FVE3011

# Image fundamentals, Neural Network

담당교수: 최 학남 (xncui@inha.ac.kr)

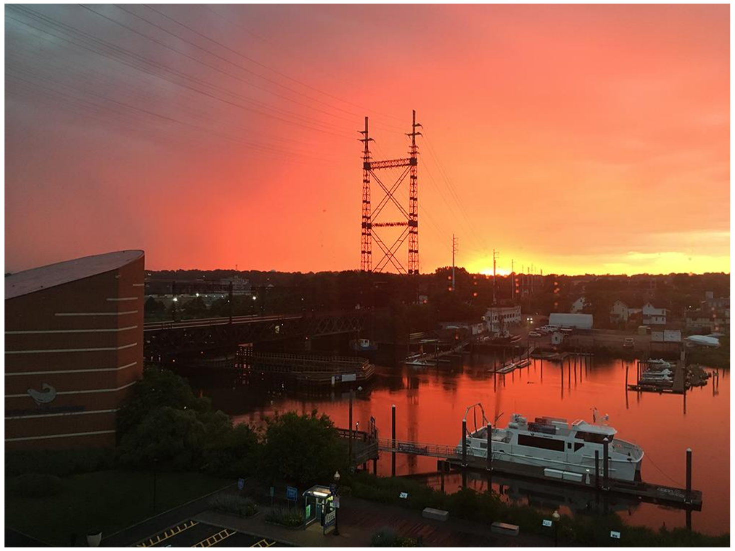
인하대학교

## Contents

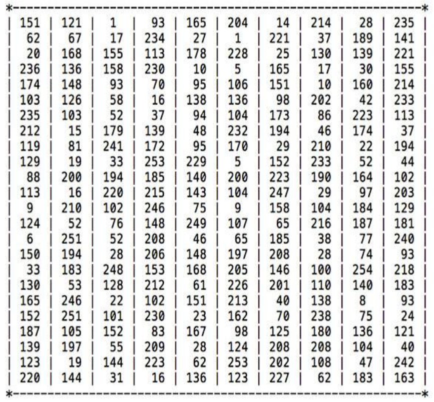
- Image fundamentals
- Neural Network

## Pixels

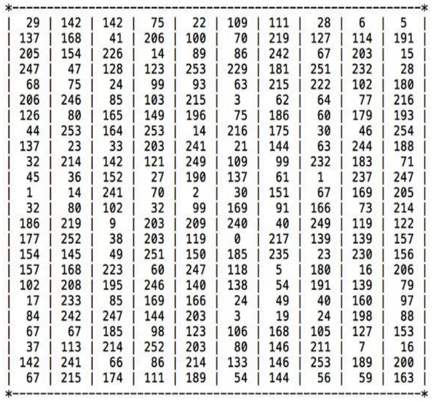
- Pixels are the raw building blocks of an image. Every image consists of a set of pixels.



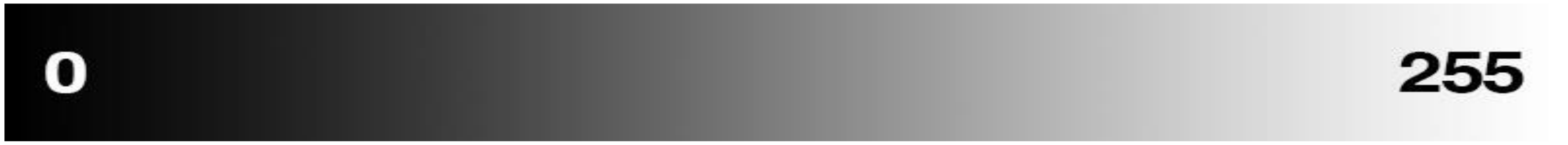
This image is 1,000 pixels wide and 750 pixels tall, for a total of 1‹000x750=750‹000 total pixels.



| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 151 | 121 | 1 | 93 | 165 | 204 | 14 | 214 | 28 | 235 |
| 62 | 67 | 17 | 234 | 27 | 1 | 221 | 37 | 189 | 141 |
| 20 | 168 | 155 | 113 | 178 | 228 | 25 | 130 | 139 | 221 |
| 236 | 136 | 158 | 230 | 10 | 5 | 165 | 17 | 30 | 155 |
| 174 | 148 | 93 | 70 | 95 | 106 | 151 | 10 | 160 | 214 |
| 103 | 126 | 58 | 16 | 138 | 136 | 98 | 202 | 42 | 233 |
| 235 | 103 | 52 | 37 | 94 | 104 | 173 | 86 | 223 | 113 |
| 212 | 15 | 179 | 139 | 48 | 232 | 194 | 46 | 174 | 37 |
| 119 | 81 | 241 | 172 | 95 | 170 | 29 | 210 | 22 | 194 |
| 129 | 19 | 33 | 253 | 229 | 5 | 152 | 233 | 52 | 44 |
| 88 | 200 | 194 | 185 | 140 | 200 | 223 | 190 | 164 | 102 |
| 113 | 16 | 220 | 215 | 143 | 104 | 247 | 29 | 97 | 203 |
| 9 | 210 | 102 | 246 | 75 | 9 | 158 | 104 | 184 | 129 |
| 124 | 52 | 76 | 148 | 249 | 107 | 65 | 216 | 187 | 181 |
| 6 | 251 | 52 | 208 | 46 | 65 | 185 | 38 | 77 | 240 |
| 150 | 194 | 28 | 206 | 148 | 197 | 208 | 28 | 74 | 93 |
| 33 | 183 | 248 | 153 | 168 | 205 | 146 | 100 | 254 | 218 |
| 130 | 53 | 128 | 212 | 61 | 226 | 201 | 110 | 140 | 183 |
| 165 | 246 | 22 | 102 | 151 | 213 | 40 | 138 | 8 | 93 |
| 152 | 251 | 101 | 230 | 23 | 162 | 70 | 238 | 75 | 24 |
| 187 | 105 | 152 | 83 | 167 | 98 | 125 | 180 | 136 | 121 |
| 139 | 197 | 55 | 209 | 28 | 124 | 208 | 208 | 104 | 40 |
| 123 | 19 | 144 | 223 | 62 | 253 | 202 | 108 | 47 | 242 |
| 220 | 144 | 31 | 16 | 136 | 123 | 227 | 62 | 183 | 163 |



| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 29 | 142 | 142 | 75 | 22 | 109 | 111 | 28 | 6 | 5 |
| 137 | 168 | 41 | 206 | 100 | 70 | 219 | 127 | 114 | 191 |
| 205 | 154 | 226 | 14 | 89 | 86 | 242 | 67 | 203 | 15 |
| 247 | 47 | 128 | 123 | 253 | 229 | 181 | 251 | 232 | 28 |
| 68 | 75 | 24 | 99 | 93 | 63 | 215 | 222 | 102 | 180 |
| 206 | 246 | 85 | 103 | 215 | 3 | 62 | 64 | 77 | 216 |
| 126 | 80 | 165 | 149 | 196 | 75 | 186 | 60 | 179 | 193 |
| 44 | 253 | 164 | 253 | 14 | 216 | 175 | 30 | 46 | 254 |
| 137 | 23 | 33 | 203 | 241 | 21 | 144 | 63 | 244 | 188 |
| 32 | 214 | 142 | 121 | 249 | 109 | 99 | 232 | 183 | 71 |
| 45 | 36 | 152 | 27 | 190 | 137 | 61 | 1 | 237 | 247 |
| 1 | 14 | 241 | 70 | 2 | 30 | 151 | 67 | 169 | 205 |
| 32 | 80 | 102 | 32 | 99 | 169 | 91 | 166 | 73 | 214 |
| 186 | 219 | 9 | 203 | 209 | 240 | 40 | 249 | 119 | 122 |
| 177 | 252 | 38 | 203 | 119 | 0 | 217 | 139 | 139 | 157 |
| 154 | 145 | 49 | 251 | 150 | 185 | 235 | 23 | 230 | 156 |
| 157 | 168 | 223 | 60 | 247 | 118 | 5 | 180 | 16 | 206 |
| 102 | 208 | 195 | 246 | 140 | 138 | 54 | 191 | 139 | 79 |
| 17 | 233 | 85 | 169 | 166 | 24 | 49 | 40 | 160 | 97 |
| 84 | 242 | 247 | 144 | 203 | 3 | 19 | 24 | 198 | 88 |
| 67 | 67 | 185 | 98 | 123 | 106 | 168 | 105 | 127 | 153 |
| 37 | 113 | 214 | 252 | 203 | 80 | 146 | 211 | 7 | 16 |
| 142 | 241 | 66 | 86 | 214 | 133 | 146 | 253 | 189 | 200 |
| 67 | 215 | 174 | 111 | 189 | 54 | 144 | 56 | 59 | 163 |

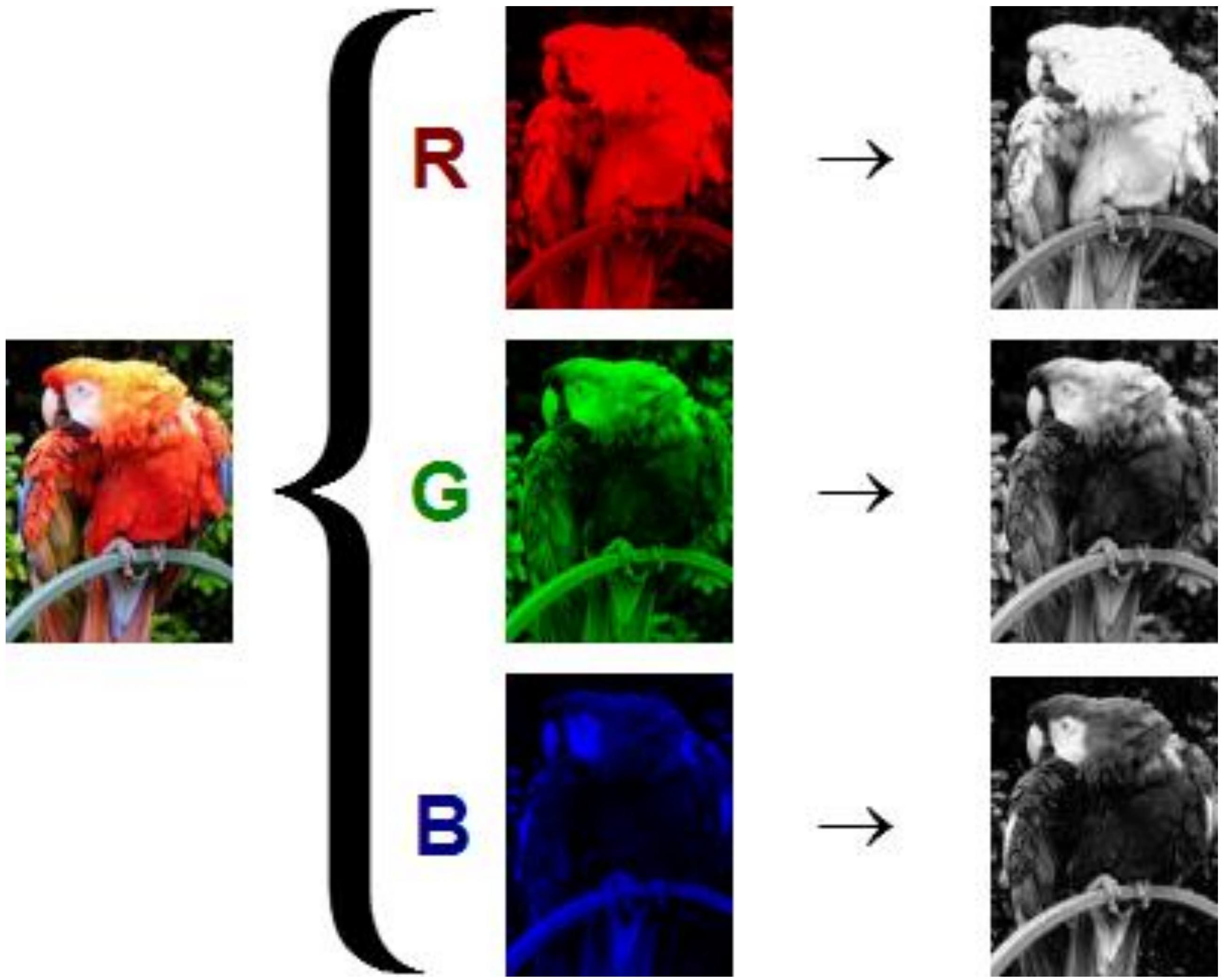


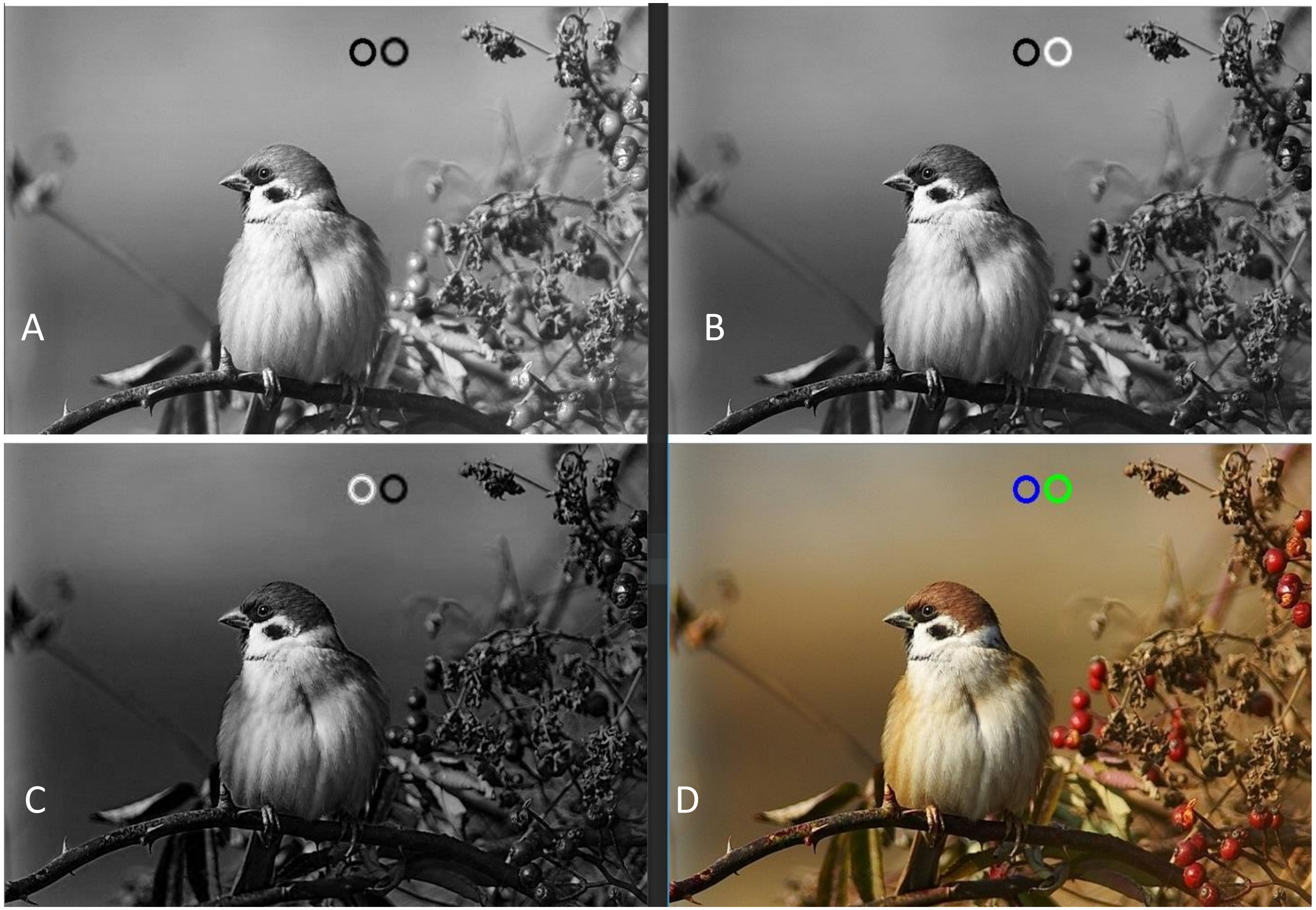
# Gray and Color image

Gray image: 256x256x**1**

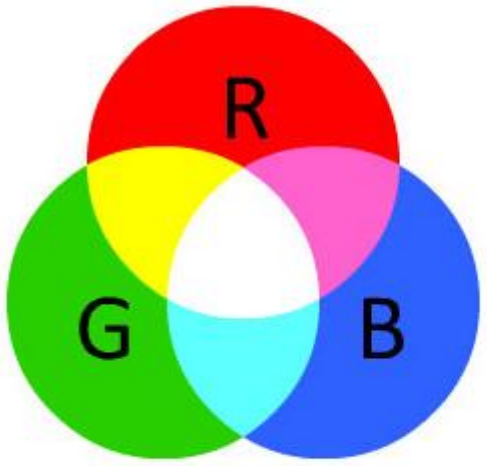
Color image: 256x256x**3**

# Color image

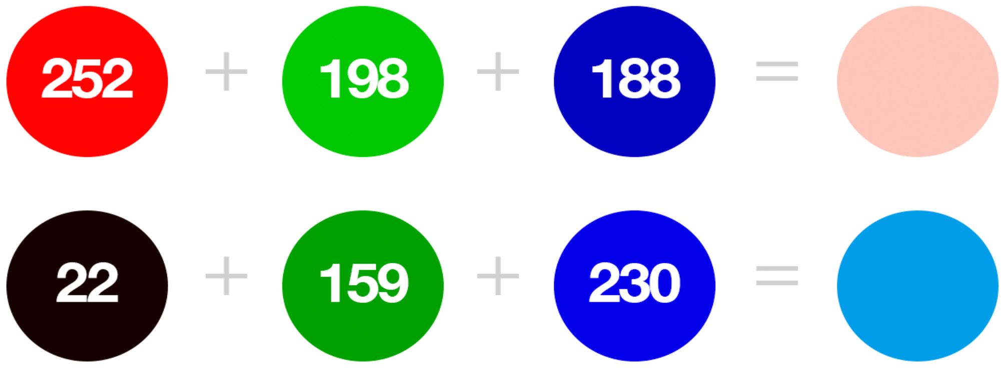



A
B
C
D

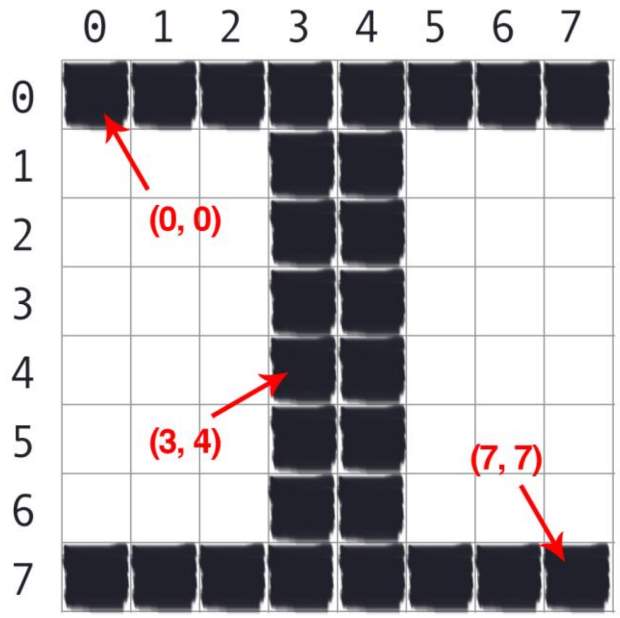
# Color image



Left: The RGB color space is additive. The more red, green and blue you mix together, the closer you get to white. Right: The RGB cube.



# Image coordinate system



```
import cv2
image = cv2.imread("example.png")
print(image.shape)
cv2.imshow("Image", image)
cv2.waitKey(0)
```

```
(b, g, r) = image[20, 100] # accesses pixel at x=100, y=20
(b, g, r) = image[75, 25] # accesses pixel at x=25, y=75
(b, g, r) = image[90, 85] # accesses pixel at x=85, y=90
```

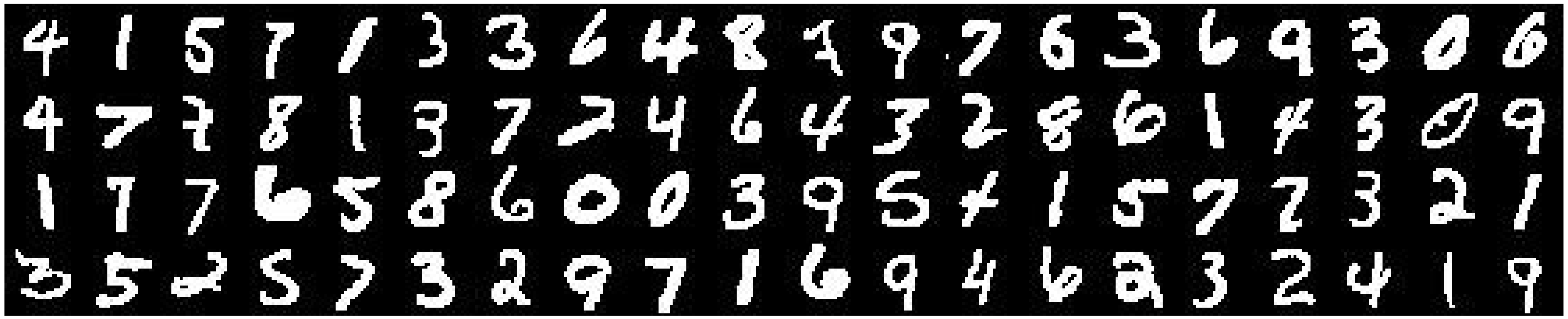
The letter “I” placed on a piece of graph paper. Pixels are accessed by their (x‹y)-coordinates, where we go x columns to the right and y rows down, keeping in mind that Python is zero-indexed.

## RGB and BGR ordering

- OpenCV stores RGB channels in reverse order.
  - While we normally think in terms of Red, Green, and Blue, OpenCV actually stores the pixel values in Blue, Green, Red order.
- Why does OpenCV do this?
  - The answer is simply historical reasons. Early developers of the OpenCV library chose the BGR color format **because the BGR ordering was popular among camera manufacturers** and other software developers at the time .

# Datasets - MNIST

- MNIST
  - A sample of the MNIST dataset. The goal of this dataset is to correctly classify the handwritten digits, 0-9.
  - Consists of **60,000 training images and 10,000 testing images**.
  - Each feature vector is **784-dim**, corresponding to the **28x28** grayscale pixel intensities of the image.
  - These grayscale pixel intensities are **unsigned integers**, falling into the range [0~255].

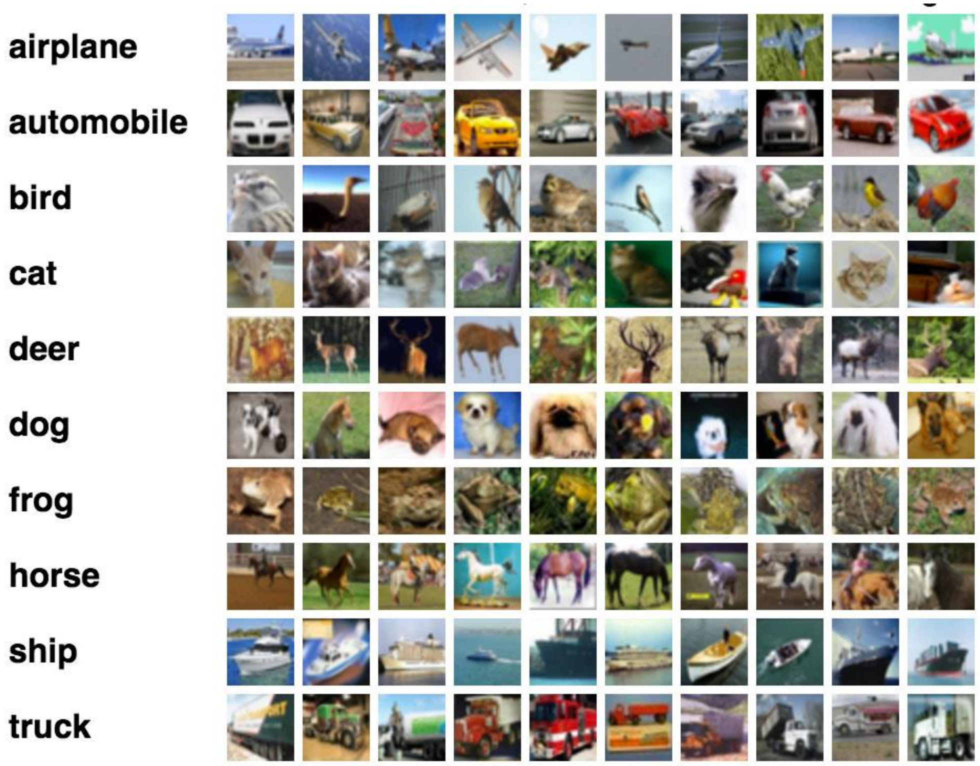


# Datasets - Animals

- A sample of the 3-class animals dataset consisting of 1,000 images per **dog, cat, and panda** class respectively for a total of **3,000 images**.

# Datasets – CIFAR-10

- CIFAR-10 consists of **60,000**, 32x32x3 (RGB) images resulting in a feature vector dimensionality of 3072.

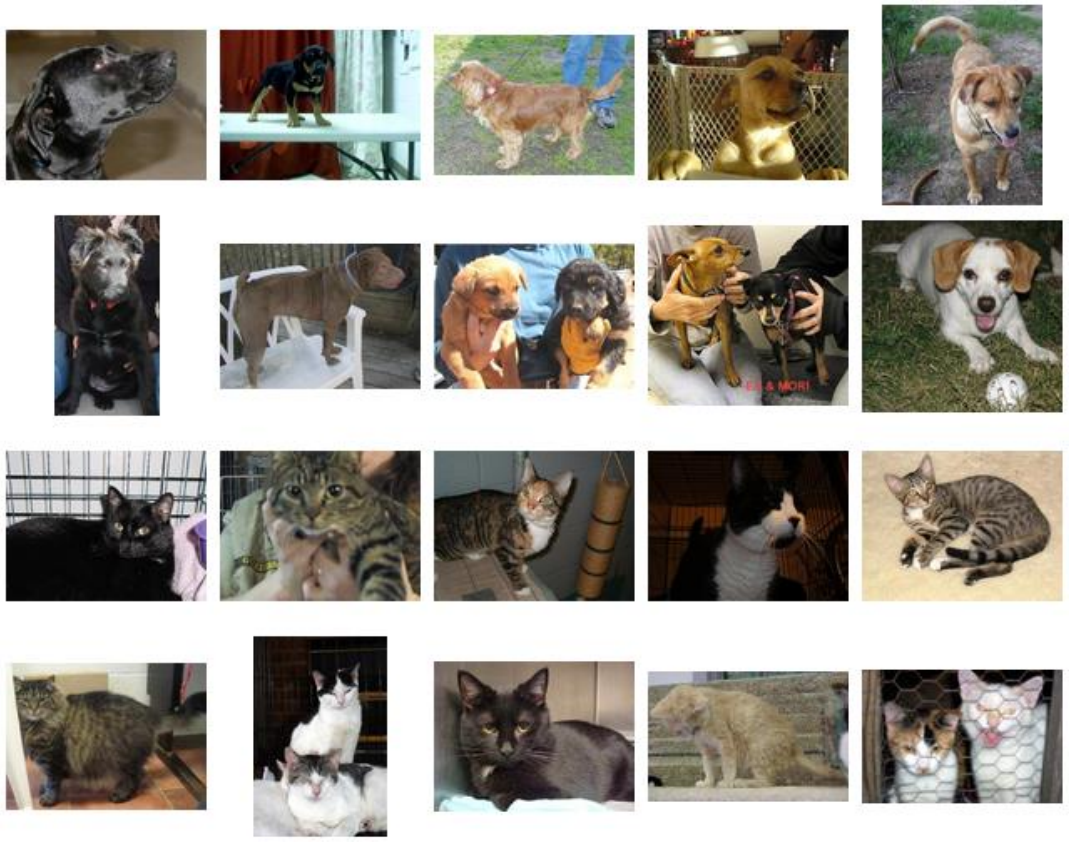


# Datasets – SMILES

- SMILES dataset consists of images of faces that are either smiling or not smiling.
- In total, there are **13,165 grayscale** images in the dataset, with each image having a size of 64x64.

# Datasets – Kaggle :dogs vs cats

- A total of **25,000 images** are provided to train your algorithm with **varying image resolutions**.



# Datasets – flowers-17

- The Flowers-17 dataset is a 17 category dataset with **80 images per class** .

# Datasets – CALTECH-101

- Introduced by Fei-Fei et al. [53] in 2004, the CALTECH-101 dataset is a popular benchmark dataset for object detection.
- The dataset of 8,677 images includes 101 categories spanning a diverse range of objects, including elephants, bicycles, soccer balls, and even human brains, just to name a few. The CALTECH-101 dataset exhibits heavy class imbalances

## Datasets – Tiny ImageNet 200

- There are a total of **200 image classes** in this dataset with **500 images for training, 50 images for validation, and 50 images for testing per class**. Each image has been preprocessed and cropped to **64x64x3** pixels making it easier for students to focus on deep learning techniques rather than computer vision preprocessing functions.

# Datasets –ImageNet

- ImageNet is actually a project aimed at labeling and categorizing images into almost **22,000 categories** based on a defined set of words and phrases.
- At the time of this writing, there are over **14 million images** in the ImageNet project.

# Datasets –ImageNet

- ImageNet Large Scale Visual Recognition Challenge (ILSVRC)
  - A collage of ImageNet examples put together by Stanford University.
  - This dataset is massive with over **1.2 million images and 1,000 possible object categories**.
  - ImageNet is considered the de facto standard for benchmarking image classification algorithms.

## Datasets – Adience

- total of 26,580 images are included in the dataset with ages ranging from 0-60.

# Datasets - Kaggle: Facial Expression Recognition Challenge(FER)

- A total of 35,888 images are provided in the FER challenge with the goal to label a given
- facial expression into seven different categories:
  1. Angry
  2. Disgust (sometimes grouped in with "Fear" due to class imbalance)
  3. Fear
  4. Happy
  5. Sad
  6. Surprise
  7. Neutral

# Datasets - Stanford Cars

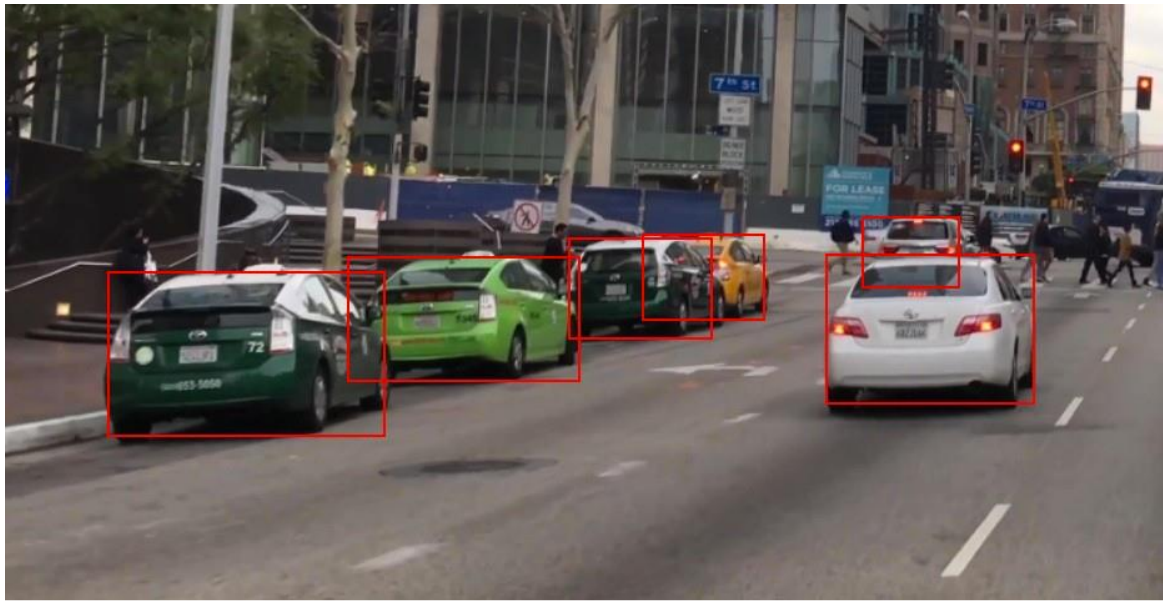
- The Stanford Cars Dataset consists of **16,185 images with 196 vehicle** make and model classes.

# Datasets - LISA Traffic Signs

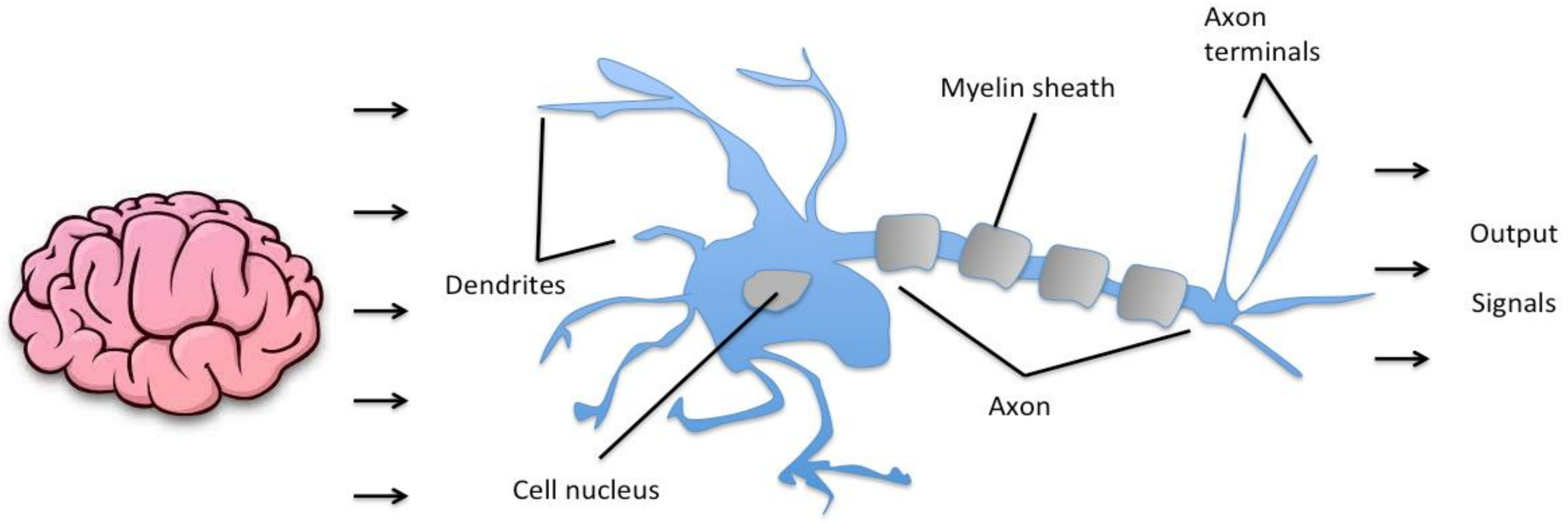
- The LISA Traffic Signs datasets consists of **47 different United States traffic signs** with **7,855 annotations over 6,610 frames**.

# Datasets -Front/Rear View Vehicles

- The Front/Rear View Vehicles dataset comes from **Davis King's dlib library** and was hand annotated by King for usage in a demonstration of his max-margin object detection algorithm.

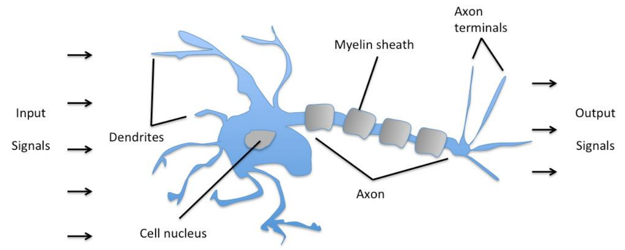


# Relation to Biology

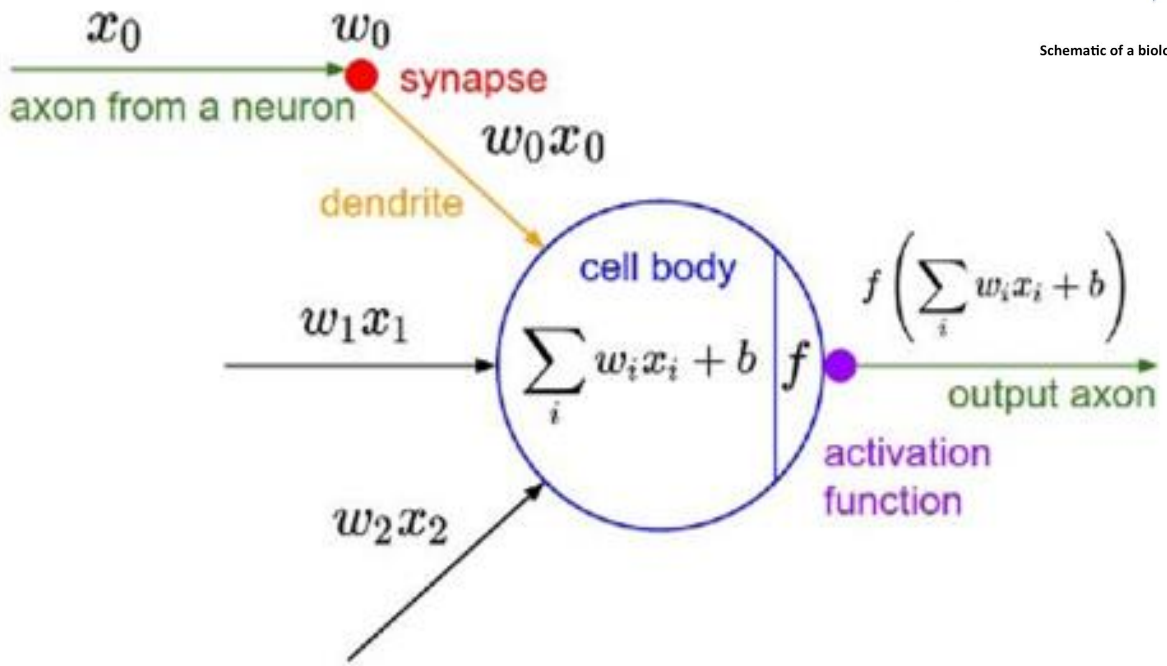


**Schematic of a biological neuron.**

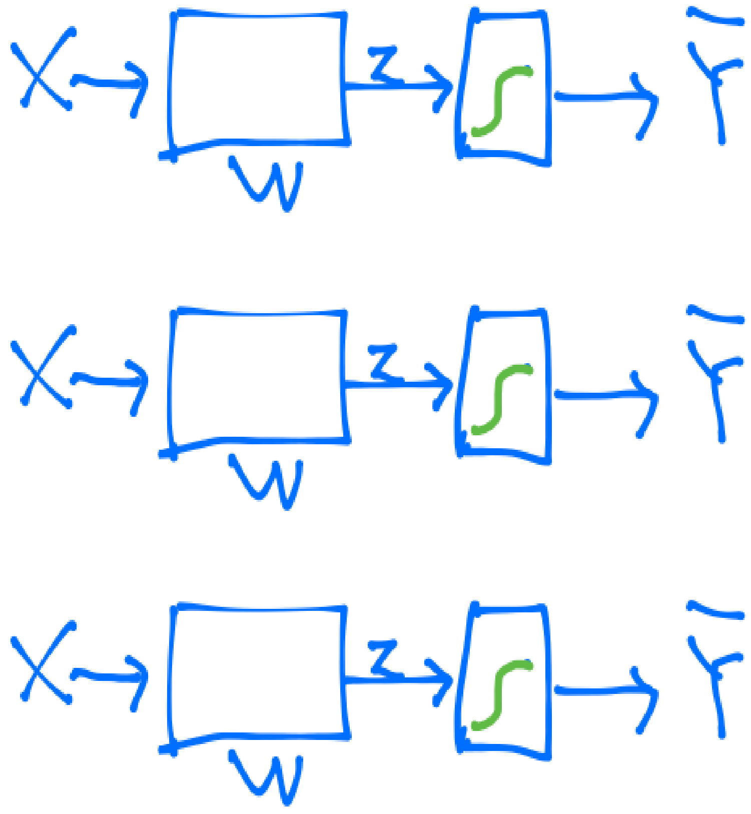
# Activation functions

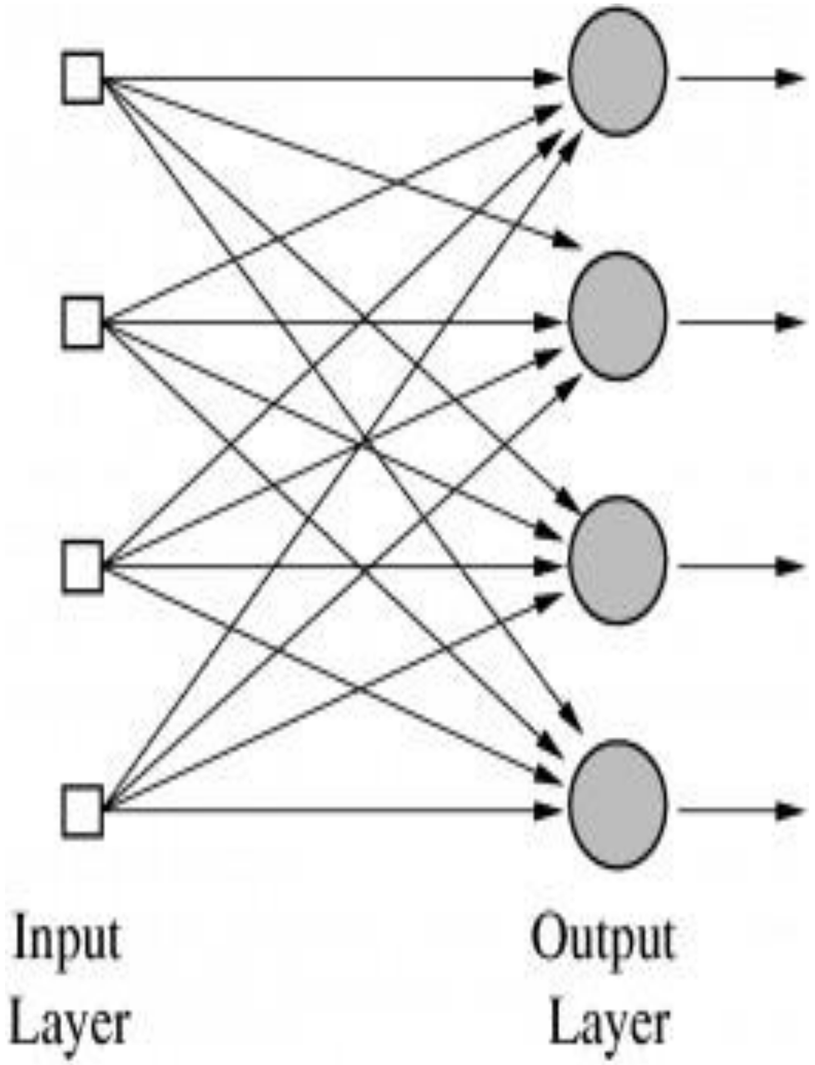


Schematic of a biological neuron.

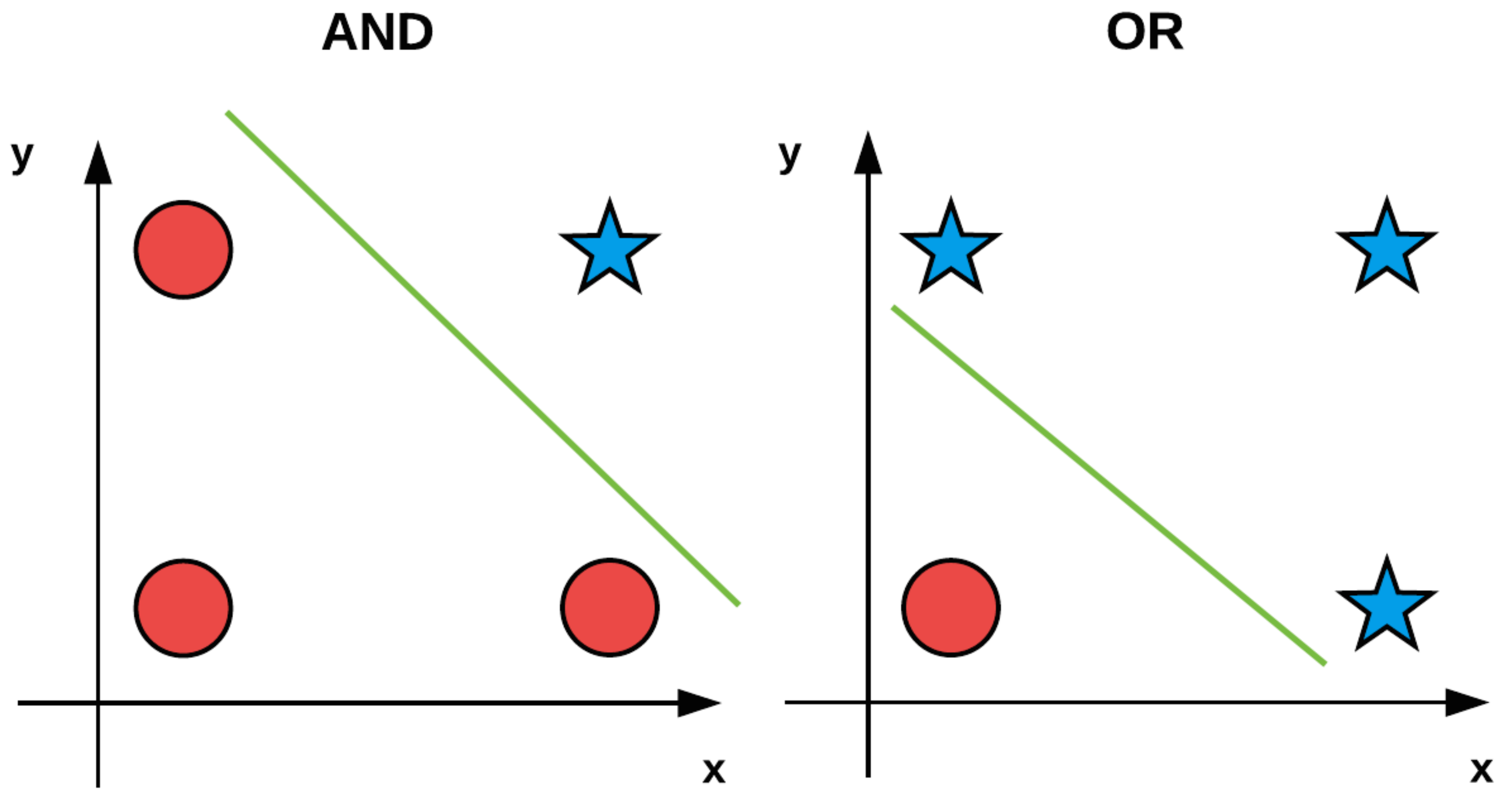


# Logistic regression units





# AND/OR problem: linearly separable?



# Exercise:

- Design the model to solve the AND problem

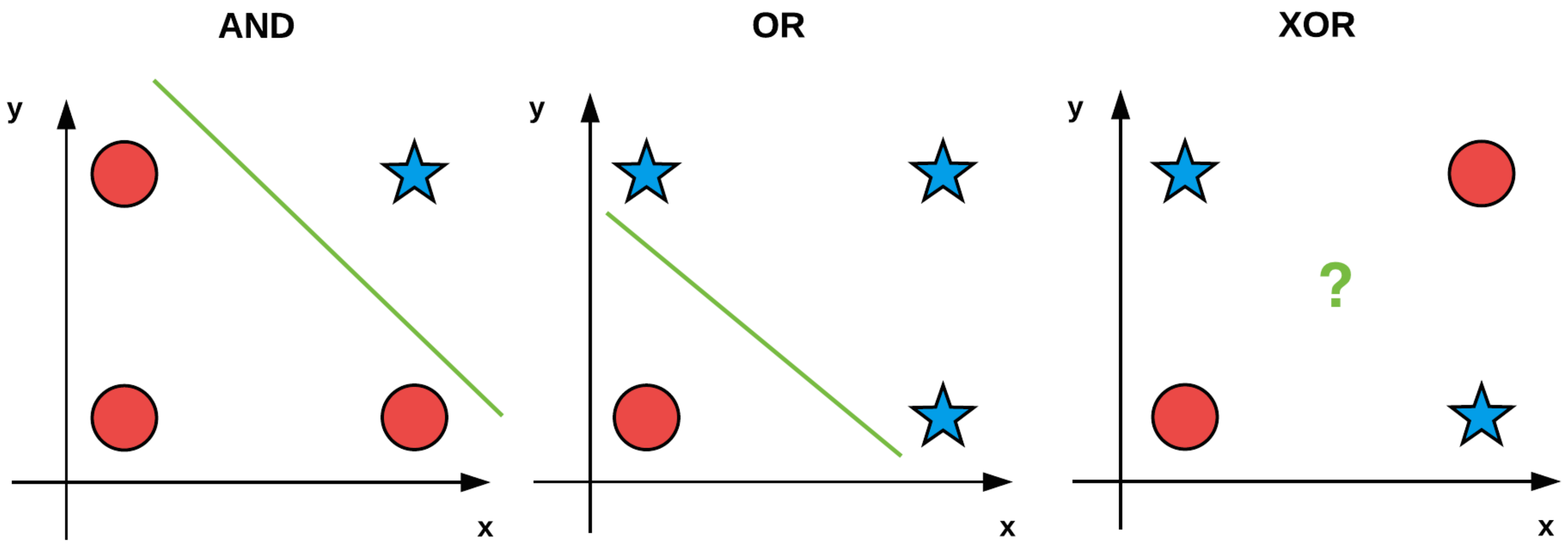
| $x_0$ | $x_1$ | $x_0 \& x_1$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

- Design the model to solve the OR problem

| $x_0$ | $x_1$ | $x_0 \| x_1$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

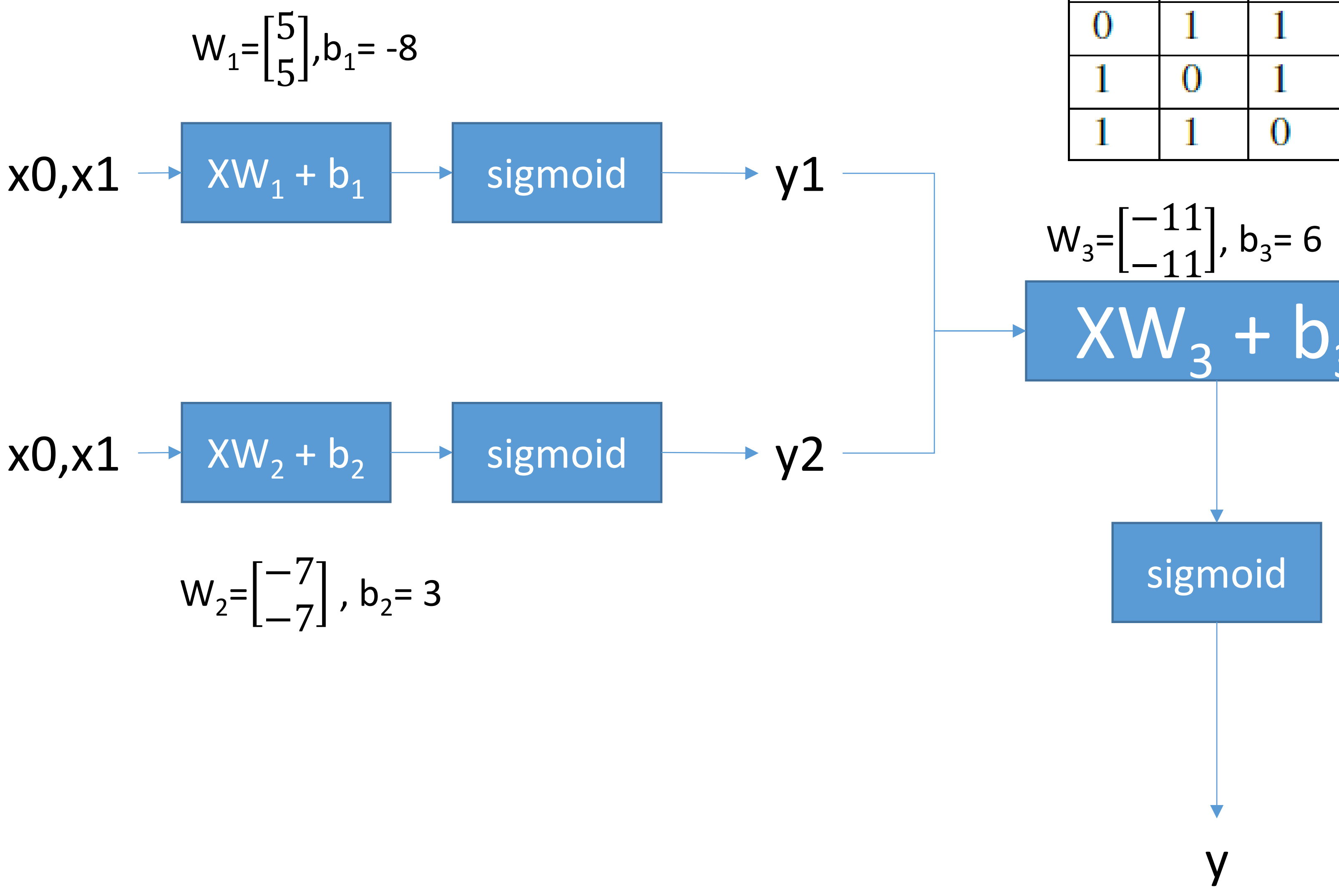
## XOR problem: linearly separable?

| $x_0$ | $x_1$ | $x_0 \wedge x_1$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |



# Neural net to solve XOR problem

| $x_0$ | $x_1$ | $x_0 \wedge x_1$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

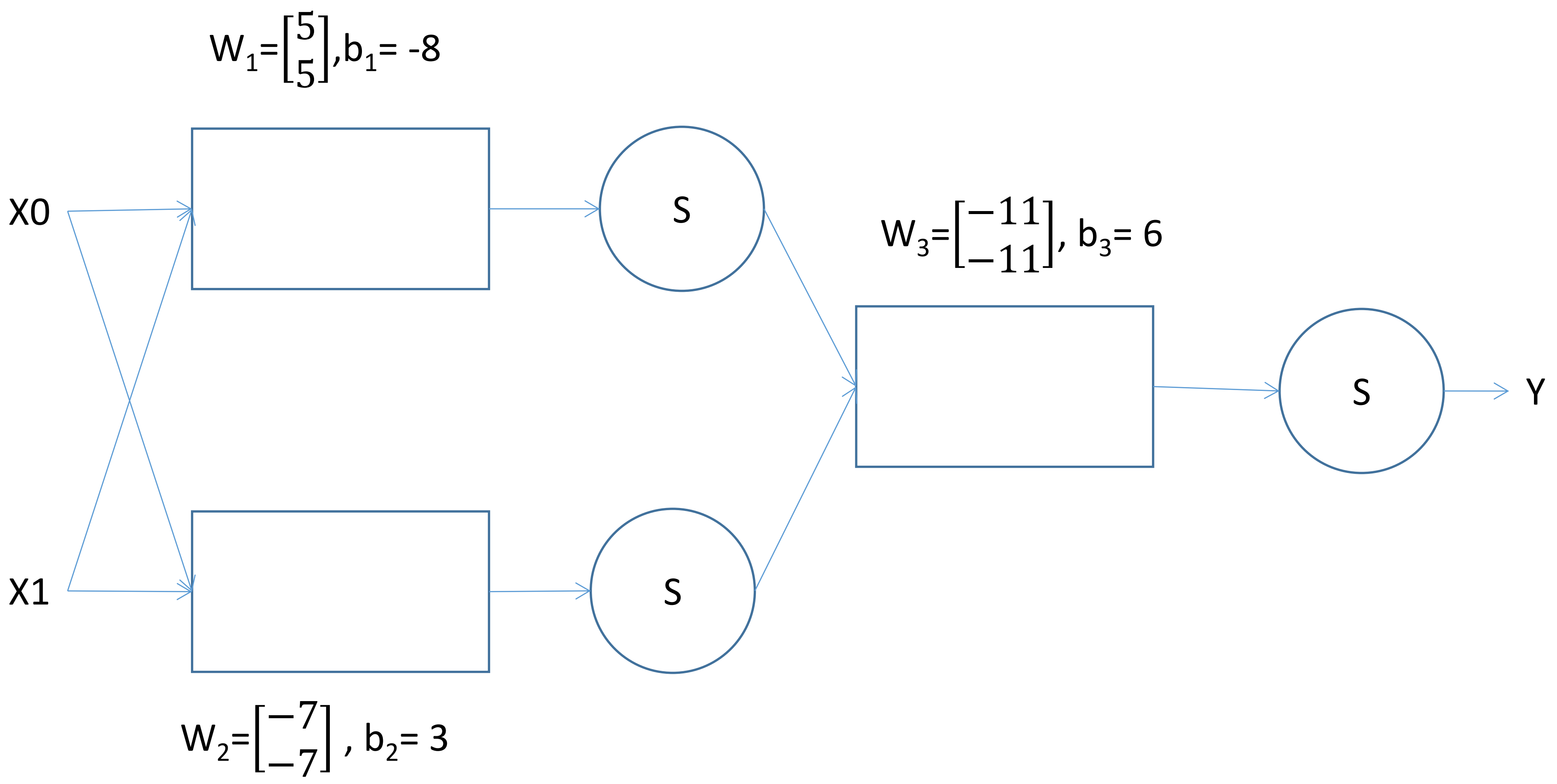


$W_1=\begin{bmatrix}5\\5\end{bmatrix}, b_1= -8$
x0,x1
$XW_1 + b_1$
sigmoid
y1
$W_3=\begin{bmatrix}-11\\-11\end{bmatrix}, b_3= 6$
$XW_3 + b_3$
x0,x1
$XW_2 + b_2$
sigmoid
y2
sigmoid
$W_2=\begin{bmatrix}-7\\-7\end{bmatrix}, b_2= 3$
y

## Neural net to solve XOR problem

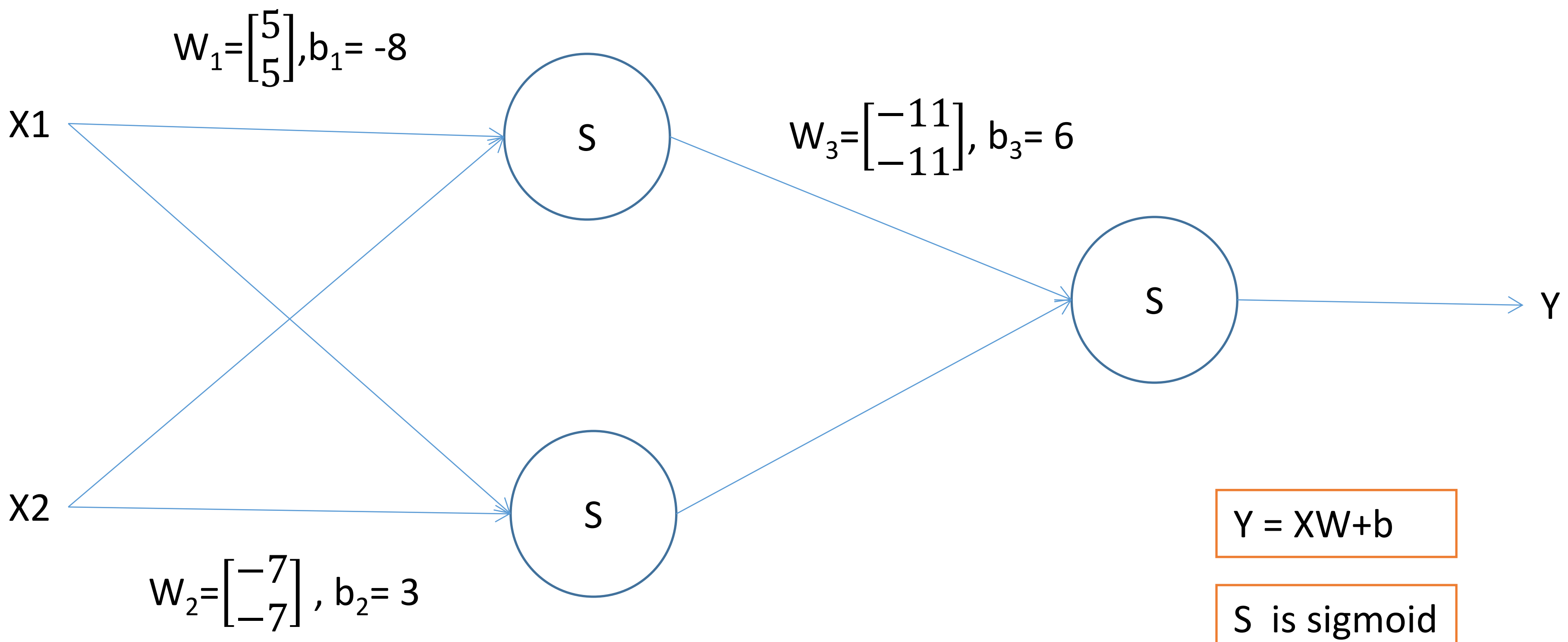
- [x0, x1] = [0, 0]
- [x0, x1] = [0, 1]
- [x0, x1] = [1, 0]
- [x0, x1] = [1, 1]

# Forward propagation



$W_1=\begin{bmatrix}5\\5\end{bmatrix}, b_1= -8$
X0
X1
S
S
$W_3=\begin{bmatrix}-11\\-11\end{bmatrix}, b_3= 6$
S
Y
$W_2=\begin{bmatrix}-7\\-7\end{bmatrix}, b_2= 3$

# Forward propagation



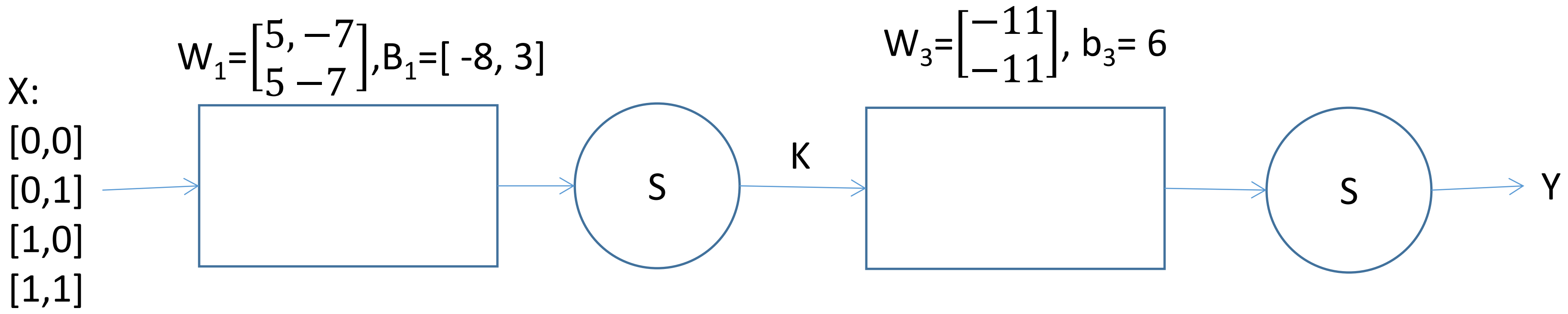
$W_1=\begin{bmatrix}5\\5\end{bmatrix}, b_1= -8$
X1
S
$W_3=\begin{bmatrix}-11\\-11\end{bmatrix}, b_3= 6$
S
Y
X2
S
$W_2=\begin{bmatrix}-7\\-7\end{bmatrix}, b_2= 3$

Y = XW+b

S is sigmoid

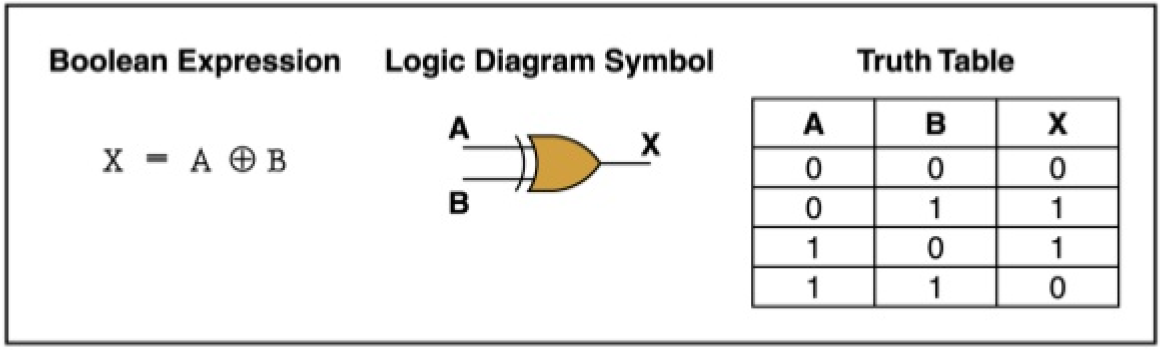
# Neural network

- Matrix form



X:
[0,0]
[0,1]
[1,0]
[1,1]
$W_1=\begin{bmatrix}5, -7\\ 5\ -7\end{bmatrix}, B_1=[-8, 3]$
S
K
$W_3=\begin{bmatrix}-11\\ -11\end{bmatrix}, b_3= 6$
S
Y

## XOR data set


Boolean Expression

$X = A \oplus B$

Logic Diagram Symbol

Truth Table

| A | B | X |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

```
x_data = np.array([[0, 0], [0, 1], [1, 0], [1, 1]], dtype=np.float32)
y_data = np.array([[0],    [1],    [1],    [0]], dtype=np.float32)
```

## Pytorch practice 1: XOR with logistic regression

```
x_data = np.array([[0, 0], [0, 1], [1, 0], [1, 1]], dtype=np.float32)
y_data = np.array([[0], [1], [1], [0]], dtype=np.float32)

X = Variable(torch.from_numpy(x_data))
Y = Variable(torch.from_numpy(y_data))

# Hypothesis using sigmoid
linear = torch.nn.Linear(2, 1, bias=True)
sigmoid = torch.nn.Sigmoid()
model = torch.nn.Sequential(linear, sigmoid)

optimizer = torch.optim.SGD(model.parameters(), lr=0.1)

for step in range(10001):
    optimizer.zero_grad()
    hypothesis = model(X)
    # cost/loss function
    cost = -(Y * torch.log(hypothesis) + (1 - Y)
             * torch.log(1 - hypothesis)).mean()
    cost.backward()
    optimizer.step()

    if step % 100 == 0:
        print(step, cost.data.numpy())

# Accuracy computation
# True if hypothesis>0.5 else False
predicted = (model(X).data > 0.5).float()
accuracy = (predicted == Y.data).float().mean()
print("\nHypothesis: ", hypothesis.data.numpy(), "\nCorrect: ", predicted.numpy(), "\nAccuracy: ", accuracy)
```

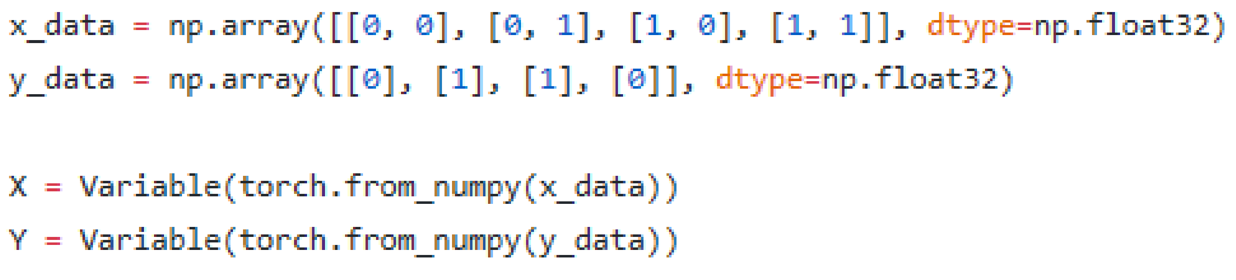
```
Hypothesis:  [[ 0.49999997]
 [ 0.5       ]
 [ 0.5       ]
 [ 0.5       ]]
Correct:  [[ 0.]
 [ 0.]
 [ 0.]
 [ 0.]]
Accuracy:  0.5
```

# Pytorch practice2: XOR with Neural Network

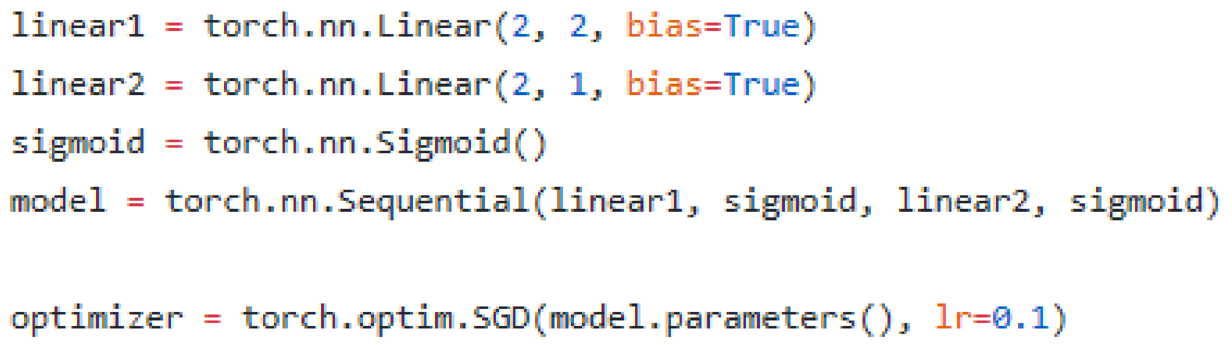
```python
x_data = np.array([[0, 0], [0, 1], [1, 0], [1, 1]], dtype=np.float32)
y_data = np.array([[0], [1], [1], [0]], dtype=np.float32)

X = Variable(torch.from_numpy(x_data))
Y = Variable(torch.from_numpy(y_data))

linear1 = torch.nn.Linear(2, 2, bias=True)
linear2 = torch.nn.Linear(2, 1, bias=True)
sigmoid = torch.nn.Sigmoid()
model = torch.nn.Sequential(linear1, sigmoid, linear2, sigmoid)

optimizer = torch.optim.SGD(model.parameters(), lr=0.1)

for step in range(10001):
    optimizer.zero_grad()
    hypothesis = model(X)
    # cost/loss function
    cost = -(Y * torch.log(hypothesis) + (1 - Y)
             * torch.log(1 - hypothesis)).mean()
    cost.backward()
    optimizer.step()

    if step % 100 == 0:
        print(step, cost.data.numpy())

# Accuracy computation
# True if hypothesis>0.5 else False
predicted = (model(X).data > 0.5).float()
accuracy = (predicted == Y.data).float().mean()
print("\nHypothesis: ", hypothesis.data.numpy(), "\nCorrect: ", predicted.numpy(), "\nAccuracy: ", accuracy)
```

```
Hypothesis:  [[ 0.0216833 ]
 [ 0.97211885]
 [ 0.97253156]
 [ 0.04630803]]
Correct:  [[ 0.]
 [ 1.]
 [ 1.]
 [ 0.]]
Accuracy:  1.0
```

## Pytorch practice2: XOR with Neural Network



```
x_data = np.array([[0, 0], [0, 1], [1, 0], [1, 1]], dtype=np.float32)
y_data = np.array([[0], [1], [1], [0]], dtype=np.float32)

X = Variable(torch.from_numpy(x_data))
Y = Variable(torch.from_numpy(y_data))
```



```
linear1 = torch.nn.Linear(2, 2, bias=True)
linear2 = torch.nn.Linear(2, 1, bias=True)
sigmoid = torch.nn.Sigmoid()
model = torch.nn.Sequential(linear1, sigmoid, linear2, sigmoid)

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