Deep Learning for Vision & Language

Machine Learning II: SGD, Generalization, Regularization

RICE UNIVERSITY

About the class

- COMP 646: Deep Learning for Vision and Language
- Instructor: **Vicente** Ordóñez (Vicente Ordóñez Román)
- Website: https://www.cs.rice.edu/~vo9/deep-vislang
- Location: Herzstein Hall 210
- Times: Tuesdays and Thursdays
  from 4pm to 5:15pm
- Office Hours: Tuesdays 10am to 11am (DH3098)
- Teaching Assistants: **Arnold, Jefferson, Sangwon, Gaotian**
- Discussion Forum: Piazza (Sign-up Link on Rice Canvas and Class Website)

Teaching Assistants (TAs)

Jefferson Hernandez

Mondays 2:30pm
DH 3036

Sangwon Seo

Wednesdays 10am
DH 3002

Gaotian Wang

Wednesdays 3pm
DH 3036

Arnold Kazadi

Thursdays 11am
DH 3036

Assignment 1

- Assignment 1 is released and is available on the class website.

Grading for this class: COMP 646

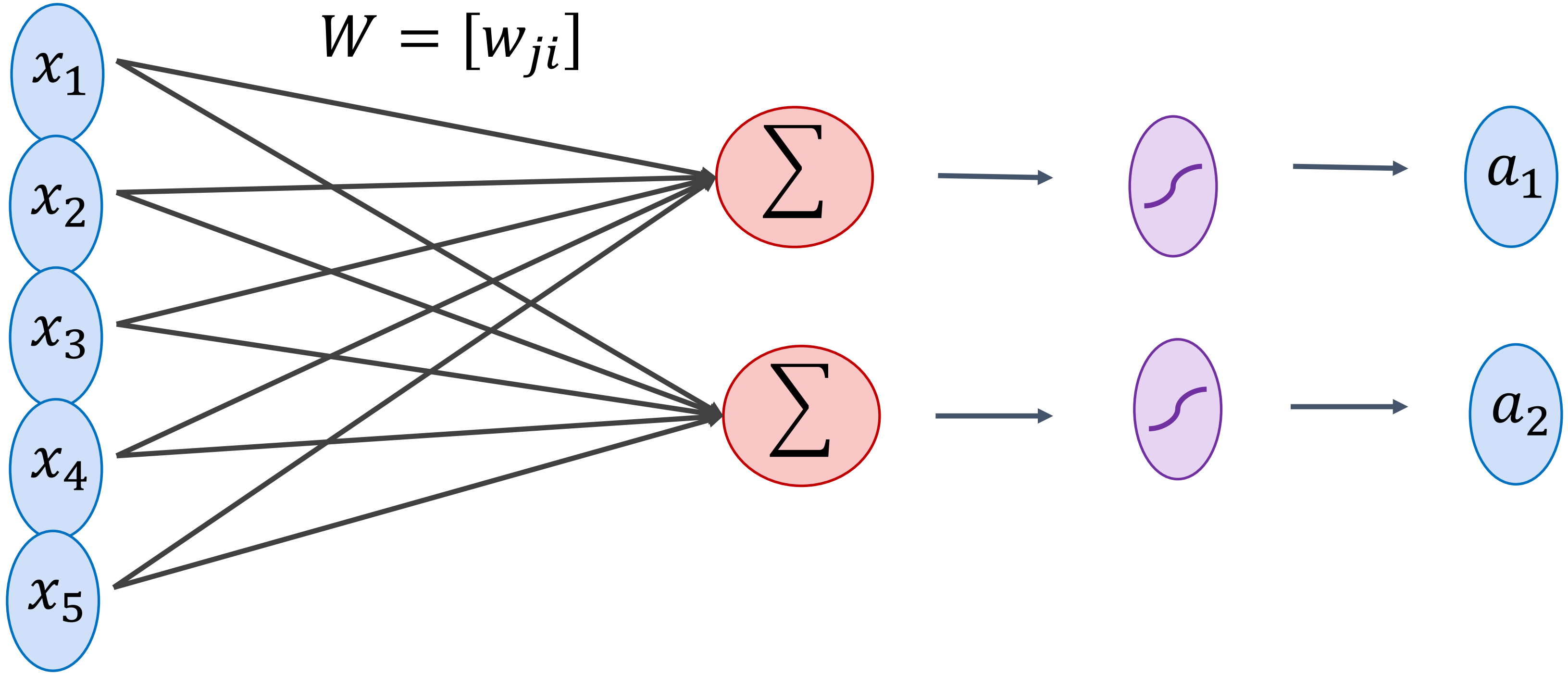
- Assignments: 30pts (3 assignments: 10pts + 10pts + 10pts)
- Class Project: 60pts
- Quiz: 10pts

Total: 100pts

- Grade cutoffs: no stricter than the following:
  A [between 90% and 100%], B [between 80% and 90%),
  C [between 70% and 80%), D [between 55% and 70%),
  F [less than 55%)

Neural Network with One Layer

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



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

Gradient Descent

$\lambda = 0.01$

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

expensive

Initialize w and b randomly

for e = 0, 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)

$\lambda = 0.01$

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

Initialize w and b randomly

for e = 0, num_epochs **do**

for b = 0, 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.

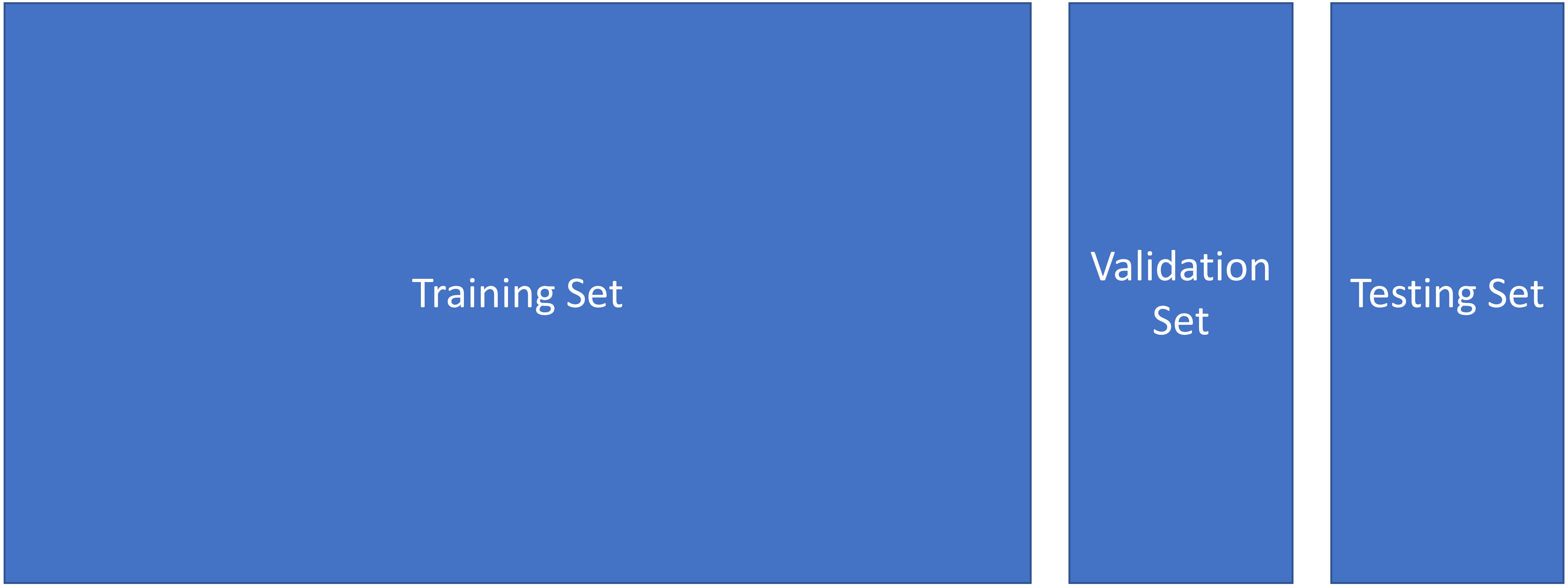
end

end

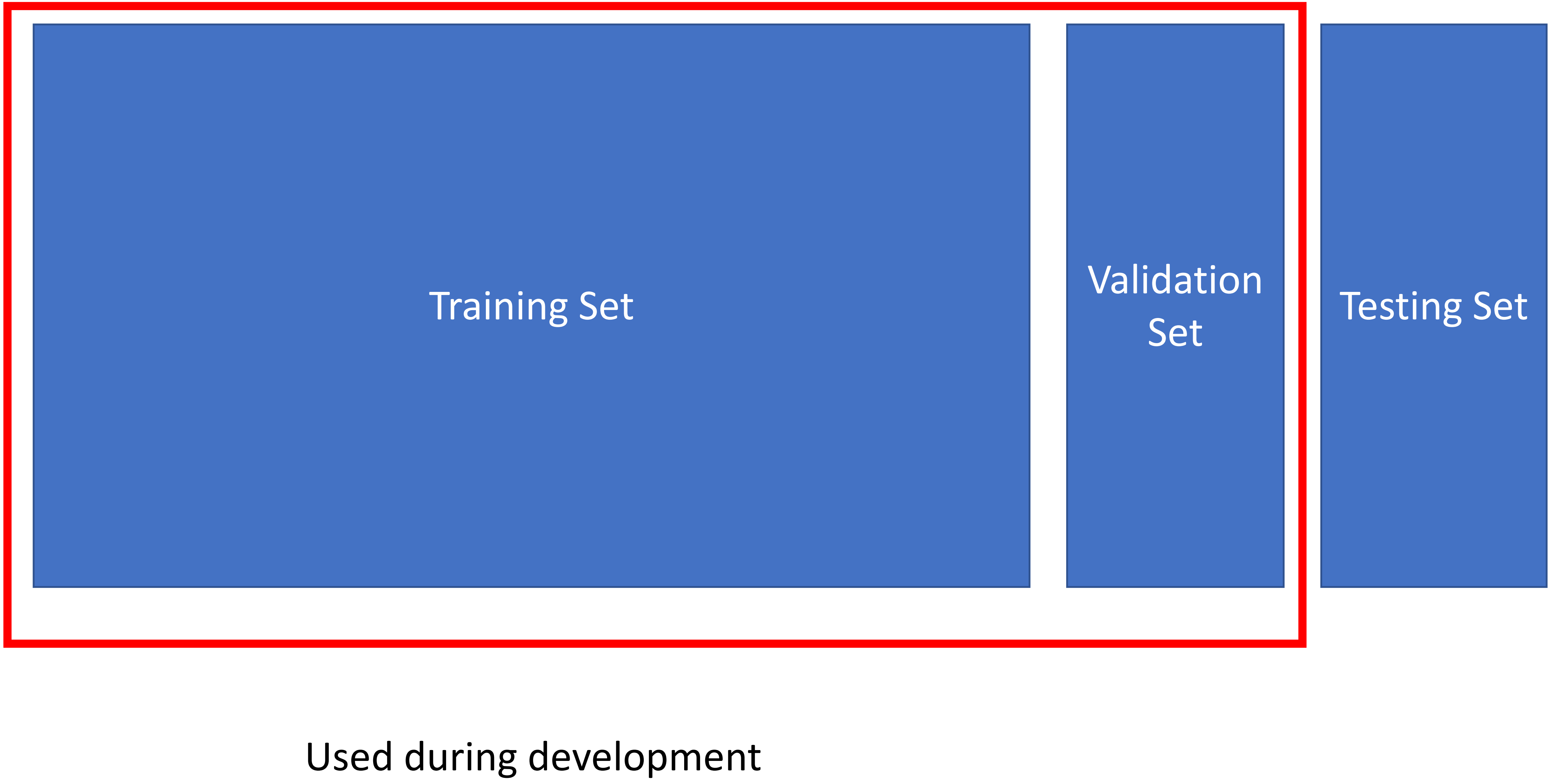
Stochastic Gradient Descent

- How to choose the right batch size B?
- How to choose the right learning rate lambda?
- 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?

Training, Validation (Dev), Test Sets



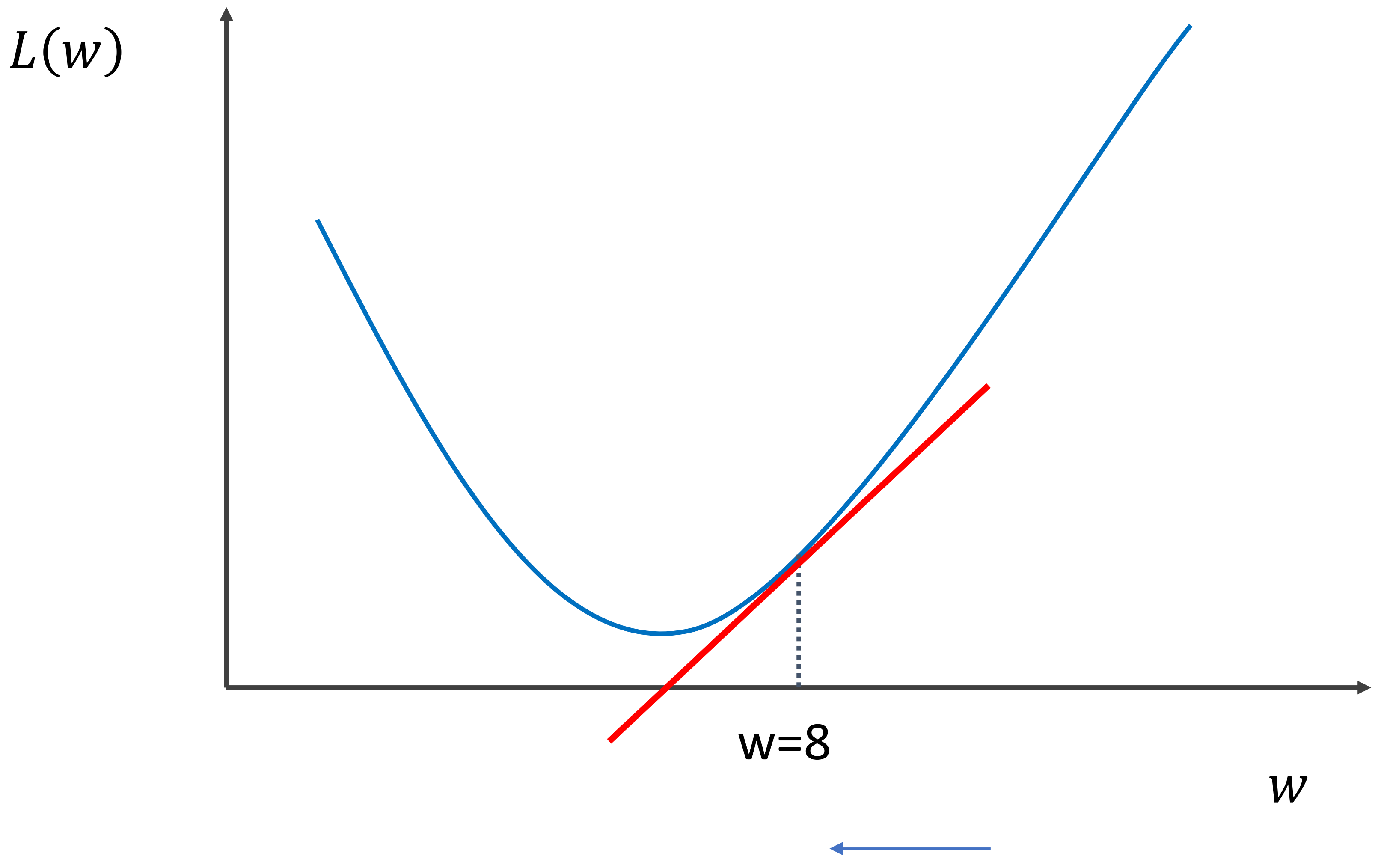
Training, Validation (Dev), Test Sets



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.

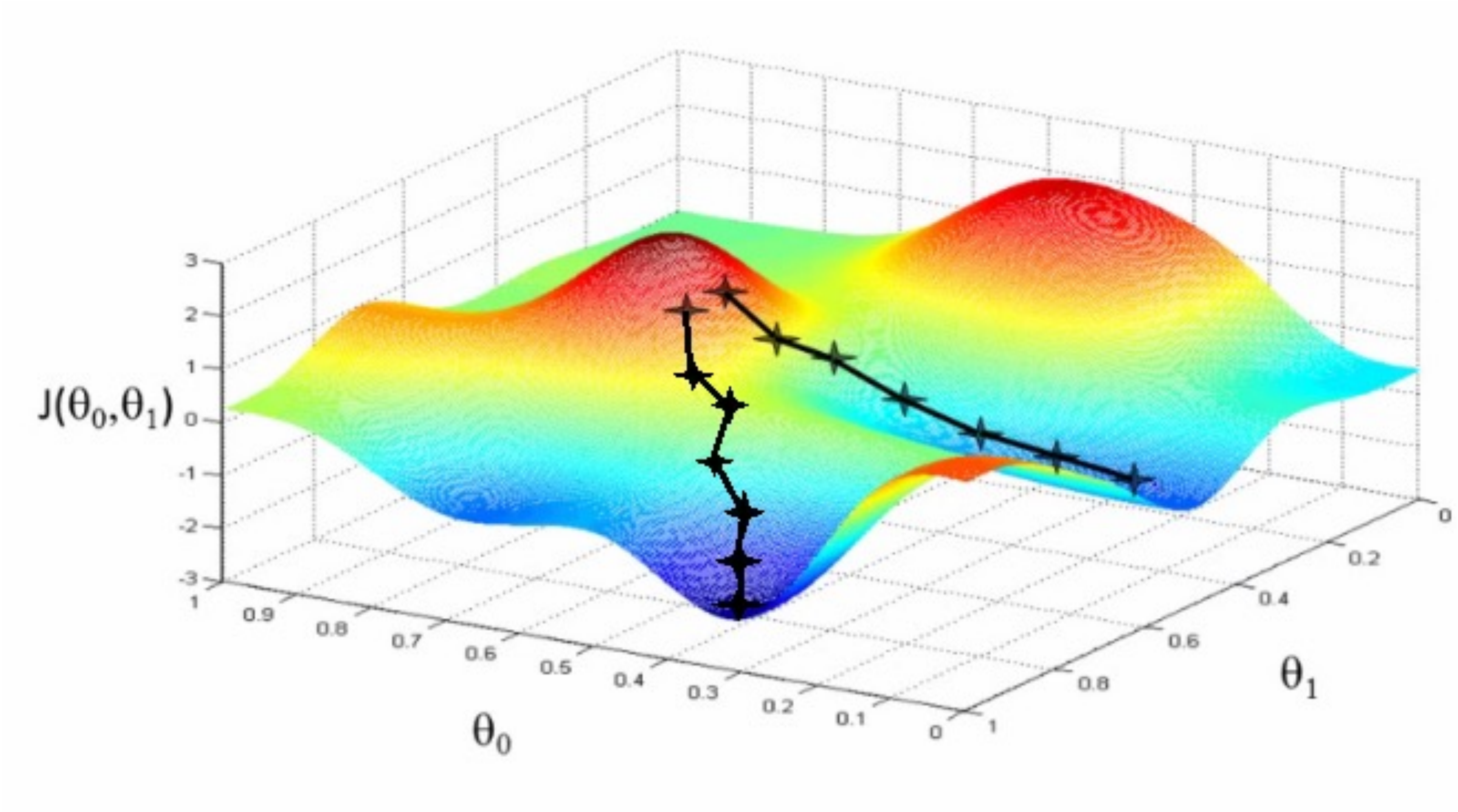
Gradient Descent



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

3. Recompute w as:

w = w – lambda * (dL / dw)



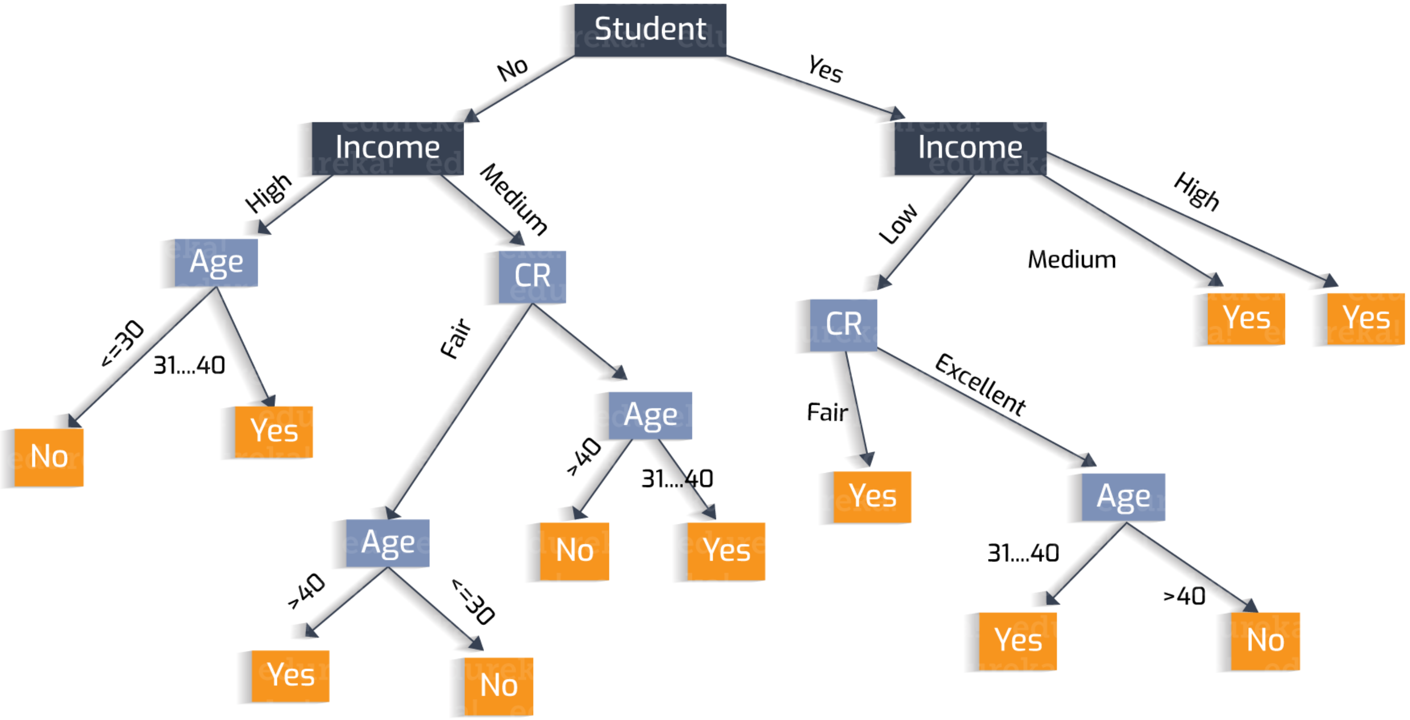
Source: Andrew Ng

In this class we will mostly rely on...

- K-nearest neighbors
- $\boxed{\text{Linear classifiers}}$
- Naïve Bayes classifiers
- Decision Trees
- Random Forests
- Boosted Decision Trees
- $\boxed{\text{Neural Networks}}$

Why?

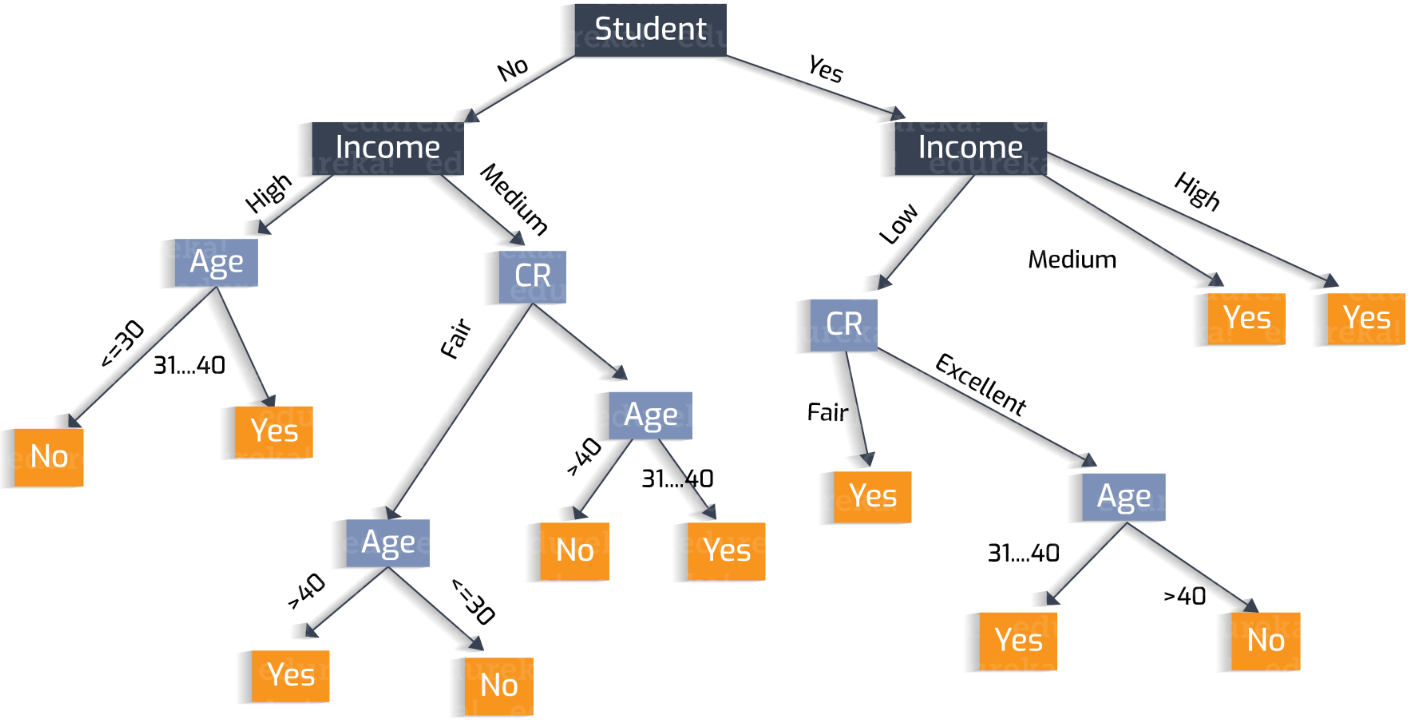
- Decisions Trees



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

Why?

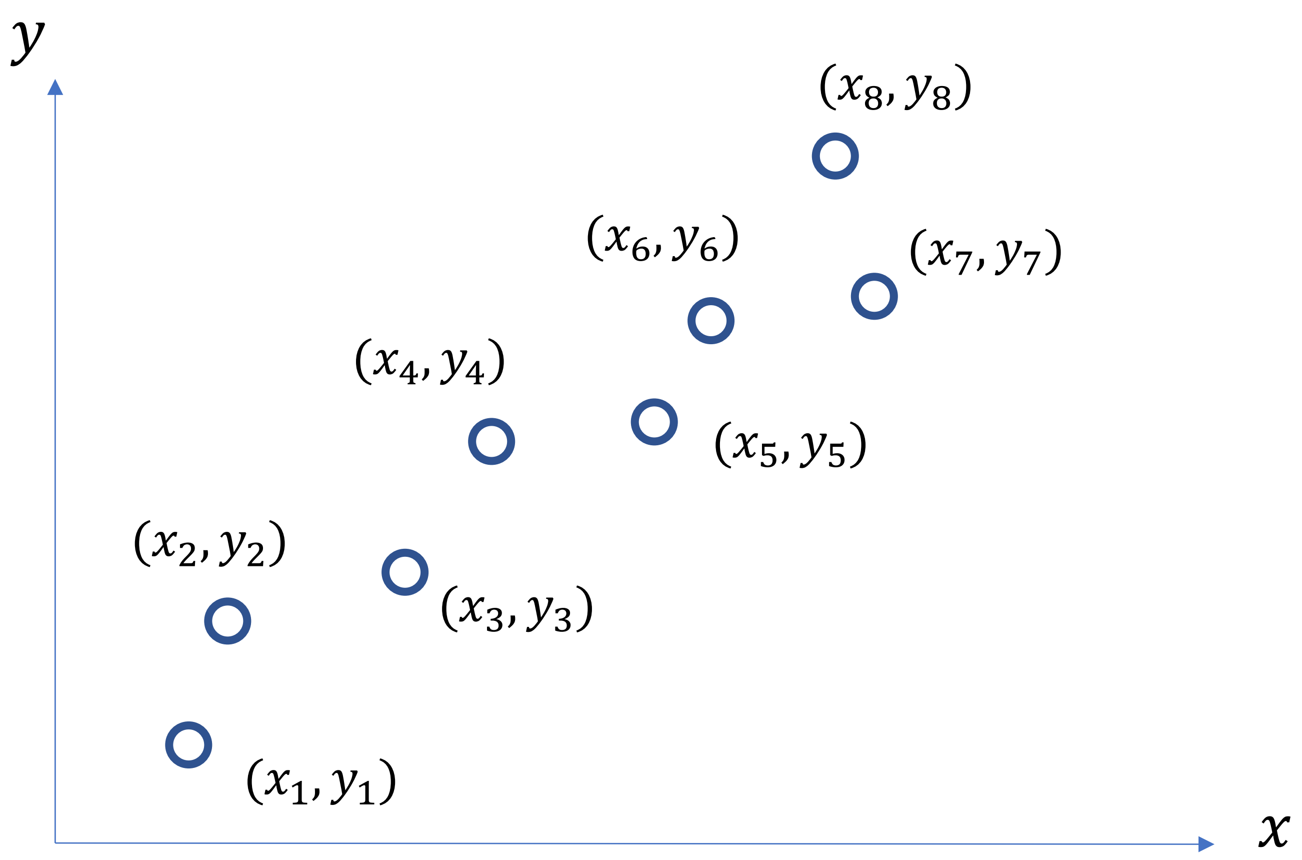
- Decisions Trees are great because they are often interpretable.
- However, they usually deal better with categorical data – not input pixel data.



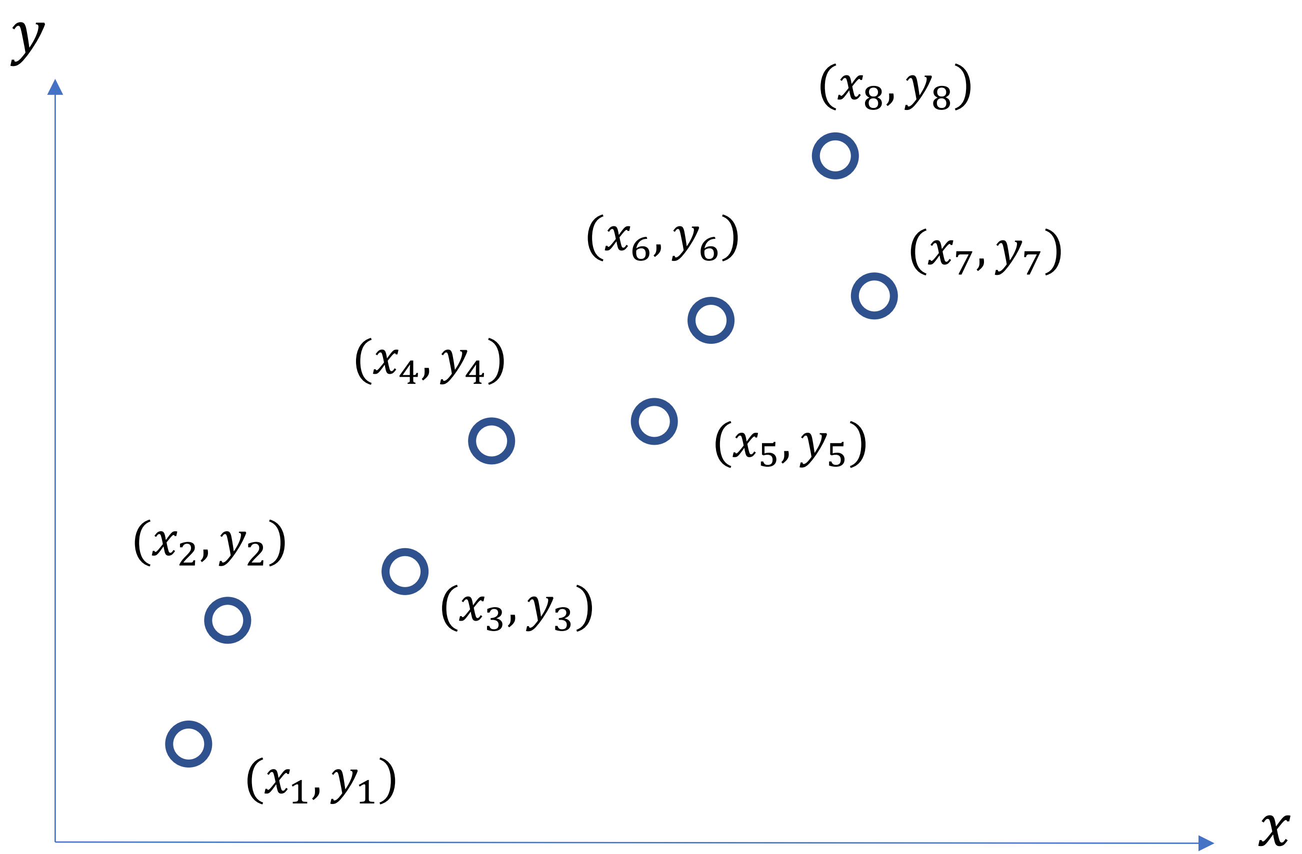
https://heartbeat.fritz.ai/understanding-the-mathematics-behind-decision-trees-22d86d55906 by Nikita Sharma

How to pick the right model?

Linear Regression – 1 output, 1 input

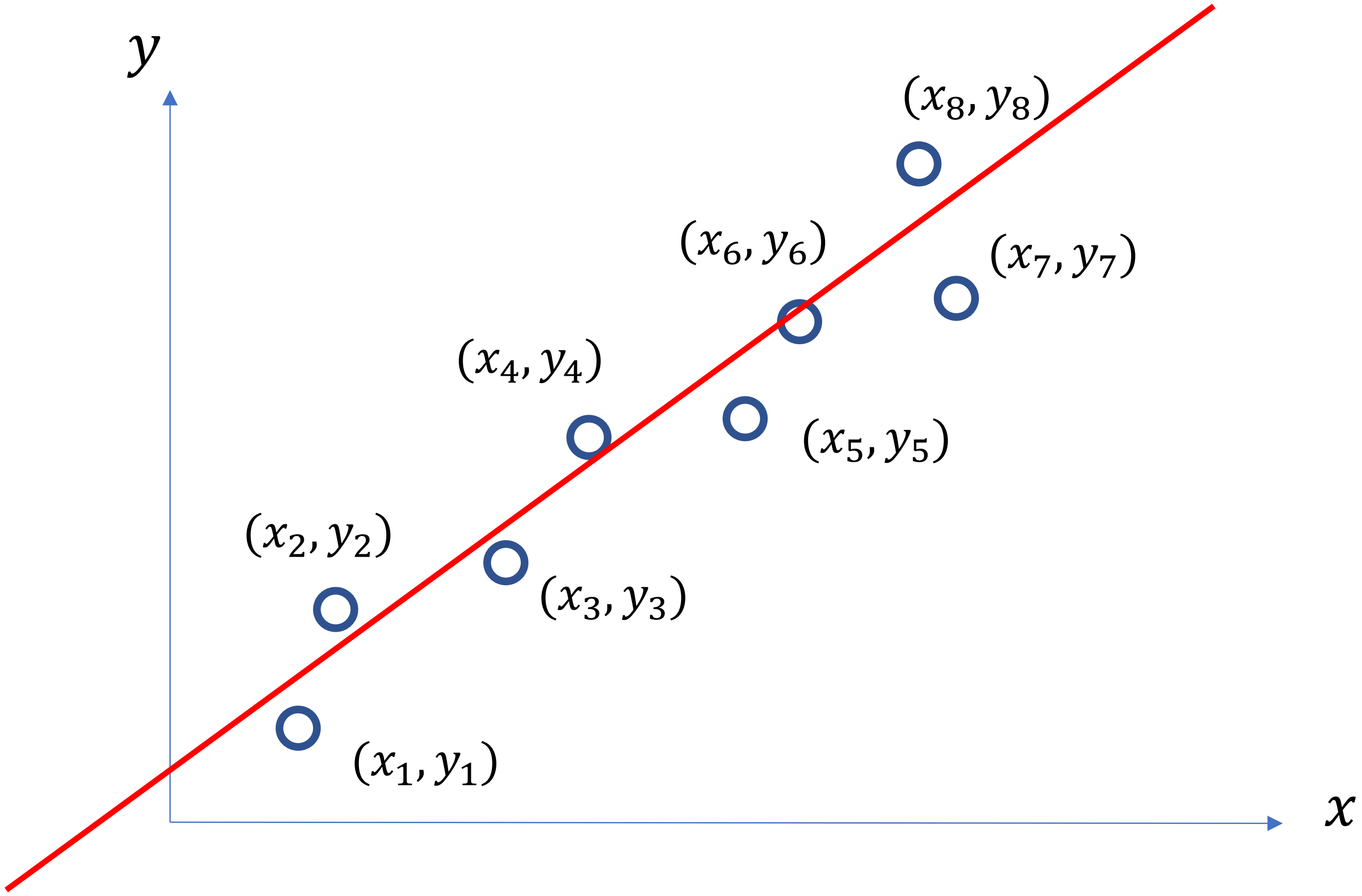


Linear Regression – 1 output, 1 input



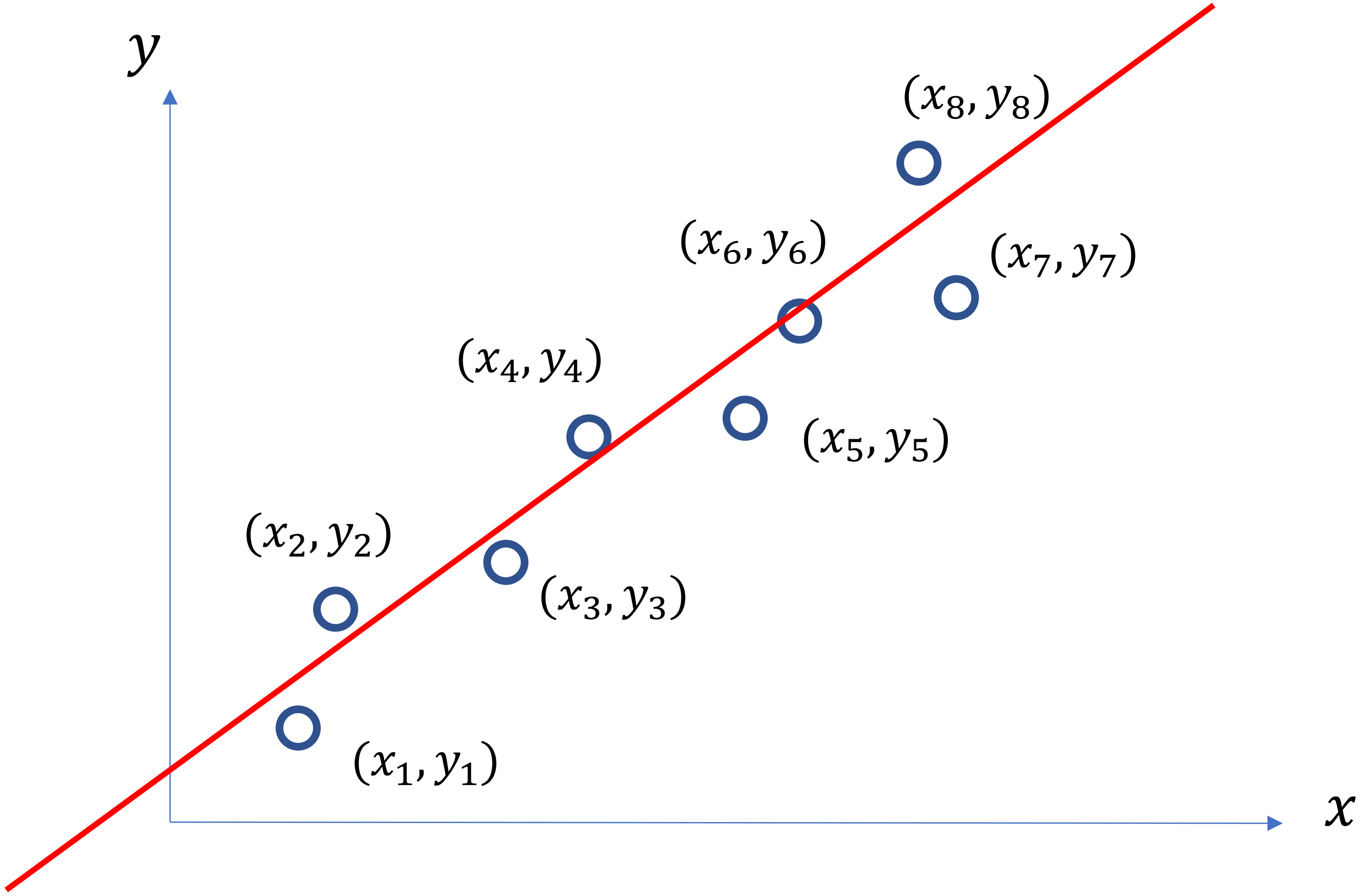
Model: $\hat{y} = wx + b$

Linear Regression – 1 output, 1 input



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

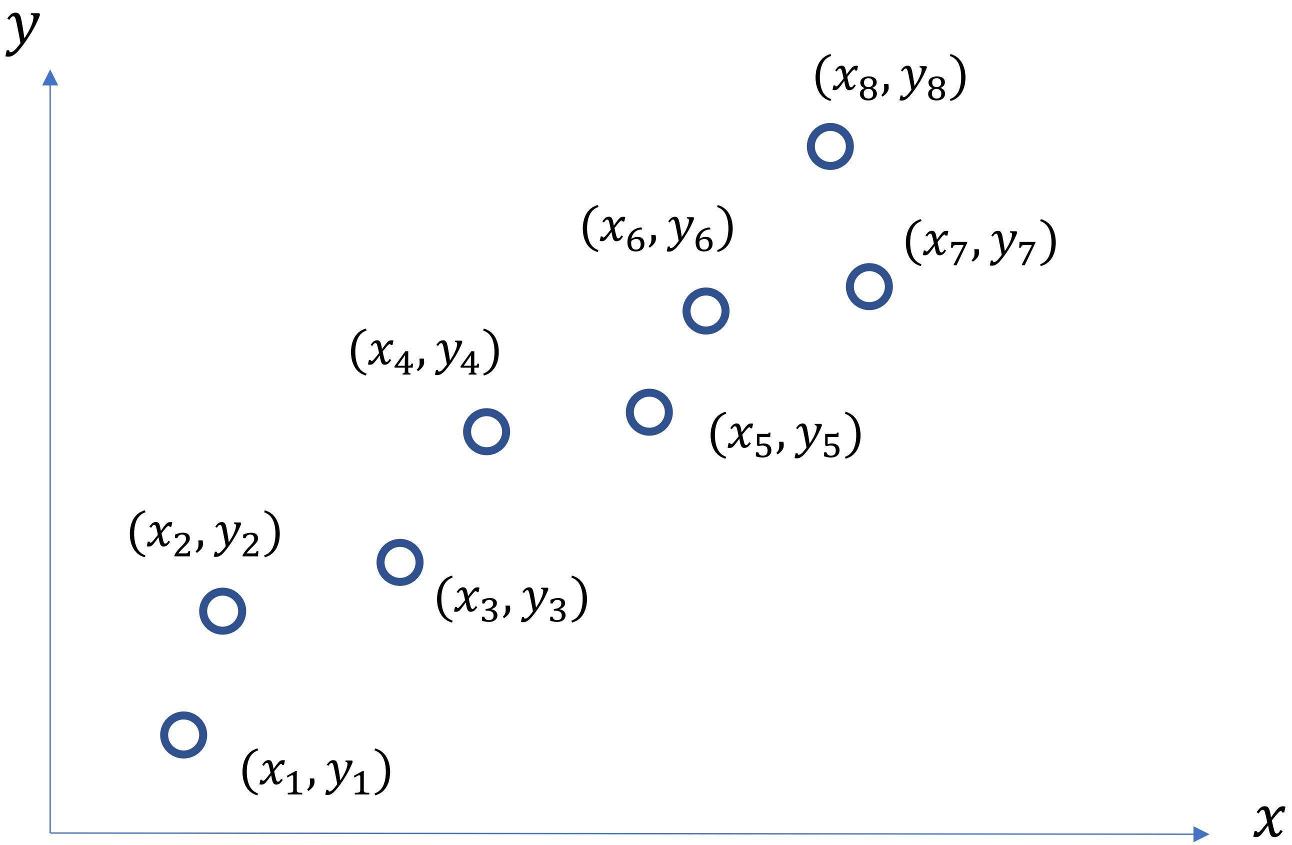
Linear Regression – 1 output, 1 input



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

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

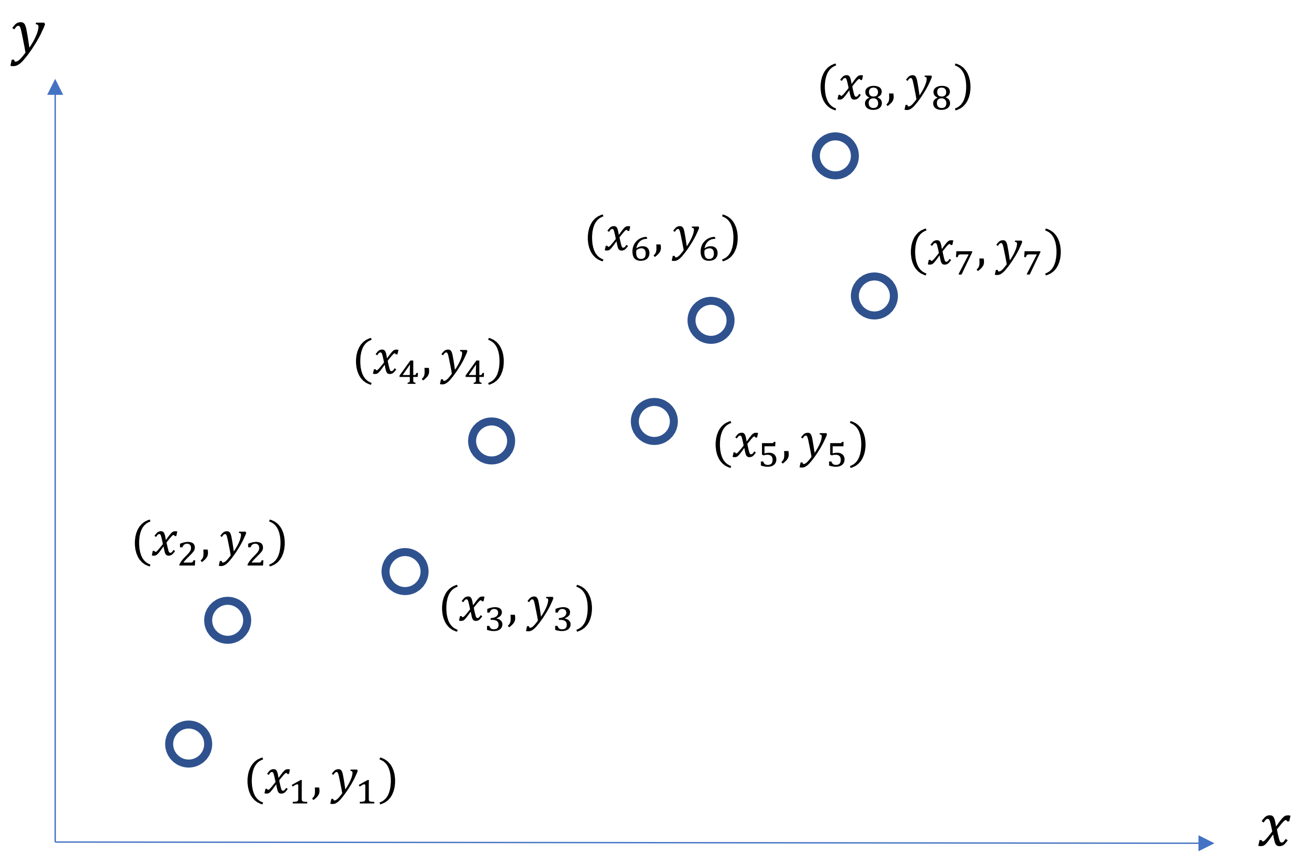
Quadratic Regression



Model: $\hat{y} = w_1 x^2 + w_2 x + b$

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

n-polynomial Regression

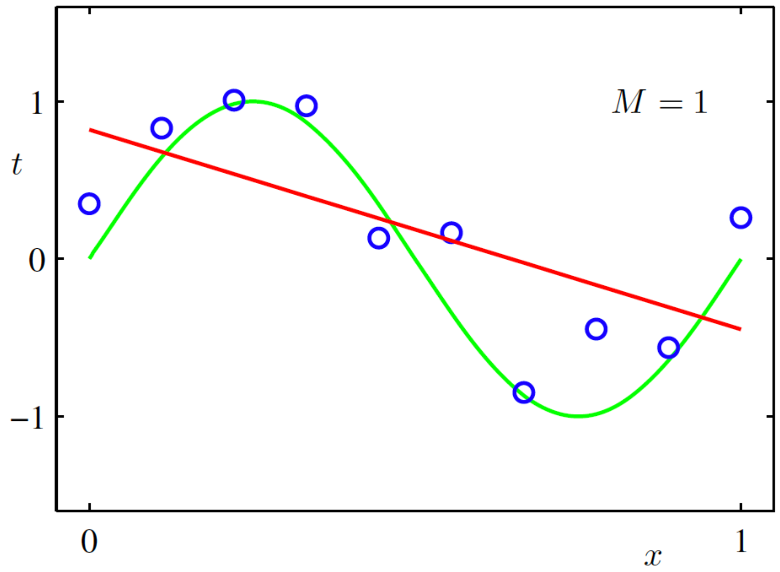


Model: $\hat{y} = w_n x^n + \cdots + w_1 x + b$

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

Overfitting

f is linear

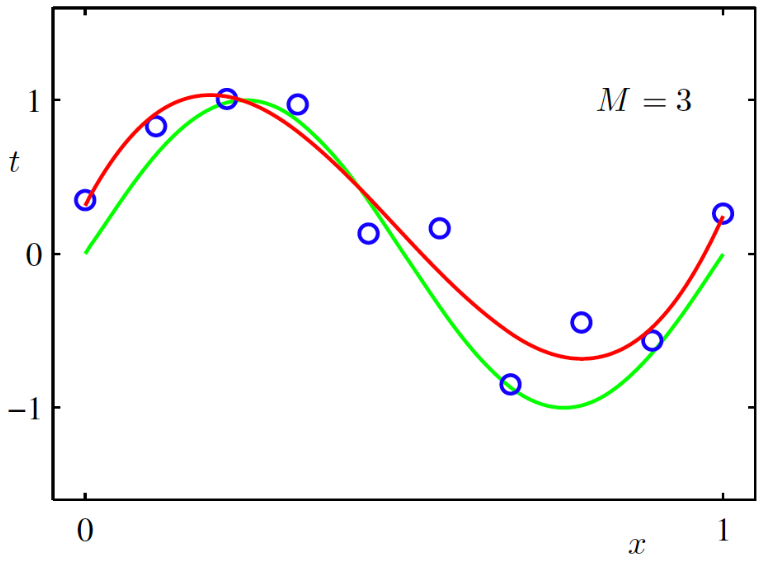


$Loss(w)$ is high

Underfitting

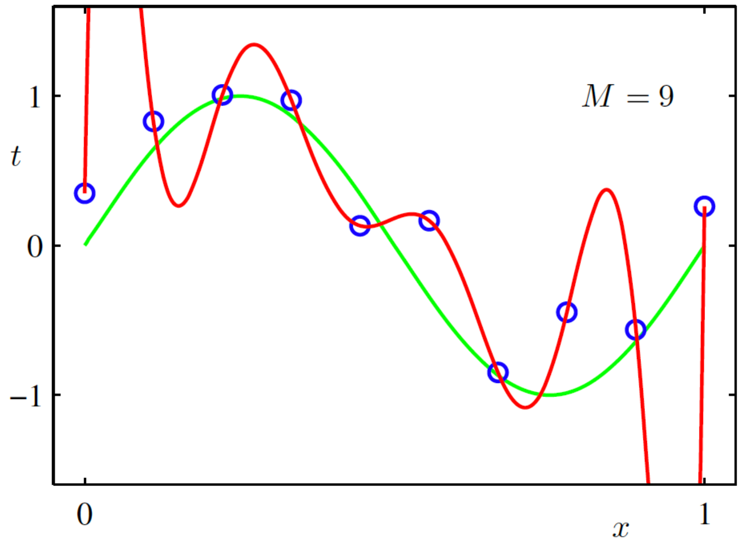
High Bias

f is cubic



$Loss(w)$ is low

f is a polynomial of degree 9



$Loss(w)$ is zero!

Overfitting

High Variance

Christopher M. Bishop – Pattern Recognition and Machine Learning

(mini-batch) Stochastic Gradient Descent (SGD)

$\lambda = 0.01$

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

Initialize w and b randomly

for e = 0, num_epochs **do**

for b = 0, 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:

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

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

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, num_epochs **do**

for b = 0, num_batches **do**

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

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

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

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

end

end

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, num_epochs **do**

for b = 0, num_batches **do**

Compute: $\boxed{dl(w, b)/dw}$ and $\boxed{dl(w, b)/db}$

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, num_epochs **do**

for b = 0, num_batches **do**

Compute: $\boxed{dl(w,b)/dw}$ and $\boxed{dl(w,b)/db}$

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, num_epochs **do**

for b = 0, num_batches **do**

Compute: $\boxed{dl(w, b)/dw}$ and $\boxed{dl(w, b)/db}$

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

$\lambda = 0.01 \qquad \tau = 0.9$

Initialize w and b randomly

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

global v

for e = 0, num_epochs **do**

for b = 0, num_batches **do**

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

$\boxed{\text{Compute:} \quad v = \tau v + dl(w, b)/dw + \alpha w}$

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

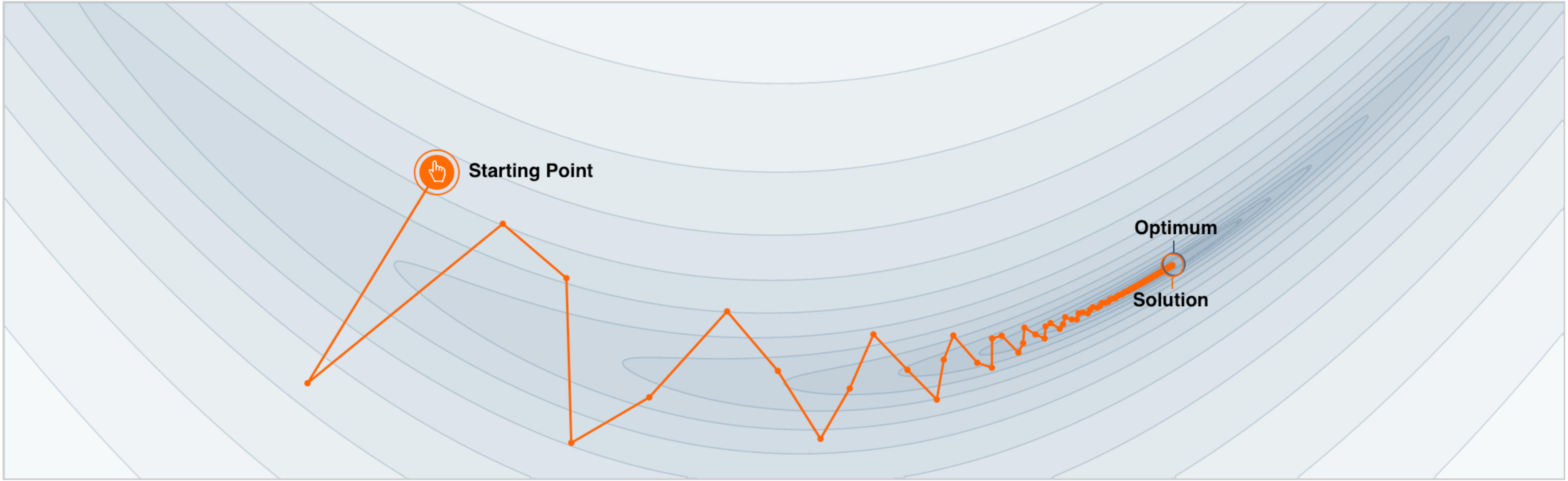
Print: $l(w, b)$ // Useful to see if this is becoming smaller or not.

end

end

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

More on Momentum



Step-size α = 0.0050

0   0.003   0.006

Momentum β = 0.77

0.00   0.500   0.990

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/

Supervised Learning - Classification

Training Data

cat

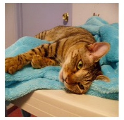
dog

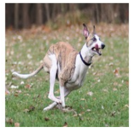
cat

.
.
.

bear

Test Data





.
.
.

Supervised Learning - Classification

Training Data

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

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

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

$\vdots$

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

Supervised Learning - Classification

Training Data

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

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:

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

Questions?