Learning (III)

Mingsheng Long

Tsinghua University

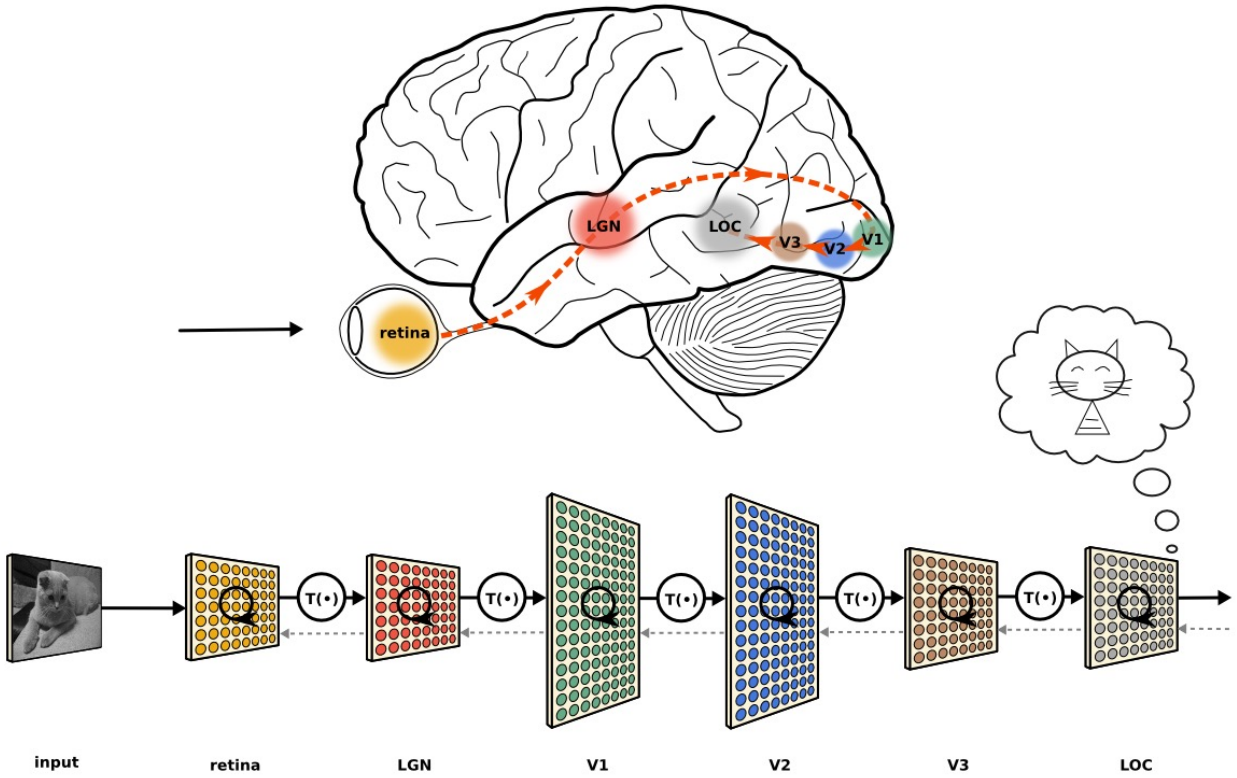
Spring 2023

Outline

- **Deep Learning**
- Multiplayer Perceptrons (MLP)
  - Backpropagation
  - Training Strategies
- Convolutional Neural Network (CNN)
  - Training Strategies
  - Standard Architectures

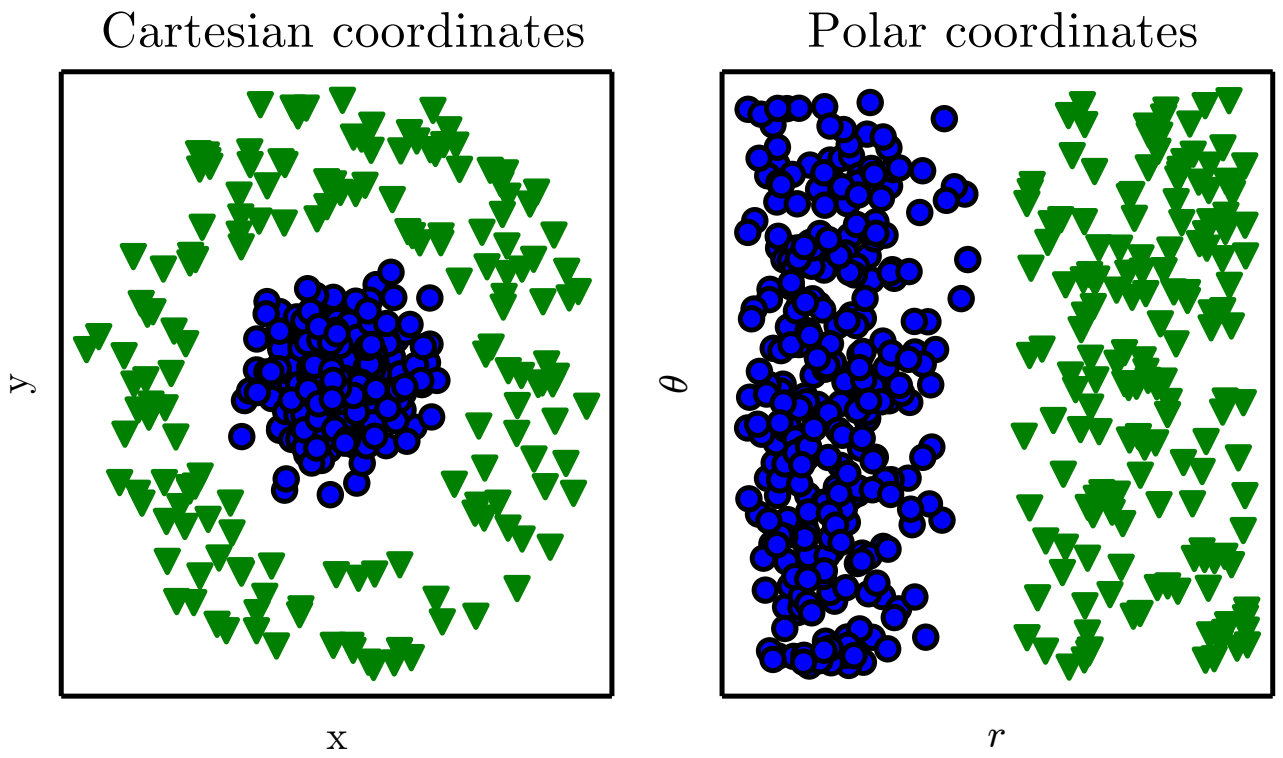
Why Deep Learning

- Deep learning is algorithms that model high-level abstractions in data using architectures consisting of **multiple nonlinear transformations**.



Features Really Matter

How many ways to turn linearly inseparable case into a separable one?



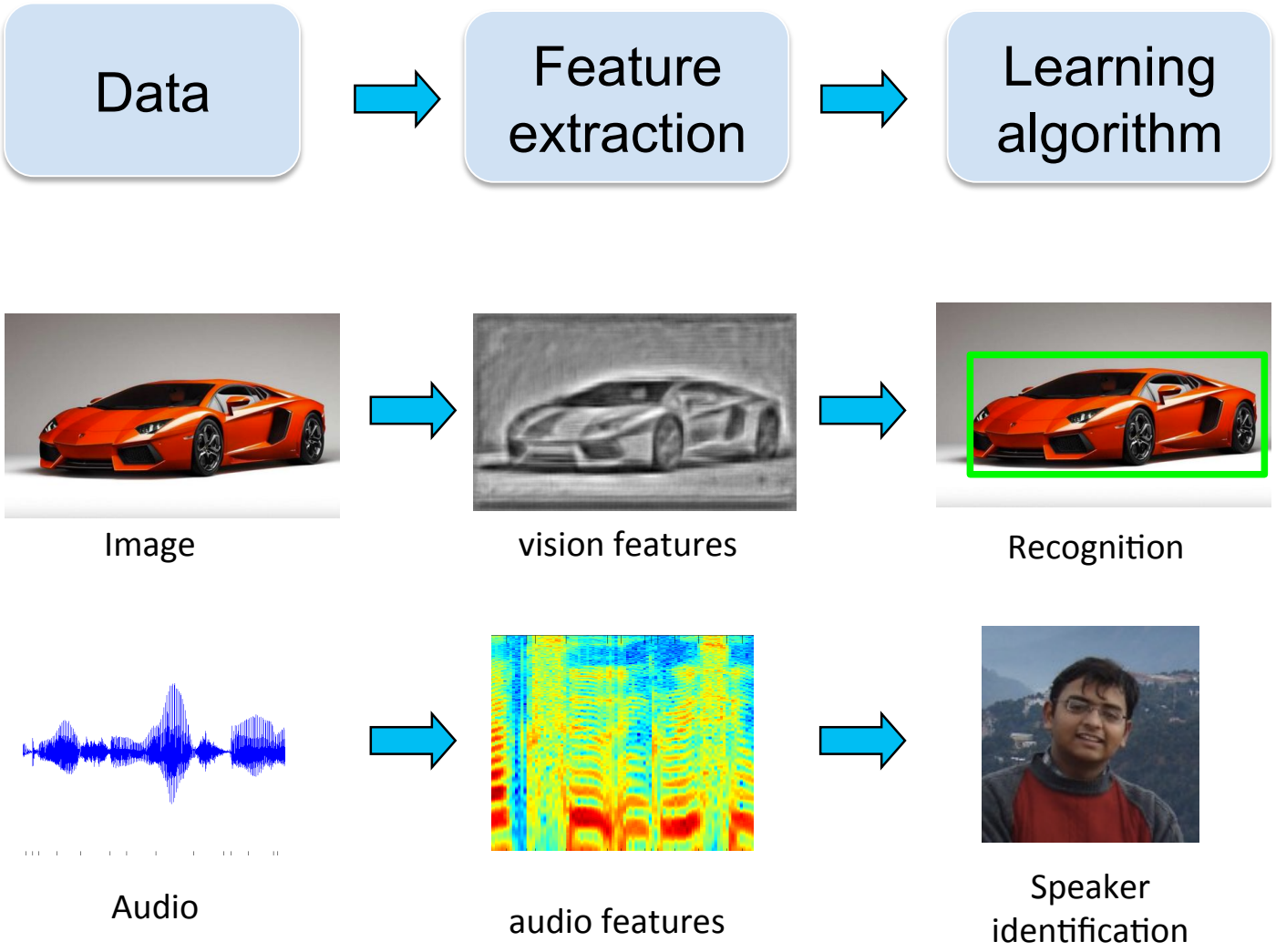
$$x = r\cos(\theta)$$
$$y = r\sin(\theta)$$

How about:

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1$$

Jitendra Malik, et al. R-CNN: Regions with Convolutional Neural Network Features. CVPR 2014

Engineered Features



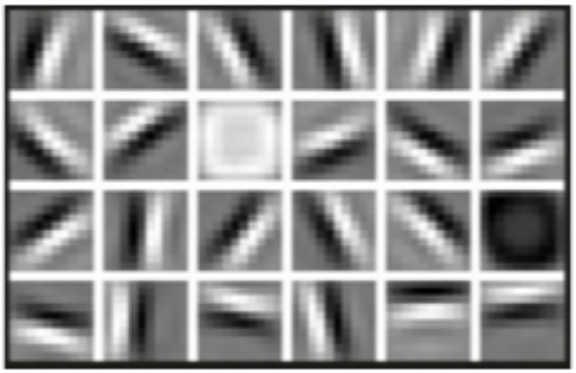
Learned Features

Hand engineered features are time consuming, brittle and not scalable.

(有多少智能，就有多少人工)

Can we learn the underlying features directly from data?

Low Level Features



Lines & Edges

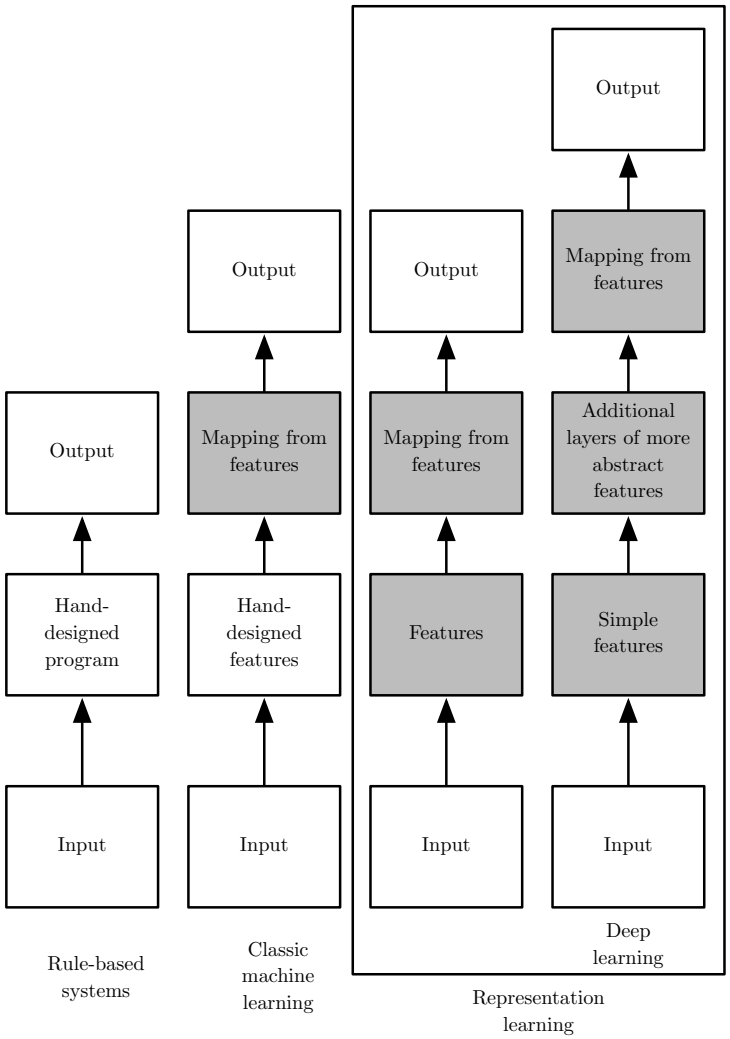
Mid Level Features

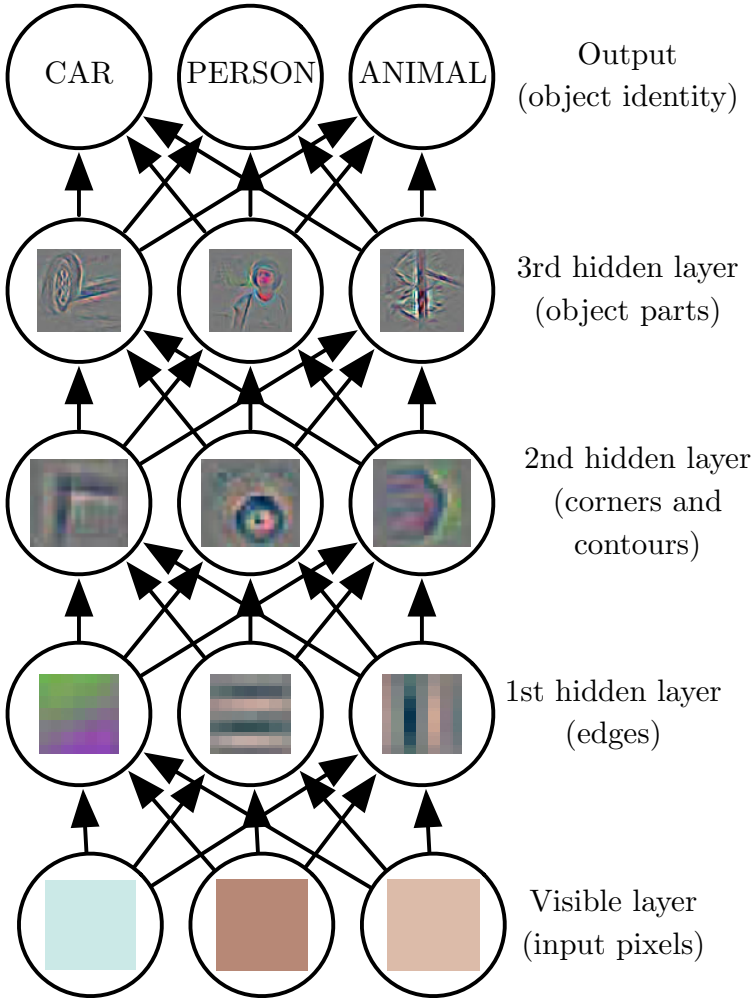
Eyes & Nose & Ears

High Level Features

Facial Structure

Feature Learning with Abstraction

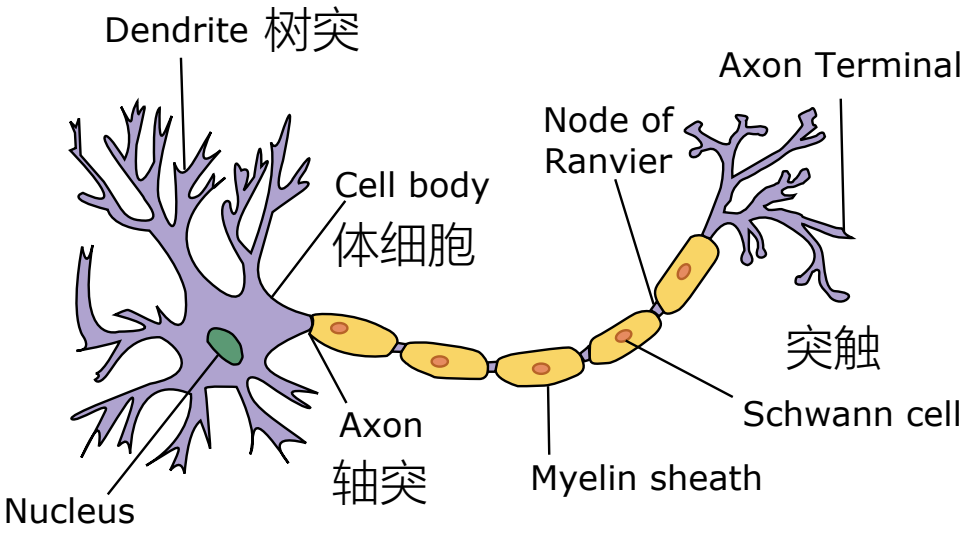


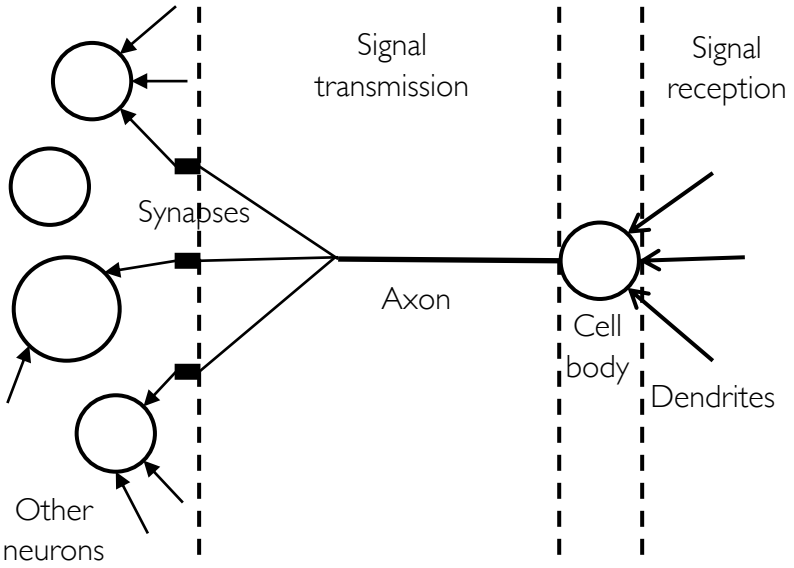


Outline

- Deep Learning
- **Multiplayer Perceptrons (MLP)**
  - Backpropagation
  - Training Strategies
- Convolutional Neural Network (CNN)
  - Training Strategies
  - Standard Architectures

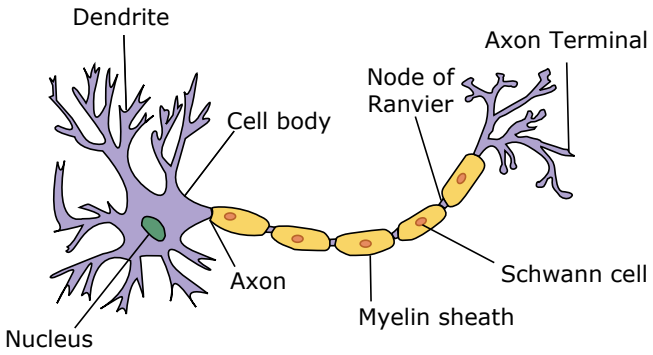
Brain and Neuron





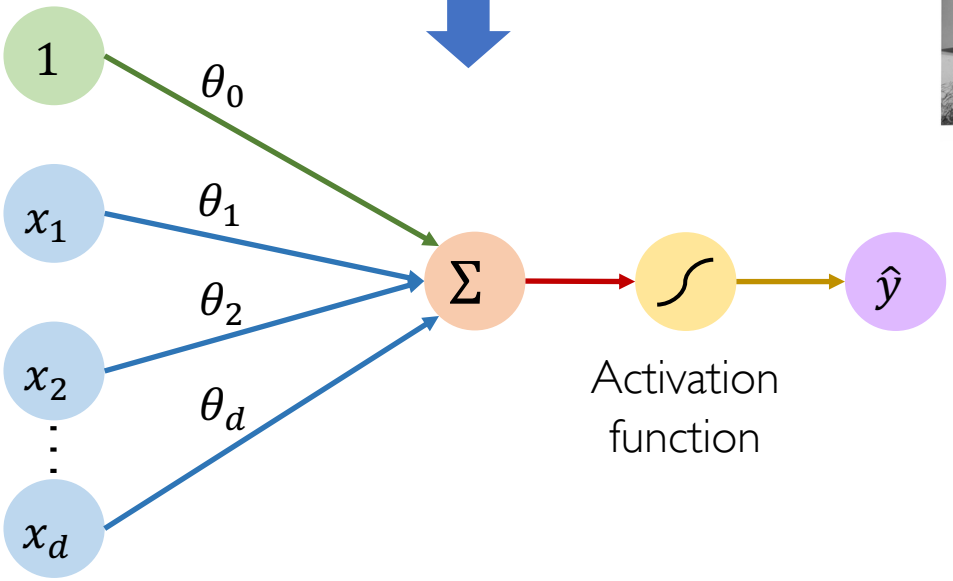
- We estimate around 10^{11} the number of neurons in the human brain:
  - They receive information from other neurons through their **dendrites** (树突)
  - They process the information in their cell body (**soma**, 体细胞)
  - They send information through a "cable" called an **axon** (轴突)
  - The point of connection between the axon and other neurons' dendrites are called **synapses** (突触)

Perceptron



Rosenblatt 1958
An psychologist ☺

Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms



Linear combination of inputs

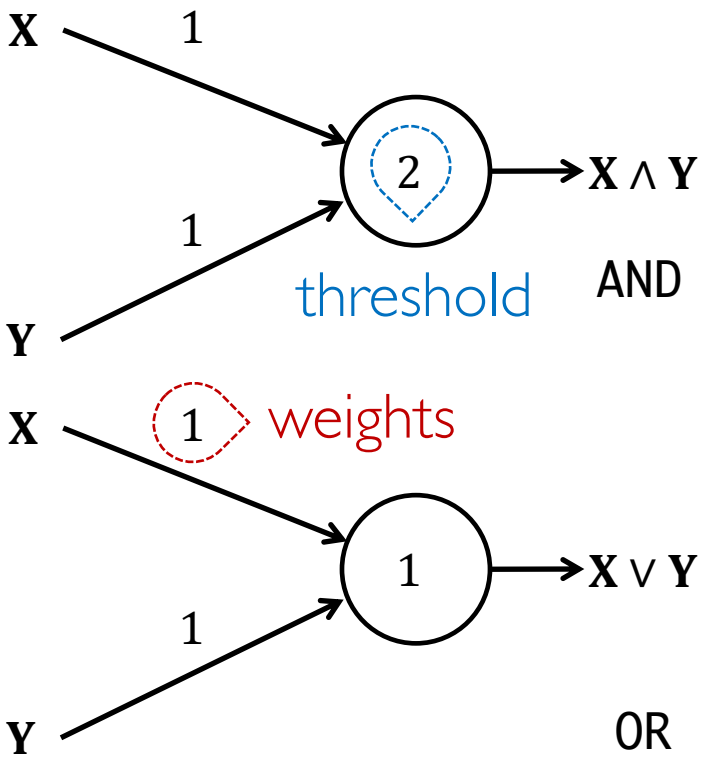
Output

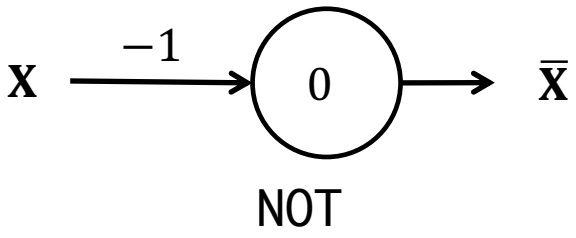
$$\hat{y} = g\left(\theta_0 + \sum_{i=1}^{d} x_i \theta_i\right)$$

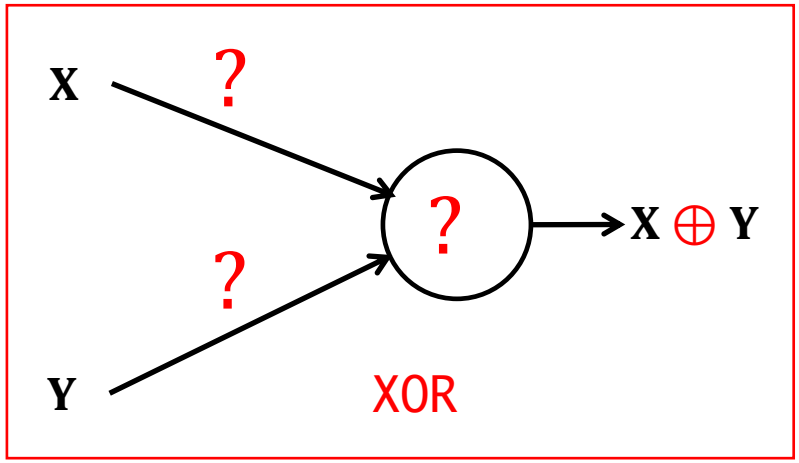
Non-Linearity Activation function

Bias

Perceptron: Boolean Functions

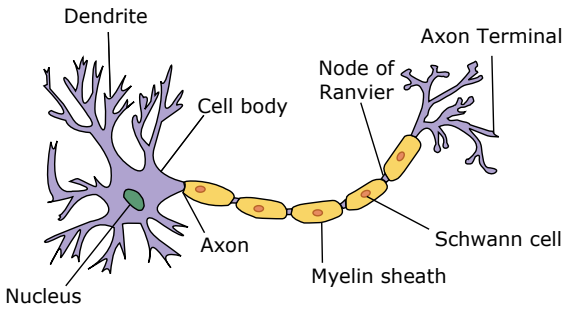




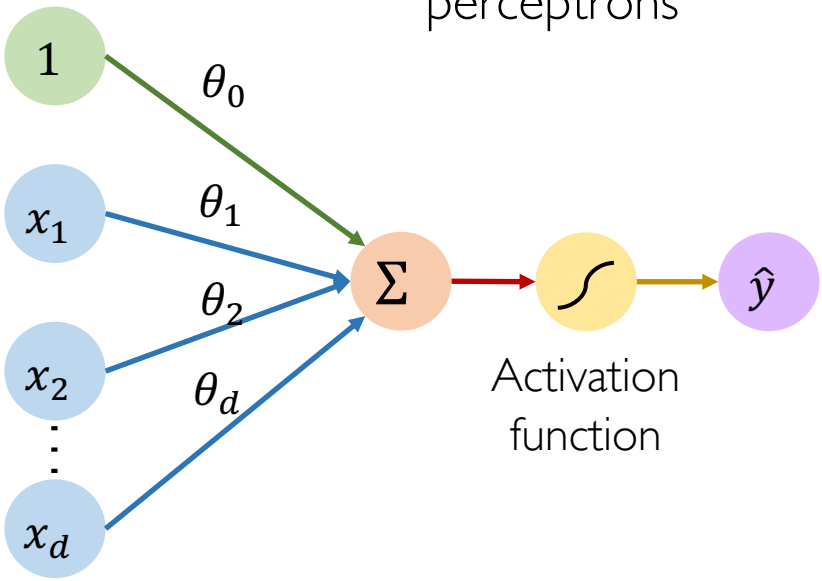


- **Rosenblatt:** Perceptron can represent any Boolean logics 😀
- **Minsky:** Wait, just want to know how to represent **XOR**? 🤔

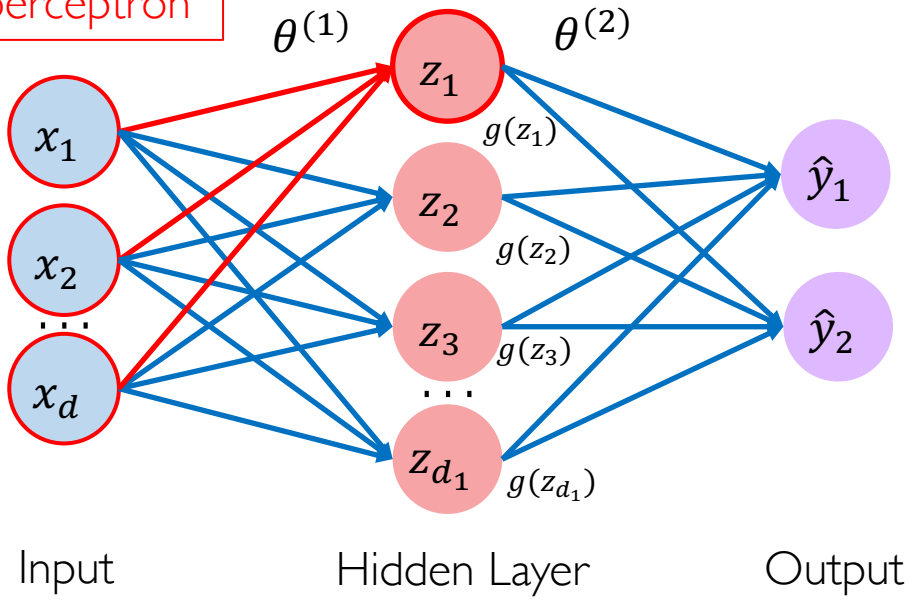
Multilayer Perceptrons (MLP)



MLP is composition of perceptrons

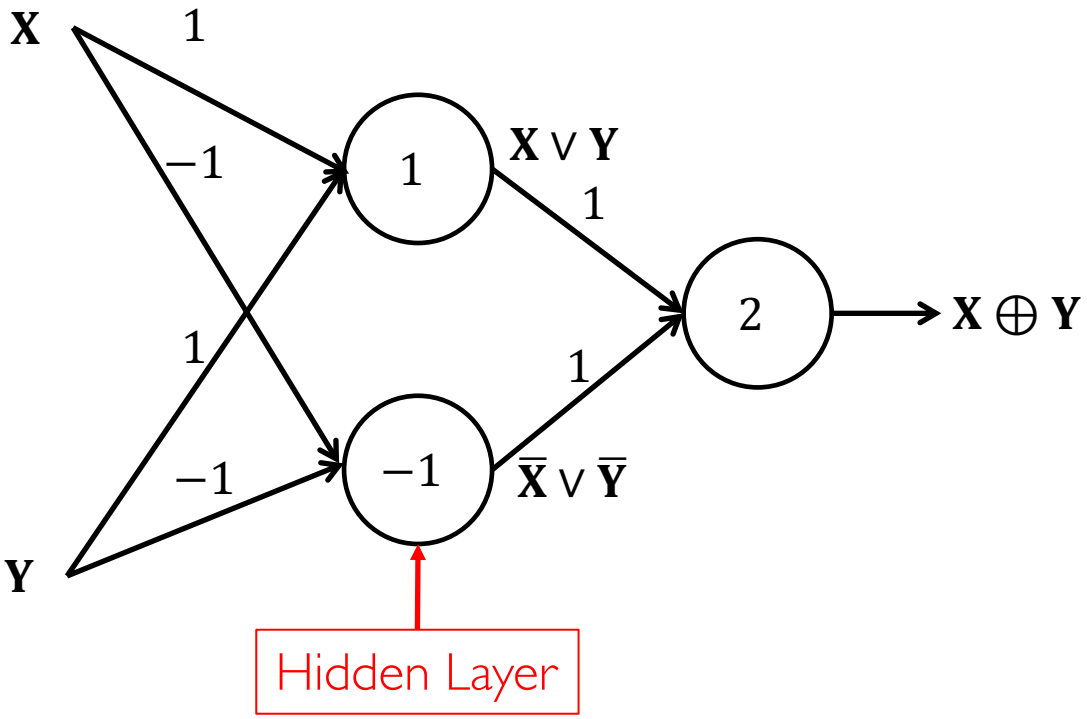


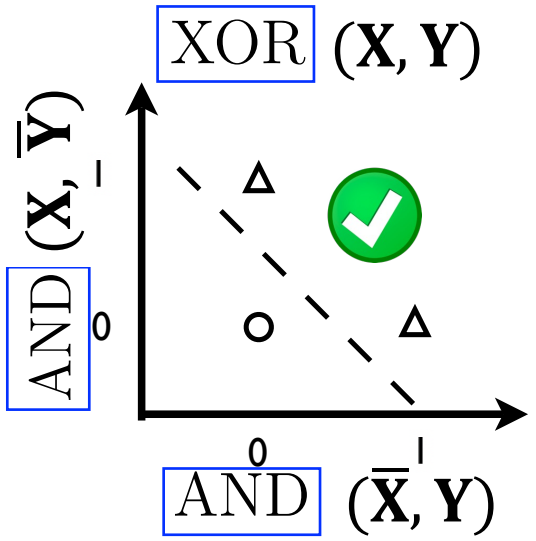
A single perceptron



MLP for XOR

$$X \oplus Y = (X \vee Y)\&(\bar{X} \vee \bar{Y})$$



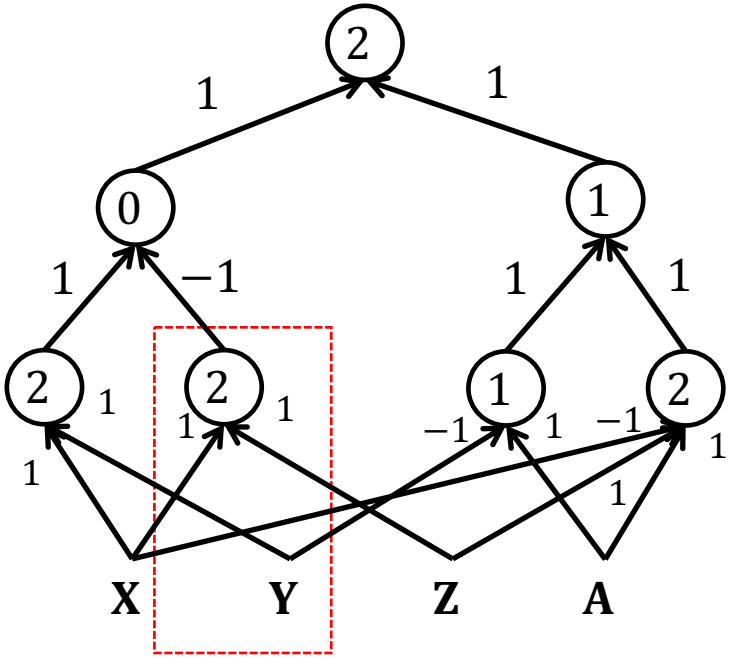


Two-layer Perceptrons

Marvin Minsky and Seymour Papert. Perceptrons. An Introduction to Computational Geometry. 1969.

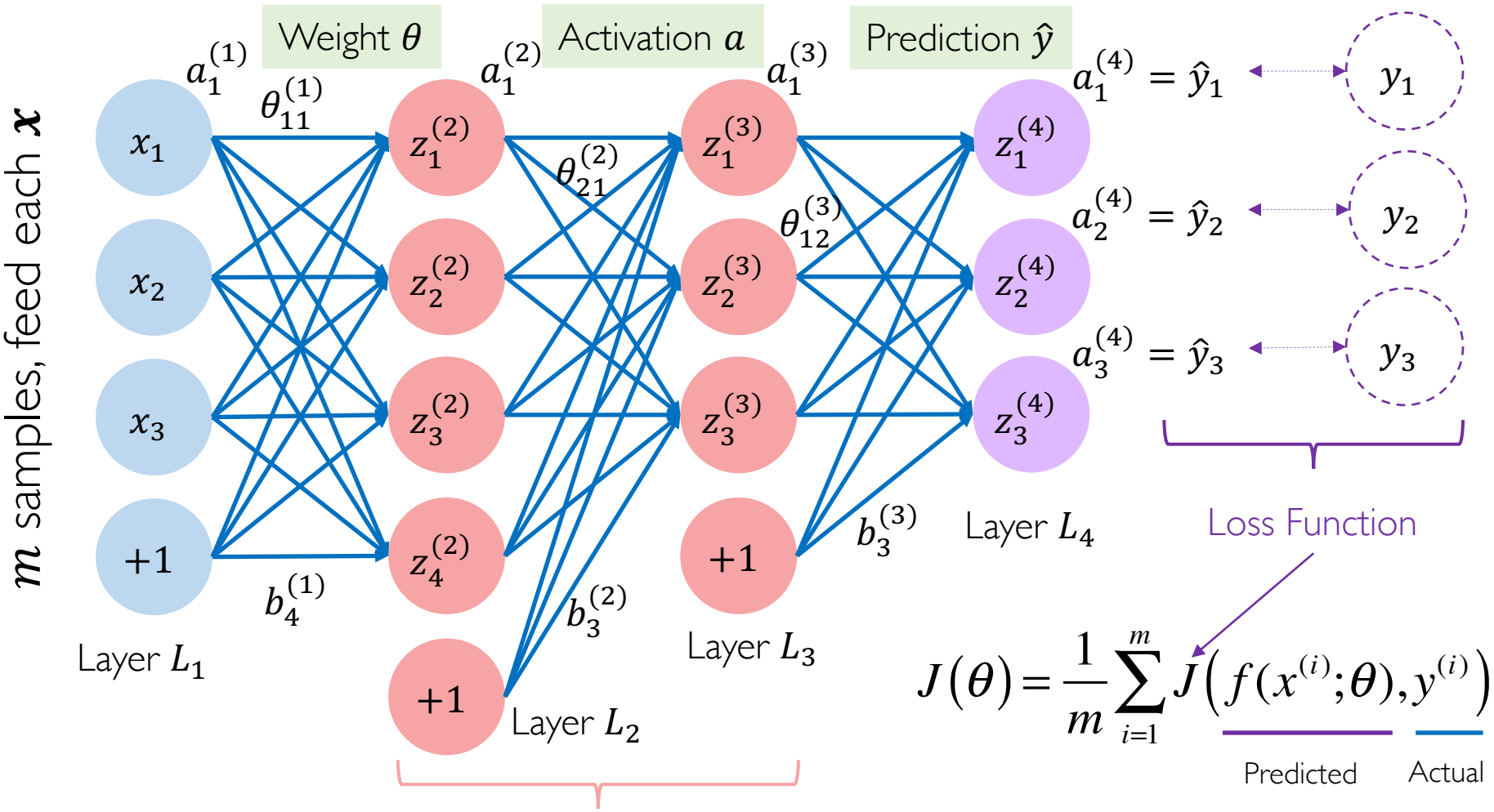
MLP: Boolean Functions

$$((A\&\bar{X}\&Z)|(A\&\bar{Y}))\&\left((X\&Y)|\overline{(X\&Z)}\right)$$



- MLP is universal to represent arbitrarily complex Boolean functions
- The connections in MLP can be sparse - a phenomenon in the Brain

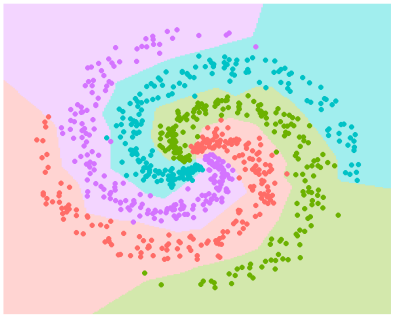
Multilayer Perceptrons (MLP)



$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} J\left(f(x^{(i)};\theta), y^{(i)}\right)$$

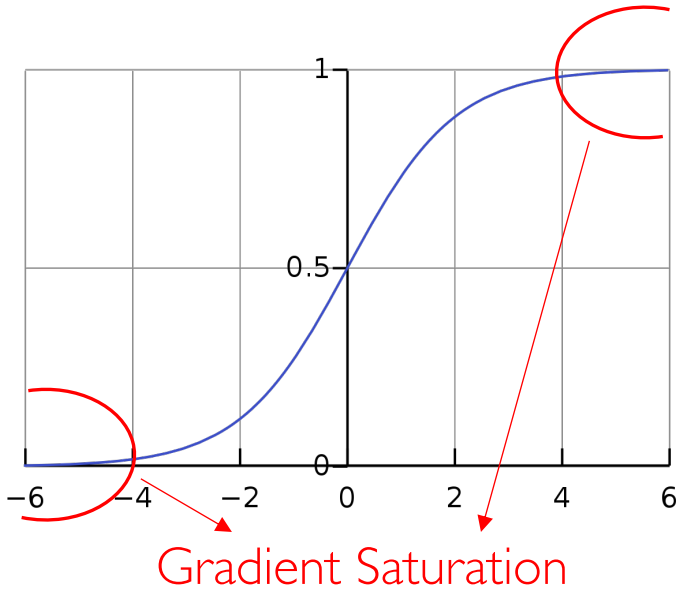
Activation Functions

Non-linearity through activation functions $\hat{y} = g(\theta_0 + \boldsymbol{X}^{\mathrm{T}}\boldsymbol{\theta})$



sigmoid function

$$g(z) = \sigma(z) = \frac{1}{1 + e^{-z}}$$

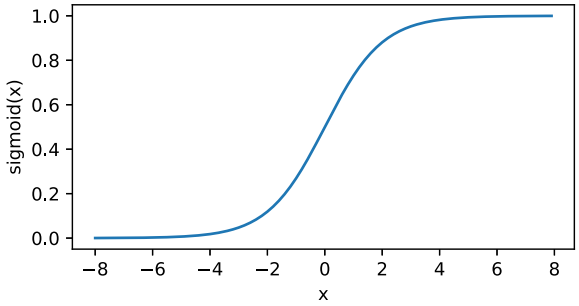


Inputs Weights Sum Non-Linearity Output

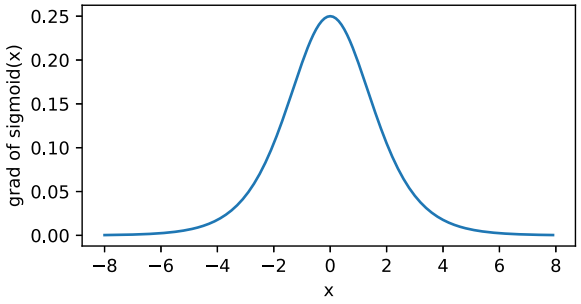
Activation Functions

Sigmoid

$$g(z) = \frac{1}{1 + e^{-z}}$$



$$g'(z) = g(z)\big(1 - g(z)\big)$$

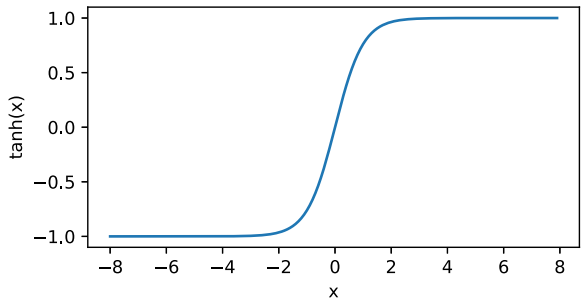


Range in (0, 1), probability

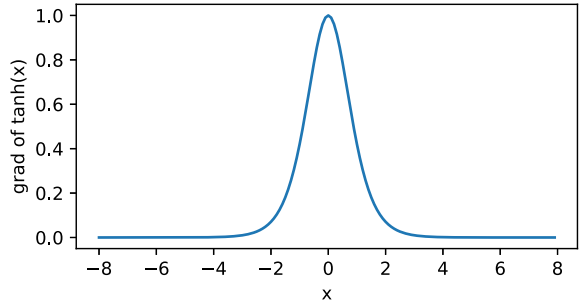
Relative smaller gradient

Hyperbolic Tangent (tanh)

$$g(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$



$$g'(z) = 1 - g(z)^2$$

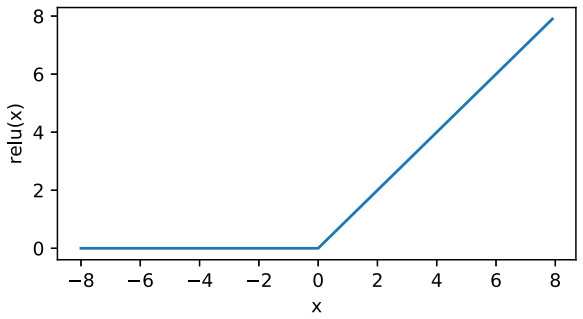


Zero-centered

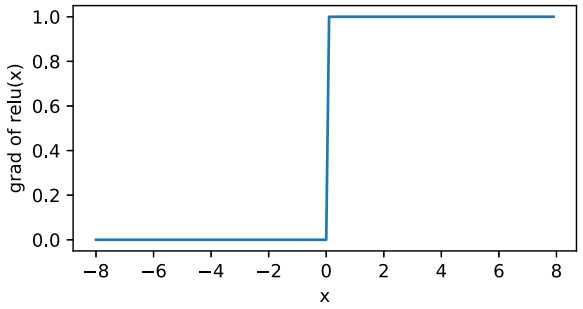
Relatively larger saturation region

Rectified Linear Unit (ReLU)

$$g(z) = \max(0, z)$$



$$g'(z) = \mathrm{sgn}(z)$$



Time-efficient and faster convergence, but neurons are prone to death.

Activation Functions

- Softmax function

$$g(\boldsymbol{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}}$$

Multiple output
Categorical distribution

rooster

cat

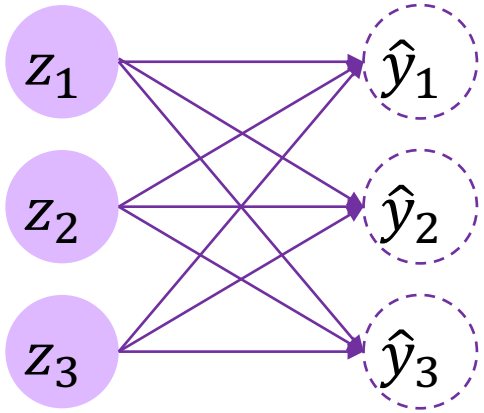
dog

donkey

  - Winner takes all
  - The biggest one dominates the outputs

- Exponent arithmetic
  - May cause numerical overflow
  - For numerical stability, implementation is

$$g(\boldsymbol{z})_i = \frac{e^{z_i - z_m}}{\sum_{j=1}^{k} e^{z_j - z_m}}, m = \text{argmax}(z_j)$$



$z_j \in (-\infty, +\infty)$ $\hat{y}_j \in [0,1], \sum_{j=1}^{k} \hat{y}_j = 1$

Loss Functions

- For each data point $\boldsymbol{x}^{(i)} \in \mathbb{R}^d$
  - Its prediction $\hat{\boldsymbol{y}} = \boldsymbol{a}^{(n_l)} = f_\theta\left(\boldsymbol{x}^{(i)}\right) \in \mathbb{R}^k$
- Its ground truth label $y^{(i)} \in \{1, \ldots, k\}$
  - One-hot coding $\boldsymbol{y}$: $y_j = 1$ if $y^{(i)} = j$ and $y_j = 0$ otherwise
- How good is $\hat{\boldsymbol{y}}$ to fit $\boldsymbol{y}$? (goodness-of-fit)
- Entropy $H(q) = -\sum_{j=1}^{k} q_j \log q_j$ $\quad \boxed{H(q,p) = \mathrm{KL}(q||p) + H(q)}$
  - Amount of bits to encode information (uncertainty) in q
- Relative-entropy $\mathrm{KL}(q||p) = -\sum_{j=1}^{k} q_j \log p_j - H(q)$
  - Amount of extra bits to encode information in p given q
- Cross-entropy $H(q,p) = -\sum_{j=1}^{k} q_j \log p_j$

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

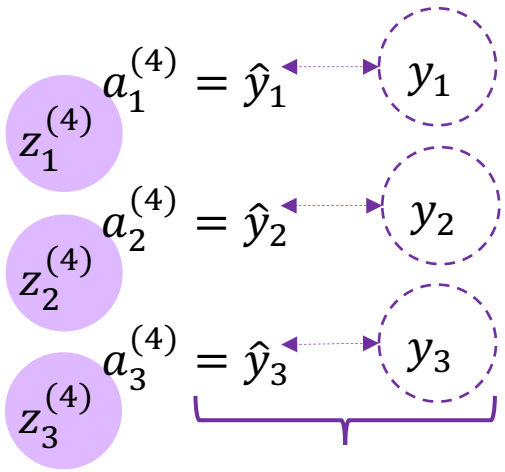
Cost Function

- Softmax function in the output layer

$$\hat{\boldsymbol{y}} = \boldsymbol{a}^{(n_l)} = f_\theta(\boldsymbol{x}^{(i)}) = \begin{bmatrix} p(y^{(i)} = 1 | \boldsymbol{x}^{(i)}; \boldsymbol{\theta}) \\ p(y^{(i)} = 2 | \boldsymbol{x}^{(i)}; \boldsymbol{\theta}) \\ \vdots \\ p(y^{(i)} = k | \boldsymbol{x}^{(i)}; \boldsymbol{\theta}) \end{bmatrix} = \frac{1}{\sum_{j=1}^{k} \exp(z_j^{(n_l)})} \begin{bmatrix} \exp(z_1^{(n_l)}) \\ \exp(z_2^{(n_l)}) \\ \vdots \\ \exp(z_k^{(n_l)}) \end{bmatrix}$$

- Cross-entropy $J(q, p) = -\sum_{j=1}^{k} q_j \log p_j$
  - loss $J(\boldsymbol{y}, \hat{\boldsymbol{y}}) = -\sum_{j=1}^{k} y_j \log \hat{y}_j$
  - $\boldsymbol{y}$ follows one-hot coding
  - $y_j = 1$ if $y^{(i)} = j$ and $y_j = 0$ otherwise
- Cost function

$$\min J(\theta) = -\frac{1}{m} \sum_{i=1}^{m} \left[\sum_{j=1}^{k} \mathbf{1}\{y^{(i)} = j\} \log \frac{\exp(z_j^{(n_l)})}{\sum_{j'=1}^{k} \exp(z_{j'}^{(n_l)})} \right]$$



Outline

- Deep Learning
- Multiplayer Perceptrons (MLP)
  - **Backpropagation**
  - Training Strategies
- Convolutional Neural Network (CNN)
  - Training Strategies
  - Standard Architectures

Gradient-Based Training

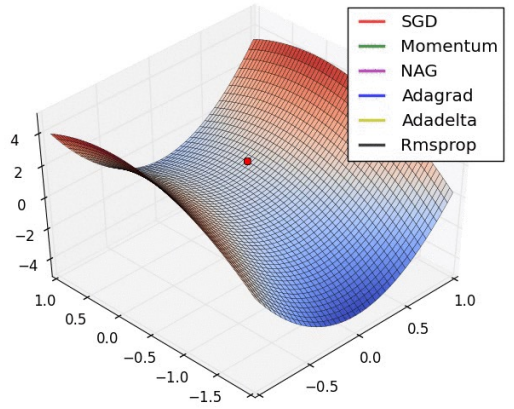
$$\arg\min_{\theta} O(\mathcal{D};\theta) = \sum_{i=1}^{m} L\left(y_i, f\left(x_i\right);\theta\right) + \Omega(\theta)$$

Structural Risk Minimization | Model | Data | Hypothesis | Parameters

- Iterative Algorithm (Convergence Guarantee)

```
for (t = 1 to T) {
    1. ForwardPropagation()
    2. BackwardPropagation()
    3. θ^(t+1) ← θ^(t) − η∇_θ O(D; θ^(t))
}
```

Parameter Updates



Gradient Descent (GD)

$$J(\theta, b) = \left[\frac{1}{m}\sum_{i=1}^{m} J\left(\theta, b; x^{(i)}, y^{(i)}\right)\right]$$

- Gradient descent

$$- \theta_{ij}^{(l)} = \theta_{ij}^{(l)} - \eta \frac{\partial}{\partial \theta_{ij}^{(l)}} J(\theta, b)$$

$$- b_i^{(l)} = b_i^{(l)} - \eta \frac{\partial}{\partial b_i^{(l)}} J(\theta, b)$$

$$J\left(\theta, b; x^{(i)}, y^{(i)}\right) = -\sum_{j=1}^{k} \mathbf{1}\{y^{(i)} = j\} \log \frac{\exp(z_j^{(n_l)})}{\sum_{j'=1}^{k} \exp(z_{j'}^{(n_l)})}$$

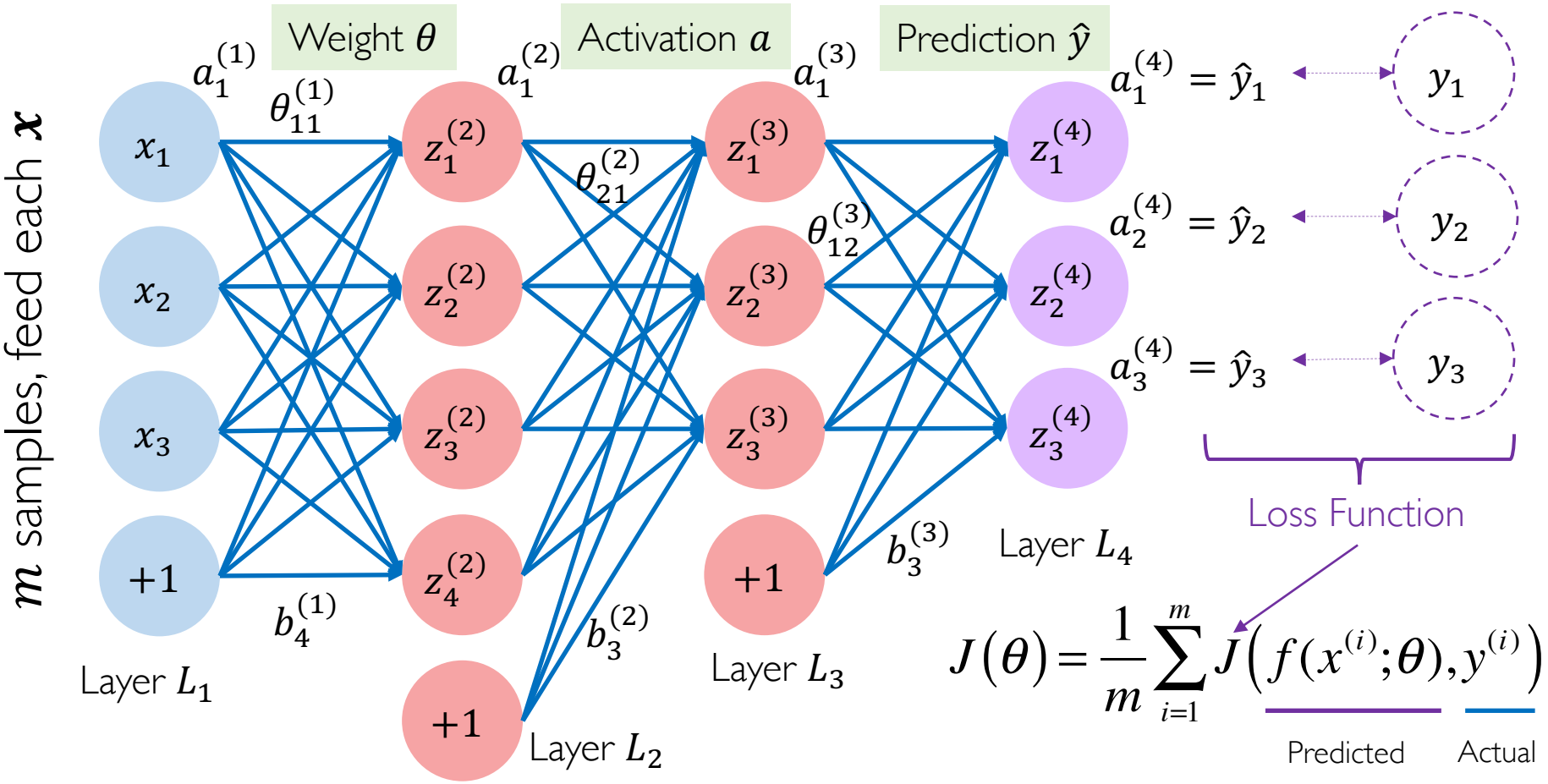
$$\frac{\partial J(\theta, b)}{\partial z_j^{(n_l)}} = -\left(\mathbf{1}\{y^{(i)} = j\} - p\left(y^{(i)} = j | x^{(i)}; \theta\right)\right)$$

How to compute the derivatives for parameters in hidden layers?

$$- \frac{\partial}{\partial \theta_{ij}^{(l)}} J(\theta, b) = \frac{1}{m}\sum_{i=1}^{m} \frac{\partial}{\partial \theta_{ij}^{(l)}} J\left(\theta, b; x^{(i)}, y^{(i)}\right)$$

$$- \frac{\partial}{\partial b_i^{(l)}} J(\theta, b) = \frac{1}{m}\sum_{i=1}^{m} \frac{\partial}{\partial b_i^{(l)}} J\left(\theta, b; x^{(i)}, y^{(i)}\right)$$

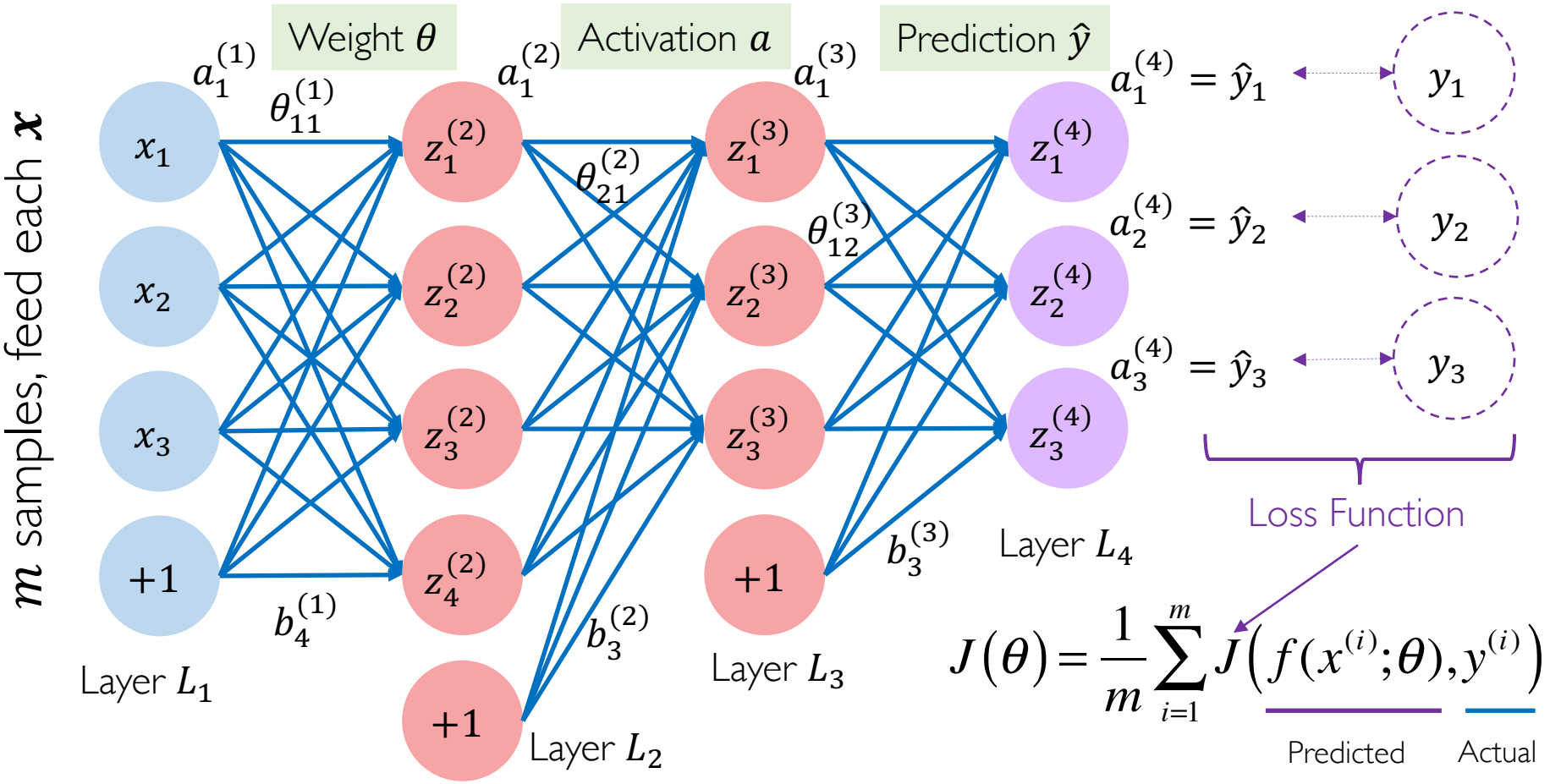
Step 1: Forward Propagation



$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} J\left(f(x^{(i)};\theta), y^{(i)}\right)$$

- Forward propagate all $(x, y)'s$ to compute activations and objective $J(\theta, b)$

Step 2: Backward Propagation

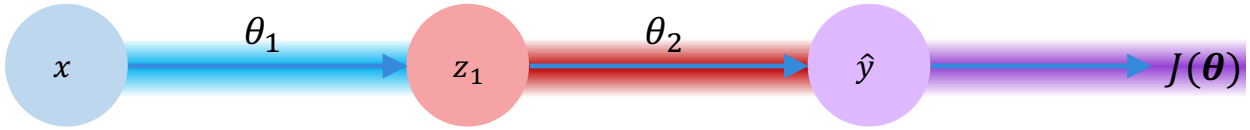


- Backward propagate the gradients $\nabla J(\theta, b)$ to update parameters in all layers

Backpropagation (BP) Algorithm

- The backpropagation algorithm applies the chain rule of calculus to the parameters $\boldsymbol{\theta}$ of each layer

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



$$\nabla_{\boldsymbol{x}} z = \left(\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}}\right)^{\top} \nabla_{\boldsymbol{y}} z,$$

Jacobian matrix

$$\frac{\partial J(\boldsymbol{\theta})}{\partial \theta_1} = \frac{\partial J(\boldsymbol{\theta})}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial z_1} * \frac{\partial z_1}{\partial \theta_1}$$

- Back propagation is an efficient implementation of the chain rule
  - Uses **dynamic programming** (table filling)
  - Avoids re-computing repeated subexpressions (in dependency)
  - Speed vs memory tradeoff (sensitive to #samples $\boldsymbol{m}$)

Step 3: Parameter Updates

$$J(\theta, b) = \left[\frac{1}{m}\sum_{i=1}^{m} J\left(\theta, b; x^{(i)}, y^{(i)}\right)\right]$$

- Gradient descent

  - $\theta_{ij}^{(l)} = \theta_{ij}^{(l)} - \eta \frac{\partial}{\partial \theta_{ij}^{(l)}} J(\theta, b)$

  - $b_i^{(l)} = b_i^{(l)} - \eta \frac{\partial}{\partial b_i^{(l)}} J(\theta, b)$

  - $\frac{\partial}{\partial \theta_{ij}^{(l)}} J(\theta, b) = \frac{1}{m}\sum_{i=1}^{m} \frac{\partial}{\partial \theta_{ij}^{(l)}} J\left(\theta, b; x^{(i)}, y^{(i)}\right)$

  - $\frac{\partial}{\partial b_i^{(l)}} J(\theta, b) = \frac{1}{m}\sum_{i=1}^{m} \frac{\partial}{\partial b_i^{(l)}} J\left(\theta, b; x^{(i)}, y^{(i)}\right)$

$$J\left(\theta, b; x^{(i)}, y^{(i)}\right) = -\sum_{j=1}^{k} \mathbf{1}\{y^{(i)} = j\} \log \frac{\exp(z_j^{(n_l)})}{\sum_{j'=1}^{k} \exp(z_{j'}^{(n_l)})}$$

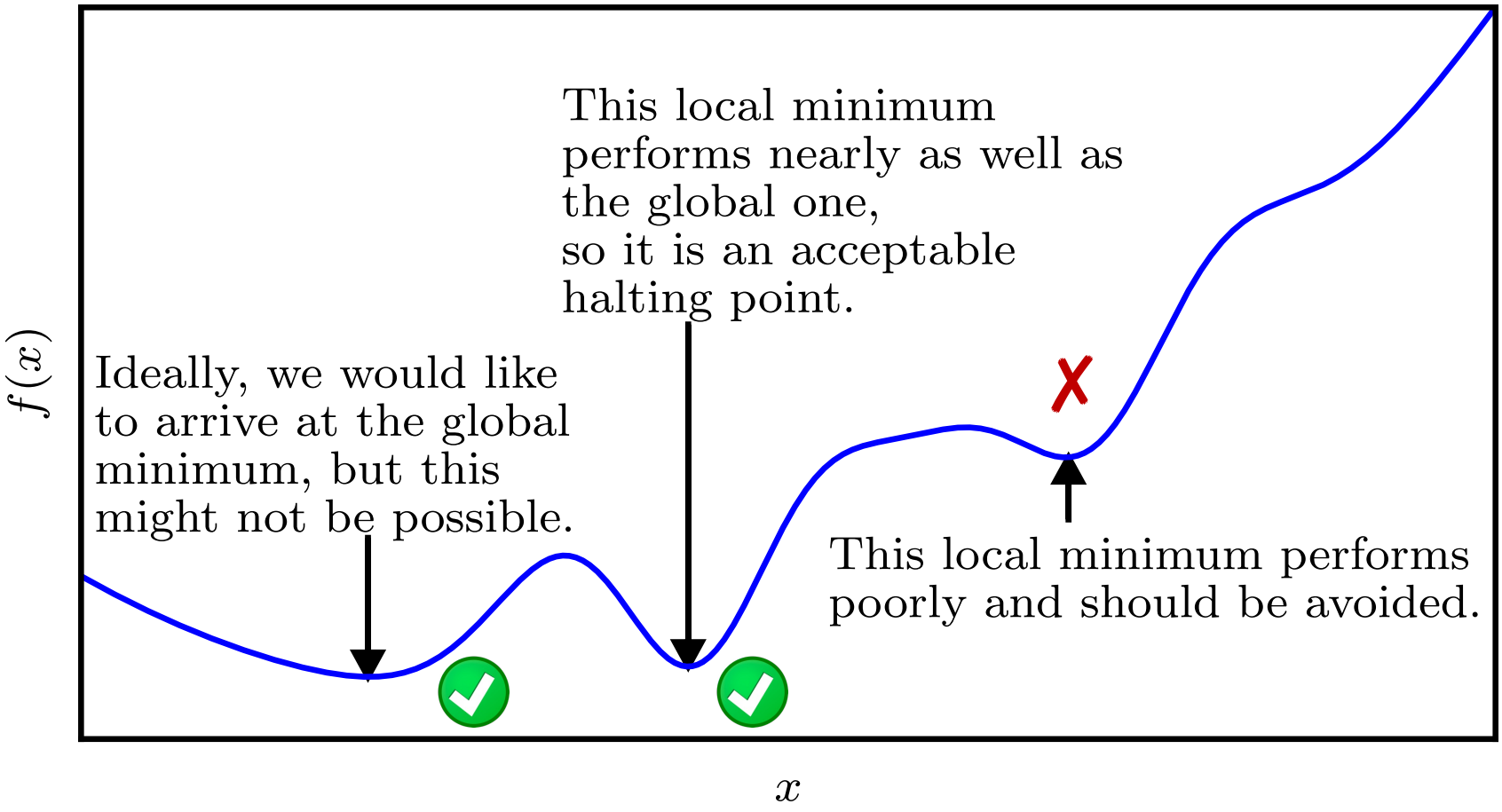
$$\frac{\partial J(\theta, b)}{\partial z_j^{(n_l)}} = -\left(\mathbf{1}\{y^{(i)} = j\} - p\left(y^{(i)} = j | x^{(i)}; \theta\right)\right)$$

Computing all these by hand is painful

Outline

- Deep Learning
- Multiplayer Perceptrons (MLP)
  - Backpropagation
  - **Training Strategies**
- Convolutional Neural Network (CNN)
  - Training Strategies
  - Standard Architectures

Optimization in Practice



Y. Bengio. Practical Recommendations for Gradient-Based Training of Deep Architectures. https://arxiv.org/pdf/1206.5533.pdf

Stochastic Gradient Descent (SGD)

$$J(\theta, b) = \left[\frac{1}{m}\sum_{i=1}^{m} J\left(\theta, b; x^{(i)}, y^{(i)}\right)\right]$$

- Stochastic Gradient Descent

  - $\theta_{ij}^{(l)} = \theta_{ij}^{(l)} - \eta \frac{\partial}{\partial \theta_{ij}^{(l)}} J(\theta, b)$

  ✅ Stochasticity helps escaping from saddle points.

  - $b_i^{(l)} = b_i^{(l)} - \eta \frac{\partial}{\partial b_i^{(l)}} J(\theta, b)$

  Too expensive when m is very large.

  - $\frac{\partial}{\partial \theta_{ij}^{(l)}} J(\theta, b) = \left[\frac{1}{m}\sum_{i=1}^{m} \frac{\partial}{\partial \theta_{ij}^{(l)}} J\left(\theta, b; x^{(i)}, y^{(i)}\right)\right]$

  For each iteration, compute gradients for a mini-batch (小批量) of data points

  - $\frac{\partial}{\partial b_i^{(l)}} J(\theta, b) = \frac{1}{m}\sum_{i=1}^{m} \frac{\partial}{\partial b_i^{(l)}} J\left(\theta, b; x^{(i)}, y^{(i)}\right)$

  For each epoch, Shuffle (洗牌) the dataset

SGD with Momentum

$$J(\theta, b) = \left[\frac{1}{m}\sum_{i=1}^{m} J\left(\theta, b; x^{(i)}, y^{(i)}\right)\right]$$

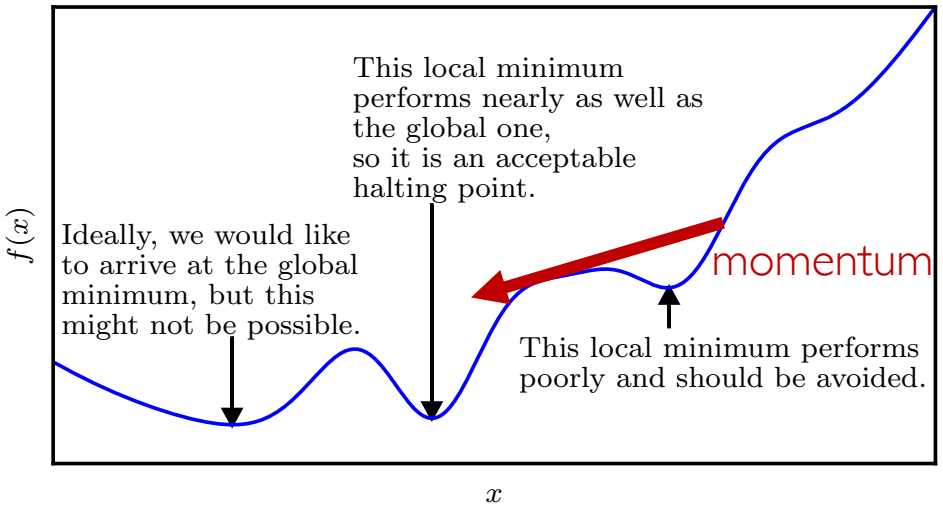
- Stochastic Gradient Descent

$$- \theta_{ij}^{(l)} = \theta_{ij}^{(l)} - \eta \frac{\partial}{\partial \theta_{ij}^{(l)}} J(\theta, b)$$

- With momentum

$$- \theta_{ij}^{(l)} = \theta_{ij}^{(l)} - \eta \Delta$$

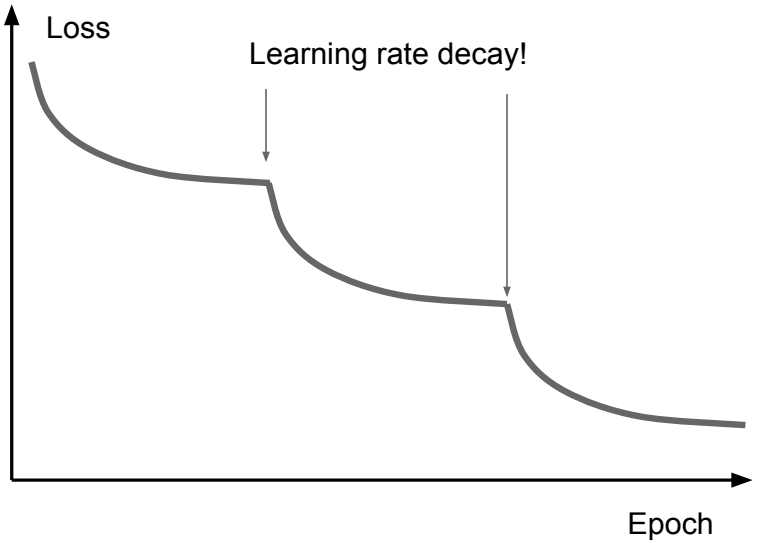
$$- \Delta = \beta \Delta + \frac{\partial}{\partial \theta_{i}^{(l)}} J(\theta, b)$$



- Many optimization methods (e.g. RMSProp, ADAM)

Learning Rate Decay

- All SGD algorithms take learning rate η as a hyperparameter
- Though some algorithms can adjust learning rate adaptively, a good choice of learning rate η could result in better performance
- To make network converge **stably and quickly**, we could set learning rate that **decays over time**



- Exponential decay strategy:

$$\eta = \eta_0 e^{-kt}$$

- 1/t decay strategy:

$$\eta = \eta_0 / (1 + kt)$$

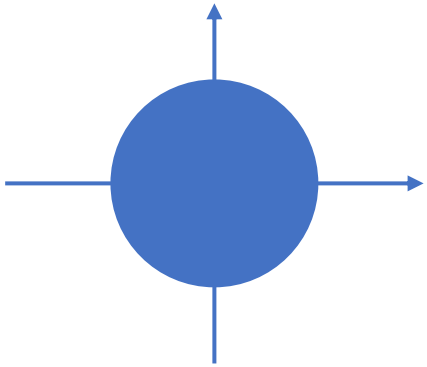
- Step strategy: decay very T iterations (widely used in practice)

Weight Decay

- *L2* regularization:

$$\Omega(\theta) = \frac{\lambda}{2} \sum_{l=1}^{n_l-1} \sum_{i=1}^{s_l} \sum_{j=1}^{s_{l+1}} (\theta_{ji}^{(l)})^2$$

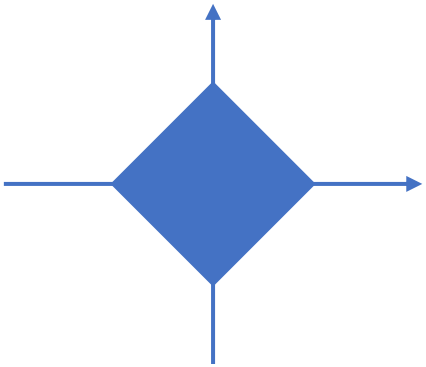
$$\frac{\partial}{\partial \theta^{(l)}} \Omega(\theta) = \lambda \theta^{(l)}$$



- *L1* regularization:

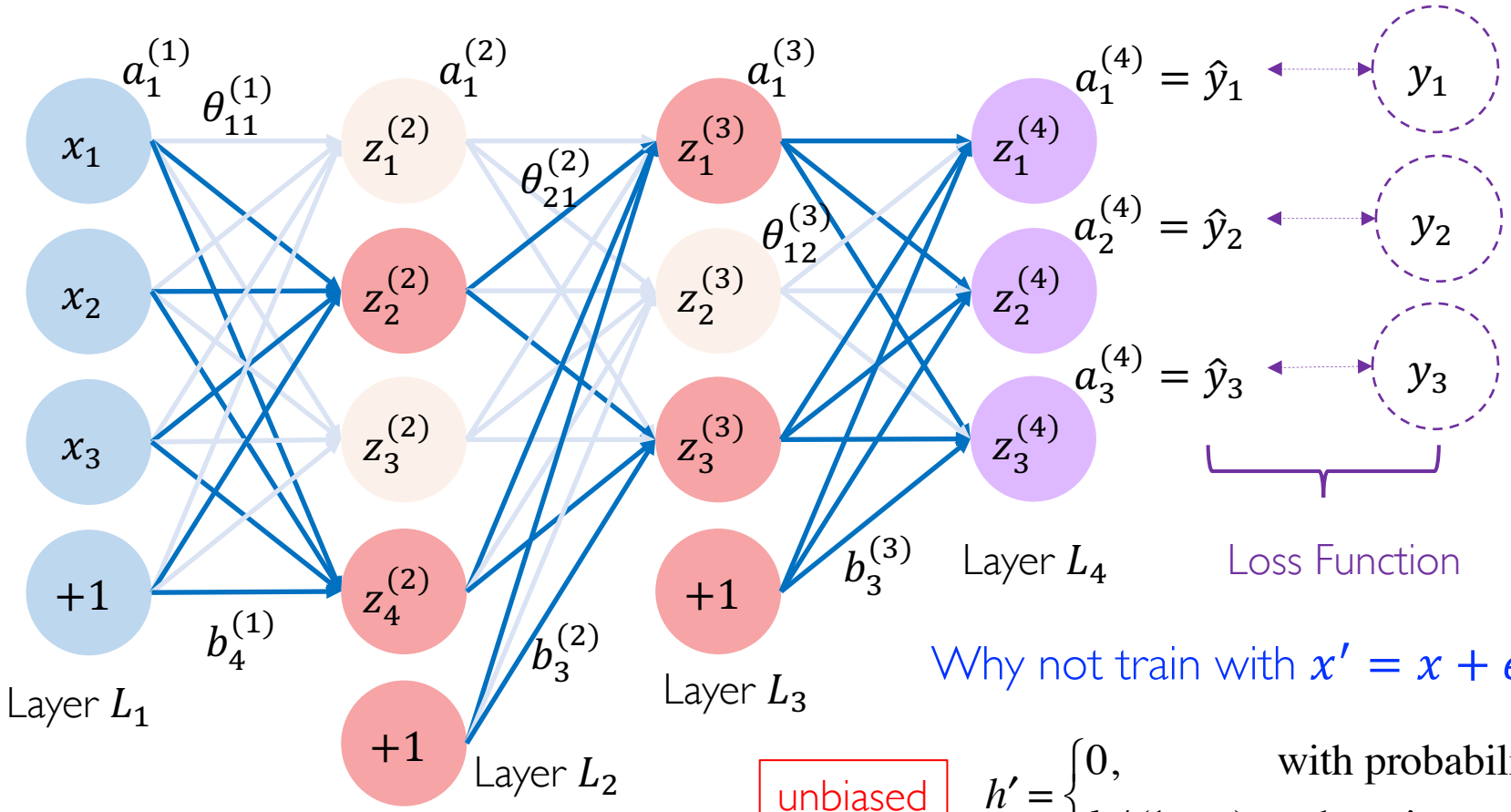
$$\Omega(\theta) = \lambda \sum_{l=1}^{n_l-1} \sum_{i=1}^{s_l} \sum_{j=1}^{s_{l+1}} |\theta_{ji}^{(l)}|$$

$$\frac{\partial}{\partial \theta^{(l)}} \Omega(\theta)_{ji} = \lambda (1_{\theta_{ji}^{(l)} > 0} - 1_{\theta_{ji}^{(l)} < 0})$$



Dropout

- During training, randomly drop 50% of activations in each layer to $\mathbf{0}$



Why not train with $x' = x + \epsilon$?

unbiased

$$h' = \begin{cases} 0, & \text{with probability } p \\ h/(1-p), & \text{otherwise} \end{cases}$$

Weight Initialization

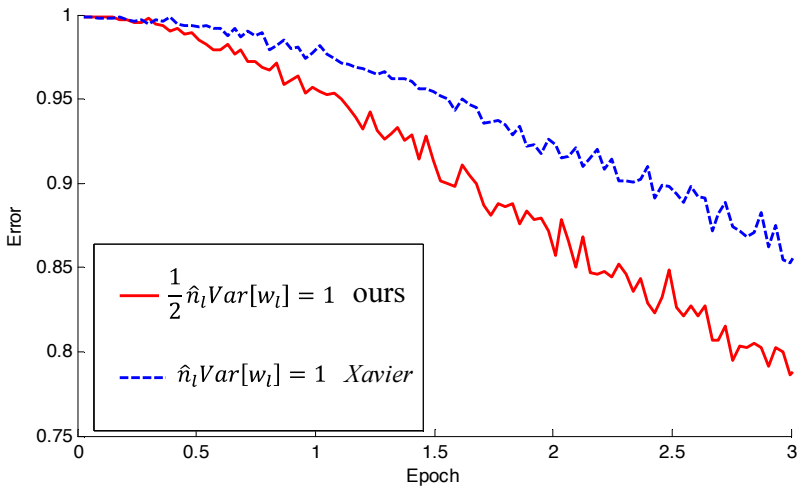
multiplicative effect through layers

Proper initialization avoids reducing or magnifying the magnitudes of signals exponentially

Xavier Initialization

(linear activations)

$$\mathrm{Var}(w) = 1 / n_{\text{in}}$$

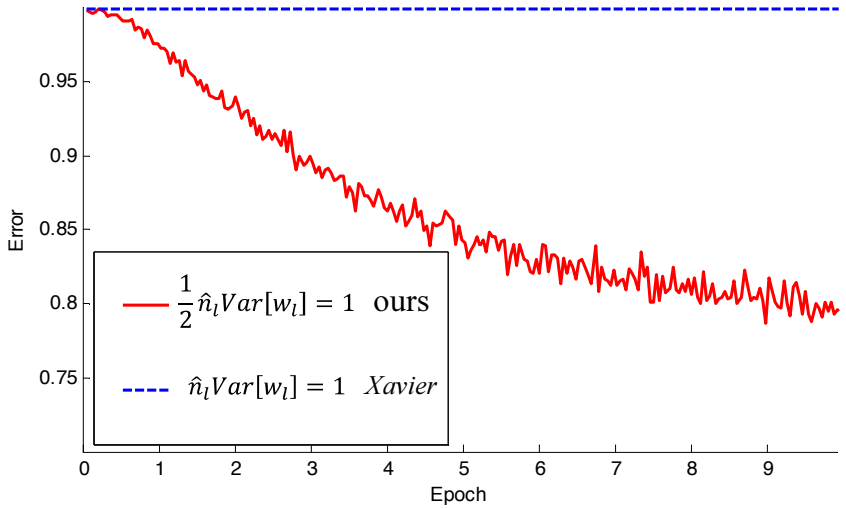


22-layer ReLU network

He Initialization

(ReLU activations)

$$\mathrm{Var}(w) = 2 / n_{\text{in}}$$

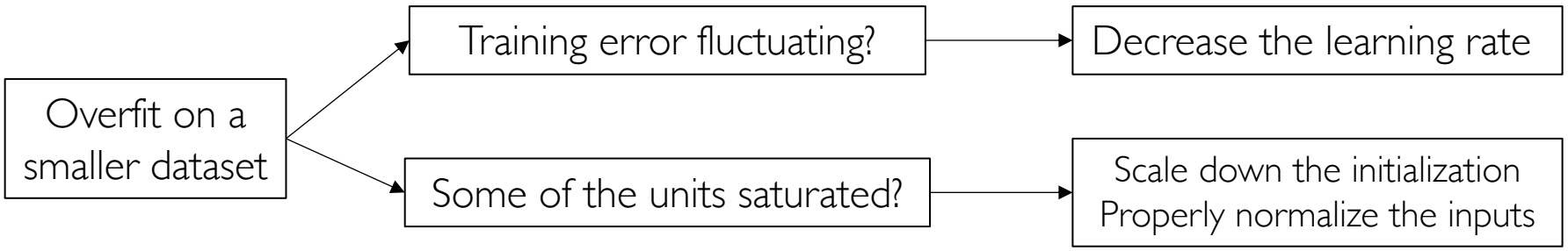


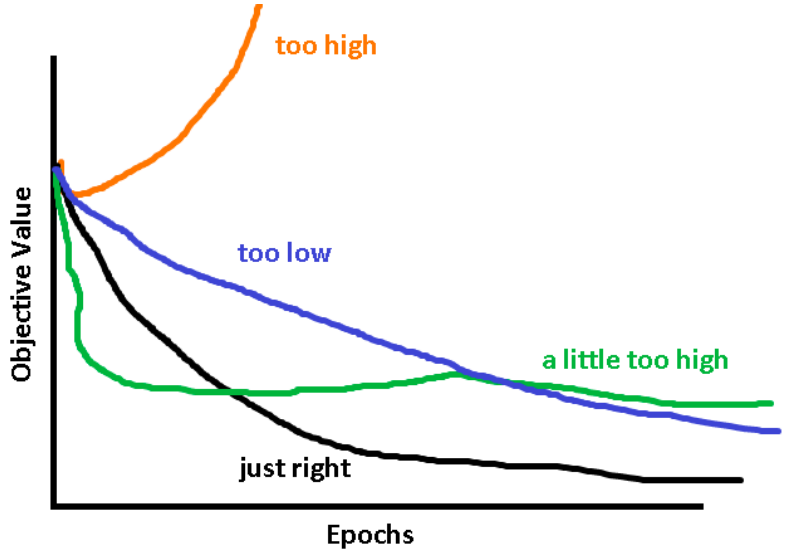
30-layer ReLU network

Glorot X, Bengio Y. Understanding the difficulty of training deep feedforward neural networks. JMLR, 2010, 9:249-256.

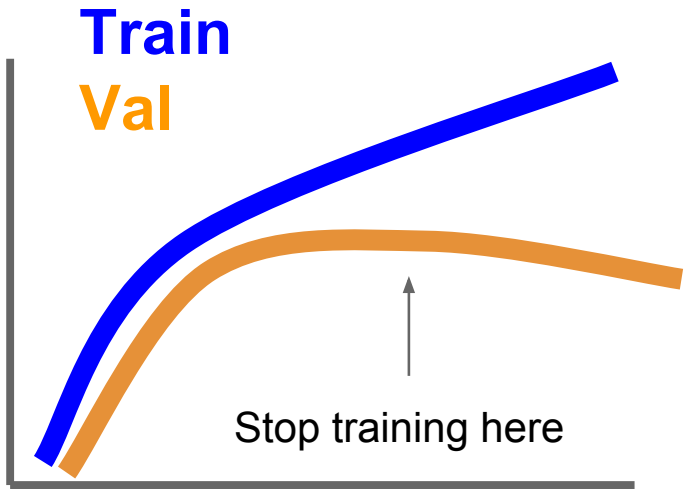
He K, Zhang X, Ren S, et al. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification[J]. 2015:1026-1034.

Babysitting Learning





learning rate



early stopping

Outline

- Deep Learning
- Multiplayer Perceptrons (MLP)
  - Backpropagation
  - Training Strategies
- **Convolutional Neural Network (CNN)**
  - Training Strategies
  - Standard Architectures

What Are They

Nose, Eyes, Mouth

A beautiful woman

Wheels, License Plate, Headlights

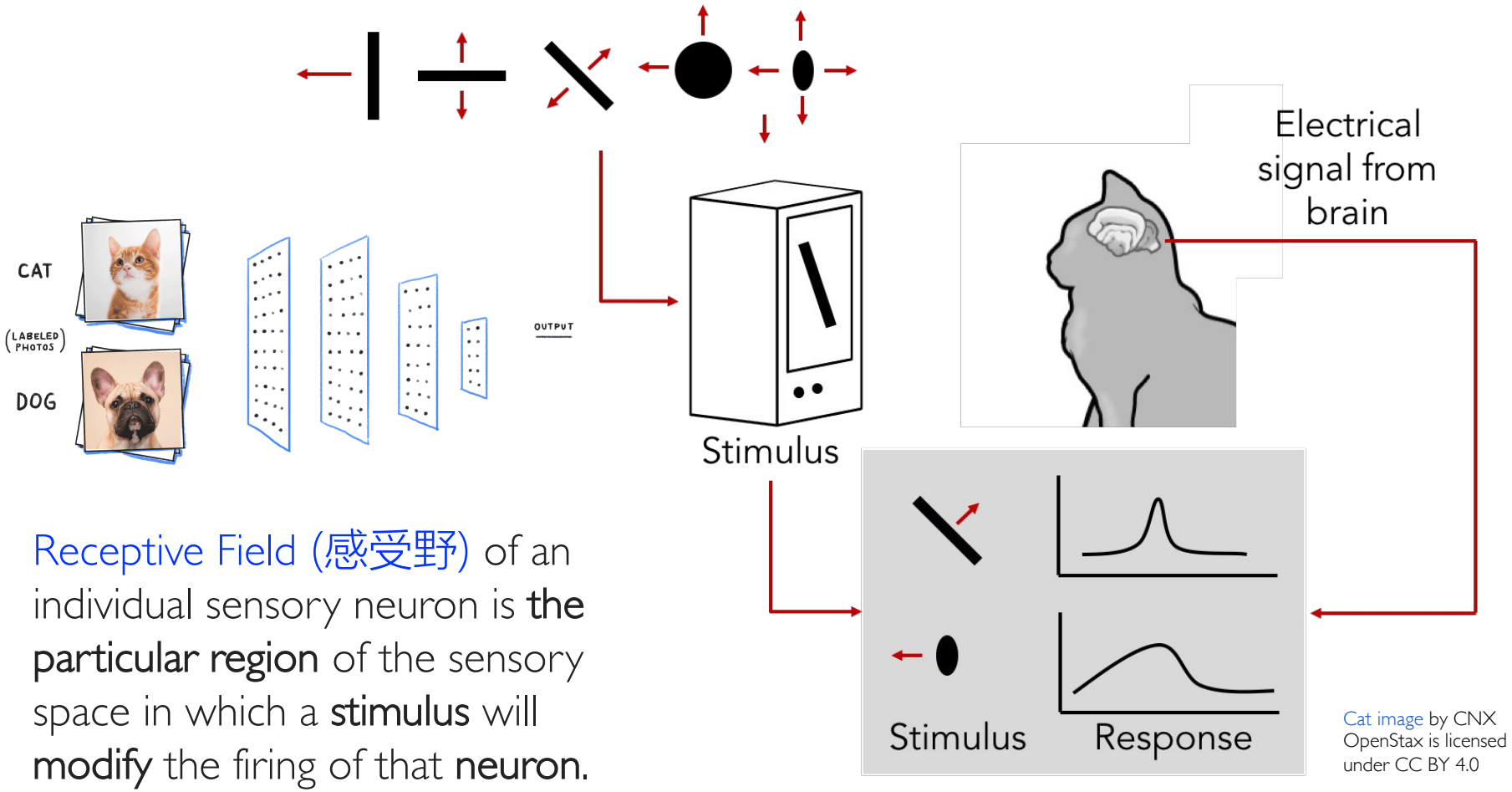
An vehicle off the road

Door, Windows

A simple villa

Humans are good at making multi-aspect descriptions of an image, but machines cannot.

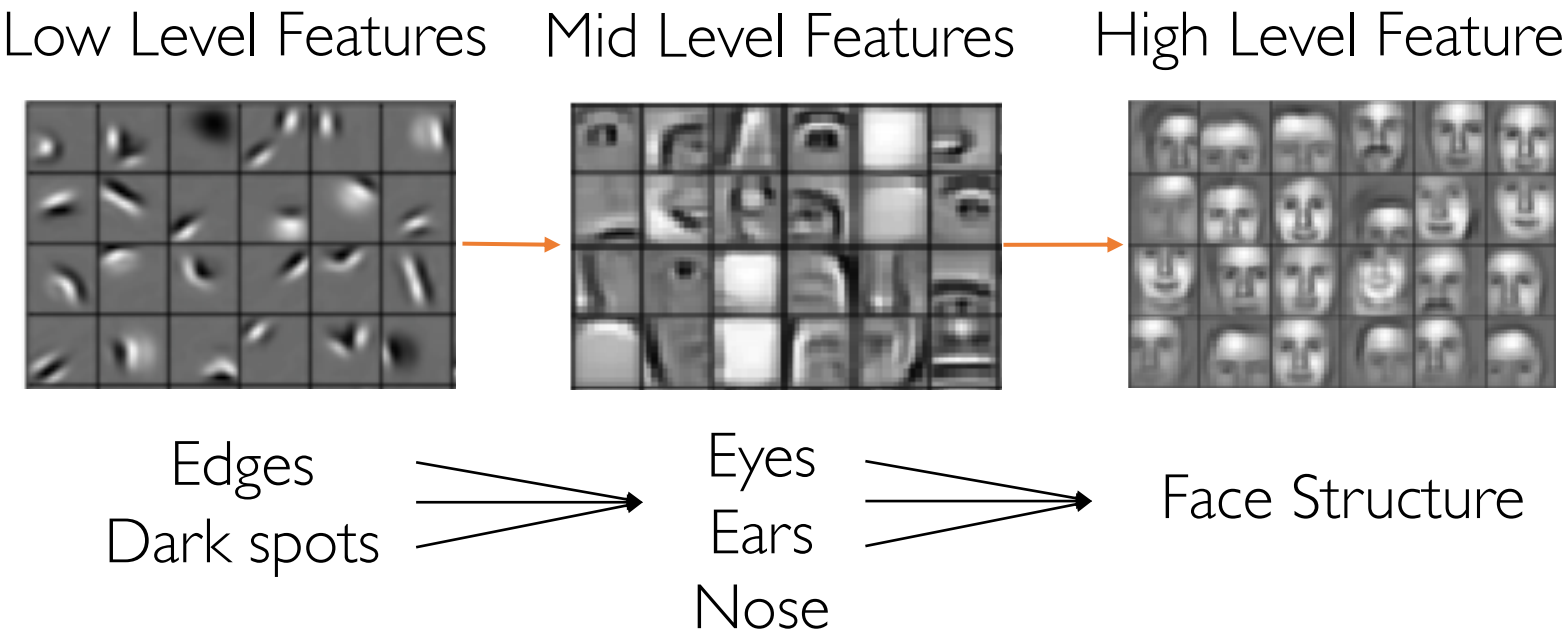
Human Vision



Receptive Field (感受野) of an individual sensory neuron is **the particular region** of the sensory space in which a **stimulus** will **modify** the firing of that **neuron**.

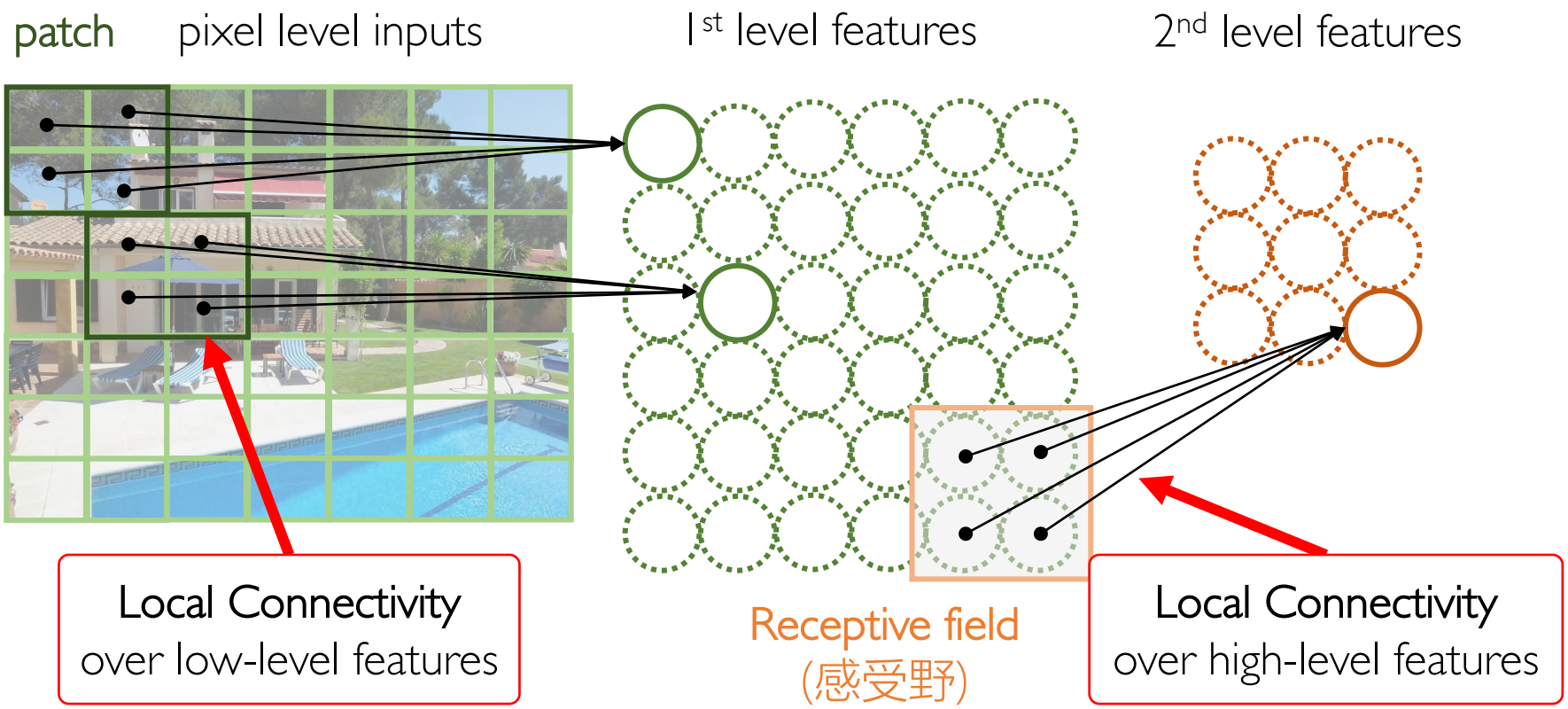
DH Hubel ,TN Wiesel. "Receptive fields of single neurons in the cat's striate cortex ." Journal of Physiology, 1959.

Machine Vision



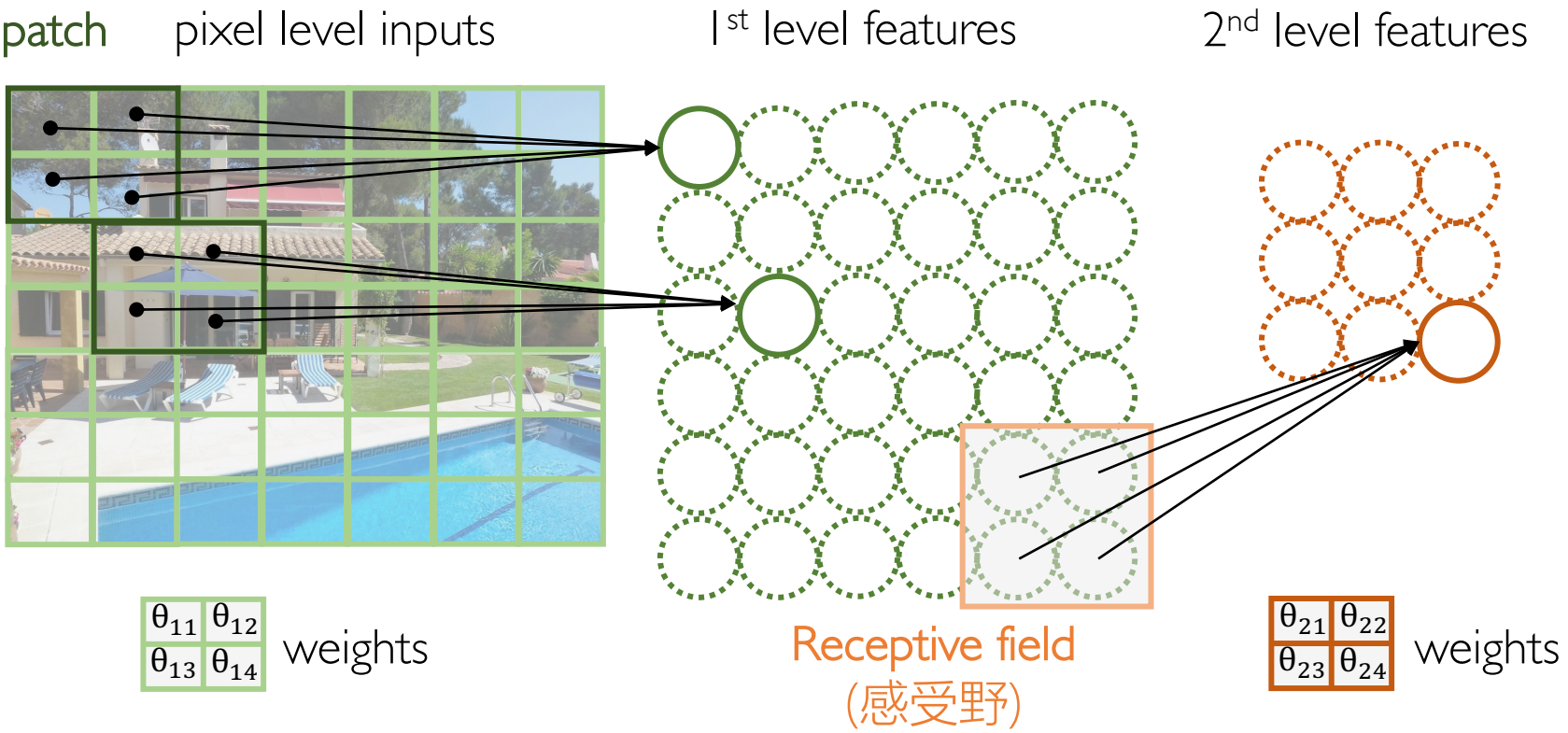
To mimic the human visual system, we need to learn hierarchy of feature representations directly from data without hand engineering.

Idea 1: Local Connectivity



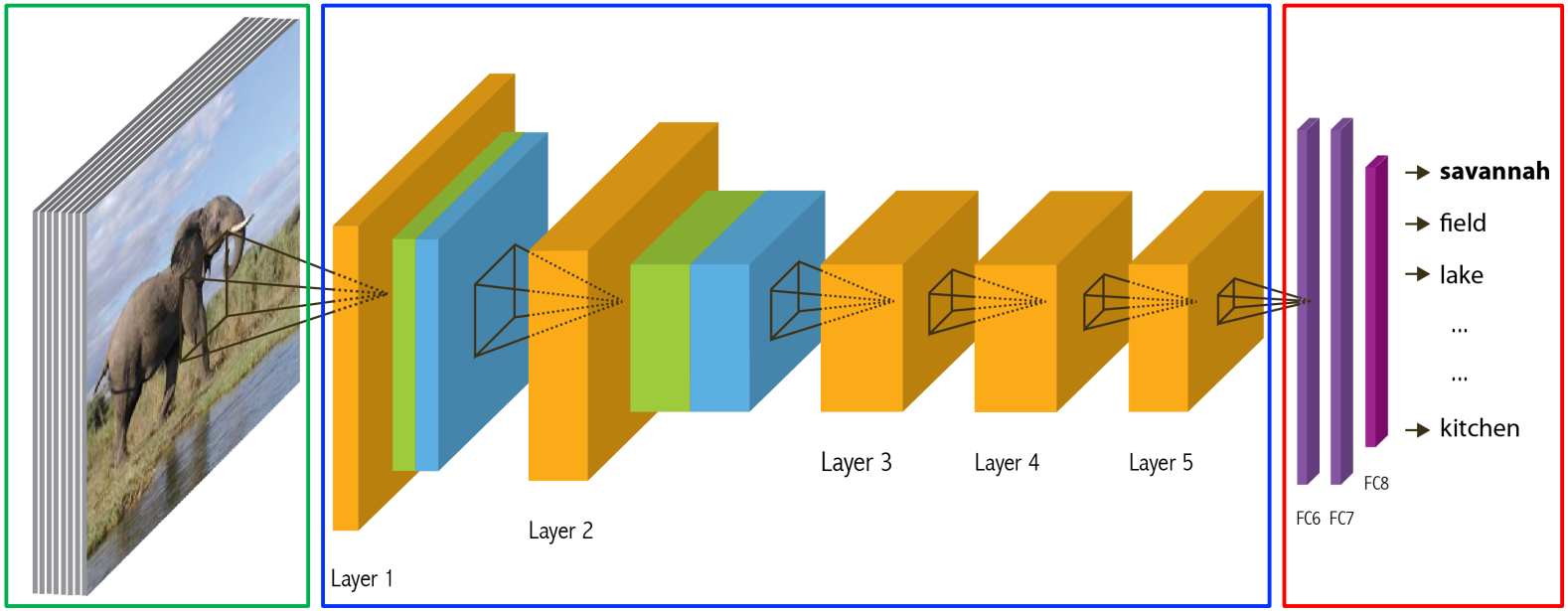
- **[Locality Assumption]:** local information is enough for recognition
- Connect each neuron to only a local region of the input volume.

Idea 2: Parameter Sharing



- **[Shift Invariance Assumption]:** If a feature is useful at spatial position (x, y), then it should also be useful for all positions (x', y')
- Share the weights of sliding window over different spatial locations

Convolutional Neural Network



Scan the Image

Generate hierarchy of features

Recognize by high level features

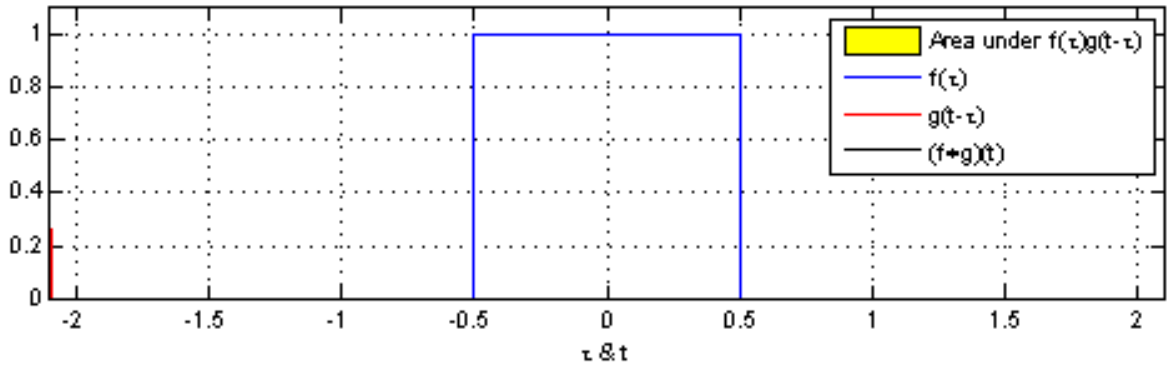
Convolution

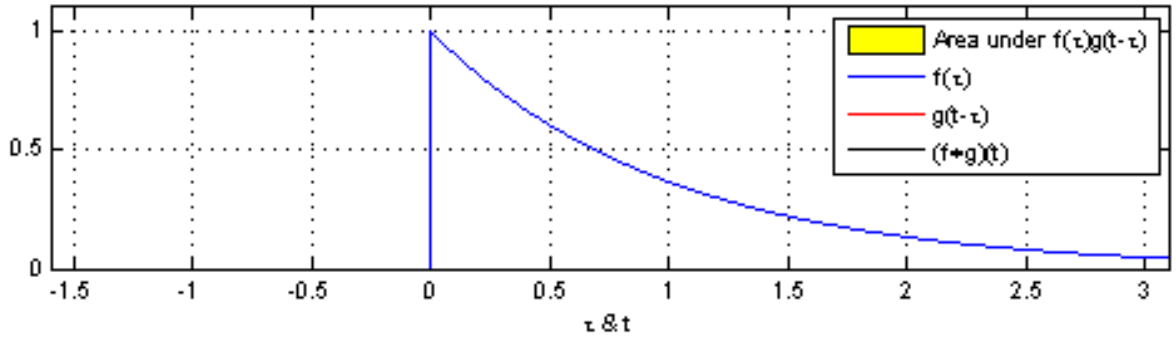
Continuous functions:

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

Discrete functions:

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





As t changes, the weighting function g emphasizes different parts of the input function f.

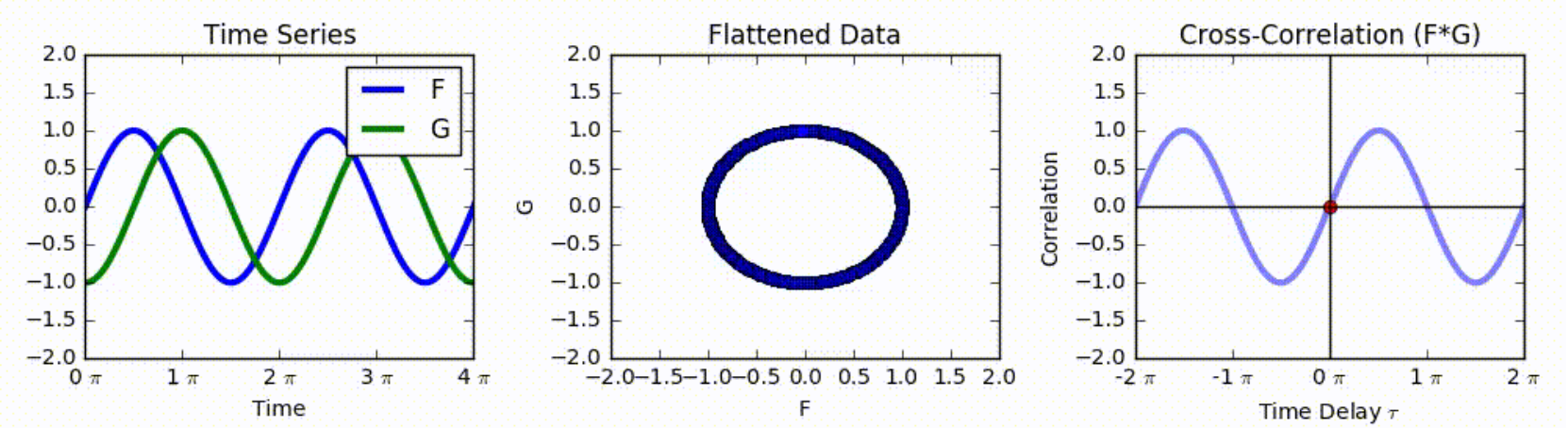
In signals processing, **convolution** is used with input signal and impulse response (脉冲) to produce an output signal.

Cross-Correlation

- Cross-correlation (sliding inner product) is a similarity measure of two series $f(\tau), g(\tau)$ as a function of the lag of one relative to the other.

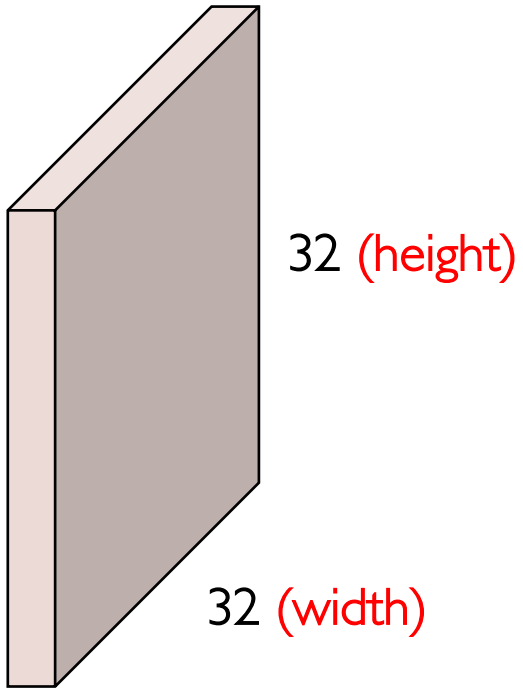
$$(f \star g)(\tau) \triangleq \int_{-\infty}^{\infty} \overline{f(t)} g(t+\tau)\, dt$$

$$(f \star g)[n] \triangleq \sum_{m=-\infty}^{\infty} \overline{f[m]} g[m+n]$$



Notations

32x32x3 image (width x height x RGB)



32 (height)

32 (width)

3

depth

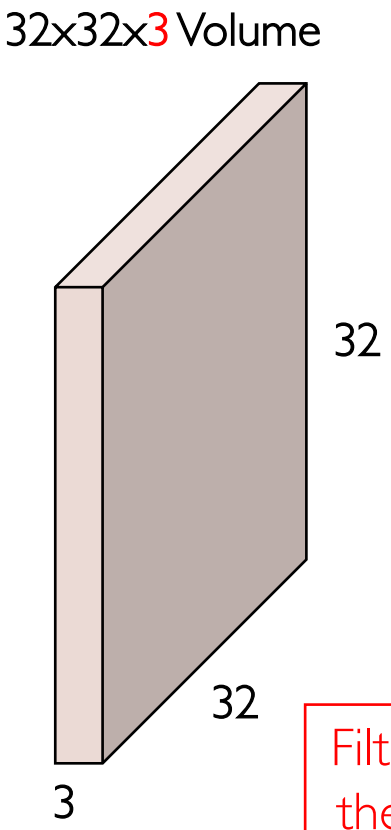
channels

Volume

Tensor

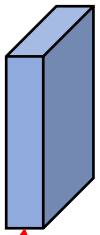
Input channels: R/G/B

Convolution: Local Connection



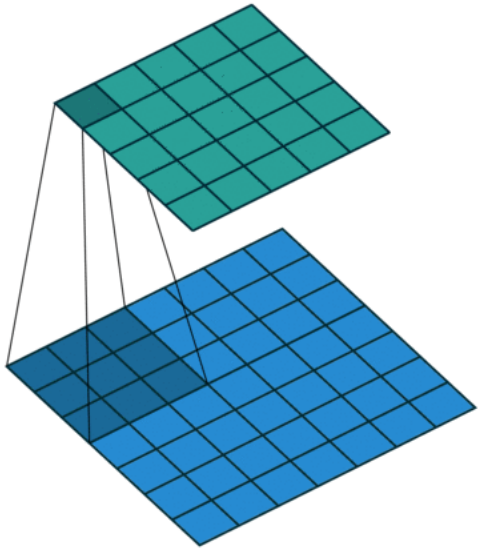
kernel size

5x5x3 filter (kernel)



Note: In other conv, this does not necessarily hold

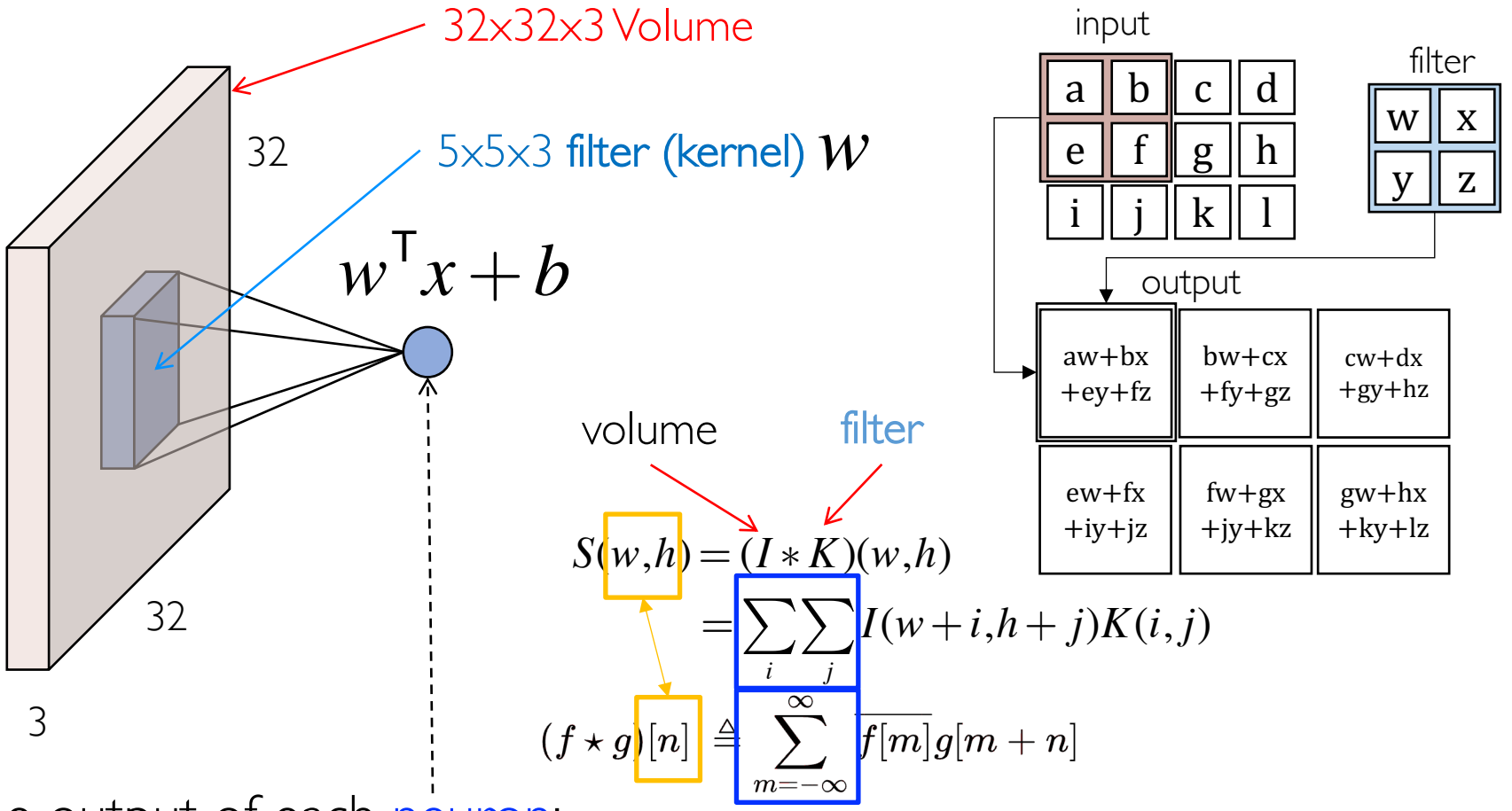
Filter always extends the full **depth** of the **input** volume (use all features)



A simple picture of 3x3x1 convolution

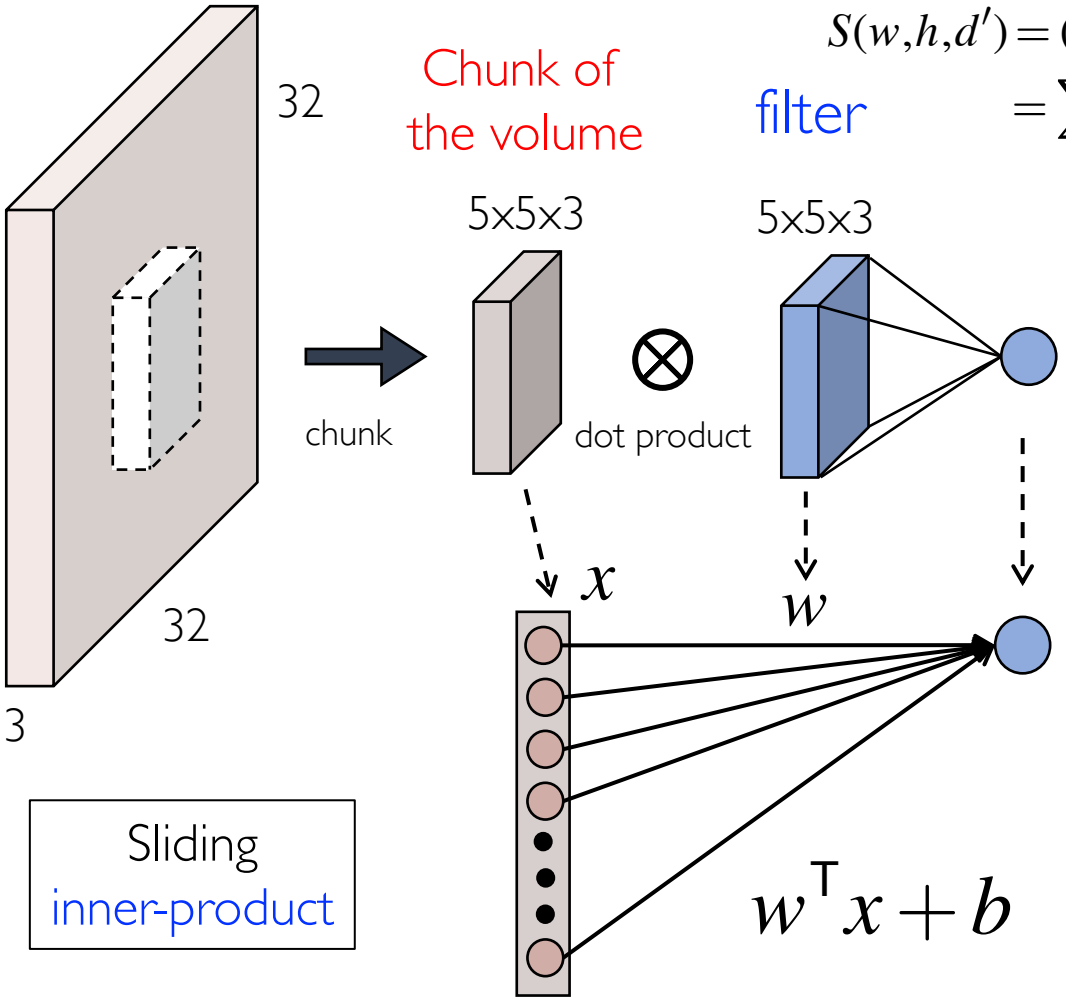
Convolve the filter with the image i.e. "**slide over** the image spatially, and computing the **dot products**"

Convolution: Local Connection



- The output of each neuron:
  - Dot product between filter and a small 5x5x3 chunk of the volume

Convolution: Local Connection



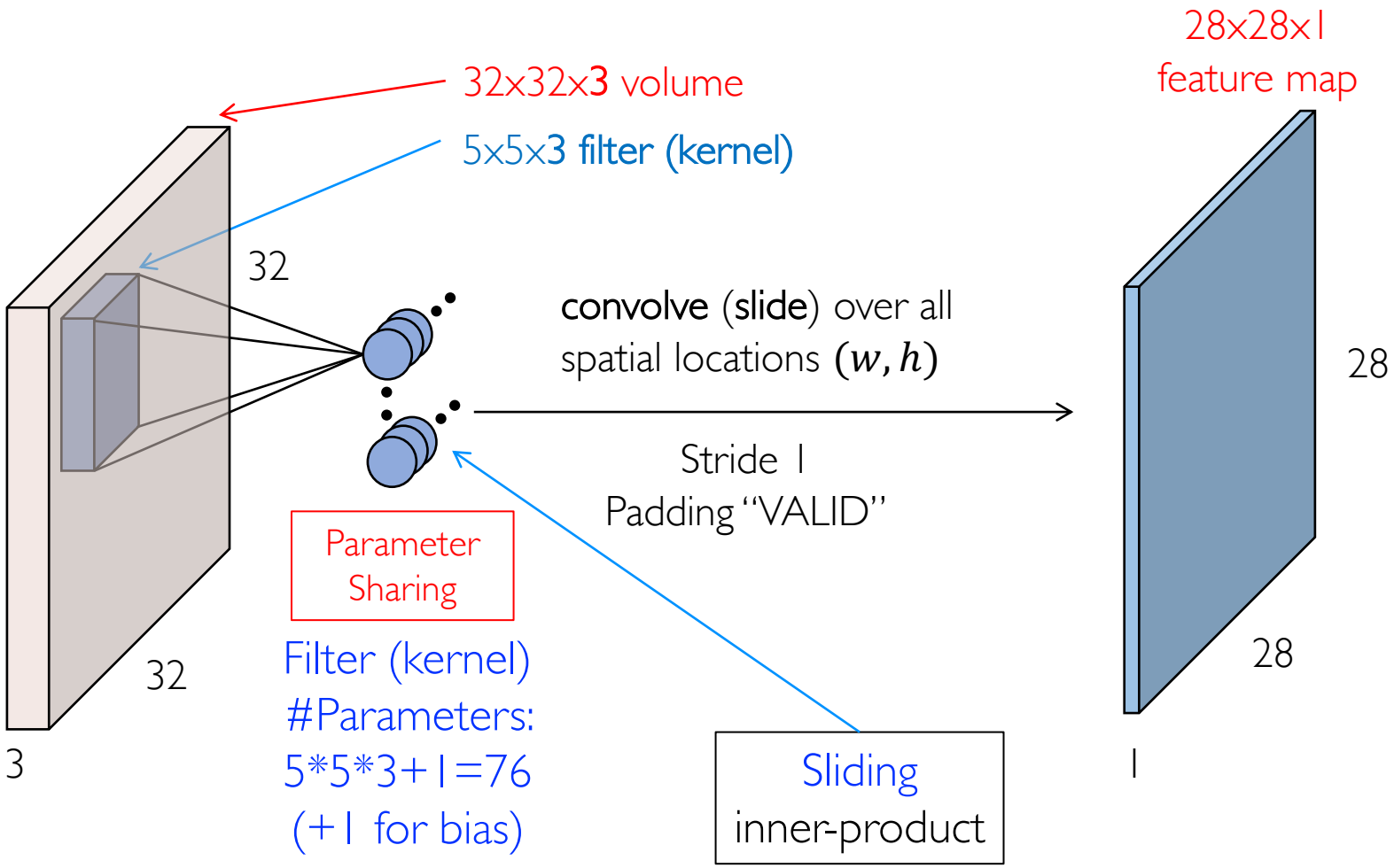
$$S(w,h,d') = (I * K)(w,h,d')$$
$$= \sum_d \sum_i \sum_j I(w+i,h+j,d) K(i,j,d)$$

Chunk of the volume

filter (kernel)

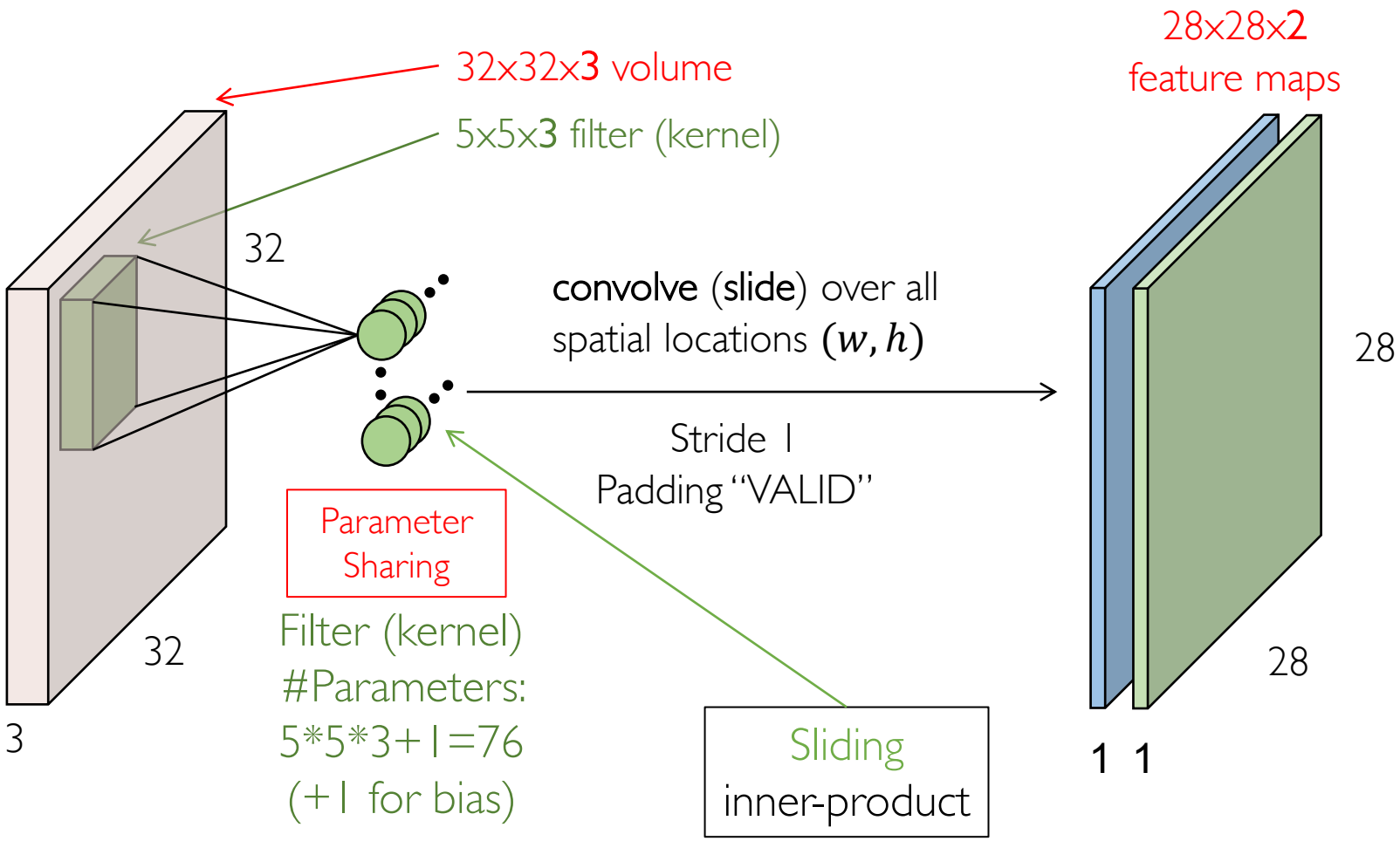
The output of each neuron: Dot product between filter and a small 5x5x3 chunk of the volume

Convolution: Parameter Sharing



Not that no "sliding" in practice, we add neuron for each location (w, h)

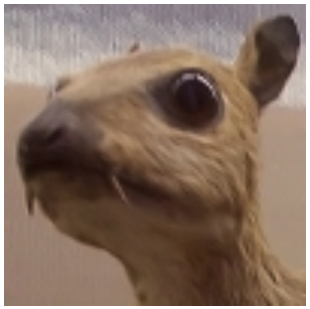
Convolution: Parameter Sharing



Not that no "sliding" in practice, we add neuron for each location (w, h)

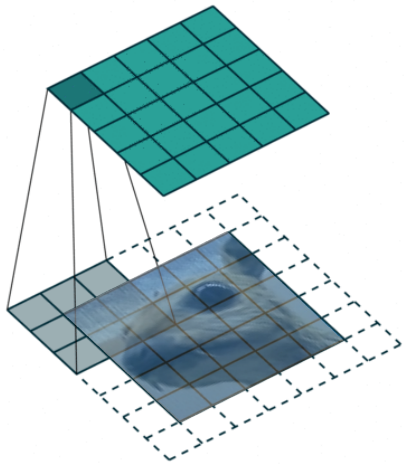
Convolution: Examples

- Convolutions with different parameters will lead to different effects

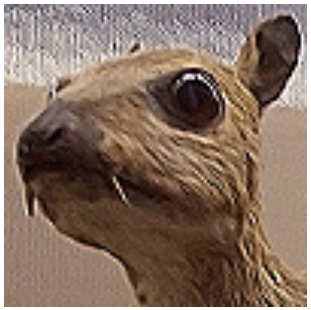


$$\begin{pmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{pmatrix}$$

Edge Detection



$$\begin{pmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{pmatrix}$$

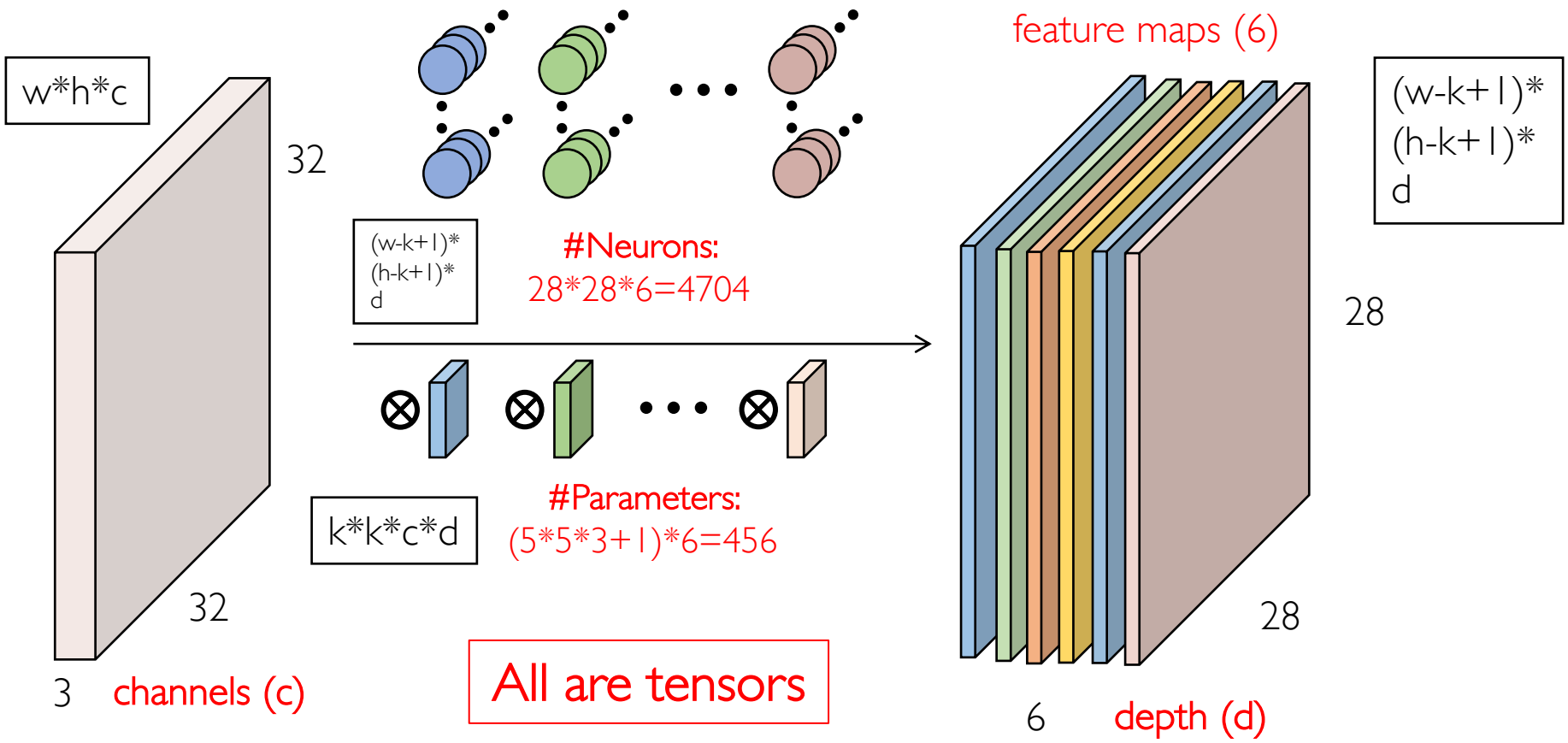


Sharpen

$$\frac{1}{16}\begin{pmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{pmatrix}$$

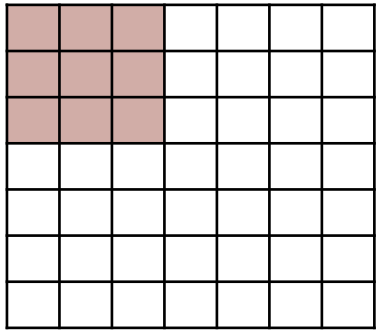
Gaussian Blur

Convolution: Multiple Feature Maps



We stack all feature maps up to get a "new image" (**representation**) of size 28x28x6. (However, we cannot "see" the new image since it has more than 3 R/G/B channels)

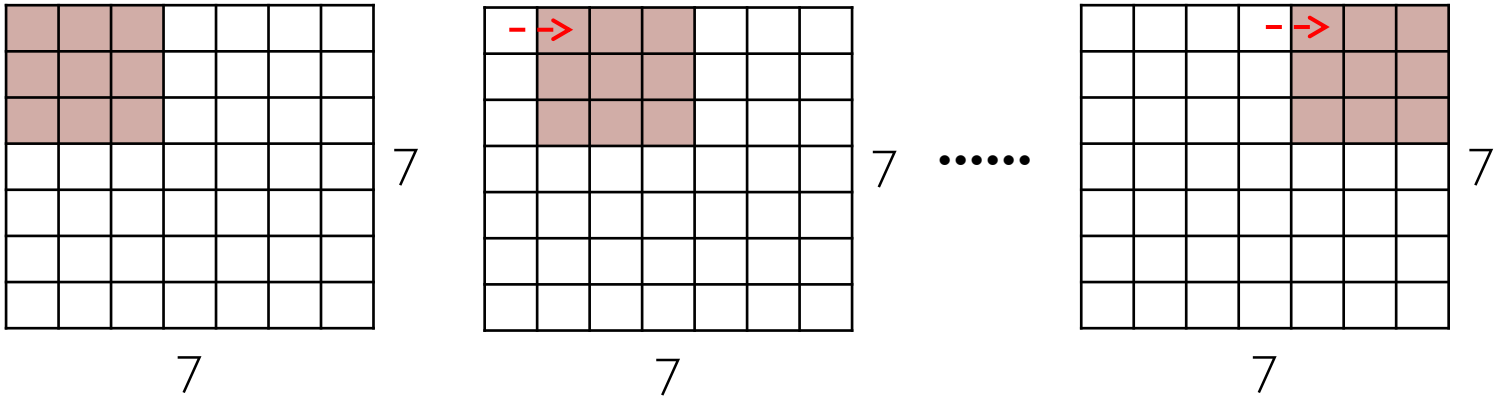
Stride



7x7xD input
3x3xD filter

Stride 1

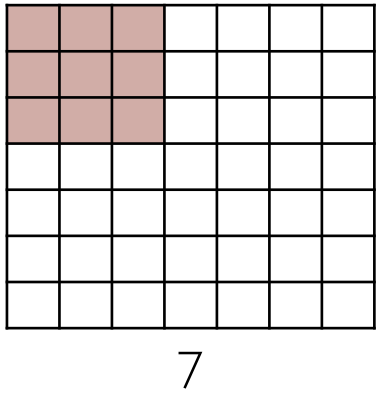
Stride 1



..... → 5x5 output

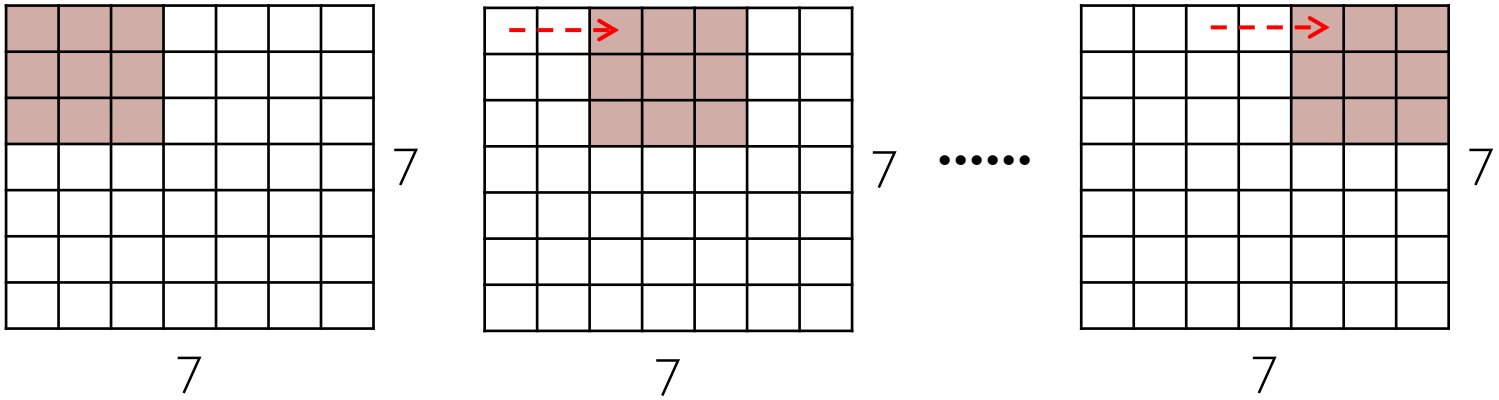
- In some cases we want to **reduce the spatial resolution** ($W \times H$). In this case we might want to subsample the output. Strides address this.

Stride



7x7xD input
3x3xD filter

Stride 2 Stride 2

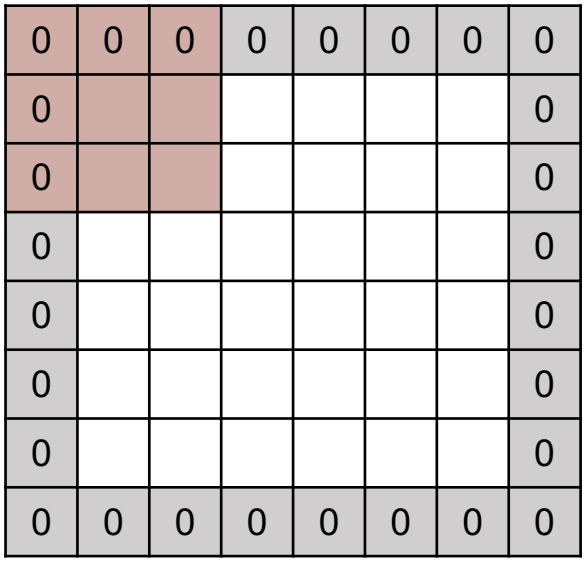


..... → 3x3 output

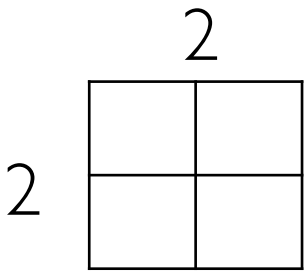
- The stride is performed over the spatial dimensions for each channel.

Padding

Input 7x7



Stride 3
3x3 Filter



(Pad P)

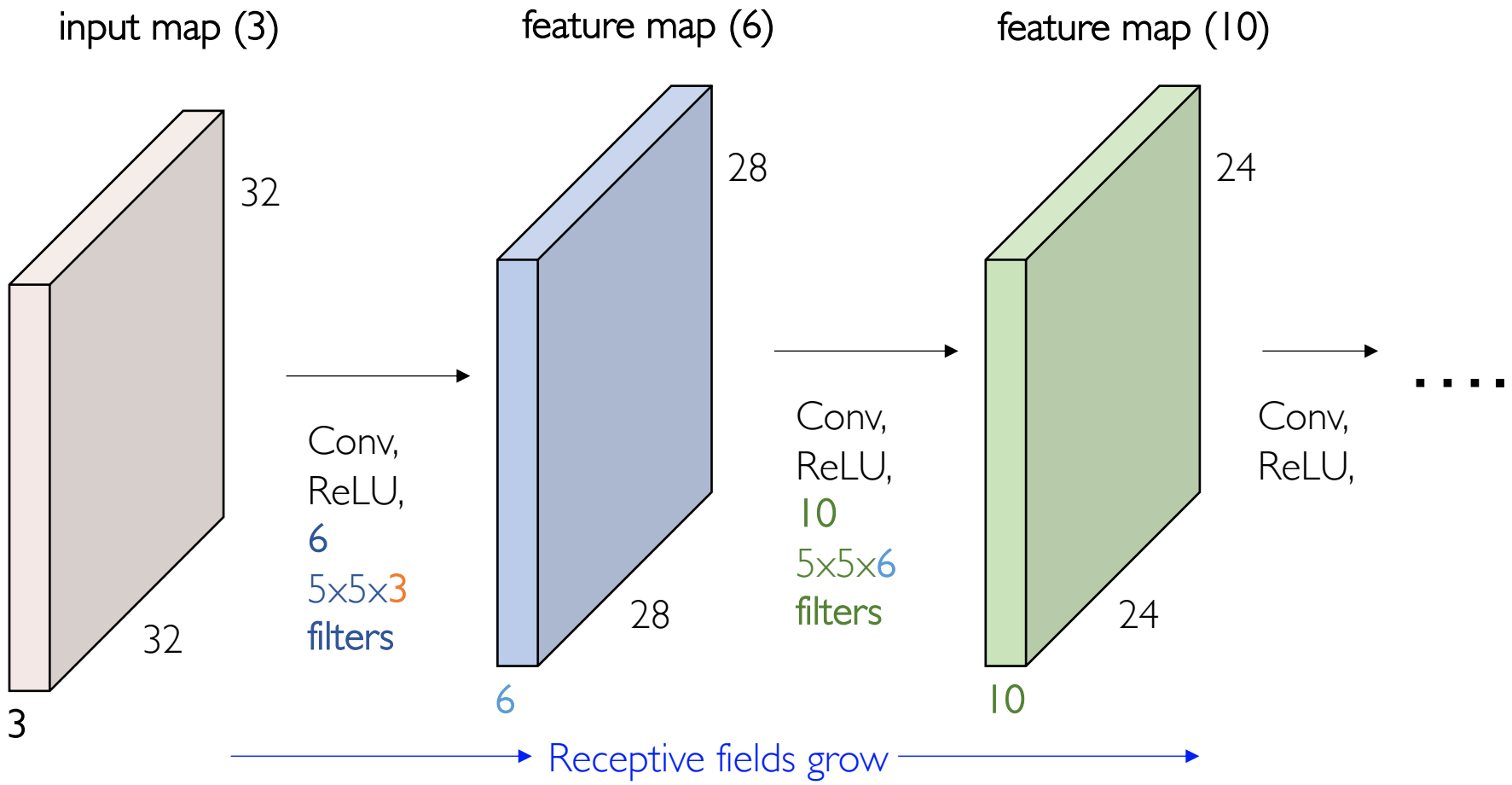
Add *P* circles of zeros outside of the original map

$$\text{height}_{\text{new}} = \lceil (\text{height-filter+2*pad}) / \text{stride} \rceil + 1$$

$$\text{width}_{\text{new}} = \lceil (\text{width-filter+2*pad}) / \text{stride} \rceil + 1$$

- Multiple layers of convolutions reduce the spatial sizes $W \times H$ and the information available at the boundary. Padding mitigates this problem.

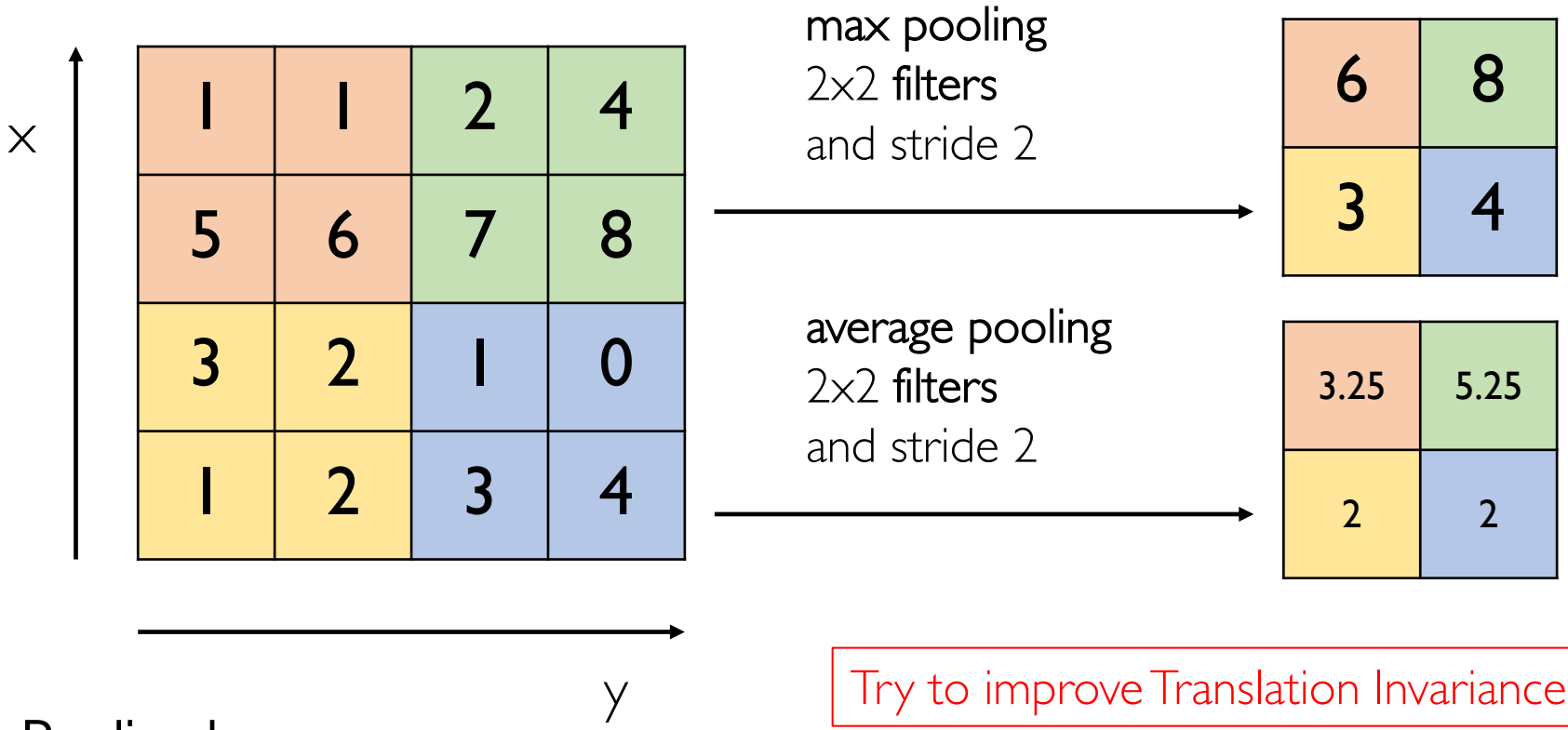
Convolutional Layer



CNN is a stack of Convolutional Layers, interspersed with activation functions

Note: Shrinking too fast in the spatial dimensions is not good, doesn't work well.

Pooling

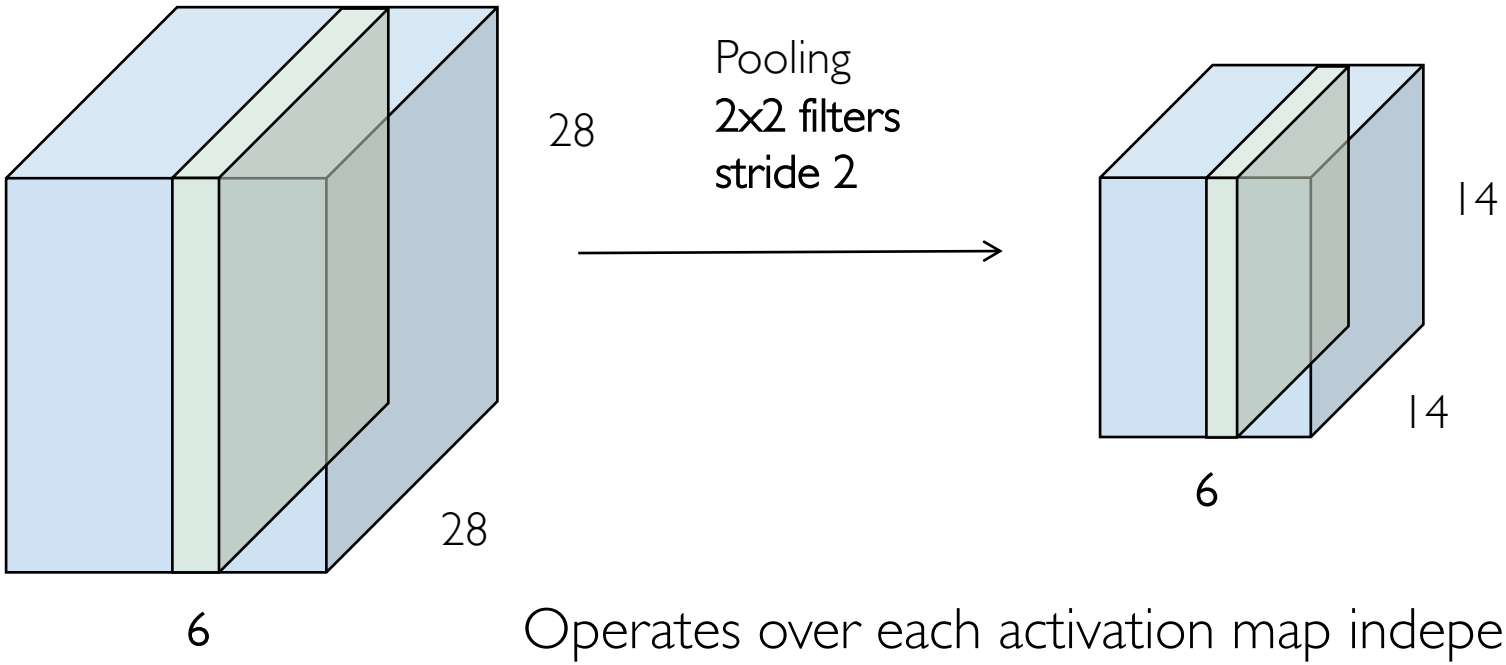


Try to improve Translation Invariance

- Pooling layers
  - Alleviate excessive sensitivity of convolutional layers to locations
  - Reduce the resolution of images through the processing pipeline.

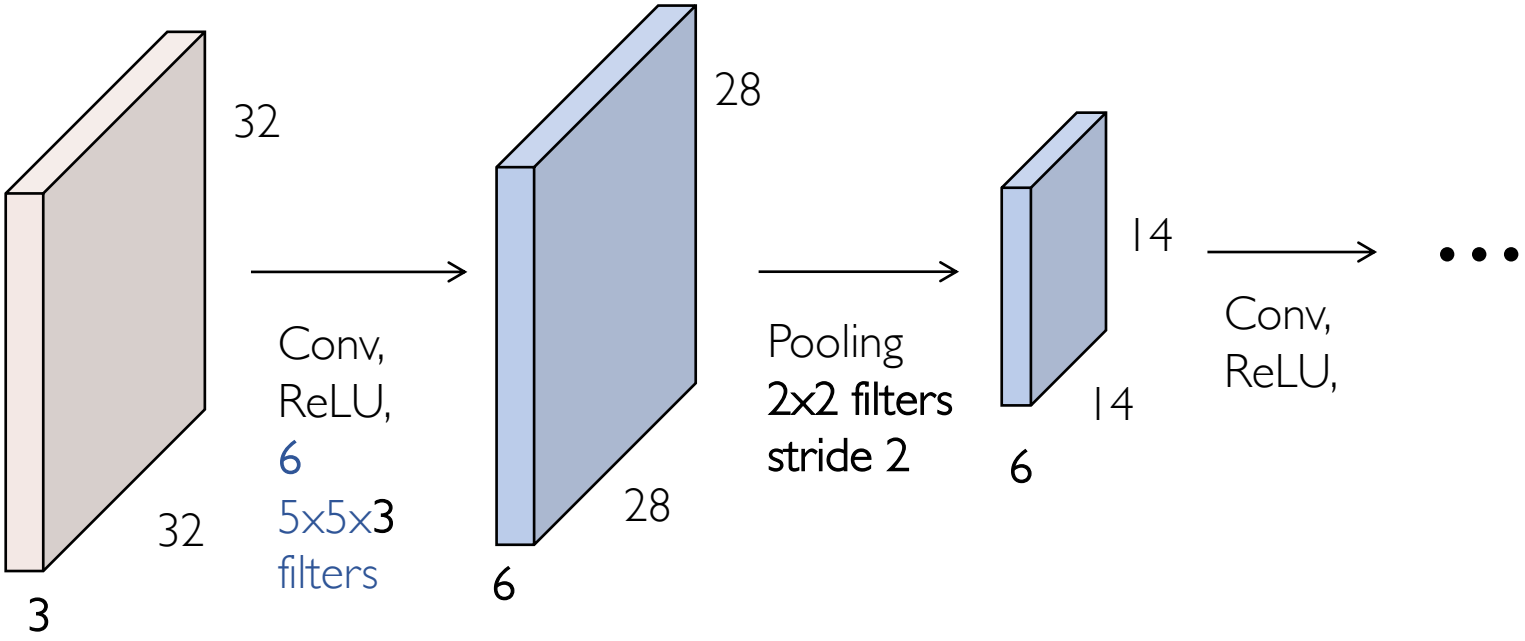
Pooling

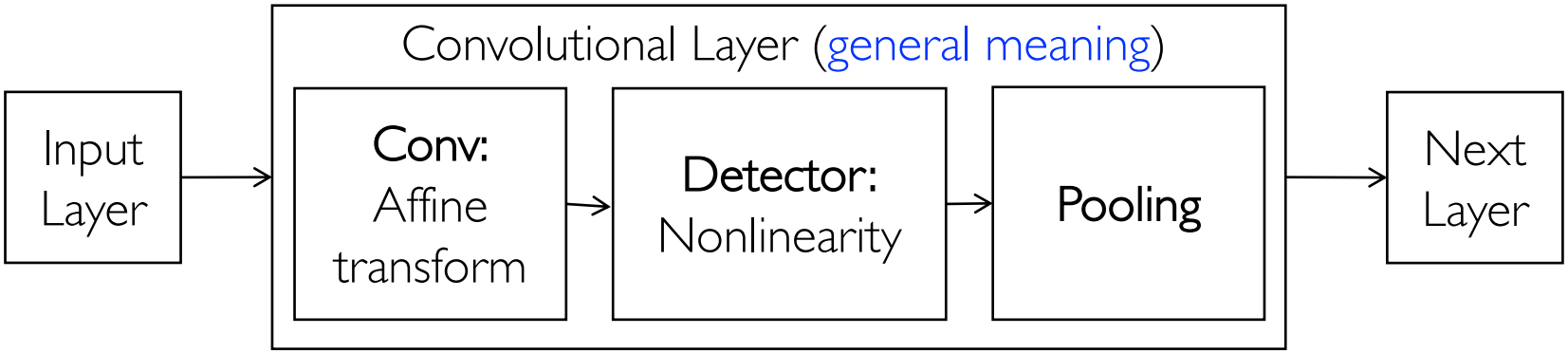
Try to improve Translation Invariance
But may fail.



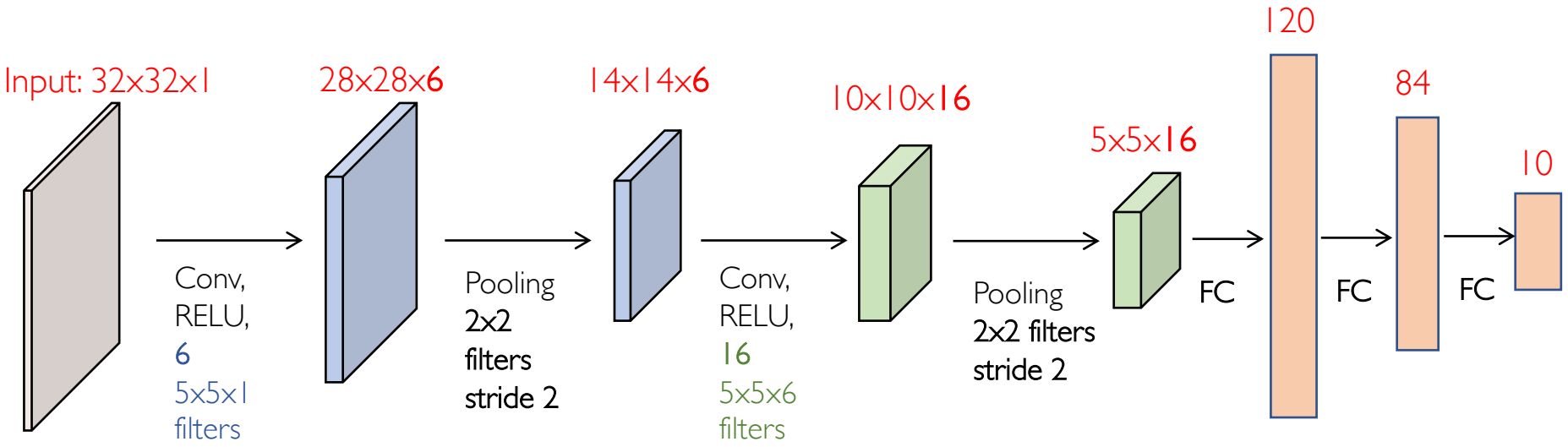
Operates over each activation map independently:
Pooling reduces dimensions of width and height.
Depth keeps unchanged before and after pooling

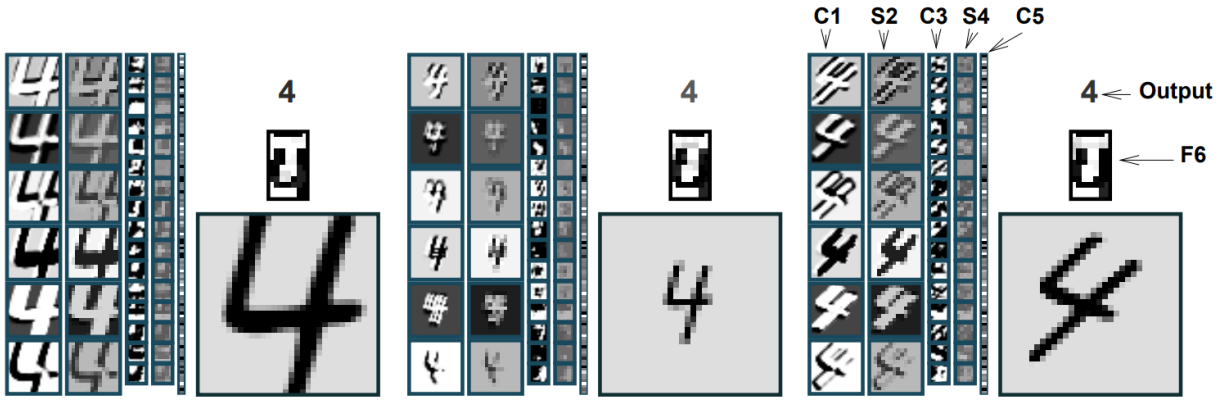
Convolutional Network: Typical Layer





LeNet





Y. LeCun, L. Bottou, Y. Bengio, Gradient-based learning applied to document recognition, Proc. IEEE, 1998. (Cited 53305)

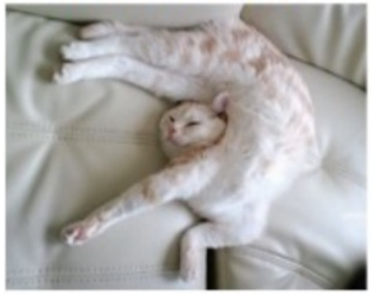
Outline

- Deep Learning
- Multiplayer Perceptrons (MLP)
  - Backpropagation
  - Training Strategies
- Convolutional Neural Network (CNN)
  - **Training Strategies**
  - Standard Architectures

Difficulties in Practice

Background clutter

Deformation



Flipped Picture

Viewpoint variation

Intra-class variation

Scale variation

?

Illumination conditions

Occlusion

Data Augmentation

- Get **more data** for training
- Give the model **high adaptability**

Color Jittering

PCA Jittering

Random Scale

Random Crop

Scale jittering

Horizontal/Vertical Flip

Shift

Rotation/Reflection

Label shuffle

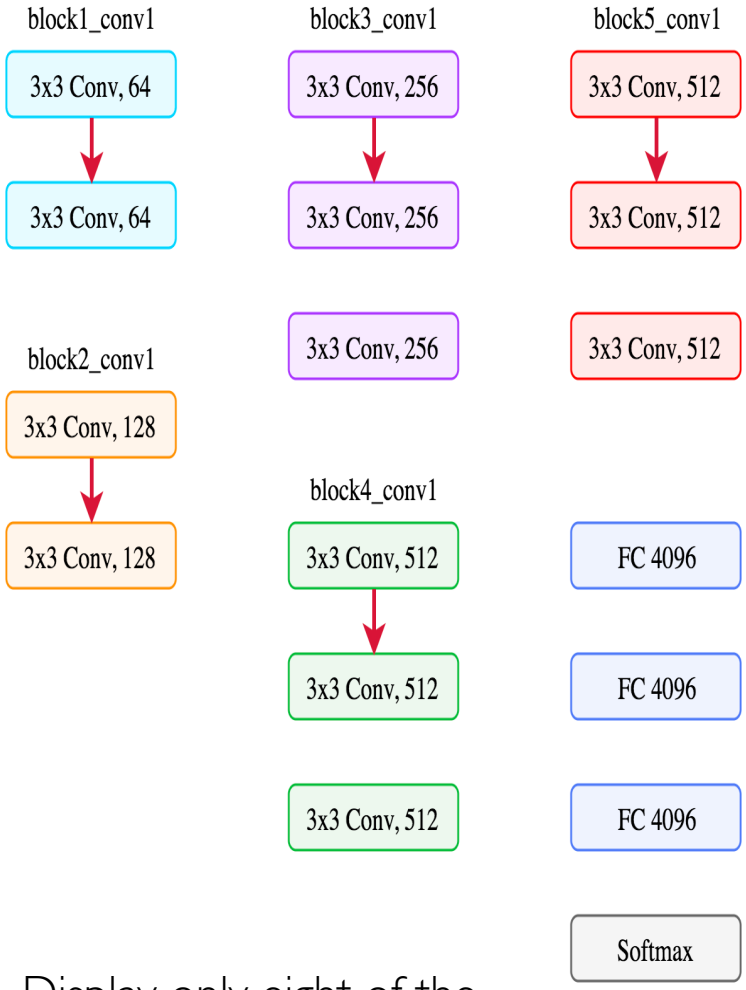
Noise

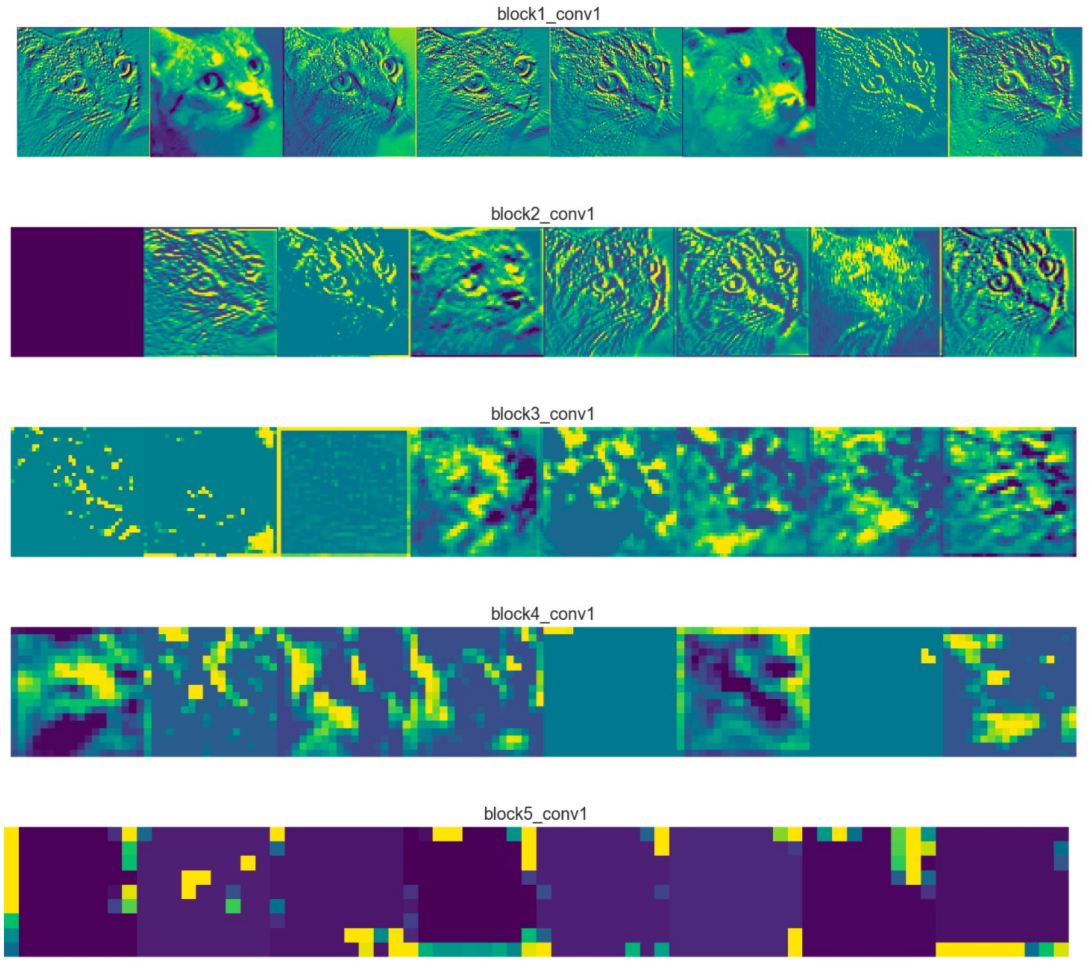







Feature Visualization

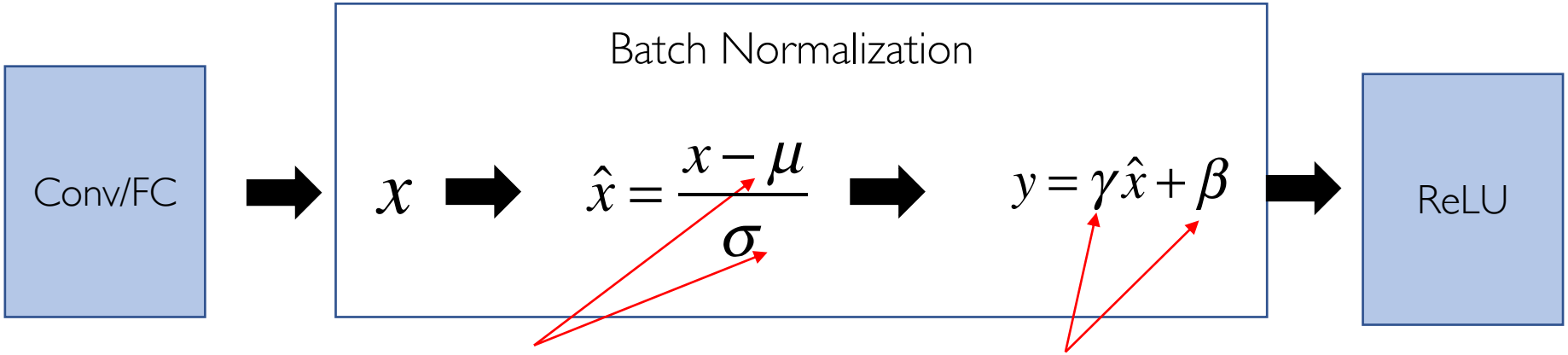




Display only eight of the 64/128/256/512 feature maps

https://github.com/utkuozbulak/pytorch-cnn-visualizations

Batch Normalization (BN)



after FC or Conv layers

Mean and std of x in each running mini-batch

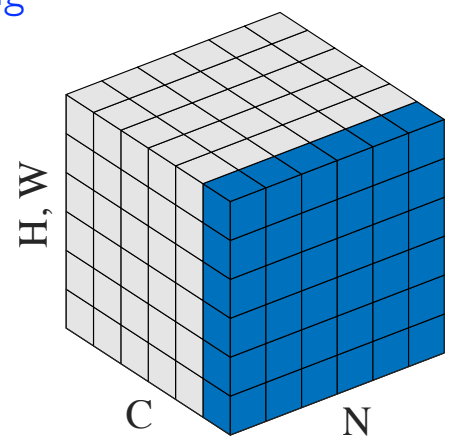
Parameters to be learned

If set $\gamma = \sqrt{\mathrm{Var}(x)}, \beta = \mathrm{E}(x)$

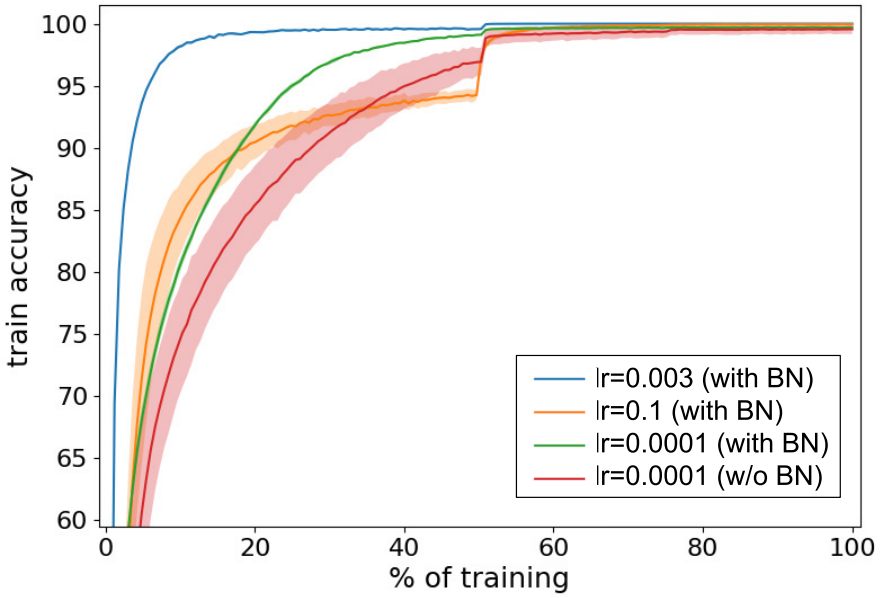
→ identity mapping

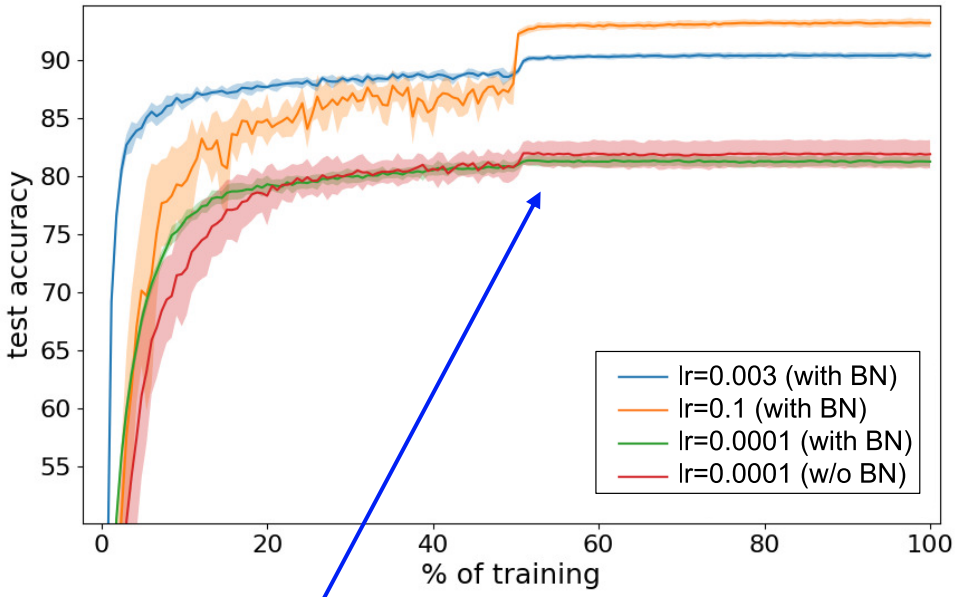
before the nonlinearity

- Normalize over mini-batch examples and nodes
- For a mini-batch of size n and feature maps of size w*h, the n*w*h elements are normalized together, and **each feature map** has a pair of **γ** and **β**
- At inference, use exponential moving average (EMA)



Batch Normalization (BN)





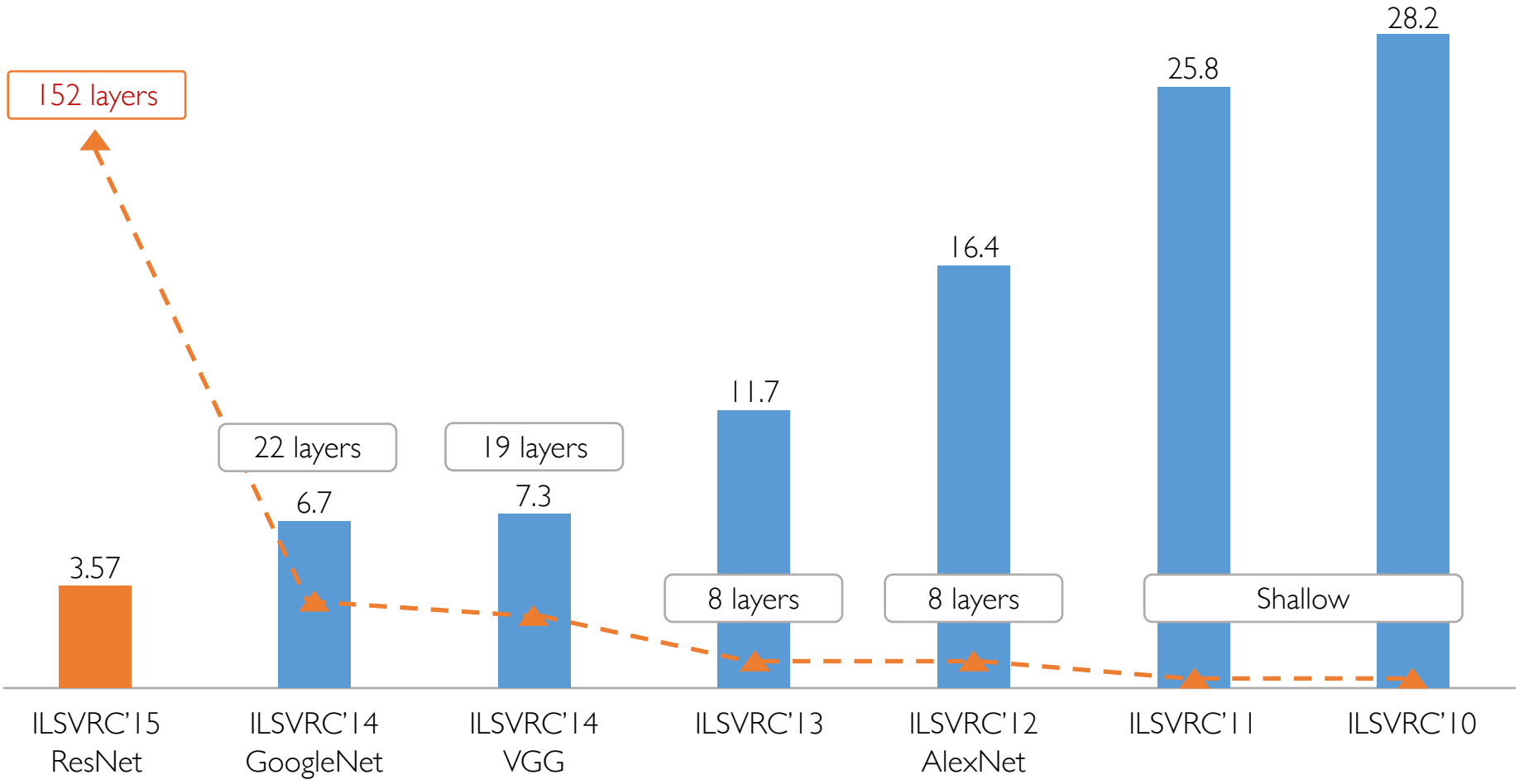
- The figure shows that with comparable learning rates result in **comparable testing accuracies** with and w/o BN.
- Larger learning rates yield **higher test accuracy** for BN networks, and **diverge** for unnormalized networks (not shown).

Bjorck, Nils, et al. Understanding batch normalization. **NIPS** 2018.

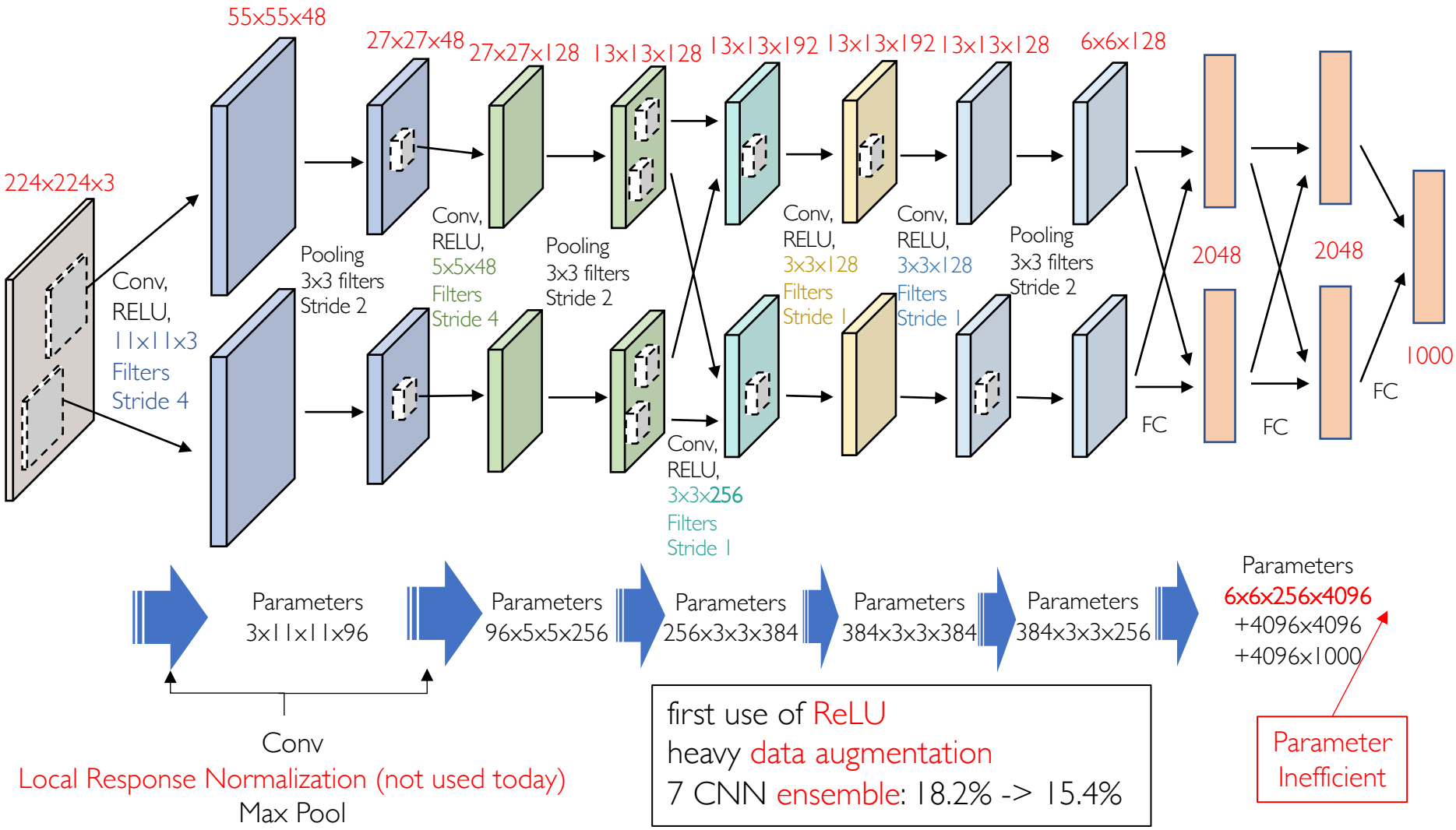
Outline

- Deep Learning
- Multiplayer Perceptrons (MLP)
  - Backpropagation
  - Training Strategies
- Convolutional Neural Network (CNN)
  - Training Strategies
  - **Standard Architectures**

ImageNet ILSVRC

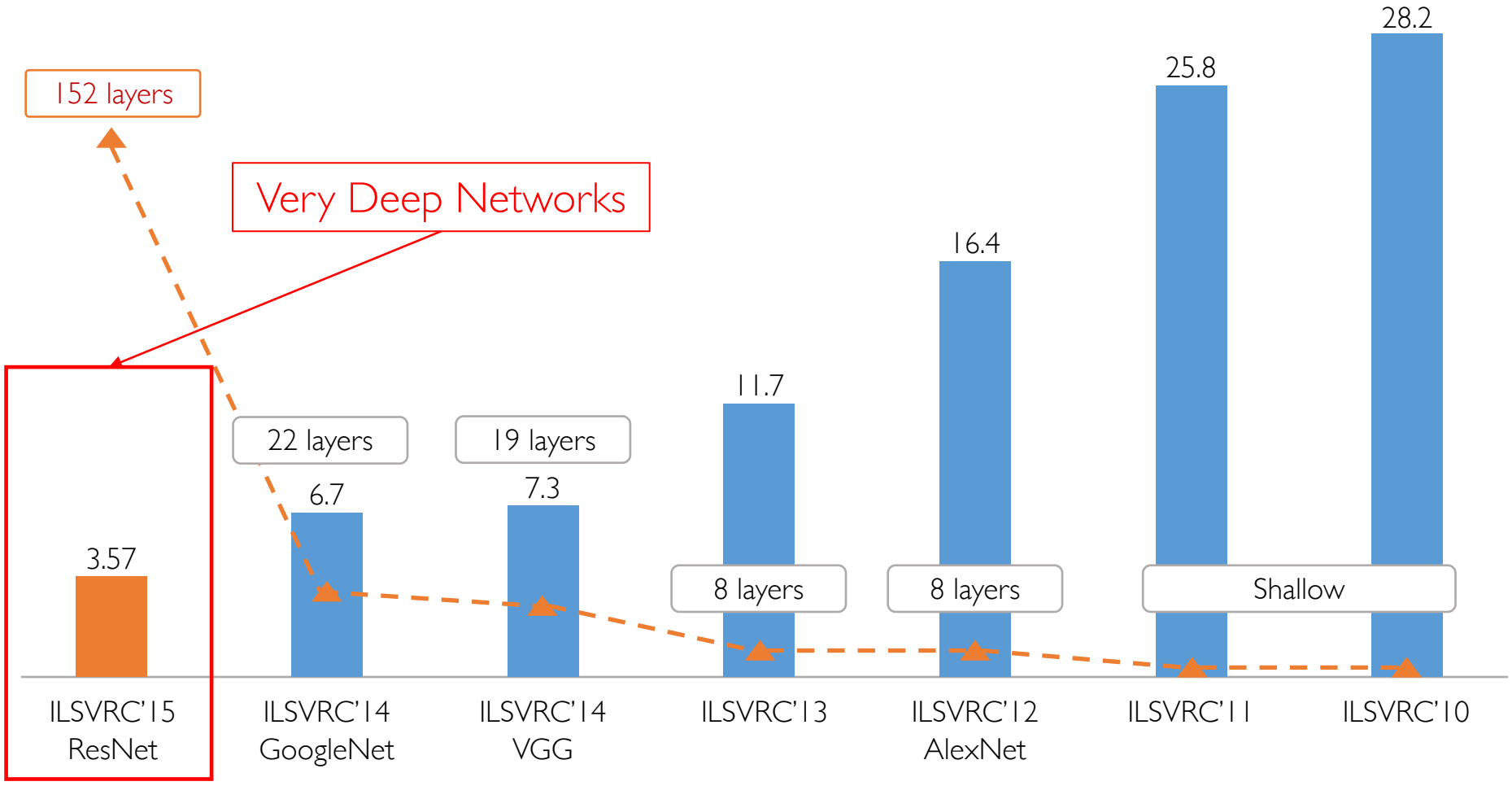


AlexNet



K., Alex, I. Sutskever, and G. E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. NIPS, 2012. (Cited 129154)

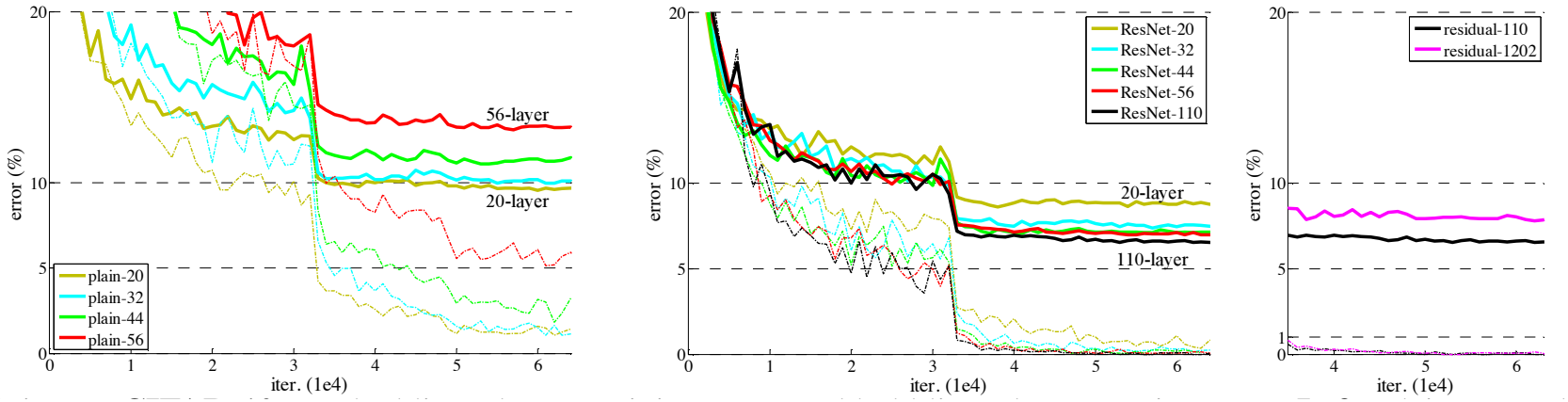
ImageNet ILSVRC



ImageNet Classification top-5 error (%)

Residual Network (ResNet)

- What happens when we continue stacking deeper layers on a plain convolutional neural network?

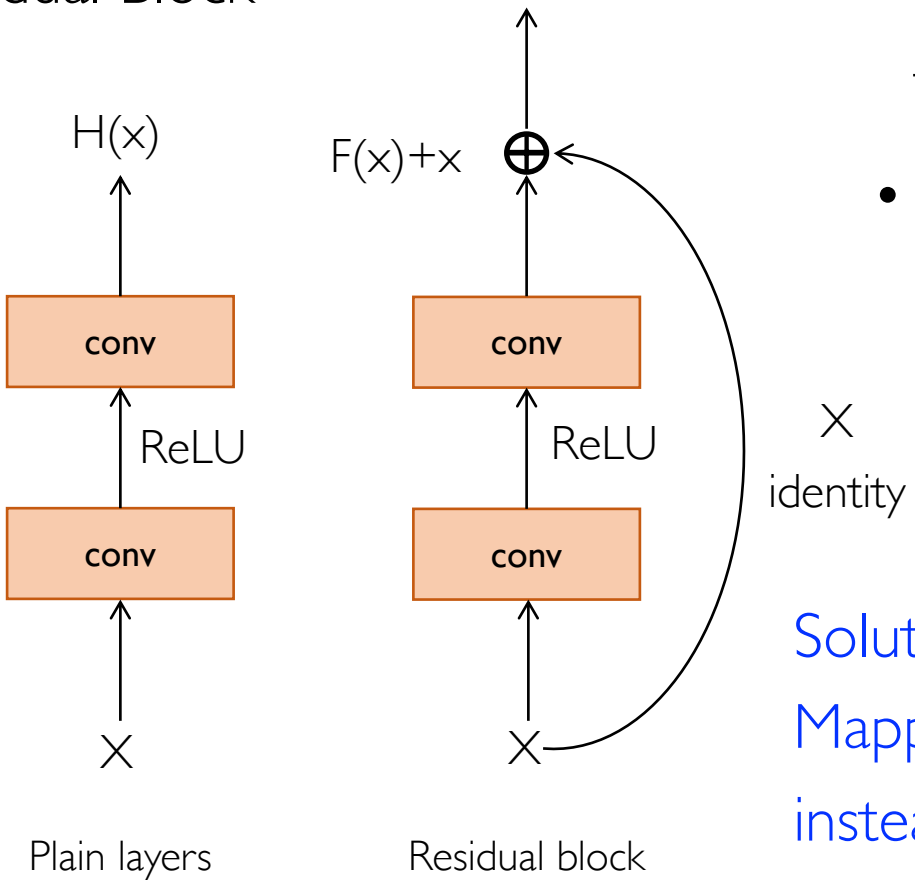


- 56-layer model performs worse on both training and test error
  - The deeper model performs worse, but not caused by overfitting!
- Deeper models are harder to optimize. It is an optimization issue.

K. He et al. Deep Residual Learning for Image Recognition. CVPR 2016. (Best Paper) (Cited 156992)

ResNet

- Residual Block



- Copying the learned layers from the shallower model
- Setting additional layers to identity mapping.

Solution:

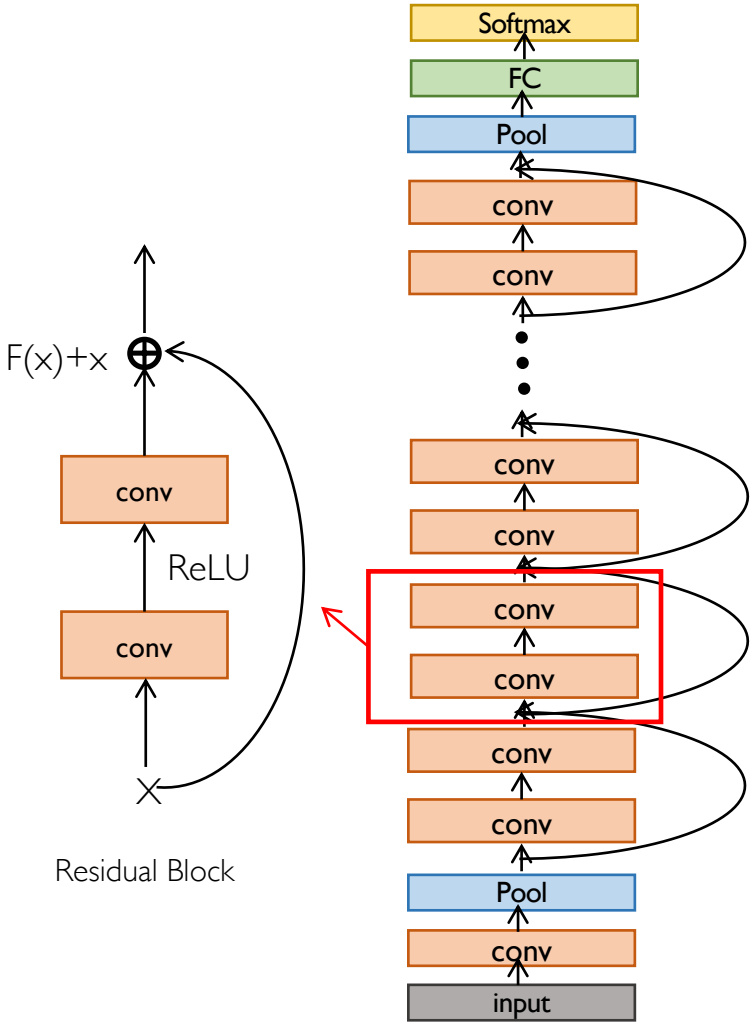
Mapping F(x) = H(x) – x,

instead of H(x) directly

K. He et al. Deep Residual Learning for Image Recognition. CVPR 2016. (Best Paper)

ResNet

- Full ResNet architecture:
  - Stack residual blocks
  - Every residual block has two 3x3 conv layers
  - Periodically, double #filters and downsample spatially using stride 2 (/2 in each dimension)
  - Additional conv layer at the beginning
  - Global average pooling at the end (only FC layer to output classes)

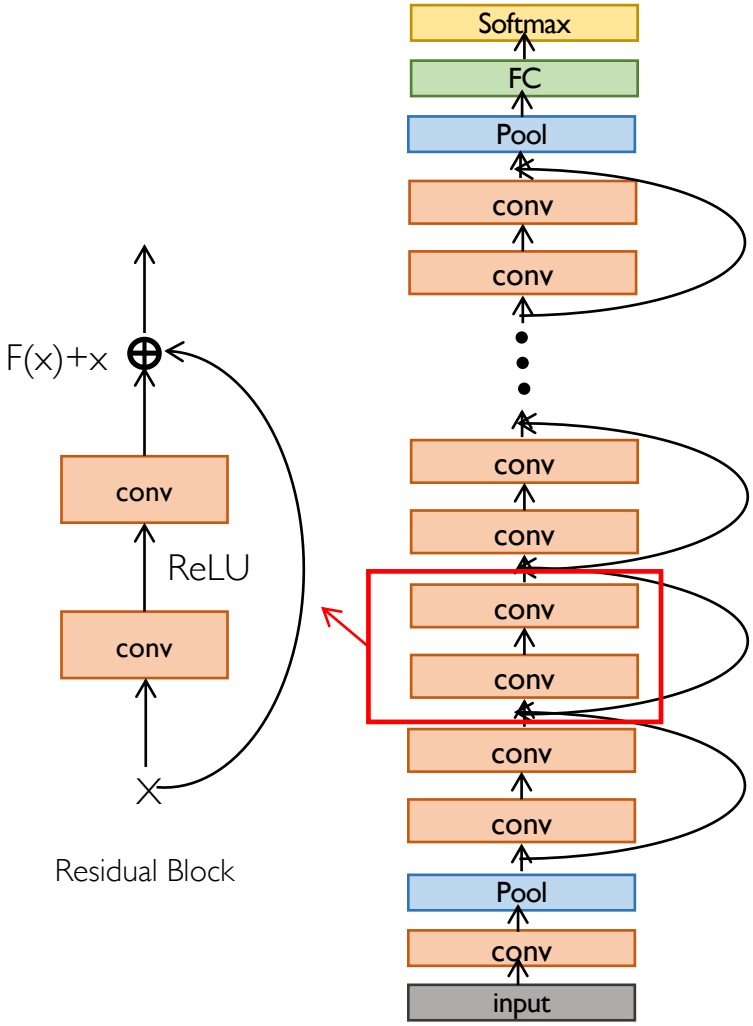


K. He et al. Deep Residual Learning for Image Recognition. CVPR 2016. (Best Paper)

ResNet

- Experimental Results
  - Able to train very deep networks without degrading (152 layers on ImageNet, 1202 on CIFAR)
  - Deeper networks now achieve lower training error as expected

- 1st place in all ILSVRC (3.6% top 5 error, better than human performance) and COCO 2015 competitions



K. He et al. Deep Residual Learning for Image Recognition. CVPR 2016. (Best Paper)

Thank You

Questions?

Mingsheng Long
mingsheng@tsinghua.edu.cn
http://ise.thss.tsinghua.edu.cn/~mlong

答疑：东主楼11区413室