Visual Computing Institute | RWTH AACHEN UNIVERSITY

Elements of Machine Learning & Data Science

Winter semester 2023/24

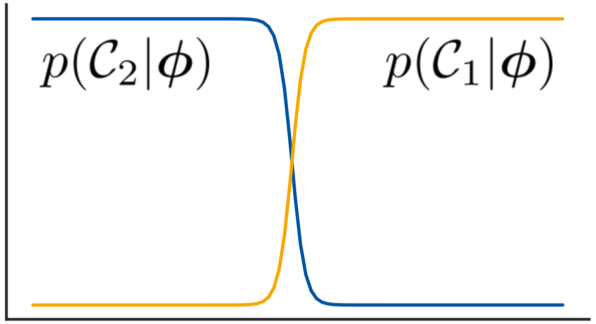
Lecture 17 – Support Vector Machines I

12.12.2023

Prof. Bastian Leibe

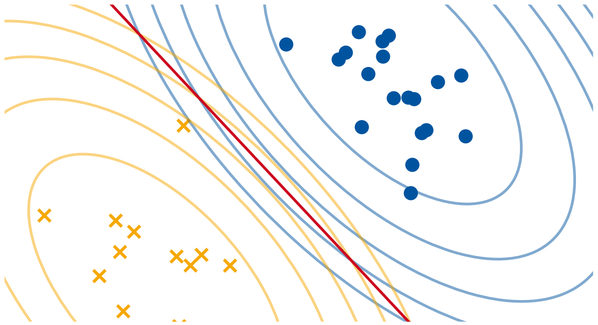
Machine Learning Topics

1. Introduction to ML
2. Probability Density Estimation
3. Linear Discriminants
4. Linear Regression
5. **Logistic Regression**
6. Support Vector Machines
7. AdaBoost
8. Neural Network Basics

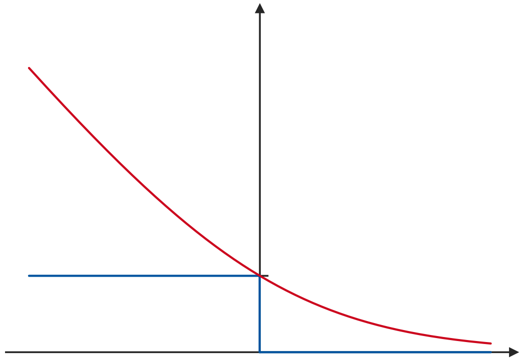


$$y(\boldsymbol{\phi}) = p(\mathcal{C}_1|\boldsymbol{\phi}) = \sigma(\mathbf{w}^{\mathsf{T}}\boldsymbol{\phi} + w_0)$$

Logistic Regression Formulation

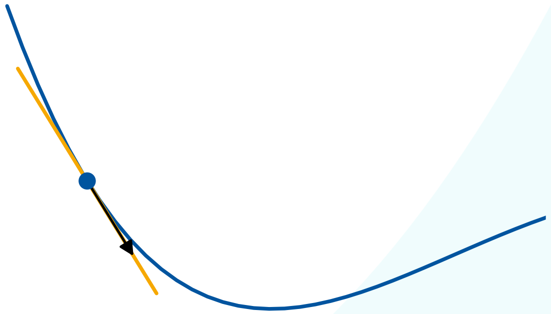


Parameter Efficiency



$$E(\mathbf{w}) = -\ln p(\mathbf{t}|\mathbf{w})$$

Cross-Entropy Error

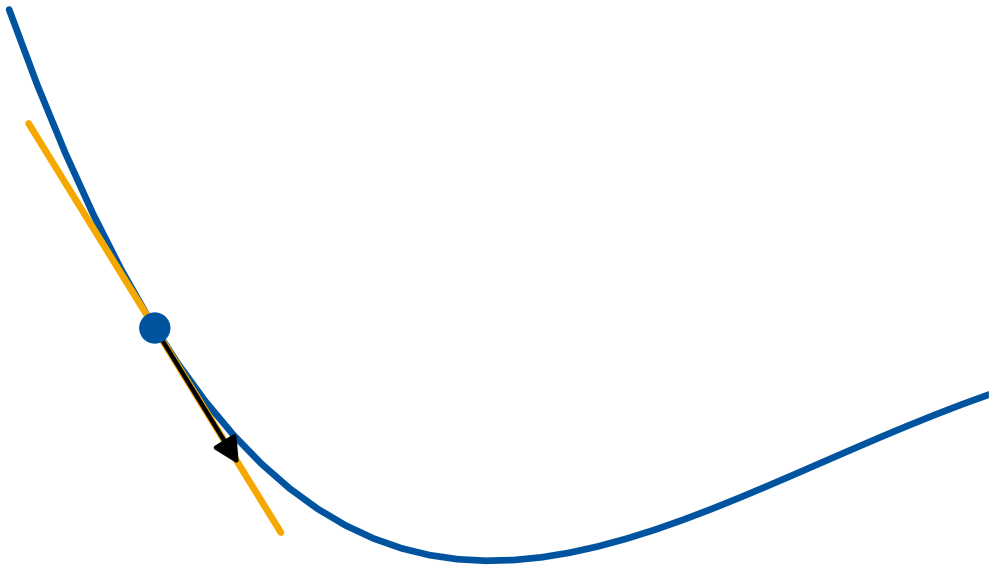


$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta\nabla E(\mathbf{w})$$

Iterative Optimization

Logistic Regression

1. Logistic Regression Formulation
2. Motivation and Background
3. **Iterative Optimization**
4. First-Order Gradient Descent
5. Second-Order Gradient Descent
6. Error Function Analysis

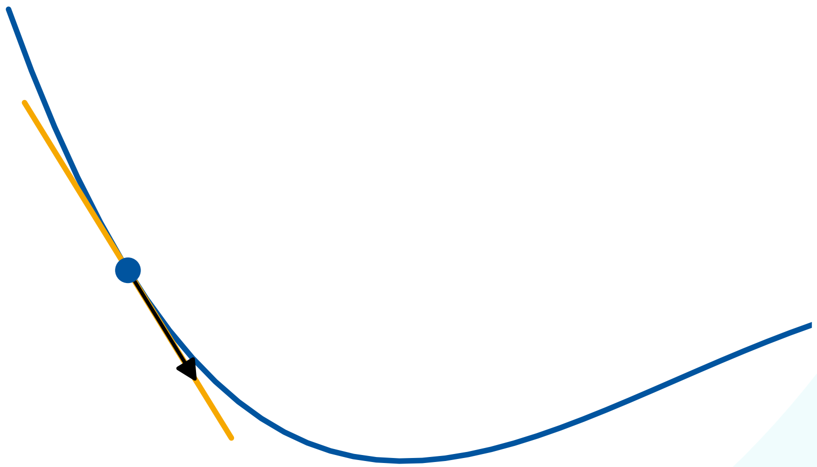


Recap: Iterative Optimization

- In general, generalized linear discriminants with nonlinear activation and/or basis functions can no longer be optimized in closed form.

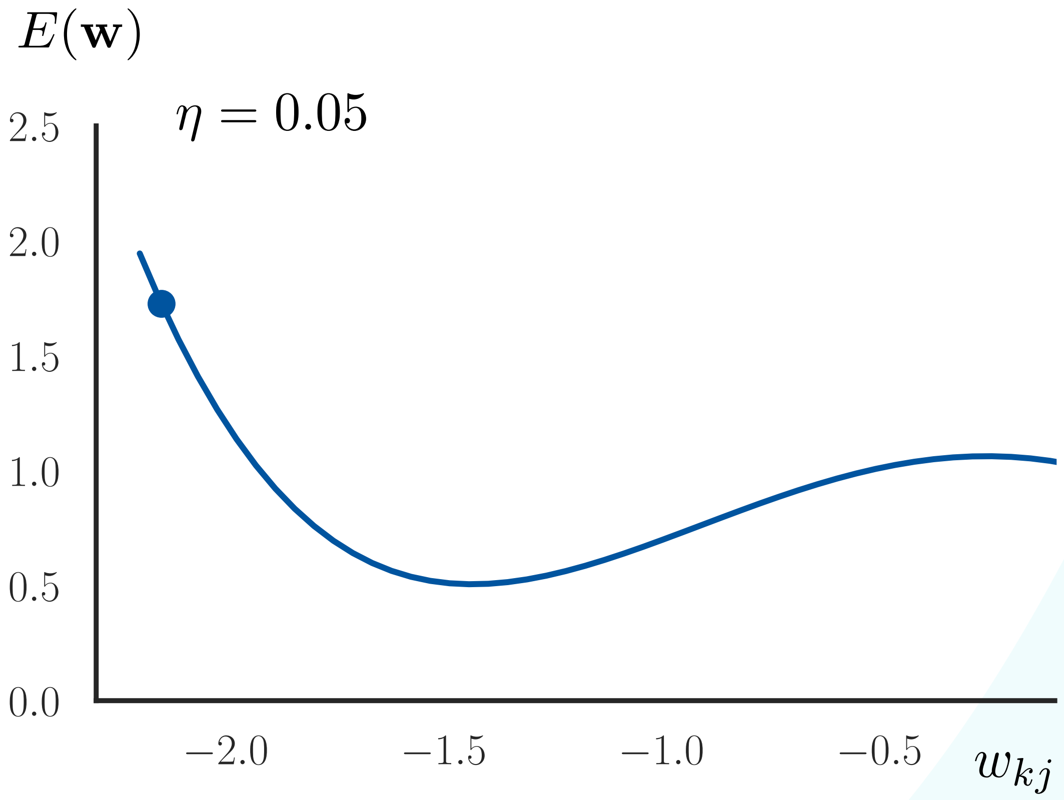
$$y_k(\mathbf{x}) = g\left(\sum_{j=0}^{M} w_{kj}\phi_j(\mathbf{x})\right) = g\left(\mathbf{w}^{\mathsf{T}}\phi(\mathbf{x})\right)$$

- Instead, we use iterative optimization schemes.
- Here: Gradient Descent.
  - Start with initial guess for parameter values.
  - Move towards a minimum of the error function by following the direction of steepest descent.
  - Iterate until convergence



Idea: Gradient Descent

- Start with an initial guess of parameter values $w_{kj}^{(0)}$.

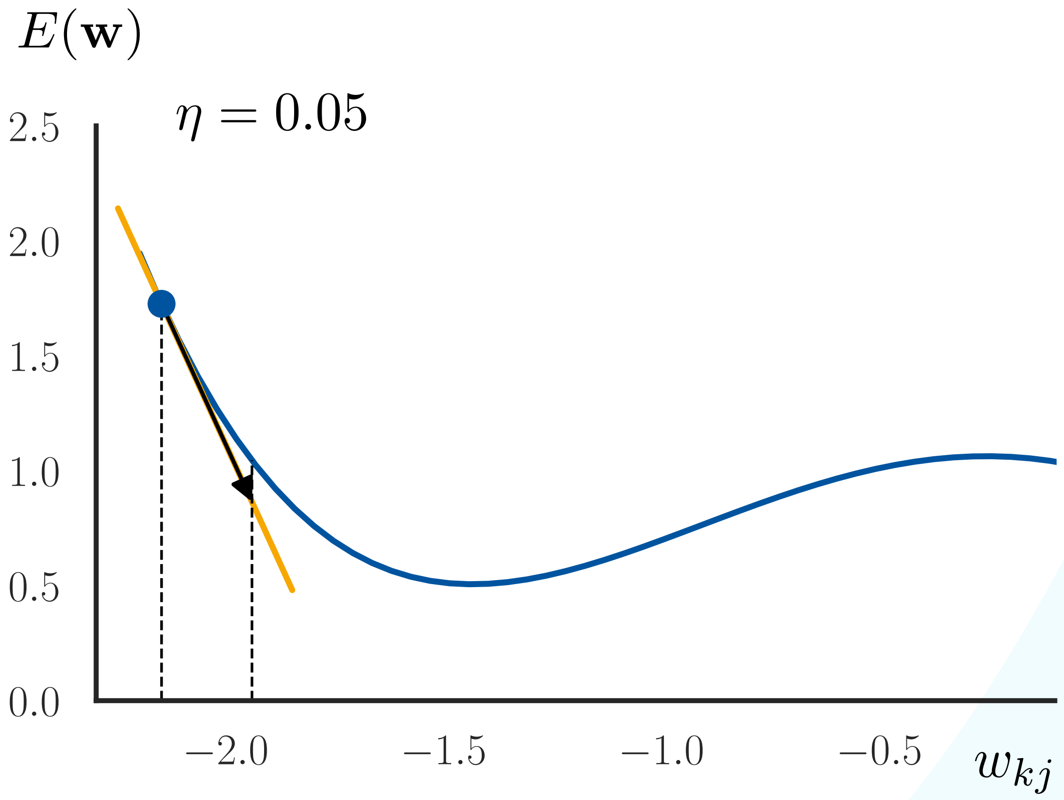


Idea: Gradient Descent

- Start with an initial guess of parameter values $w_{kj}^{(0)}$.
- Follow the gradient to move to a (local) minimum:

$$w_{kj}^{(\tau+1)} = w_{kj}^{(\tau)} - \eta \left. \frac{\partial E(\mathbf{w})}{\partial w_{kj}} \right|_{\mathbf{w}^{(\tau)}}$$

- η is called the learning rate.
- This corresponds to a 1st-order Taylor expansion.
  - I.e., we approximate the error function by its tangent plane around the current point $\mathbf{w}^{(\tau)}$.
- Repeat this procedure for a number of steps.

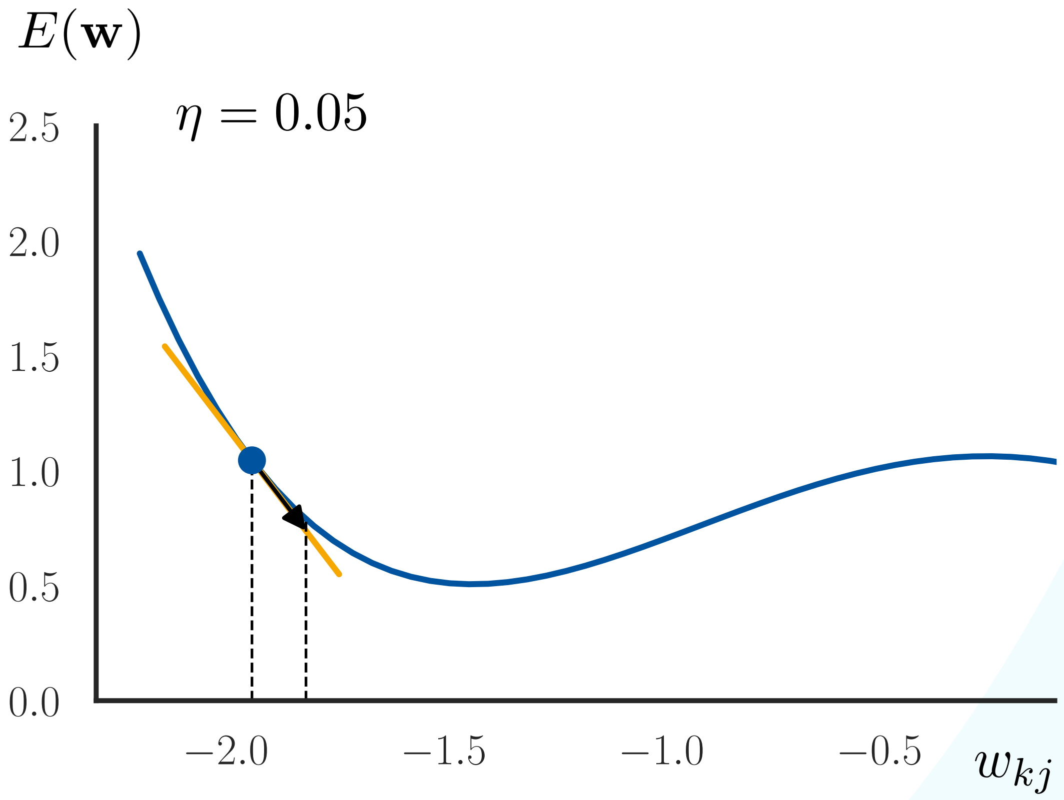


Idea: Gradient Descent

- Start with an initial guess of parameter values $w_{kj}^{(0)}$.
- Follow the gradient to move to a (local) minimum:

$$w_{kj}^{(\tau+1)} = w_{kj}^{(\tau)} - \eta \left. \frac{\partial E(\mathbf{w})}{\partial w_{kj}} \right|_{\mathbf{w}^{(\tau)}}$$

- η is called the learning rate.
- This corresponds to a 1st-order Taylor expansion.
  - I.e., we approximate the error function by its tangent plane around the current point $\mathbf{w}^{(\tau)}$.
- Repeat this procedure for a number of steps.

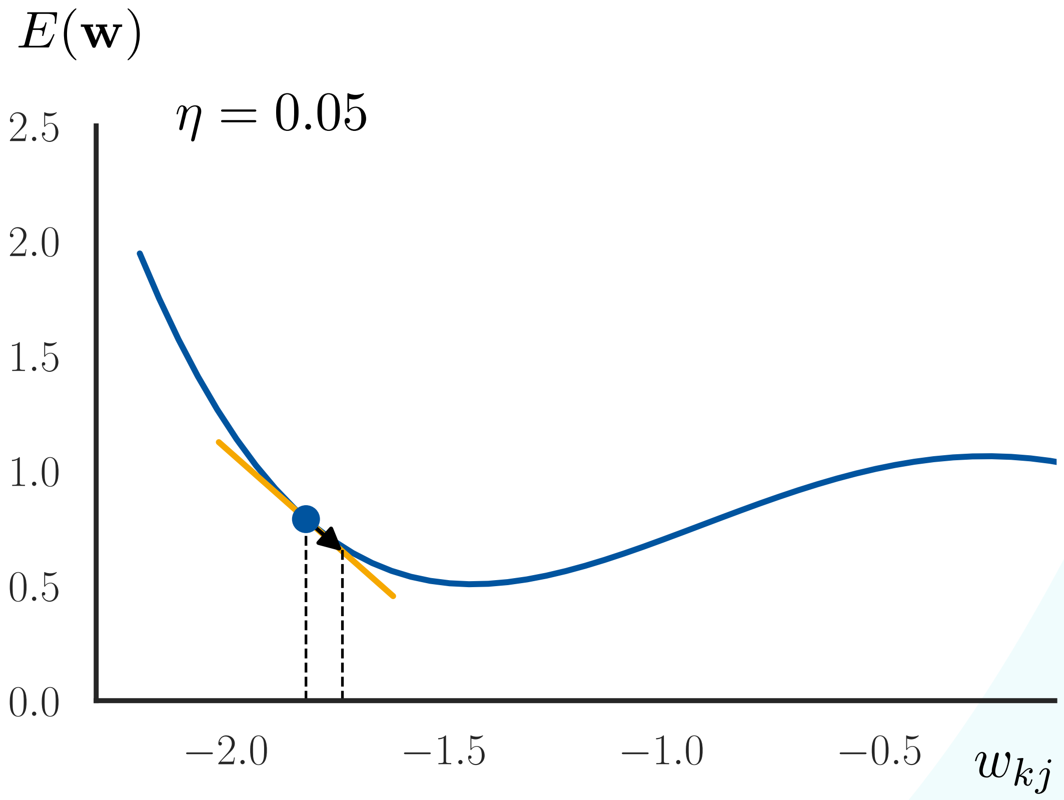


Idea: Gradient Descent

- Start with an initial guess of parameter values $w_{kj}^{(0)}$.
- Follow the gradient to move to a (local) minimum:

$$w_{kj}^{(\tau+1)} = w_{kj}^{(\tau)} - \eta \left. \frac{\partial E(\mathbf{w})}{\partial w_{kj}} \right|_{\mathbf{w}^{(\tau)}}$$

- η is called the learning rate.
- This corresponds to a 1st-order Taylor expansion.
  - I.e., we approximate the error function by its tangent plane around the current point $\mathbf{w}^{(\tau)}$.
- Repeat this procedure for a number of steps.

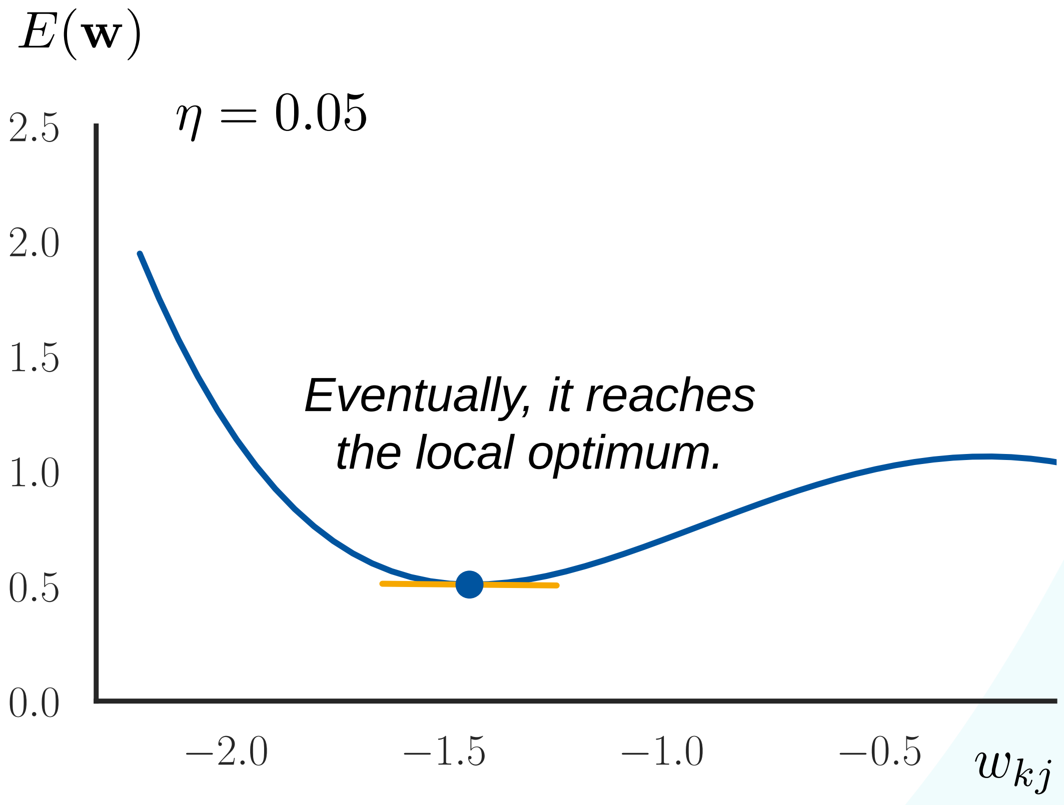


Idea: Gradient Descent

- Start with an initial guess of parameter values $w_{kj}^{(0)}$.
- Follow the gradient to move to a (local) minimum:

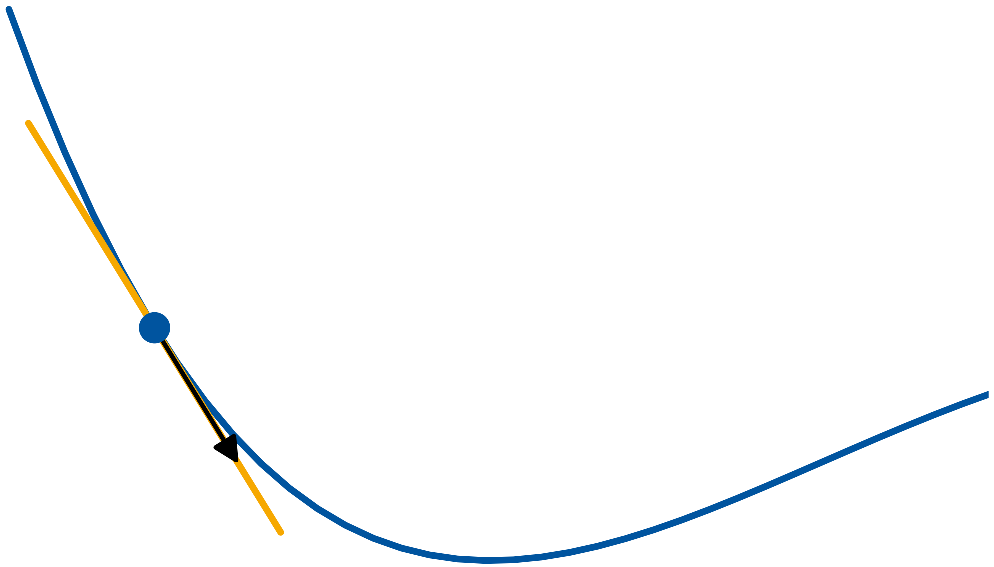
$$w_{kj}^{(\tau+1)} = w_{kj}^{(\tau)} - \eta \left.\frac{\partial E(\mathbf{w})}{\partial w_{kj}}\right|_{\mathbf{w}^{(\tau)}}$$

- η is called the learning rate.
- This corresponds to a 1st-order Taylor expansion.
  - I.e., we approximate the error function by its tangent plane around the current point $\mathbf{w}^{(\tau)}$.
- Repeat this procedure for a number of steps.



Logistic Regression

1. Logistic Regression Formulation
2. Motivation and Background
3. Iterative Optimization
4. **First-Order Gradient Descent**
5. Second-Order Gradient Descent
6. Error Function Analysis



First-order Optimization

- Logistic regression uses the binary cross-entropy error:

$$E(\mathbf{w}) = -\sum_{n=1}^{N} \big(t_n \ln y(\mathbf{x}_n; \mathbf{w}) + (1 - t_n) \ln(1 - y(\mathbf{x}_n; \mathbf{w}))\big)$$

- Properties
  - Convex function, so it has a unique minimum
  - But no closed-form solution

- We need to use iterative methods for optimization
  - Let’s try (first-order) gradient descent:

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla E(\mathbf{w})$$

Gradient of the Cross-Entropy Error

$$E(\mathbf{w}) = -\sum_{n=1}^{N} (t_n \ln y_n + (1 - t_n) \ln(1 - y_n))$$

$$y_n = y(\mathbf{x}_n; \mathbf{w})$$

$$\nabla E(\mathbf{w}) = -\sum_{n=1}^{N} \left(t_n \frac{\frac{\partial}{\partial \mathbf{w}} y_n}{y_n} + (1 - t_n) \frac{\frac{\partial}{\partial \mathbf{w}} (1 - y_n)}{(1 - y_n)} \right)$$

$$= -\sum_{n=1}^{N} \left(t_n \frac{\cancel{y_n}(1 - y_n)}{\cancel{y_n}} \phi_n + (1 - t_n) \frac{y_n \cancel{(1 - y_n)}}{\cancel{(1 - y_n)}} \phi_n \right)$$

$$= -\sum_{n=1}^{N} ((t_n - \cancel{t_n y_n} - y_n + \cancel{t_n y_n}) \phi_n)$$

$$= \sum_{n=1}^{N} (y_n - t_n) \phi_n$$

$$\sigma'(a) = \sigma(a)(1 - \sigma(a))$$
$$\frac{\partial y_n}{\partial \mathbf{w}} = y_n (1 - y_n) \phi_n$$

$$\phi_n = \phi(\mathbf{x}_n)$$

- The gradient for logistic regression is

$$\nabla E(\mathbf{w}) = \sum_{n=1}^{N} (y_n - t_n)\boldsymbol{\phi}_n$$

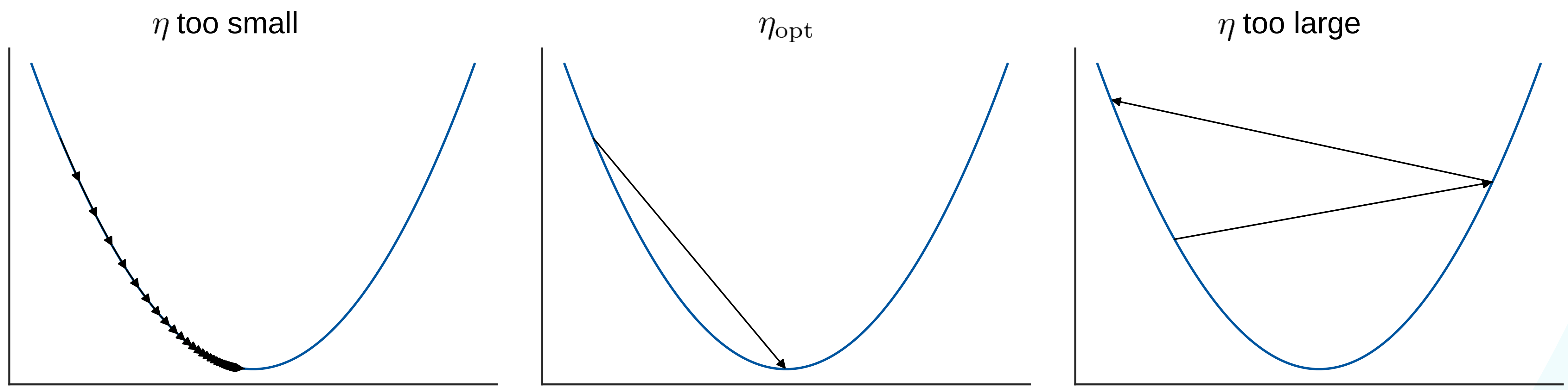
- We can plug this into gradient descent:

$$\begin{aligned} \mathbf{w}^{(\tau+1)} &= \mathbf{w}^{(\tau)} - \eta \nabla E(\mathbf{w}) \\ &= \mathbf{w}^{(\tau)} - \eta \sum_{n=1}^{N} (y_n - t_n)\boldsymbol{\phi}_n \end{aligned}$$

How should we choose the learning rate?

- This update rule is known as the Delta rule (= LMS rule)
  - *Simply feed back the input data points, weighted by the classification error.*

Effects of the learning rate

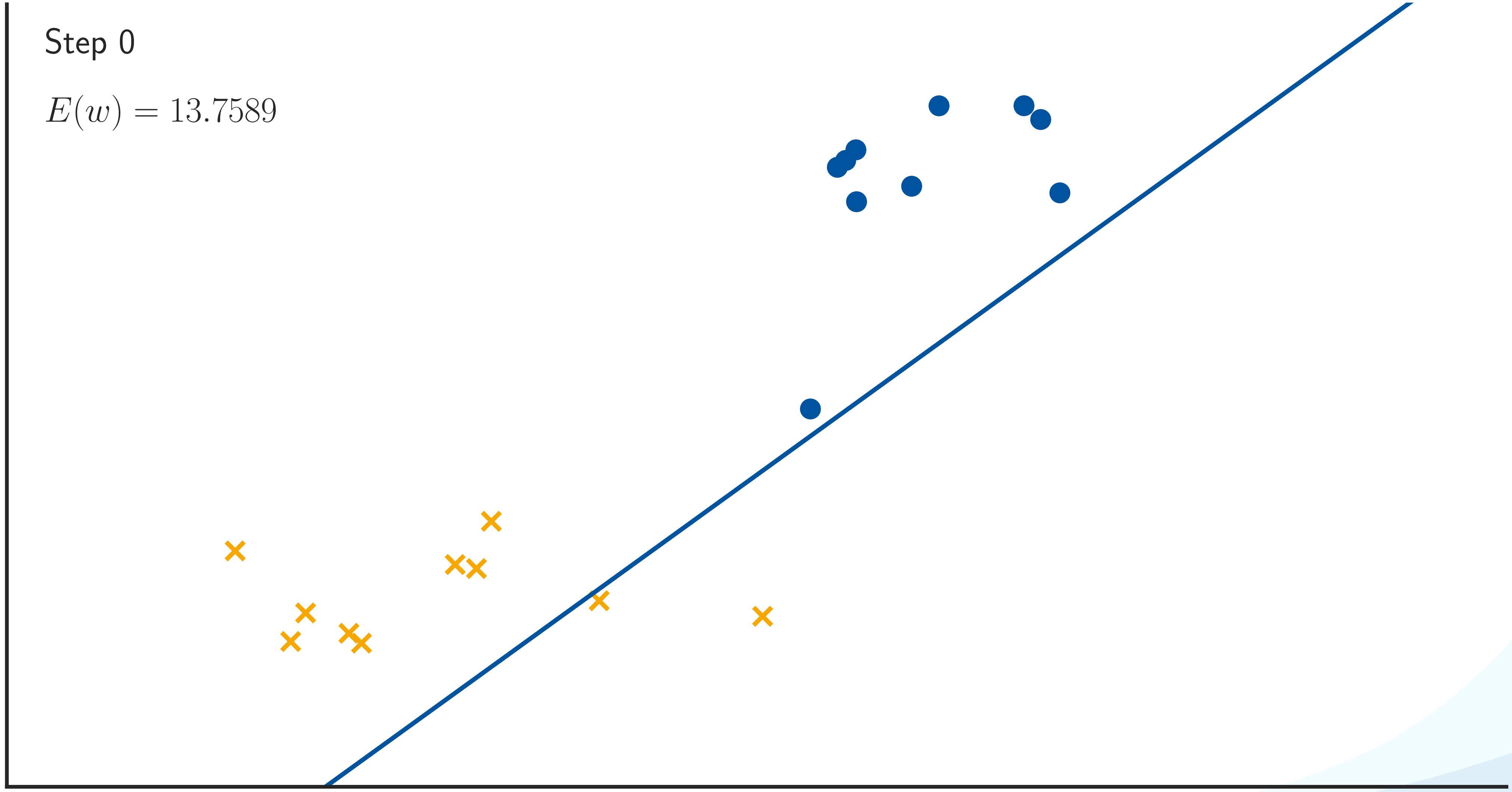


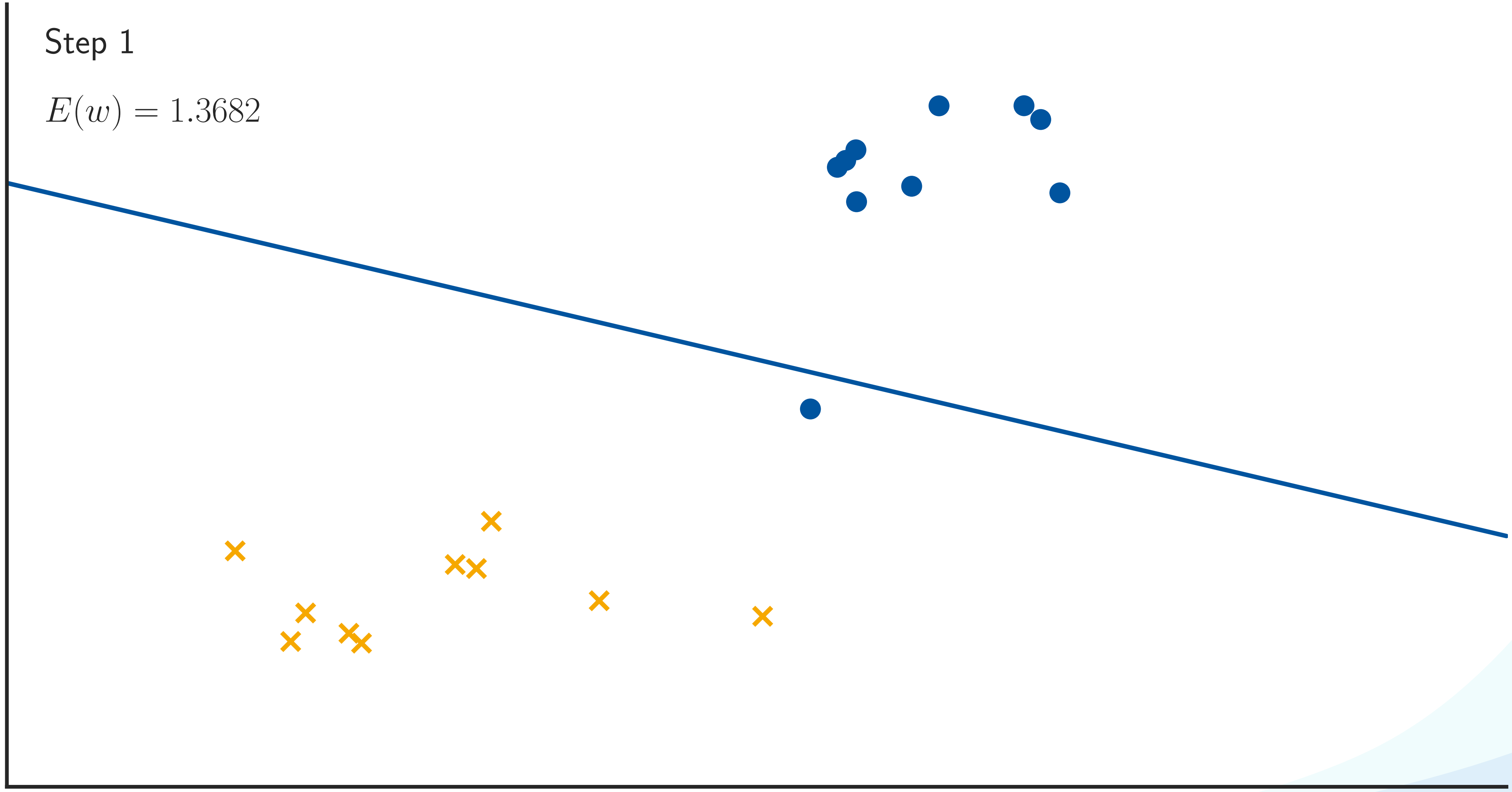
Convergence is slow

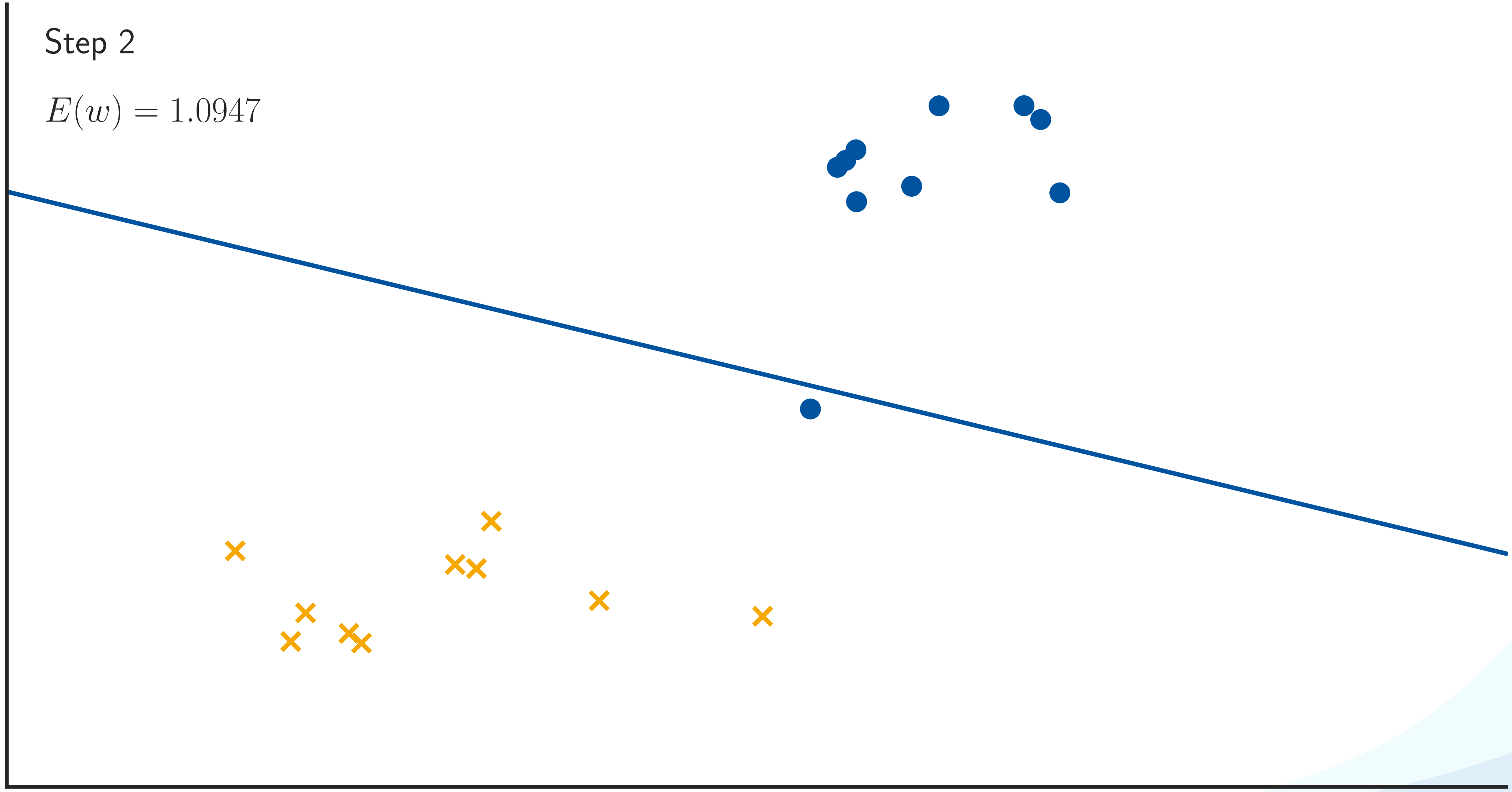
Converges ideally in a single step

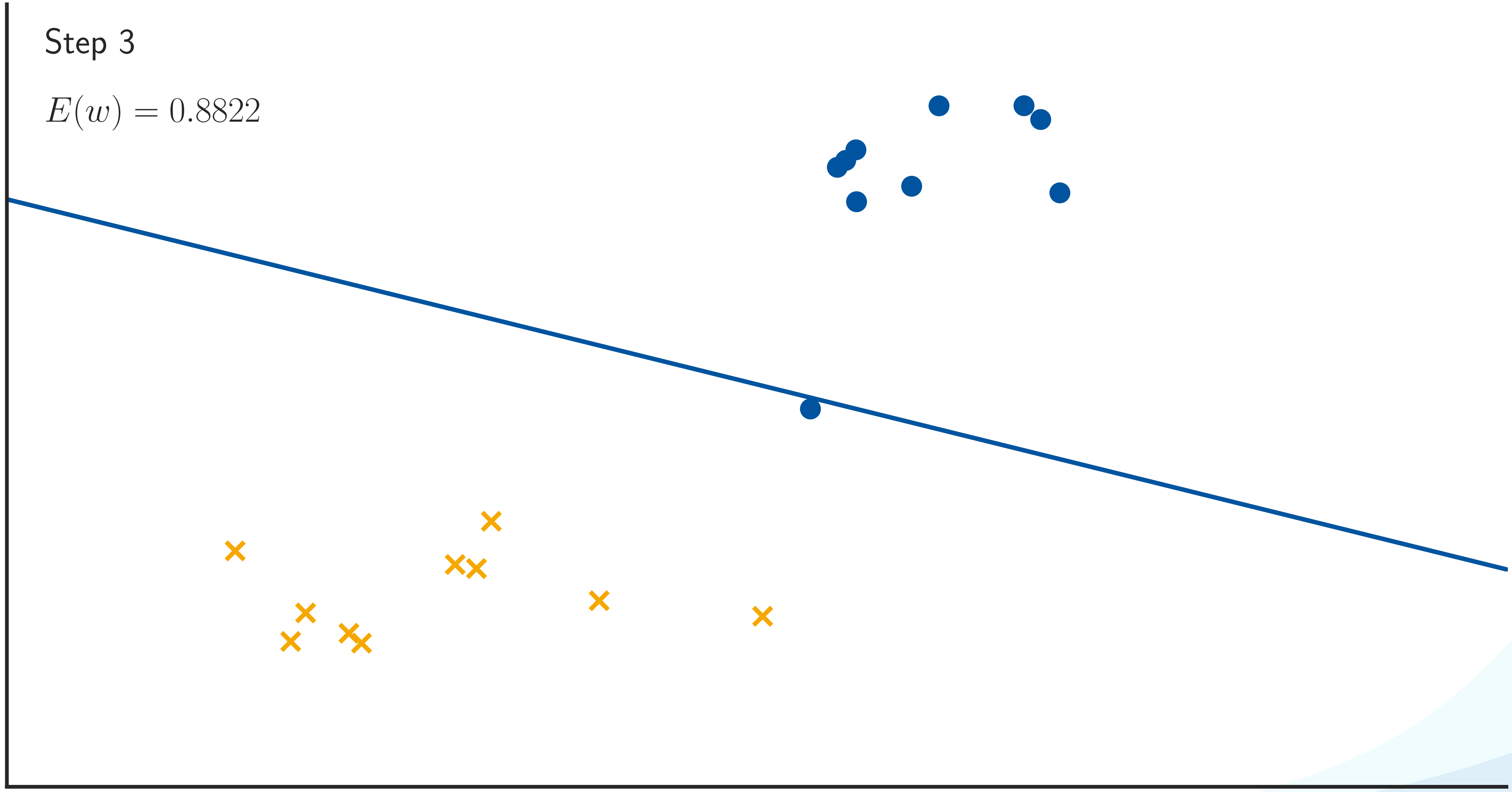
Might not converge

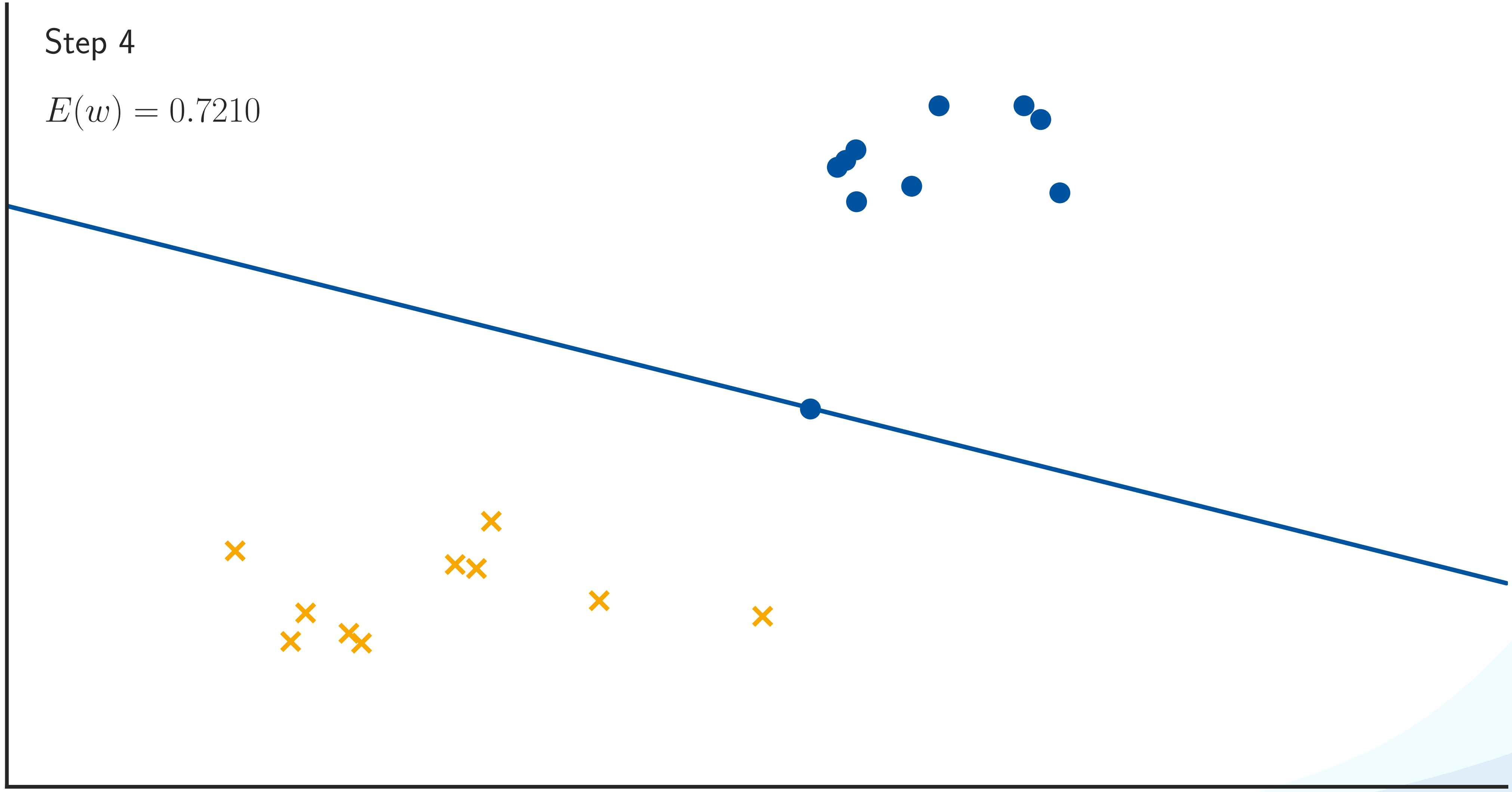
Example: Logistic Regression with Gradient Descent

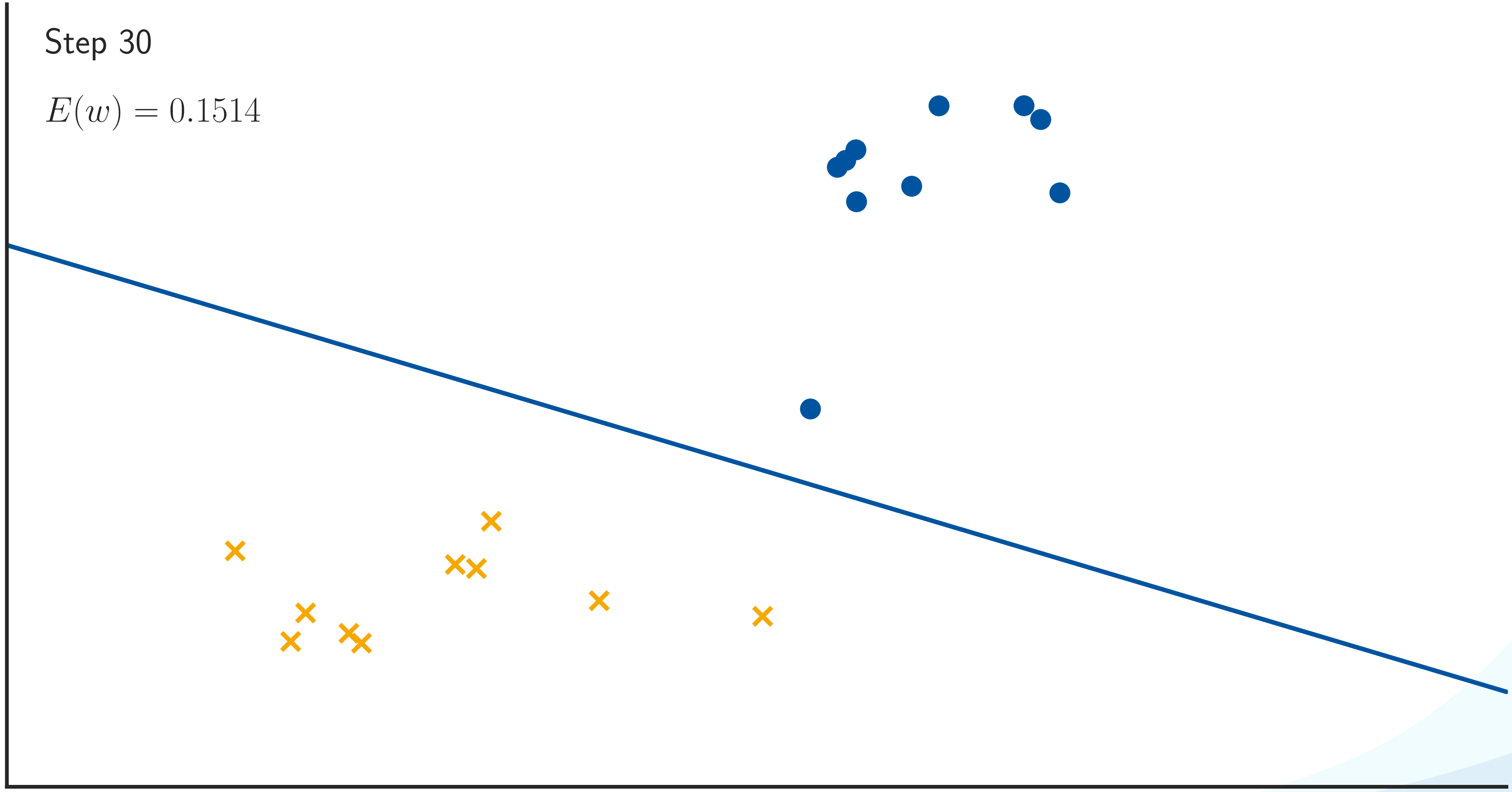


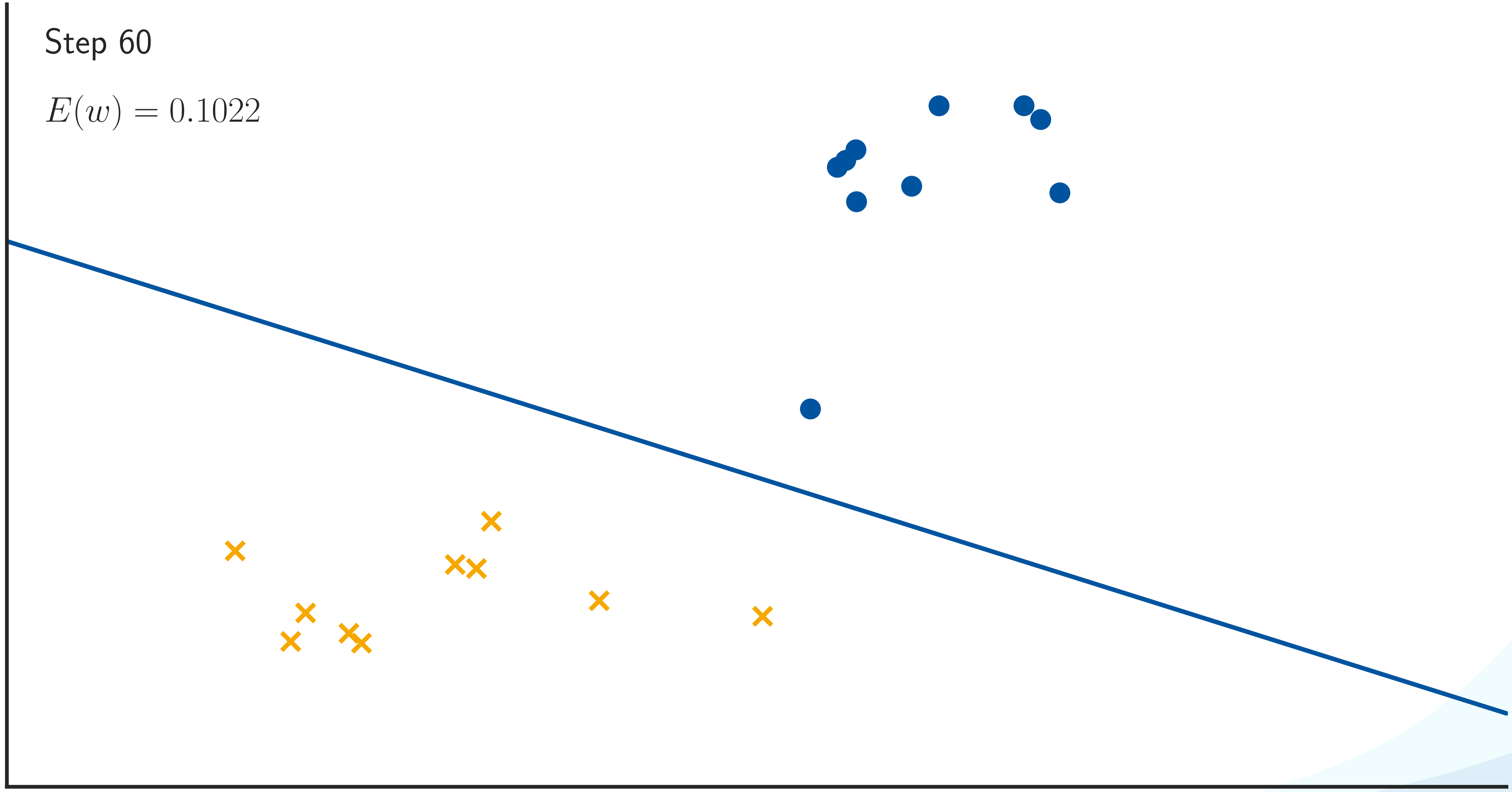

Step 1
$E(w) = 1.3682$

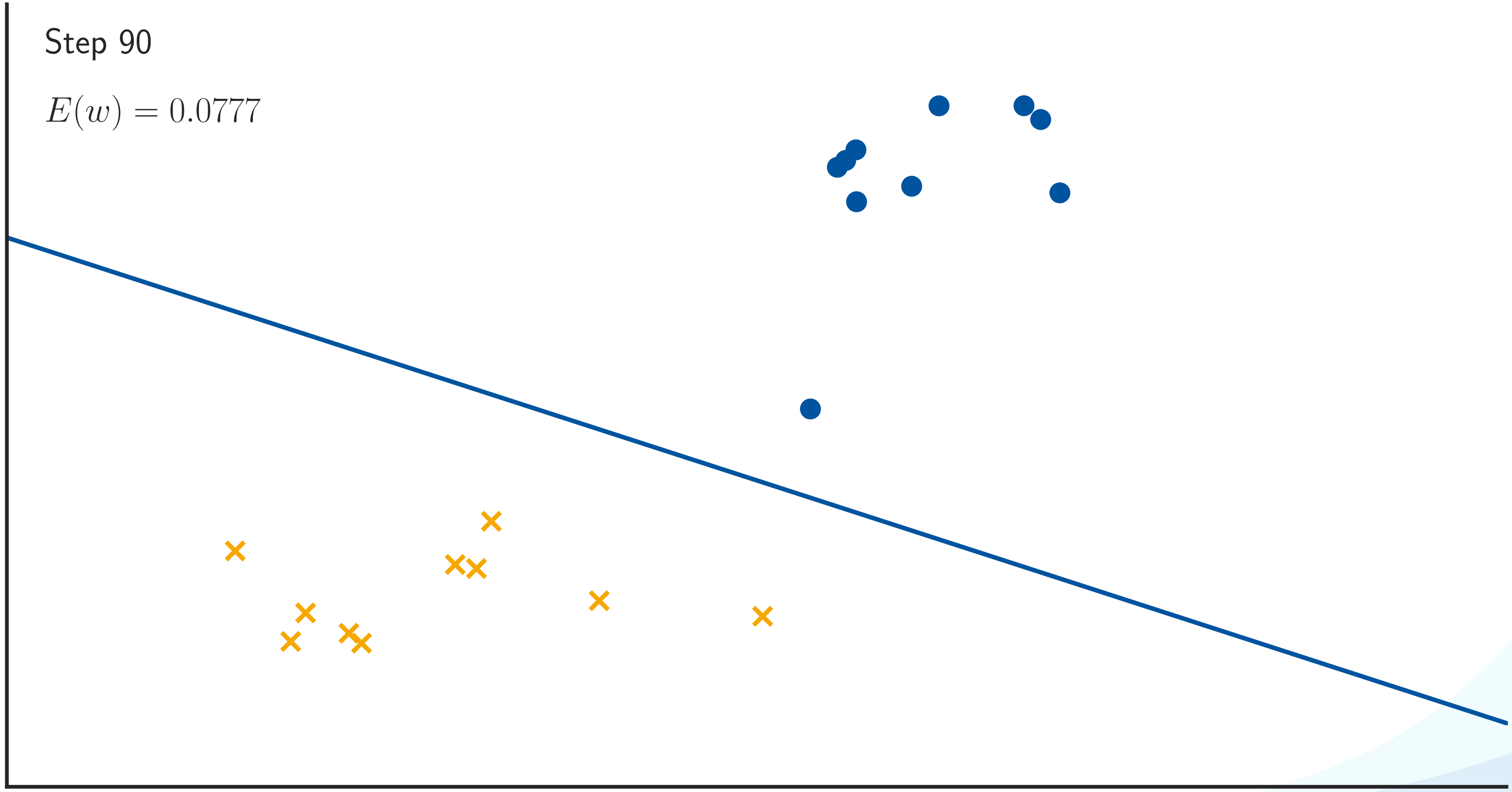

Step 2
$E(w) = 1.0947$


Step 3
$E(w) = 0.8822$


Step 4
$E(w) = 0.7210$


Step 30
$E(w) = 0.1514$


Step 60
$E(w) = 0.1022$


Step 90
$E(w) = 0.0777$

Discussion: Logistic Regression with Gradient Descent

Advantages

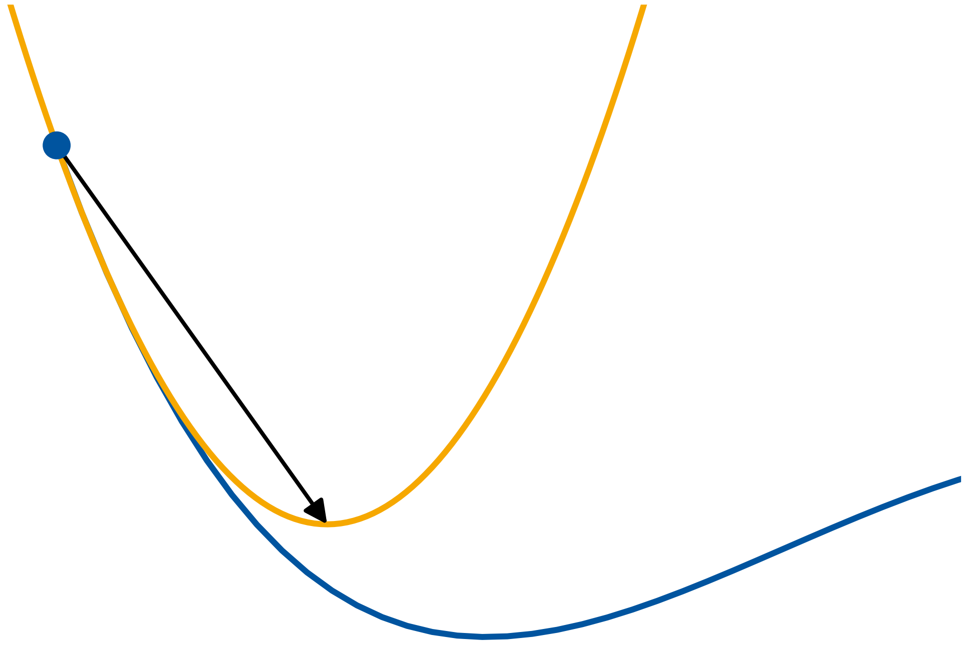
- Simple iterative optimization scheme with a familiar update rule (Delta rule).

Limitations

- Slow convergence
- Need to choose a suitable learning rate.

Logistic Regression

1. Logistic Regression Formulation
2. Motivation and Background
3. Iterative Optimization
4. First-Order Gradient Descent
5. **Second-Order Gradient Descent**
6. Error Function Analysis



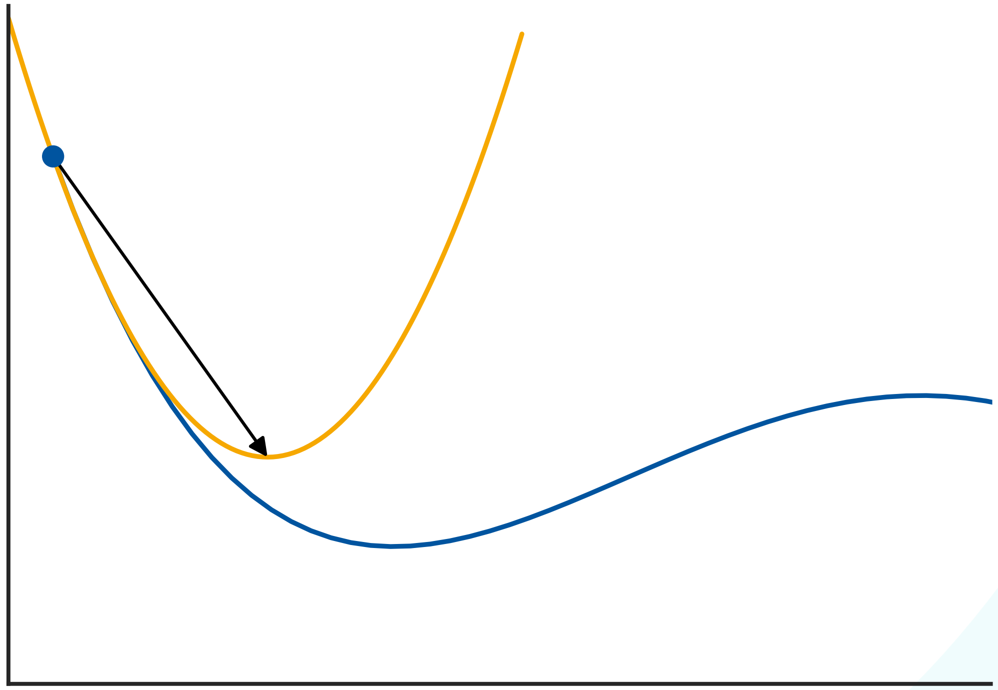
Second-Order Optimization

- So far, we have optimized the cross-entropy error with gradient descent:

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla E(\mathbf{w})$$

- This is a first-order approximation, and it heavily depends on the learning rate η.

- Instead, we can apply a second-order optimization scheme that converges faster and is independent of the learning rate.



Newton-Raphson Gradient Descent

- Second-order Newton-Raphson update scheme:

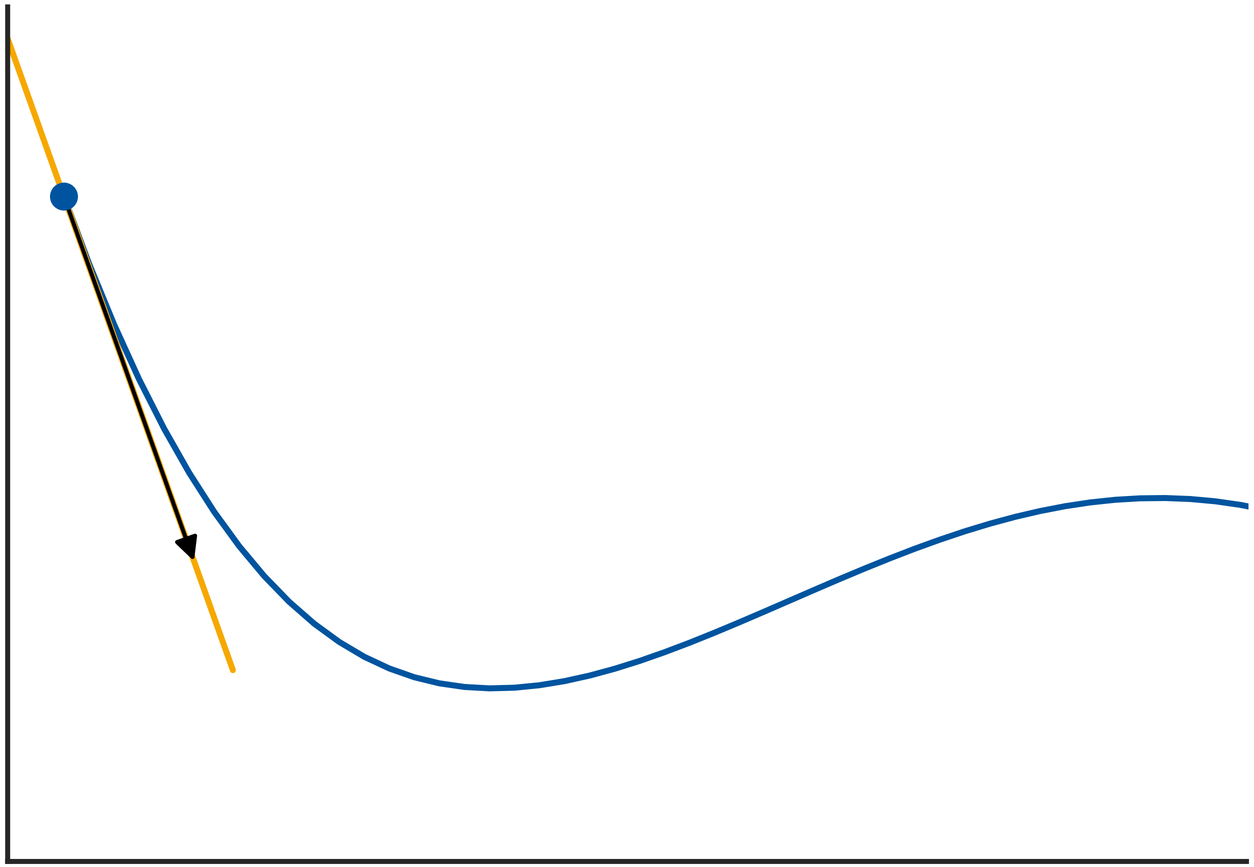
$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \mathbf{H}^{-1}\nabla E(\mathbf{w})$$

- Here, $\mathbf{H} = \nabla\nabla E(\mathbf{w})$ is the Hessian matrix, i.e., the matrix of second derivatives:

$$\mathbf{H}_{ij} = \frac{\partial^2 E(\mathbf{w})}{\partial w_i \partial w_j}$$

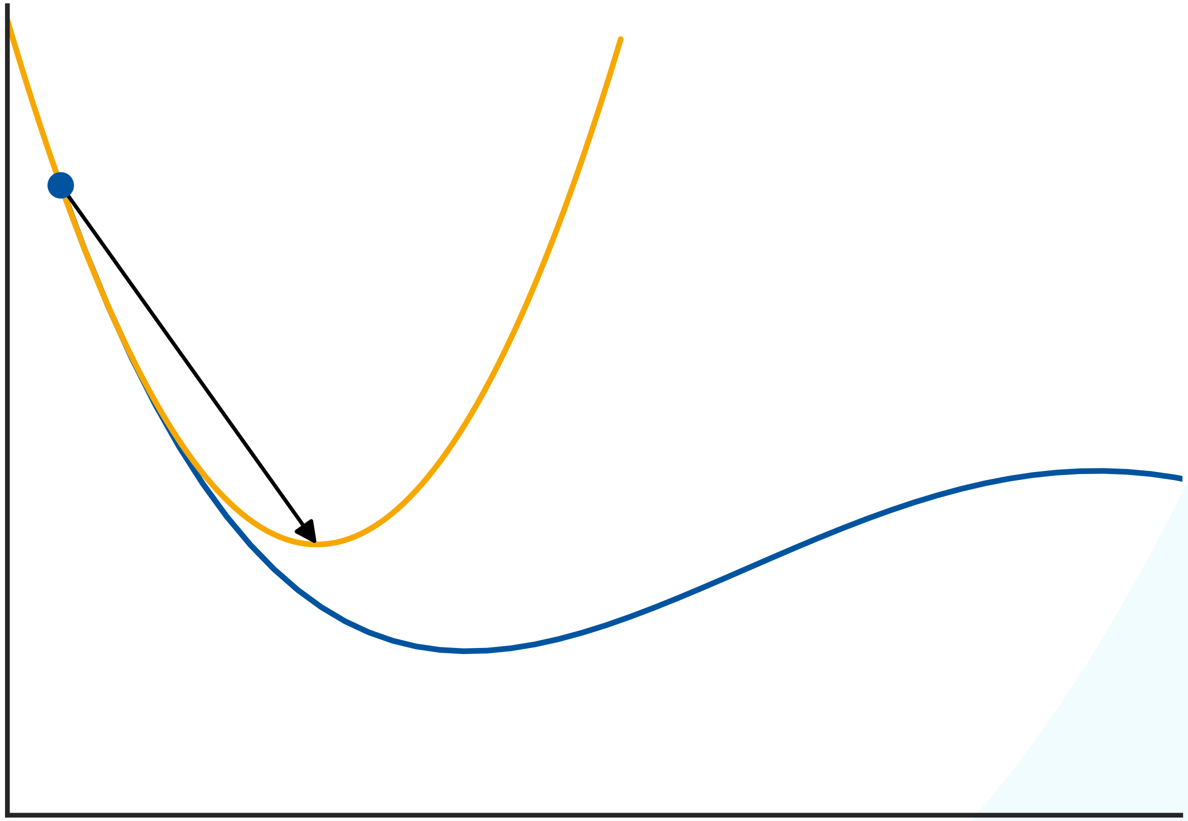
- Properties
  - Local quadratic approximation
  - Much faster convergence by taking into account the curvature of the error function.

Intuition



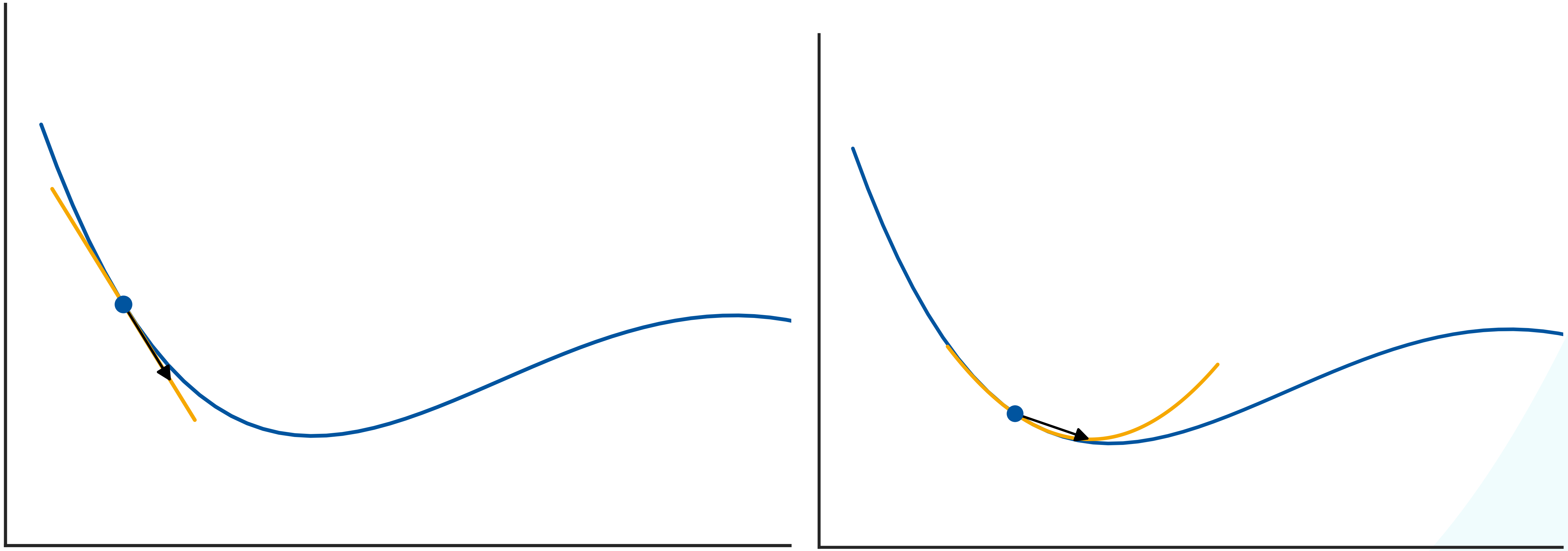
First-Order

$\eta = 0.005$



Second-Order

Intuition

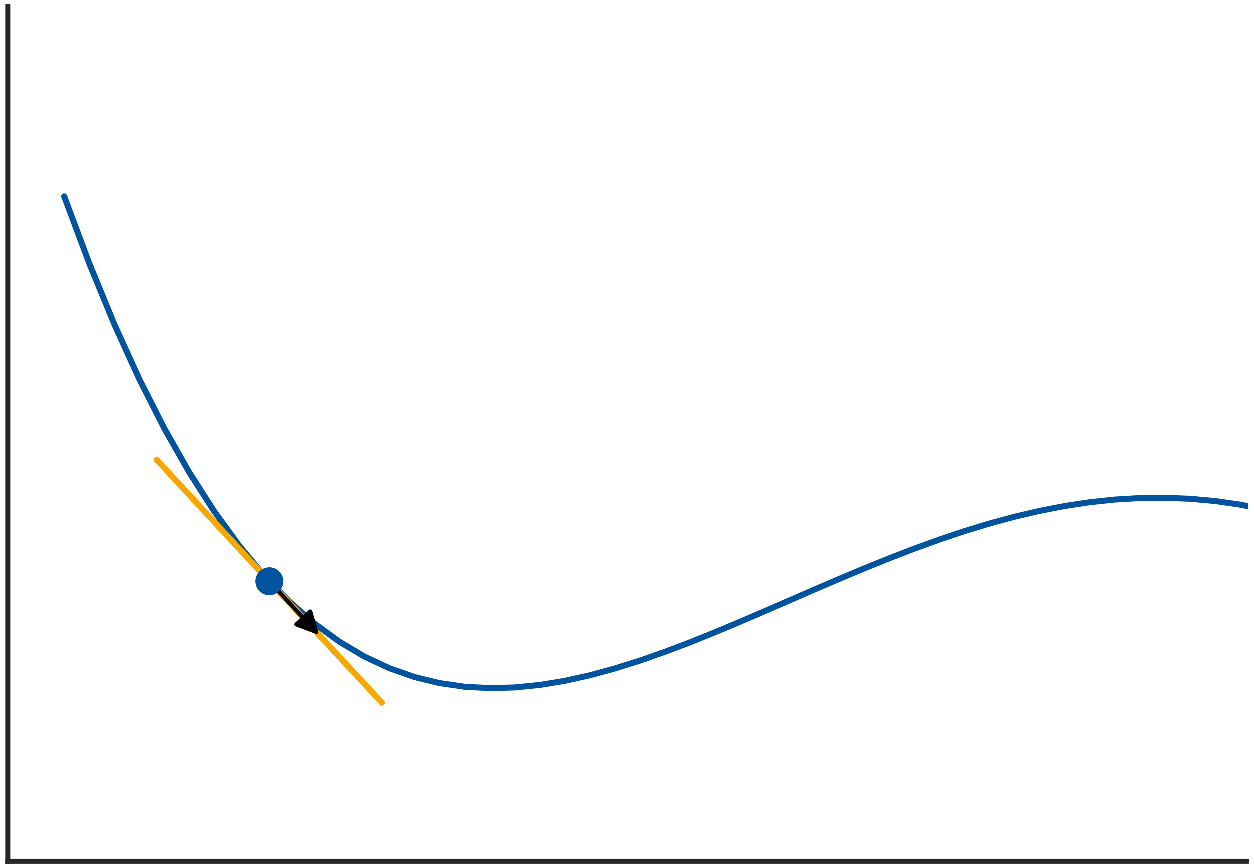


First-Order

$\eta = 0.005$

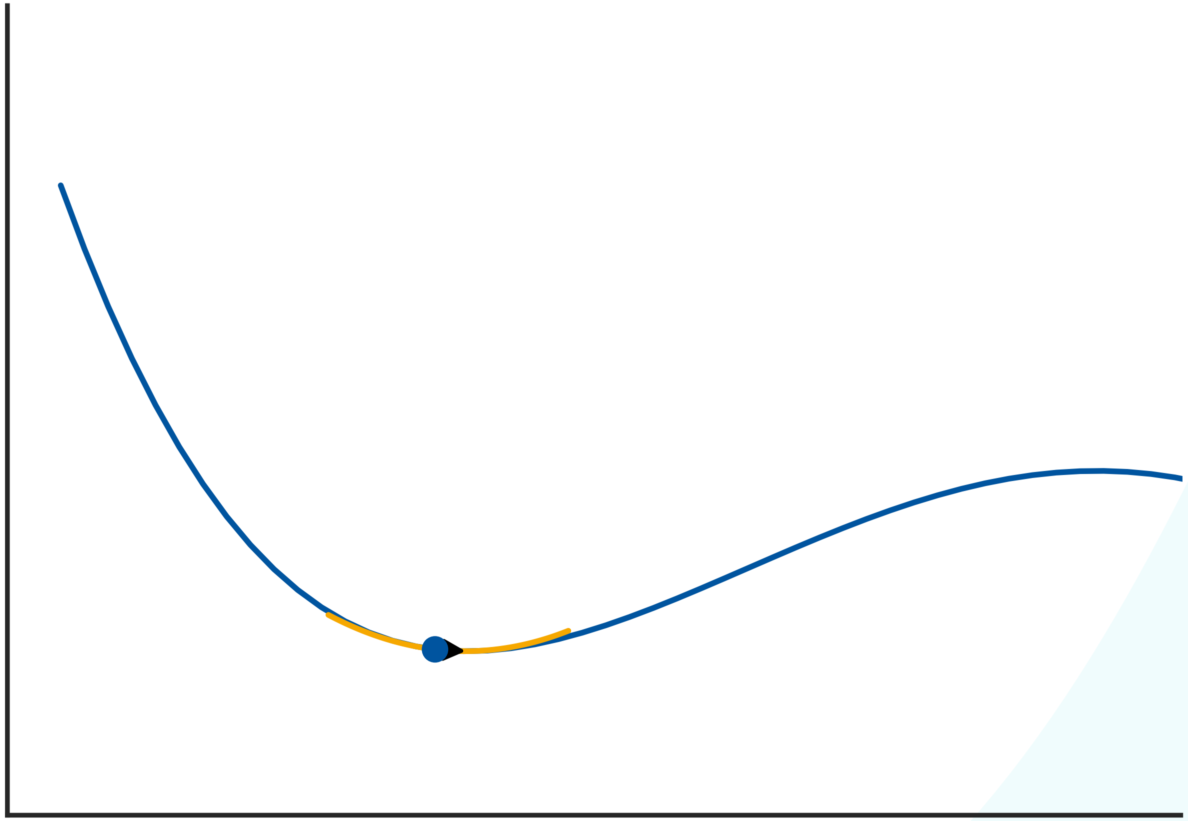
Second-Order

Intuition



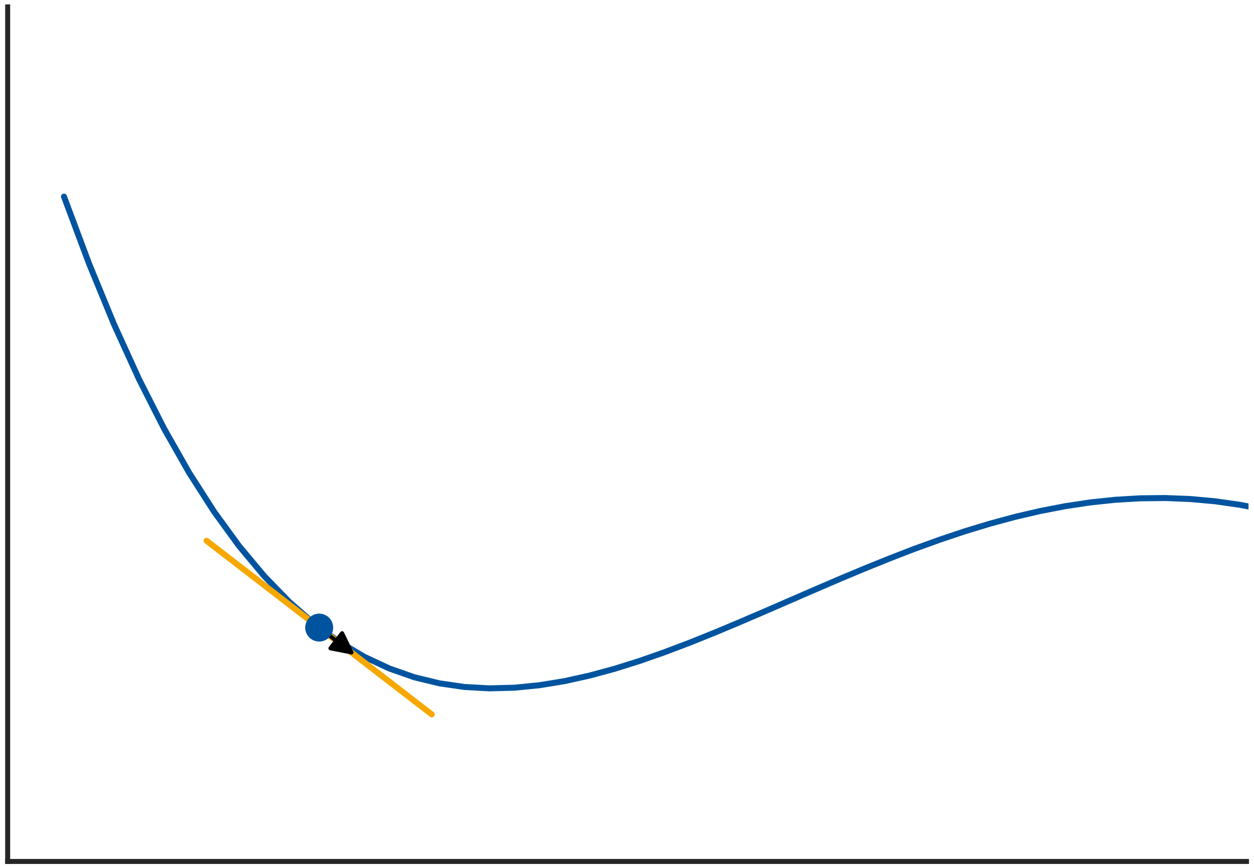
First-Order

$\eta = 0.005$



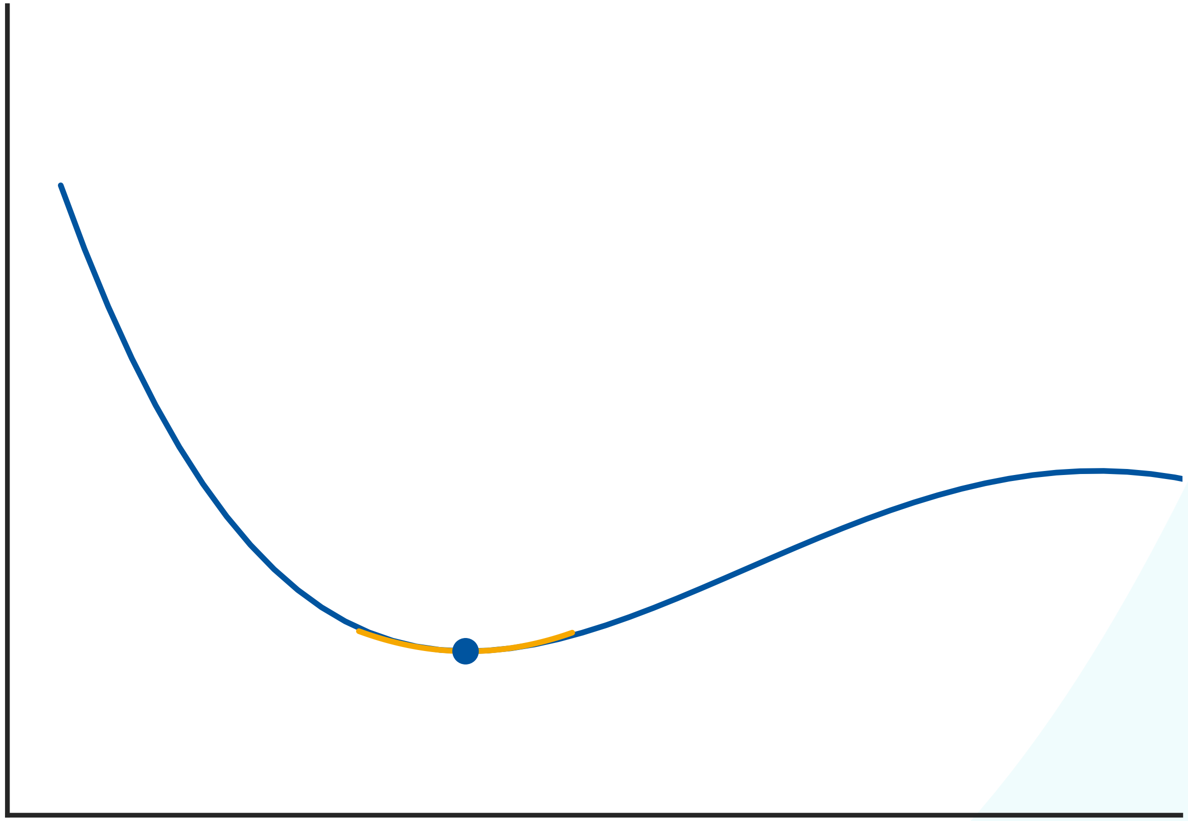
Second-Order

Intuition



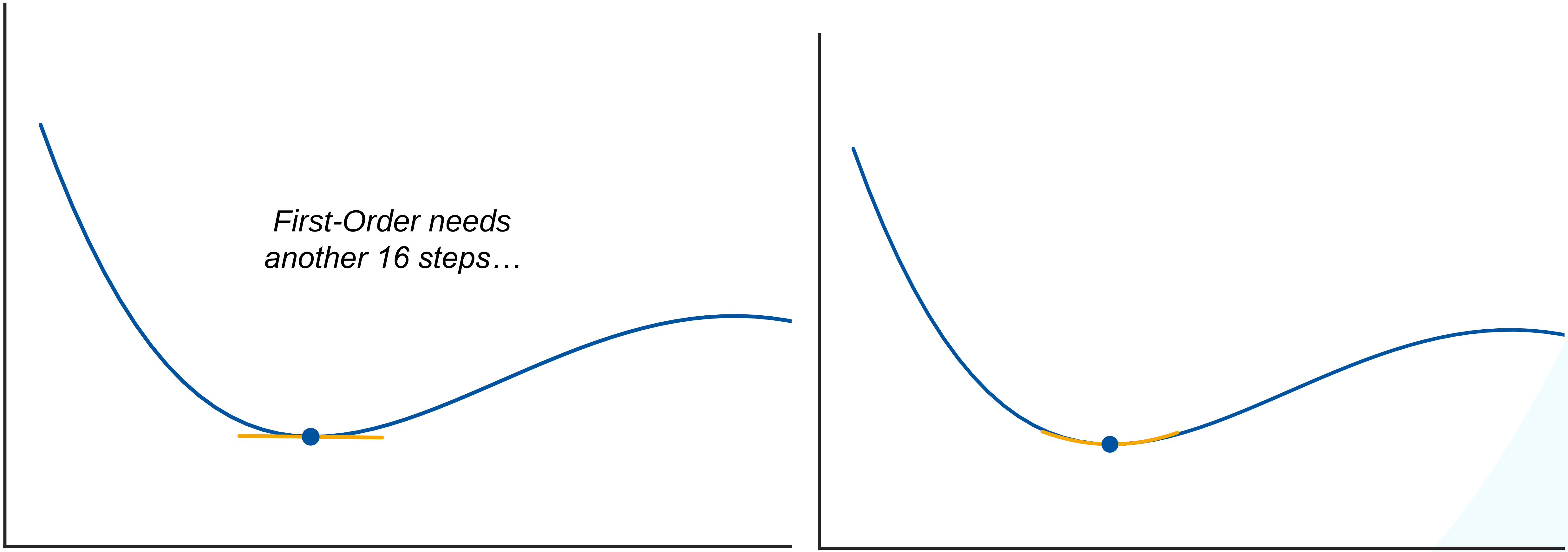
First-Order

$\eta = 0.005$



Second-Order

Intuition



First-Order
$\eta = 0.005$

Second-Order

Newton-Raphson for Least-Squares

- First, we apply it to least-squares:

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}(\mathbf{w}^\top \phi_n - t_n)^2$$

$$\nabla E(\mathbf{w}) = \sum_{n=1}^{N}(\mathbf{w}^\top \phi_n - t_n)\phi_n \quad = \boldsymbol{\Phi}^\top\boldsymbol{\Phi}\mathbf{w} - \boldsymbol{\Phi}^\top\mathbf{t}$$

$$\mathbf{H} = \nabla\nabla E(\mathbf{w}) = \sum_{n=1}^{N}\phi_n\phi_n^\top = \boldsymbol{\Phi}^\top\boldsymbol{\Phi}$$

$$\boldsymbol{\Phi} = \begin{pmatrix}\phi_1^\top \\ \vdots \\ \phi_N^\top\end{pmatrix}$$

- Resulting update scheme (normal equations):

$$\begin{aligned}\mathbf{w}^{(\tau+1)} &= \mathbf{w}^{(\tau)} - (\boldsymbol{\Phi}^\top\boldsymbol{\Phi})^{-1}(\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\mathbf{w}^{(\tau)} - \boldsymbol{\Phi}^\top\mathbf{t}) \\ &= (\boldsymbol{\Phi}^\top\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^\top\mathbf{t}\end{aligned}$$

This is the closed-form solution of the least-squares objective!

Newton-Raphson for the Cross-Entropy Error

- Now, let's try Newton-Raphson on the cross-entropy error function:

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}(t_n \ln y_n + (1-t_n)\ln(1-y_n))$$

$$\nabla E(\mathbf{w}) = \sum_{n=1}^{N}(y_n - t_n)\boldsymbol{\phi}_n \quad = \boldsymbol{\Phi}^\top(\mathbf{y}-\mathbf{t})$$

$$\mathbf{H} = \nabla\nabla E(\mathbf{w}) = \sum_{n=1}^{N} y_n(1-y_n)\boldsymbol{\phi}_n\boldsymbol{\phi}_n^\top = \boldsymbol{\Phi}^\top\mathbf{R}\boldsymbol{\Phi}$$

$$\sigma'(a) = \sigma(a)(1-\sigma(a))$$

$$\frac{\partial y_n}{\partial \mathbf{w}} = y_n(1-y_n)\boldsymbol{\phi}_n$$

- where $\mathbf{R} \in \mathbb{R}^{N\times N}$ is an $N \times N$ diagonal matrix with $R_{nn} = y_n(1-y_n)$.
- The Hessian now depends on $\mathbf{w}$ through the weighting matrix $\mathbf{R}$.

Iteratively Reweighted Least Squares (IRLS)

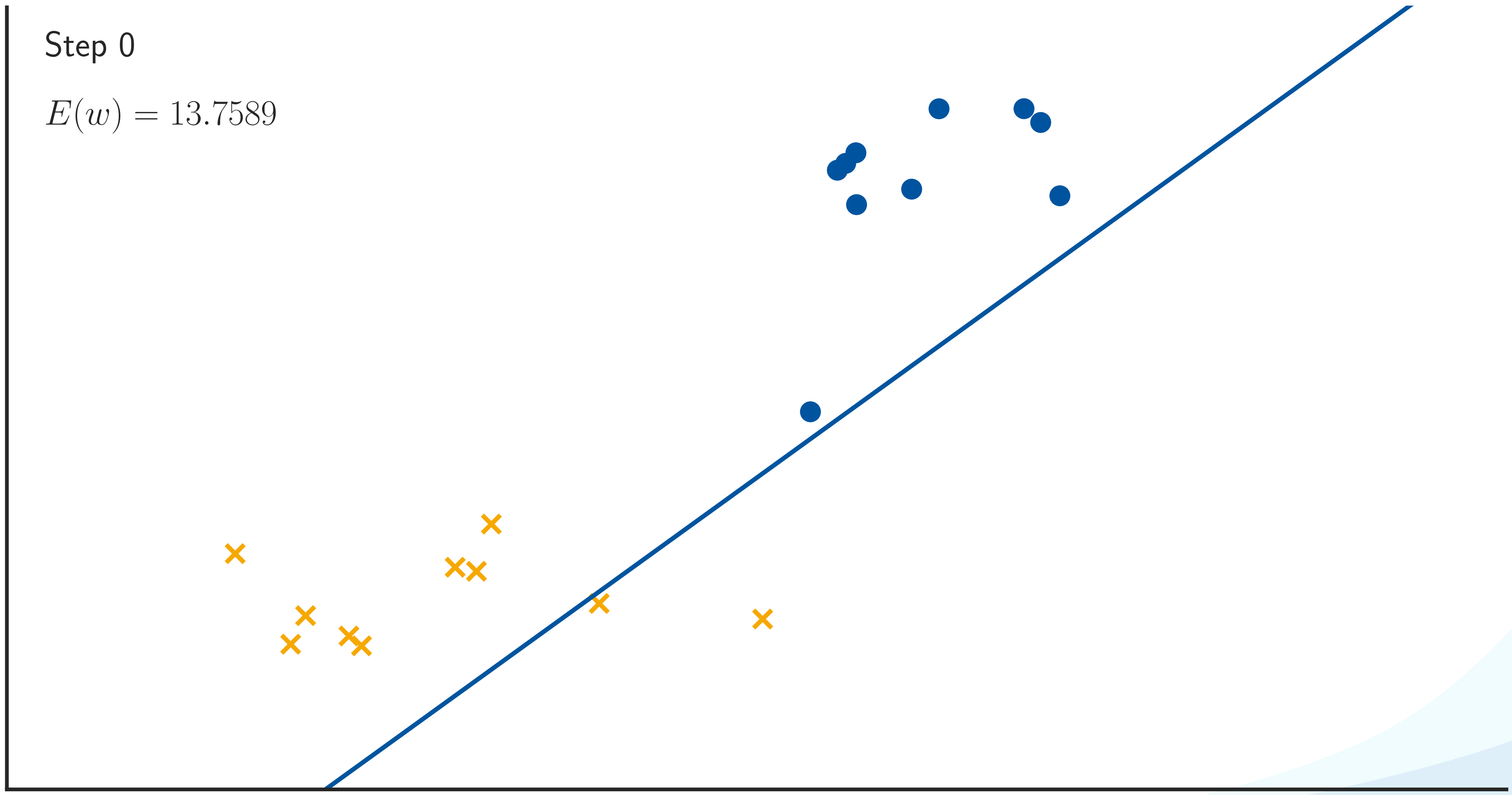
- Update equations:

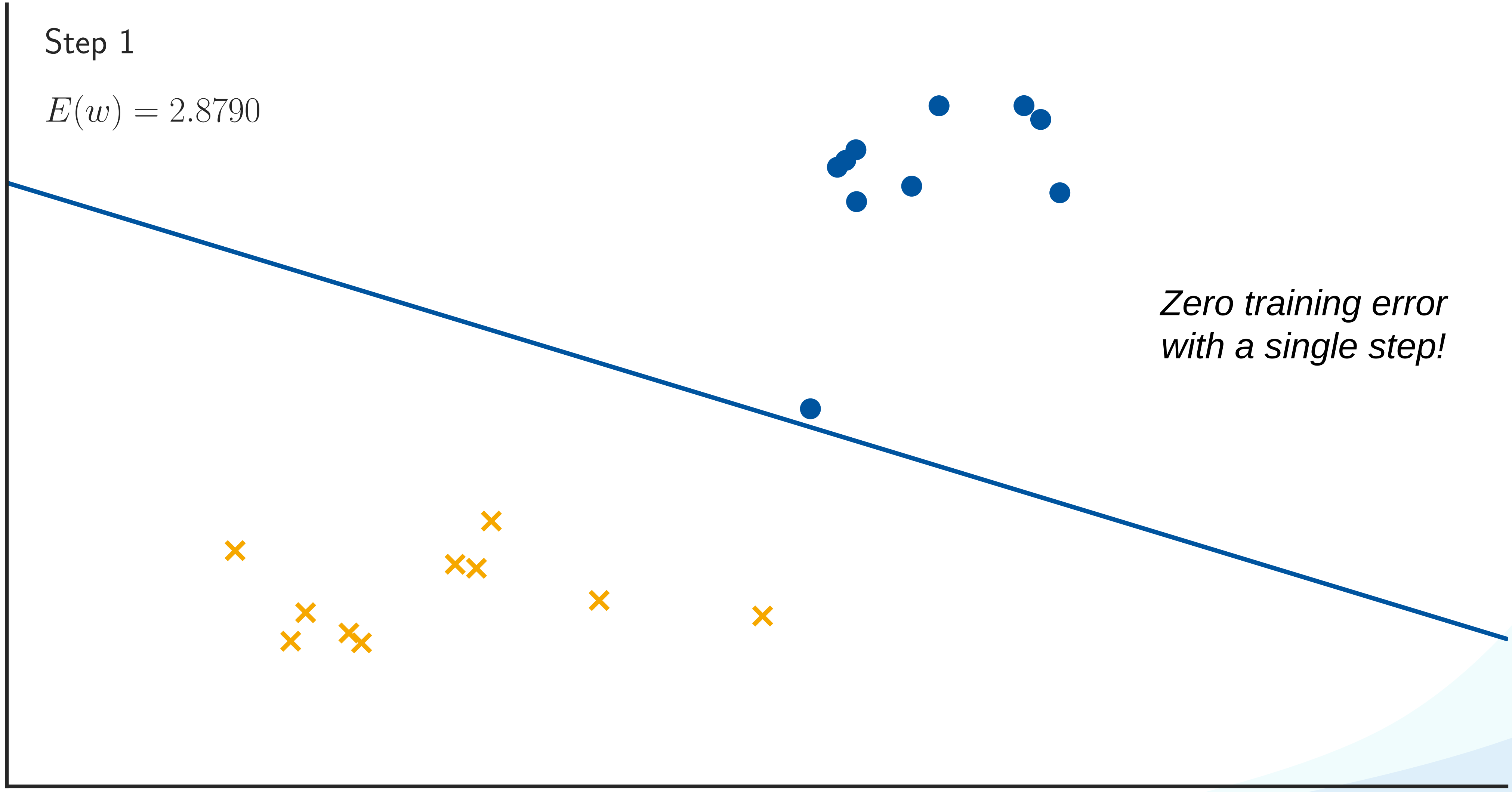
$$\begin{aligned}\mathbf{w}^{(\tau+1)} &= \mathbf{w}^{(\tau)} - (\mathbf{\Phi}^{\mathsf{T}}\mathbf{R}\mathbf{\Phi})^{-1}\mathbf{\Phi}^{\mathsf{T}}(\mathbf{y}-\mathbf{t}) \\ &= (\mathbf{\Phi}^{\mathsf{T}}\mathbf{R}\mathbf{\Phi})^{-1}\left(\mathbf{\Phi}^{\mathsf{T}}\mathbf{R}\mathbf{\Phi}\mathbf{w}^{(\tau)} - \mathbf{\Phi}^{\mathsf{T}}(\mathbf{y}-\mathbf{t})\right) \\ &= (\mathbf{\Phi}^{\mathsf{T}}\mathbf{R}\mathbf{\Phi})^{-1}\mathbf{\Phi}^{\mathsf{T}}\mathbf{R}\mathbf{z}\end{aligned}$$

$$\text{with} \quad \mathbf{z} = \mathbf{\Phi}\mathbf{w}^{(\tau)} - \mathbf{R}^{-1}(\mathbf{y}-\mathbf{t})$$

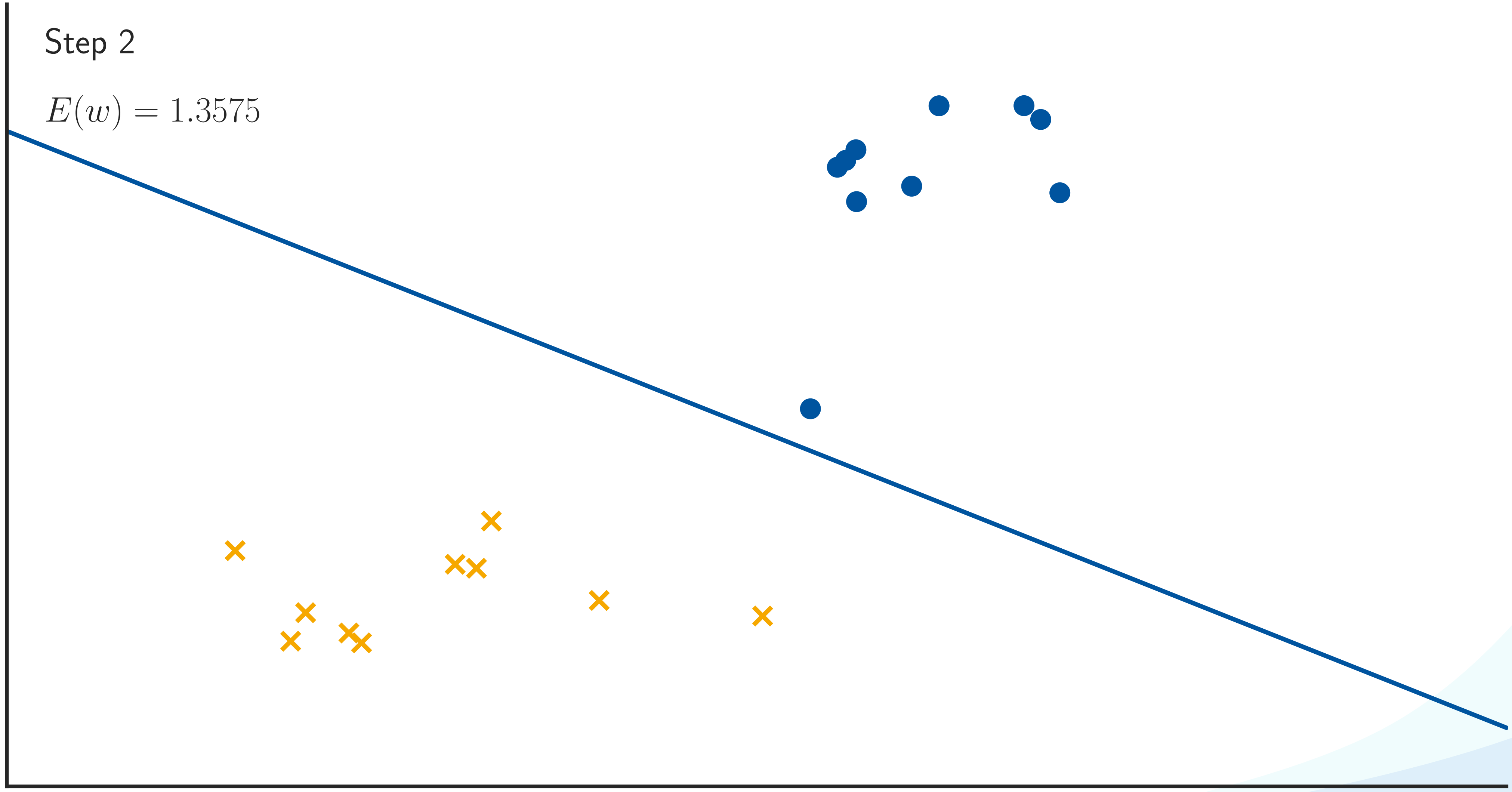
- Very similar form (normal equations).
  - But now with non-constant weighting matrix $\mathbf{R}$ (depends on $\mathbf{w}$).
  - Need to apply normal equations iteratively.
  - This is called Iteratively Reweighted Least-Squares (IRLS).

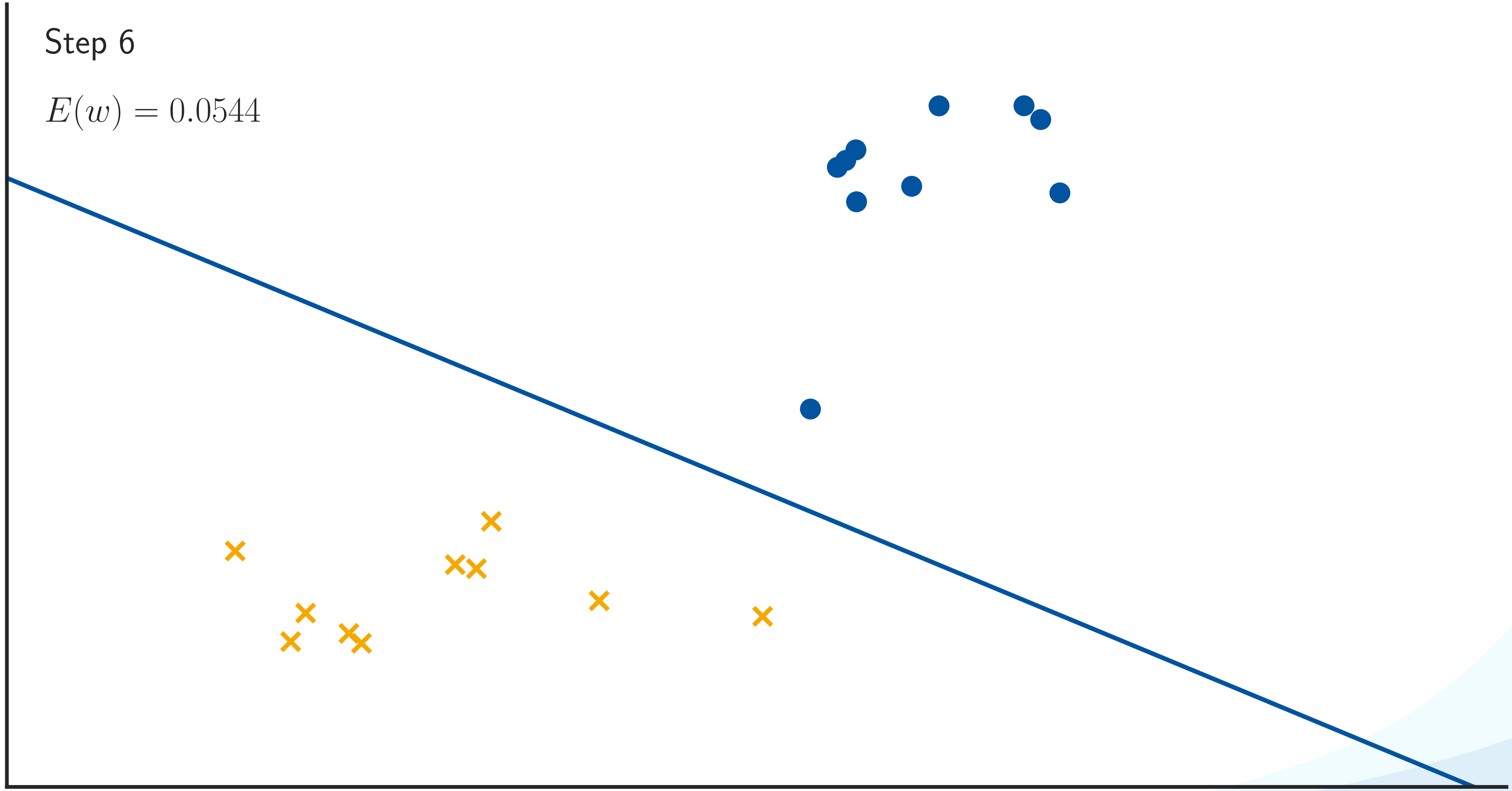
Example: Logistic Regression with IRLS

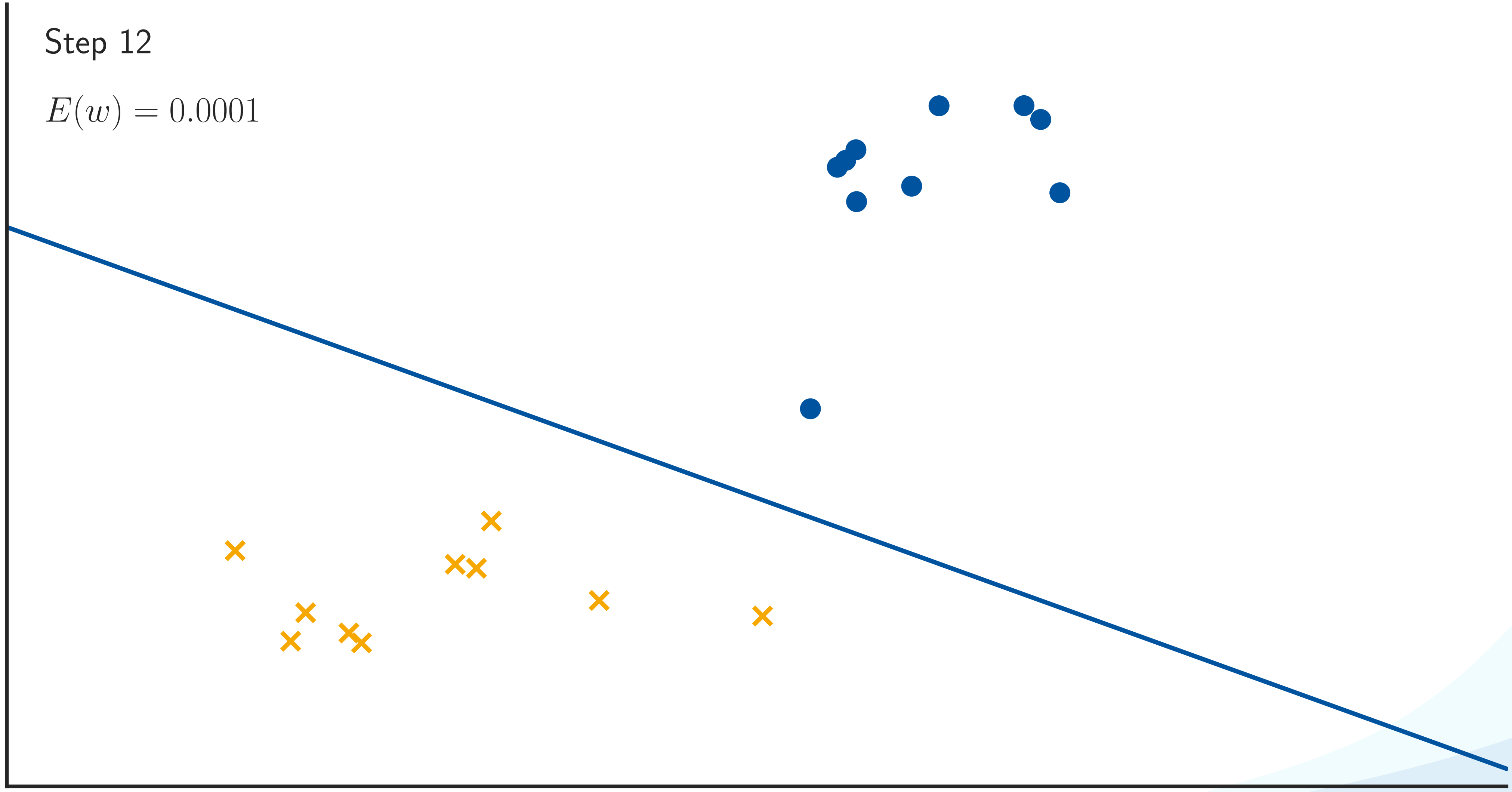




Zero training error with a single step!


Step 2
$E(w) = 1.3575$


Step 6
$E(w) = 0.0544$


Step 12
$E(w) = 0.0001$

Discussion: Second-Order Optimization

Advantages

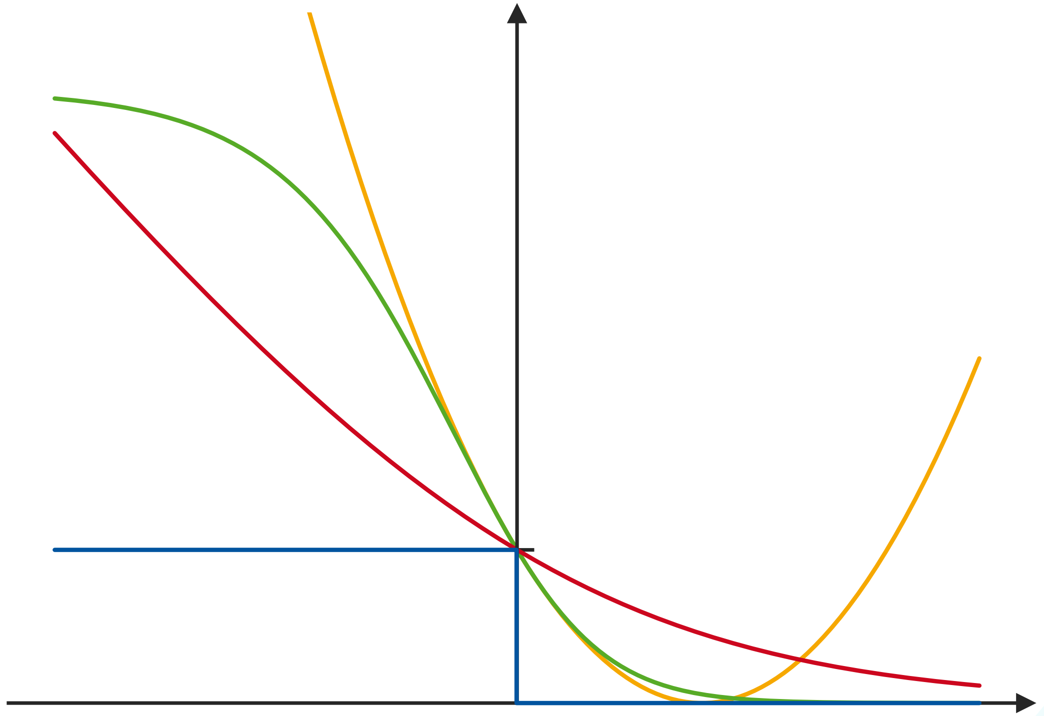
- Faster convergence than first-order methods

Limitations

- Second-order approach, relies on computing second derivatives.
- Computing (and inverting) the Hessian matrix is expensive for problems with many parameters.

Logistic Regression

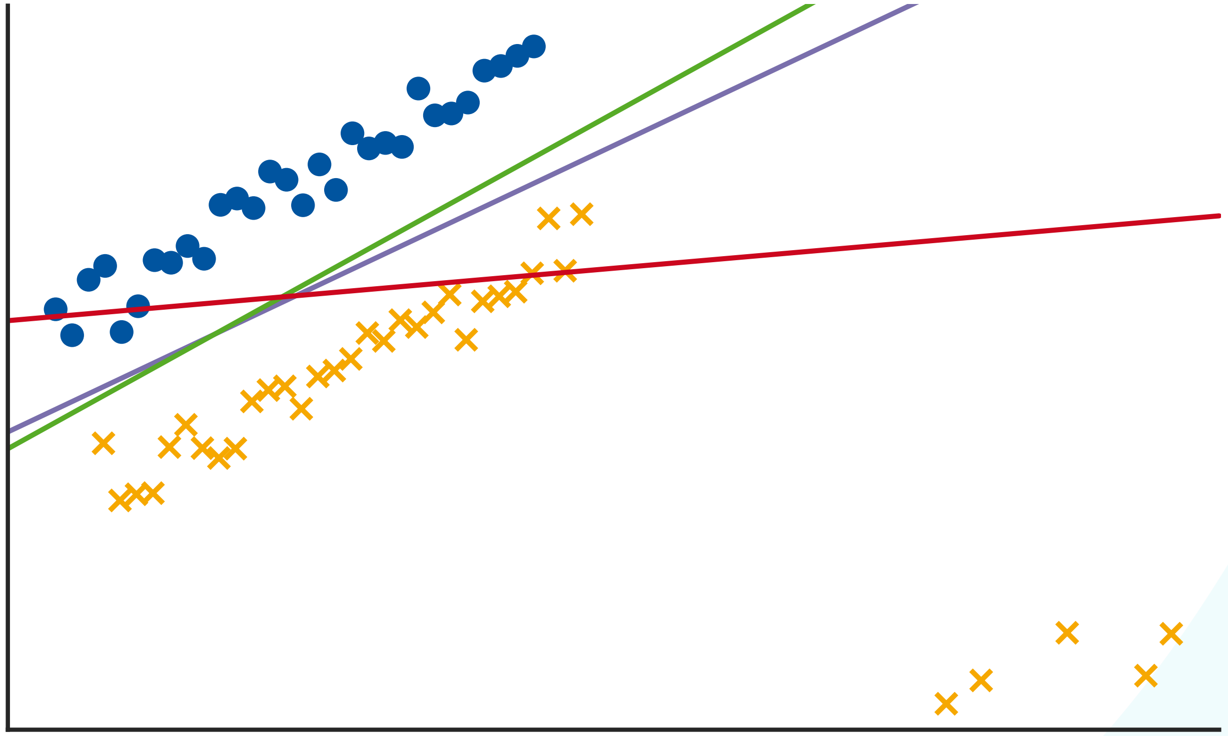
1. Logistic Regression Formulation
2. Motivation and Background
3. Iterative Estimation
4. First-Order Gradient Descent
5. Second-Order Gradient Descent
6. **Error Function Analysis**



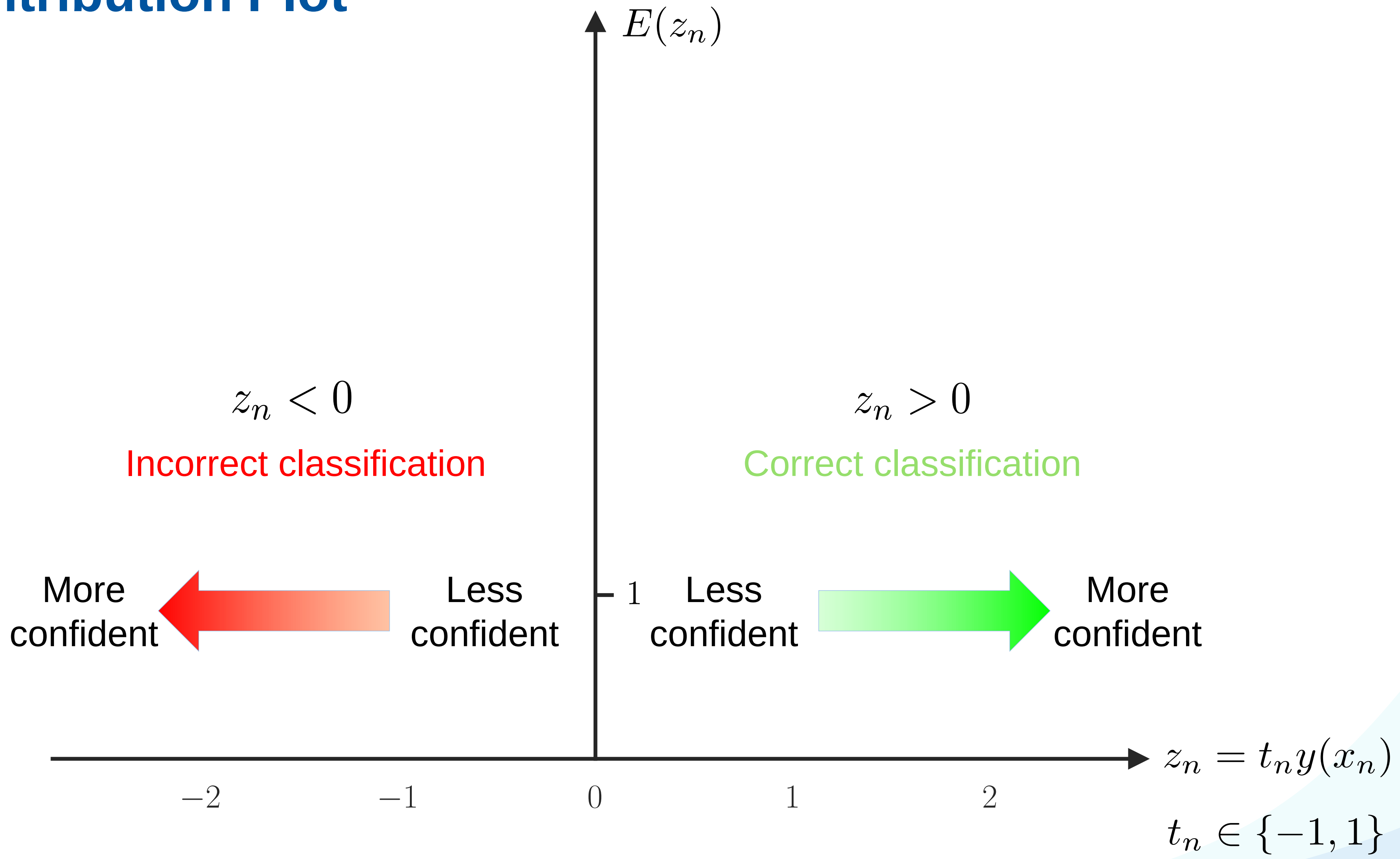
Error Function Analysis

- We have seen how to learn generalized linear discriminant models by optimizing an error function.
  - We observed problems with least-squares classification based on the squared error function.
  - We have seen that logistic regression behaves more robustly.

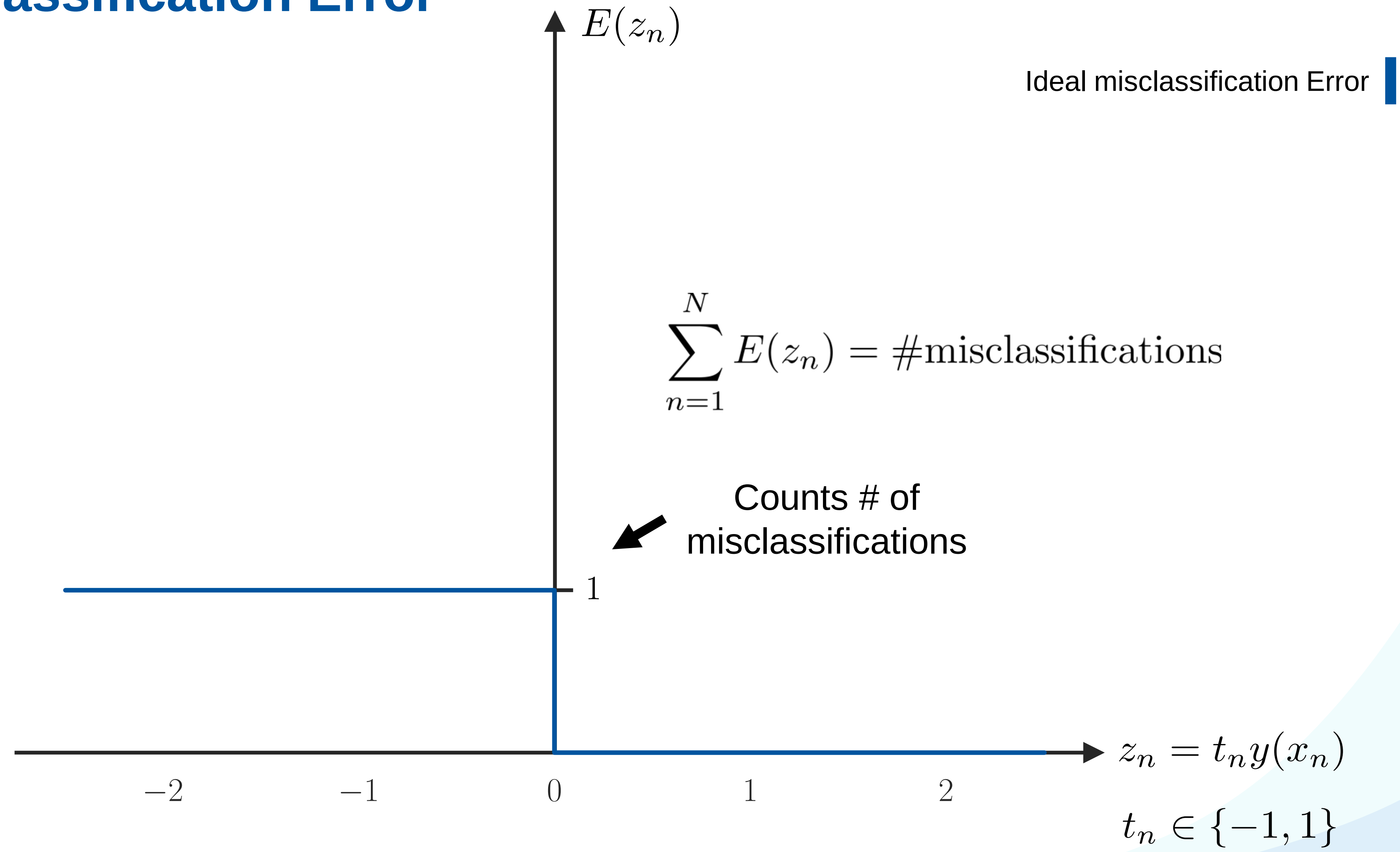
- *Let's analyze the cross-entropy error in more detail…*



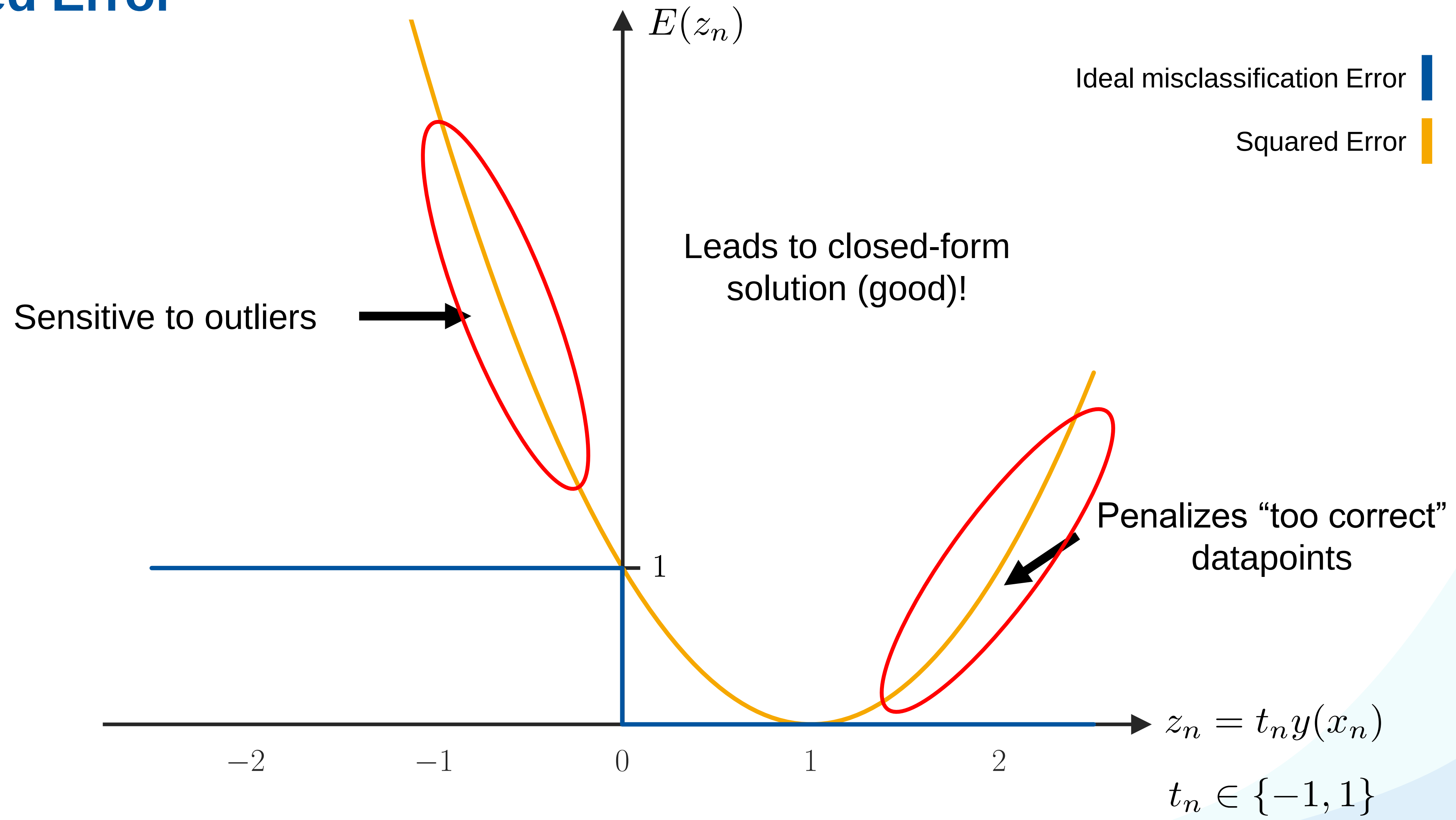
Error Contribution Plot



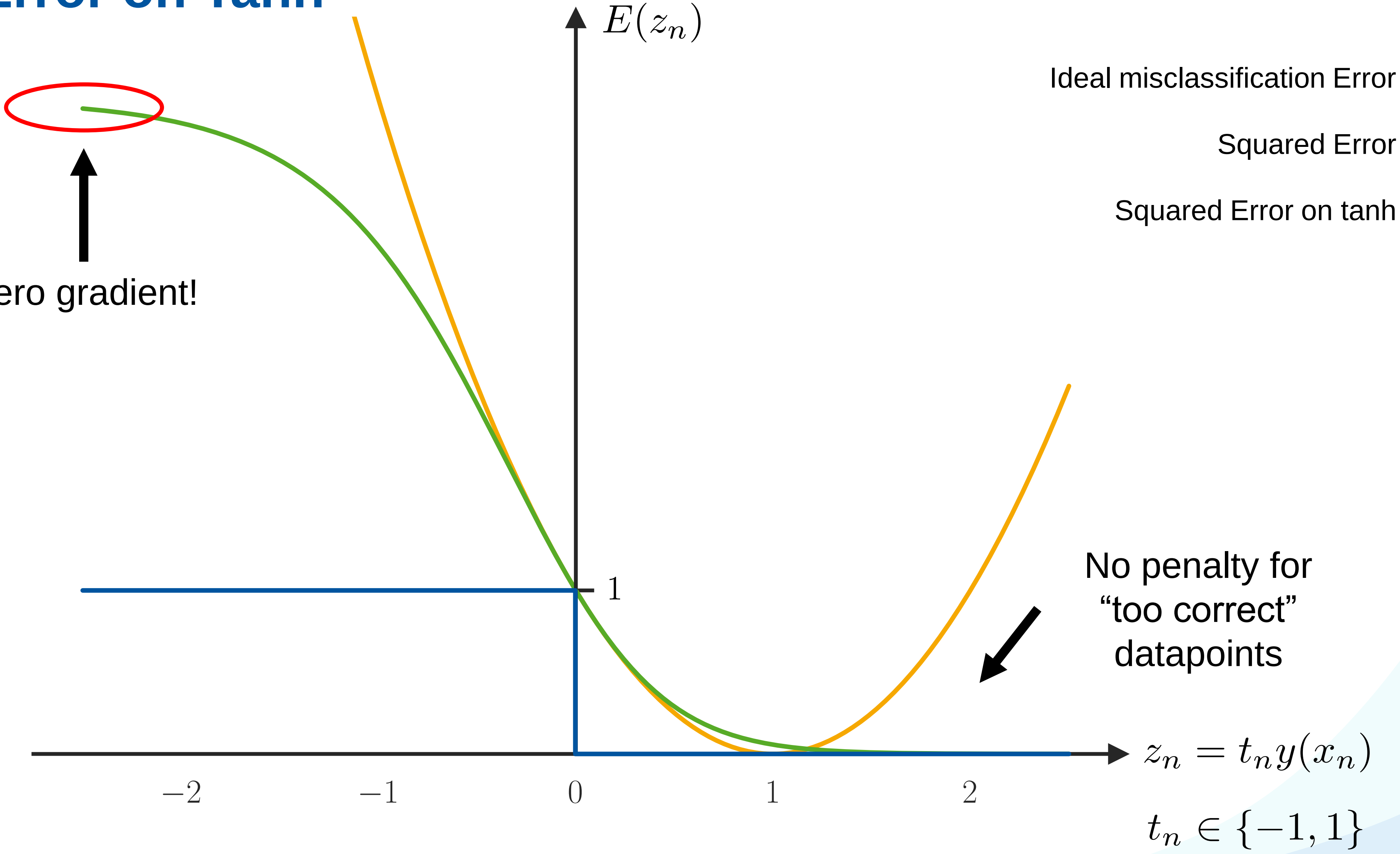
Ideal Misclassification Error



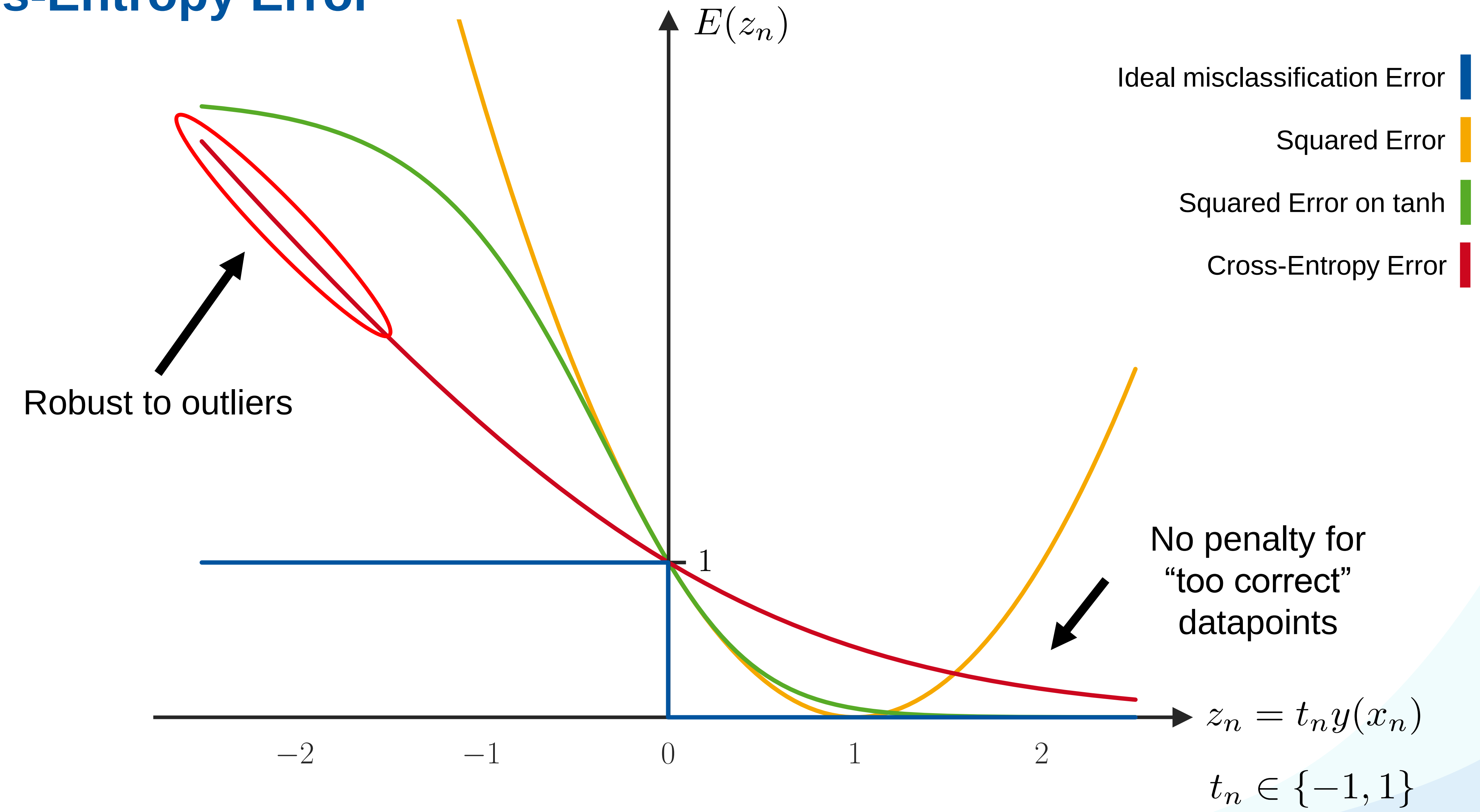
Squared Error



Squared Error on Tanh



Cross-Entropy Error



Discussion: Cross-Entropy Error

Advantages

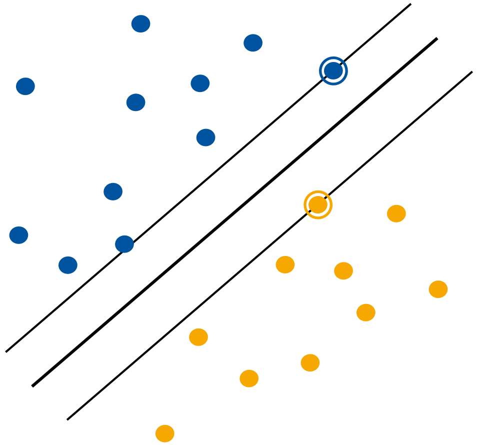
- Minimizer of this error corresponds to class posteriors
- Convex function, unique minimum exists
- Robust to outliers

Limitations

- No closed-form solution, requires iterative estimation

Machine Learning Topics

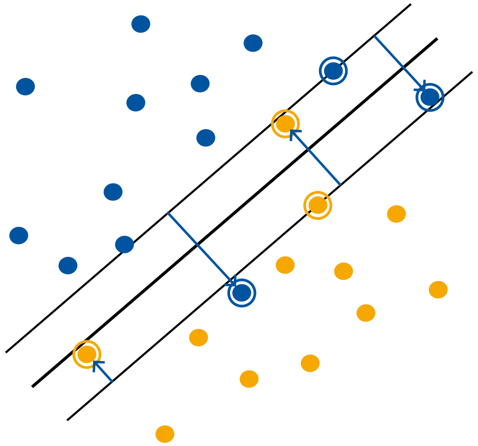
1. Introduction to ML
2. Probability Density Estimation
3. Linear Discriminants
4. Linear Regression
5. Logistic Regression
6. **Support Vector Machines**
7. AdaBoost
8. Neural Network Basics



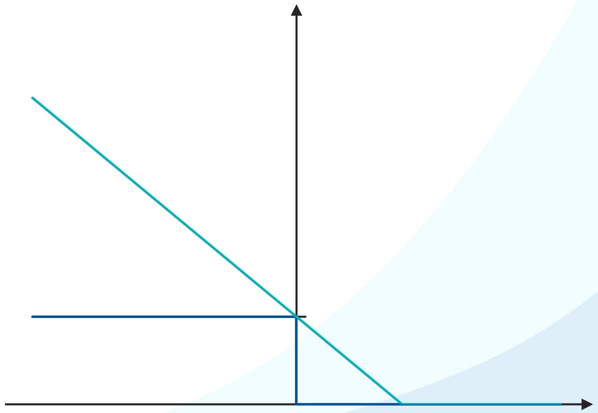
Maximum Margin Classification

$$L_p(\mathbf{w}, b, \mathbf{a})$$
$$L_d(\mathbf{a})$$

Primal & Dual Form



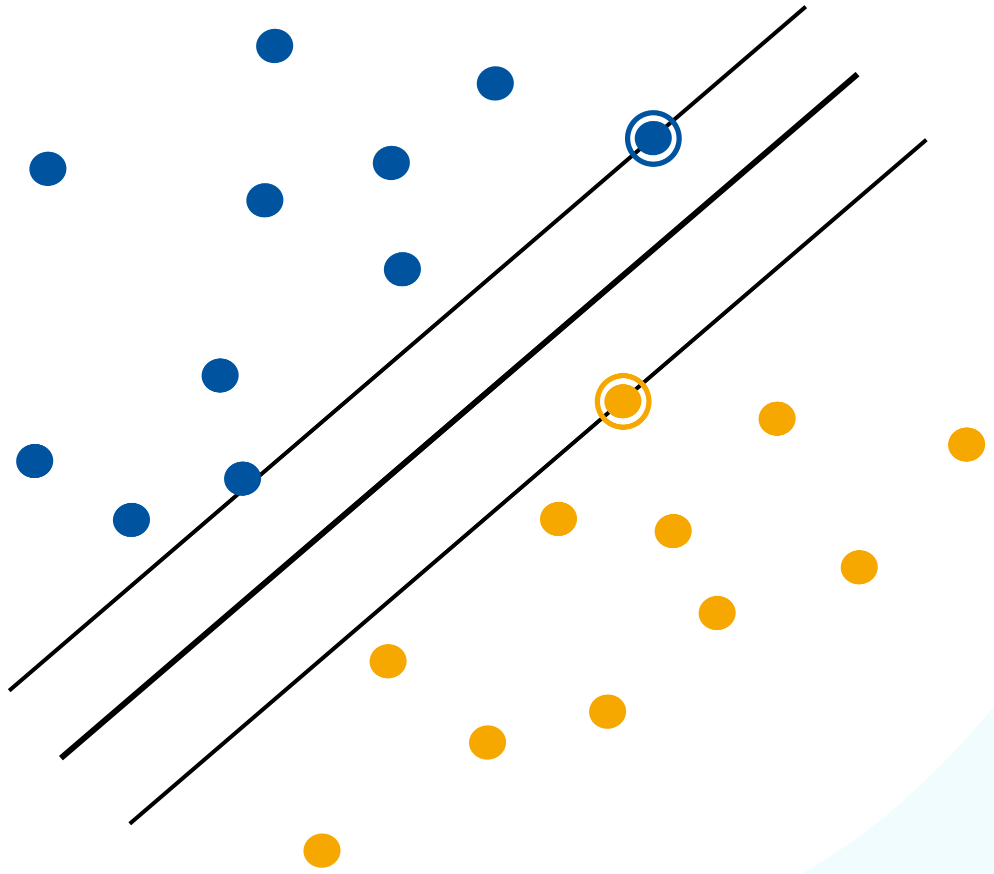
Soft-Margin SVM



Hinge Loss

Support Vector Machines

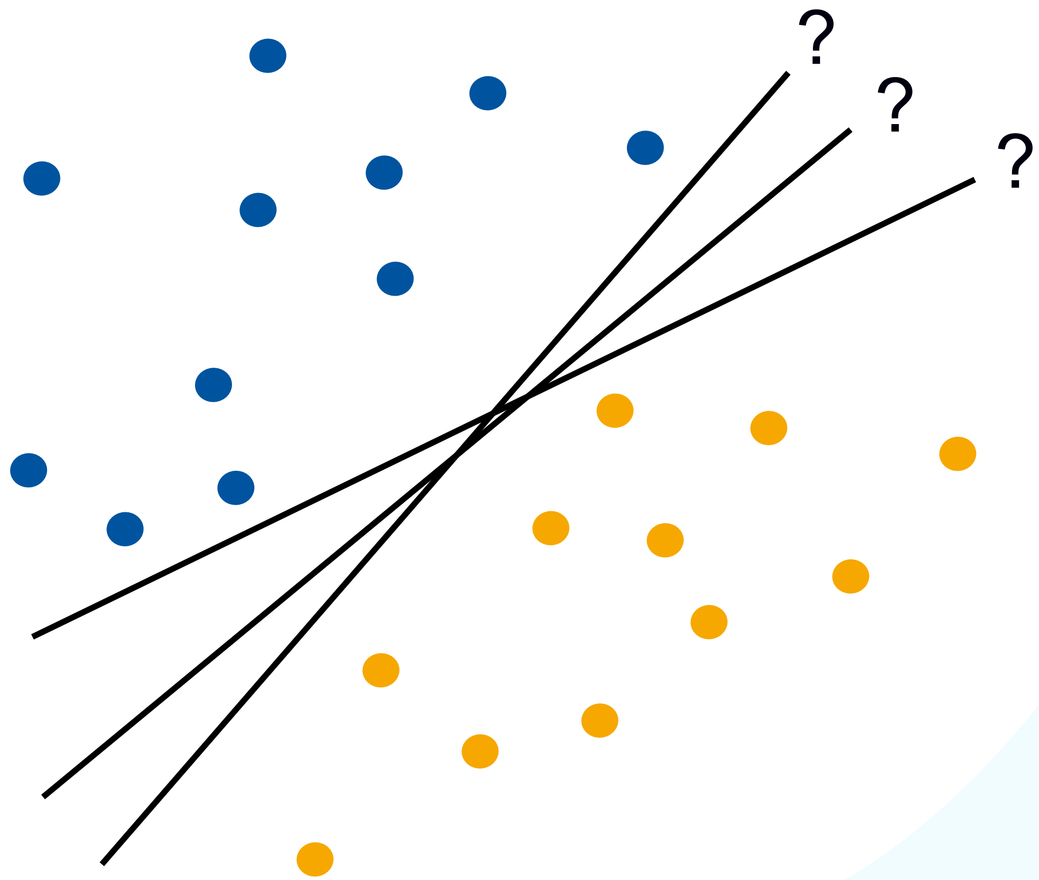
1. **Maximum Margin Classification**
2. Primal Formulation
3. Dual Formulation
4. Soft-Margin SVMs
5. Non-linear SVMs
6. Error Function Analysis



Maximum Margin Classification

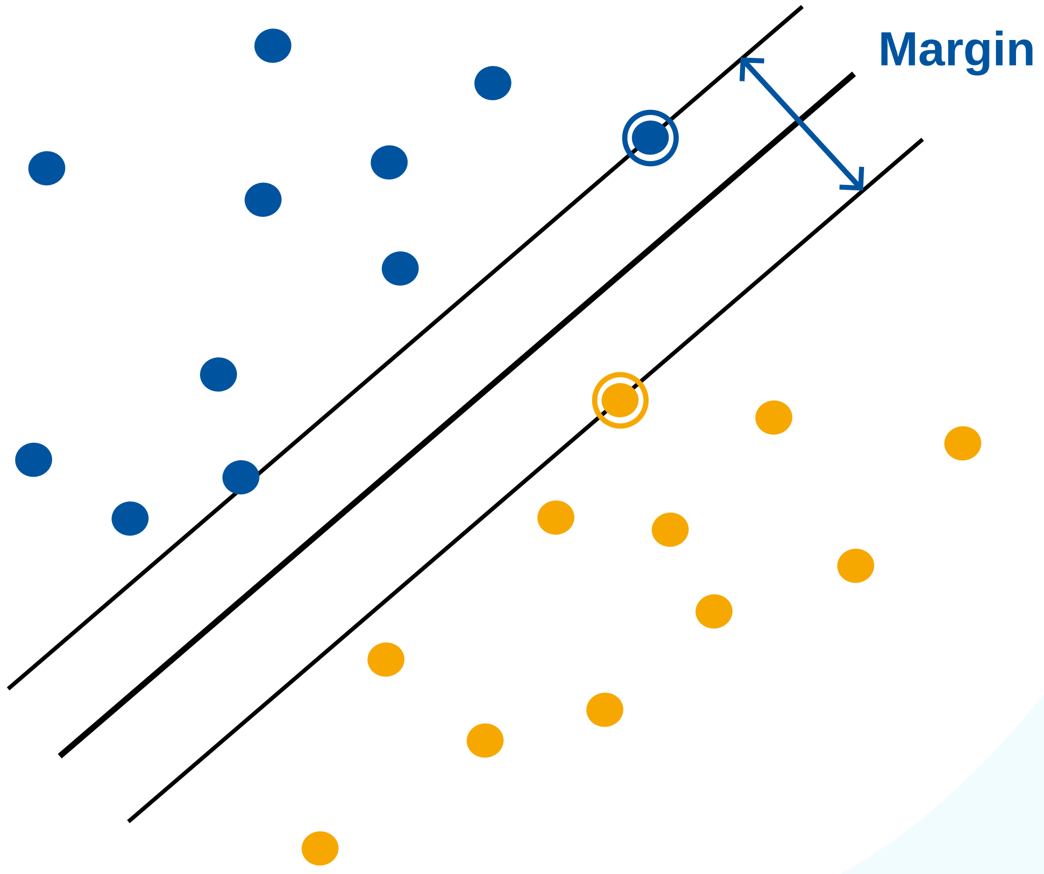
- Overfitting is often a problem with linearly separable data
  - Which of the many possible decision boundaries is correct?
  - All of them have zero error on the training set…
  - However, they will perform differently on novel test data.

- *How can we select the classifier with the best generalization performance?*



Maximum Margin Classification

- Intuitively, we want to choose the classifier which leaves maximal “safety room” for future data points.
- This classifier has the largest margin between positive and negative points.
- It can be shown: The larger the margin, the lower the capacity for overfitting.

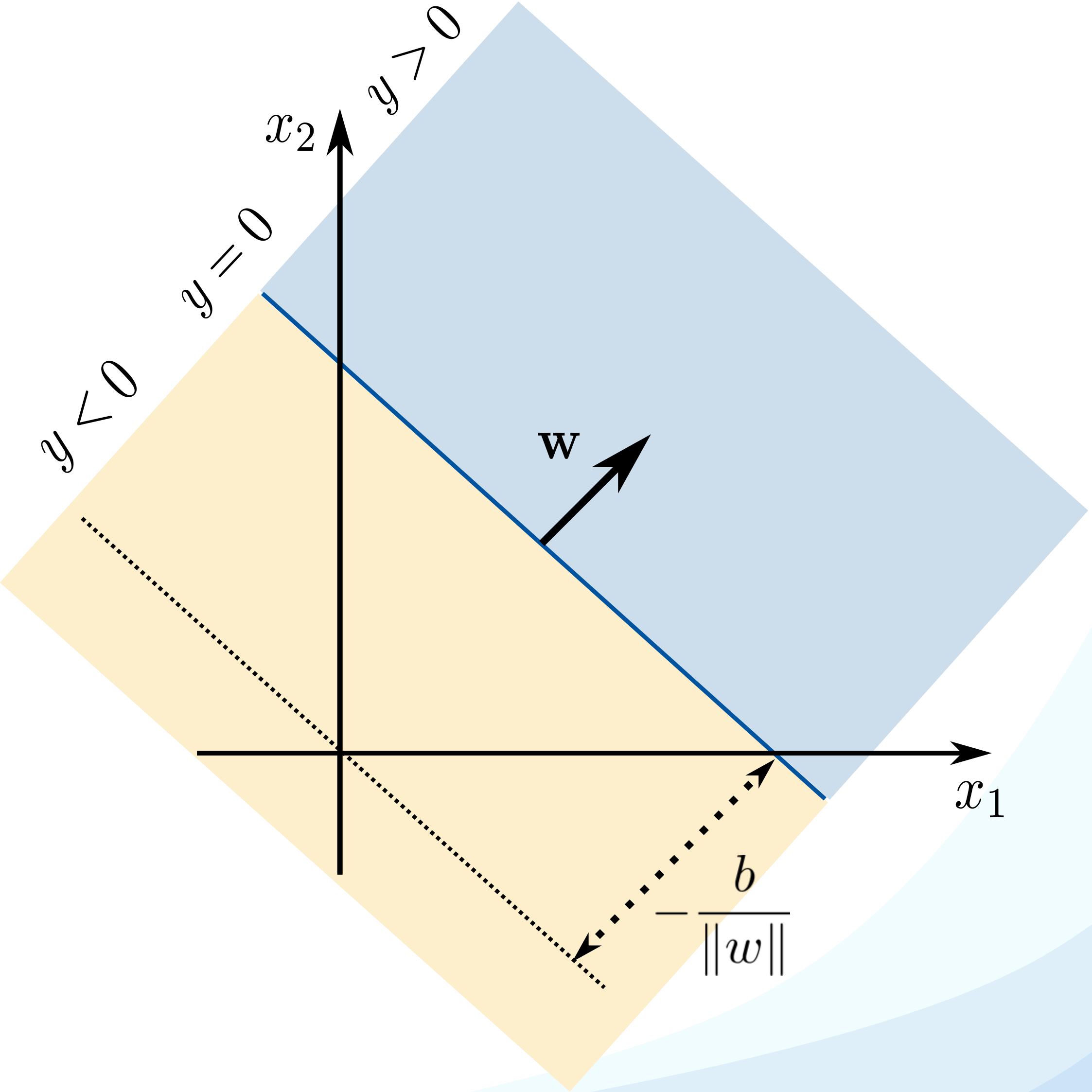


Intuition

- Let's first consider linearly separable data:
  - N training data points $\{(\mathbf{x}_i, t_i)\}_{i=1}^N, \ \mathbf{x}_i \in \mathbb{R}^D$
  - Binary labels $t_i \in \{-1, 1\}$
- A linear discriminant function models a hyperplane separating the data:

$$y(\mathbf{x}) = \mathbf{w}^\mathsf{T}\mathbf{x} + b$$

- *Note that we denote the bias explicitly with* b.
- Decision rule
  - Decide for $\mathcal{C}_1$ if $y(\mathbf{x}) > 0$, else for $\mathcal{C}_2$.



Support Vector Machines

- Assuming linearly separable data, we can always find a hyperplane with

$$\mathbf{w}^{\mathsf{T}}\mathbf{x}_n + b \geq +1 \text{ for } t_n = +1$$
$$\mathbf{w}^{\mathsf{T}}\mathbf{x}_n + b \leq -1 \text{ for } t_n = -1$$

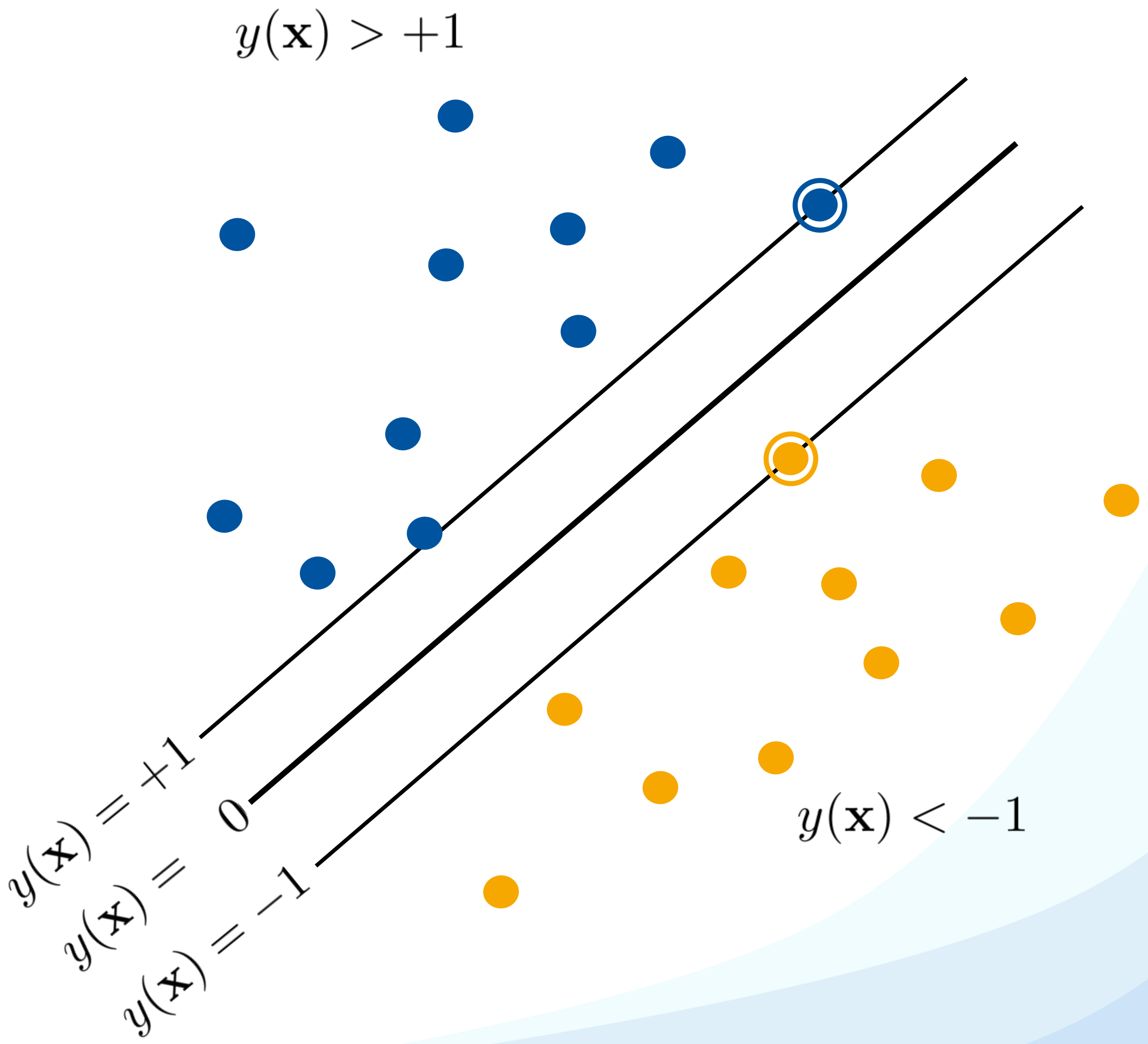
- In short:

$$t_n(\mathbf{w}^{\mathsf{T}}\mathbf{x}_n + b) \geq 1 \quad \forall n$$

- We can rescale $\mathbf{w}$ such that the equation holds exactly for the points on the margin:

$$t_n(\mathbf{w}^{\mathsf{T}}\mathbf{x}_n + b) = 1$$

- There will be at least one such point on either side.



- We can choose $\mathbf{w}$ such that

$$\mathbf{w}^{\mathsf{T}}\mathbf{x}_n + b = +1 \text{ for one } t_n = +1$$

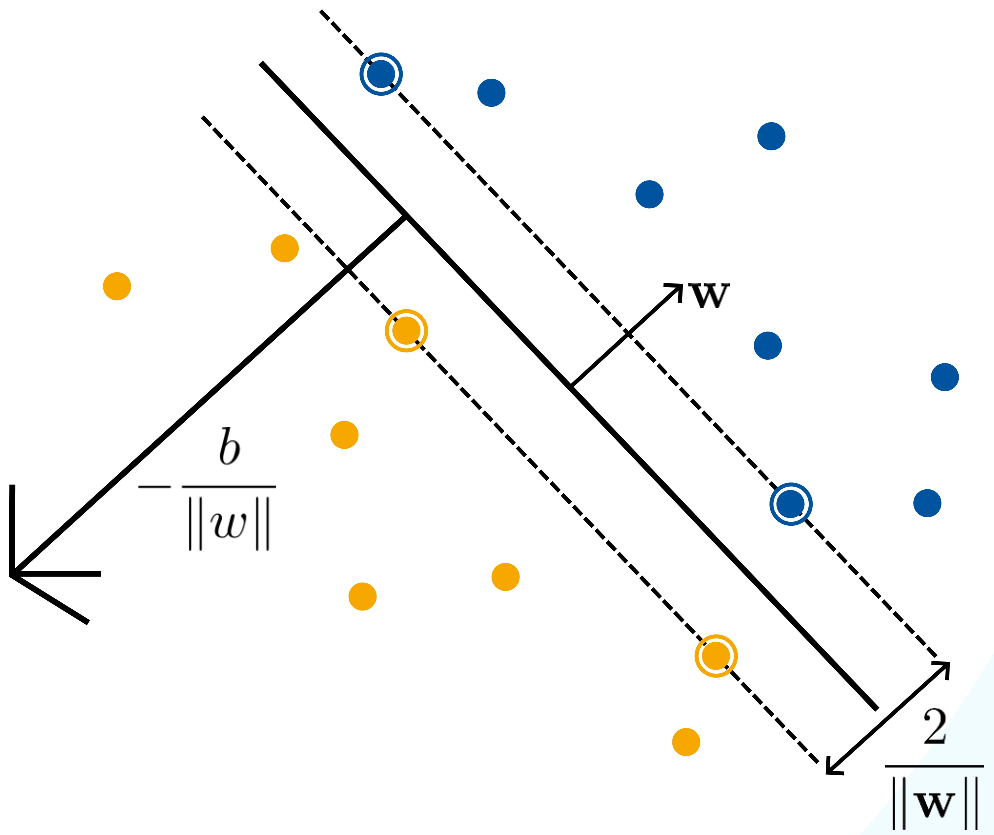
$$\mathbf{w}^{\mathsf{T}}\mathbf{x}_n + b = -1 \text{ for one } t_n = -1$$

- The distance between those hyperplanes is then the margin:

$$d_- = d_+ = \frac{1}{\|\mathbf{w}\|}$$

$$d_- + d_+ = \frac{2}{\|\mathbf{w}\|}$$

$\Rightarrow$ Maximize the margin by minimizing $\|\mathbf{w}\|^2$



- Optimization problem
  - Find the hyperplane with maximum margin by optimizing:

$$\underset{\mathbf{w},b}{\arg\min}\ \frac{1}{2}\|\mathbf{w}\|^2$$

"Maximize the margin"

such that

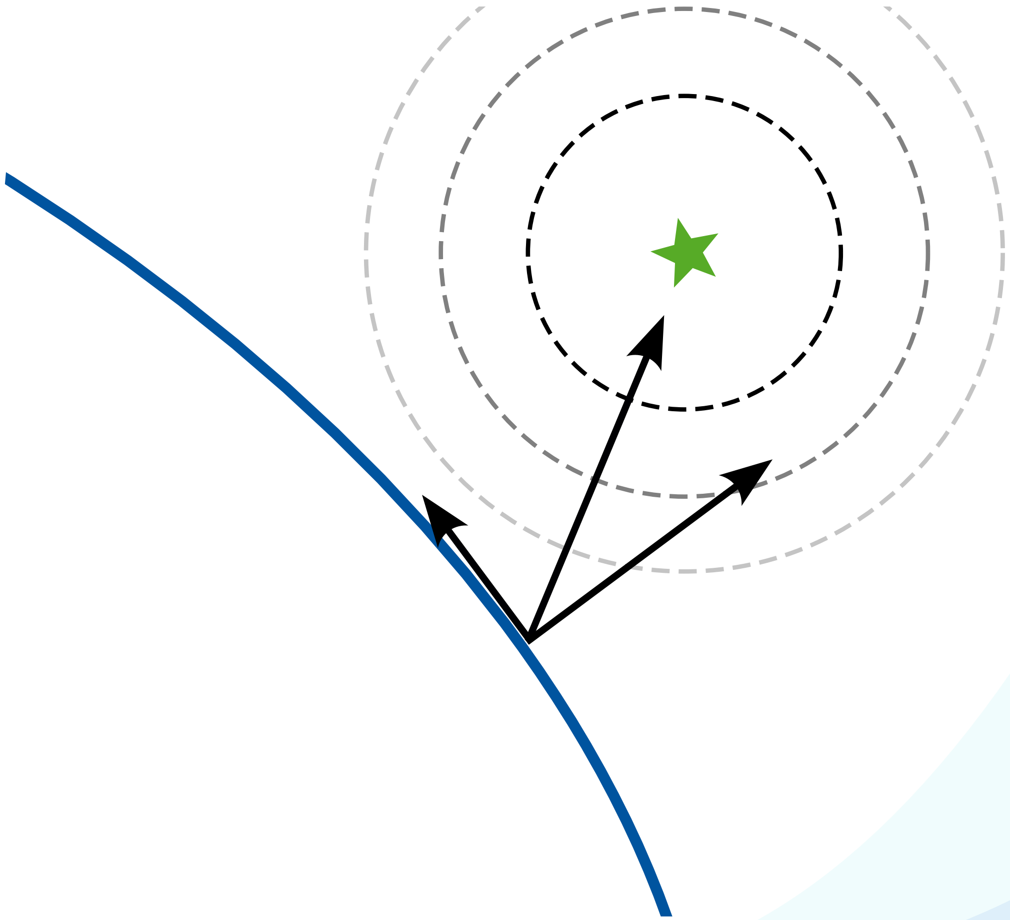
$$t_n(\mathbf{w}^{\mathsf{T}}\mathbf{x}_n + b) \geq 1 \quad \forall n$$

"such that each point is on the correct side of the margin"

- This is a quadratic programming problem with linear constraints.

Support Vector Machines

1. Maximum Margin Classification
    a) **Constrained Optimization**
2. Primal Formulation
3. Dual Formulation
4. Soft-Margin SVMs
5. Non-linear SVMs
6. Error Function Analysis



Constrained Optimization

- Recall the SVM objective:

$$\arg\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2, \quad \text{such that} \quad t_n(\mathbf{w}^\mathsf{T}\mathbf{x}_n + b) \geq 1 \quad \forall n$$

- This is a constrained optimization problem.
  - We want to optimize an objective $K(\mathbf{x})$ subject to constraints $f(\mathbf{x})$:

$$\operatorname{opt}_{\mathbf{x}} K(\mathbf{x})$$ min *or* max

such that $f(\mathbf{x}) = 0$ *equality constraints*

$f(\mathbf{x}) \geq 0$ *inequality constraints*

SVM

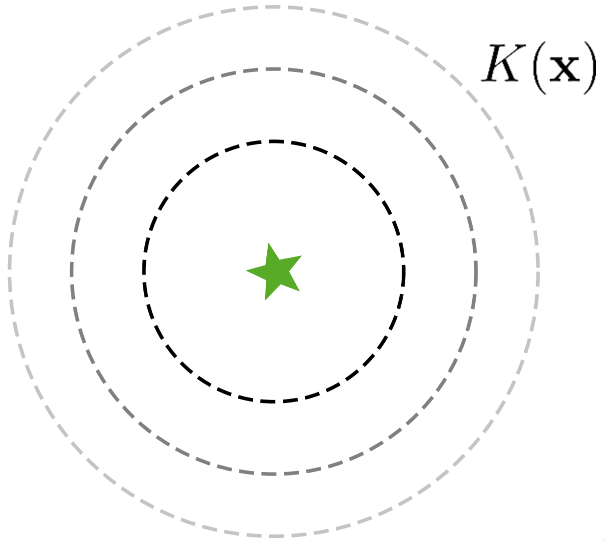
$$K(\mathbf{w}) = \frac{1}{2}\|\mathbf{w}\|^2$$

$$f_n(\mathbf{w}) = t_n(\mathbf{w}^\mathsf{T}\mathbf{x}_n + b) - 1 \geq 0 \quad \forall n$$

- We can solve such constrained optimization problems using the technique of Lagrange multipliers.

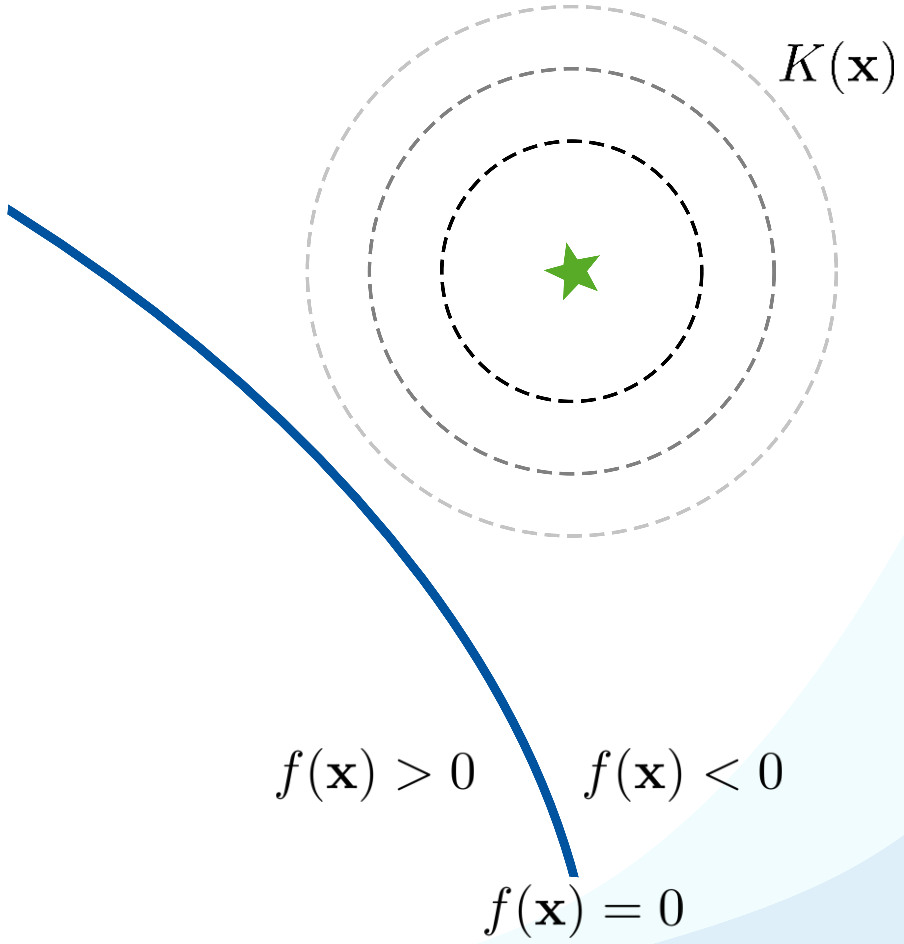
Lagrange Multipliers

- We want to maximize $K(\mathbf{x})$



Lagrange Multipliers

- We want to maximize $K(\mathbf{x})$ subject to constraints $f(\mathbf{x}) = 0$



Lagrange Multipliers

- We want to maximize $K(\mathbf{x})$ subject to constraints $f(\mathbf{x}) = 0$
- We can only move along $\nabla_{||} K = \nabla K + \lambda \nabla f$, with $\lambda \neq 0$.
- Add the constraints to the objective by introducing auxiliary variables λ:

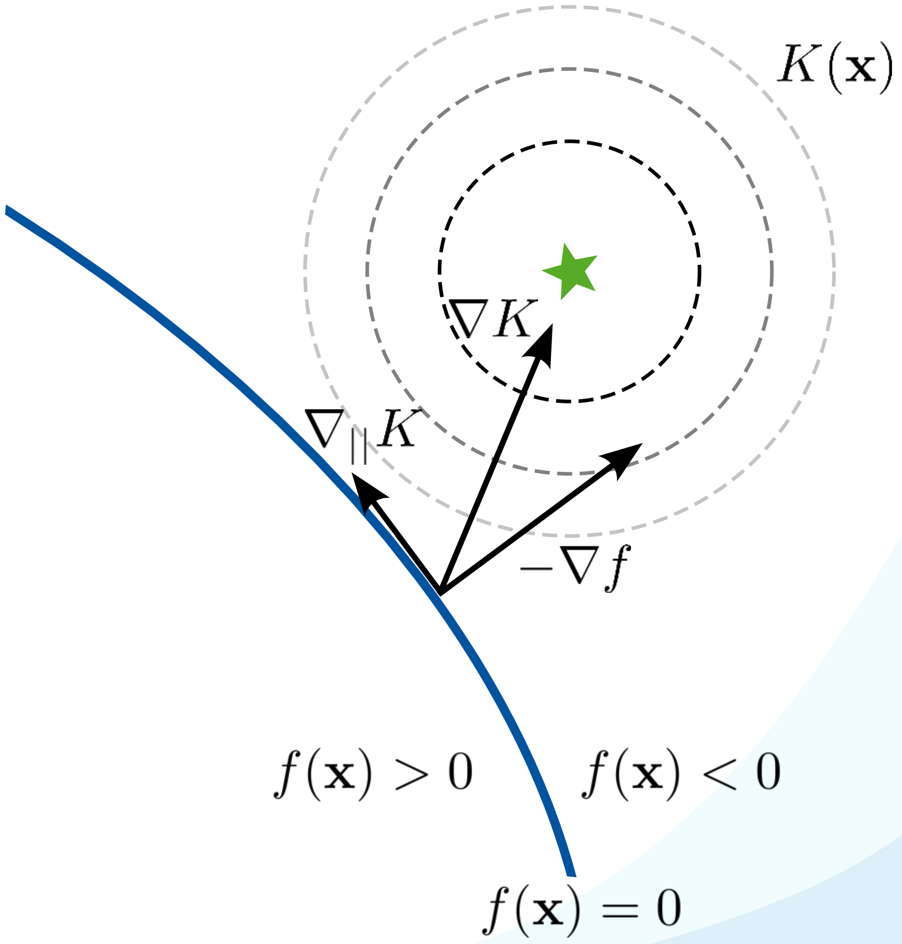
$$\mathcal{L}(\mathbf{x}, \lambda) = K(\mathbf{x}) + \lambda f(\mathbf{x})$$

- $\mathcal{L}$ is called the Lagrangian form of the optimization problem, and λ is referred to as a Lagrange multiplier.
- Optimize $\mathcal{L}$:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{x}} = \nabla_{||} K \stackrel{!}{=} 0$$

$$\frac{\partial \mathcal{L}}{\partial \lambda} = f(\mathbf{x}) \stackrel{!}{=} 0$$

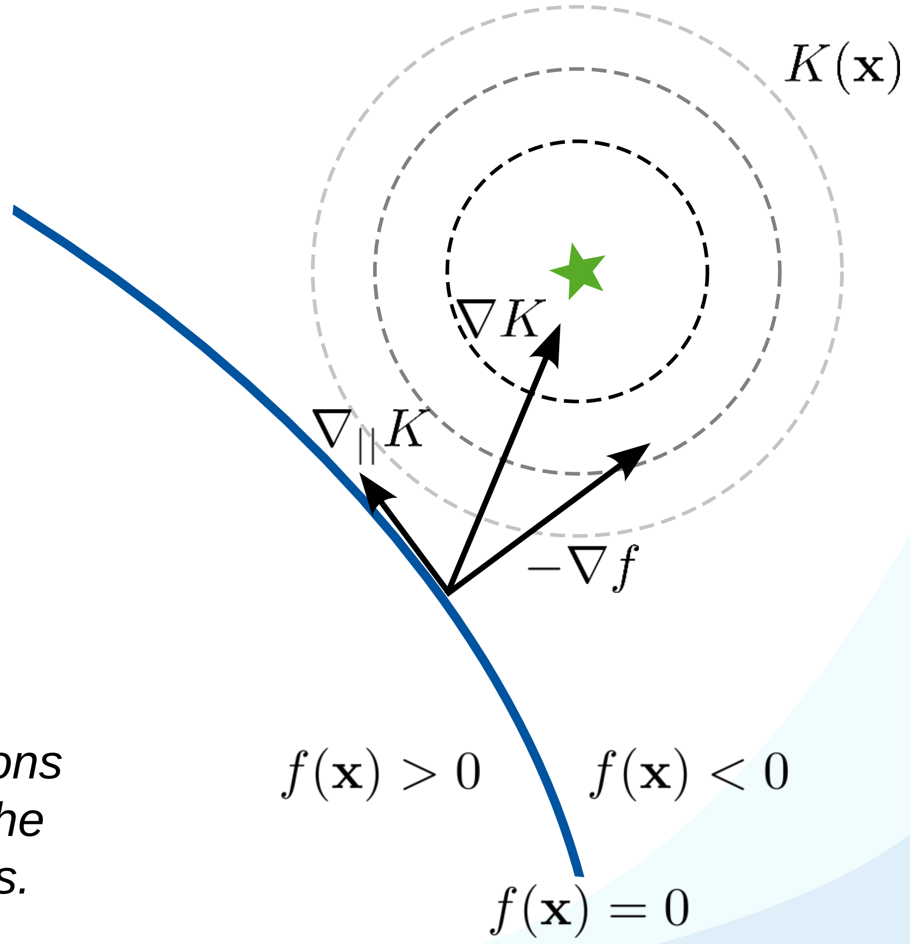
The objective is maximized while satisfying the constraints.



Inequality Constraints

- Now let's use inequality constraints $f(\mathbf{x}) \geq 0$.
- Optimize $\mathcal{L}(\mathbf{x}, \lambda) = K(\mathbf{x}) + \lambda f(\mathbf{x})$
  - Two cases
    - – Solution lies on boundary: $\Rightarrow f(\mathbf{x}) = 0$ for some $\lambda > 0$
    - – Solution lies inside $f(\mathbf{x}) > 0$: $\Rightarrow$ Constraint inactive: $\lambda = 0$

In both cases: $\lambda f(\mathbf{x}) = 0$



- Karush-Kuhn-Tucker (KKT) conditions:

$$\lambda \geq 0$$
$$f(\mathbf{x}) \geq 0$$
$$\lambda f(\mathbf{x}) = 0$$

All valid solutions need to fulfill the KKT conditions.

Maximization vs. Minimization

- Note: differences for maximization vs. minimization.
- If we want to maximize $K(\mathbf{x})$ subject to $f(\mathbf{x}) \geq 0$, we optimize the Lagrangian form

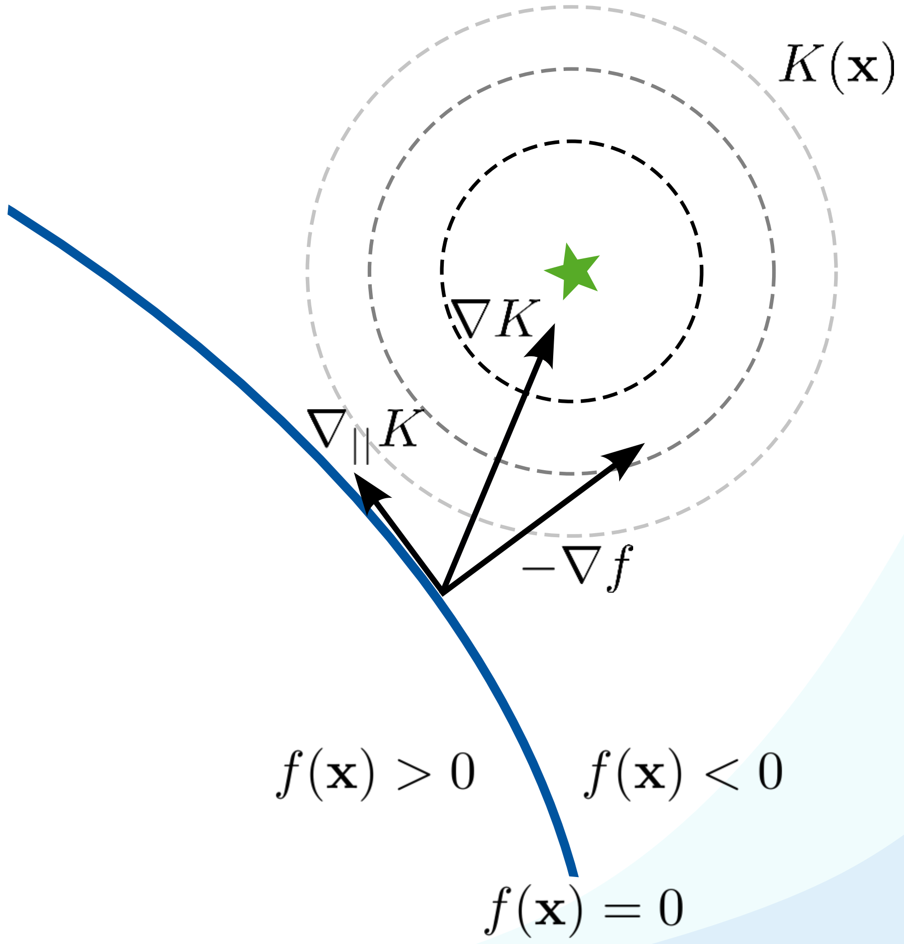
$$\mathcal{L}(\mathbf{x}, \lambda) = K(\mathbf{x}) + \lambda f(\mathbf{x})$$

  – *maximize* w.r.t. $\mathbf{x}$
  – *minimize* w.r.t. λ

- If we want to minimize $K(\mathbf{x})$ subject to $f(\mathbf{x}) \geq 0$, we optimize the Lagrangian form

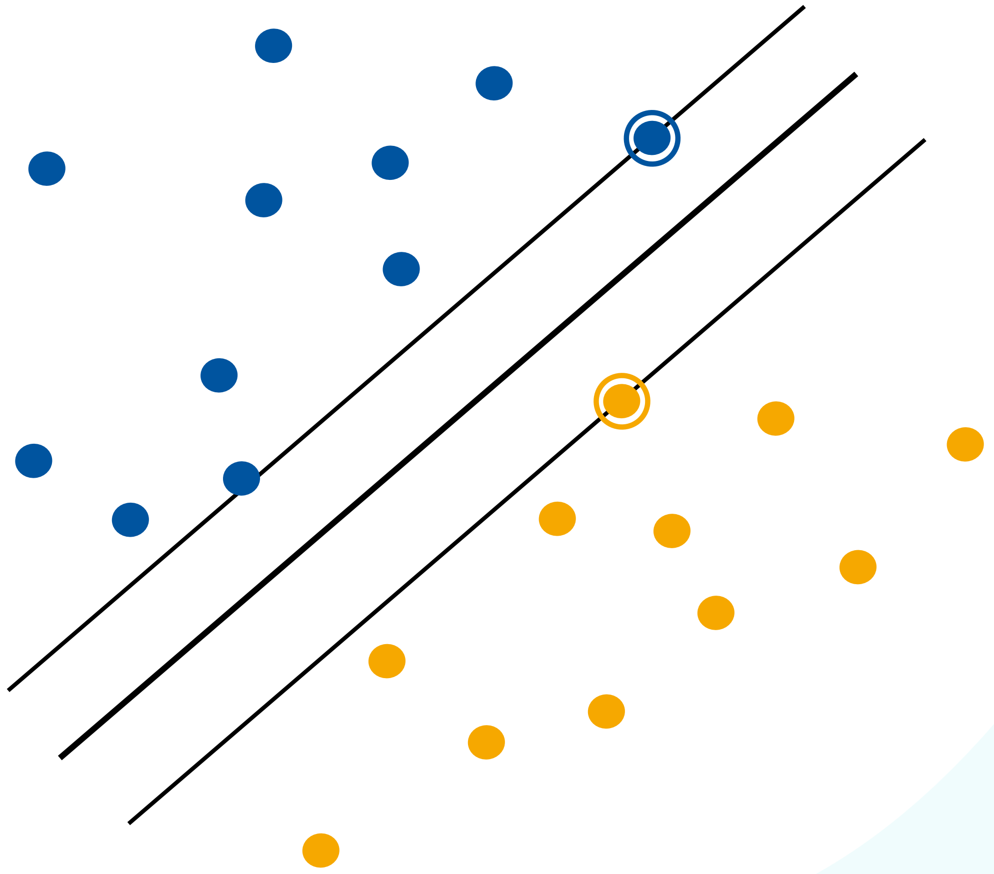
$$\mathcal{L}(\mathbf{x}, \lambda) = K(\mathbf{x}) - \lambda f(\mathbf{x})$$

  – *minimize* w.r.t. $\mathbf{x}$
  – *maximize* w.r.t. λ



Support Vector Machines

1. Maximum Margin Classification
2. **Primal Formulation**
3. Dual Formulation
4. Soft-Margin SVMs
5. Non-linear SVMs
6. Error Function Analysis



Primal SVM Formulation

- Recall the SVM objective:

$$\arg\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2, \quad \text{such that} \quad t_n(\mathbf{w}^\mathsf{T}\mathbf{x}_n + b) \geq 1 \quad \forall n$$

- We introduce positive Lagrange multipliers $a_n \geq 0$ and get the primal form of SVMs:

$$L(\mathbf{w}, b, \mathbf{a}) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{n=1}^{N} a_n \left[t_n(\mathbf{w}^\mathsf{T}\mathbf{x}_n + b) - 1\right]$$

Necessary and sufficient conditions:

$$\begin{aligned} a_n &\geq 0 \\ t_n(\mathbf{w}^\mathsf{T}\mathbf{x}_n + b) - 1 &\geq 0 \\ a_n[t_n(\mathbf{w}^\mathsf{T}\mathbf{x}_n + b) - 1] &= 0 \end{aligned}$$

KKT conditions:

$$\begin{aligned} \lambda &\geq 0 \\ f(\mathbf{x}) &\geq 0 \\ \lambda f(\mathbf{x}) &= 0 \end{aligned}$$

Lagrangian Formulation

- We want to minimize the primal form:

$$L(\mathbf{w}, b, \mathbf{a}) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{n=1}^{N} a_n \left[t_n(\mathbf{w}^\mathsf{T}\mathbf{x}_n + b) - 1\right]$$

$$\frac{\partial L(\mathbf{w}, b, \mathbf{a})}{\partial b} = \sum_{n=1}^{N} a_n t_n$$

$$\frac{\partial L(\mathbf{w}, b, \mathbf{a})}{\partial \mathbf{w}} = \mathbf{w} - \sum_{n=1}^{N} a_n t_n \mathbf{x}_n$$

- Setting the gradients for $\mathbf{w}, b$ to zero, we get:

$$\frac{\partial L}{\partial b} = 0 \Rightarrow \sum_{n=1}^{N} a_n t_n = 0$$

$$\frac{\partial L}{\partial \mathbf{w}} = 0 \Rightarrow \mathbf{w} = \sum_{n=1}^{N} a_n t_n \mathbf{x}_n$$

- The hyperplane is computed as a linear combination of training examples:

$$\mathbf{w} = \sum_{n=1}^{N} a_n t_n \mathbf{x}_n$$

$$\frac{\partial L}{\partial \mathbf{w}} = 0 \Rightarrow \mathbf{w} = \sum_{n=1}^{N} a_n t_n \mathbf{x}_n$$

- Additionally, the solution needs to fulfill

$$a_n \left[t_n (\mathbf{w}^\mathsf{T} \mathbf{x}_n + b) - 1 \right] = 0$$

KKT conditions:

$$a_n \geq 0$$
$$t_n (\mathbf{w}^\mathsf{T} \mathbf{x}_n + b) - 1 \geq 0$$
$$a_n [t_n (\mathbf{w}^\mathsf{T} \mathbf{x}_n + b) - 1] = 0$$

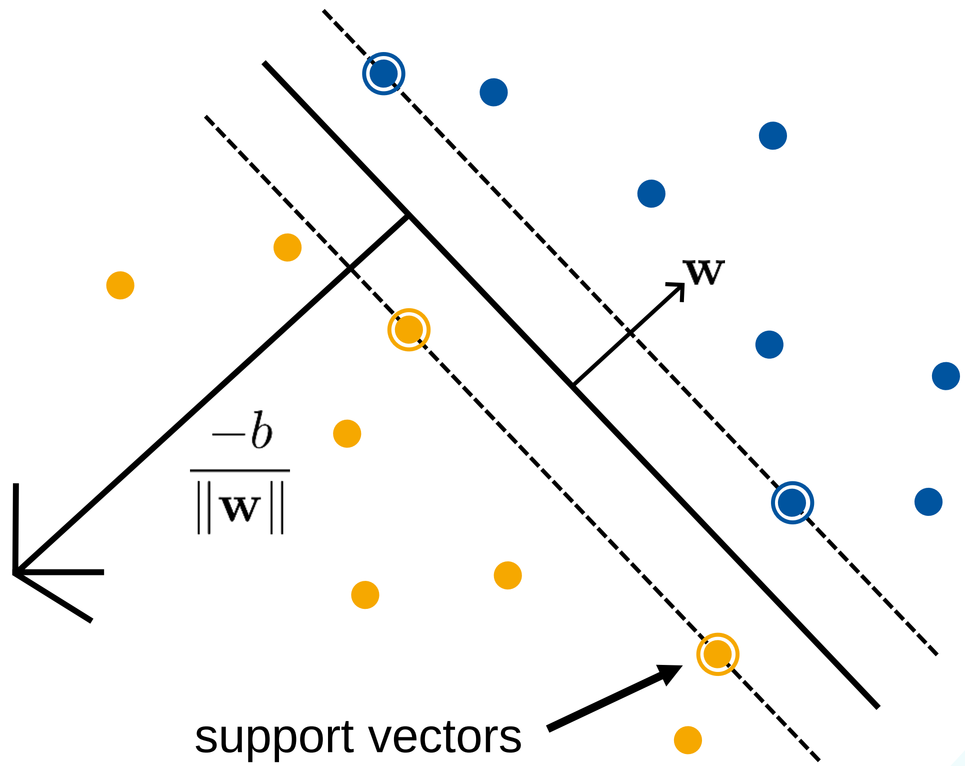
- This implies $a_n > 0$ only for those points for which

$$\left[t_n (\mathbf{w}^\mathsf{T} \mathbf{x}_n + b) - 1 \right] = 0$$

Only some data points influence the decision boundary!

Intuition

- The training points with $a_n > 0$ are called support vectors.
- They are the points on the margin.
- This makes the SVM robust to "too correct" points!



- We still need to find b.
- Observation: Any support vector $\mathbf{x}_n$ satisfies

$$t_n y(\mathbf{x}_n) = t_n \left(\sum_{m \in \mathcal{S}} a_m t_m \mathbf{x}_m^{\mathsf{T}} \mathbf{x}_n + b \right) = 1$$

- Using $t_n^2 = 1$, we can derive

$$b = t_n - \sum_{m \in \mathcal{S}} a_m t_m \mathbf{x}_m^{\mathsf{T}} \mathbf{x}_n$$

- In practice, it is more robust to average over all support vectors:

$$b = \frac{1}{N_{\mathcal{S}}} \sum_{n \in \mathcal{S}} \left(t_n - \sum_{m \in \mathcal{S}} a_m t_m \mathbf{x}_m^{\mathsf{T}} \mathbf{x}_n \right)$$

Advantages

- SVMs yield a linear classifier with "guaranteed" generalization capability.
- Convex optimization, yields globally optimal solution.
- Solution depends only on a subset of the input data points, the support vectors.
- Automatic robustness against "too correct" data points.

Limitations

- Need to solve quadratic programming problem: time complexity for that is cubic in the number of variables.
- Here: Time complexity is in $\mathcal{O}(D^3)$.
- Scaling to high-dimensional data is difficult.

References and Further Reading

- More information about SVMs is available in Chapter 7.1 of Bishop's book.

Christopher M. Bishop
Pattern Recognition and Machine Learning
Springer, 2006