中山大學
SUN YAT-SEN UNIVERSITY

Lecture 15.
Neural Networks

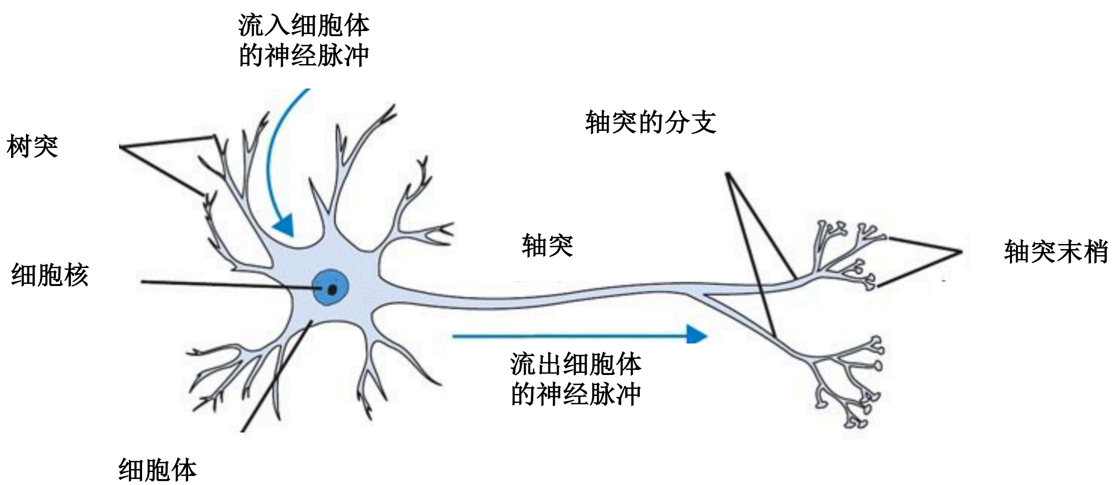
Pattern Recognition and Computer Vision

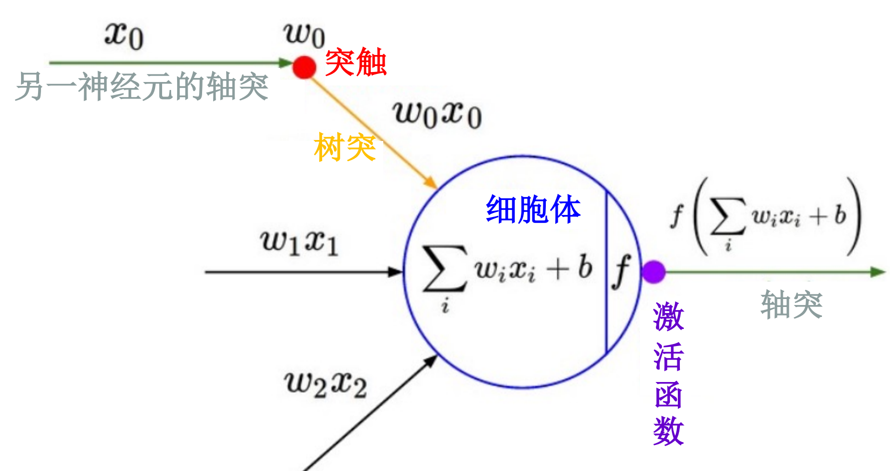
Guanbin Li,
School of Computer Science and Engineering, Sun Yat-Sen University

扫码签到

Perceptron

一个神经元通常具有多个**树突**，主要用来接受传入信息；而**轴突**只有一条，轴突尾端有许多轴突末梢可以给其他多个神经元传递信息。轴突末梢跟其他神经元的树突产生连接，从而传递信号。这个连接的位置在生物学上叫做“**突触**”。



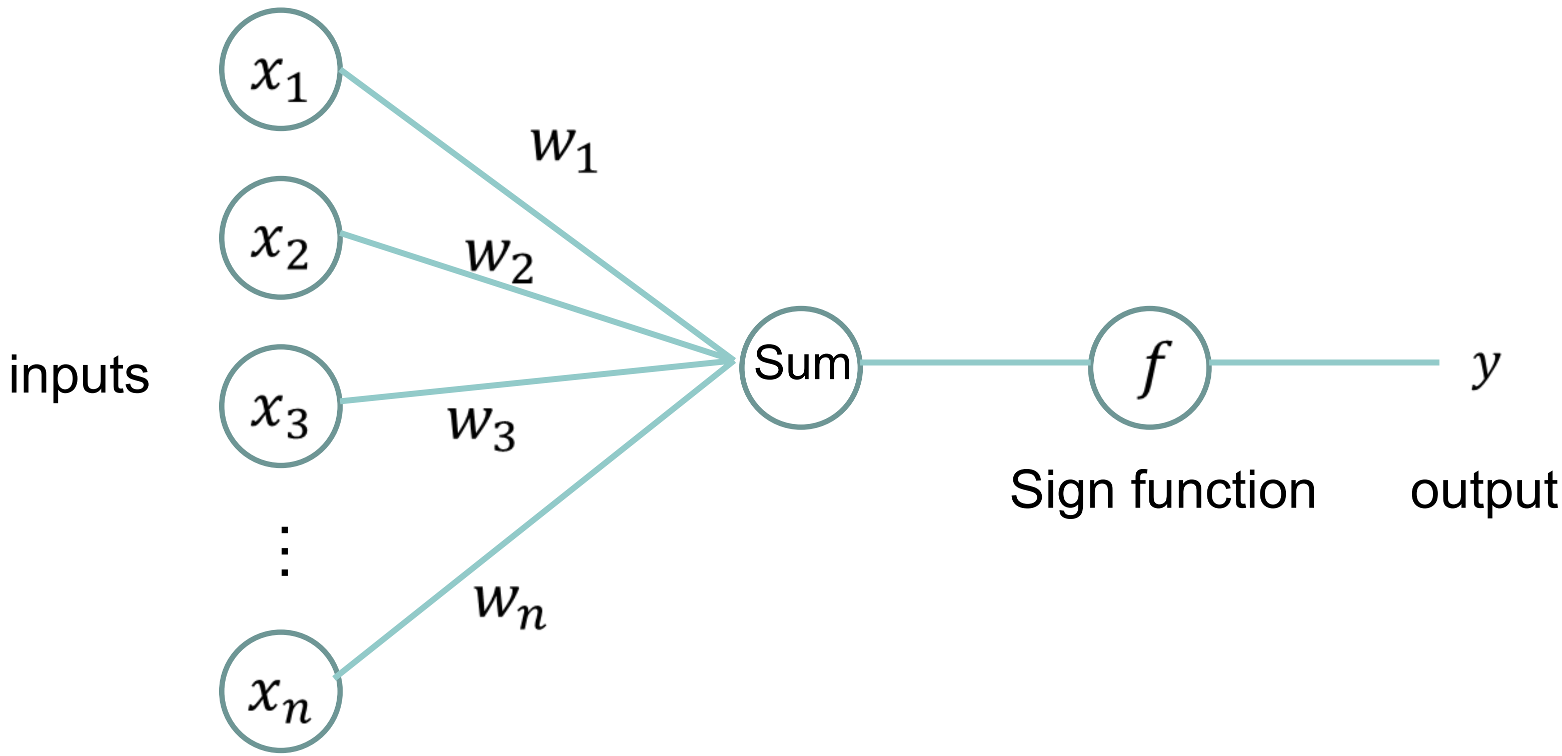


输入：权重W，初始值X

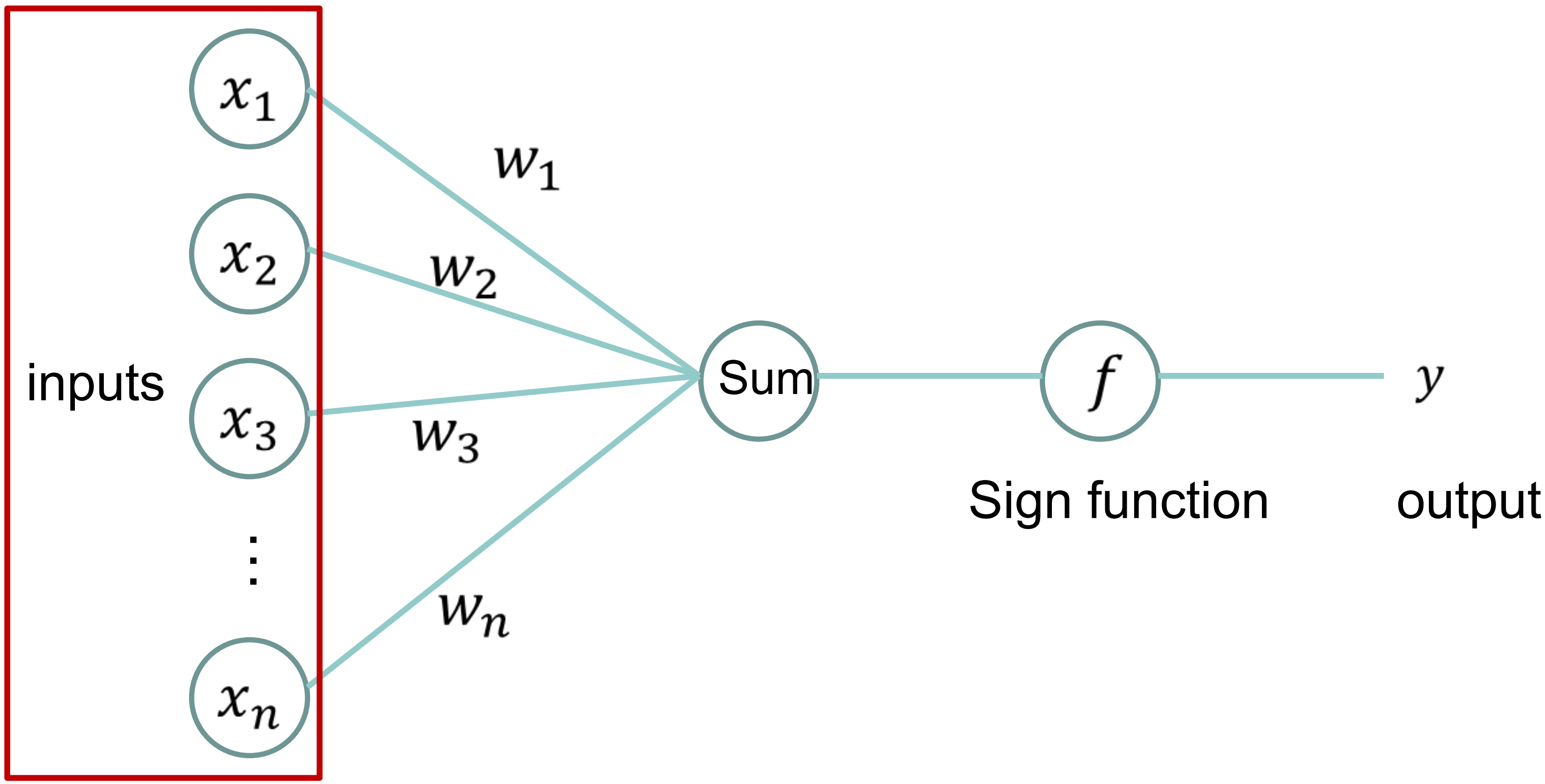
Score function：$s = WX$

输出：$Y = f(WX)$，其中f是激活函数

Perceptron



Perceptron



What are the inputs?

Perceptron

- For image classification

inputs

Raw pixels
PCA
LDA
…

x_1

x_2

x_3

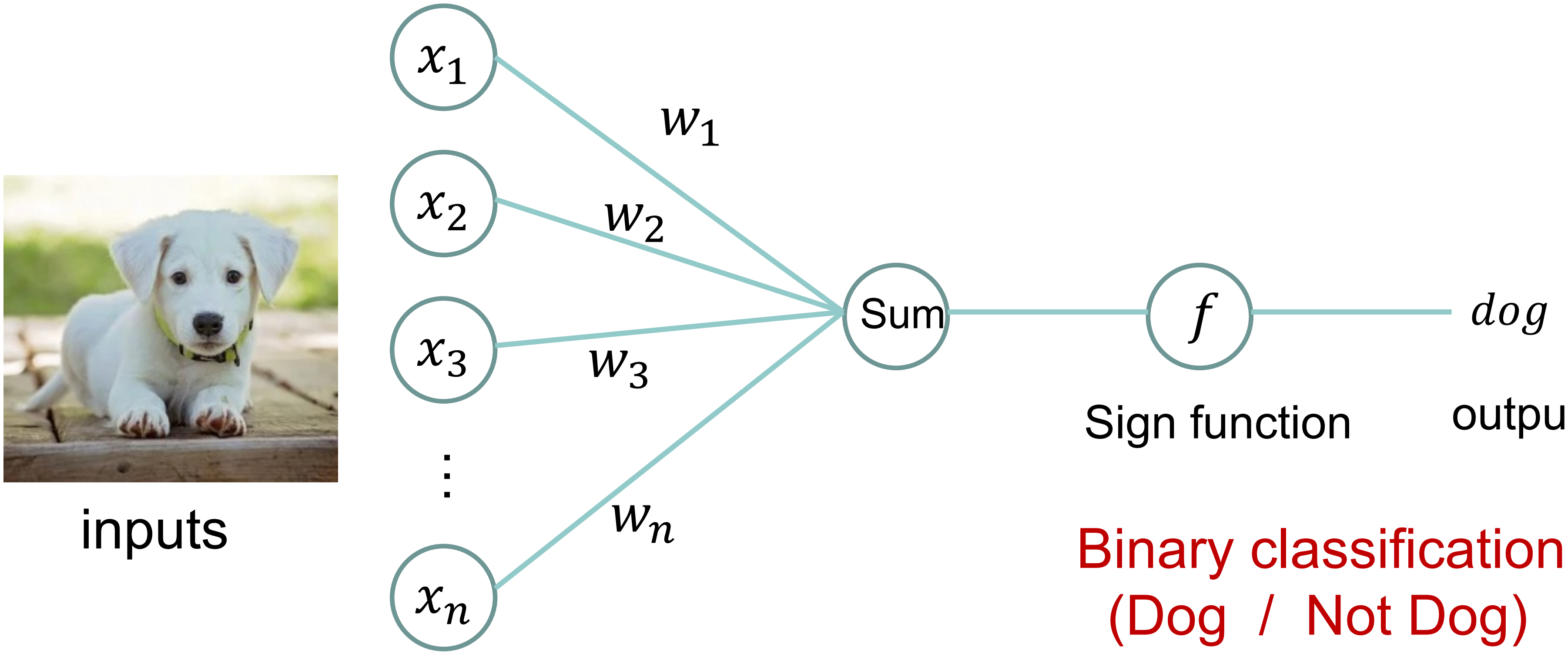
$\vdots$

x_n

Feature Vector

Perceptron

- For image classification



Binary classification
(Dog / Not Dog)

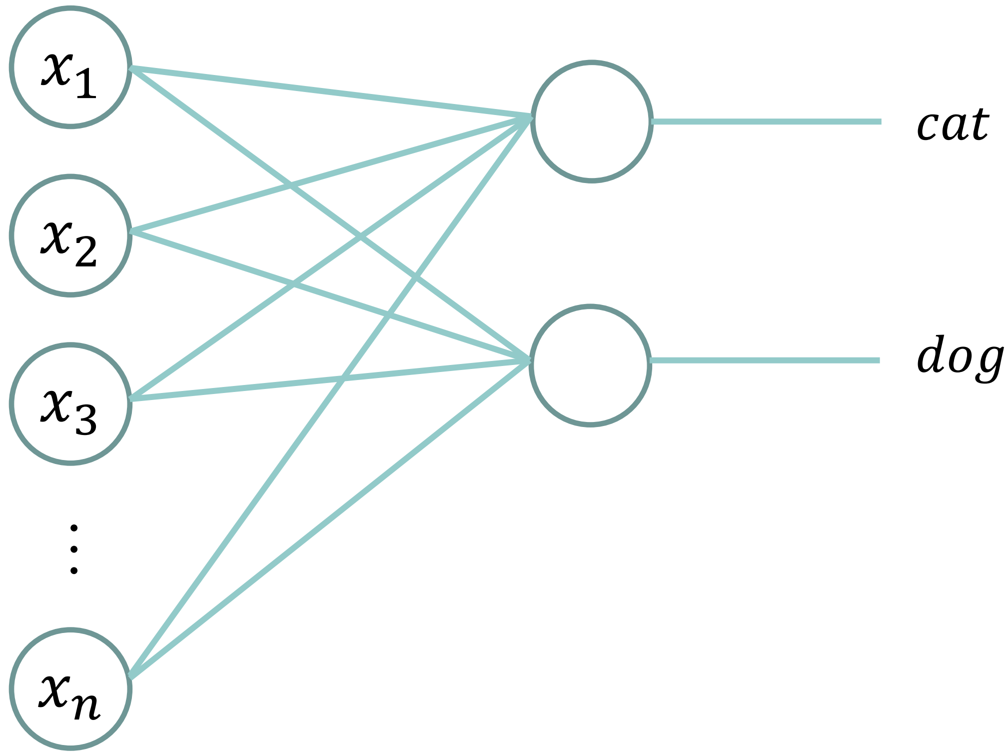
How do we handle multi-class classification task?

Add more perceptrons!

Perceptron

- Two perceptrons

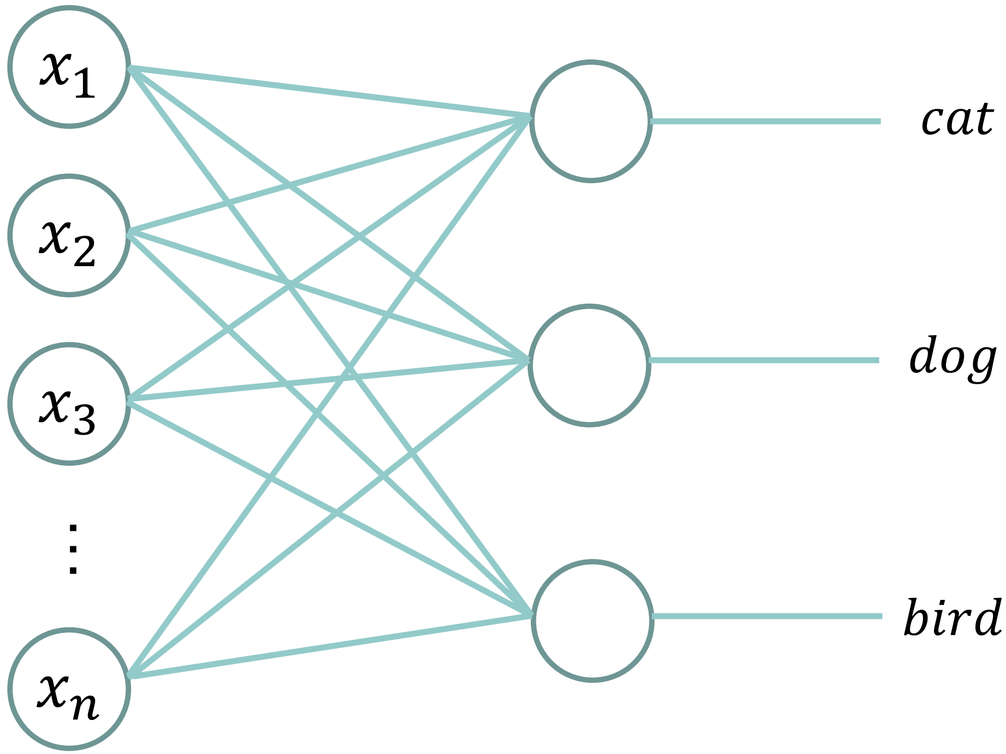
inputs



Perceptron

- Three perceptrons

inputs



Linear classifier is a set of perceptrons

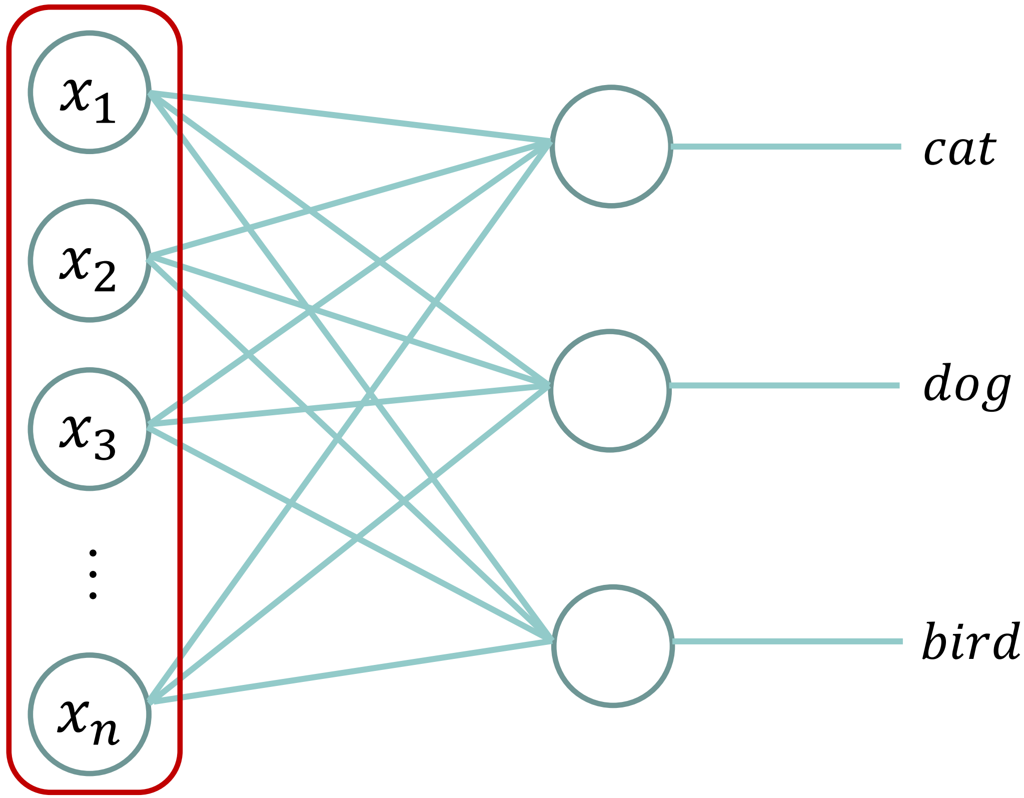
What we will learn today?

- Perceptron
- Linear classifier
- Loss function
- Gradient descent and backpropagation
- Neural networks

Linear classifier

- Input layer

inputs

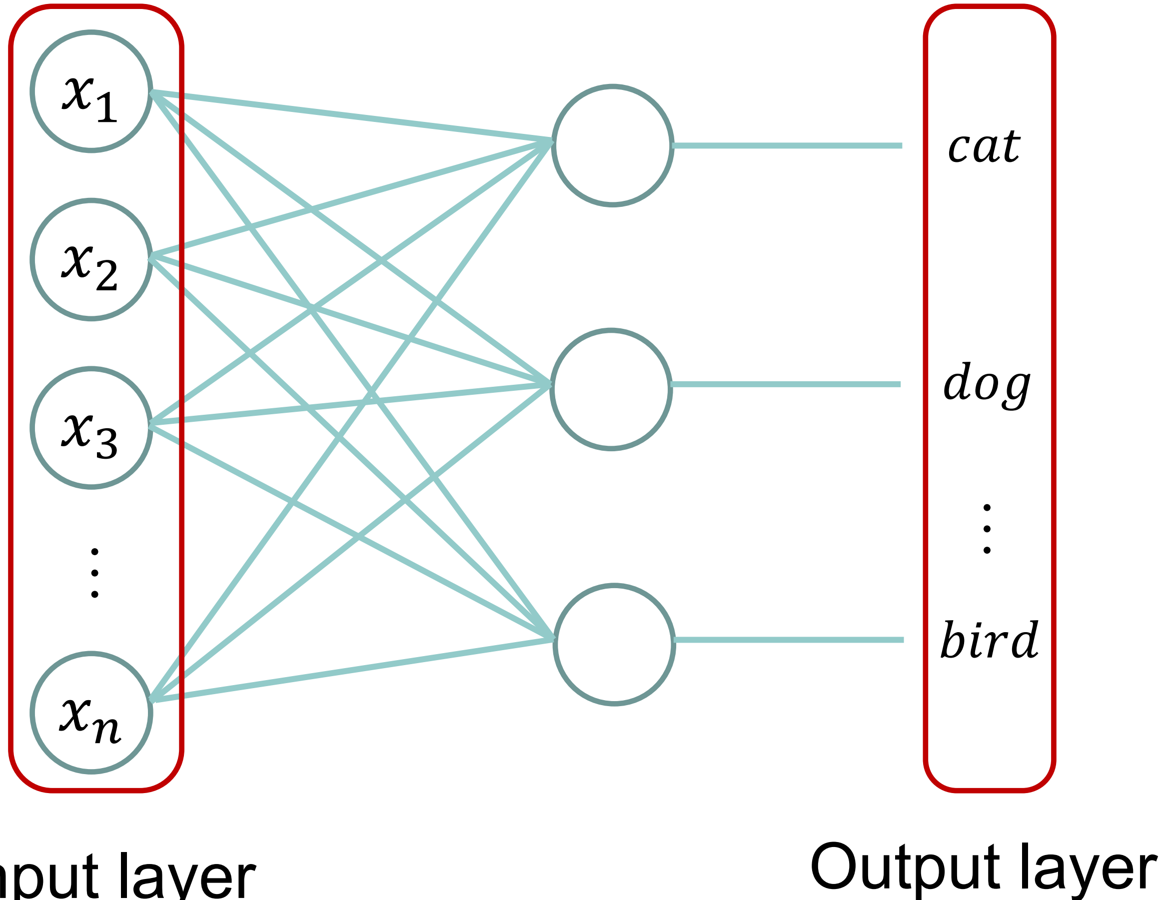


Input layer

Linear classifier

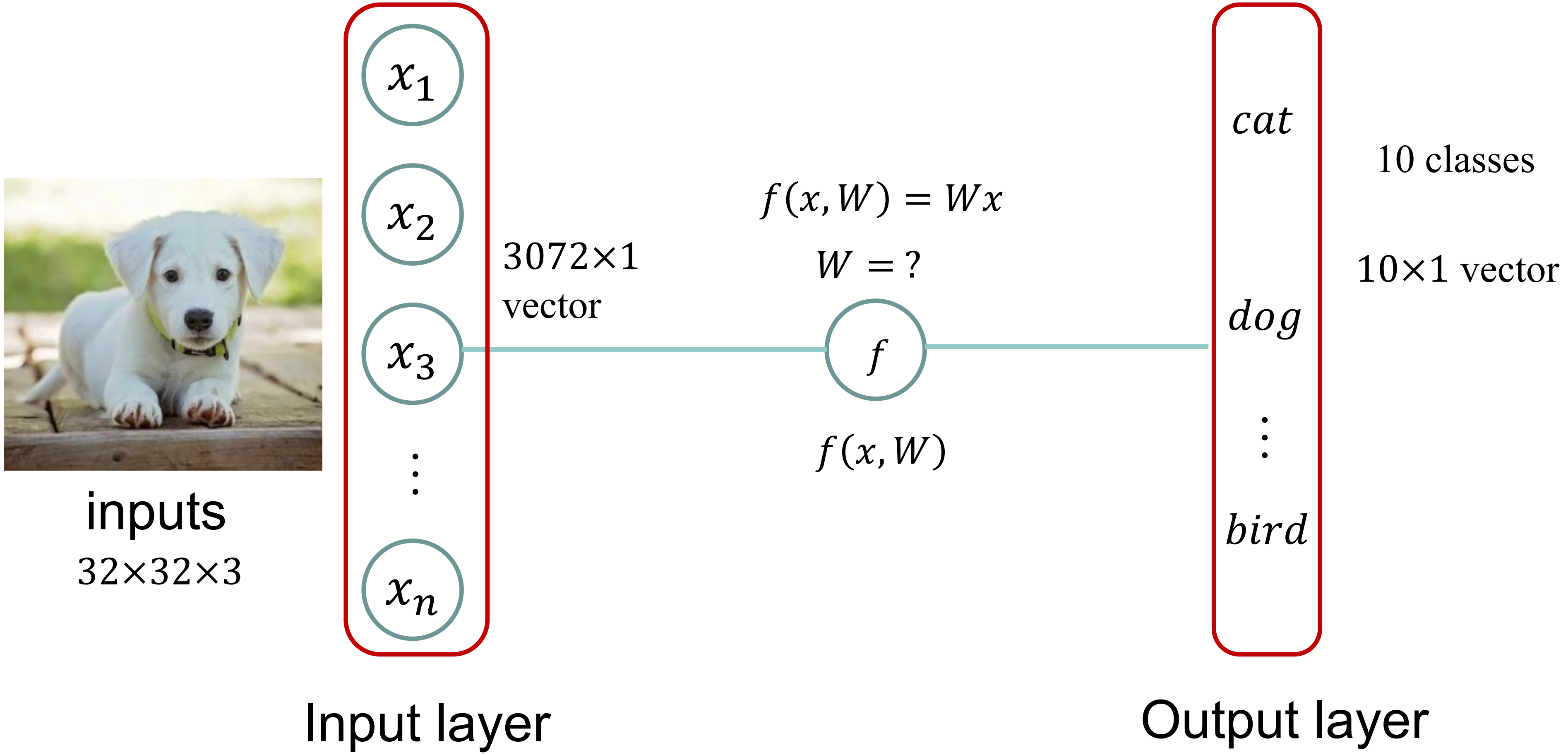
- Output layer

inputs



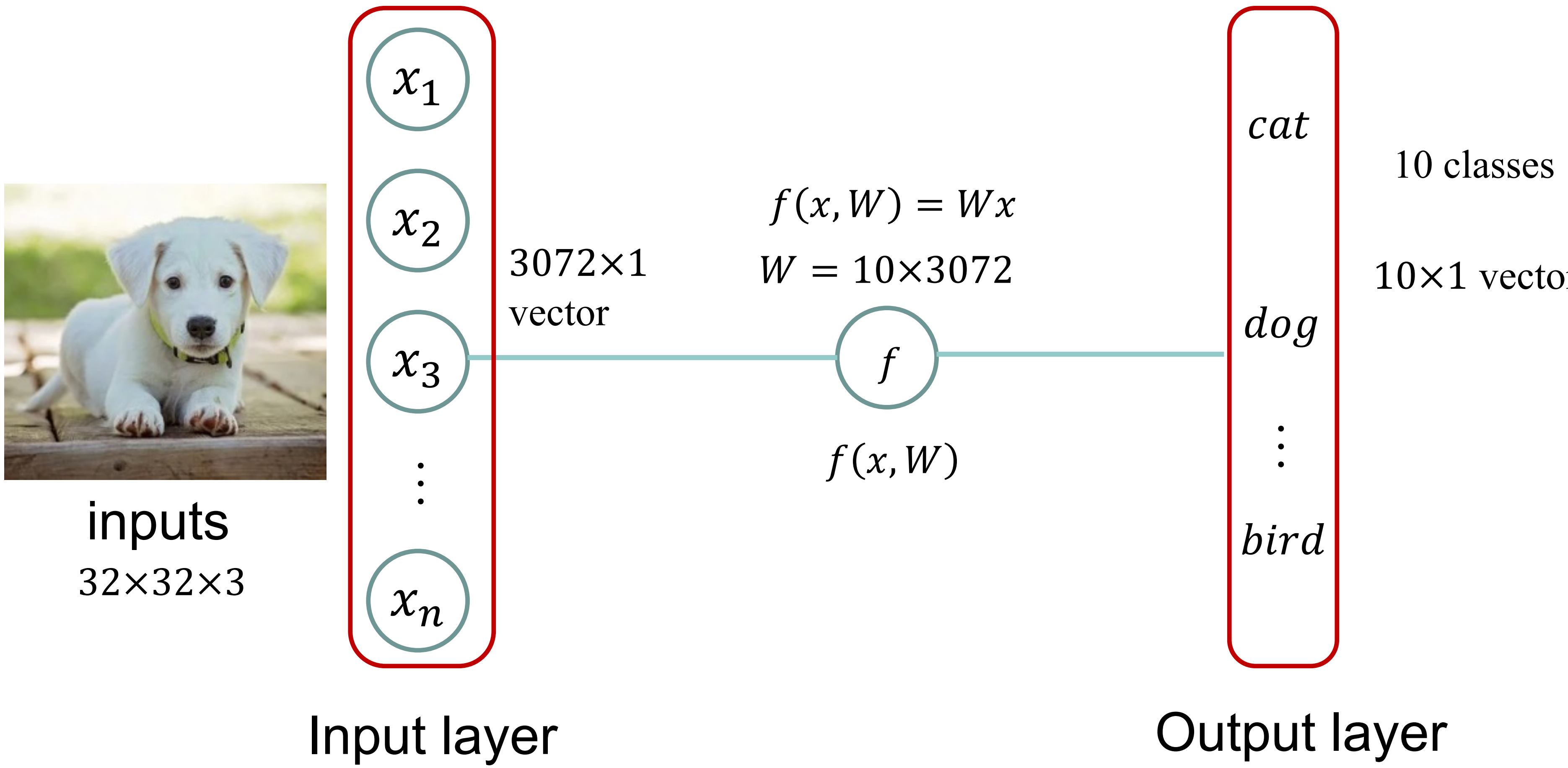
Linear classifier

- Mathematical formulation



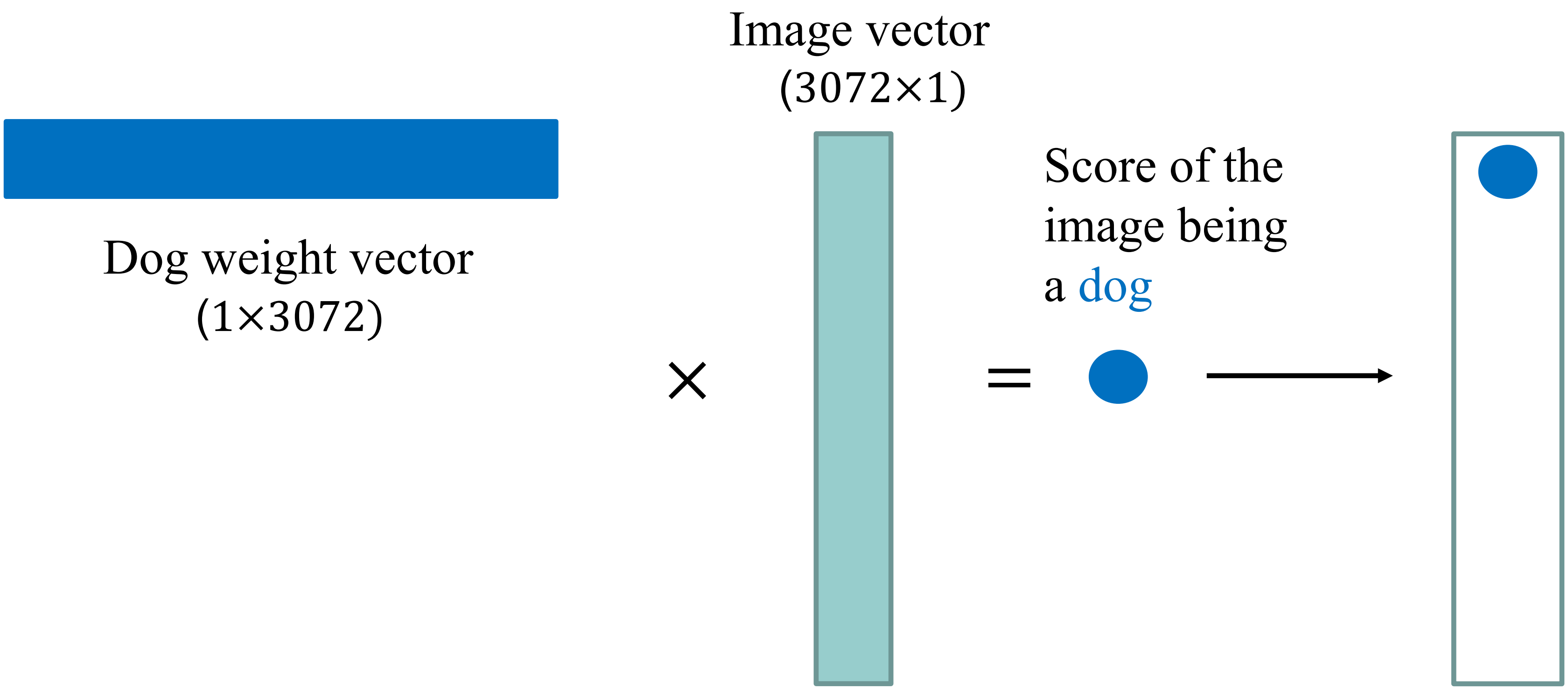
Linear classifier

- Mathematical formulation



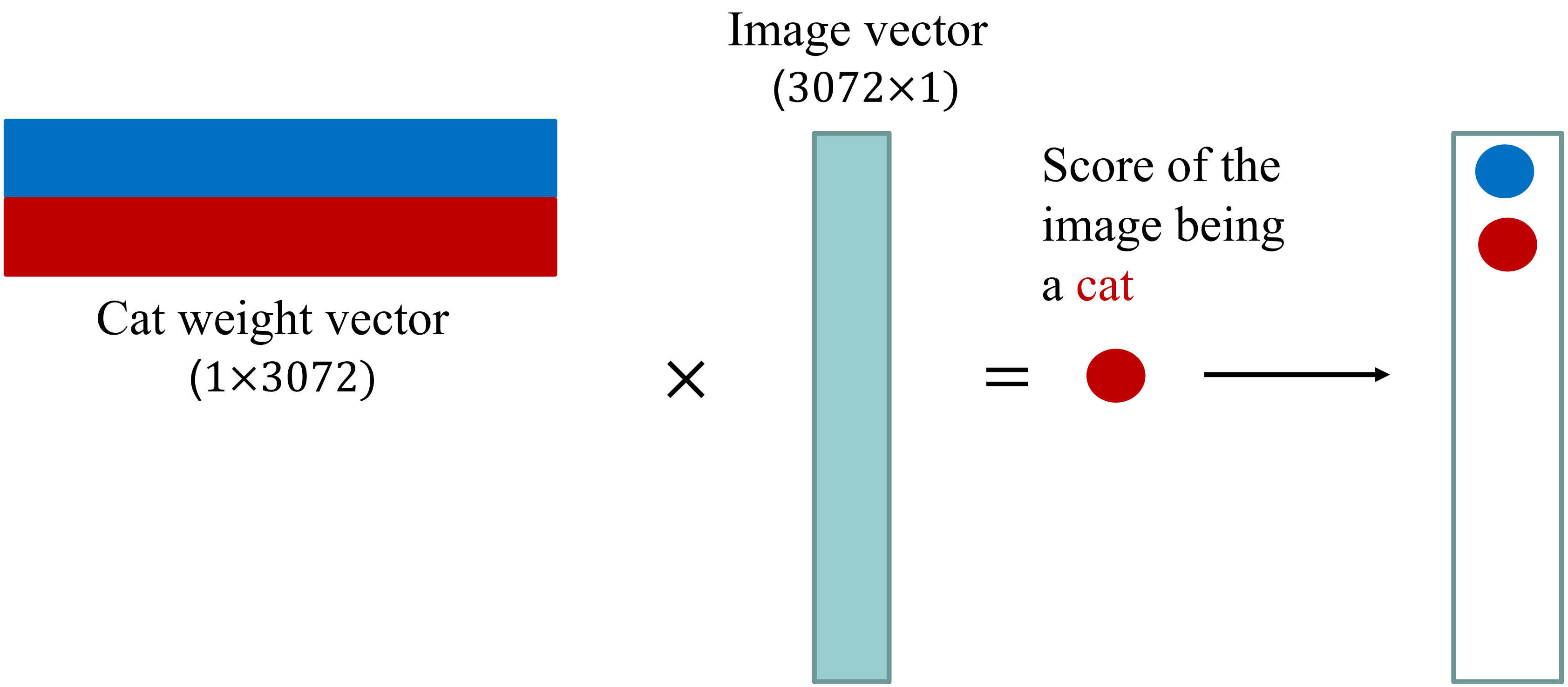
Linear classifier

- Classification task



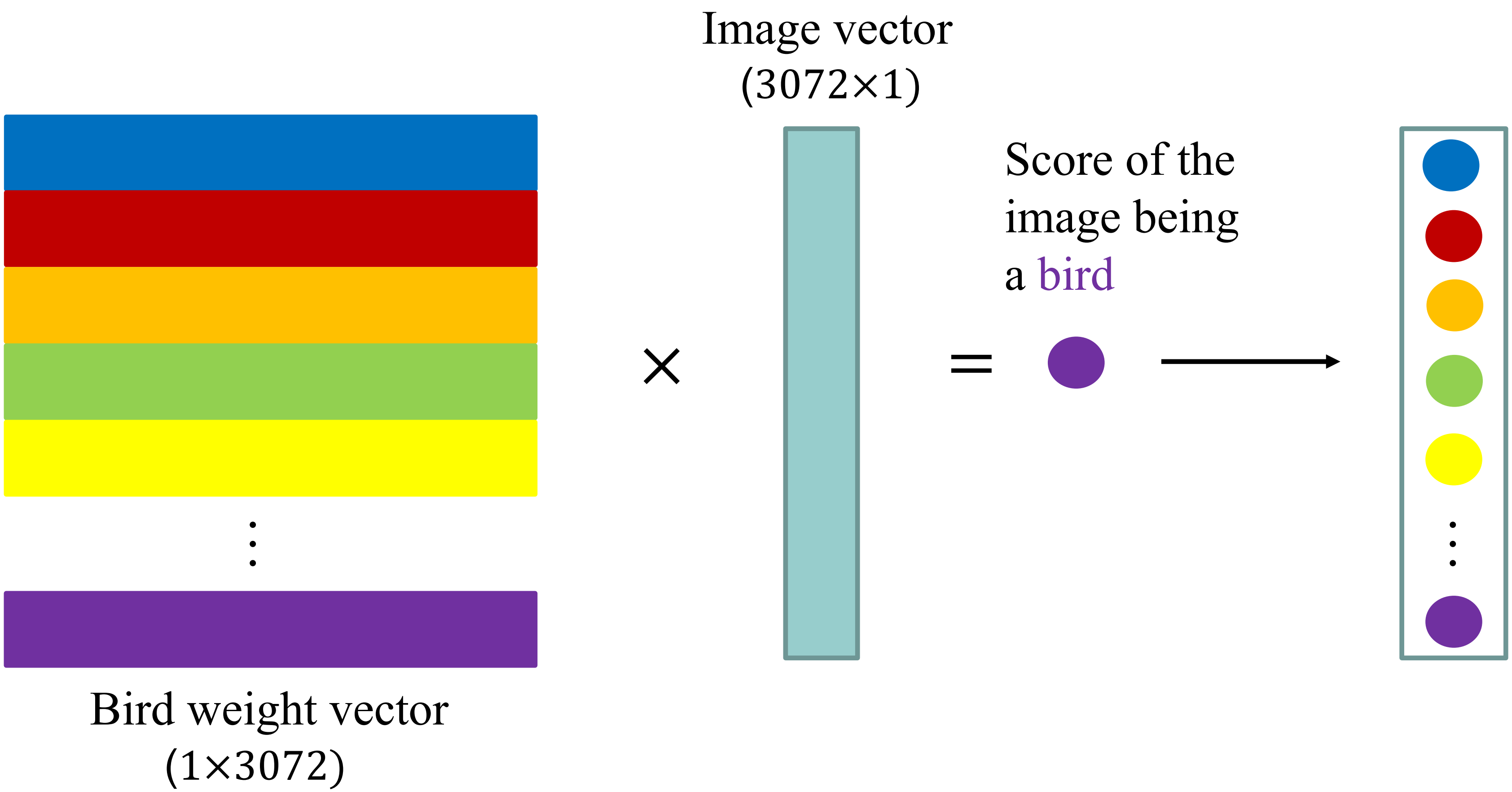
Linear classifier

- Classification task



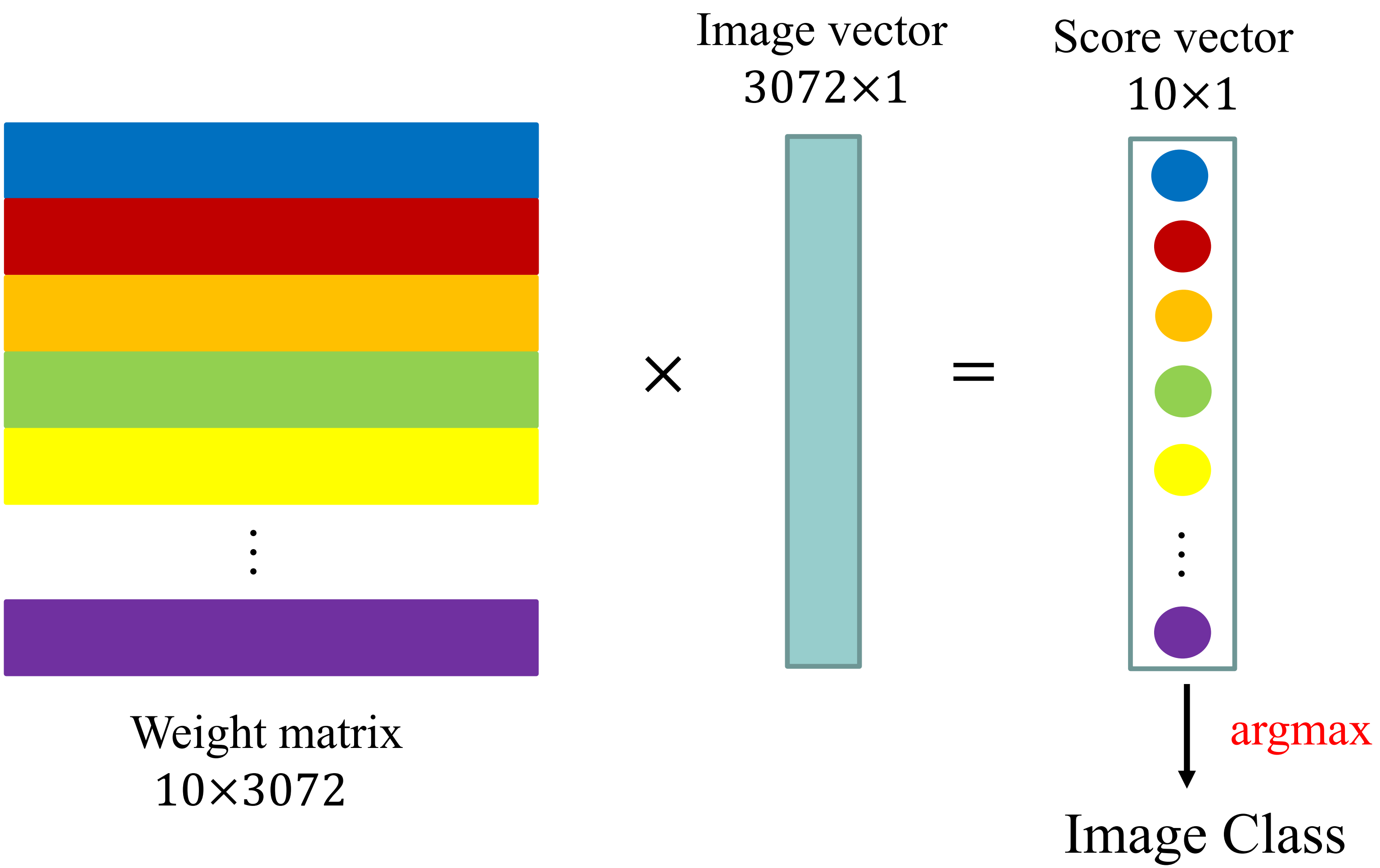
Linear classifier

- Classification task



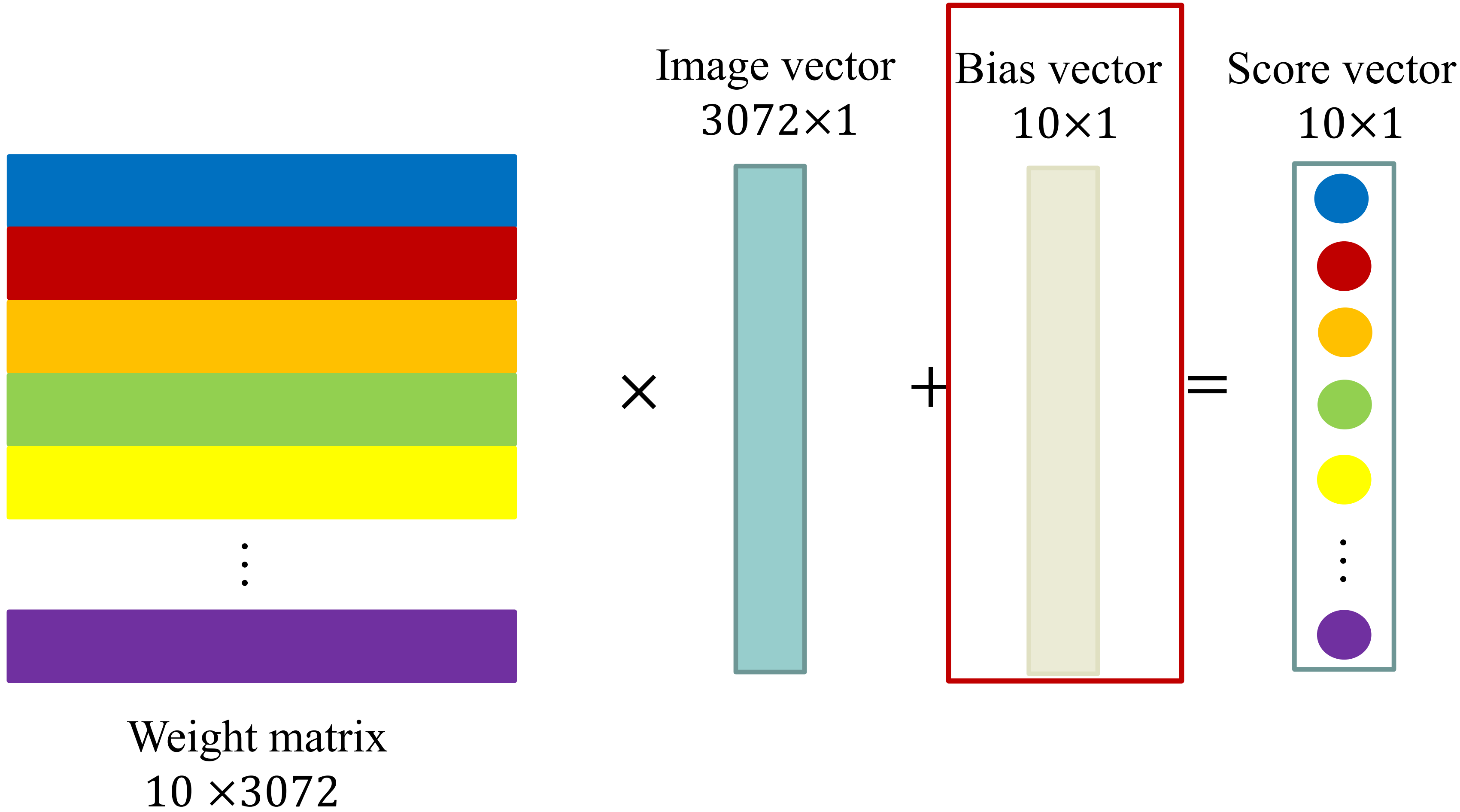
Linear classifier

- Classification task



Linear classifier

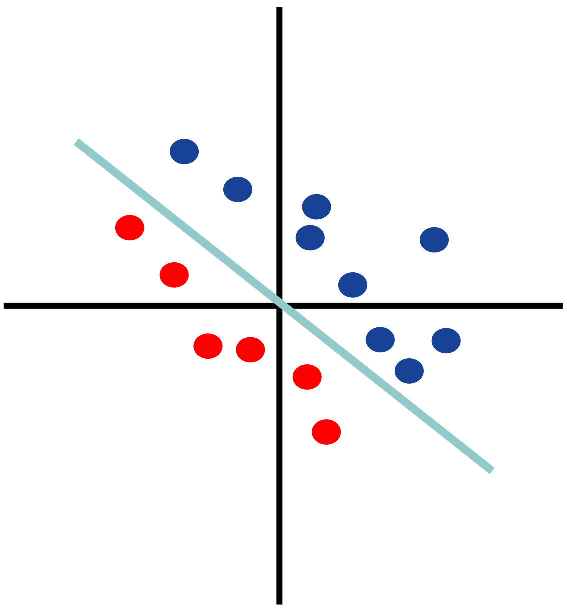
- Classification task



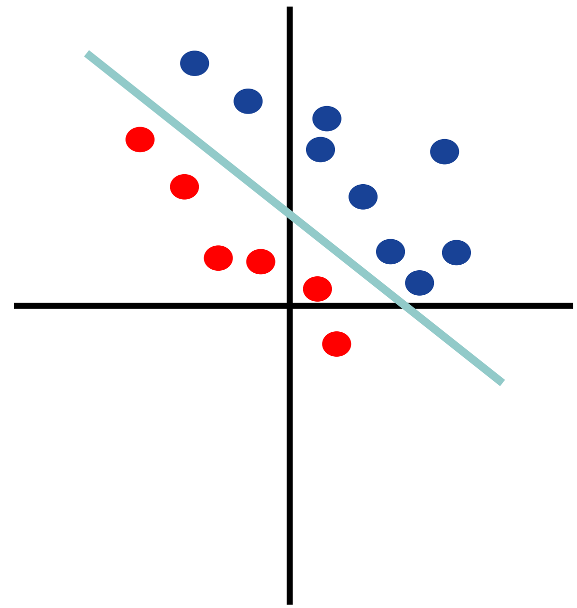
Linear classifier

- Bias can enhance the learning ability of the network

$$y = Wx$$

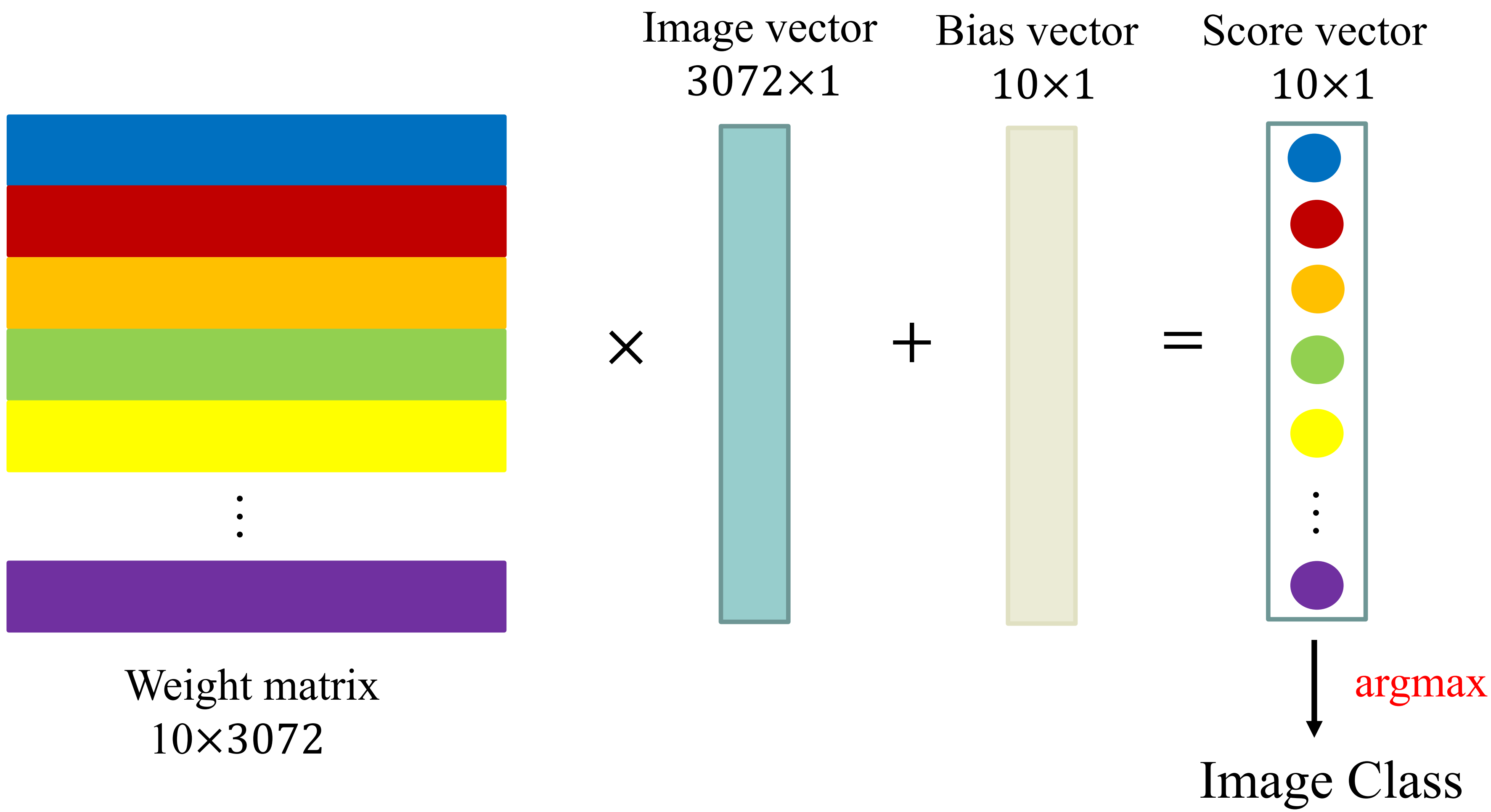


$$y = Wx + b$$



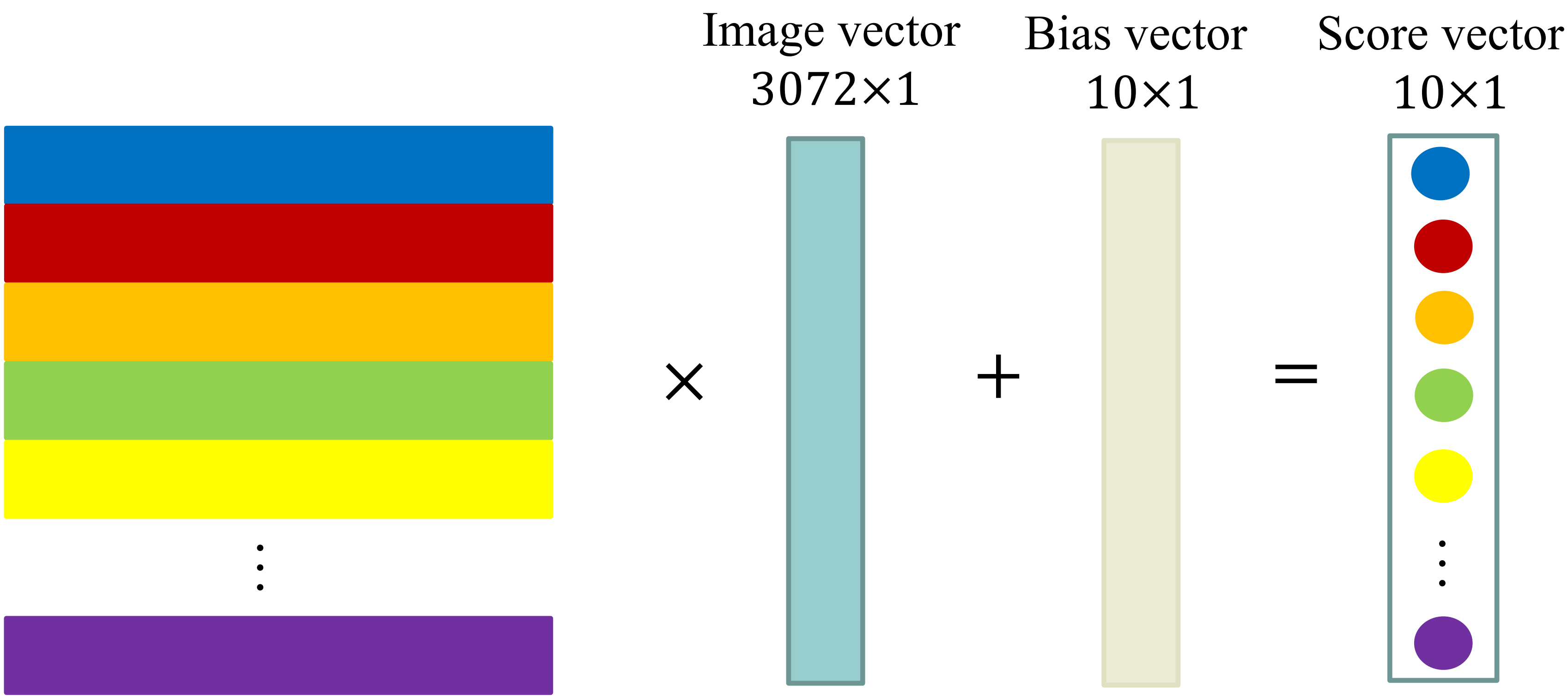
Linear classifier

- Classification task



Linear classifier

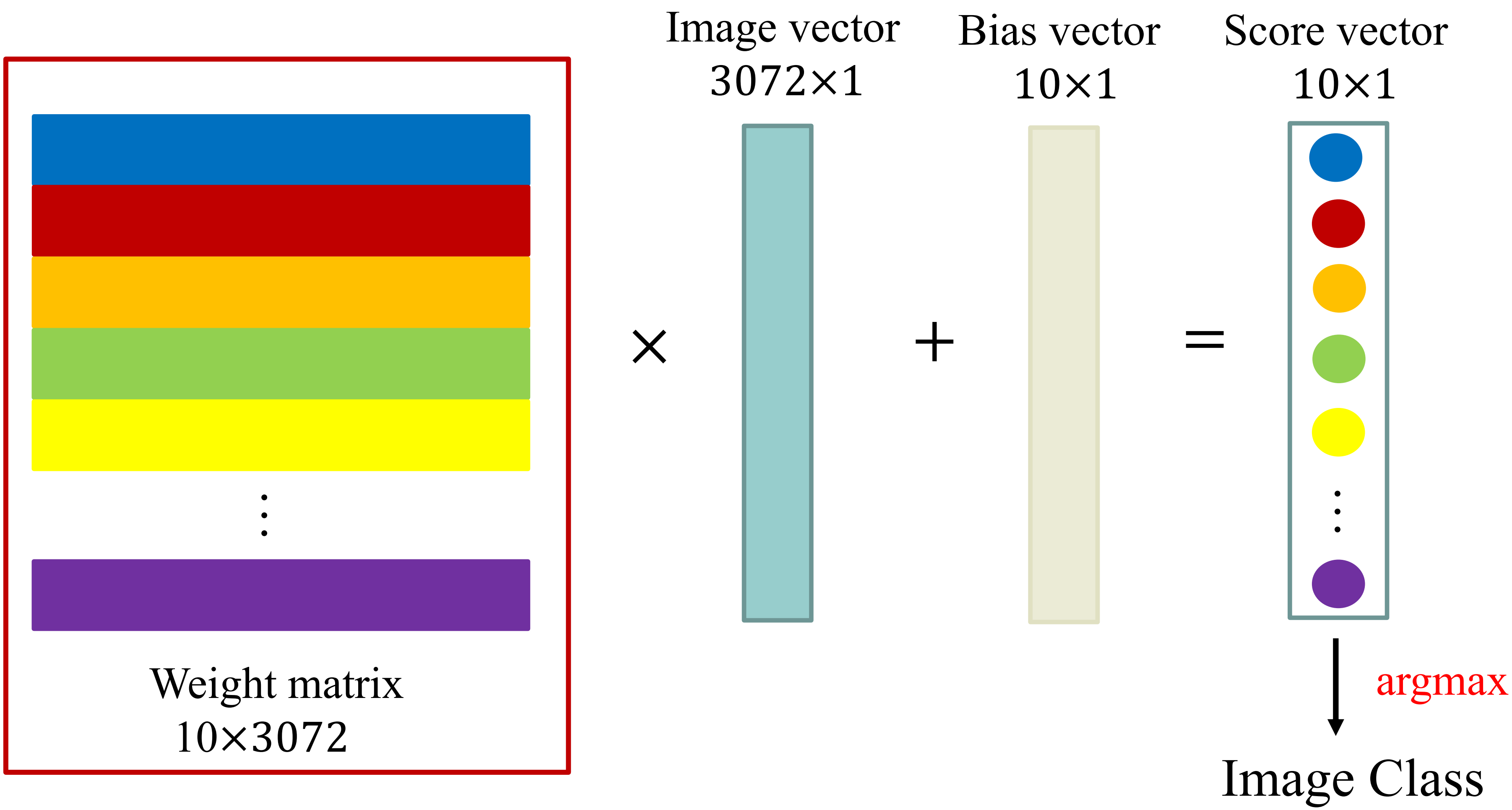
- Classification task



How do the classifier get the highest score for the right class?

Linear classifier

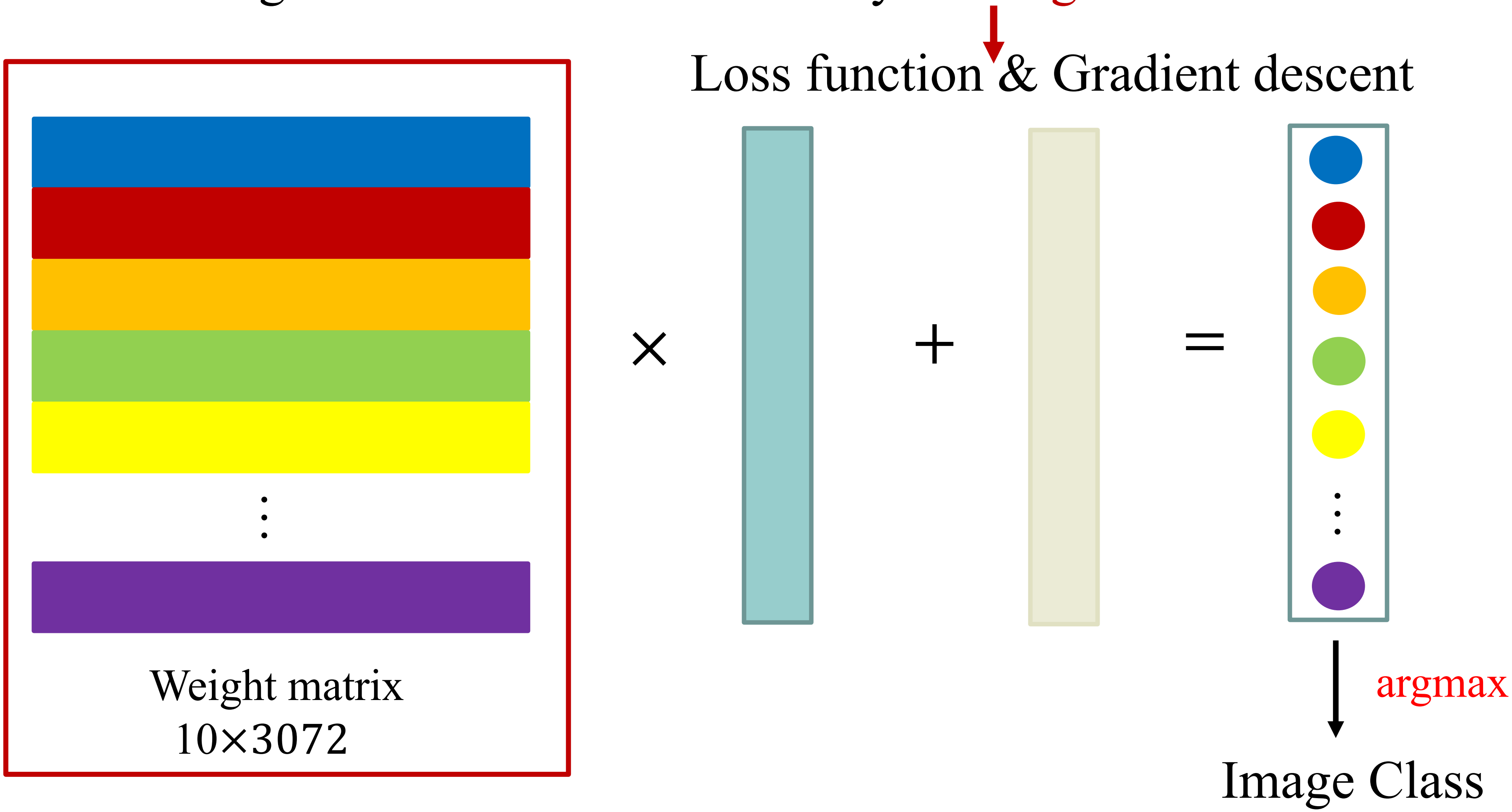
- The key of classification task is the weight matrix



Linear classifier

- The key of classification task is the weight matrix

  The weight matrix can be obtained by training!



What we will learn today?

- Perceptron
- Linear classifier
- Loss function
- Gradient descent and backpropagation
- Neural networks

Loss function

- Loss function determines how good our classifier

Given some training examples:
$\{(x_1, y_1), (x_2, y_2), (x_3, y_3), \cdots (x_n, y_n)\}$
x_i the ith image, y_i is the corresponding integer label
(e.g. 0 for dog, 1 for cat ⋯)

and our classifier: $\hat{y} = Wx$

Loss of one example is determined as $L_i(y_i, \hat{y}_i)$
when the classifier predicts correctly ($\hat{y}_i = y_i$), the loss is low
when the classifier makes mistakes ($\hat{y}_i \neq y_i$), the loss is high

Loss function

- Loss function determines how good our classifier

Given some training examples:

$\{(x_1, y_1), (x_2, y_2), (x_3, y_3), \cdots (x_n, y_n)\}$

x_i the ith image, y_i is the corresponding integer label

(e.g. 0 for dog, 1 for cat ⋯)

and our classifier: $\hat{y} = Wx$

Loss over the dataset is the average of loss over examples:

$$Loss = \frac{1}{N} \sum_i^N L_i(y_i, \hat{y}_i)$$

Loss function

- Loss function is the key to find suitable W

Specifically, we need to find W such that:

$$min_W Loss(y, \hat{y})$$

y is the true labels, $\hat{y}$ is the model predicted labels.

Loss function

- Some popular loss functions

L1 Loss

$$L_i(y_i, \hat{y}_i) = |y_i - \hat{y}_i|$$

Squared Error Loss(L2 Loss)

$$L_i(y_i, \hat{y}_i) = (y_i - \hat{y}_i)^2$$

Zero-One Loss

$$L_i(y_i, \hat{y}_i) = 1\|y_i \neq \hat{y}_i\|$$

Hinge Loss

$$L_i(y_i, \hat{y}_i) = \max(0, 1 - y_i\hat{y}_i)$$

Loss function

- Softmax Loss (multi-class classification task)

- Softmax function allows us to treat the outputs of a model as probabilities for each class.
- Common way of measuring distance between probability distributions is Kullback-Leibler (KL) divergence.

$$D_{KL} = \sum_{y} P(y) log \frac{P(y)}{Q(y)}$$

P is the ground truth distribution and

Q is the output score distribution

Loss function

- KL divergence

$$D_{KL} = \sum_{y} P(y) log \frac{P(y)}{Q(y)}$$

In our case, P is only non-zero for the correct class.

For example, consider the case where we only have 3 classes:

1	dog
0	cat
0	bird

Ground Truth

Loss function

- KL divergence

$$D_{KL} = \sum_{y} P(y) log \frac{P(y)}{Q(y)}$$

$$= -logQ(y) \text{ when } y = dog$$

$$= -logProb(f(x_i, W) = y_i)$$

1	dog
0	cat
0	bird

Ground Truth

Loss function

- KL divergence

$$L_i = -logProb(f(x_i, W) == y_i)$$

Our linear classifier:

$$\hat{y} = wx$$

There is no limits on the output space.

Meaning that the model can generate outputs >1 or <0 .

3.2
5.1
-1.7

Model Outputs

1	dog
0	cat
0	bird

Ground Truth

Loss function

- Softmax function

$$L_i = -logProb(f(x_i, W) == y_i)$$

We need to convert the outputs into probability ranges [0,1].

3.2
5.1
-1.7

Model Outputs

1	dog
0	cat
0	bird

Ground Truth

Loss function

- Softmax function

$$L_i = -logProb(f(x_i, W) == y_i)$$

We need to convert the outputs into probability ranges [0,1].

Solution: Softmax: $Prob(f(x_i, W) == k) = \frac{e^{\hat{y}k}}{\sum_j e^{\hat{y}j}}$

Model Outputs	Ground Truth	
3.2	1	dog
5.1	0	cat
-1.7	0	bird

Loss function

- Softmax function

$$L_i = -logProb(f(x_i, W) == y_i)$$

We need to convert the outputs into probability ranges [0,1].

Solution: Softmax: $Prob(f(x_i, W) == k) = \frac{e^{\hat{y}k}}{\sum_j e^{\hat{y}j}}$

Model Outputs	exp →	norm →	↔ loss	Ground Truth	
3.2	24.5	0.13		1	dog
5.1	168	0.87		0	cat
-1.7	0.18	0.00		0	bird

Loss function

- Softmax function

$$L_i = -logProb(f(x_i, W) == y_i)$$

In this case, loss is: $L_i = -\log(0.13) = 2.04$

The dog probability closes to 1, the loss closes to 0.

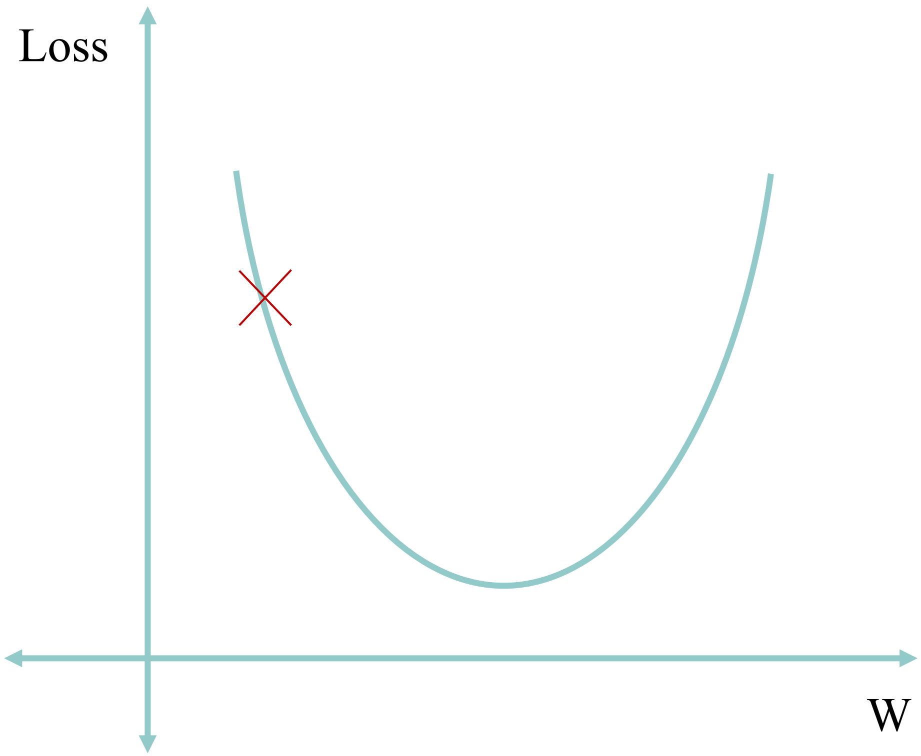
Model Outputs				Probabilities		Ground Truth	
3.2	exp →	24.5	norm →	0.13	← loss →	1	dog
5.1		168		0.87		0	cat
-1.7		0.18		0.00		0	bird

What we will learn today?

- Perceptron
- Linear classifier
- Loss function
- Gradient descent and backpropagation
- Neural networks

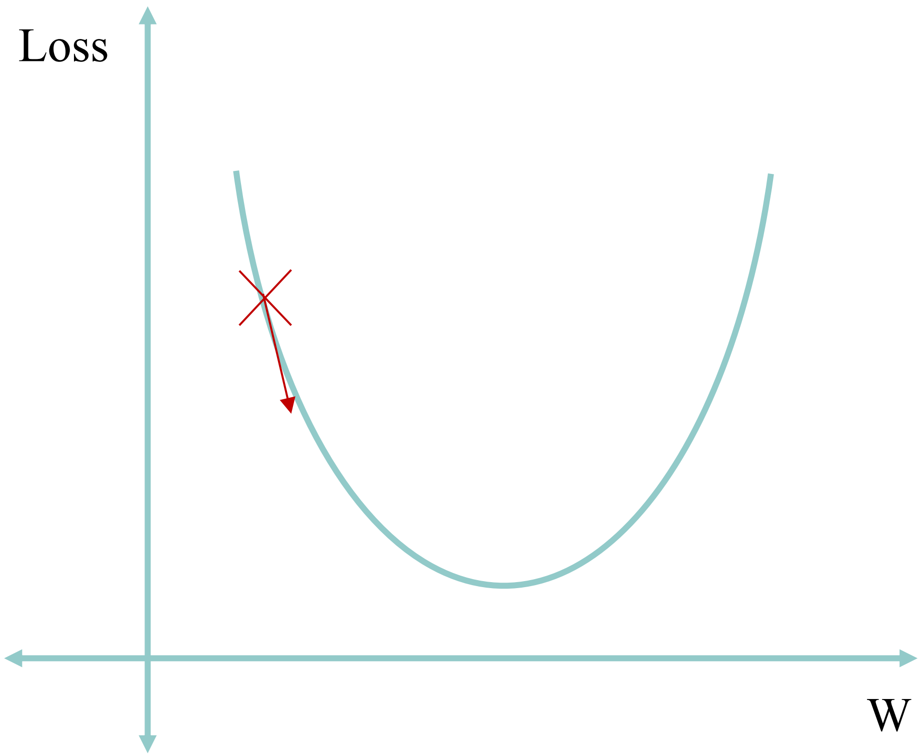
Gradient descent

- Visualization: Minimizing loss



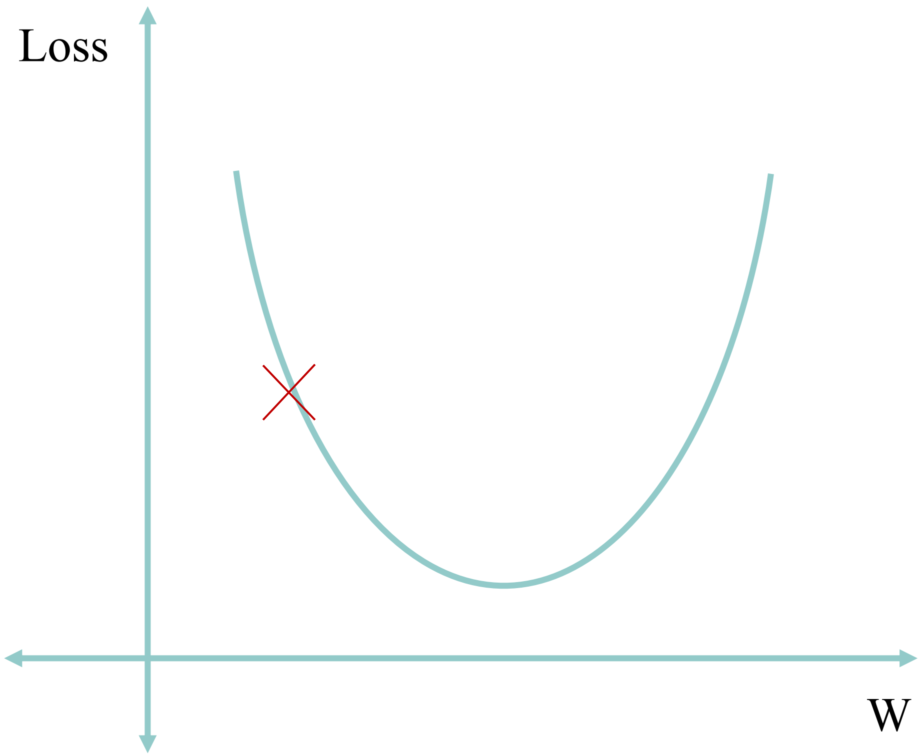
Gradient descent

- Visualization: Minimizing loss



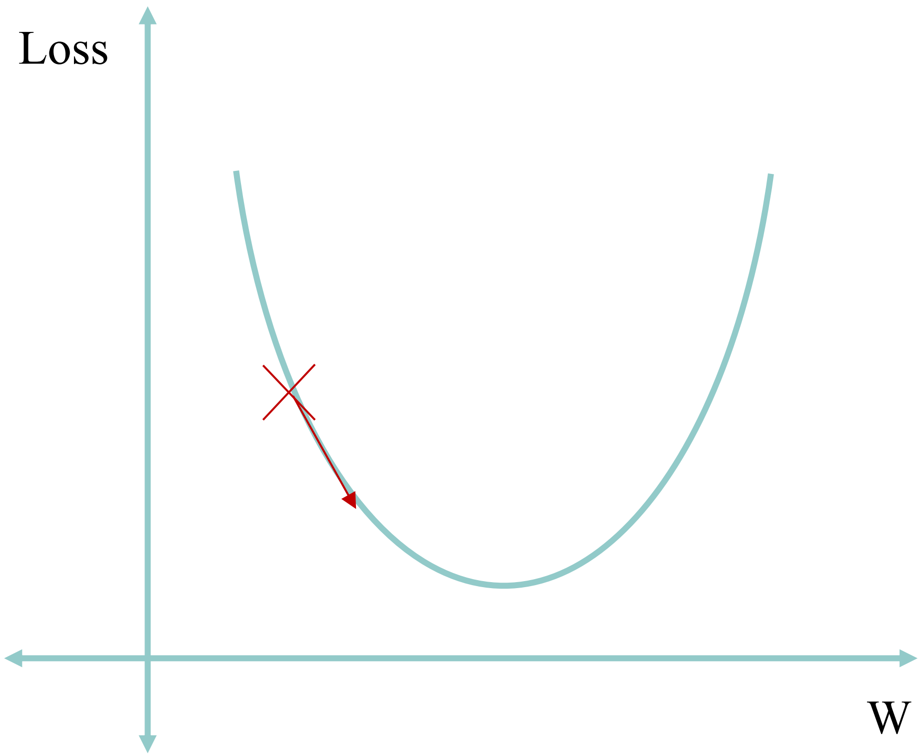
Gradient descent

- Visualization: Minimizing loss



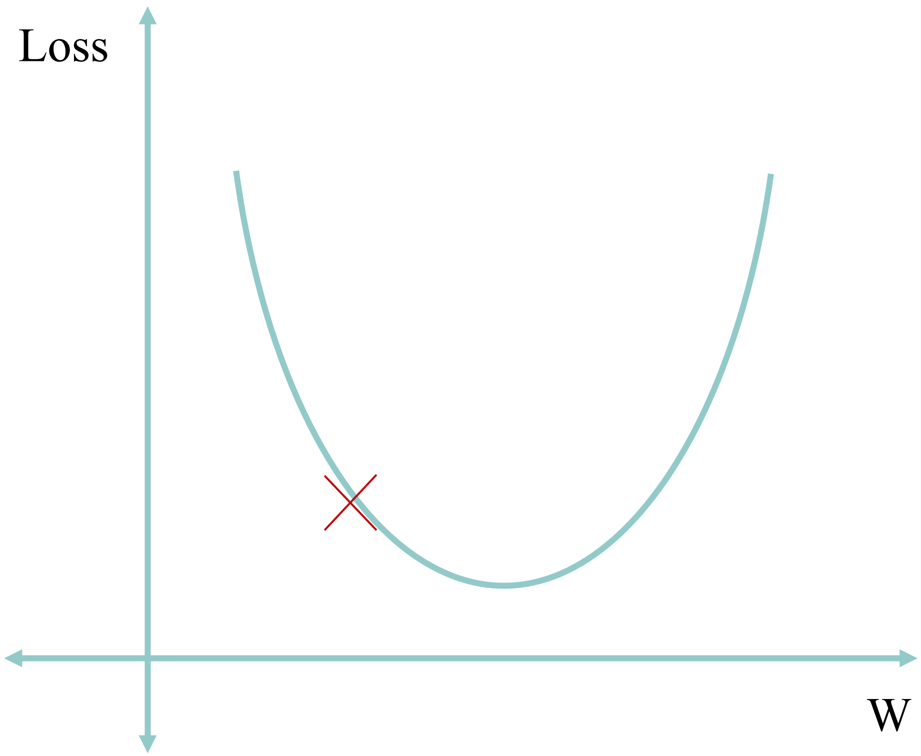
Gradient descent

- Visualization: Minimizing loss



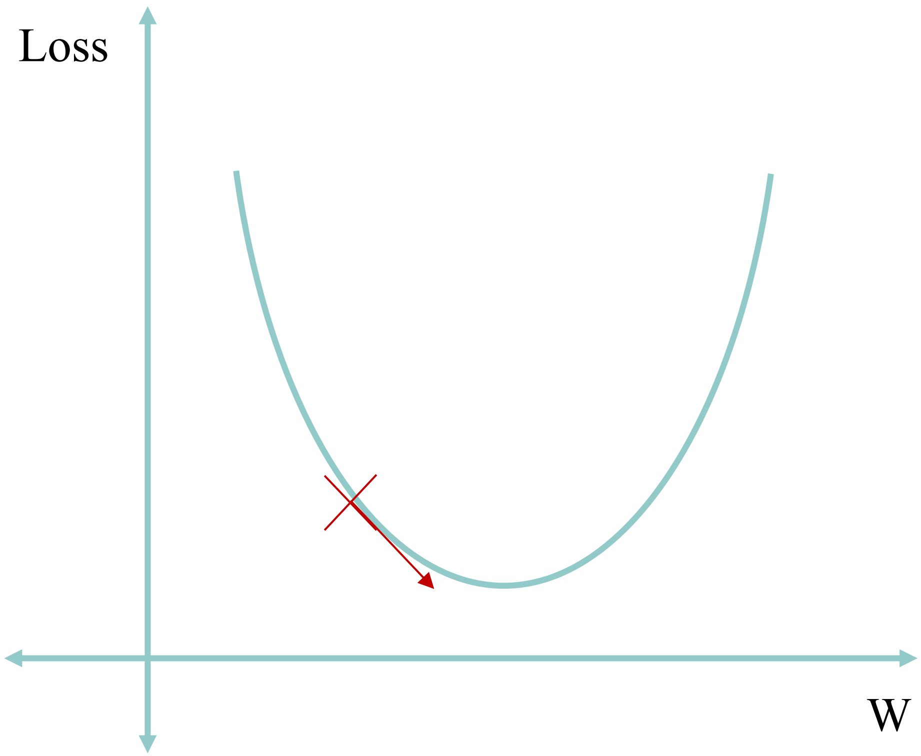
Gradient descent

- Visualization: Minimizing loss



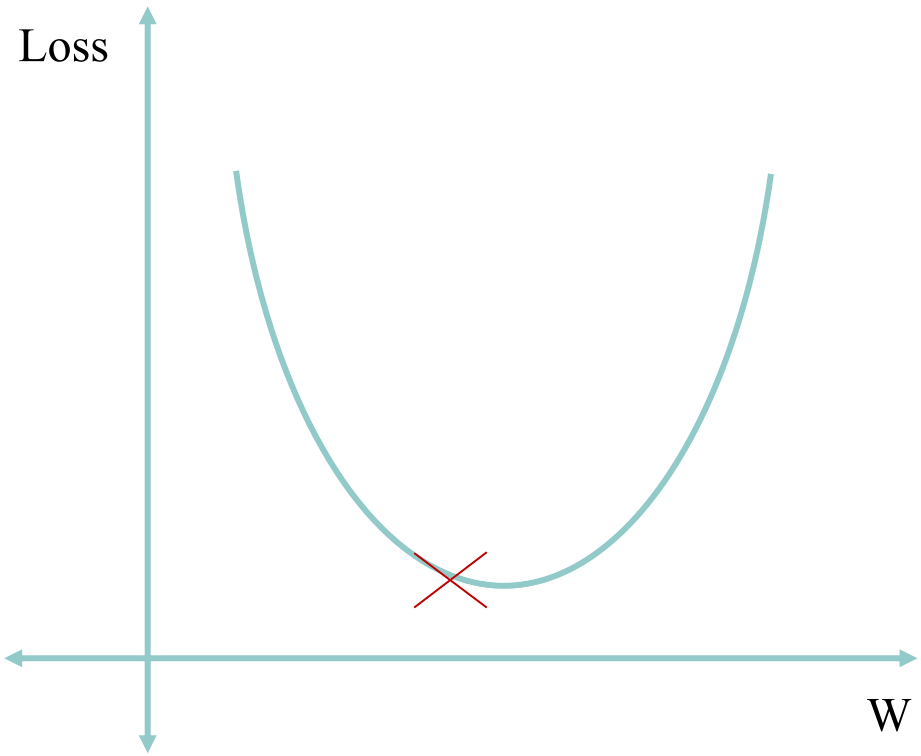
Gradient descent

- Visualization: Minimizing loss



Gradient descent

- Visualization: Minimizing loss



Gradient descent

- Gradient Descent Pseudocode

for _ in $\{0, \cdots, num\_epochs\}$:

$L = 0$

for x_i, y_i in data:

$\hat{y}_i = f(x_i, W)$

$L \mathrel{+}= L_i(y_i, \hat{y}_i)$

$\frac{dL}{dW} = ???$

$W := W - \alpha \frac{dL}{dW}$

Learning rate

Gradient descent

- Partial derivative of loss to update weights

Given training data point (x, y), the linear classifier formula is: $\hat{y}_i = \text{Wx}$

Let's assume that the correct label is class k, implying $y = k$

$$Loss = L(\hat{y}, y) = -log \frac{e^{\hat{y}k}}{\sum_j e^{\hat{y}j}} \text{ (Softmax loss)}$$

$$= -\hat{y}_k + log \sum_j e^{\hat{y}j}$$

Calculating the loss $\frac{dL}{dW}$ is hard mathematically. But we can use the chain rule to make it simpler:

$$\frac{dL}{dW} = \frac{dL}{d\hat{y}} \frac{d\hat{y}}{dW}$$

Gradient descent

- Partial derivative of loss to update weights

Given training data point (x, y), the linear classifier formula is: $\hat{y}_i = \mathrm{Wx}$
Let's assume that the correct label is class k, implying $y = k$

$$Loss = L(\hat{y}, y) = -log\,\frac{e^{\hat{y}k}}{\sum_j e^{\hat{y}j}} \quad \text{(Softmax loss)}$$

$$= -\hat{y}_k + log\sum_j e^{\hat{y}j}$$

Calculating the loss $\frac{dL}{dW}$ is hard mathematically. But we can use the chain rule to make it simpler:

$$\frac{dL}{dW} = \frac{dL}{d\hat{y}}\frac{d\hat{y}}{dW}$$

We know that $\frac{d\hat{y}}{dW} = \mathrm{x}$, but what about $\frac{dL}{d\hat{y}}$?

Gradient descent

- Partial derivative of loss to update weights

$$L = -\hat{y}_k + log \sum_j e^{\hat{y}j}$$

To calculate $\frac{dL}{d\hat{y}}$, we need to consider two cases:

Case1:

$$\frac{dL}{d\hat{y}_k} = -1 + \frac{e^{\hat{y}k}}{\sum_j e^{\hat{y}j}}$$

Case2:

$$\frac{dL}{d\hat{y}_{l\neq k}} = \frac{e^{\hat{y}l}}{\sum_j e^{\hat{y}j}}$$

Gradient descent

- Partial derivative of loss to update weights

Put it all together:

$$\frac{dL}{dW} = \frac{dL}{d\hat{y}}\frac{d\hat{y}}{dW}$$

$$\frac{dL}{dW} = \begin{pmatrix} \frac{e^{\hat{y}0}}{\sum_j e^{\hat{y}j}} \\ \cdots \\ -1 + \frac{e^{\hat{y}k}}{\sum_j e^{\hat{y}j}} \\ \cdots \\ \frac{e^{\hat{y}3071}}{\sum_j e^{\hat{y}j}} \end{pmatrix} x$$

Gradient descent

- Gradient Descent Pseudocode

for _ in $\{0, \cdots, num\_epochs\}$:

$L = 0$

for x_i, y_i in data:

$\hat{y}_i = f(x_i, W)$

$L \mathrel{+}= L_i(y_i, \hat{y}_i)$

$\frac{dL}{dW} = We\ know\ how\ to\ calculate\ this\ now!$

$W := W - \alpha \frac{dL}{dW}$

Gradient descent

- Gradient Descent Pseudocode

for _ in $\{0, \cdots, num\_epochs\}$:

$L = 0$

for x_i, y_i in data:

$\hat{y}_i = f(x_i, W)$

$L \mathrel{+}= L_i(y_i, \hat{y}_i)$

$\frac{dL}{dW} = We\ know\ how\ to\ calculate\ this\ now!$

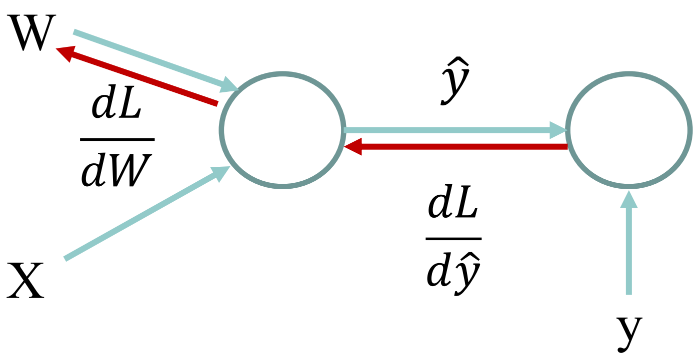
$W := W - \alpha \frac{dL}{dW}$

After num_epochs, W is well suited to the classification task

Backprop

- Backprop – another way of computing gradients

  - visualize the computation as a graph
  - Compute the forward pass to calculate the loss.
  - Compute all gradients for each computation backwards



$$\hat{y} = Wx$$

$$L = Loss(\hat{y}, y)$$

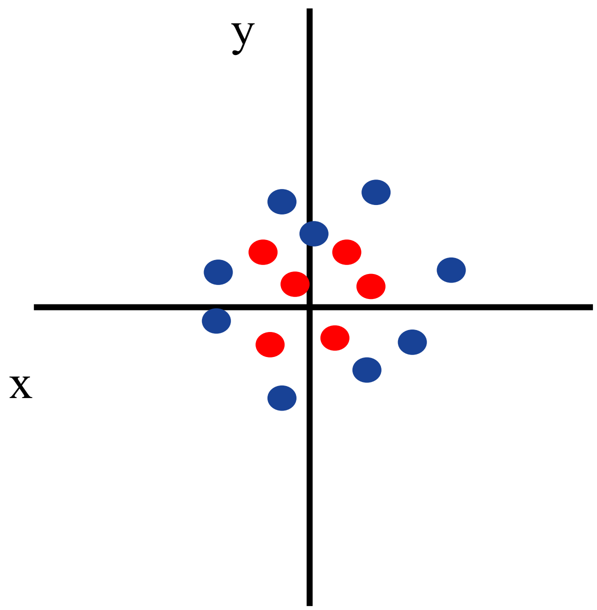
$$\frac{dL}{dW} = \frac{dL}{d\hat{y}} \frac{d\hat{y}}{dW}$$

What we will learn today?

- Perceptron
- Linear classifier
- Loss function
- Gradient descent and backpropagation
- Neural networks

Neural networks

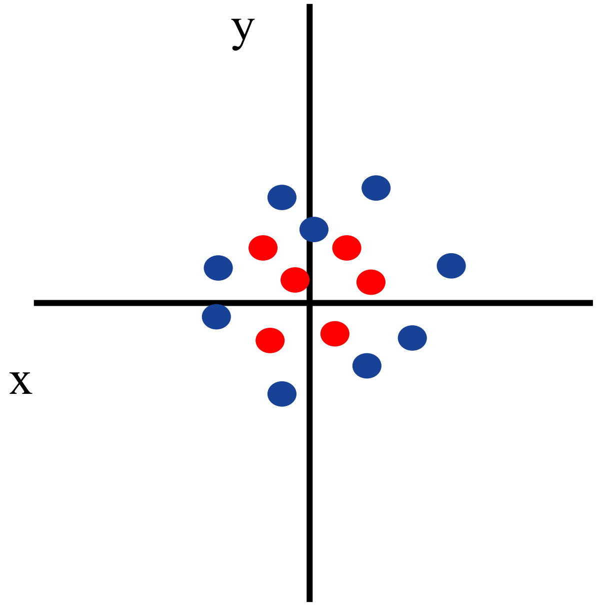
- Features sometimes might not be linearly separable



What should we do in this case?

Neural networks

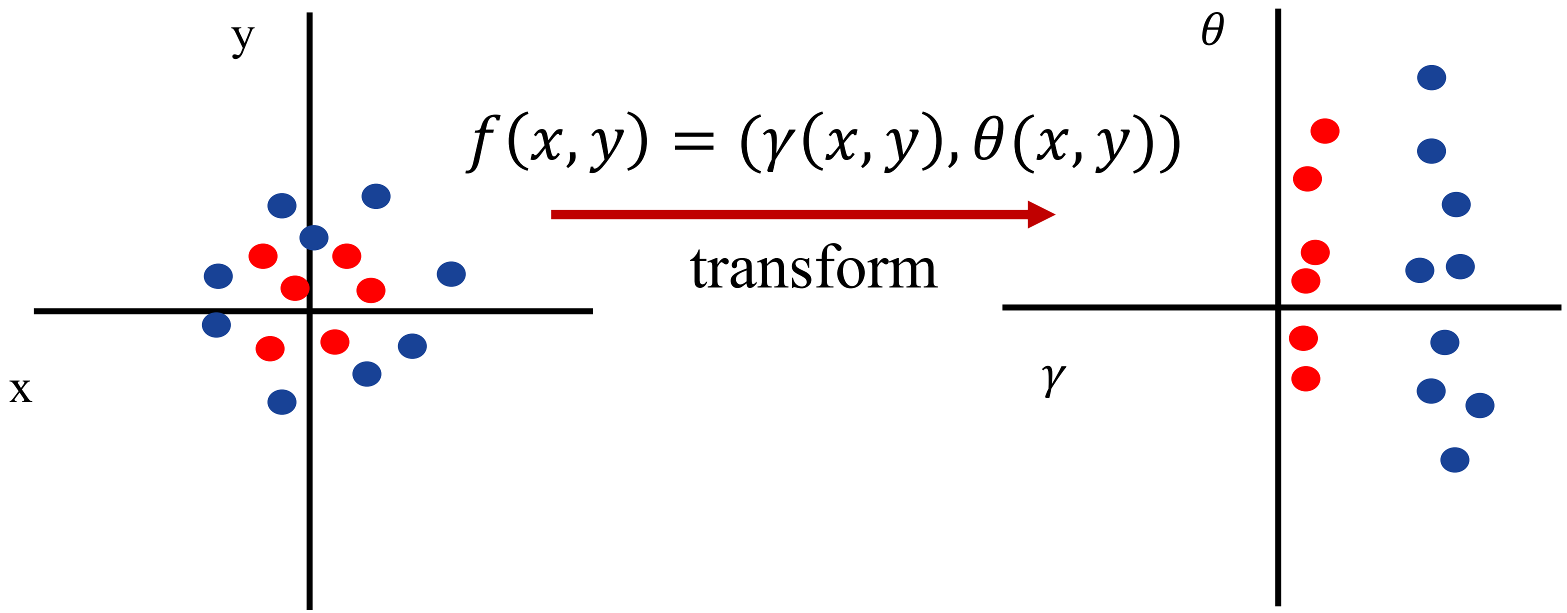
- Features sometimes might not be linearly separable



We can transform it to be linearly separable!

Neural networks

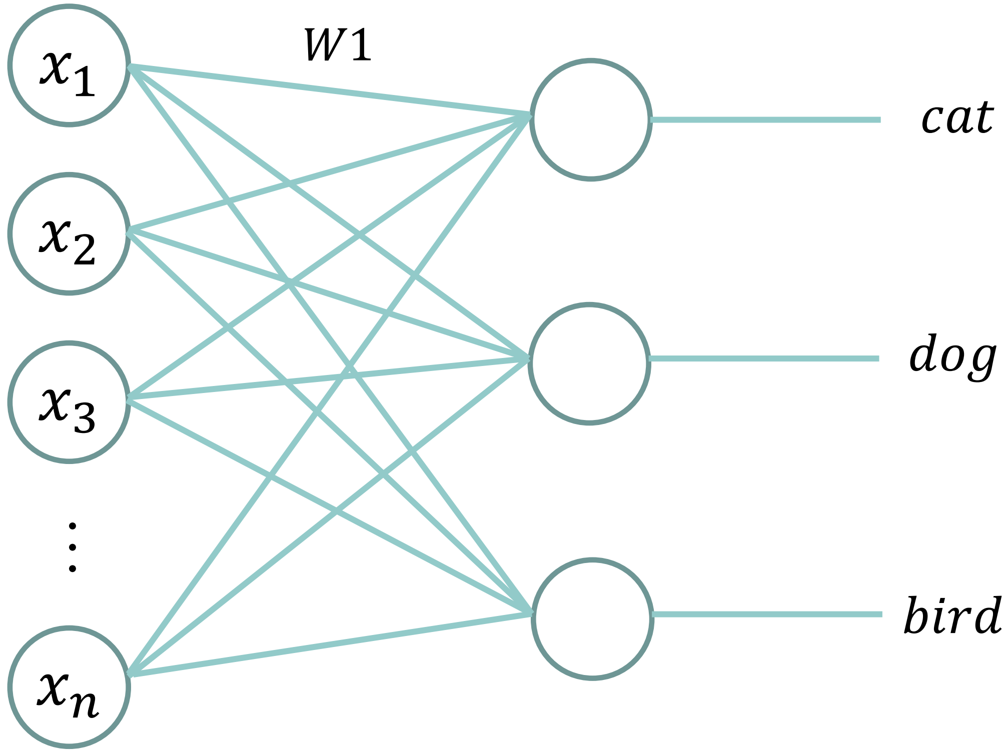
- Features sometimes might not be linearly separable



Neural networks

- Recall: Our linear classifier

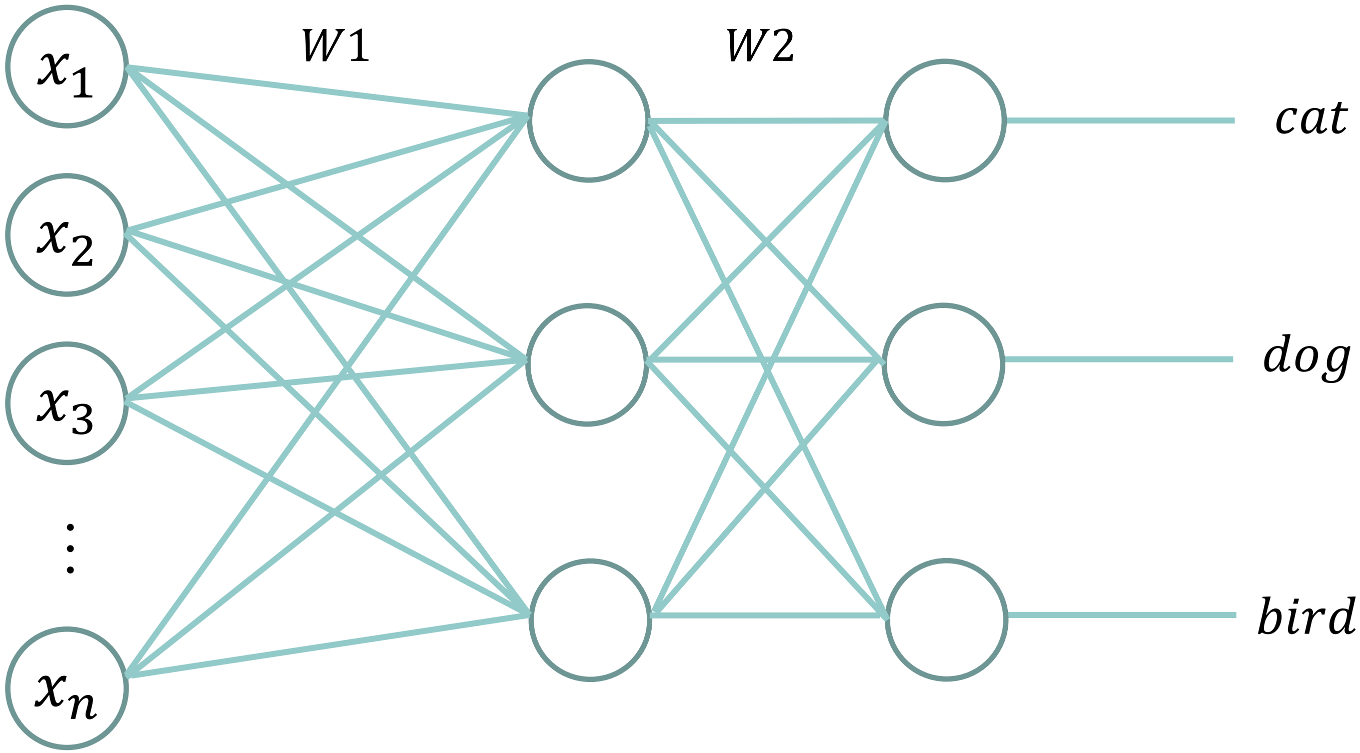
inputs



Neural networks

- Transform the features by adding another layer

inputs



Neural networks

- 2-layer network: mathematical formula

linear classifier:

$$y = Wx$$

2-layer network:

$$y = W_2(\max(0, W_1 x))$$

We know the size of $x = 3072 \times 1$ and $y = 10 \times 1$

So:

$$W_1 = h \times 3072 \text{ and } W_2 = 10 \times h$$

h is a new hyperparameter!

Neural networks

- 2-layer network: mathematical formula

linear classifier:

$$y = Wx$$

Activation function

2-layer network:

$$y = W_2(\boxed{\max}(0, W_1 x))$$

We know the size of $x = 3072 \times 1$ and $y = 10 \times 1$

So:

$$W_1 = h \times 3072 \text{ and } W_2 = 10 \times h$$

h is a new hyperparameter!

Neural networks

- 2-layer network: mathematical formula

linear classifier:

$$y = Wx$$

Activation function

2-layer network:

$$y = W_2(\boxed{\max}(0, W_1 x))$$

Why is the activation function necessary?

If we remove it, the neural network turns to be a linear classifier

$$y = W_2 W_1 x = Wx$$

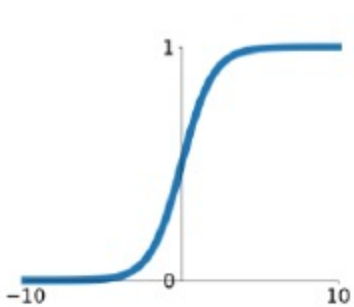
Neural networks

- Activation function

It allows models to learn complex transformations for features.
Choosing the right activation function is important!

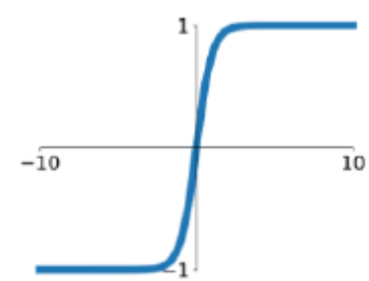
a. Sigmoid (binary classification)

$$\delta(x) = \frac{1}{1+e^{-x}}$$



b. tanh

$$\tanh(x)$$



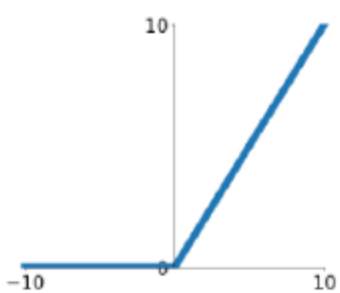
Neural networks

- Activation function

It allows models to learn complex transformations for features.
Choosing the right activation function is important!

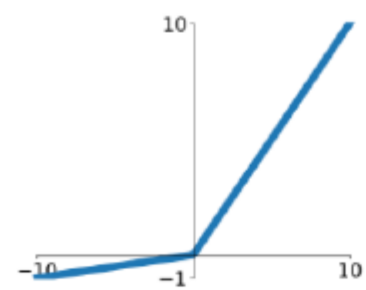
c. ReLU (Rectified Linear Unit, ReLU)

$\max(0, x)$



d. Leaky ReLU

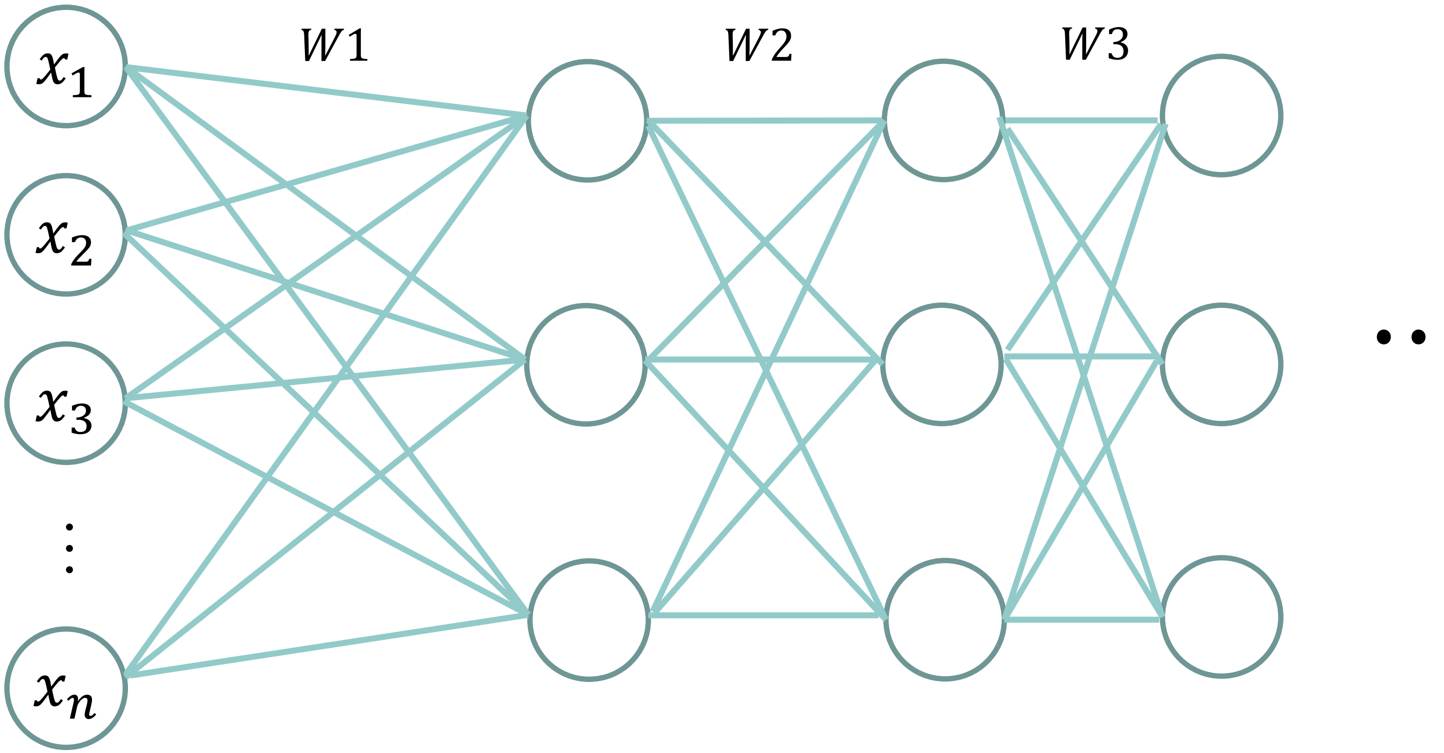
$\max(0.1x, x)$



Neural networks

- Multi-layers Neural networks

inputs



What we have learned today?

- Perceptron
- Linear classifier
- Loss function
- Gradient descent and backpropagation
- Neural networks

What we have learned today?

- Perceptron----Inspiration from Biology
- Linear classifier----A set of perceptrons
- Loss function----Determines how good our classifier
- Gradient descent and backpropagation----Training
- Neural networks---Handle complex feature distribution

Questions?