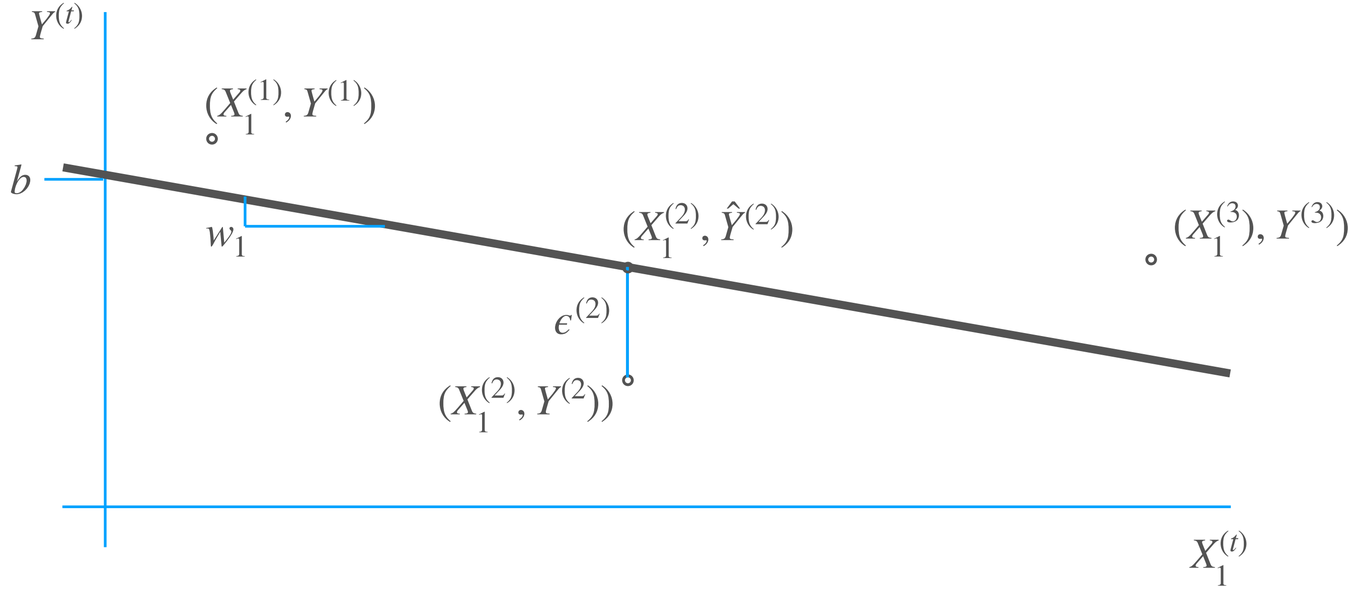
# *Reminder*

- HW1 due tomorrow

# *Linear regression*




- Regression = supervised prediction of *real* labels
  - data: features $X^{(t)} \in \mathbb{R}^d$, labels $Y^{(t)} \in \mathbb{R}$, $t = 1 : T$
- *Linear* regression = predict a linear function of features
  - model: predict $\hat{Y}^{(t)} = w \cdot X^{(t)} = \sum_i w_i X_i^{(t)}$ or $w \cdot X^{(t)} + b$
  - can omit $b$ if we have a constant feature

# *Linear regression*

- Objective:
  - minimize sum of squared errors:

$$\frac{1}{2}\sum_{t=1}^{T}(\underbrace{Y^{(t)}-\hat{Y}^{(t)}}_{\epsilon^{(t)}})^2$$

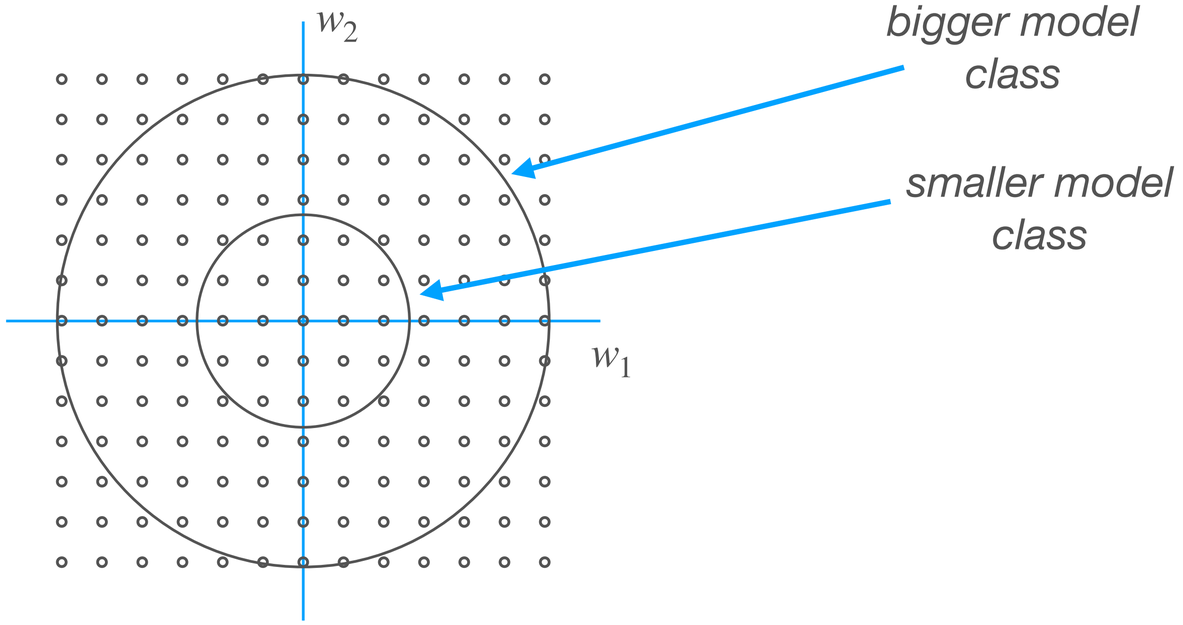
  - plus optional regularizer (see next slides)
- E.g., minimize

$$L(w)=\frac{1}{2}\sum_{t=1}^{T}(Y^{(t)}-\hat{Y}^{(t)})^2+\frac{\lambda}{2}\|w\|_2^2$$

# *Model complexity*

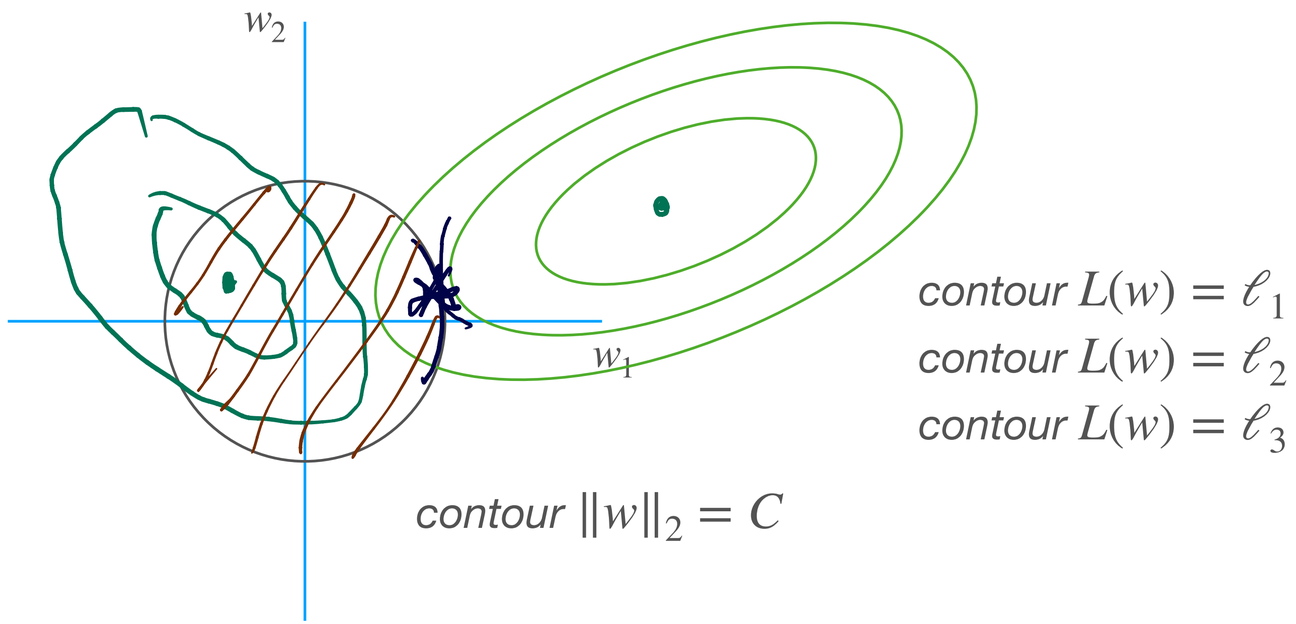
*Terminology: **model** can mean either the structure or the structure plus fitted parameters*
*To distinguish, **model class** vs. **trained/fitted model***

*Trained model = hypothesis*
*One or more model classes make up hypothesis space*




- Model complexity (for model selection):
  - how many features/weights
  - how many *nonzero* weights
  - *size* of weights (there are more effectively-distinct weight vectors in $\{w \mid \|w\| \leq 100\}$ than in $\{w \mid \|w\| \leq 1\}$)

# Model selection

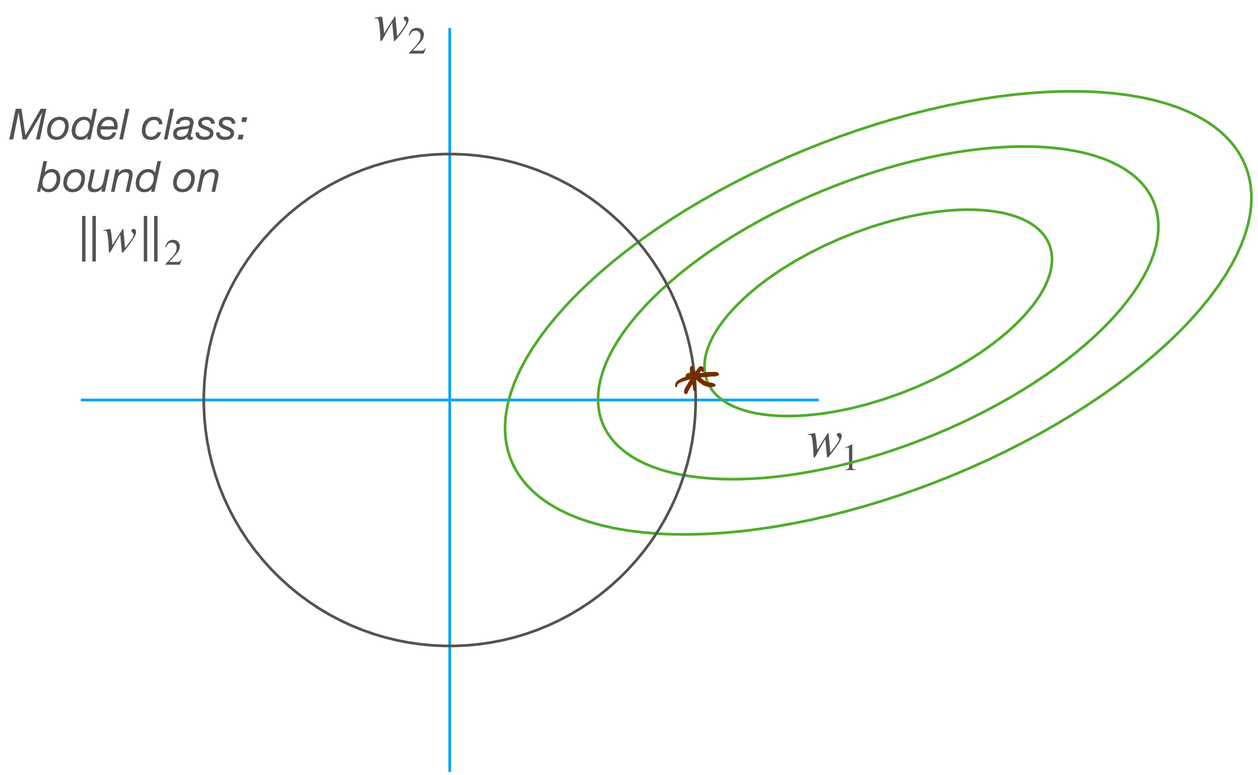



- To limit complexity, can limit (e.g.) norm of weight vector
- Corresponding model fitting problem:

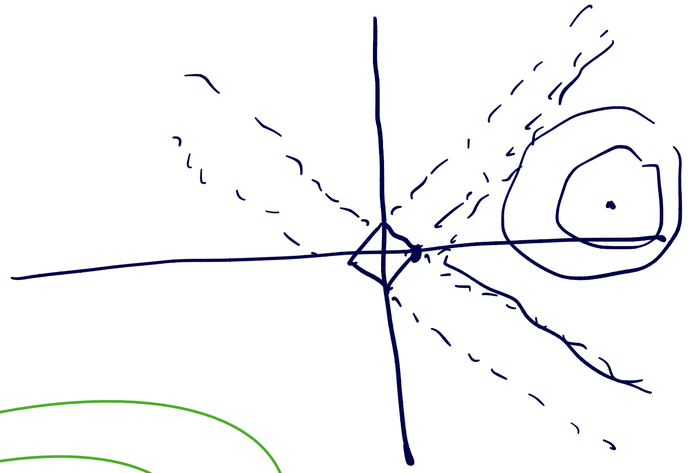
$$\min_w L(w) = \frac{1}{2}\sum_{t=1}^{T}(Y^{(t)} - X^{(t)} \cdot w)^2 \quad \text{s.t.} \quad \|w\|_2 \le C$$

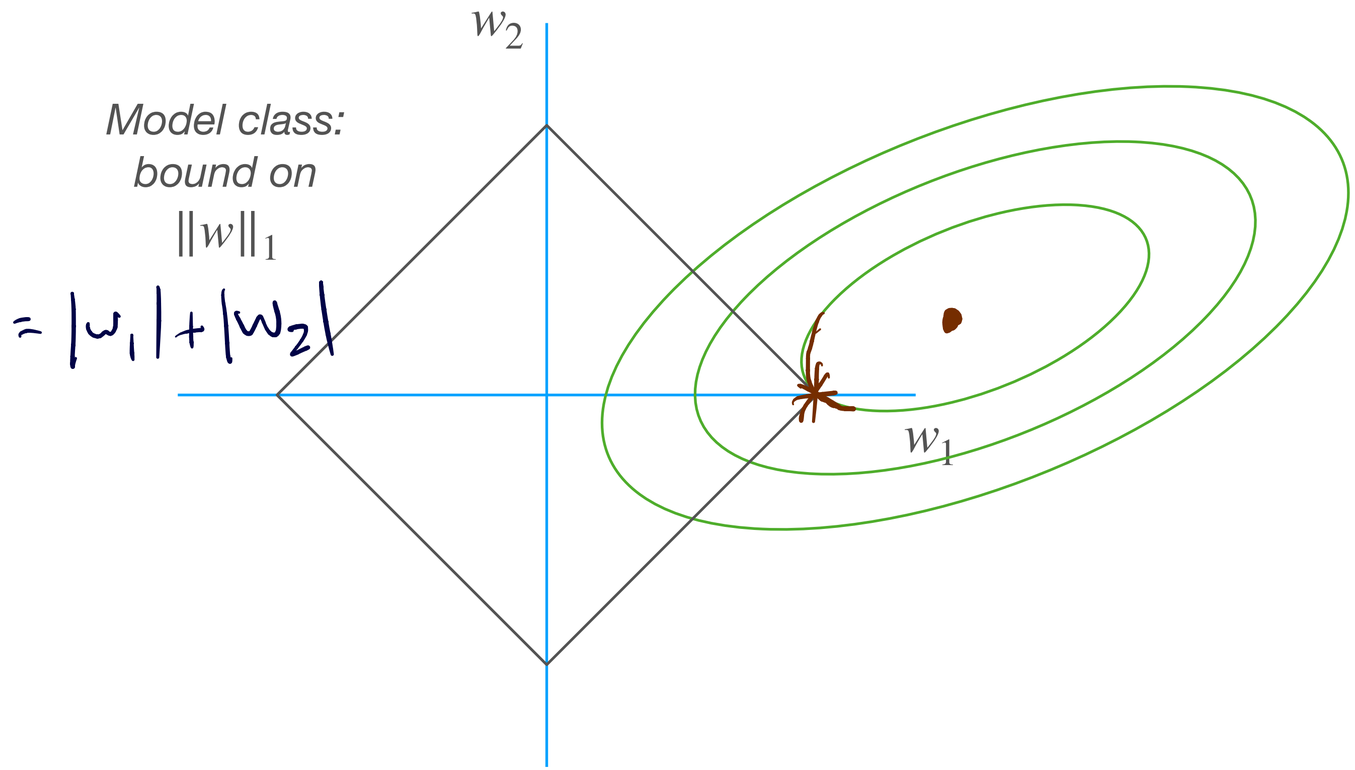
- Model selection = picking $C$

# *Size (norm) and count of nonzeros are related*

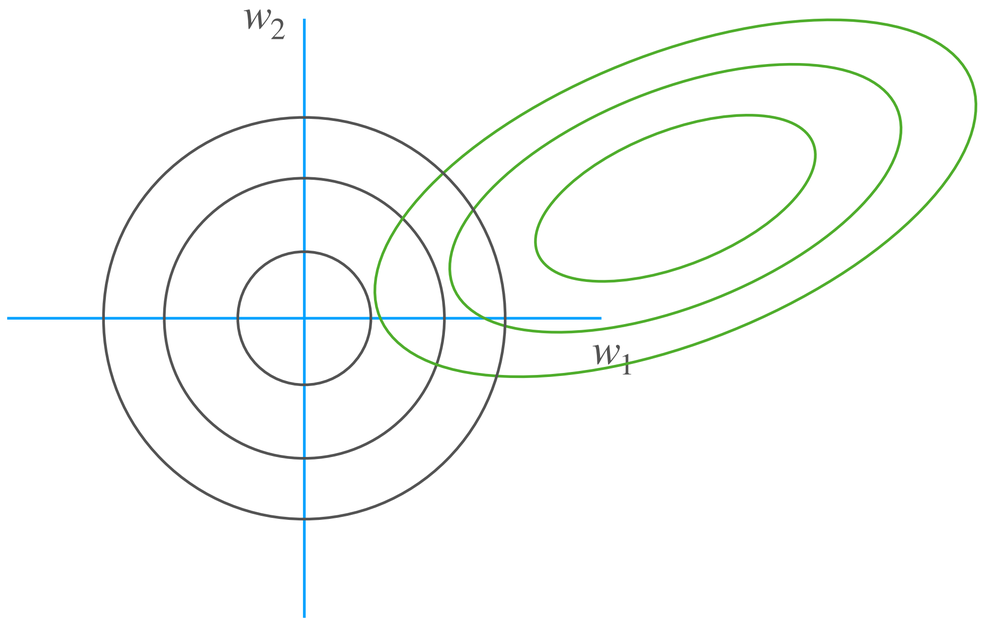


# *Size (norm) and count of nonzeros are related*





# *Regularization by penalty vs. constraint*




- Instead of constrained minimization

$$\min_w L(w) = \frac{1}{2}\sum_{t=1}^{T}(Y^{(t)} - X^{(t)} \cdot w)^2 \quad \text{s.t.} \quad \|w\|_2 \leq C$$

- we can try to directly trade off $L(w)$ and $\|w\|_2$ in optimizer (avoid outer model selection loop):

$$\min_w L(w) = \frac{1}{2}\sum_{t=1}^{T}(Y^{(t)} - X^{(t)} \cdot w)^2 + \frac{\lambda}{2}\|w\|_2^2$$

# *Bias-variance tradeoff*

- Regularization parameters $\lambda$, $C$ trade off underfitting vs. overfitting
  - underfitting is often called ***bias:*** the correct answer is not an option, so we pick something that might be far from it
  - overfitting shows up as ***variance*** of parameters $w$: more choices → more variability in which one we select

regression: SSE = $\text{bias}^2$ + variance

# *Overfitting, bias, variance example*

```
n = 10
k = 7
X = np.random.normal(size=(n, k))
Y = np.random.normal(size=(n, 1))
w, _, _, _ = np.linalg.lstsq(X, Y, rcond=None)
(np.linalg.norm(w), np.mean((Y - X@w)**2))
```

- What is the smallest MSE this code should return if there were no selection bias? → 1
- What is the actual best $w$? → $w = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$

# *Overfitting, bias, variance example*

```
n = 10
k = 7
X = np.random.normal(size=(n, k))
Y = np.random.normal(size=(n, 1))
w, _, _, _ = np.linalg.lstsq(X, Y, rcond=None)
(np.linalg.norm(w), np.mean((Y - X@w)**2))
```

- What is the smallest MSE this code should return if there were no selection bias?
- What is the actual best $w$?

```
# three runs (n=10, k=7):

(1.729258520013365, 0.0940395032823642 1)
(1.9706932994777069, 0.07379394277037196)
(3.591128626783351, 0.47238275606461944)
```

# *Overfitting, bias, variance example*

```
n = 10
k = 7
X = np.random.normal(size=(n, k))
Y = np.random.normal(size=(n, 1))
w, _, _, _ = np.linalg.lstsq(X, Y, rcond=None)
(np.linalg.norm(w), np.mean((Y - X@w)**2))
```

- What is the smallest MSE this code should return if there were no selection bias?
- What is the actual best $w$?

```
# three runs (n=10, k=7):

(1.729258520013365, 0.0940395032823642 1)
(1.9706932994777069, 0.07379394277037196)
(3.591128626783351, 0.47238275606461944)

# three runs (n=20, k=2):

(0.20006773436081007, 0.8932440732695813)
(0.3765035073938464, 0.9200354120567253)
(0.576420124635776 1, 0.7614382010070438)
```

# *Linear regression: summary of objective*

- Objective:
  - minimize sum of squared errors: $\frac{1}{2}\sum_{t=1}^{T}(Y^{(t)} - \hat{Y}^{(t)})^2$
  - plus optional regularizer: e.g., $L_2$, $L_1$, both
  - $\|w\|_2^2$ called *ridge regression*
  - $\|w\|_1$ called *LASSO*
  - if we use both, *elastic net*
- E.g., minimize

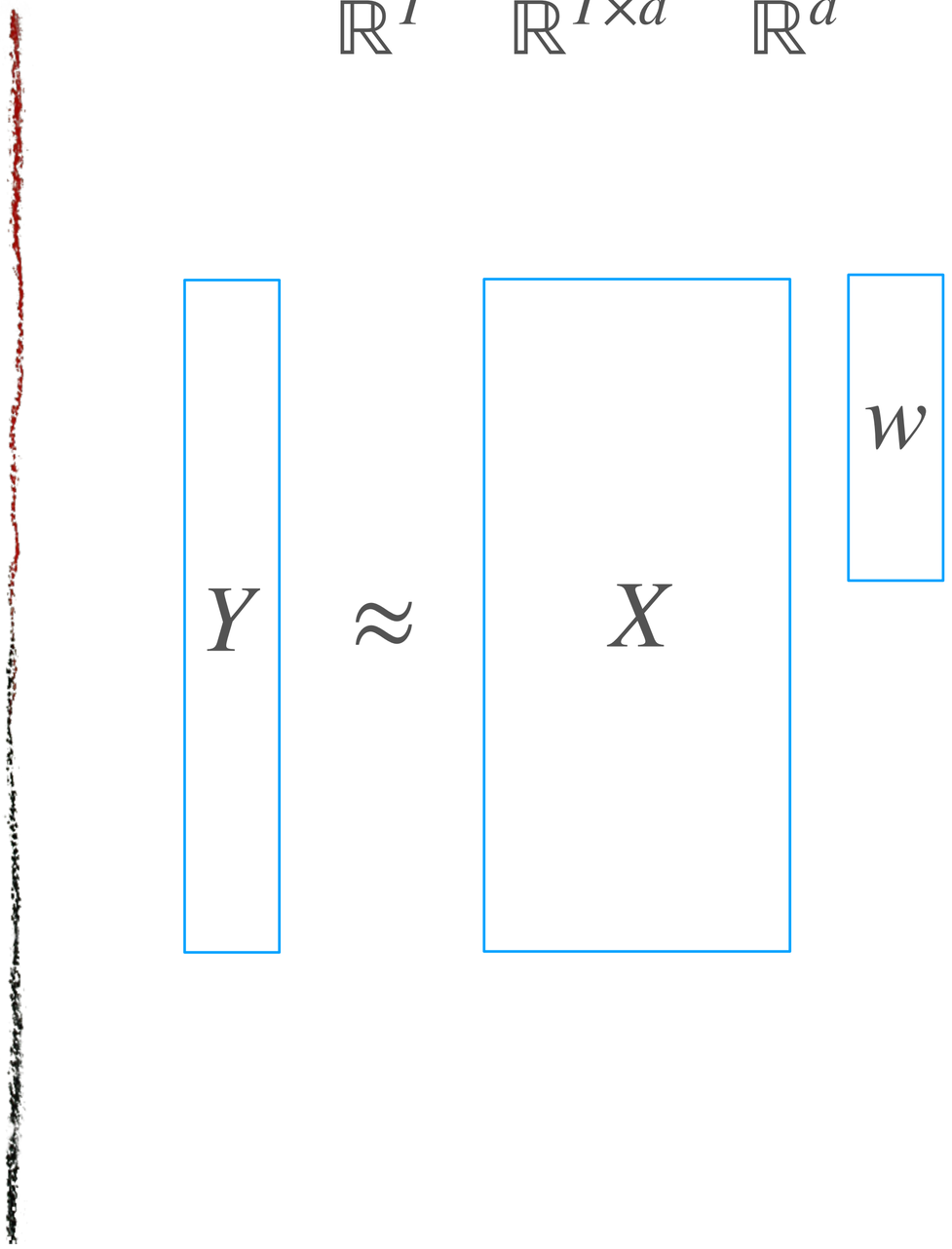
$$L(w) = \frac{1}{2}\sum_{t=1}^{T}(Y^{(t)} - \hat{Y}^{(t)})^2 + \frac{\lambda}{2}\|w\|_2^2 + \beta\|w\|_1$$

# *Solving it*

- To find best $w$, several options:
  - minimize symbolically by setting $\nabla_w L$ to 0
    - sub-choices: QR, SVD, Cholesky
  - iterative optimizer
    - sub-choices: stochastic vs. full gradient
    - 1st-order like SGD or Adam
    - 2nd-order like Newton
    - in-between like L-BFGS
    - tools like momentum, rescaling
  - choice of covariance form or kernel form

# Schematic

$\mathbb{R}^T \quad \mathbb{R}^{T \times d} \quad \mathbb{R}^d$



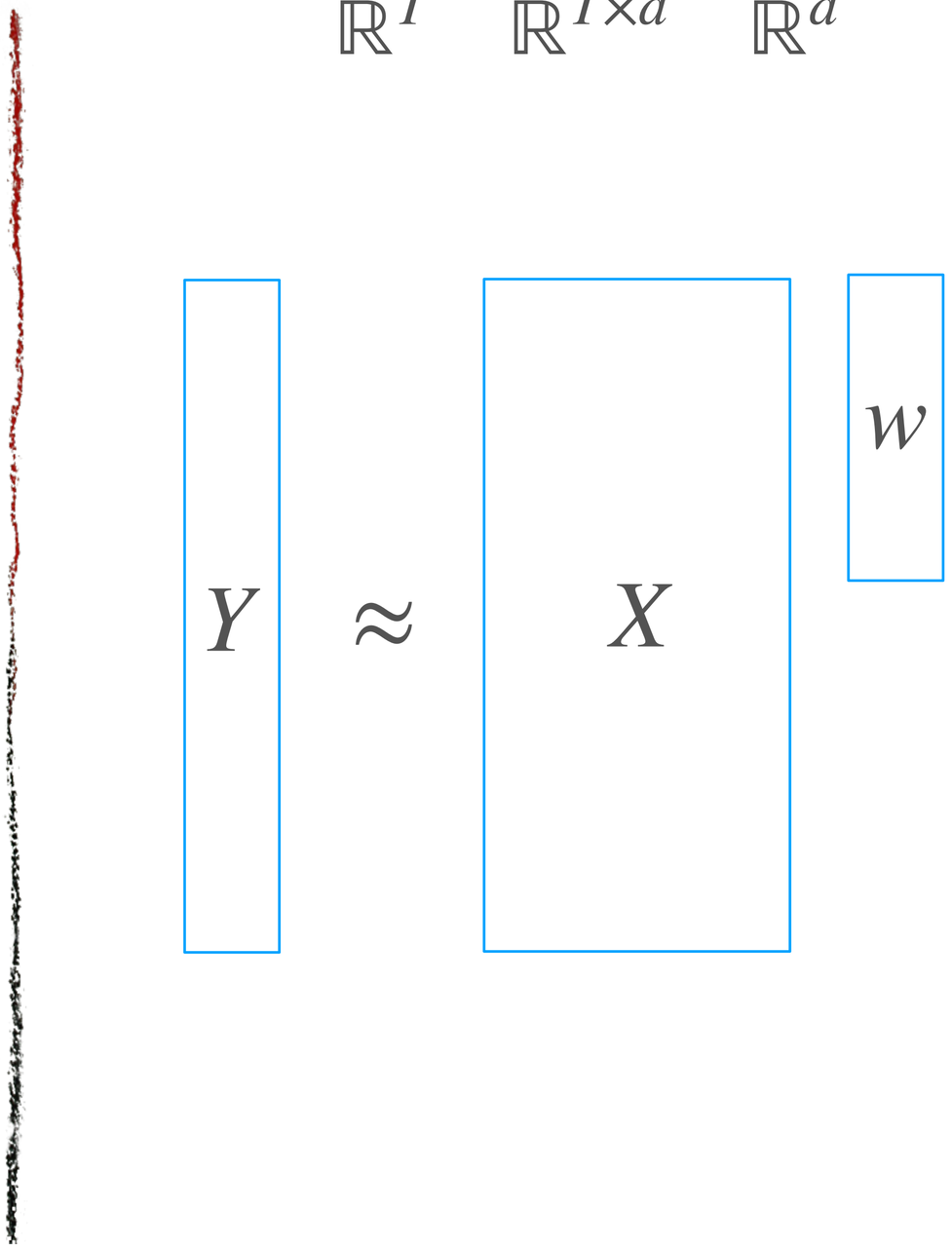

features $X$ labels $y$

| Family History | Resting Blood Pressure | Cholesterol | Cost (k$) |
|---|---|---|---|
| Yes | Low | Normal | 0.1 |
| No | Medium | Normal | 0.2 |
| No | Low | Abnormal | 10.3 |
| Yes | Medium | Normal | 8.7 |
| Yes | High | Abnormal | 13.1 |
| No | Low | Normal | 0.1 |
| No | Medium | Normal | 0.4 |
| Yes | Medium | Abnormal | 9.6 |

- data matrix $X$, label vector $Y$; rows are $X^{(t)}$, $Y^{(t)}$

# Schematic

$\mathbb{R}^T \quad \mathbb{R}^{T \times d} \quad \mathbb{R}^d$



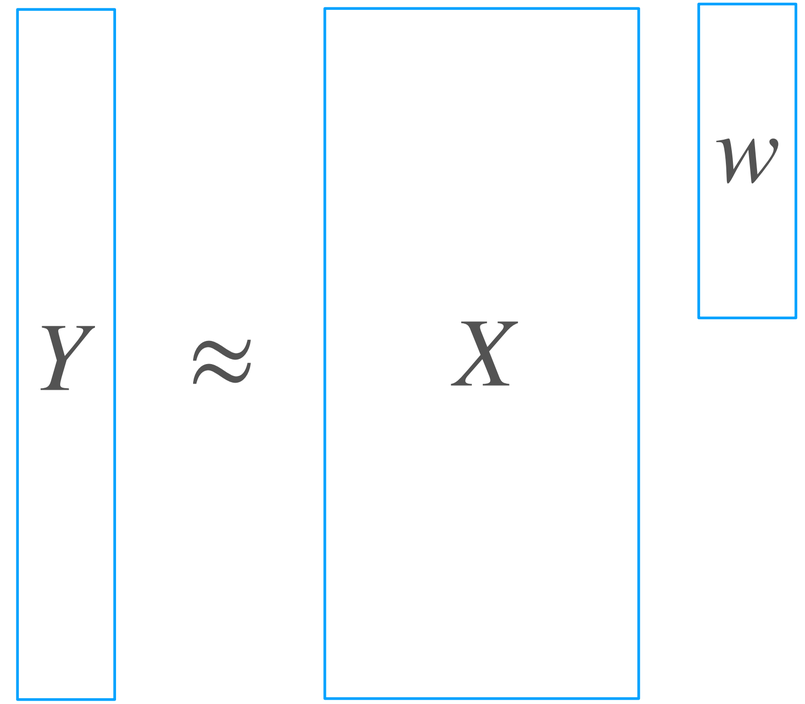

features $X$ labels $y$

| | Family History | Resting Blood Pressure | Cholesterol | Cost (k$) |
|---|---|---|---|---|
| 1 | Yes | Low | Normal | 0.1 |
| 1 | No | Medium | Normal | 0.2 |
| 1 | No | Low | Abnormal | 10.3 |
| 1 | Yes | Medium | Normal | 8.7 |
| 1 | Yes | High | Abnormal | 13.1 |
| 1 | No | Low | Normal | 0.1 |
| 1 | No | Medium | Normal | 0.4 |
| 1 | Yes | Medium | Abnormal | 9.6 |

- data matrix $X$, label vector $Y$; rows are $X^{(t)}$, $Y^{(t)}$

# Schematic

$$\mathbb{R}^T \quad \mathbb{R}^{T \times d} \quad \mathbb{R}^d$$




features $X$ labels $y$

| | Family History | Resting Blood Pressure | Cholesterol | Cost (k$) |
|---|---|---|---|---|
| 1 | Yes | Low | Normal | 0.1 |
| 1 | No | Medium | Normal | 0.2 |
| 1 | No | Low | Abnormal | 10.3 |
| 1 | Yes | Medium | Normal | 8.7 |
| 1 | Yes | High | Abnormal | 13.1 |
| 1 | No | Low | Normal | 0.1 |
| 1 | No | Medium | Normal | 0.4 |
| 1 | Yes | Medium | Abnormal | 9.6 |

$$L(w) = \|Y - Xw\|_F^2 \quad [+ R(w)]$$

↳ or 2

- data matrix $X$, label vector $Y$; rows are $X^{(t)}$, $Y^{(t)}$

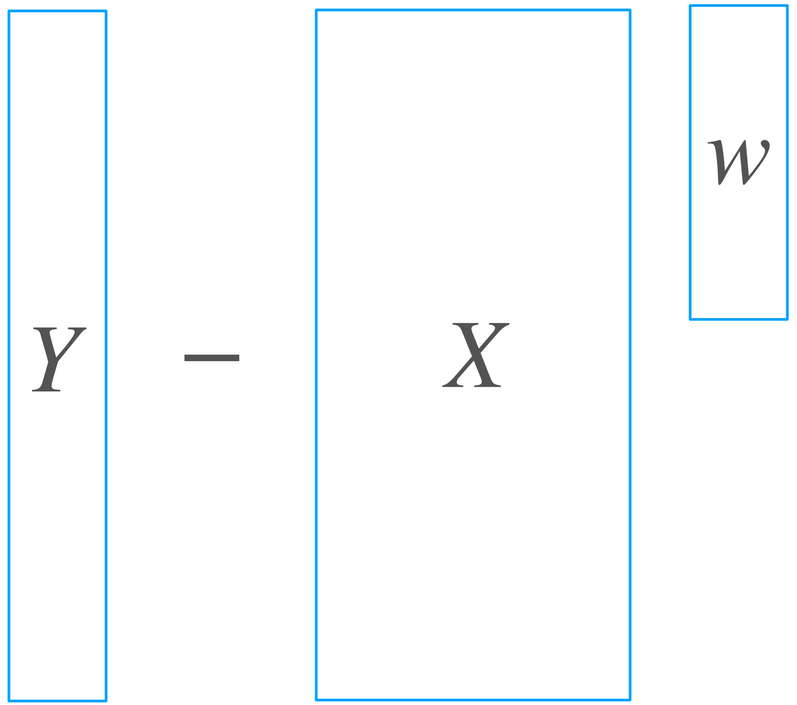
***MOO!***

- Previous slides illustrate *model, objective, optimizer* for linear regression
- Often three good questions to ask when learning a new ML method — maybe not complete, but a good place to start

# *MOO examples*

- Linear regression
  - model = linear functions of features
  - objective = sum of squared errors (+ regularizer)
  - optimizer = exact or SGD
- ID3
  - model = decision trees
  - objective = mutual information
  - optimizer = greedy top-down induction

# Gradients




- For any of the above, need gradients
  - $L(w) = \frac{\lambda}{2}\|w\|^2 + \frac{1}{2}\sum_{t=1}^{T}(Y^{(t)} - w \cdot X^{(t)})^2$

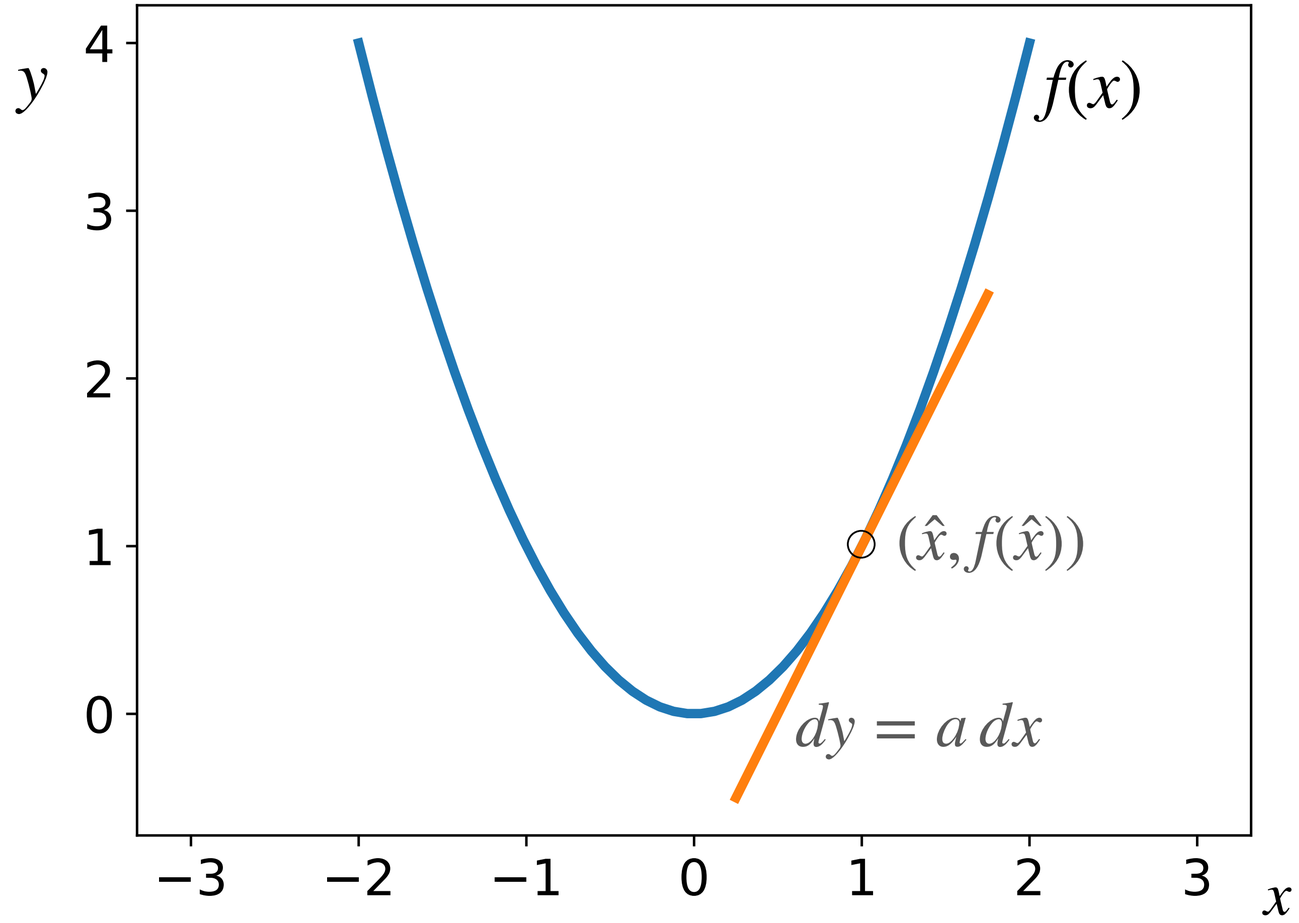
    $= \frac{\lambda}{2}\|w\|^2 + \frac{1}{2}\|Y - Xw\|^2$
  - $\nabla L(w) =$ ?

# *Derivatives everywhere*

- The way we derive many ML methods:
  - start from objective to minimize (here, sum of squared errors plus optional regularizer)
  - take a few derivatives ($0 = d\,\text{objective} / dw$)
  - find optimum (e.g., by setting to 0 and solving, or SGD)
  - done!
- Will continue to be a common pattern
- Review of intuition and tools for high-d derivatives, so we can do it fast

# Derivatives notation

*dx, dy are called* ***differentials***




$m = n = 1$

$dy = y - f(\hat{x})$

$dx = x - \hat{x}$

- Function $y = f(x)$
  - $x \in \mathbb{R}^n$, $y \in \mathbb{R}^m$, $f \in \mathbb{R}^n \to \mathbb{R}^m$
- Derivative is a *linear fn* that *locally* approximates $f$ near $\hat{x}$
  - $dy = a\,dx$, where $a \in \mathbb{R}$ is the derivative

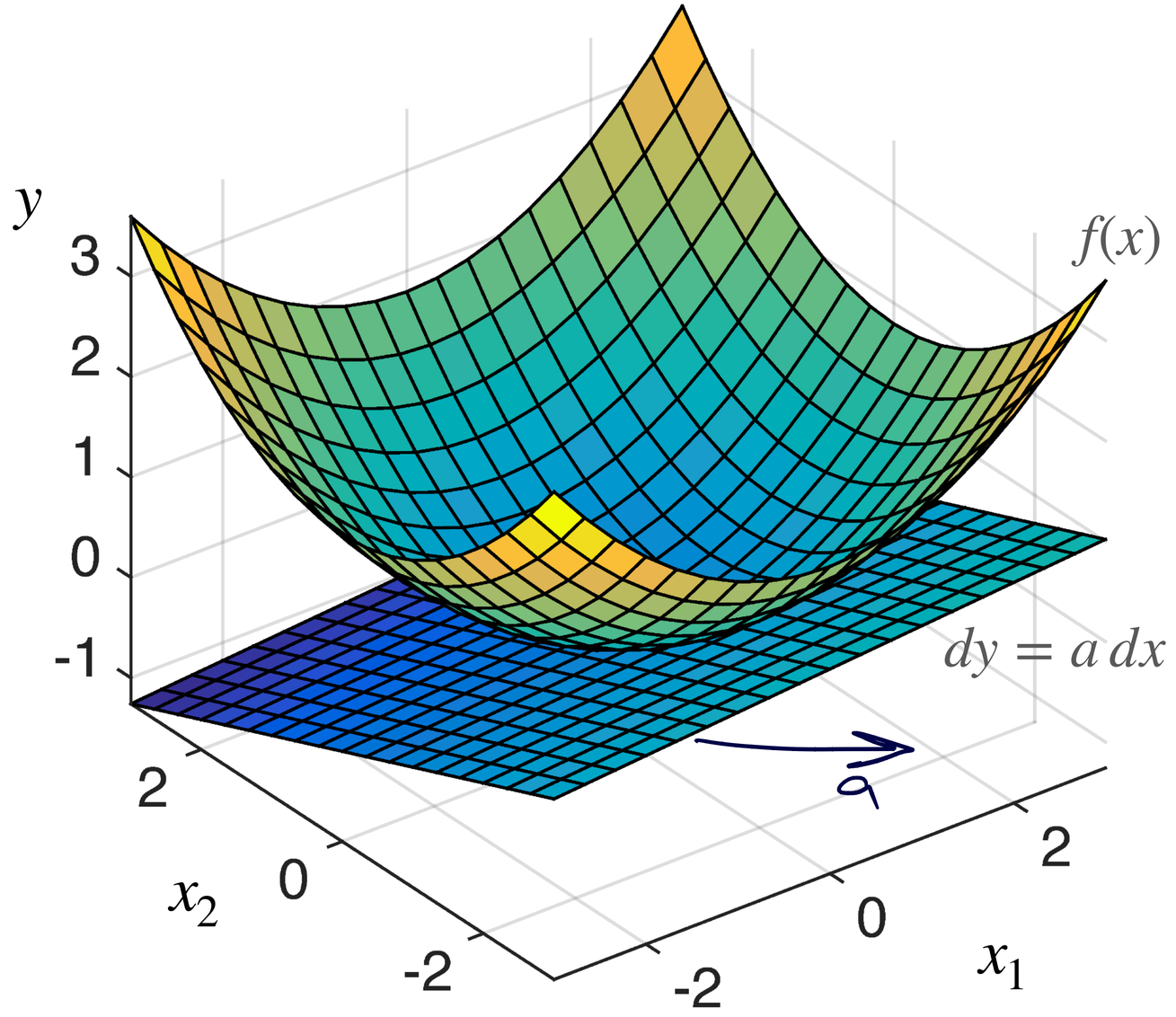
# *Compare older vs. newer notation*

$dy = a\,dx$

$\frac{dy}{dx} = a$

- Older version:
  - $\frac{dy}{dx}$ is atomic: derivative of $y$ wrt $x$
  - $dx$ and $dy$ are meaningless on their own
  - focused on infinitesimals: change in $x$ or $y \to 0$
- Newer version:
  - $dx$ and $dy$ have separate meaning: terms in a linear function (the Taylor expansion)
  - $dx$ and $dy$ can be any size: linear function is well defined even for large changes in $x$ or $y$ (though accuracy of Taylor approximation can be bad if $dx$, $dy$ are big)
- Heuristic to convert: “divide through” or “multiply through” by $dx$ (only makes sense in scalar case, but often succeeds in converting notation anyway)

# *Higher dimensions*

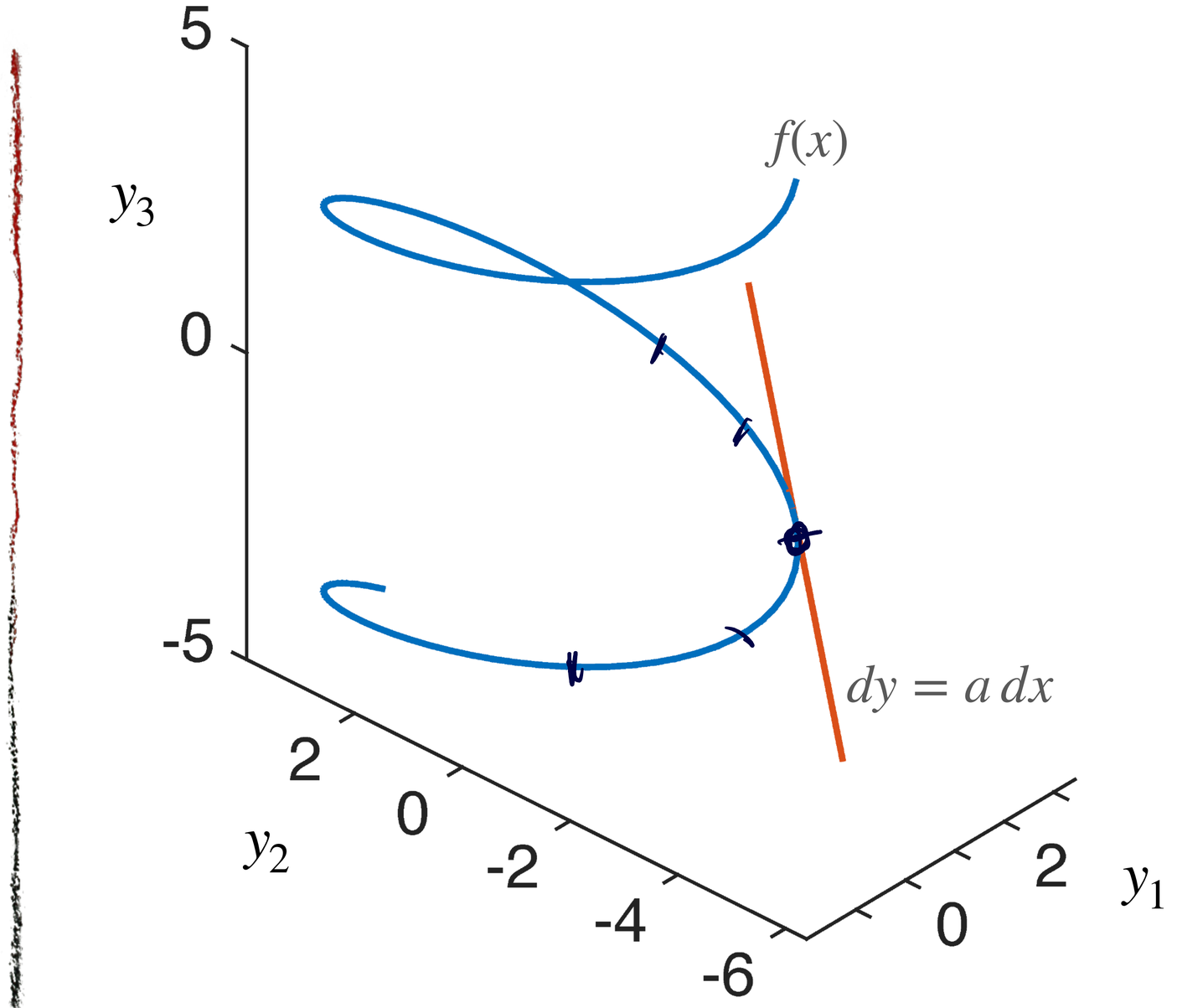



$\mathbb{R}^n \to \mathbb{R}^m$

$m = 1, n = 2$

Function $y = f(x)$ $dy = a\,dx$ $a \in \mathbb{R}^{1\times 2}$

# *Higher dimensions*




$\mathbb{R}^n \to \mathbb{R}^m$

$m = 3, n = 1$

Function $y = f(x)$ $dy = a\,dx$ $a \in \mathbb{R}^{3\times 1}$

# *Notation for linear functions*

output shape (derivative of)

| | input shape (with respect to) Scalar | Vector | Matrix |
|---|---|---|---|
| Scalar | $ds = a\,dt$ | $ds = u^T dv$ | $ds = \mathrm{tr}(M^T dN)$ |
| Vector | $du = v\,ds$ | $du = M\,dv$ | $\times$ |
| Matrix | $dM = N\,ds$ | $\times$ | $\times$ |

- $a, s, t$: scalars  $u, v$: column vectors  $M, N$: matrices
- note: $ds = a\,dt$ is a synonym for $a = \frac{ds}{dt}$, etc.
- $\times$ means no common notation; see torch.einsum

# *Notation for linear functions*

| output shape (derivative of) \ input shape (with respect to) | Scalar | Vector | Matrix |
|---|---|---|---|
| Scalar | $ds = a\,dt$ | gradient | $ds = \mathrm{tr}(M^T dN)$ |
| Vector | velocity | Jacobian | × |
| Matrix | $dM = N\,ds$ | × | × |

- $a, s, t$: scalars  $u, v$: column vectors  $M, N$: matrices
- note: $ds = a\,dt$ is a synonym for $a = \frac{ds}{dt}$, etc.
- × means no common notation; see torch.einsum

# *Inner product*

- With scalar output $s$, for any shape of input $X$, can write $ds = \langle A, dX \rangle$ where $A, X, dX$ all have same shape
  - $\langle \cdot, \cdot \rangle$ is ***inner product***, in our case $\sum_{\text{index lists } I} A_I \, dX_I$
  - e.g., if $A, dX$ are $m \times n$ matrices, then

$$\langle A, dX \rangle = \sum_{i=1}^{m} \sum_{j=1}^{n} A_{ij} \, dX_{ij}$$

- Expressions on previous slide are equivalent to above
  - most complex is $\text{tr}(M^\top dN)$
  - tr is sum of diagonal elements of a square matrix

$$\left[M^\top dN\right]_{jk} =$$

  - sum of diagonal elements means set $j = k$, sum over $j$:

# *Shape conventions*

- Derivative = coefficient of $ds, dt, dv, dN$ on RHS
- Ambiguity: can write $\langle M, dN \rangle$ or $\text{tr}(M^\top dN)$
  - is the derivative $M$ or $M^\top$?
- Convention: $x$, $y$, and derivative $\frac{dy}{dx}$ are ***tensors*** (multidimensional arrays of numbers)
- In derivative $\frac{dy}{dx}$:
  - the dimensions of $y$ come first, then $x$
  - in same order as dimensions of $y$ and $x$
- For example: if $y$ is $3 \times 2$ and $x$ is $4 \times 6 \times 5$ then $\frac{dy}{dx}$ is $3 \times 2 \times 4 \times 6 \times 5$

# *Special cases*

- If $y$ is a scalar, then $\frac{dy}{dx}$ is the same shape as $x$
  - derivative of a scalar wrt a $5 \times 3$ matrix is a $5 \times 3$ matrix
- If $x$ and $y$ are vectors, $\frac{dy}{dx}$ (the Jacobian) is the same shape as a linear function from $x$ to $y$
  - derivative of a $3$-vector wrt a $6$-vector is a $3 \times 6$ matrix

# *Useful identities*

*compatible dimensions:*

- *2nd dimension of $A$ = 1st dimension of $B$*
- *2nd dimension of $B$ = 1st dimension of $C$*
- *2nd dimension of $C$ = 1st dimension of $A$*

- scalar = scalar$^\top$ = tr(scalar)
- Trace rotation
  - $\text{tr}(ABC) = \text{tr}(CAB)$ when $A, B, C$ have compatible dimensions
- Transpose-Hadamard
  - $A \circ B =$ Hadamard (componentwise) product,

    $$[A \circ B]_{ij} = A_{ij} B_{ij}$$
  - $A^\top (B \circ C) = (A \circ B)^\top C$

# *Tools for derivatives: linearity, chain, product*

- Linearity:
  - $d(ax + b) = a\,dx + db$, $d(AX + B) = A\,dX + dB$, …
- Chain rule for $f(g(x))$
  - old notation: $\frac{df}{dx} = \frac{df}{dg}\frac{dg}{dx}$ (“cancel the $dg$”)
  - new: if $df = A\,dg$, $dg = B\,dx$ then $df = AB\,dx$
- Product rule for $f(x)\,g(x)$
  - $d(fg) = df\,g + f\,dg$
  - works for any kind of product (matrix product, vector cross product, elementwise (Hadamard) product, vector convolution, Khatri-Rao product …)
  - unlike scalar case, order can matter (products are not necessarily commutative)

# *Partial vs. total derivatives*

- In a function of several variables, partial derivative imagines changing one of them, holding others fixed
  - e.g., $\frac{\partial}{\partial x}(x+y)^2 =$ $2(x+y)$ holding $y$ fixed

- New notation: any variable that appears with $d$ in RHS is changing, others are fixed
  - e.g., $d(x+y+z)^2 =$ holding $z$ fixed

$$2(x+y+z)\underbrace{d(x+y+z)}_{(dx+dy)}$$

# *Example: chain rule with multiple variables*

$du = \sin x \, dx$

$dv = 2x \, dx$

- Ex: $y = (u+v)^3, \; u = \cos x, \; v = x^2$

$$dy = 3(u+v)^2 (du + dv)$$
$$= 3(u+v)^2 (\sin x + 2x) \, dx$$

- Sometimes called ***law of total derivatives***:

$$df(u(x), v(x), w(x)) = f^{(1)}(u, v, w) \, u'(x) \, dx + f^{(2)}(u, v, w) \, v'(x) \, dx + f^{(3)}(u, v, w) \, w'(x) \, dx$$

# Gradient practice: Linear regression in one slide

$$L = \tfrac{1}{2}\|Y - Xw\|^2 = \tfrac{1}{2}(Y - Xw)^\top (Y - Xw)$$

$$L = \tfrac{1}{2}(Y^\top Y - 2Y^\top Xw + w^\top X^\top Xw)$$

$$dL = \tfrac{1}{2}(0 - 2Y^\top X\,dw + dw^\top X^\top Xw + w^\top X^\top X\,dw)$$

$$dL = -Y^\top X\,dw + w^\top X^\top X\,dw$$

$$= (w^\top X^\top X - Y^\top X)\,dw$$

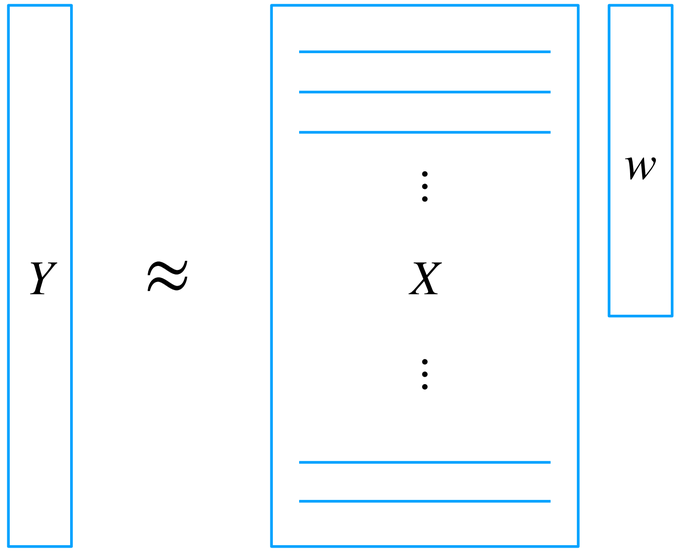
$$0 = X^\top Xw - X^\top Y$$

$$\boxed{X^\top Xw = X^\top Y}$$

normal equations

# *Exact solution for ridge regression*

$X \in \mathbb{R}^{T \times d}$
$Y \in \mathbb{R}^{T}$
$w \in \mathbb{R}^{d}$




- Predictions are $\hat{Y} = Xw$
- Objective is $\frac{\lambda}{2}\|w\|^2 + \frac{1}{2}\|Y - Xw\|^2$
- Gradient is $\lambda w + X^\top(Xw - Y)$
- Exact solution is $(\lambda I + X^\top X)w = X^\top Y$
  - implemented in `torch.linalg.lstsq`

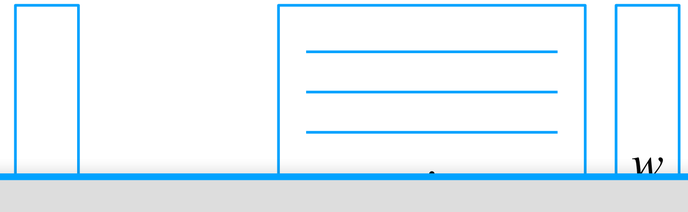
or ^p.

# *Exact solution for ridge regression*

$X \in \mathbb{R}^{T \times d}$

$Y \in \mathbb{R}^{T}$

$w \in \mathbb{R}^{d}$




- Pred
- Obje
- Grad
- Exact solution is $(X^T X + \lambda I)^{-1} X^T Y$
  - implemented in `torch.linalg.lstsq`

*Why use torch.linalg.lstsq? Handles (many) special cases: over- or under-determined, rank-deficient, poor conditioning*

*It would be several days of lectures to describe algorithms and best practices for all of these*

*Please remember: don't write inv() unless you are sure none of the special cases will bite you*

https://forms.gle/hoenoXZA57MZ4GS3A

# Poll

- In which of the following situations is selection bias the *largest*:
  - A: We roll 3 six-sided dice and take the largest
  - B: We roll 10 six-sided dice and take the largest
  - C: We roll 10 six-side dice and take the largest; the dice are weighted so that the mean is still $3\frac{1}{2}$, but extreme outcomes 1 & 6 have lower probability
  - D: We roll 15 six-sided dice; 5 of them are normal, but 10 are weighted so that they always come up 1 or 2
  - E: All the above are about the same

- Bias = $\mathbb{E}$(observed value of selected die) – $\mathbb{E}$(that same die if we roll it again)