Neural Networks

Shikui Tu

Department of Computer Science and Engineering, Shanghai Jiao Tong University

2021-05-14

Outline

- Training the neural networks
  - Backpropagation (BP) algorithm
- Convolutional neural networks (CNN)
  - Overview
  - Network details: convolution, pooling, activation, loss functions
- ResNet
- DenseNet
- RNN

Overall neural network function

- Overall network function (using sigmoidal activation function)

$$y_k(\mathbf{x}, \mathbf{w}) = \sigma\left(\sum_{k=1}^{M} w_{kj}^{(2)} h\left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}\right) + w_{k0}^{(2)}\right)$$

  where weight parameters are grouped into **w**

- Neural network is a function that maps **x** to **y**, controlled by adjustable **w**.

Error Function

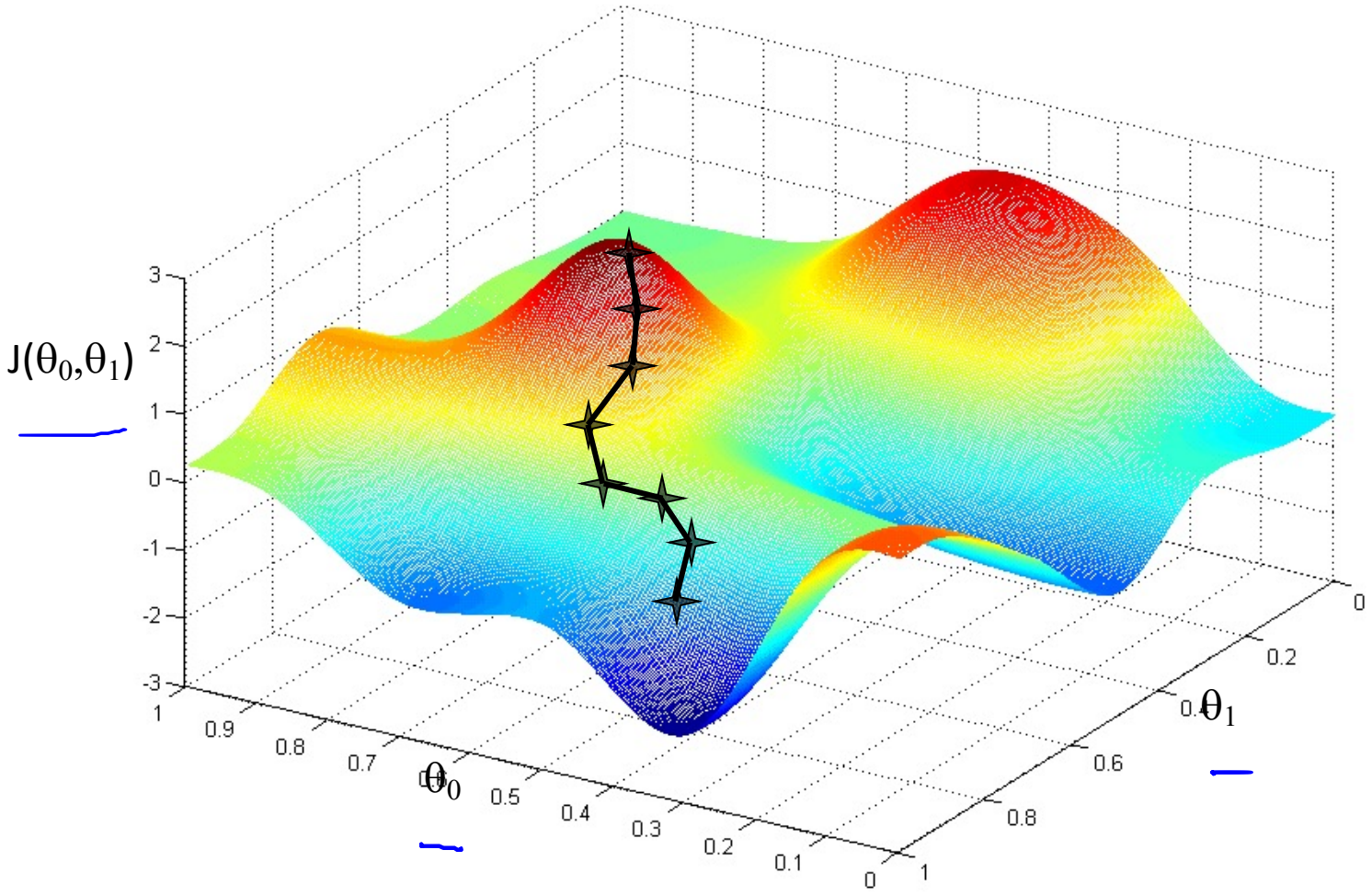
- For a set of N training samples $\{\mathbf{x}_n, \mathbf{t}_n\}_{n=1}^{N}$

$$E(\mathbf{w}) = \sum_{n=1}^{N} E_n(\mathbf{w})$$

- How to evaluate the derivative for one sample pair in the error function?

$$\nabla E_n(\mathbf{w})$$

Gradient Descent



Ng, A. Linear Regression with One Variable [PowerPoint slides]

Error function for linear function

For a linear function

$$y_j = \sum_i w_{ji} x_i$$

together with a squared error function

$$E_n = \frac{1}{2} \sum_j (y_{nj} - t_{nj})^2$$

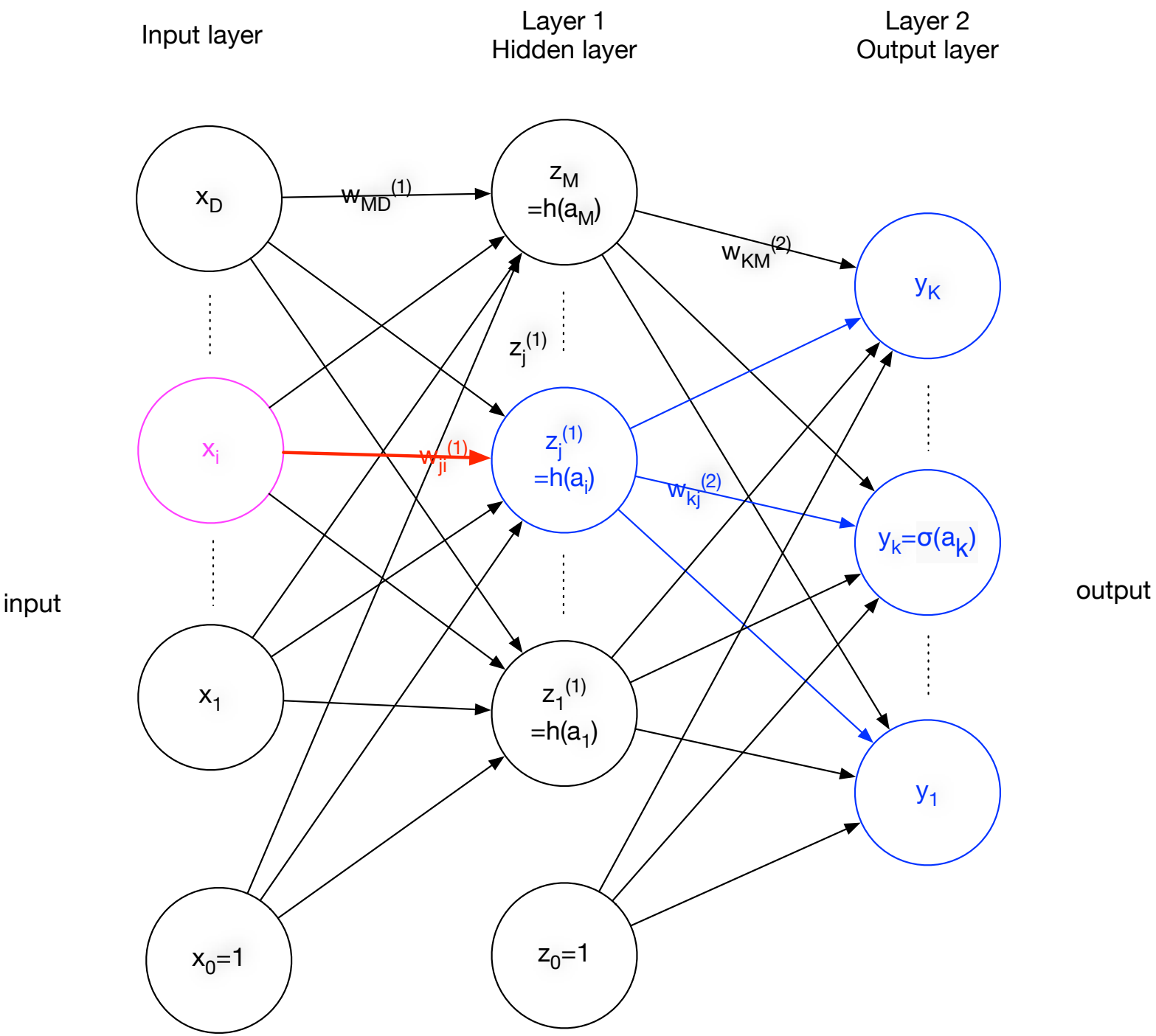
where $y_{nj} = y_j(\mathbf{x}_n, \mathbf{w})$. The gradient w.r.t. w_{ji} is

$$\frac{\partial E_n}{\partial w_{ji}} = (y_{nj} - t_{nj}) x_{ni}$$

which can be interpreted as a *local* computation involved with the product of

- An 'error' signal $y_{nj} - t_{nj}$ associated with the *output* end of the link w_{ji}
- The variable x_{ni} associated with the *input* end of the link

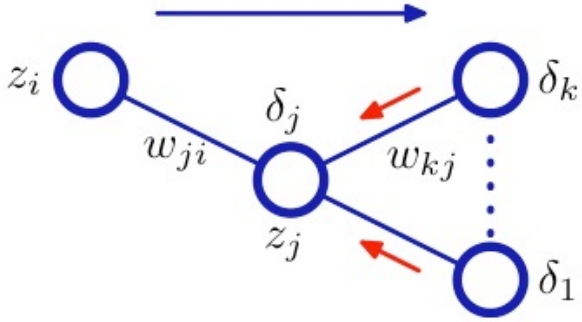
We will show that this property also holds for more complicated FFNN.


Input layer
Layer 1
Hidden layer
Layer 2
Output layer
input
output
x_D
x_i
x_1
$x_0=1$
z_M
$=h(a_M)$
$z_j^{(1)}$
$z_1^{(1)}$
$=h(a_1)$
$z_0=1$
y_K
$y_k=\sigma(a_k)$
y_1
$w_{MD}^{(1)}$
$w_{ji}^{(1)}$
$w_{KM}^{(2)}$
$w_{kj}^{(2)}$

Error backpropagation (1/3)

Note that E_n depends on the weight w_{ji} only via the summed input a_j to unit j. We can apply the chain rule for partial derivative to give

$$\frac{\partial E_n}{\partial w_{ji}} = \frac{\partial E_n}{\partial a_j}\frac{\partial a_j}{\partial w_{ji}}$$



Introduce a useful notation

$$\delta_j \equiv \frac{\partial E_n}{\partial a_j}$$

where δ's are referred to as *errors*. Because $a_j = \sum_i w_{ji} z_i$, we have

$$\frac{\partial a_j}{\partial w_{ji}} = z_i$$

Therefore

$$\frac{\partial E_n}{\partial w_{ji}} = \delta_j z_i$$

Error backpropagation (2/3)

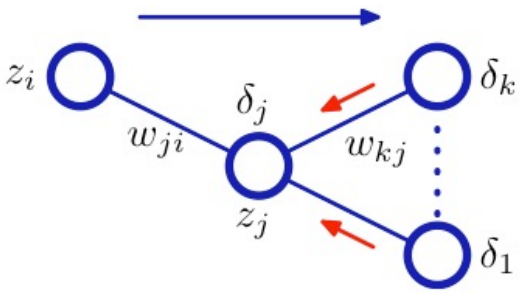
To evaluate the δ's for the hidden units at level (l), we again use the chain rule

$$\delta_j^{(l)} \equiv \frac{\partial E_n}{\partial a_j^{(l)}} = \sum_k \frac{\partial E_n}{\partial a_k^{(l+1)}} \frac{\partial a_k^{(l+1)}}{\partial a_j^{(l)}}$$

where the sum runs over all units k to which unit j sends connection.

Since we have defined $\delta_k^{l+1} \equiv \frac{\partial E_n}{\partial a_k^{(l+1)}}$, and $a_k^{(l+1)} = \sum_j w_{kj}^{(l)} h(a_j^{(l)})$, we obtain the *backpropagation* formula

$$\delta_j^{(l)} = h'(a_j^{(l)}) \sum_k w_{kj}^{(l)} \delta_k^{(l+1)}$$



Error propagation (3/3)

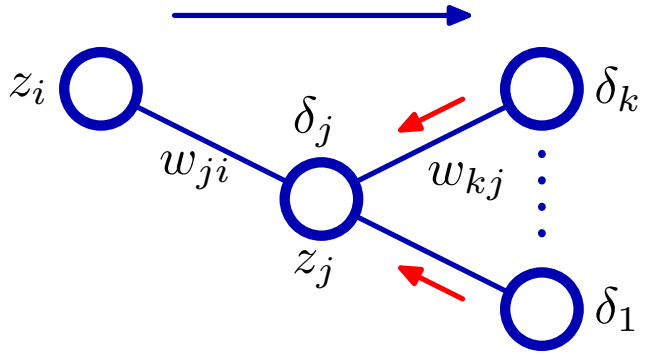
For the squared error function and for the output units, we have

$$\delta_k \equiv \frac{\partial E_n}{\partial a_k} = y_k - t_k$$

The simplified error backpropagation formula

$$\delta_j = h'(a_j) \sum_k w_{kj} \delta_k$$

is a weighted sum of the errors.



Error propagation algorithm

- Forward propagation: use equation

$$a_k = \sum_j w_{kj} z_j = \sum_j w_{kj} h(a_j)$$

to find the activations of all the hidden and output units

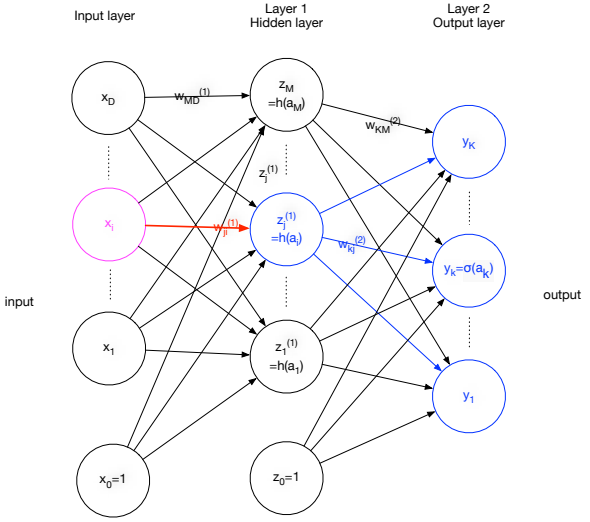
- Evaluate δ_k for all the output units using $\delta_k = y_k - t_k$
- Backpropagate δ's using

$$\delta_j = h'(a_j) \sum_k w_{kj} \delta_k$$

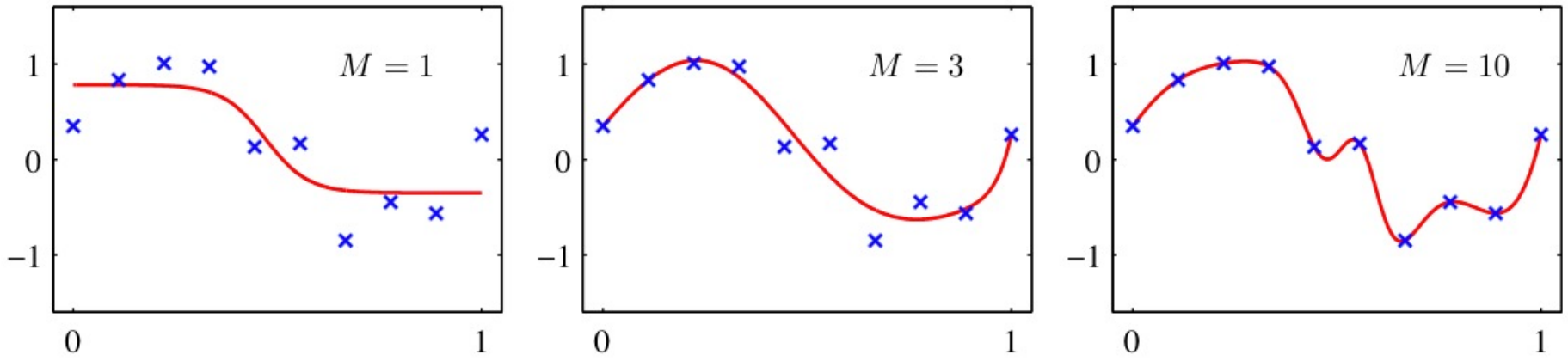
to obtain δ_j for each hidden unit in the network

- Evaluate derivatives

$$\frac{\partial E_n}{\partial w_{ji}} = \delta_j z_i$$

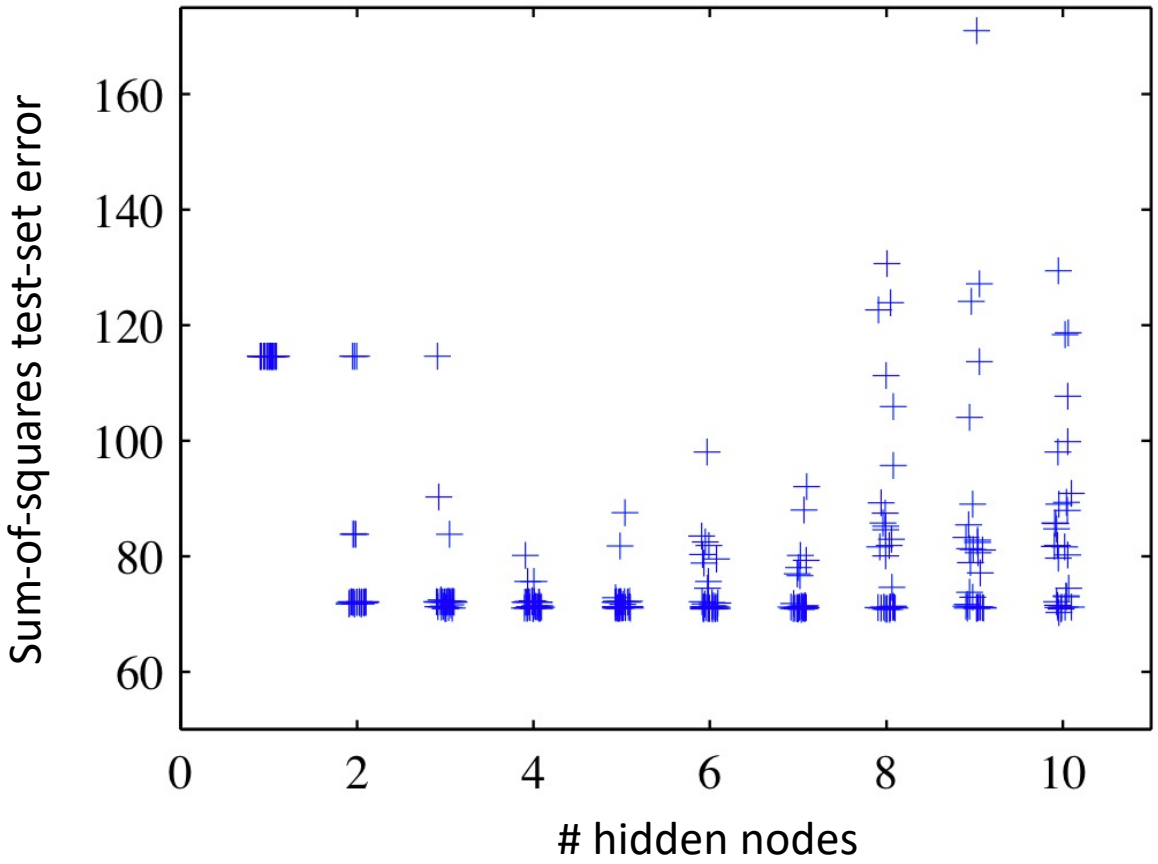


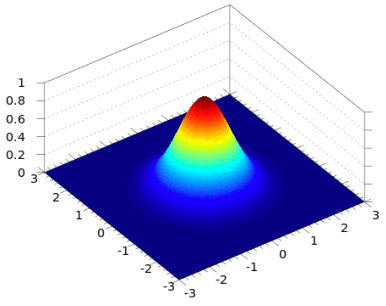
Regularization in Neural Nets



Local Minima

- Test error as a function of the number of hidden units. 30 random starts for each network size.





PRML

Regularization

- The simplest regularizer is a Gaussian prior, with a new objective function

$$\widetilde{E}(\mathbf{w}) = E(\mathbf{w}) + \frac{1}{2}\mathbf{w}^T\mathbf{w}$$

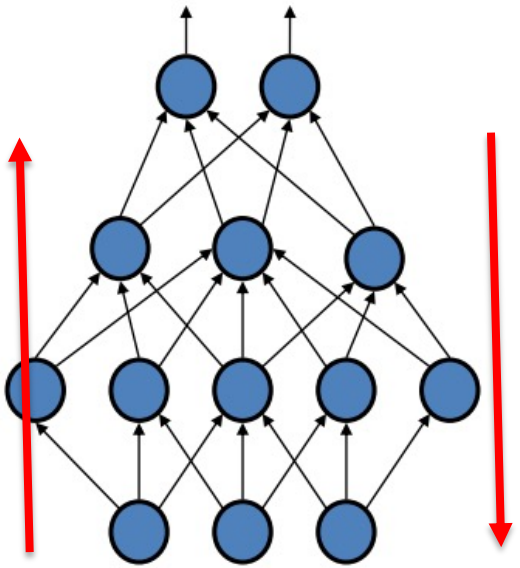
Can be interpreted from Bayesian learning perspective.

What is wrong with back-propagation?

- It requires labeled training data.
  - Almost all data is unlabeled.
- The brain needs to fit about 10^14 connection weights in only about 10^9 seconds.
  - Unless the weights are highly redundant, labels cannot possibly provide enough information.
- The learning time does not scale well
  - It is very slow in networks with multiple hidden layers.
- The neurons need to send two different types of signal
  - Forward pass: signal = activity = y
  - Backward pass: signal = dE/dy
- Gradient descent with backpropagation is not guaranteed to find the global minimum of the error function, but only a local minimum; also, it has trouble crossing plateaus in the error function landscape.

Deep Learning (present solution)

- Train first layer using data without the labels (unsupervised)
- Freeze the first layer parameters and start training the second layer using the output of first layer input
  - repeat so as many layers as desired

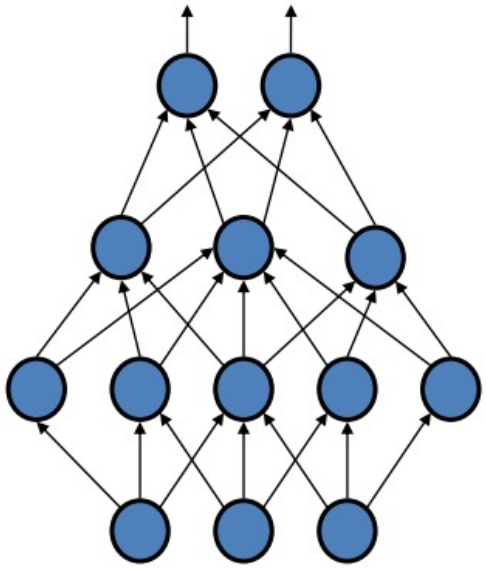


- Use the final outputs as inputs to a supervised layer/model and train the last supervised layer(s) (early weights frozen)

Unfreeze all weights and fine tune the full network by training with a supervised way, given the preprocessed weight settings

Question

- What are the problems of fully-connected multiple-layer neural networks?



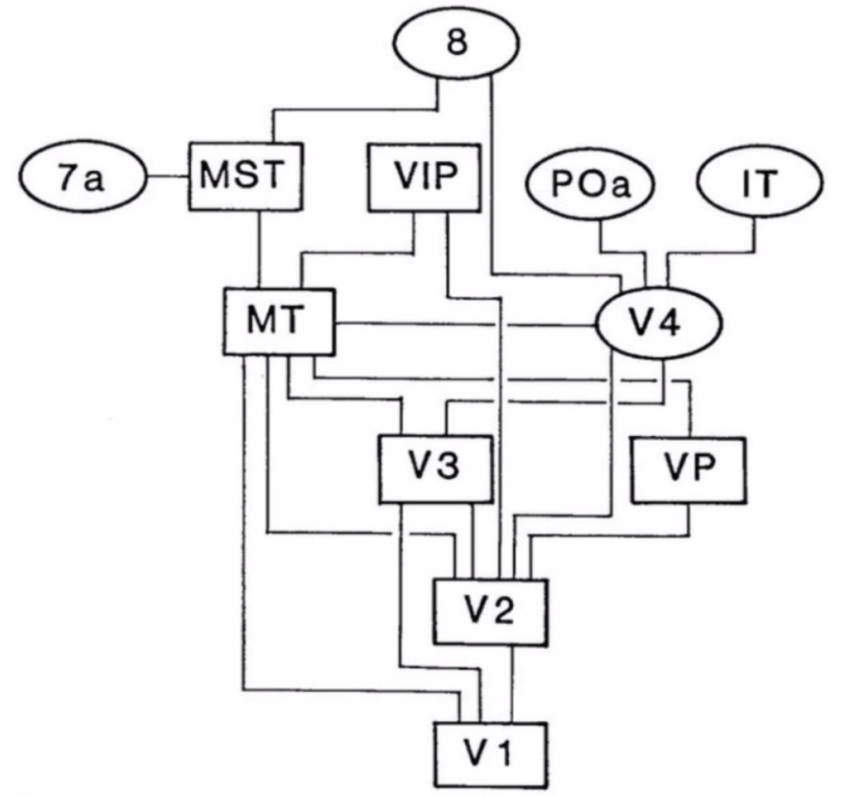
Outline

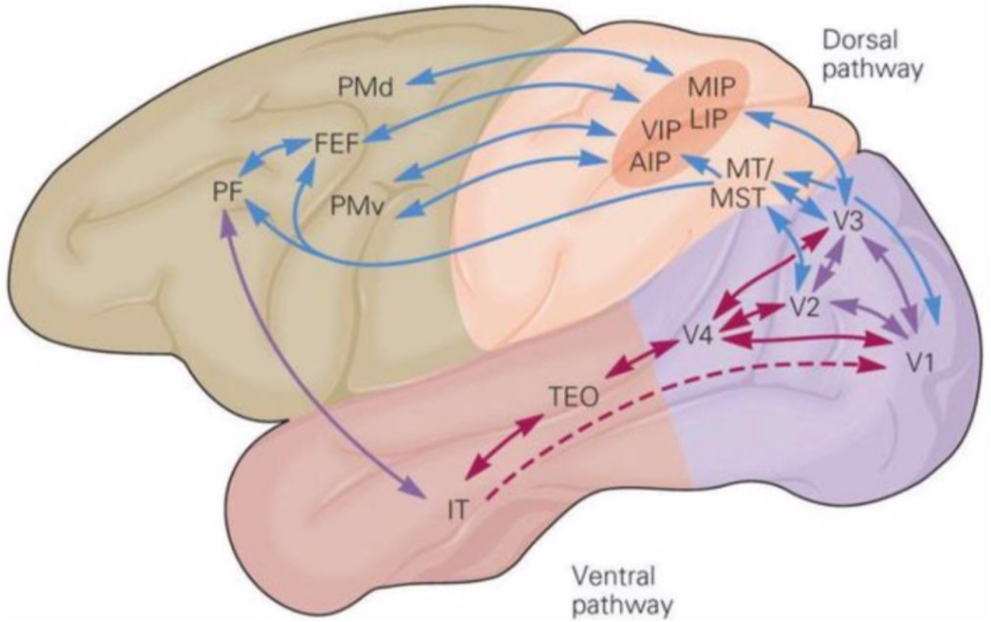
- Training the neural networks
  - Backpropagation (BP) algorithm

- Convolutional neural networks (CNN)
  - Overview
  - Network details: convolution, pooling, activation, loss functions

- ResNet
- DenseNet
- RNN

Visual cortex





Starting from V1 primary visual cortex, visual signal is transmitted upwards, becoming more complicated and abstract.

视觉信息处理过程

- 生物视觉信息处理的大致过程
  - –0ms: 光刺激发生
  - –10ms: 视网膜光电转换和编码
  - –35ms: 视网膜“模数“转换和神经脉冲
  - –45ms: 外侧膝状体(LGN)中继转发
  - –55ms: V1 方向选择细胞响应
  - –85ms: V4 大范围整合
  - –100ms: 人脸选择细胞响应
  - –160-220ms: 对象分类识别

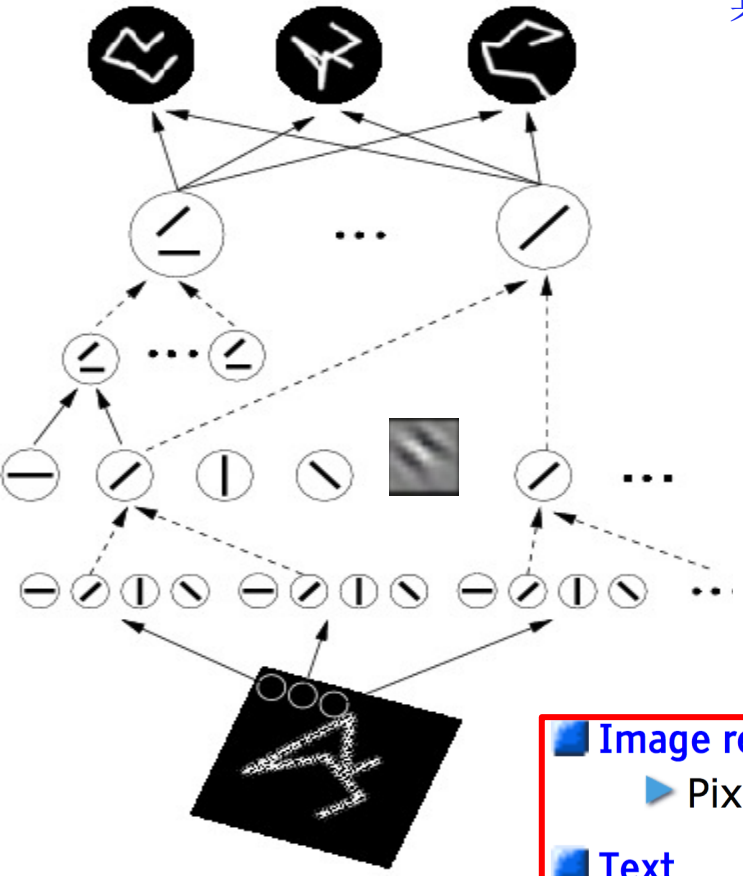
“只因在人群中多看了你一眼”

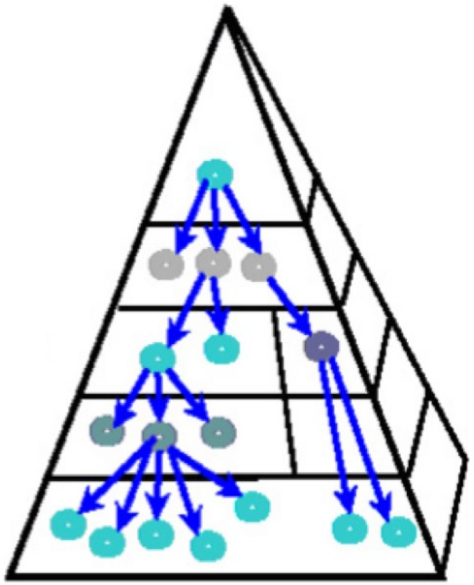
Maunsell and Gibson 1992; Raiguel et al. 1989; Nowak et al. 1995; Schmolesky et al. 1998; Thorpe, Fize& Marlot 1996

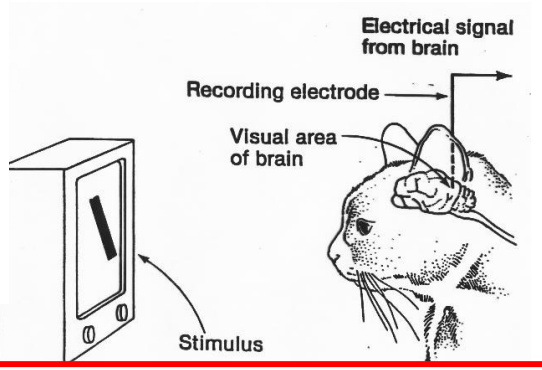
Feature Detection Theory 特征检测理论（1960）

Hubel Wiesel

具备广泛的普遍性！



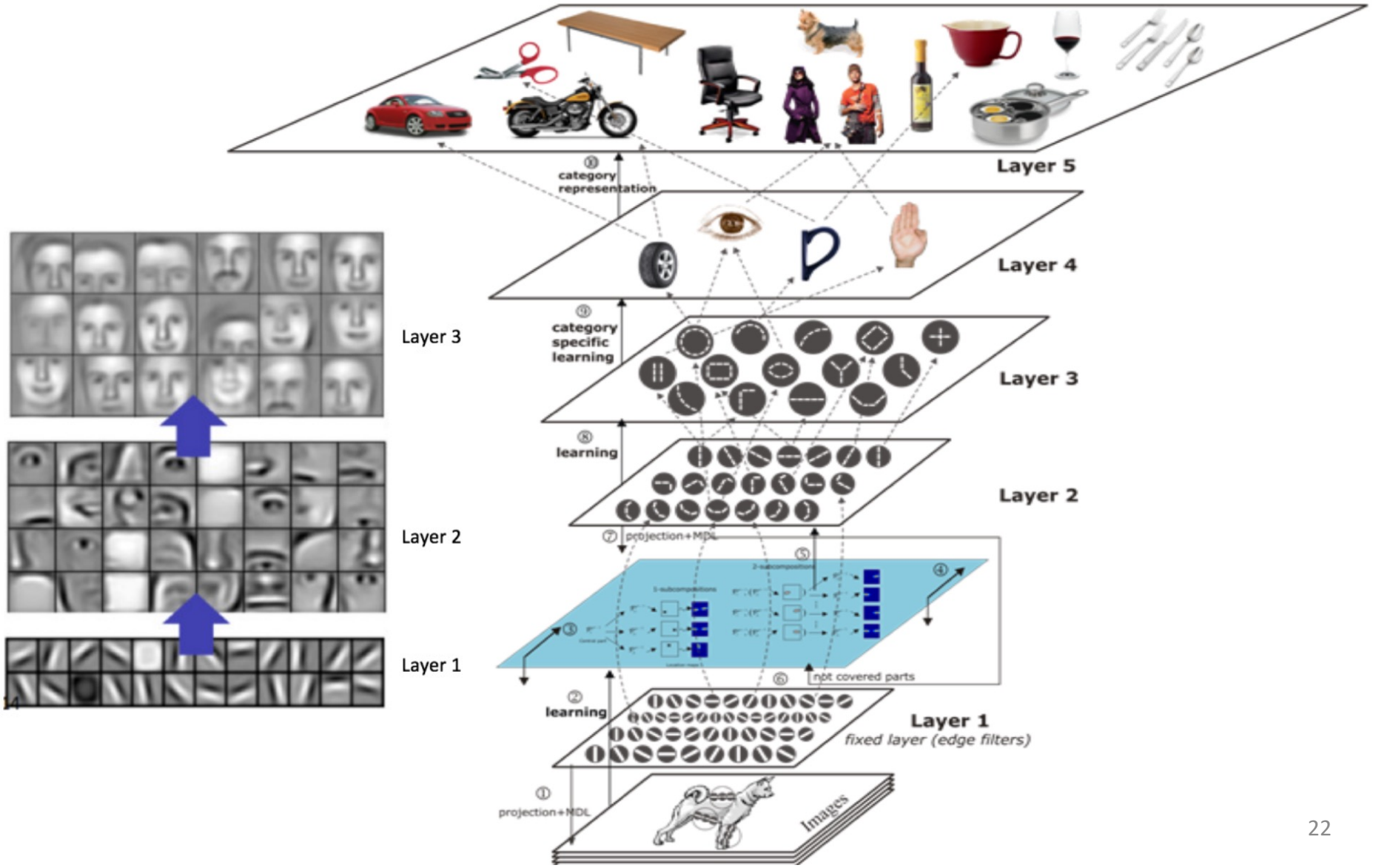




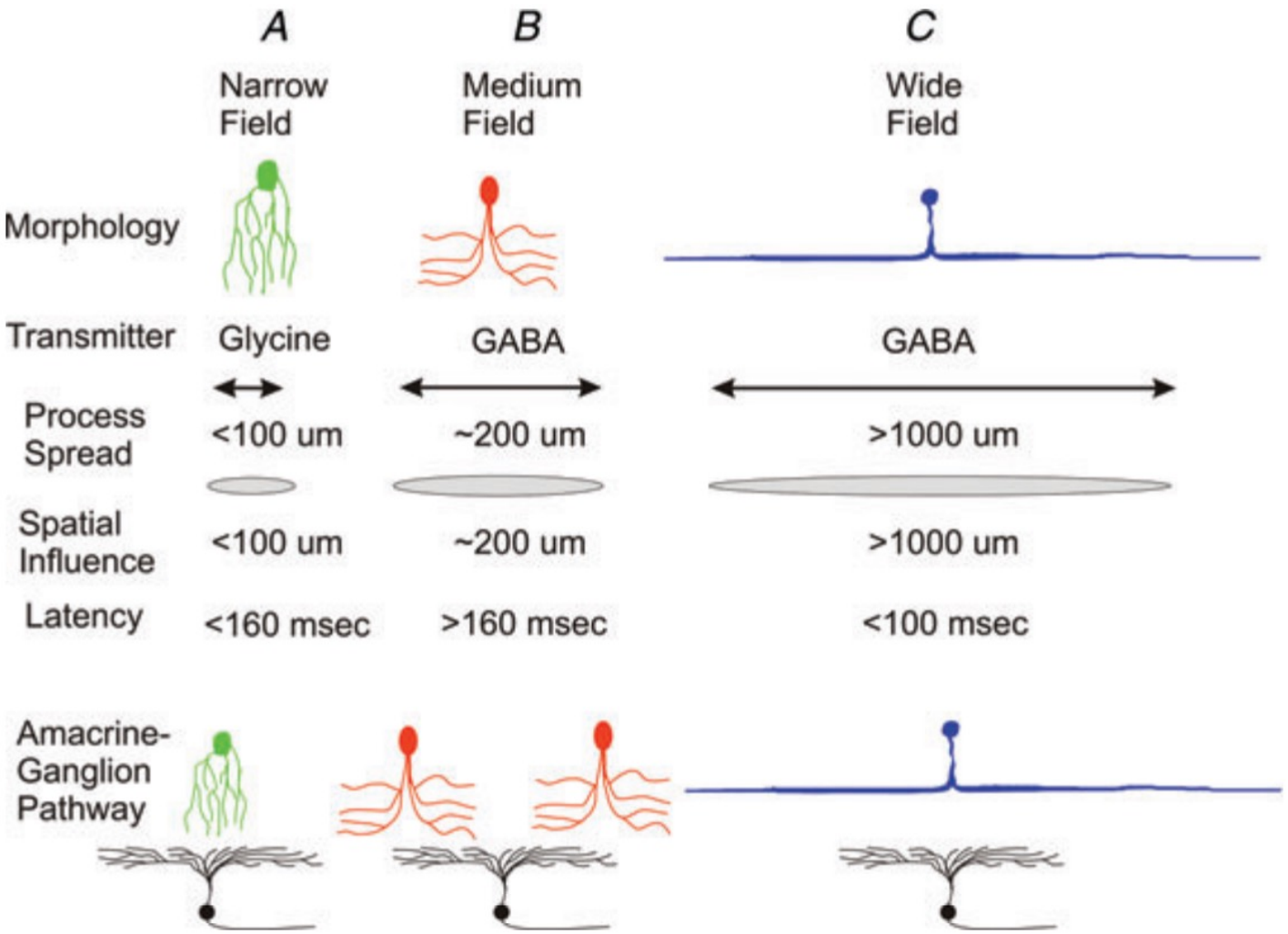
- **Image recognition**
  - Pixel → edge → texton → motif → part → object
- **Text**
  - Character → word → word group → clause → sentence → story
- **Speech**
  - Sample → spectral band → sound → … → phone → phoneme → word →

Slide from Lei Xu

Hierarchical representations

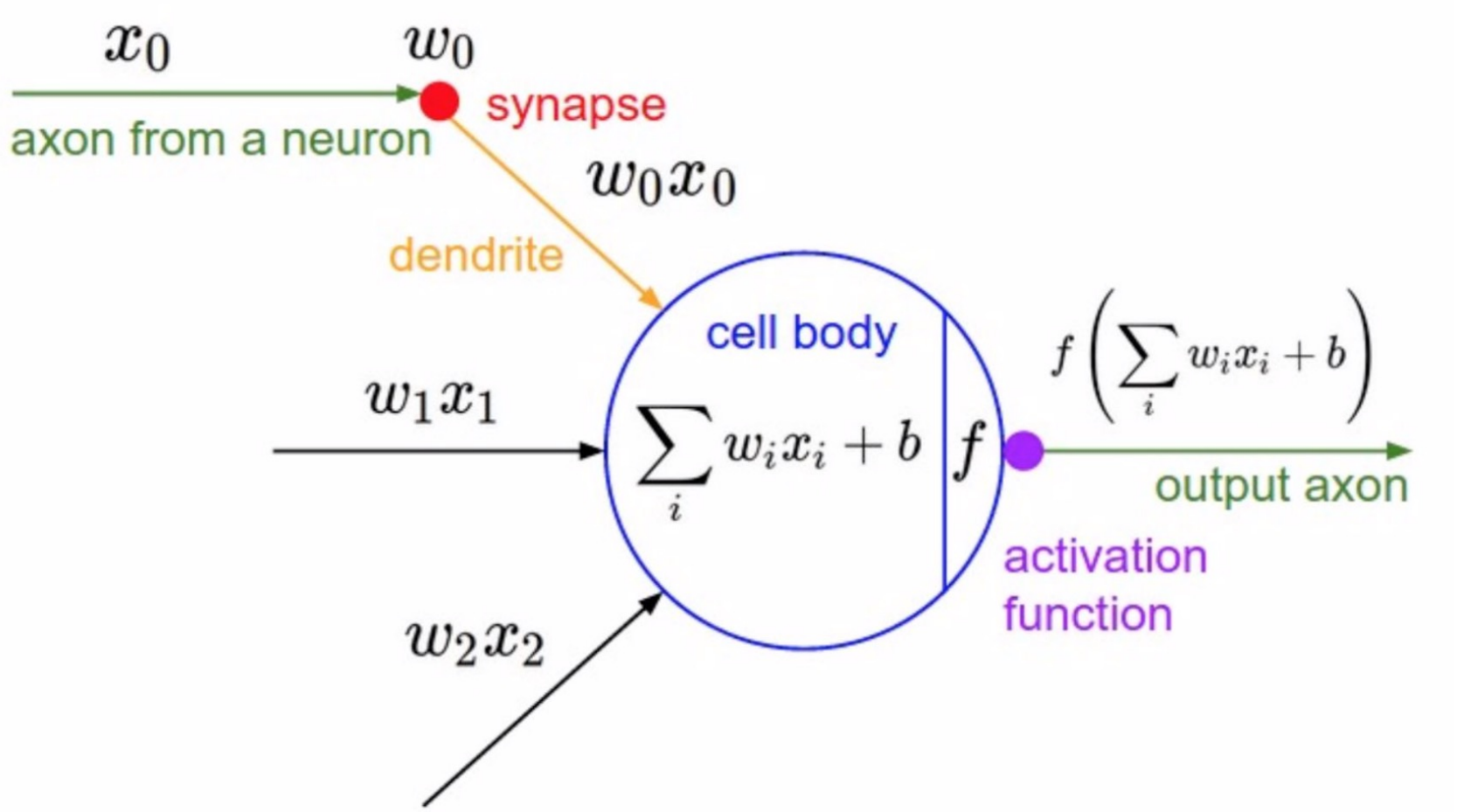


narrow field, medium field and wide field amacrine cells

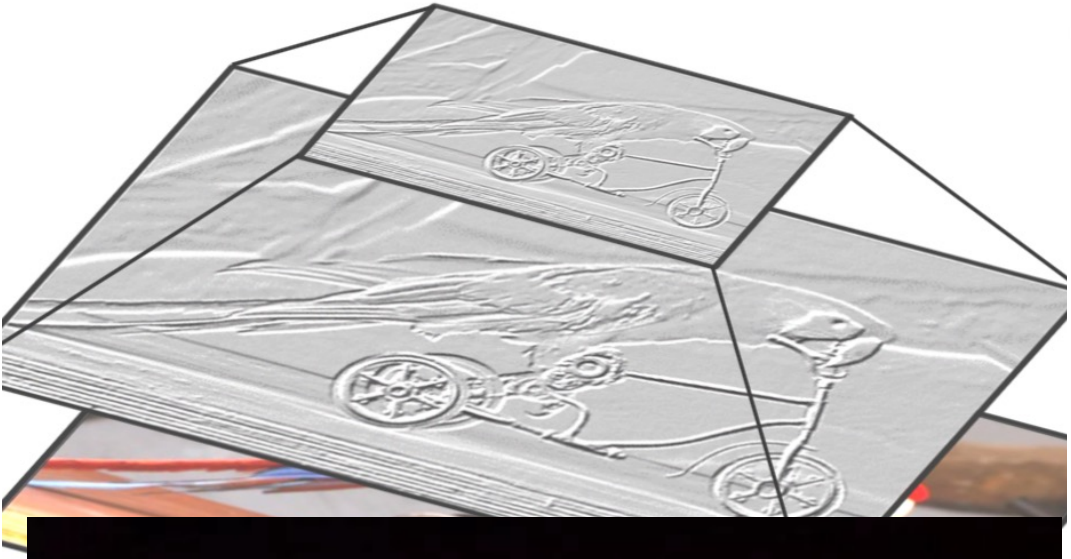


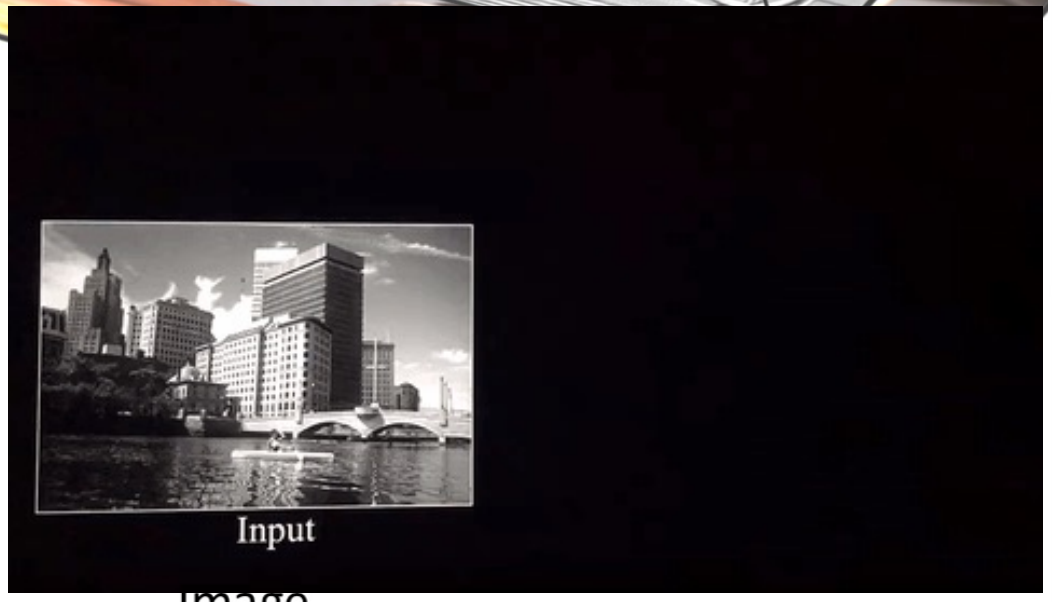
Frank S. Werblin, J Physiol 589.15 (2011) pp 3691–3702

The neuron model

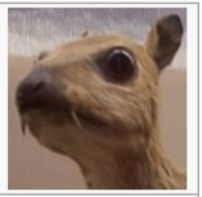
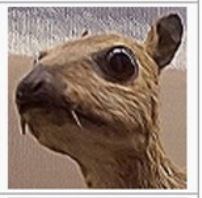
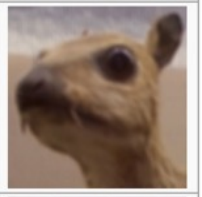
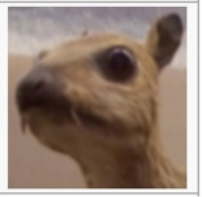


A Basic Module of the **Convolutional Neural Nets**





Slide from Lei Xu

Operation	Filter	Convolved Image
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur (approximation)	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

Basic functions: *constant, successor, projection*

$$f(x_1,\cdots,x_k)=n,\quad f(x)=x+1,\quad P_i(x_1,\cdots,x_k)=x_i$$

Composition operator

$$h\circ(g_1,\ldots,g_m)\overset{\text{def}}{=}f(x_1,\ldots,x_k)=h(g_1(x_1,\ldots,x_k),\ldots,g_m(x_1,\ldots,x_k)).$$

Primitive recursion operator $\rho(g,h)\overset{\text{def}}{=}f(y,x_1,\ldots,x_k)$ where

$$f(0,x_1,\ldots,x_k)=g(x_1,\ldots,x_k)$$

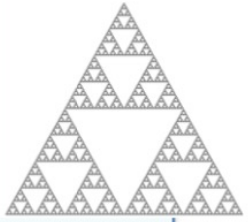
$$f(y+1,x_1,\ldots,x_k)=h(y,f(y,x_1,\ldots,x_k),x_1,\ldots,x_k)$$

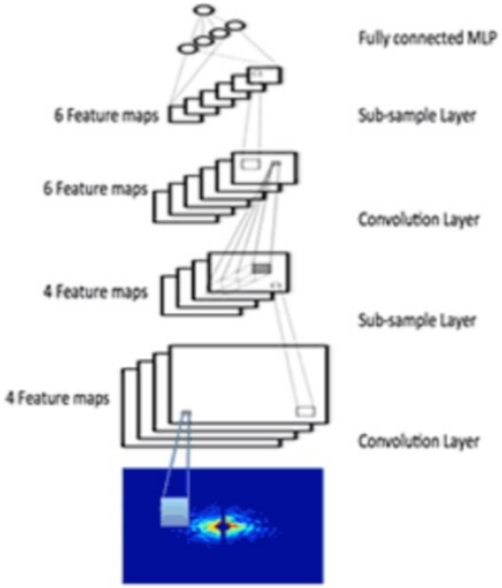
Minimization operator

Given a (k+1)-ary total function $f(y,x_1,\ldots,x_k)$

$$\mu(f)(x_1,\ldots,x_k)=z \overset{\text{def}}{\iff} f(z,x_1,\ldots,x_k)=0 \quad \text{and}$$

$$f(i,x_1,\ldots,x_k)>0 \quad \text{for } i=0,\ldots,z-1.$$





$$f(x,0)=g(x)$$

$$f(x,y+1)=h(x,y,f(x,y))$$

$$h(x,y,x)=z+x^y x$$

$$div(x,y)=\begin{cases}\text{integer portion of } x/y \text{ if } y\neq 0\\ \text{underfined if } y=0\end{cases}$$

$$f(\bar{x},3)=h(\bar{x},2,f(\bar{x},2))$$

$$f(\bar{x},2)=h(\bar{x},1,f(\bar{x},1))$$

$$f(\bar{x},1)=h(\bar{x},0,f(\bar{x},0))$$

$$f(\bar{x},0)=g(\bar{x})$$

Slide from Lei Xu

Some influential CNN architectures

- **LeNet (1990s):** LeNet-5: a pioneering 7-level convolutional network by LeCun et al. in 1998
- **1990s to 2012:** In the years from late 1990s to early 2010s convolutional neural network were in incubation. As more and more data and computing power became available, tasks that convolutional neural networks could tackle became more and more interesting.
- **AlexNet (2012) –** In 2012, Alex Krizhevsky (and others) released AlexNet which was a deeper and much wider version of the LeNet and won by a large margin the difficult ImageNet Large Scale Visual Recognition Challenge (ILSVRC) in 2012. It was a significant breakthrough with respect to the previous approaches and the current widespread application of CNNs can be attributed to this work.
- **ZF Net (2013) –** The ILSVRC 2013 winner was a Convolutional Network from Matthew Zeiler and Rob Fergus. It became known as the ZFNet (short for Zeiler & Fergus Net). It was an improvement on AlexNet by tweaking the architecture hyperparameters.
- **GoogLeNet (2014) –** The ILSVRC 2014 winner was a Convolutional Network from Szegedy et al. from Google. Its main contribution was the development of an *Inception Module* that dramatically reduced the number of parameters in the network (4M, compared to AlexNet with 60M).
- **VGGNet (2014) –** The runner-up in ILSVRC 2014 was the network that became known as the VGGNet. Its main contribution was in showing that the depth of the network (number of layers) is a critical component for good performance.
- **ResNets (2015) –** Residual Network developed by Kaiming He (and others) was the winner of ILSVRC 2015. ResNets are currently by far state of the art Convolutional Neural Network models and are the default choice for using ConvNets in practice (as of May 2016).
- **DenseNet (August 2016) –** Recently published by Gao Huang (and others), the Densely Connected Convolutional Network has each layer directly connected to every other layer in a feed-forward fashion. The DenseNet has been shown to obtain significant improvements over previous state-of-the-art architectures on five highly competitive object recognition benchmark tasks.

Early CNN: LeNet-5

LeNet-5, a pioneering 7-level convolutional network by LeCun et al. in 1998, that classifies digits, was applied by several banks to recognize hand-written numbers on checks digitized in 32x32 pixel images.

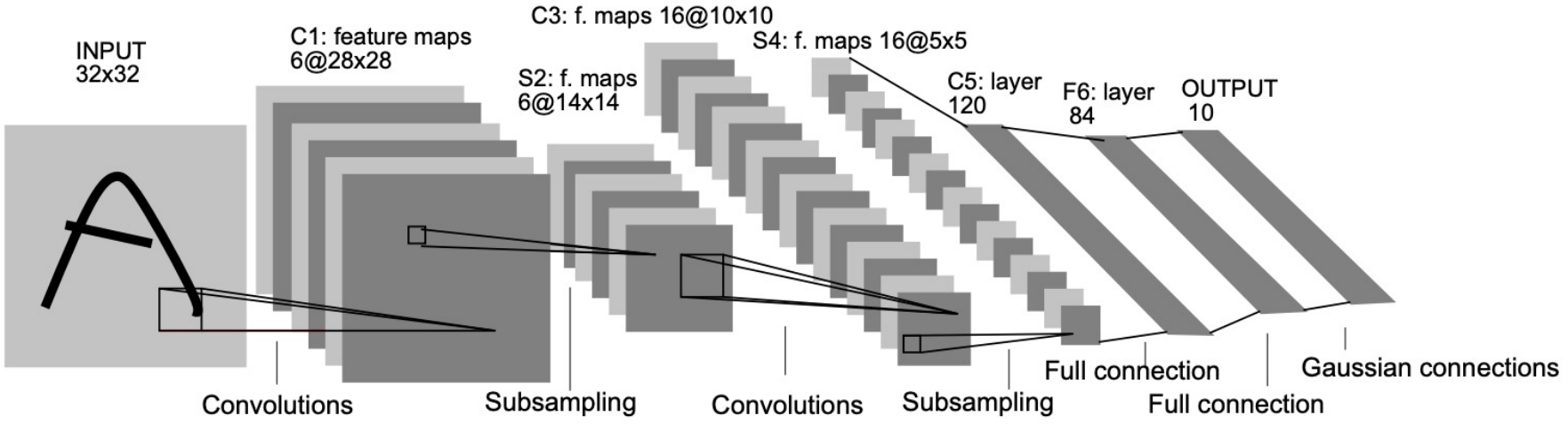


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

LeCun, Yann; Léon Bottou; Yoshua Bengio; Patrick Haffner (1998). "Gradient-based learning applied to document recognition" (PDF). Proceedings of the IEEE. 86 (11): 2278–2324.

Philip Marlowe
6381 Hollywood Blvd # 615
Los Angeles, CA 90028

PORTLAND OR 970
16 JAN 2014 PM 1 L

Dave Fennie
c/o Letter, inc
509 Cascade Ave, Suite H
Hood River, OR 97031

9703120608O

CARROLL O'CONNOR
BUSINESS ACCOUNT
% NANAS, STERN, BIERS AND CO.
9454 WILSHIRE BLVD., STE. 405 273-2501
BEVERLY HILLS, CALIF. 90212

11725

715

March 10 1980

16-24/63
1220

PAY TO THE ORDER OF Pollard-Wittman-Robb Chevrolet $5000.00

Five thousand xx/00 DOLLARS

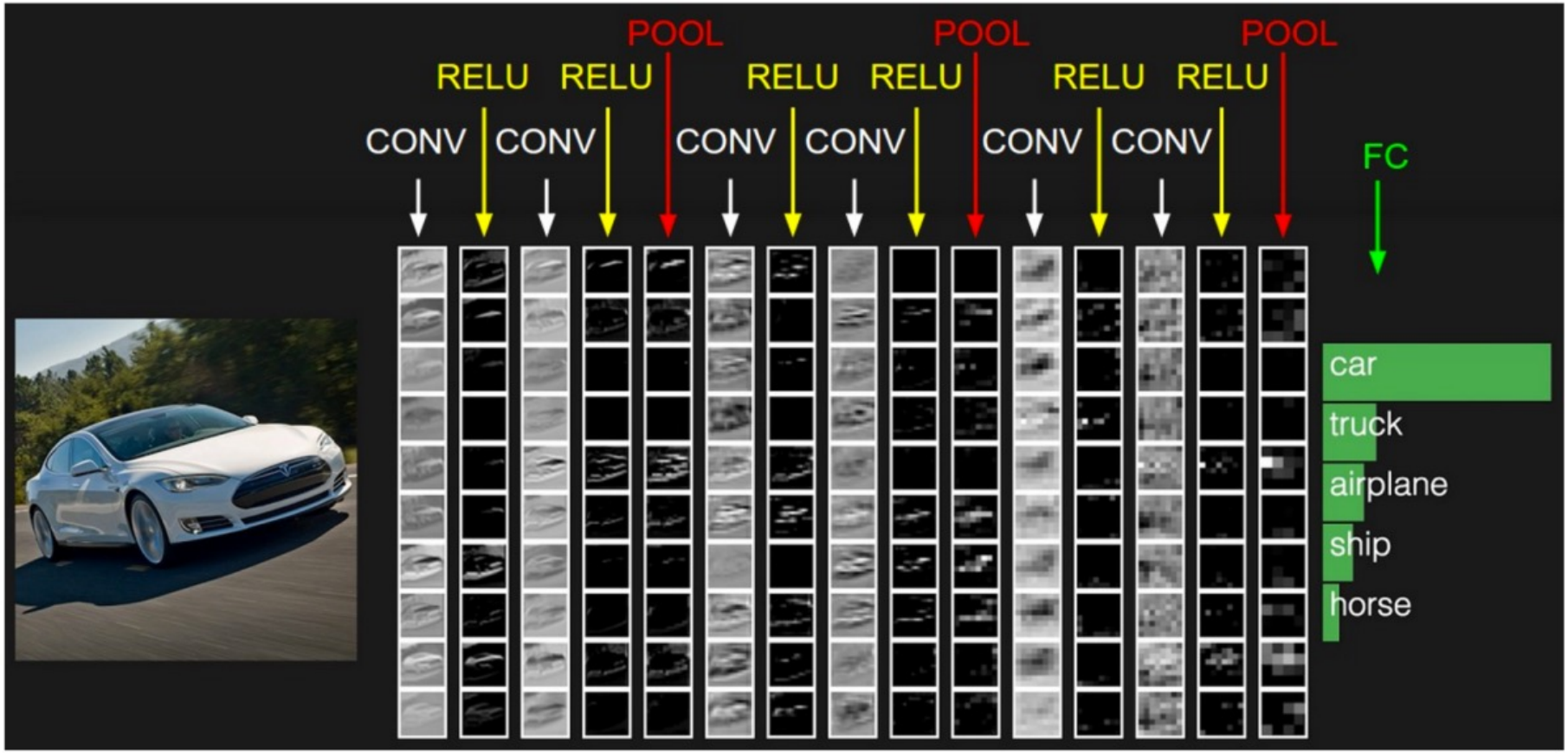
WILSHIRE-DOHENY OFFICE
WELLS FARGO BANK
NATIONAL ASSOCIATION
9101 WILSHIRE BOULEVARD
BEVERLY HILLS, CALIFORNIA 90211
SINCE 1852

MEMO deposit 1980 Chev. pickup for HCO.

Carroll O'Connor

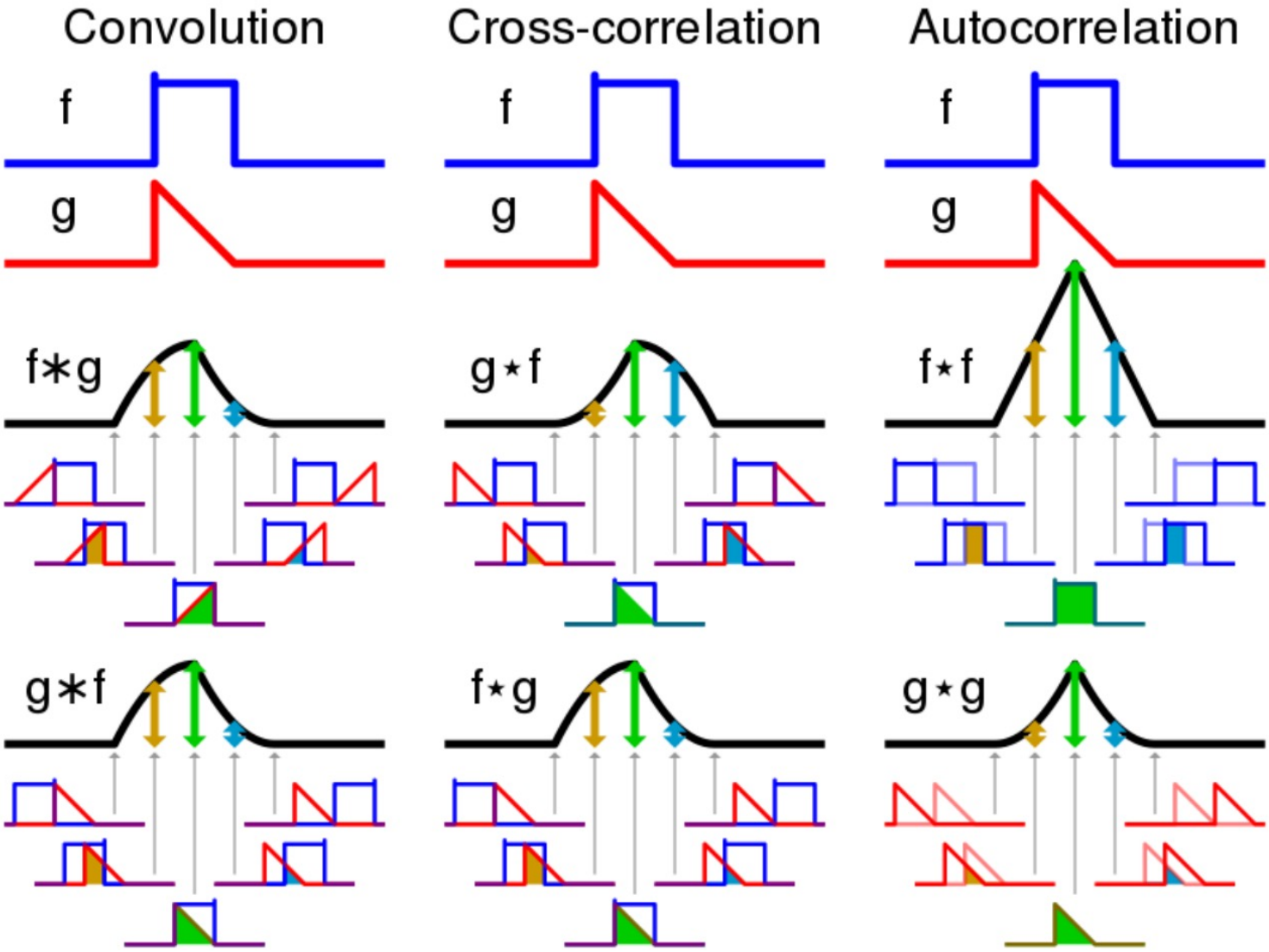
⑆122000024⑆715 0635 11875⑈ ⑈0000500000⑈

A typical modern CNN



In fact, some of the best performing ConvNets today have tens of Convolution and Pooling layers! Also, it is not necessary to have a Pooling layer after every Convolutional Layer.

Convolution

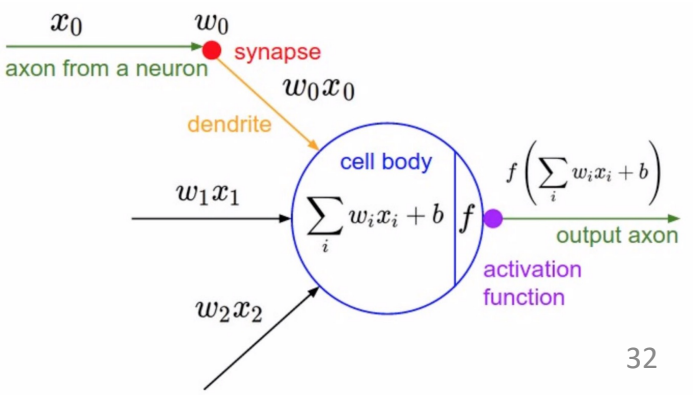


$$(f * g)(t) \triangleq \int_{-\infty}^{\infty} f(\tau) g(t - \tau)\, d\tau.$$

Affine transformations

- A vector is received as input and is multiplied with a matrix to produce an output (to which a bias vector is usually added before passing the result through a non-linearity).

- Applicable to any type of input
  - flattened into a vector
  - Image, sound clip, etc.



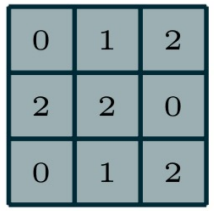
V. Dumoulin and F. Visin, arXiv:1603.07285, 2016

Affine transformations are not good for image-like structured data

- Images, sound clips and many other similar kinds of data have an intrinsic structure.
- They are stored as multi-dimensional arrays.
- They feature one or more axes for which ordering matters (e.g., width and height axes for an image, time axis for a sound clip).
- One axis, called the channel axis, is used to access different views of the data (e.g., the red, green and blue channels of a color image, or the left and right channels of a stereo audio track).

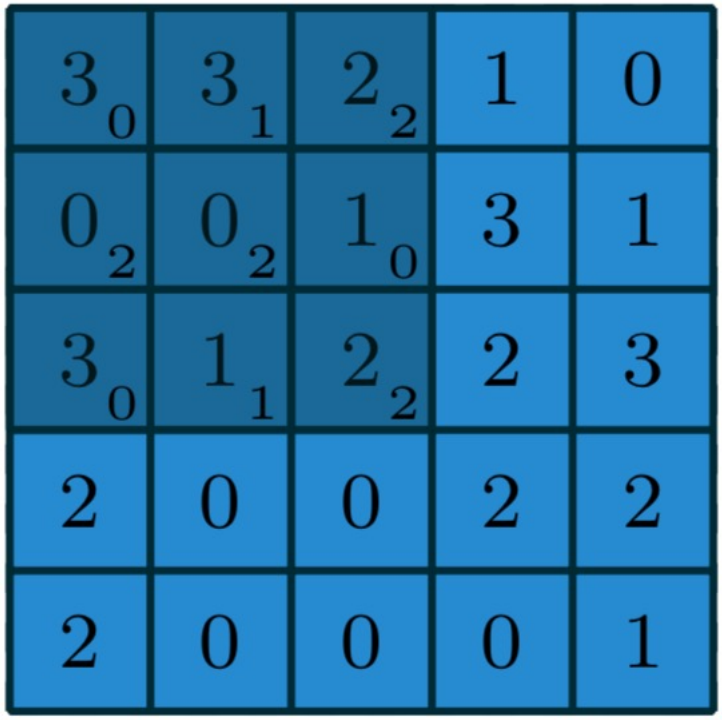
V. Dumoulin and F. Visin, arXiv:1603.07285, 2016

Discrete convolution

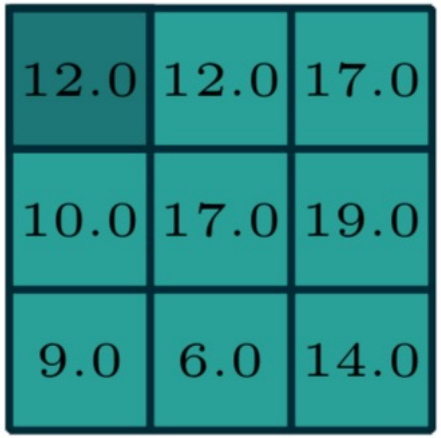


Kernel (3x3)

Input feature map



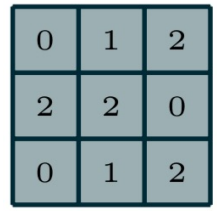
Output feature map



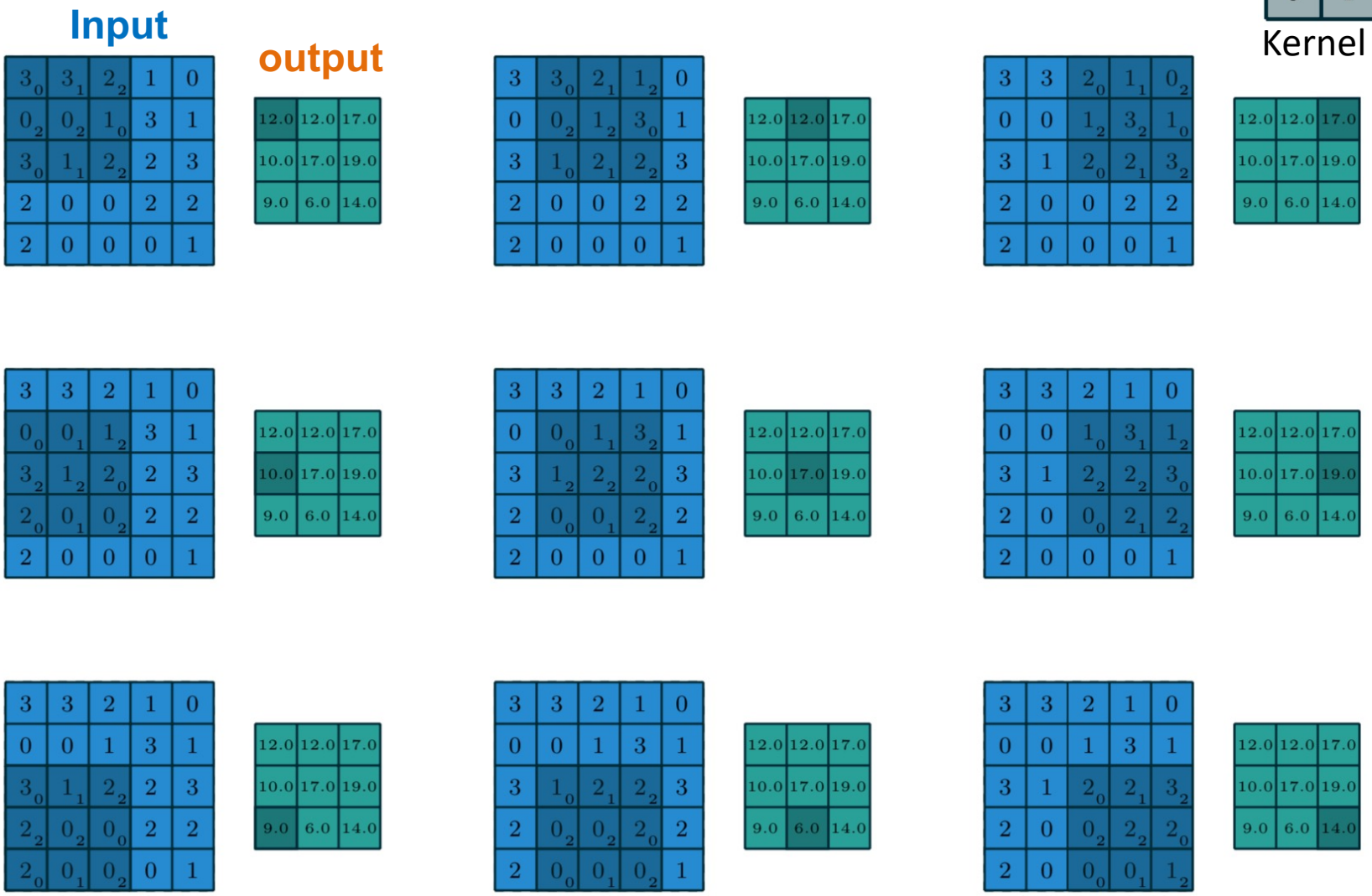
0 x 3 + 1 x 3 + 2 x 2
+ 2 x 0 + 2 x 0 + 0 x 1
+ 0 x 3 + 1 x 1 + 2 x 2
= 12.0

V. Dumoulin and F. Visin, arXiv:1603.07285, 2016

Discrete convolution



Kernel (3x3)



V. Dumoulin and F. Visin, arXiv:1603.07285, 2016

N-D convolution (N=2,3,…)

axis $j = 1, 2, \ldots, N$.

$n \equiv$ number of output feature maps,

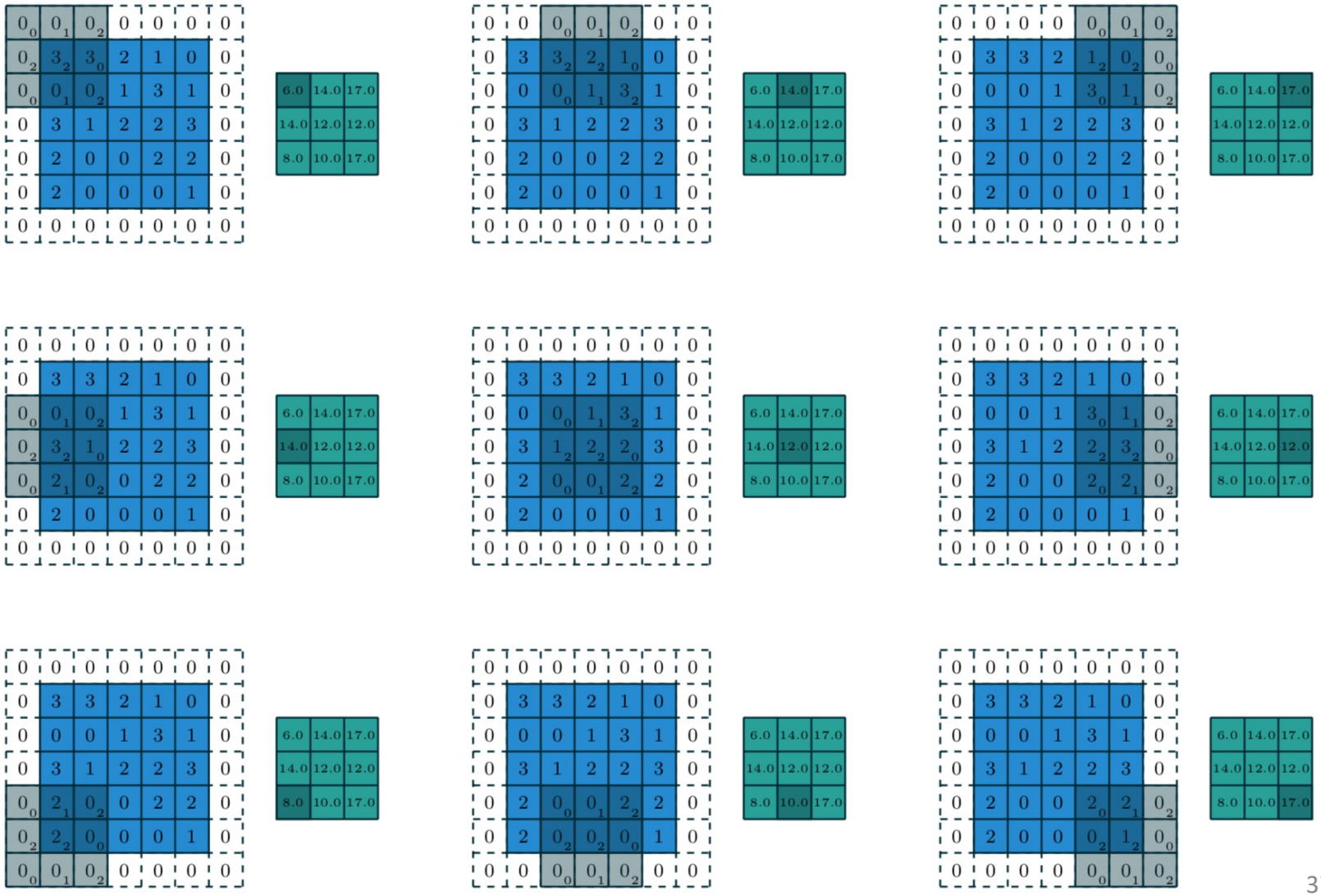
$m \equiv$ number of input feature maps,

$k_j \equiv$ kernel size along axis j.

- i_j: input size along axis j,
- k_j: kernel size along axis j,
- s_j: stride (distance between two consecutive positions of the kernel) along axis j,
- p_j: zero padding (number of zeros concatenated at the beginning and at the end of an axis) along axis j.

Example with zero paddings

for $N = 2$, $i_1 = i_2 = 5$, $k_1 = k_2 = 3$, $s_1 = s_2 = 2$, and $p_1 = p_2 = 1$.

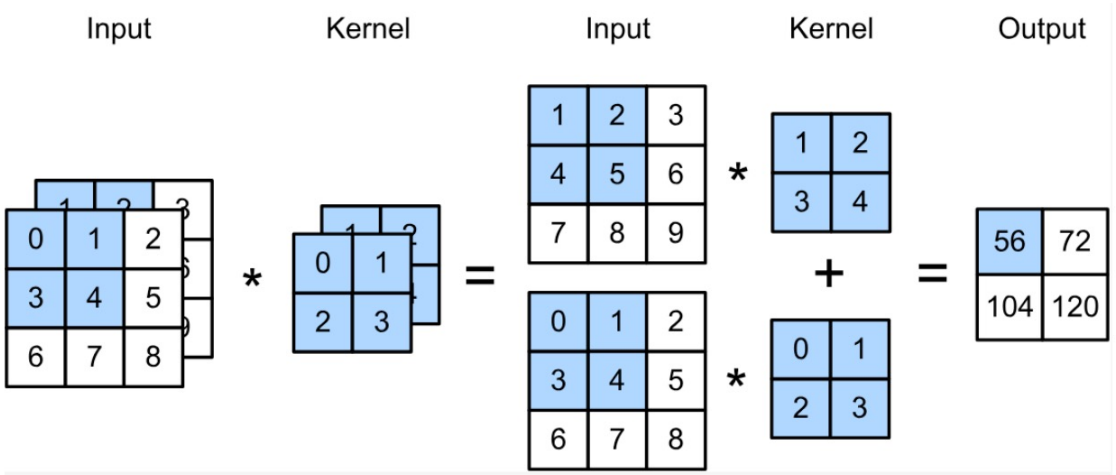


Multiple Input Channels

Red Green Blue

Multiple Input Channels

- One kernel for each channel, and then sum results over channels

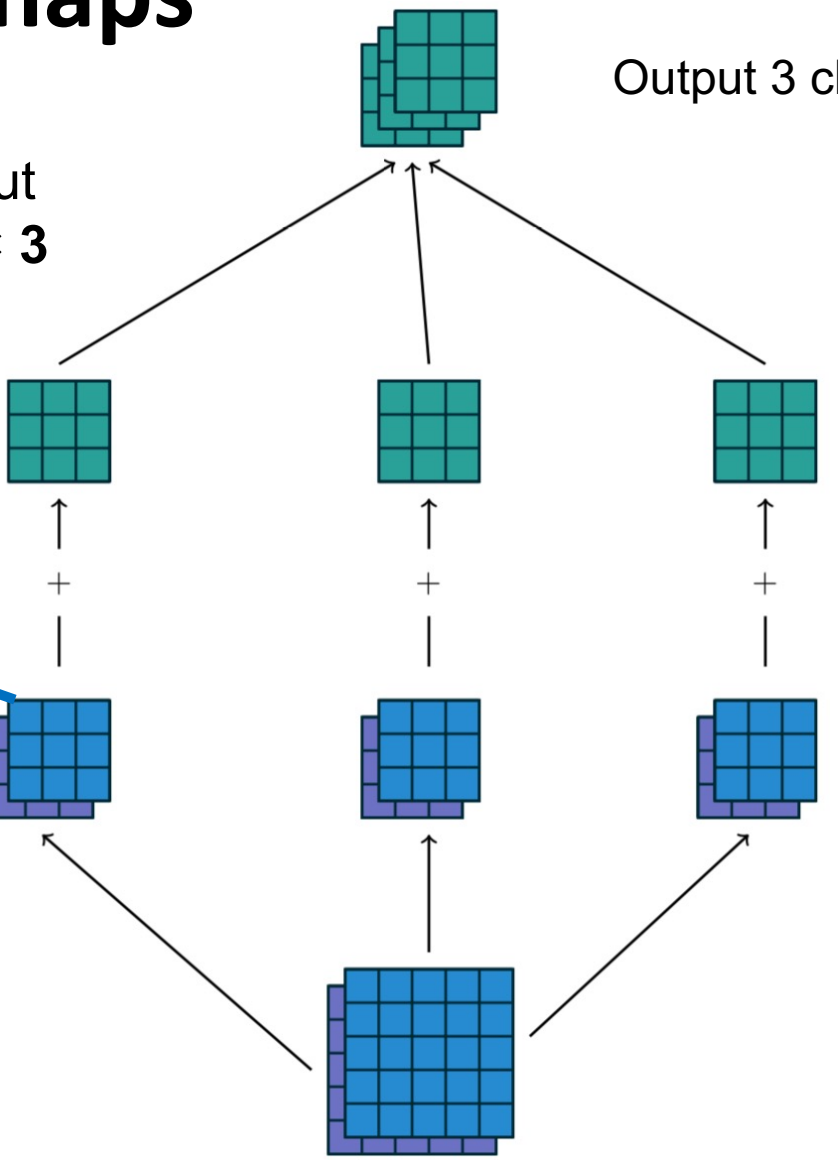


$$(1\times 1+2\times 2+4\times 3+5\times 4)$$
$$+(0\times 0+1\times 1+3\times 2+4\times 3)$$
$$=56$$

Example with three output feature maps

A convolution mapping from two input feature maps to three output feature maps using a 3 × 2 × **3 × 3** collection of kernels **w**.

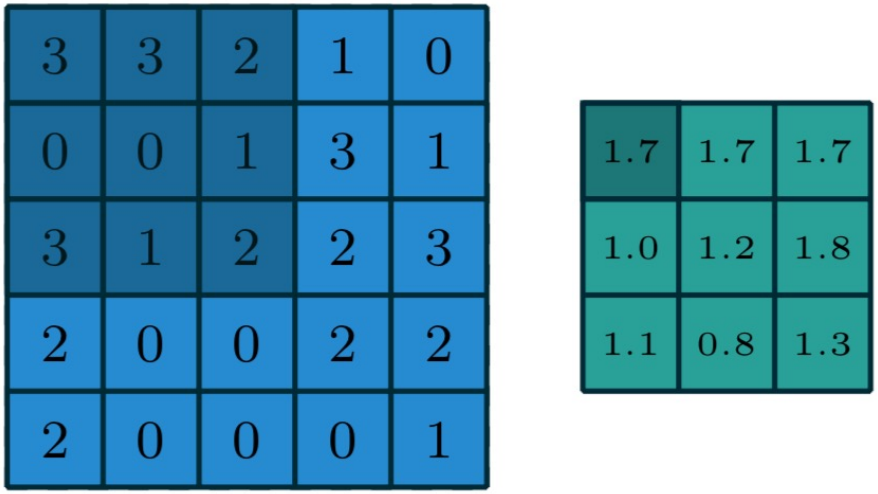
- input feature map 1 is convolved with kernel $\mathbf{w}_{1,1}$.
- input feature map 2 is convolved with kernel $\mathbf{w}_{1,2}$.



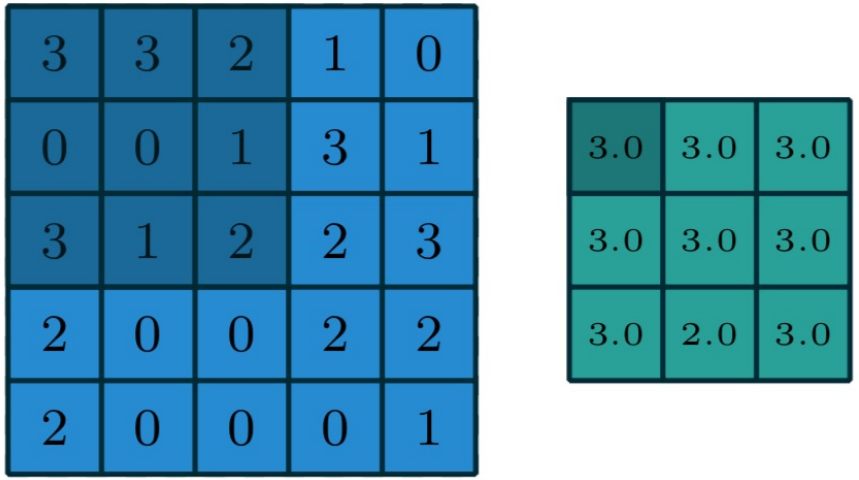
Pooling

- Pooling operations reduce the size of feature maps by using some function to summarize subregions, such as taking the **average** or the **maximum** value.

Average pooling

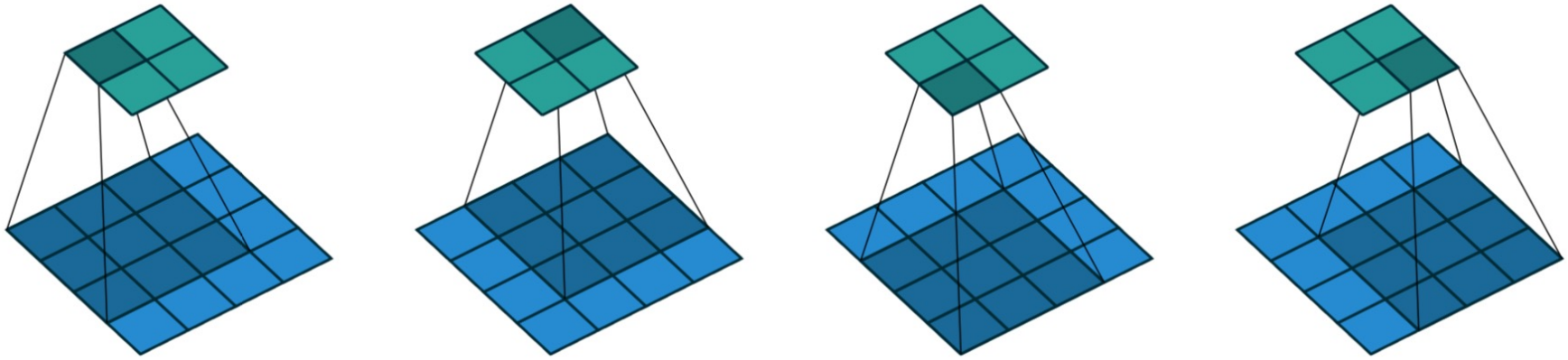


Maximum pooling



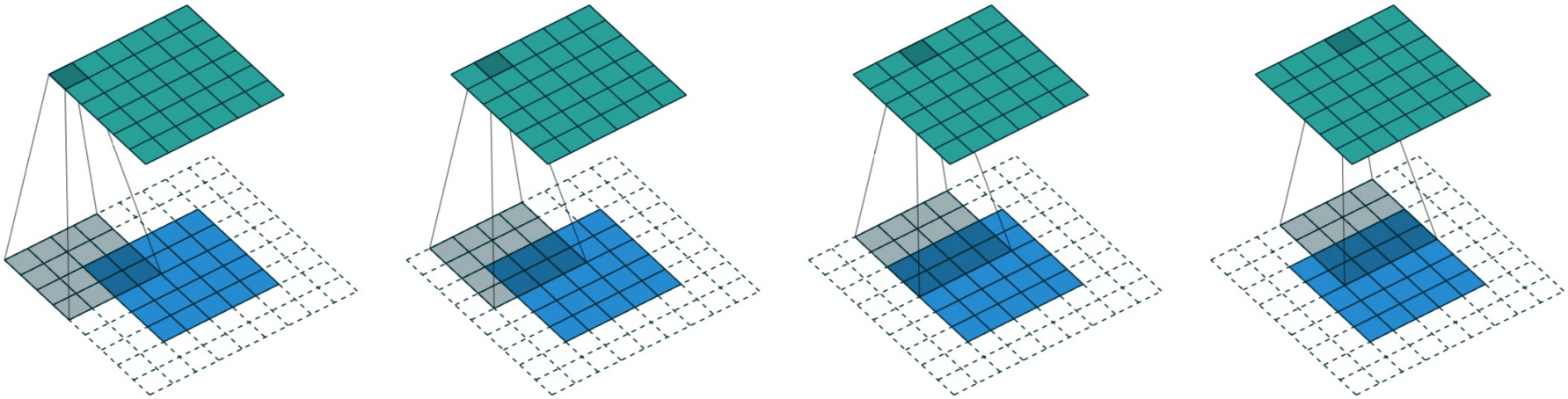
Example -- No padding, no strides

Convolving a 3 × 3 kernel over a 4 × 4 input using unit strides (i.e., i = 4, k = 3, s = 1 and p = 0).

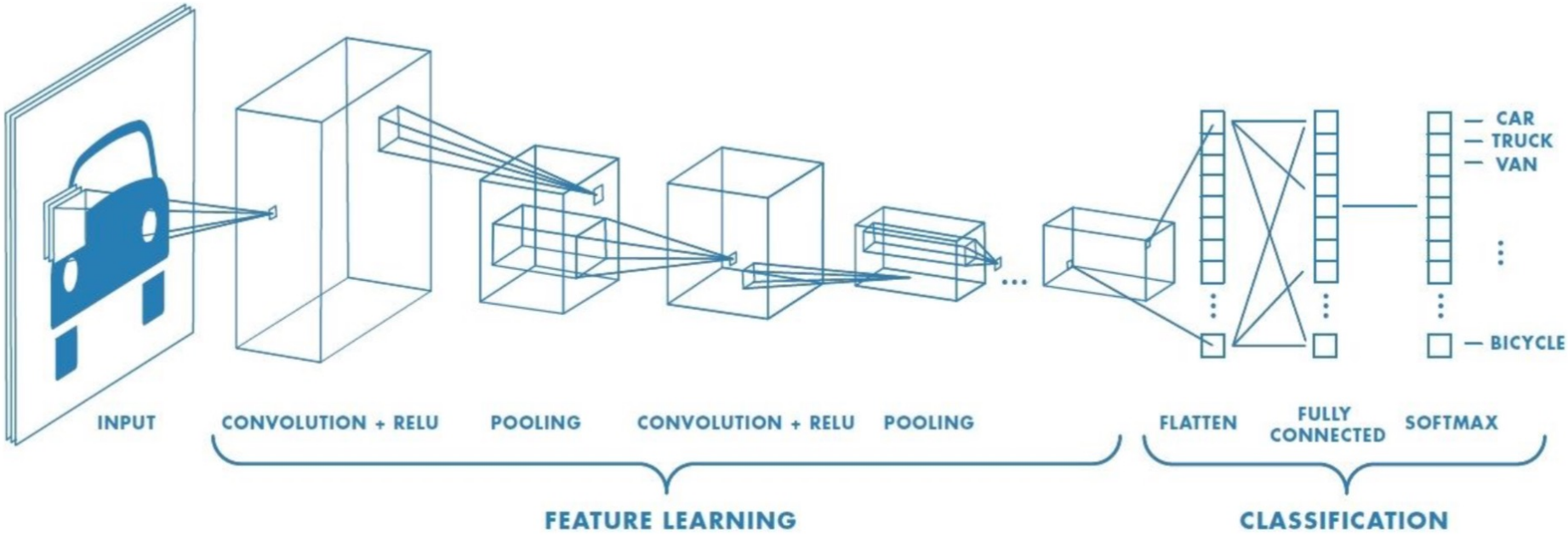


Example -- Arbitrary padding, no strides

Convolving a 4 × 4 kernel over a 5 × 5 input padded with a 2 × 2 border of zeros using unit strides (i.e., i = 5, k = 4, s = 1 and p = 2).

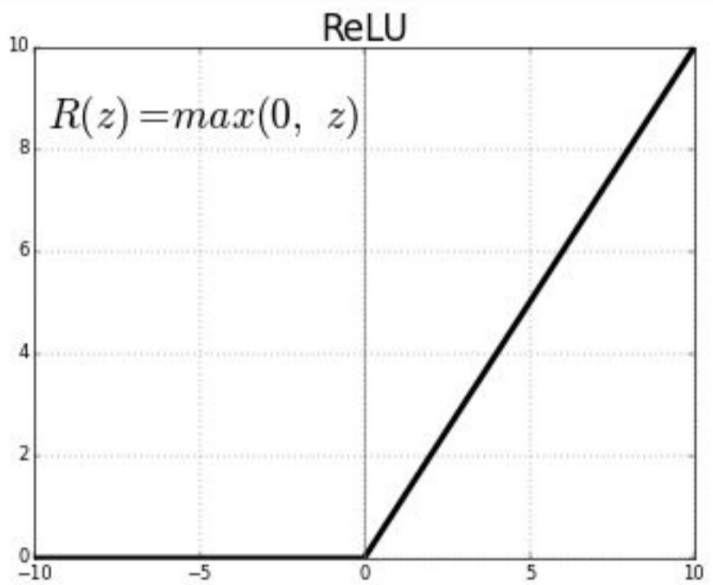


Sample architecture of CNN



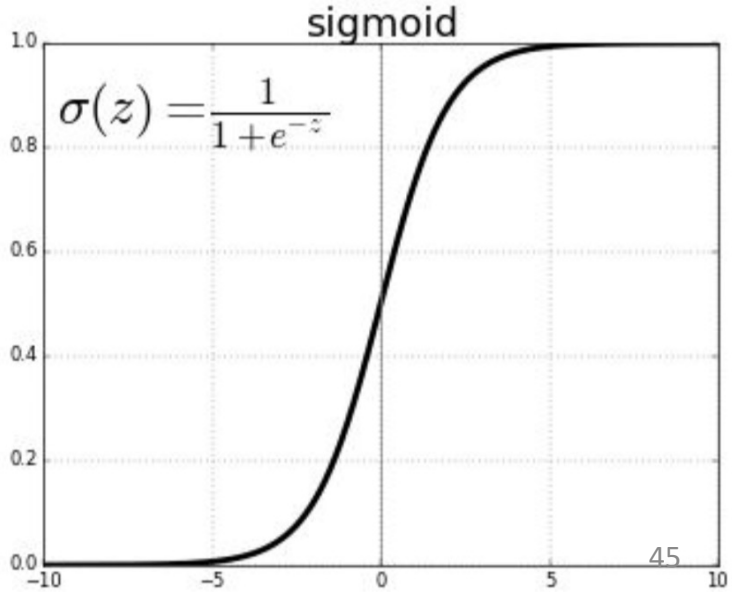
Activation layer

- Used to increase non-linearity of the network without affecting receptive fields of conv layers
- Prefer **ReLU**, results in faster training
- **LeakyReLU** addresses the vanishing gradient problem



Other types:

- Leaky ReLU,
- Randomized Leaky ReLU,
- Parameterized ReLU
- Exponential Linear Units (ELU),
- Scaled Exponential Linear Units Tanh,
- hardtanh, softtanh, softsign, softmax, softplus...



Softmax

- A special kind of activation layer, usually at the end of FC layer outputs
- Can be viewed as a fancy **normalizer** (a.k.a. Normalized exponential function)
- Produce a **discrete probability distribution** vector
- Very convenient when combined with cross-entropy loss

$$P(y = j \mid \mathbf{x}) = \frac{e^{\mathbf{x}^\mathsf{T}\mathbf{w}_j}}{\sum_{k=1}^{K} e^{\mathbf{x}^\mathsf{T}\mathbf{w}_k}}$$

- Sample vector **x**
- Weight vector **w**

Loss layer

- **L1, L2** loss
- **Cross-Entropy** loss (works well for classification, e.g., image classification)
- **Hinge** Loss
- **Huber** Loss, more resilient to outliers with smooth gradient
- **Minimum Squared Error** (works well for regression task, e.g., Behavioral Cloning)

$$J = -\sum_{i=1}^{N} y_i \log(h_\theta(x_i)) + (1 - y_i) \log(1 - h_\theta(x_i))$$

$$p(y_i|x_i) = [h_\theta(x_i)]^{(y_i)}[1 - h_\theta(x_i)]^{(1-y_i)}$$

$$p(y_i = 1|x_i) = h_\theta(x_i) \qquad p(y_i = 0|x_i) = 1 - h_\theta(x_i)$$

$$\sum_i max(0, 1 - y_i * h_\theta(x_i))$$

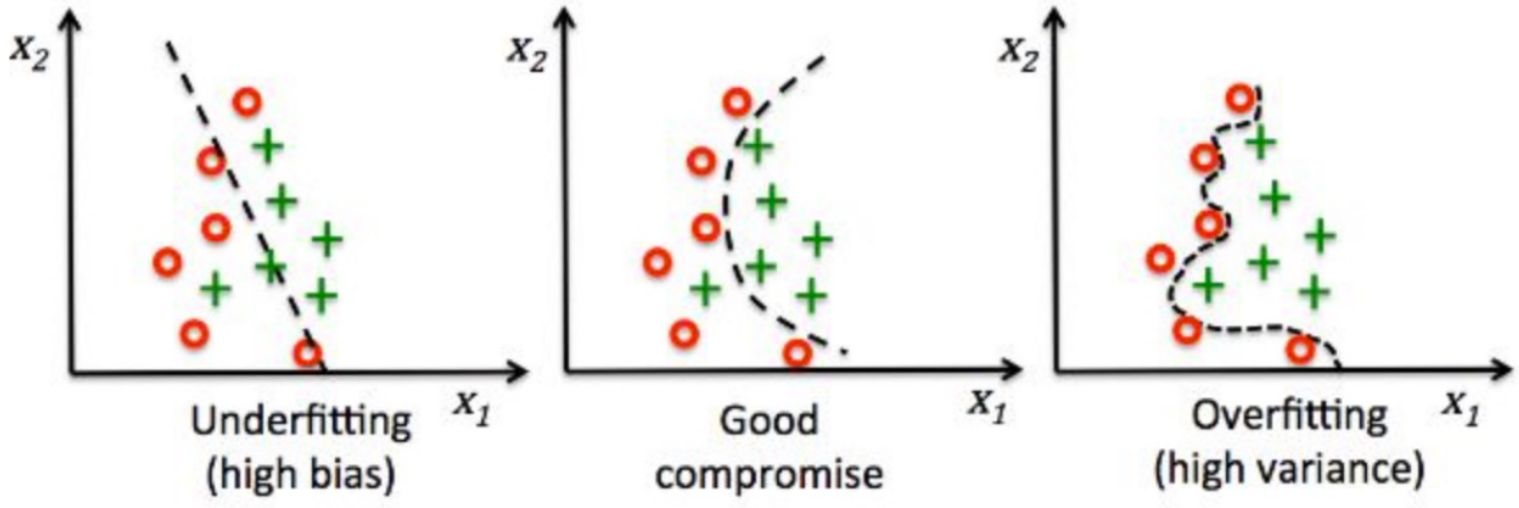
$$L_\delta(y, f(x)) = \begin{cases} \frac{1}{2}(y - f(x))^2 & \text{for} |y - f(x)| \le \delta, \\ \delta |y - f(x)| - \frac{1}{2}\delta^2 & \text{otherwise.} \end{cases}$$

$$J = \frac{1}{N} \sum_{i=1}^{N} (y_i - h_\theta(x_i))^2$$

https://rohanvarma.me/Loss-Functions/

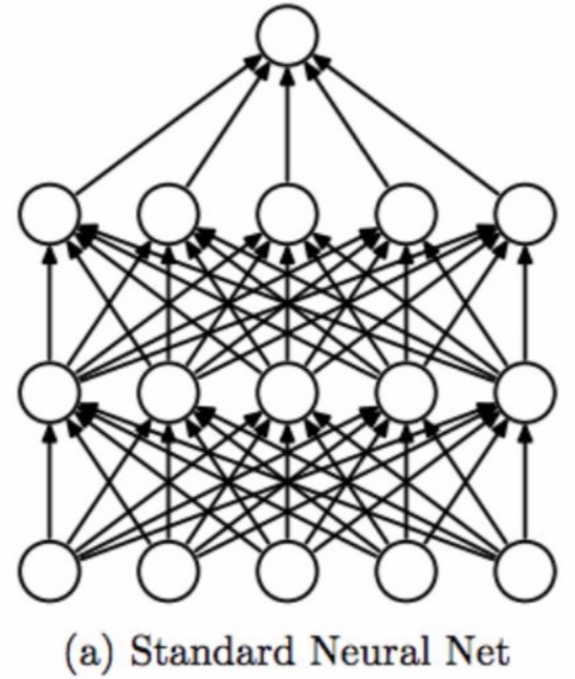
Regularization

- L1 / L2
- Dropout
- Batch norm
- Gradient clipping
- Max norm constraint

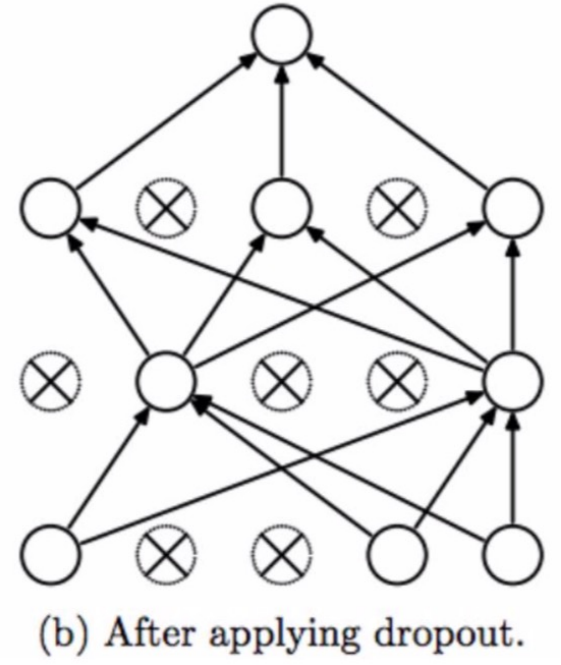


Dropout

- During training, randomly ignore activations by probability ***p***
- During testing, use all activations but scale them by ***p***
- Effectively prevent overfitting by reducing correlation between neurons



(a) Standard Neural Net



(b) After applying dropout.

Batch Normalization

- Makes networks robust to bad initialization of weights
- Usually inserted right before activation layers
- Reduce covariance shift by normalizing and scaling inputs
- The scale and shift parameters are trainable to avoid losing stability of the network

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

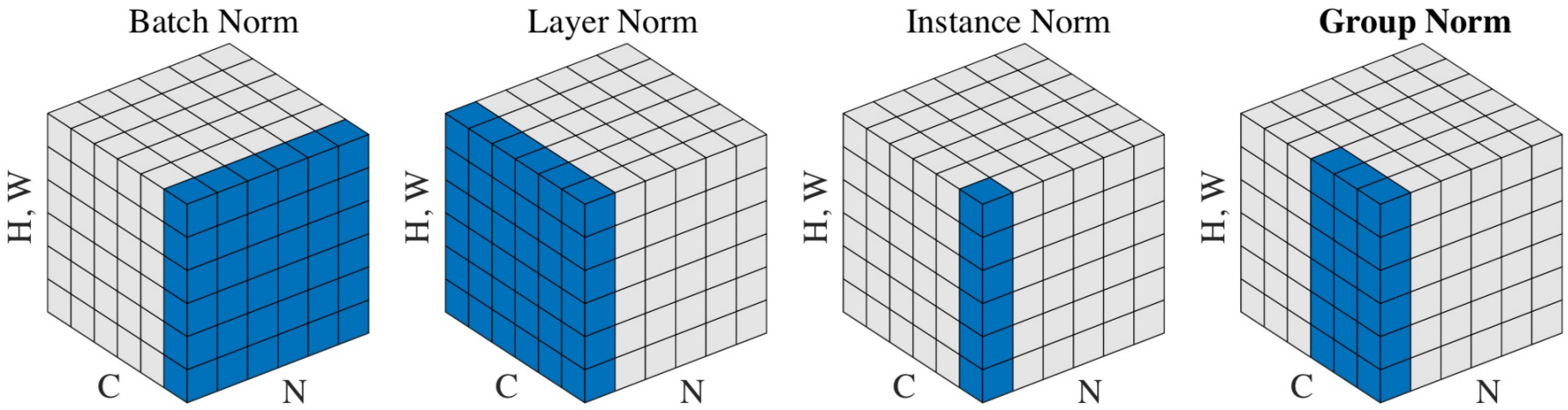
$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.



Group Normalization

Yuxin Wu Kaiming He

Facebook AI Research (FAIR)

{yuxinwu,kaiminghe}@fb.com

arXiv:1803.08494v3

https://mlexplained.com/2018/11/30/an-overview-of-normalization-methods-in-deep-learning/

Thank you!