

Deep Learning for Vision & Language

Machine Learning II: Optimization / Regularization /
Generalization



RICE UNIVERSITY

ML Classifier / Regression models

- K-nearest neighbors
- Linear classifier / Linear regression
- Naïve Bayes classifiers
- Decision Trees
- Random Forests
- Boosted Decision Trees
- Neural Networks

Linear Regression

Example: Hollywood movie data

input variables x

output variables y

production costs	promotional costs	genre of the movie	box office first week	total book sales	total revenue USA	total revenue international
$x_1^{(1)}$	$x_2^{(1)}$	$x_3^{(1)}$	$x_4^{(1)}$	$x_5^{(1)}$	$x_6^{(1)}$	$x_7^{(1)}$
$x_1^{(2)}$	$x_2^{(2)}$	$x_3^{(2)}$	$x_4^{(2)}$	$x_5^{(2)}$	$x_6^{(2)}$	$x_7^{(2)}$
$x_1^{(3)}$	$x_2^{(3)}$	$x_3^{(3)}$	$x_4^{(3)}$	$x_5^{(3)}$	$x_6^{(3)}$	$x_7^{(3)}$
$x_1^{(4)}$	$x_2^{(4)}$	$x_3^{(4)}$	$x_4^{(4)}$	$x_5^{(4)}$	$x_6^{(4)}$	$x_7^{(4)}$
$x_1^{(5)}$	$x_2^{(5)}$	$x_3^{(5)}$	$x_4^{(5)}$	$x_5^{(5)}$	$x_6^{(5)}$	$x_7^{(5)}$

Linear Regression

Example: Hollywood movie data

input variables x

output variables y

production costs	promotional costs	genre of the movie	box office first week	total book sales	total revenue USA	total revenue international
$x_1^{(1)}$	$x_2^{(1)}$	$x_3^{(1)}$	$x_4^{(1)}$	$x_5^{(1)}$	$y_1^{(1)}$	$y_2^{(1)}$
$x_1^{(2)}$	$x_2^{(2)}$	$x_3^{(2)}$	$x_4^{(2)}$	$x_5^{(2)}$	$y_1^{(2)}$	$y_2^{(2)}$
$x_1^{(3)}$	$x_2^{(3)}$	$x_3^{(3)}$	$x_4^{(3)}$	$x_5^{(3)}$	$y_1^{(3)}$	$y_2^{(3)}$
$x_1^{(4)}$	$x_2^{(4)}$	$x_3^{(4)}$	$x_4^{(4)}$	$x_5^{(4)}$	$y_1^{(4)}$	$y_2^{(4)}$
$x_1^{(5)}$	$x_2^{(5)}$	$x_3^{(5)}$	$x_4^{(5)}$	$x_5^{(5)}$	$y_1^{(5)}$	$y_2^{(5)}$

Linear Regression

Example: Hollywood movie data

input variables x

output variables y

	production costs	promotional costs	genre of the movie	box office first week	total book sales	total revenue USA	total revenue international
Training data	$x_1^{(1)}$	$x_2^{(1)}$	$x_3^{(1)}$	$x_4^{(1)}$	$x_5^{(1)}$	$y_1^{(1)}$	$y_2^{(1)}$
	$x_1^{(2)}$	$x_2^{(2)}$	$x_3^{(2)}$	$x_4^{(2)}$	$x_5^{(2)}$	$y_1^{(2)}$	$y_2^{(2)}$
	$x_1^{(3)}$	$x_2^{(3)}$	$x_3^{(3)}$	$x_4^{(3)}$	$x_5^{(3)}$	$y_1^{(3)}$	$y_2^{(3)}$
Test data	$x_1^{(4)}$	$x_2^{(4)}$	$x_3^{(4)}$	$x_4^{(4)}$	$x_5^{(4)}$	$y_1^{(4)}$	$y_2^{(4)}$
	$x_1^{(5)}$	$x_2^{(5)}$	$x_3^{(5)}$	$x_4^{(5)}$	$x_5^{(5)}$	$y_1^{(5)}$	$y_2^{(5)}$

Linear Regression

$$\hat{y} = \sum_i w_i x_i$$

$$\hat{y} = W^T x$$

Prediction,
Inference,
Testing

$$D = \{(x^{(d)}, y^{(d)})\}$$

$$L(W) = \sum_{d=1}^{|D|} l(\hat{y}^{(d)}, y^{(d)})$$

$$W^* = \operatorname{argmin} L(W)$$

Training,
Learning,
Parameter
estimation
Objective
minimization

Linear Regression

$$\hat{y}_j = \sum_i w_{ji} x_i$$

$$\hat{y} = W^T x$$

Prediction,
Inference,
Testing

$$D = \{(x^{(d)}, y^{(d)})\}$$

$$L(W) = \sum_{d=1}^{|D|} \sum_j l(\hat{y}_j^{(d)}, y_j^{(d)})$$

$$W^* = \operatorname{argmin} L(W)$$

Training,
Learning,
Parameter
estimation
Objective
minimization

Linear Regression – Least Squares

$$\hat{y}_j = \sum_i w_{ji} x_i$$

$$\hat{y} = W^T x$$

$$D = \{(x^{(d)}, y^{(d)})\}$$

$$L(W) = \sum_{d=1}^{|D|} \sum_j (\hat{y}_j^{(d)} - y_j^{(d)})^2$$

$$W^* = \operatorname{argmin} L(W)$$

Training,
Learning,
Parameter
estimation
Objective
minimization

Linear Regression – Least Squares

$$L(W) = \sum_{d=1}^{|D|} \sum_j \left(\hat{y}_j^{(d)} - y^{(d)} \right)^2$$

$$\hat{y}_j^{(d)} = \sum_i w_{ji} x_i^{(d)}$$

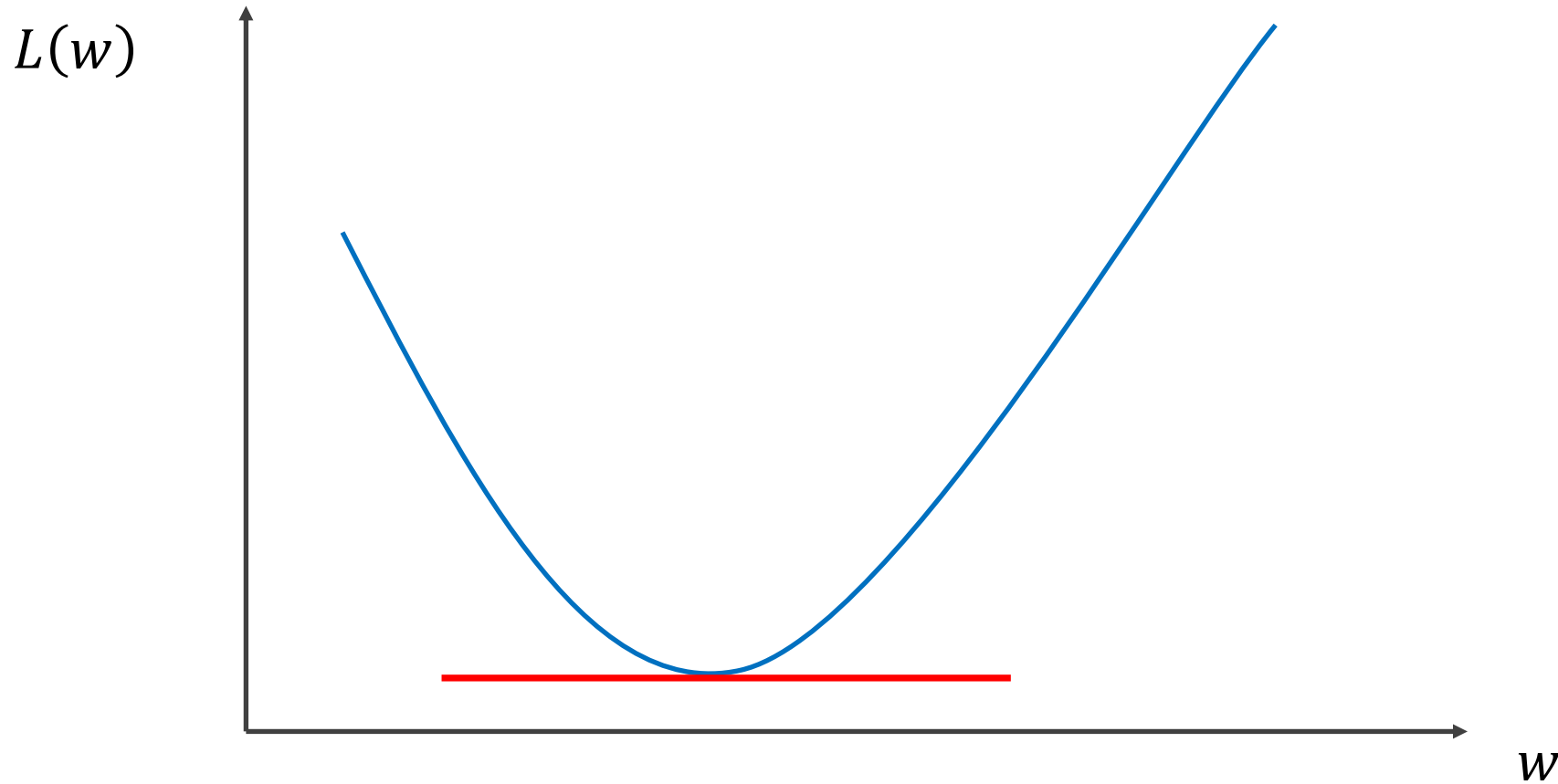
$$L(W) = \sum_{d=1}^{|D|} \sum_j \left(\sum_i w_{ji} x_i^{(d)} - y^{(d)} \right)^2$$

Linear Regression – Least Squares

$$L(W) = \sum_{d=1}^{|D|} \sum_j \left(\sum_i w_{ji} x_i^{(d)} - y^{(d)} \right)^2$$

$$W^* = \operatorname{argmin} L(W)$$

How to find the minimum of a function $L(w)$?



$$\frac{\partial L(w)}{\partial w} = 0$$

Linear Regression – Least Squares

$$\frac{\partial L(W)}{\partial w_{ji}} = \frac{\partial}{\partial w_{ji}} \left(\sum_{d=1}^{|D|} \sum_j \left(\sum_i w_{ji} x_i^{(d)} - y^{(d)} \right)^2 \right)$$

$$\frac{\partial L(W)}{\partial w_{ji}} = 0$$

...

$$W = (X^T X)^{-1} X^T Y$$

Linear Regression – Least Squares

$$W = (X^T X)^{-1} X^T Y$$

$$W = \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \end{bmatrix}$$

$$X = \begin{bmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \\ x_{41} & x_{42} & x_{43} \\ x_{51} & x_{52} & x_{53} \end{bmatrix}$$

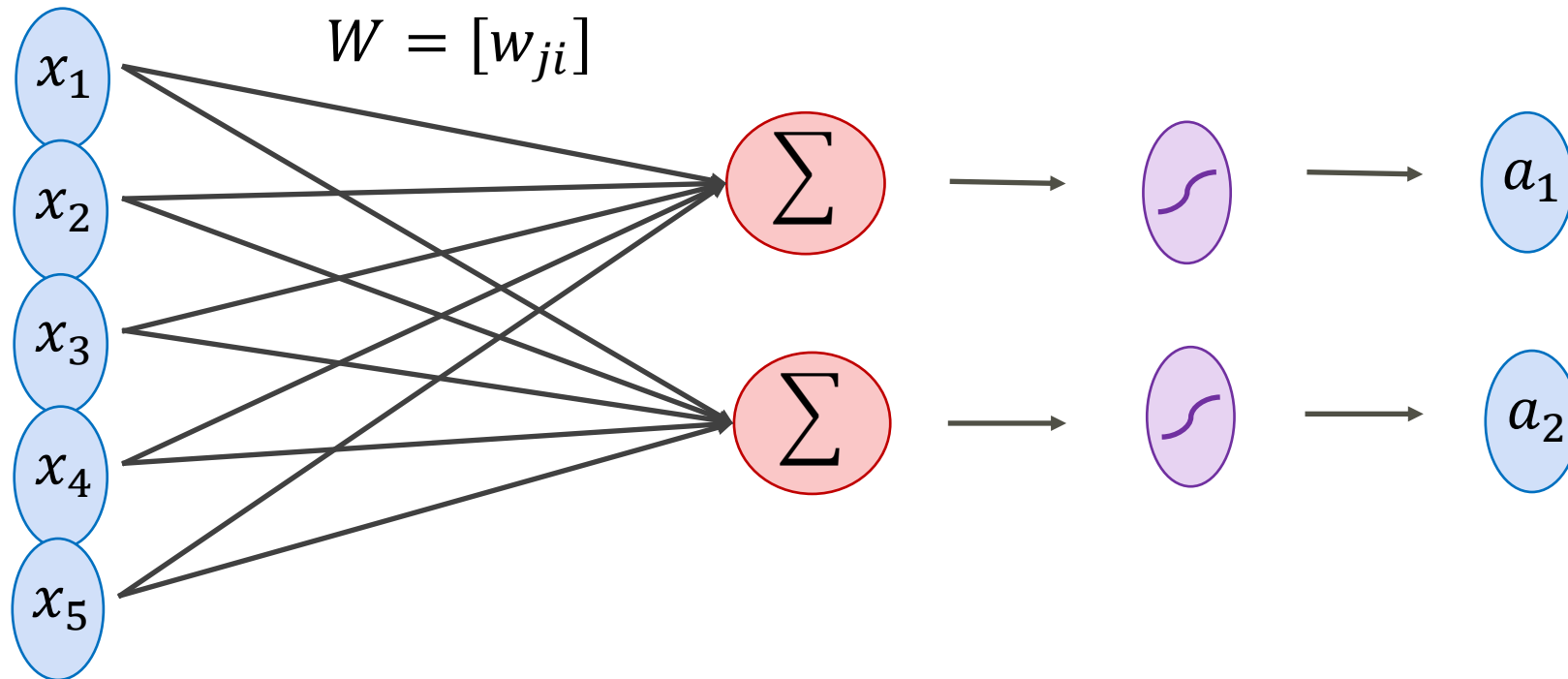
$$Y = \begin{bmatrix} y_{11} & y_{12} \\ y_{21} & y_{22} \\ y_{31} & y_{32} \\ y_{41} & y_{42} \\ y_{51} & y_{52} \end{bmatrix}$$

ML Classifier / Regression models

- K-nearest neighbors
- Linear classifier / Linear regression
- Naïve Bayes classifiers
- Decision Trees
- Random Forests
- Boosted Decision Trees
- Neural Networks

Neural Network with One Layer

$$\text{sigmoid}(z) = \frac{1}{1 + e^{-z}}$$



$$a_j = \text{sigmoid}(\sum_i w_{ji}x_i + b_j)$$

Neural Network with One Layer

$$L(W, b) = \sum_{d=1}^{|D|} (a^{(d)} - y^{(d)})^2$$

$$a_j^{(d)} = \text{sigmoid}\left(\sum_i w_{ji} x_i^{(d)} + b_j\right)$$

← Bias parameters

$$L(W, b) = \sum_{j,d} \left(\text{sigmoid}\left(\sum_i w_{ji} x_i^{(d)} + b_j\right) - y_j^{(d)} \right)^2$$

Neural Network with One Layer

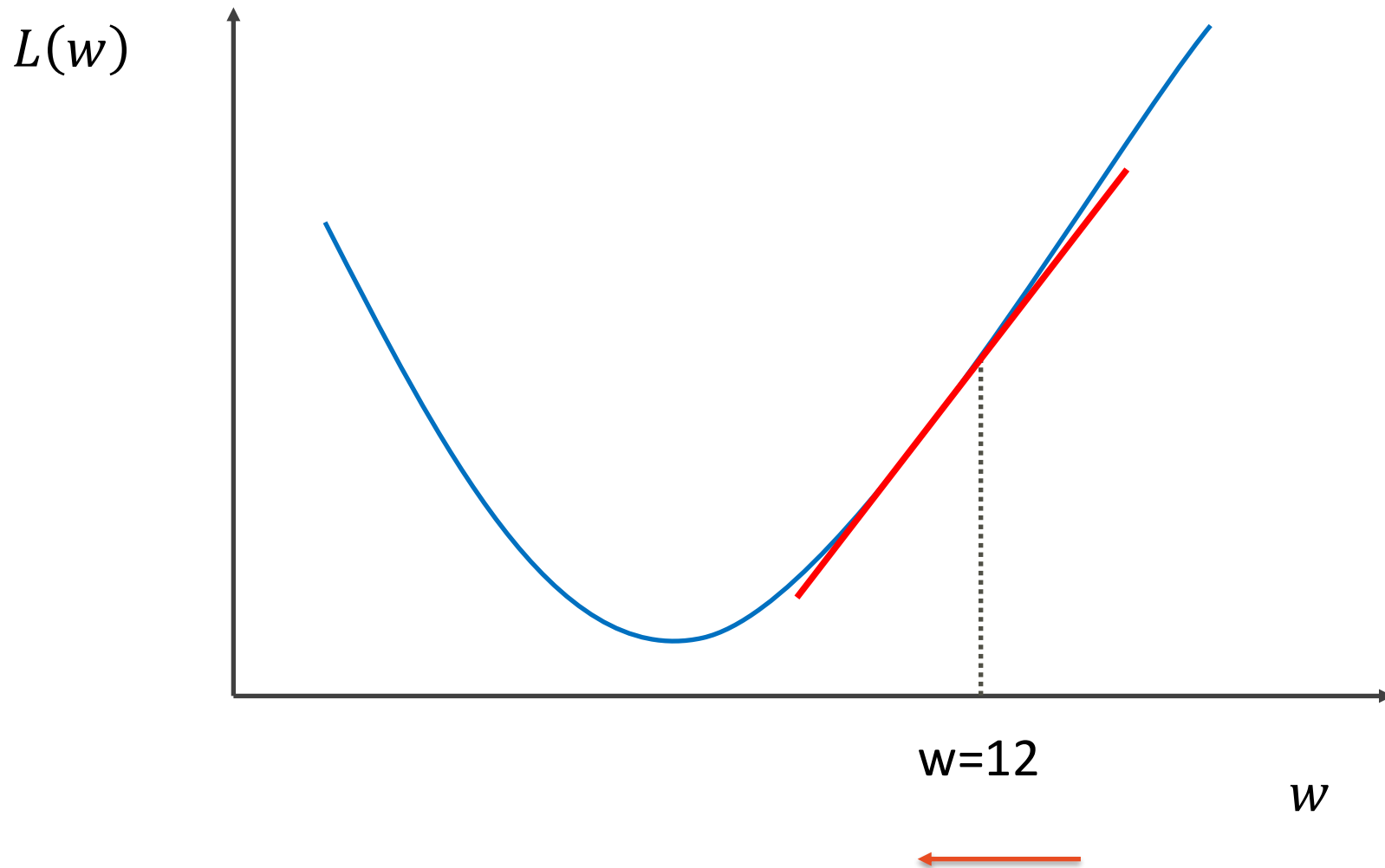
$$L(W, b) = \sum_{j,d} \left(\text{sigmoid} \left(\sum_i w_{ji} x_i^{(d)} + b_j \right) - y_j^{(d)} \right)^2$$

$$\frac{\partial L}{\partial w_{uv}} = 0$$

(1) We can compute this derivative but often there will be no closed-form solution for W when $dL/dw = 0$

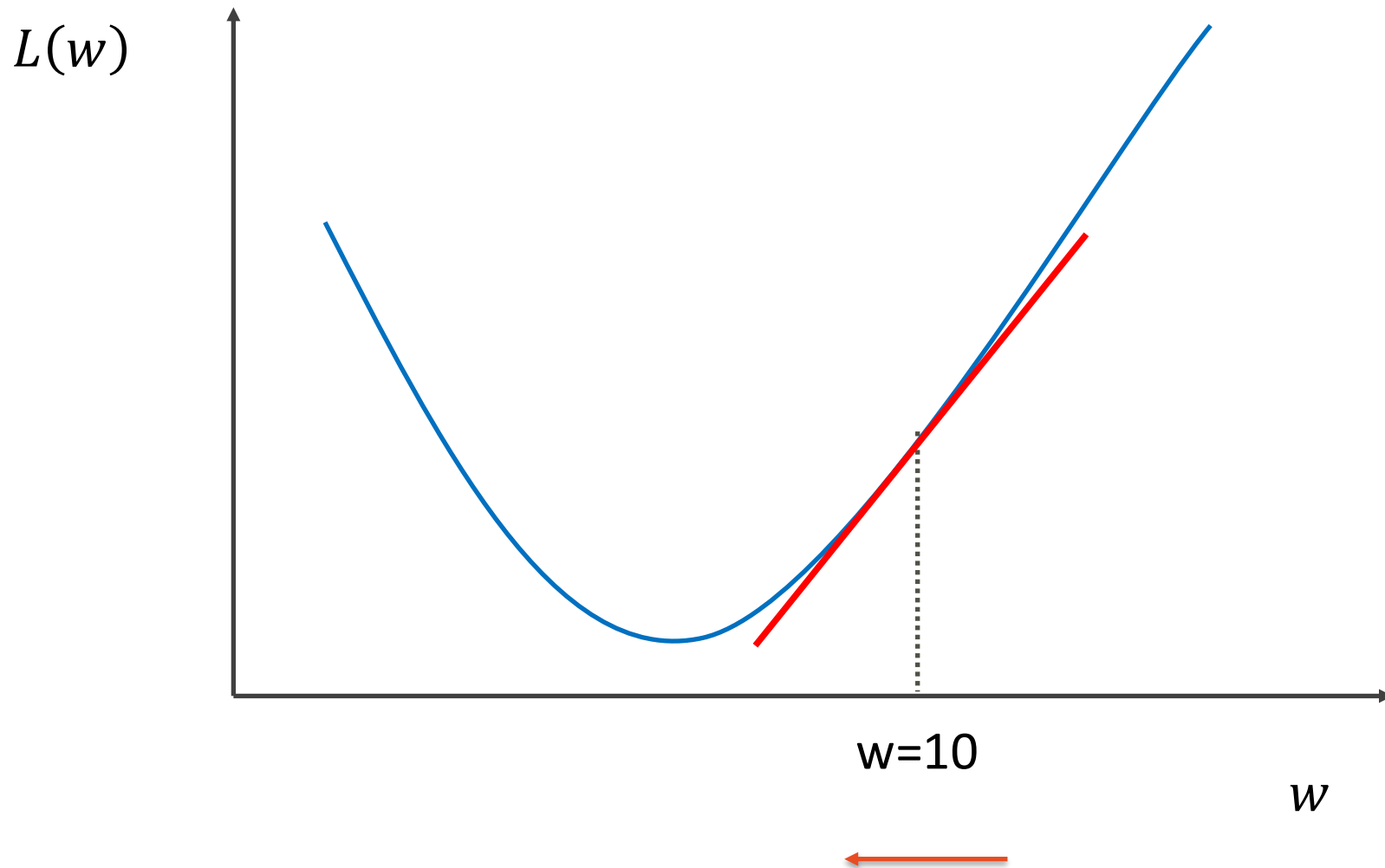
(2) Also, even for linear regression where the solution was $W = (X^T X)^{-1} X^T Y$, computing this expression might be expensive or infeasible. e. g. think of computing $(X^T X)^{-1}$ for a very large dataset with a million x_i

Gradient Descent



1. Start with a random value of w (e.g. $w = 12$)
2. Compute the gradient (derivative) of $L(w)$ at point $w = 12$. (e.g. $dL/dw = 6$)
3. Recompute w as:
$$w = w - \text{lambda} * (dL / dw)$$

Gradient Descent

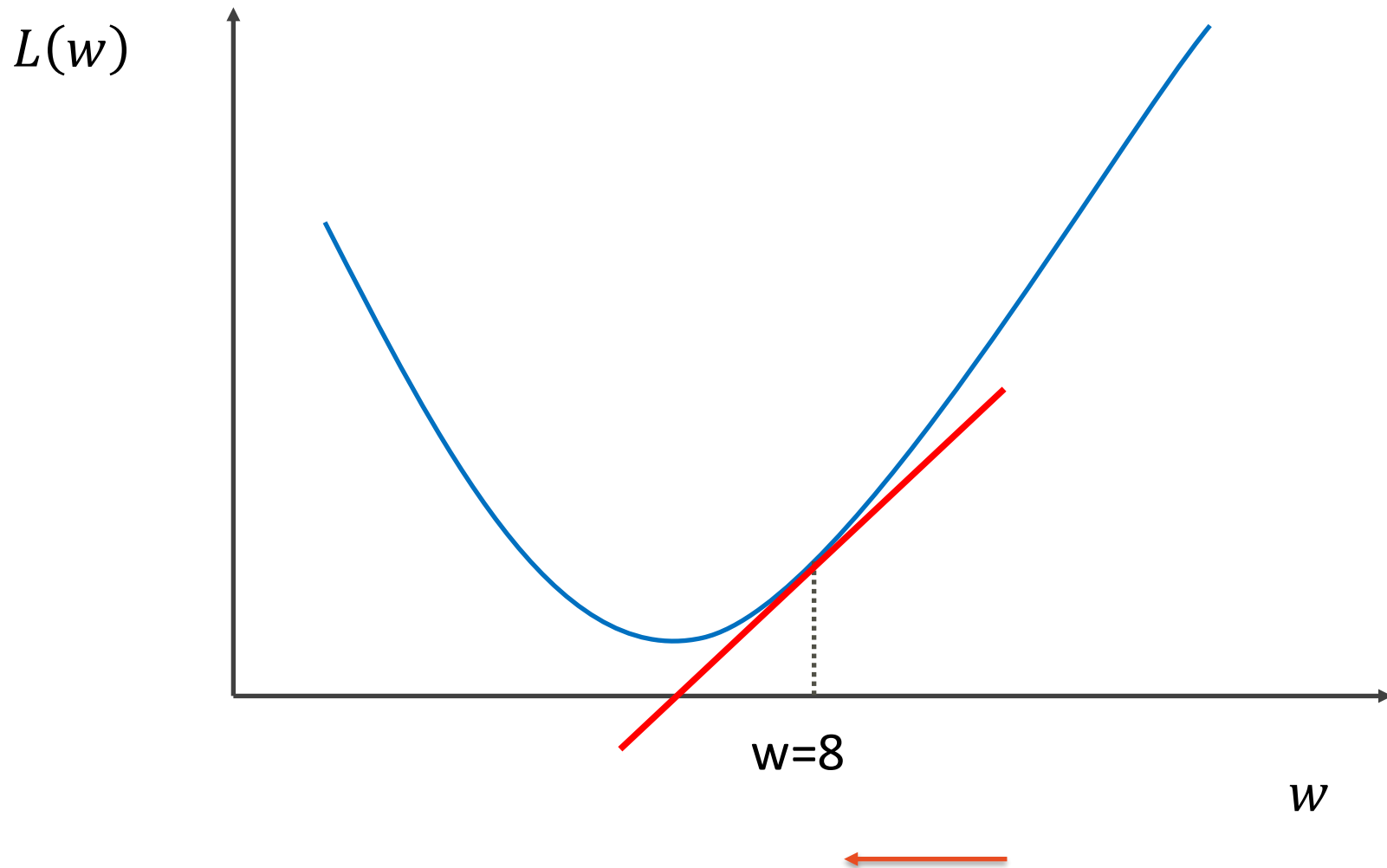


2. Compute the gradient (derivative) of $L(w)$ at point $w = 12$. (e.g. $dL/dw = 6$)

3. Recompute w as:

$$w = w - \text{lambda} * (dL / dw)$$

Gradient Descent



2. Compute the gradient (derivative) of $L(w)$ at point $w = 12$. (e.g. $dL/dw = 6$)

3. Recompute w as:

$$w = w - \text{lambda} * (dL / dw)$$

Gradient Descent

$\lambda = 0.01$

Initialize w and b randomly

$$L(w, b) = \sum_{i=1}^n l(w, b)$$

expensive

for $e = 0, \text{ num_epochs}$ **do**

 Compute: $dL(w, b)/dw$ and $dL(w, b)/db$

 Update w : $w = w - \lambda dL(w, b)/dw$

 Update b : $b = b - \lambda dL(w, b)/db$

 Print: $L(w, b)$ // Useful to see if this is becoming smaller or not.

end

Stochastic Gradient Descent (mini-batch)

$\lambda = 0.01$

Initialize w and b randomly

$$L_B(w, b) = \sum_{i=1}^B l(w, b)$$

for $e = 0, \text{num_epochs}$ **do**

for $b = 0, \text{num_batches}$ **do**

 Compute: $dL_B(w, b)/dw$ and $dL_B(w, b)/db$

 Update w : $w = w - \lambda dl(w, b)/dw$

 Update b : $b = b - \lambda dl(w, b)/db$

 Print: $L_B(w, b)$ // Useful to see if this is becoming smaller or not.

end

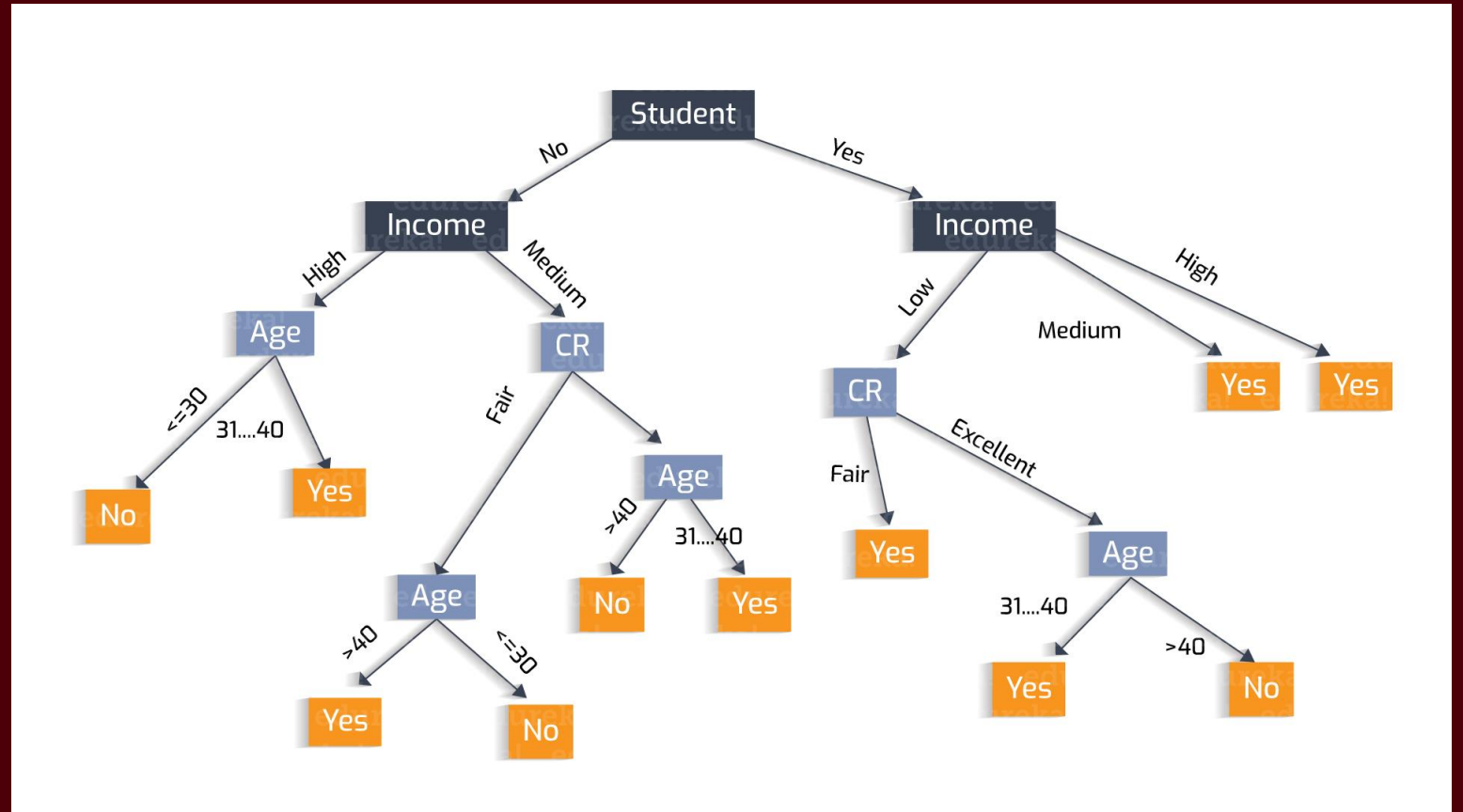
end

In this class we will mostly rely on...

- K-nearest neighbors
- Linear classifiers
- Naïve Bayes classifiers
- Decision Trees
- Random Forests
- Boosted Decision Trees
- Neural Networks

Why?

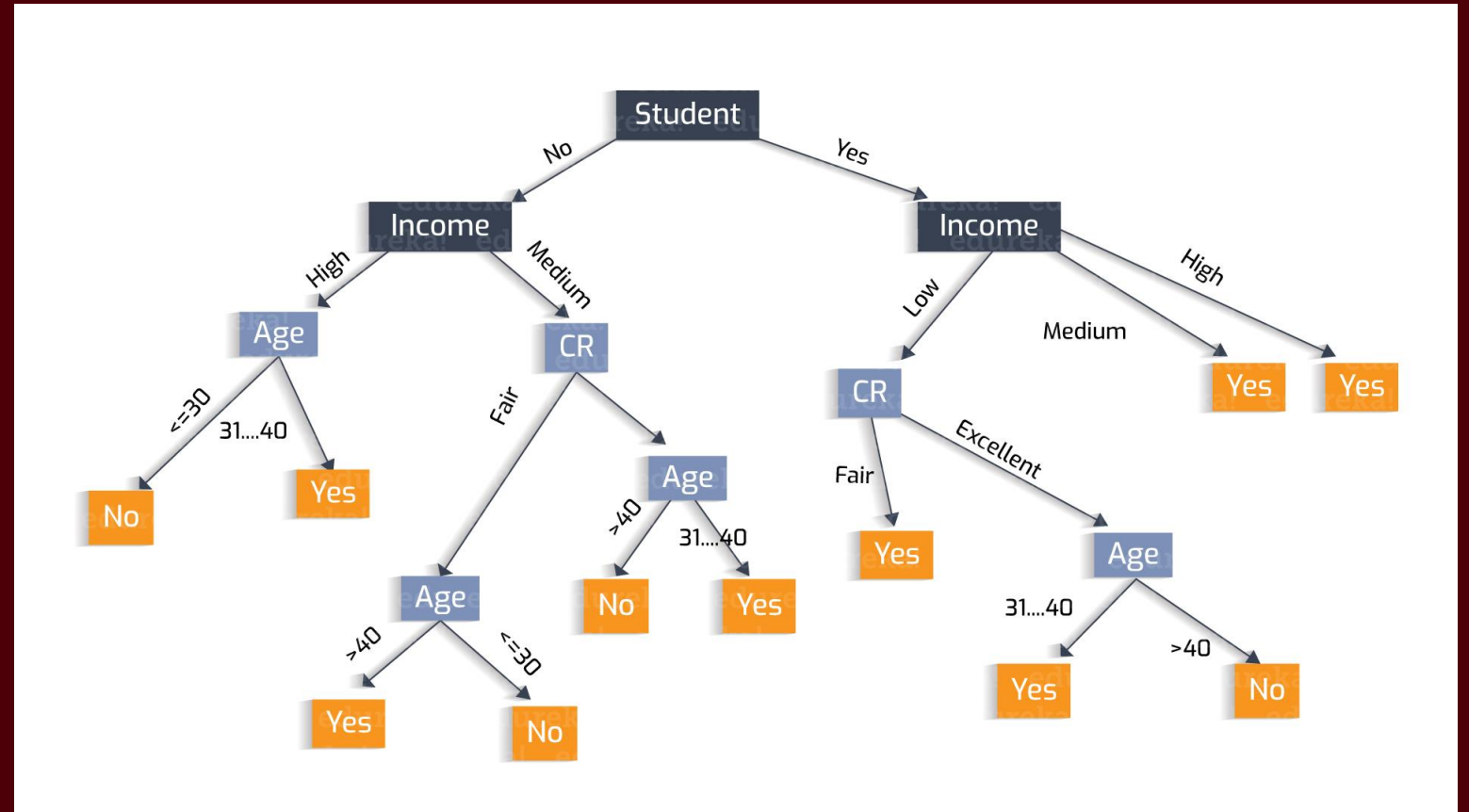
- Decisions Trees



<https://heartbeat.fritz.ai/understanding-the-mathematics-behind-decision-trees-22d86d55906> by Nikita Sharma

Why?

- Decisions Trees are great because they are often interpretable.
- However, they usually deal better with categorical data – not input pixel data.



<https://heartbeat.fritz.ai/understanding-the-mathematics-behind-decision-trees-22d86d55906> by Nikita Sharma

Regression vs Classification

Regression

- Labels are continuous variables – e.g. distance.
- Losses: Distance-based losses, e.g. sum of distances to true values.
- Evaluation: Mean distances, correlation coefficients, etc.

Classification

- Labels are discrete variables (1 out of K categories)
- Losses: Cross-entropy loss, margin losses, logistic regression (binary cross entropy)
- Evaluation: Classification accuracy, etc.

Supervised Learning - Classification

Training Data



cat



dog



cat

-
-
-



bear

Test Data



-
-
-



Supervised Learning - Classification

Training Data

$$x_1 = [\text{img}] \quad y_1 = [\text{cat}]$$

$$x_2 = [\text{img}] \quad y_2 = [\text{dog}]$$

$$x_3 = [\text{img}] \quad y_3 = [\text{cat}]$$

•
•
•

$$x_n = [\text{img}] \quad y_n = [\text{bear}]$$

Supervised Learning - Classification

Training Data

inputs	targets / labels / ground truth	predictions
$x_1 = [x_{11} \ x_{12} \ x_{13} \ x_{14}]$	$y_1 = 1$	$\hat{y}_1 = 1$
$x_2 = [x_{21} \ x_{22} \ x_{23} \ x_{24}]$	$y_2 = 2$	$\hat{y}_2 = 2$
$x_3 = [x_{31} \ x_{32} \ x_{33} \ x_{34}]$	$y_3 = 1$	$\hat{y}_3 = 2$
⋮		
⋮		
⋮		
$x_n = [x_{n1} \ x_{n2} \ x_{n3} \ x_{n4}]$	$y_n = 3$	$\hat{y}_n = 1$

We need to find a function that maps x and y for any of them.

$$\hat{y}_i = f(x_i; \theta)$$

How do we "learn" the parameters of this function?

We choose ones that makes the following quantity small:

$$\sum_{i=1}^n Cost(\hat{y}_i, y_i)$$

Stochastic Gradient Descent

- How to choose the right batch size B ?
- How to choose the right learning rate λ ?
- How to choose the right loss function, e.g. is least squares good enough?
- How to choose the right function/classifier, e.g. linear, quadratic, neural network with 1 layer, 2 layers, etc?

Linear Regression

Example: Hollywood movie data

input variables x

output variables y

training
data

	production costs	promotional costs	genre of the movie	box office first week	total book sales	total revenue USA	total revenue international
	$x_1^{(1)}$	$x_2^{(1)}$	$x_3^{(1)}$	$x_4^{(1)}$	$x_5^{(1)}$	$y_1^{(1)}$	$y_2^{(1)}$
	$x_1^{(2)}$	$x_2^{(2)}$	$x_3^{(2)}$	$x_4^{(2)}$	$x_5^{(2)}$	$y_1^{(2)}$	$y_2^{(2)}$
	$x_1^{(3)}$	$x_2^{(3)}$	$x_3^{(3)}$	$x_4^{(3)}$	$x_5^{(3)}$	$y_1^{(3)}$	$y_2^{(3)}$
test data	$x_1^{(4)}$	$x_2^{(4)}$	$x_3^{(4)}$	$x_4^{(4)}$	$x_5^{(4)}$	$y_1^{(4)}$	$y_2^{(4)}$
	$x_1^{(5)}$	$x_2^{(5)}$	$x_3^{(5)}$	$x_4^{(5)}$	$x_5^{(5)}$	$y_1^{(5)}$	$y_2^{(5)}$

Training, Validation (Dev), Test Sets



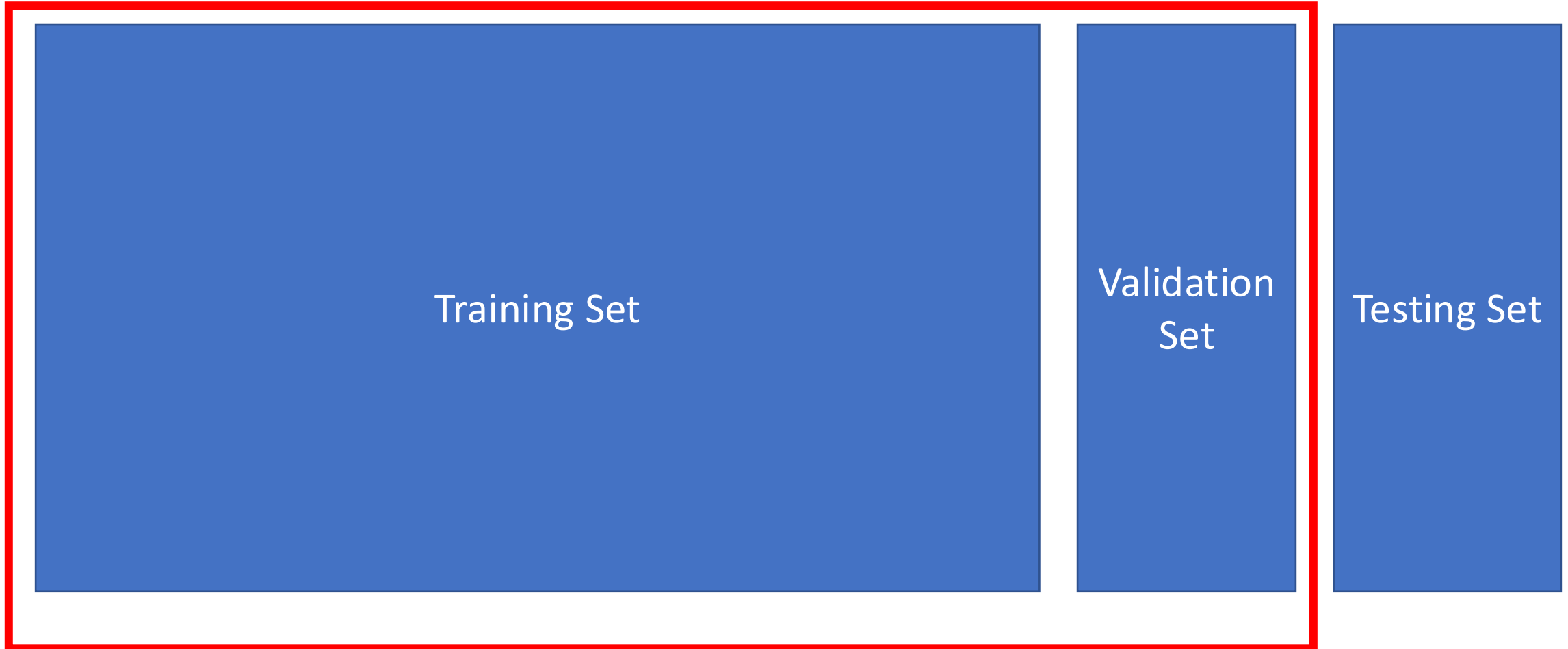
The diagram consists of three blue rectangular blocks arranged horizontally. The first block on the left is significantly larger than the other two, which are of equal size and positioned to its right. Each block contains white text centered within it.

Training Set

Validation
Set

Testing Set

Training, Validation (Dev), Test Sets



Used during development

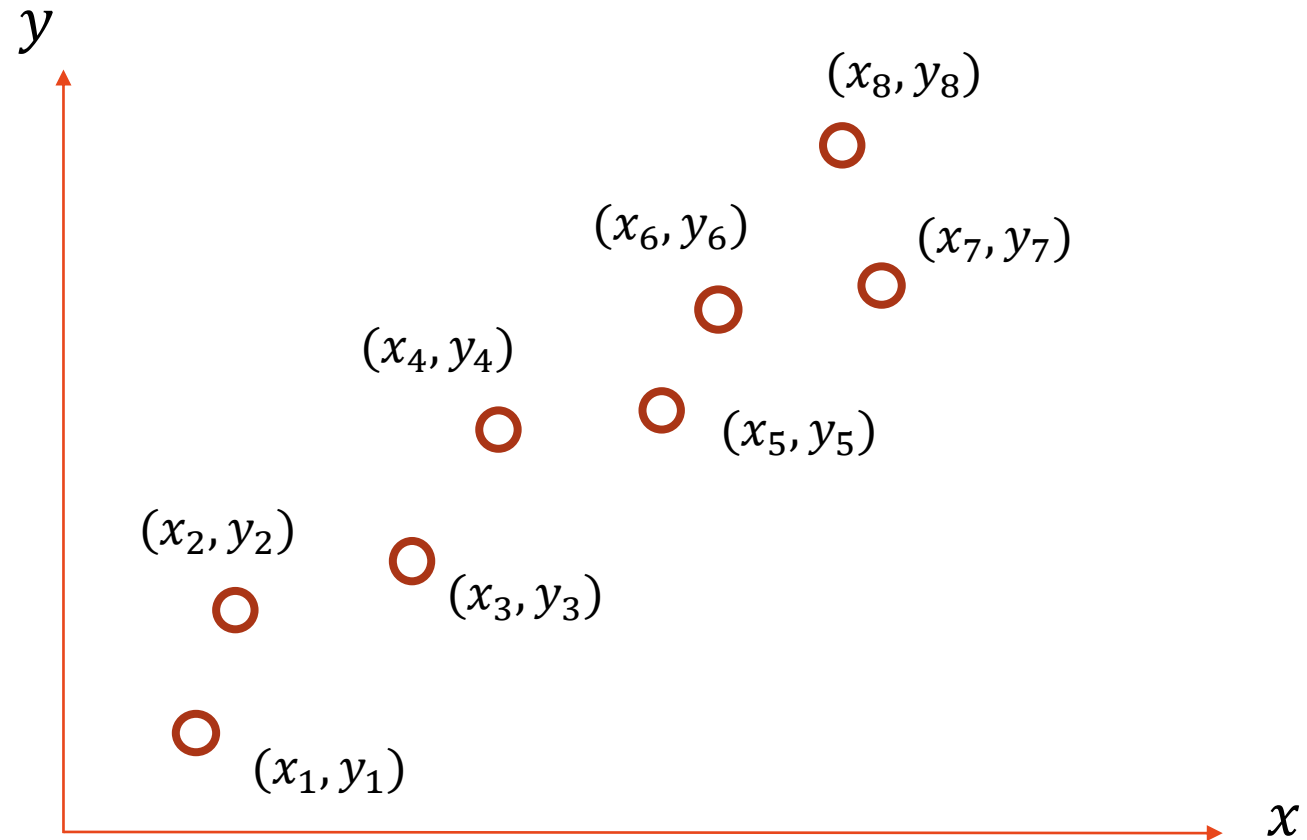
Training, Validation (Dev), Test Sets



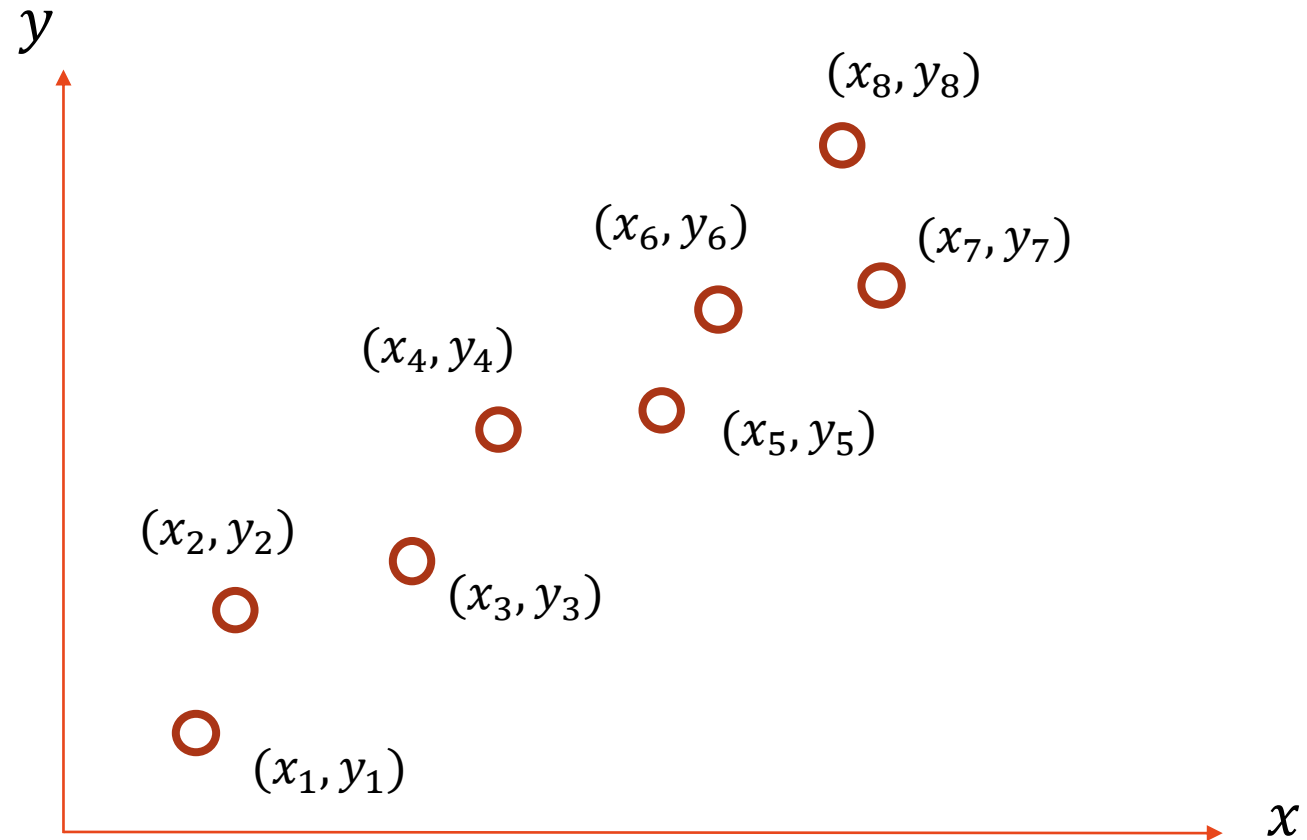
Only to be used for evaluating the model at the very end of development and any changes to the model after running it on the test set, could be influenced by what you saw happened on the test set, which would invalidate any future evaluation.

HOW TO PICK THE RIGHT MODEL?

Linear Regression – 1 output, 1 input

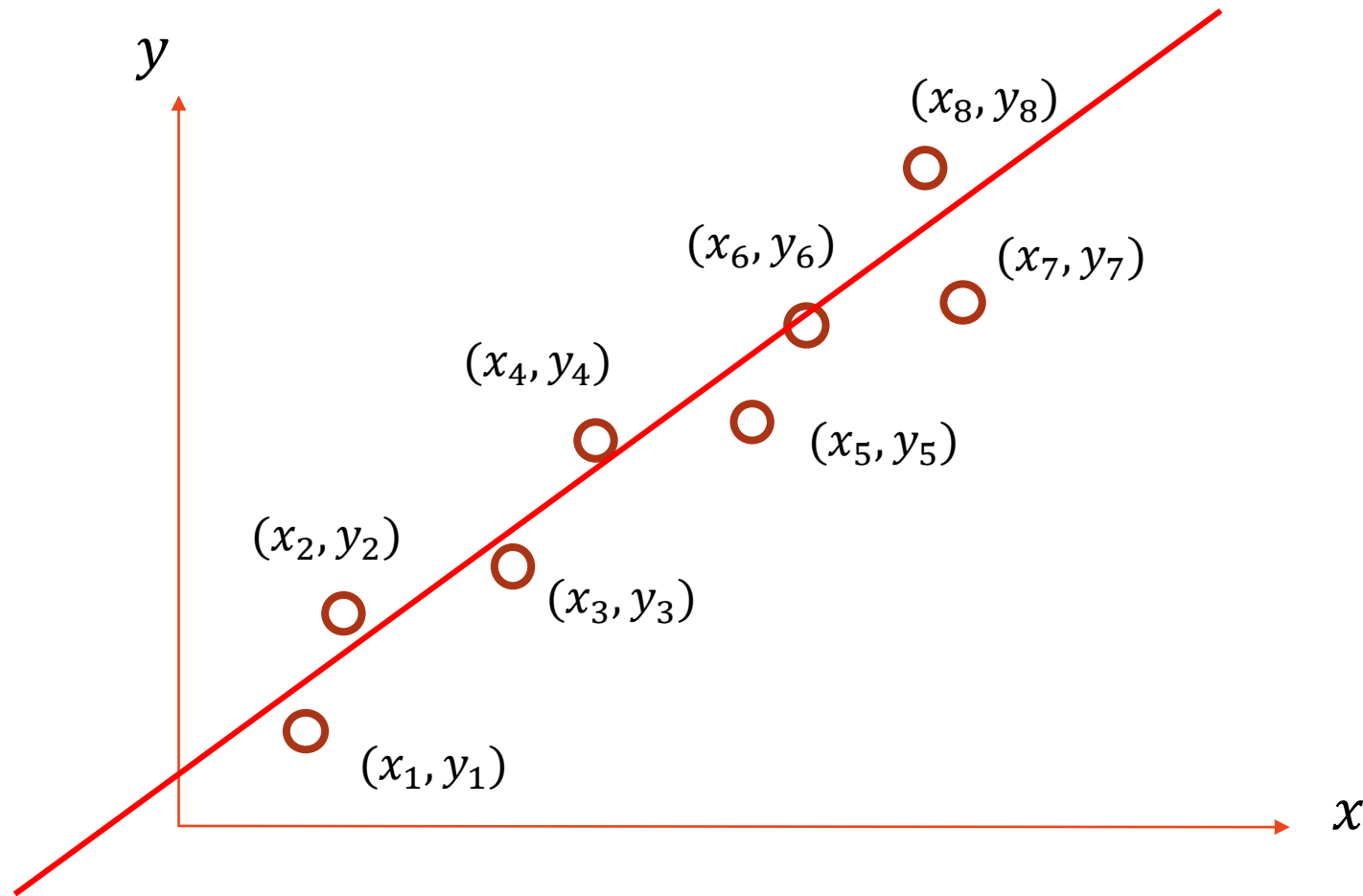


Linear Regression – 1 output, 1 input



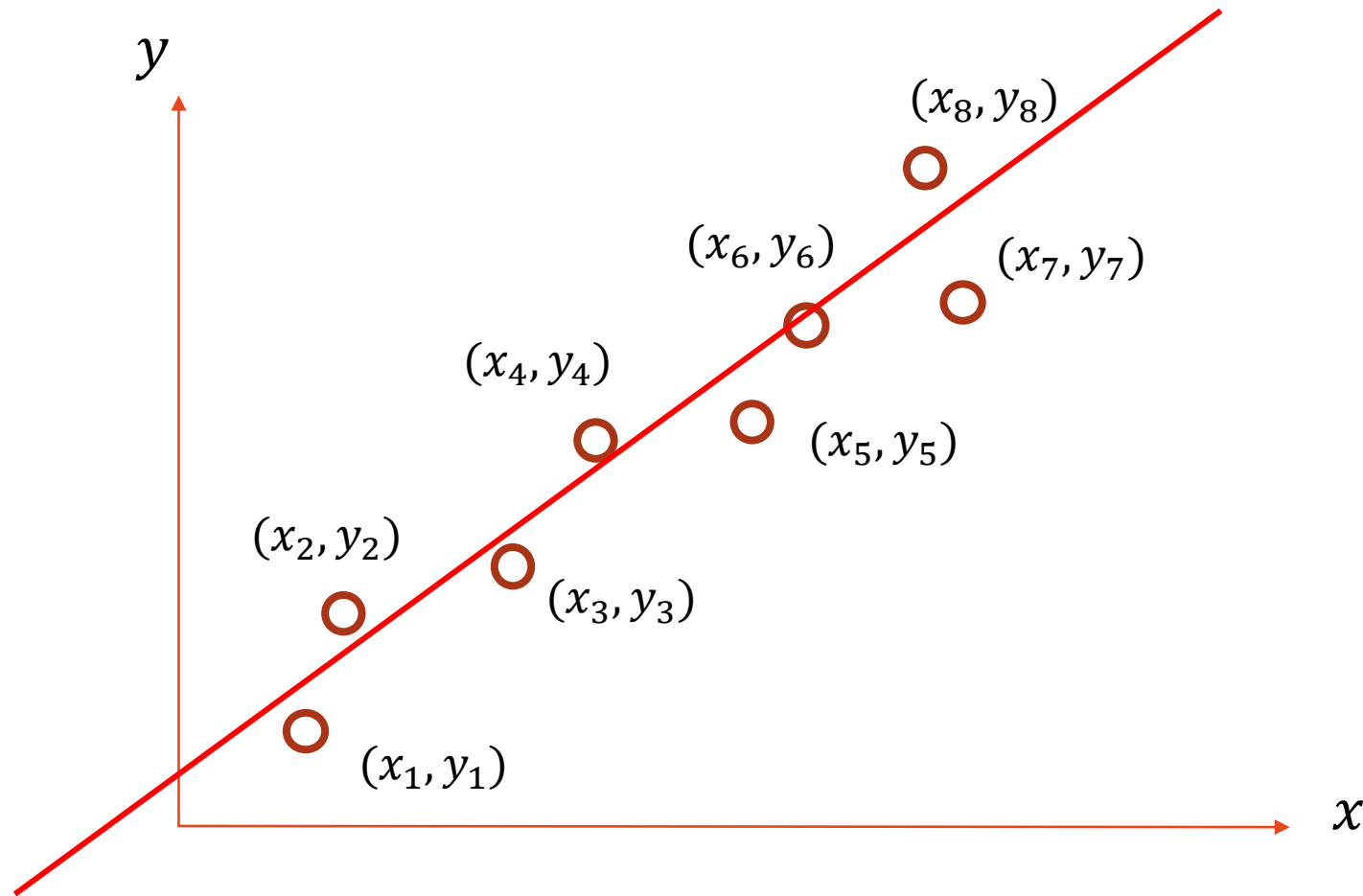
Model: $\hat{y} = wx + b$

Linear Regression – 1 output, 1 input



Model: $\hat{y} = wx + b$

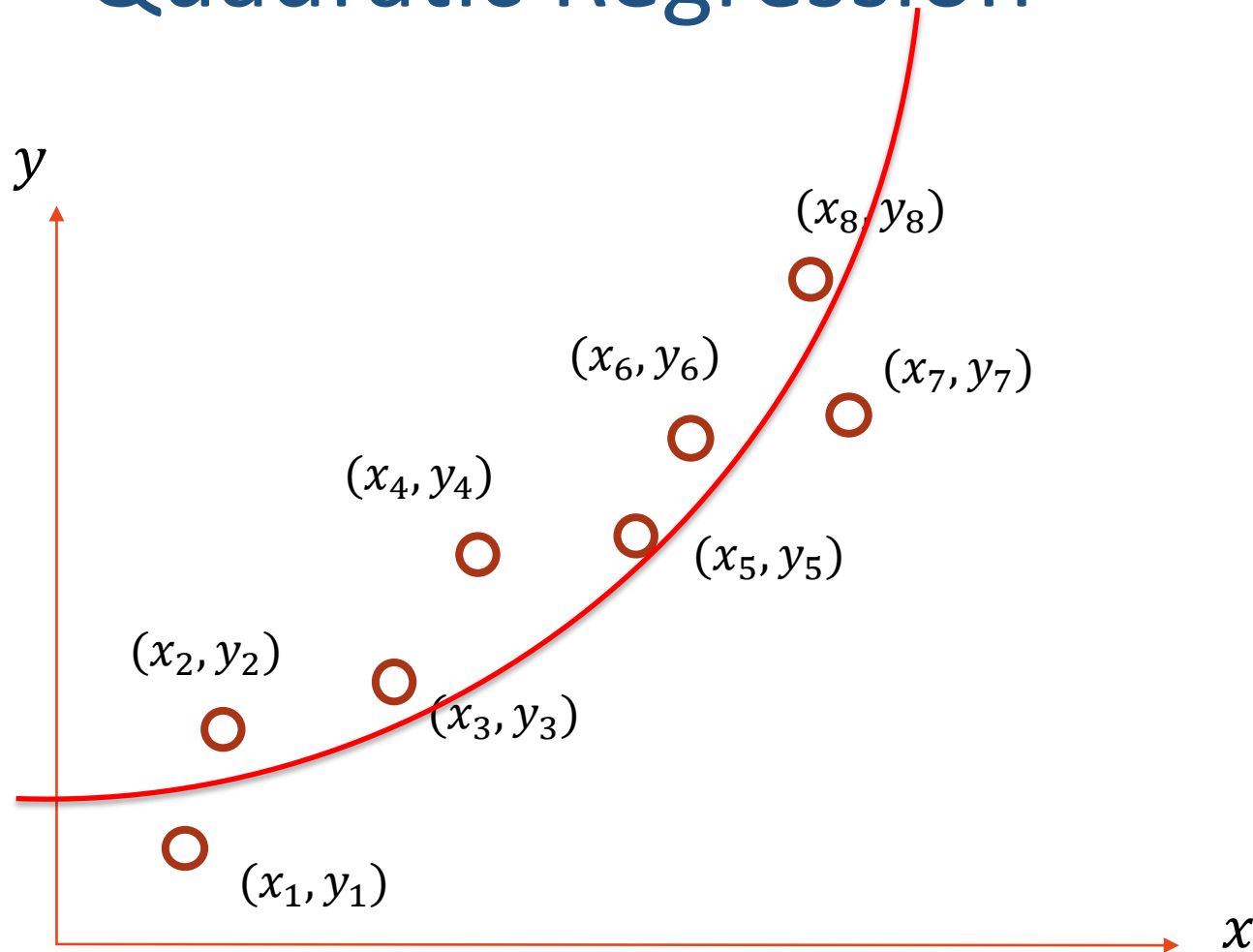
Linear Regression – 1 output, 1 input



Model: $\hat{y} = wx + b$

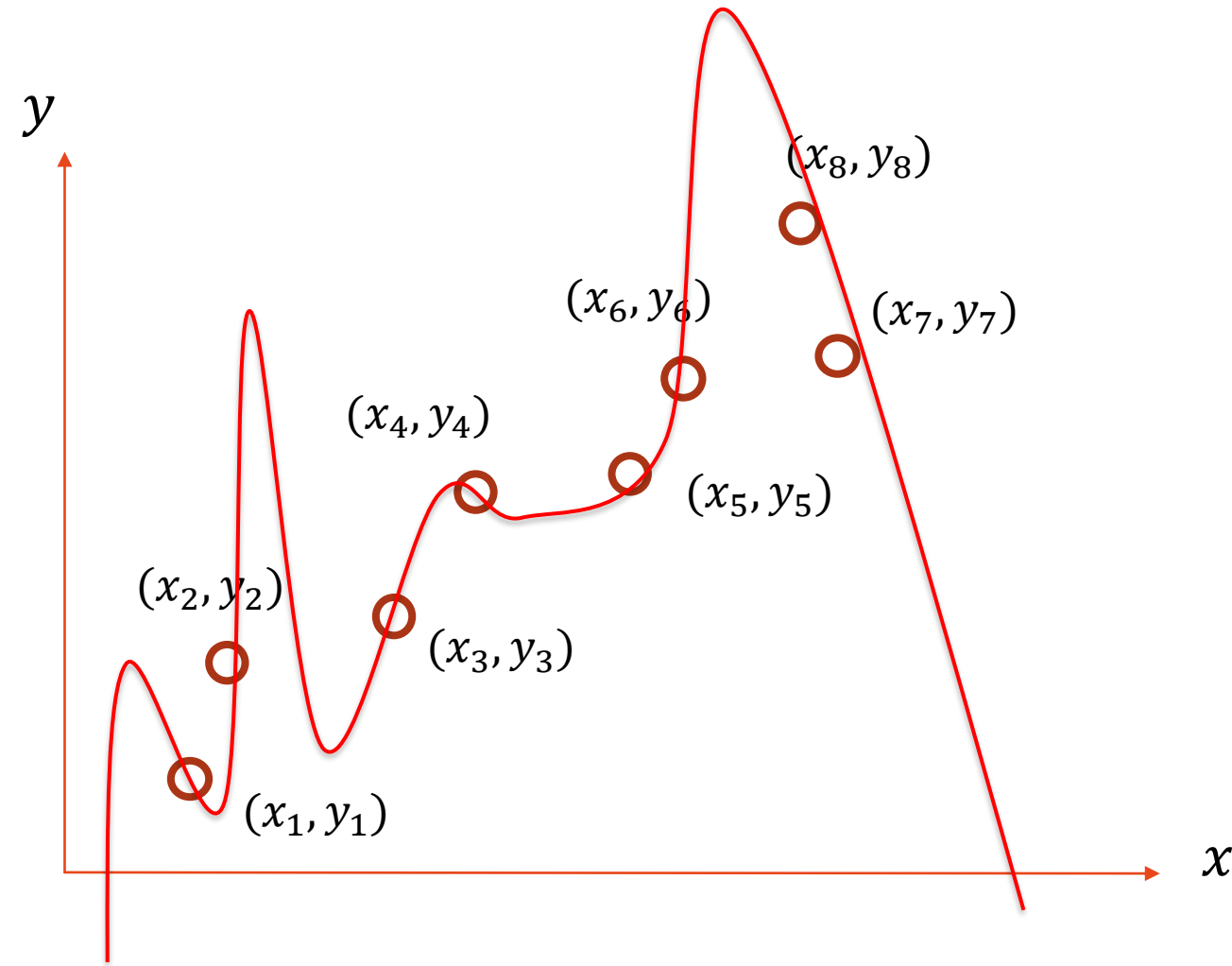
Loss: $L(w, b) = \sum_{i=1}^{i=8} (\hat{y}_i - y_i)^2$

Quadratic Regression



Model: $\hat{y} = w_1x^2 + w_2x + b$ Loss: $L(w, b) = \sum_{i=1}^{i=8} (\hat{y}_i - y_i)^2$

n-polynomial Regression

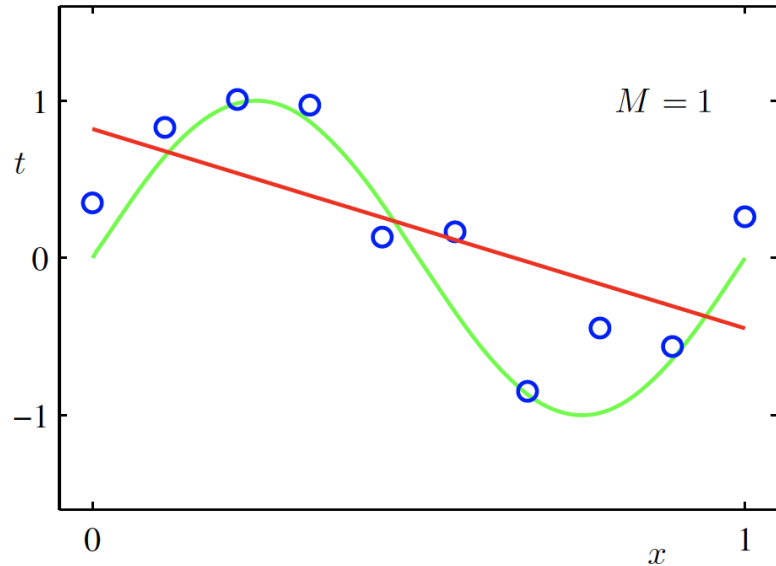


Model: $\hat{y} = w_n x^n + \dots + w_1 x + b$

Loss: $L(w, b) = \sum_{i=1}^{i=8} (\hat{y}_i - y_i)^2$

Overfitting

f is linear

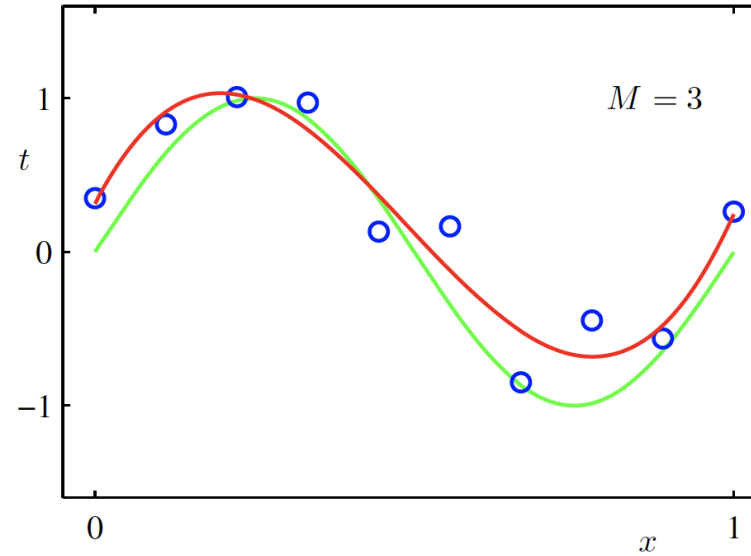


$Loss(w)$ is high

Underfitting

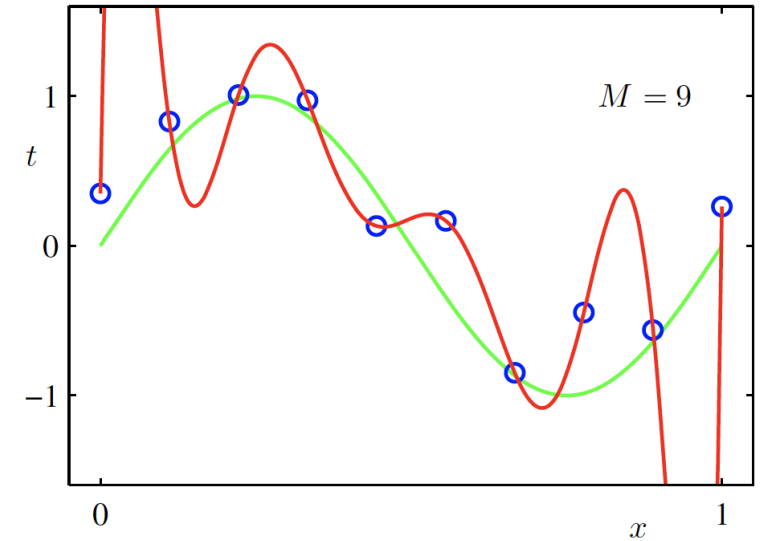
High Bias

f is cubic



$Loss(w)$ is low

f is a polynomial of degree 9

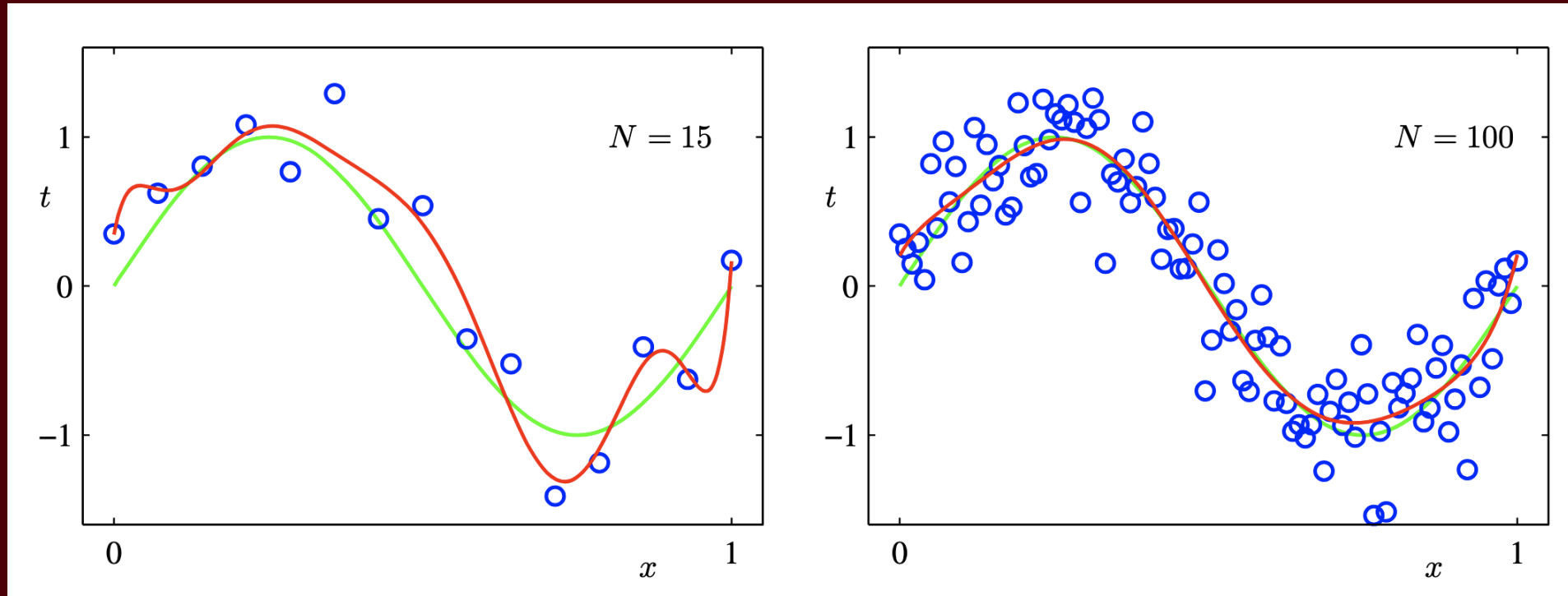


$Loss(w)$ is zero!

Overfitting

High Variance

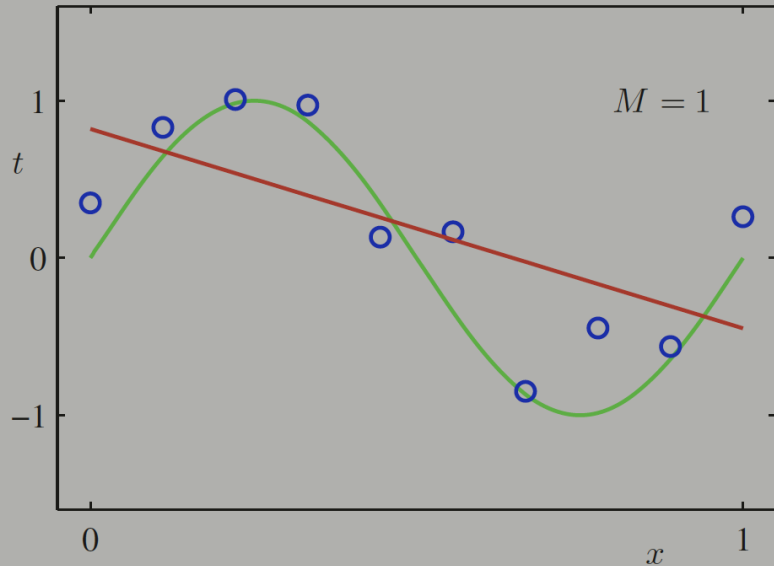
What to do about overfitting?



Add more training data

Overfitting

f is linear

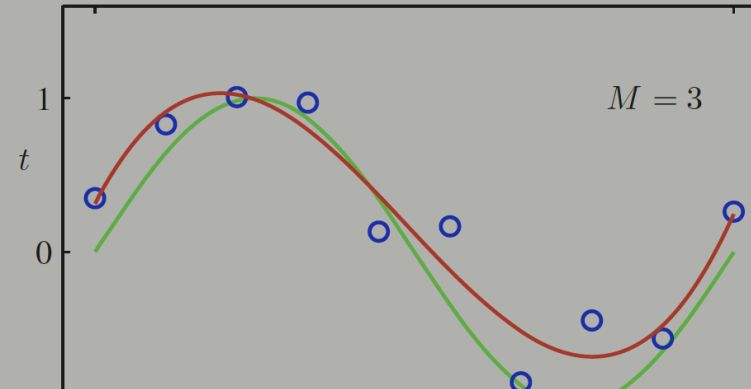


$Loss(w)$ is high

Underfitting

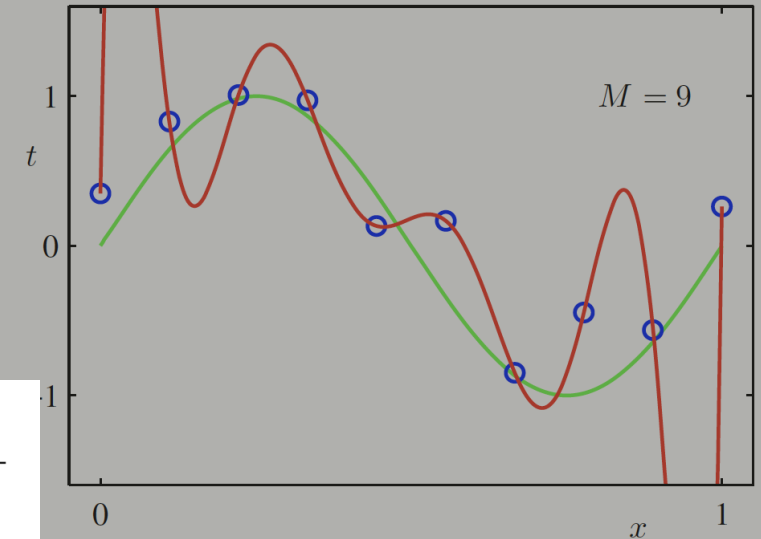
High Bias

f is cubic



	$M = 1$	$M = 6$	$M = 9$
w_0^*	0.82	0.31	0.35
w_1^*	-1.27	7.99	232.37
w_2^*		-25.43	-5321.83
w_3^*		17.37	48568.31
w_4^*			-231639.30
w_5^*			640042.26
w_6^*			-1061800.52
w_7^*			1042400.18
w_8^*			-557682.99
w_9^*			125201.43

f is a polynomial of degree 9



$Loss(w)$ is zero!

Overfitting

High Variance

(mini-batch) Stochastic Gradient Descent (SGD)

$\lambda = 0.01$

Initialize w and b randomly

$$l(w, b) = \sum_{i \in B} Cost(w, b)$$

for $e = 0, \text{num_epochs}$ **do**

for $b = 0, \text{num_batches}$ **do**

 Compute: $dl(w, b)/dw$ and $dl(w, b)/db$

 Update w : $w = w - \lambda dl(w, b)/dw$

 Update b : $b = b - \lambda dl(w, b)/db$

 Print: $l(w, b)$ // Useful to see if this is becoming smaller or not.

end

end

Regularization

- Large weights lead to large variance. i.e. model fits to the training data too strongly.
- Solution: Minimize the loss but also try to keep the weight values small by doing the following:

$$\text{minimize} \quad L(w, b) + \alpha \sum_i |w_i|^2$$

Regularization

- Large weights lead to large variance. i.e. model fits to the training data too strongly.
- Solution: Minimize the loss but also try to keep the weight values small by doing the following:

minimize $L(w, b) + \alpha \sum_i |w_i|^2$

Regularizer term
e.g. L2- regularizer

SGD with Regularization (L-2)

$$\lambda = 0.01$$

$$l(w, b) = l(w, b) + \alpha \sum_i |w_i|^2$$

Initialize w and b randomly

for $e = 0, \text{num_epochs}$ **do**

for $b = 0, \text{num_batches}$ **do**

Compute: $dl(w, b)/dw$ and $dl(w, b)/db$

Update w : $w = w - \lambda dl(w, b)/dw - \lambda \alpha w$

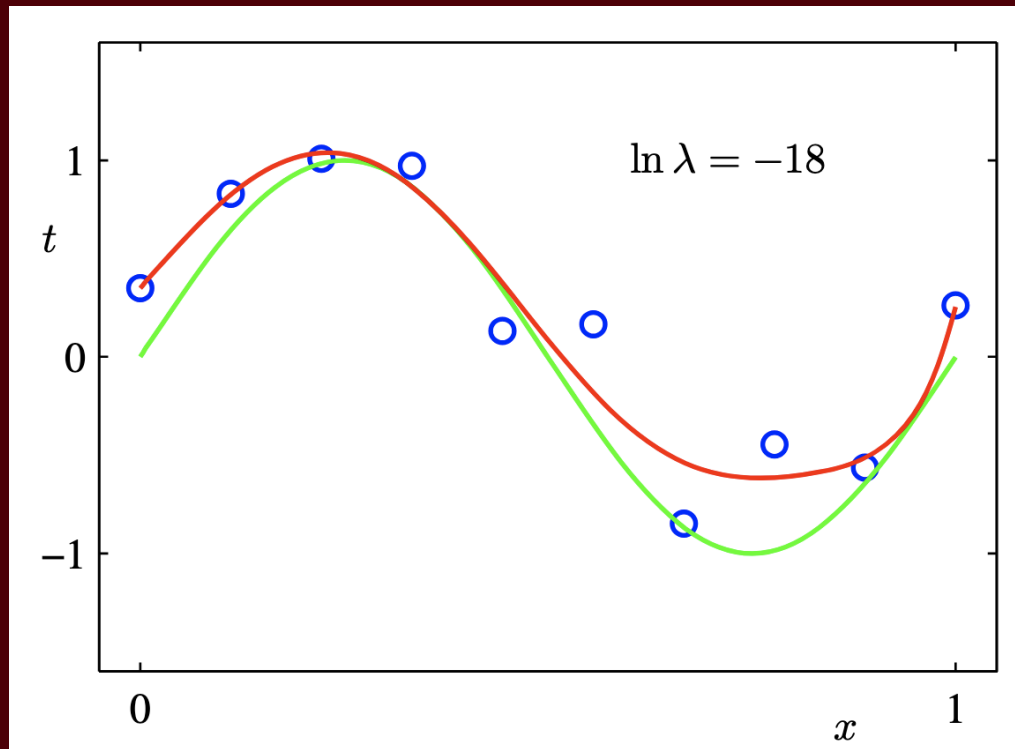
Update b : $b = b - \lambda dl(w, b)/db - \lambda \alpha w$

Print: $l(w, b)$ // Useful to see if this is becoming smaller or not.

end

end

Regularization: As Shown in Bishop's ML Book



	$\ln \lambda = -18$
w_0^*	0.35
w_1^*	4.74
w_2^*	-0.77
w_3^*	-31.97
w_4^*	-3.89
w_5^*	55.28
w_6^*	41.32
w_7^*	-45.95
w_8^*	-91.53
w_9^*	72.68

$$\tilde{E}(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^N \{y(x_n, \mathbf{w}) - t_n\}^2 + \frac{\lambda}{2} \|\mathbf{w}\|^2$$

Revisiting Another Problem with SGD

$$\lambda = 0.01$$

$$l(w, b) = l(w, b) + \alpha \sum_i |w_i|^2$$

Initialize w and b randomly

for $e = 0, \text{num_epochs}$ **do**

for $b = 0, \text{num_batches}$ **do**

Compute: $dl(w, b)/dw$ and $dl(w, b)/db$

Update w : $w = w - \lambda dl(w, b)/dw - \lambda \alpha w$

Update b : $b = b - \lambda dl(w, b)/db - \lambda \alpha w$

Print: $l(w, b)$ // Useful to see if this is becoming smaller or not.

end

end

These are only approximations to the true gradient with respect to $L(w, b)$

Revisiting Another Problem with SGD

$$\lambda = 0.01$$

$$l(w, b) = l(w, b) + \alpha \sum_i |w_i|^2$$

Initialize w and b randomly

for $e = 0, \text{num_epochs}$ **do**

for $b = 0, \text{num_batches}$ **do**

Compute: $dl(w, b)/dw$ and $dl(w, b)/db$

Update w : $w = w - \lambda dl(w, b)/dw - \lambda \alpha w$

Update b : $b = b - \lambda dl(w, b)/db - \lambda \alpha w$

Print: $l(w, b)$ // Useful to see if this is becoming smaller or not.

end

end

This could lead to “un-learning” what has been learned in some previous steps of training.

Solution: Momentum Updates

$$\lambda = 0.01$$

$$l(w, b) = l(w, b) + \alpha \sum_i |w_i|^2$$

Initialize w and b randomly

for $e = 0, \text{num_epochs}$ **do**

for $b = 0, \text{num_batches}$ **do**

Compute: $dl(w, b)/dw$ and $dl(w, b)/db$

Update w : $w = w - \lambda dl(w, b)/dw - \lambda \alpha w$

Update b : $b = b - \lambda dl(w, b)/db - \lambda \alpha w$

Print: $l(w, b)$ // Useful to see if this is becoming smaller or not.

end

end

Keep track of previous gradients in an accumulator variable! and use a weighted average with current gradient.

Solution: Momentum Updates

$$\lambda = 0.01 \quad \tau = 0.9$$

Initialize w and b randomly

$$l(w, b) = l(w, b) + \alpha \sum_i |w_i|^2$$

global v

for $e = 0, \text{num_epochs}$ **do**

for $b = 0, \text{num_batches}$ **do**

Compute: $dl(w, b)/dw$

Compute: $v = \tau v + dl(w, b)/dw + \alpha w$

Update w : $w = w - \lambda v$

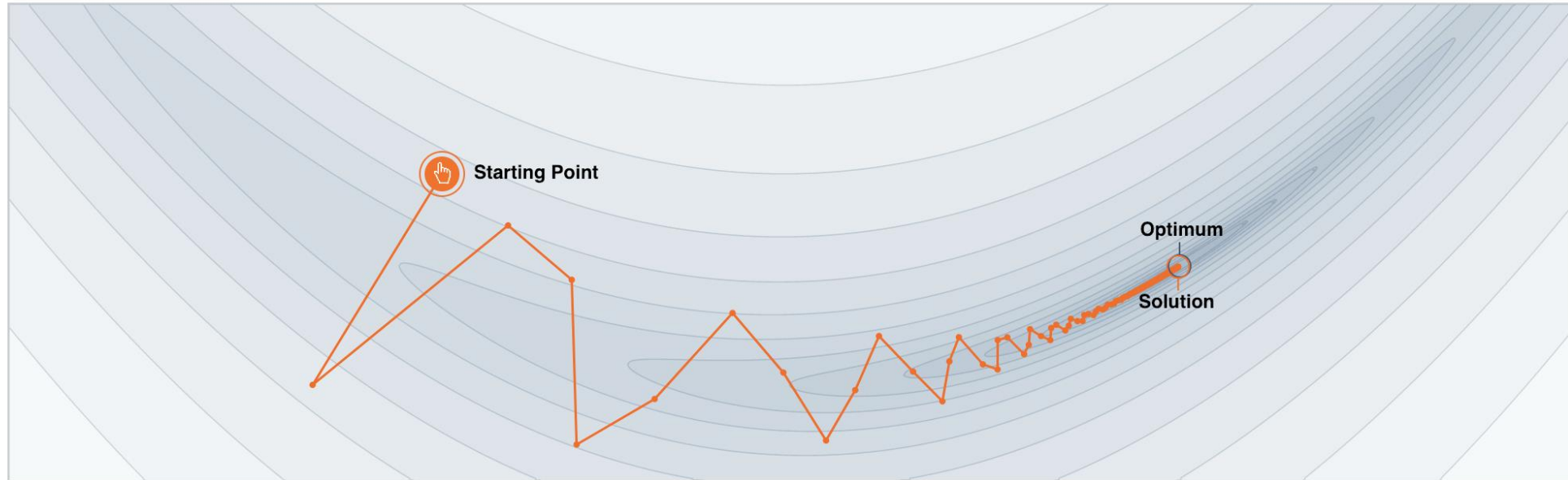
Print: $l(w, b)$ // Useful to see if this is becoming smaller or not.

end

end

Keep track of previous gradients in an accumulator variable! and use a weighted average with current gradient.

More on Momentum



Step-size $\alpha = 0.0050$



Momentum $\beta = 0.77$



We often think of Momentum as a means of dampening oscillations and speeding up the iterations, leading to faster convergence. But it has other interesting behavior. It allows a larger range of step-sizes to be used, and creates its own oscillations. What is going on?

<https://distill.pub/2017/momentum/>

Questions?