Introduction to Regression

Srinandan (“Sri”) Dasmahapatra

COMP3223

Supervised Learning: labelled data

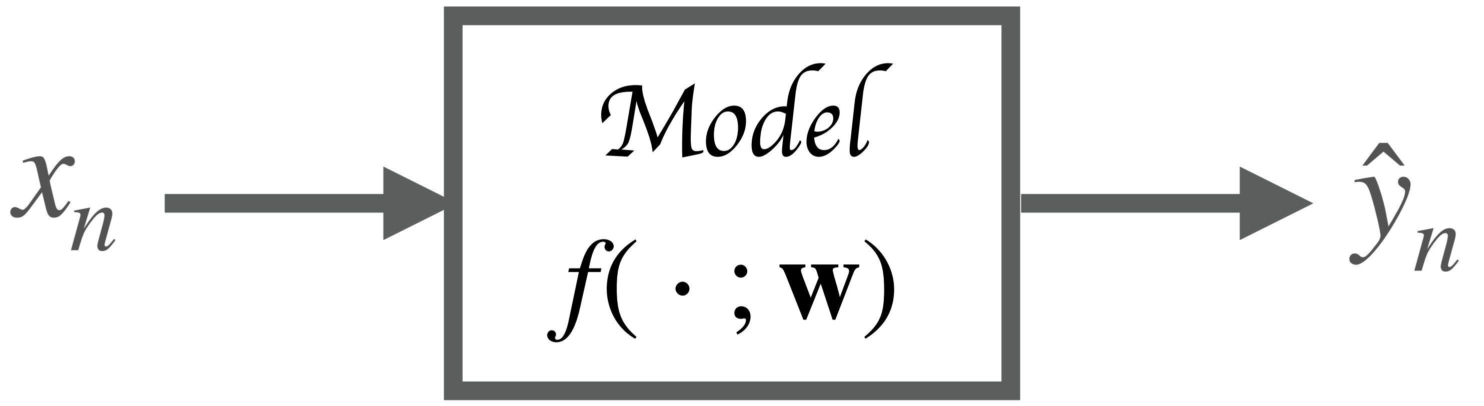
Compare labels with predictions

$d(\hat{y}_n, y_n)$: How far is prediction $\hat{y}_n$ from actual data y_n?

- Given data $\mathscr{D}$ construct model $f(\,\cdot\,; \mathbf{w})$ such that the "distance" between model output and real "output" is small

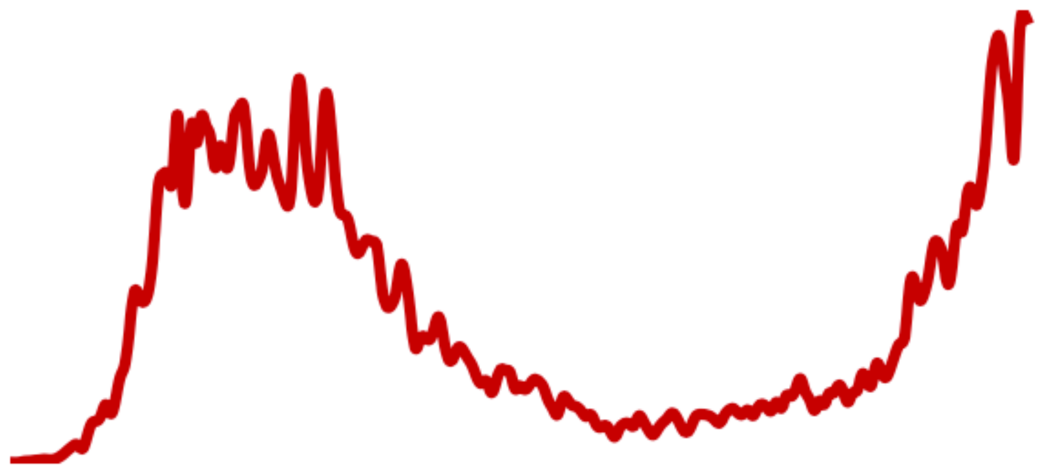
- Learning = model construction by **minimising** loss $L(\mathbf{w}) = \sum_{n=1}^{N} d(\hat{y}_n, y_n)$

$$\mathscr{D} := \{(x_n, y_n)\}, n = 1, \ldots, N$$



$$\hat{y}_n = f(x_n; \mathbf{w})$$

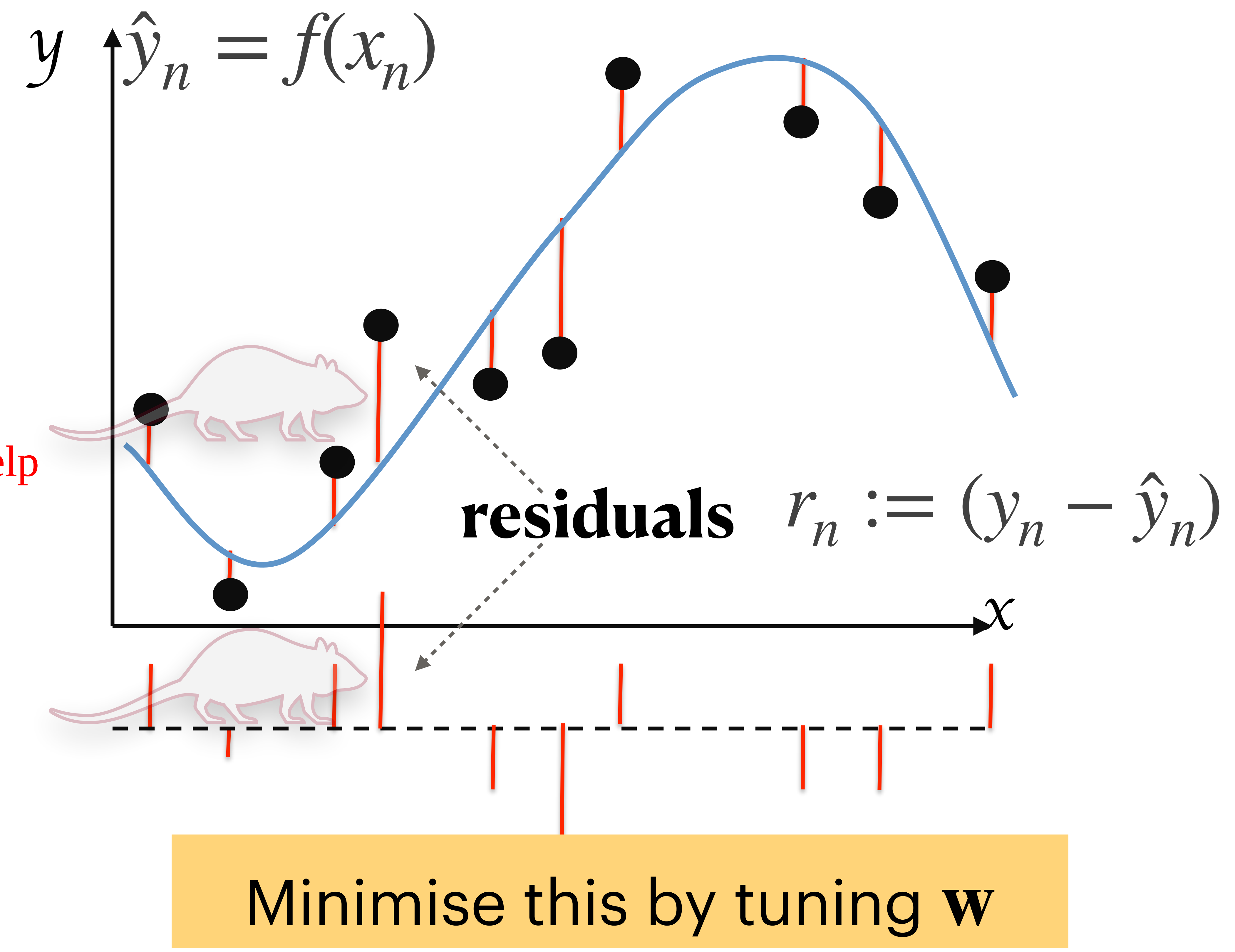
- $y = f(\,\cdot\,; \mathbf{w})$ continuous (*e.g.*, y=23.4), $f(\,\cdot\,; \mathbf{w})$ is a **regression** model



Core idea in ML: Reduce mismatch between model prediction and data

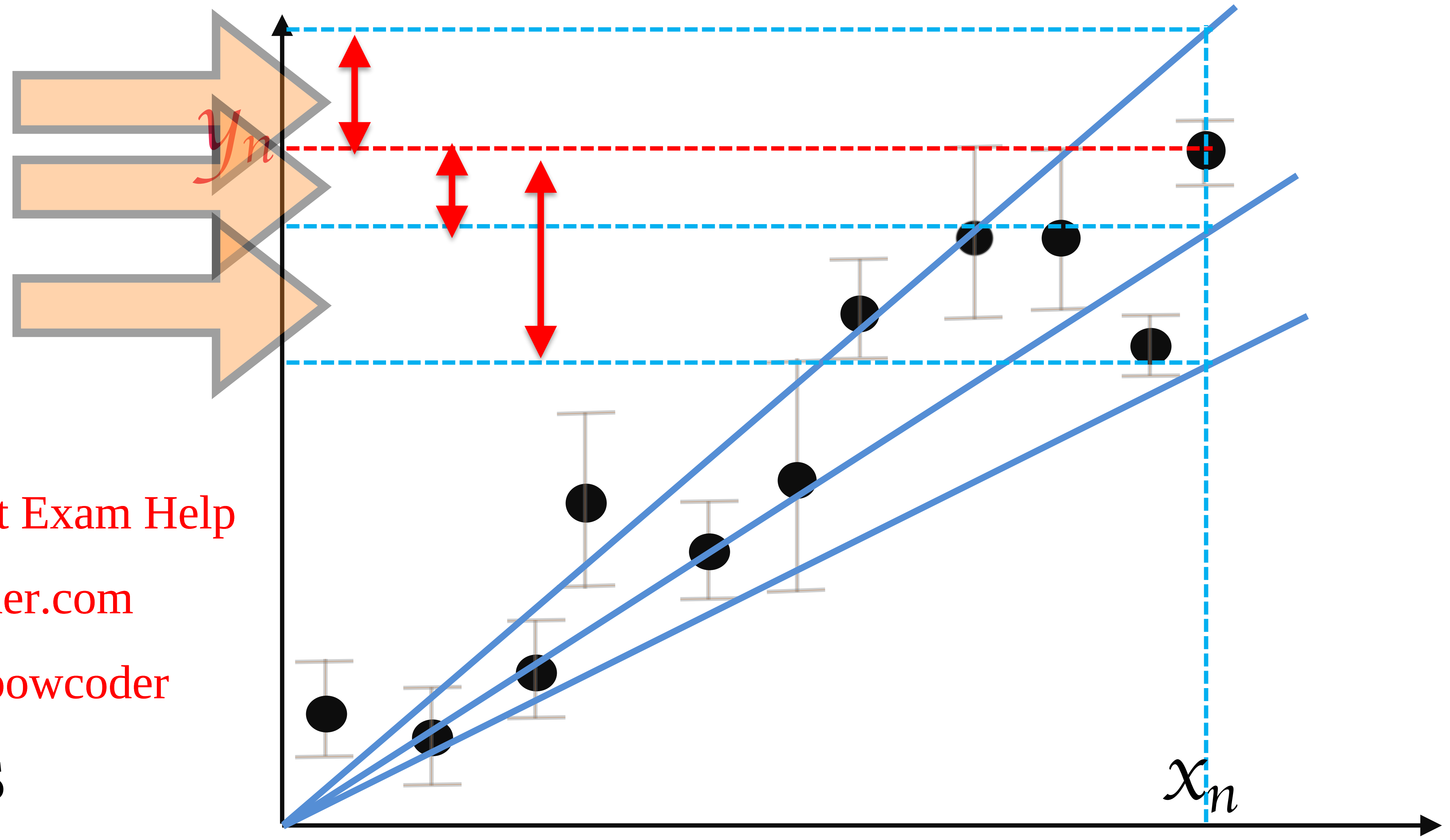
Squared residual loss

- Regression models minimise **residuals** — deviations of model predictions from outputs in training data
- $y_n = \hat{y}_n + \overbrace{(y_n - \hat{y}_n)}^{r_n} = f(x_n; \mathbf{w}) + r_n(\mathbf{w})$
- Contribution to loss function:
  $l_n(\mathbf{w}) = (y_n - f(x_n; \mathbf{w}))^2 = r_n^2(\mathbf{w})$.
- Average loss: $L(\mathbf{w}) = \frac{1}{N}\sum_{n=1}^{N} l_n(\mathbf{w})$



Choose weight for minimum loss

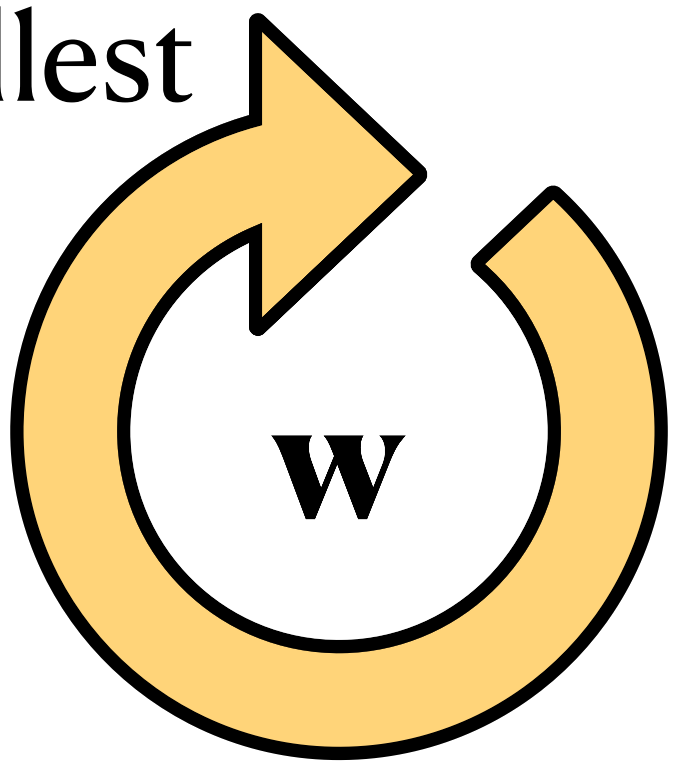
Example: find slope of straight line



- Fit $y=w\,x$ to data, slope w of the line is the weight/parameter to be learnt
- Loss L = sum of squares of residuals (in red), for 3 possible choices of slopes — $\{w_1, w_2, w_3\}$: the 3 residuals for input $\boldsymbol{x_n}$ is shown
- Choose the slope that gives the smallest value from $\{L(w_1), L(w_2), L(w_3)\}$

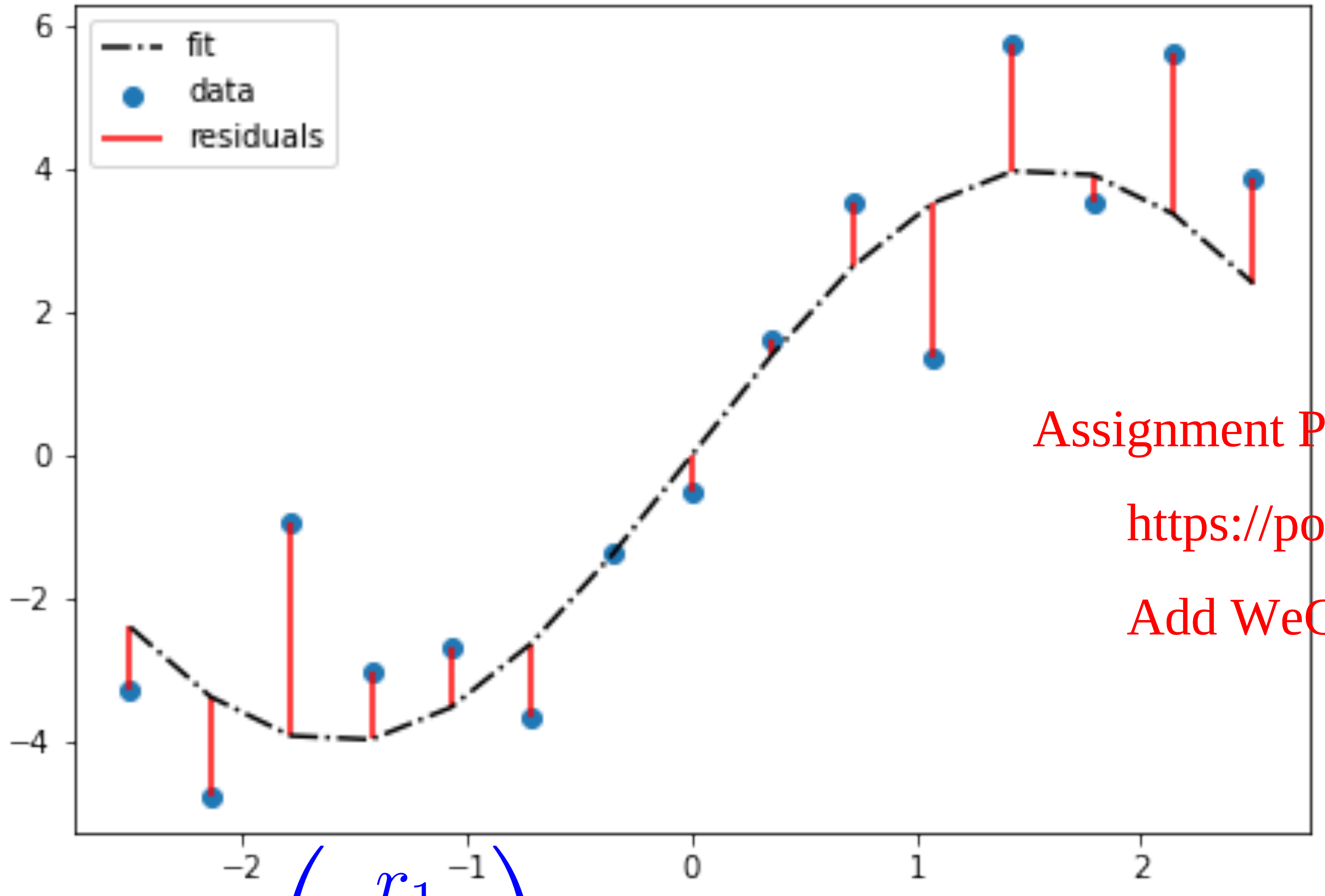
$$l_n(w) = r_n^2 = (wx_n - y_n)^2$$

$$L(w) = \frac{1}{N}\sum_{n=1}^{N} l_n(w)$$



Loss function needs a distance: introducing the norm

Treat all data points as collective unit



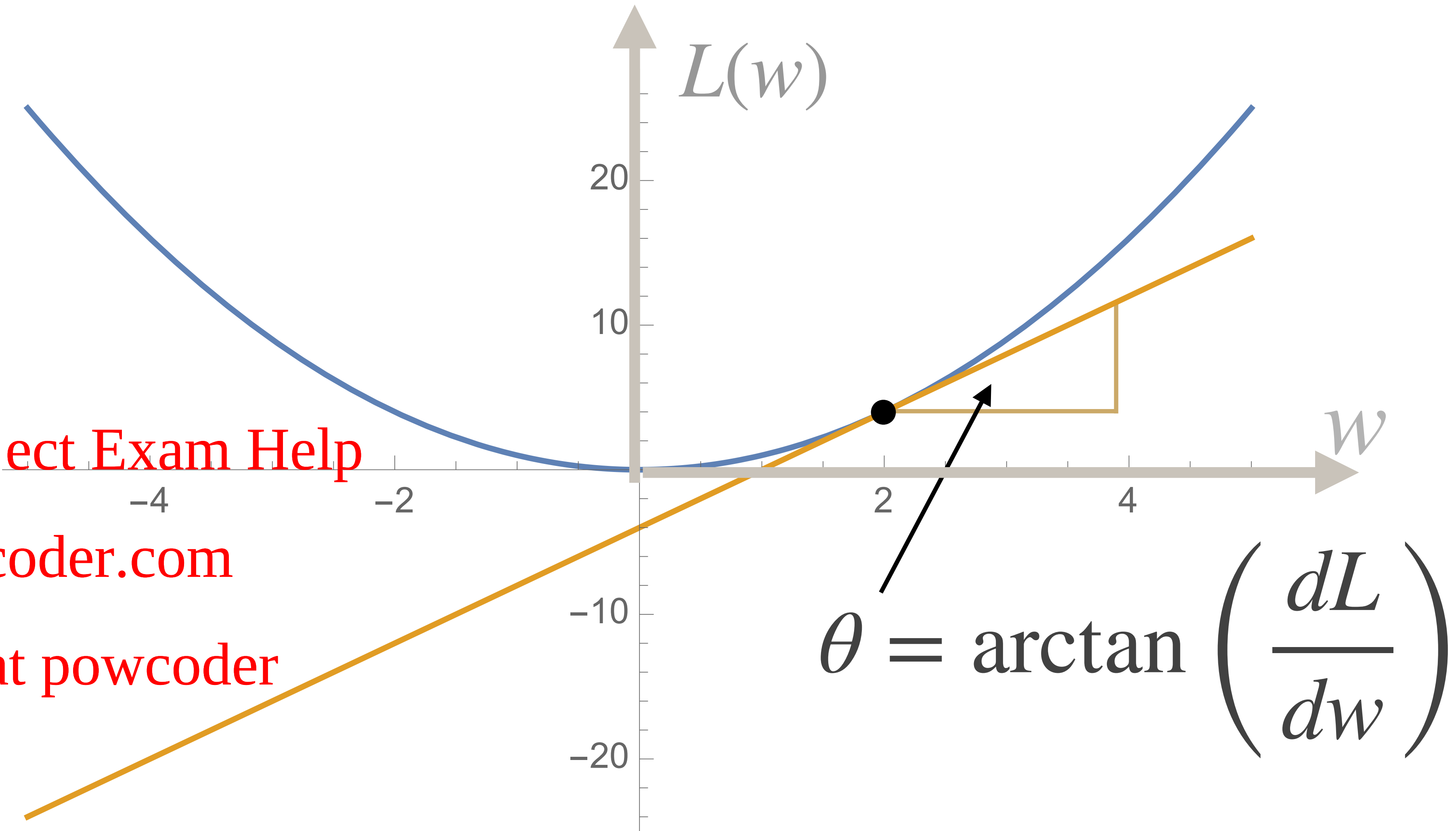
$$\mathbf{r} = \begin{pmatrix} r_1 \\ r_2 \\ \vdots \\ r_N \end{pmatrix}, \quad \|\mathbf{r}\|^2 = \sum_{n=1}^{N} r_n^2$$

Loss = (1/N) $(\text{length})^2$ of N-dimensional residual vector

Update weights to reduce loss: gradient descent

Differentiable loss function: $l_n(w) = r_n^2 = (wx_n - y_n)^2$

- Slope can take any real value $w \in \mathbb{R}$
- Iterate $w^{(t)} \mapsto w^{(t+1)}$, $t = 0, 1, \ldots,$
- Update weights $w^{(t)}$ to $w^{(t+1)}$ so that $L(w^{(t+1)}) < L(w^{(t)})$
- Change weights in the direction **opposite** to the slope of the loss function (we want to **reduce** the loss, hence **descent**)



linear approximation of loss function

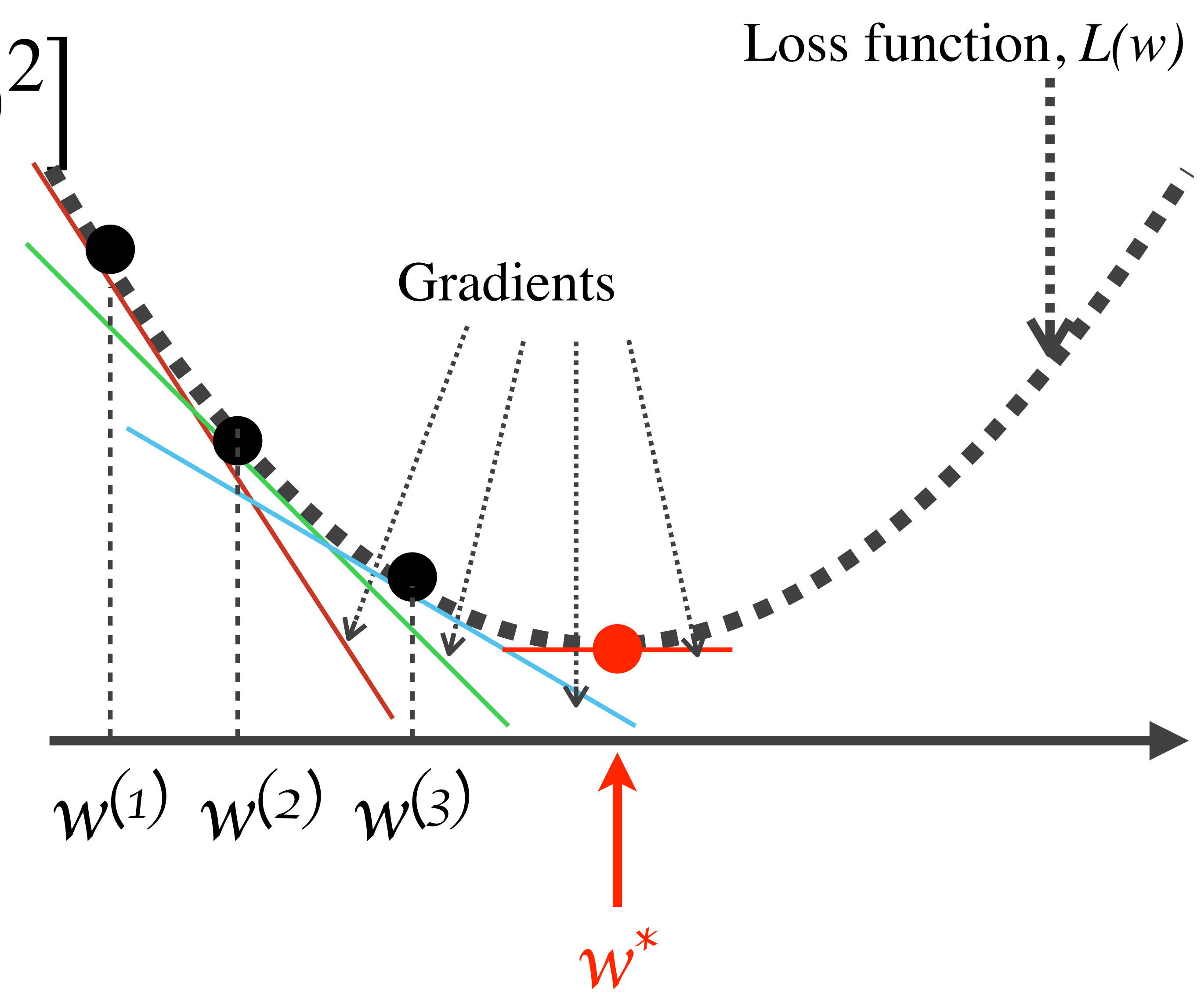
$$w^{(t+1)} = w^{(t)} + \eta\left(\frac{dL(w^{(t)})}{dw}\right), \eta < 0$$

Linear Regression: solving for zero gradient of loss

Closed form solution exists: linear algebra

$(w^2x_1^2 -$ $(w^2x_N^2 - 2wx_Ny_N + y_N^2)$

- $L(w) = (1/N)\left[(wx_1 - y_1)^2 + (wx_2 - y_2)^2 + \cdots + (wx_N - y_N)^2\right]$
- Loss function is quadratic in w
- $L(w) = aw^2 + bw + c$
- Follow gradients until minimum reached
- Solution for weights: set gradient = 0

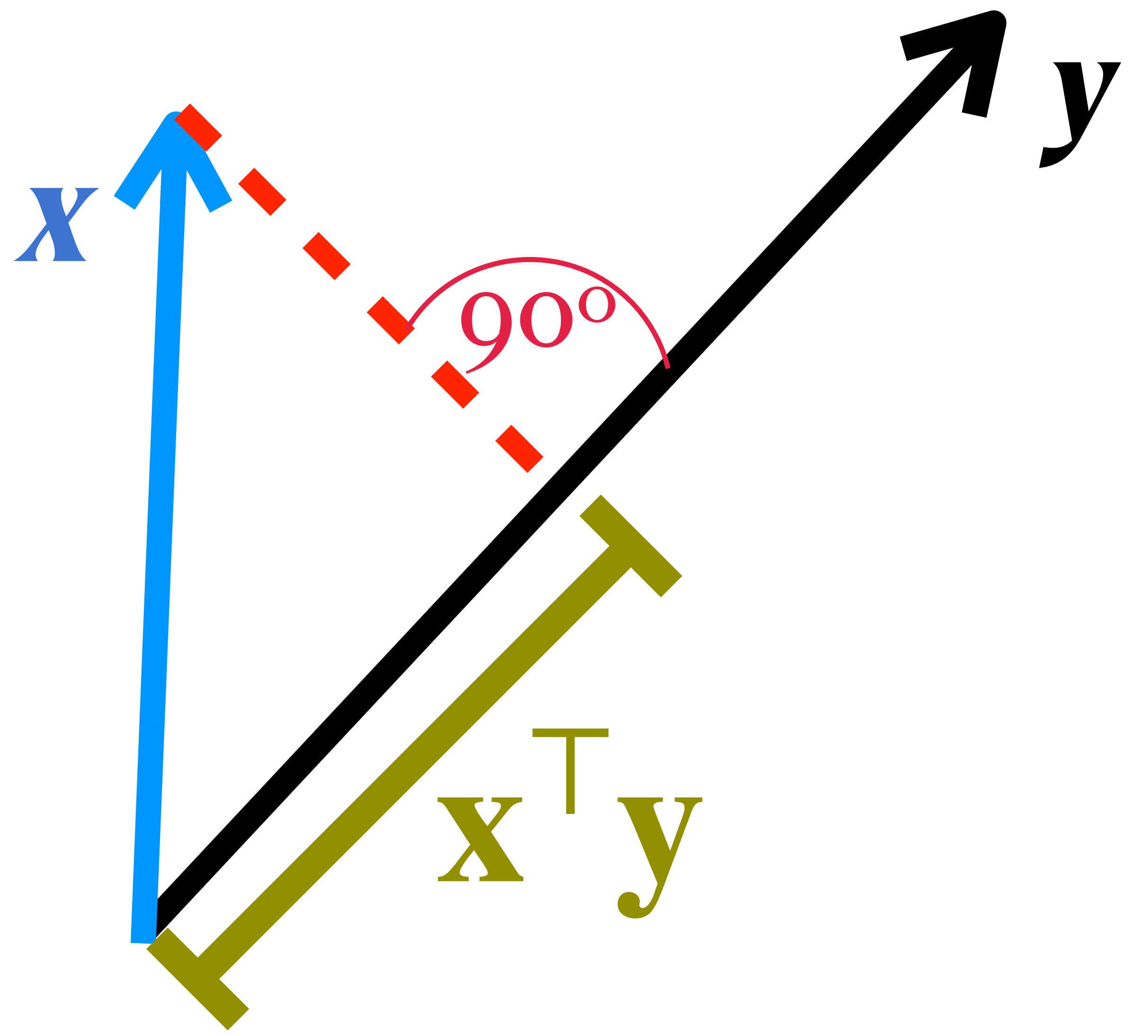


$$0 = \left.\frac{\partial L(w)}{\partial w}\right|_{w=w^*} \implies w^* = -b/(2a)$$

Exercise: differential calculus

Closed form solution to linear regression weights in terms of vector products

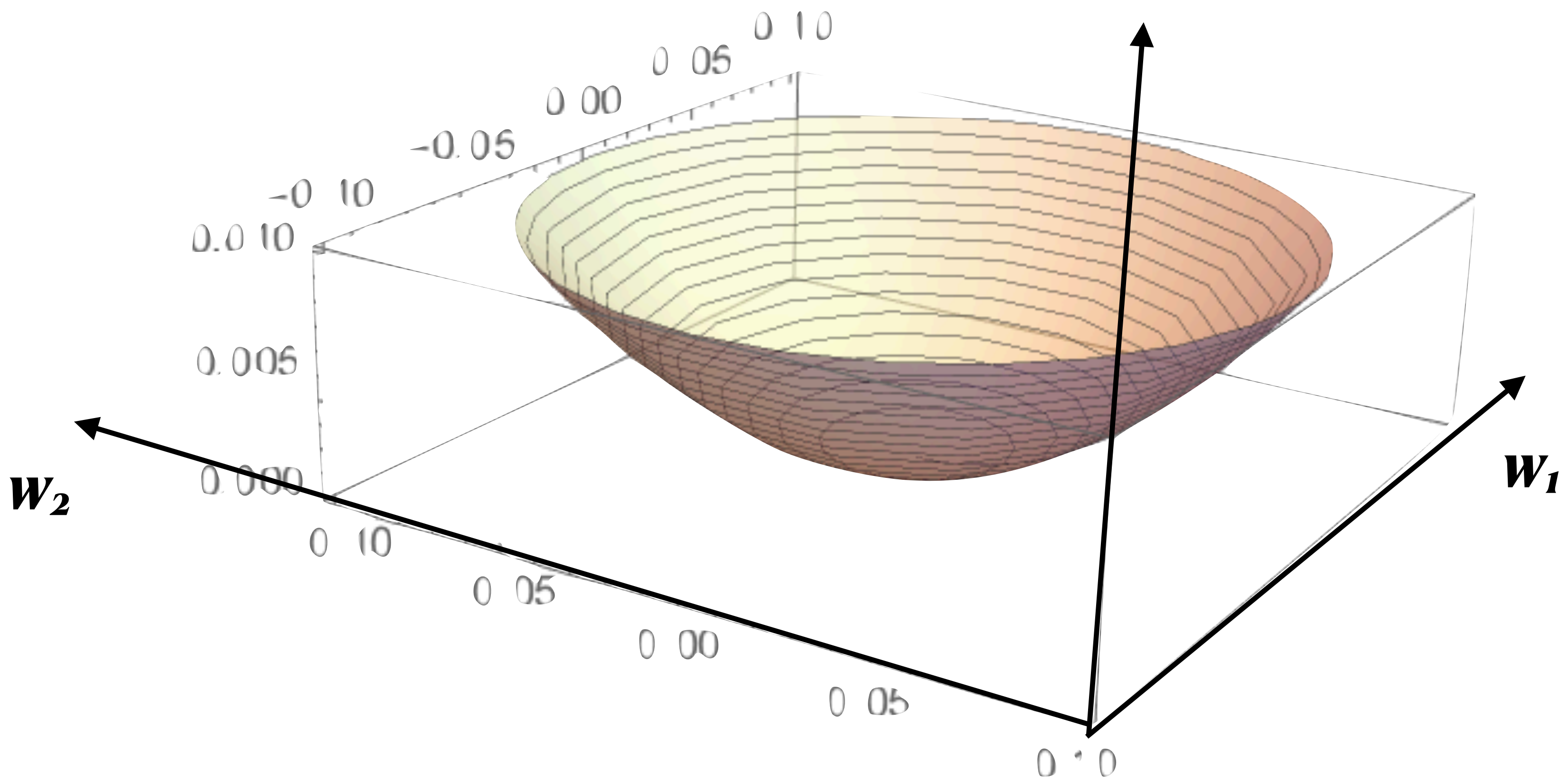
- $L(w) = (1/N)\left[(wx_1 - y_1)^2 + (wx_2 - y_2)^2 + \cdots + (wx_N - y_N)^2\right]$
- **Exercise**: In $L(w) = aw^2 + bw + c$, show
  - $a = (1/N)[x_1^2 + x_2^2 + \cdots + x_N^2]$
  - $b = (-2/N)[x_1y_1 + x_2y_2 + \cdots + x_Ny_N]$
  - $0 = \left.\frac{\partial L(w)}{\partial w}\right|_{w=w*} \implies w* = -b/(2a) = \frac{\mathbf{x}^\top \mathbf{y}}{\mathbf{x}^\top \mathbf{x}}$



Reduce loss by gradient descent: optimisation

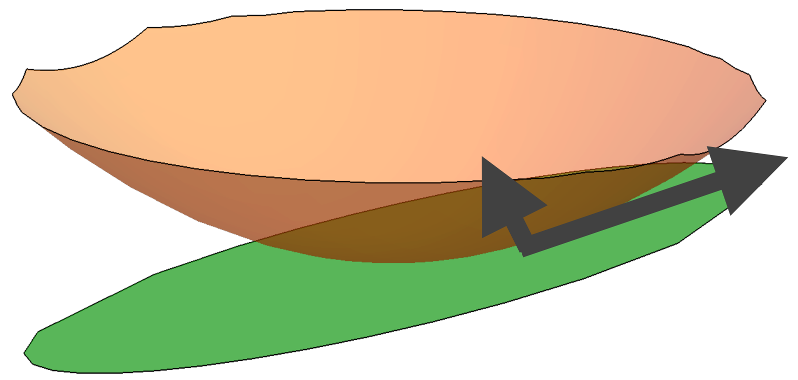
Same idea in higher dimensions (more adjustable weights)

$$\text{loss}_n = (y_n - (w_1 x_{n,1} + w_2 x_{n,2}))^2, \mathbf{x} = (x_1, x_2) \in \mathbb{R}^2$$



Evaluate partial derivatives (gradient) to choose direction of weight updates

$$L(\mathbf{w}) = \mathbb{E}_{(\mathbf{x}_n, y_n) \sim \mathcal{D}} \text{loss}_n(\mathbf{w}), \text{ mean loss}$$



$$(\nabla_{\mathbf{w}} L)_i = \frac{\partial L}{\partial w_i}$$

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \nabla_{\mathbf{w}} L$$

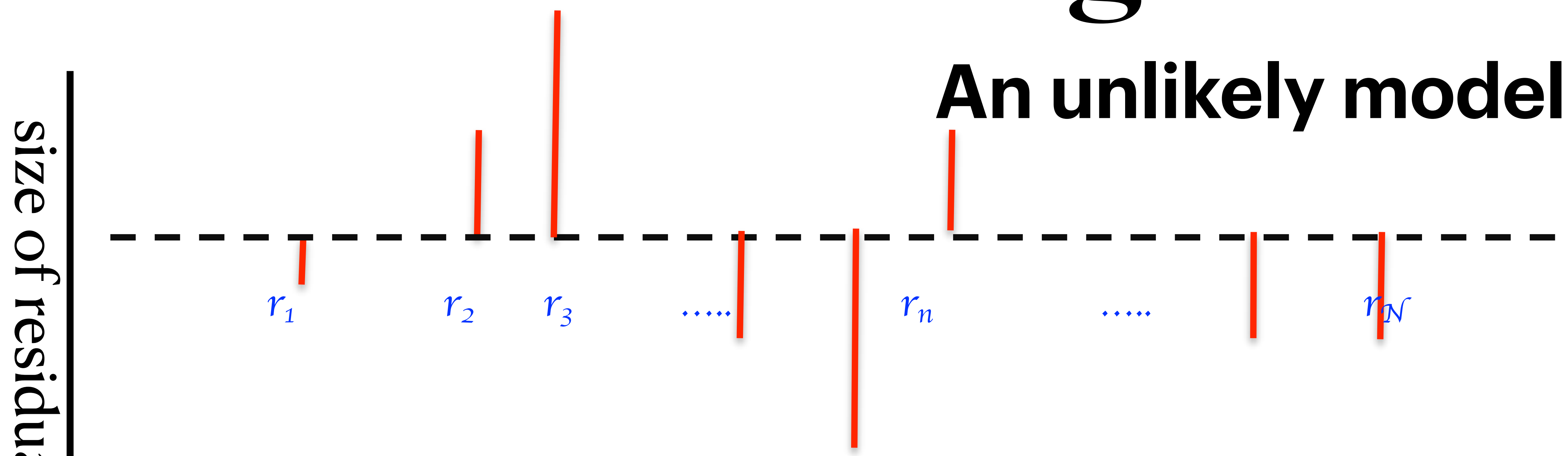
Automatic differentiation

No need to differentiate by hand, except to understand (lab exercises)

- Autograd, JAX — AD libraries in python
- $f : x \mapsto f(x);\ \dfrac{dy}{dx} = \lim_{\delta x \to 0} \dfrac{f(x+\delta x) - f(x)}{\delta x} = \lim_{\delta x \to 0} \dfrac{f(x+\delta x) - f(x-\delta x)}{2 \cdot \delta x}$
- Product: $y(x) = f(x) \cdot g(x); y'(x) = f'(x) \cdot g(x) + f(x) \cdot g'(x)$
- Quotient: $y(x) = f(x)/g(x); y'(x) = [f'(x) \cdot g(x) - f(x) \cdot g'(x)]/g(x)^2$
- Composition: $y(x) = f(g(x)), \dfrac{dy}{dx} = \dfrac{df}{dg}\dfrac{dg}{dx}$
- Program $f : x \mapsto f(x)$ forward mode AD $(f, Df) : x \mapsto (f(x), f'(x))$ e.g., $\text{sq} : x \mapsto (x^2, 2x)$
- Taylor series: $f(x_0 + \delta x) = f(x_0) + \delta x \left.\dfrac{df}{dx}\right|_{x_0} + \dfrac{1}{2}(\delta x)^2 \left.\dfrac{d^2 f}{dx^2}\right|_{x_0} + \cdots$, note $(\delta x)^2 \ll \delta x$

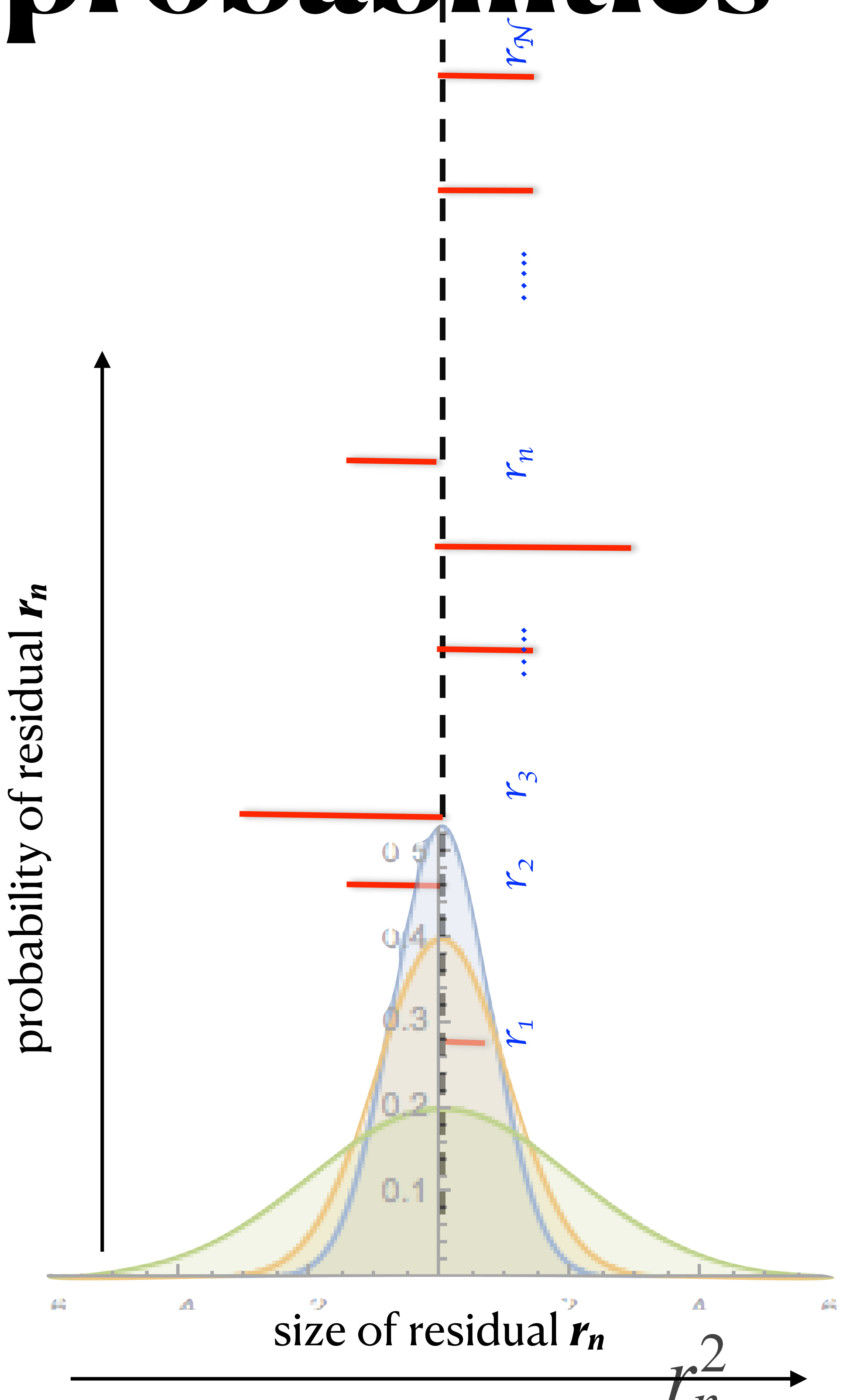
Later: view large distances as small probabilities

An unlikely model has large residuals



- Different interpretation of the learning task
  A. If $|r_n|$ is large $p(r_n)$ is small (low probability)
  B. Reducing $|r_n|$ makes $p(r_n)$ large (high probability)

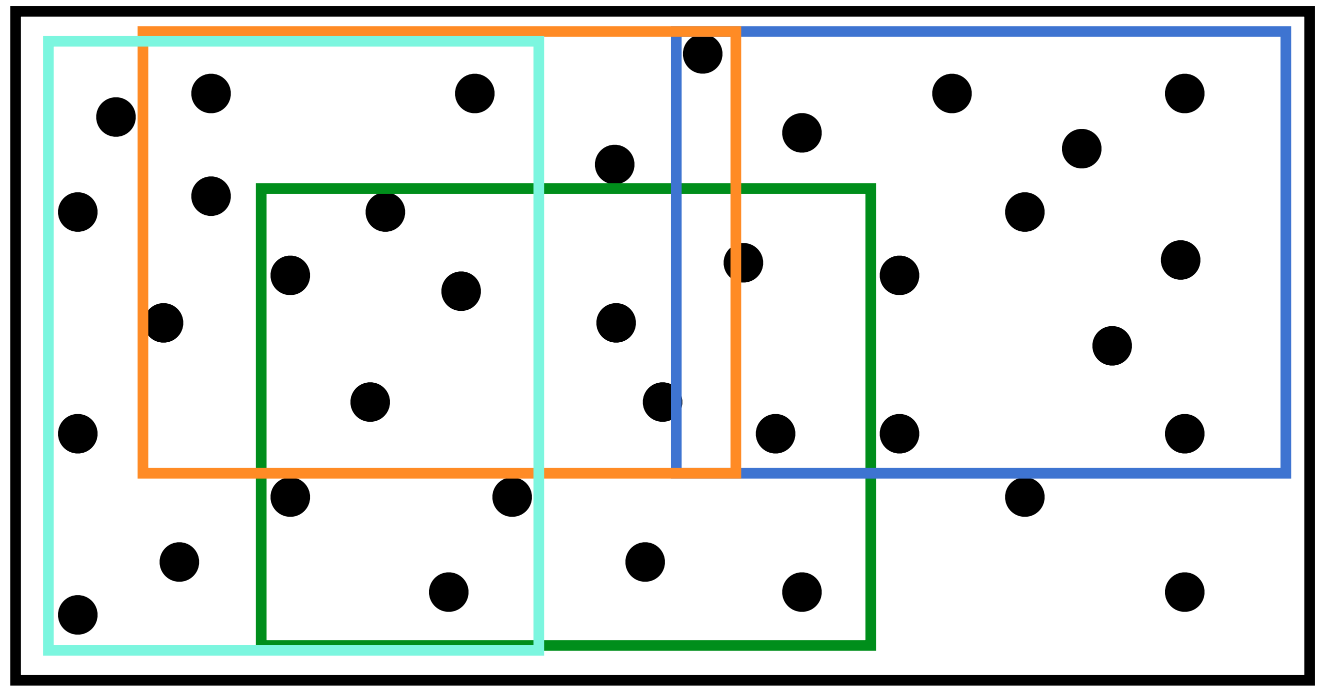
- Probability of what? $f(\,\cdot\,; \mathbf{w})$ evaluated on data $\mathcal{D} := \{(x_n, y_n)\}_{n=1,\ldots,N}$, called model **likelihood**
- Maximum likelihood estimation: estimation of weights to achieve objective of large likelihood



$$p(r_n) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{r_n^2}{2\sigma^2}}$$

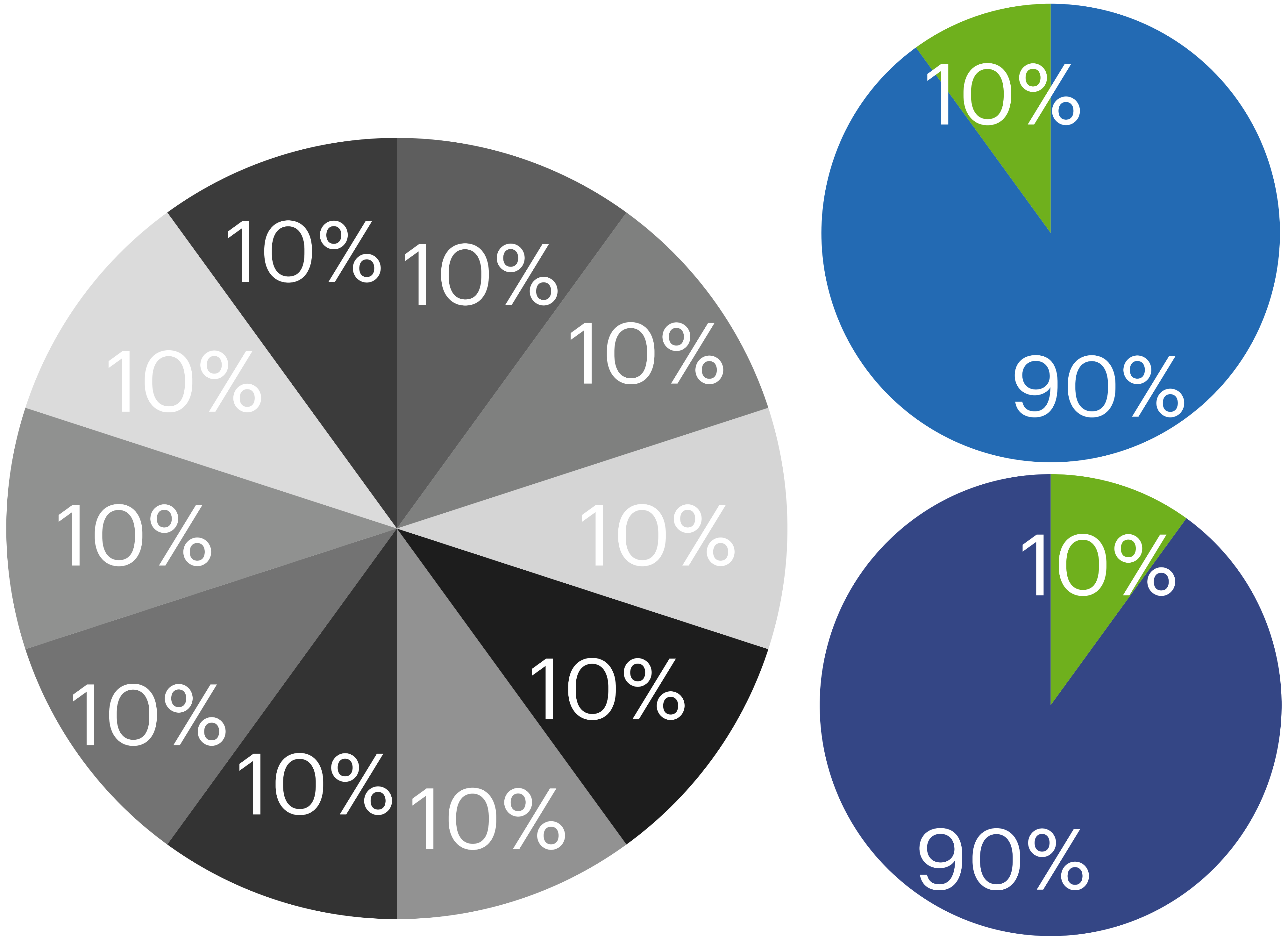
Learning or memorising?

Evaluating ML models: Cross-validation



Coloured boxes: possible training sets

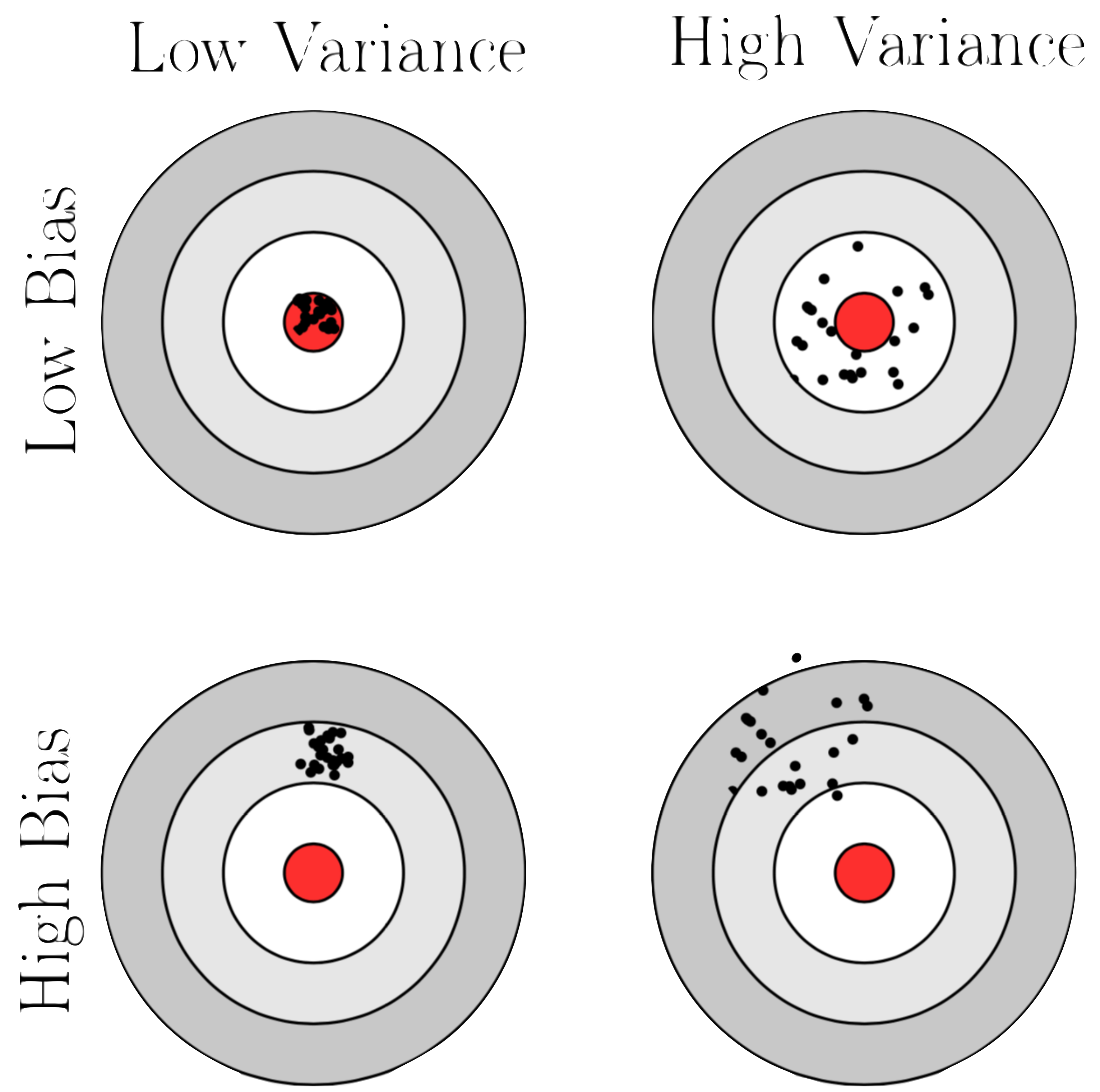
- Evaluate model performance on test data (**generalisation**)
- Different training sets can lead to different model parameters with different predictions, hence different residuals
- Characterise model on **distribution** of residuals trained on different subsets of available training data
- **Cross validation**

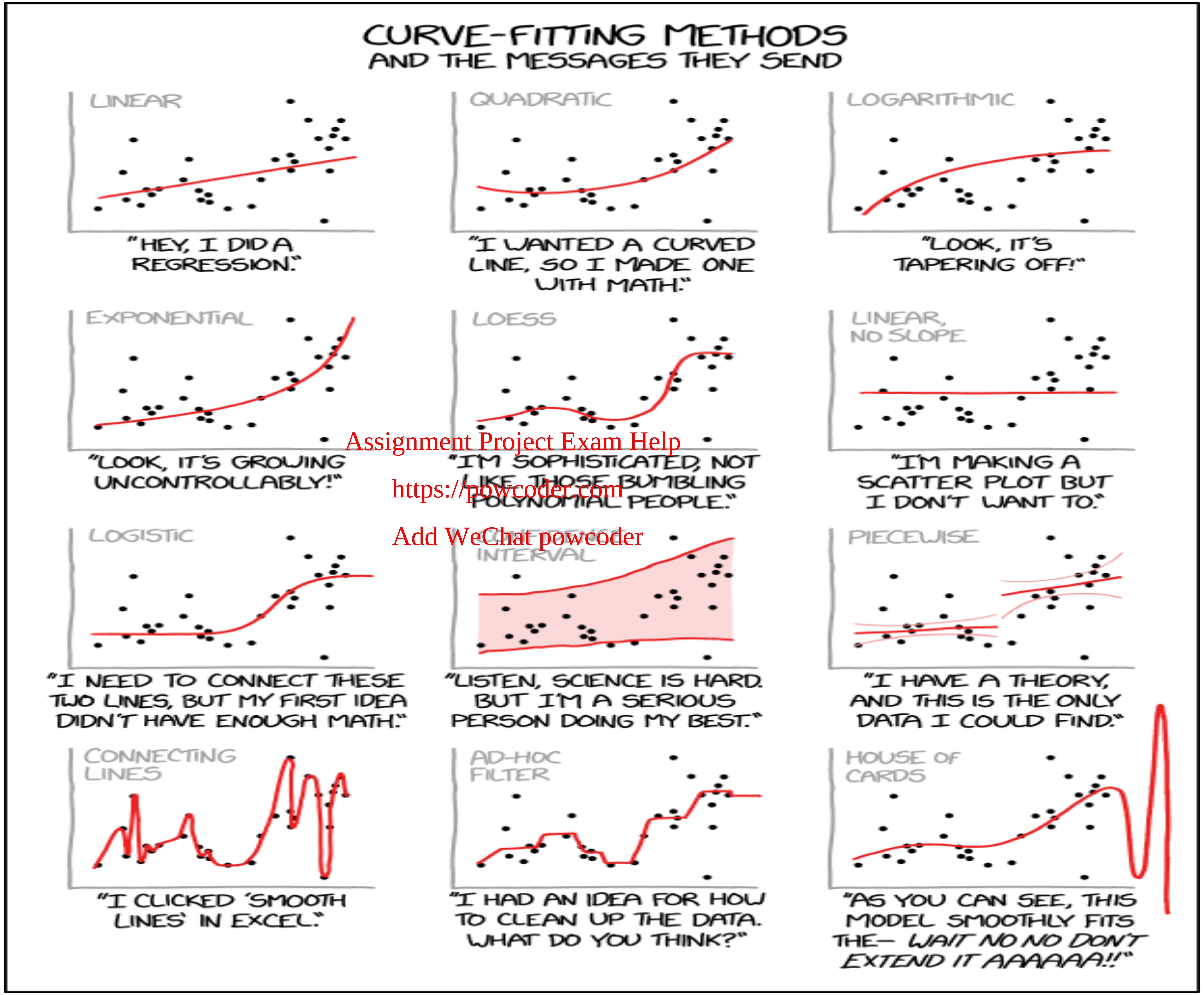


Learning or memorising? Bias-variance tradeoff

Criteria for evaluation of ML models

- Different training sets can lead to different models with different model predictions and different residuals
- Bias: Deviation of average of predictions from truth
- Variance: variability of model predictions on different parts of test set
- Trade-off

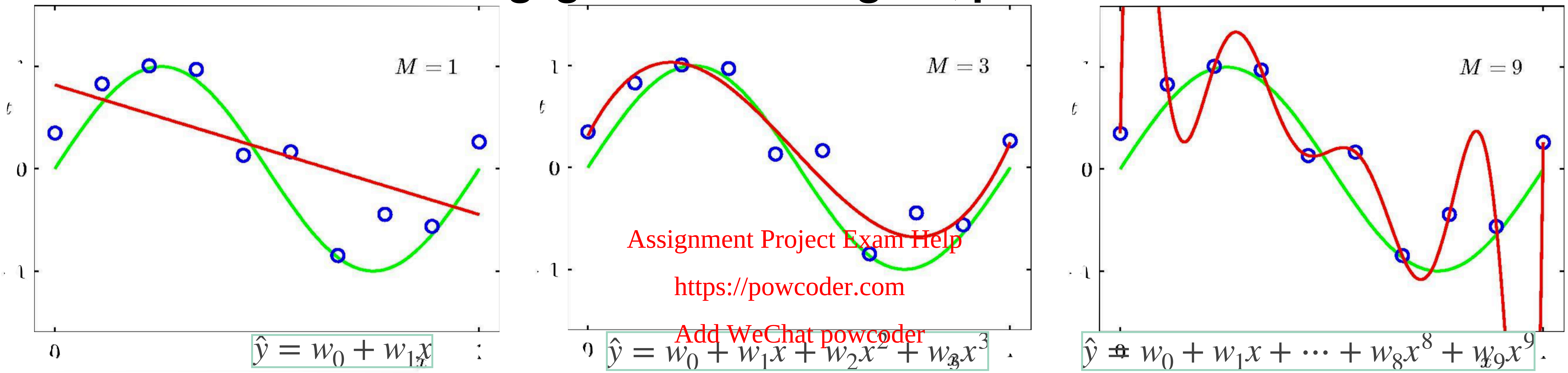




Curve-fitting,
no interpretation

Polynomial fits with high degrees tend to overfit

Overfitting: good on training set, poor on test set

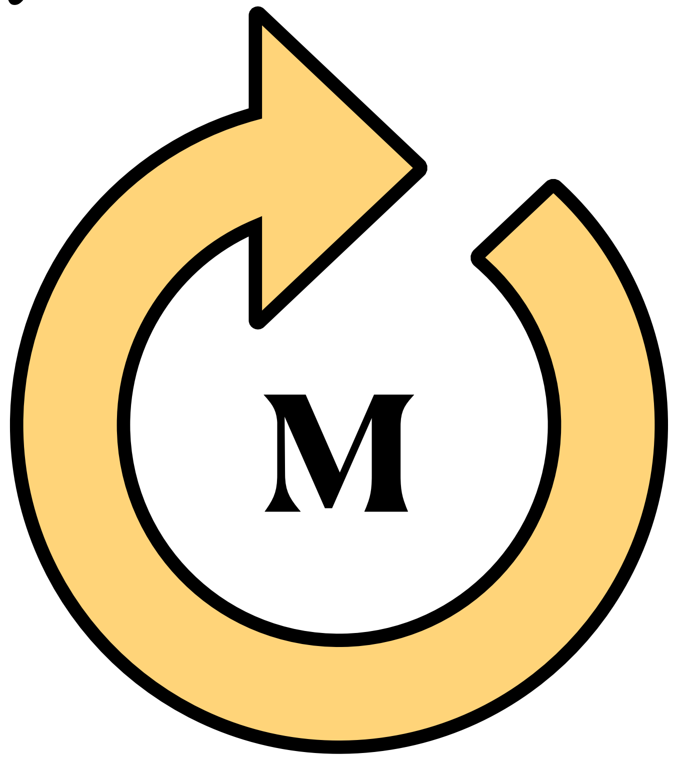


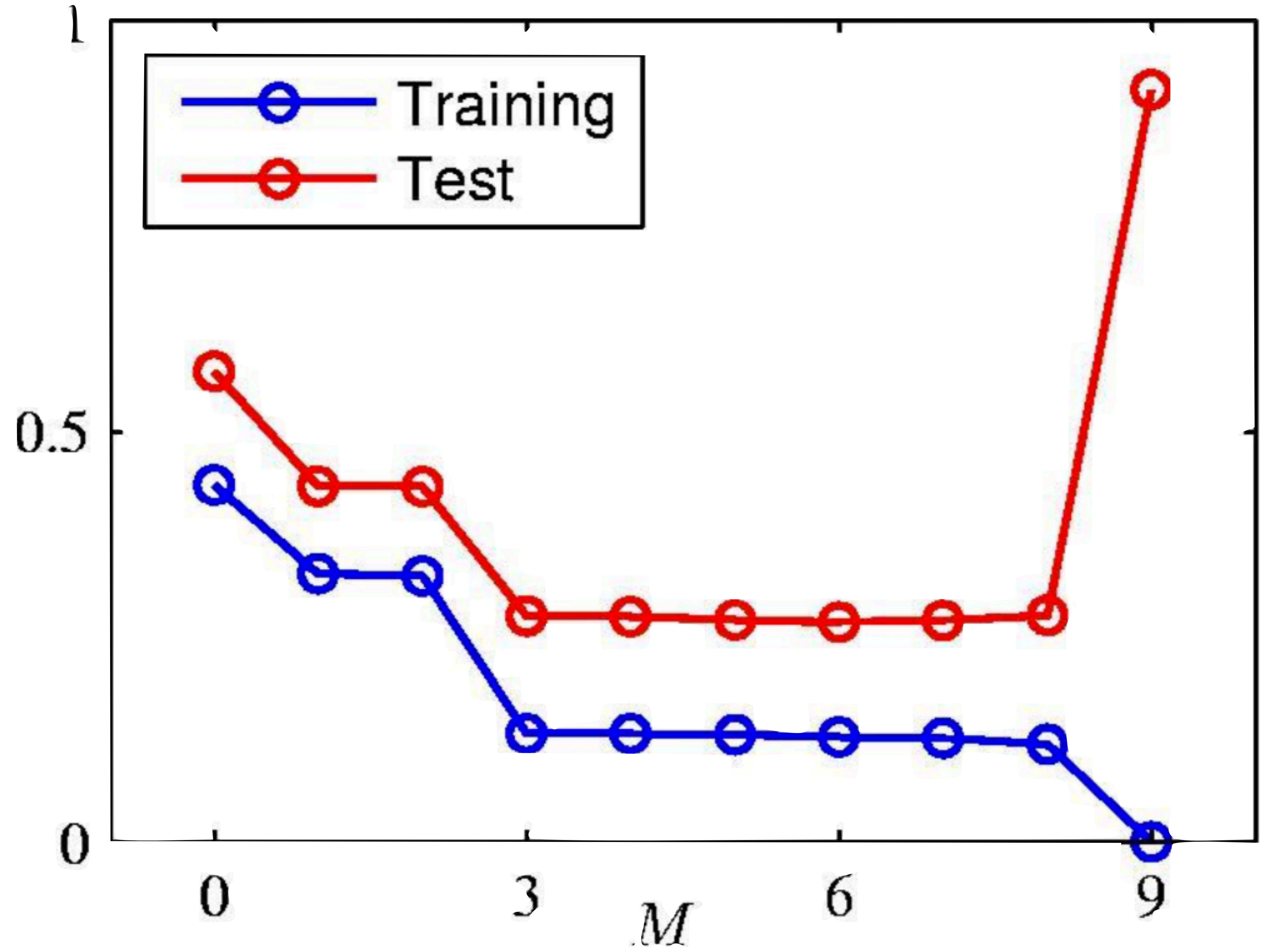
$\hat{y} = w_0 + w_1 x$

$\hat{y} = w_0 + w_1 x + w_2 x^2 + w_3 x^3$

$\hat{y} = w_0 + w_1 x + \cdots + w_8 x^8 + w_9 x^9$

$$\hat{y}(x; \mathbf{w}) = w_0 + w_1 x + w_2 x^2 + \cdots + w_M x^M$$





From C Bishop, PRML

Weights learned by minimising loss

Polynomial models of high degree have large weights

	$M = 0$	$M = 1$	$M = 3$	$M = 9$
$w_0^\star$	0.19	0.82	0.31	0.35
$w_1^\star$		-1.27	7.99	232.37
$w_2^\star$			-25.43	-5321.83
$w_3^\star$			17.37	48568.31
$w_4^\star$				-231639.30
$w_5^\star$				640042.26
$w_6^\star$				-1061800.52
$w_7^\star$				1042400.18
$w_8^\star$				-557682.99
$w_9^\star$				125201.43

From C Bishop, PRML

Regularisation: Penalty for model complexity

Complex models are believed to overfit

$$\text{Loss}_{\mathcal{D}}(\mathbf{w}) = \mathbb{E}_{(\mathbf{x}_n, y_n)\sim\mathcal{D}}(y_n - \hat{y}(\mathbf{x}_n; \mathbf{w}))^2$$

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \text{Loss}_{\mathcal{D}}(\mathbf{w}) + \lambda||\mathbf{w}||^2$$

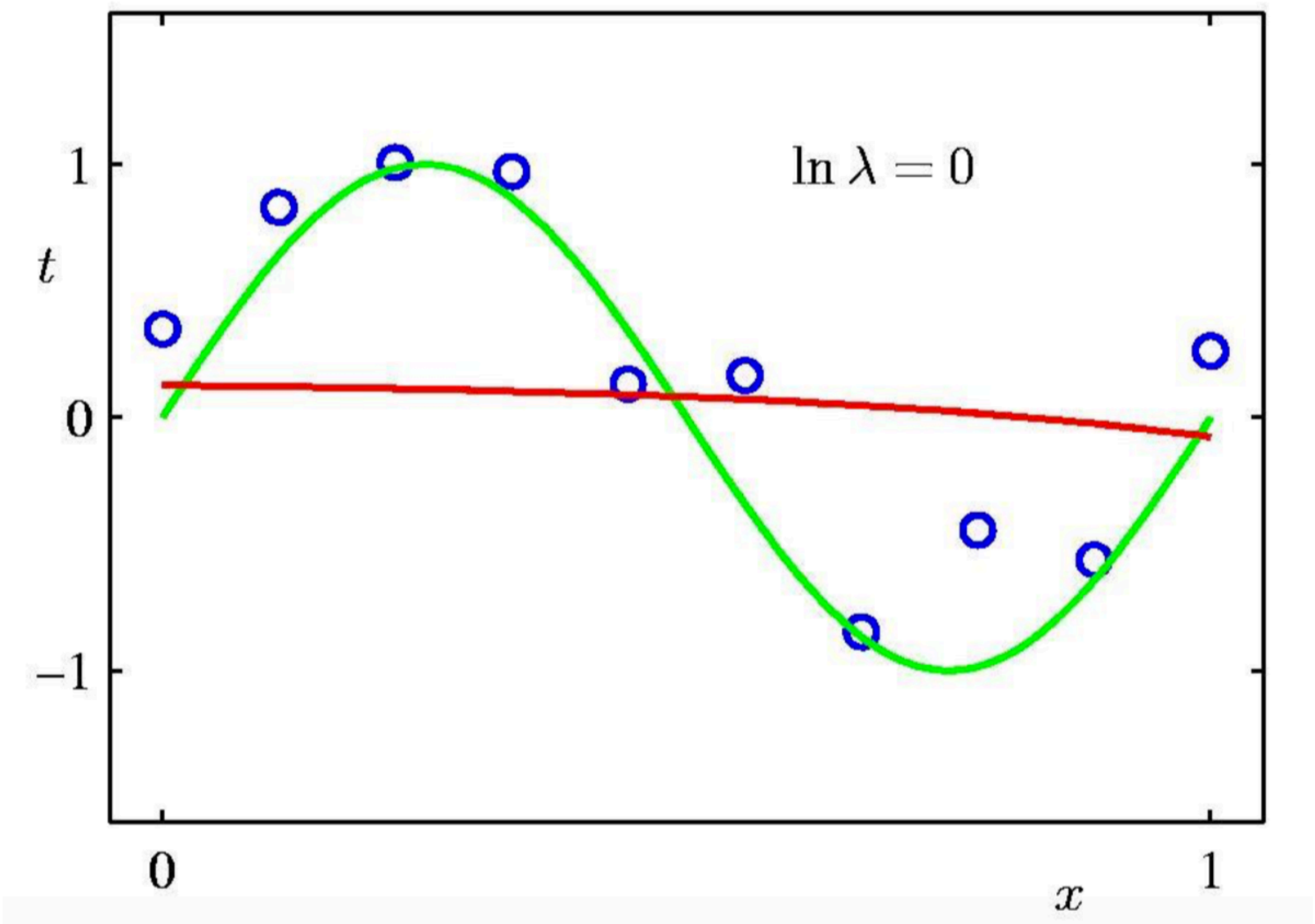
Minimise a combination of two factors

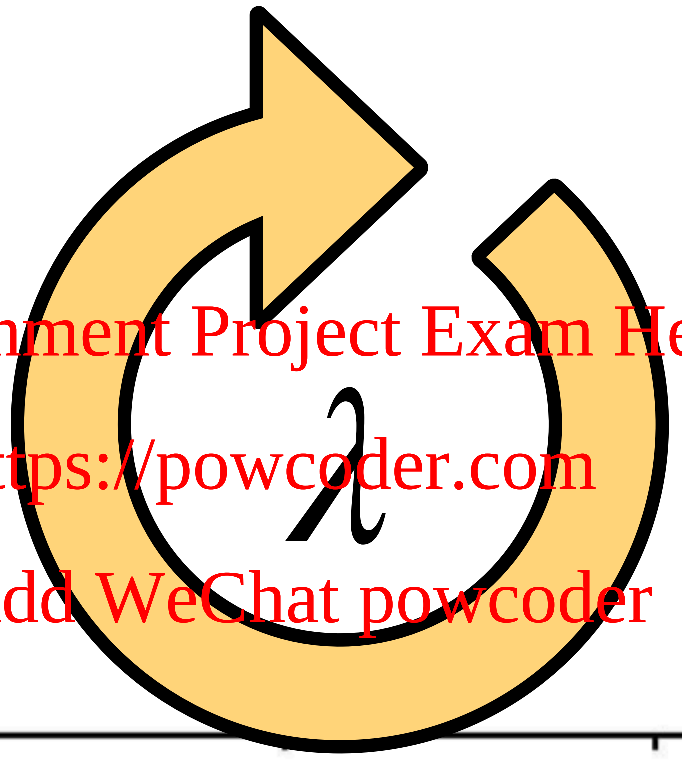
1. — mismatch of model prediction to labelled data (Loss)
2. — data-independent term that depends on model alone (**regularisation**)

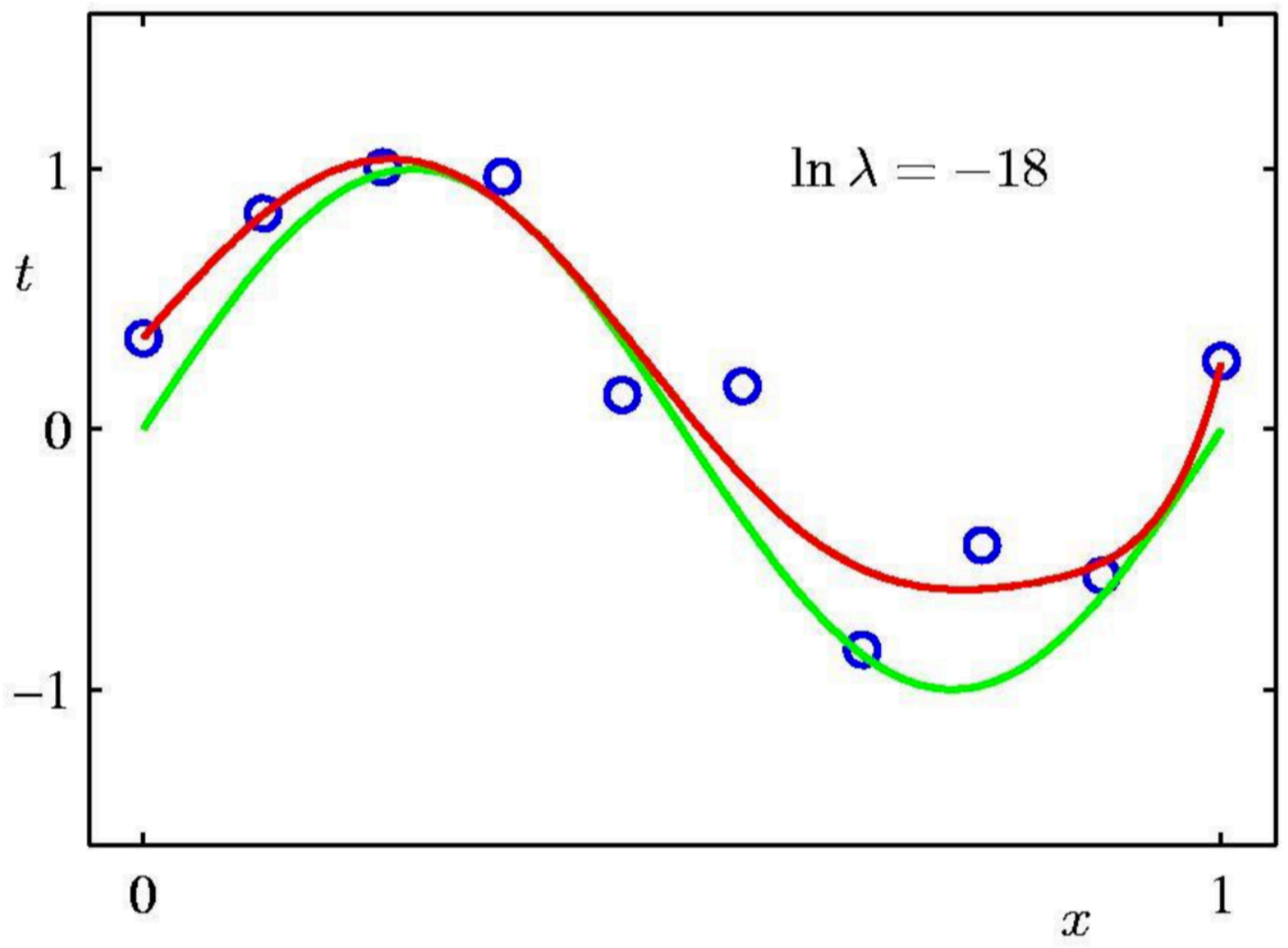
Relative penalties for loss and weight length

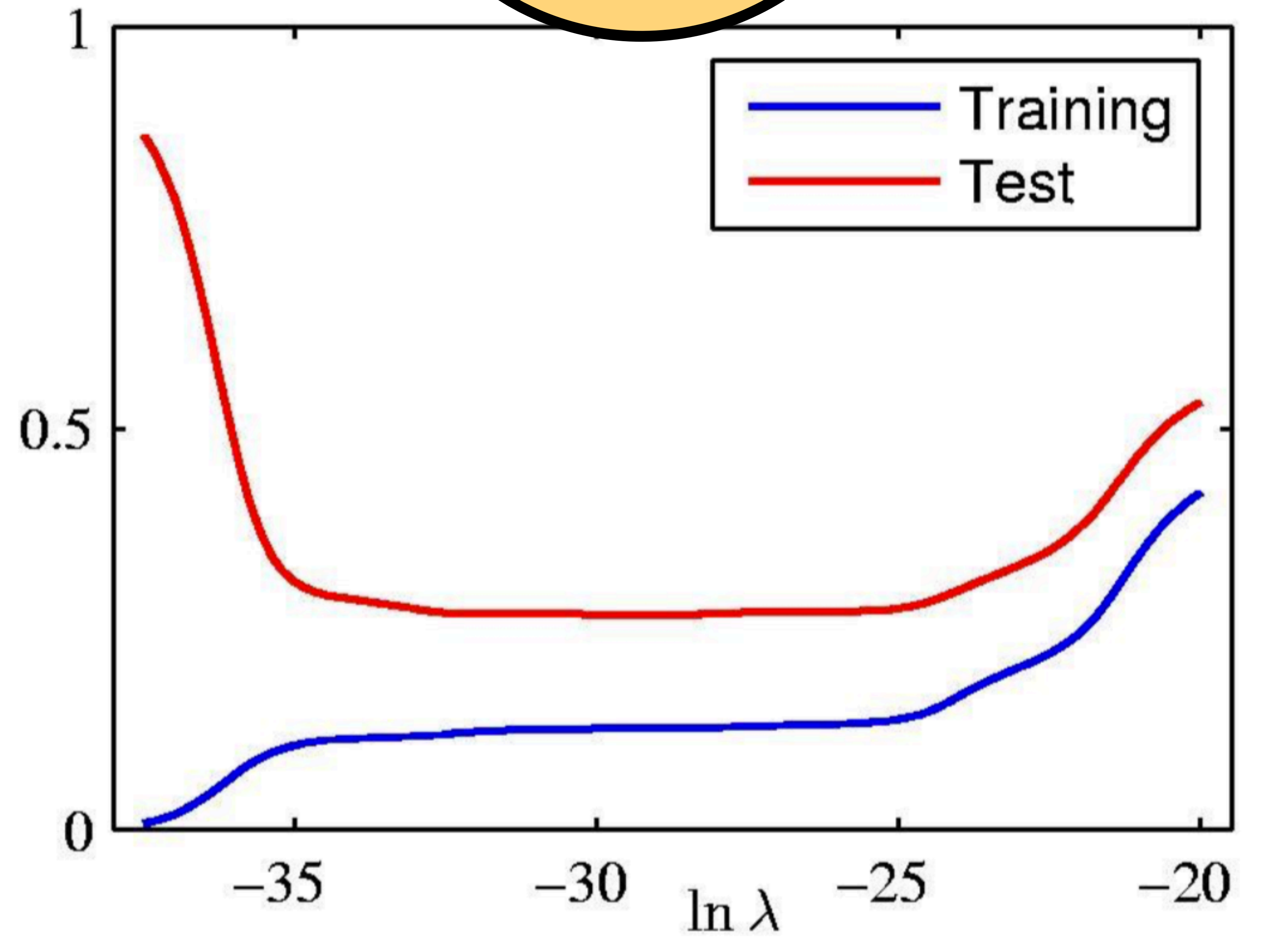
How big should model penalty λ be?

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \mathrm{Loss}_{\mathscr{D}}(\mathbf{w}) + \lambda\|\mathbf{w}\|^2$$









Read Section 1.1, p.4
C Bishop, PRML

Supervised Learning: reduce loss

Summary

- Predict and correct using weight-adjustable functions
- Optimise weights using loss function — learning as optimisation
- Interpret weight spaces in terms of mathematical framework of linear algebra
- Interpret distances and loss metrics in terms of probability theory
- Evaluate performance in terms of generalisation (bias/variance)
- Introduce regularisation for generalisation