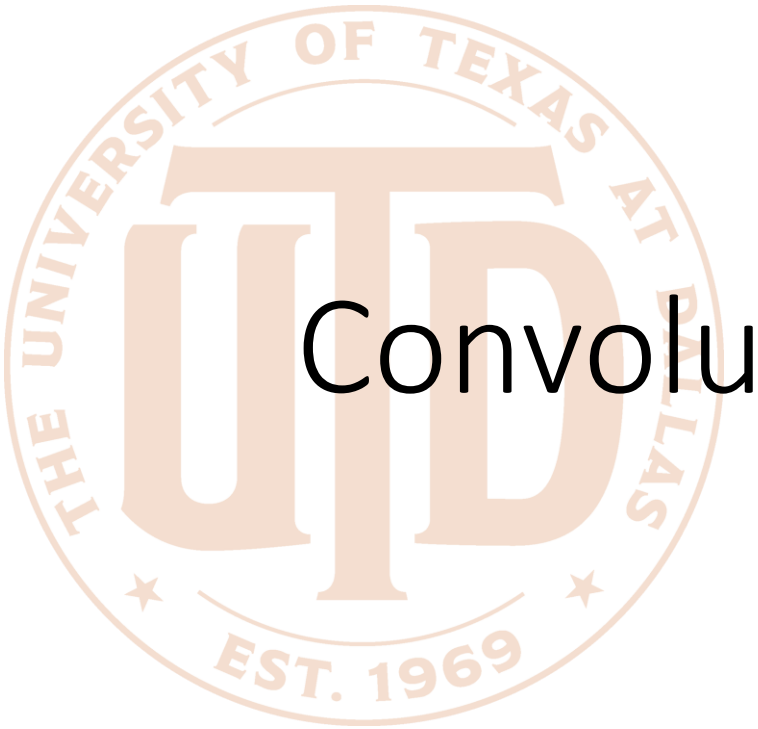


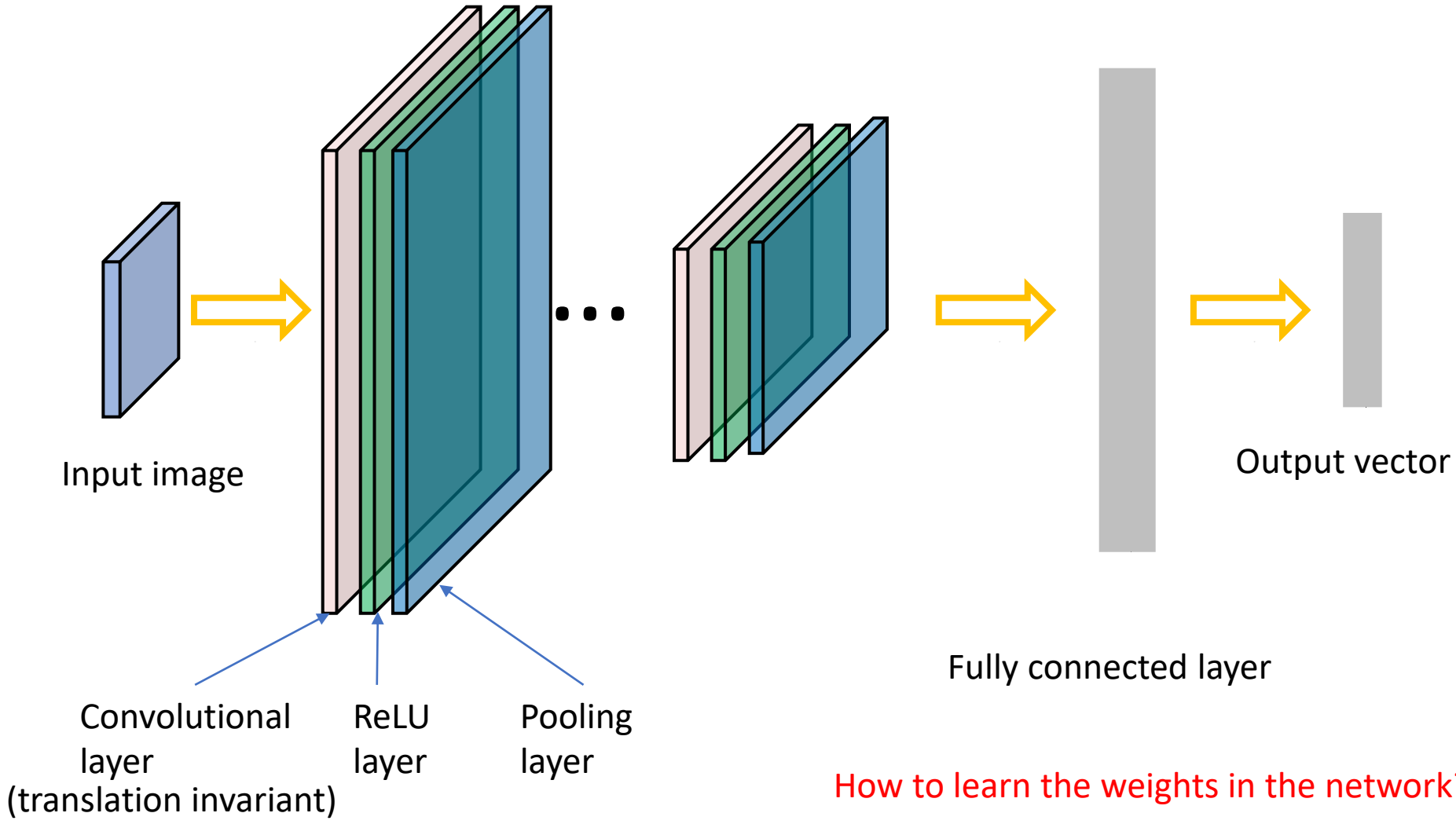
Convolutional Neural Networks III: Network Training

CS 4391 Introduction Computer Vision

Instructor Yu Xiang

The University of Texas at Dallas

Convolutional Neural Networks



How to learn the weights in the network?

Supervised Learning

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

Image

Object class

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

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

Input Output

Image Classification

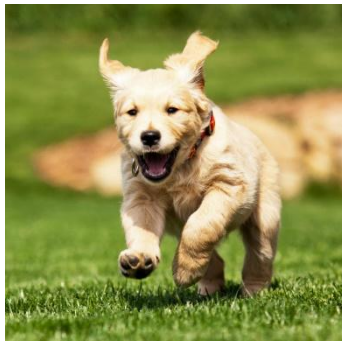
- ImageNet dataset
  - Training: 1.2 million images
  - Testing and validation: 150,000 images
  - 1000 categories

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

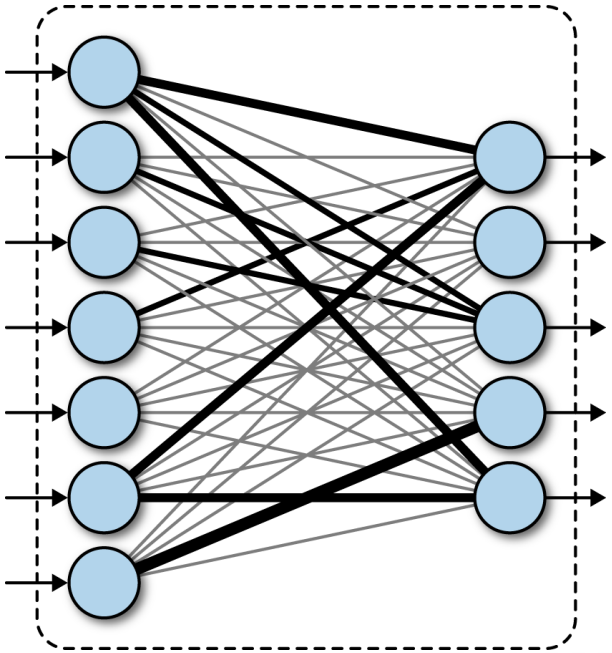
Image Classification

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

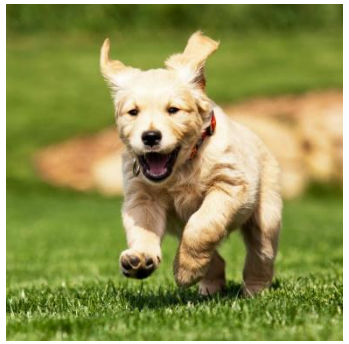
- Training data $\{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^{N}$

Image label

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

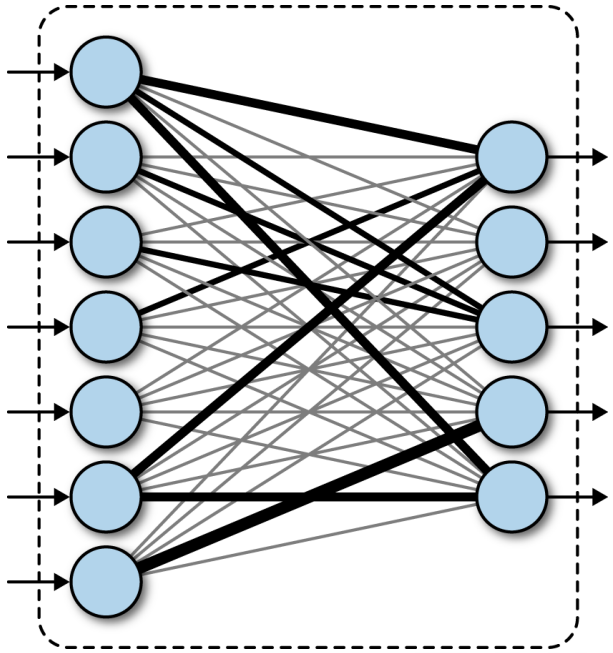
Ground truth category

Image Classification



$224 \times 224 \times 3$

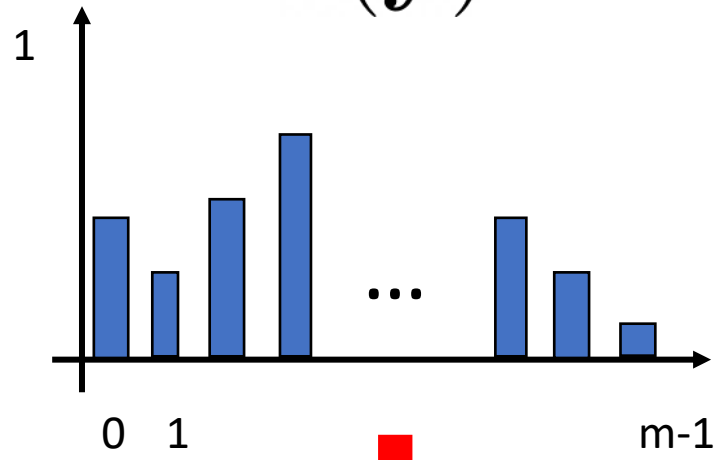
$W : m \times n$



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

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

$\sigma(\mathbf{y})$



The goal of training

Ground truth

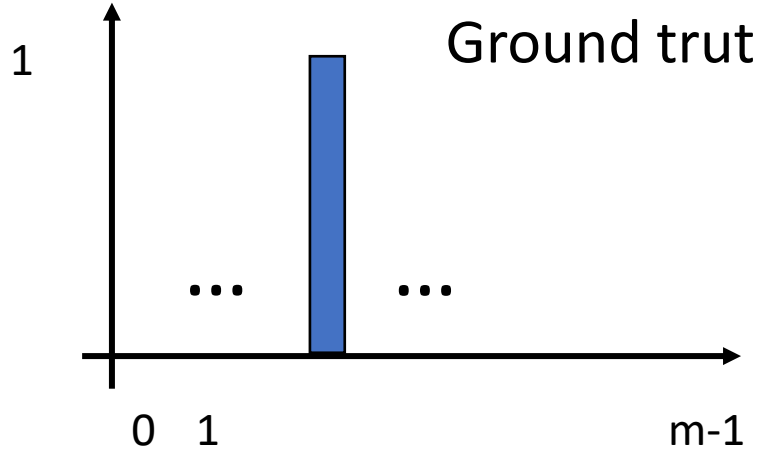


Image Classification

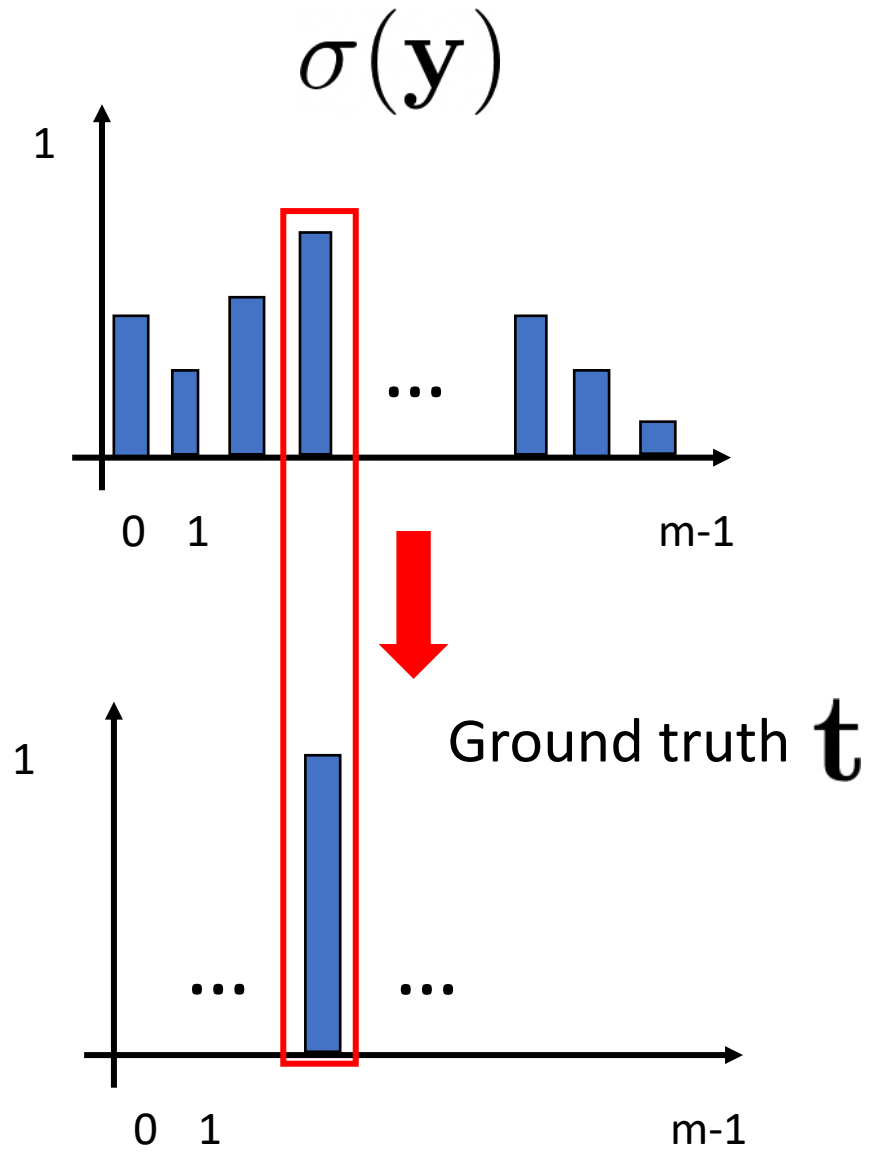
- Cross entropy loss function

Cross entropy between two distributions
(measure distance between distributions)

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

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

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



Training

- Cross entropy loss function

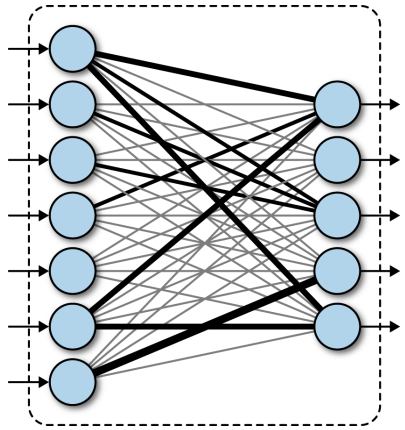
$$\mathbf{y} = W\mathbf{x}$$

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

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



Training

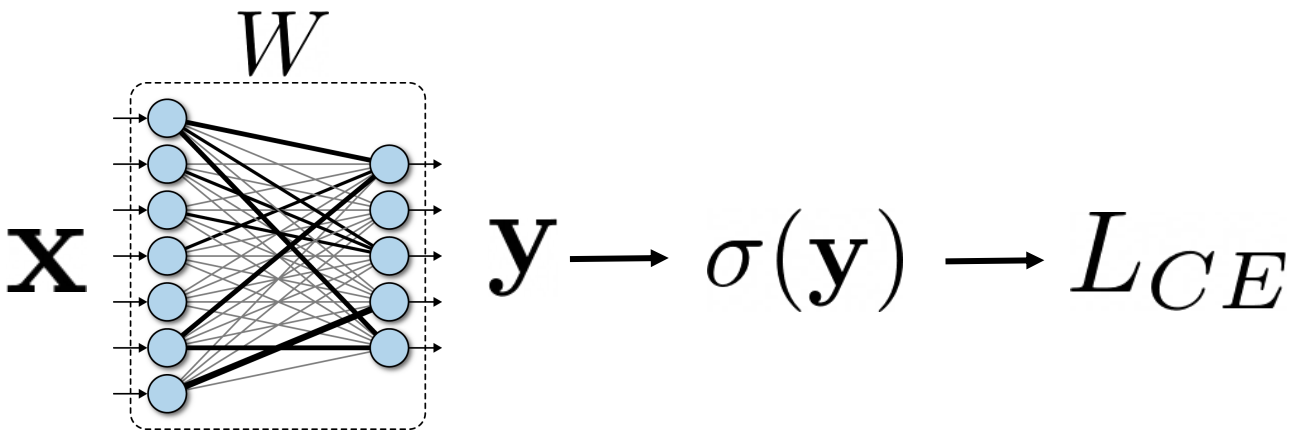
- Gradient descent

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

Learning rate

- Chain rule

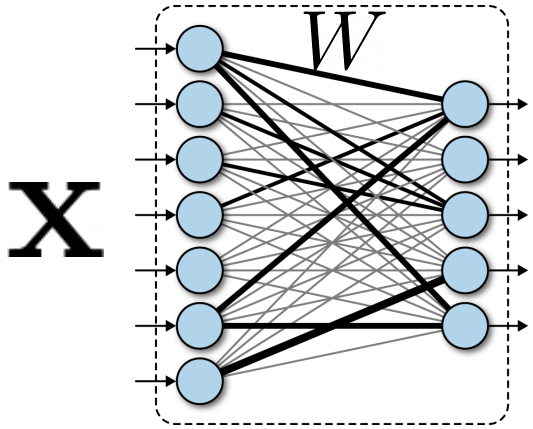
$$\frac{\partial L}{\partial W} = \frac{\partial L}{\partial \sigma(\mathbf{y})} \frac{\partial \sigma(\mathbf{y})}{\partial \mathbf{y}} \frac{\partial \mathbf{y}}{\partial W}$$



Training

- Gradient descent

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



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

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

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

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

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

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

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

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

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

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

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

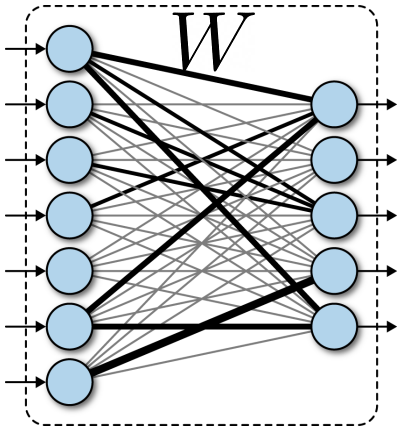
$$\delta_{ij} = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases}$$

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

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

Learning rate

Back-propagation

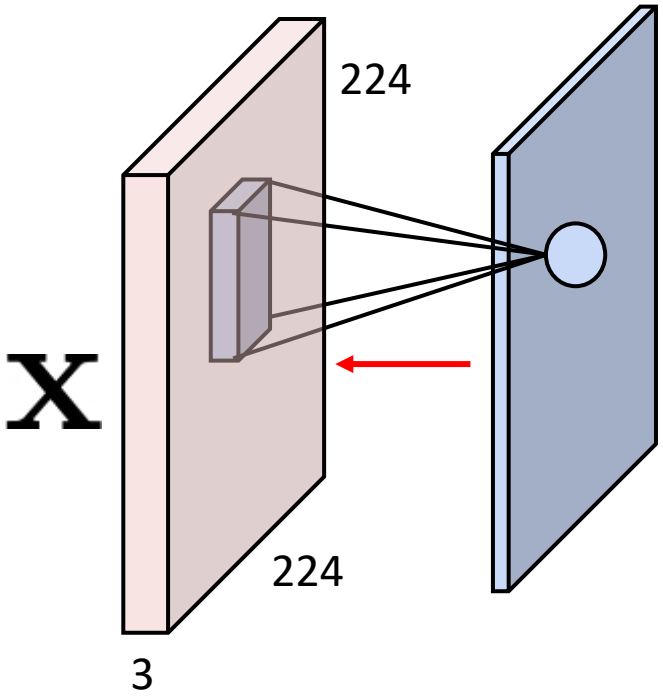


$$\mathbf{X} \quad \mathbf{y} \rightarrow \sigma(\mathbf{y}) \rightarrow L$$

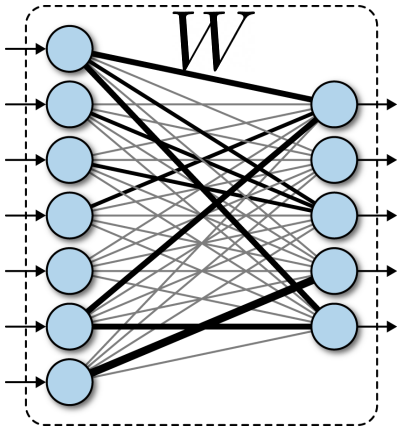
$$\frac{\partial \mathbf{y}}{\partial W} \qquad \frac{\partial \sigma(\mathbf{y})}{\partial \mathbf{y}} \qquad \frac{\partial L}{\partial \sigma(\mathbf{y})}$$

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

Back-propagation



$$\mathbf{X}$$



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

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

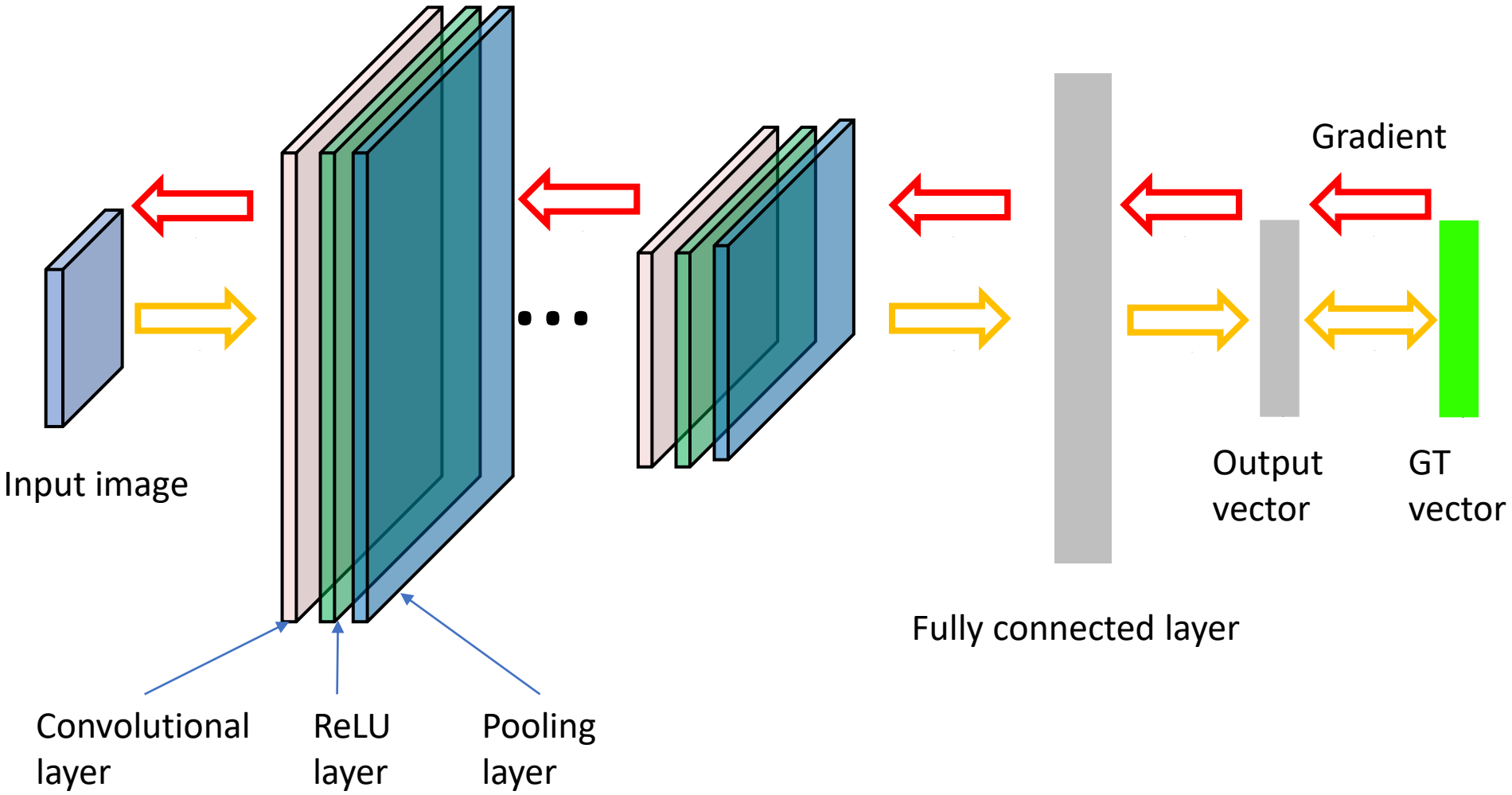
Conv filter

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

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

PyTorch Basics

- https://docs.pytorch.org/tutorials/beginner/basics/intro.html

- Install PyTorch https://pytorch.org/get-started/locally/

Working with Data

```
import torch
from torch import nn
from torch.utils.data import DataLoader
from torchvision import datasets
from torchvision.transforms import ToTensor
```

```
# Download training data from open datasets.
training_data = datasets.FashionMNIST(
    root="data",
    train=True,
    download=True,
    transform=ToTensor(),
)

# Download test data from open datasets.
test_data = datasets.FashionMNIST(
    root="data",
    train=False,
    download=True,
    transform=ToTensor(),
)
```

Working with Data

```
batch_size = 64

# Create data loaders.
train_dataloader = DataLoader(training_data, batch_size=batch_size)
test_dataloader = DataLoader(test_data, batch_size=batch_size)

for X, y in test_dataloader:
    print(f"Shape of X [N, C, H, W]: {X.shape}")
    print(f"Shape of y: {y.shape} {y.dtype}")
    break
```

Out:

```
Shape of X [N, C, H, W]: torch.Size([64, 1, 28, 28])
Shape of y: torch.Size([64]) torch.int64
```

Creating Models

```
device = torch.accelerator.current_accelerator().type if torch.accelerator.is_available() else "cpu"
print(f"Using {device} device")

# Define model
class NeuralNetwork(nn.Module):
    def __init__(self):
        super().__init__()
        self.flatten = nn.Flatten()
        self.linear_relu_stack = nn.Sequential(
            nn.Linear(28*28, 512),
            nn.ReLU(),
            nn.Linear(512, 512),
            nn.ReLU(),
            nn.Linear(512, 10)
        )

    def forward(self, x):
        x = self.flatten(x)
        logits = self.linear_relu_stack(x)
        return logits

model = NeuralNetwork().to(device)
print(model)
```

Optimizing Model Parameters

```
loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr=1e-3)
```

```
def train(dataloader, model, loss_fn, optimizer):
    size = len(dataloader.dataset)
    model.train()
    for batch, (X, y) in enumerate(dataloader):
        X, y = X.to(device), y.to(device)

        # Compute prediction error
        pred = model(X)
        loss = loss_fn(pred, y)

        # Backpropagation
        loss.backward()
        optimizer.step()
        optimizer.zero_grad()

        if batch % 100 == 0:
            loss, current = loss.item(), (batch + 1) * len(X)
            print(f"loss: {loss:>7f}  [{current:>5d}/{size:>5d}]")
```

Testing the Model

```
def test(dataloader, model, loss_fn):
    size = len(dataloader.dataset)
    num_batches = len(dataloader)
    model.eval()
    test_loss, correct = 0, 0
    with torch.no_grad():
        for X, y in dataloader:
            X, y = X.to(device), y.to(device)
            pred = model(X)
            test_loss += loss_fn(pred, y).item()
            correct += (pred.argmax(1) == y).type(torch.float).sum().item()
    test_loss /= num_batches
    correct /= size
    print(f"Test Error: \n Accuracy: {(100*correct):>0.1f}%, Avg loss: {test_loss:>8f} \n")
```

Saving and Loading Models

```
torch.save(model.state_dict(), "model.pth")
print("Saved PyTorch Model State to model.pth")
```

```
model = NeuralNetwork().to(device)
model.load_state_dict(torch.load("model.pth", weights_only=True))
```

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/