

# Neural networks

COMS 4771 Fall 2023

**Feature maps revisited**

Justification for simple statistical models (e.g., logistic regression):

- ▶ They are reasonable with a judicious choice of features or feature map
- ▶ In linear models, best prediction of  $Y$  given  $X = x$  is based entirely on

$$w^\top \varphi(x)$$

where  $\varphi$  is the feature map

1 / 33

Weierstrass approximation theorem: For any continuous function  $f: \mathbb{R}^d \rightarrow \mathbb{R}$ , any bounded region  $B \subset \mathbb{R}^d$ , and any  $\varepsilon > 0$ , there exists a polynomial  $g: \mathbb{R}^d \rightarrow \mathbb{R}$  such that

$$\max_{x \in B} |f(x) - g(x)| \leq \varepsilon$$

- ▶ Polynomials give good approximations uniformly over an interval (Cf. Taylor's theorem: only guarantees local approximations)
- ▶ Universal justification of polynomial expansion + linear functions
- ▶ Caveat: Degree of  $g$  may be large (e.g., growing with  $d$  and  $1/\varepsilon$ )
  - ▶ Somewhat ameliorated by kernel methods + regularization

2 / 33

Kernel machine: function learned by kernel method

$$g(x) = \sum_{i=1}^n \alpha_i k(x, x^{(i)})$$

where  $k(\cdot, \cdot)$  is the kernel function, and  $x^{(1)}, \dots, x^{(n)}$  are the training examples

3 / 33

Stone-Weierstrass approximation theorem: For any continuous function  $f: \mathbb{R}^d \rightarrow \mathbb{R}$ , any bounded region  $B \subset \mathbb{R}^d$ , and any  $\varepsilon > 0$ , there exists a function  $g: \mathbb{R}^d \rightarrow \mathbb{R}$  of the form

$$g(x) = \sum_{i=1}^p \alpha_i \exp(x^\top w^{(i)})$$

such that

$$\max_{x \in B} |f(x) - g(x)| \leq \varepsilon$$

- ▶ Can replace “exp” with other “activation functions”
- ▶ Caveat:  $p$  may be large
- ▶ Another interpretation: linear function  $\alpha^\top \varphi(x)$  with feature map

$$\varphi(x) = (\exp(x^\top w^{(1)}), \dots, \exp(x^\top w^{(p)}))$$

Except the  $w^{(i)}$ 's may need to depend on  $f$

- ▶ This kind of function is called a (two-layer) neural network

4 / 33

Kernel machine

$$g(x) = \sum_{i=1}^n \alpha_i \kappa(x, x^{(i)})$$

- ▶ Only  $\alpha_i$ 's are learned using data

(Two-layer) neural network

$$g(x) = \sum_{i=1}^p \alpha_i \exp(x^\top w^{(i)})$$

- ▶ Both  $\alpha_i$ 's and  $w^{(i)}$ 's are learned
- ▶ Can use  $p > n$

**Neural networks as straight-line programs**

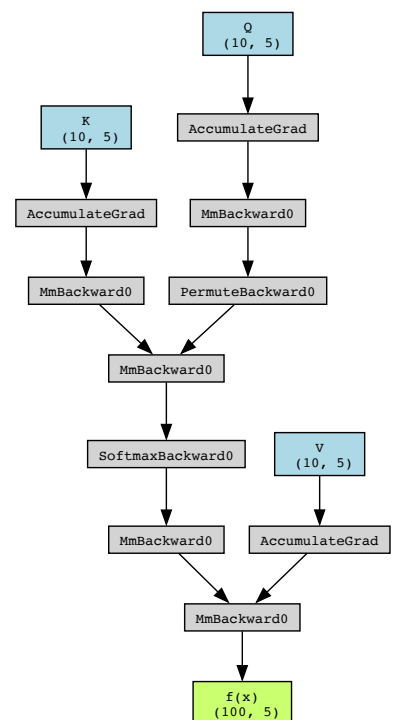
## Very abbreviated history:

- ▶ McCulloch and Pitts (early 1940s):  
Neural networks as computational model for brain
- ▶ Arnold and Kolmogorov (late 1950s):  
Solved Hilbert's 13th problem (about polynomial roots) using neural networks
- ▶ **Modern use of neural networks with Linnainmaa's autodiff (early 1970s) started with Werbos (early 1980s)**
- ▶ Many other researchers have since discovered other approximation-theoretic properties and practical uses of neural networks (e.g., Cybenko, Rumelhart and Hinton, LeCun)

Today, for machine learning purposes: a [neural network](#) is any function  $f$  such that  $f(x)$  can be computed by a straight-line program

```
K = torch.randn(d, p, requires_grad=True)
Q = torch.randn(d, p, requires_grad=True)
V = torch.randn(d, p, requires_grad=True)
```

```
def f(x):
    k = x @ K
    q = x @ Q
    a = torch.softmax(k @ q.T, dim=1)
    return a @ x @ V
```



### Example:

$$f(x) = \alpha_0 + \sum_{i=1}^p \alpha_i \sigma(x^\top w^{(i)} + b^{(i)})$$

$$v_1 := \underline{\hspace{10em}}$$

$$v_2 := \underline{\hspace{10em}}$$

⋮

$$v_p := \underline{\hspace{10em}}$$

$$\hat{y} := \alpha_0 + \alpha_1 \times v_1 + \alpha_2 \times v_2 + \cdots + \alpha_p \times v_p$$

- ▶  $v_1, \dots, v_p$  called [hidden units](#) (antiquated terminology)
- ▶ Using modern numerical software (e.g., pytorch):

$$\hat{y} := \alpha_0 + \alpha^\top \sigma(Wx + b)$$

( $W \in \mathbb{R}^{p \times d}$ ,  $b, \alpha \in \mathbb{R}^p$ ,  $\alpha_0 \in \mathbb{R}$ , and  $\sigma: \mathbb{R} \rightarrow \mathbb{R}$  is applied component-wise)

8 / 33

In practice, neural network “architectures” (i.e., program “templates”) are built using/composing component modules

Simplest module is [fully-connected layer](#):

$$h \mapsto \sigma(Wh + b)$$

(affine transformation followed by non-linear transformation)

Some examples of  $\sigma$ :

- ▶ [Rectified linear unit](#):  $\text{relu}(t) = [t]_+ = \max\{0, t\}$
- ▶ [Hyperbolic tangent](#):  $\text{tanh}(t) = 2 \text{logistic}(t) - 1$
- ▶ [Softmax](#):  $\text{softmax}: \mathbb{R}^k \rightarrow \mathbb{R}^k$ , where  $\text{softmax}(u)_i = \frac{\exp(u_i)}{\sum_{j=1}^k \exp(u_j)}$

9 / 33

## Training neural networks

**Problem:** How to fit neural network  $f$  (with parameters  $\theta$ ) to training data?

- ▶ A few more lines in straight-line program gives

$$J := \sum_{i=1}^n \text{loss}(f(x^{(i)}), y^{(i)})$$

```
loss = torch.nn.NLLLoss(reduction='sum')  
J = loss(f(x), y)
```

- ▶ So autodiff can compute gradient of  $J$  with respect to all parameters  $\theta$
- ▶ This enables use of gradient-based optimization algorithms!

**Major challenge:** objective function  $J(\theta)$  might not be convex, so use of gradient-based optimization is more complicated (e.g., initialization, step sizes)

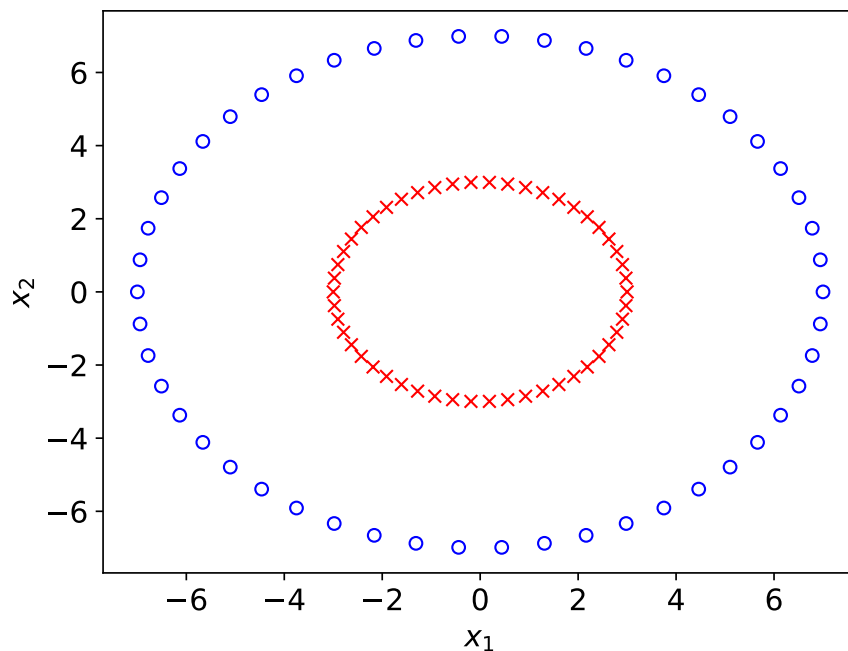
- ▶ Many tips and tricks (e.g., “Efficient BackProp”, LeCun et al, 1998)

- ▶ Experimentation may still be required

## Synthetic example



Data: classes are two concentric circles, 50 examples per class



12 / 33

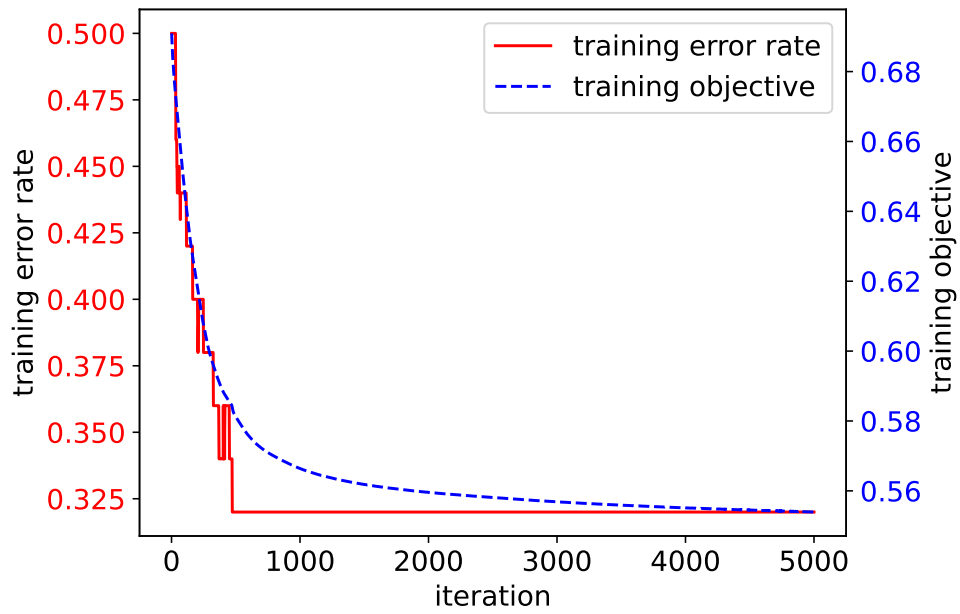
- ▶ Feature transformation: standardization
- ▶ Neural net:  $f(x) = \text{softmax}(A \text{relu}(Wx + b) + c)$ 
  - ▶ Parameters:  $W \in \mathbb{R}^{p \times 2}$ ,  $b \in \mathbb{R}^p$ ,  $A \in \mathbb{R}^{2 \times p}$ ,  $c \in \mathbb{R}^2$   
(We will vary the “width”  $p$ )
  - ▶  $k$ -th output is prediction of  $\Pr(Y = k \mid X = x)$
- ▶ Use gradient descent on average logarithmic loss on training data
  - ▶ Random initialization:

$$W_{i,j}, b_i \stackrel{\text{i.i.d.}}{\sim} \text{N}(0, \frac{1}{3}), \quad A_{i,j}, c_i \stackrel{\text{i.i.d.}}{\sim} \text{N}(0, \frac{2}{p+1})$$

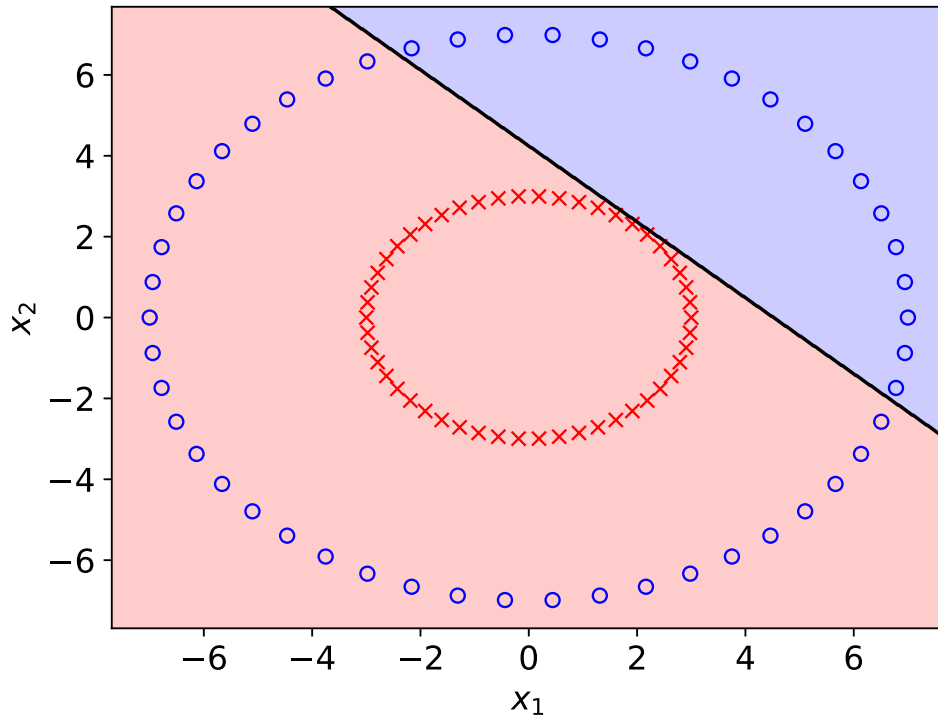
- ▶ Step size:  $\eta_t = 0.1$

13 / 33

Results:  $p = 2$



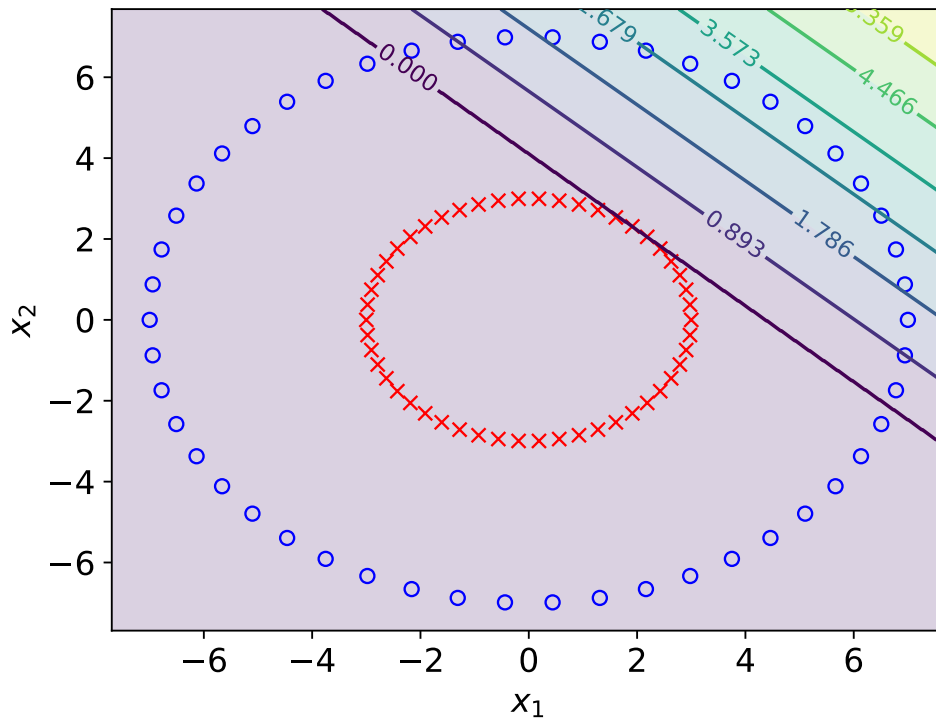
Results:  $p = 2$



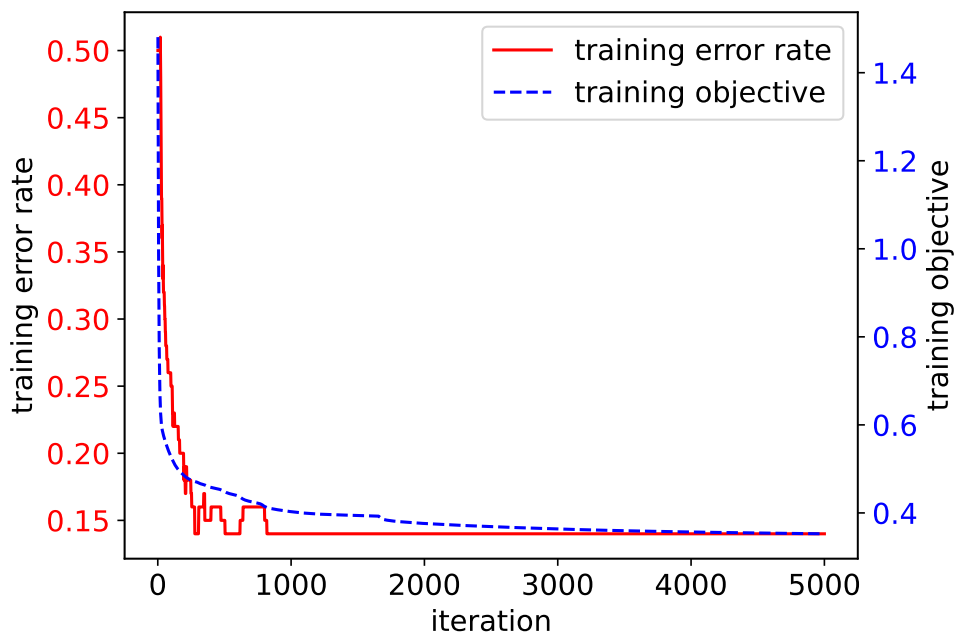
Results:  $p = 2$

- ▶ Behaves like a linear classifier
- ▶ First component of  $\text{relu}(Wx + b)$  is constant (0) over training data
- ▶ Only second component of  $\text{relu}(Wx + b)$  varies over training data

Results:  $p = 2$  — second component of  $\text{relu}(Wx + b)$

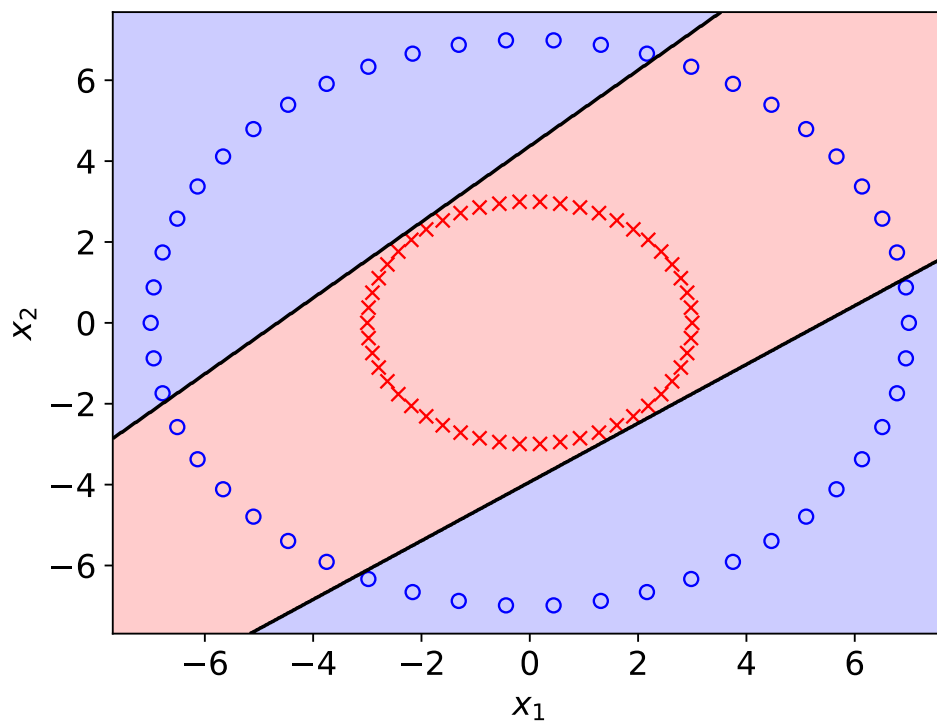


Results:  $p = 2$  (different initialization)



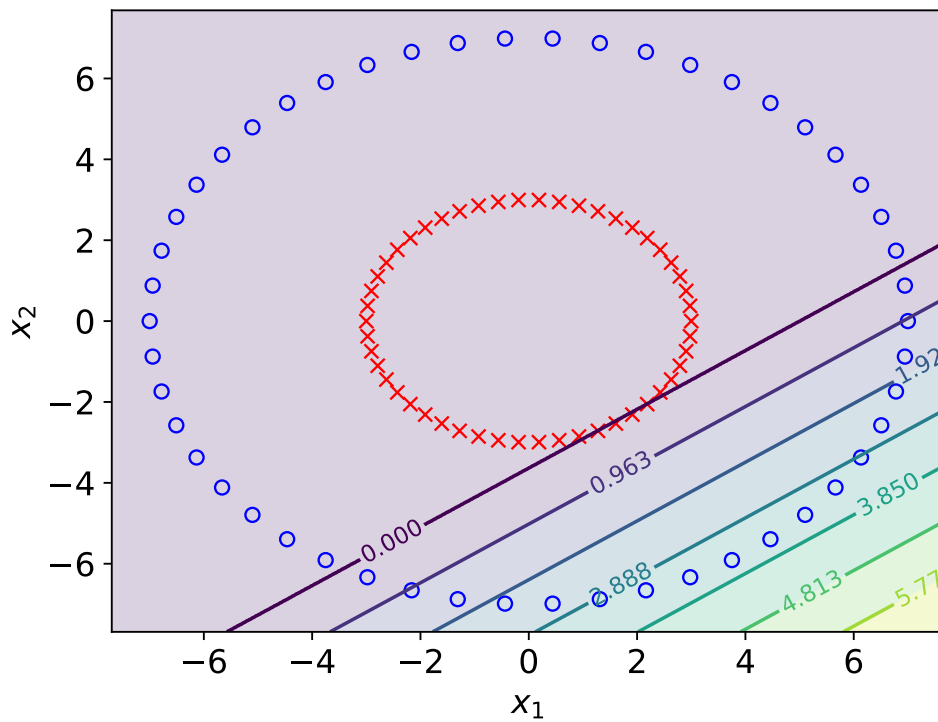
18 / 33

Results:  $p = 2$  (different initialization)

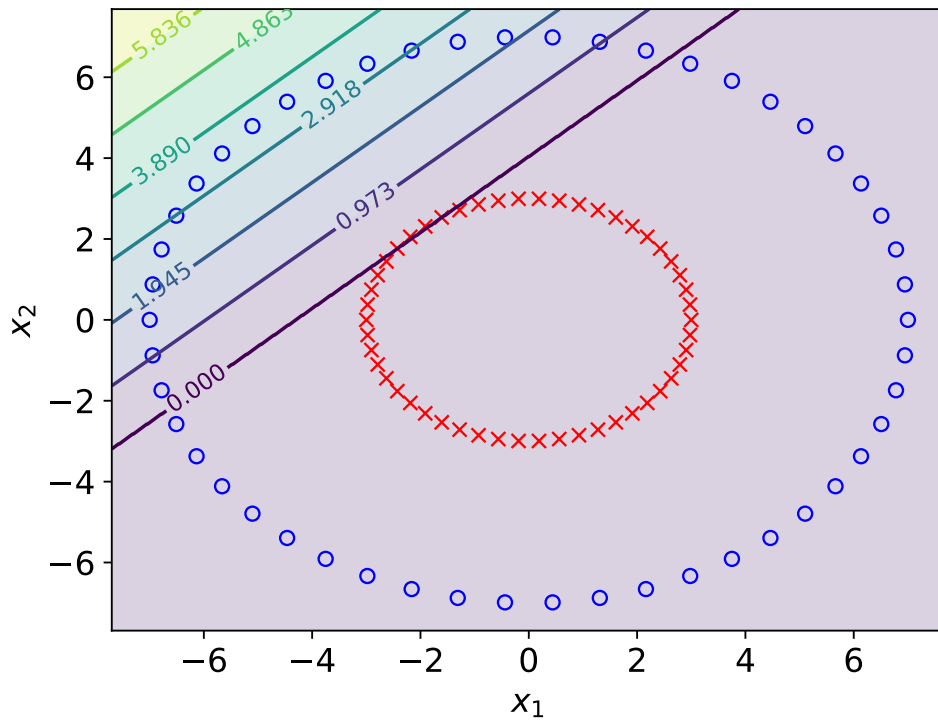


19 / 33

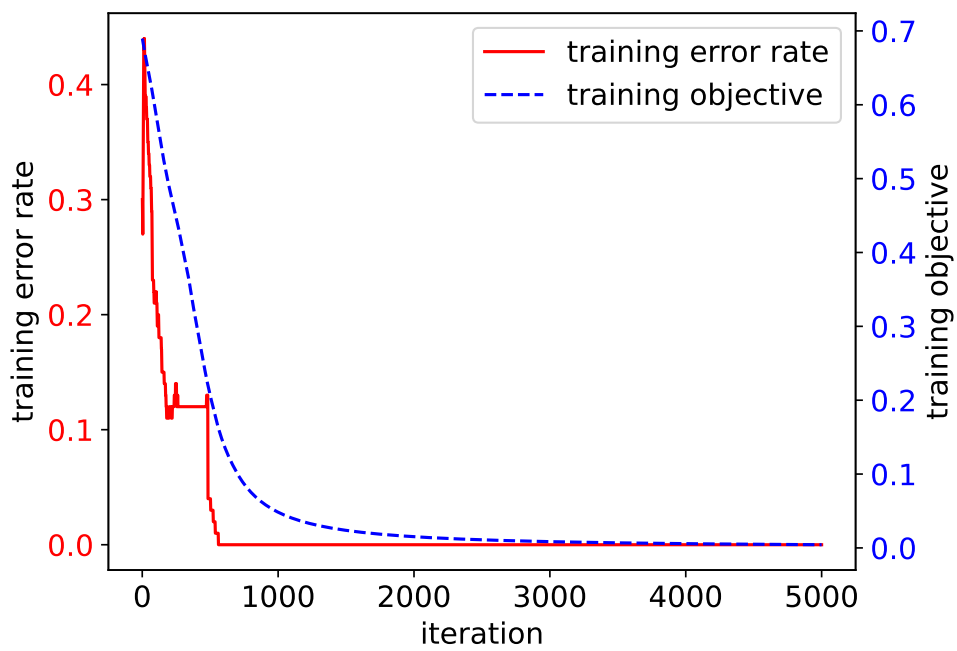
Results:  $p = 2$  (different initialization) — first component of  $\text{relu}(Wx + b)$



Results:  $p = 2$  (different initialization) — second component of  $\text{relu}(Wx + b)$

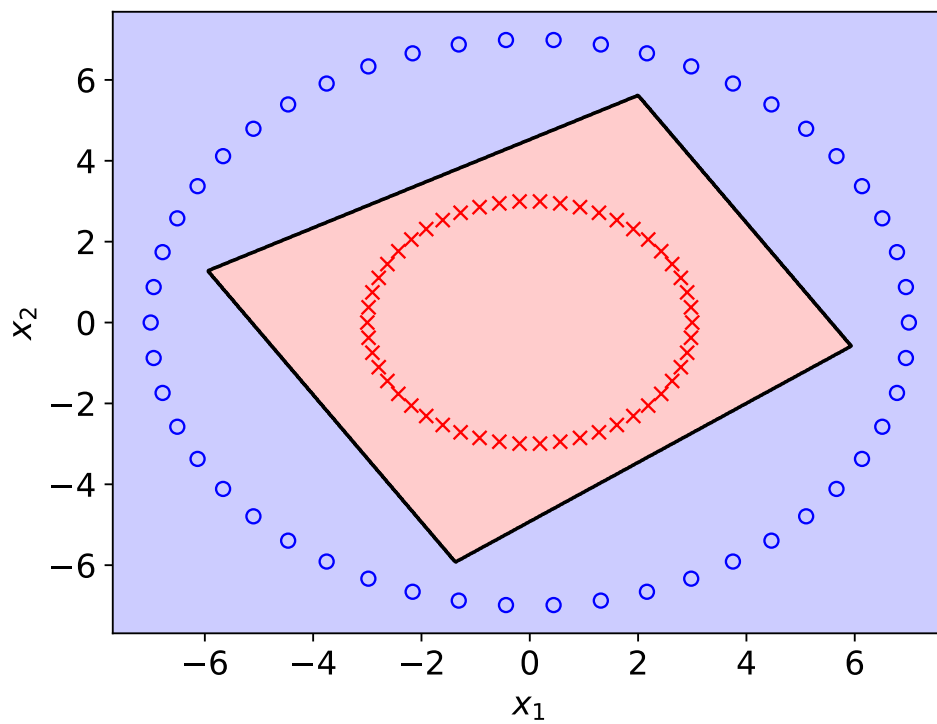


Results:  $p = 3$



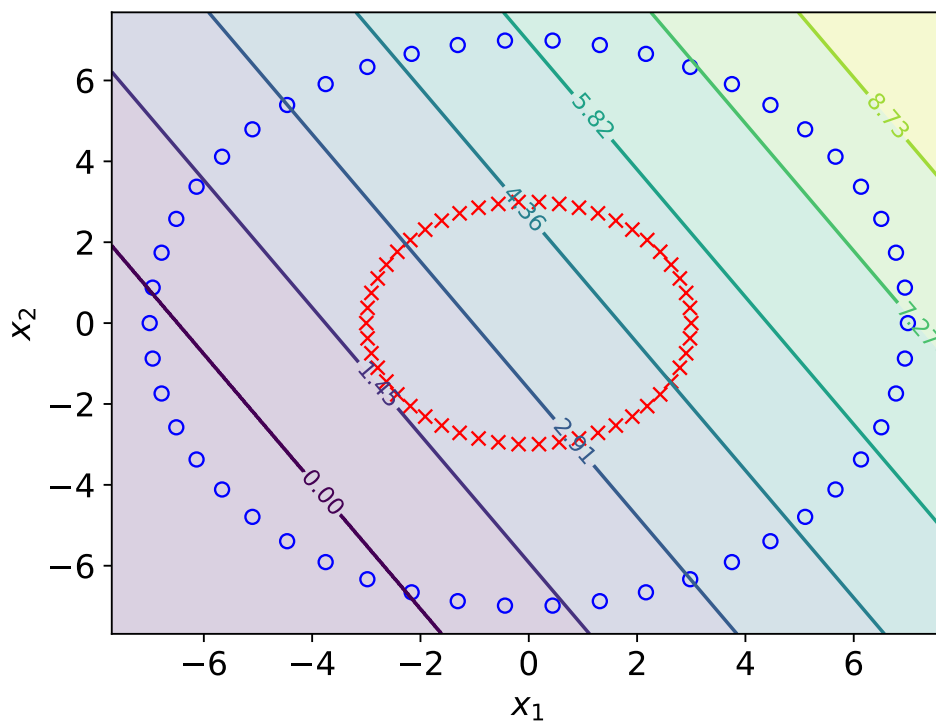
22 / 33

Results:  $p = 3$



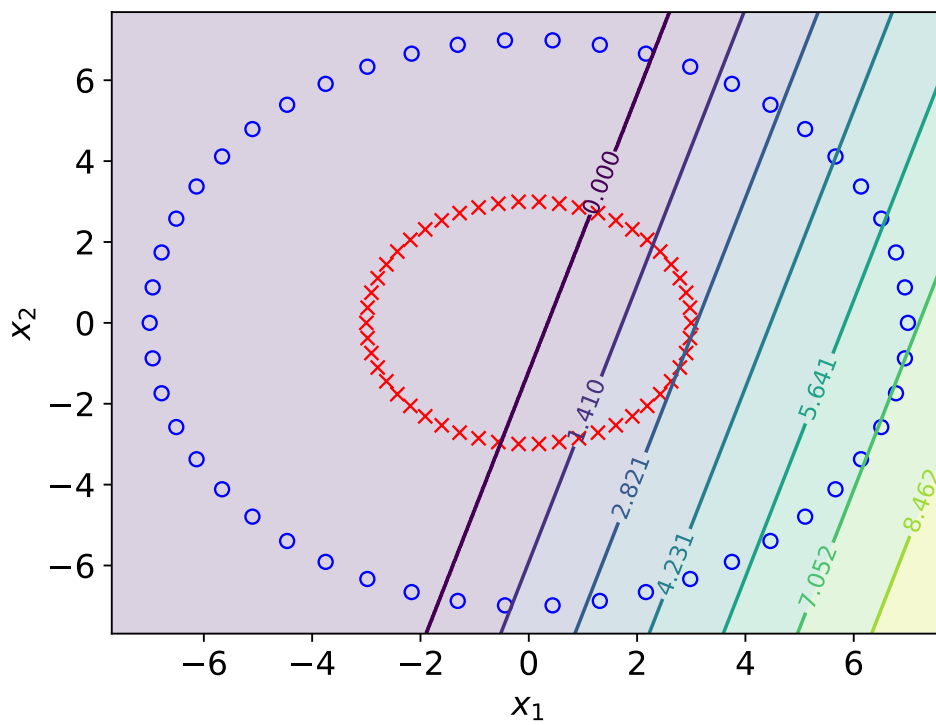
23 / 33

Results:  $p = 3$  — first component of  $\text{relu}(Wx + b)$



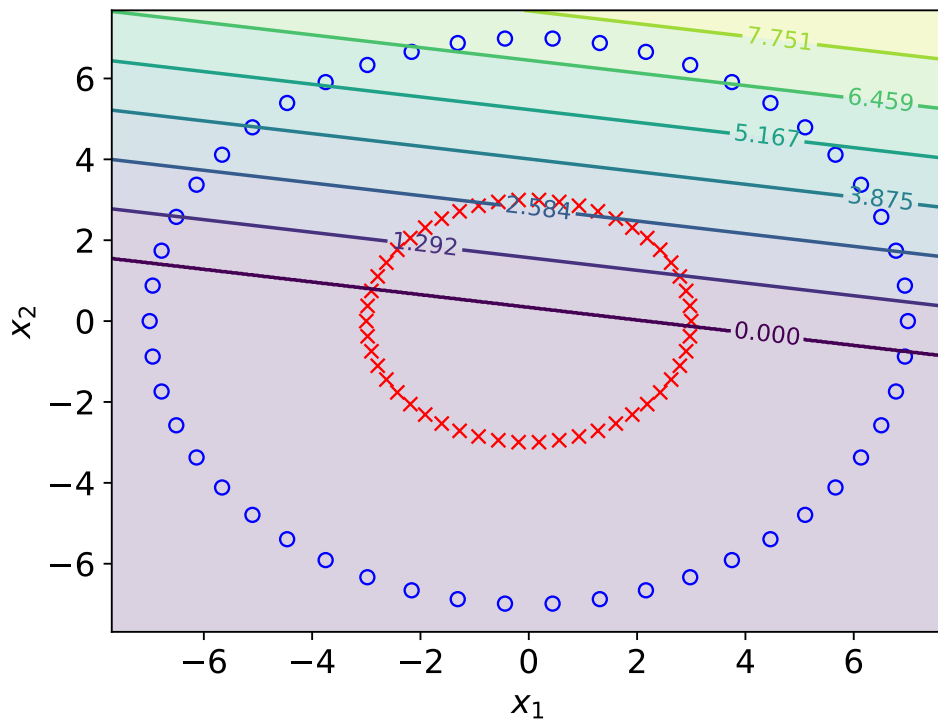
24 / 33

Results:  $p = 3$  — second component of  $\text{relu}(Wx + b)$

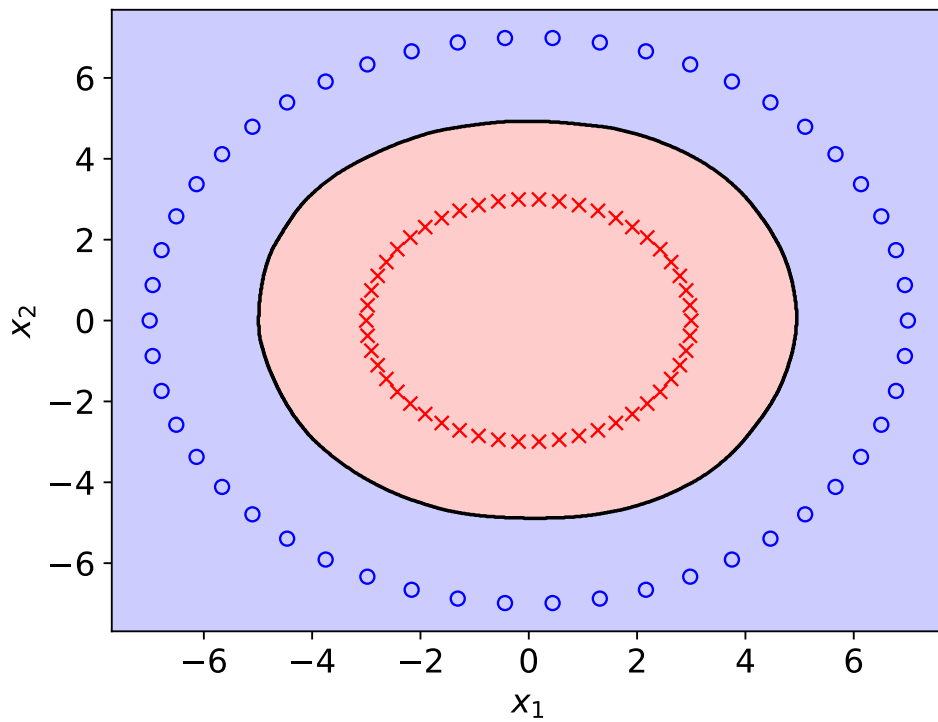


25 / 33

Results:  $p = 3$  — third component of  $\text{relu}(Wx + b)$



Results:  $p = 1000$





## Iris data classifier

- ▶ Features:

$$x_1 = \text{sepal width} / \text{sepal length}, \quad x_2 = \text{petal width} / \text{petal length}$$

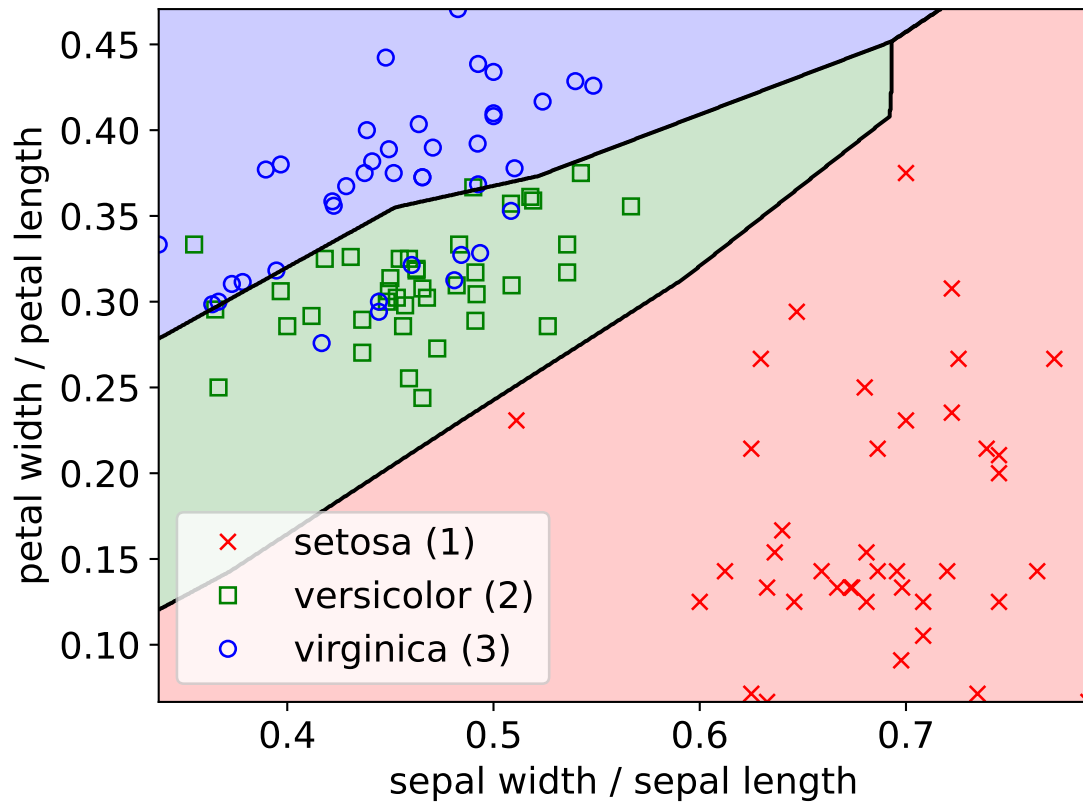
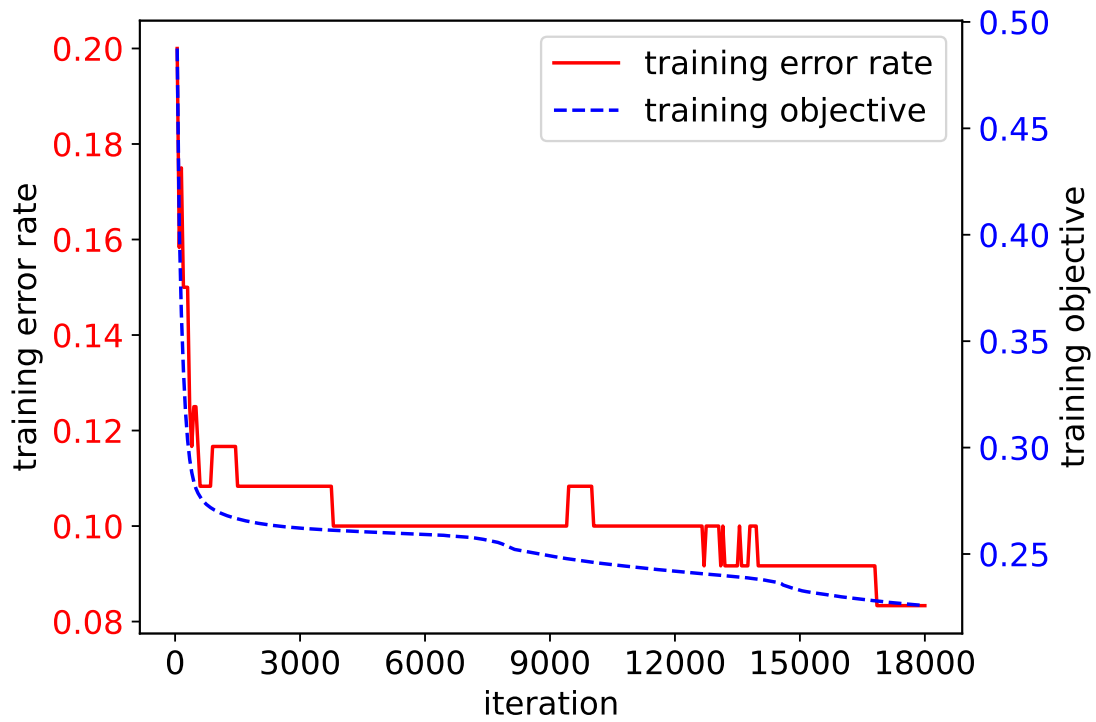
- ▶ Neural net:  $f(x) = \text{softmax}(A \text{relu}(Wx + b) + c)$

- ▶ Parameters:  $W \in \mathbb{R}^{10 \times 2}$ ,  $b \in \mathbb{R}^{10}$ ,  $A \in \mathbb{R}^{2 \times 10}$ ,  $c \in \mathbb{R}^2$

- ▶  $k$ -th output is prediction of  $\Pr(Y = k | X = x)$

- ▶ Feature transformation and training procedure: same as in synthetic example

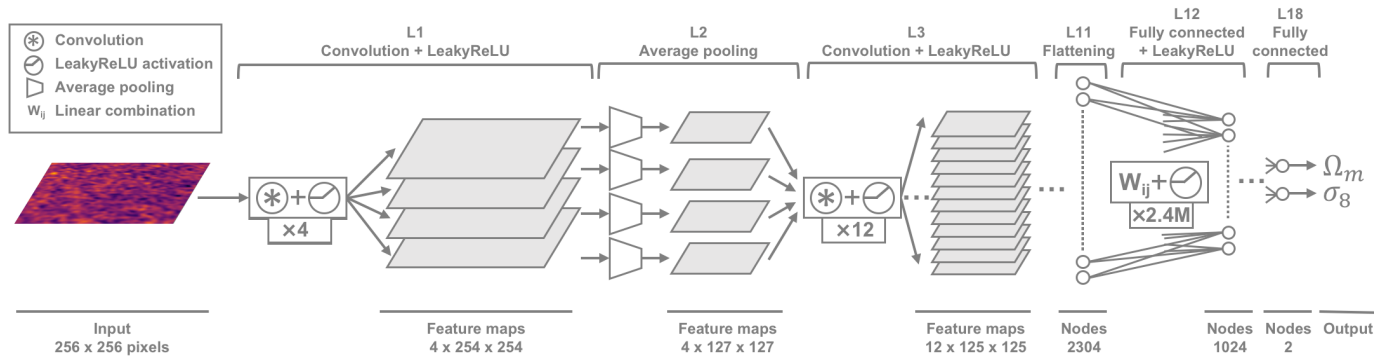
- ▶ Training error rate: 8.33%, test error rate: 10.0%



## Deep learning lifestyle

- ▶ Since 2012, use of neural networks++ has exploded in machine learning
- ▶ Called “deep learning” due to use of large and “deep” neural networks
- ▶ Key factors in latest resurgence and success:
  - ▶ Graphics processing units (GPUs) to speed-up matrix operations
  - ▶ Easy-to-use numerical software with autodiff (e.g., pytorch)
  - ▶ Large benchmark datasets (e.g., ImageNet)

Practice largely guided by heuristics and extensive experimentation



Many different architectural components, e.g.:

- ▶ Convolutional layer (in convolutional neural networks)
- ▶ Attention module (in transformer networks)