Convolutional Neural Networks II

CS 4391 Introduction Computer Vision

Professor Yu Xiang

The University of Texas at Dallas

Supervised Learning

$\mathbf{x}$ $\Rightarrow$  $\Rightarrow$ $\mathbf{y}$

Image

Object class

$$f(\mathbf{x})$$

Training Data $\{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^{N}$

Input

Output

Convolutional Neural Networks

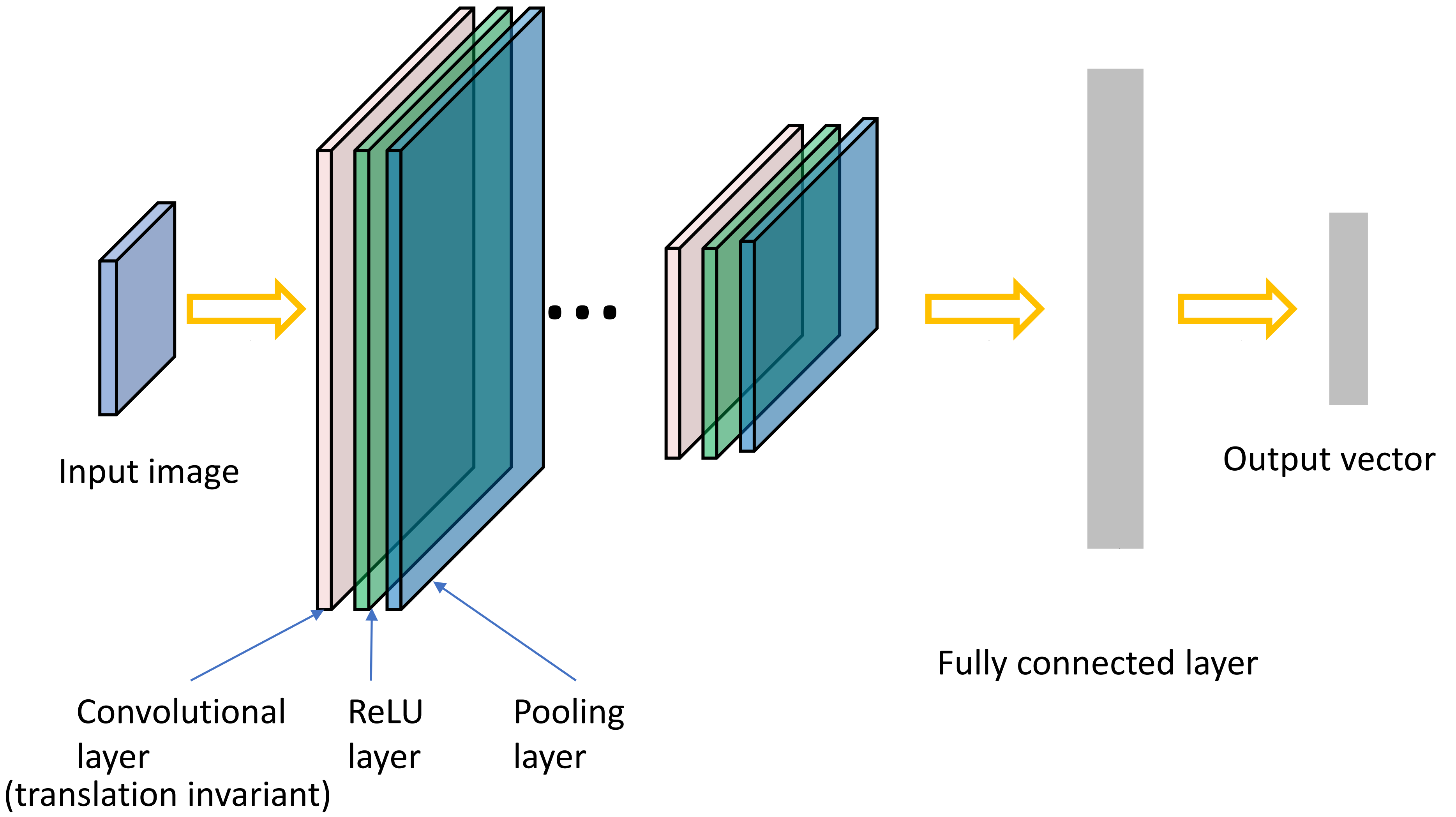


Image Classification

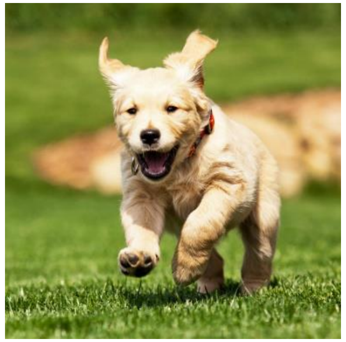
- ImageNet dataset
  - Training: 1.2 million images
  - Testing and validation: 150,000 images
  - 1000 categories

n02119789: kit fox, Vulpes macrotis
n02100735: English setter
n02096294: Australian terrier
n02066245: grey whale, gray whale, devilfish, Eschrichtius gibbosus, Eschrichtius robustus
n02509815: lesser panda, red panda, panda, bear cat, cat bear, Ailurus fulgens
n02124075: Egyptian cat
n02417914: ibex, Capra ibex
n02123394: Persian cat
n02125311: cougar, puma, catamount, mountain lion, painter, panther, Felis concolor
n02423022: gazelle

https://image-net.org/challenges/LSVRC/2012/index.php

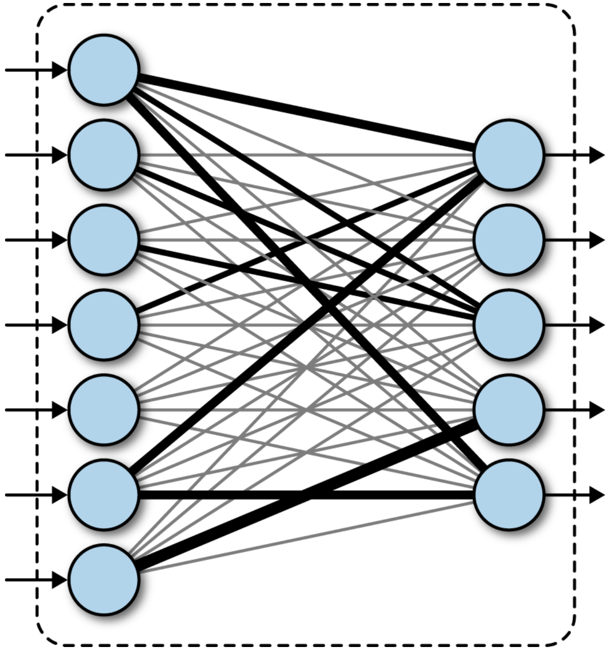
Image Classification

Let's consider only using one FC layer



$224 \times 224 \times 3$

$W : m \times n$



$x : n \times 1$ 150528

$y : m \times 1$ 1000

$$\mathbf{y} = W\mathbf{x}$$

$\sigma(\mathbf{y})$ Probability distribution

Softmax function

$$\sigma(\mathbf{y})_i = \frac{e^{y_i}}{\sum_j^m e^{y_i}}$$

Image Classification

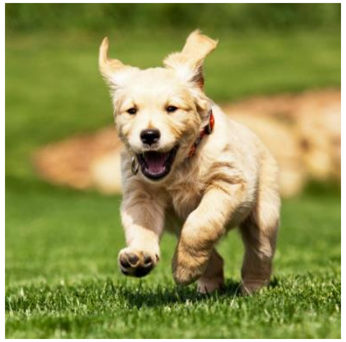
- Training data $\{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^{N}$

Image label

- One-hot vector $\mathbf{y}_i = 000\ldots1\ldots000$

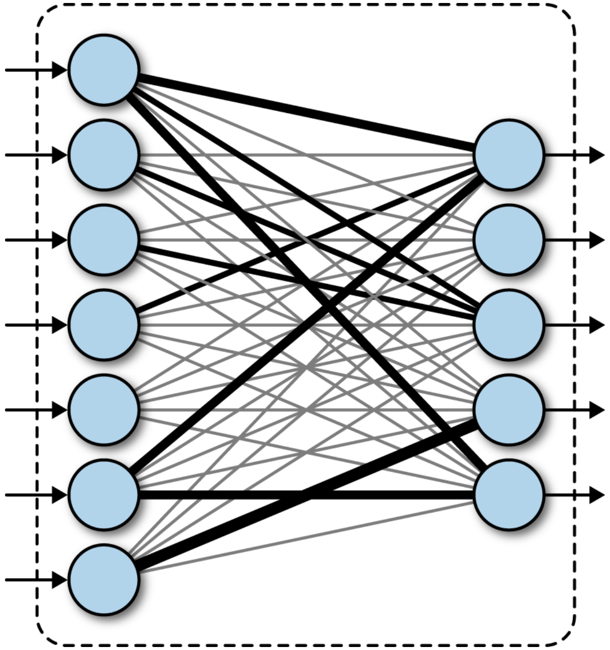
Ground truth category

Image Classification



$224 \times 224 \times 3$

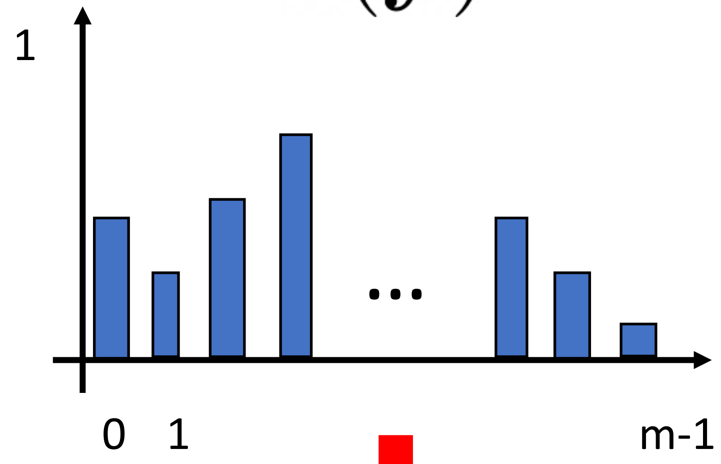
$W : m \times n$



$x : n \times 1$ $\quad$ $y : m \times 1$

$$\mathbf{y} = W\mathbf{x}$$

$\sigma(\mathbf{y})$



The goal of training

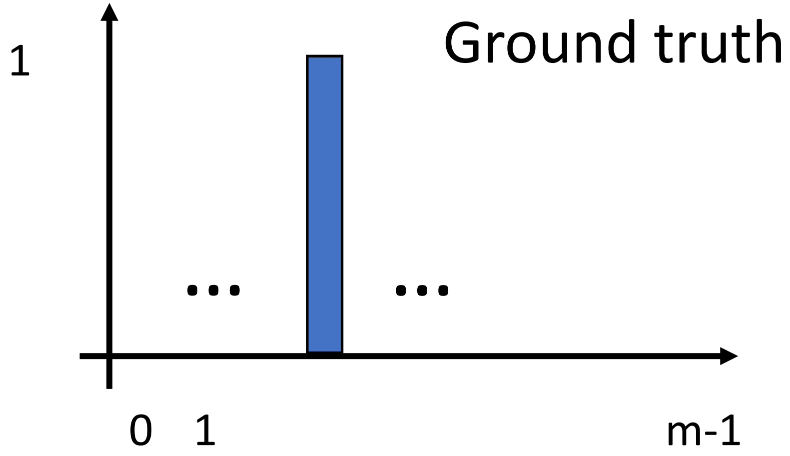


Image Classification

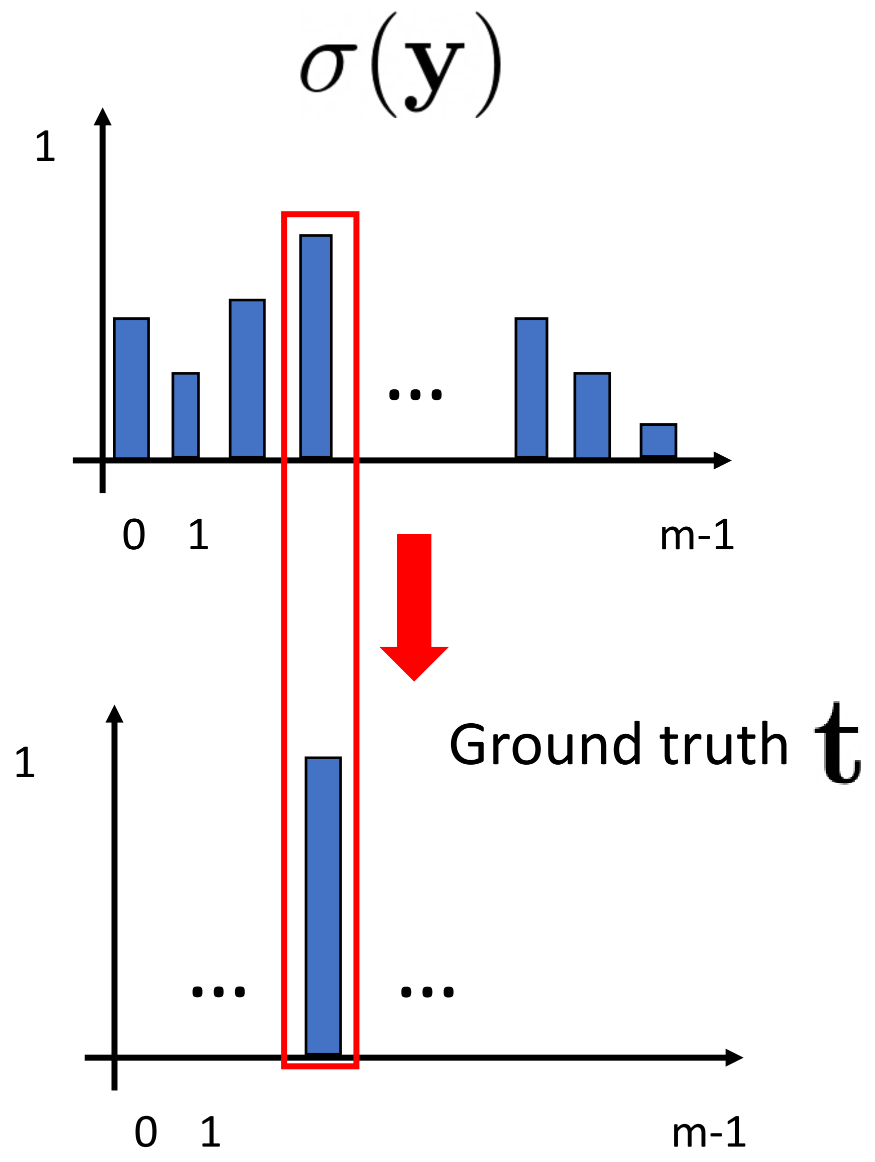
- Cross entropy loss function

Cross entropy between two distributions
(measure distance between distributions)

$$H(p,q) = -\mathrm{E}_p[\log q]$$

$$H(p,q) = -\sum_{x\in\mathcal{X}} p(x)\,\log q(x)$$

$$L_{CE} = -\sum_{i=0}^{m-1} t_i \log \sigma(\mathbf{y})_i$$



Training

- Cross entropy loss function

$$\mathbf{y} = W\mathbf{x}$$

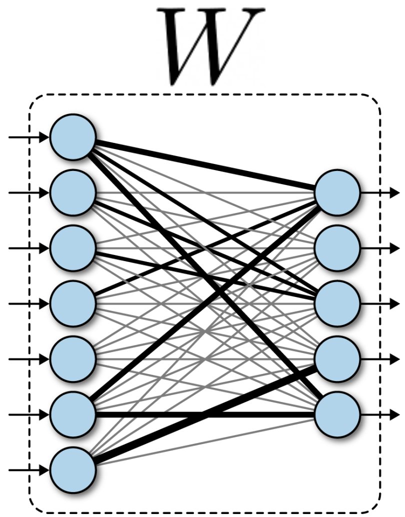
Minimize

$$L_{CE} = -\sum_{i=0}^{m-1} t_i \log \sigma(\mathbf{y})_i$$

$$\sigma(\mathbf{y})_i = \frac{e^{y_i}}{\sum_j^m e^{y_i}}$$

With respect to weights W

$\mathbf{x}$



$$\mathbf{y} \longrightarrow \sigma(\mathbf{y}) \longrightarrow L_{CE}$$

Training

- Gradient descent

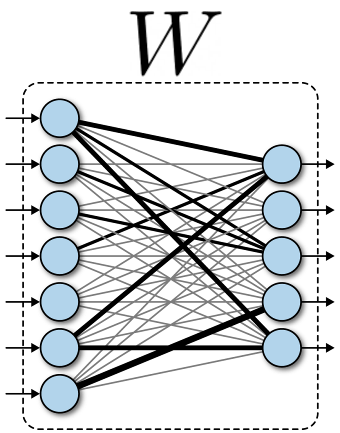
$$W \leftarrow W - \gamma \frac{\partial L}{\partial W}$$

Learning rate

- Chain rule

$$\frac{\partial L}{\partial W} = \frac{\partial L}{\partial \sigma(\mathbf{y})} \frac{\partial \sigma(\mathbf{y})}{\partial \mathbf{y}} \frac{\partial \mathbf{y}}{\partial W}$$

$\mathbf{x}$



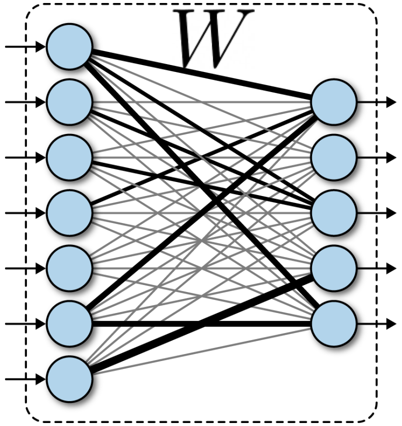
$$\mathbf{y} \rightarrow \sigma(\mathbf{y}) \rightarrow L_{CE}$$

Training

- Gradient descent

$$L_{CE} = -\sum_{i=0}^{m-1} t_i \log \sigma(\mathbf{y})_i = -\mathbf{t} \cdot \log \sigma(\mathbf{y})$$

$$\mathbf{x}$$



$$\mathbf{y} \longrightarrow \sigma(\mathbf{y}) \longrightarrow L_{CE}$$

How to compute gradient?

$$\frac{\partial L}{\partial \mathbf{y}} \quad \begin{bmatrix} \frac{\partial L}{y_1} & \frac{\partial L}{y_2} & \cdots & \frac{\partial L}{y_m} \end{bmatrix}$$

$$1 \times m$$

Training

$$L_{CE} = -\sum_{i=0}^{m-1} t_i \log \sigma(\mathbf{y})_i = -\mathbf{t} \cdot \log \sigma(\mathbf{y})$$

$$\sigma(\mathbf{y})_i = \frac{e^{y_i}}{\sum_j^m e^{y_i}}$$

- Chain rule

$$\frac{\partial L}{\partial \mathbf{y}} = \frac{\partial L}{\partial \sigma(\mathbf{y})} \cdot \frac{\partial \sigma(\mathbf{y})}{\partial \mathbf{y}}$$

$$1 \times m \qquad 1 \times m \qquad m \times m$$

$$\frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \begin{bmatrix} \nabla f_1(\mathbf{x}) \\ \nabla f_2(\mathbf{x}) \\ \cdots \\ \nabla f_m(\mathbf{x}) \end{bmatrix} = \begin{bmatrix} \frac{\partial}{\partial \mathbf{x}} f_1(\mathbf{x}) \\ \frac{\partial}{\partial \mathbf{x}} f_2(\mathbf{x}) \\ \cdots \\ \frac{\partial}{\partial \mathbf{x}} f_m(\mathbf{x}) \end{bmatrix} = \begin{bmatrix} \frac{\partial}{\partial x_1} f_1(\mathbf{x}) & \frac{\partial}{\partial x_2} f_1(\mathbf{x}) & \ldots & \frac{\partial}{\partial x_n} f_1(\mathbf{x}) \\ \frac{\partial}{\partial x_1} f_2(\mathbf{x}) & \frac{\partial}{\partial x_2} f_2(\mathbf{x}) & \ldots & \frac{\partial}{\partial x_n} f_2(\mathbf{x}) \\ & & \cdots & \\ \frac{\partial}{\partial x_1} f_m(\mathbf{x}) & \frac{\partial}{\partial x_2} f_m(\mathbf{x}) & \ldots & \frac{\partial}{\partial x_n} f_m(\mathbf{x}) \end{bmatrix}$$

Jacobian matrix

$$\frac{\partial L}{\partial \sigma(\mathbf{y})} = -\mathbf{t} \cdot \frac{1}{\sigma(\mathbf{y})}$$

$$\frac{\partial \sigma(\mathbf{y})_i}{\partial y_j} = \sigma(\mathbf{y})_i(\delta_{ij} - \sigma(\mathbf{y})_j) \qquad \delta_{ij} = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases}$$

https://eli.thegreenplace.net/2016/the-softmax-function-and-its-derivative/

Training

- Gradient descent

$$L_{CE} = -\sum_{i=0}^{m-1} t_i \log \sigma(\mathbf{y})_i = -\mathbf{t} \cdot \log \sigma(\mathbf{y})$$

$$\mathbf{y} = W\mathbf{x}$$

$$\frac{\partial L}{\partial W} = \frac{\partial L}{\partial \sigma(\mathbf{y})} \frac{\partial \sigma(\mathbf{y})}{\partial \mathbf{y}} \frac{\partial \mathbf{y}}{\partial W}$$

$$\frac{\partial L}{\partial \sigma(\mathbf{y})} = -\mathbf{t} \cdot \frac{1}{\sigma(\mathbf{y})}$$

$$\frac{\partial \sigma(\mathbf{y})_i}{\partial y_j} = \sigma(\mathbf{y})_i(\delta_{ij} - \sigma(\mathbf{y})_j)$$

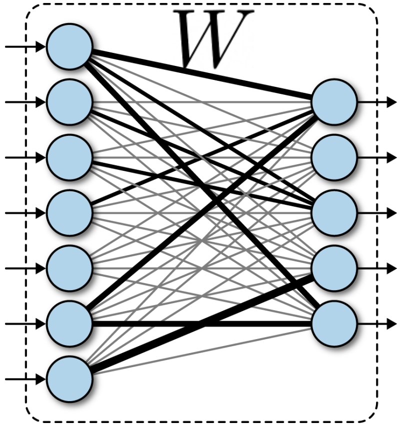
$$\delta_{ij} = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases}$$

$$\frac{\partial y_i}{\partial W_{jk}} = \begin{cases} 0 & \text{if } i \neq j \\ x_k & \text{otherwise} \end{cases}$$

$$W \leftarrow W - \gamma \frac{\partial L}{\partial W}$$

Learning rate

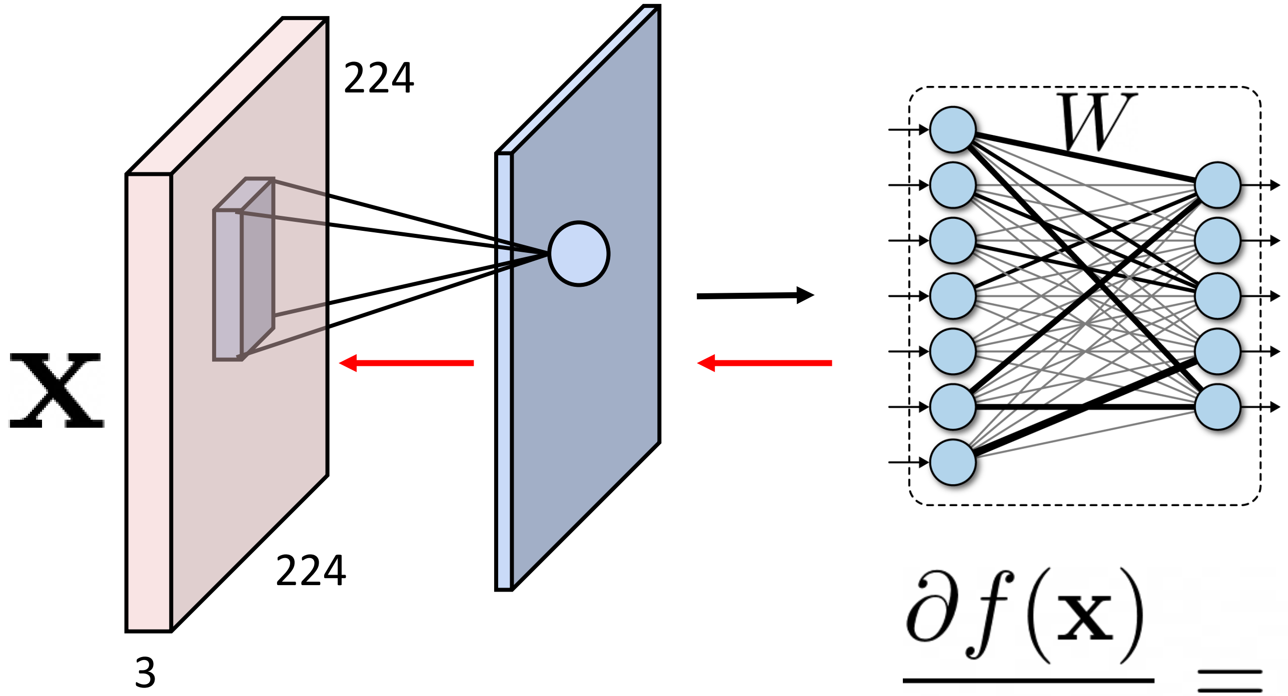
Back-propagation

$\mathbf{x}$  $\mathbf{y} \longrightarrow \sigma(\mathbf{y}) \longrightarrow L$

$$\frac{\partial \mathbf{y}}{\partial W} \qquad \frac{\partial \sigma(\mathbf{y})}{\partial \mathbf{y}} \qquad \frac{\partial L}{\partial \sigma(\mathbf{y})}$$

$$\frac{\partial L}{\partial W} = \frac{\partial L}{\partial \sigma(\mathbf{y})} \frac{\partial \sigma(\mathbf{y})}{\partial \mathbf{y}} \frac{\partial \mathbf{y}}{\partial W}$$

Back-propagation



$$\mathbf{y} \rightarrow \sigma(\mathbf{y}) \rightarrow L$$

$$f(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b$$

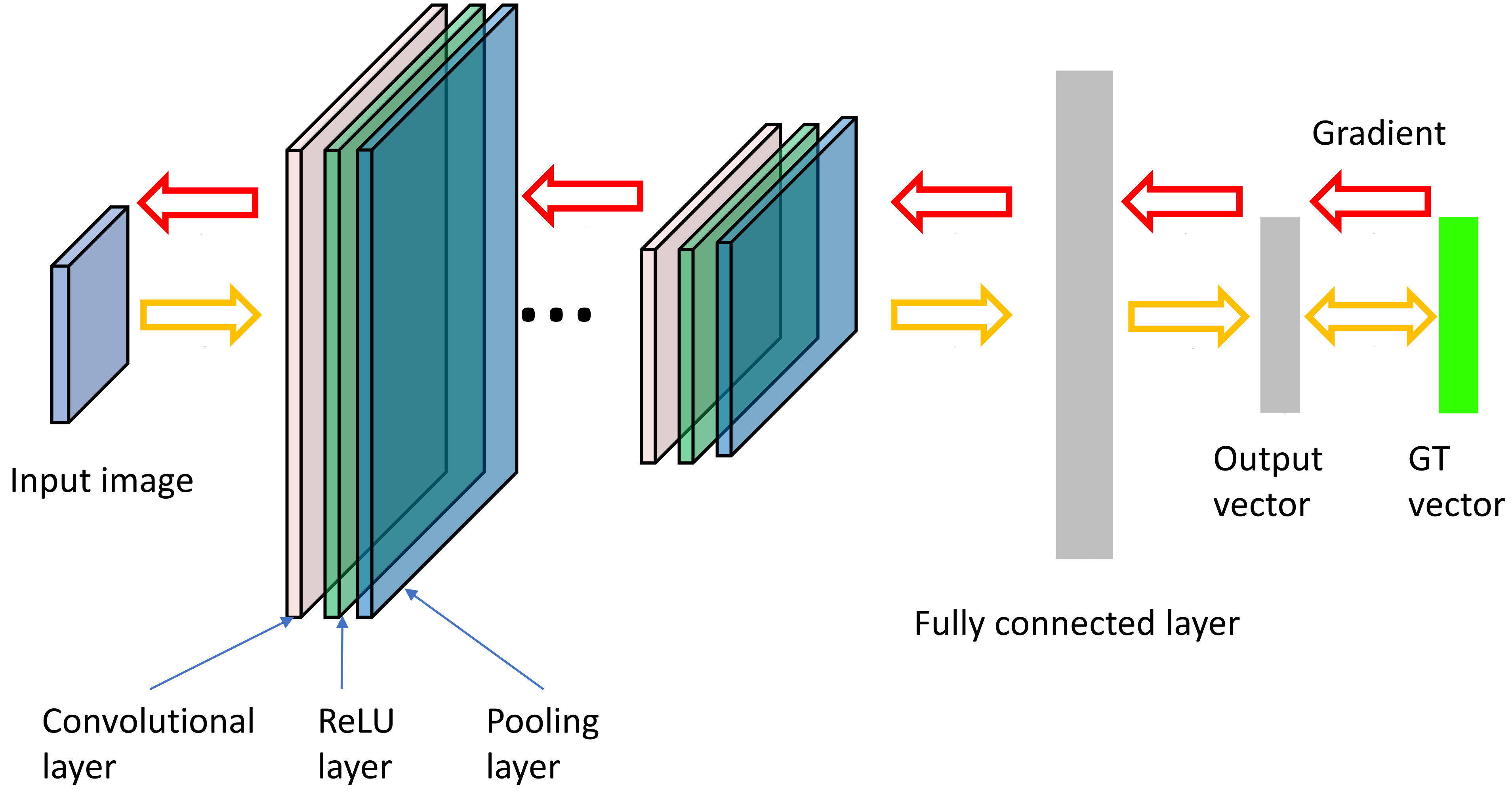
Conv filter

Bias

$$\frac{\partial f(\mathbf{x})}{\partial \mathbf{w}} = \mathbf{x}^T \qquad \frac{\partial f(\mathbf{x})}{\partial b} = 1$$

$$\frac{\partial L}{\partial \mathbf{w}} = \sum_i \frac{\partial L}{\partial f(\mathbf{x}_i)} \frac{\partial f(\mathbf{x}_i)}{\partial \mathbf{w}}$$

Training: back-propagate errors



Back-propagation

- For each layer in the network, compute local gradients (partial derivative)
  - Fully connected layers
  - Convolution layers
  - Activation functions
  - Pooling functions
  - Etc.

- Use chain rule to combine local gradients for training

$$\frac{\partial L}{\partial W} = \frac{\partial L}{\partial \sigma(\mathbf{y})} \frac{\partial \sigma(\mathbf{y})}{\partial \mathbf{y}} \frac{\partial \mathbf{y}}{\partial W}$$

Further Reading

- Stanford CS231n, lecture 3 and lecture 4, http://cs231n.stanford.edu/schedule.html
- Deep learning with PyTorch https://pytorch.org/tutorials/beginner/deep_learning_60min_blitz.html
- Dropout: A Simple Way to Prevent Neural Networks from Overfitting https://jmlr.org/papers/v15/srivastava14a.html
- Matrix Calculus: https://explained.ai/matrix-calculus/