3010 Artificial Intelligence

Lecture 7 Perceptron and neural networks

Slides courtesy of Hiroshi Noji
(with some minor modifications by MS)

What is machine learning?

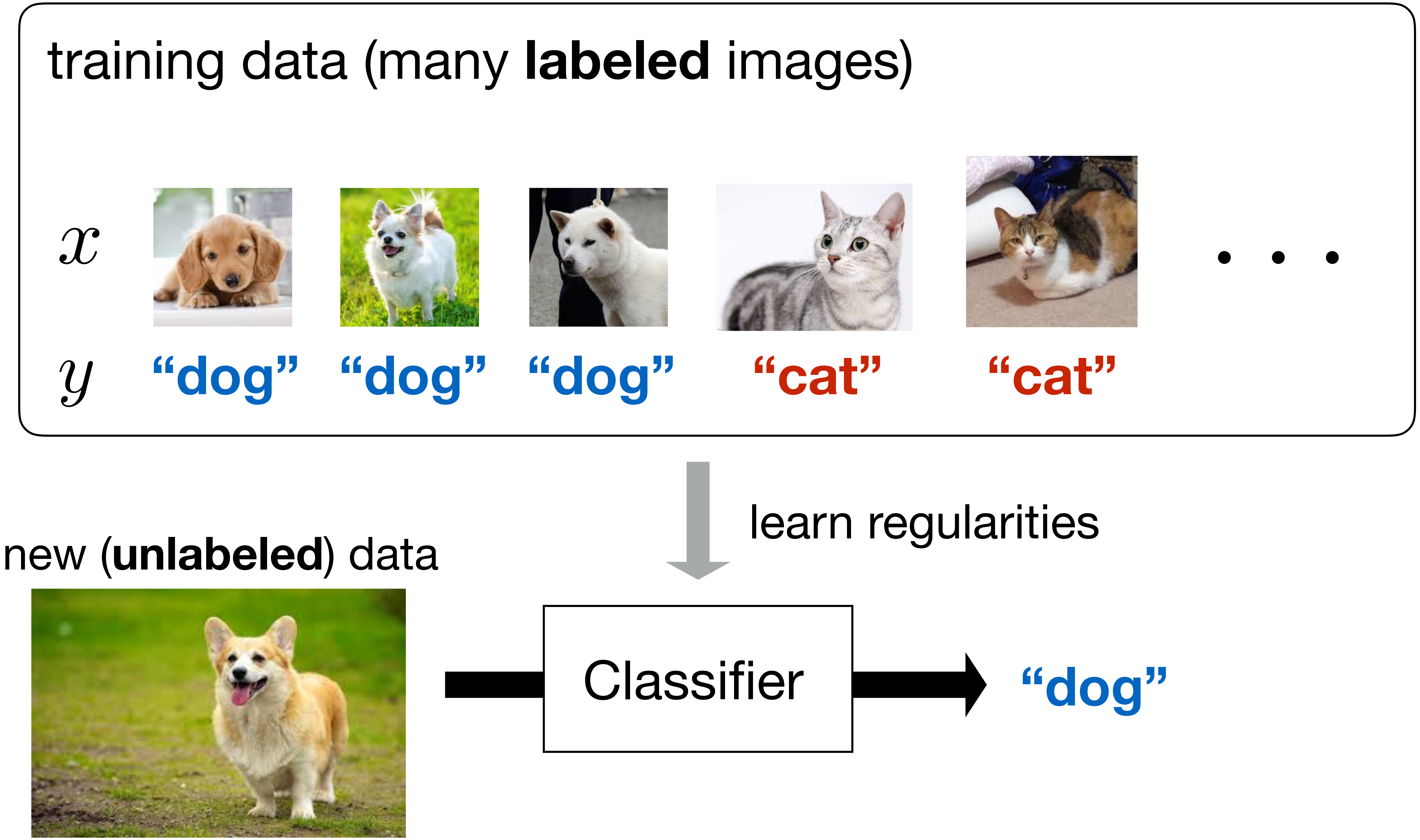
- Neural networks are one technique for machine learning, in particular for **supervised machine learning**

- Problem: learn a function f that maps input $x \in \mathcal{X}$ to $y \in \mathcal{Y}$

$$f(x) \to y$$

- The most typical problem is **classification**
  - $\mathcal{Y} = \{1, \cdots, K\}$ … a discrete set of (categorical) **class labels**
  - x is classified into one of $\{1, \cdots, K\}$
  - In other words, f assigns a label y to an input x
  - A classifier learns the mapping f from many pairs of (x, y), called the **training data**

Computer vision



Sentiment analysis

x	y
The final season was a massive disappointment	negative
Their computer animated faces are very expressive	positive

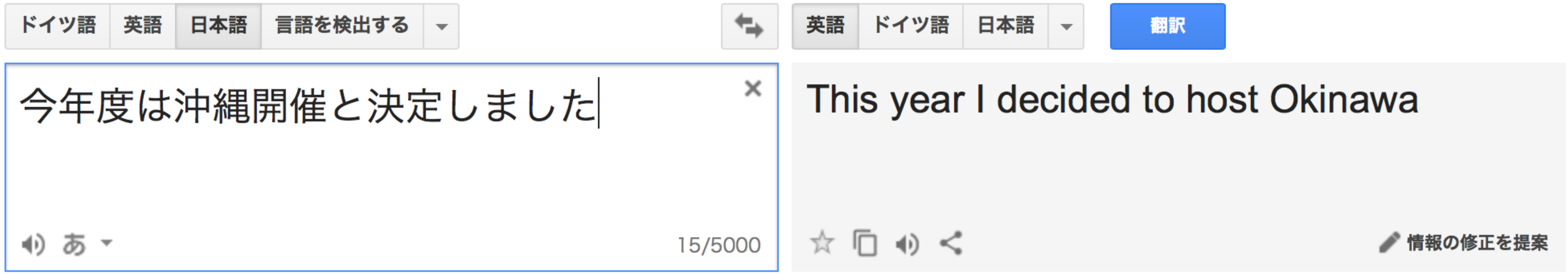
- Task: Is the opinion expressed in an utterance negative, or positive?
- Training data: Many labeled sentences
- At test time, predict the opinion of a new utterance

By far the finest Chardonnay I have tasted to date …

- Many problems can be formulated as binary classification:
  - E-mail spam filtering; document (news) classification

Machine translation

x y

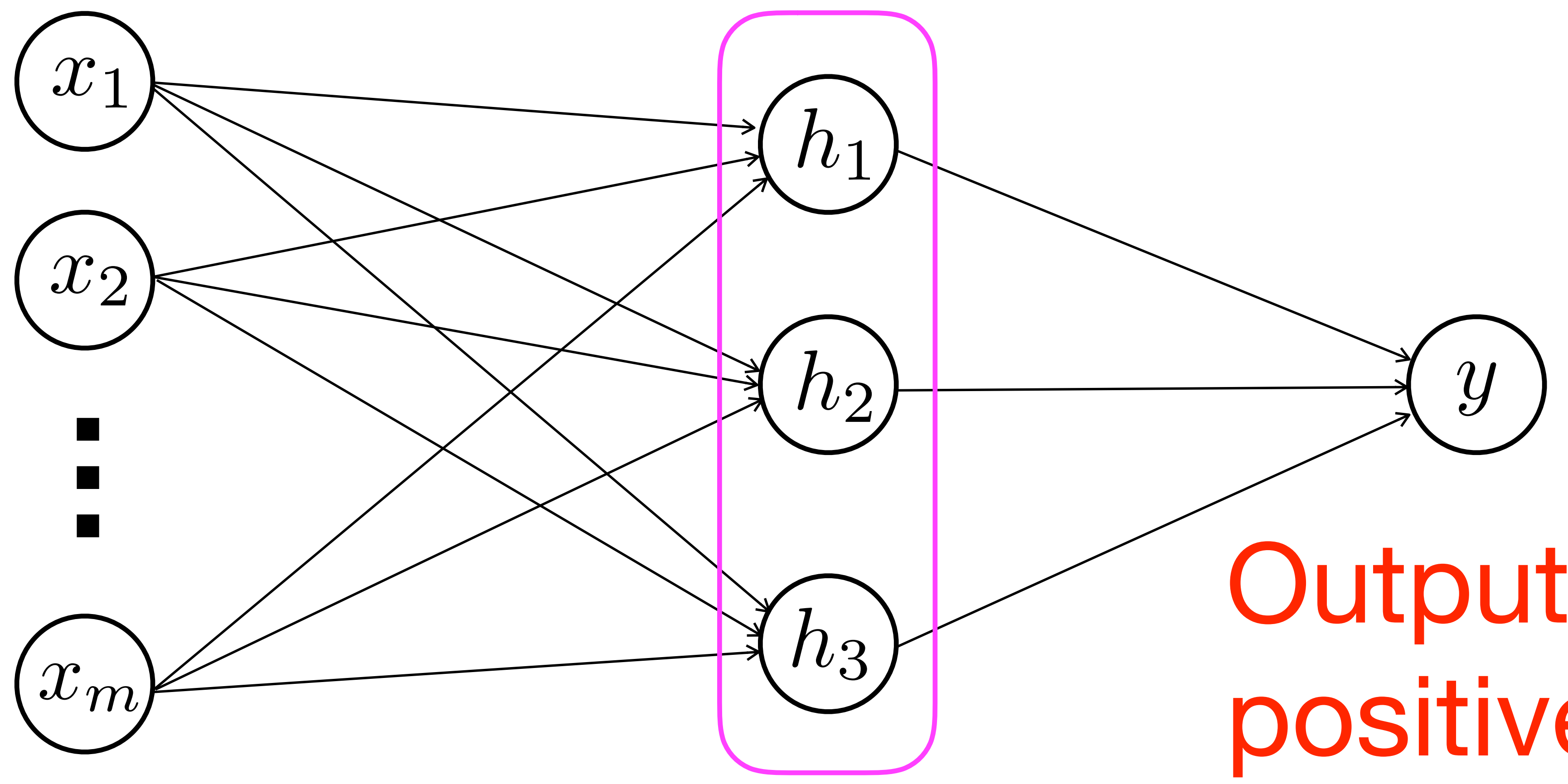


- Task: Translate a sentence in one language to another
- Training data: Large amount of translation pairs
- Quality of Google translation has significantly improved since neural networks are introduced
  - Especially seq-to-seq models;
    details are given in Sequential Data Mining course

What is an artificial neural network?

Raw input:
"… finest Chardonnay I've tasted"

Convert to an input vector



Output:
positive/negative

hidden vector

- A technique for machine learning ≒ classification
  - Other techniques: SVM, decision trees, etc
- Characteristics: the input is converted to a hidden vector, and then the output label (y) is calculated

Neural networks

- Neural networks (NNs) become more and more important for all areas related to machine learning
- In many areas, NN-based systems outperform other machine learning methods by large margin

Why are NNs successful?

- NN itself is a classic (old) idea in AI, but early attempts were not very successful
  - first appears in 1940’s
  - extensively studied in 1970’s - 1990’s, but due to its higher **representation power**, learning NNs was quite difficult
  - in 2000’s, other machine learning techniques, such as SVM, gain much popularity, and studies on **NNs were not mainstream**
- Gradually succeeded since 2006 ~
  - Several innovations, such as dropout, have been invented
  - Most important is the recent advance of **machine power** (e.g., GPGPU), which enables handling of **big data**

So …

- Neural nets are a powerful and very important tool for machine learning
- This and the next class cover the basics of NNs

Topics covered in this course

Today

- Perceptron
  - Simplest form of neural networks
  - Feed-forward neural networks (multilayer perceptrons)

June 3

- Training neural networks
  - Stochastic gradient descent, back propagation, etc

Outline

- Linear classification and perceptron
- 2-layer neural networks (multi-layer perceptron)
- Next week: how to learn neural networks

Perceptron?

- Perceptron is a **learning algorithm** for **linear classification**
- Linear classification is a basic yet important problem in machine learning (next)
  - Many practical problems can be formulated as a linear classification problem
- In contrast, **multi-layer** neural networks are **non-linear** models
  - Highly effective if trained properly - mathematically and computationally more involved

Binary classification

- Recall: machine learning aims to a learn function f that maps an input $x \in \mathcal{X}$ to an output $y \in \mathcal{Y}$

$$f(x) \to y$$

- Here let us assume $\mathcal{Y} = \{-1, +1\}$
  - This problem is called **binary classification**

- Perceptron can be best understood in terms of binary classification
  - Generalization to multi-class classification is discussed later

Linear classification

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

- Assume the input $\mathbf{x}$ is an m-dimensional vector $(\mathcal{X} = \mathbb{R}^m)$

$$\mathbf{x} = (x_1, x_2, \cdots, x_m)^\top$$

- Linear classification model has the following parameters:
  - weight vector $\mathbf{w} = (w_1, w_2, \cdots, w_m)^\top \in \mathbb{R}^m$
  - bias parameter (scalar) $b \in \mathbb{R}$
- For linear classification, f is defined as:

$$f(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} + b = w_1 x_1 + w_2 x_2 + \cdot + w_m x_m + b$$

  - and, output $y = +1$ when $f(\mathbf{x}) \geq 0$; and $y = -1$ when $f(\mathbf{x}) < 0$

Notes on the bias term

$$f(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} + b = w_1 x_1 + w_2 x_2 + \cdot + w_m x_m + b$$

$$f(\mathbf{x}) \geq 0 \Rightarrow \; y = +1$$

$$f(\mathbf{x}) < 0 \Rightarrow \; y = -1$$

- The bias b is always added regardless of the input
  - Positive value of b means y tends to be +1 a priori
  - Negative value of b means the opposite

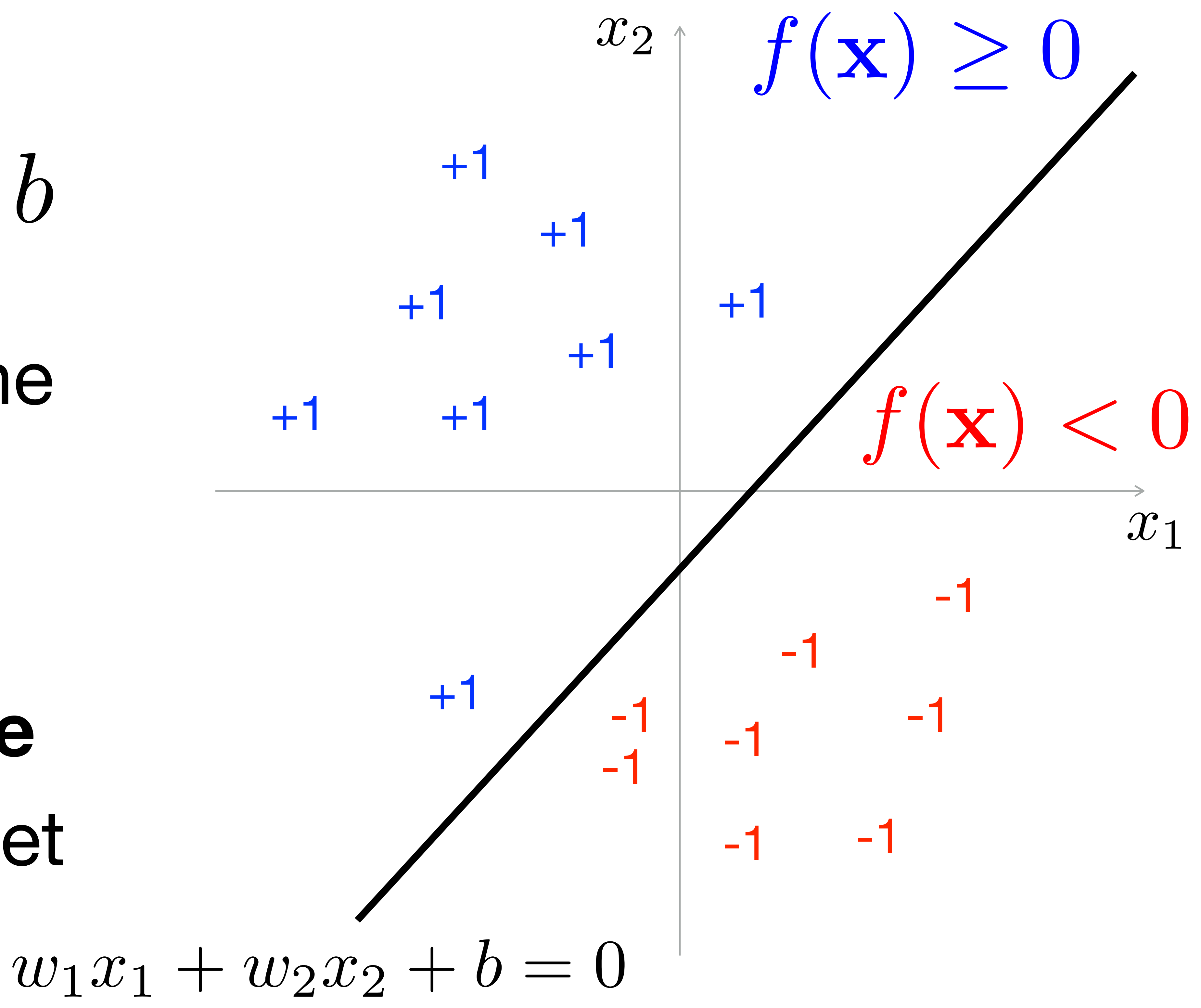
Interpretation

$$f(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} + b = w_1 x_1 + w_2 x_2 + \cdot + w_m x_m + b$$
$$f(\mathbf{x}) \geq 0 \Rightarrow \ y = +1$$
$$f(\mathbf{x}) < 0 \Rightarrow \ y = -1$$

- When $m = 2$

$$f(\mathbf{x}) = w_1 x_1 + w_2 x_2 + b$$

- Each data (x_1, x_2) is a point on the 2-dimensional space
- The goal of learning is to find (w_1, w_2, b) that can **separate** the labeled data in the training set



Problem: find parameters

$$f(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} + b = w_1 x_1 + w_2 x_2 + \cdot + w_m x_m + b$$

$$f(\mathbf{x}) \geq 0 \Rightarrow \ y = +1$$

$$f(\mathbf{x}) < 0 \Rightarrow \ y = -1$$

- Given many labeled data $(\mathbf{x}_i, y_i)$, we want to find $\mathbf{w}$ and b that can correctly classify these training data
  - In other words, all data should satisfy $f(\mathbf{x}_i) \cdot y_i > 0$ after training
- Note: our goal is to correctly predict the label of **unseen data**
  - Intuition: If we get a classifier that can classify the training data, we expect that that can classify the unseen data as well
  - There is a problem of **overfitting**; we discuss it later

Example: spam filtering

- So far we have abstracted the input and output as $\mathbf{x}$ and y

- Spam filtering is an example of document classification
  - Given an **e-mail text**, classify whether it is spam or not
  - A typical way to mapping the text into vector $\mathbf{x}$ is **bag-of-words**:

… $\boxed{\text{click}}$ here to receive $10.5 million $\boxed{\text{dollars}}$ …

$$\mathbf{x} = \begin{pmatrix} 0 \\ 0 \\ \color{red}{1} \\ \vdots \\ \color{blue}{1} \\ 0 \\ \vdots \end{pmatrix}$$

dimension = size of vocabulary

- **Each word** corresponds to **one-dimension** of $\mathbf{x}$

Weight vector: Intuition

$$f(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} + b = w_1 x_1 + w_2 x_2 + \cdot + w_m x_m + b$$

... click here to receive $10.5 million dollars ...

$$\mathbf{x} = \begin{pmatrix} 0 \\ 0 \\ 1 \\ \vdots \\ 1 \\ 0 \\ \vdots \end{pmatrix} \quad \mathbf{w} = \begin{pmatrix} 1.1 \\ -0.1 \\ -13.8 \\ \vdots \\ -5.2 \\ 0.2 \\ \vdots \end{pmatrix}$$

- After training, probably the weight w_i for x_i that likely occurs in the spam would be negative (with large absolute value)
- How can we obtain such weights?

[Rosenblatt 1957]

Perceptron algorithm

Initialize to $\mathbf{w} \leftarrow (0, 0, \cdots, 0); b \leftarrow 0$

Loop:

  Randomly pick up $(\mathbf{x}_i, y_i)$

  $s \leftarrow y_i \cdot (\mathbf{w} \cdot \mathbf{x} + b)$

  if $s < 0$: # failed to predict

    $\mathbf{w} \leftarrow \mathbf{w} + y_i \mathbf{x}_i;\ b \leftarrow b + y_i$

until convergence

Why does perceptron work?

- Recall: if the prediction is correct, $y_i f(\mathbf{x}) = y_i(\mathbf{w} \cdot \mathbf{x} + b) \geq 0$

- **Updated weights** when the prediction is failed:

$$\mathbf{w}' = \mathbf{w} + y_i \mathbf{x}_i; \; b' = b + y_i$$

- Let us predict the label of $\mathbf{x}_i$ again with this parameter:

$$\begin{aligned} f'(\mathbf{x}) &= \mathbf{w}' \cdot \mathbf{x}_i + b' = (\mathbf{w} + y_i \mathbf{x}_i) \cdot \mathbf{x}_i + (b + y_i) \\ &= \underbrace{\mathbf{w} \cdot \mathbf{x}_i + b}_{\text{last prediction}} + y_i \underbrace{(\mathbf{x}_i \cdot \mathbf{x}_i + 1)}_{\text{always positive}} \end{aligned}$$

- Thus: $y_i f'(\mathbf{x}) = \underbrace{y_i(\mathbf{w} \cdot \mathbf{x} + b)}_{y_i f(\mathbf{x})} + \boxed{y_i^2(\mathbf{x}_i \cdot \mathbf{x}_i + 1)} > y_i f(\mathbf{x})$

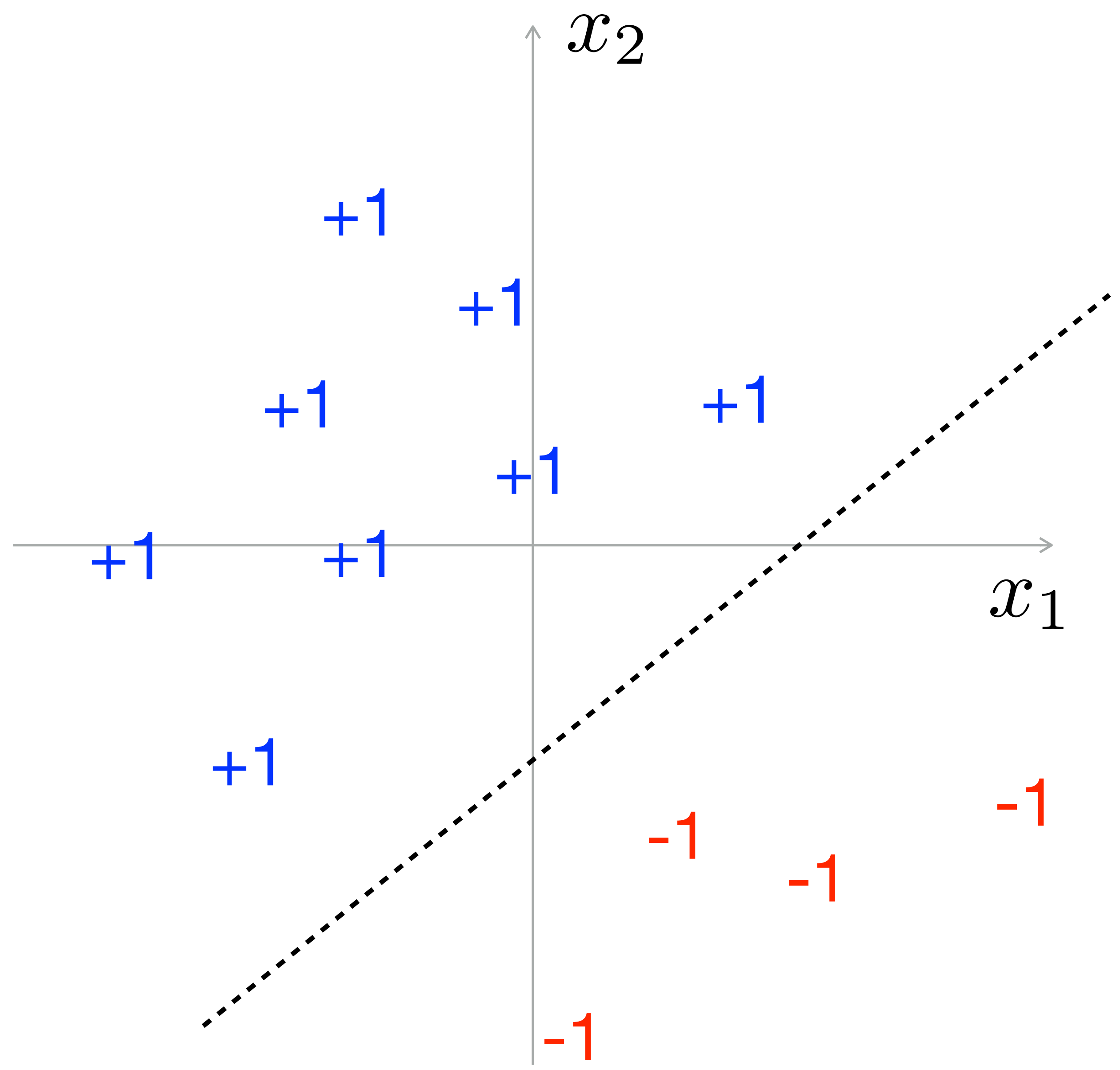
push to positive

$\Rightarrow$ better prediction for $\mathbf{x}_i$

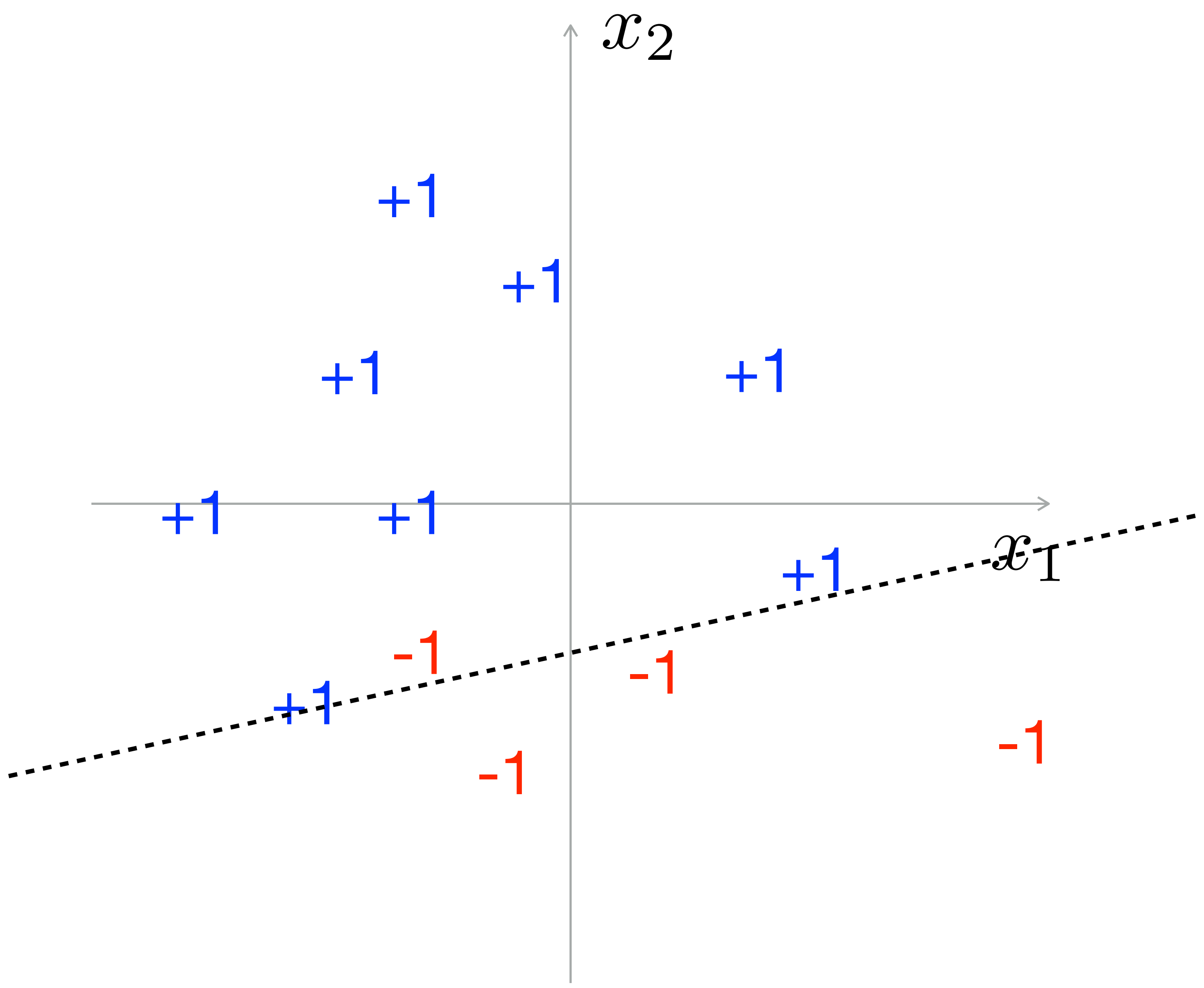
Does perceptron converge?

- Question: By updating parameters to classify **only the last example**, can we eventually classify all the training data?

- Answer: Yes, if the training data is linearly separable
  - This is known as the perceptron convergence theorem



Linearly separable

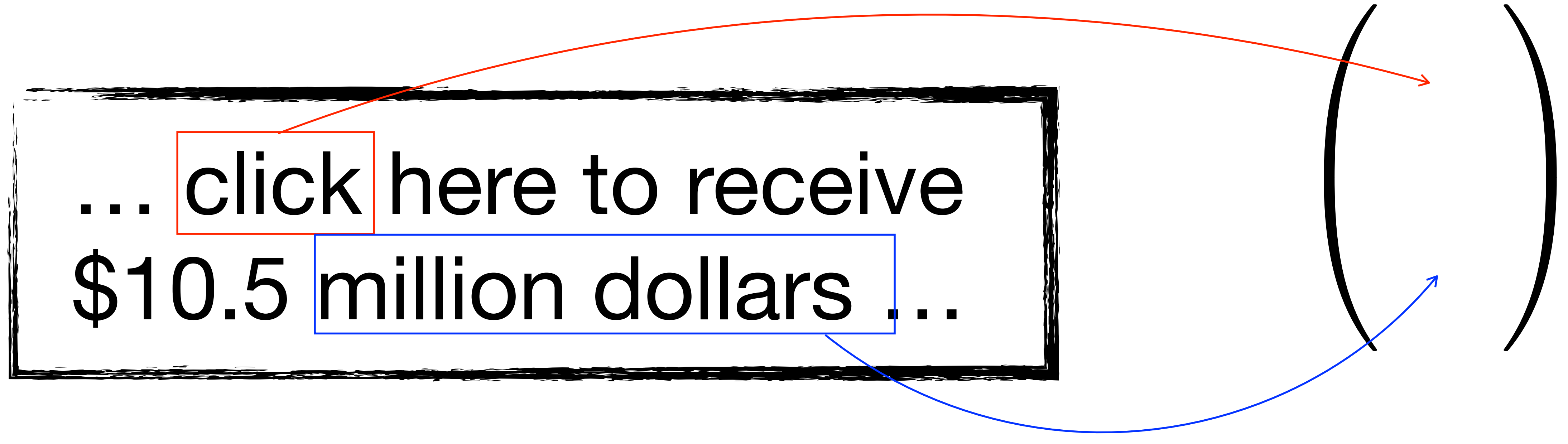


Not linearly separable

Is the perceptron a practical tool?

- Perceptron has been criticized because of its limitation to the linear classification problem [Minsky, 1969]
  - For a long time, it has been thought to be too primitive

- Perceptron **gained popularity** again in 2000s
  - For problems involving structured data (e.g., sequence labeling), perceptron is an effective tool [Collins, 2002]
  - We may modify the data linearly separable by extending the mapping from the input to the vector $\mathbf{x}$ (**feature extraction**)



Count consecutive-words (bi-grams) as well

Averaged perceptron [Schapire and Freund 1999]

- For perceptron, averaging all weights at each step works improves performance

$$\mathbf{w}^{(0)} \to \mathbf{w}^{(1)} \to \cdots \to \mathbf{w}^{(N)} \Rightarrow \boxed{\mathbf{w} = \frac{1}{N}\sum_{i=0}^{N} \mathbf{w}^{(i)}}$$

An efficient algorithm for averaging

Initialize to:

$\mathbf{w}_0 \leftarrow (0, 0, \cdots, 0); b_0 \leftarrow 0$

$\mathbf{w}_a \leftarrow (0, 0, \cdots, 0); b_a \leftarrow 0$

$c \leftarrow 1$

Loop:

Randomly pick up $(\mathbf{x}_i, y_i)$

if $y_i \cdot (\mathbf{w}_0 \cdot \mathbf{x} + b_0) < 0$:

$\mathbf{w}_0 \leftarrow \mathbf{w}_0 + y_i \mathbf{x}_i; b_0 \leftarrow b_0 + y_i$

$\mathbf{w}_a \leftarrow \mathbf{w}_a + c y_i \mathbf{x}_i; b_a \leftarrow b_a + c y_i$

$c \leftarrow c + 1$

until convergence

$\mathbf{w} \leftarrow \mathbf{w}_0 - \mathbf{w}_a / c;\ b \leftarrow b_0 - b_a / c$

Multi-class classification

- So far the output is binary: $y \in \mathcal{Y} = \{-1, +1\}$
- For multi-class classification: $\mathcal{Y} = \{1, 2, \cdots, K\}$
- Weight vector is prepared for each class:

$$\mathbf{w}_1, \mathbf{w}_2, \cdots, \mathbf{w}_K$$
$$b_1, b_2, \cdots, b_K$$

- Then, the output is the class with the highest score:

$$\hat{y} = \arg\max_{y \in \mathcal{Y}} \mathbf{w}_k \cdot \mathbf{x} + b_k$$

- How to obtain these parameters?

Multi-class perceptron

Initialize to $\mathbf{w}_k \leftarrow (0, 0, \cdots, 0);\ b_k \leftarrow 0$

Loop:

Randomly pick up $(\mathbf{x}_i, y_i)$

$\hat{y} = \arg\max_{y \in \mathcal{Y}} \mathbf{w}_k \cdot \mathbf{x} + b_k$

if $\hat{y} \neq y_i$: # failed to predict

$\boxed{\mathbf{w}_{y_i} \leftarrow \mathbf{w}_{y_i} + \mathbf{x}_i;\ b_{y_i} \leftarrow b_{y_i} + 1}$

$\mathbf{w}_{\hat{y}} \leftarrow \mathbf{w}_{\hat{y}} - \mathbf{x}_i;\ b_{\hat{y}} \leftarrow b_{\hat{y}} - 1$

until convergence

Increase the score for the correct label

Summary on perceptron

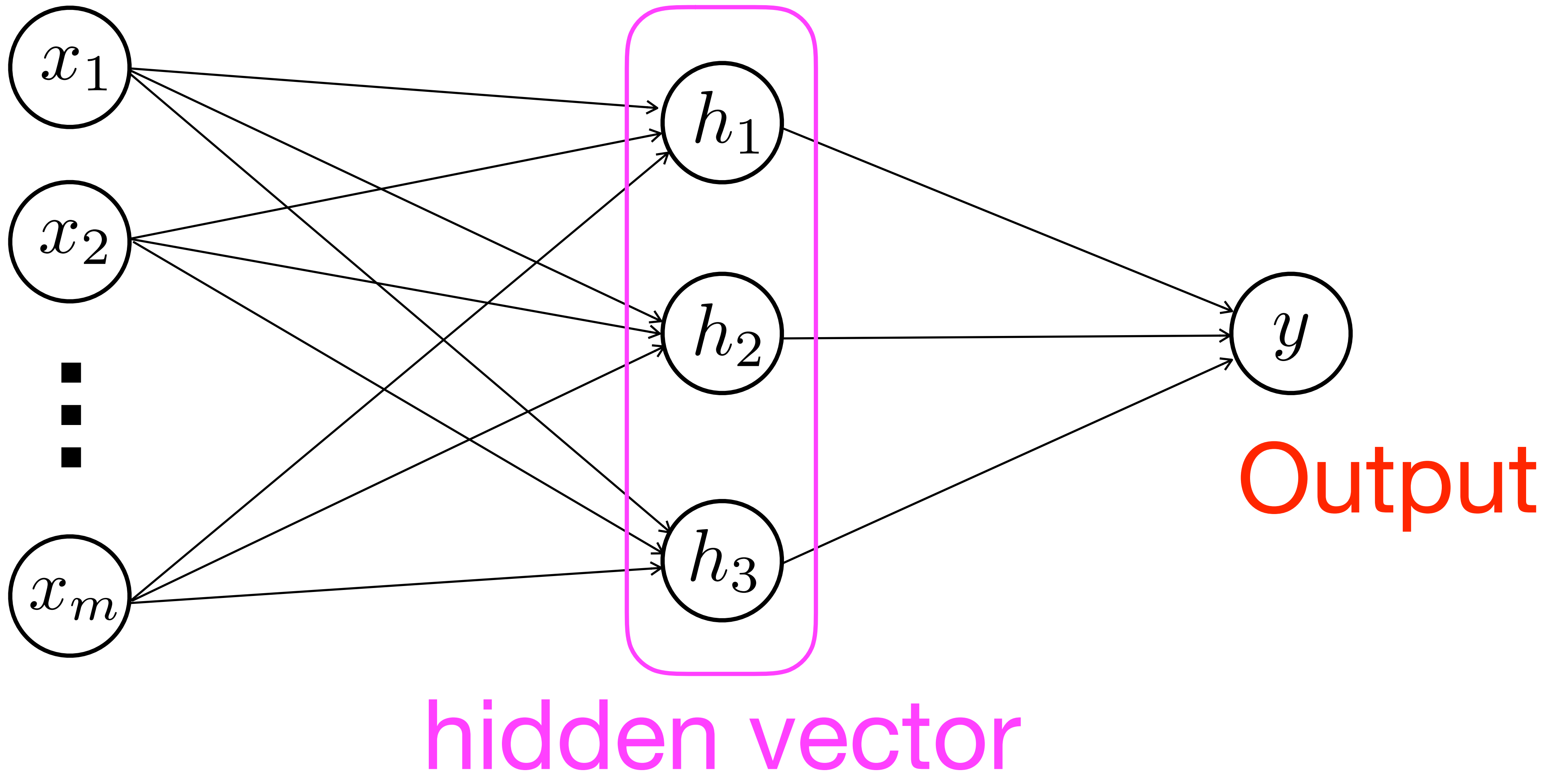
- The simplest, and easy-to-implement learning algorithm for linear classification
- But still a practical tool for many applications
  - Weight averaging is important in practice

What is a multi-layer neural network?

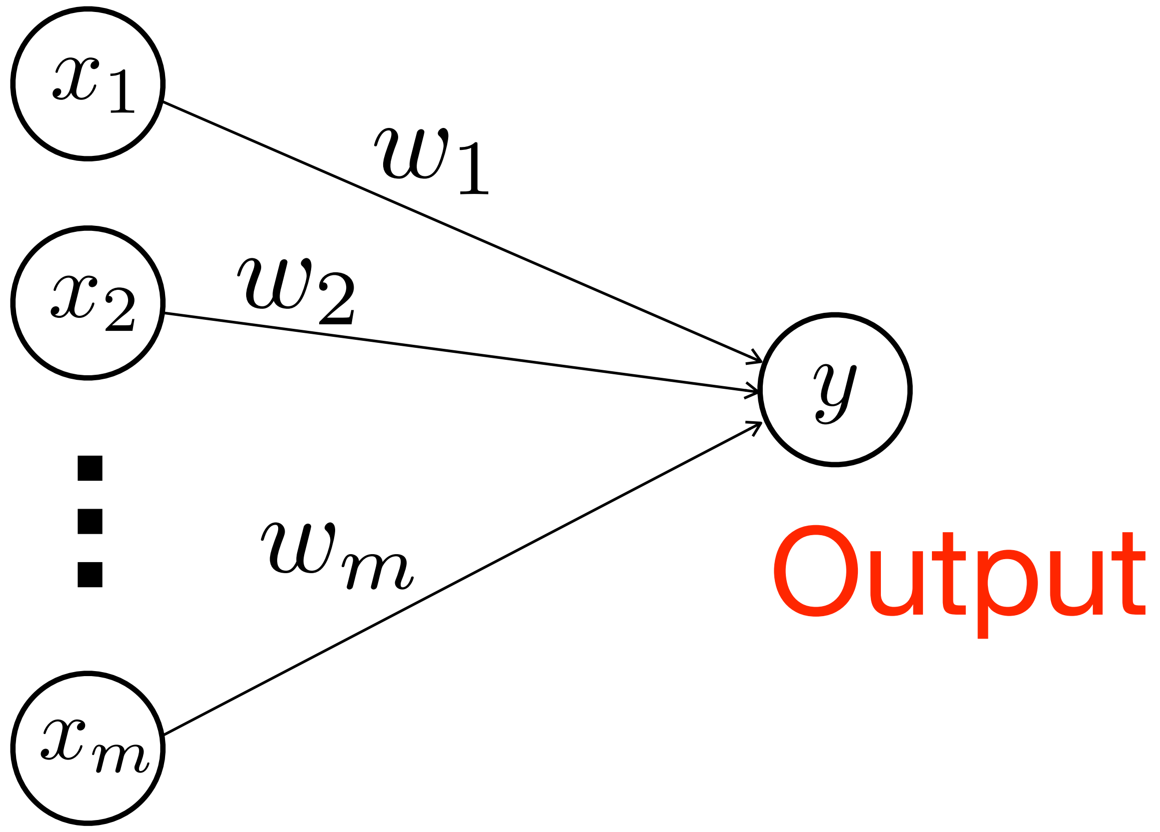
- Recall the figure in the introduction

Raw input:
"... finest Chardonnay I've tasted"

Convert to an input vector

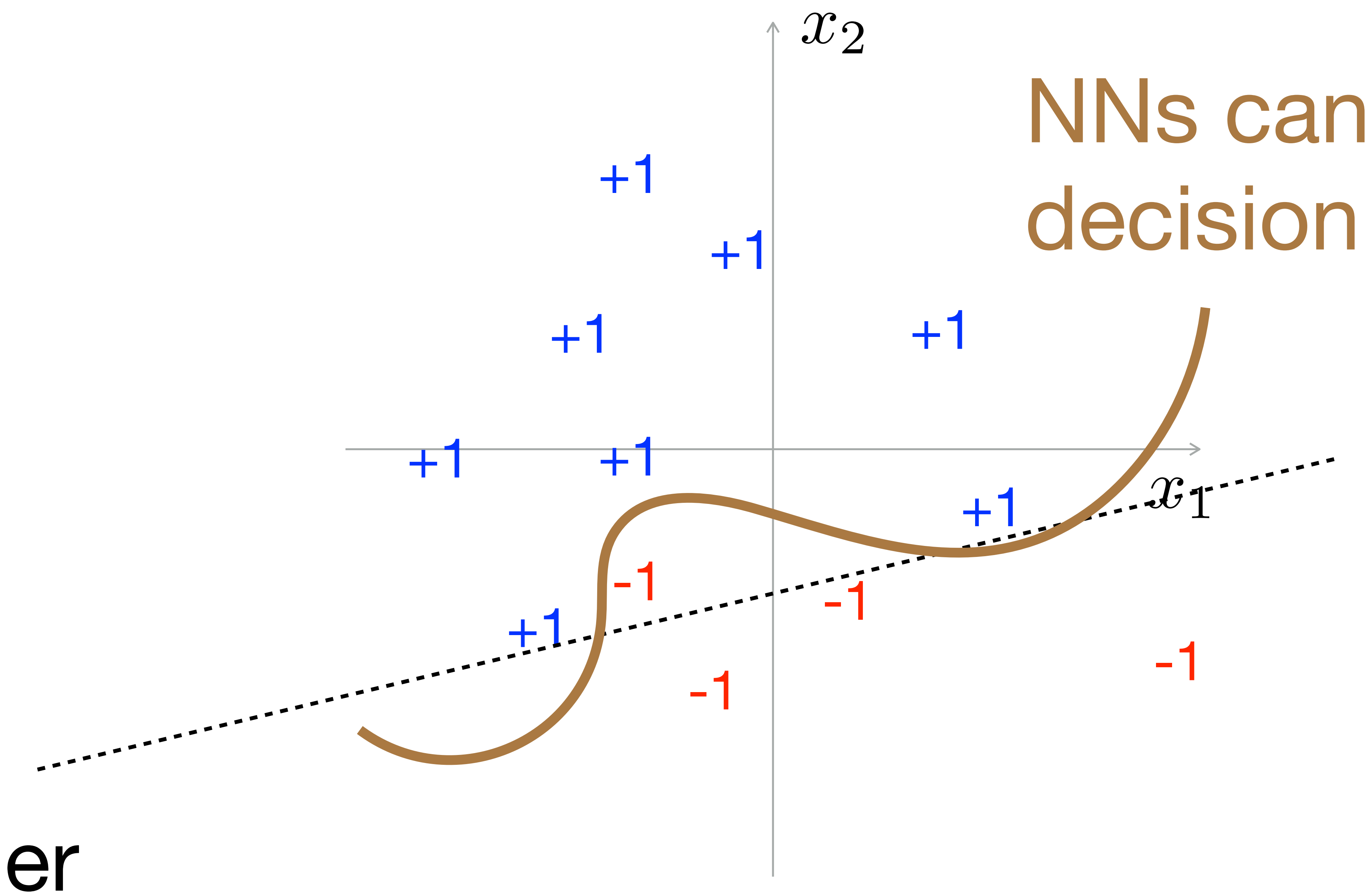


- Perceptron looks like:



- The difference is the existence of **hidden vector**

NNs can do non-linear classification

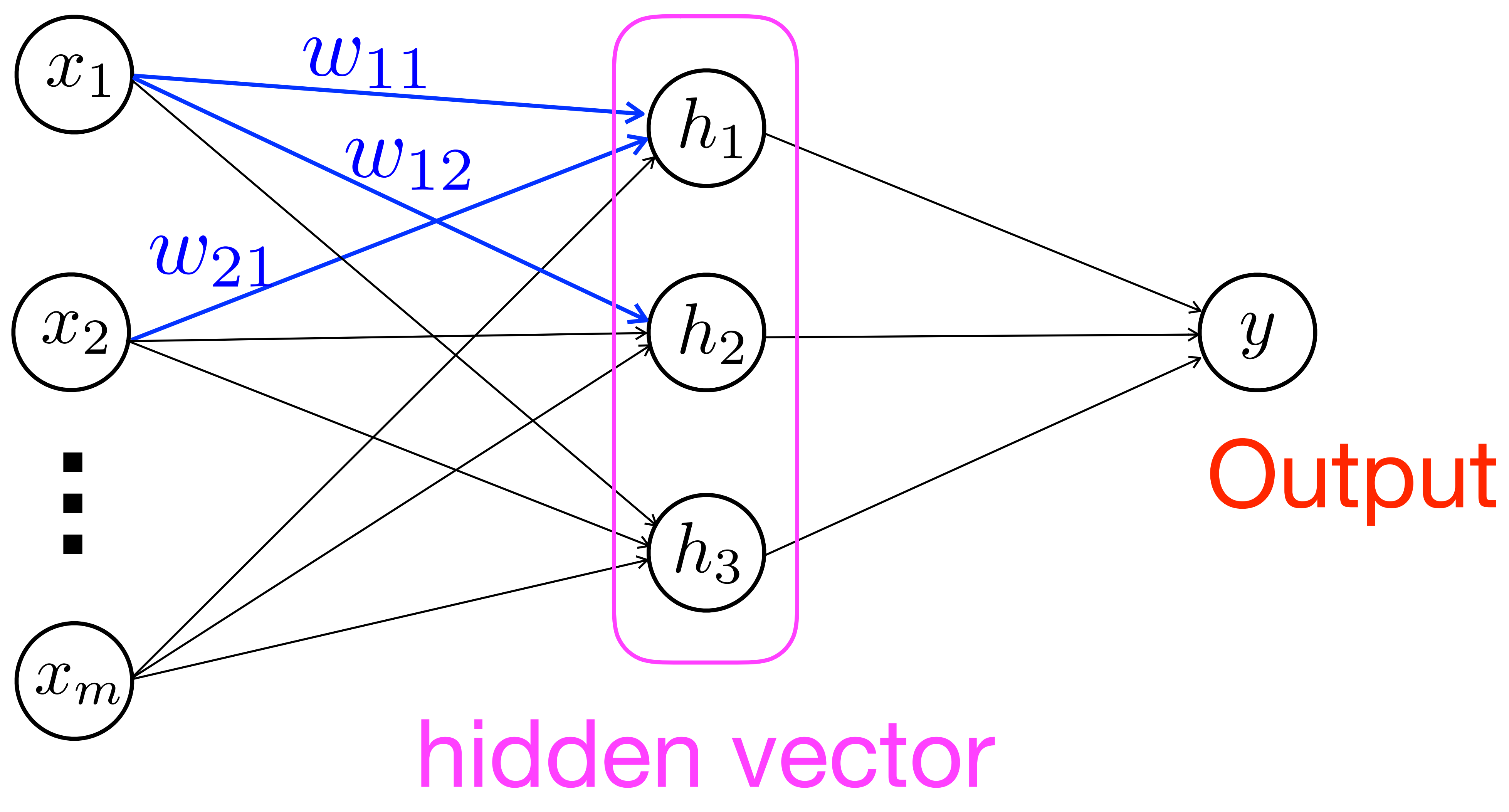


NNs can find a non-linear decision boundary

Linear classifier fails to separate

Not linearly separable

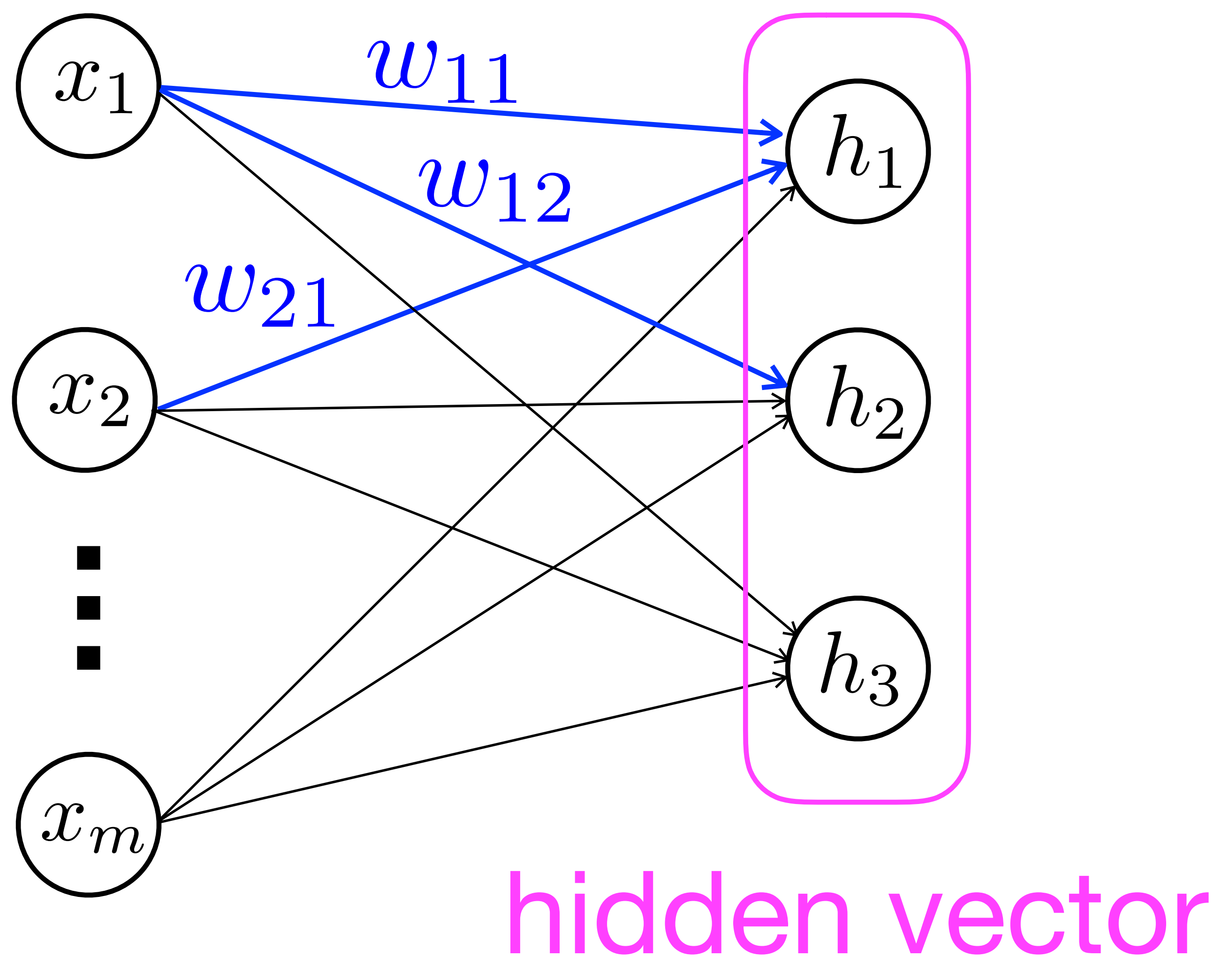
2-layer neural networks



- h_i is obtained in two steps:
  - $a_i = \sum_j w_{ji} x_j$
  - $h_i = g(a_i)$
  - g is some non-linear function (later), called **(non-linear) activity function**
- the last layer $h_i \rightarrow y$ looks like perceptron

Using matrices

$$\mathbf{W} = \begin{pmatrix} w_{11} & w_{21} & \cdots & w_{m1} \\ w_{12} & w_{22} & \cdots & w_{m2} \\ w_{13} & w_{23} & \cdots & w_{m3} \end{pmatrix}$$

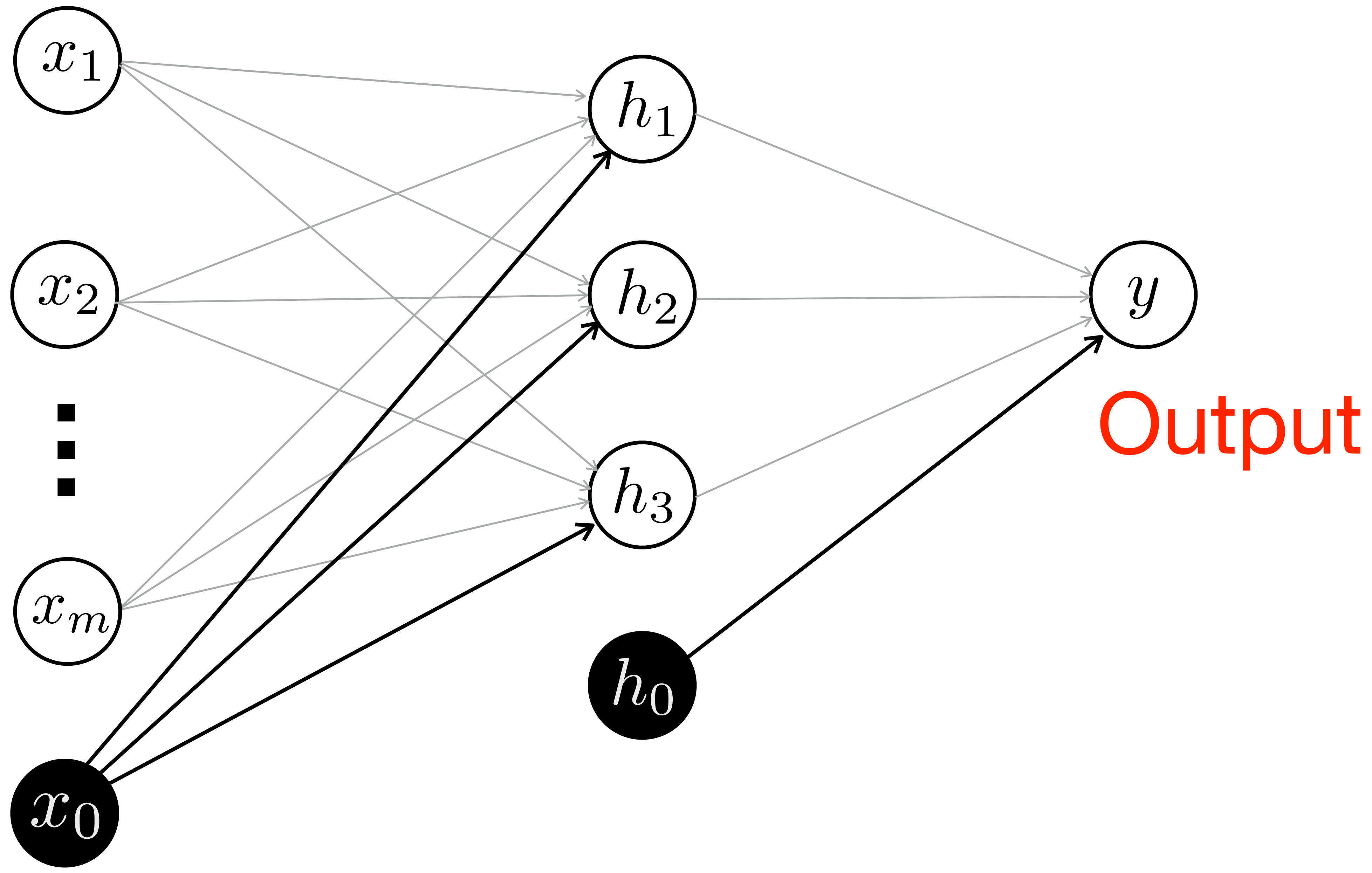


$$\mathbf{h} = \begin{pmatrix} h_1 \\ h_2 \\ h_3 \end{pmatrix}$$

$$\mathbf{h} = g(\mathbf{W}\mathbf{x})$$

- g is applied for each component
- With GPUs, matrix calculation is very fast

Handling of the bias term

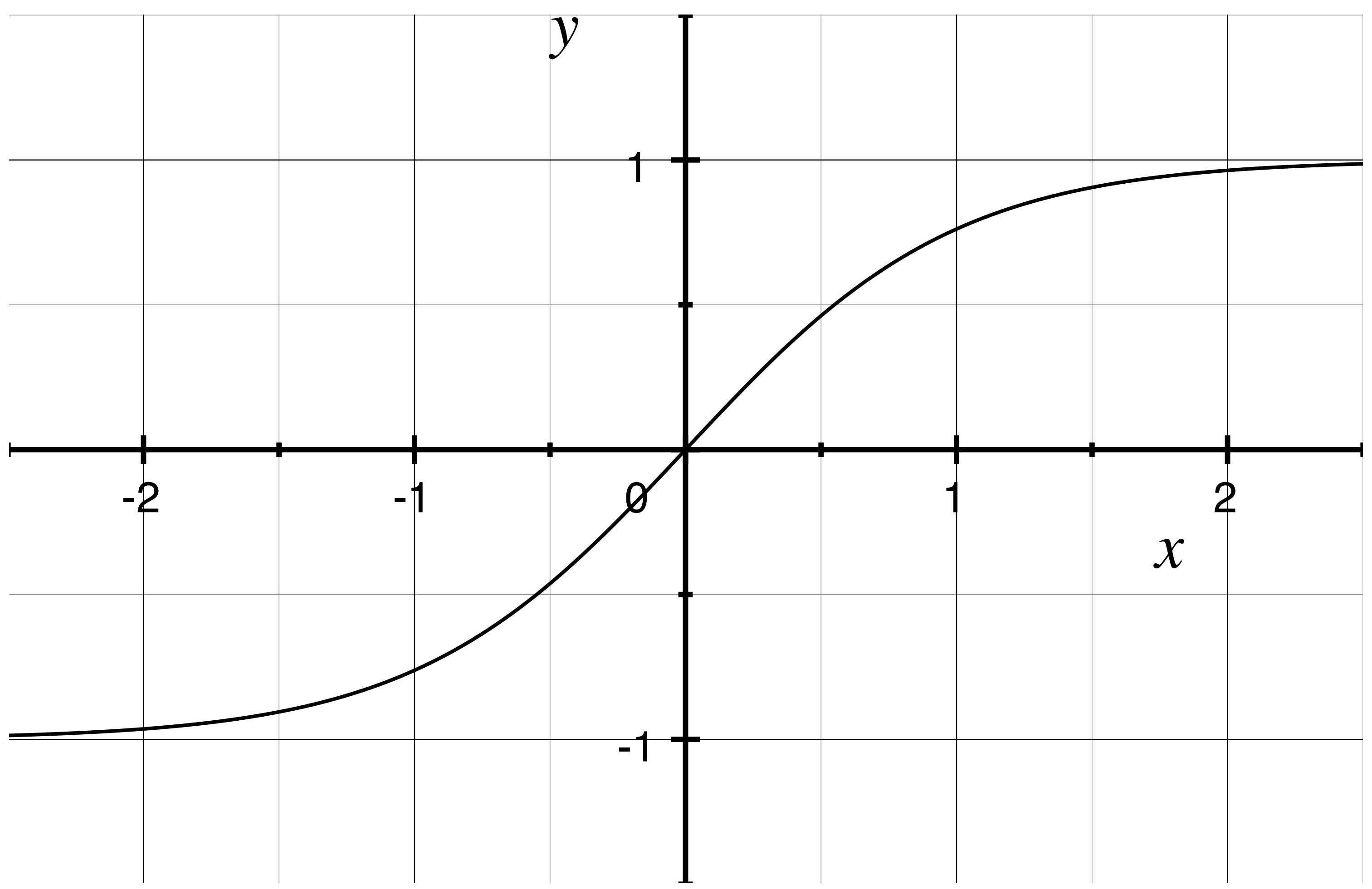


- We have omitted the bias term but that should be considered
  - We can think that the input at each layer contains a bias term, which is always **1** (the weights are updated during training)
- We consider in this way in the following

Non-linear activation function

$$\mathbf{h} = g(\mathbf{W}\mathbf{x})$$

- Activation function (g) converts the weighted input with a non-linear function
  - Essential for the higher expressive power of NNs



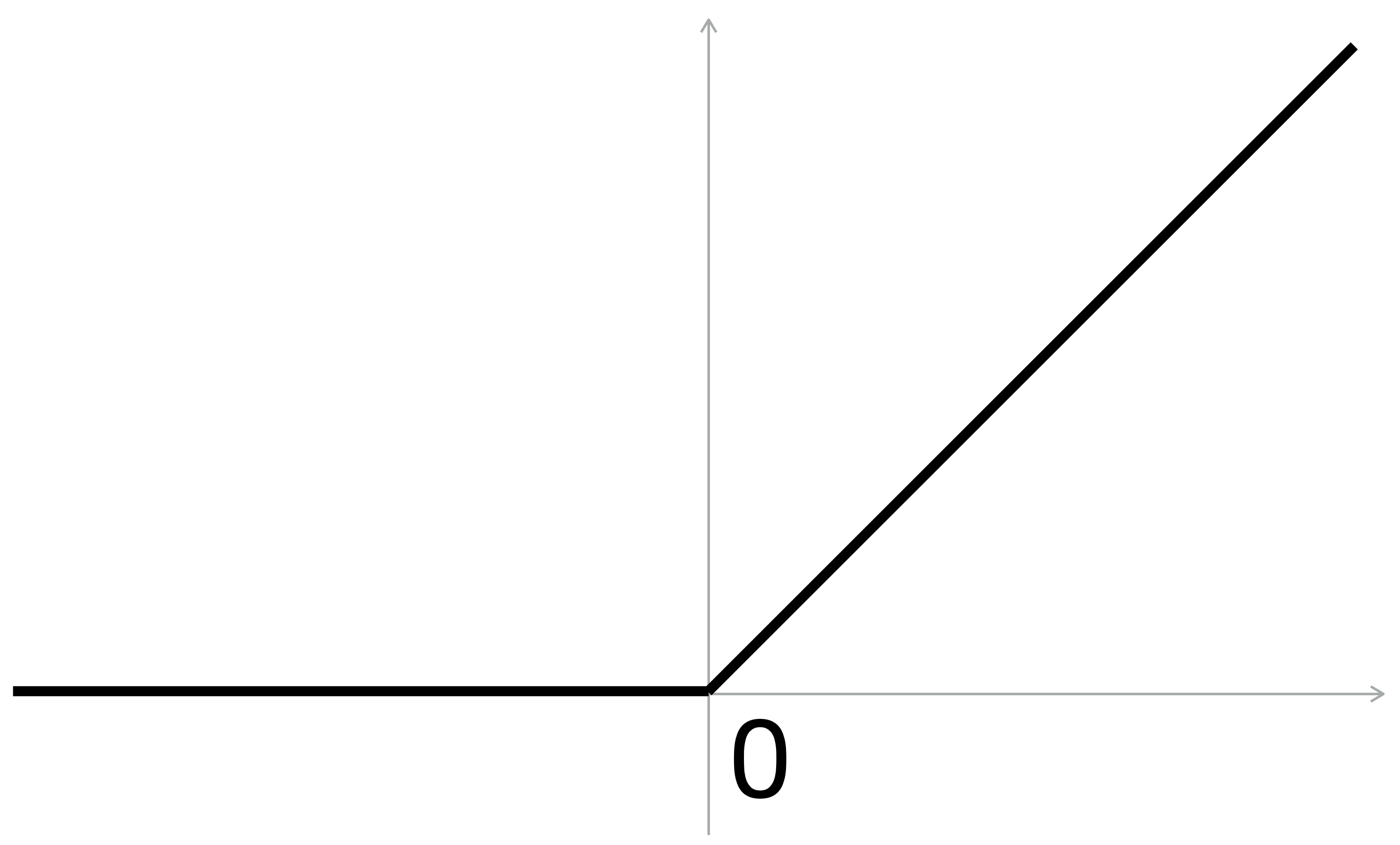
$$y = \tanh(x)$$

- Several non-linear functions can be applied
  - but importantly, the function should be differentiable
  - needed for training with back-prop (next week)
- One popular choice is **tanh** non-linearity

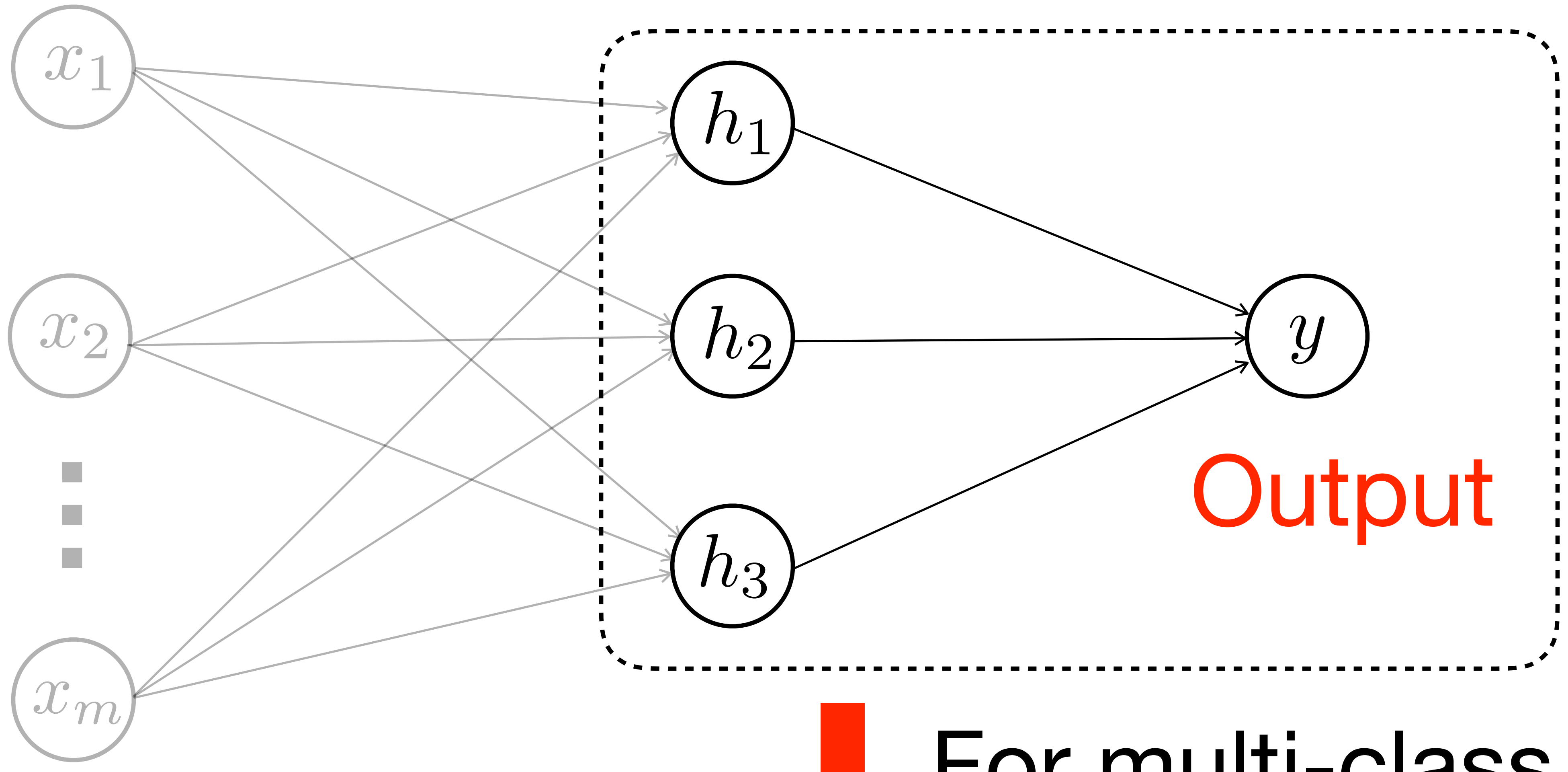
ReLU: Rectified linear unit

- A simpler non-linear activation function, called ReLU, is getting popular
  - Often performs better than other activation functions

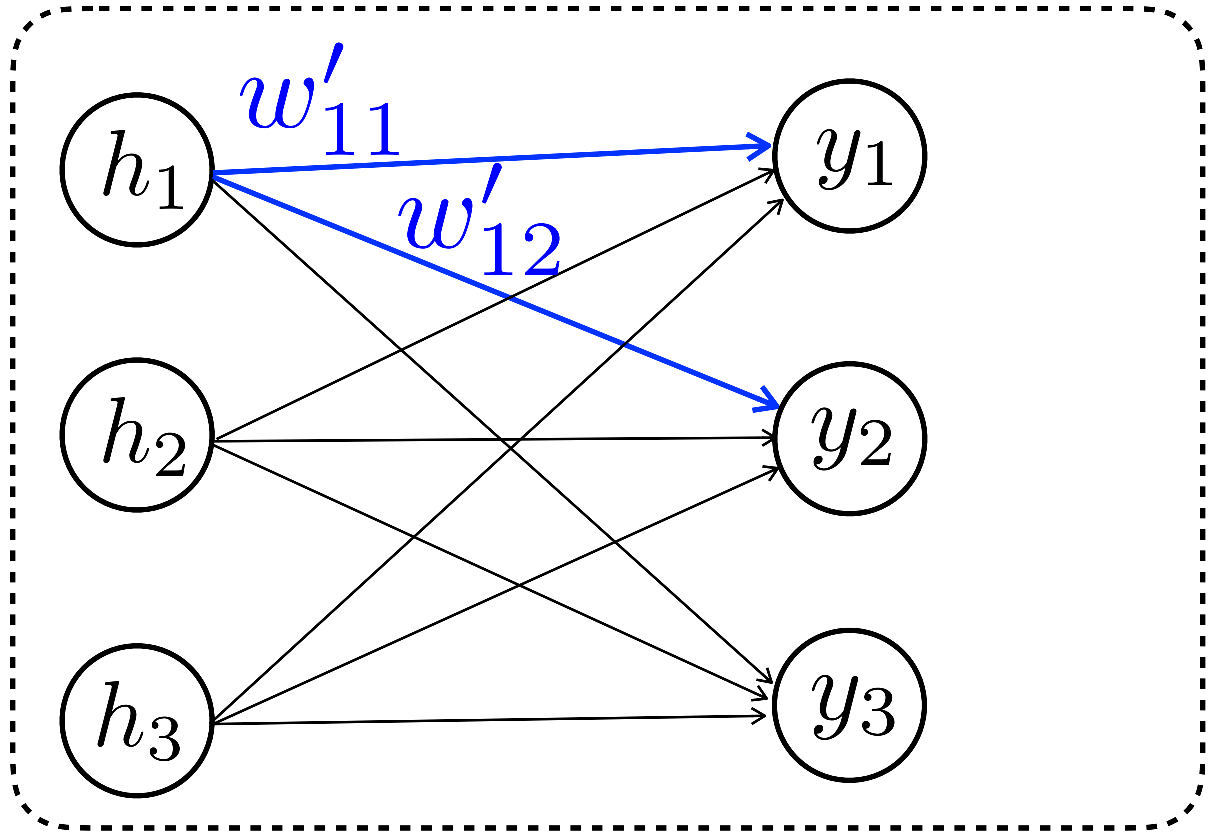
$$\mathrm{ReLU}(x) = \begin{cases} x & \text{if } x > 0; \\ 0 & \text{otherwise.} \end{cases}$$



Output layer



For multi-class classification, this actually looks like:



$$y_i = \sum_j w'_{ji} h_i$$

- Output is i-th label with the highest y_i
- Similarly to $\mathbf{h}$, $\mathbf{y}$ can be written as: $\mathbf{y} = \mathbf{W}'\mathbf{h}$

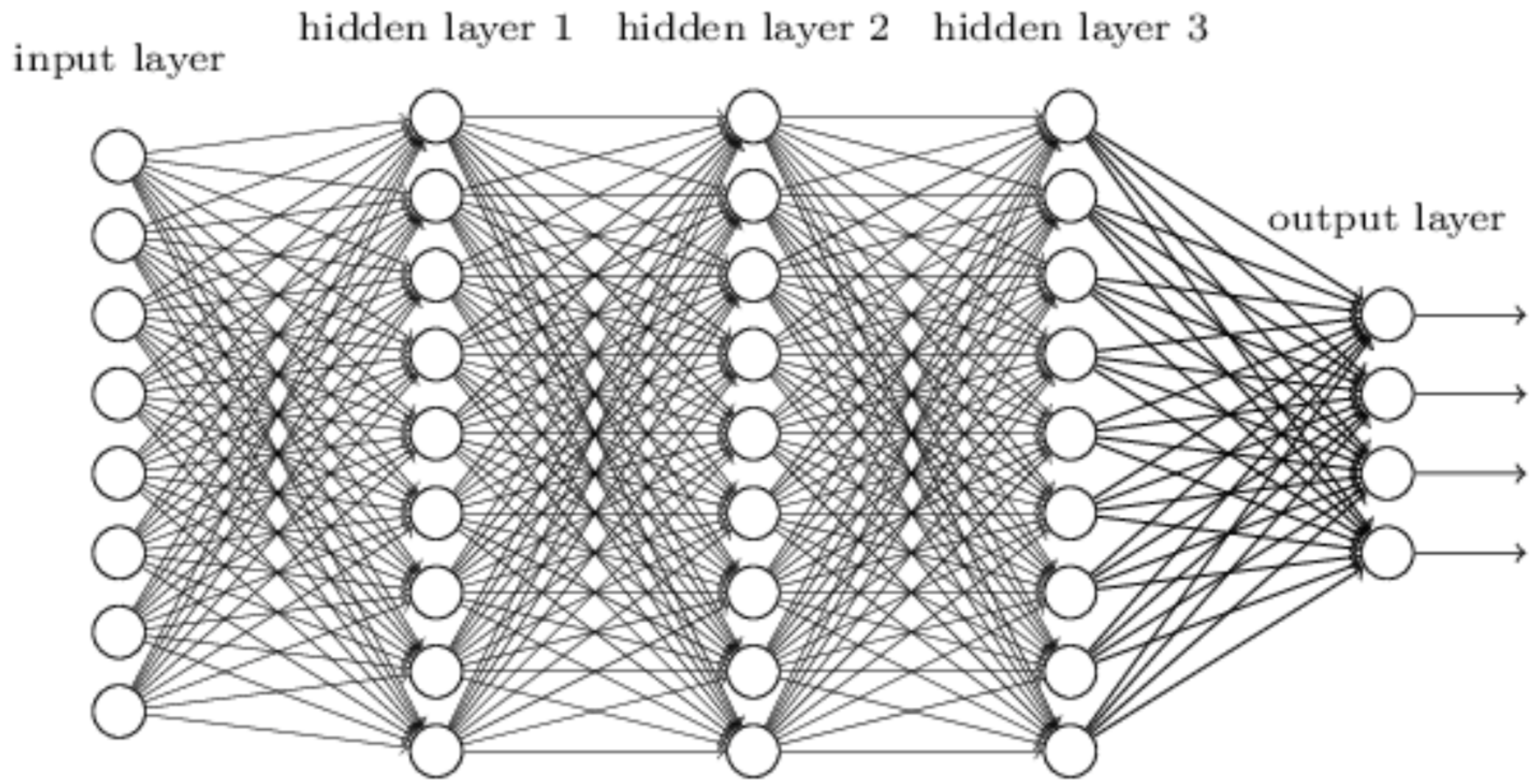
Softmax

- Often the output $\mathbf{y}$ is transformed with a function
  - A popular transform is **softmax**:

$$\mathrm{softmax}(y_i) = \frac{\exp(y_i)}{\sum_j \exp(y_j)}$$

  - $\exp$ is often used in machine learning to make the value non-negative
  - By this, all outputs are in [0, 1]; and the sum is 1.0
    - We can interpret the value as a probability to choose i
    - This is nice when we want to train the parameters (for using some loss-function, such as cross-entropy; next week)

What is **deep** learning?



- Adding more and more layers to the neural network
  - The expressive power rapidly increases with the number of hidden layers
  - But learning becomes much more challenging
  - Learning (deep) neural networks is the main topic of the next week

Summary

- Machine learning
  - The problem of learning a mapping from an input to the output using the labeled training data
- Perceptron
  - A simplest form of neural networks for classification
  - Limitation: cannot perform non-linear classification
- Deep Neural networks
  - Hidden layers introduce non-linearity, which allows non-linear classification
  - Training becomes much harder ⇒ main topic of the next week