

Convolutional Neural Networks II

CS 6384 Computer Vision

Professor Yu Xiang

The University of Texas at Dallas

Supervised Learning



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

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

Input

Output

Convolutional Neural Networks

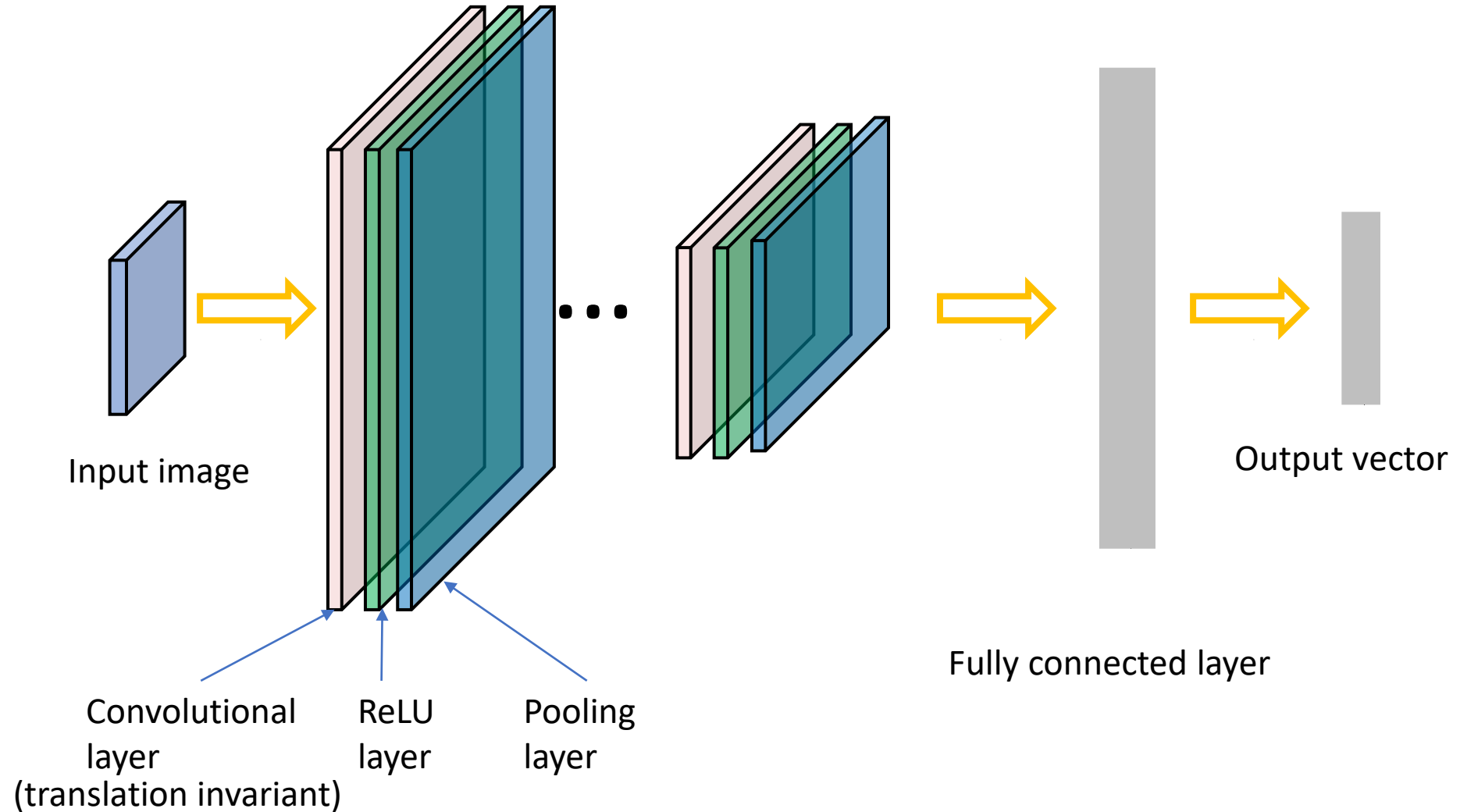


Image Classification

- 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*

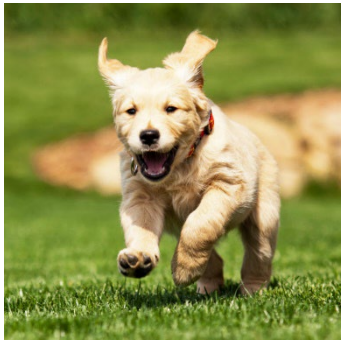
n02423022: gazelle



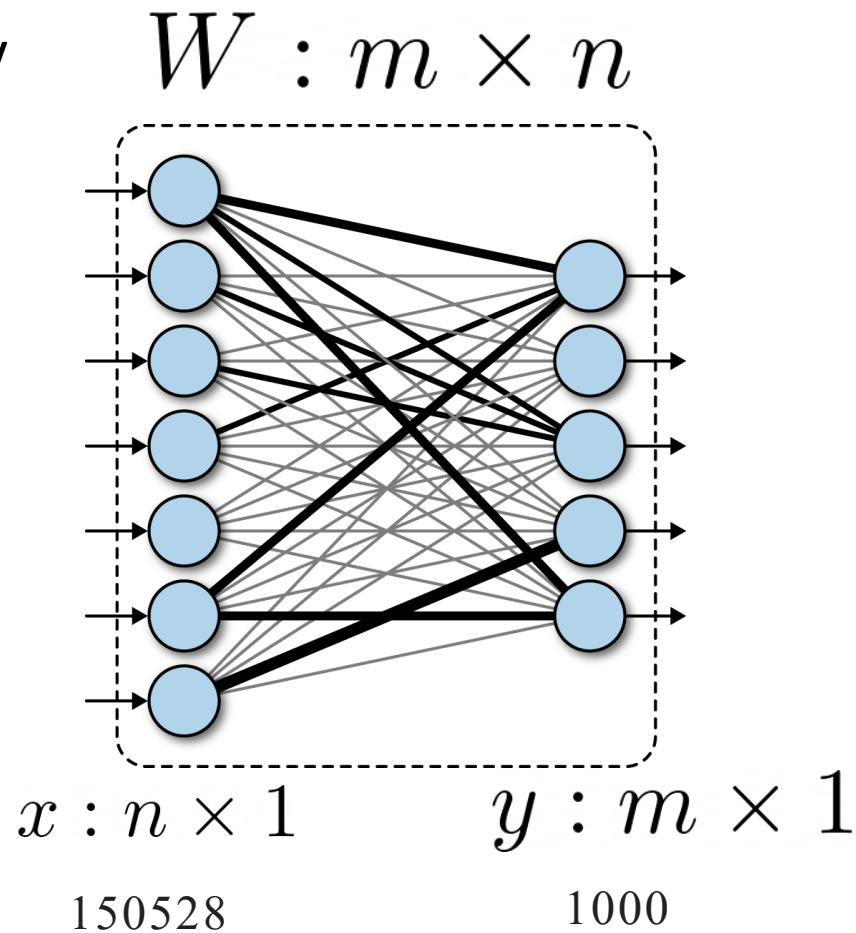
<https://image-net.org/challenges/LSVRC/2012/index.php>

Image Classification

Let's consider only using one FC layer



$224 \times 224 \times 3$




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

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

Softmax function

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

Image Classification

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

Image label


- One-hot vector $\mathbf{y}_i = 000 \dots 1 \dots 000$

Ground true category

Image Classification

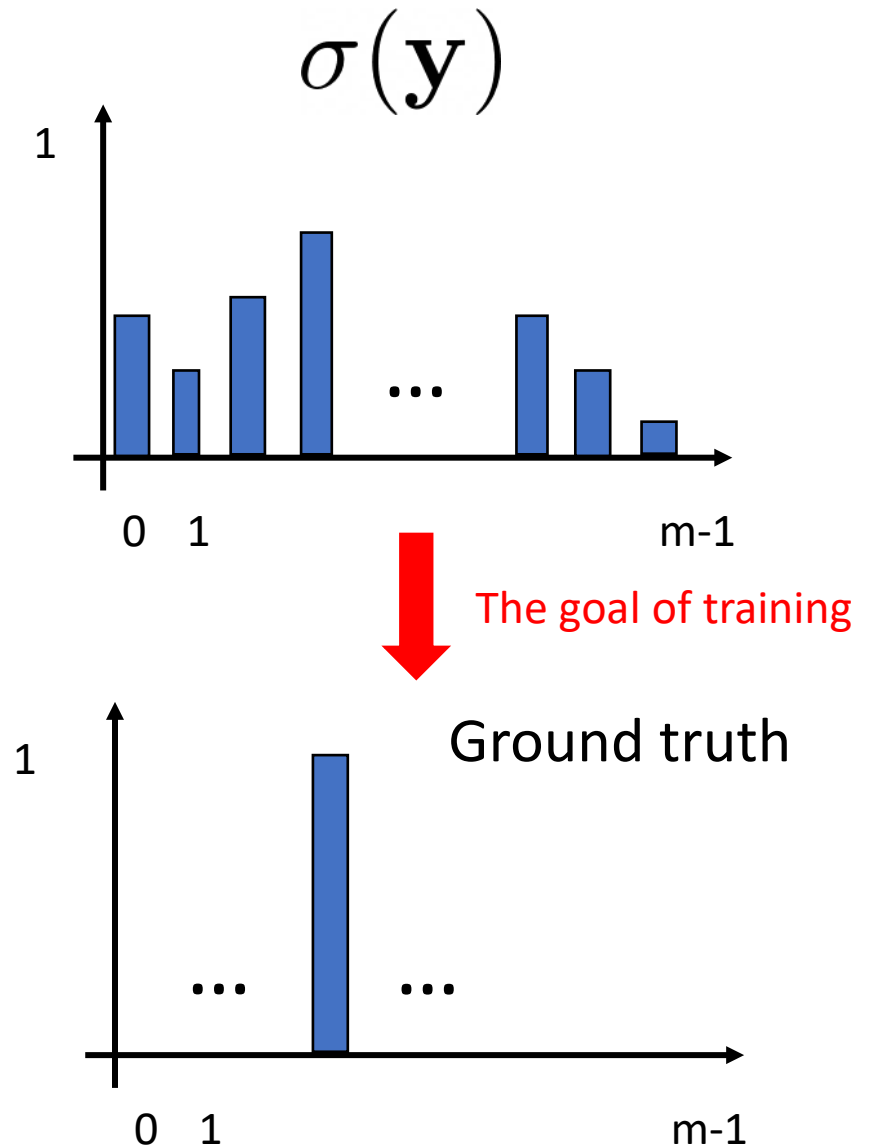
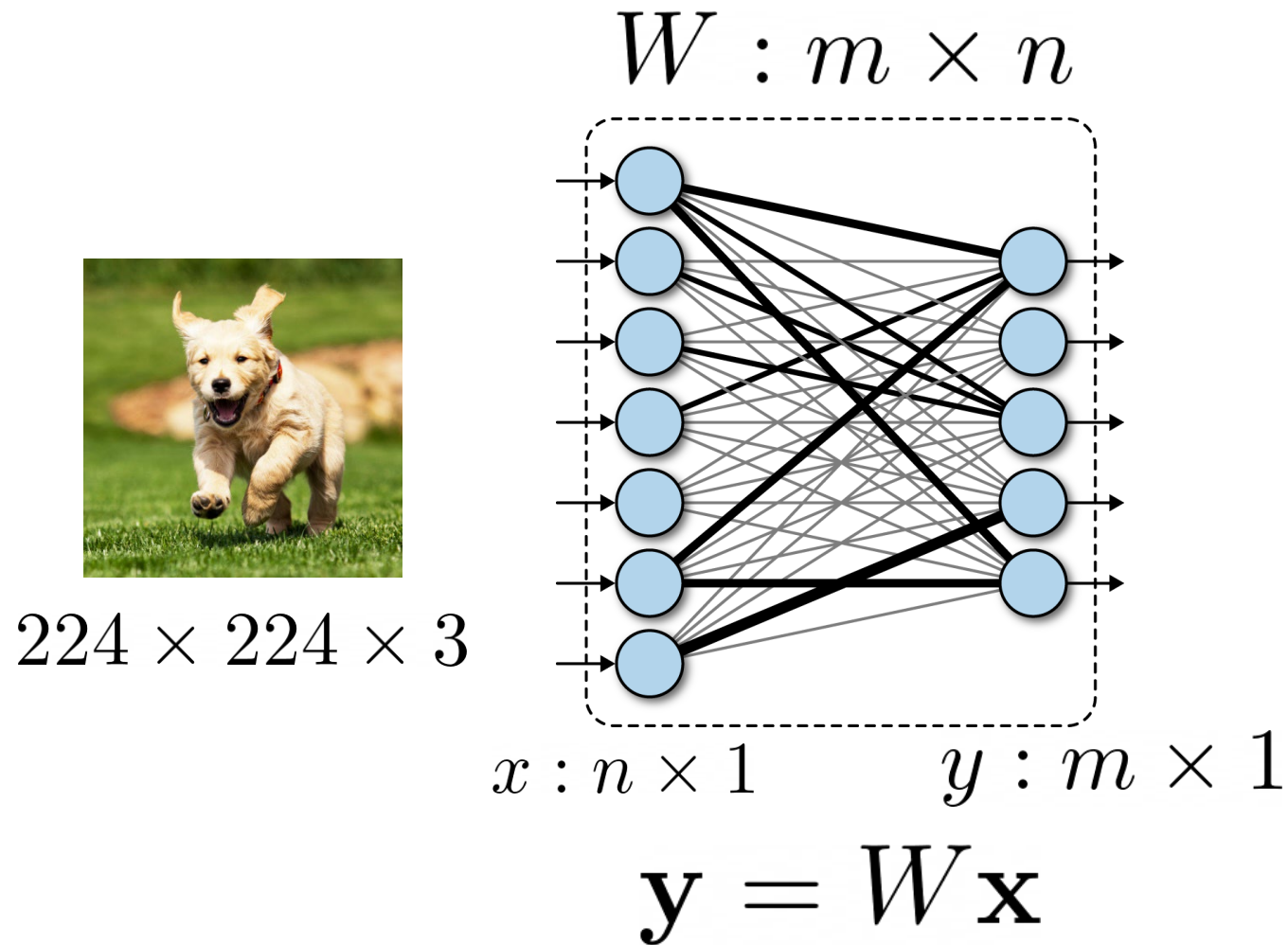


Image Classification

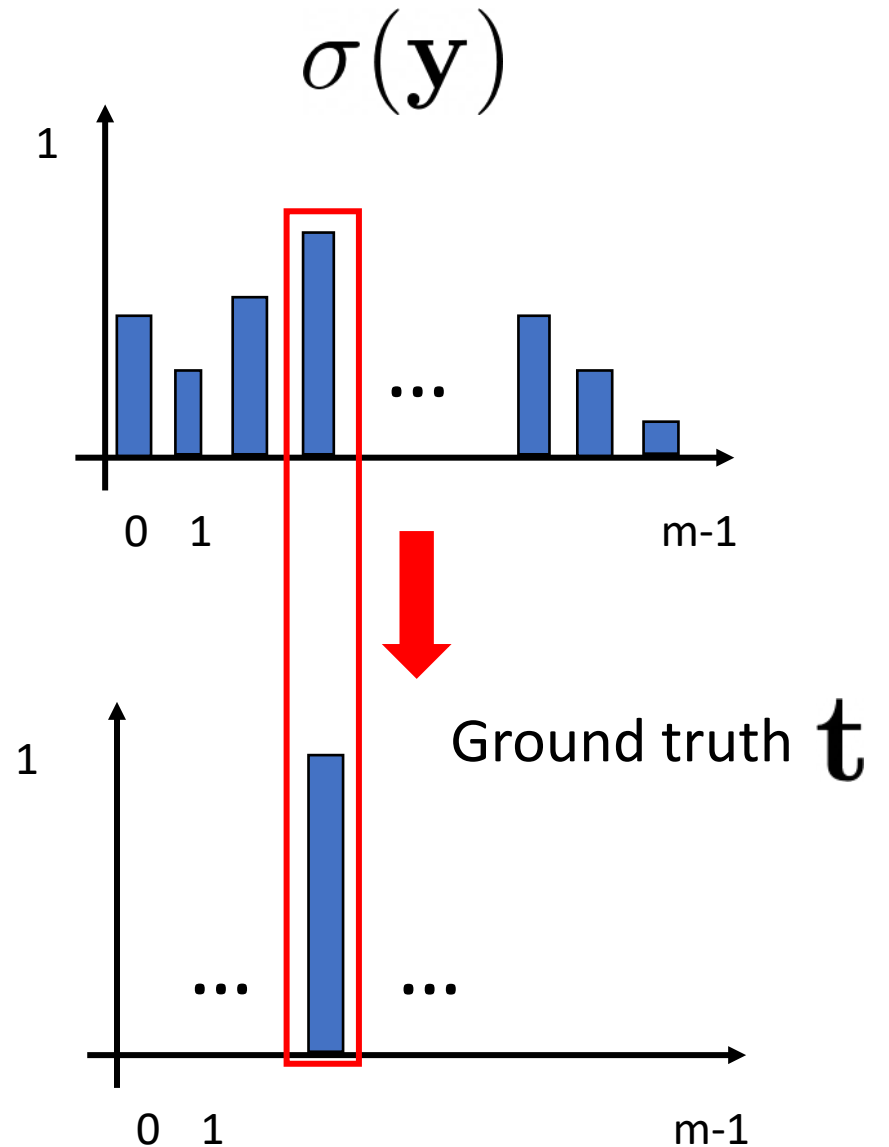
- Cross entropy loss function

Cross entropy between two distributions
(measure distance between distributions)

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

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

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



Training

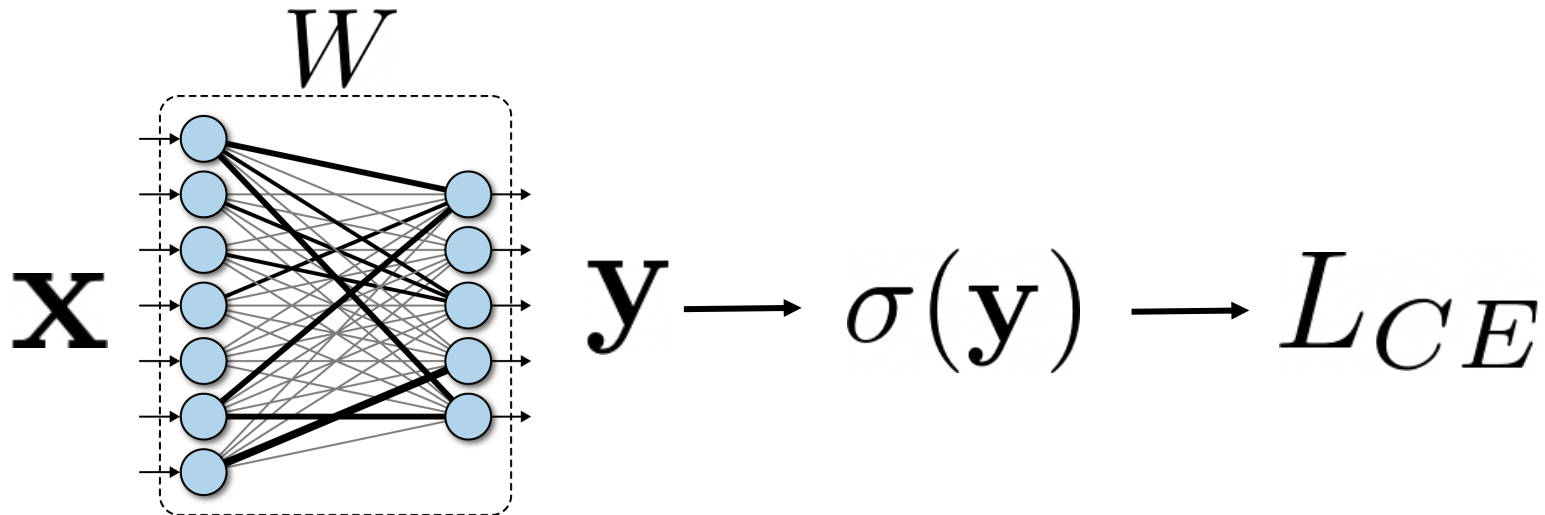
- Cross entropy loss function

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

With respect to weights W

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

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



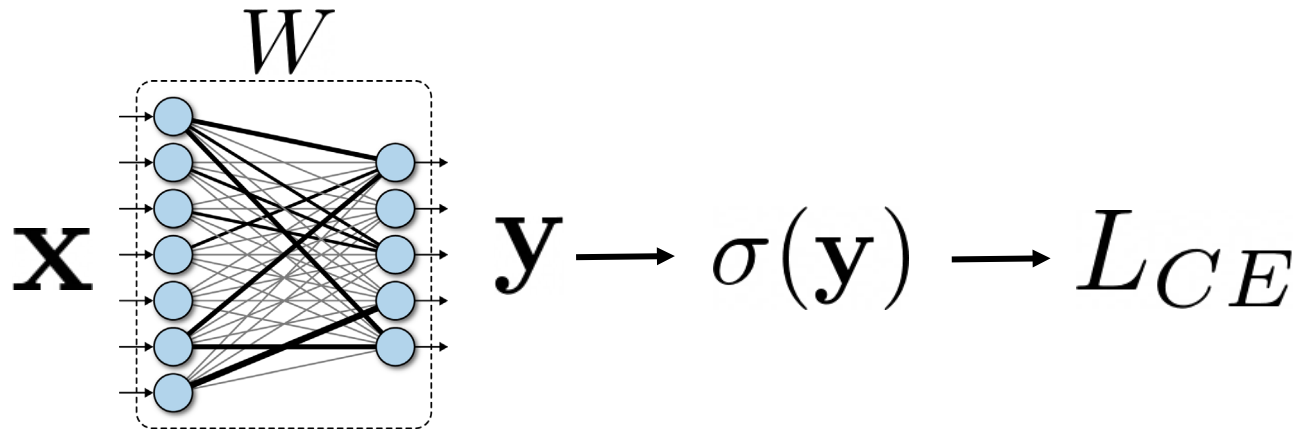
Training

- Gradient descent

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

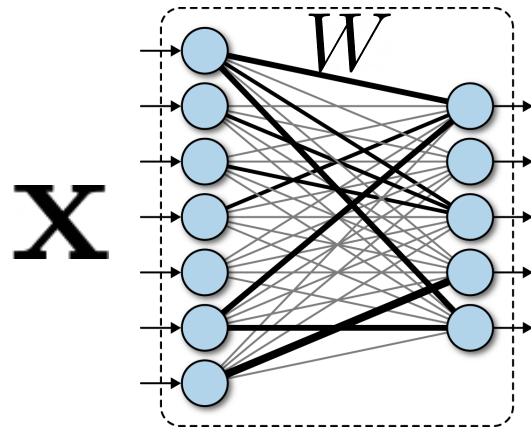
- Chain rule

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



Training

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



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

How to compute gradient?

$$\frac{\partial L}{\partial \mathbf{y}} \left[\frac{\partial L}{y_1} \quad \frac{\partial L}{y_2} \quad \cdots \quad \frac{\partial L}{y_m} \right]$$

$$1 \times m$$

Training

- Chain rule

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

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

$$\begin{array}{ccc} \frac{\partial L}{\partial \mathbf{y}} & = & \frac{\partial L}{\partial \sigma(\mathbf{y})} \cdot \frac{\partial \sigma(\mathbf{y})}{\partial \mathbf{y}} \\ 1 \times m & & 1 \times m \quad m \times m \end{array} \quad \frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \begin{bmatrix} \nabla f_1(\mathbf{x}) \\ \nabla f_2(\mathbf{x}) \\ \vdots \\ \nabla f_m(\mathbf{x}) \end{bmatrix} = \begin{bmatrix} \frac{\partial}{\partial \mathbf{x}} f_1(\mathbf{x}) \\ \frac{\partial}{\partial \mathbf{x}} f_2(\mathbf{x}) \\ \vdots \\ \frac{\partial}{\partial \mathbf{x}} f_m(\mathbf{x}) \end{bmatrix} = \begin{bmatrix} \frac{\partial}{\partial x_1} f_1(\mathbf{x}) & \frac{\partial}{\partial x_2} f_1(\mathbf{x}) & \dots & \frac{\partial}{\partial x_n} f_1(\mathbf{x}) \\ \frac{\partial}{\partial x_1} f_2(\mathbf{x}) & \frac{\partial}{\partial x_2} f_2(\mathbf{x}) & \dots & \frac{\partial}{\partial x_n} f_2(\mathbf{x}) \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial}{\partial x_1} f_m(\mathbf{x}) & \frac{\partial}{\partial x_2} f_m(\mathbf{x}) & \dots & \frac{\partial}{\partial x_n} f_m(\mathbf{x}) \end{bmatrix}$$

Jacobian matrix

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

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

Training

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

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

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

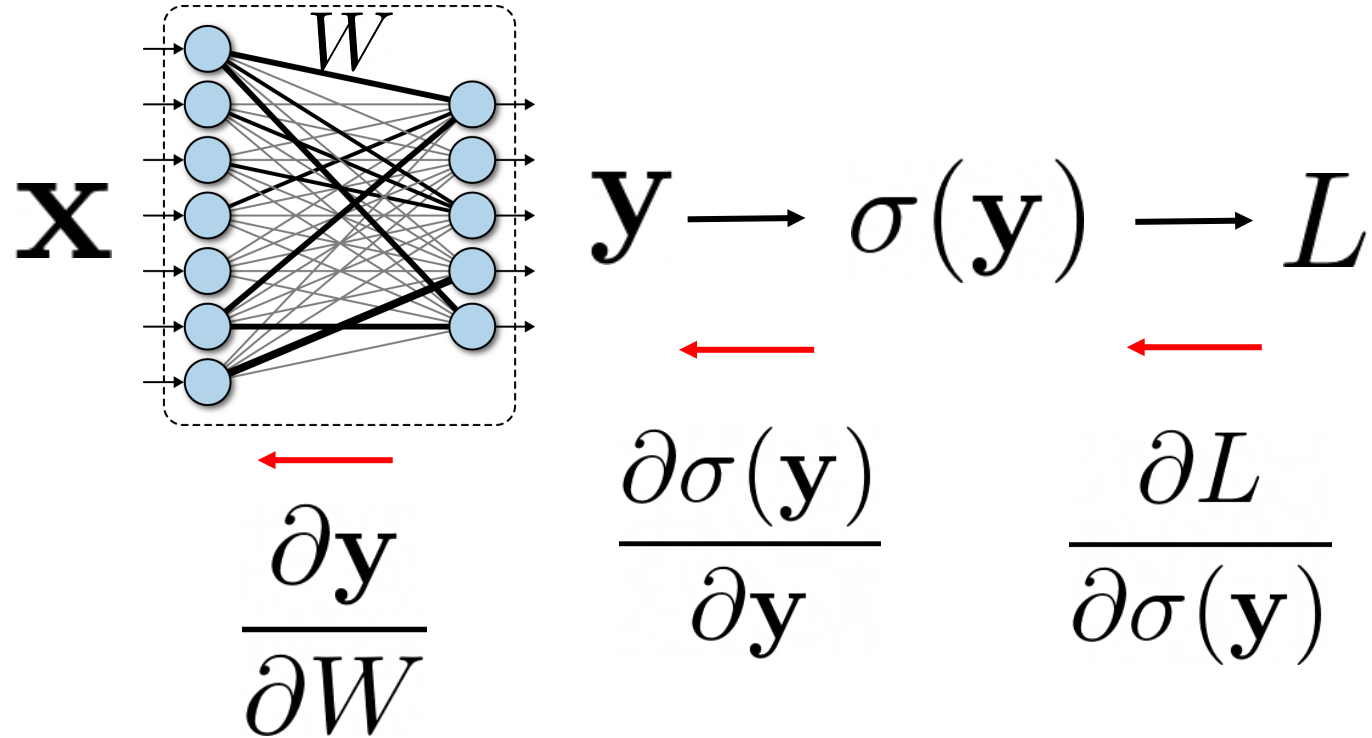
$$\frac{\partial L}{\partial \sigma(\mathbf{y})} = -\mathbf{t} \cdot \frac{1}{\sigma(\mathbf{y})} \quad \frac{\partial \sigma(\mathbf{y})_i}{\partial y_j} = \sigma(\mathbf{y})_i (\delta_{ij} - \sigma(\mathbf{y})_j) \quad \delta_{ij} = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases}$$

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

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

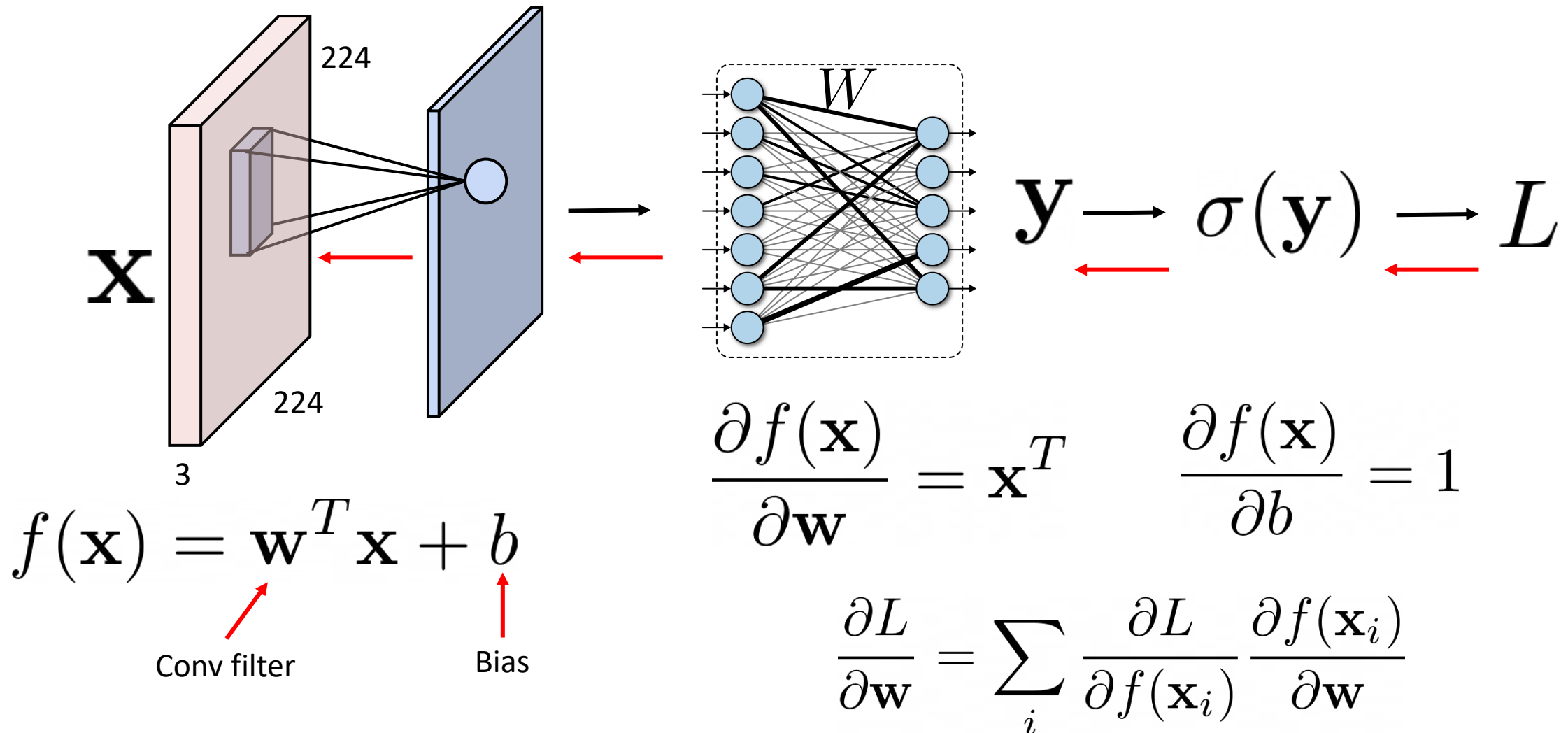
Learning rate

Back-propagation

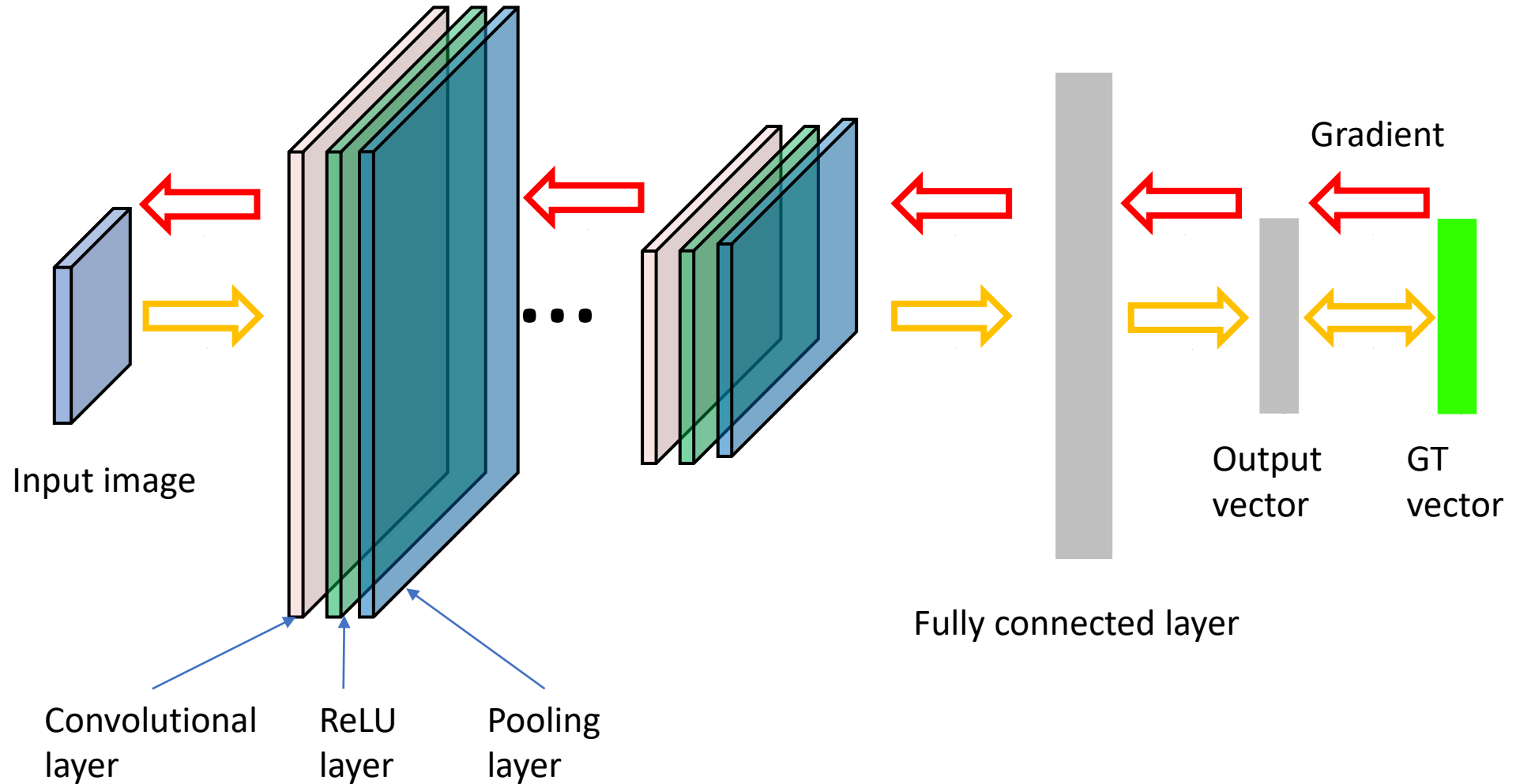


$$\frac{\partial L}{\partial W} = \frac{\partial L}{\partial \sigma(\mathbf{y})} \frac{\partial \sigma(\mathbf{y})}{\partial \mathbf{y}} \frac{\partial \mathbf{y}}{\partial W}$$

Back-propagation



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

$$L_{CE} = - \sum_{i=0}^{m-1} \underset{\substack{\uparrow \\ \text{Binary} \\ \text{ground} \\ \text{truth label}}}{t_i} \log \sigma(\underset{\substack{\uparrow \\ \text{Logit}}}{\mathbf{y}})_i$$

- Hinge loss for binary classification

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

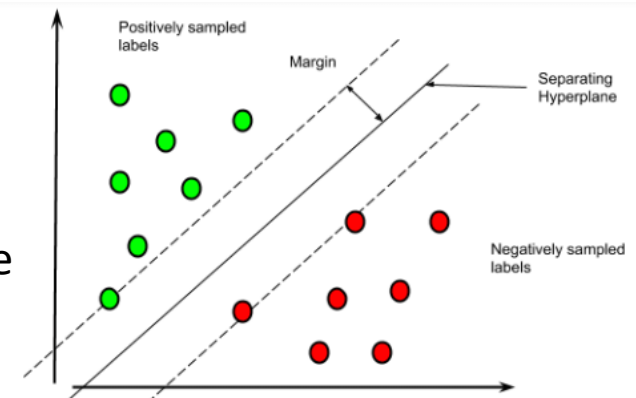
ground truth label $t \in \{-1, +1\}$

\uparrow
Classification score

$y \geq 0$ 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

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

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

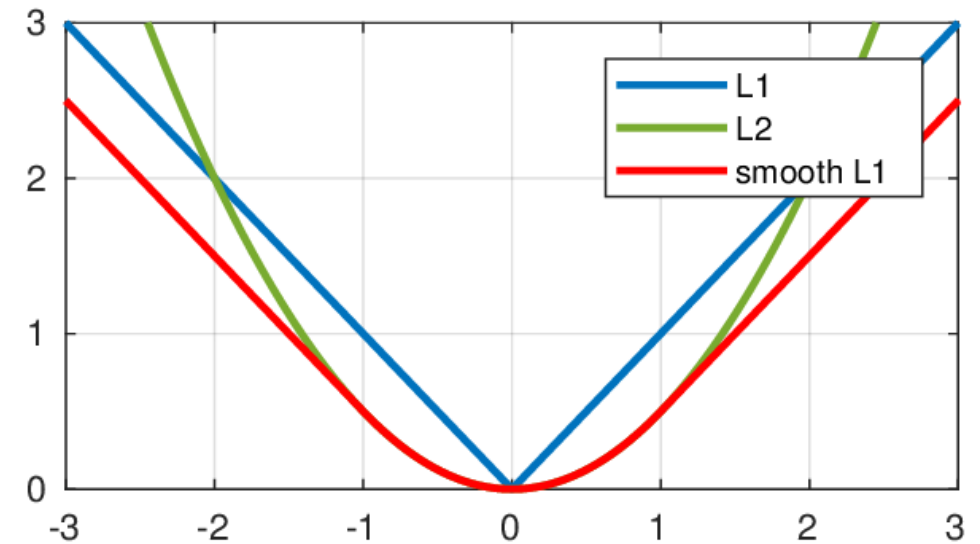
$$f(y, \hat{y}) = \sum_{i=1}^N (y_i - \hat{y}_i)^2$$

Regression Loss Functions

- Smooth L1 loss

$$\text{smooth } L_1(x) = \begin{cases} 0.5x^2 & \text{if } |x| < 1 \\ |x| - 0.5 & \text{otherwise} \end{cases}$$

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

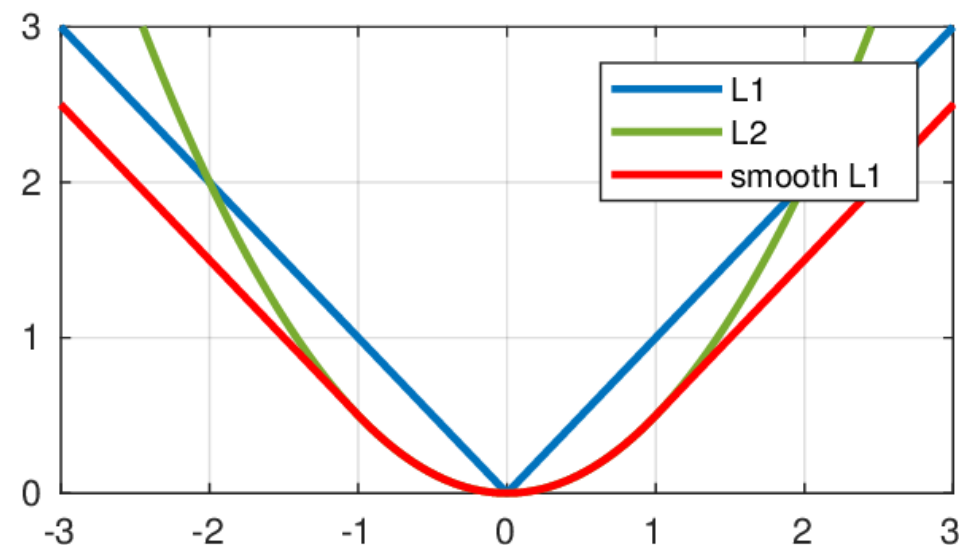


Regression Loss Functions

- Huber loss
 - Generalization of smooth L1 loss ($\delta = 1$)

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

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



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)

$$W \leftarrow W - \underset{\substack{\uparrow \\ \text{Learning rate}}}{\gamma} \frac{\partial L}{\partial W}$$

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

momentum Learning rate

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

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

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

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

3. Updating rule

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

Learning rate

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

Adaptive learning rate

Case Study: Training AlexNet

- Data augmentation
 - Extracting random 224x224 patches from 256x256 images

- Change RGB intensities

$$\begin{bmatrix} I_{xy}^R, I_{xy}^G, I_{xy}^B \end{bmatrix}^T + \begin{bmatrix} \mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3 \end{bmatrix} \begin{bmatrix} \alpha_1 \lambda_1, \alpha_2 \lambda_2, \alpha_3 \lambda_3 \end{bmatrix}^T$$

Eigen vectors
of 3x3 covariance
matrix of RGB values
on training set

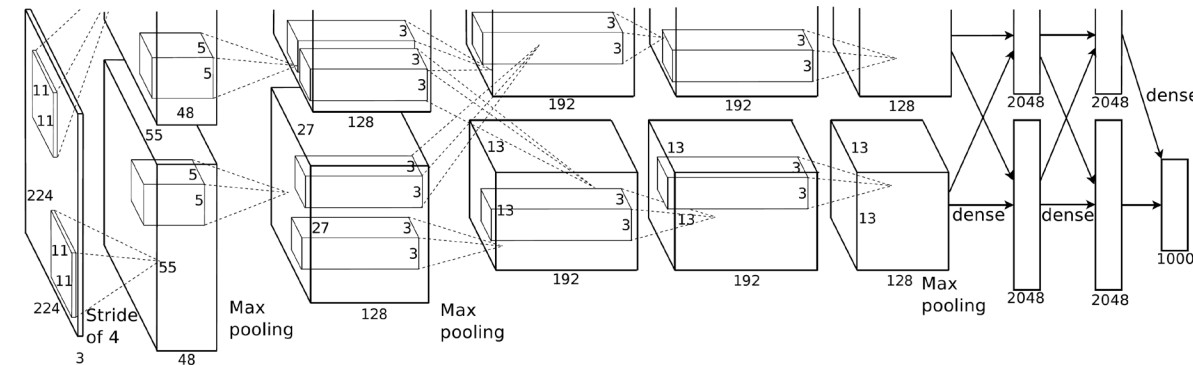
Random variable
 $N(0, 0.1)$

Eigen values

covariance matrix

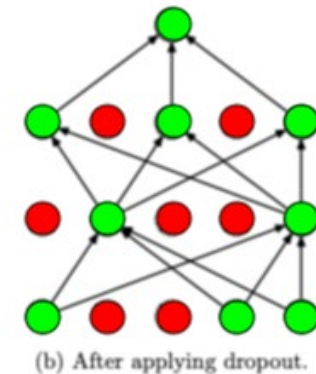
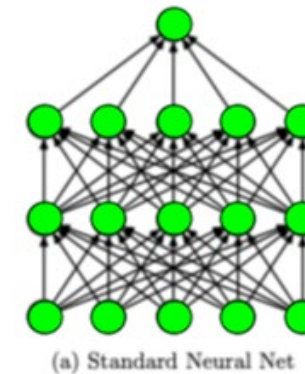
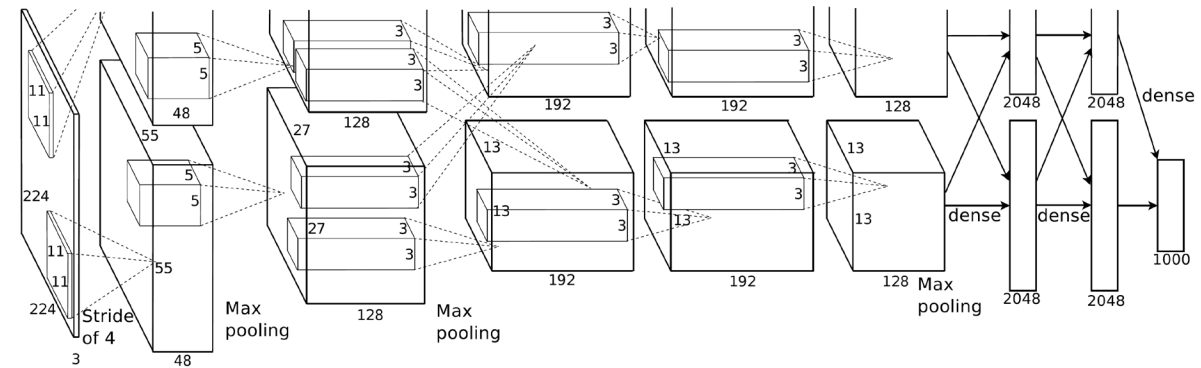
$$S = \frac{1}{n-1} \sum_{i=1}^n (X_i - \bar{X})(X_i - \bar{X})'$$

<https://papers.nips.cc/paper/2012/hash/c399862d3b9d6b76c8436e924a68c45b-Abstract.html>



Case Study: Training AlexNet

- Dropout
 - Set to zero the output of each hidden neuron with probability 0.5
 - Apply to the first two FC layers
 - Prevent overfitting



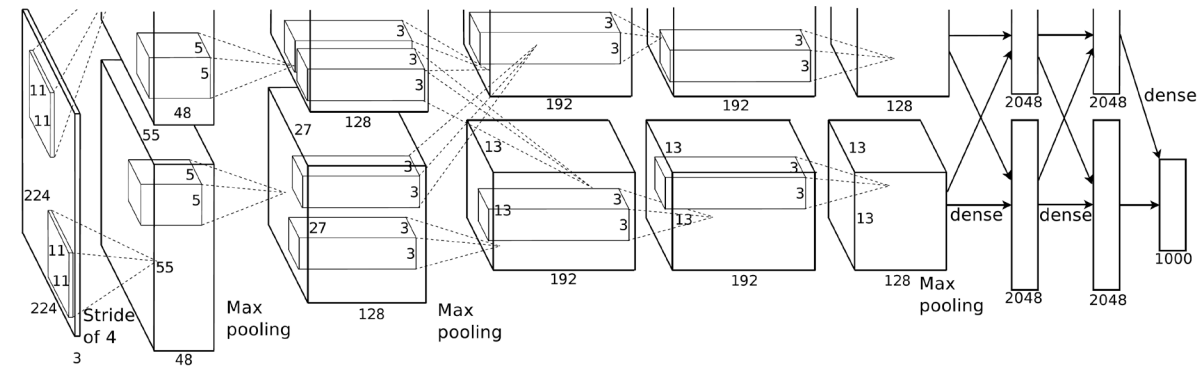
<https://papers.nips.cc/paper/2012/hash/c399862d3b9d6b76c8436e924a68c45b-Abstract.html>

Case Study: Training AlexNet

- Batch size: 128
- Updating rule

$$w_{i+1} := w_i + v_{i+1}$$

$$v_{i+1} := \underset{\text{Momentum}}{0.9} \cdot v_i - \underset{\text{Weight Decay}}{0.0005} \cdot \epsilon \cdot w_i - \underset{\text{Learning rate}}{\epsilon} \cdot \underset{\text{Gradient}}{\left\langle \frac{\partial L}{\partial w} \Big|_{w_i} \right\rangle_{D_i}}$$



Five to six days on two NVIDIA GTX 580 3GB GPUs, 2012

<https://papers.nips.cc/paper/2012/hash/c399862d3b9d6b76c8436e924a68c45b-Abstract.html>

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/>