COMP701 Nature Inspired Computing

Artificial Neural Networks: Training Practice

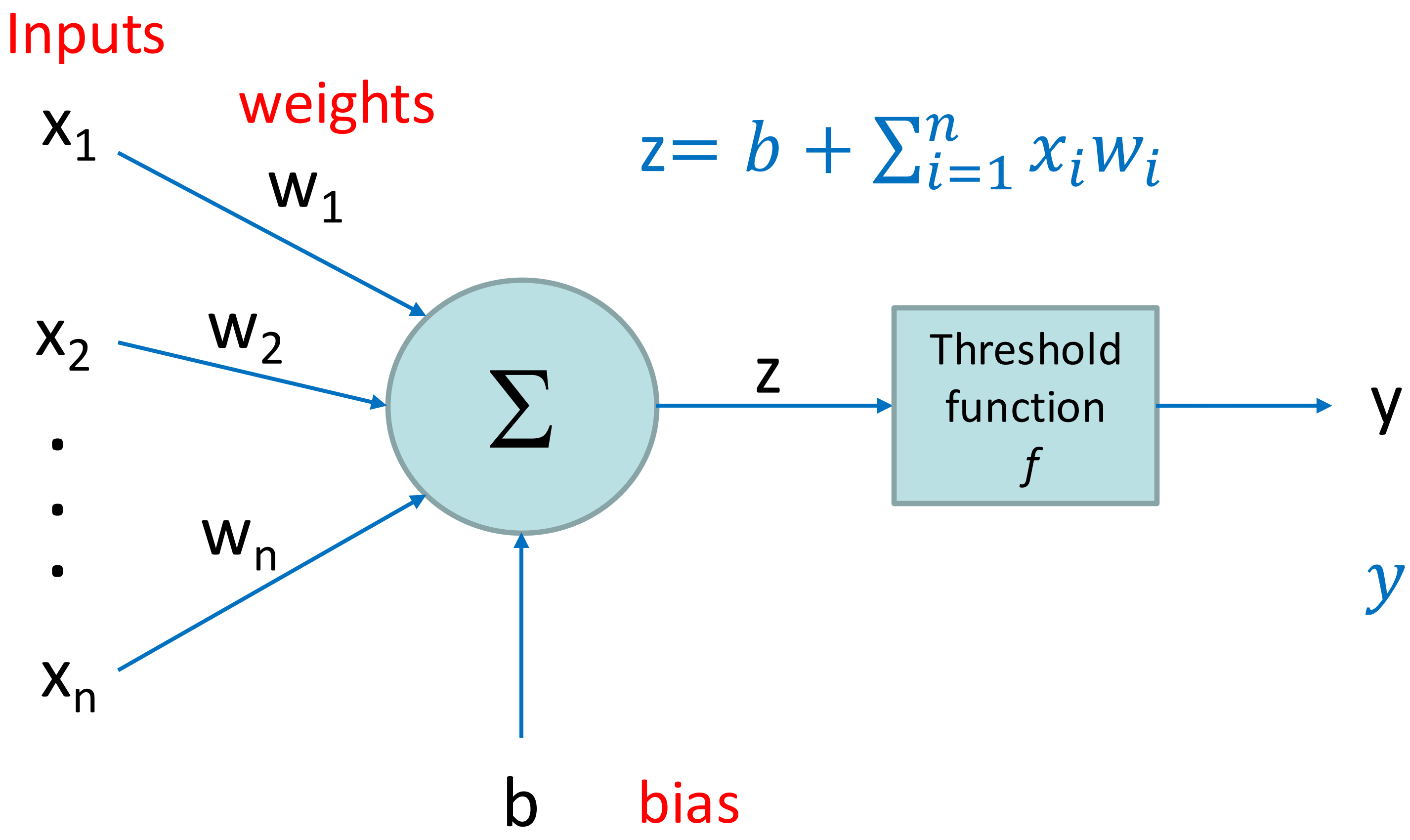
Recap

- Evolution Algorithms (biology-inspired)
  ~~Assignment Part 1~~

- Swarm Algorithms (social-inspired)
  Assignment Part 2 (Due Sep 27, next Friday)

- Neural Networks (brain-inspired)
  Assignment Part 3

Recap: Perceptron



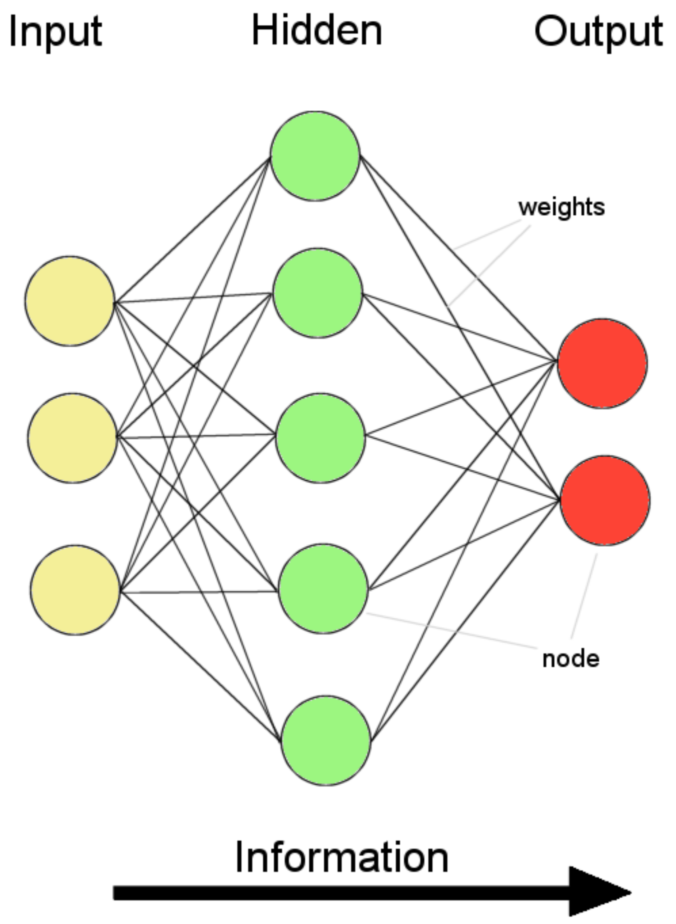
$$z = b + \sum_{i=1}^{n} x_i w_i$$

$$y = f(z)$$

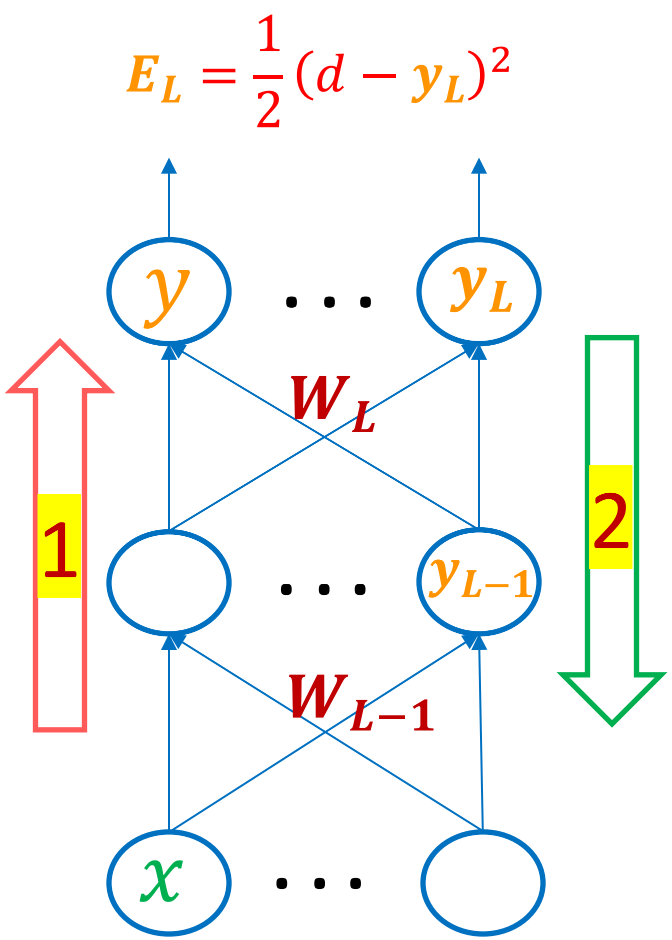
Recap: Neural Networks

Artificial neurons are connected together to form an *artificial neural network*

The *architecture* of an artificial neural network is the way the layers are organized



Backpropagation (BP) Algorithm



Training Pipeline: data $(x_i, y_i)_{i=1}^{N}$

1. Forward pass

Input x $(i.e., y_0)$ with W_1 … W_L to obtain y_1 … y_L

2. Backward pass

Output y_L and E_L with the chain rule to update W_1 … W_L

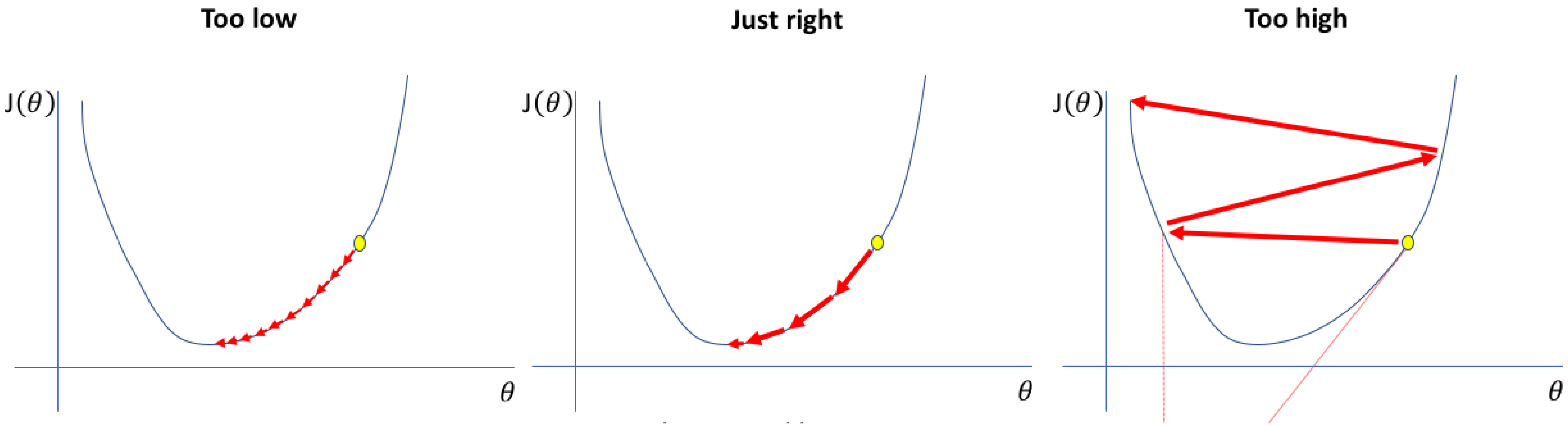
Iterate 1 and 2 until E_L is small or after certain number of iters

Training Practice

- Momentum
- Vanishing Gradient
- Mini-batch Updates
- Cross-entropy Loss

Momentum

- Convergence can be very slow
- Ways to speed up convergence:
  - Increase learning rate
    - weights may oscillate if set too high



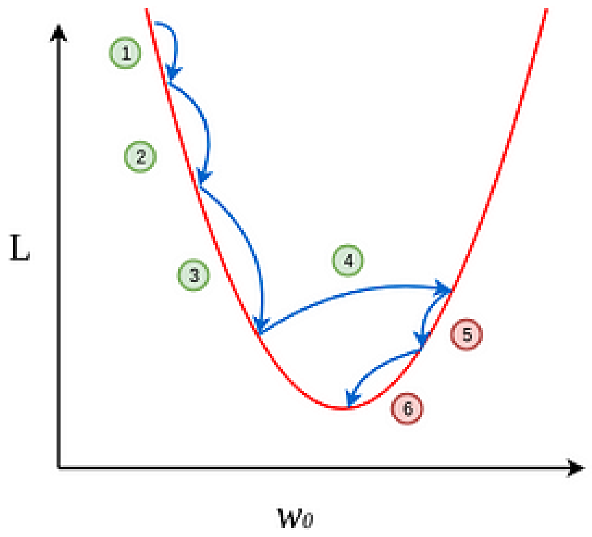
Momentum

- Convergence can be very slow
- Ways to speed up convergence:
  - Introduce a momentum term

Keep a portion of previous change

$$\Delta w_{ij}(t) = -\eta \frac{\partial E}{\partial w_{ij}(t)} + \alpha \Delta w_{ij}(t-1)$$

Usually around 0.5



Vanishing Gradient 1

- If the weights W are large, y_j becomes large
  $f(y_j)$ approaches 1, and $f'(y_j) = \frac{\partial E_L}{\partial W_L} \to 0$
- Hence weights do not change: $W = W + \frac{\partial E_L}{\partial W_L}$
- Possible solution: Add a small constant, say 0.1, to f'

Vanishing Gradient 2

- Error signal is attenuated as it goes backwards through multiple layers
- Consequently, input-to-hidden weights learn more slowly than hidden-to-output weights

- Possible solution: use <u>different learning rates</u> for different layers

Mini-Batch Update

- Dataset $(x_i, y_i)_{i=1}^{N}, e.g., N = 10{,}000$
- Online update – (one by one)
  - Weights are updated after *each* training pattern
  - Computationally demanding
- Mini-Batch update:
  - Divide training dataset into small batches (mini-batch)
  - Weights are accumulated and updated for each *mini-batch* data
  - Both efficient and more accurate

  E.g., batch_size=8, 16, …, 256

Variations of Gradient Algorithms

- Stochastic Gradient Descent (SGD) – Batch update
- Root Mean Square Propagation (RMSprop)
- Adaptive Gradient Algorithm (Adagrad)
- Adadelta – improved version of Adagrad
- Adaptive Moment Estimation (Adam) – keeps separate learning rate for each weight
- Adamax – variant of Adam

Don’t worry, PyTorch has all the implementations!!!!

Cross-entropy Loss Function

- An alternative to squared error suitable for binary outputs

$$E = \frac{1}{2}(d - y)^2$$

- Cross entropy function:

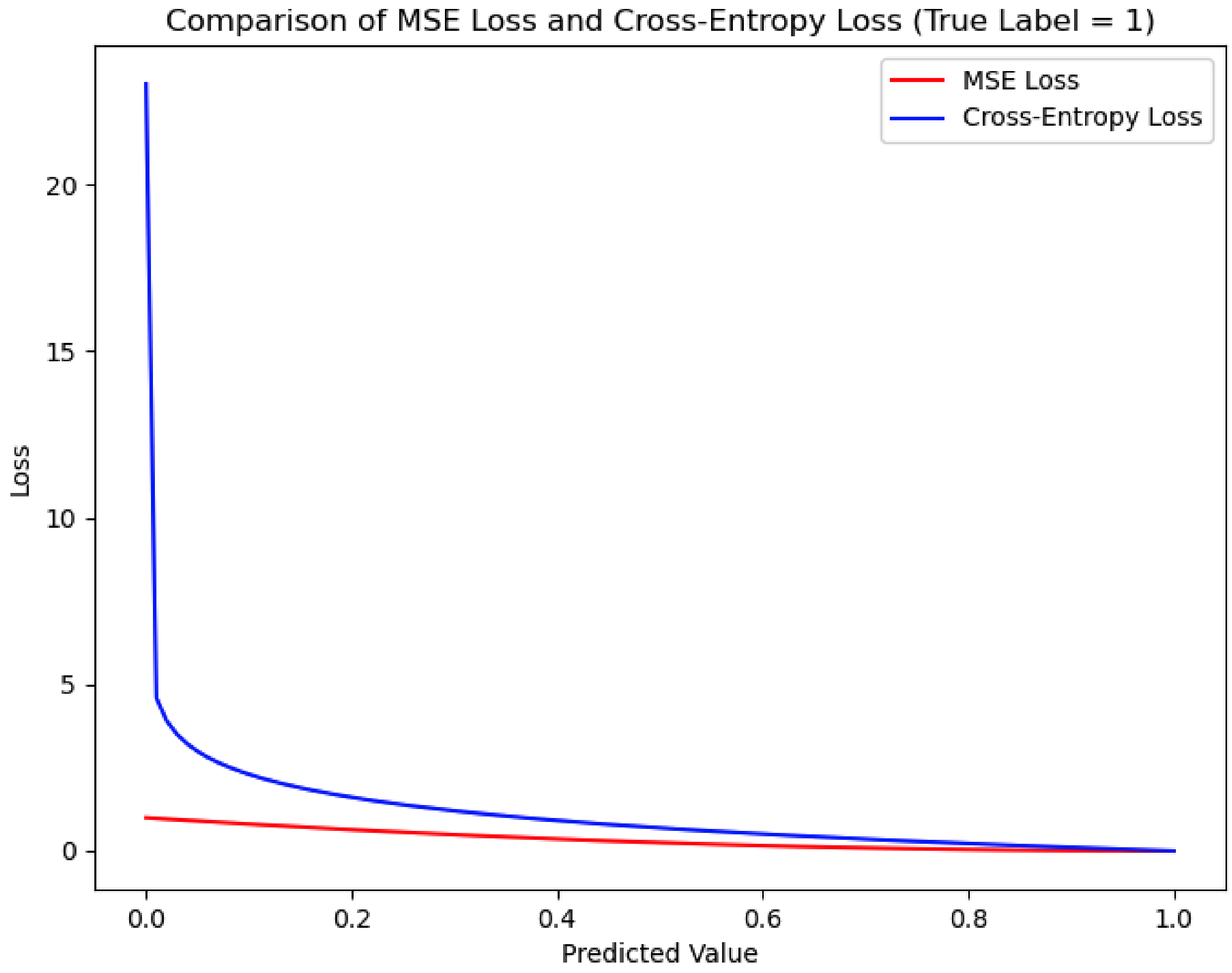
$$E = \sum_p \left[d^p \log \frac{d^p}{y^p} + (1 - d^p) \log \frac{1 - d^p}{1 - y^p} \right]$$

Target output

Computed output

- Heavily penalizes very wrong outputs
- Leads to faster convergence for some problems and avoids local minima

Cross-entropy Loss Function

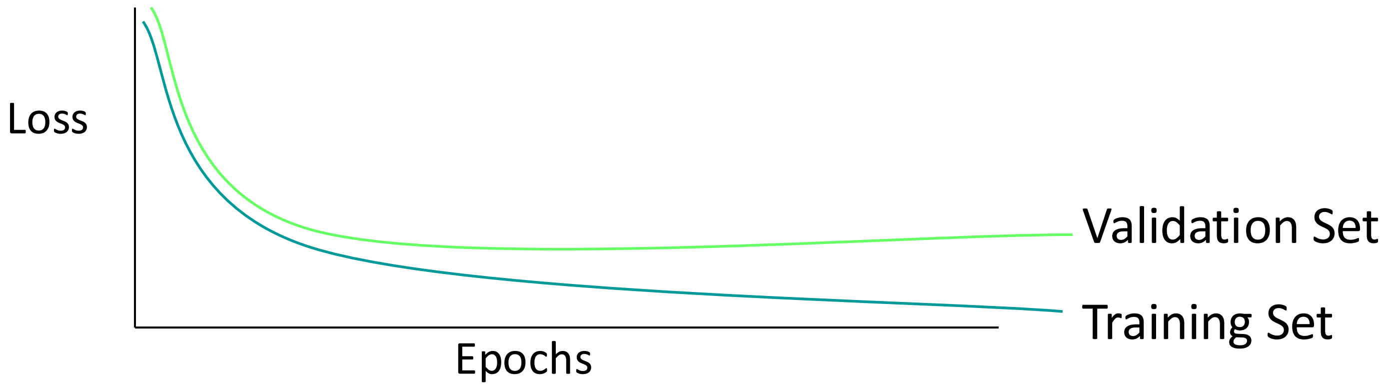


Overfitting

- The model produces very small errors for the training set but large errors for non-training ("unseen") data
- Could be due to the model is too large for the given amount of training data
- Or the model simply remembers every training data point

Overcome Overfitting

- Use more training data
- Use a validation set – a set of data separate from the training and test sets – stop when there is no more improvement to the validation set



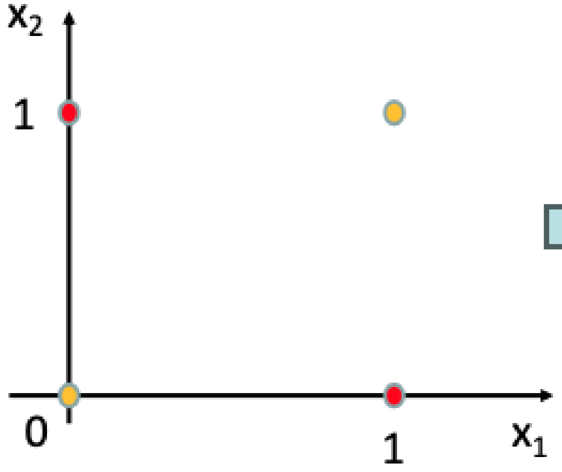
Any Question so far?

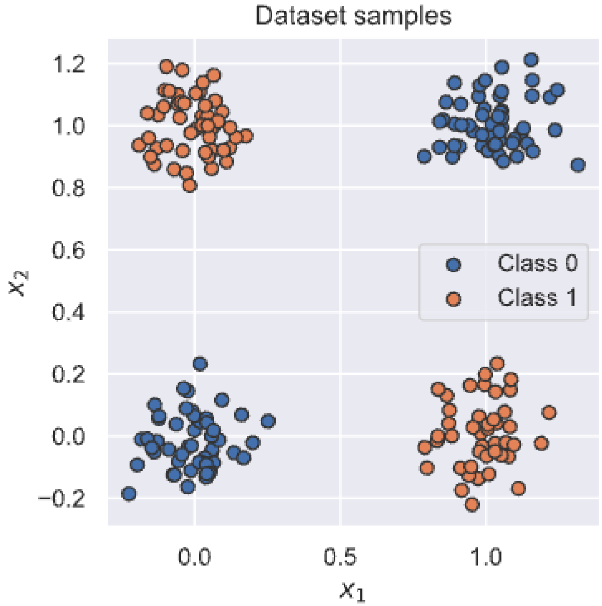
Exclusive OR Classification Problem

PyTorch for XOR classification (7.3)

x_1	x_2	y
0	1	1
1	0	1
0	0	0
1	1	0

Truth table



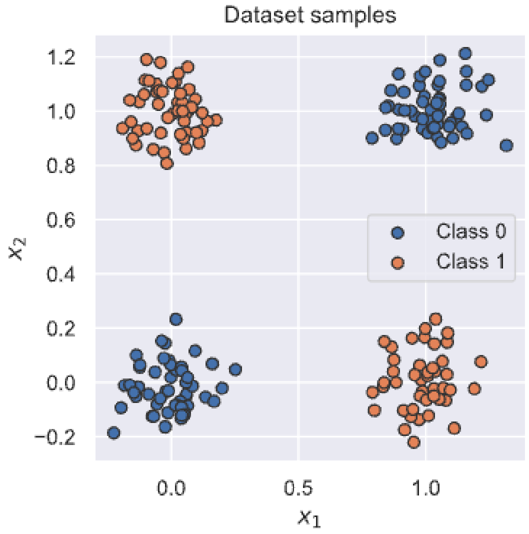


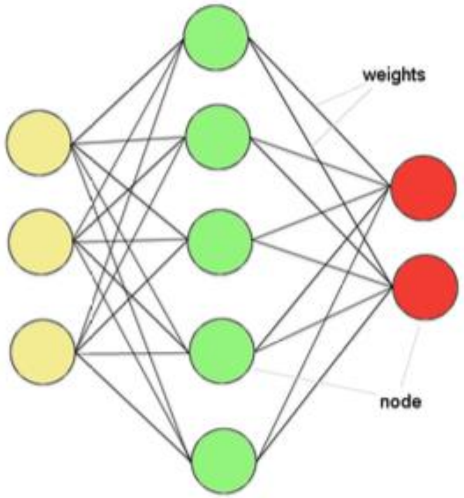
Simulation data

Recap of ANN and PyTorch

PyTorch for XOR classification (7.3)

x_1	x_2	y
0	1	1
1	0	1
0	0	0
1	1	0





Class 0?
Class 1?

Input: (x1, x2)

Model: Multilayer Networks

Output: y = 0 or 1

Recap of ANN and PyTorch

Input: (x1, x2)

The Data

```
import torch.utils.data as data
```

```
class XOR_Dataset(data.Dataset):
```

⇩

```
train_dataset = XOR_Dataset(size=2500)
train_data_loader = data.DataLoader(train_dataset,
```

Model: Multilayer Networks

The Model

```
class XOR_Classifier(nn.Module):

    def __init__(self, num_inputs, num_hidden, num_outputs):
        super().__init__()
        # Initialize modules needed for this network
        self.linear1 = nn.Linear(num_inputs, num_hidden)
        self.activation = nn.Tanh()
        self.linear2 = nn.Linear(num_hidden, num_outputs)

    def forward(self, x):
        # Compute output given an input
        x = self.linear1(x)
        x = self.activation(x)
        x = self.linear2(x)
        return x
```

Output: preds = 0 or 1

Prediction

```
for data_inputs, data_labels in data_loader:

    preds = model(data_inputs)
```

```
loss = loss_module(preds, data_labels.float())

optimizer.zero_grad()
loss.backward()

optimizer.step()
```

```
loss_module = nn.BCEWithLogitsLoss()
optimizer = torch.optim.SGD(model.parameters(), lr=0.1)
```

Important Note

- Make sure **PyTorch** is Installed (Workshop 7)!
- Alternatively, you can use Google **Colab**
- Finish **Task 7.3** (Basic code for Assignment Part 3)

Any Question so far?

COMP815 Nature Inspired Computing

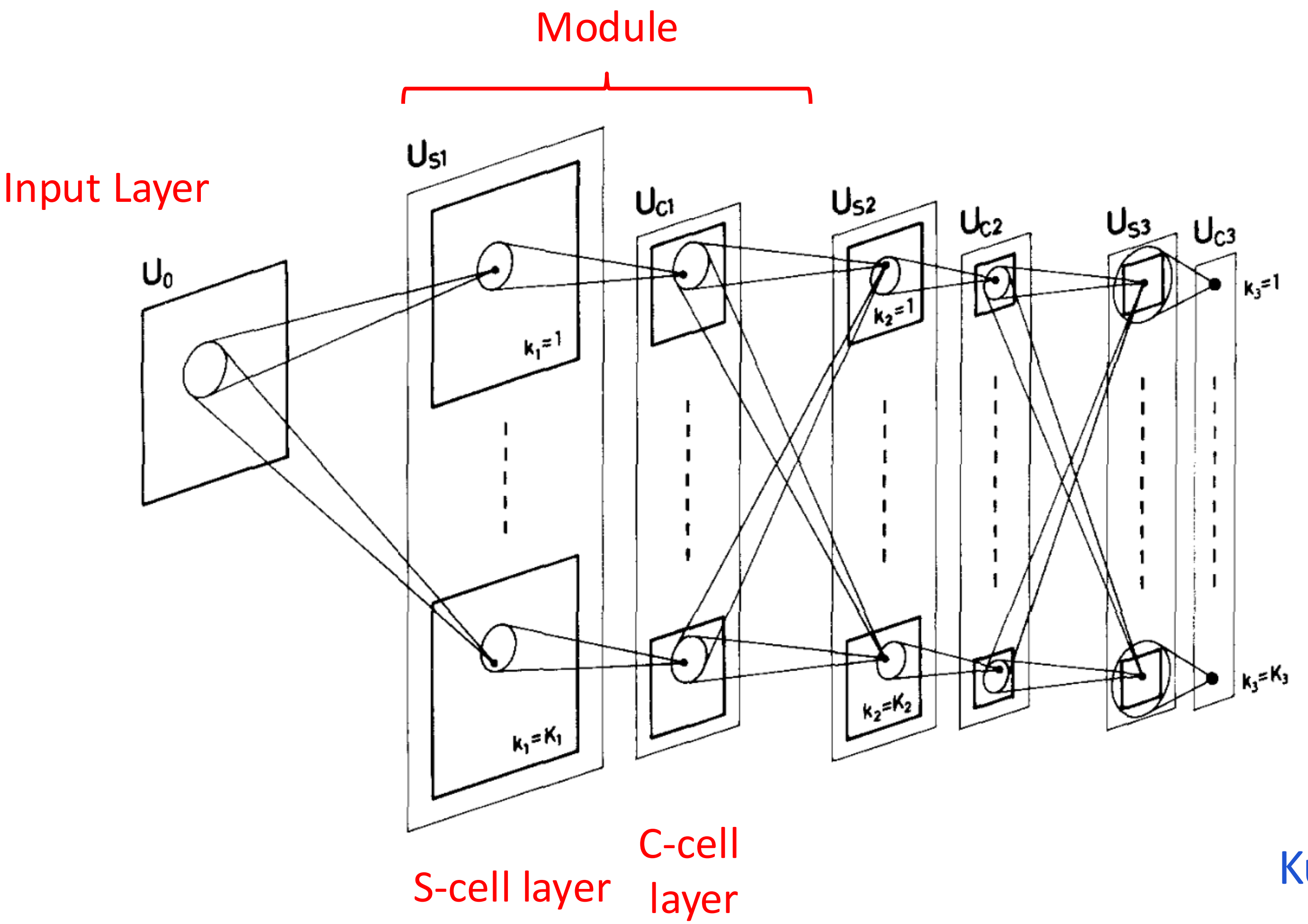
Convolutional Neural Networks

Visual Cortex

- Research by Hubel and Wiesel (Nobel Prize 1981) on the visual cortex in 1960s

- Two kinds of cells
  - Simple cells – respond to edges and lines of particular orientation in certain parts of a scene
  - Complex cells – respond to edges and lines at any location in the scene

Neocognitron



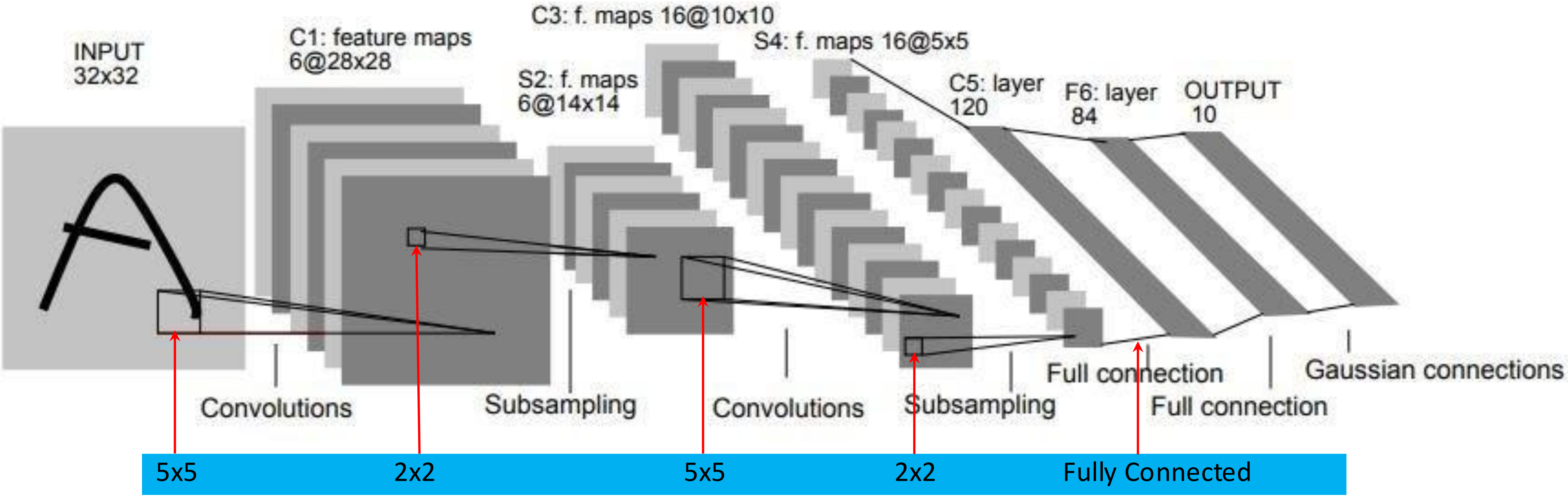
Kunihiko Fukushima 1980s

Pick up local features

Aggregate these features into representation in a local neighbourhood

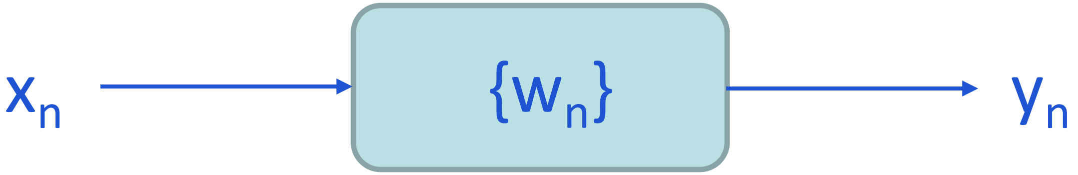
LeNet

The first Convolutional Neural Network



Linear Convolution

- Linear convolution in linear systems theory:



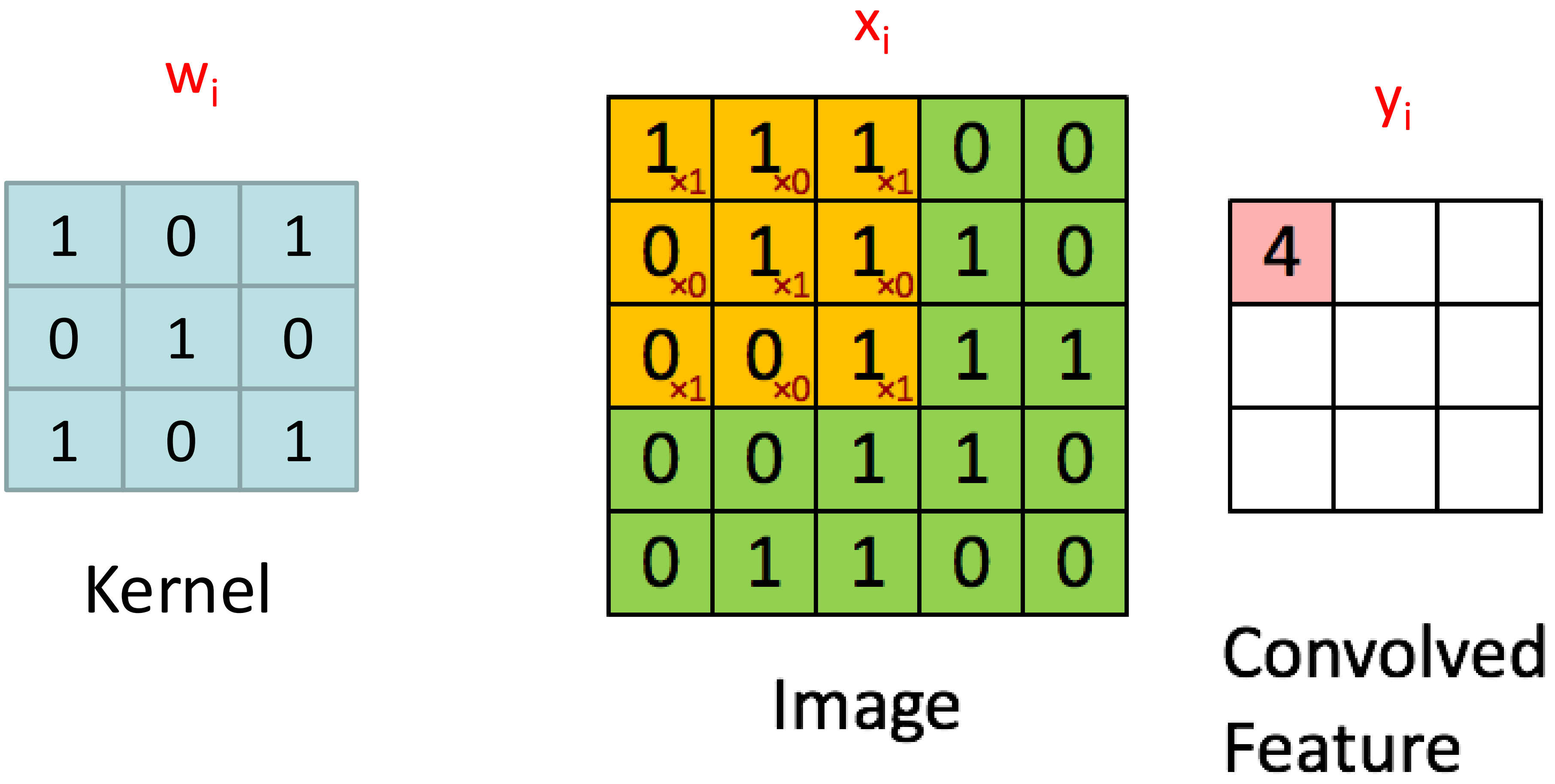
$$y_n = \sum_{i=0}^{N-1} w_i x_{N-i}$$

- Can also be extended to higher dimensions

- The "convolution" used in the convolution layer is actually a *correlation*:

$$y_n = \sum_{i=0}^{N-1} w_i x_i$$

Example



w_i
1 0 1
0 1 0
1 0 1
Kernel
x_i
$1_{\times 1}$ $1_{\times 0}$ $1_{\times 1}$ 0 0
$0_{\times 0}$ $1_{\times 1}$ $1_{\times 0}$ 1 0
$0_{\times 1}$ $0_{\times 0}$ $1_{\times 1}$ 1 1
0 0 1 1 0
0 1 1 0 0
Image
y_i
4
Convolved Feature