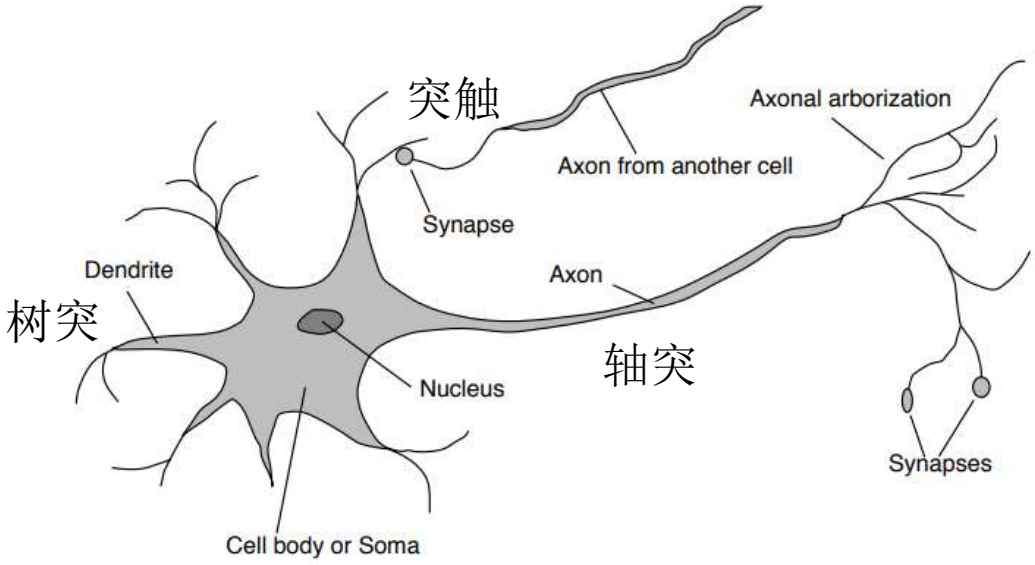


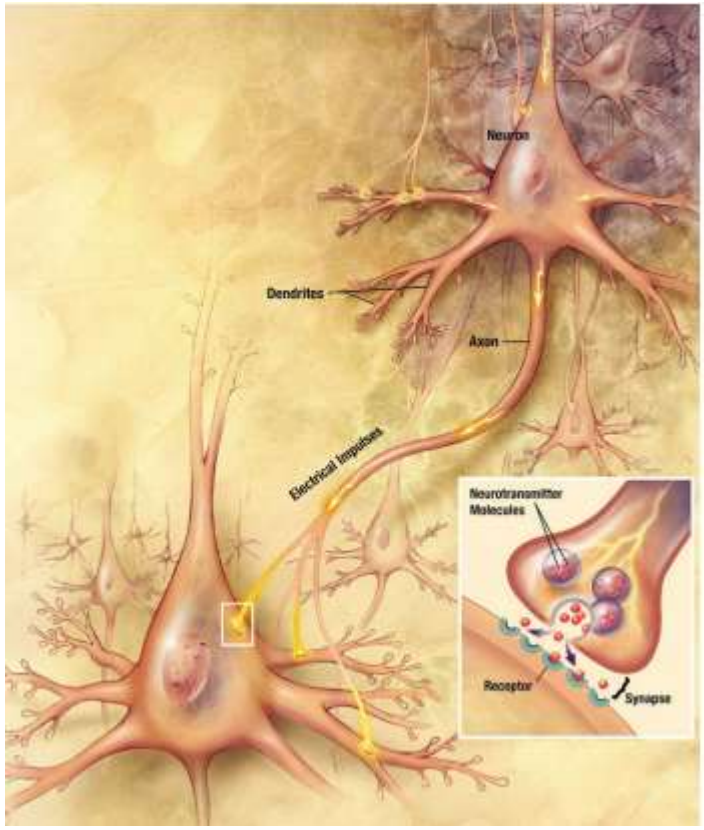
周次	星期	时间段	节次	教学班	机位数	课程	软件	机房分配信息	操作
13	星期三	上午	3	202320243B58A003003	28	模式识别(双语)	Python	五楼5#机房	
13	星期三	上午	4	202320243B58A003003	28	模式识别(双语)	Python	五楼5#机房	

Neural Function

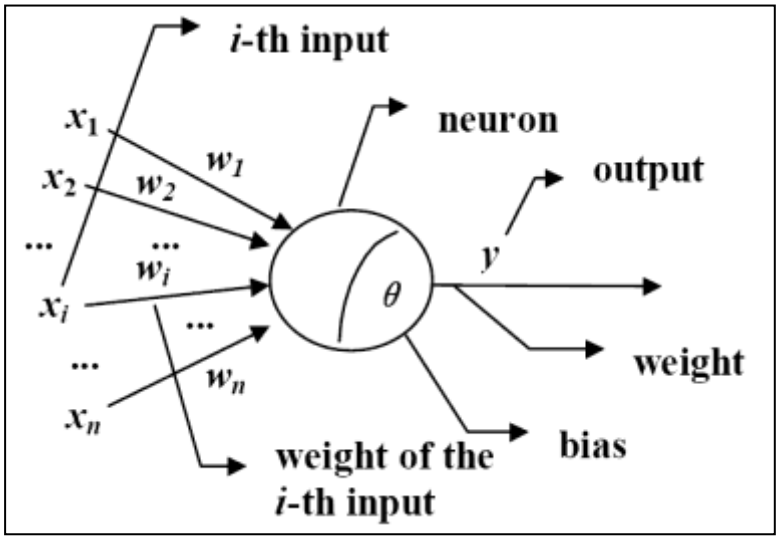
- Our brain functions (thoughts) occur as the result of firing of neurons(神经元)
- Neurons connect to each other through synapses(突触), which propagate **electrical impulses**(电脉冲) by releasing **neurotransmitters**(神经递质)
  - Synapses can be **excitatory** (potenıal-increasing，兴奋性) or **inhibitory** (potenıal-decreasing，抑制性), and have varying activation thresholds
  - Learning occurs as a result of the synapses' **plasticity** (可塑性): They exhibit long-term changes in connection strength
- There are about 10^{11} neurons and about 10^{14} synapses in the human brain!

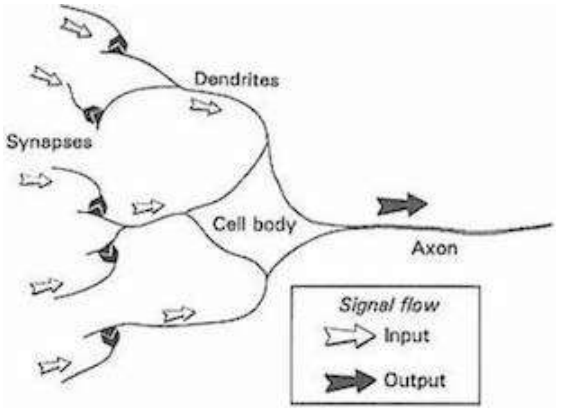
Neurons Biology

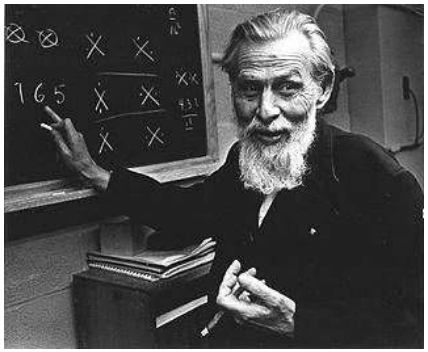




The M-P Neuron Model







Warren S. McCulloch
(1898-1969)

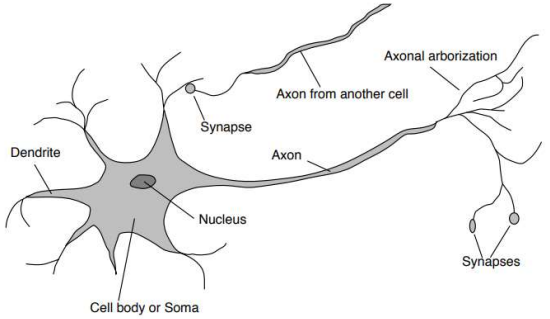
Walter Pitts
(1923-1969)

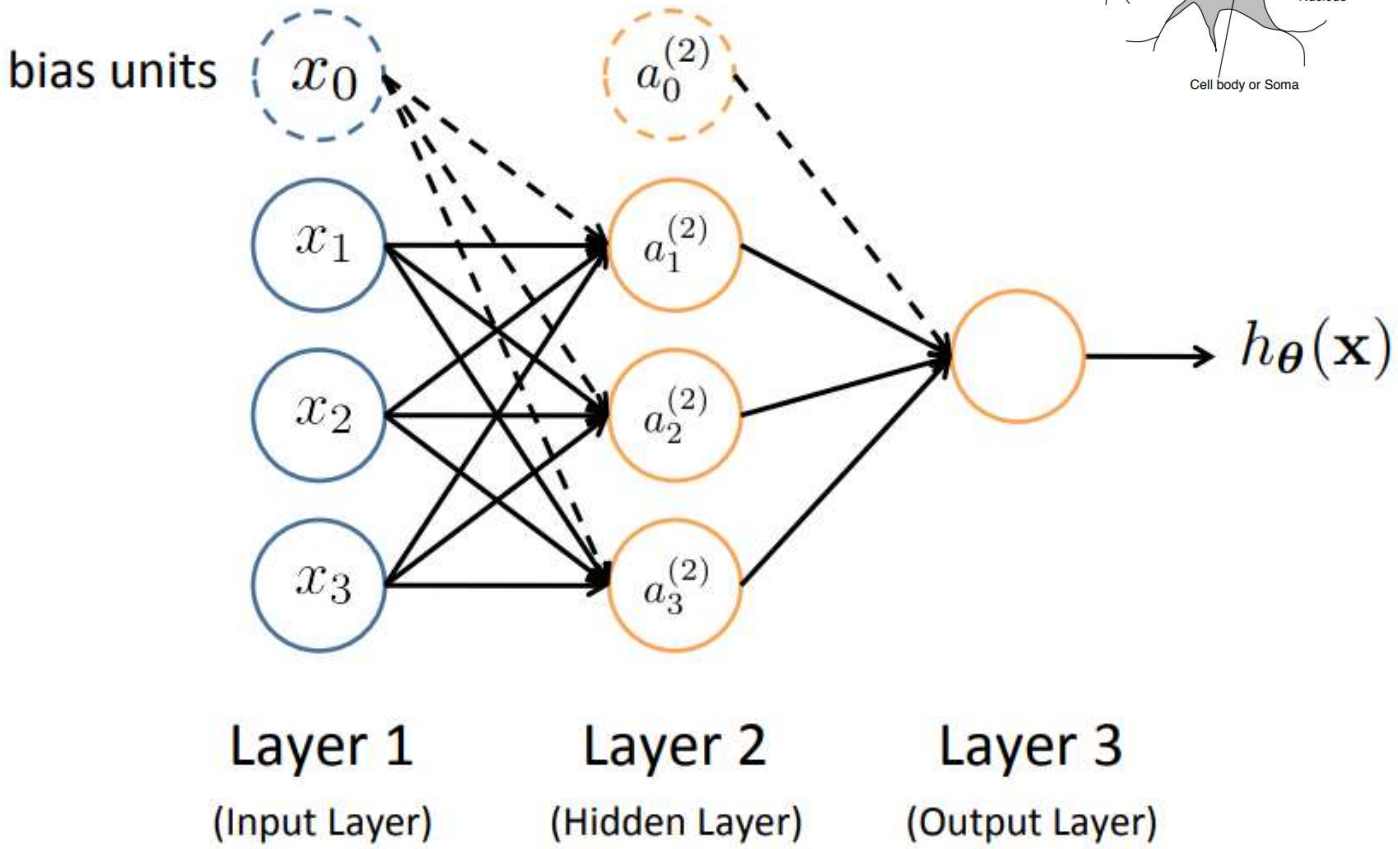
The M-P neuron model

- **Input**: $x_i\,(1 \le i \le n)$
- **Weight**: $w_i\,(1 \le i \le n)$
- **Bias**: θ
- **Activation function**: $f(\cdot)$
- **Output**: y

$$y = f\left(\sum_{i=1}^{n} w_i \cdot x_i - \theta\right)$$

Neural Network



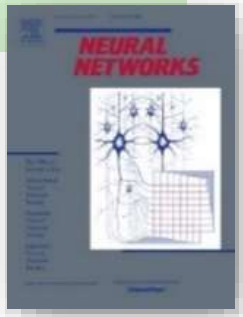


Artificial Neural Networks (ANN)

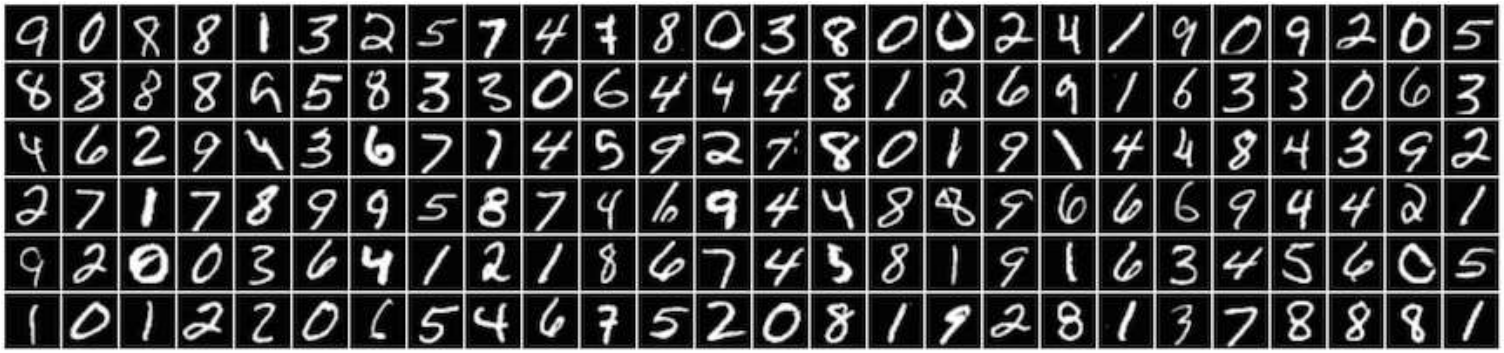
"Artificial Neural Networks (ANN) are ***massively parallel interconnected networks of simple (usually adaptive) elements and their hierarchical organizations*** *which are intended to interact with the objects of the real world in the same way as biological nervous systems do"*

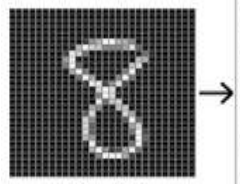
人工神经网络(ANN)是由简单（通常是自适应）元素及其分层组织组成的大规模并行互连网络，旨在以与生物神经系统相同的方式与现实世界的对象进行交互

- *T. Kohonen. An introduction to neural computing.* ***Neural Networks****, 1988, 1(1): 3-16.*

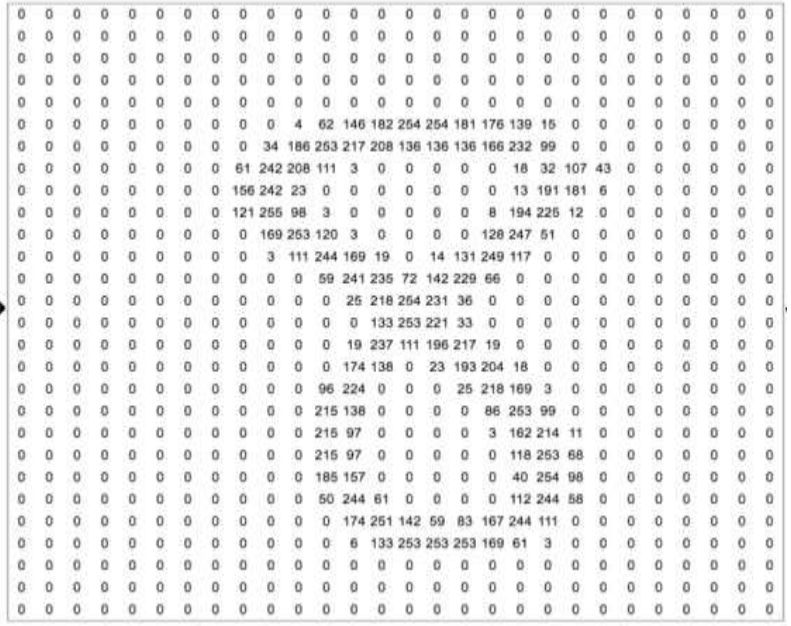


Handwritten Digit Recognition

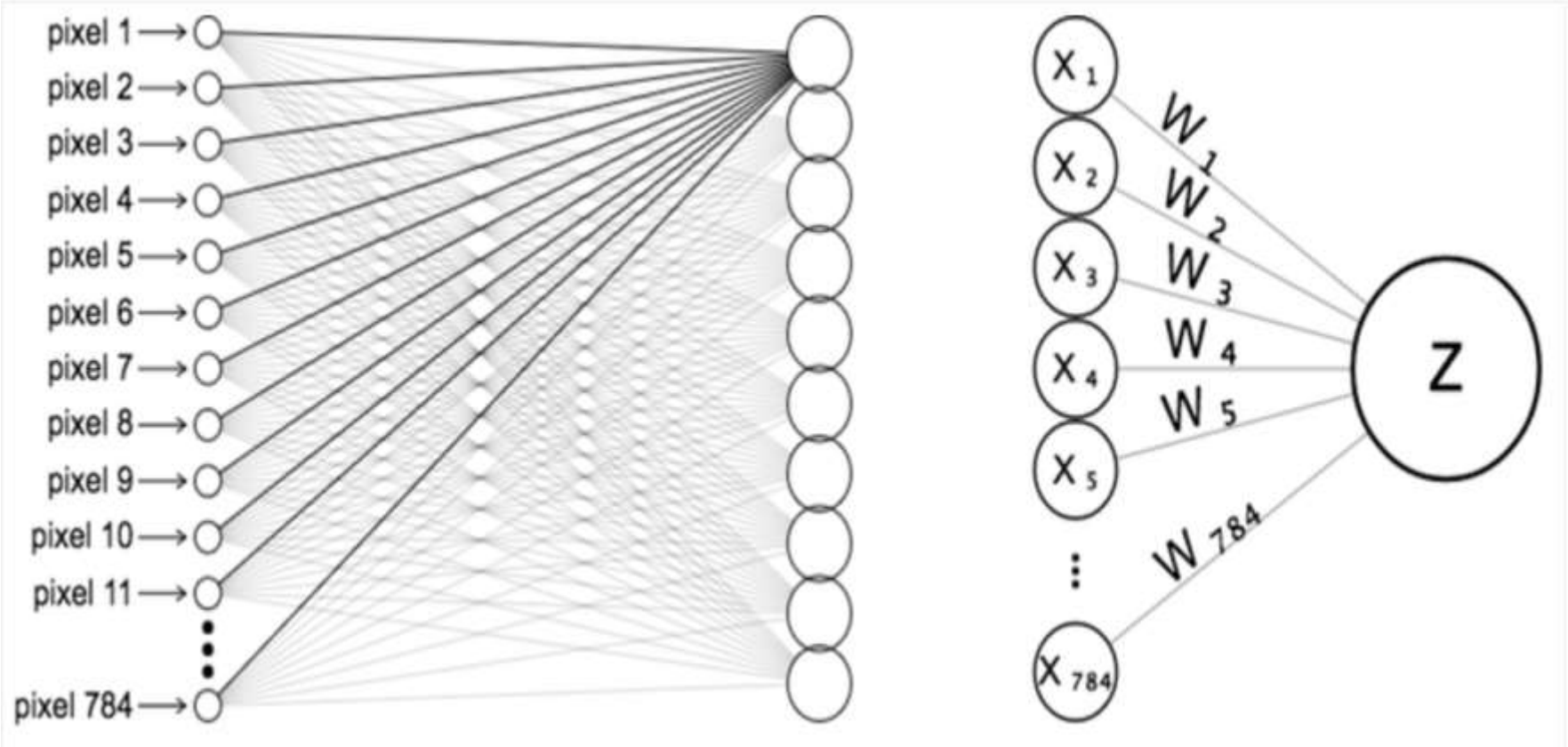




28 x 28
784 pixels

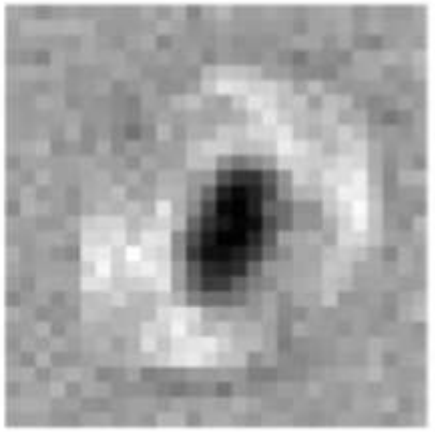


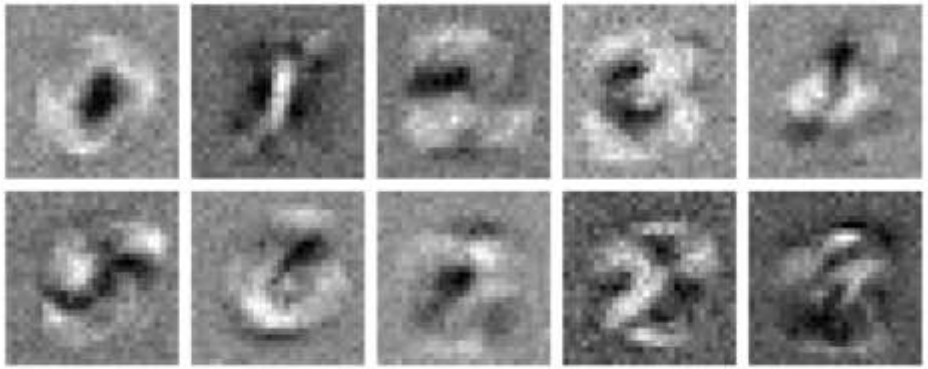
Each image is "unrolled" into a vector $\mathbf{x}$ of pixel intensities



$$\begin{bmatrix} X_1 & X_2 & X_3 & \cdots & X_{28} \\ X_{29} & X_{30} & X_{31} & & X_{56} \\ X_{57} & X_{58} & X_{59} & & X_{84} \\ \vdots & & & \ddots & \\ X_{757} & X_{758} & X_{759} & & X_{784} \end{bmatrix} \circ \begin{bmatrix} W_1 & W_2 & W_3 & \cdots & W_{28} \\ W_{29} & W_{30} & W_{31} & & W_{56} \\ W_{57} & W_{58} & W_{59} & & W_{84} \\ \vdots & & & \ddots & \\ W_{757} & W_{758} & W_{759} & & W_{784} \end{bmatrix} = Z$$

W_1	W_2	W_3	. . .	W_{28}
W_{29}	W_{30}	W_{31}		W_{56}
W_{57}	W_{58}	W_{59}		W_{84}
⋮			⋱	
W_{757}	W_{758}	W_{759}		W_{784}

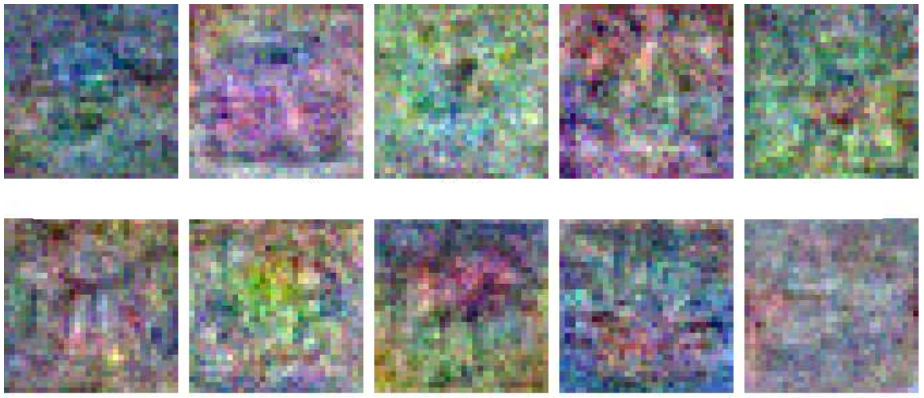




3-nearest-neighbor = 2.4% error
400–300–10 unit MLP = 1.6% error
LeNet (1998): 768–192–30–10 unit MLP = 0.9% error
SVMs: ≈ 0.6% error

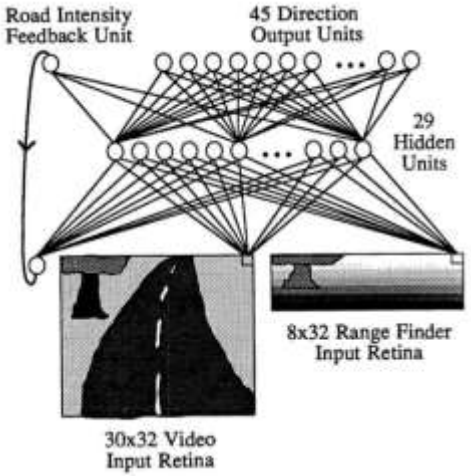
CIFAR10 Dataset Performances

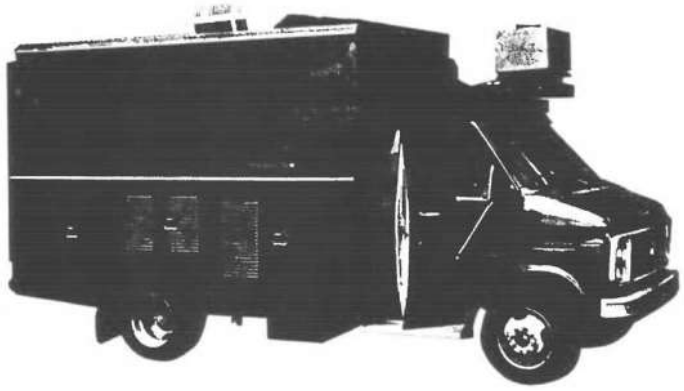
airplanes, automobiles, birds, cats, deer, dogs, frogs, horses, ships, trucks



An "Ancient" Self-Driving Car

- ALVINN
  - An Autonomous Land Vehicle in a Neural Network
  - Published in NIPS, 1988
    - Yes, NIPS is the conference where Vladimir Vapnik got 3 of his SVM paper rejected ☺



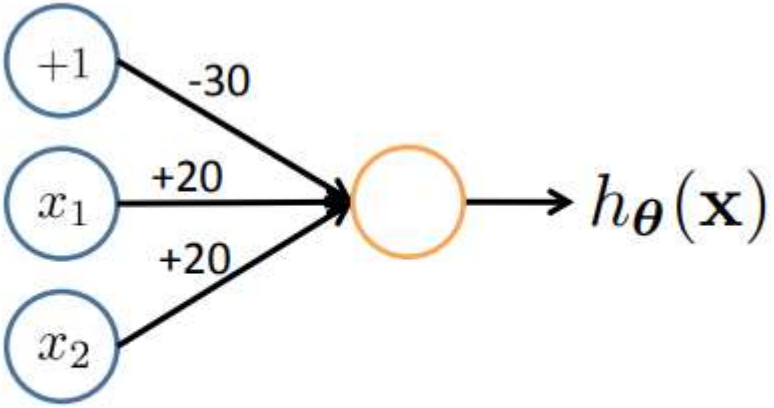


Representing Boolean Functions

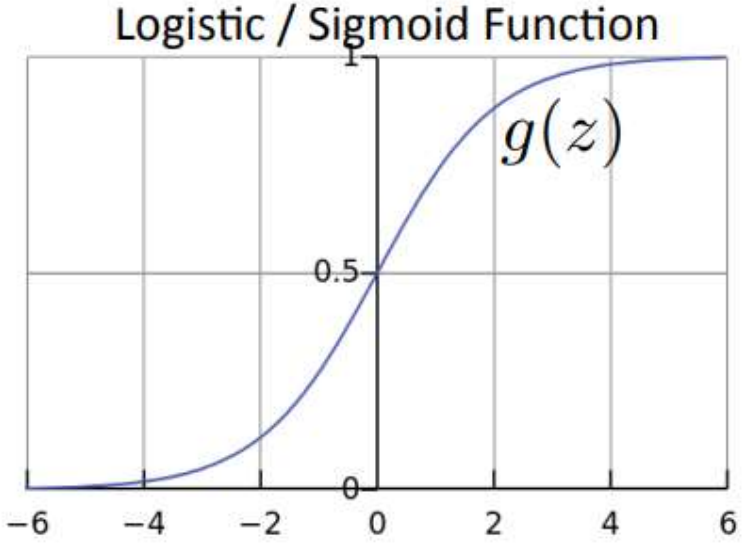
Simple example: AND

$x_1, x_2 \in \{0, 1\}$

$y = x_1 \text{ AND } x_2$

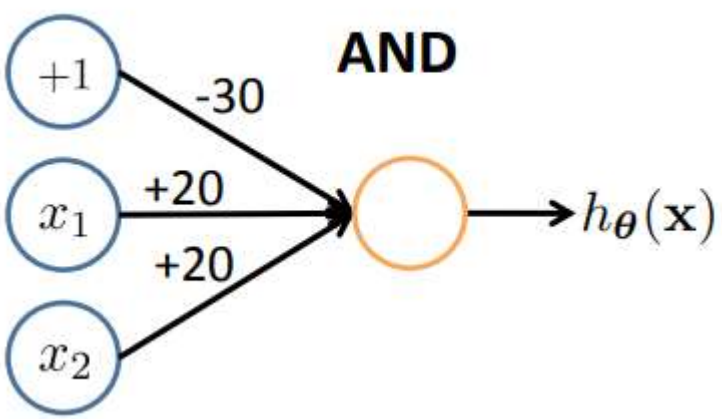


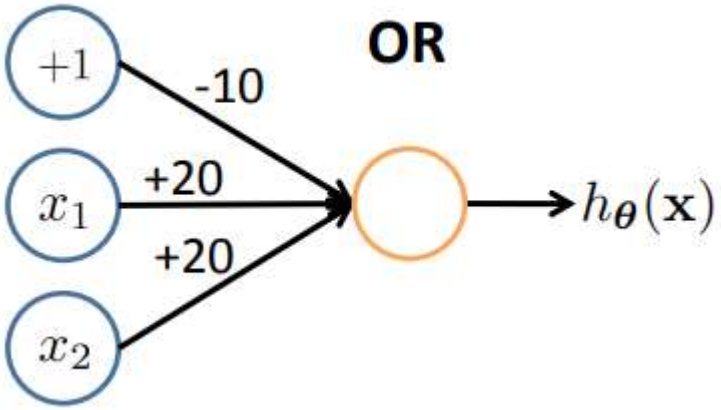
$$\mathrm{h}_\Theta(\mathbf{x}) = g(-30 + 20x_1 + 20x_2)$$

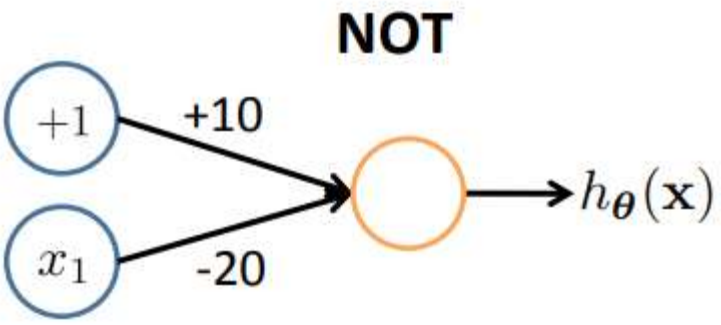


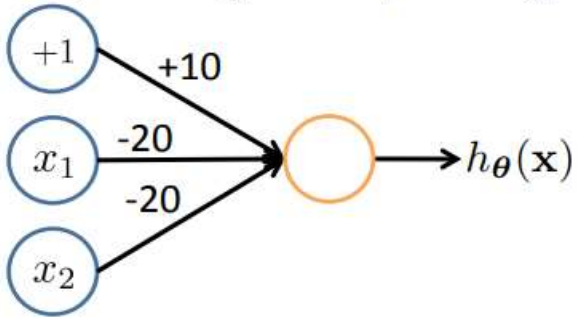
x_1	x_2	$\mathrm{h}_\Theta(\mathbf{x})$
0	0	$g(-30) \approx 0$
0	1	$g(-10) \approx 0$
1	0	$g(-10) \approx 0$
1	1	$g(10) \approx 1$

Representing Boolean Functions



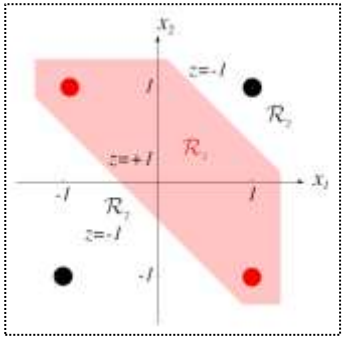






或非门

The XOR Problem

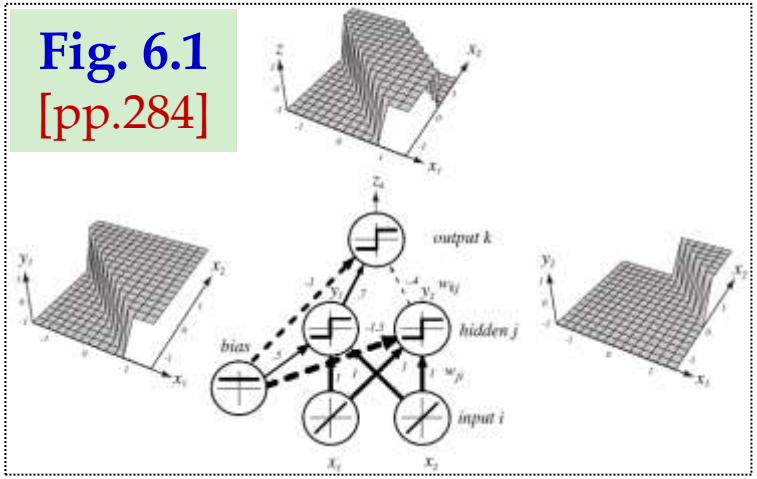


The XOR ("异或") problem

- **Decide +1 if** $x_1 \cdot x_2 = 1$
- **Decide -1 if** $x_1 \cdot x_2 = -1$

linearly inseparable

Fig. 6.1 [pp.284]



A 2-2-1 three-layer ANN

$(d = 2;\ n_H = 2)$

$$net_j = \sum_{i=1}^{d} w_{ji} x_i + w_{j0} = \mathbf{w}_j^t \mathbf{x}$$ *input to the hidden unit*

$$y_j = f(net_j)$$ *activation of the hidden unit*

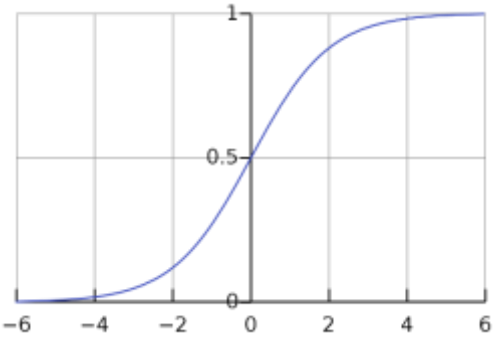
$$f(net) = \mathrm{Sgn}(net) = \begin{cases} 1 \text{ if } net \geq 0 \\ -1 \text{ if } net < 0 \end{cases}$$ *activation function*

$$net_k = \sum_{j=1}^{n_H} w_{kj} y_j + w_{k0} = \mathbf{w}_k^t \mathbf{y}$$ *input to the output unit*

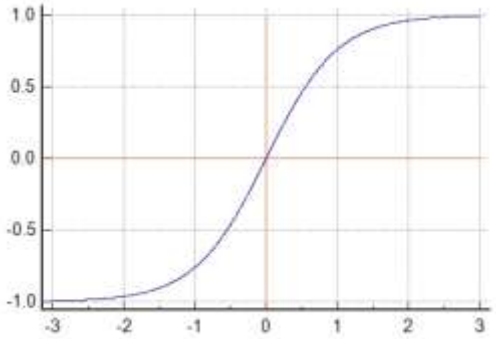
$$z_k = f(net_k)$$ *activation of the output unit*

Feedforward Neural Network (Cont.)

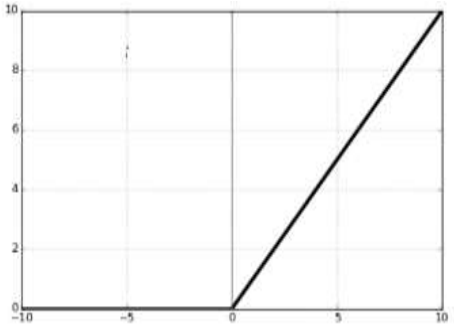
Activation function



Sigmoid $f(x) = \frac{1}{1+e^{-x}}$



tanh $f(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$



ReLU $f(x) = \max(0, x)$

Expressive power of ANN ⇒ ***theoretical ("can", not "how")***

One layer of hidden units with sigmoid activation function is sufficient for approximating any function with finitely many discontinuities to arbitrary precision

具有sigmoid 激活函数的一层隐藏单元足以将任何具有有限多个不连续点的函数逼近到任意精度

- *K. Hornik, M. Stinchcombe, H. L. White. Multilayer feedforward neural networks are universal approximators. **Neural Networks**, 1989, 2(5): 359-366.*

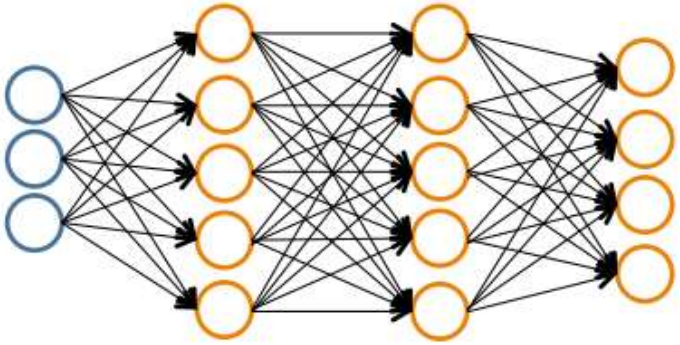
Multiple Output Units: One-vs-Rest

Pedestrian

Car

Motorcycle

Truck



$$h_\Theta(\mathbf{x}) \in \mathbb{R}^K$$

We want:

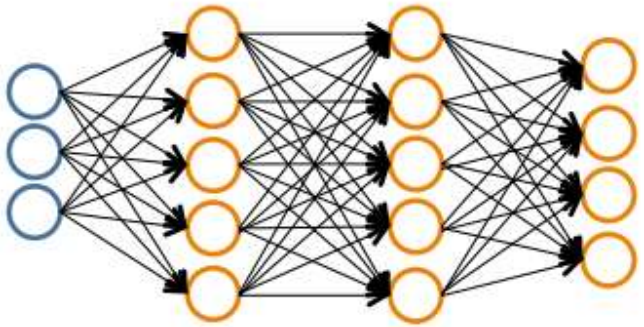
$$h_\Theta(\mathbf{x}) \approx \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$ when pedestrian

$$h_\Theta(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$ when car

$$h_\Theta(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$ when motorcycle

$$h_\Theta(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$ when truck

Multiple Output Units: One-vs-Rest



$h_\Theta(\mathbf{x}) \in \mathbb{R}^K$

We want:

$h_\Theta(\mathbf{x}) \approx \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$ when pedestrian

$h_\Theta(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$ when car

$h_\Theta(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$ when motorcycle

$h_\Theta(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$ when truck

- Given $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n)\}$

  Must convert labels to 1-of-K representation

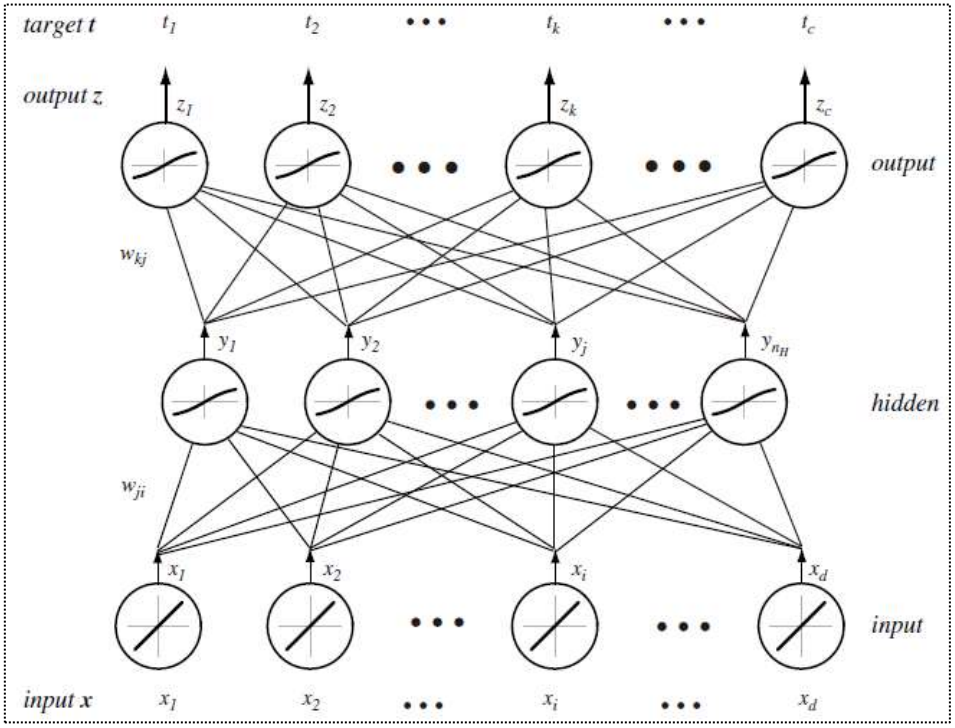
  – e.g., $y_i = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$ when motorcycle, $y_i = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$ when car, etc.

Feedforward (前馈) Neural Network

Settings A d-n_H-c fully connected three-layer network

d: # features n_H: # hidden neurons c: # output neurons

$\mathbf{x} = (x_1, x_2, \ldots, x_d)^t$: training pattern $\mathbf{t} = (t_1, t_2, \ldots, t_c)^t$: desired output



Parameters to be learned

w_{ji}: **input-to-hidden** layer weight

(i-th feature to j-th hidden unit)

w_{kj}: **hidden-to-output** layer weight

(j-th hidden to k-th output unit)

$$(1 \le i \le d;\ 1 \le j \le n_H;\ 1 \le k \le c)$$

$$\mathbf{w} = (w_{11}, \ldots, w_{n_H d}, \ldots, w_{c n_H})^t$$

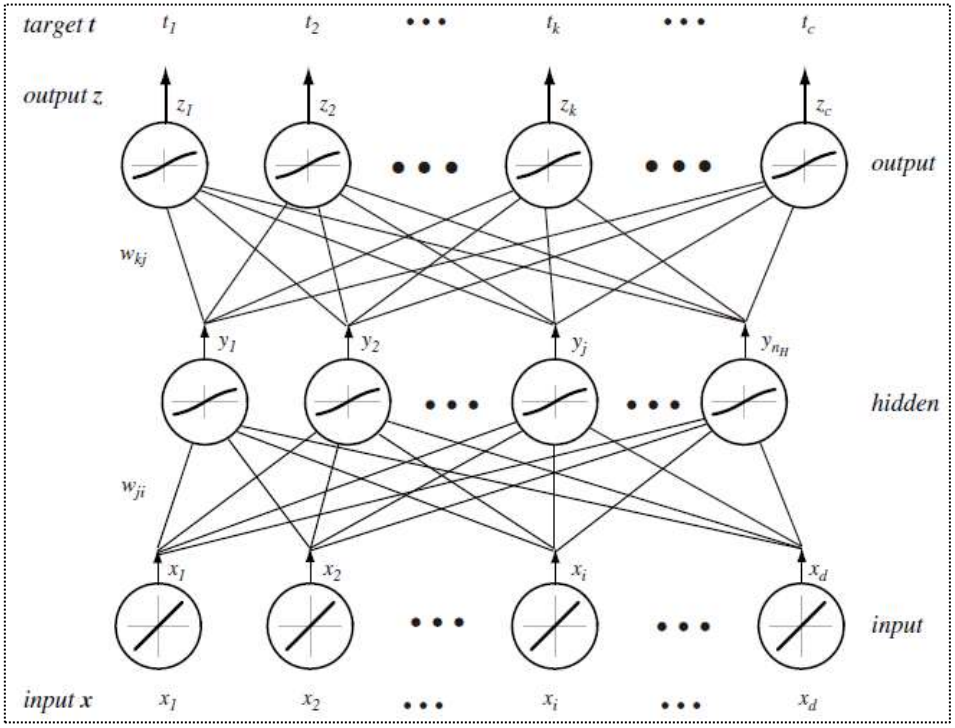
$$\#\ parameters\ in\ \mathbf{w} : n_H(d+c)$$

Feedforward Neural Network (Cont.)

Settings A d-n_H-c fully connected three-layer network

d: # features n_H: # hidden neurons c: # output neurons

$\mathbf{x} = (x_1, x_2, \ldots, x_d)^t$: training pattern $\mathbf{t} = (t_1, t_2, \ldots, t_c)^t$: desired output



Feedforward procedure

$$net_j = \sum_{i=1}^{d} w_{ji} x_i \quad (1 \le j \le n_H)$$

$$y_j = f(net_j) \quad (1 \le j \le n_H)$$

$$net_k = \sum_{j=1}^{n_H} w_{kj} y_j \quad (1 \le k \le c)$$

$$z_k = f(net_k) \quad (1 \le k \le c)$$

$g_k(\mathbf{x}) = z_k$ *(discriminant function)*

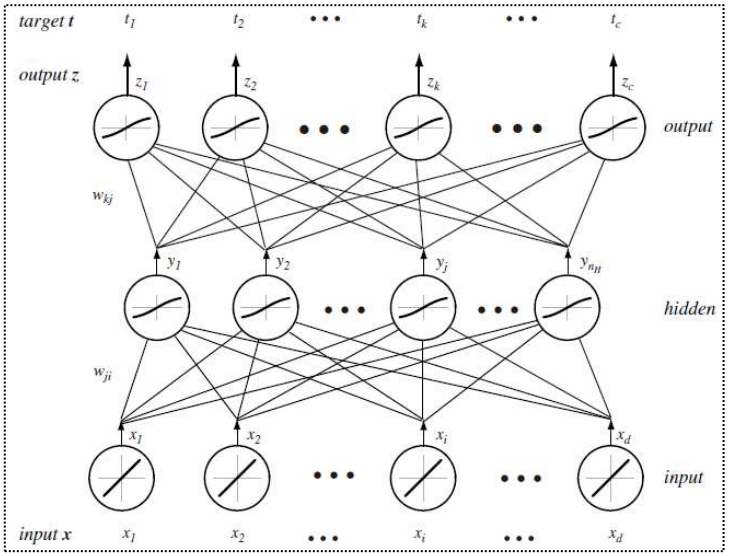
$$= f\left(\sum_{j=1}^{n_H} w_{kj} f\left(\sum_{i=1}^{d} w_{ji} x_i\right)\right)$$

Backpropagation (反向传播) Algorithm

Settings A d-n_H-c fully connected three-layer network

d: # features n_H: # hidden neurons c: # output neurons **w: weights**

$\mathbf{x} = (x_1, x_2, \ldots, x_d)^t$: training pattern $\mathbf{t} = (t_1, t_2, \ldots, t_c)^t$: desired output



Criterion function

$$J(\mathbf{w}) = \frac{1}{2}\sum_{k=1}^{c}(t_k - z_k)^2 = \frac{1}{2}\|\mathbf{t} - \mathbf{z}\|^2$$

Gradient descent

$$\mathbf{w} \leftarrow \mathbf{w} + \Delta\mathbf{w}$$

$$\Delta\mathbf{w} = -\eta\frac{\partial J}{\partial \mathbf{w}} \quad \left(\Delta w_{pq} = -\eta\frac{\partial J}{\partial w_{pq}}\right)$$

P. J. Werbos. Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences. PhD Thesis, Harvard University, 1974.

Backpropagation Algorithm (Cont.)

Settings A d-n_H-c fully connected three-layer network

d: # features n_H: # hidden neurons c: # output neurons **w: weights**

$\mathbf{x} = (x_1, x_2, \ldots, x_d)^t$: training pattern $\mathbf{t} = (t_1, t_2, \ldots, t_c)^t$: desired output

w_{kj}: **hidden-to-output** layer weight *(k-th output unit back to j-th hidden unit)*

$$\frac{\partial J}{\partial w_{kj}} = \frac{\partial J}{\partial net_k}\frac{\partial net_k}{\partial w_{kj}} = \frac{\partial J}{\partial net_k}\frac{\sum_{j=1}^{n_H} w_{kj} y_j}{\partial w_{kj}} = -\delta_k y_j$$

$$\delta_k = -\frac{\partial J}{\partial net_k} = -\frac{\partial J}{\partial z_k}\frac{\partial z_k}{\partial net_k}$$

$$= -\frac{\partial\left(\frac{1}{2}\sum_{k=1}^{c}(t_k - z_k)^2\right)}{\partial z_k}\frac{\partial f(net_k)}{\partial net_k} = (t_k - z_k) f'(net_k)$$

$$\Delta w_{kj} = -\eta\frac{\partial J}{\partial w_{kj}} = \eta\delta_k y_j = \eta(t_k - z_k) f'(net_k) y_j$$

Sigmoid $f' = f(1 - f)$

tanh $f' = 1 - f^2$

Backpropagation Algorithm (Cont.)

Settings A d-n_H-c fully connected three-layer network

d: # features n_H: # hidden neurons c: # output neurons **w: weights**

$\mathbf{x} = (x_1, x_2, \ldots, x_d)^t$: training pattern $\mathbf{t} = (t_1, t_2, \ldots, t_c)^t$: desired output

w_{ji}: **input-to-hidden** layer weight *(j-th hidden unit back to i-th feature)*

$$\frac{\partial J}{\partial w_{ji}} = \frac{\partial J}{\partial y_j}\frac{\partial y_j}{\partial net_j}\frac{\partial net_j}{\partial w_{ji}} = \frac{\partial J}{\partial y_j}\frac{\partial y_j}{\partial net_j}\frac{\partial\left(\sum_{i=1}^{d} w_{ji}x_i\right)}{\partial w_{ji}} = \frac{\partial J}{\partial y_j} f'(net_j)x_i = -\delta_j x_i$$

$$\frac{\partial J}{\partial y_j} = \frac{\partial}{\partial y_j}\left[\frac{1}{2}\sum_{k=1}^{c}(t_k - z_k)^2\right] = -\sum_{k=1}^{c}(t_k - z_k)\frac{\partial z_k}{\partial y_j} = -\sum_{k=1}^{c}(t_k - z_k)\frac{\partial z_k}{\partial net_k}\frac{\partial net_k}{\partial y_j}$$

$$= -\sum_{k=1}^{c}(t_k - z_k) f'(net_k) w_{kj} = -\sum_{k=1}^{c} w_{kj}\delta_k$$

$$\Delta w_{ji} = -\eta\frac{\partial J}{\partial w_{ji}} = \eta\delta_j x_i = -\eta\frac{\partial J}{\partial y_j} f'(net_j)x_i = \eta\left[\sum_{k=1}^{c} w_{kj}\delta_k\right] f'(net_j)x_i$$

Backpropagation Algorithm (Cont.)

Settings A d-n_H-c fully connected three-layer network

d: # features n_H: # hidden neurons c: # output neurons **w: weights**

$\mathbf{x} = (x_1, x_2, \ldots, x_d)^t$: training pattern $\mathbf{t} = (t_1, t_2, \ldots, t_c)^t$: desired output

Forward procedure

$$net_j = \sum_{i=1}^{d} w_{ji} x_i \quad (1 \le j \le n_H)$$

$$y_j = f(net_j) \quad (1 \le j \le n_H)$$

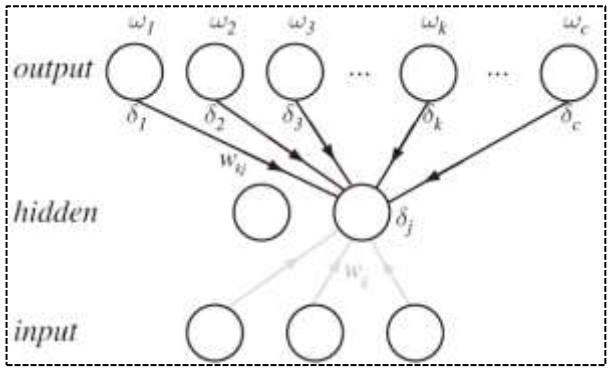
$$net_k = \sum_{j=1}^{n_H} w_{kj} y_j \quad (1 \le k \le c)$$

$$z_k = f(net_k) \quad (1 \le k \le c)$$

Backpropagation procedure

$$\delta_k = (t_k - z_k) f'(net_k) \quad (1 \le k \le c)$$

$$\delta_j = f'(net_j) \left[\sum_{k=1}^{c} w_{kj} \delta_k\right] \quad (1 \le j \le n_H)$$



δ_k, δ_j: *neuron unit's* ***sensitivity***

Backpropagation Algorithm (Cont.)

Stochastic training

One sample is randomly selected from the training set, and the weights are updated by presenting the chosen pattern to the network

1. **begin initialize** $n_H,\ \mathbf{w},\ \text{criterion}\ \theta,\ \eta,\ m \leftarrow 0$
2. **do** $m \leftarrow m + 1$
3. $\mathbf{x}^m \leftarrow$ randomly chosen training sample
4. Invoke the forward and backpropagation procedures on $\mathbf{x}^m$ to obtain $\delta_k (1 \le k \le c)$, y_j and $\delta_j\ (1 \le j \le n_H)$
5. $w_{ji} \leftarrow w_{ji} + \eta \delta_j x_i; \quad w_{kj} \leftarrow w_{kj} + \eta \delta_k y_j$
6. **until** $\|\nabla J(\mathbf{w})\| \le \theta$
7. **return** w
8. **end**

Stochastic backpropagation

Backpropagation Algorithm (Cont.)

Batch training

All samples in the training set are presented to the network at once, and the weights are updated in ***one epoch***

Settings A d-n_H-c fully connected three-layer network

d: # features n_H: # hidden neurons c: # output neurons **w: weights**

$\mathcal{D} = \{(\mathbf{x}^m, \mathbf{t}^m) \mid 1 \le m \le n\}$: training set consisting of n samples

$\mathbf{x}^m = (x_1, x_2, \ldots, x_d)^t$: training pattern $\mathbf{t}^m = (t_1, t_2, \ldots, t_c)^t$: desired output

(WLOG, the superscript m is ignored for elements of $\mathbf{x}^m$ *and* $\mathbf{t}^m$*)*

$$J(\mathbf{w}) = \frac{1}{2}\|\mathbf{t} - \mathbf{z}\|^2 \Rightarrow J(\mathbf{w}) = \frac{1}{2}\sum_{m=1}^{n}\|\mathbf{t}^m - \mathbf{z}^m\|^2$$

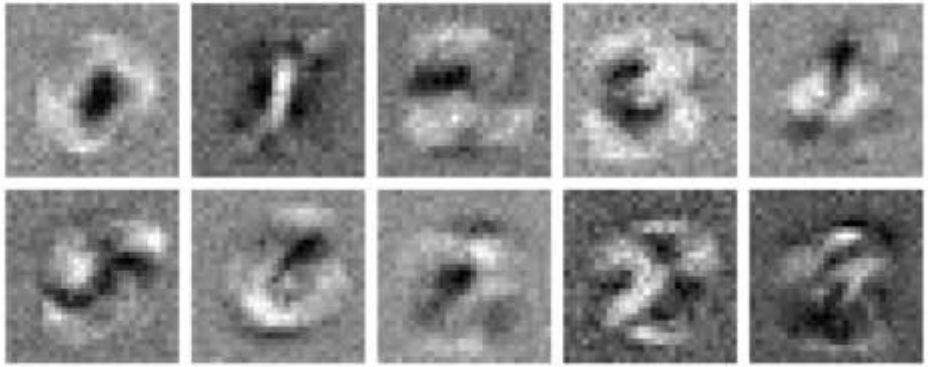
Backpropagation Algorithm (Cont.)

1. **begin initialize** $n_H,\ \mathbf{w},\ \text{criterion}\ \theta,\ \eta,\ r \leftarrow 0$
2. **do** $r \leftarrow r + 1$ *(increment epoch)*
3. $m \leftarrow 0;\ \Delta w_{ji} \leftarrow 0;\ \Delta w_{kj} \leftarrow 0$
4. **do** $m \leftarrow m + 1$
5. $\mathbf{x}^m \leftarrow$ the *m*-th sample in the training set
6. Invoke the forward and backpropagation procedures on $\mathbf{x}^m$ to obtain $\delta_k\ (1 \le k \le c)$, y_j and $\delta_j\ (1 \le j \le n_H)$
7. $\Delta w_{ji} \leftarrow \Delta w_{ji} + \eta \delta_j x_i;\quad \Delta w_{kj} \leftarrow \Delta w_{kj} + \eta \delta_k y_j$
8. **until** $m = n$
9. $w_{ji} \leftarrow w_{ji} + \Delta w_{ji};\ w_{kj} \leftarrow w_{kj} + \Delta w_{kj}$
10. **until** $\|\nabla J(\mathbf{w})\| \le \theta$
11. **return** **w**
12. **end**

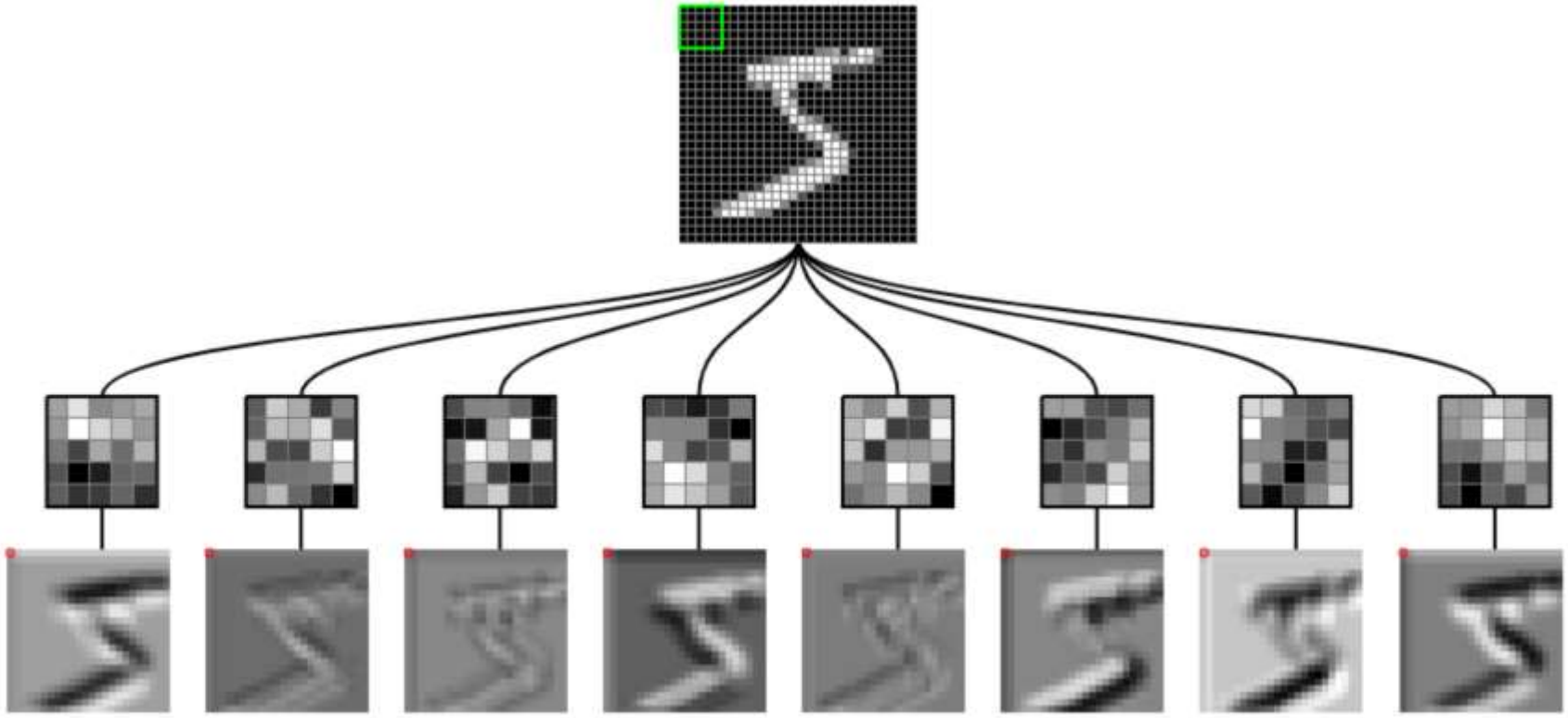
Batch backpropagation

Convolution Neural Network

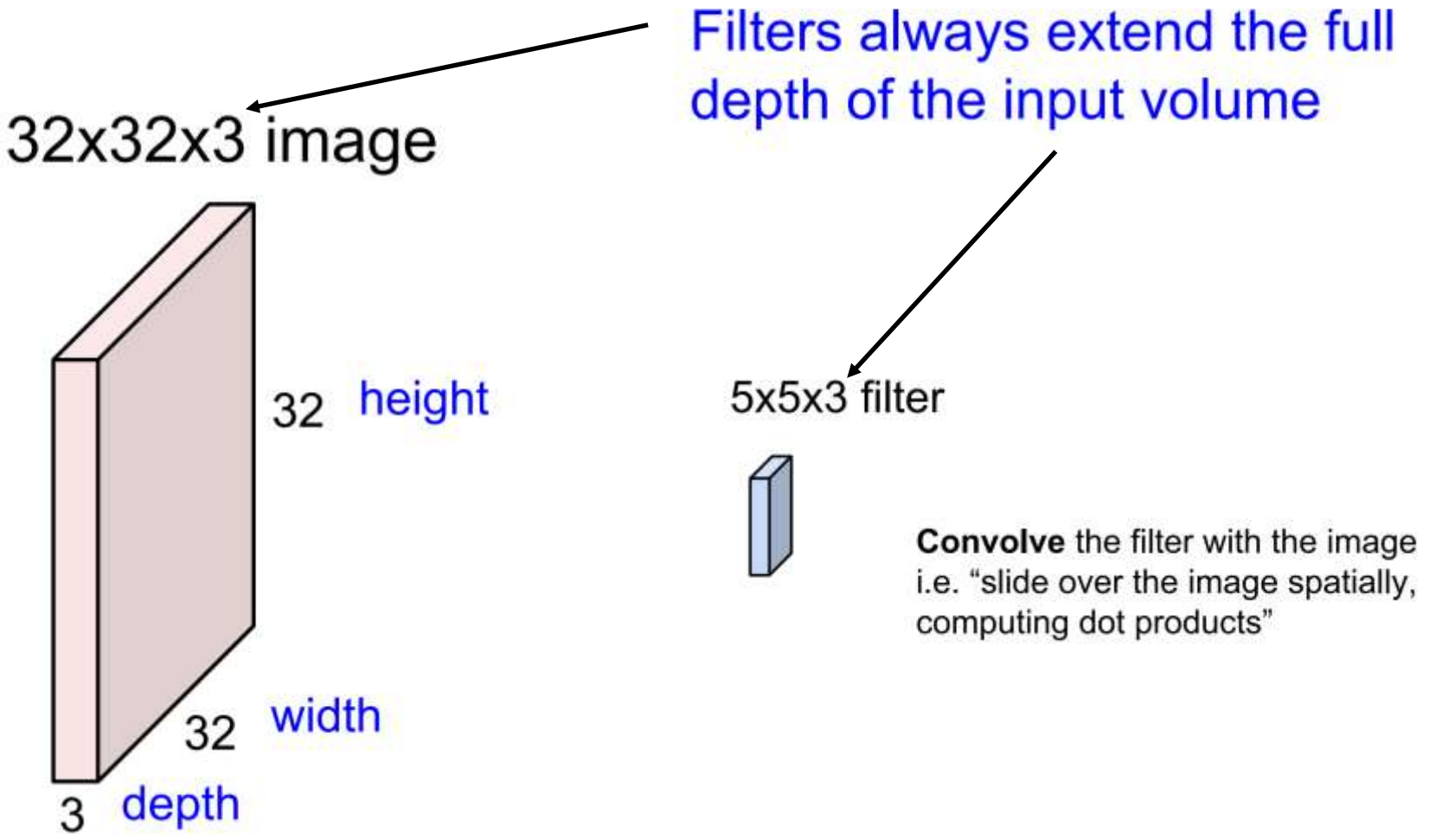
Convolution?



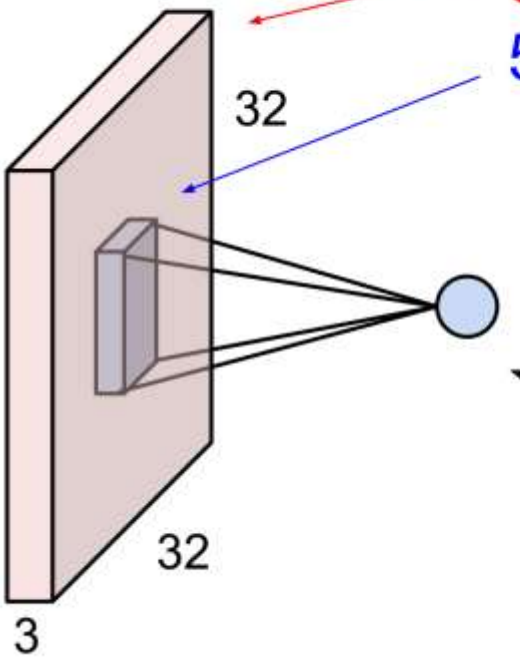
- Slides adopted from Fei-Fei
  Johnson

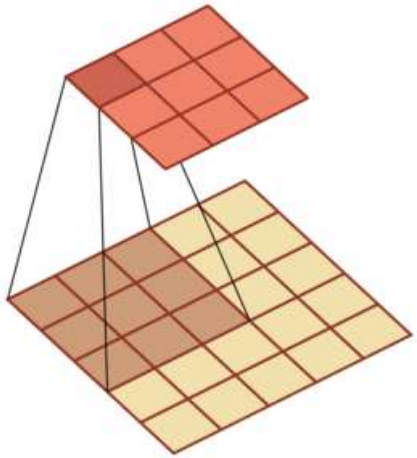


Convolution Layer



Convolution Filter

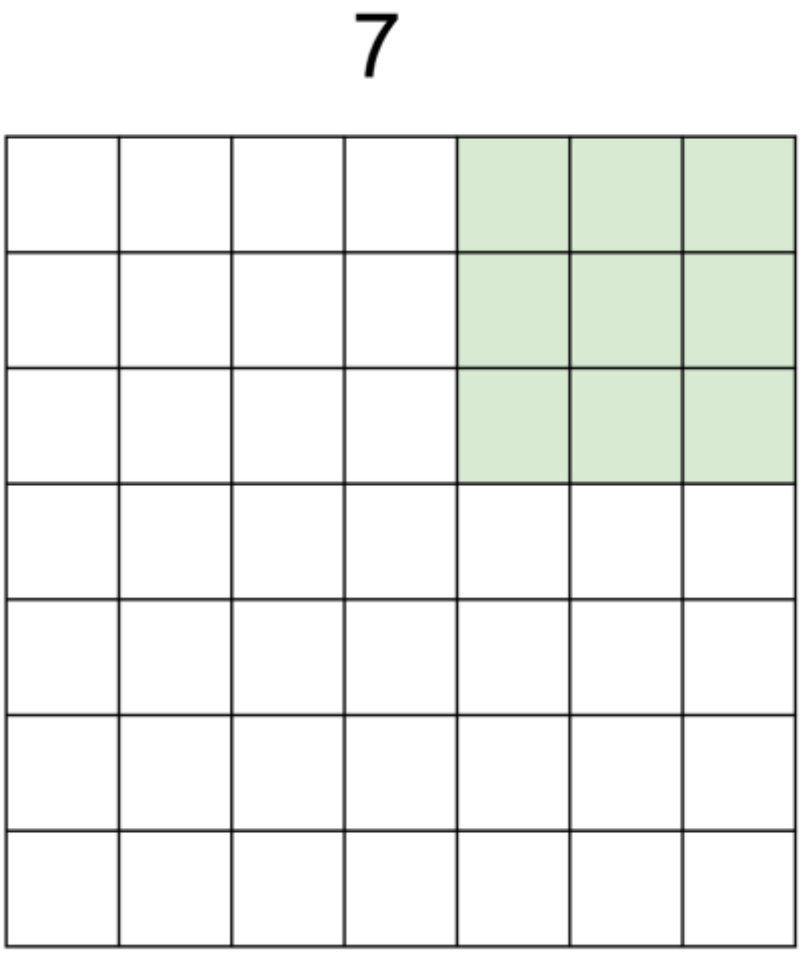




1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)

$$w^T x + b$$

Inspecting Convolution Filter

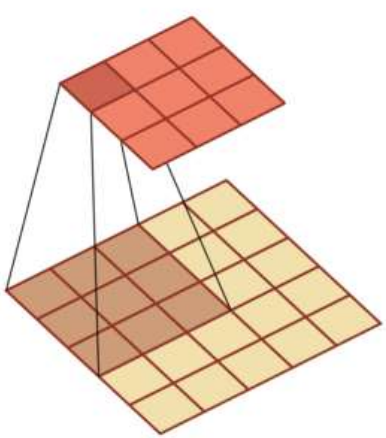


7x7 input (spatially)
assume 3x3 filter

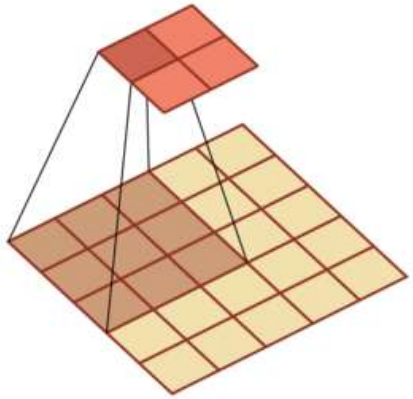
=> 5x5 output

Inspecting Convolution Filter

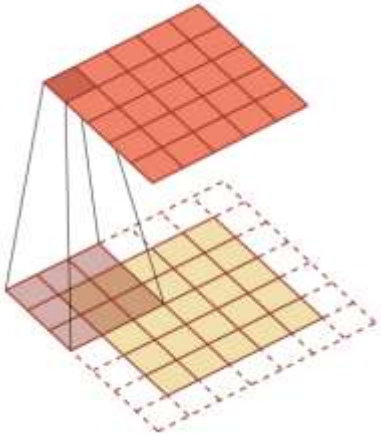
- More examples:



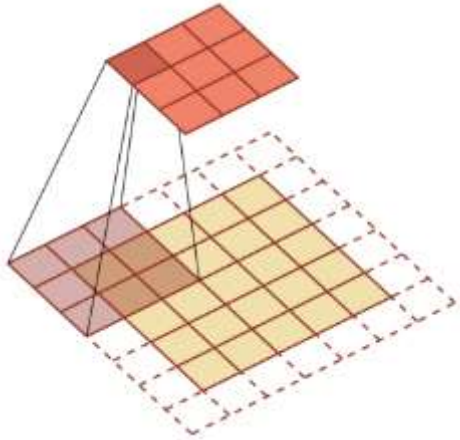
Stride 1,
zero padding 0



Stride 2,
zero padding 0

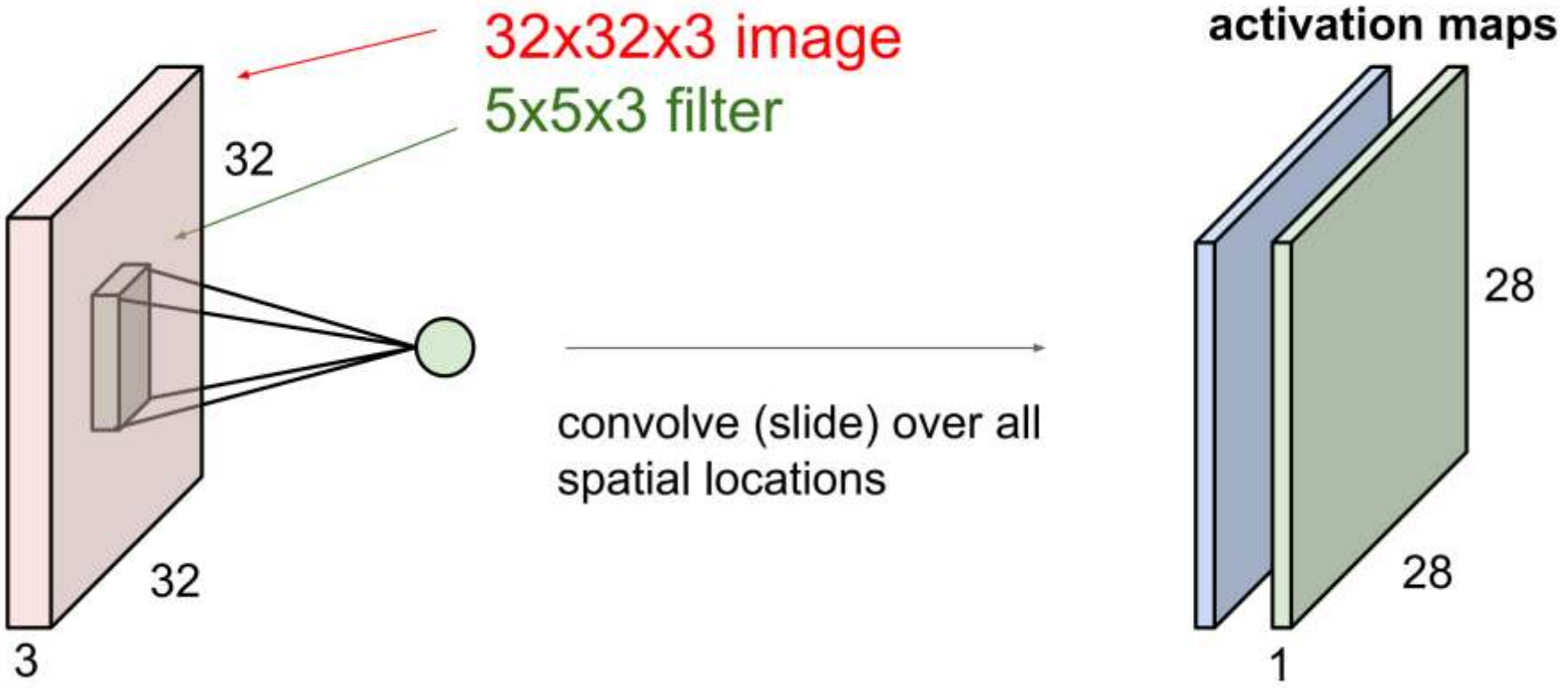


Stride 1,
zero padding 1



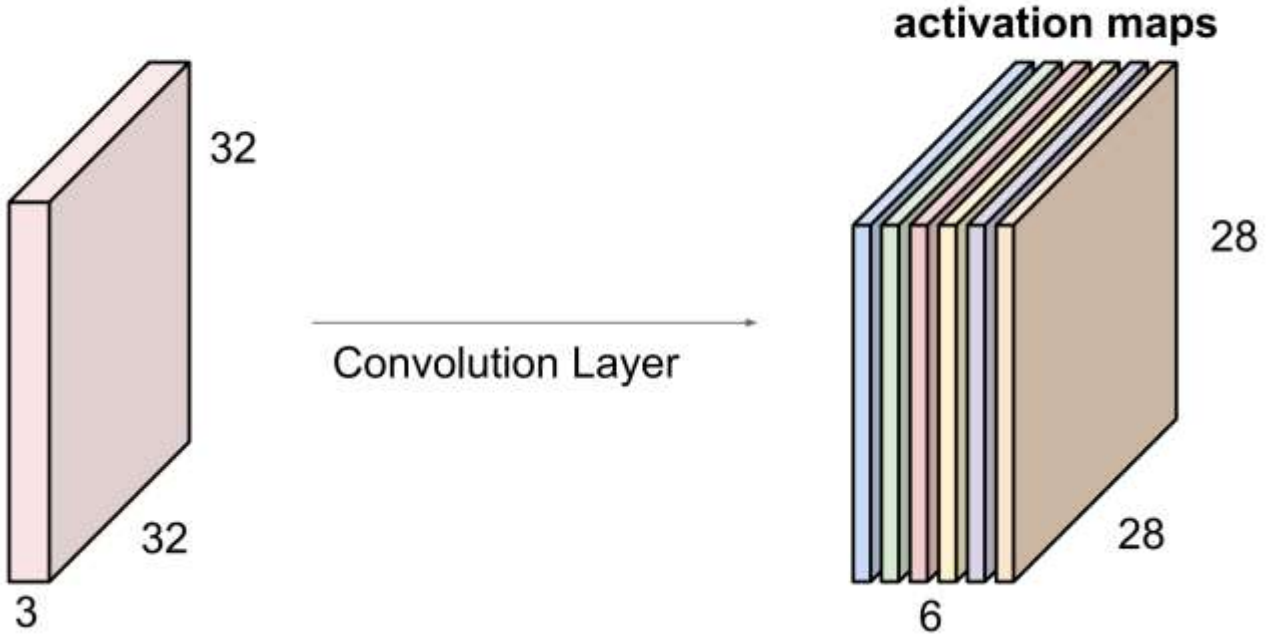
Stride 2,
zero padding 1

More Convolution Filters



Stacking Convolution Filters

For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:

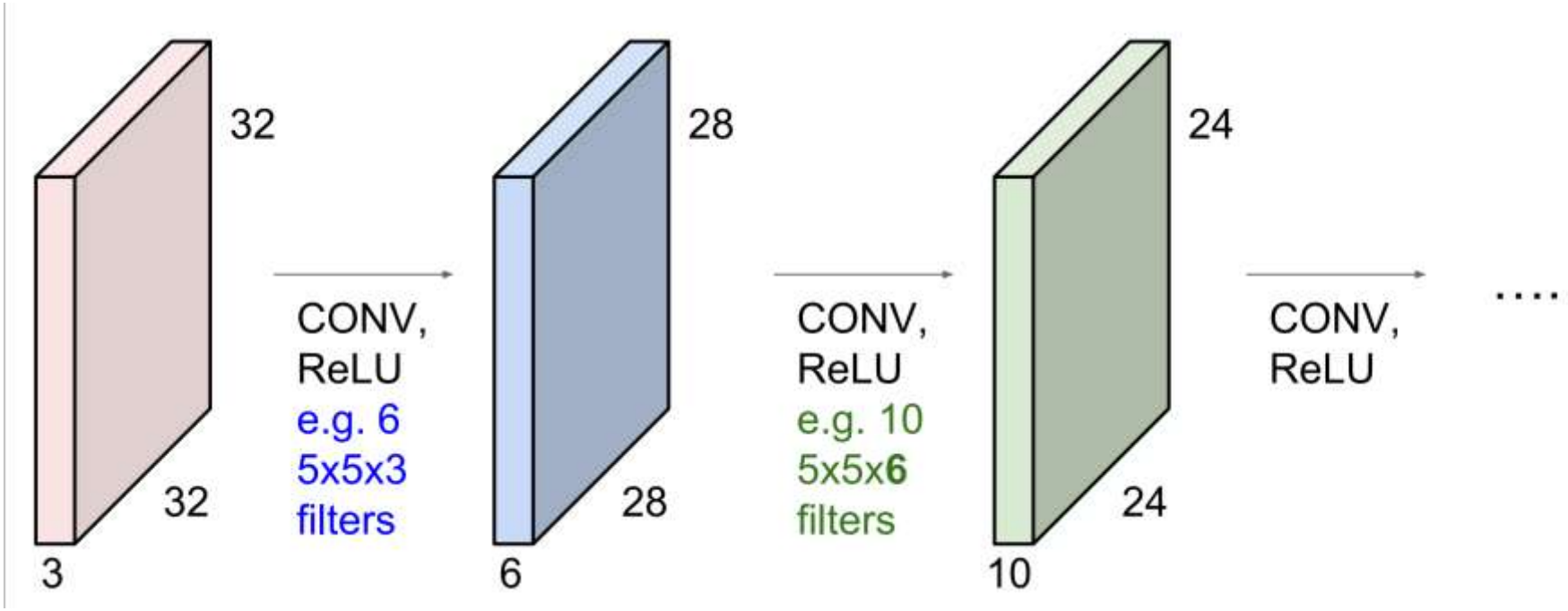


We stack these up to get a "new image" of size 28x28x6!

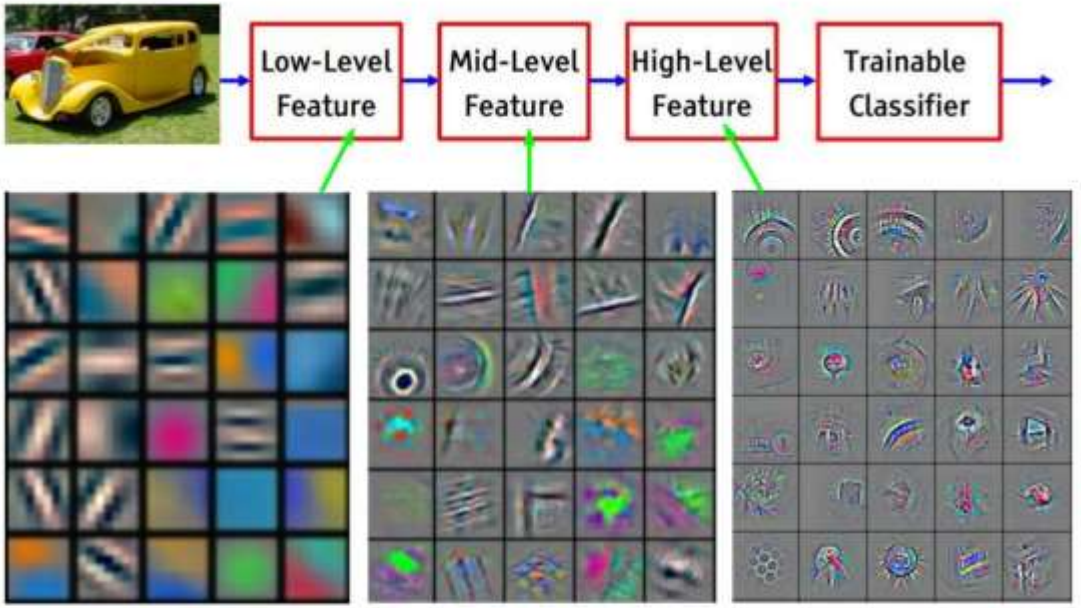
Demo of Convolution Filters

- https://ml4a.github.io/demos/convolution_all/

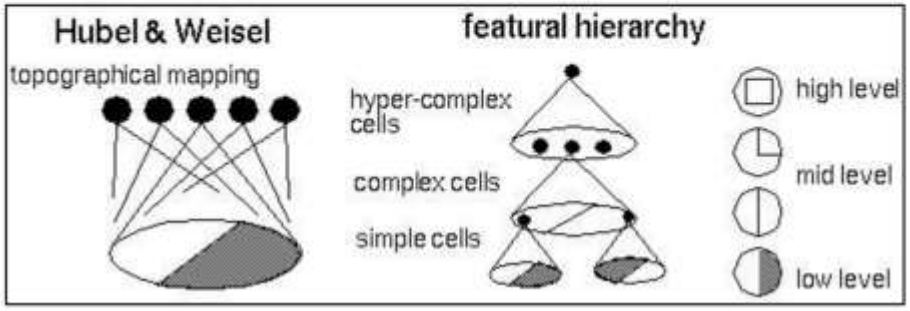
Convolutional Network



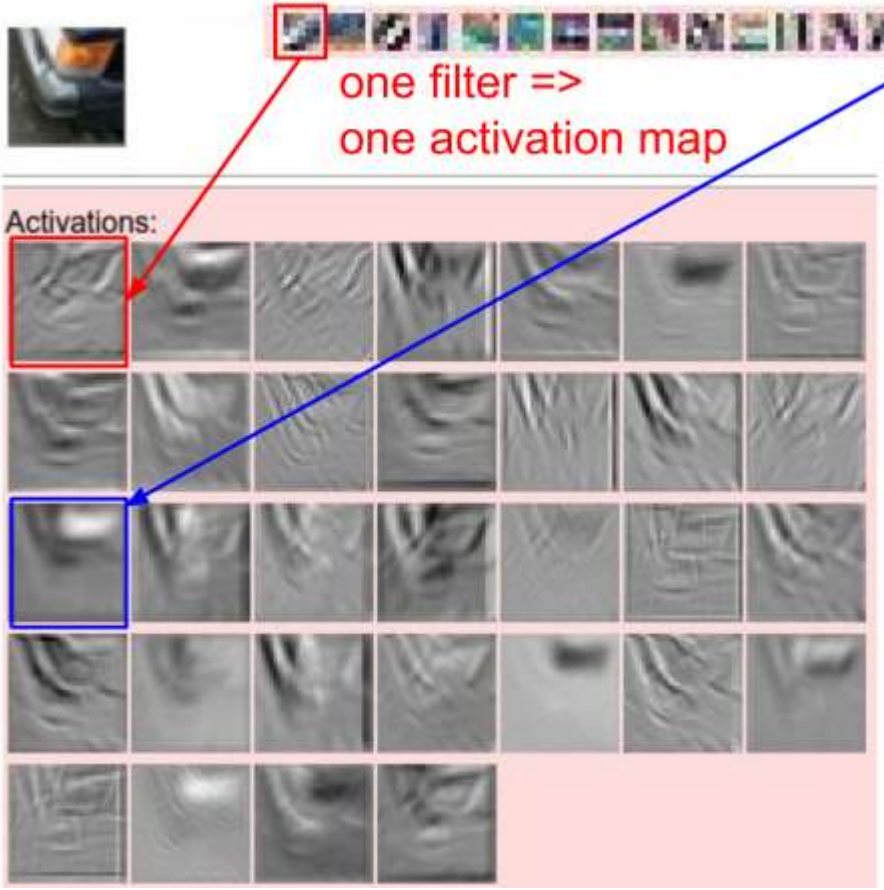
Visualizing Convolutional Filters



Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]



Visualizing Convolutional Features



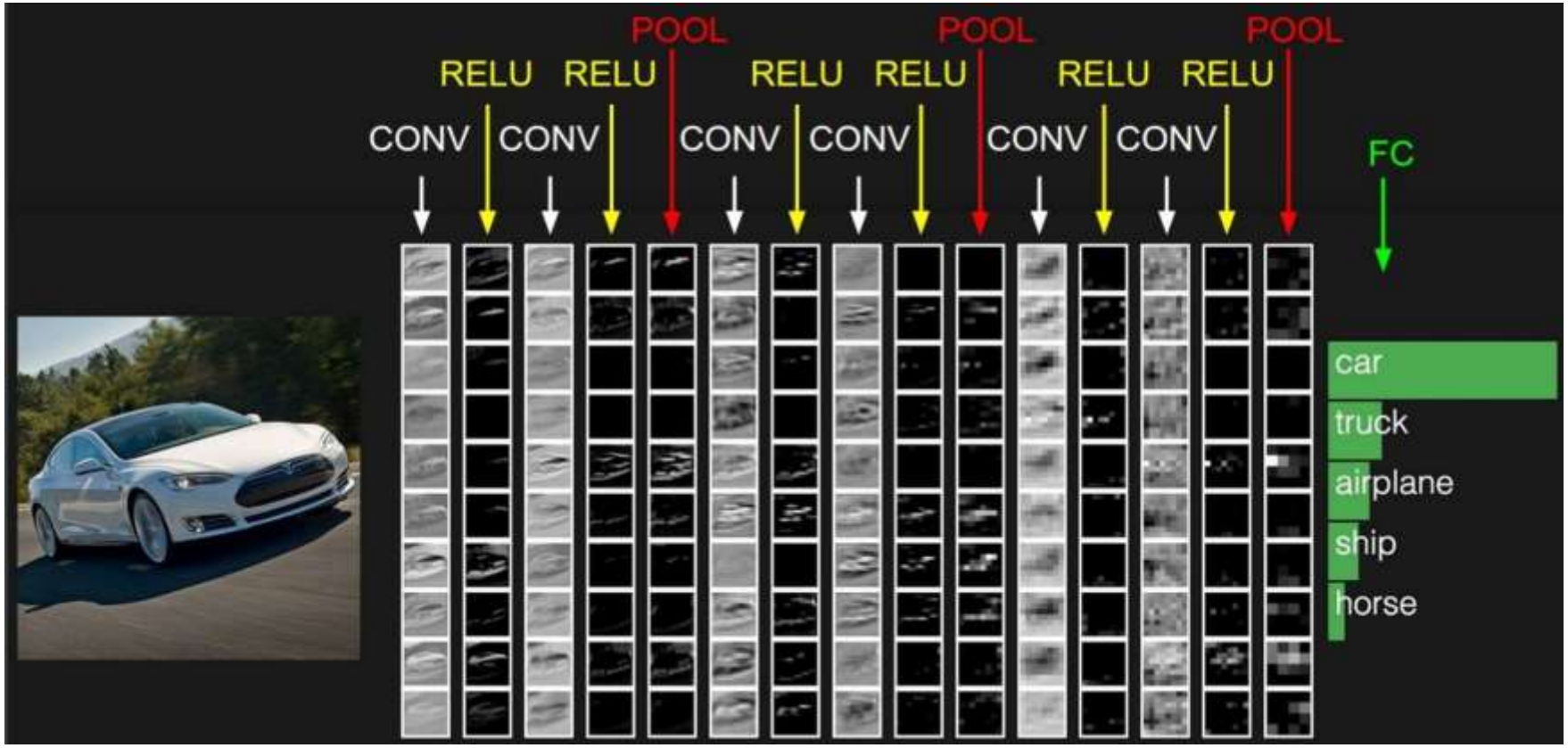
example 5x5 filters
(32 total)

We call the layer convolutional because it is related to convolution of two signals:

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

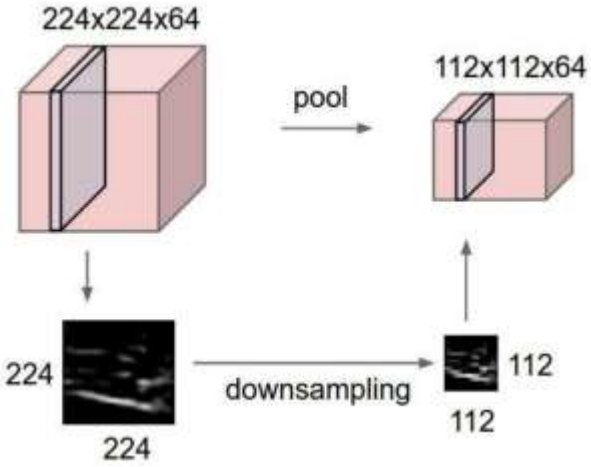
elementwise multiplication and sum of a filter and the signal (image)

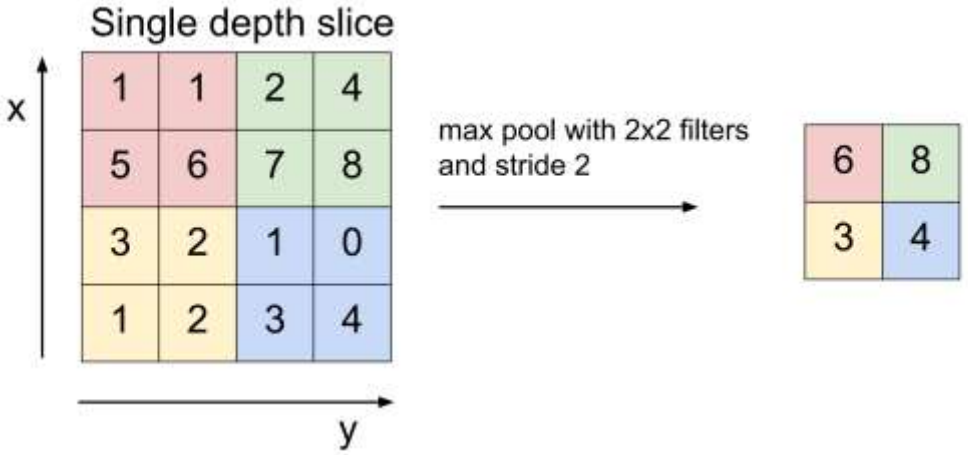
Visualizing Convolutional Features



Pooling Layers

- Makes the representations smaller and more manageable
- Operates over each activation map independently





AlexNet

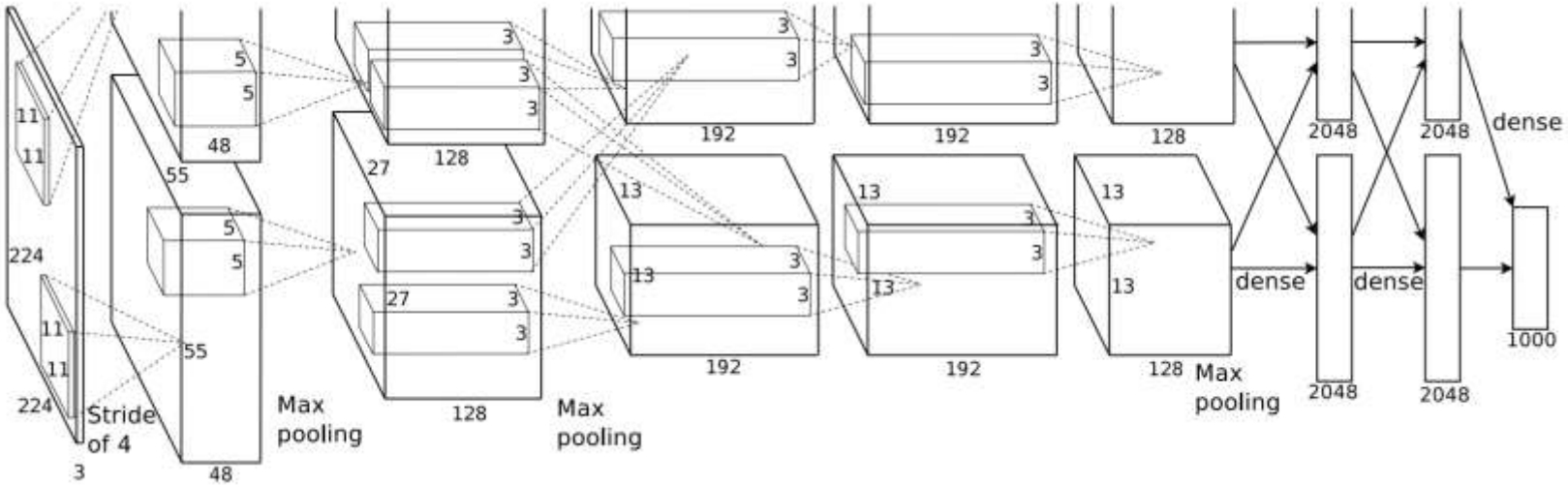


Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network's input is 150,528-dimensional, and the number of neurons in the network's remaining layers is given by 253,440–186,624–64,896–64,896–43,264–4096–4096–1000.

Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. *Neural Information Processing Systems 2012.* ***Cited by: 107214***

Historical Development

- Brief History
  - 1960s, Hubel and Wisel showed that cat visual cortices contain neurons that respond to small regions of the visual field.
  - 1980, Kunihiko Fukushima introduced **Neocognitron**, brought convolutional layers to neural networks (published at NIPS in 1987)
  - 1989, Yann LeCun et al. used back-propagation to learn CNN
  - 1990, Yamaguchi et al. introduced max pooling
  - 1998, Yann LeCun invented LeNet-5, a 7-layer CNN which has been applied to digit recognition on bank cheques
  - 2004, K. S. Oh and K. Jung introduced the idea of using GPU
  - 2012, Alex Krizhevsky et al. won ImageNet 2012 with AlexNet

Summary

- Artificial neural networks
  - The M-P neuron model
  - Feedforward neural network
  - Expressive power of ANN
- Backpropagation algorithm
  - Criterion function, activation function
  - Feedforward procedure
  - Backpropagation procedure
  - Stochastic/Batch mode