Machine Learning Fundamentals - LS2026
Deep Learning and Generalization

Czech Technical University in Prague
V. Franc

The linear model

A linear model maps input $\mathbf{x} \in \mathbb{R}^d$ to output via

$$f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + b$$

- Regression: $\hat{y} = f(\mathbf{x})$;
- Classification: $\hat{y} = \sigma(f(\mathbf{x}))$ for some non-linearity σ (e.g. sigmoid, sign).

Non-linear data mapping: Increase expressive power of linear model:

$$f(\mathbf{x}) = \mathbf{w}^\top \boldsymbol{\phi}(\mathbf{x}) + b$$

Linear in $\boldsymbol{\phi}(\mathbf{x})$, possibly non-linear in $\mathbf{x}$.

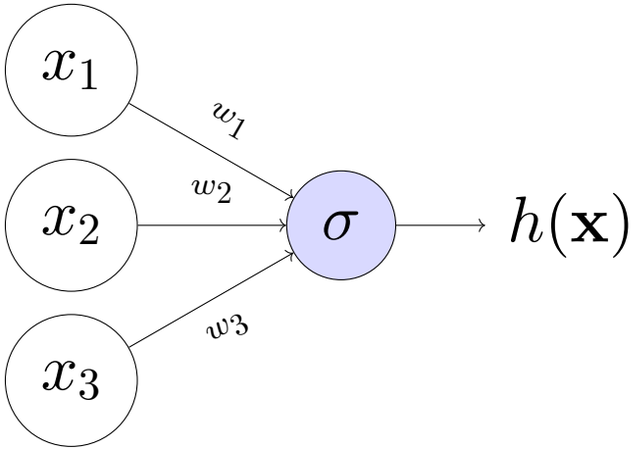
- **Kernel methods:** $\boldsymbol{\phi}$ is fixed implicitly, via kernel k.
- **Classical ML:** $\boldsymbol{\phi}$ is hand-engineered (SIFT, MFCC, n-grams, . . .).

Key question: Can we *learn* the feature map $\boldsymbol{\phi}$ from data, instead of designing it?

The single neuron

A neuron is a linear model followed by a non-linearity:

$$h(\mathbf{x}) = \sigma\left(\mathbf{w}^\top \mathbf{x} + b\right)$$



Common choices for σ: ReLU $\max(0, z)$, sigmoid, tanh.

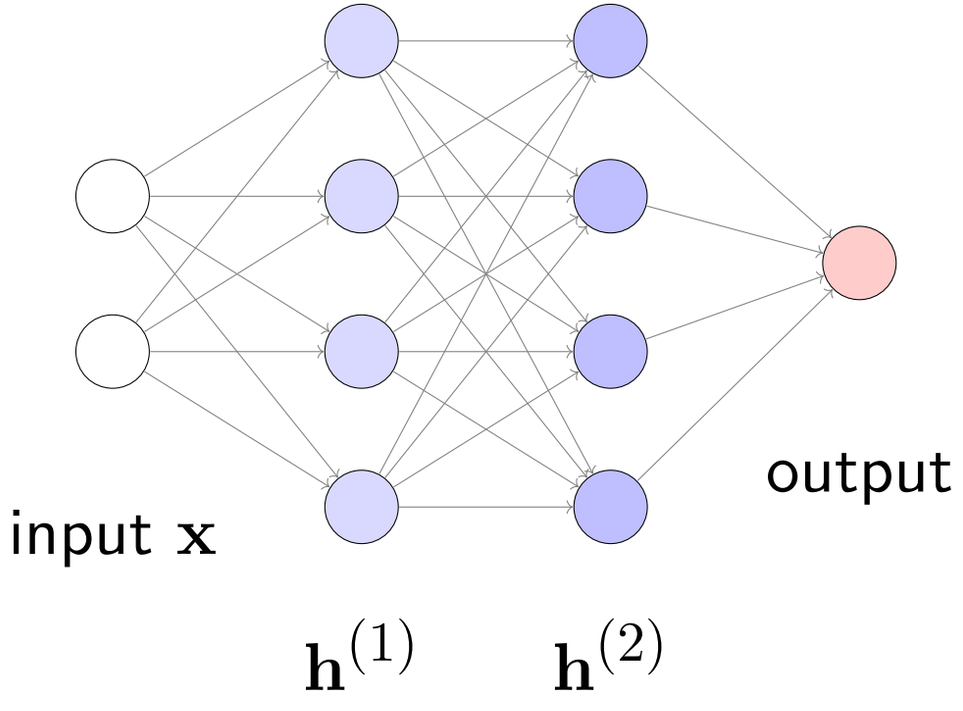
One neuron alone = linear model. The power comes from *stacking*.

Feed forward neural network

A feedforward network with L layers:

$$\mathbf{h}^{(l)} = \sigma\left(\mathbf{W}^{(l)}\mathbf{h}^{(l-1)} + \mathbf{b}^{(l)}\right), \qquad l = 1, \ldots, L$$

with $\mathbf{h}^{(0)} = \mathbf{x}$ and final output $f(\mathbf{x}) = \mathbf{h}^{(L)}$.



Depth = repeated composition of blocks (linear + non-linearity).

Deep learning as learned feature extraction

Rewrite the network as

$$f(\mathbf{x};\boldsymbol{\theta}) = \underbrace{\mathbf{W}^{(L)}}_{\text{linear classifier}} \cdot \underbrace{\boldsymbol{\phi}(\mathbf{x};\boldsymbol{\theta})}_{\text{learned feature map}} + \mathbf{b}^{(L)}$$

where $\boldsymbol{\phi}(\mathbf{x};\boldsymbol{\theta}) = \mathbf{h}^{(L-1)}$ is the output of the penultimate layer.

Key insight: A deep network is a linear model on top of a learned feature representation.

The lower layers learn $\boldsymbol{\phi}$; the last layer is just a linear classifier / regressor.

Expressive power

Universal approximation theorem (Cybenko 1989; Hornik 1991):

A feedforward network with one hidden layer and a non-polynomial activation can approximate any continuous function on a compact set, to arbitrary accuracy, given enough width.

- Width alone suffices in principle (e.g. kernel SVM/regression).
- In practice, *depth* is much more parameter-efficient: deep networks express functions that would need exponentially wide shallow networks.

Training neural networks

Empirical risk minimization: Given training data $T_m = ((\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m))$ find parameters $\boldsymbol{\theta} \in \mathbb{R}^d$ by solving:

$$\min_{\boldsymbol{\theta}} \frac{1}{m} \sum_{i=1}^{m} \ell\big(f(\mathbf{x}_i; \boldsymbol{\theta}), y_i\big)$$

Same skeleton as linear models like SVMs or logistic regression.

Differences:

- Learning objective is *non-convex* in $\boldsymbol{\theta}$ – no global guarantee.
- Optimized with stochastic gradient descent + backpropagation.
- Gradients computed by chain rule through the layers.

Despite non-convexity, SGD reliably finds good solutions in practice.

Gradient descent and stochastic gradient descent

Goal: minimize empirical risk

$$L(\boldsymbol{\theta}) = \frac{1}{m}\sum_{i=1}^{m} \ell\big(f(\mathbf{x}_i;\boldsymbol{\theta}), y_i\big).$$

Gradient descent (GD)

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta \nabla L(\boldsymbol{\theta}_t)$$

- Uses the *full* gradient: $\nabla L(\boldsymbol{\theta}) = \frac{1}{n}\sum_i \nabla \ell_i(\boldsymbol{\theta})$
- Deterministic; each step is exact but costs $\mathcal{O}(m)$.
- Smooth, monotone descent on convex losses.

Stochastic gradient descent (SGD)

$$\boldsymbol{\theta}_{t+1} = \theta_t - \eta \nabla \ell_{i_t}(\boldsymbol{\theta}_t)$$

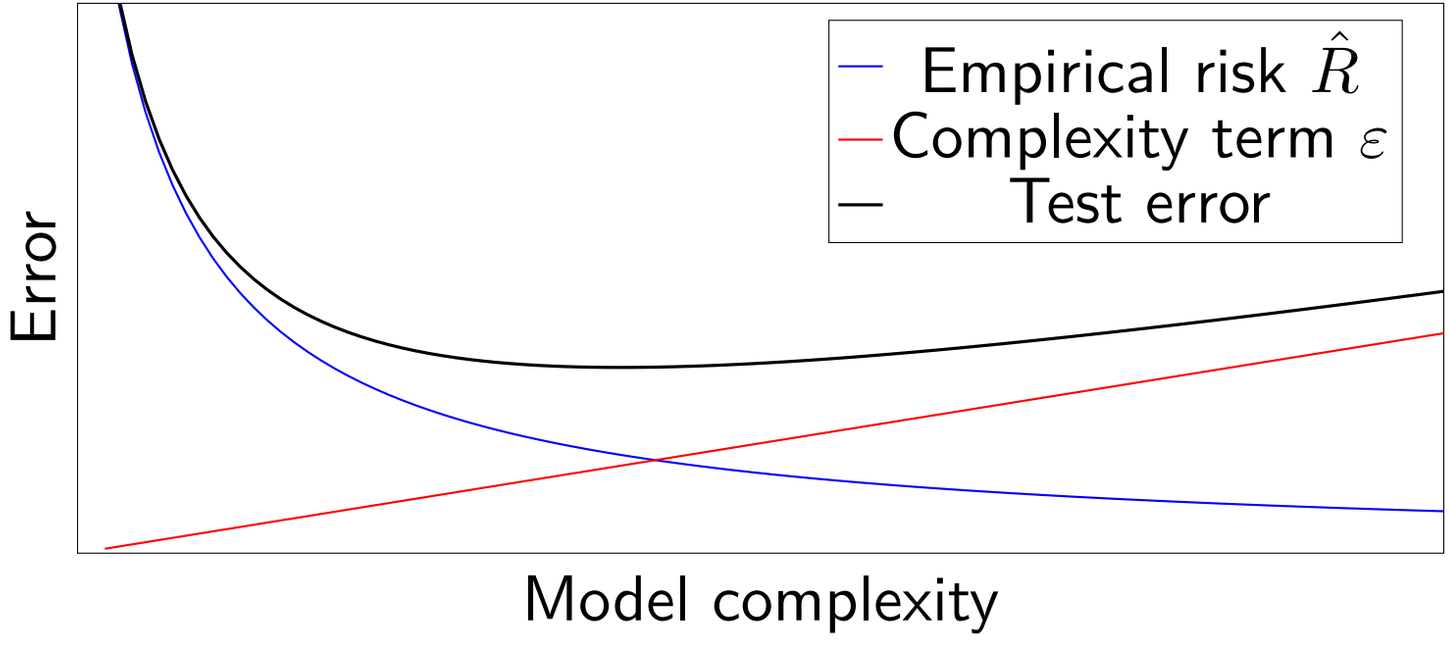
- Sample i_t (or a mini-batch B_t) at each step.
- Unbiased estimator: $\mathbb{E}\nabla \ell_{i_t} = \nabla L$.
- Each step is cheap ($\mathcal{O}(1)$ or $\mathcal{O}(|B|)$) and *noisy*.

Convergence requires either a decreasing step size $\eta_t \to 0$ (Robbins–Monro: $\sum \eta_t = \infty$, $\sum \eta_t^2 < \infty$) or iterate averaging.

The classical generalization theory

The VC generalization bound: with probability at least $1-\delta$:

$$R(h,p) \leq \hat{R}(T_m,h) + \varepsilon\left(\mathrm{VCdim}(\mathcal{H}), \frac{1}{m}, \frac{1}{\delta}\right)$$



- Generalization depends on hypothesis class $\mathcal{H}$ and the number of training examples m.
- To complex hypothesis class $\mathcal{H} \Rightarrow$ overfitting and poor generalization.

Learning algorithms explained by the classical gen. theory

Structural Risk Minimization Construct a nested sequence of hypothesis classes $\mathcal{H}_1 \subset \mathcal{H}_2 \subset \cdots \subset \mathcal{H}_K$, and choose capacity to balance fit and the class complexity:

$$\min_{k=1,\ldots,K} \min_{h \in \mathcal{H}_k} \left[\hat{R}(T_m, h) + \varepsilon\left(\mathrm{VCdim}(\mathcal{H}_k), \frac{1}{m}, \frac{1}{\delta}\right) \right]$$

Regularized risk minimization Select a parametrized hypothesis class $\mathcal{H} = \{h(\cdot;\boldsymbol{\theta}) \colon \mathcal{X} \to \mathcal{Y} \mid \boldsymbol{\theta} \in \Theta\}$ and learn the parameters by solving:

$$\min_{\boldsymbol{\theta}} \left[\frac{\lambda}{2} \|\boldsymbol{\theta}\|^2 + \hat{R}(T_m, \boldsymbol{\theta}) \right]$$

where $\hat{R}(T_m, \boldsymbol{\theta})$ is the empirical risk.

Support Vector Machines: $\mathcal{H} = \{h(x;\boldsymbol{\theta}) = \mathrm{sign}(\langle \boldsymbol{\theta}, \boldsymbol{\phi}(x) \rangle) \mid \boldsymbol{\theta} \in \Theta\}$ and $\hat{R}(T_m, \boldsymbol{\theta}) = \frac{1}{m} \sum_{i=1}^{m} \max\{0, 1 - y_i \langle \boldsymbol{\theta}, \boldsymbol{\phi}(x_i) \rangle\}$.

The puzzle of deep learning

Classical wisdom: More parameters than samples $\Rightarrow$ overfitting and poor generalization.

Modern deep networks often satisfy the following:

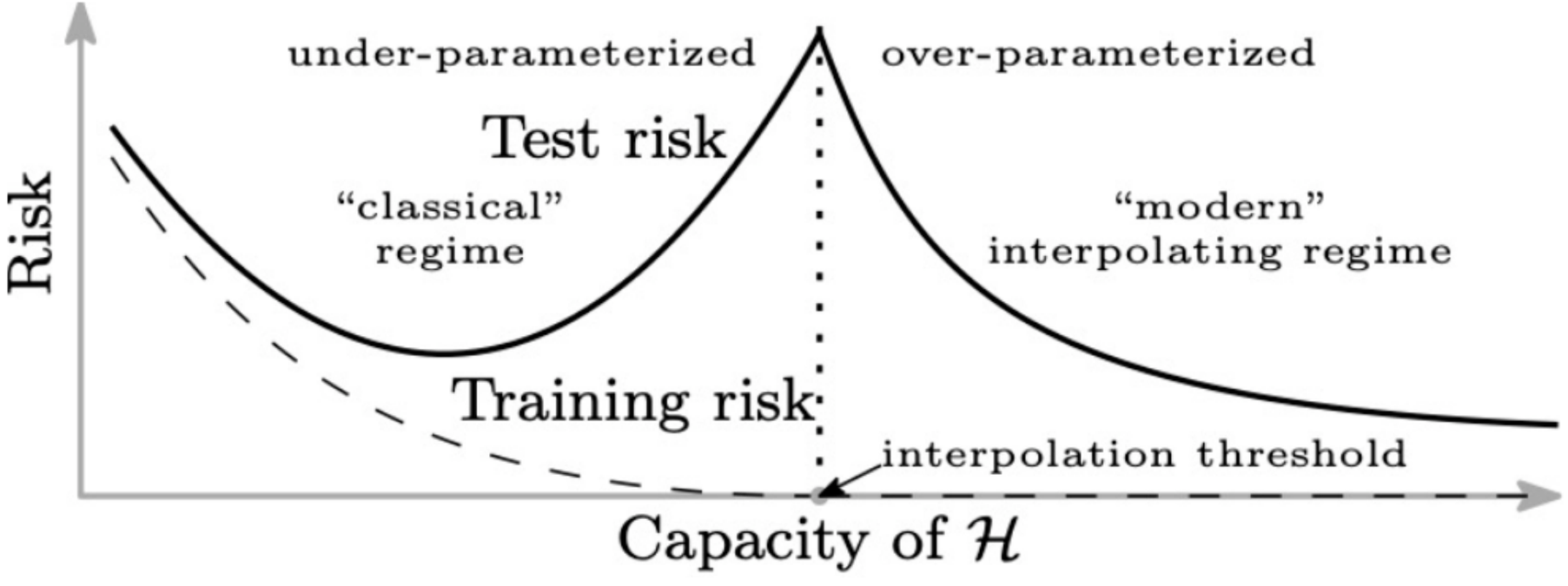
- The number of parameters is much larger than the number of training samples;
- The training error can be driven to nearly zero;
- The architecture has enough capacity to memorize random labels;
- The test performance can still be excellent.

Puzzle: If the class can memorize anything, why does it learn useful predictor ?

Remark: Recall the memorizing learning algorithm we used as an example of ERM failure.

Double descent
Belkin et al. 2019, Nakkiran et al. 2022

Phenomenon observed in many deep learning problems:



Overparameterization: A model is overparameterized when it has many more degrees of freedom than constrains. For neural networks often $p \gg m$.

Interpolation: A model interpolates the training set if it reaches zero empirical error $\hat{R}(T_m, h) = 0$.

The **interpolation threshold** is the point where the model becomes able to fit the training data exactly. DL models often operate beyond this threshold.

Explicit vs. implicit regularization

Explicit regularization

- Regularized Risk Minimization (e.g. SVM, LASSO, Ridge regression):

$$\min_{\boldsymbol{\theta}} \left[\frac{\lambda}{2} \|\boldsymbol{\theta}\|^2 + \hat{R}(T_m, \boldsymbol{\theta}) \right]$$

- Weight decay (L_2).
- Dropout.
- Data augmentation.
- Early stopping.

Zhang et al. 2017: *remove all explicit regularization* and networks still generalize.

Implicit regularization

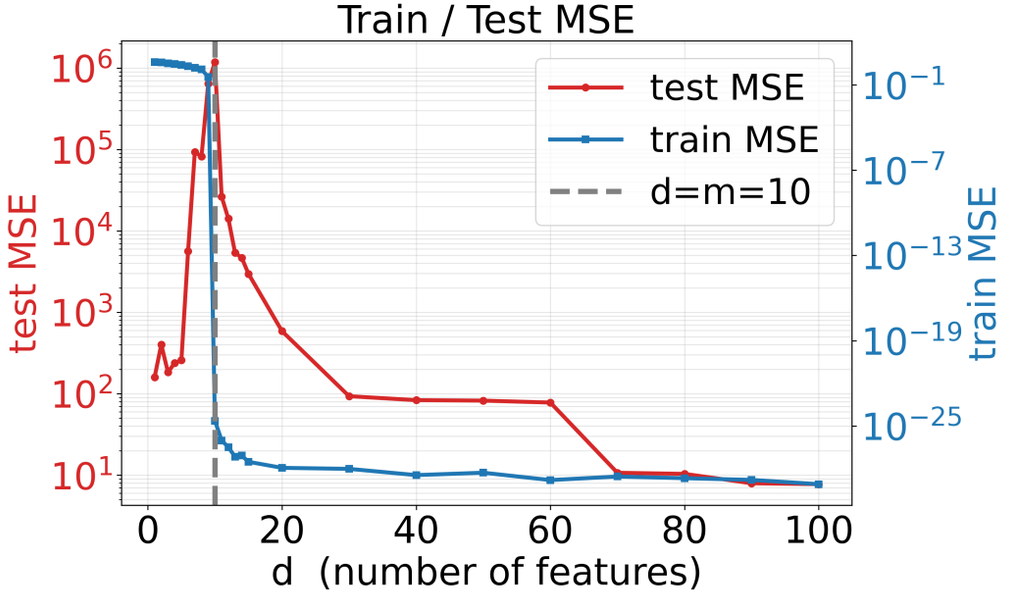
The optimizer and the architecture themselves bias learning toward "simple" solutions, even without explicit penalties.

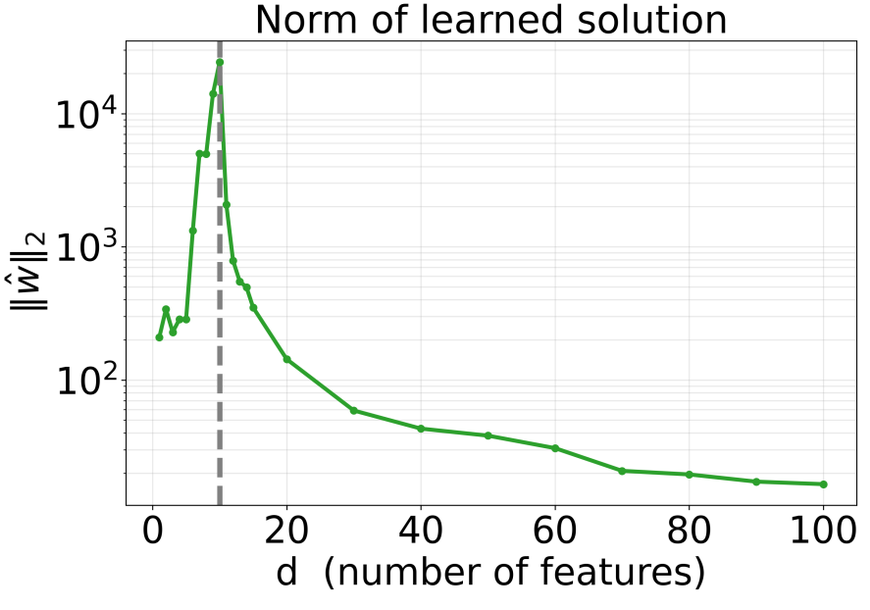
Double descent in linear models
Belkin et al. (2019); Hastie et al. (2022)

Double descent is *not* a deep-learning phenomenon — it appears in plain minimum-norm linear regression.

Setup: Given training data $(\mathbf{X}_m \in \mathbb{R}^{m\times d}, \mathbf{y}_m \in \mathbb{R}^m)$, learn the linear regression model $f(\mathbf{x}, \mathbf{w}) = \mathbf{x}^\top \mathbf{w}$ by solving the minimum-norm least squares:

$$\min_{\mathbf{w}} \|\mathbf{w}\| \qquad \text{s.t.} \qquad \mathbf{w} \in \arg\min_{\mathbf{w}'} \|\mathbf{y}_m - \mathbf{X}_m \mathbf{w}'\|^2$$





- $d = m$: unique interpolator forced to absorb all label noise.
- $d > m$: many interpolators; min-norm solution is smoother.

Overparameterized linear regression
Hastie et al. 2022

Overparameterized problem: If $\mathbf{X}_m \in \mathbb{R}^{m \times d}$ has a full rank and $d > m$ then there are infinitely many $\mathbf{w}$ such that $\|\mathbf{y}_m - \mathbf{X}_m \mathbf{w}\| = 0$.

Empirical Risk Minimization: Learning the linear regression model $f(\mathbf{x}, \mathbf{w}) = \mathbf{x}^\top \mathbf{w}$ involves:

$$\hat{\mathbf{w}}_{\text{OLS}} = \arg\min_{\mathbf{w}} \frac{1}{m} \sum_{i=1}^{m} (\mathbf{x}_i^\top \mathbf{w} - y_i)^2 = \arg\min_{\mathbf{w}} \|\mathbf{y}_m - \mathbf{X}_m \mathbf{w}\|^2 \qquad \text{(OLS)}$$

Minimum-norm least squares:

$$\hat{\mathbf{w}}_{\text{MNLS}} = \arg\min_{\mathbf{w}\,:\,\mathbf{X}_m^\top \mathbf{w} = \mathbf{y}_m} \|\mathbf{w}\| = \mathbf{X}_m^\top (\mathbf{X}_m \mathbf{X}_m^\top)^{-1} \mathbf{y}_m$$

Implicit regularization: Gradient descent from zero initialization applied to solving (OLS) converges to the minimum-norm interpolating solution $\hat{\mathbf{w}}_{\text{MNLS}}$.

Logistic regression on linearly separable data
Soudry et al. 2018

ML learning: Learning the logistic regression classifier $h(\mathbf{x}, \mathbf{w}) = \text{sign}(\mathbf{x}^\top \mathbf{w})$:

$$\hat{\mathbf{w}}_{\text{ML}} = \arg\min_{\mathbf{w}} \frac{1}{m} \sum_{i=1}^{m} \log(1 + \exp(-y_i \mathbf{w}^\top \mathbf{x}_i)) \qquad \text{(LR)}$$

Hard-Margin SVM: Learning the SVM classifier $h(\mathbf{x}, \mathbf{w}) = \text{sign}(\mathbf{x}^\top \mathbf{w})$:

$$\hat{\mathbf{w}}_{\text{SVM}} = \arg\min_{\mathbf{w}} \|\mathbf{w}\|^2 \quad \text{s.t.} \quad y_i \mathbf{w}^\top \mathbf{x}_i \geq 1 \,, \forall i \,.$$

Implicit regularization: Gradient descent applied to solve (ERM) converges to solution which has the same direction as the Hard-Margin SVM solution $\hat{\mathbf{w}}_{\text{SVM}}$:

Although $\|\mathbf{w}_t\| \to \infty$, the *direction* converges:

$$\frac{\mathbf{w}_t}{\|\mathbf{w}_t\|} \longrightarrow \frac{\hat{\mathbf{w}}_{\text{SVM}}}{\|\hat{\mathbf{w}}_{\text{SVM}}\|}$$

Does the implicit-bias story extend to deep networks?

Linear models: clean theorems. GD on OLS $\Rightarrow$ minimum-norm interpolating solution. GD on logistic loss $\Rightarrow$ max-margin solution.

Deep nonlinear networks: the picture is fragmentary.

- **Theory in restricted regimes:**
  - Homogeneous ReLU networks $\Rightarrow$ KKT points of max-margin.
  - Deep linear networks $\Rightarrow$ low-rank / low nuclear-norm bias.
- **Empirical support:**
  - Learned norms / margins correlate with generalization.
  - Margin grows during training, even after zero training error.
  - SGD finds solutions that can be drastically compressed.

Summary. No general theorem covers arbitrary architectures and optimizers. The implicit-bias story extends in *spirit* — with partial theory and substantial empirical support. This is an active open problem.

Architectural inductive bias

Choice of architecture *is* choice of the hypothesis class $\mathcal{H}$.

- **CNNs**: weight sharing $\Rightarrow$ translation equivariance across positions
- **RNNs**: weight sharing $\Rightarrow$ translation equivariance across time step
- **Transformers**: permutation equivariance + pairwise relational structure.
- **Graph NNs**: permutation equivariance over nodes.

Each architecture is a strictly smaller $\mathcal{H}$ than an unconstrained MLP of the same parameter budget. The architectures exploit symmetries and sharing patterns that match structure in the data.

Summary

- Classical theory (VC, SRM): generalization depends on the hypothesis class.
- Deep learning requires to refine the *theory*:
  - complexity should be measured by norms / margins, not parameter count;
  - generalization depends on the optimizer and the hypothesis class (architecture);
  - implicit bias is central.
- A complete predictive theory of deep generalization **does not yet exist**.
- This is one of the most active areas in modern ML theory.