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

$$w^{\mathsf{T}}\varphi(x)$$

  where φ is the feature map

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

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

- 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

Kernel machine: function learned by kernel method

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

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

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

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

such that

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

- Can replace "$\exp$" with other "activation functions"
- Caveat: p may be large
- Another interpretation: linear function $\alpha^\mathsf{T} \varphi(x)$ with feature map

$$\varphi(x) = (\exp(x^\mathsf{T} w^{(1)}), \ldots, \exp(x^\mathsf{T} w^{(p)}))$$

  Except the $w^{(i)}$'s may need to depend on f
- This kind of function is called a (two-layer) neural network

Kernel machine

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

- Only α_i's are learned using data

(Two-layer) neural network

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

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

Neural networks as straight-line programs

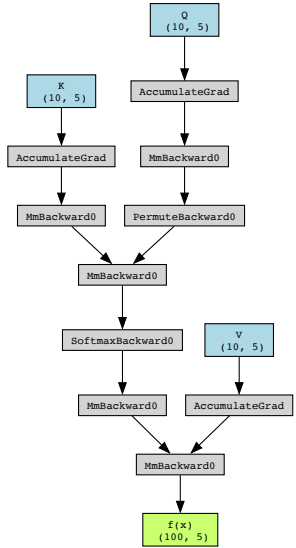
Very abbreviated history:

- 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

```python
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:

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

$$\begin{aligned} v_1 &:= \underline{\qquad\qquad} \\ v_2 &:= \underline{\qquad\qquad} \\ &\vdots \\ v_p &:= \underline{\qquad\qquad} \\ \hat{y} &:= \alpha_0 + \alpha_1 \times v_1 + \alpha_2 \times v_2 + \cdots + \alpha_p \times v_p \end{aligned}$$

- $v_1, \ldots, v_p$ called hidden units (antiquated terminology)
- Using modern numerical software (e.g., pytorch):

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

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

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 σ:

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

Training neural networks

Problem: How to fit neural network f (with parameters θ) 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 θ
- This enables use of gradient-based optimization algorithms!

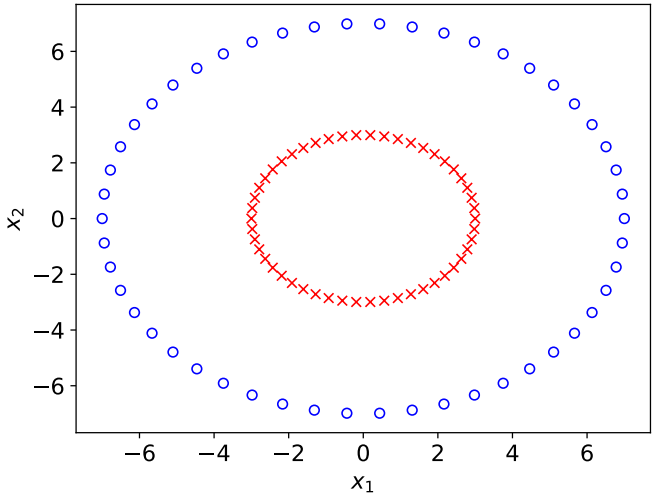
Major challenge: objective function $J(\theta)$ might not be convex, so use of gradient-based optimization is more complicated (e.g., initialization, step sizes)

- 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

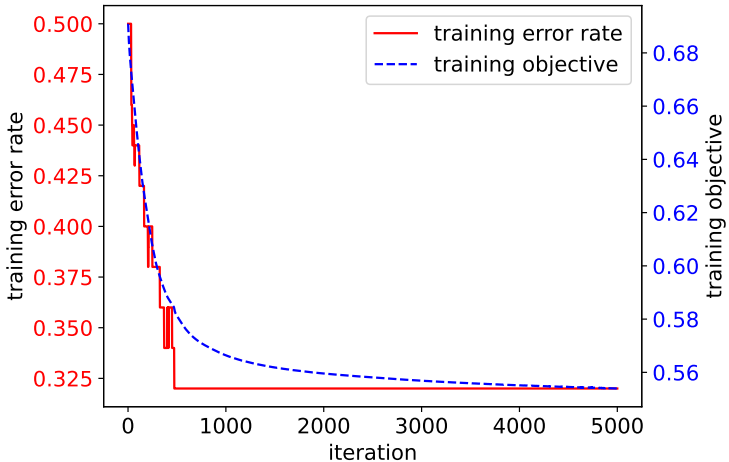


- Feature transformation: standardization
- Neural net: $f(x) = \mathrm{softmax}(A\,\mathrm{relu}(Wx + b) + c)$
  - Parameters: $W \in \mathbb{R}^{p\times 2}$, $b \in \mathbb{R}^p$, $A \in \mathbb{R}^{2\times p}$, $c \in \mathbb{R}^2$
    (We will vary the "width" p)
  - k-th output is prediction of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

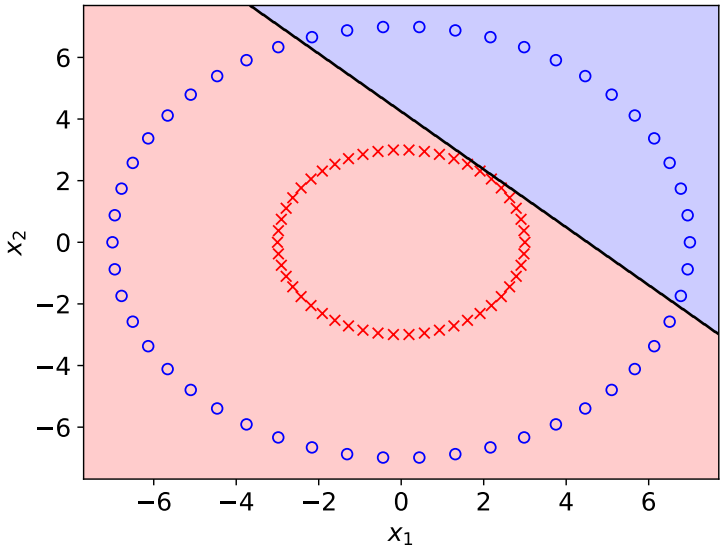
$$W_{i,j}, b_i \overset{\text{i.i.d.}}{\sim} \mathrm{N}(0, \tfrac{1}{3}), \quad A_{i,j}, c_i \overset{\text{i.i.d.}}{\sim} \mathrm{N}(0, \tfrac{2}{p+1})$$

  - Step size: $\eta_t = 0.1$

Results: $p = 2$


training error rate
training objective
0.500
0.475
0.450
0.425
0.400
0.375
0.350
0.325
training error rate
0.68
0.66
0.64
0.62
0.60
0.58
0.56
training objective
0
1000
2000
3000
4000
5000
iteration

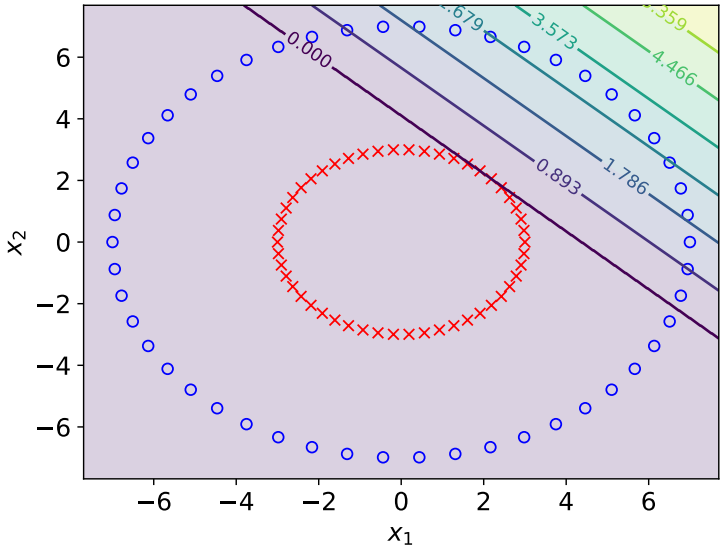
Results: $p = 2$



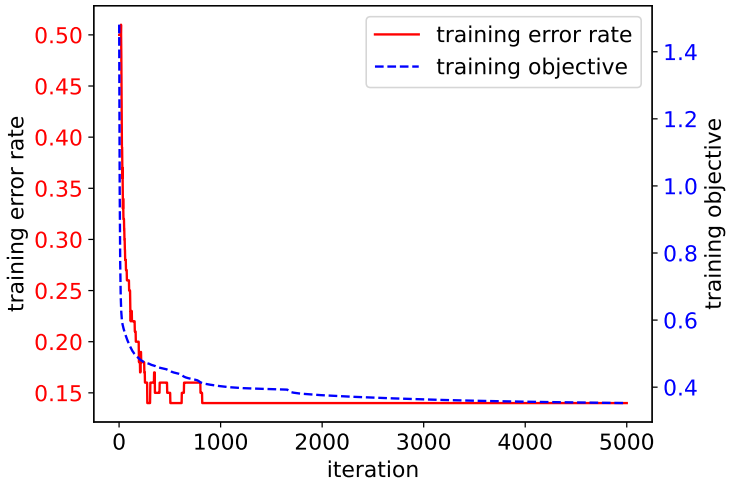
Results: $p = 2$

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

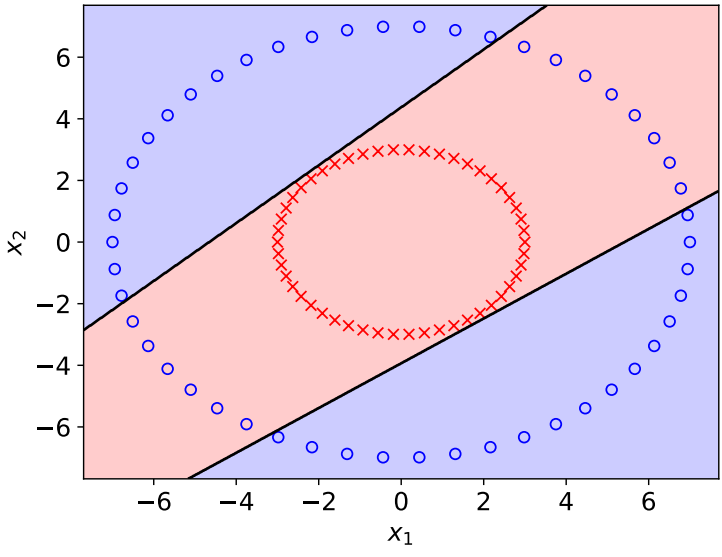
Results: $p = 2$ — second component of $\mathrm{relu}(Wx + b)$



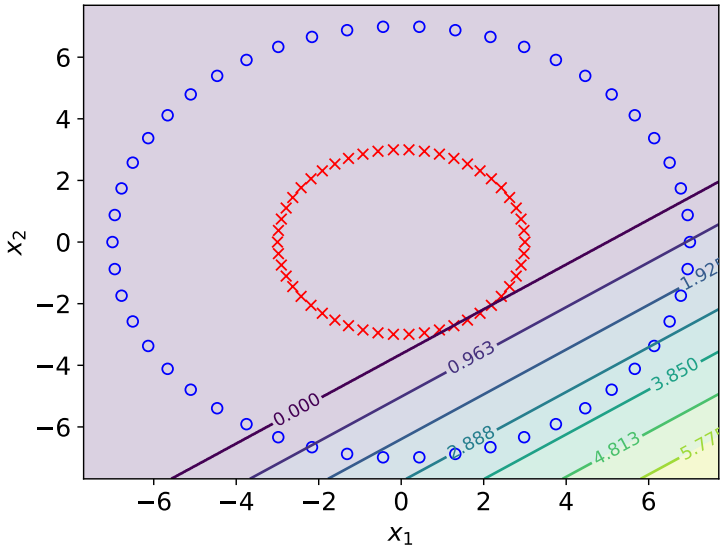
Results: $p = 2$ (different initialization)



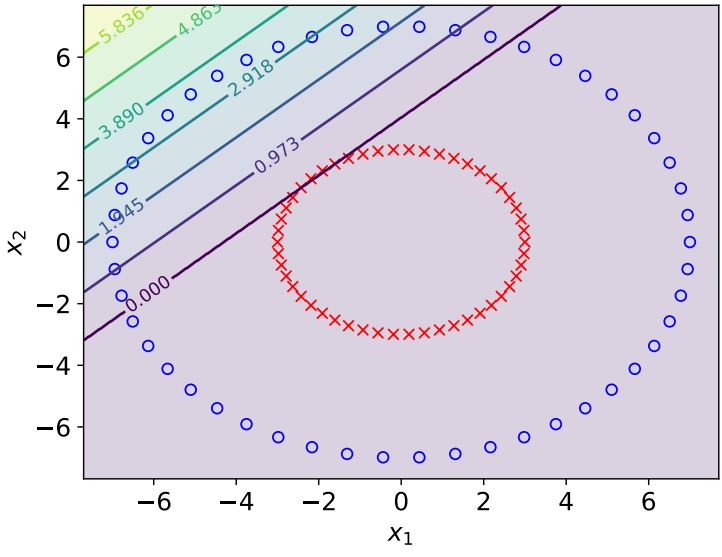
Results: $p = 2$ (different initialization)



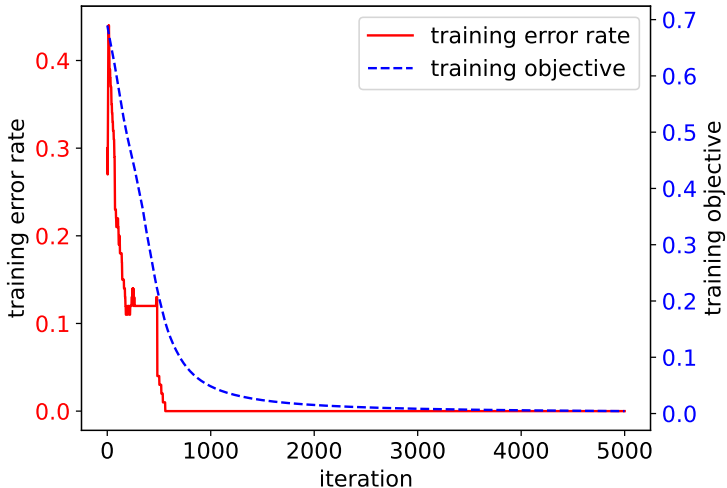
Results: $p = 2$ (different initialization) — first component of $\mathrm{relu}(Wx + b)$



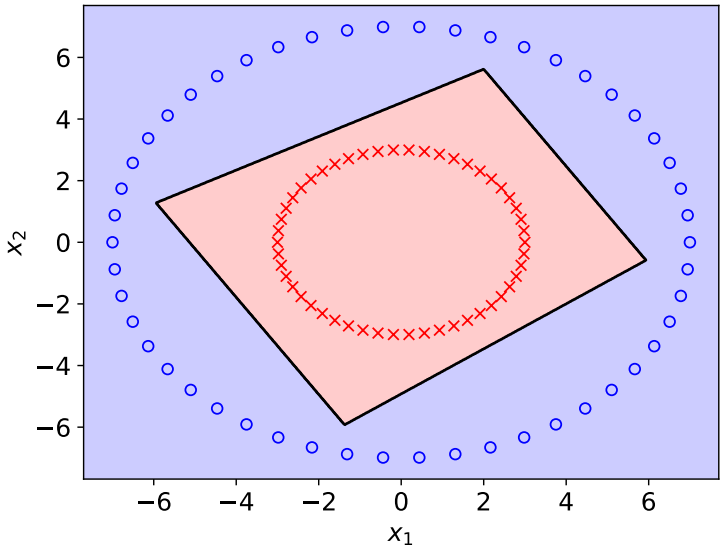
Results: $p = 2$ (different initialization) — second component of $\mathrm{relu}(Wx + b)$



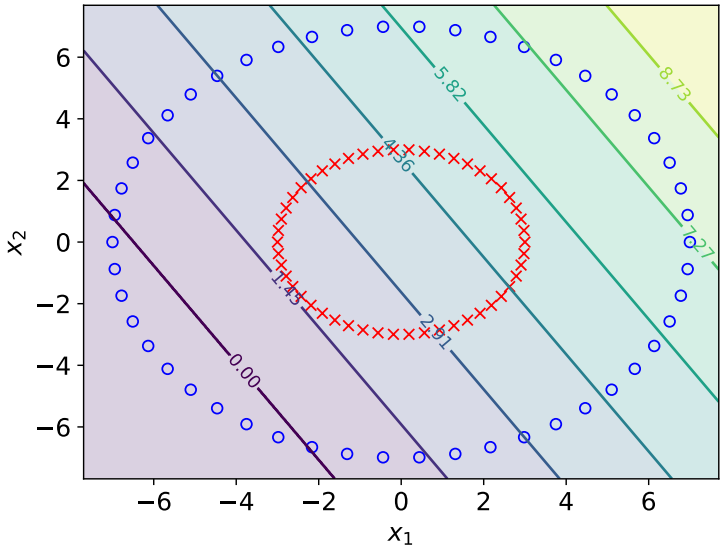
Results: $p = 3$


training error rate
training objective
training error rate
training objective
iteration
0.0
0.1
0.2
0.3
0.4
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0
1000
2000
3000
4000
5000

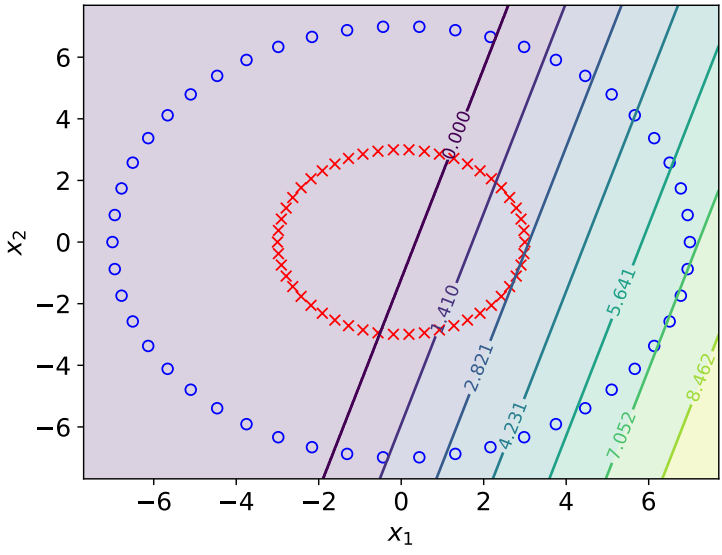
Results: $p = 3$



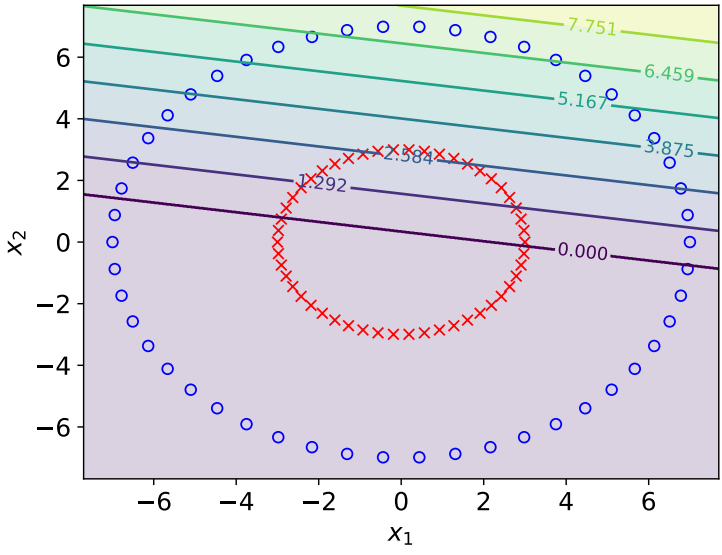
Results: $p = 3$ — first component of $\mathrm{relu}(Wx + b)$



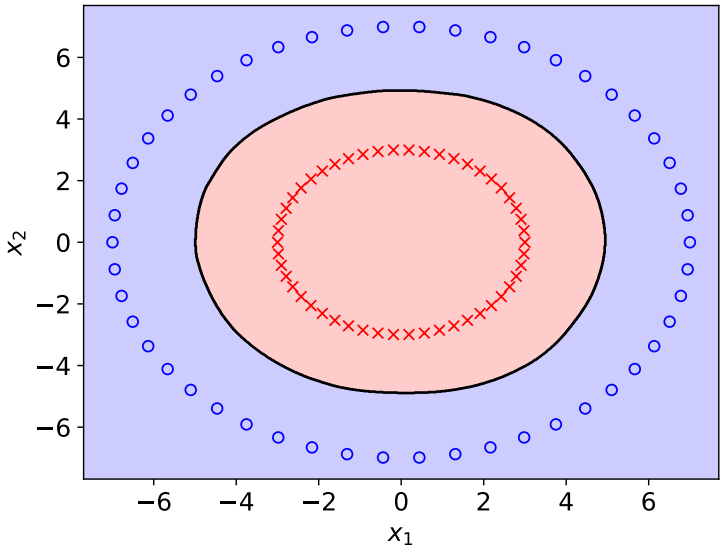
Results: $p = 3$ — second component of $\mathrm{relu}(Wx + b)$



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



Results: $p = 1000$

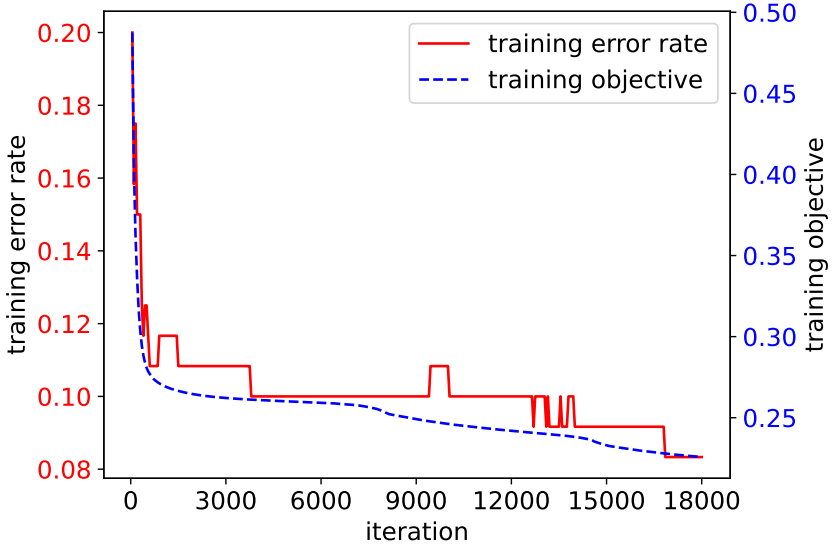


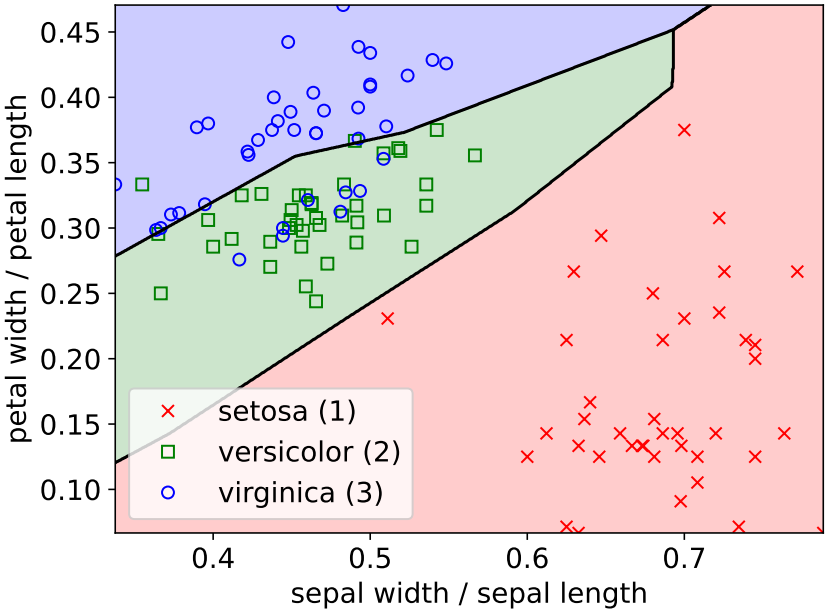
Iris data classifier

- Features:

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

- Neural net: $f(x) = \mathrm{softmax}(A\,\mathrm{relu}(Wx + b) + c)$
  - 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 $\Pr(Y = k \mid X = x)$
- Feature transformation and training procedure: same as in synthetic example
- Training error rate: 8.33%, test error rate: 10.0%

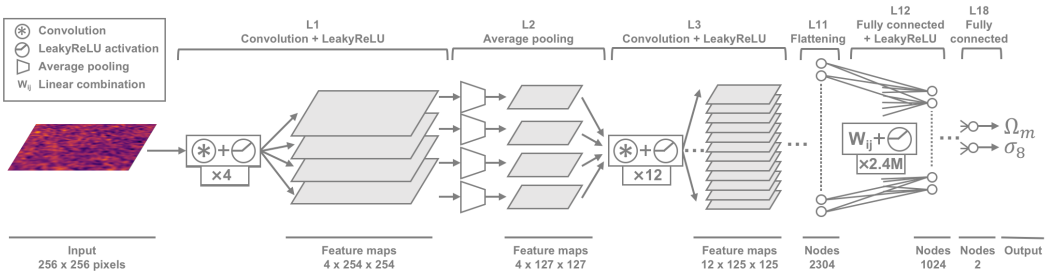

training error rate
training objective
0.20
0.18
0.16
0.14
0.12
0.10
0.08
0.50
0.45
0.40
0.35
0.30
0.25
0
3000
6000
9000
12000
15000
18000
iteration


petal width / petal length
sepal width / sepal length
0.10
0.15
0.20
0.25
0.30
0.35
0.40
0.45
0.4
0.5
0.6
0.7
setosa (1)
versicolor (2)
virginica (3)

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)