CENTER FOR
Brains
Minds+
Machines

Biological
Discovery
in Woods Hole

INTRODUCTION TO OPTIMIZATION

CBMM Summer School

Aug 12, 2018

What you will learn

- What optimization is used for
- Different optimization concepts
- Commonly used terminology, marked with: 
- In the notes: pointers to how to perform optimization in common programming languages (R, Python, Matlab)

Important terms

- Likelihood
- Maximum likelihood estimate
- Cost function
- Gradient
- Gradient descent
- Global / local minima
- Convex / non-convex functions
- Differentiable functions
- Stochastic gradient descent
- Regularization
- Sparse coding
- Momentum

Materials and notes

http://bit.do/IntroOptim

Find notes at:
http://cbmm.mit.edu/summer-school/2018/resources

Agenda

- Likelihood and cost functions
- Single variable optimization
- Multi-variable optimization
- Optimization for machine learning
  - Stochastic gradient descent
  - Regularization
  - Sparse coding
  - Momentum

LIKELIHOOD & COST FUNCTIONS

What is the likelihood?

Likelihood: The probability of observing your data given a particular model

$$P(D|\theta)$$

A set of data

$$D = \{d_1, d_2, \ldots, d_N\}$$

A model and its associated parameterization

Example: Balls in urns

Suppose we have an urn with white and black balls, but we don't know the proportion. We pull some with replacement and get:

BBWBWWBWWWBW

What is the probability of getting this sequence if the balls were equally distributed – P(Black) = 0.5?

$$P(D \mid \theta = 0.5) = 0.5 * 0.5 * (1 - 0.5) * \ldots = 0.5^5 * (1 - 0.5)^7 = 2.44 * 10^{-4}$$

What is the probability of getting this sequence if the urn was 75% filled with black balls?

$$P(D \mid \theta = 0.75) = 0.75 * 0.75 * (1 - 0.75) * \ldots = 0.75^5 * (1 - 0.75)^7 = 1.45 * 10^{-5}$$

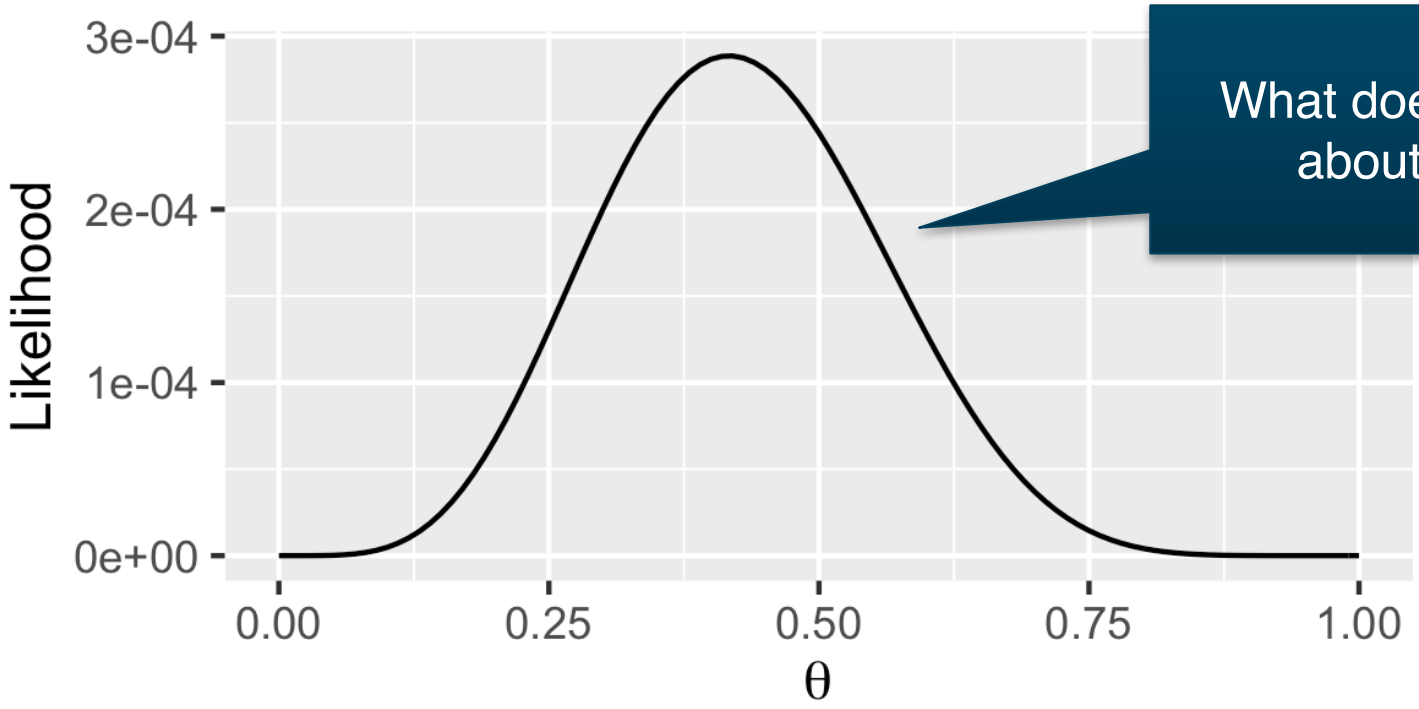
Example: Balls in urns

Suppose we have an urn with white and black balls, but we don't know the proportion. We pull some with replacement and get:

BBWBWWBWWWBW

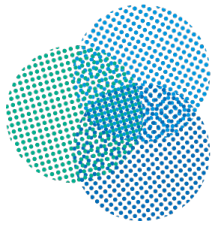
We can abstract this function and view it for any (allowed) value of the probability (θ):

$$L(\theta \mid D) = \theta^5 * (1-\theta)^7$$



What does this tell us about the urn?

Maximum likelihood estimator



Maximum Likelihood Estimator: The value of the parameters of a given model that maximizes the likelihood function for a set of data

$$\hat{\theta}_{mle} = \underset{\theta}{\operatorname{argmax}}\, L(\theta \mid D)$$

Example: Balls in urns

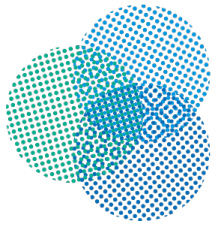
Suppose we have an urn with white and black balls, but we don't know the proportion. We pull some with replacement and get:

BBWBWWBWWWBW

$\Theta_{mle} = 5/12$

How do we find this value more generally?

Cost functions



Cost function / loss function: A function that maps a set of events into a number that represents the "cost" of that event occurring

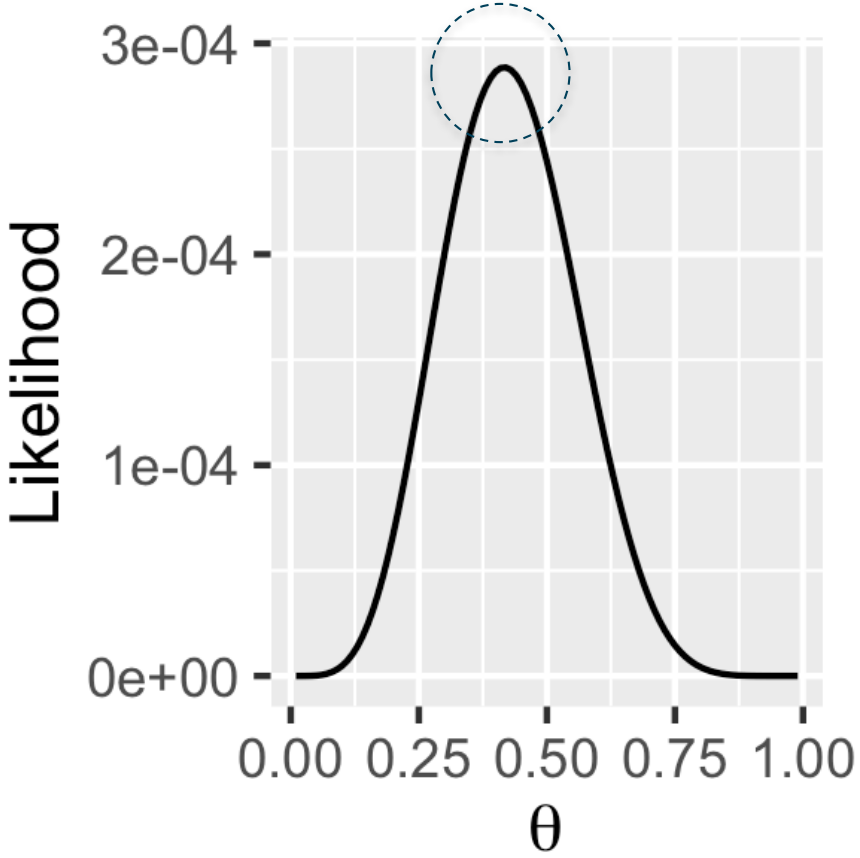
- More general than likelihood

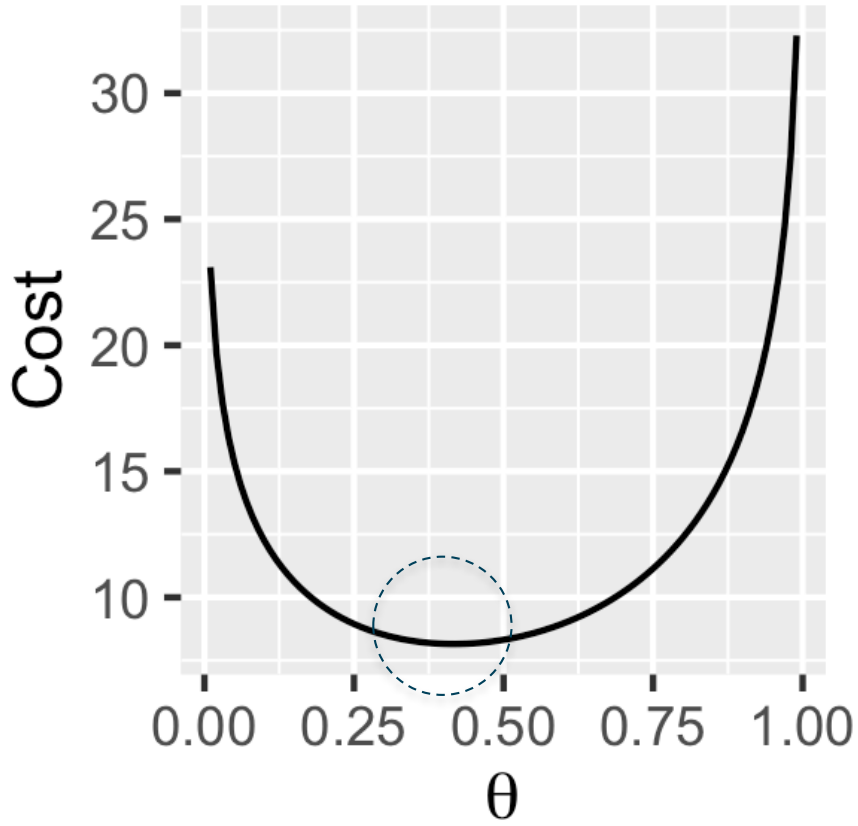
- The "cost" of likelihood is typically the negative log-likelihood:

$$C(\theta, D) = -\log(L(\theta \mid D))$$

Likelihood -> Cost

$$C(\theta, D) = -\log(L(\theta \mid D))$$





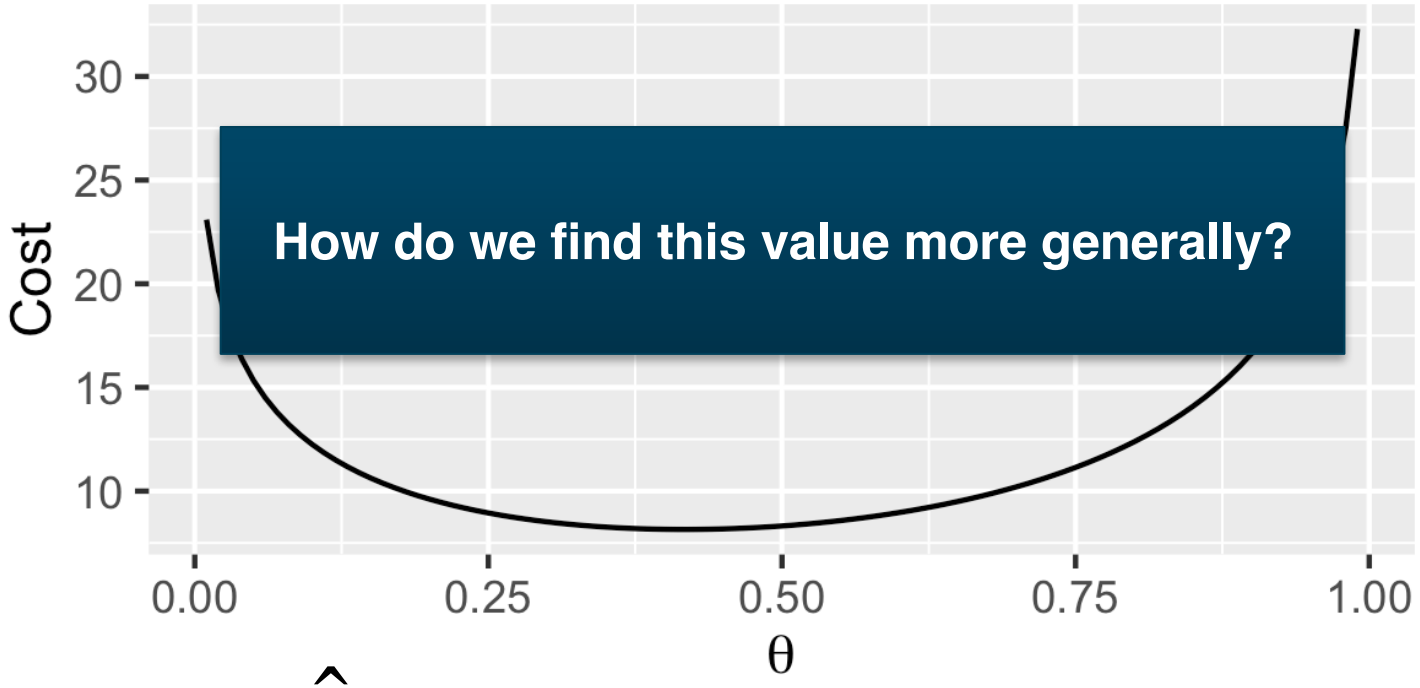
$$\hat{\theta}_{mle} = \underset{\theta}{\operatorname{argmax}}\, L(\theta \mid D) = \underset{\theta}{\operatorname{argmin}}\, C(\theta, D)$$

SINGLE-VARIABLE OPTIMIZATION

Back to the urn problem…

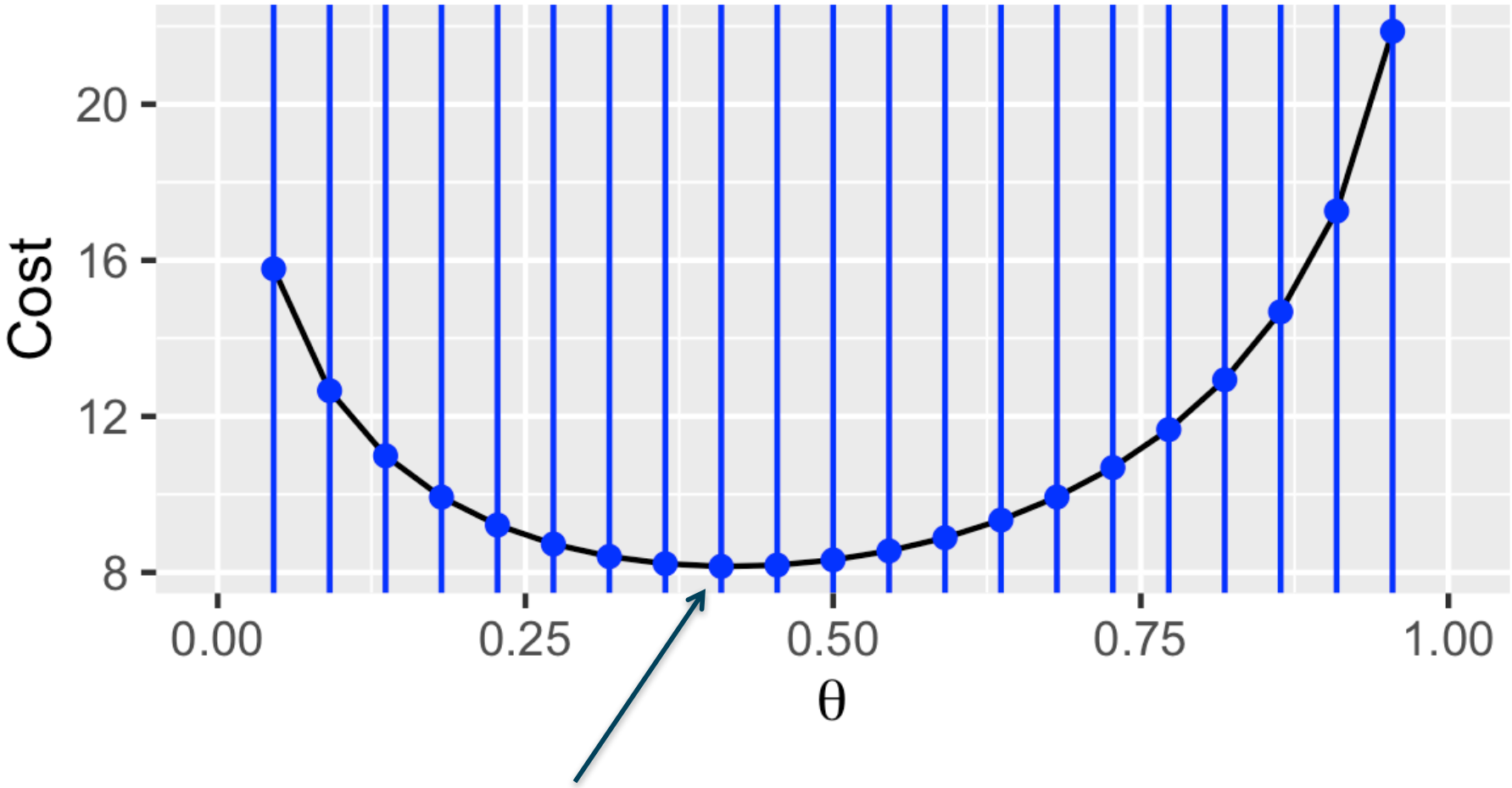
Suppose we have an urn with white and black balls, but we don't know the proportion. We pull some with replacement and get:

BBWBWWBWWWBW



$$\hat{\theta}_{mle} = \underset{\theta}{\operatorname{argmin}} C(\theta, D)$$

Grid search (brute force)



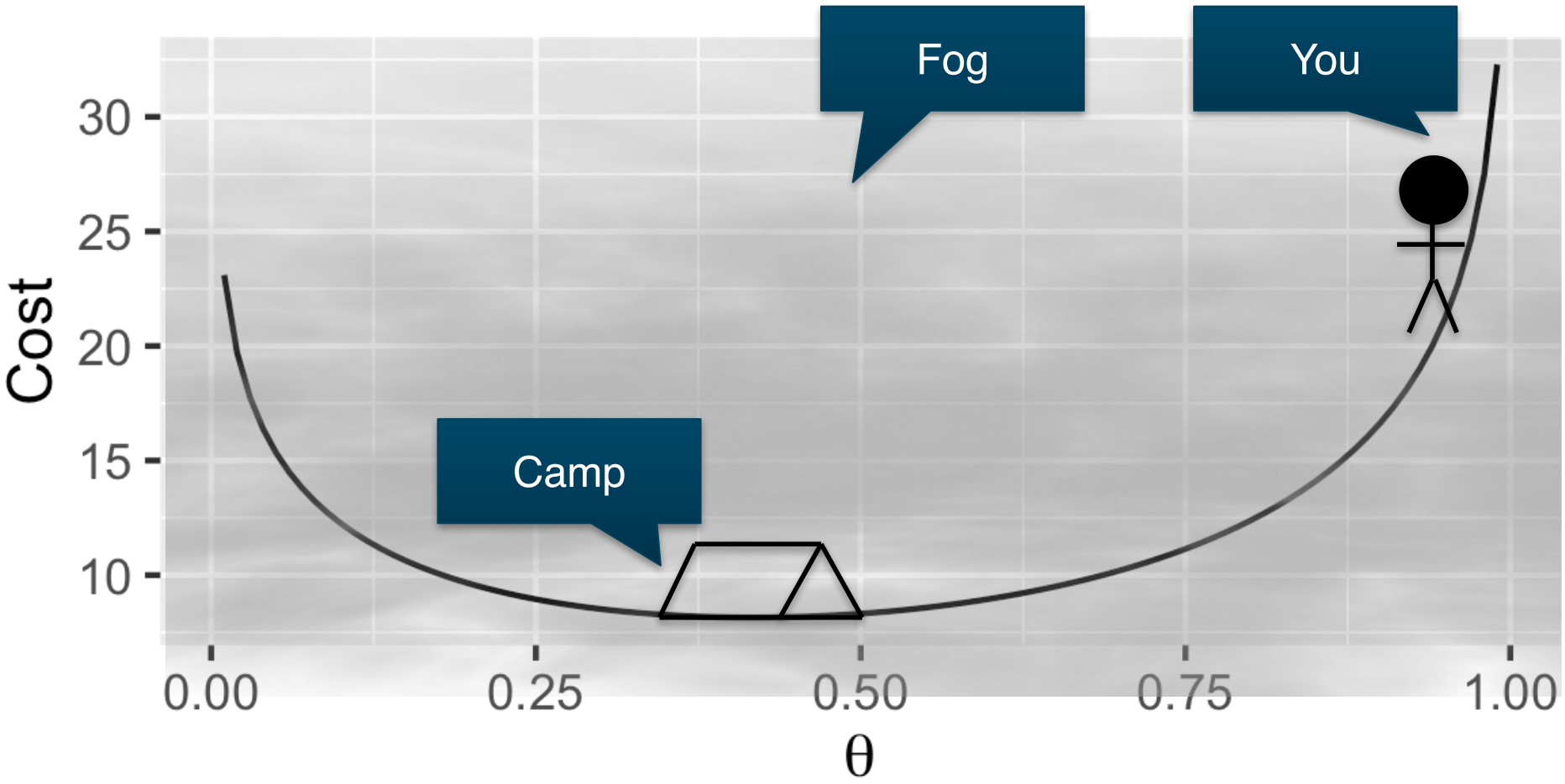
Here's the minimum!

Grid search (brute force)

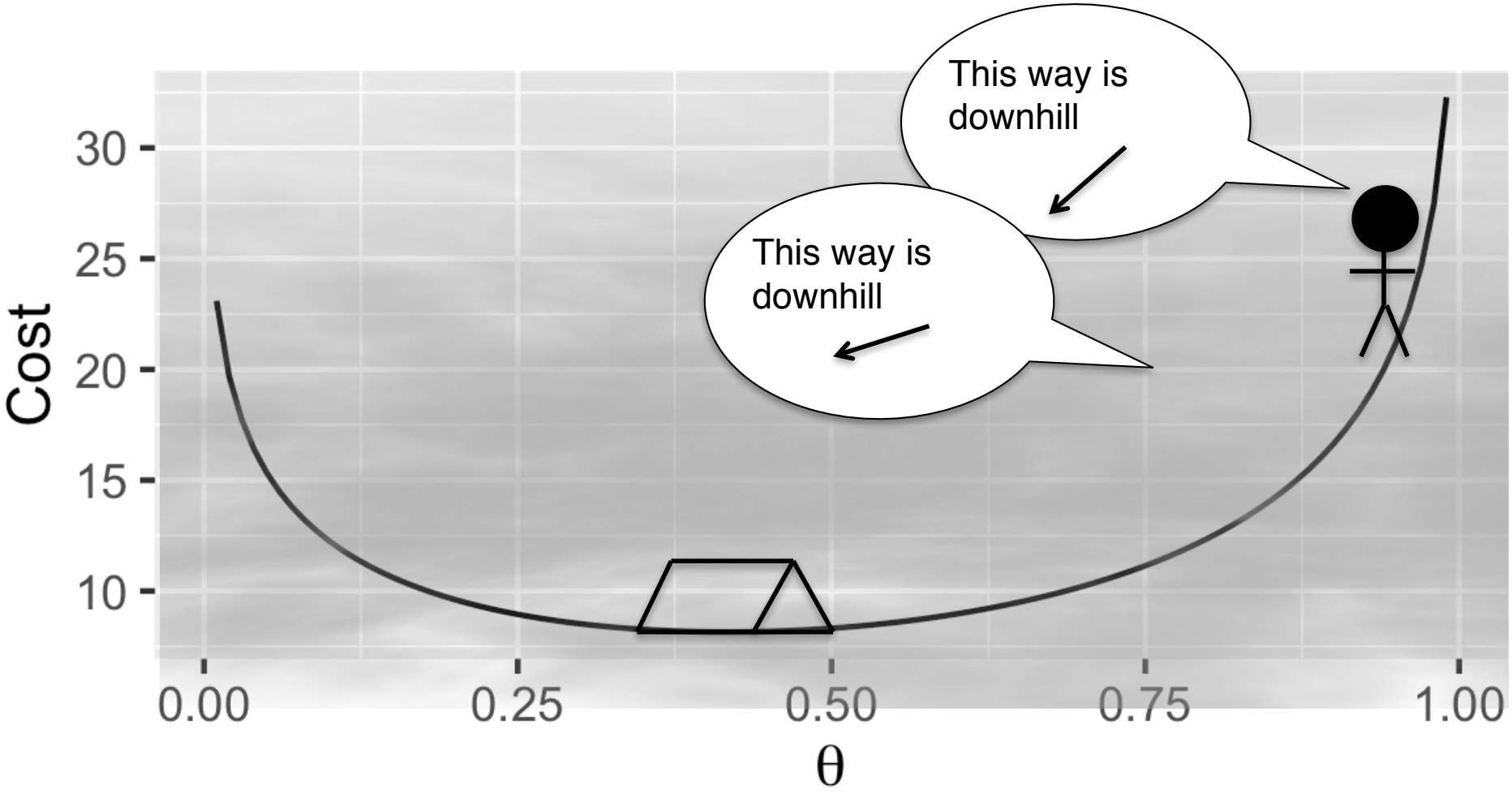
- Pros:
  - Guaranteed to get you close to the minimum (with fine enough grid)
  - Easy to implement

- Cons:
  - Inefficient
  - Only really works for 1-2 parameters

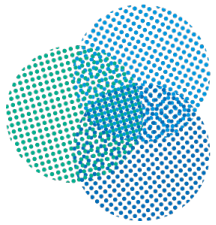
Gradient descent



Gradient descent



Gradient descent



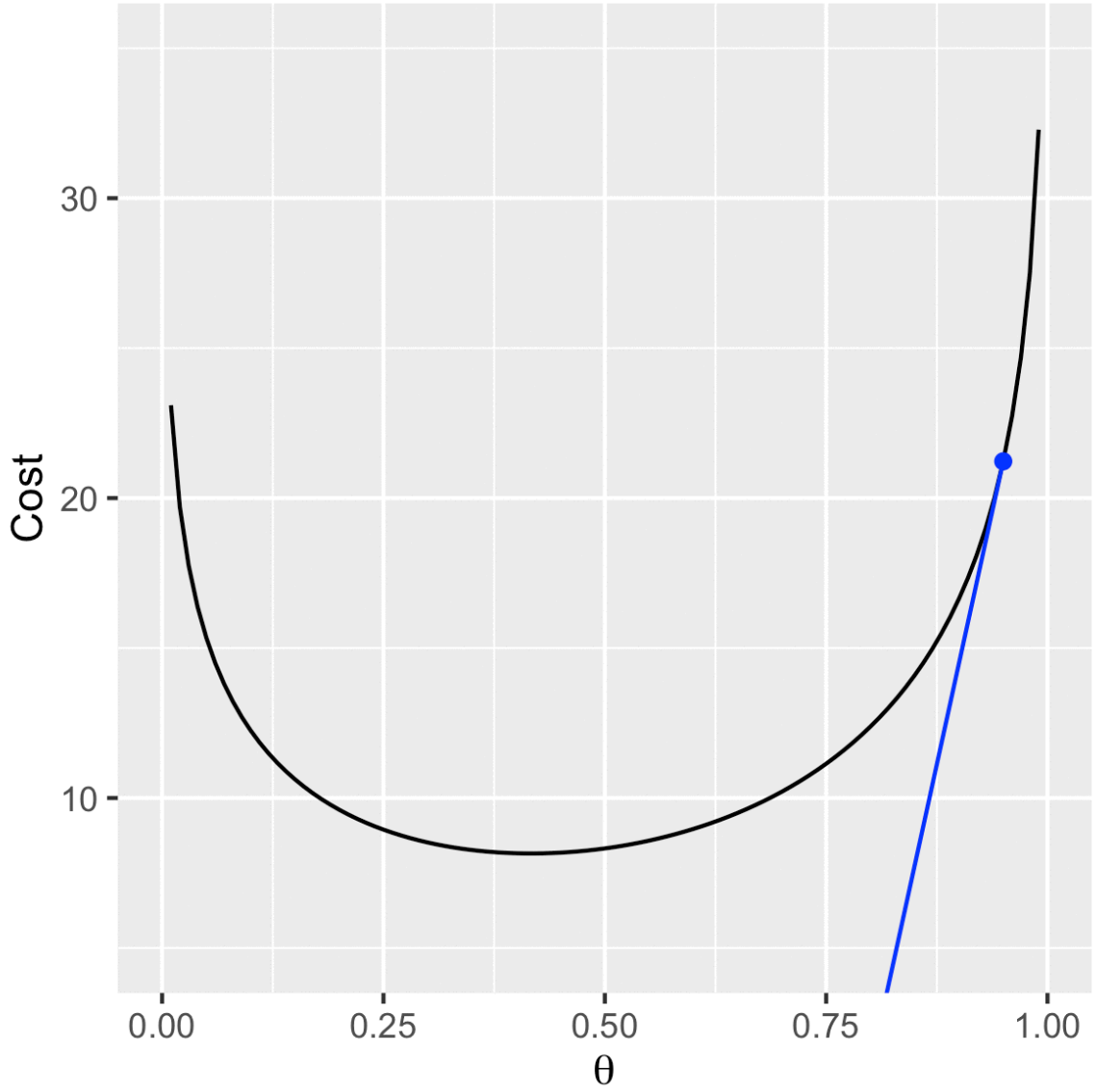
Gradient descent: An optimization technique where parameters are updated proportionally to the negative gradient to continually step "downhill" in the cost function

$$\theta_{t+1} = \theta_t - \gamma \nabla C(\theta_t)$$

Step-size parameter

$$\nabla C = \frac{dC}{d\theta}$$

Gradient descent



Gradient descent

```
theta = .95
gamma = .002
tau = .000001

urn.gradient = function(theta) {
  return(7/(1-theta) - 5/theta)
}

prior.cost = urn.cost(theta)

# Anything to do with plot.thetas or plot.costs can be ignored - this is just for graphing purposes
plot.thetas = theta
plot.costs = prior.cost

running = TRUE
while(running) {
  grad = urn.gradient(theta)
  theta = theta - gamma * grad
  new.cost = urn.cost(theta)
  plot.thetas = c(plot.thetas, theta)
  plot.costs = c(plot.costs, new.cost)
  if(abs(new.cost - prior.cost)< tau) {
    running = FALSE
  } else {
    prior.cost = new.cost
  }
}
# This is our solution
writeLines(paste("Gradient descent value:",theta,sep=' '))
```

But we had to calculate the gradient!

```
## Gradient descent value: 0.417056684695153
```

≈ 5 / 12

Gradient descent

But what if we don't know how to calculate the gradient?

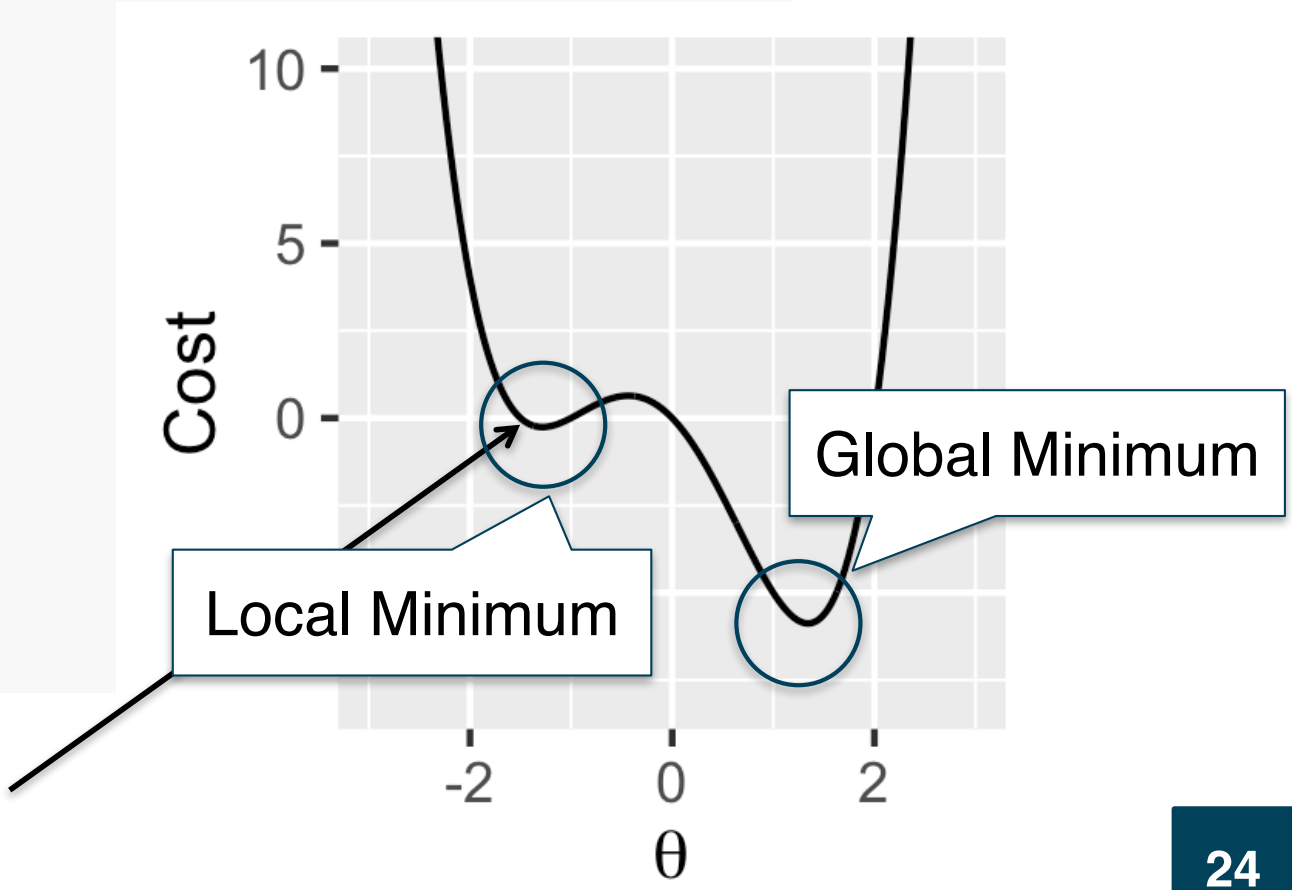
… We approximate it!

$$\nabla C(\theta) \approx \frac{C(\theta + \epsilon) - C(\theta - \epsilon)}{2\epsilon}$$

Local vs. global minima

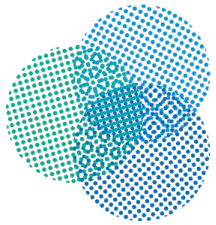
$$C(\theta) = \theta * (\theta + 1) * (\theta + 1.5) * (\theta - 2)$$

```
arbitrary.cost = function(theta) {
  return(theta*(theta+1)*(theta+1.5)*(theta-2))
  }
arbitrary.gradient = function(theta, epsilon = 0.001) {
  return( (arbitrary.cost(theta+epsilon) - arbitrary.cost(theta-epsilon)) / (2*epsilon) )
}

theta = -2.5
gamma = .0001
tau = .000001

prior.cost = arbitrary.cost(theta)

running = TRUE
while(running) {
  grad = arbitrary.gradient(theta)
  theta = theta - gamma * grad
  new.cost = arbitrary.cost(theta)
  if(abs(new.cost - prior.cost)< tau) {
    running = FALSE
  } else {
    prior.cost = new.cost
  }
}
print(theta)
```



θ = -1.294

Local vs. global minima



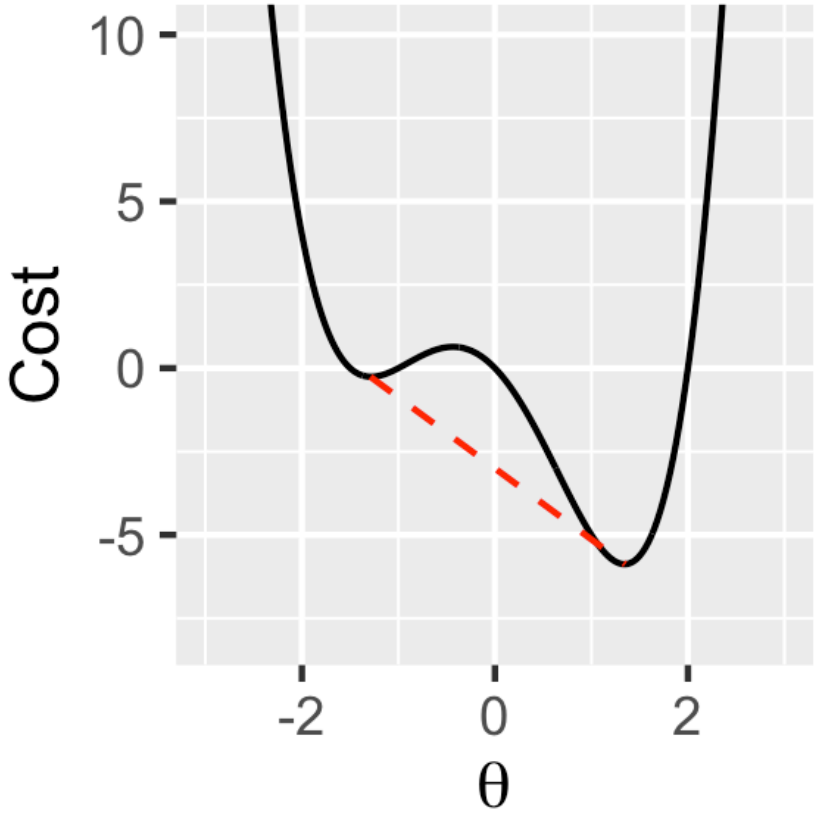
Local minimum: a point $x*$ is a local minimum if it is the lowest value of the function within some range centered on $x*$

Global minimum: a point $x*$ is a global minimum if it is the lowest value of the function across all allowable values of the parameter

Convex vs. non-convex functions

Convex function: A function is convex if for every line segment drawn between two points on the function, that line segment passes above the graph

Non-convex function: A function that is not convex



A minimum on a convex function is guaranteed to be a global minimum!

Implementation

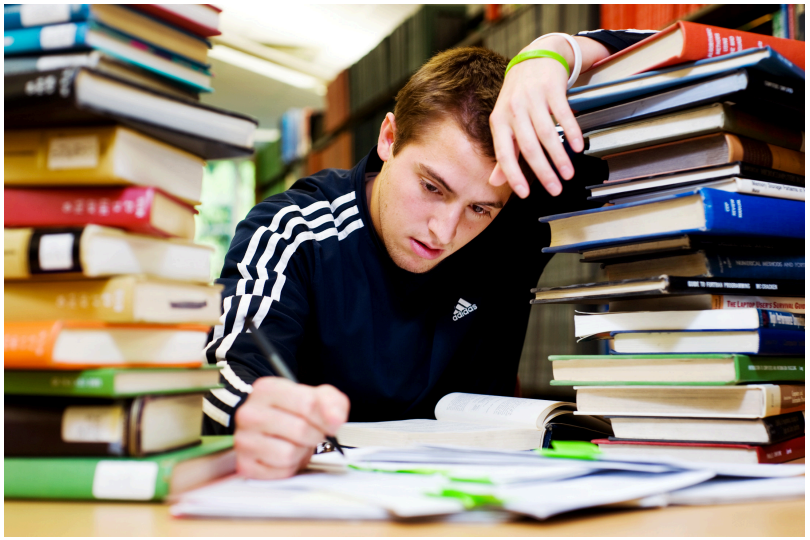
Language	Single-var
R	*optimize*
Python*	*minimize_scalar*
Matlab	*fmindbnd*

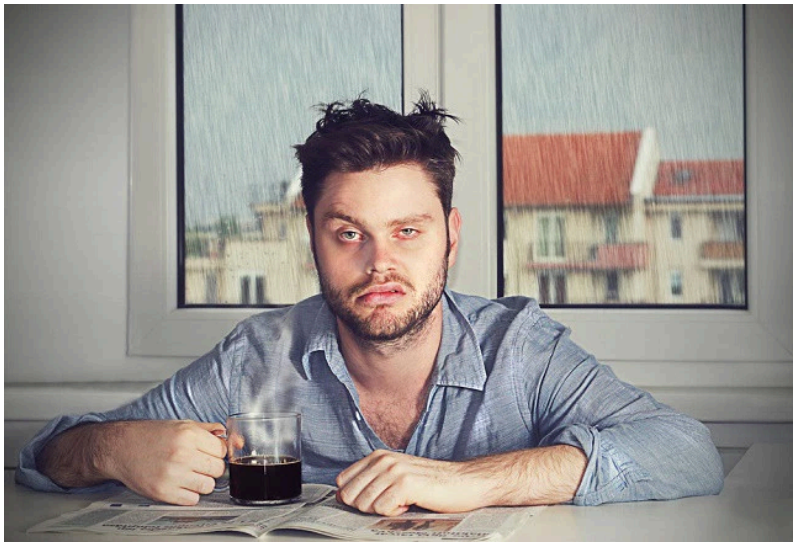
*: Python optimization functions require the “scipy” package

The implementations of optimization use more advanced algorithms than discussed here. See the notes for further details!

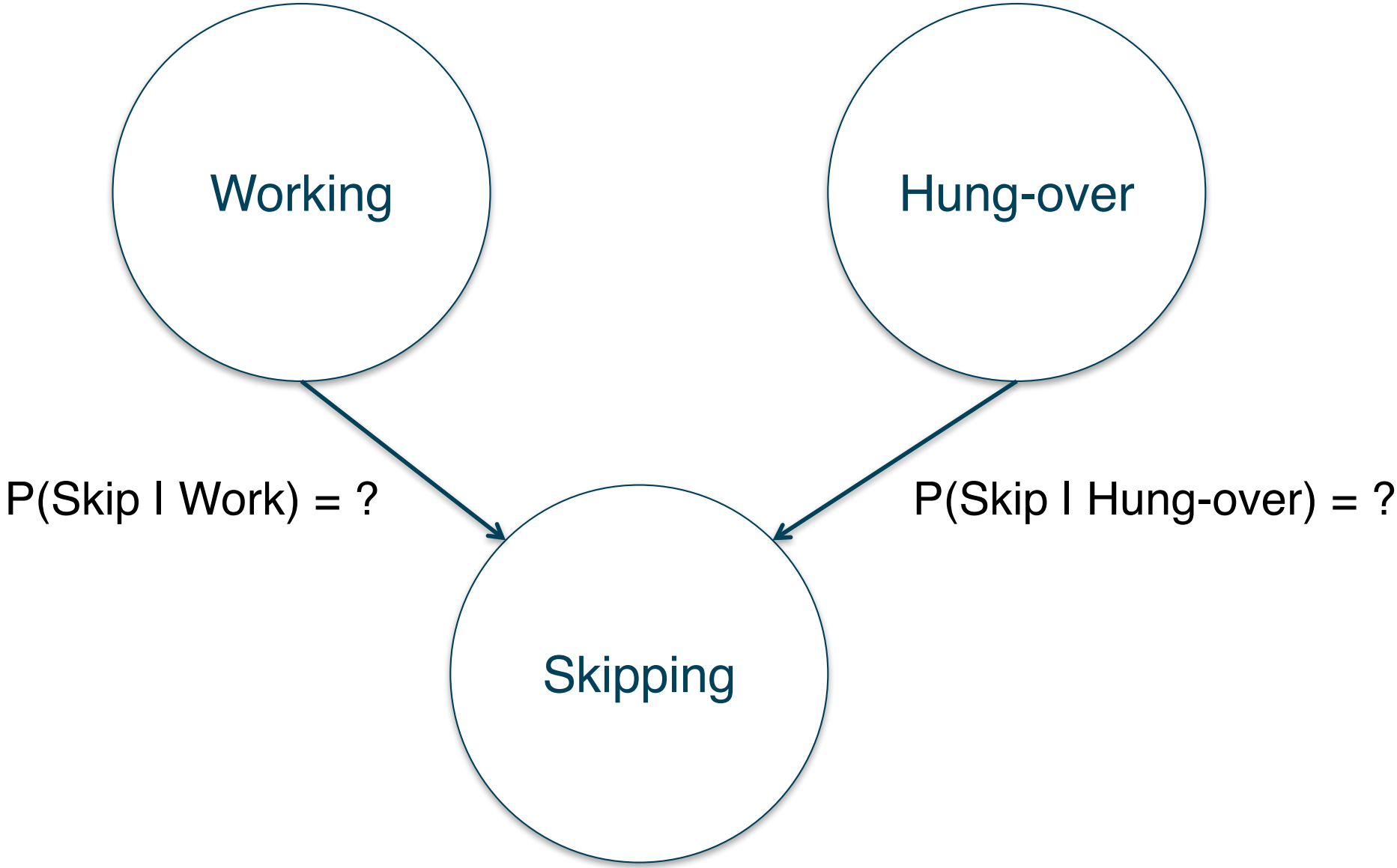
MULTI-VARIABLE OPTIMIZATION

Lecture attendance problem

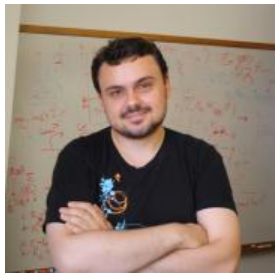
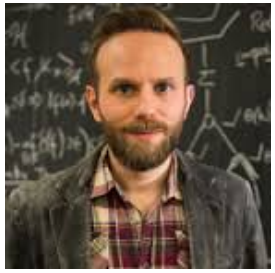




Lecture attendance problem



Lecture attendance problem

	Working	Hung-over	Skipping
	✔	✖	✖
	✔	✔	✖
	✖	✖	✖
	✖	✔	✔

Lecture attendance problem

Attending

	HungO	~HungO
Work	42	17
~Work	57	26

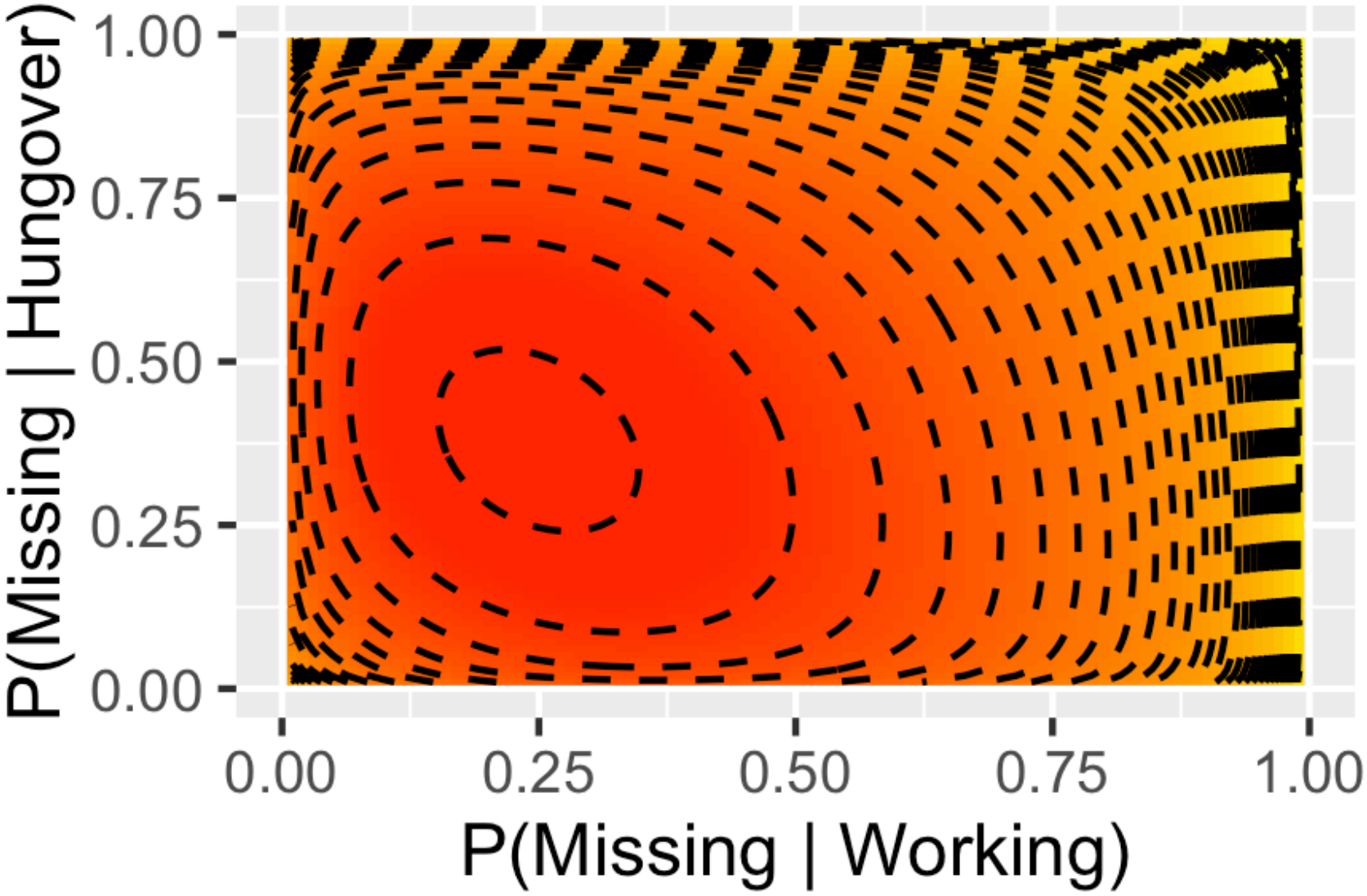
Skipping

	HungO	~HungO
Work	0	10
~Work	18	30

$$P(Skip) = \begin{cases} 0 & if\ not\ working\ \&\ not\ hungover \\ \theta_{work} & if\ working\ \&\ not\ hungover \\ \theta_{hungover} & if\ not\ working\ \&\ hungover \\ \theta_{work} + \theta_{hungover} - \theta_{work} * \theta_{hungover} & if\ working\ \&\ hungover \end{cases}$$

$$C([\theta_{work}, \theta_{hungover}]) = \sum \begin{cases} -log\big(P(Skip)\big) & if\ skipping \\ -log\big(1 - P(Skip)\big) & if\ not\ skipping \end{cases}$$

Lecture attendance problem

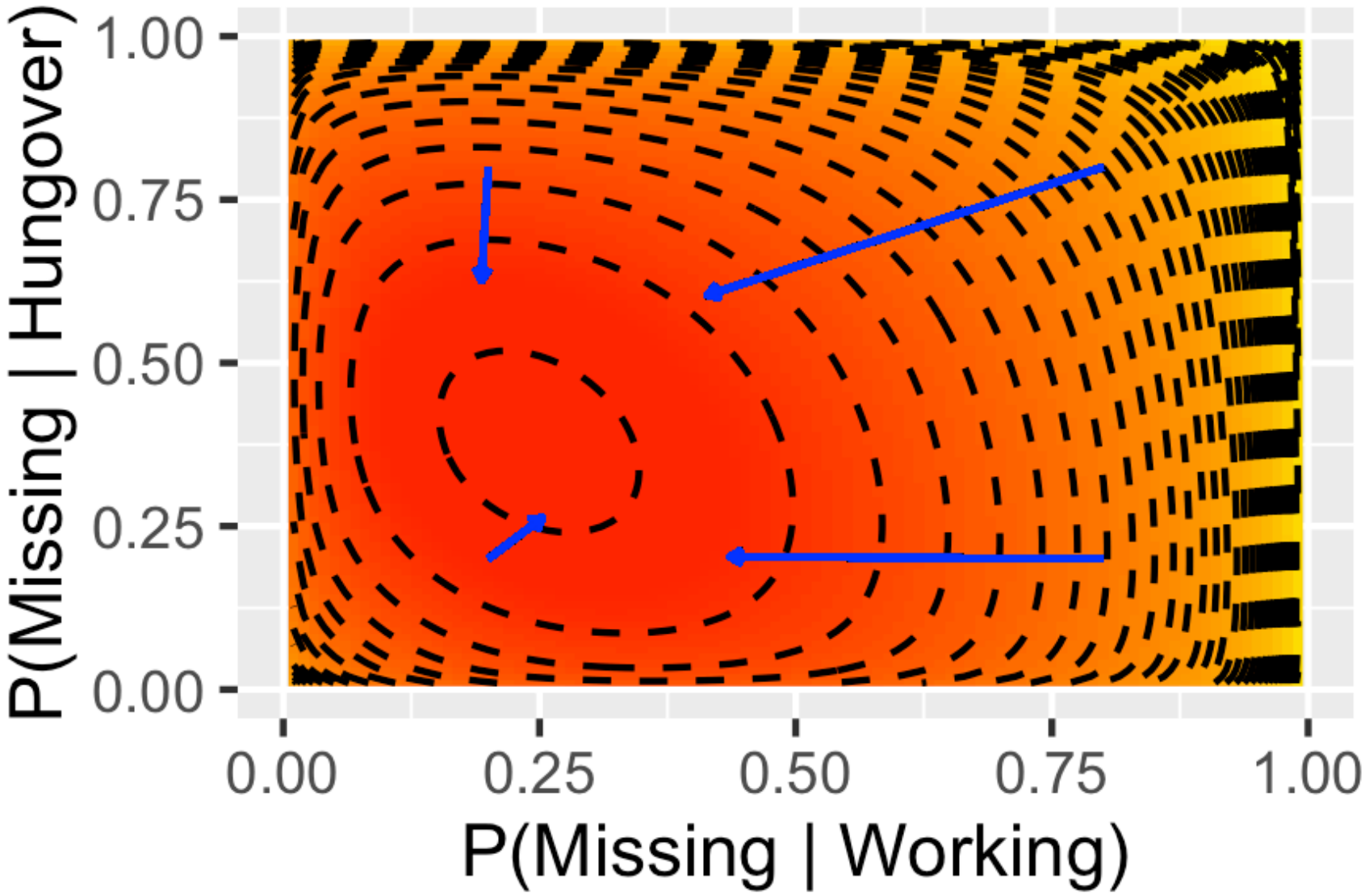


Multi-dimensional gradients

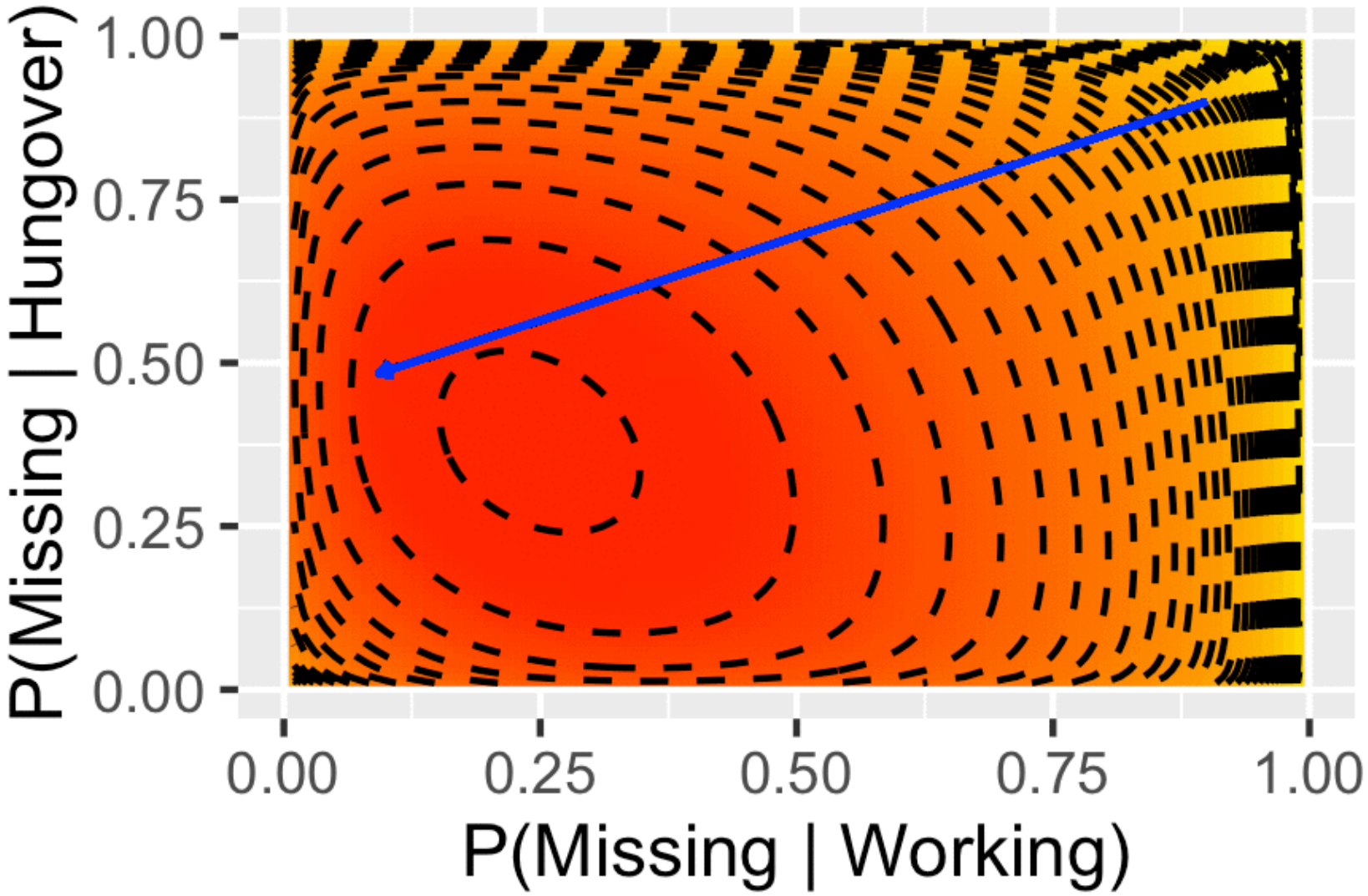
Almost the same as single-dimensional, but as a vector to represent both the magnitude and direction:

$$\nabla C(\theta) = \left[\frac{\partial C}{\partial \theta_0}, \frac{\partial C}{\partial \theta_1}, \ldots, \frac{\partial C}{\partial \theta_N} \right]$$

Multi-dimensional gradients



Multi-dimensional gradient descent



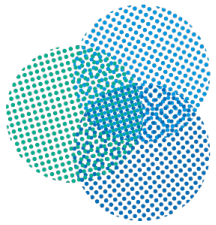
Multi-dimensional gradient descent

```
theta = c(.9, .9)
gamma = .0001
tau = .01

prior.cost = attendance.cost(theta)
running = TRUE

# For showing the descent
plt.xs = theta[1]
plt.ys = theta[2]

while(running) {
  grad = attendance.gradient(theta)
  theta = theta - gamma * grad
  plt.xs = c(plt.xs, theta[1])
  plt.ys = c(plt.ys, theta[2])
  new.cost = attendance.cost(theta)
  if(abs(new.cost - prior.cost) < tau) {
    running = FALSE
  } else {
    prior.cost = new.cost
  }
}
```

```
## Theta_Work: 0.241490536756828
## Theta_Hungover: 0.416981437389993
```

$\theta_{work} = 0.3$
$\theta_{hungover} = 0.4$

Differentiable functions



Differentiable: A function is differentiable if there exists a function that provides the gradient at every allowable value of its parameters

Implementation

Language	Single-var	Multi-var
R	*optimize*	*optim*
Python*	*minimize_scalar*	*minimize*
Matlab	*fmindbnd*	*fminsearch*

*: Python optimization functions require the “scipy” package

The implementations of optimization use more advanced algorithms than discussed here. See the notes for further details!

OPTIMIZATION FOR MACHINE LEARNING

Optimization for machine learning

- Stochastic gradient descent
- Regularization
- Sparse coding
- Momentum

Stochastic gradient descent

- What happens when the cost function is expensive?

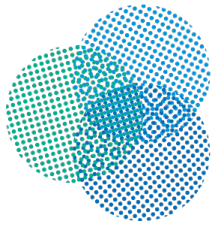
>14 million images!

- You want to fit your model to the entire dataset… but computing the entire cost function thousands of times would take forever!

Stochastic gradient descent

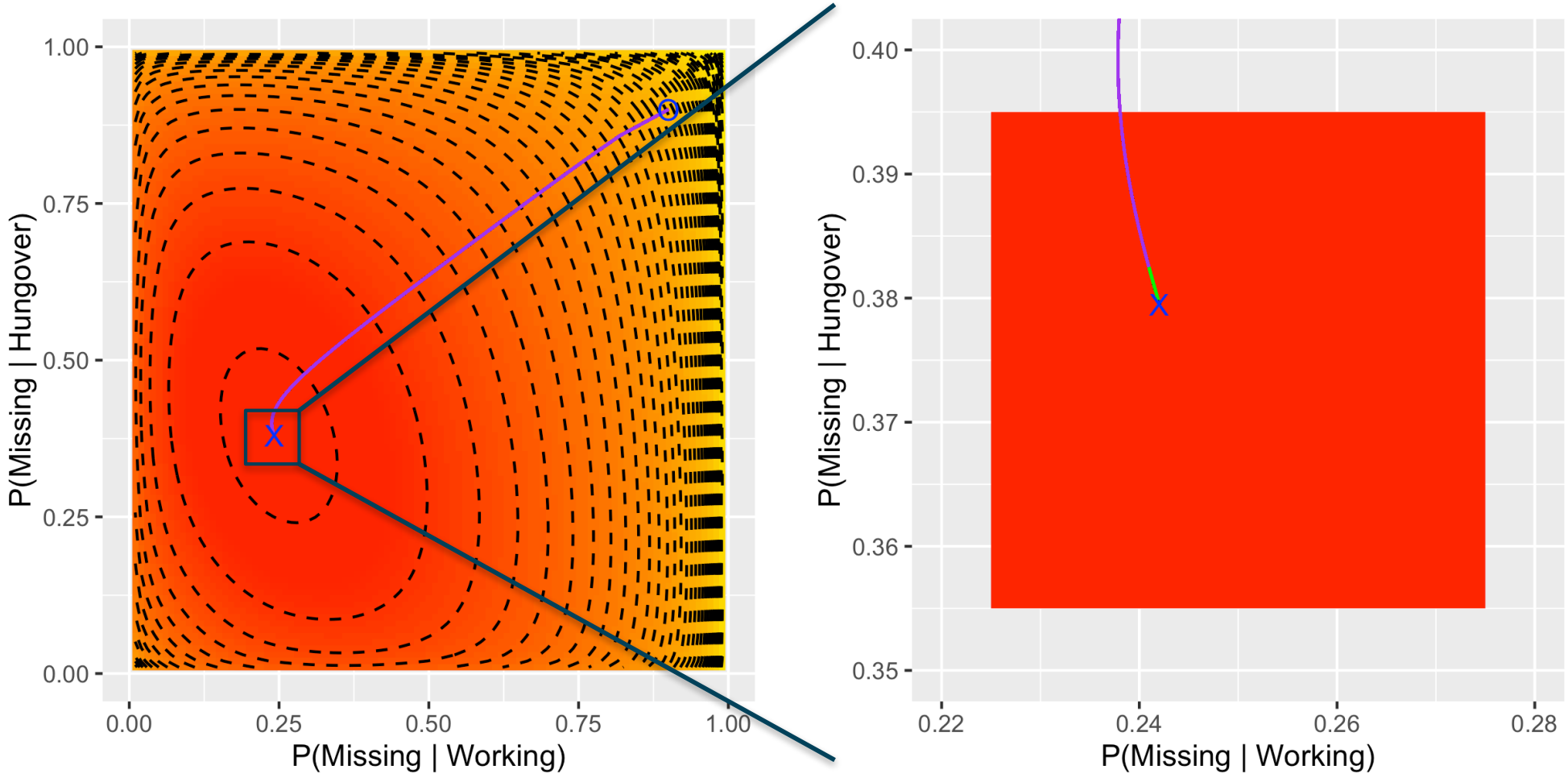
If your cost function is decomposable into costs from each individual observation, you can approximate the gradient by calculating the cost on a random subset of the data

Stochastic gradient descent



Stochastic gradient descent: Performing gradient descent iteratively using subsets of the full data set to calculate the cost function

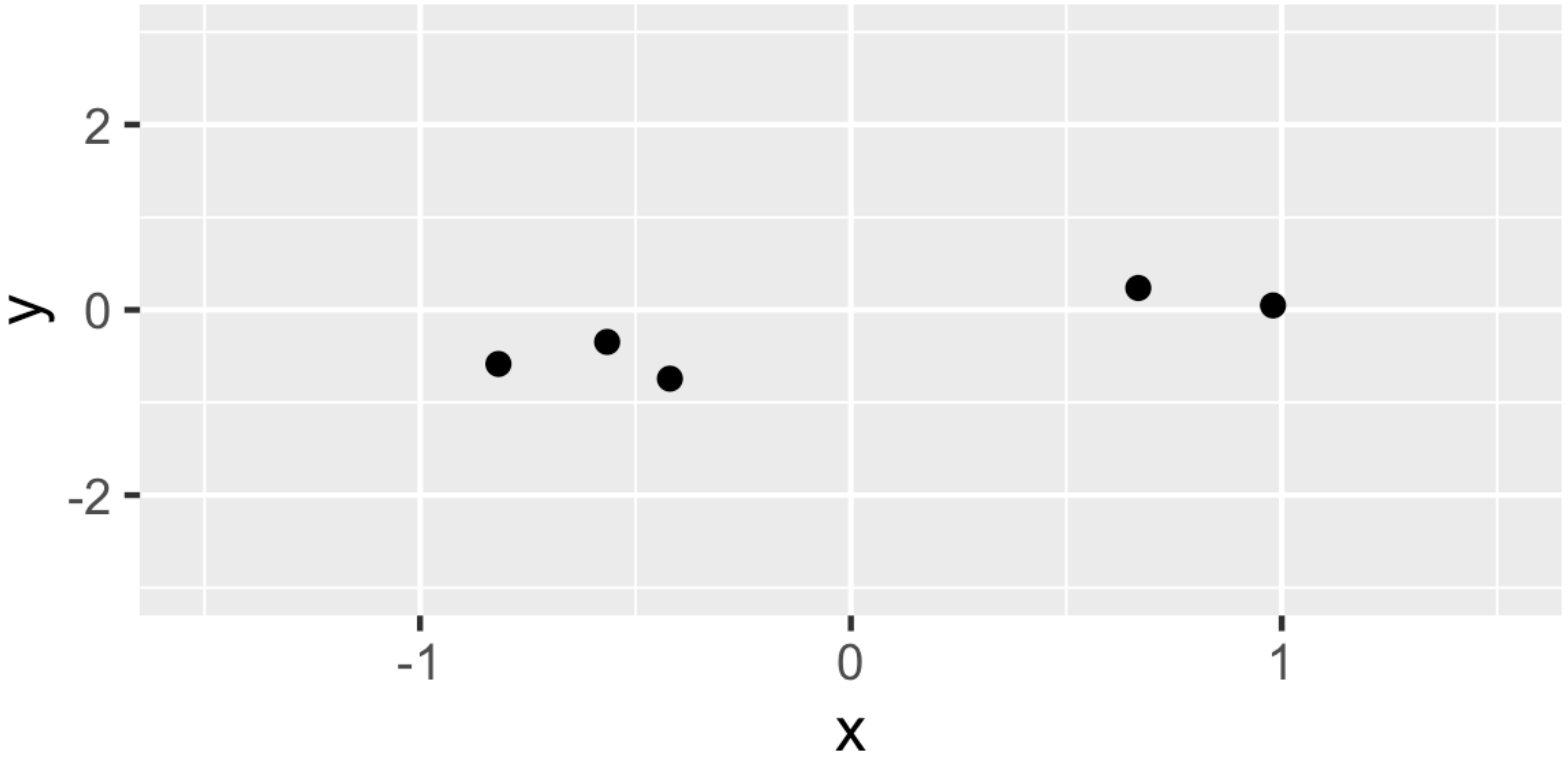
Stochastic gradient descent



Regularization

Polynomial machine:

$$y_i = b_0 + b_1 * x_i + b_2 * x_i^2 + b_3 * x_i^3 + b_4 * x_i^4 + b_5 * x_i^5 + b_6 * x_i^6 + b_7 * x_i^7 + \epsilon$$

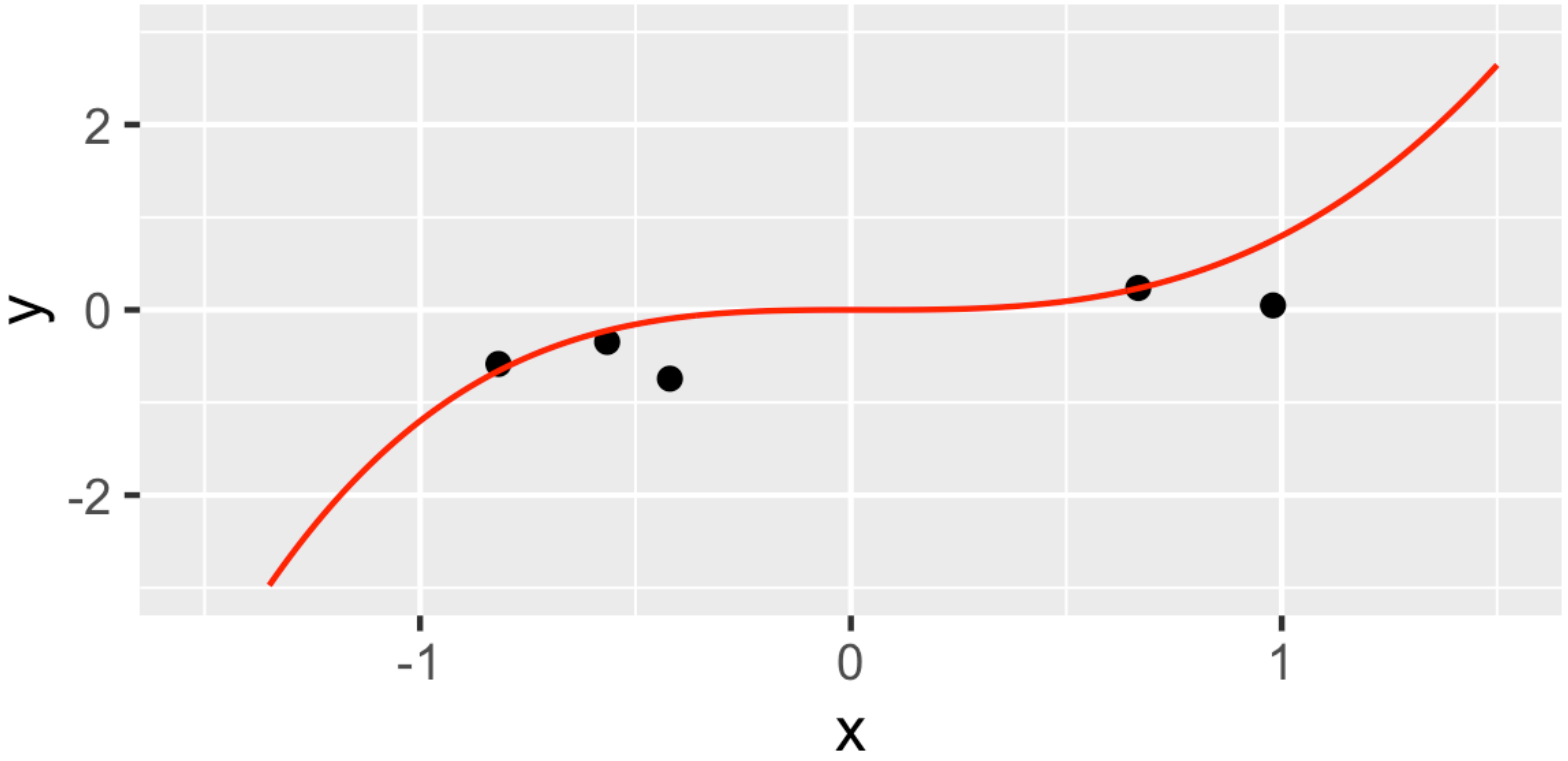


Regularization

Polynomial machine:

$$y = -0.1 * x^2 + x^3 - 0.1 * x^4 + \epsilon$$

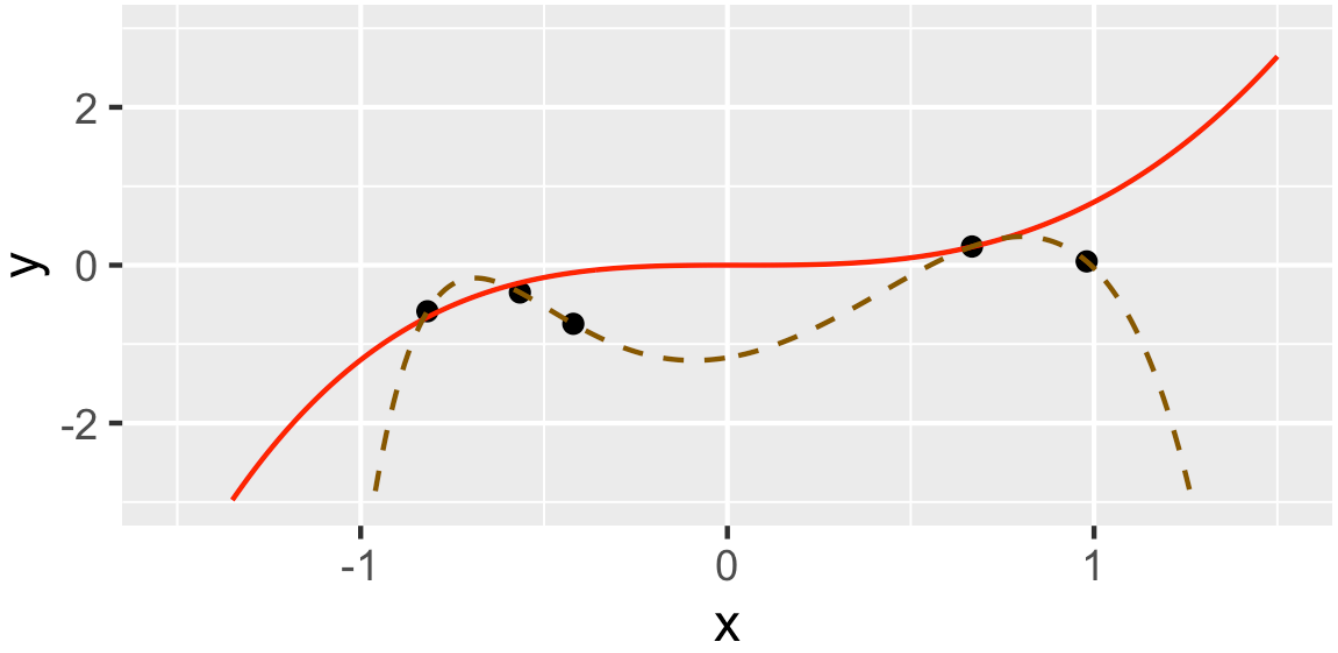
$$\epsilon \sim \mathcal{N}(0, 0.6)$$



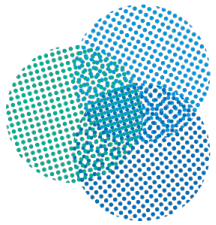
Regularization

$$\hat{y} = \sum_{n=0\ldots7} b_n * x^n$$

$$C_0(\theta, y) = \sum [(y_i - \hat{y}_i)^2]$$



Regularization



Regularization: Imposing an additional cost that is related to the magnitude of the parameters

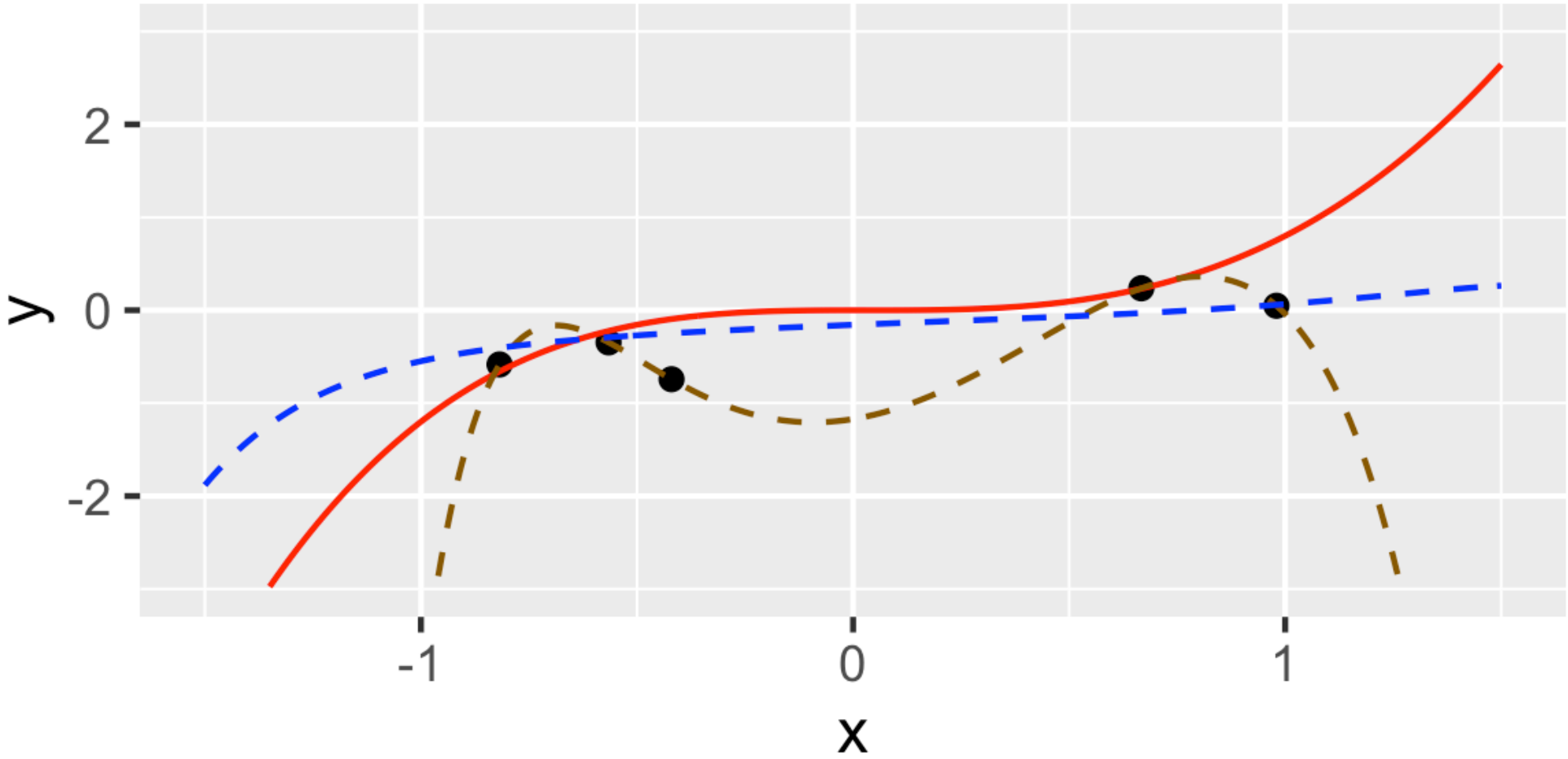
$$C(\theta, D) = C_0(\theta, D) + \lambda * R(\theta)$$

Strength of regularization

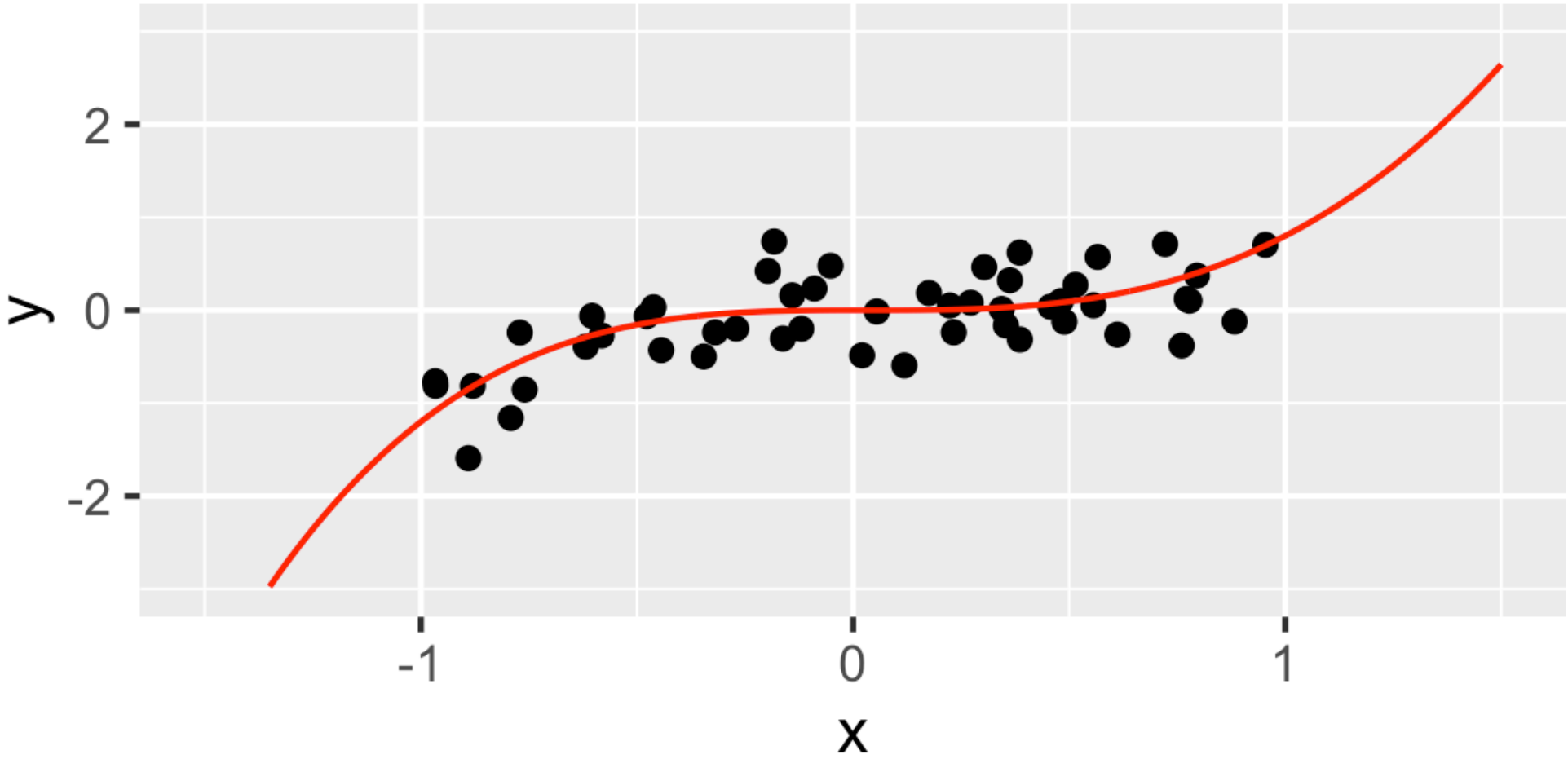
Regularization term – e.g., L_2:

$$R(\theta) = \sum \left(||\theta_i||^2 \right)$$

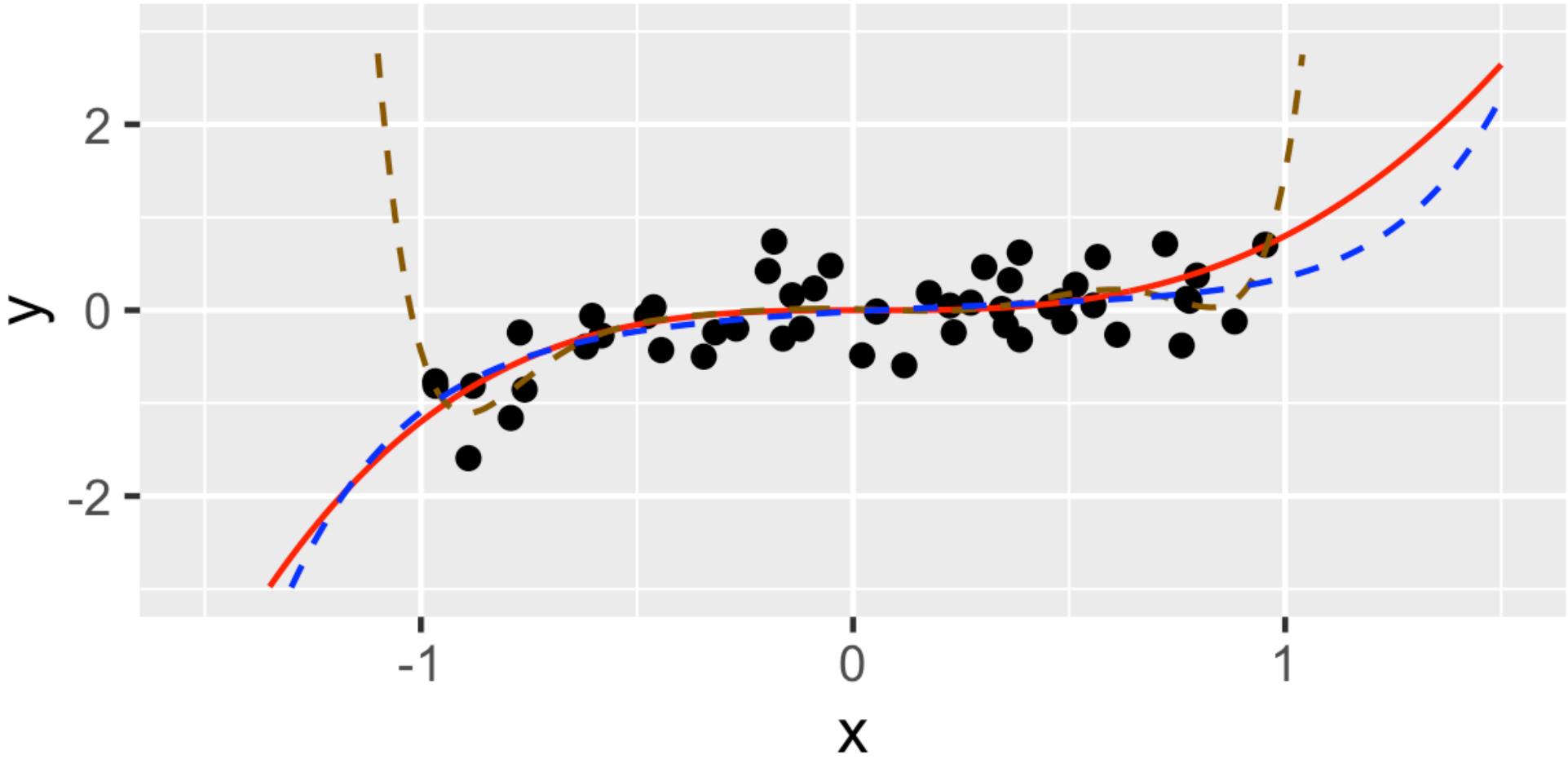
Regularization



Regularization

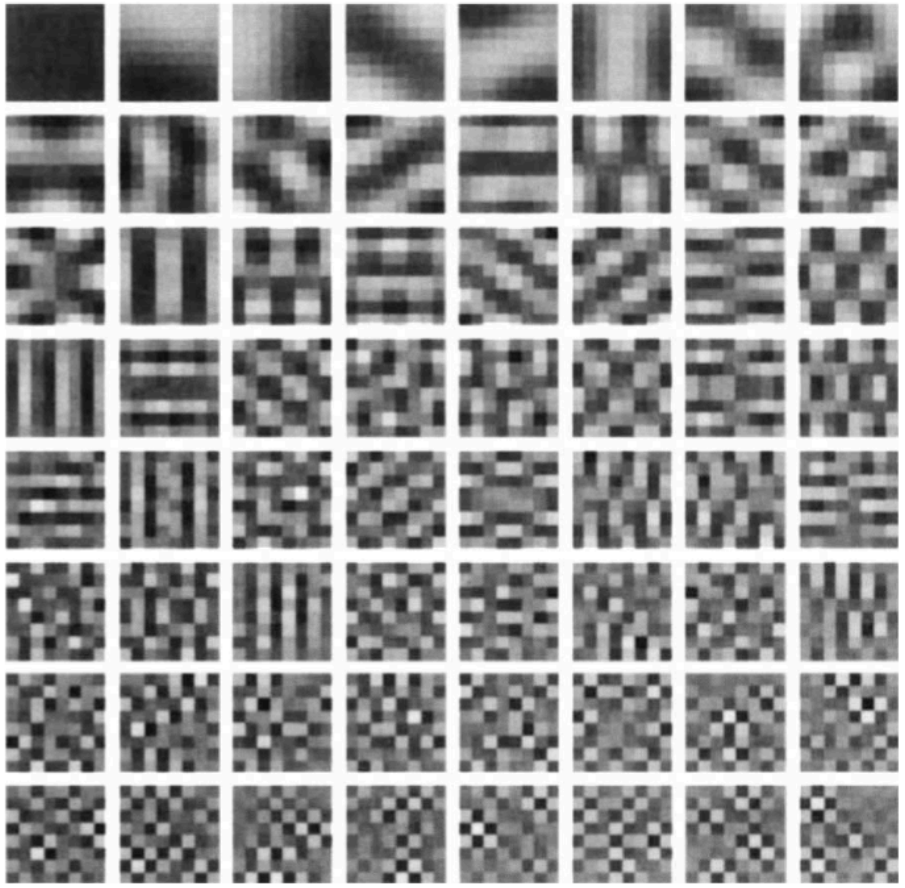


Regularization



Sparse coding

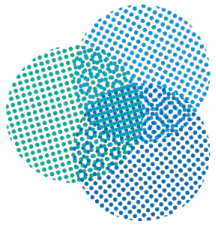
$$I(x,y) = \sum_i a_i \phi_i(x,y)$$



Olshousen & Fields (1996)

- This lets you recreate all natural images very well
- But each image requires contributions from every basis image
- Inefficient if:
  - Cost of activation (neural spiking)
  - Composition of features

Sparse coding



Sparse coding: Imposing an additional cost that is related to the magnitude of activated units

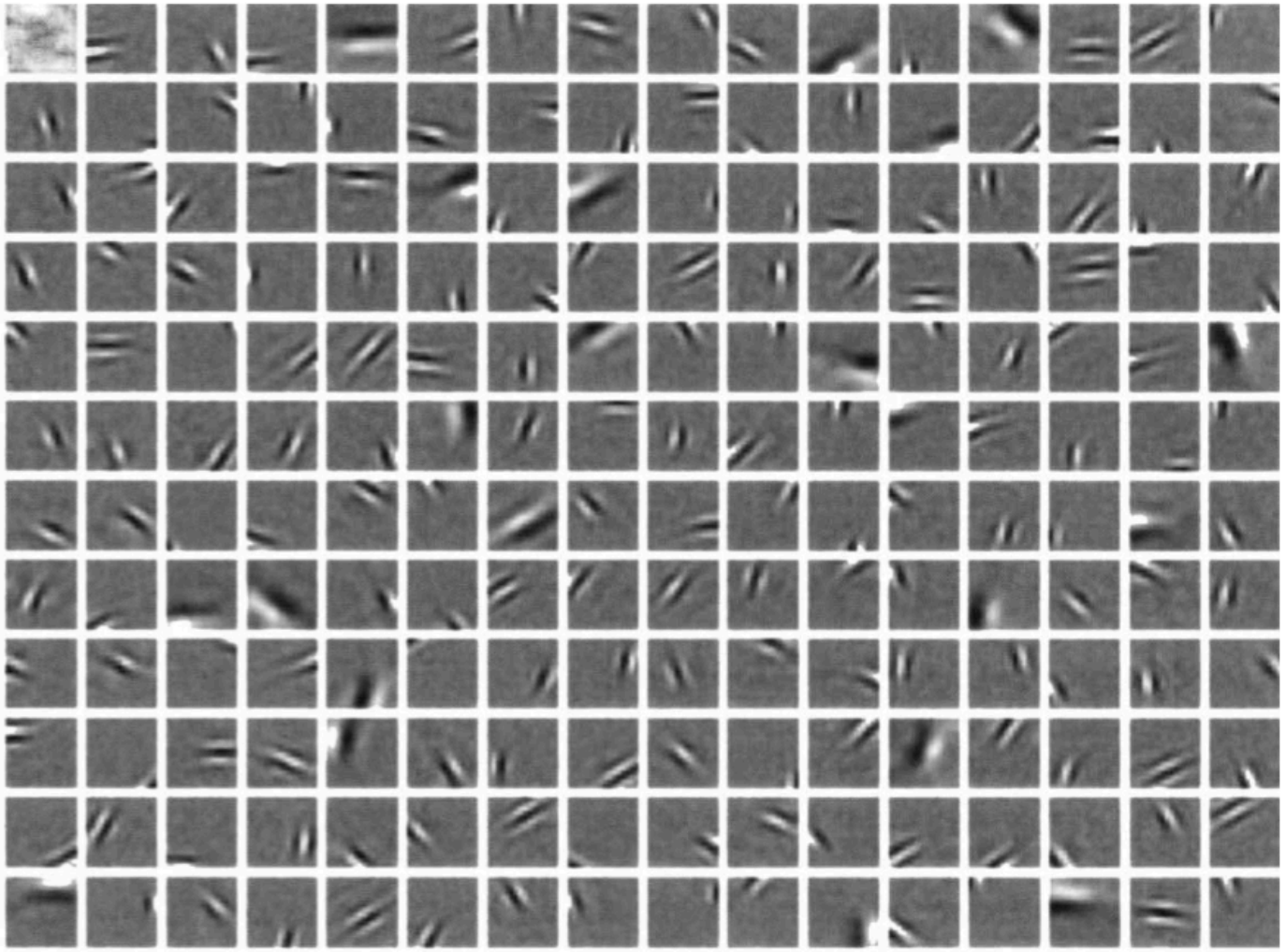
$$C(\theta, D) = C_0(\theta, D) + \lambda * S(A)$$

L_1 sparsity

Log-penalty

$$S(A) = \sum_{1 \ldots N} abs(a_i) \qquad S(A) = \sum_{1 \ldots N} log(1 + a_i^2)$$

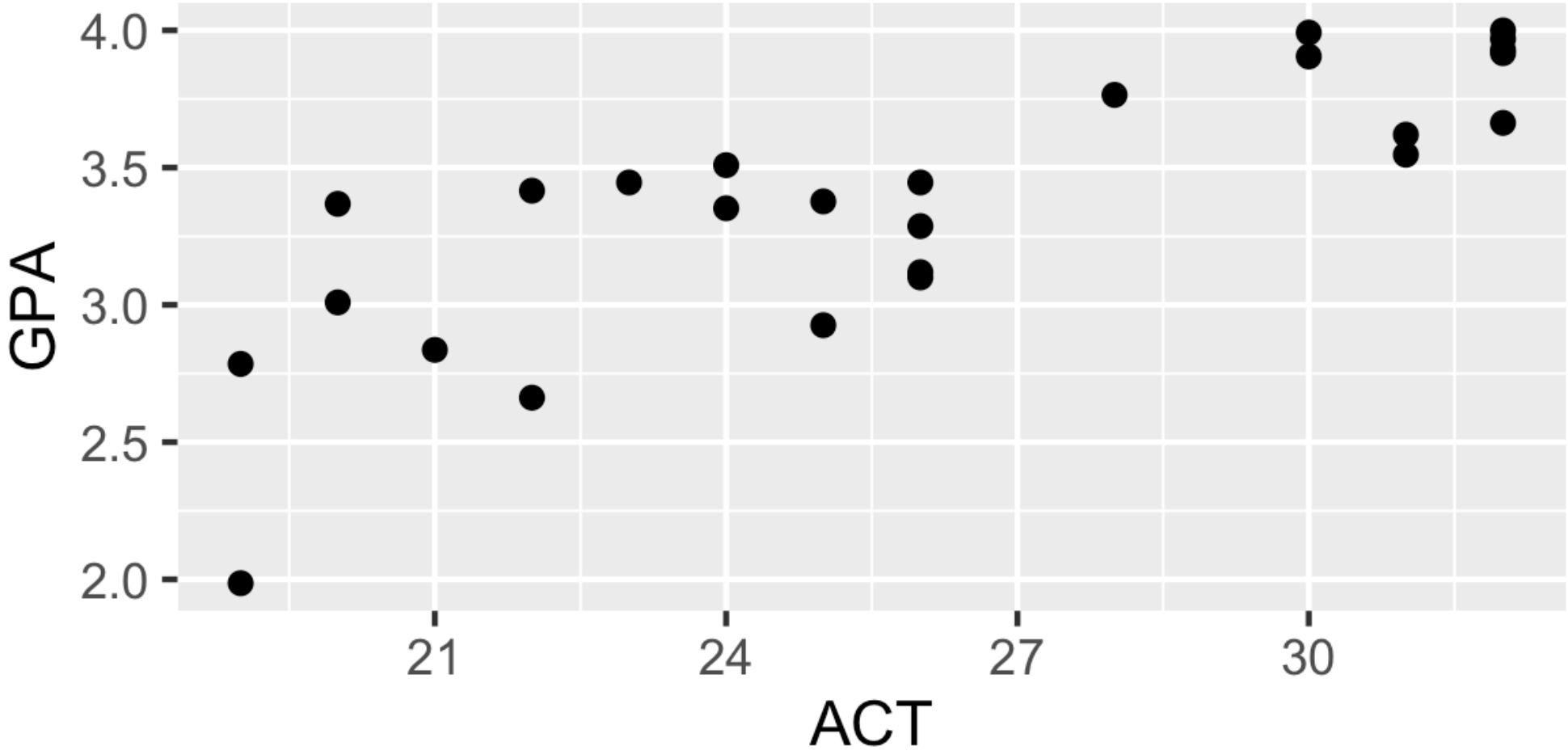
Sparse coding



Olshousen & Fields (1996)

Momentum

Can we predict college GPA from ACT scores?

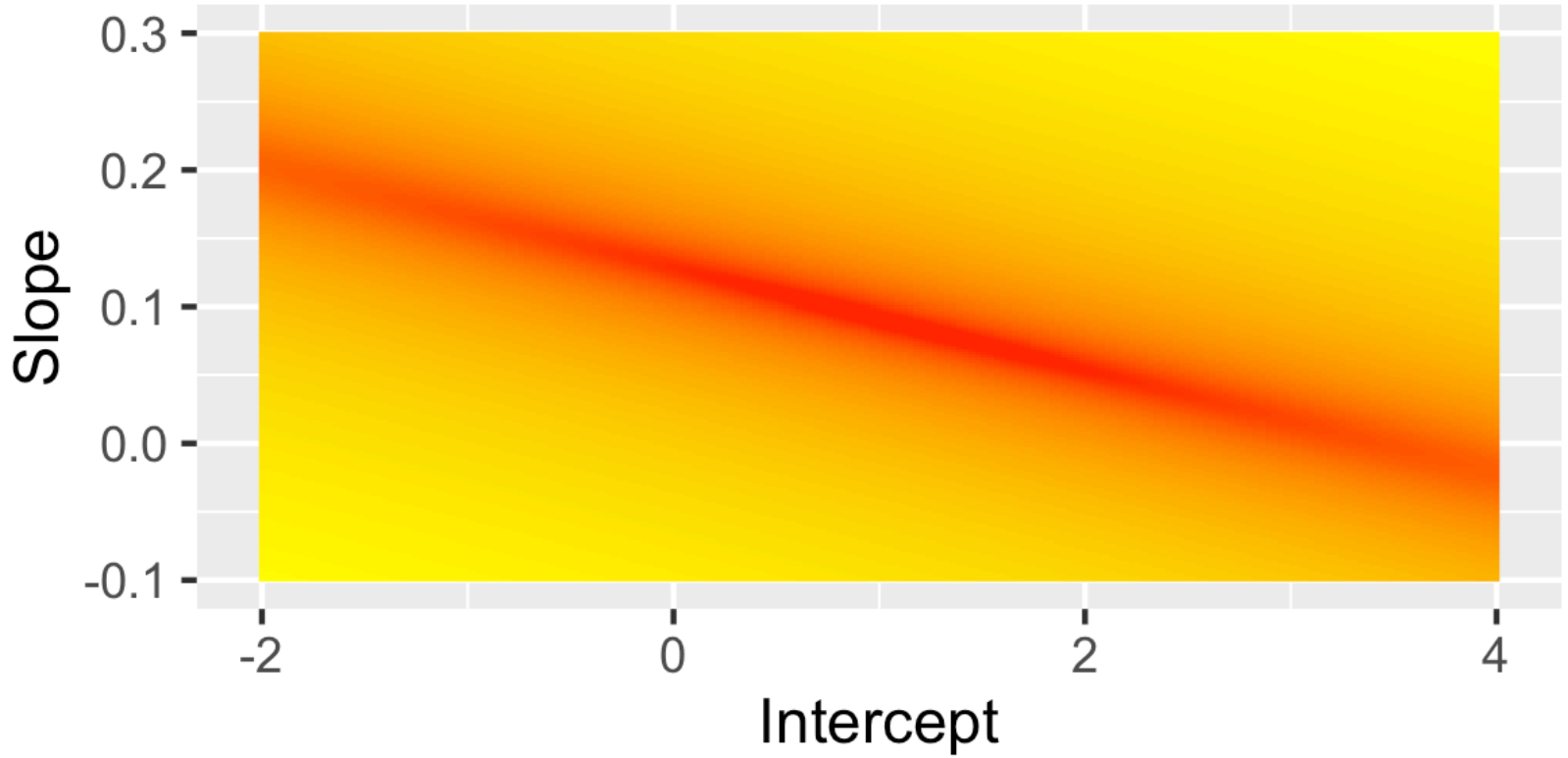


http://www.calvin.edu/~stob/data/actgpanona.csv

Momentum

$$\widehat{GPA} = b_0 + b_1 * ACT$$

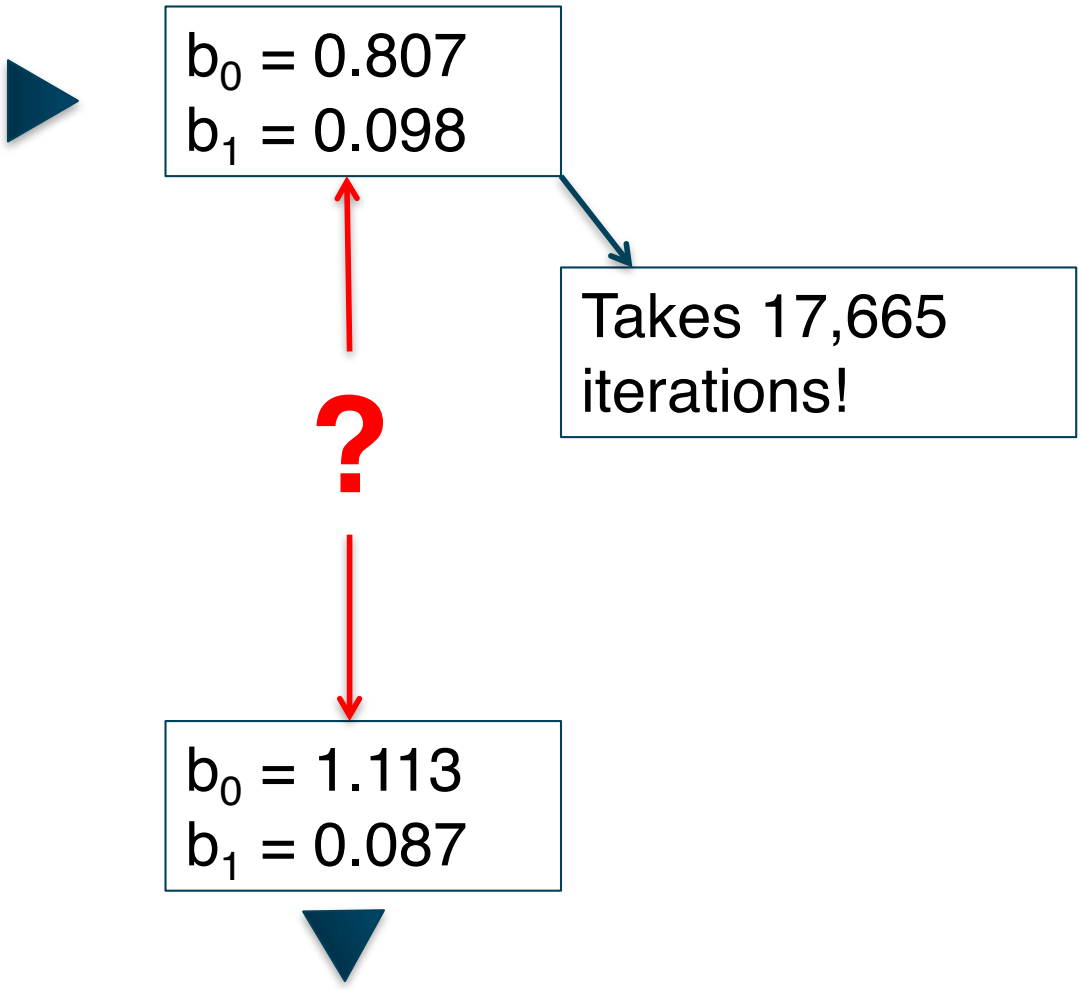
$$C([b_0, b_1]) = \sum (GPA - \widehat{GPA})^2$$



Momentum

Gradient descent

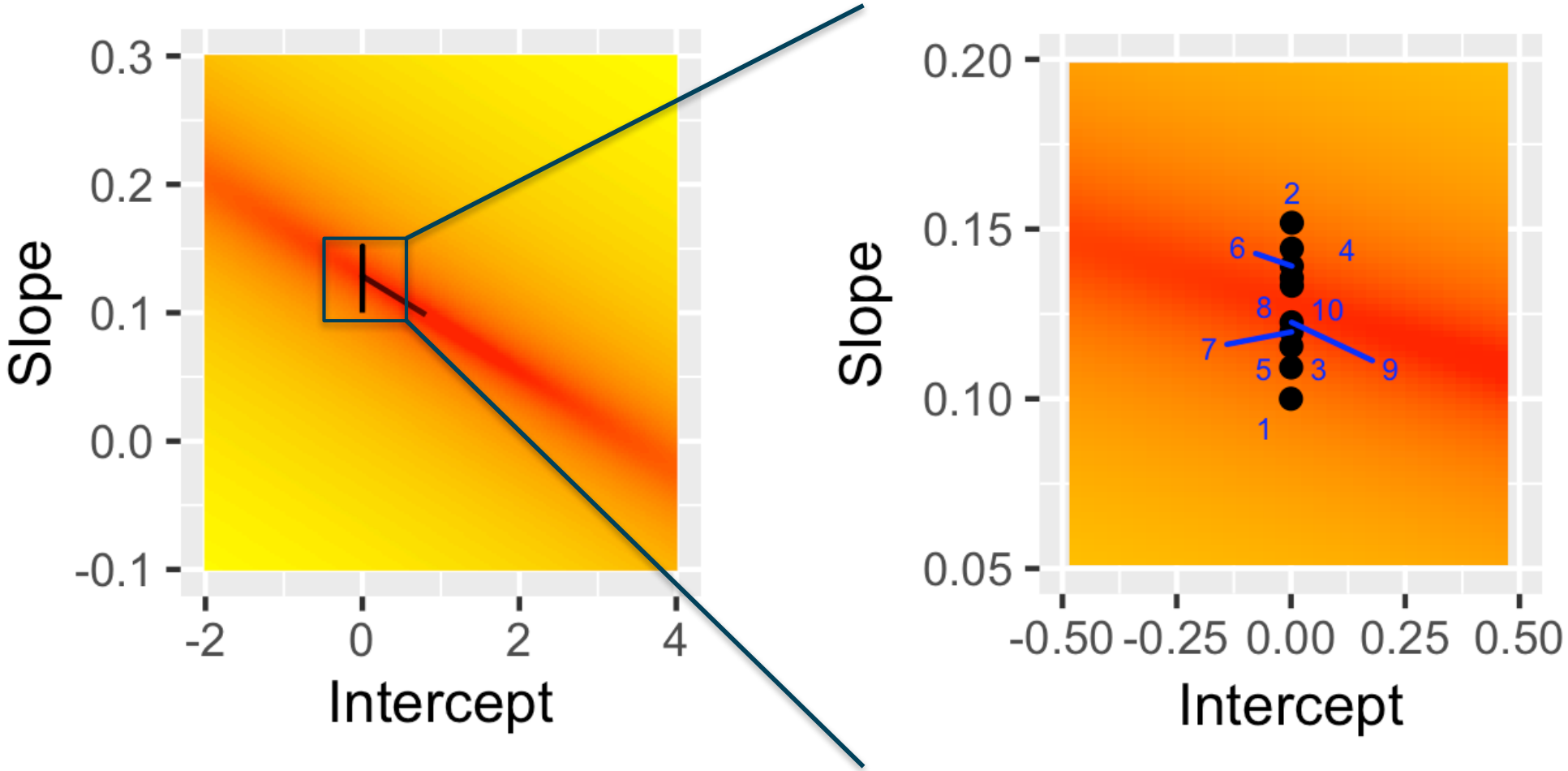
```
gpa.gradient = function(params, epsilon = .001) {
  intercept = params[1]
  slope = params[2]
  i.plus = gpa.cost(c(intercept + epsilon, slope))
  i.minus = gpa.cost(c(intercept - epsilon, slope))
  s.plus = gpa.cost(c(intercept, slope + epsilon))
  s.minus = gpa.cost(c(intercept, slope - epsilon))
  return(c((i.plus - i.minus) / (2*epsilon),
           (s.plus - s.minus) / (2*epsilon)))
}

theta = c(0,0.1)
gamma = .00005
tau = .00001

theta.vals = list(theta)

prior.cost = gpa.cost(theta)
running = TRUE
while(running) {
  grad = gpa.gradient(theta)
  theta = theta - gamma * grad
  theta.vals = append(theta.vals, list(theta))
  new.cost = gpa.cost(theta)
  if(abs(new.cost - prior.cost) < tau) {
    running = FALSE
  } else {
    prior.cost = new.cost
  }
}
writeLines(paste("b0: ",theta[1],"\nb1: ",theta[2],sep=''))
```



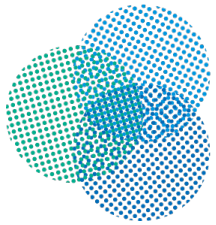
Linear regression

```
gpa.lm = lm(data=gpa.data, GPA ~ ACT)
writeLines(paste("From regression:\nb0: ",coef(gpa.lm)[1],"\nb1: ",coef(gpa.lm)[2],sep=''))
```

Momentum


Slope
0.3
0.2
0.1
0.0
-0.1
-2
0
2
4
Intercept
Slope
0.20
0.15
0.10
0.05
-0.50
-0.25
0.00
0.25
0.50
Intercept
1
2
3
4
5
6
7
8
9
10

Momentum



Momentum: Modifying gradient descent such that your next step is a combination of the gradient and the previous step

$$\Delta\theta_t = -\gamma * \nabla C(\theta_t)$$

Gradient contribution

Momentum contribution

Momentum

Gradient descent

```
gpa.gradient = function(params, epsilon = .001) {
  intercept = params[1]
  slope = params[2]
  i.plus = gpa.cost(c(intercept + epsilon, slope))
  i.minus = gpa.cost(c(intercept - epsilon, slope))
  s.plus = gpa.cost(c(intercept, slope + epsilon))
  s.minus = gpa.cost(c(intercept, slope - epsilon))
  return(c((i.plus - i.minus) / (2*epsilon),
           (s.plus - s.minus) / (2*epsilon)))
}

theta = c(0,0.1)
gamma = .00005
tau = .00001

theta.vals = list(theta)

prior.cost = gpa.cost(theta)
running = TRUE
while(running) {
  grad = gpa.gradient(theta)
  theta = theta - gamma * grad
  theta.vals = append(theta.vals, list(theta))
  new.cost = gpa.cost(theta)
  if(abs(new.cost - prior.cost) < tau) {
    running = FALSE
  } else {
    prior.cost = new.cost
  }
}
writeLines(paste("b0: ",theta[1],"\nb1: ",theta[2],sep=''))
```

$b_0 = 0.807$
$b_1 = 0.098$
$N = 17{,}665$

Gradient descent + momentum

```
theta = c(0,0.1)
gamma = .00005
alpha = .95
tau = .00001

theta.vals.mom = list(theta)
prior.cost.mom = gpa.cost(theta)
prior.dtheta = 0
running = TRUE
while(running) {
  grad = gpa.gradient(theta)
  dtheta = -gamma * grad + alpha * prior.dtheta
  theta = theta + dtheta
  theta.vals.mom = append(theta.vals.mom, list(theta))
  new.cost = gpa.cost(theta)
  if(abs(new.cost - prior.cost) < tau) {
    running = FALSE
  } else {
    prior.cost = new.cost
    prior.dtheta = dtheta
  }
}
writeLines(paste("b0: ",theta[1],"\nb1: ",theta[2],sep=''))
```

$b_0 = 1.045$
$b_1 = 0.090$
$N = 1{,}884$

$b_0 = 1.113$
$b_1 = 0.087$

Linear regression

```
gpa.lm = lm(data=gpa.data, GPA ~ ACT)
writeLines(paste("From regression:\nb0: ",coef(gpa.lm)[1],"\nb1: ",coef(gpa.lm)[2],sep=''))
```

SUMMARY

Important terms

- Likelihood
- Maximum likelihood estimate
- Cost function
- Gradient
- Gradient descent
- Global / local minima
- Convex / non-convex functions
- Differentiable functions
- Stochastic gradient descent
- Regularization
- Sparse coding
- Momentum

QUESTIONS?