Lecture 2 — Introduction to Machine Learning

Machine learning is a very interdisciplinary subject, so it can be approached from different angles ("buckets"). These lectures use the **probability** and **computer science** buckets. Later lectures add others, such as **neuroscience**.

Contents

1. What is an ML algorithm?
2. Formal setup of supervised learning
3. The hypothesis space
4. Constructing loss functions via maximum likelihood
5. Regression → Mean Squared Error
6. Binary classification → Binary Cross-Entropy
7. Linear regression: closed-form solution
8. Regularized least squares
9. Linear classification: logistic regression
10. Optimizing the loss: gradient descent
11. Summary

---

1. What is an ML algorithm?

Definition. A computer program learns from **experience E**, on a **task T**, with respect to a **performance measure P**, when its performance P on the task T improves with experience E.

The **task T** is what the program has to do. The two main examples are classification, $f : \mathbb{R}^n \to \mathbb{Z}_c$ (assigning one of c classes), and regression, $f : \mathbb{R}^n \to \mathbb{R}^m$, but there are many others.

The **performance measure P** quantifies how well the algorithm is performing. The simplest one is **accuracy**, the proportion of examples predicted correctly. More often we use a **loss function**, which usually comes from a *maximum likelihood principle*, as we will see in Section 4.

The **experience E** is what the program learns from, and it can be of two kinds. In **unsupervised** learning, the experience is the overall structure of the datapoints given. In **supervised** learning, the experience comes as a set of (x, y) pairs, an input and its output. The line between the two is often blurred. From here on we focus on supervised learning.

---

2. Formal setup of supervised learning

Inputs live in an **instance space** $\mathcal{X}$ and are drawn as $x^{(i)} \sim p(x)$. Outputs live in an **output space** $\mathcal{Y}$ and are produced by "the supervisor", a conditional distribution $g(y \mid x)$, so that $y^{(i)} \sim g(y \mid x^{(i)})$. The dataset is then a collection of P input–output pairs, called *patterns*:

$$\mathcal{D} = \{(x^{(i)}, y^{(i)})\}_{i=1}^{P}, \qquad (x^{(i)}, y^{(i)}) \sim g(y \mid x)\, p(x) \,\hat{=}\, p_{\text{data}}$$

The joint distribution p_{data} is very much related to the goal of learning: we want to find a function $f(x)$ that captures the supervisor $g(y \mid x)$.

---

3. The hypothesis space

Where do we look for $f(x)$? We search in a **hypothesis space**, a family of functions indexed by parameters θ:

$$\mathcal{H} : \mathbb{R}^n \to \mathcal{F}, \qquad \theta \mapsto f(\,\cdot\,; \theta)$$

The simplest example is the space of linear functions, $\theta \mapsto \langle \theta, x \rangle_{\mathbb{R}^n}$. More generally, we can take the inner product in a Hilbert space S, $\theta \mapsto \langle \theta, x \rangle_S$, which covers for instance polynomial and Fourier functions. In practice this amounts to first mapping x through a set of features ϕ_k (polynomials or more advanced functions) and then taking a linear combination of them:

$$\langle \theta, \phi(x) \rangle_{\mathbb{R}^n} = \theta_1 \phi_1(x) + \theta_2 \phi_2(x) + \dots$$

This idea is very profound and is at the core of neural networks and NN theory. The buzzword to remember is **kernel methods**, the subject of Lecture 3, Section 7.

---

4. Constructing loss functions via maximum likelihood

A performance measure can be written in several equivalent ways, and you may encounter all of them: as $\mathcal{L} : \mathcal{H} \times \mathcal{Y} \to \mathbb{R}_+$, as $\mathcal{L} : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}_+$, or, in practice, as $\mathcal{L} : \mathbb{R}^n \to \mathbb{R}$. The real question is how to construct *valid* losses.

The idea is to choose the parameters $\hat{\theta}$ that make the observed realization of $\mathcal{D}$ as probable as possible. This is the **Maximum Likelihood Principle**:

$$\hat{\theta} = \arg\max_\theta \, p_{\text{model}}(\mathcal{D}; \theta) = \arg\max_\theta \, p_{\text{model}}(y^{(1)}, \dots, y^{(P)} \mid x^{(1)}, \dots, x^{(P)}; \theta)$$

If the samples are i.i.d., the joint likelihood factorizes into a product over patterns:

$$\hat{\theta} = \arg\max_\theta \prod_{i=1}^{P} p_{\text{model}}(y^{(i)} \mid x^{(i)}; \theta)$$

Since log is a monotonic function, taking it does not change the argmax, and it conveniently turns the product into a sum. Dividing by P is likewise harmless and gives an empirical average, which for large P approximates an expectation over the data distribution:

$$\hat{\theta} = \arg\max_\theta \sum_{i=1}^{P} \log p_{\text{model}}(y^{(i)} \mid x^{(i)}; \theta) = \arg\max_\theta \frac{1}{P} \sum_{i=1}^{P} \log p_{\text{model}}(y^{(i)} \mid x^{(i)}; \theta) \approx \arg\max_\theta \, \mathbb{E}_{p_{\text{data}}} \big[\log p_{\text{model}}(y \mid x; \theta) \big]$$

The same estimator is obtained by minimizing the Kullback–Leibler divergence between the data distribution and the model, $\hat{\theta} = \arg\min_\theta \, \mathrm{KL}[p_{\text{data}} \,\|\, p_{\text{model}}]$, where

$$\mathrm{KL}(p \,\|\, q) \mathrel{\hat{=}} \mathbb{E}_p(\log p - \log q) = \int (\log p(z) - \log q(z)) \, dp(z)$$

With this principle in hand, we can derive losses for regression and classification simply by choosing a model distribution $p_{\text{model}}(y \mid x; \theta)$.

Maximum likelihood trusts the data alone. If we also hold a belief about θ before seeing the data, a **prior** $p(\theta)$, Bayes' theorem combines the two into the **posterior** $p(\theta \mid \mathcal{D}) \propto p_{\text{model}}(\mathcal{D}; \theta)\, p(\theta)$. Maximizing the posterior instead of the likelihood is the **maximum a posteriori** (MAP) estimate, and it adds a term $-\log p(\theta)$ to the loss. Section 8 shows that this term is a regularizer.

---

5. Regression → Mean Squared Error

For regression the output space is $\mathcal{Y} = I \subset \mathbb{R}$, and a natural model is a Gaussian centred on the prediction $f(x; \theta)$ with variance σ^2:

$$p_{\text{model}}(y \mid x; \theta) = \mathcal{N}(y;\, \mu = f(x; \theta),\, \sigma^2) = \frac{1}{(2\pi\sigma^2)^{1/2}} \exp\left[-\frac{(y - f(x; \theta))^2}{2\sigma^2} \right]$$

Plugging this into the maximum likelihood estimator and taking the log of each term gives

$$\hat{\theta} = \arg\max_{\theta} \sum_{i=1}^{P} \left[-\log \sigma - \frac{1}{2} \log 2\pi - \frac{(y^{(i)} - f(x^{(i)}; \theta))^2}{2\sigma^2} \right]$$

If σ is constant, the first two terms do not depend on θ, and the factor $1/(2\sigma^2)$ is just a positive constant. Maximizing the remaining negative quantity is the same as minimizing its positive counterpart, which leaves the familiar mean squared error:

$$\boxed{\hat{\theta} = \arg\min_{\theta} \frac{1}{P} \sum_{i=1}^{P} \left(y^{(i)} - f(x^{(i)}; \theta)\right)^2 = \arg\min_{\theta} \mathrm{MSE}(\mathcal{D}; f)} \tag{1}$$

In short, **a Gaussian likelihood leads to the MSE loss.**

> Choosing the MSE therefore means assuming Gaussian noise around $f(x; \theta)$, with all the implicit assumptions of the Gaussian of Lecture 1. The Gaussian is a second-order expansion around the most likely value: it keeps only a mean and a variance (here the same σ^2 for every x) and forgets everything else. It describes the sum of many small, independent errors well, but not rare events far in the tails, skewed or heavy-tailed noise, correlated errors or bimodal outputs. In those regimes the MSE is the wrong loss: a single outlier, with its squared residual, can dominate the whole sum. Seen from the other side, among all distributions with a given mean and variance the Gaussian has maximum entropy: if mean and variance are all we know about the noise, it is the least committal choice.

---

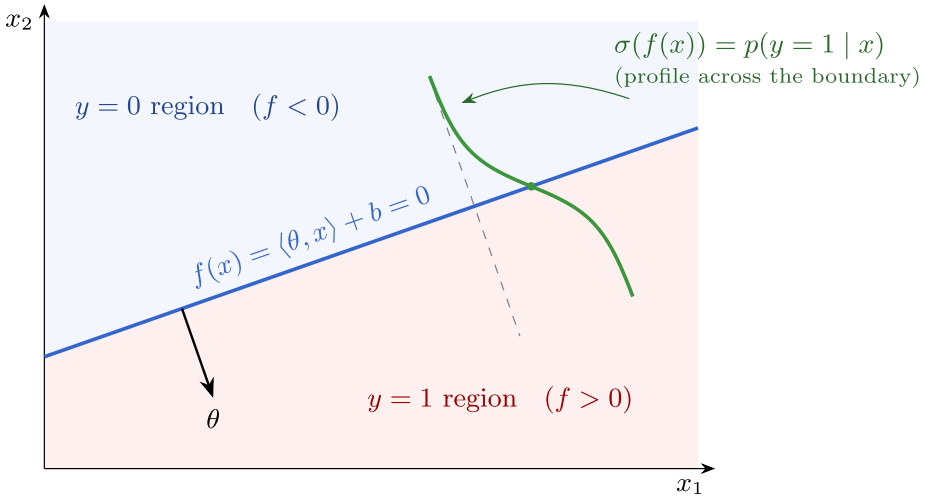
6. Binary classification → Binary Cross-Entropy

For binary classification the output space is $\mathcal{Y} = \mathbb{Z}_2 = \{0, 1\}$, and we model the output as a Bernoulli variable:

$$p_{\text{model}}(y \mid x; \theta) = [\sigma(f(x; \theta))]^y [1 - \sigma(f(x; \theta))]^{1-y}$$

This has the form $\alpha^y (1-\alpha)^{1-y}$, so $\alpha = \sigma(f(x; \theta))$ is the probability that x belongs to class $y = 1$. The function σ is the **logistic function**: its job is to map the real-valued score f into a probability, which is why in this context it is "probability-inspired" (the probability bucket).

To picture this, take $\mathcal{X} = \mathbb{R}^2$ and $f = \langle \theta, x \rangle + b$.



The line $f(x) = 0$ is the decision boundary, and the parameter vector θ is normal to it. On the side where $f < 0$ the model favours $y = 0$; on the side where $f > 0$ it favours $y = 1$. Moving across the boundary along θ, the probability $\sigma(f(x))$ of belonging to class $y = 1$ follows the green sigmoid profile: it passes through $\frac{1}{2}$ exactly on the boundary and saturates to 0 or 1 far away from it. Indeed, $\langle \theta, x \rangle + b$ is the signed distance from the plane $\langle \theta, x \rangle + b = 0$, multiplied by $\|\theta\|$: the probability is **parametrized** by the distance from a **plane**, and we divide the two classes using **linear functions**.

Taking the negative log-likelihood of this model gives

$$\hat{\theta} = \arg\min_{\theta} -\sum_{i=1}^{P} \Big[\underbrace{y^{(i)} \log \sigma\big(f(x^{(i)};\theta)\big)}_{(a)} + \underbrace{(1-y^{(i)}) \log\big(1-\sigma(f(x^{(i)};\theta))\big)}_{(b)} \Big] = \arg\min_{\theta} \mathrm{BCE}(\mathcal{D}, f)$$

This is the **Binary Cross-Entropy loss**, which goes by that name even though we obtained it from maximum likelihood.

The expression can be made more compact with a few tricks. First, relabel the classes from $\{0, 1\}$ to $\{-1, +1\}$. Then notice that term (a) only contributes when $y = +1$ and term (b) only when $y = -1$. Finally, the symmetry of the logistic function, $\sigma(-s) = 1 - \sigma(s)$, gives $\log(1 - \sigma(f)) = \log \sigma(-f)$, so both terms take the same form $\log \sigma(y \cdot f)$:

$$\boxed{\hat{\theta} = \arg\min_{\theta} \frac{1}{P} \sum_{i=1}^{P} -\log \sigma\big(y^{(i)} f(x^{(i)};\theta)\big)} \tag{2}$$

In short, **a Bernoulli likelihood leads to the BCE loss.**

---

7. Linear regression: closed-form solution

Now take a linear model, $f(x;\theta) = \langle \theta, x \rangle$, with $\theta \in \mathbb{R}^d$, $x^{(i)} \in \mathbb{R}^d$ and $y^{(i)} \in \mathbb{R}$. Stacking the inputs as rows of a design matrix and the outputs into a vector,

$$X = \begin{pmatrix} —x^{(1)}— \\ —x^{(2)}— \\ \vdots \\ —x^{(P)}— \end{pmatrix} \in \mathbb{R}^{P \times d}, \qquad Y = \begin{pmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(P)} \end{pmatrix} \in \mathbb{R}^{P},$$

the MSE loss (1) can be written compactly as

$$\frac{1}{P} \|X\theta - Y\|_2^2 = \frac{(X\theta - Y)^T (X\theta - Y)}{P}$$

To find its minimum we set the gradient to zero:

$$\nabla_\theta \left[\frac{(X\theta - Y)^T (X\theta - Y)}{P} \right] = \frac{X^T(X\theta - Y) + (X\theta - Y)^T X}{P} = \frac{X^T X\theta - X^T Y + \theta^T X^T X - Y^T X}{P} = 0$$

The two halves of the numerator are transposes of each other, so the condition reduces to the **normal equations** $X^T X\theta = X^T Y$, whose solution is

$$\boxed{\hat{\theta} = \underbrace{(X^T X)^{-1} X^T}_{X^+ \text{ (pseudo-inverse)}} Y}$$

Open question: how do the properties of $X^T X$ depend on the ratio P/d?

You can explore this question in least_squares_XtX.html (open it in a browser). Tab 1 shows what happens to the eigenvalues of X^TX, and to $\hat{\theta}$, as P crosses d. Tab 3 shows how the test error depends on P/d.

---

8. Regularized least squares

The closed form of Section 7 needs X^TX to be invertible. This fails when $P < d$, and the result is unstable when X^TX is nearly singular. The standard fix is to penalize the norm of θ to break the degeneracy in the solution space (without considering implicit regularization):

$$\boxed{\hat{\theta}_\lambda = \arg\min_{\theta\in\mathbb{R}^d} \frac{1}{P}\|X\theta - Y\|_2^2 + \lambda\|\theta\|_2^2}, \qquad \lambda \geq 0.$$

This is **regularized least squares** (RLS), also called ridge regression or Tikhonov regularization. The empirical error is our proxy for the expected error $\mathbb{E}_{p_{\text{data}}}[(y - \langle\theta, x\rangle)^2]$, which we cannot compute. The **regularizer** $\|\theta\|^2$ keeps the solution stable, and λ balances the two terms. The gradient is $-\frac{2}{P}X^T(Y - X\theta) + 2\lambda\theta$, and setting it to zero gives the regularized normal equations

$$\boxed{(X^TX + \lambda P\, I)\,\hat{\theta}_\lambda = X^TY}$$

For $\lambda = 0$ we are back to Section 7. For $\lambda > 0$ the matrix is symmetric and positive definite, therefore system can be solved.

Probabilistic view: a Gaussian prior. The MSE came from a Gaussian likelihood (Section 5). The penalty comes from a Gaussian **prior** on the parameters, centred at zero with variance τ^2:

$$p_{\text{model}}(y \mid x;\theta) = \mathcal{N}(y;\, \langle\theta, x\rangle,\, \sigma^2), \qquad p(\theta) = \mathcal{N}(\theta;\, 0,\, \tau^2 I) = \frac{1}{(2\pi\tau^2)^{d/2}} \exp\left[-\frac{\|\theta\|^2}{2\tau^2}\right].$$

The MAP estimate of Section 4 maximizes $p(\theta \mid \mathcal{D}) \propto \prod_i p_{\text{model}}(y^{(i)} \mid x^{(i)};\theta)\, p(\theta)$. Taking minus the log and dropping the constants,

$$-\log p(\theta \mid \mathcal{D}) = \frac{1}{2\sigma^2}\|X\theta - Y\|_2^2 + \frac{1}{2\tau^2}\|\theta\|_2^2 + \text{const},$$

and multiplying by $2\sigma^2/P$, which does not move the minimum, gives exactly the RLS loss:

$$\boxed{\hat{\theta}_\lambda = \arg\max_\theta\, p(\theta \mid \mathcal{D}), \qquad \lambda = \frac{\sigma^2}{P\,\tau^2}}$$

RLS is **Gaussian likelihood + Gaussian prior**, and λ is set by the ratio of the two variances. A flat prior ($\tau \to \infty$) gives $\lambda \to 0$ and the maximum likelihood solution of Section 7. A narrow prior, i.e. a strong belief that θ is small, gives a large λ. Noisy data (large σ^2) lean more on the prior, and more data (large P) lean less: the prior matters most when the data are few or unreliable.

> The prior carries its own implicit assumptions, like the likelihood. A Gaussian centred at zero says that every component of θ is small, and equally so in every direction. Another prior gives another regularizer: a Laplace prior $p(\theta) \propto e^{-\|\theta\|_1/b}$ gives the penalty $\lambda\|\theta\|_1$ (the **lasso**), which pushes many components exactly to zero.

Interlude: linear systems. Why does λ help? Take a symmetric positive semi-definite M, like X^TX, with eigendecomposition $M = VHV^T$, where $H = \text{diag}(\eta_1, \ldots, \eta_d)$ and $VV^T = I$. The solution of $Ma = b$ is $a = M^{-1}b$, with

$$M^{-1} = V\,\text{diag}\Big(\frac{1}{\eta_1}, \ldots, \frac{1}{\eta_d}\Big)V^T, \qquad (M + \lambda I)^{-1} = V\,\text{diag}\Big(\frac{1}{\eta_1 + \lambda}, \ldots, \frac{1}{\eta_d + \lambda}\Big)V^T.$$

Small eigenvalues make M^{-1} blow up: noise in b along those directions is hugely amplified. A zero eigenvalue, always present when $P < d$ since $\operatorname{rank} X^T X \leq P$, makes M not invertible at all. Adding λ shifts every eigenvalue up by λ, so the inverse always exists, and the **condition number** $\eta_{\max}/\eta_{\min}$ improves to $(\eta_{\max}+\lambda)/(\eta_{\min}+\lambda)$. This answers the open question of Section 7: λ controls the invertibility and stability of the problem.

Tab 2 of the companion page shows this shift and its effect on $\hat{\theta}_\lambda$ and on the test error.

> **Open question:** λ makes the problem well posed, but it also pulls $\hat{\theta}_\lambda$ away from the least-squares solution, towards zero. What do we pay and gain? We come back to this in Lecture 3, Section 3.

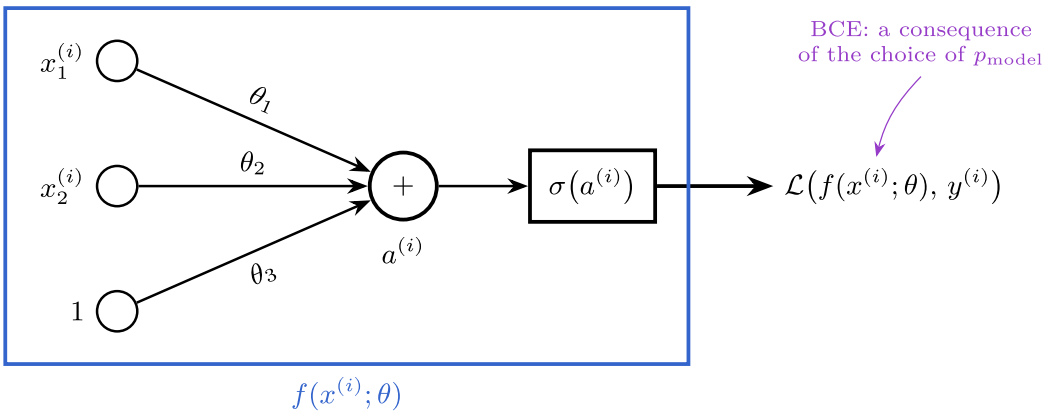
Like the learning rate of Section 10, λ is not fitted by the algorithm itself. It is an **hyperparameter**.

9. Linear classification: logistic regression

Logistic regression applies loss (2), which comes from a Bernoulli p_{model}, to a linear score $f = \langle \theta, x \rangle$. A very common trick is to append a constant 1 to every input, $x \to (1, x)$, so that the first component of θ plays the role of the bias b and no separate bias term is needed. Since $\sigma(s) = 1/(1+e^{-s})$, we have $-\log \sigma(s) = \log(1+e^{-s})$, and loss (2) becomes

$$\boxed{\mathcal{L}(\mathcal{D}, \theta) = \frac{1}{P} \sum_{i=1}^{P} \log\left(1 + e^{-y^{(i)} \langle \theta, x^{(i)} \rangle}\right)}$$

For a single pattern $x^{(i)} = (x_1^{(i)}, x_2^{(i)})$ in $\mathbb{R}^2$, the computation can be drawn as a graph. Here we append the constant 1 at the end of the input, so that the third weight θ_3 plays the role of the bias b:



The inputs are weighted and summed into the **activation**

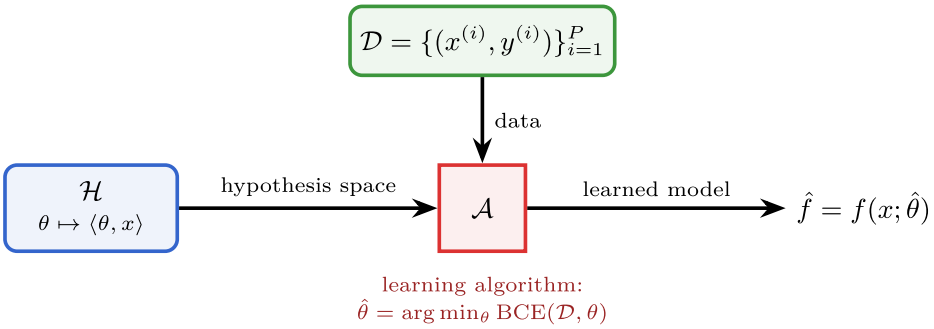
$$a^{(i)} = \theta_1 x_1^{(i)} + \theta_2 x_2^{(i)} + \theta_3 \cdot 1 = \langle \theta, (x^{(i)}, 1) \rangle,$$

which is passed through the logistic function. The result $\sigma(a^{(i)})$ is the model's probability that $y^{(i)} = +1$, and the blue box as a whole is the model $f(x^{(i)}; \theta)$. Its output goes into the loss $\mathcal{L}(f(x^{(i)};\theta), y^{(i)})$, which for labels in $\{-1, +1\}$ is $-\log \sigma(y^{(i)} a^{(i)})$, i.e. loss (2). The BCE is not an arbitrary choice: it is a **consequence of the choice of p_{model}**.

We compute this for every $i = 1, \ldots, P$ and minimize the average loss with respect to $\theta_1, \theta_2, \theta_3$. This gives $\hat{\theta}$:

$$\boxed{\hat{\theta} = \arg\min_{\theta} \frac{1}{P} \sum_{i=1}^{P} \mathcal{L}(f(x^{(i)};\theta),\, y^{(i)})}$$

Schematically, the learning algorithm $\mathcal{A}$ takes a hypothesis space and a dataset, and returns a learned model:



Here the algorithm $\mathcal{A}$ is: **minimize the loss function (binary cross-entropy) with respect to the parameters of the plane that divides the classes.**

What about **linear regression**? It is the *same* graph, with $\sigma = \mathbb{1}$ (the identity) and $\mathcal{L} = \mathrm{MSE}$, the loss that comes from a Gaussian p_{model}.

Unlike linear regression, logistic regression has no closed-form minimizer. The gradient $\nabla_\theta \mathcal{L}$ is explicit, however. Since $\frac{d}{ds}\log(1+e^{-s}) = -\frac{e^{-s}}{1+e^{-s}} = -\sigma(-s)$, the chain rule with $s = y^{(i)}\langle \theta, x^{(i)} \rangle$ gives

$$\boxed{\nabla_\theta \mathcal{L} = -\frac{1}{P}\sum_{i=1}^{P} \sigma\big(-y^{(i)}\langle \theta, x^{(i)} \rangle\big)\, y^{(i)} x^{(i)}}$$

Each pattern pulls θ towards $y^{(i)}x^{(i)}$, weighted by $\sigma(-y^{(i)}\langle \theta, x^{(i)} \rangle)$, the probability the model gives to the *wrong* class. A point classified correctly and far from the boundary contributes almost nothing; a misclassified one contributes almost fully. With labels back in $\{0, 1\}$, the same gradient reads $\frac{1}{P}\sum_i \big(\sigma(\langle \theta, x^{(i)} \rangle) - y^{(i)}\big)\, x^{(i)}$: prediction minus target, times input. This is the form of the linear regression gradient of Section 7, $\frac{2}{P}X^T(X\theta - Y)$, with σ in place of the identity. It is the same graph again. What changes is that θ now sits inside σ, so setting the gradient to zero no longer gives a linear system. Instead we find the solution iteratively with **gradient descent**, the subject of the next section.

---

10. Optimizing the loss: gradient descent

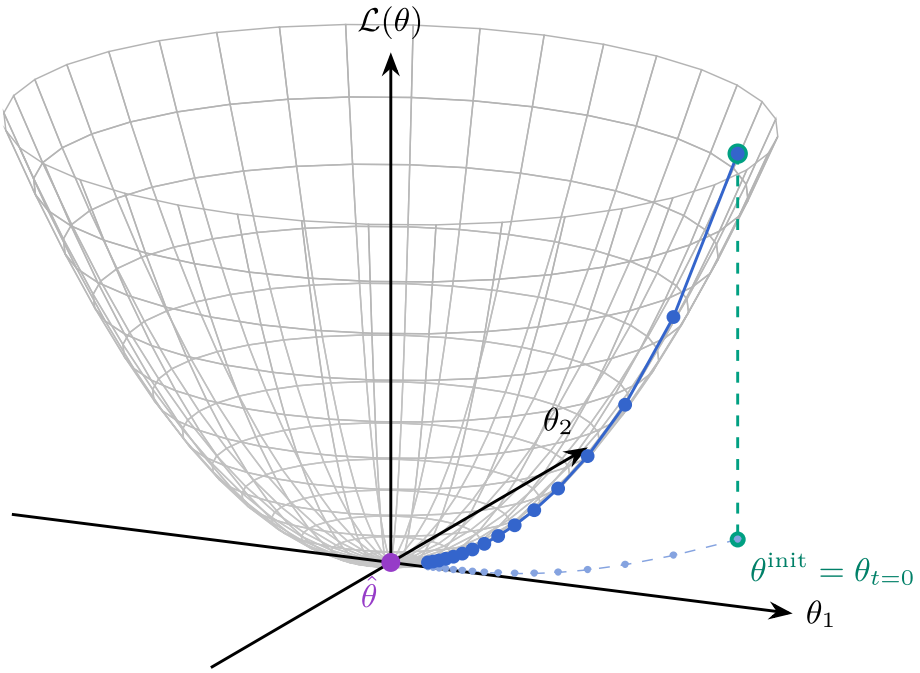
How do we actually find $\hat{\theta}$? For linear regression we could use the closed formula of Section 7, but logistic regression has none. The general tool is **gradient descent**. Start from an initial guess

$$\theta^{\text{init}} = (\theta_1^{\text{init}}, \theta_2^{\text{init}}, \theta_3^{\text{init}}) = \theta_{t=0},$$

and repeatedly take a step downhill, against the gradient of the loss:

$$\boxed{\theta_{t+1} = \theta_t - \alpha\, \nabla_\theta \frac{1}{P}\sum_{i=1}^{P} \mathcal{L}[f(x^{(i)}; \theta_t),\, y^{(i)}]}$$

The step size $\alpha > 0$ is the **learning rate**. Since $-\nabla_\theta \mathcal{L}$ is the direction in which the loss decreases fastest, each step lowers the loss as long as α is small enough.



If the problem is **convex**, every θ^{init} eventually leads to the same $\hat{\theta}$, up to numerical error and a proper choice of α. The loss surface is a single bowl with no other valleys to get stuck in. Both the MSE of linear regression and the BCE of logistic regression are convex in θ.

> The learning rate matters. If α is too small, convergence is very slow. If it is too large, the steps overshoot the bottom of the bowl, and the iterates oscillate or even diverge. Note also that α is *not* learned by gradient descent itself: it is a **hyperparameter**, and choosing it is the subject of Lecture 3, Section 5.

---

11. Summary

Problem	Output space	Likelihood model	Loss	Solution (linear f)
Regression	$\mathcal{Y} \subset \mathbb{R}$	Gaussian $\mathcal{N}(f(x;\theta), \sigma^2)$	MSE (1)	Closed form: $\hat{\theta} = (X^T X)^{-1} X^T Y$
Regularized regression (RLS)	$\mathcal{Y} \subset \mathbb{R}$	Gaussian $\mathcal{N}(f(x;\theta), \sigma^2)$, Gaussian prior $\mathcal{N}(0, \tau^2 I)$ on θ	MSE $+ \lambda\|\theta\|^2$, $\lambda = \sigma^2/(P\tau^2)$	Closed form: $\hat{\theta}_\lambda = (X^T X + \lambda P\, I)^{-1} X^T Y$
Binary classification	$\mathcal{Y} = \mathbb{Z}_2$	Bernoulli with $\sigma(f(x;\theta))$	BCE (2)	Iterative: gradient descent

An ML algorithm is defined by its task, performance measure and experience. Models are searched for in a hypothesis space, where linear combinations of features $\phi(x)$ are a powerful idea underlying kernel methods and neural networks. Loss functions are not arbitrary: they follow from maximum likelihood, which is equivalent to minimizing $\mathrm{KL}[p_{\text{data}} \| p_{\text{model}}]$. Adding a penalty $\lambda\|\theta\|^2$ to least squares makes the closed form always well posed; it is the MAP estimate with a Gaussian likelihood and a Gaussian prior on θ. Linear classification and regression are the same computational graph, with different σ and loss. The loss is minimized by gradient descent, which for convex problems reaches the same $\hat{\theta}$ from any starting point.

References

- I. Goodfellow, Y. Bengio and A. Courville, *Deep Learning*, MIT Press, 2016, available online.
- L. Rosasco, *Introductory Machine Learning Notes* (for MIT course), University of Genova.
- T. M. Mitchell, *Machine Learning*, McGraw-Hill, 1997.