

Convolutional Neural Networks II

CS 4391 Introduction 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

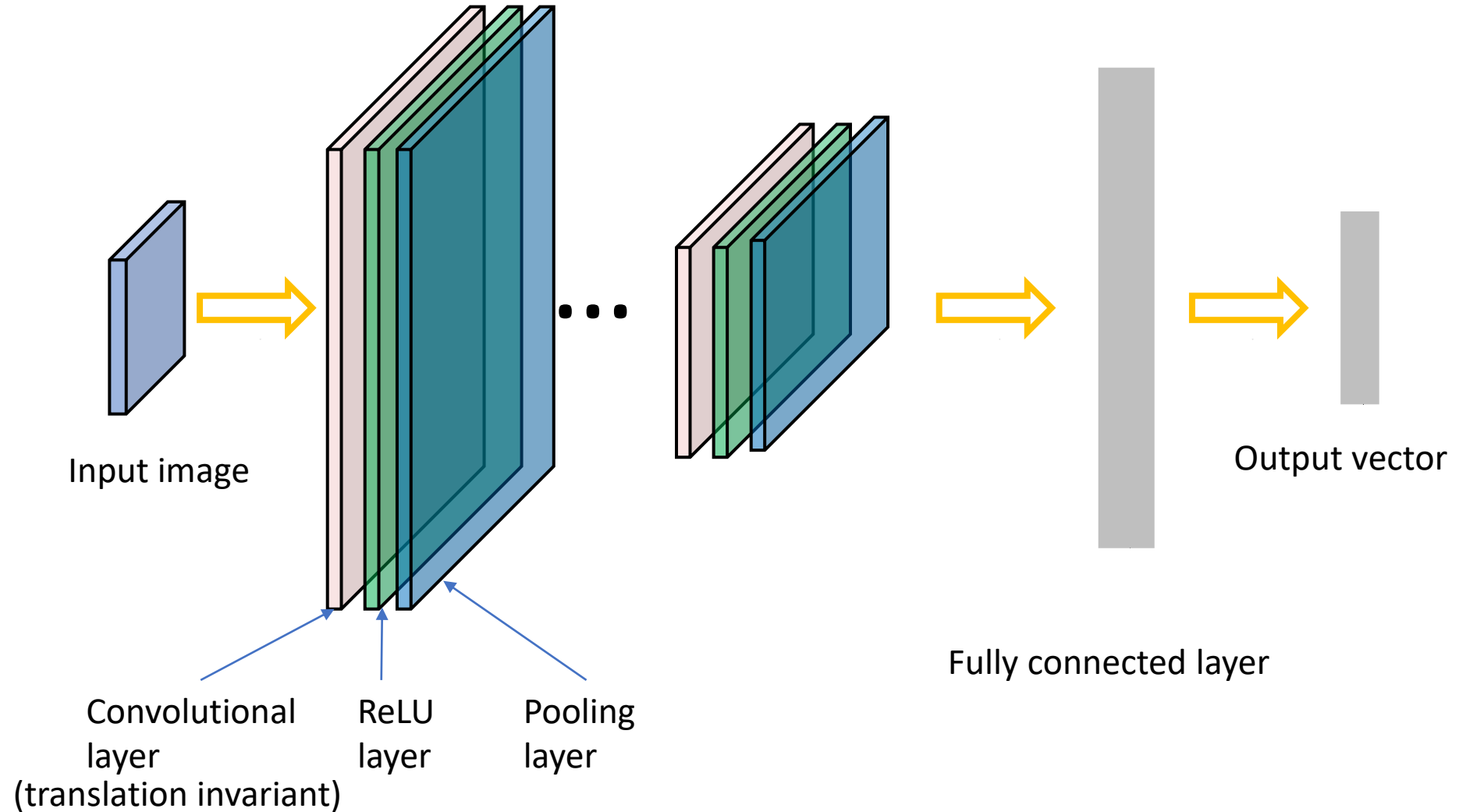


Image Classification

- 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*

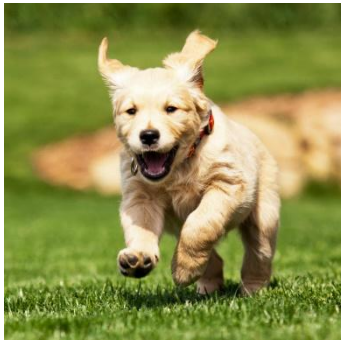
n02423022: gazelle



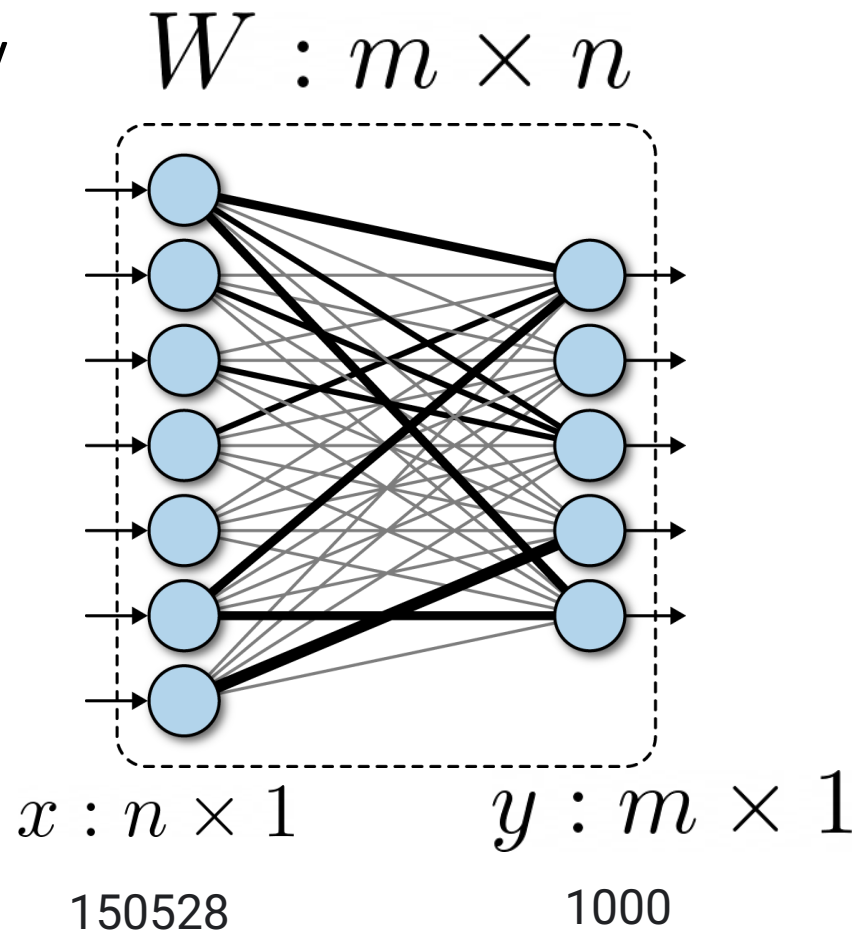
<https://image-net.org/challenges/LSVRC/2012/index.php>

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 truth category

Image Classification

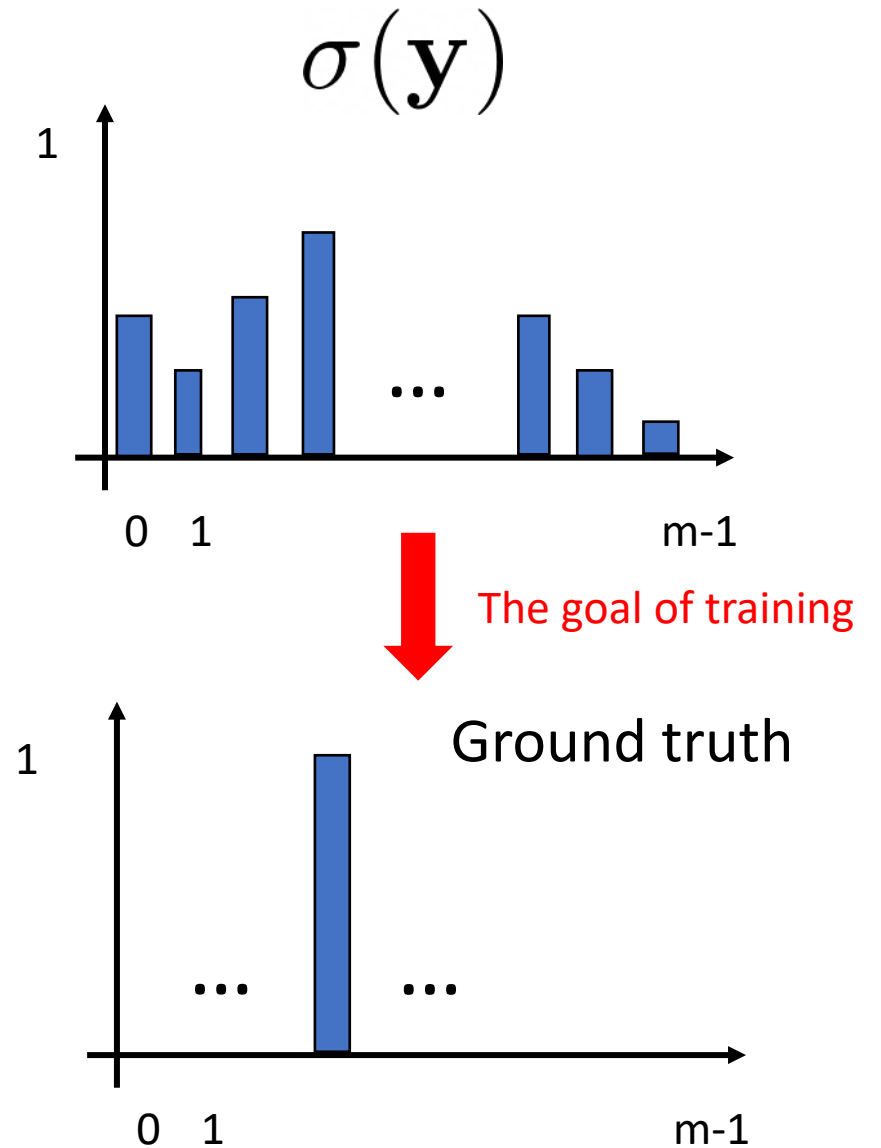
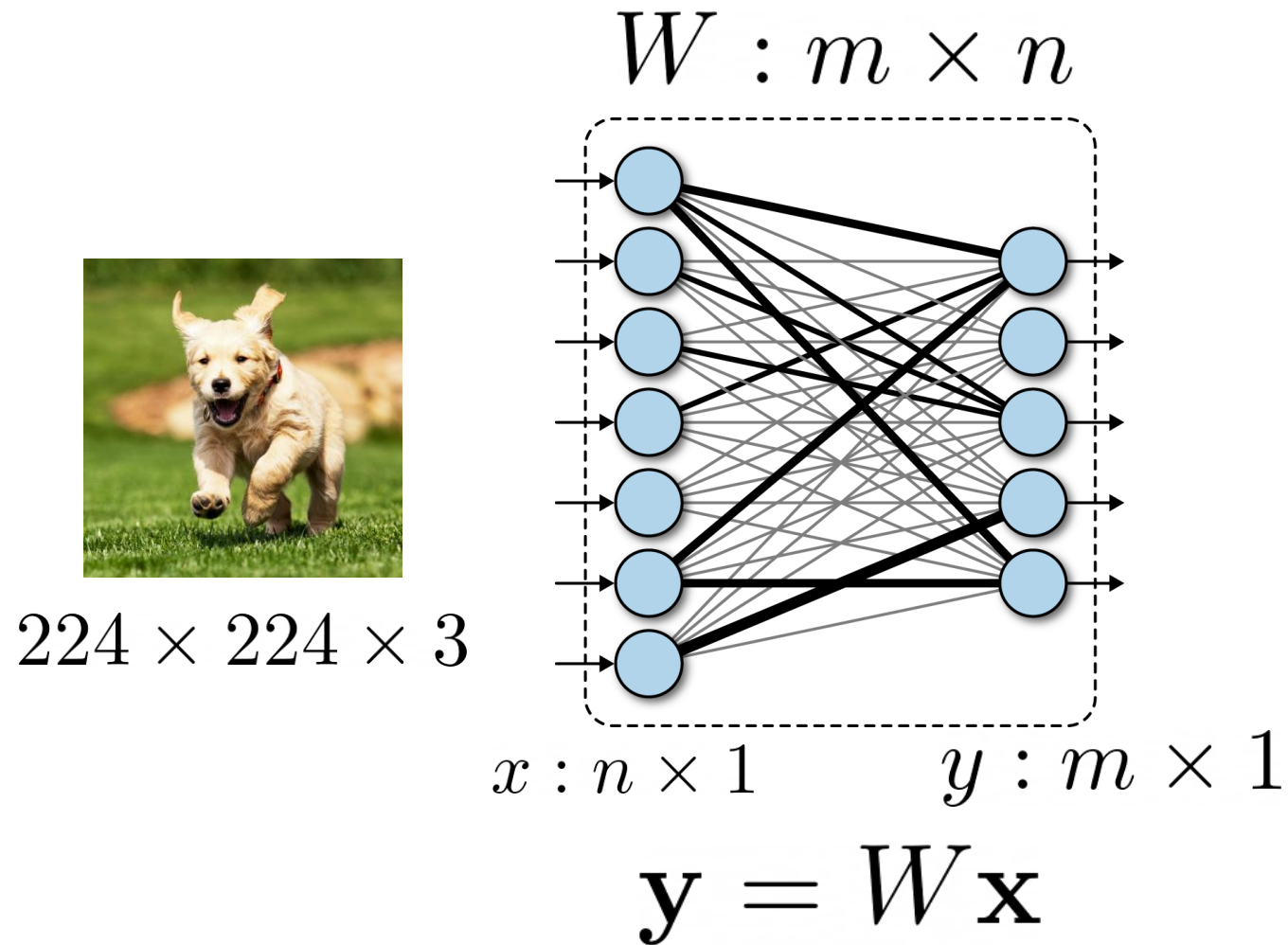


Image Classification

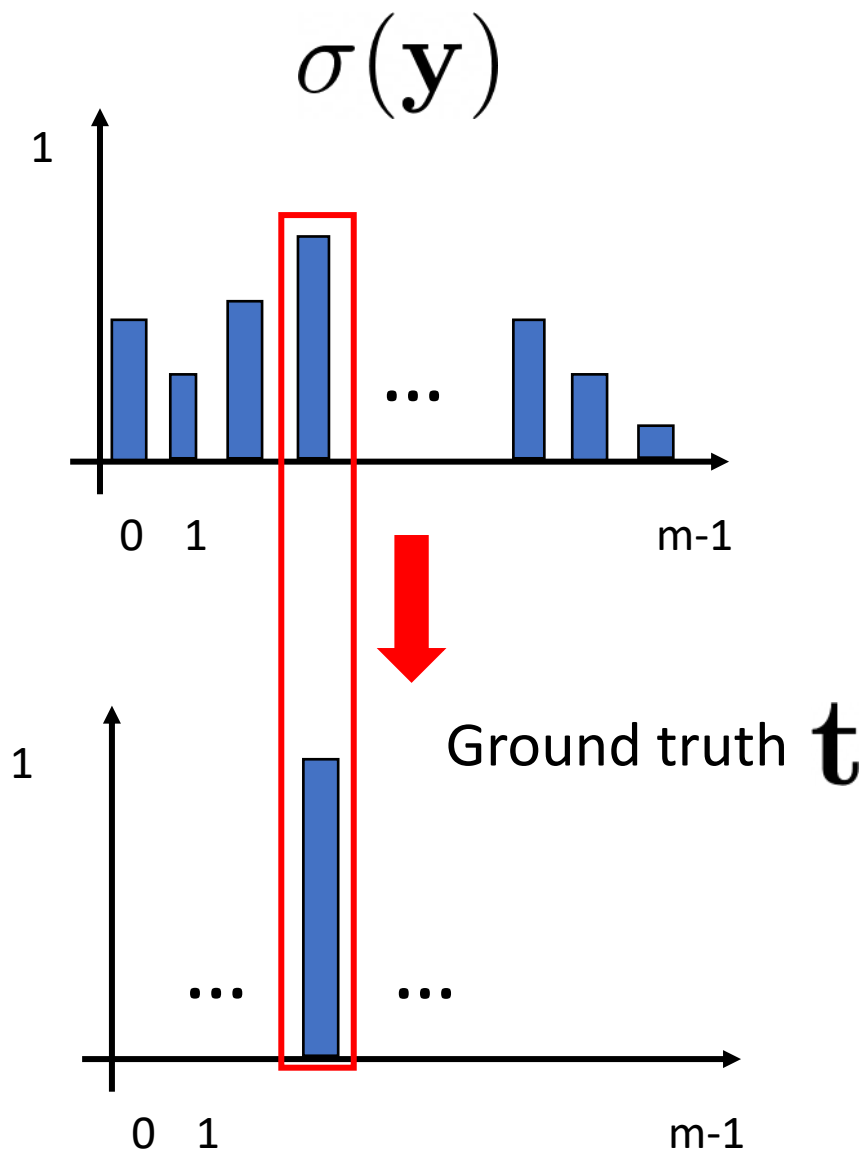
- Cross entropy loss function

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

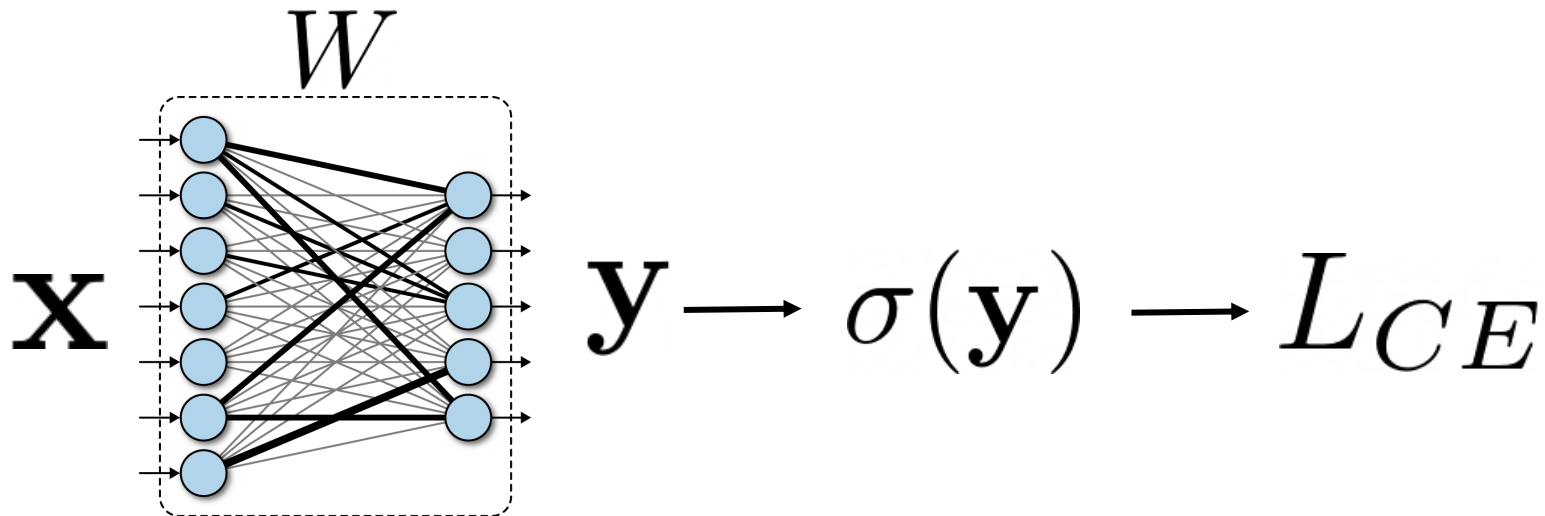
- Cross entropy loss function

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

With respect to weights W

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

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



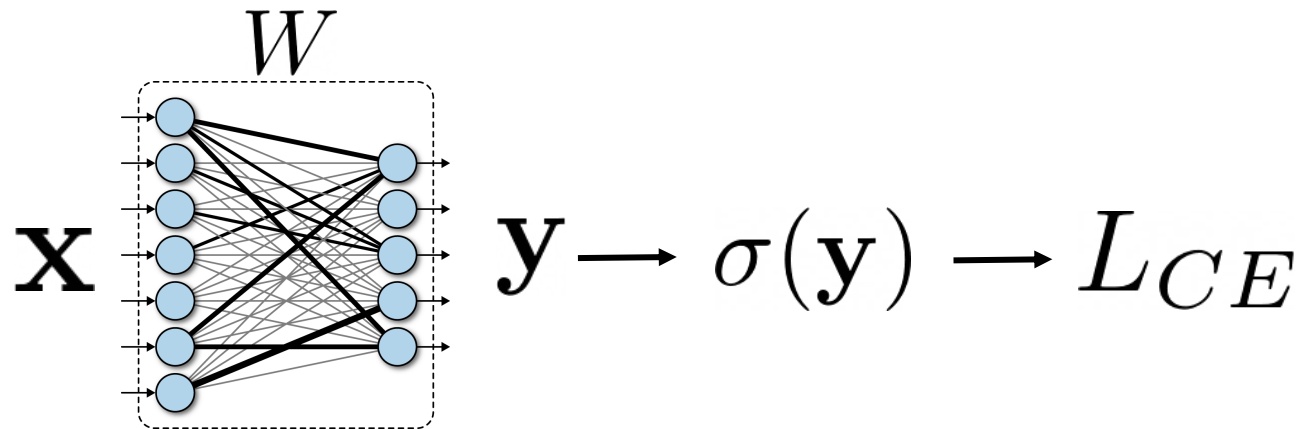
Training

- Gradient descent

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

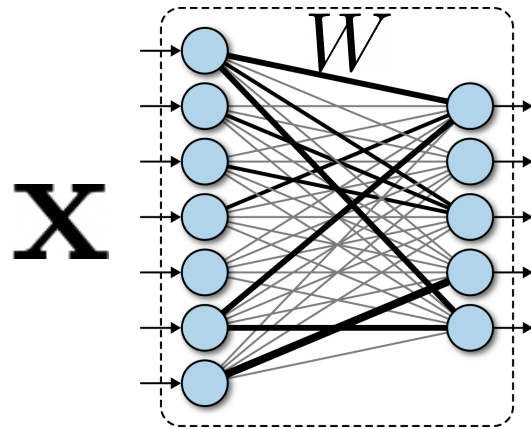
- Chain rule

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



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

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

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

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

$1 \times m \quad 1 \times m \quad m \times m$

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

Jacobian matrix

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

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

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

Training

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

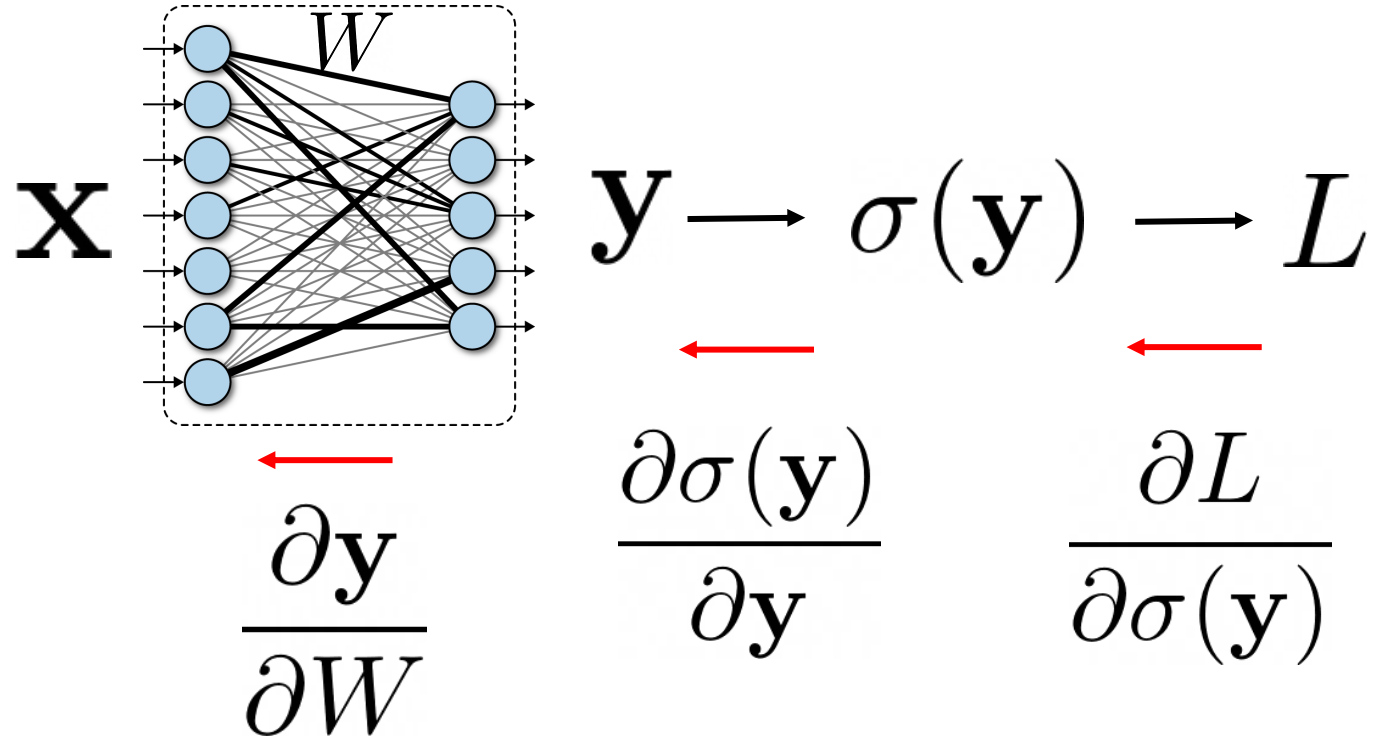
$$\frac{\partial L}{\partial W} = \frac{\partial L}{\partial \sigma(\mathbf{y})} \frac{\partial \sigma(\mathbf{y})}{\partial \mathbf{y}} \frac{\partial \mathbf{y}}{\partial W} \quad \mathbf{y} = W\mathbf{x}$$

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

$$\frac{\partial y_i}{\partial W_{jk}} = \begin{cases} 0 & \text{if } i \neq j \\ x_k & \text{otherwise} \end{cases} \quad W \leftarrow W - \gamma \frac{\partial L}{\partial W}$$

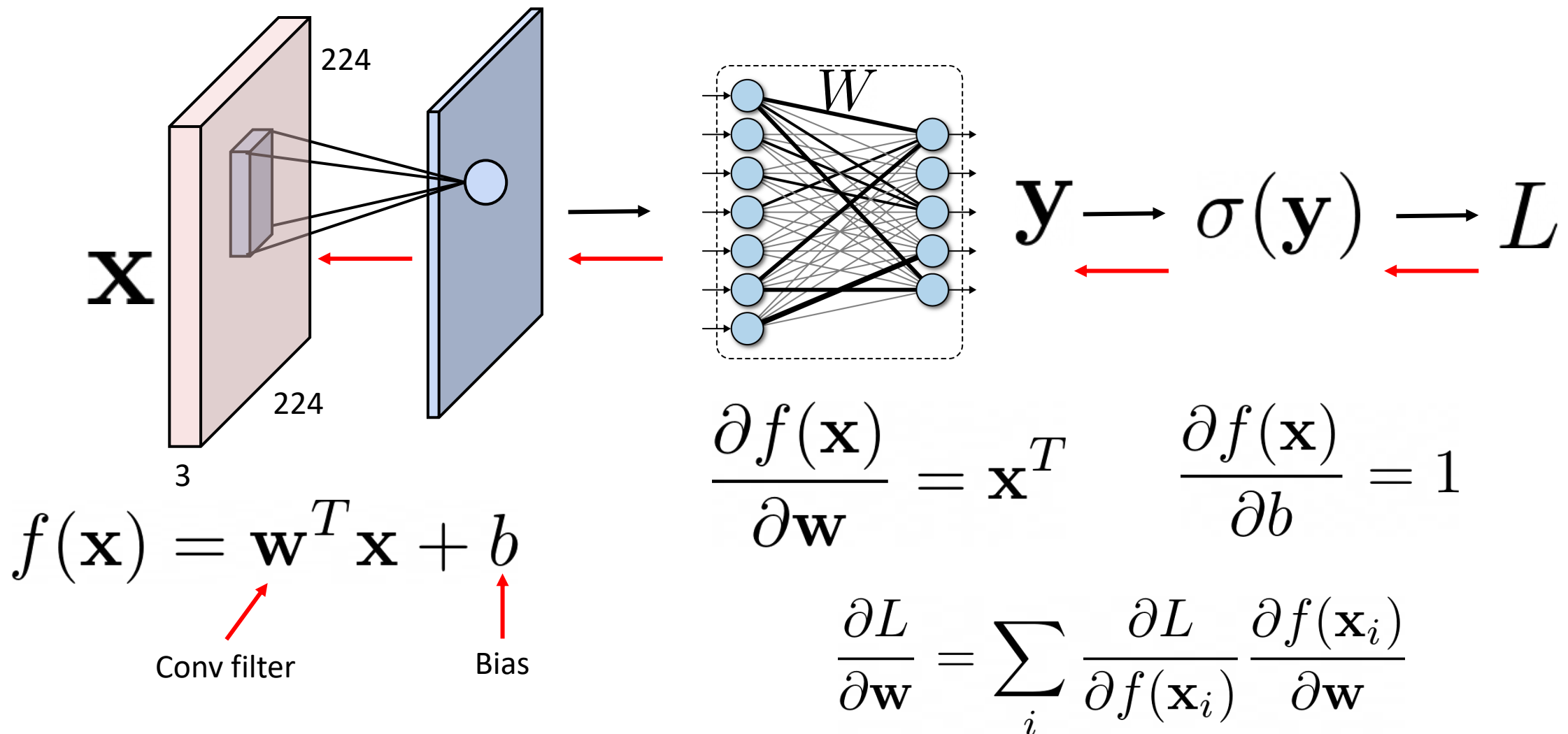
Learning rate

Back-propagation

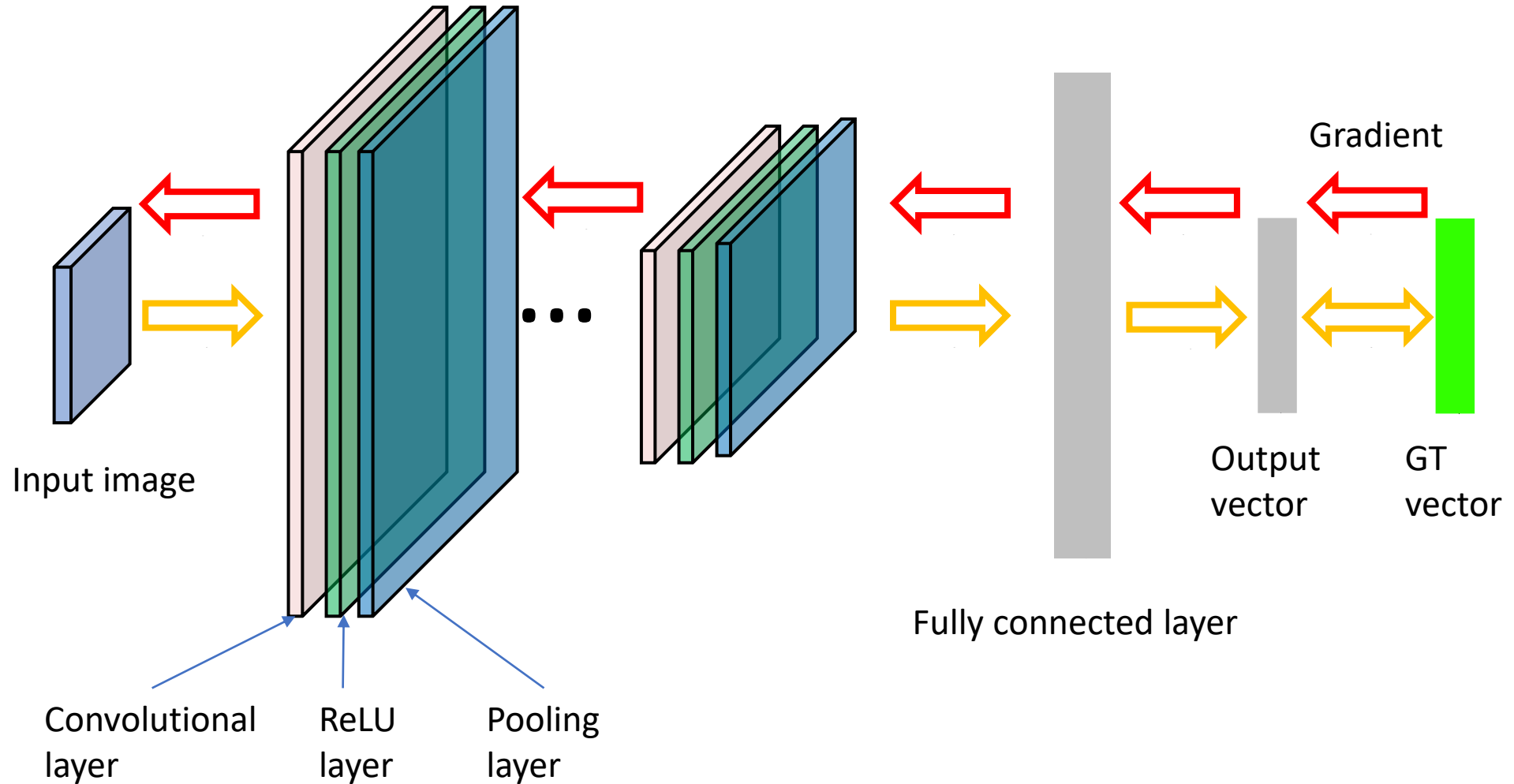


$$\frac{\partial L}{\partial W} = \frac{\partial L}{\partial \sigma(\mathbf{y})} \frac{\partial \sigma(\mathbf{y})}{\partial \mathbf{y}} \frac{\partial \mathbf{y}}{\partial W}$$

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

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

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/>