THE UNIVERSITY OF TEXAS AT DALLAS

Convolutional Neural Networks II

CS6384 Computer Vision

Professor Yapeng Tian

Department of Computer Science

Slides borrowed from Professor Yu Xiang

Midterm Exam

Date and Time: 03/06/25 **(Thursday) 11:30am-12:45pm (75 mins)**

Location: **TI Auditorium, ECSS 2.102**

Topics: first **12** lectures

Question Types: multi-choice, short answer, and long answer questions

Policy

- **Closed-book test**. But you are allowed **one A4 page (single page)** of handwritten notes
- No calculators, cell phones, or any kind of internet connection are allowed
- Talking and discussion are prohibited
- Please space yourselves so that students are evenly distributed throughout the room. There should be no one directly next to you

Supervised Learning

$\mathbf{x}$

Image

$\mathbf{y}$

Object class

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

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

Input

Output

Convolutional Neural Networks

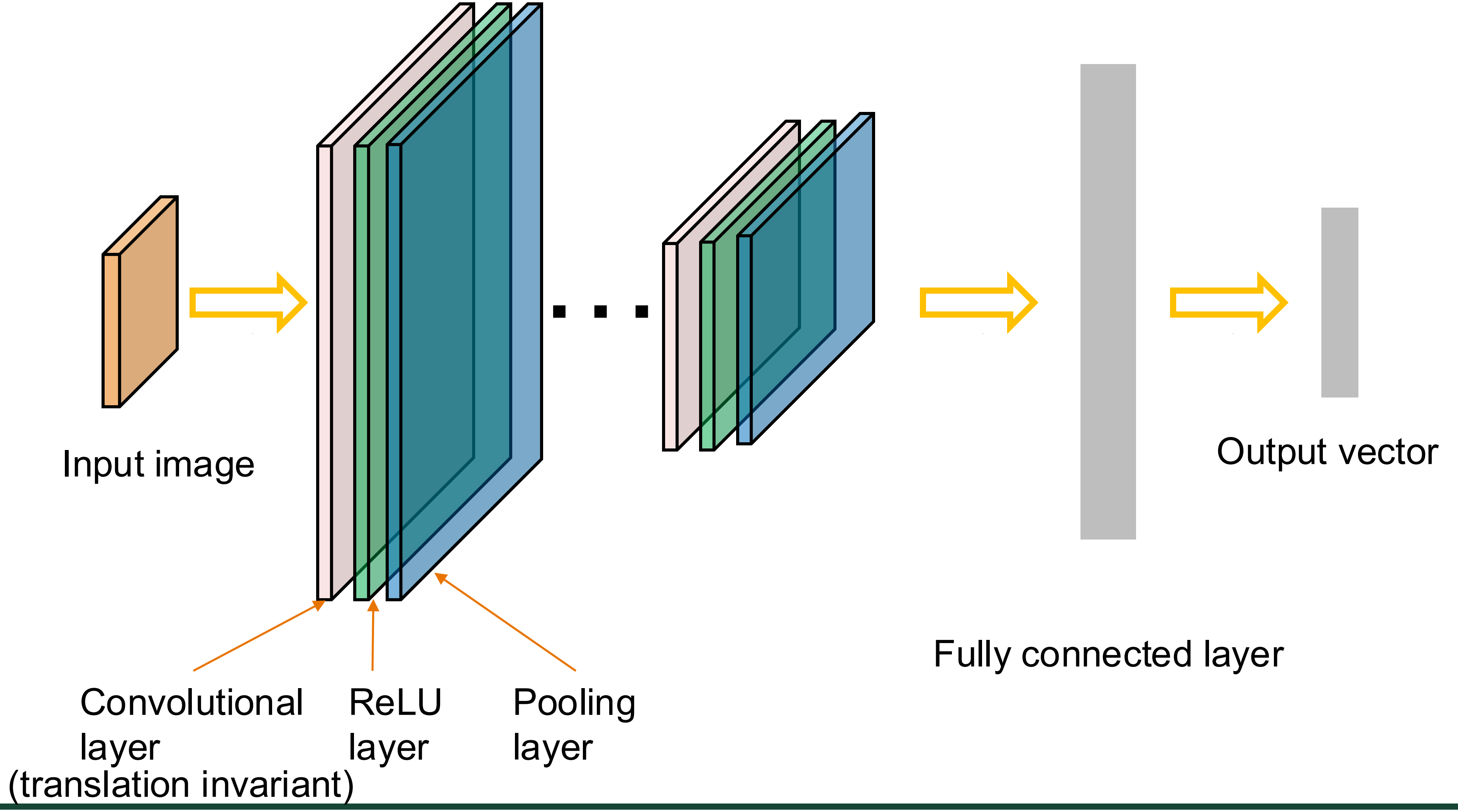


Image Classification

ImageNet dataset

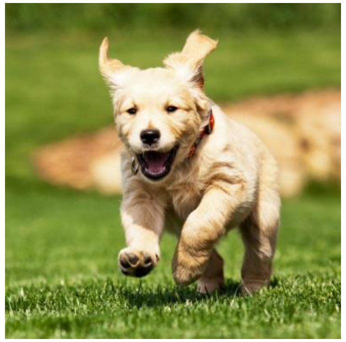
- Training: 1.2 million images
- Testing and validation: 150,000 images
- 1000 categories

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

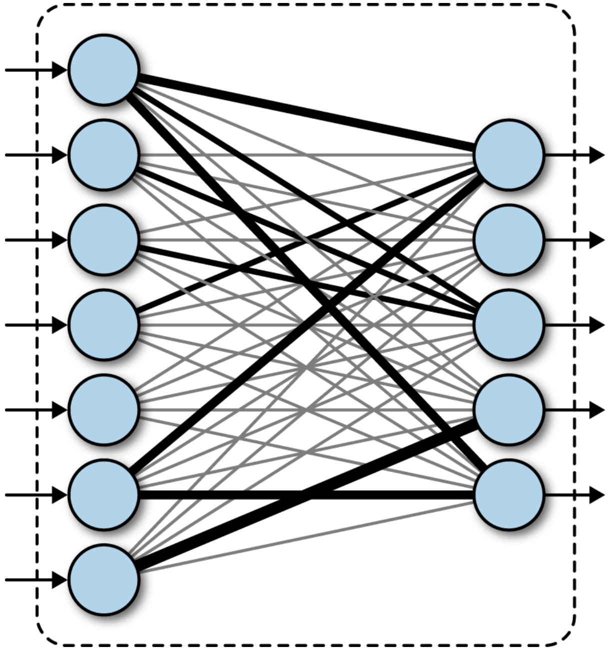
Image Classification

Let's consider only using one FC layer



$224 \times 224 \times 3$

$W : m \times n$



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

150528 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 $\{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^{N}$

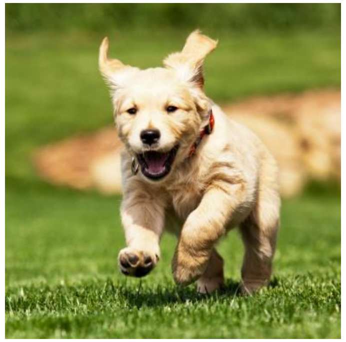
Image label

One-hot vector: if an object in k-th class exists in the image, its label will be encoded as [0, 0, 0, …, 1, …, 0, 0, 0], where only k-th element in the vector is 1

$$\mathbf{y}_i = 000\ldots1\ldots000$$

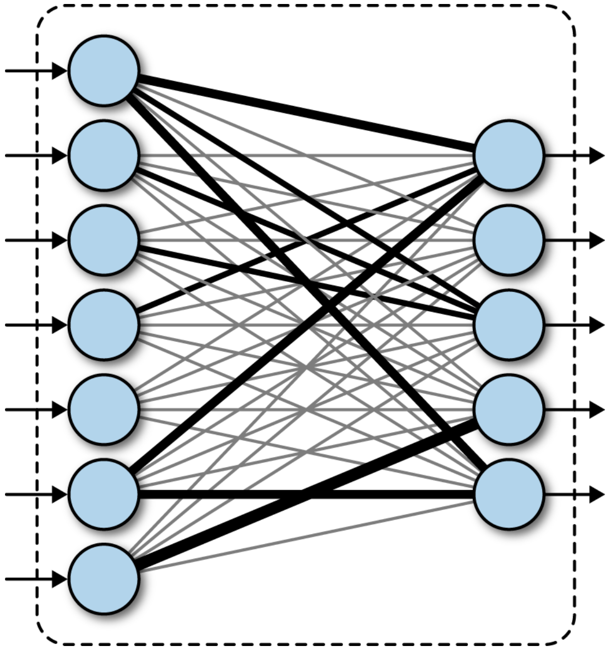
Ground truth category

Image Classification



$224 \times 224 \times 3$

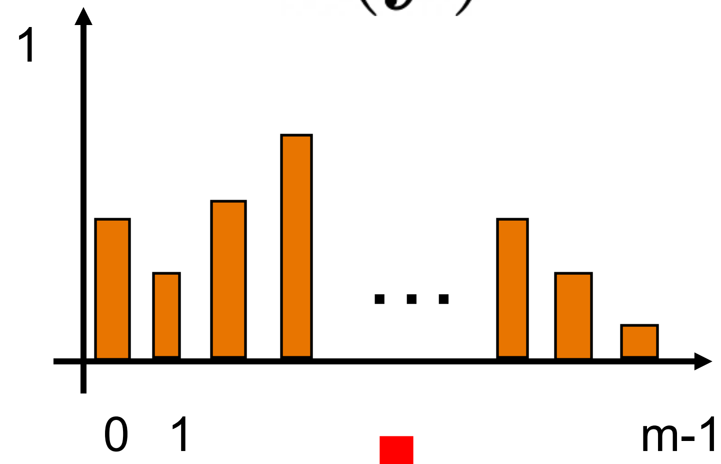
$W : m \times n$



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

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

$\sigma(\mathbf{y})$



The goal of training

Ground truth

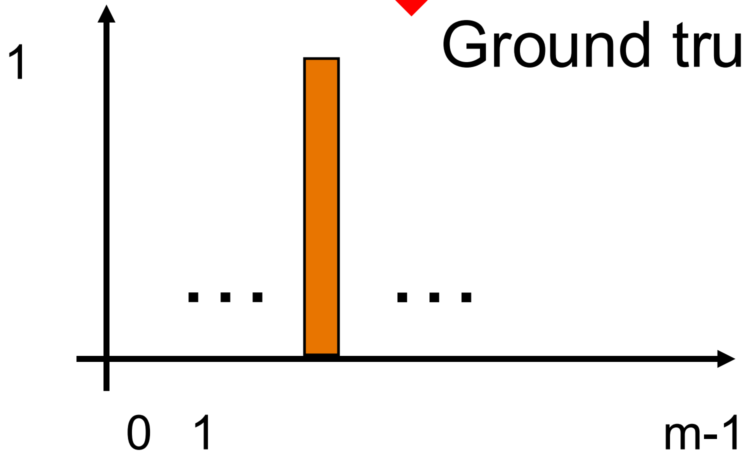


Image Classification

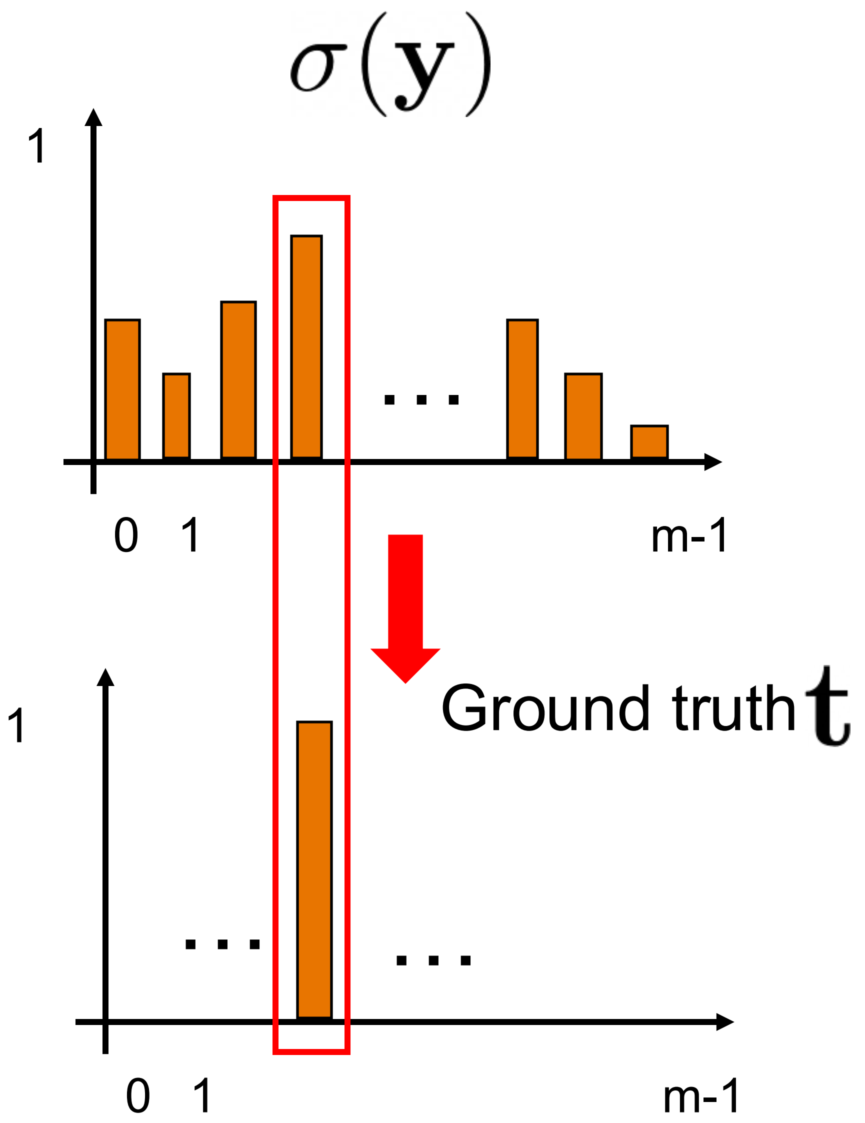
Cross entropy loss function

Cross entropy between two distributions
(measure distance between distributions)

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

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

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



Input pixels, $\mathbf{x}$

Forward propagation →

Shape: (3, 32, 32)

Feedforward output, $\mathbf{y}_i$

cat	dog	horse
5	4	2
4	2	8
4	4	1

Shape: (3,)

Softmax function →

Softmax output, $\sigma(\mathbf{y})_i$

cat	dog	horse
0.71	0.26	0.04
0.02	0.00	0.98
0.49	0.49	0.02

Shape: (3,)

https://ljvmiranda921.github.io/notebook/2017/08/13/softmax-and-the-negative-log-likelihood/

Training

Cross entropy loss function

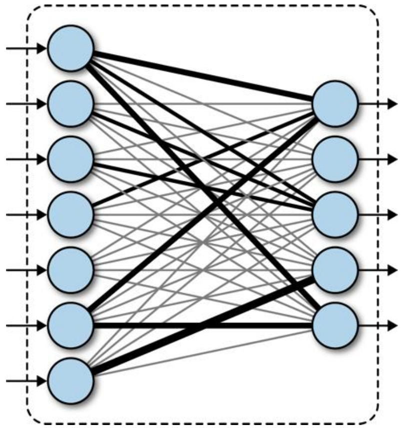
$$\mathbf{y} = W\mathbf{x}$$

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

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

With respect to weights W

W

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

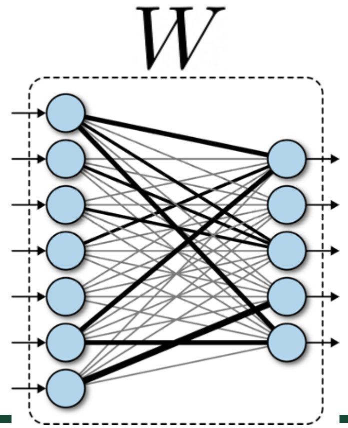
Training

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

Learning rate

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

$\mathbf{x}$



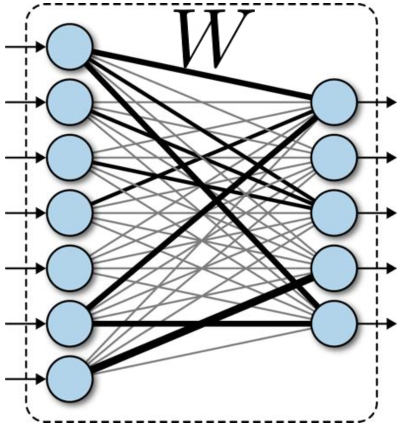
$$\mathbf{y} \longrightarrow \sigma(\mathbf{y}) \longrightarrow L_{CE}$$

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? $\dfrac{\partial L}{\partial \mathbf{y}}$

$$\begin{bmatrix} \dfrac{\partial L}{y_1} & \dfrac{\partial L}{y_2} & \dots & \dfrac{\partial L}{y_m} \end{bmatrix}$$

$$1 \times m$$

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

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

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

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

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

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

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

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

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

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

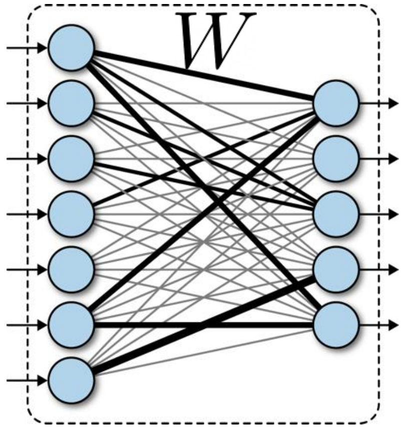
$$\delta_{ij} = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases}$$

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

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

Learning rate

Back-propagation

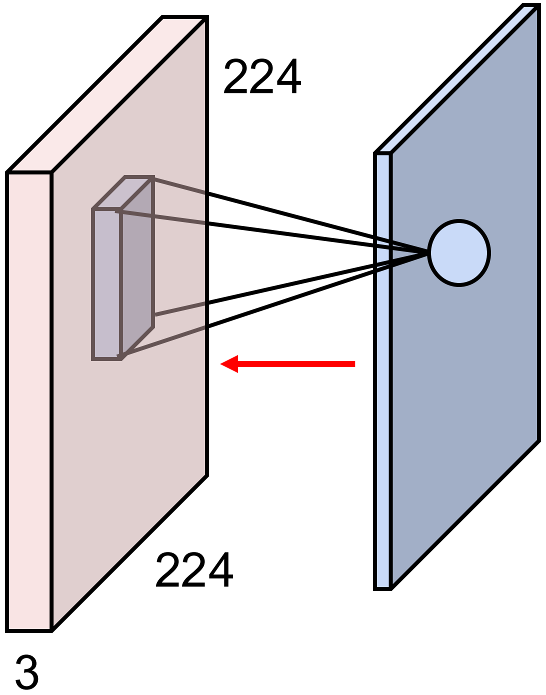


$$\mathbf{x} \quad \mathbf{y} \longrightarrow \sigma(\mathbf{y}) \longrightarrow L$$

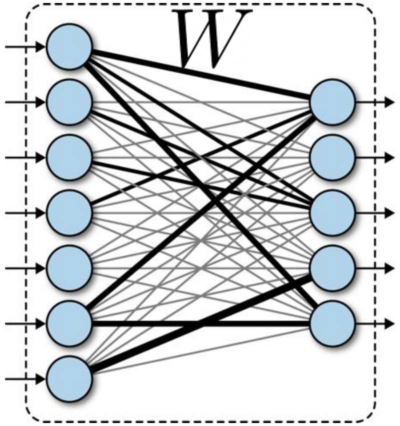
$$\frac{\partial \mathbf{y}}{\partial W} \qquad \frac{\partial \sigma(\mathbf{y})}{\partial \mathbf{y}} \qquad \frac{\partial L}{\partial \sigma(\mathbf{y})}$$

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

Back-propagation



$$\mathbf{x}$$



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

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

Conv filter

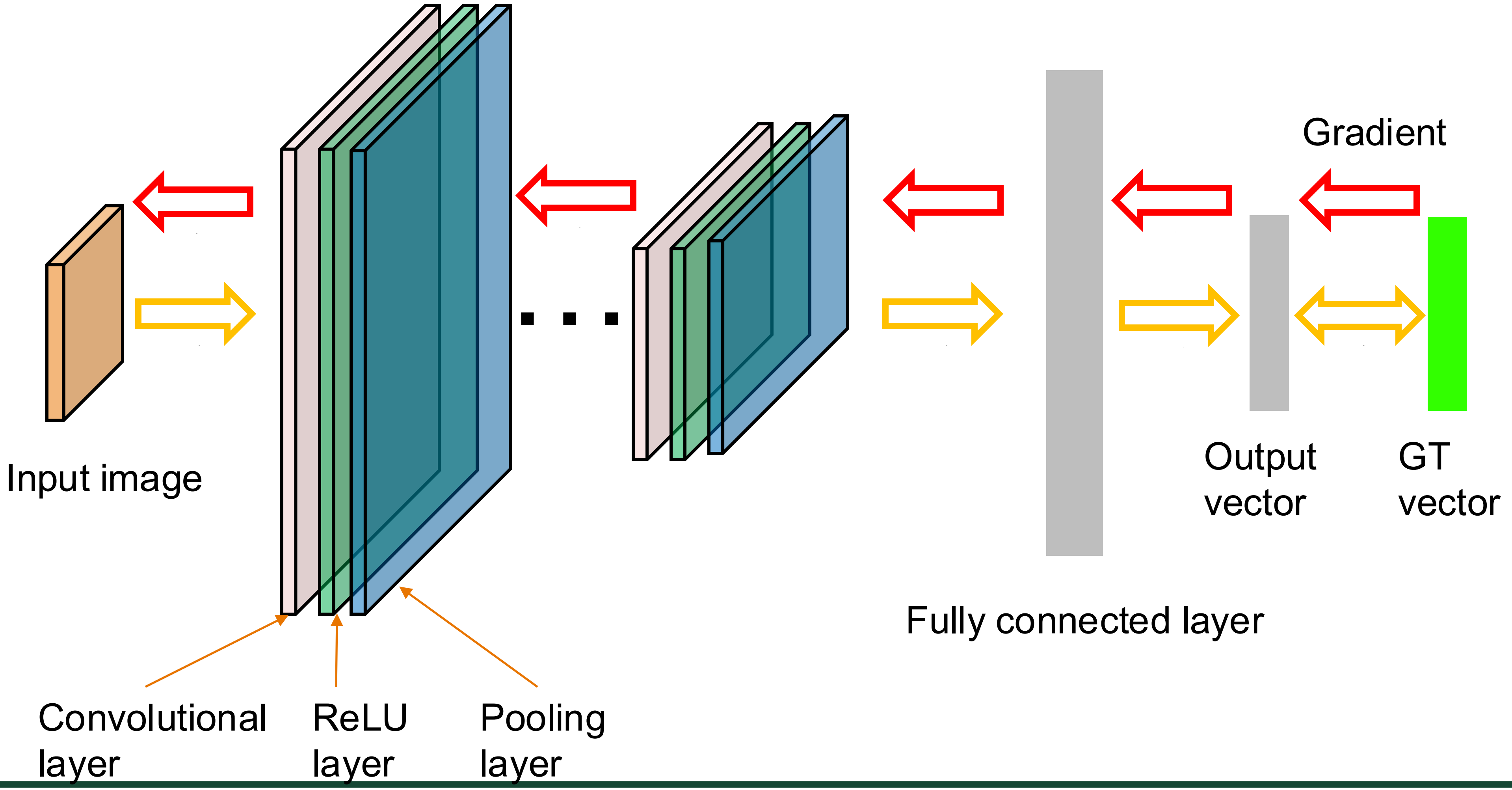
Bias

Simplified convolution computation

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

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

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

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

Classification Loss Functions

Cross entropy loss

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

Binary ground truth label

Logit

Hinge loss for binary classification

$$L = \max(0, 1 - t \cdot y)$$

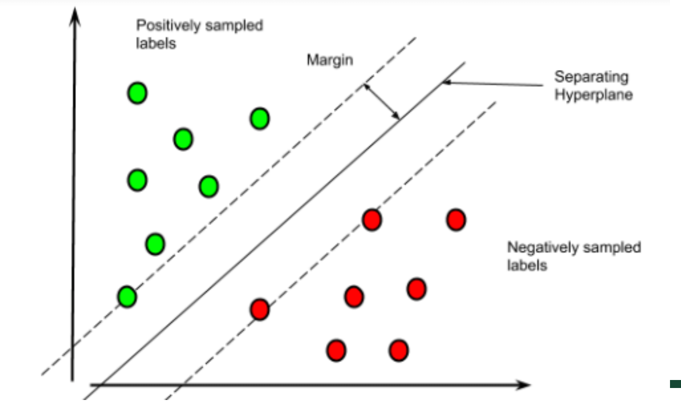
Classification score

ground truth label $t \in \{-1, +1\}$

$y \geq 0$ Predict positive

$y < 0$ Predict negative

Max margin classification



Classification Loss Functions

Hinge loss for multi-class classification

$$\ell(y) = \max(0, 1 + \max_{y \neq t} \mathbf{w}_y \mathbf{x} - \mathbf{w}_t \mathbf{x})$$

margin

Score corresponds to the most wrong label

Score corresponds to the ground truth label

https://torchmetrics.readthedocs.io/en/stable/classification/hinge_loss.html

Regression Loss Functions

Mean Absolute Loss or L1 loss

$$L_1(x) = |x|$$

$$f(y, \hat{y}) = \sum_{i=1}^{N} |y_i - \hat{y}_i|$$

Mean Square Loss or L2 loss

$$L_2(x) = x^2$$

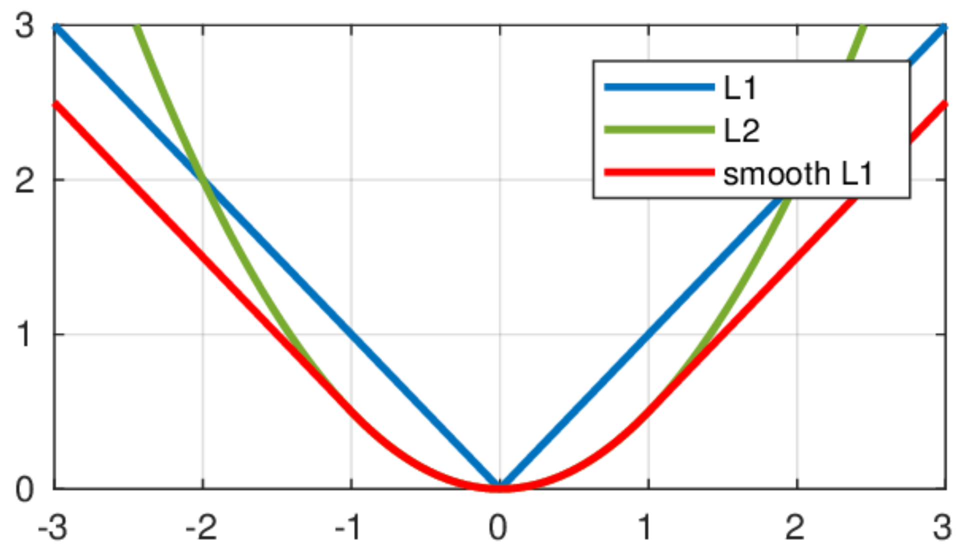
$$f(y, \hat{y}) = \sum_{i=1}^{N} (y_i - \hat{y}_i)^2$$

Regression Loss Functions

Smooth L1 loss

$$\text{smooth } \mathrm{L}_1(x) = \begin{cases} 0.5x^2 & if|x| < 1 \\ |x| - 0.5 & otherwise \end{cases}$$

$$f(y, \hat{y}) = \begin{cases} 0.5(y - \hat{y})^2 & \text{if } |y - \hat{y}| < 1 \\ |y - \hat{y}| - 0.5 & otherwise \end{cases}$$



https://pytorch.org/docs/stable/generated/torch.nn.SmoothL1Loss.html

Optimization

Gradient descent

- Gradient direction: steepest direction to increase the objective
- Can only find local minimum
- Widely used for neural network training (works in practice)
- Compute gradient with a mini-batch (Stochastic Gradient Descent, SGD)

$$W \leftarrow W - \underbrace{\gamma}_{\text{Learning rate}} \frac{\partial L}{\partial W}$$

Optimization

Gradient descent with momentum

- Add a fraction of the update vector from previous time step (momentum)
- Accelerated SGD, reduced oscillation

Image 2: SGD without momentum

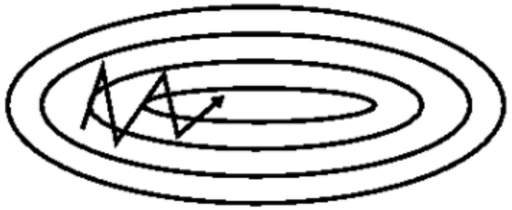


Image 3: SGD with momentum

https://ruder.io/optimizing-gradient-descent/

momentum   Learning rate

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta)$$
$$\theta = \theta - v_t$$

Optimization

Adam: Adaptive Moment Estimation

1. Exponentially decaying average of gradients and squared gradients

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$
$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

$$\beta_1 = 0.9, \ \beta_2 = 0.999$$

Start m and v from 0s

2. Bias-corrected 1st and 2nd moment estimates

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \qquad \hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

3. Updating rule

Learning rate

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t$$

$$\epsilon = 10^{-8}$$

Adaptive learning rate

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

Matrix Calculus: https://explained.ai/matrix-calculus/