First example

Philipp Krähenbühl, UT Austin

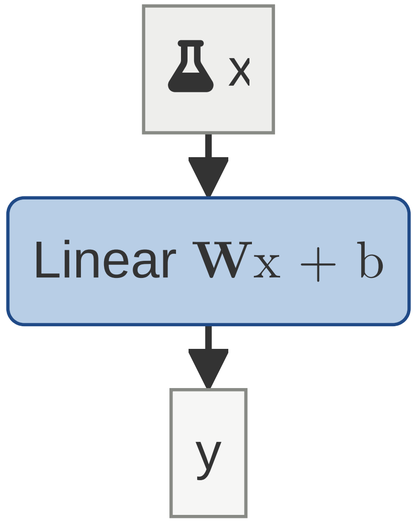
Goals

- We train our first “deep” network

Plan

Train a single layer deep network

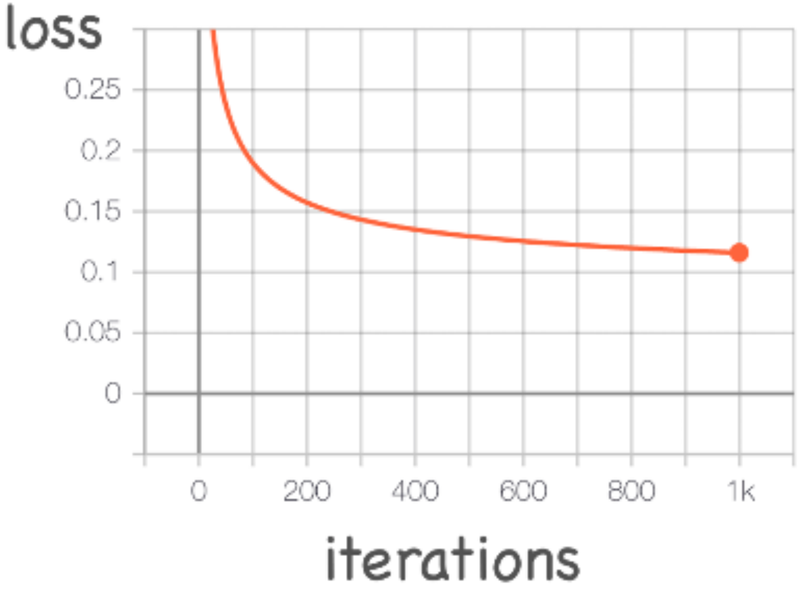
Task / Model



Dataset / Loss

$$l(\theta|\mathbf{x}, \mathbf{y})$$

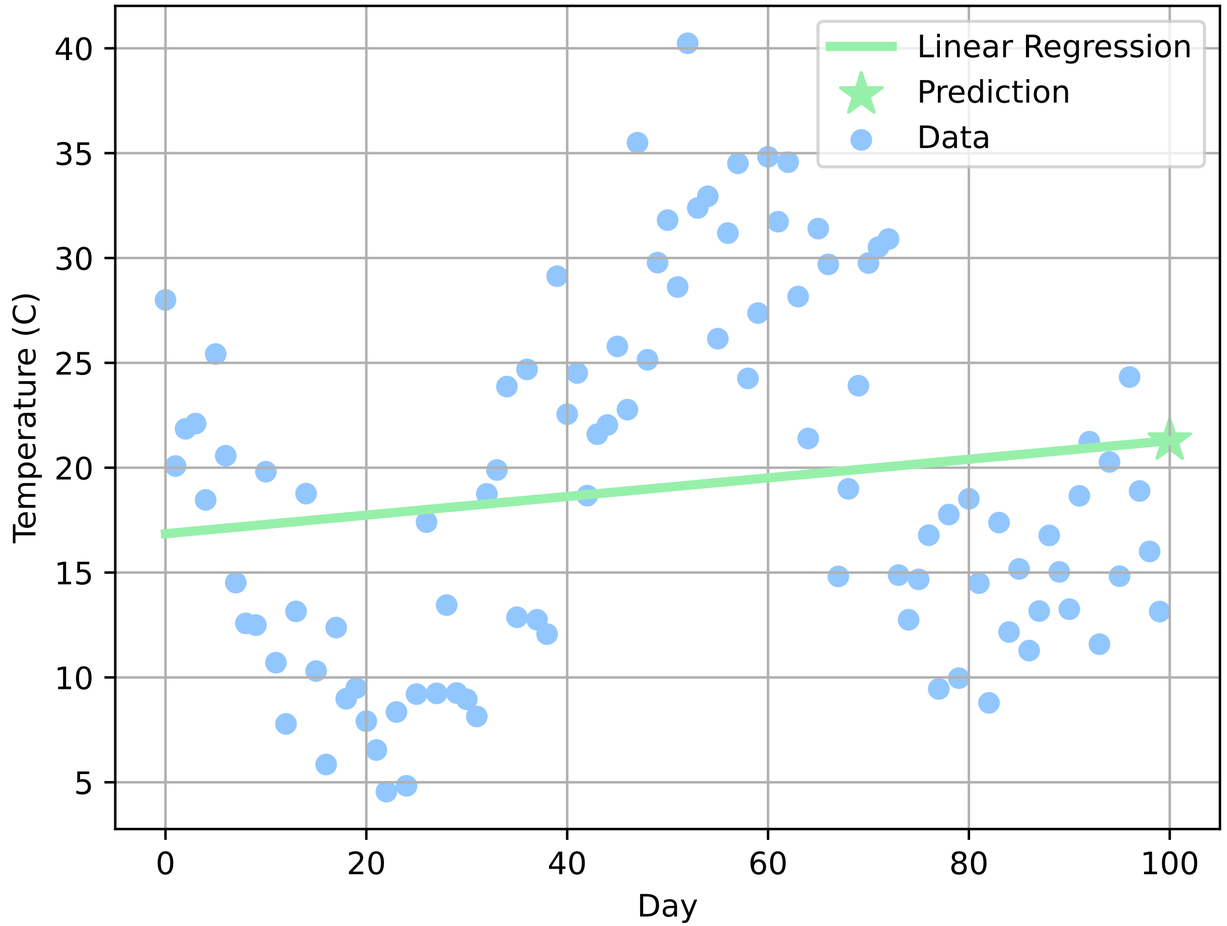
Training / Optimization



Linear regression and classification

Linear Regression

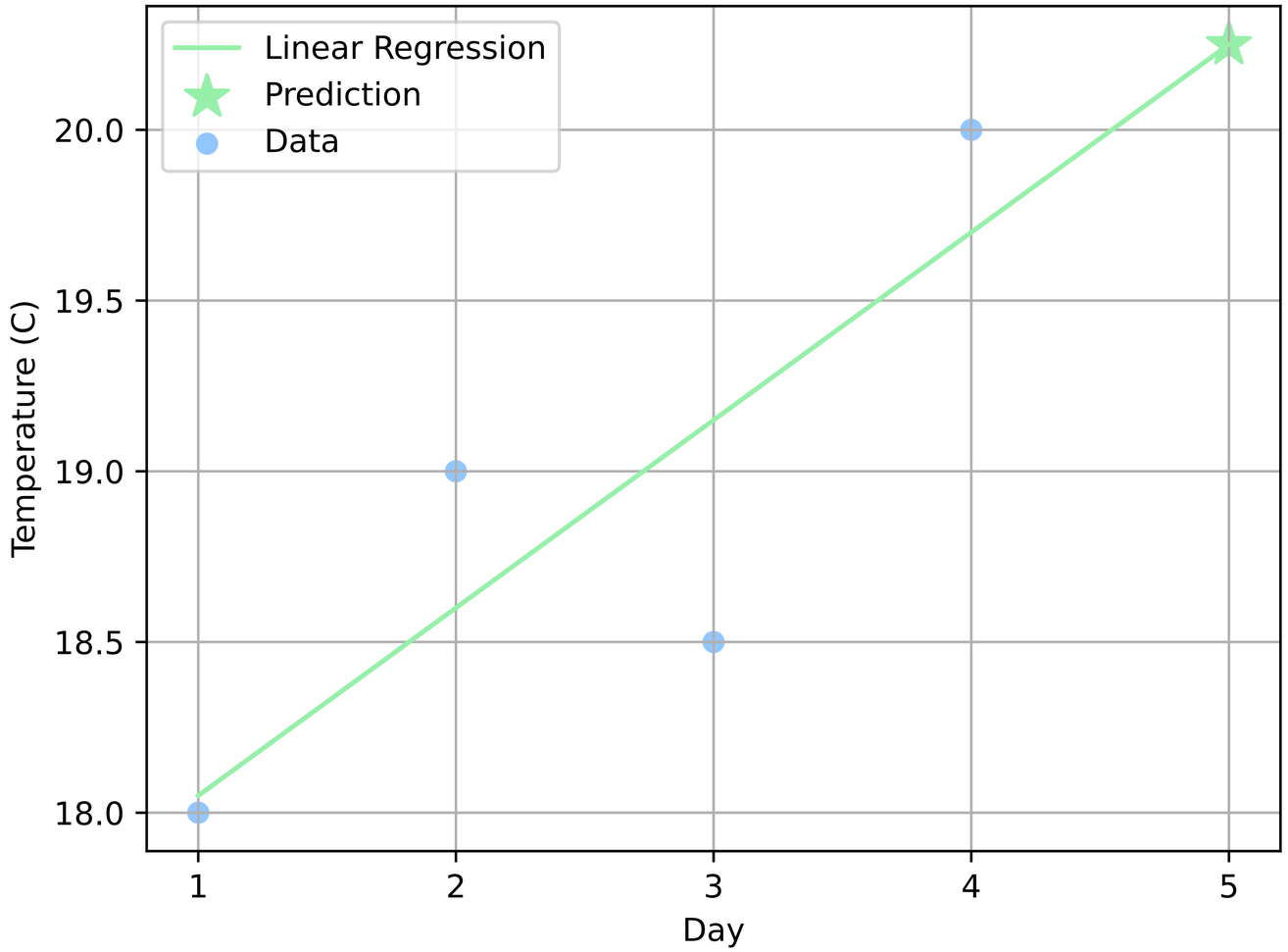
- Regression model:
  - $f_\theta : \mathbb{R}^n \rightarrow \mathbb{R}^d$
- Linear regression:
  - $f_\theta(\mathbf{x}) = \mathbf{W}\mathbf{x} + \mathbf{b}$
- Parameters:
  - $\theta = (\mathbf{W}, \mathbf{b})$



Linear Regression

Example

- Temperature forecast
  - $f(x)$: average temperature on day x

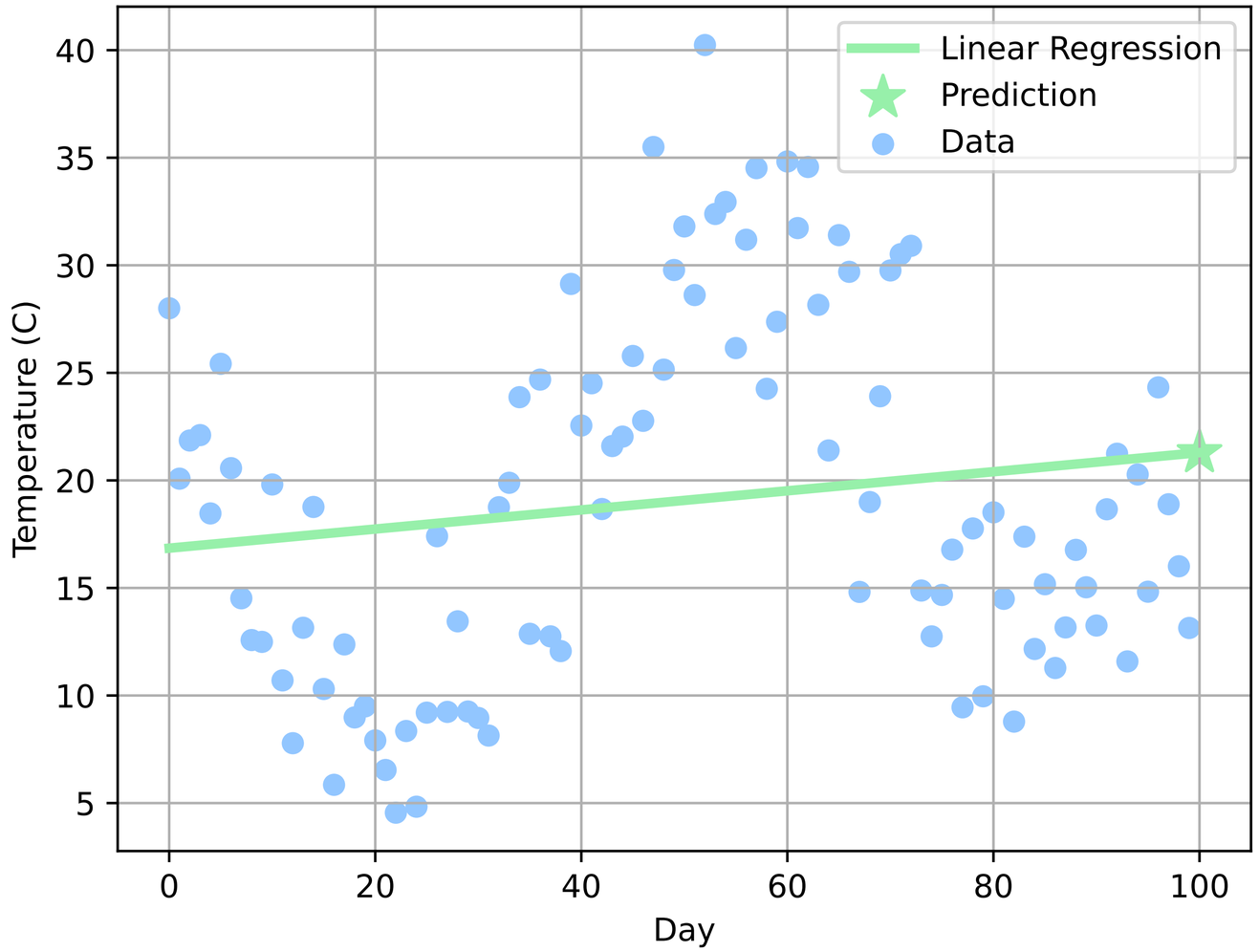


Linear Regression in PyTorch

Linear Regression

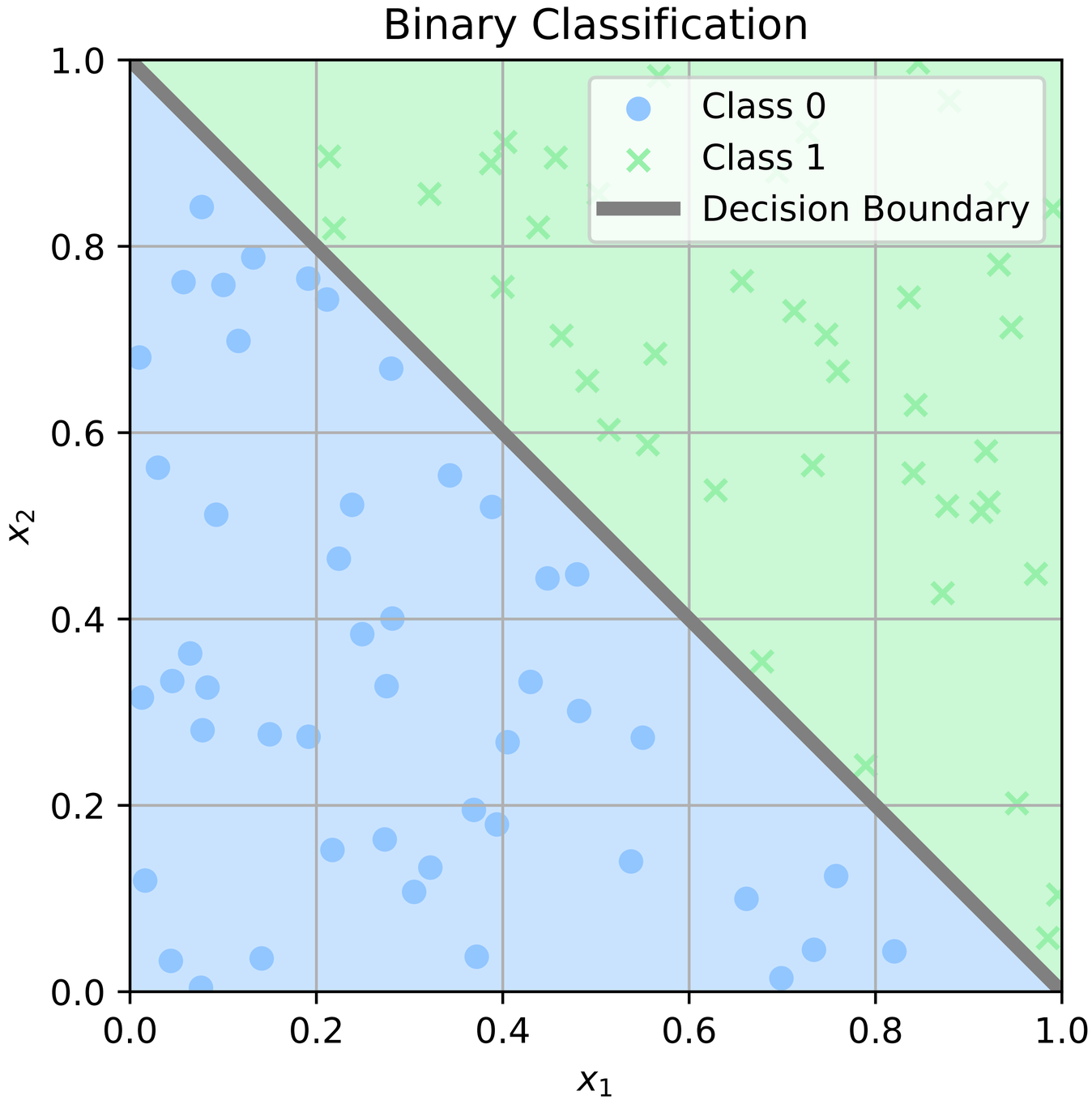
Limitations

- Cannot capture non-linear patterns
  - Cyclical functions
  - Quadratic functions
  - ...



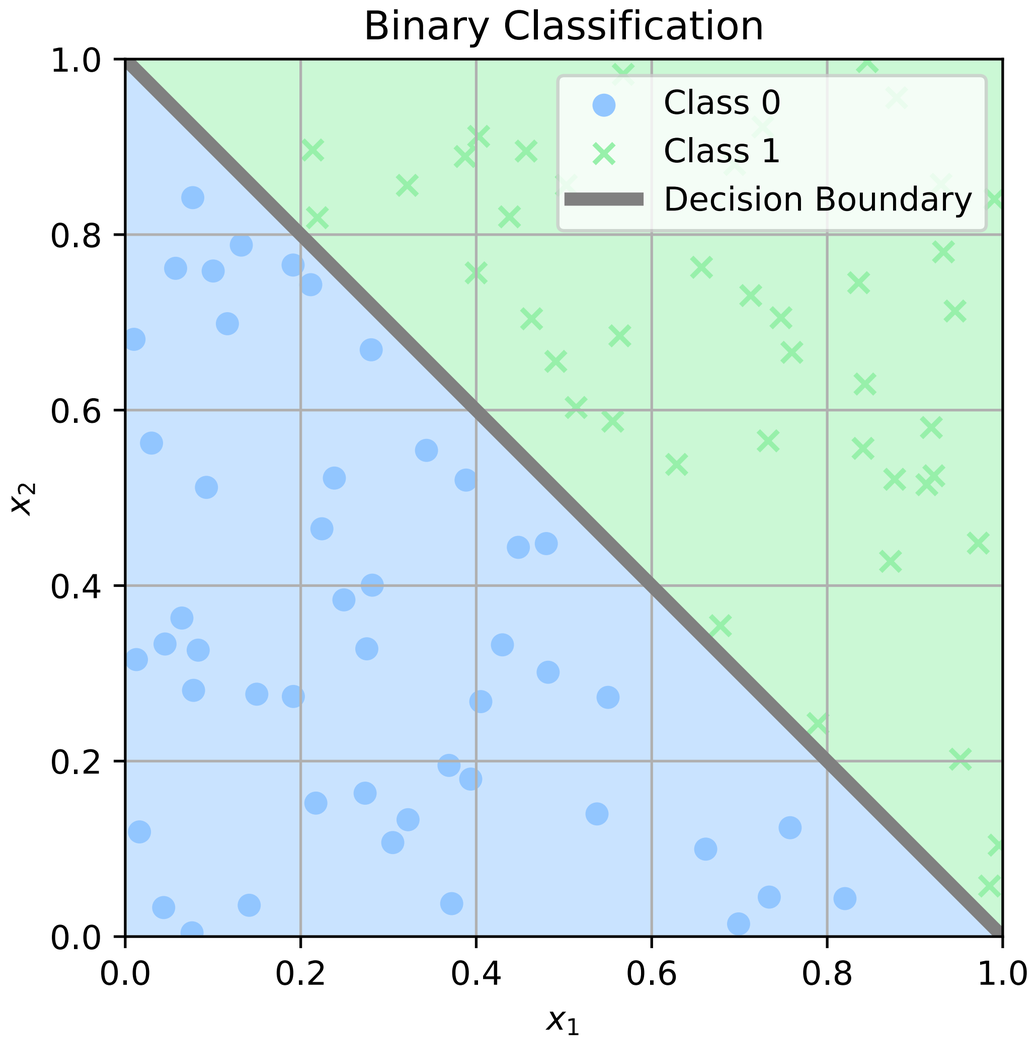
Linear Binary Classification

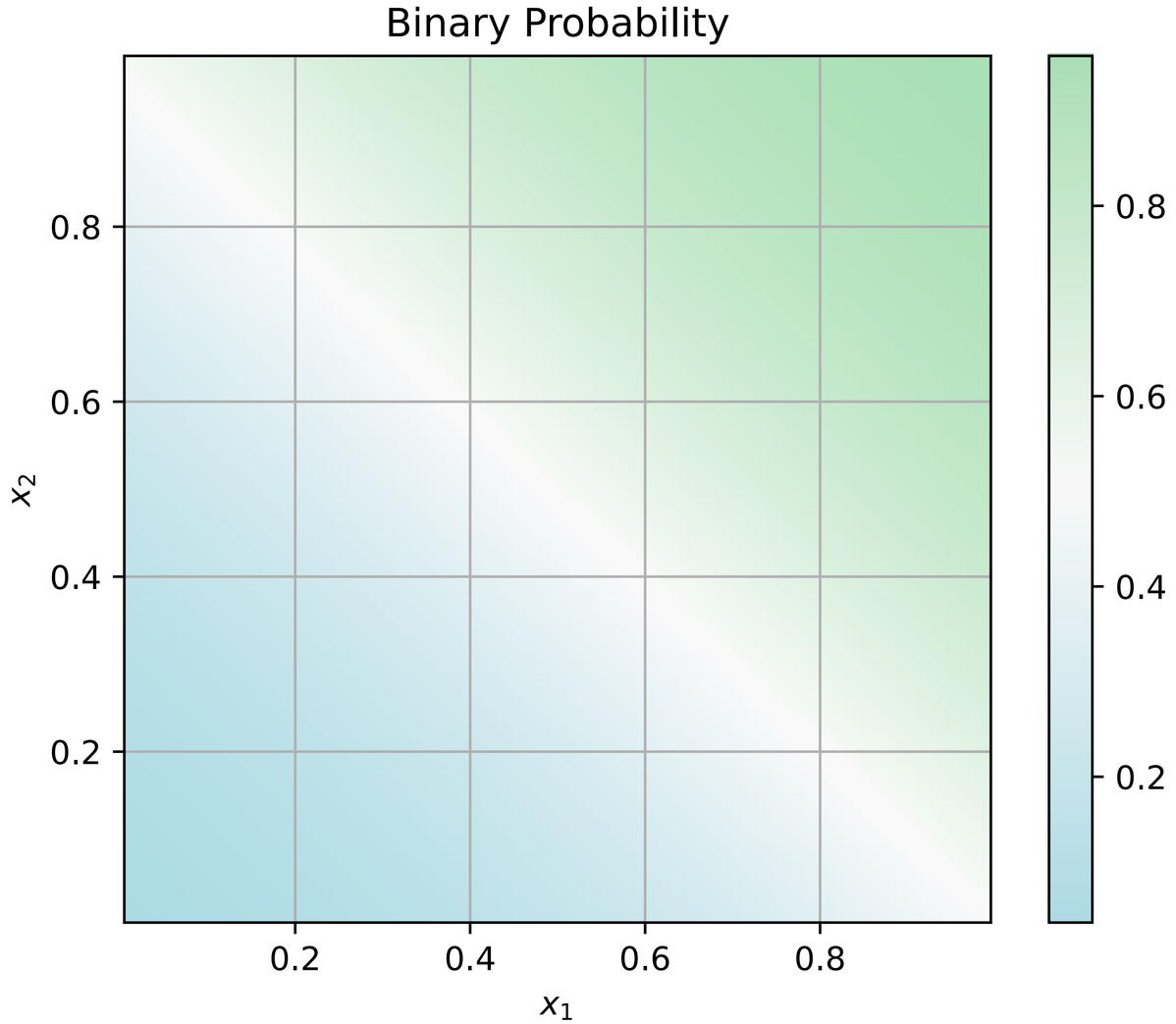
- Binary classification model:
  - $f_\theta : \mathbb{R}^n \to [0,1]$
- Linear binary classification:
  - $f_\theta(\mathbf{x}) = \sigma(W\mathbf{x} + b)$
  - $\sigma(x) = \dfrac{1}{1 + \exp(-x)}$
- Parameters:
  - $\theta = (W, b)$



Decision boundaries

$$f_\theta(\mathbf{x}) = \sigma(W\mathbf{x} + b)$$

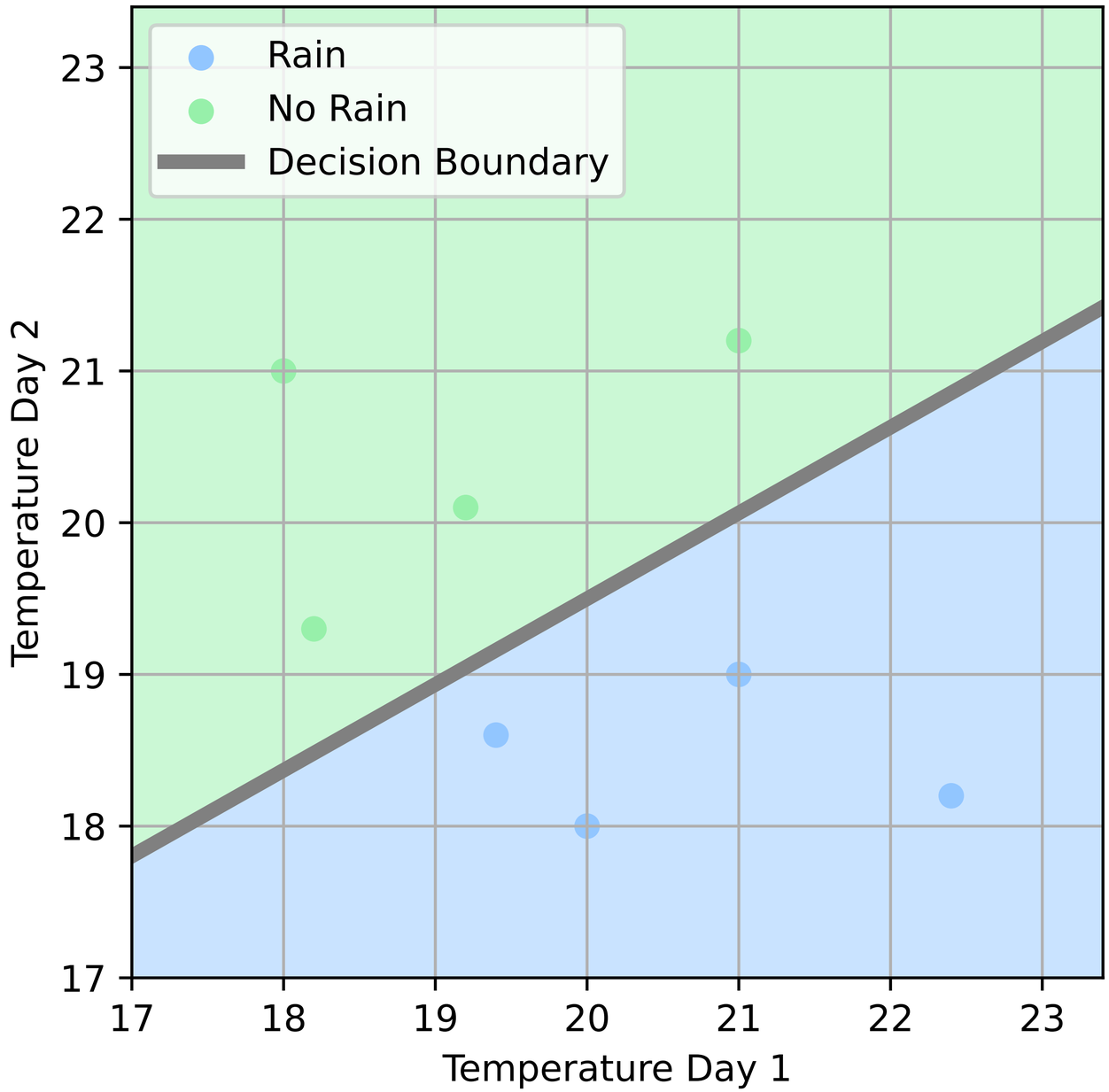




Linear Binary Classification

Example

- Input x: average daily temperature (for two consecutive days)
- Output $f(x)$: will it rain on day 2?
- $P(\text{rain}) = f_\theta(\mathbf{x}) = \sigma(W\mathbf{x} + b)$

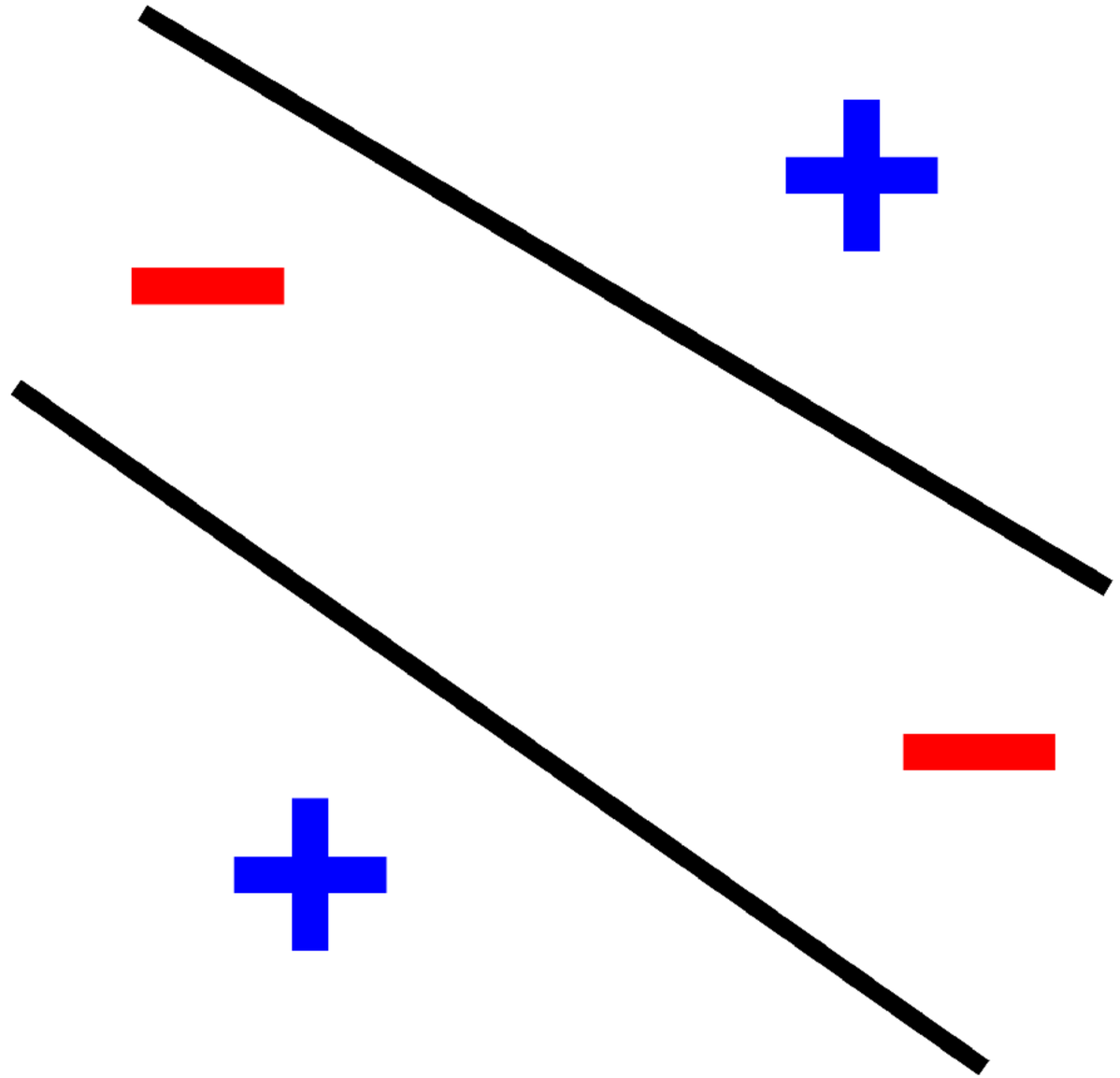


Linear Binary Classification in PyTorch

Linear Binary Classification

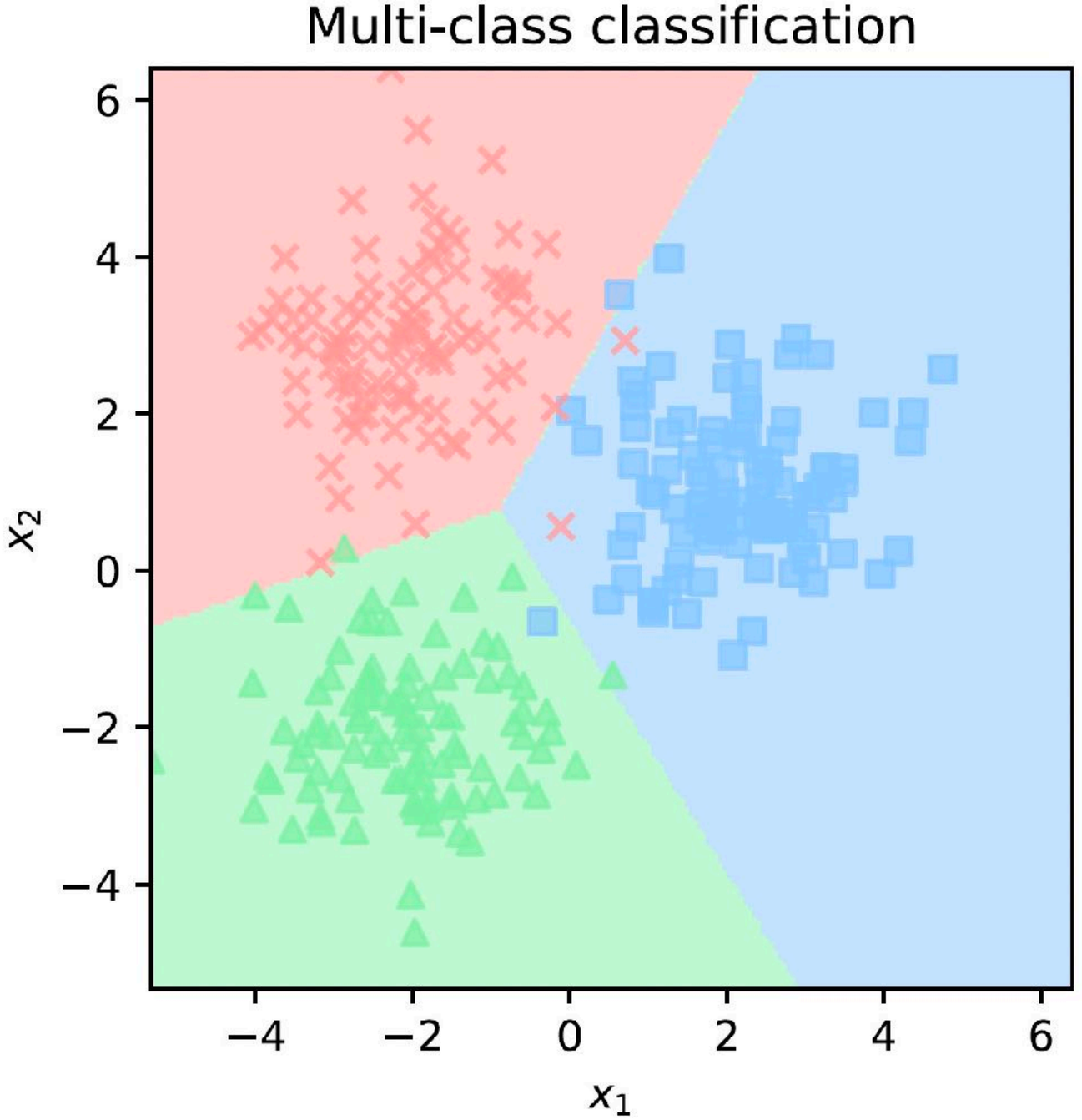
Limitations

- Only linear decision boundaries
- 2 classes only



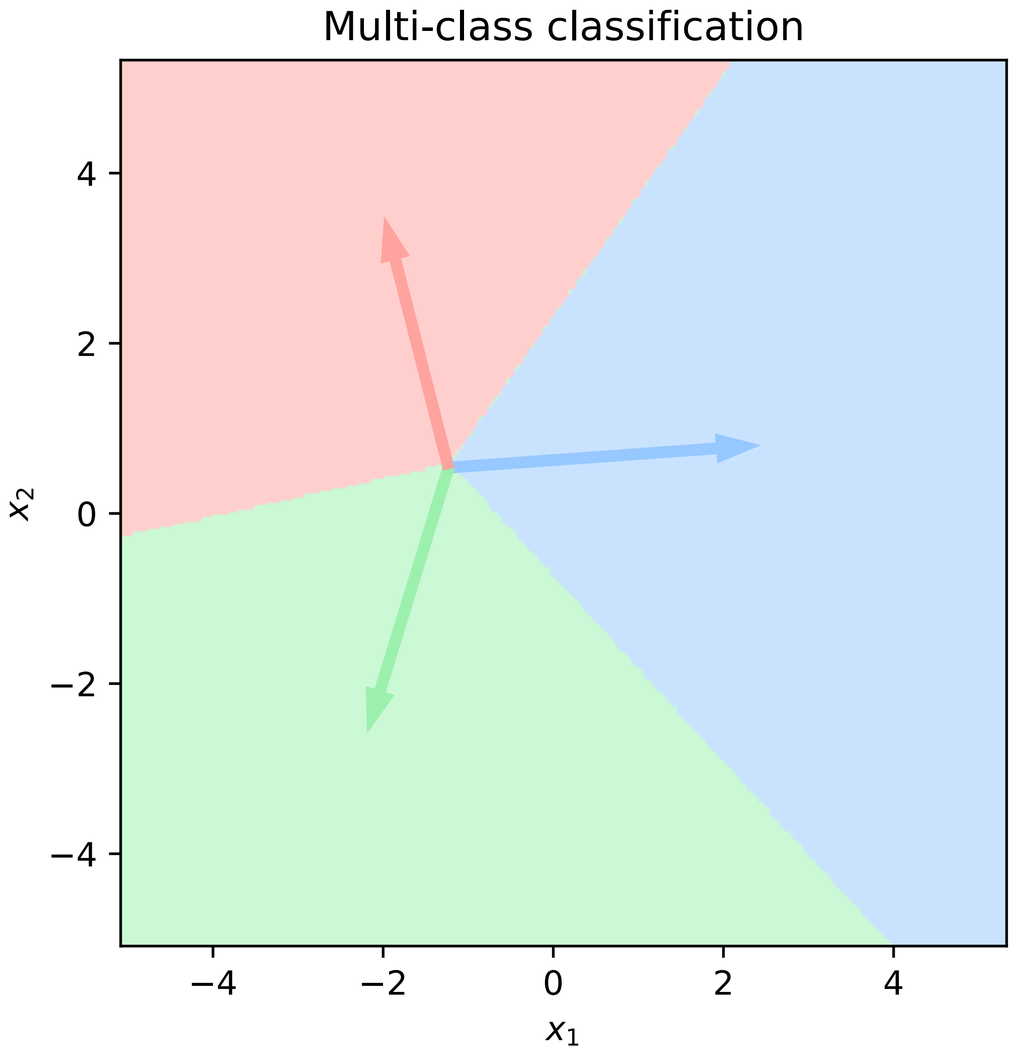
Linear Multi-Class Classification

- Multi-class Classification Model:
  - $f_\theta : \mathbb{R}^n \to \mathbb{P}^c$
    where $\mathbb{P}^c \subset \mathbb{R}_+^c$ and $\forall_{\mathbf{x} \in \mathbb{P}^c} \mathbf{x}^\top \mathbf{1} = 1$
- Linear Multi-class Classification Model:
  - $f_\theta(\mathbf{x}) = \text{softmax}(\mathbf{W}\mathbf{x} + \mathbf{b})$
  - $\text{softmax}(\mathbf{v})_i = \dfrac{\exp(\mathbf{v}_i)}{\sum_k \exp(\mathbf{v}_k)}$
- Parameters:
  - $\theta = (\mathbf{W}, \mathbf{b})$

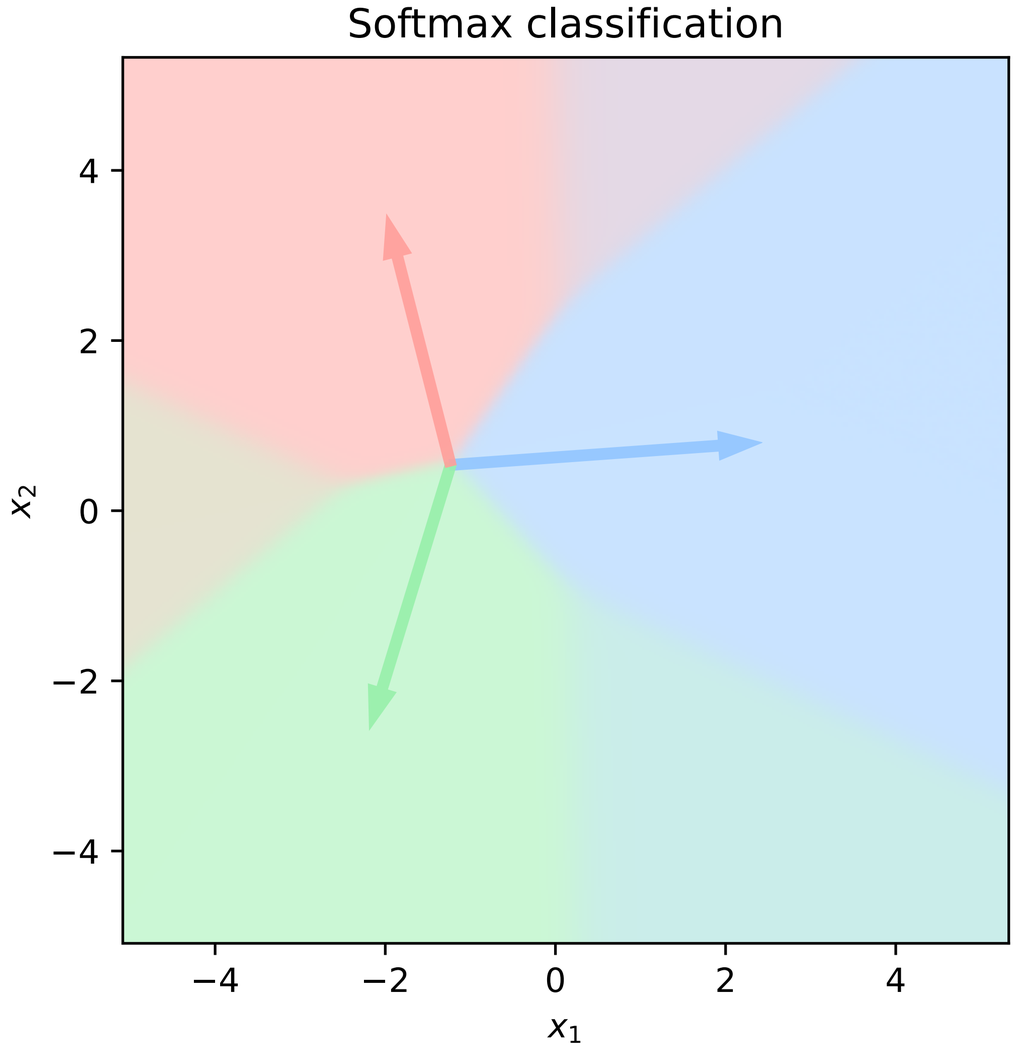


Linear Multi-Class Classification

Hard boundaries



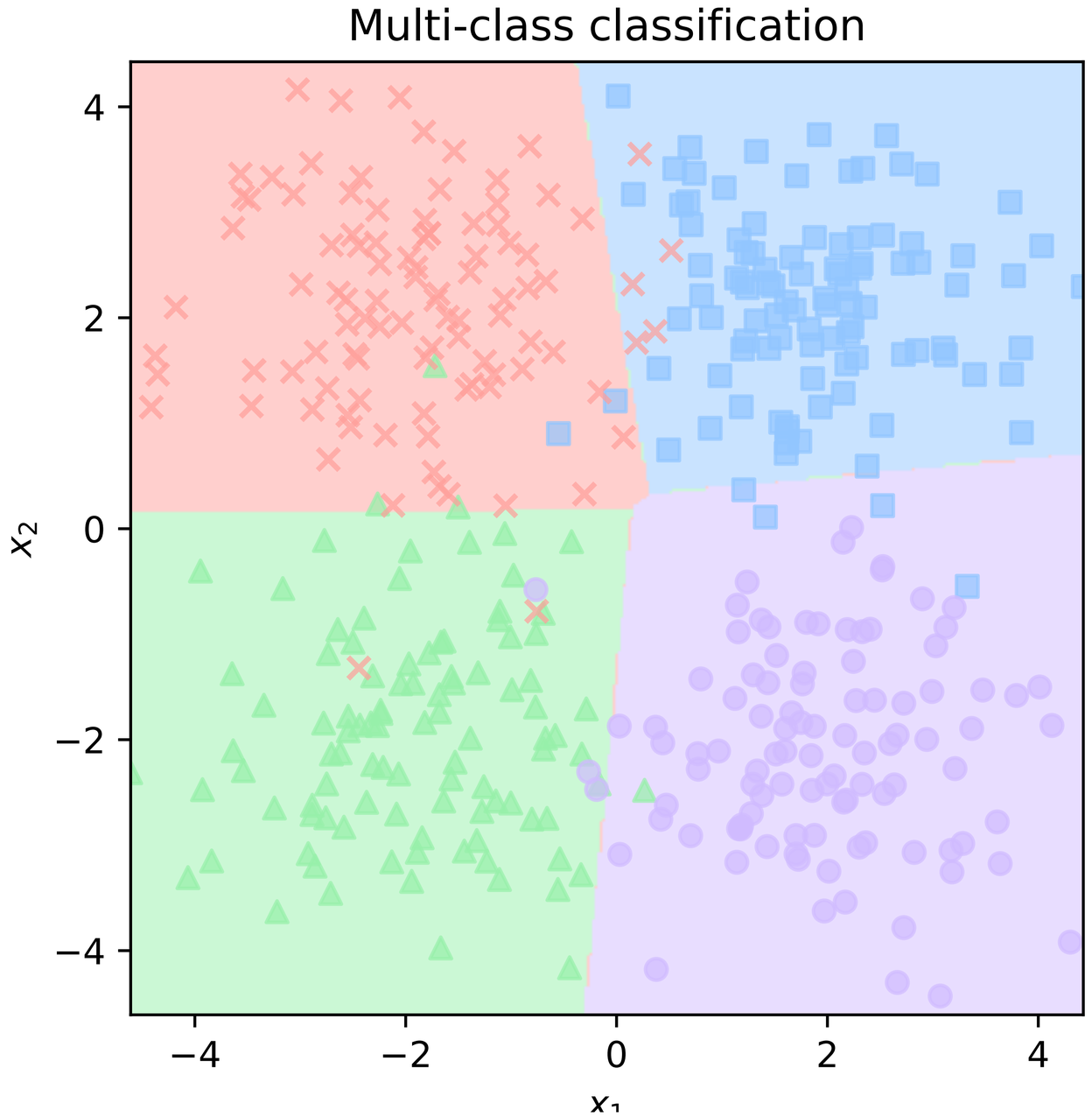
Soft boundaries



Linear Multi-Class Classification

Example

- Input x: average daily temperature (for two consecutive days)
- Output $f(x)$: precipitation (rain, snow, hail, sun)
- Prediction:
  - $P(\text{rain} \mid \mathbf{x}) = f_\theta(\mathbf{x})_1 = \text{softmax}(\mathbf{Wx} + \mathbf{b})_1$
  - $P(\text{snow} \mid \mathbf{x}) = f_\theta(\mathbf{x})_2 = \text{softmax}(\mathbf{Wx} + \mathbf{b})_2$
  - $P(\text{hail} \mid \mathbf{x}) = f_\theta(\mathbf{x})_3 = \text{softmax}(\mathbf{Wx} + \mathbf{b})_3$
  - $P(\text{sun} \mid \mathbf{x}) = f_\theta(\mathbf{x})_4 = \text{softmax}(\mathbf{Wx} + \mathbf{b})_4$

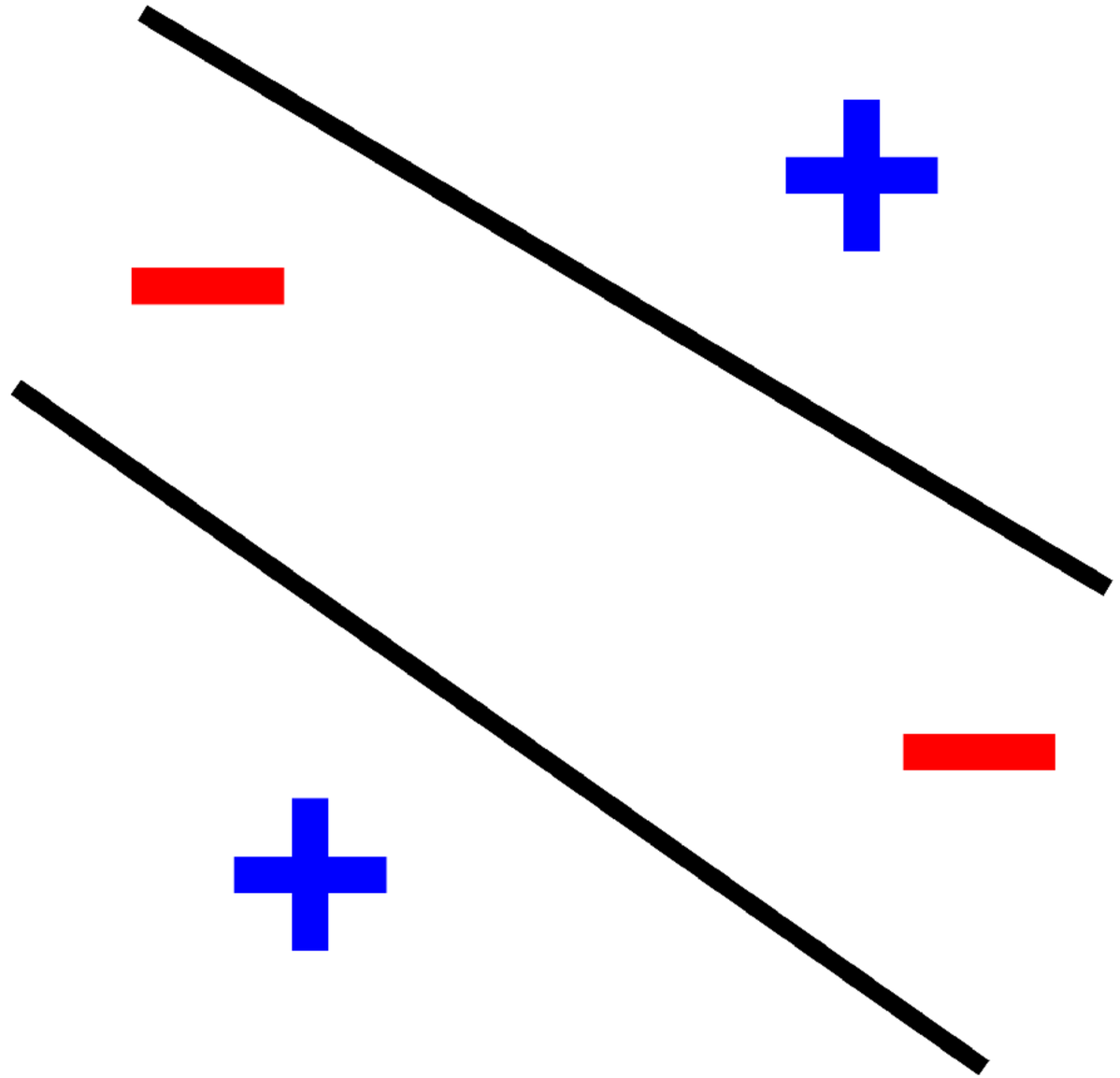


Linear Multi-class Classification in PyTorch

Linear Multi-class Classification

Limitations

- Only linear decision boundaries
  - Between pairs of classes



Multi-Class vs Multiple Binary Classification: Demo

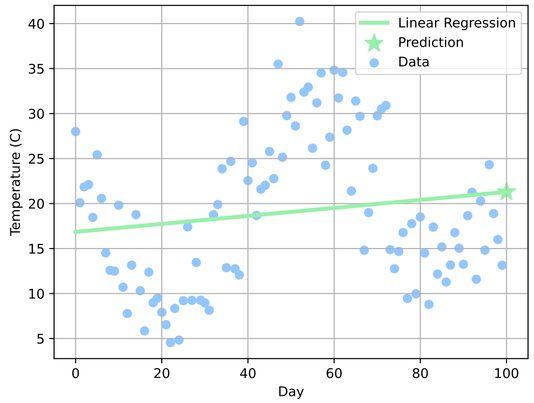
- Multi-class classification (**Softmax**):
  - Describes exactly one category
  - no negative examples
  - calibrated probabilities
  - used for mutually exclusive categories
- Examples:
  - Predicting the weather (rain, cloudy, sunny)
  - Predicting the scientific name of an animal
  - Predicting the next word in a sentence

- Multiple binary classifier (**Sigmoid**):
  - Allows for multiple categories
  - requires negative examples
  - uncalibrated probabilities
  - used for multi-label tagging
- Examples:
  - Predicting where in Texas it will rain
  - Predicting attributes of an animal
  - Predicting which books a sentence can be found in

TL;DR

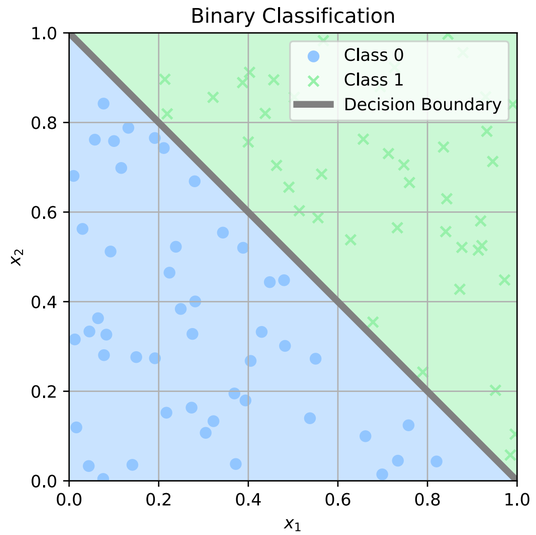
Regression

$$f_\theta(\mathbf{x}) = \mathbf{W}\mathbf{x} + \mathbf{b}$$



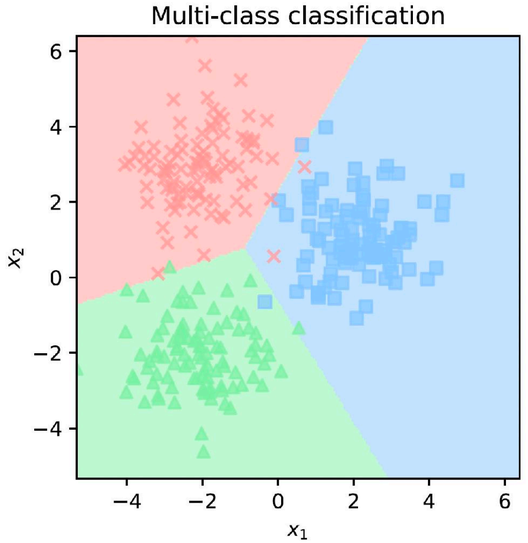
Binary Classification

$$f_\theta(\mathbf{x}) = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b})$$



Multi-class Classification

$$f_\theta(\mathbf{x}) = \text{softmax}(\mathbf{W}\mathbf{x} + \mathbf{b})$$



Question for the break

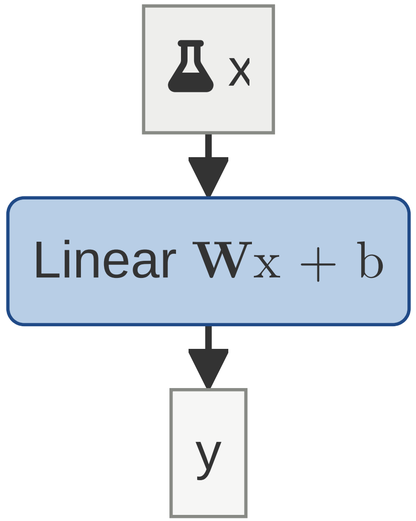
- Given a model, how do we measure how good it is?

Datasets and Losses

Plan

Train a single layer deep network

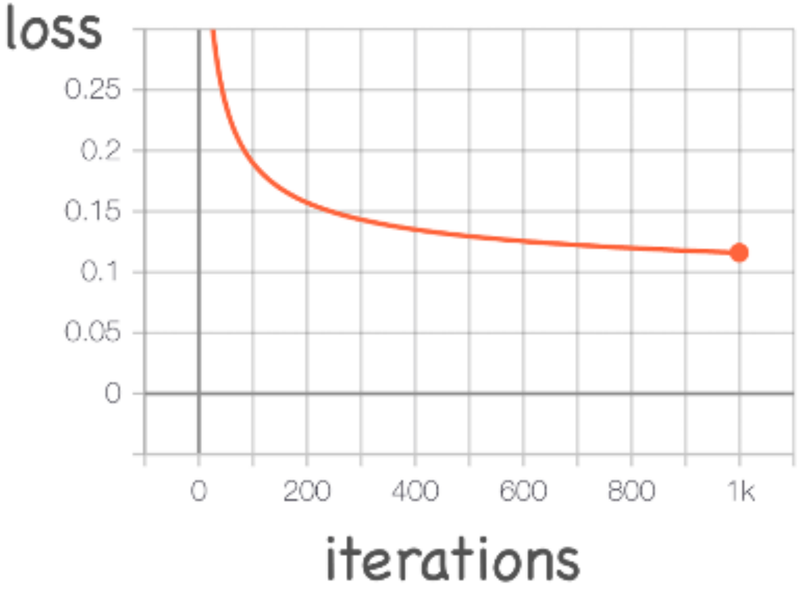
Task / Model



Dataset / Loss

$$l(\theta|\mathbf{x}, \mathbf{y})$$

Training / Optimization



What is a dataset?

- A collection of (labeled) data “samples”
  - Inputs and (desired) outputs of a model

$$\mathcal{D} = \left\{(x_i, y_i)\right\}_{i=1}^{N}$$

where $x_i \sim P(X), y_i \sim P(Y|X)$

Examples of data

$$\mathscr{D} = \{(x_i, y_i)\}_{i=1}^{N}$$

where $x_i \sim P(X), y_i \sim P(Y|X)$

- Data: Images
  Labels: Class, text description, objects, ...
- Data: Text (the entire internet)
  Labels: The next word
- Data: Audio waveform
  Labels: Transcription
- Data: Protein sequence
  Labels: 3D structure, function, ...

Given a model, how do we measure how good it is?

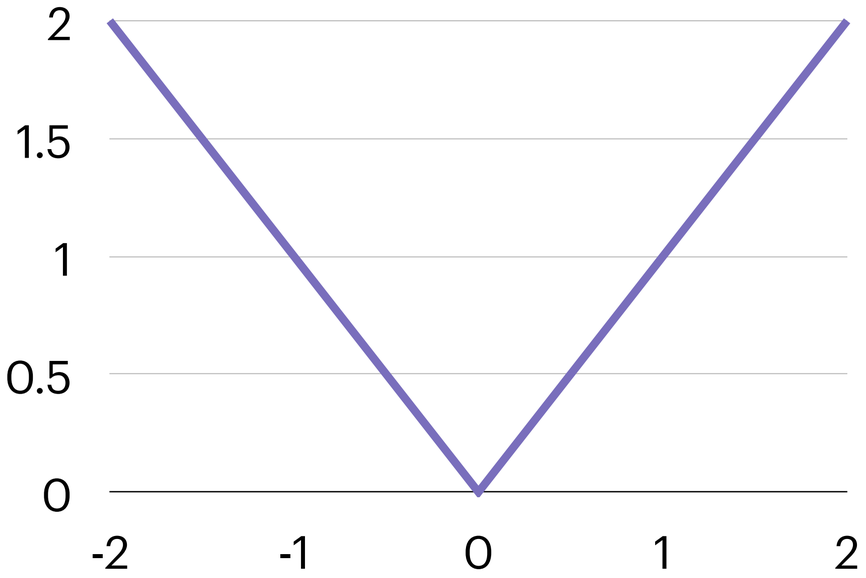
- We measure the "accuracy" of predictions on a dataset $\mathscr{D} = \left\{(x_i, y_i)\right\}_{i=1}^{N}$
- Classification: $\hat{y}_i = \arg\max_y P(y \mid x_i)$
  - Accuracy $y_i = \hat{y}_i$
- Regression: $\hat{y}_i = f(x_i)$
  - Accuracy $y_i = \hat{y}_i$?
    - Is $y_i = \hat{y}_i$ ever true?

Accuracy

- Regression: Never exactly equal
- Hard to optimize
  - Not differentiable
- Solution: Loss functions
  - Differentiable “proxy’s” for accuracy

Loss functions

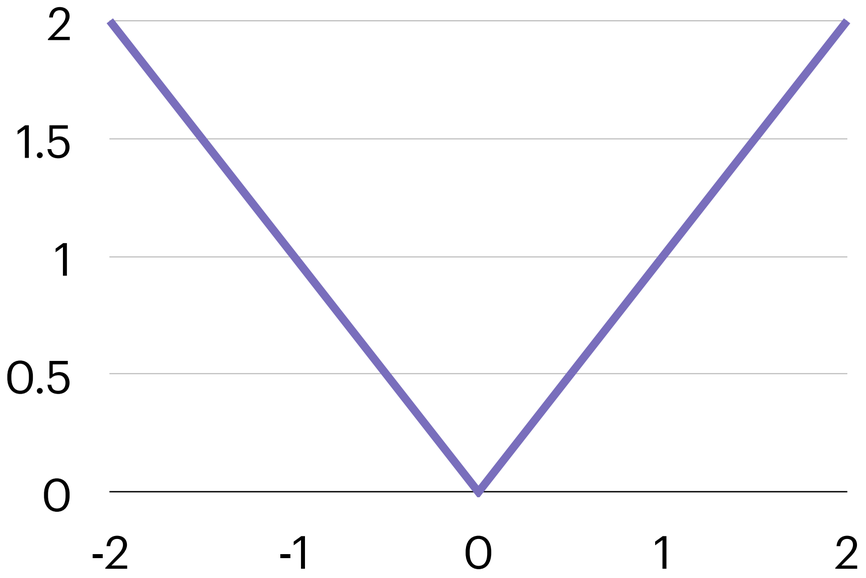
- A **loss function** measures the quality of the model
- Loss function $\ell(\theta \mid x, y)$
- Expected loss
  $L(\theta \mid \mathcal{D}) = E_{x,y \sim \mathcal{D}} \left[\ell(\theta \mid x, y) \right]$



Properties of loss functions

- Low loss - good 🙂
- High loss - bad ☹️
- Loss function over entire dataset

$$L(\theta) = L(\theta \mid \mathcal{D}) = E_{x,y \sim \mathcal{D}} \left[\ell(\theta \mid x, y) \right]$$



Loss functions

Regression

$$f_\theta(\mathbf{x}) = \mathbf{W}\mathbf{x} + \mathbf{b}$$

L2 loss

$$\ell(\theta \,|\, \mathbf{x}, y) = \|y - f_\theta(\mathbf{x})\|^2$$

Binary Classification

$$f_\theta(\mathbf{x}) = \sigma\left(\mathbf{W}\mathbf{x} + \mathbf{b}\right)$$

Binary cross entropy loss

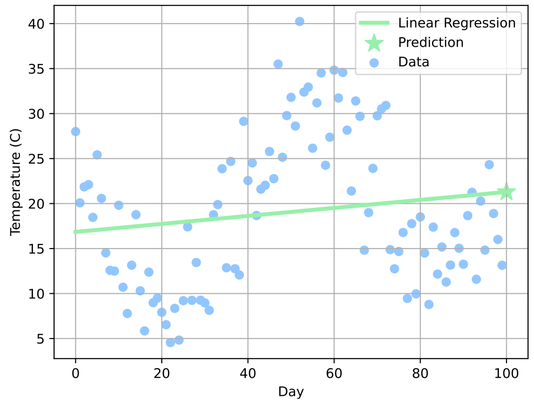
$$\begin{aligned}\ell(\theta \,|\, \mathbf{x}, y) &= \log P(y \,|\, \mathbf{x}) \\ &= -y \log \sigma(f_\theta(\mathbf{x})) \\ &\quad -(1-y)\log(1-\sigma(f_\theta(\mathbf{x})))\end{aligned}$$

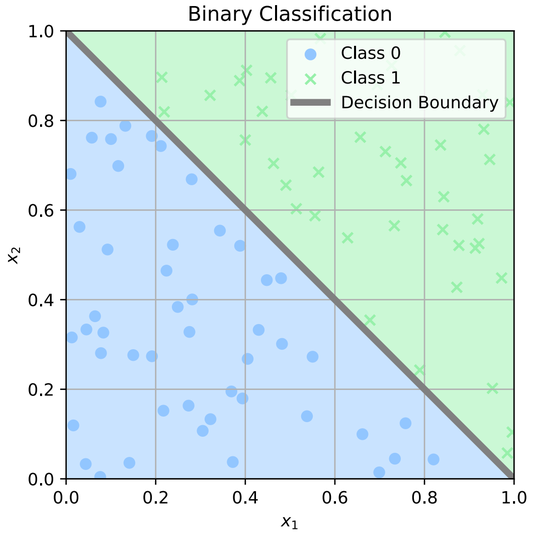
Multi-class Classification

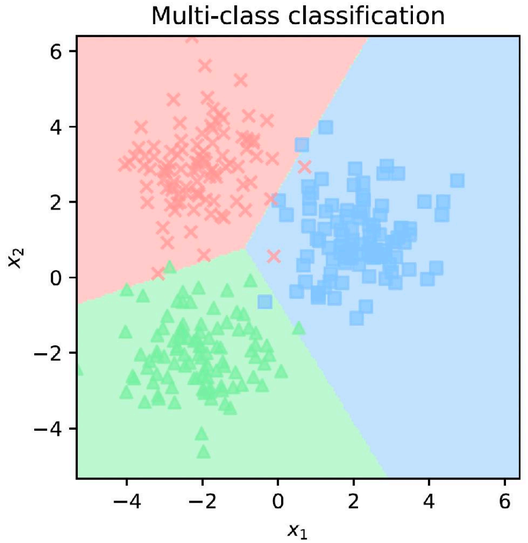
$$f_\theta(\mathbf{x}) = \text{softmax}\left(\mathbf{W}\mathbf{x} + \mathbf{b}\right)$$

Cross entropy loss

$$\begin{aligned}\ell(\theta \,|\, \mathbf{x}, y) &= -\log P(y \,|\, \mathbf{x}) \\ &= -\sum_c 1_{c=y} \log f_\theta(\mathbf{x})_c\end{aligned}$$







Loss functions in PyTorch

TL;DR

Regression

$$f_\theta(\mathbf{x}) = \mathbf{W}\mathbf{x} + \mathbf{b}$$

L2 loss

$$\ell(\theta \mid \mathbf{x}, y) = \|y - f_\theta(\mathbf{x})\|^2$$

Binary Classification

$$f_\theta(\mathbf{x}) = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b})$$

Binary cross entropy loss

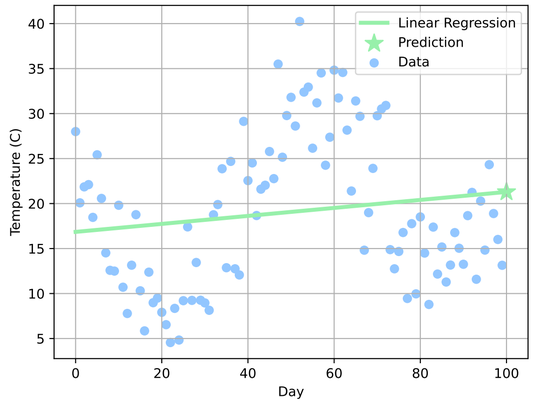
$$\begin{aligned}\ell(\theta \mid \mathbf{x}, y) &= \log P(y \mid \mathbf{x}) \\ &= -y \log \sigma(f_\theta(\mathbf{x})) \\ &\quad -(1-y)\log(1 - \sigma(f_\theta(\mathbf{x})))\end{aligned}$$

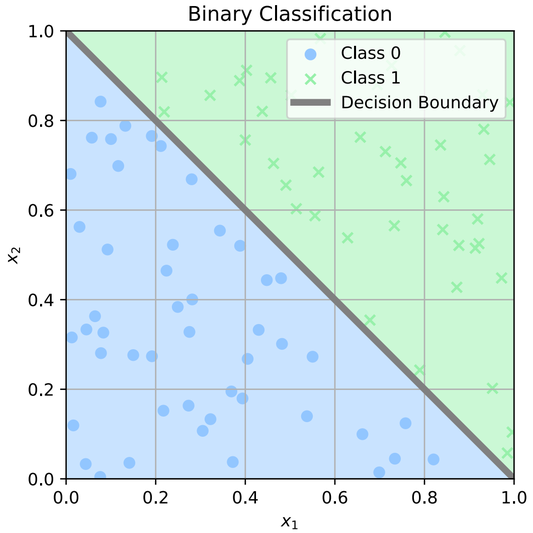
Multi-class Classification

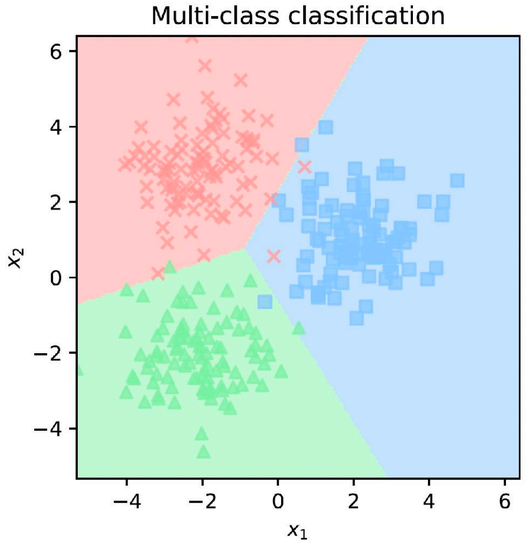
$$f_\theta(\mathbf{x}) = \text{softmax}(\mathbf{W}\mathbf{x} + \mathbf{b})$$

Cross entropy loss

$$\begin{aligned}\ell(\theta \mid \mathbf{x}, y) &= -\log P(y \mid \mathbf{x}) \\ &= -\sum_c 1_{c=y} \log f_\theta(\mathbf{x})_c\end{aligned}$$





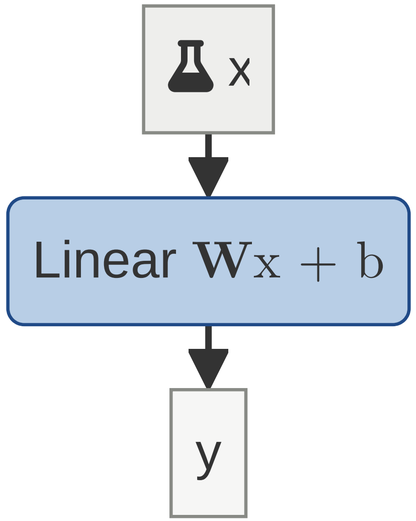


Optimization

Plan

Train a single layer deep network

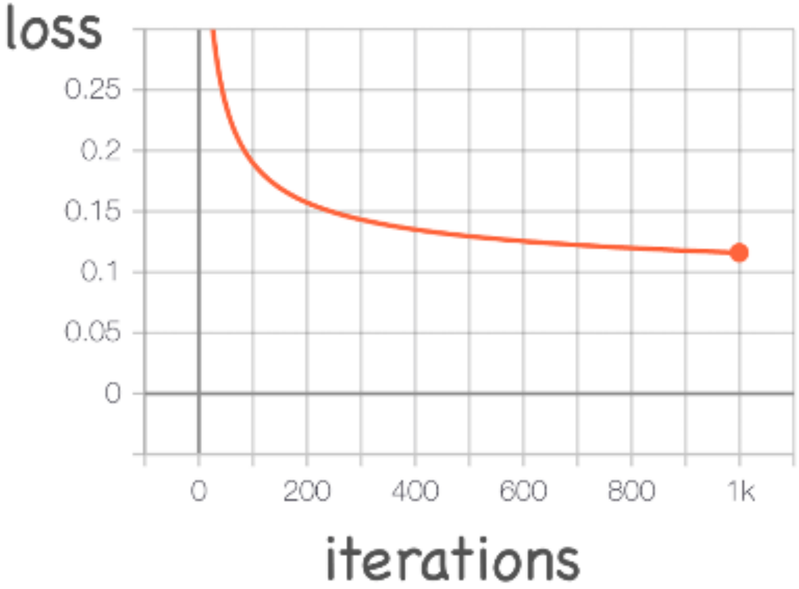
Task / Model



Dataset / Loss

$$l(\theta|\mathbf{x}, \mathbf{y})$$

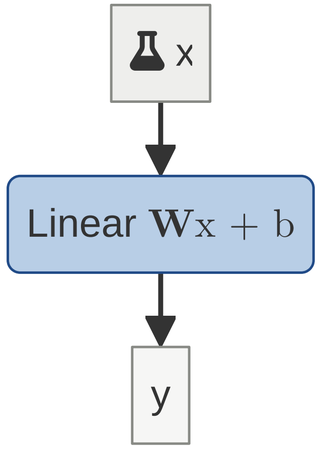
Training / Optimization



Recap: Model, Dataset, Loss

Model: $f_\theta(\mathbf{x})$

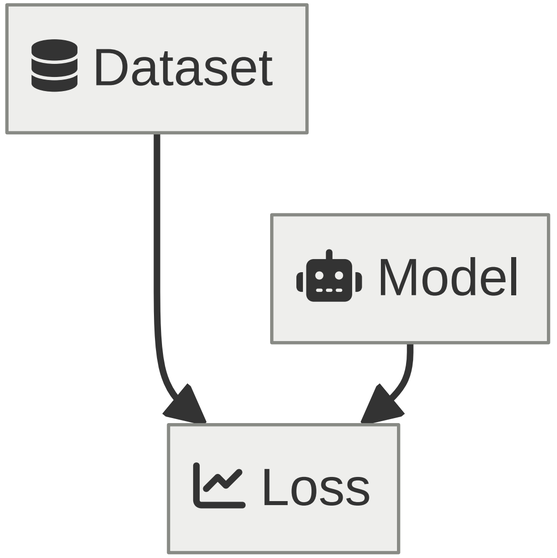
Parameters: $\theta = (\mathbf{W}, \mathbf{b})$



Expected Loss

$$L(\theta) = L(\theta \mid \mathscr{D}) = E_{x,y\sim\mathscr{D}}\left[\ell(\theta \mid x, y)\right]$$

- Low loss - good 🙂
- High loss - bad ☹️



Training

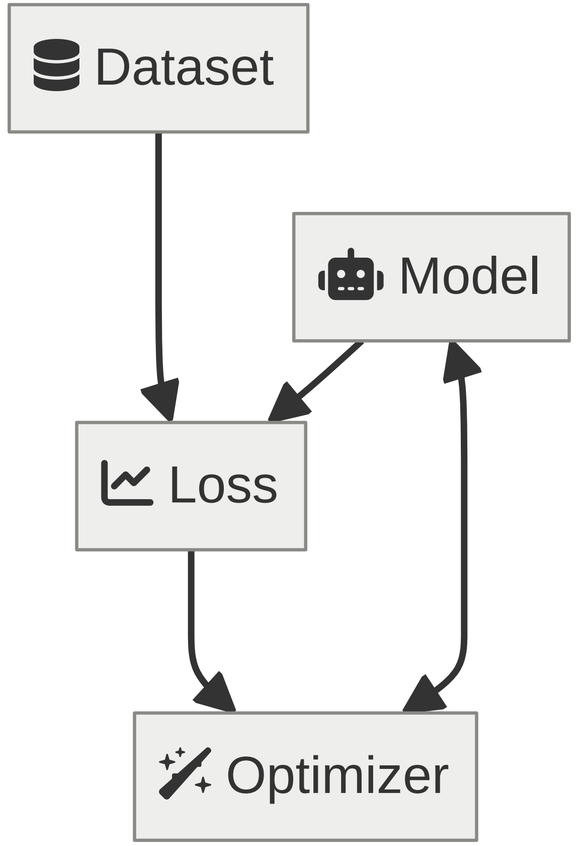
Model: $f_\theta(\mathbf{x})$ with parameters: $\theta = (\mathbf{W}, \mathbf{b})$

Expected Loss

$$L(\theta) = L(\theta \mid \mathscr{D}) = E_{x,y \sim \mathscr{D}} \left[\ell(\theta \mid x, y) \right]$$

Training objective:

- Find $\theta^* = \arg\max_\theta L(\theta \mid \mathscr{D})$



Gradient Descent

Deep learning uses **gradient descent**:

- Updates parameters via the negative gradient
- Iterative method

Update rule:

$$\theta' = \theta - \epsilon \left[\nabla L(\theta) \right]^{\top}$$

where ε is the learning rate

Pseudocode:

```
for iteration in range(n):
    J = ∇L(θ)
    θ = θ - ϵ * J.mT
```

Initialization

How should we chose the initial θ?

Random initialization

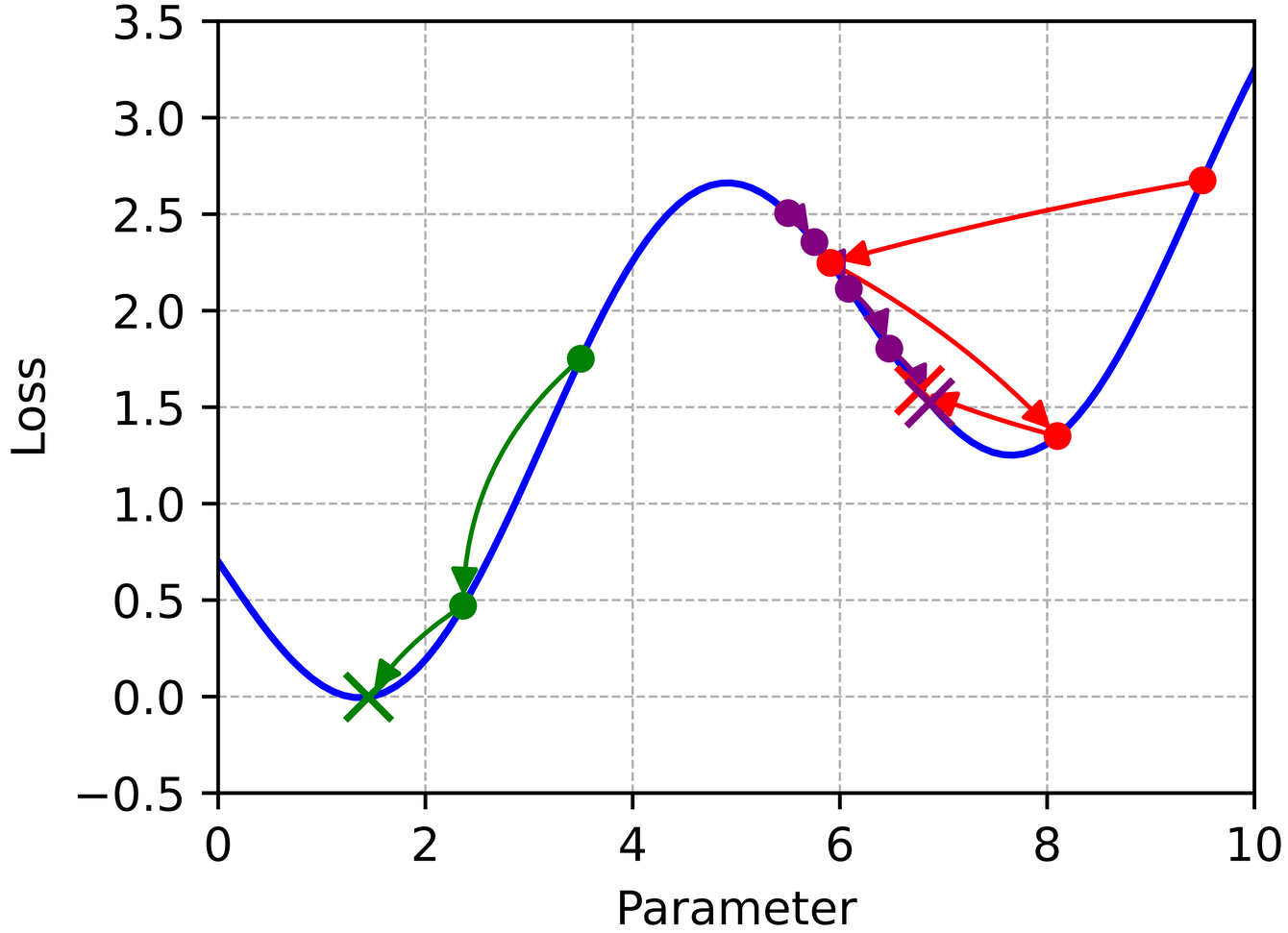
- Gaussian $\theta \sim \mathcal{N}(0,\sigma)$
- Uniform $\theta \sim U(-\sigma, \sigma)$

Pseudocode: Gradient Descent

```
θ ~ Init
for iteration in range(n):
    J = ∇L(θ)
    θ = θ - ϵ * J.mT
```

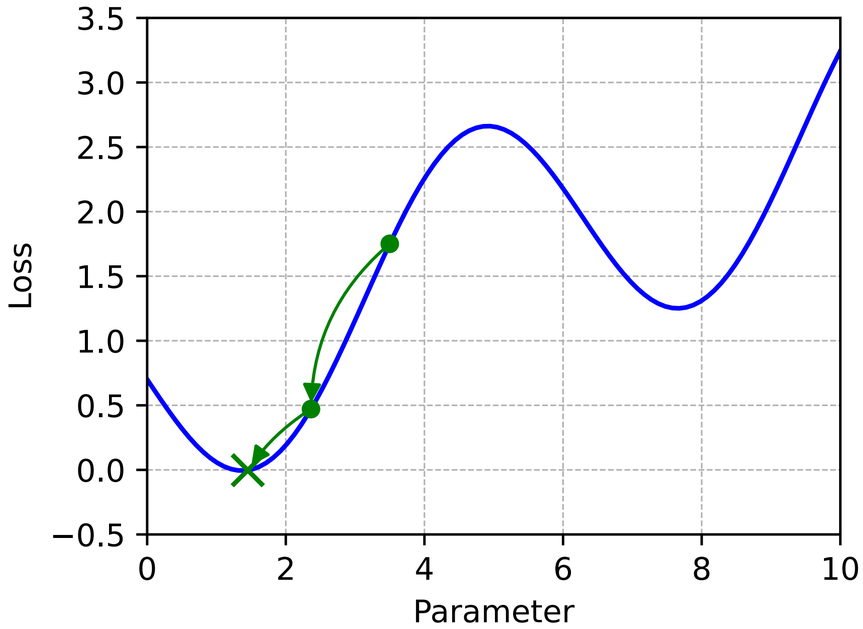
Gradient Descent in Action

Gradient Descent in Action

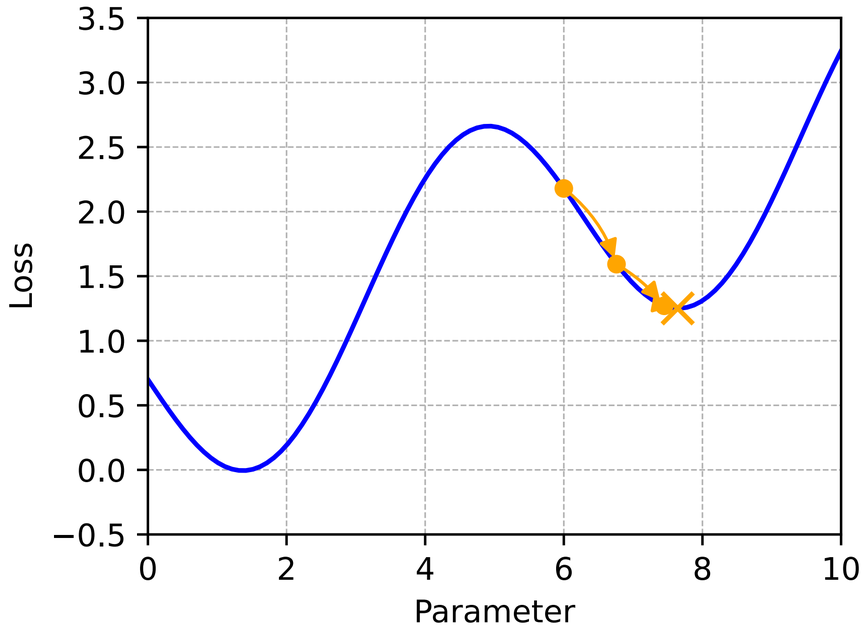


What can do wrong?

Best Case



Initialization

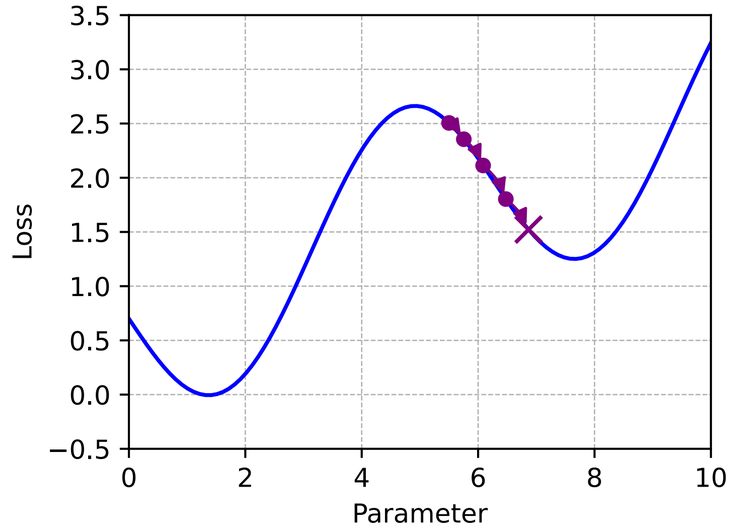


- Bumpy loss landscape
- Gets stuck in local minima

What can do wrong?

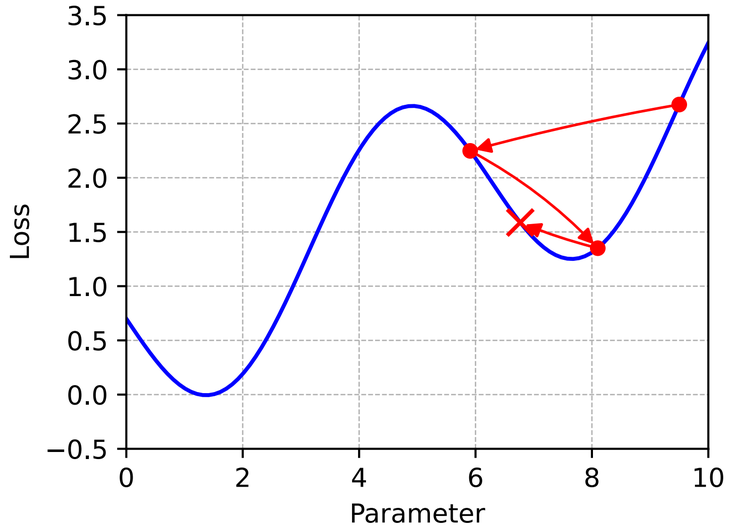
Gradient Descent: What Can Go Wrong?

Learning Rate Too Low



- Slow training

Learning Rate Too High



- Convergence
- Extreme case - NaNs!

Where do we get the gradient from?

Gradient $\nabla L(\theta|D) = E_{x,y\sim\mathcal{D}}\left[\nabla \ell(\theta|x,y)\right]$

```
θ ~ Init
for iteration in range(n):
    J = ∇L(θ)
    θ = θ - ϵ * J.mT
```

Gradient: Linear Regression

- $$\ell(\theta \mid \mathbf{x}, y) = (f_\theta(\mathbf{x}) - y)^2 = (\mathbf{w}^\top \mathbf{x} + b - y)^2$$
  $$\nabla_b \ell(\theta \mid \mathbf{x}, y) = 2(\mathbf{w}^\top \mathbf{x} + b - y)$$
  $$\nabla_{\mathbf{w}} \ell(\theta \mid \mathbf{x}, y) = 2(\mathbf{w}^\top \mathbf{x} + b - y)\mathbf{x}^\top$$

Gradient in general

Regression

$$f_\theta(\mathbf{x}) = \mathbf{W}\mathbf{x} + \mathbf{b}$$

L2 loss

$$\ell(\theta \,|\, \mathbf{x}, y) = \|y - f_\theta(\mathbf{x})\|^2$$

Binary Classification

$$f_\theta(\mathbf{x}) = \sigma\left(\mathbf{W}\mathbf{x} + \mathbf{b}\right)$$

Binary cross entropy loss

$$\begin{aligned}\ell(\theta \,|\, \mathbf{x}, y) &= \log P(y \,|\, \mathbf{x}) \\ &= -y \log \sigma(f_\theta(\mathbf{x})) \\ &\quad -(1-y)\log(1 - \sigma(f_\theta(\mathbf{x})))\end{aligned}$$

Multi-class Classification

$$f_\theta(\mathbf{x}) = \text{softmax}\left(\mathbf{W}\mathbf{x} + \mathbf{b}\right)$$

Cross entropy loss

$$\begin{aligned}\ell(\theta \,|\, \mathbf{x}, y) &= -\log P(y \,|\, \mathbf{x}) \\ &= -\sum_c 1_{c=y} \log f_\theta(\mathbf{x})_c\end{aligned}$$

General gradients

Get very complicated very quickly

TL;DR

Gradient descent (and variants): workhorse for 99% of deep learning

```
θ ~ Init
for iteration in range(n):
    J = ∇L(θ)
    θ = θ - ϵ * J.mT
```

Relies on loss and model gradient computation
(more next time)