Training Models

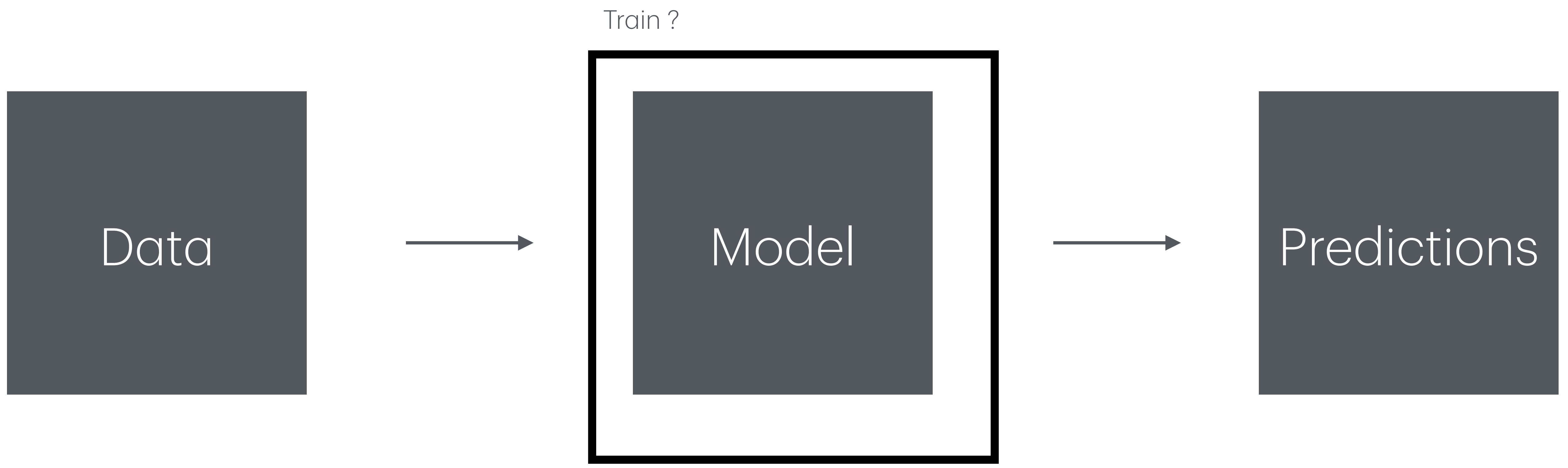
Polynomial Models

Prone to overfitting. Use this as an guide to handle real world problems

- How to detect overfitting
- Use regularization to reduce the risk of overfitting

Linear Regression

Linear based model: $\theta_0 + \theta_1 x_1 + \theta_2 x_2 \ldots$



Train model by finding models parameters that lower cost function.

Cost function is the sum of MSE of each instance prediction

Goal: Find θ vector which minimizes prediction MSE

X (Instance)	h(X) (Prediction)	Y (Target)	Error (h(X) - Y)	Error^2 (h(X) - Y)^2

MSE Error

$$\frac{1}{m}\Sigma Error_{m_i}(\,..\,)^2$$

MSE Error

Normal Equation
(Closed Form Solution)

Gradient Descent

Normal Equation: MSE Error

Option 1 uses SVD(Singular Value Decomposition) algorithm

Normal Equation (Closed Form Solution)

Option 1

$$\hat{\theta} = X^{+} \cdot y$$

X^{+} = Pseudoinverse

Option 2

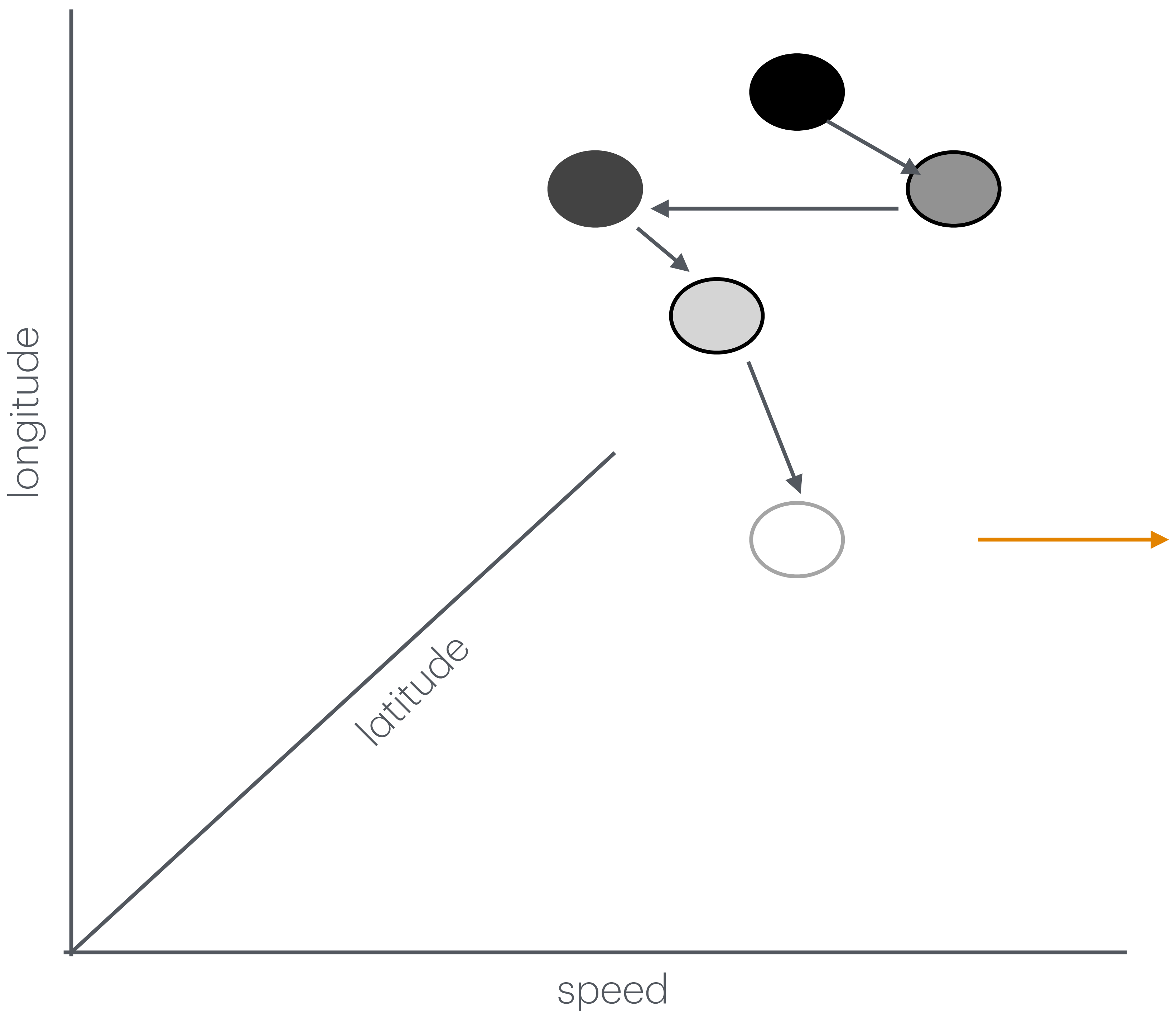
$$\hat{\theta} = (X^{T} \cdot X)^{-1} \cdot X^{T} \cdot y$$

Both options are slow for very large datasets

Gradient Descent: MSE Error

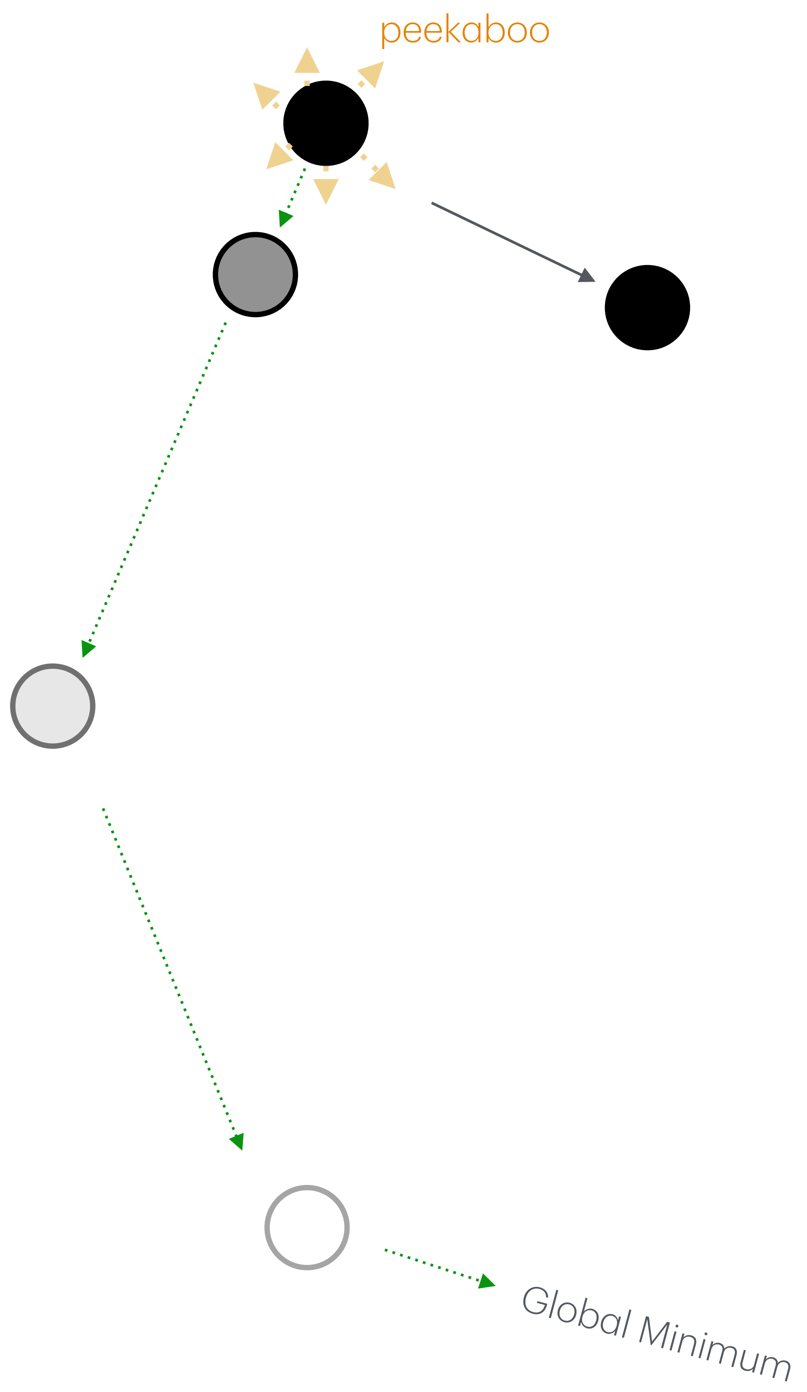
$$X_{speed} + X_{longitude} + X_{latitude}$$

$$\theta_{speed} + \theta_{longitude} + \theta_{latitude}$$ → Find solution which minimizes error



→ Global Minimum

Gradient Descent: MSE Error



Batch Gradient Descent

$$X = X_{i_{speed}} + X_{i_{longitude}} X_{i_{latitude}}$$

$$\hat{\theta} = \theta_{speed} + \theta_{longitude} + \theta_{latitude}$$

i	speed	longitude	latitude	Y
0				
1				

⋮

$$MSE(X_i, \hat{\theta})$$

$$\nabla = \frac{\partial}{\partial \theta_{speed}} + \frac{\partial}{\partial \theta_{longitude}} + \frac{\partial}{\partial \theta_{latitude}}$$

$$\nabla MSE(X_i, \hat{\theta})$$

Gradient of MSE (Cost Function)

$$\frac{\partial}{\partial \theta_{speed}} \cdot MSE(X_i, \hat{\theta}) \longrightarrow$$

$$\frac{\partial}{\partial \theta_{longitude}} \cdot MSE(X_i, \hat{\theta}) \longrightarrow$$

$$\frac{\partial}{\partial \theta_{latitude}} \cdot MSE(X_i, \hat{\theta}) \longrightarrow$$

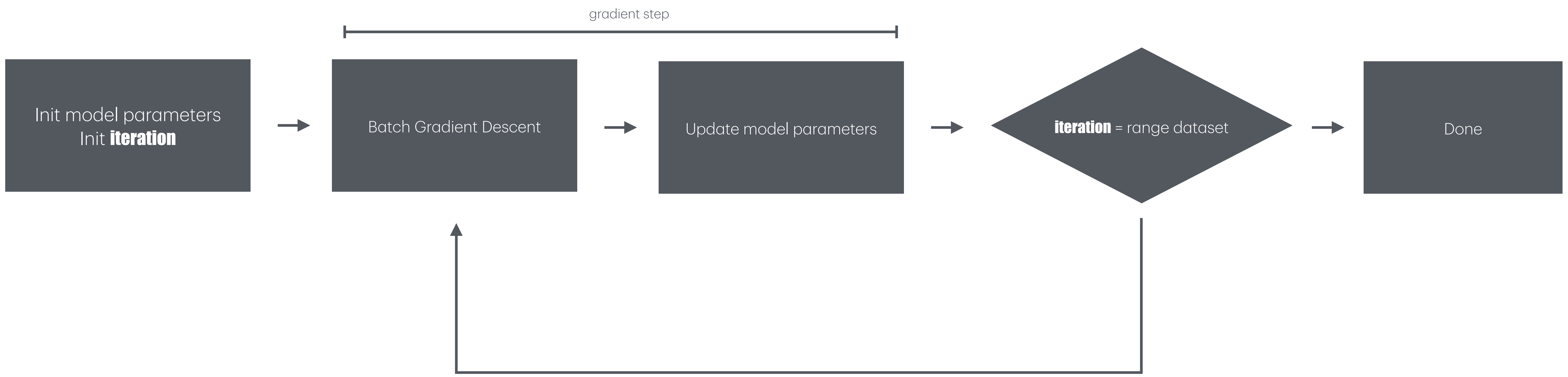
$$= \frac{2}{m} X^T (X\theta - y)$$

Requires ***entire training dataset (Batch Gradient Descent)*** *per training step,*

Vector with length equal to number of features

Scales with with feature (i.e. preferred algorithm for large number of features)

Batch Gradient Descent

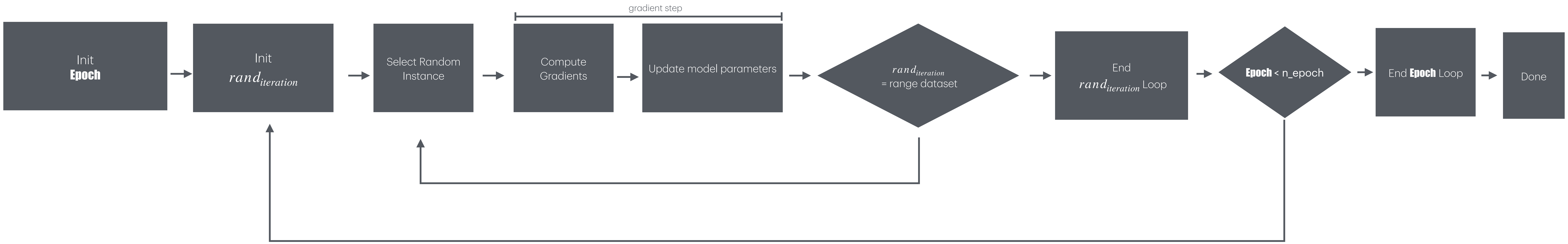


Stochastic Gradient Descent

Training instances must be
- independent
    - instances (e.g. rv) don't influence each other
- identically distributed (IID)
    - Instances (e.g. rv) share the same probability distribution

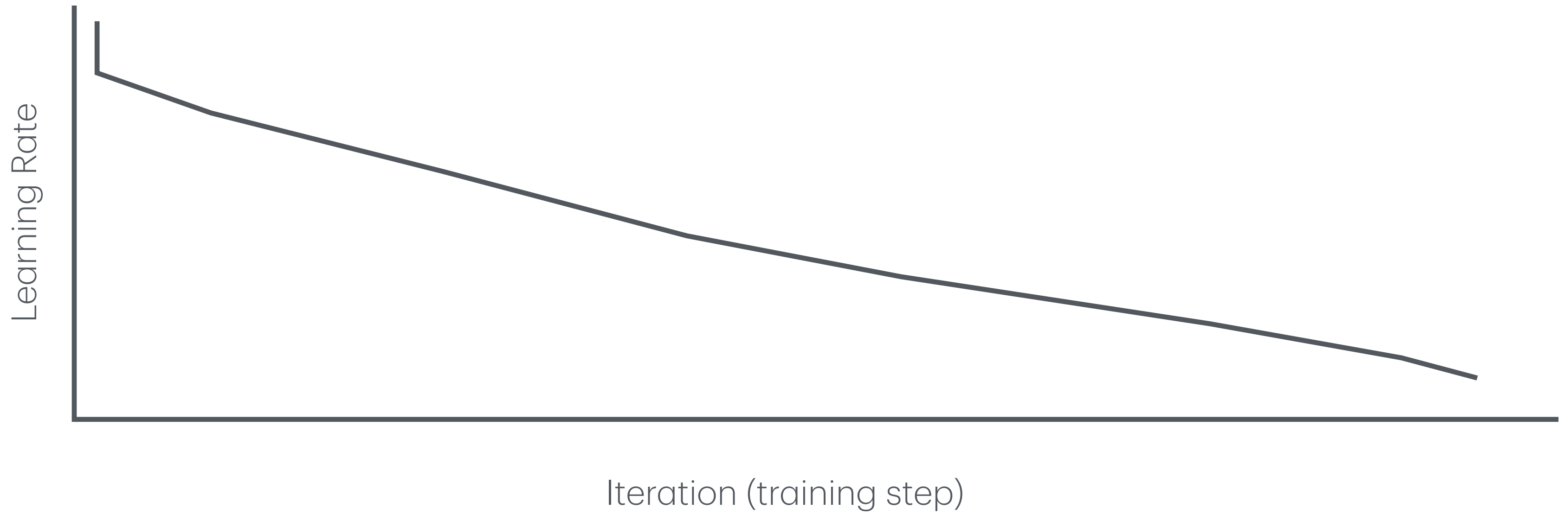
i	random i	speed	longitude	latitude	Y
0	2				
1	6				
2	1				
3	4				
4	7				
5	3				
6	5				
7					
8					

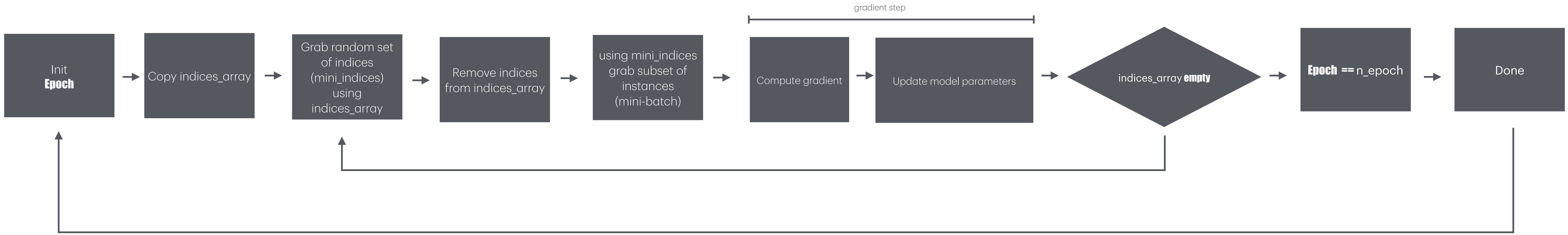
⋮



Each training step is much faster but the search for the minimum is noisy (bouncy, random)

Algorithm alone does not find true minimum. Gradually reduce learning rate during training (simulated annealing)





Algorithm provides a performance boost from hardware optimization of matrix operations (perfect for GPU matrix processing)

Prone to finding local minima if the learning schedule is bad

Data (no idea what/who generated)

Model

How complex should it be?

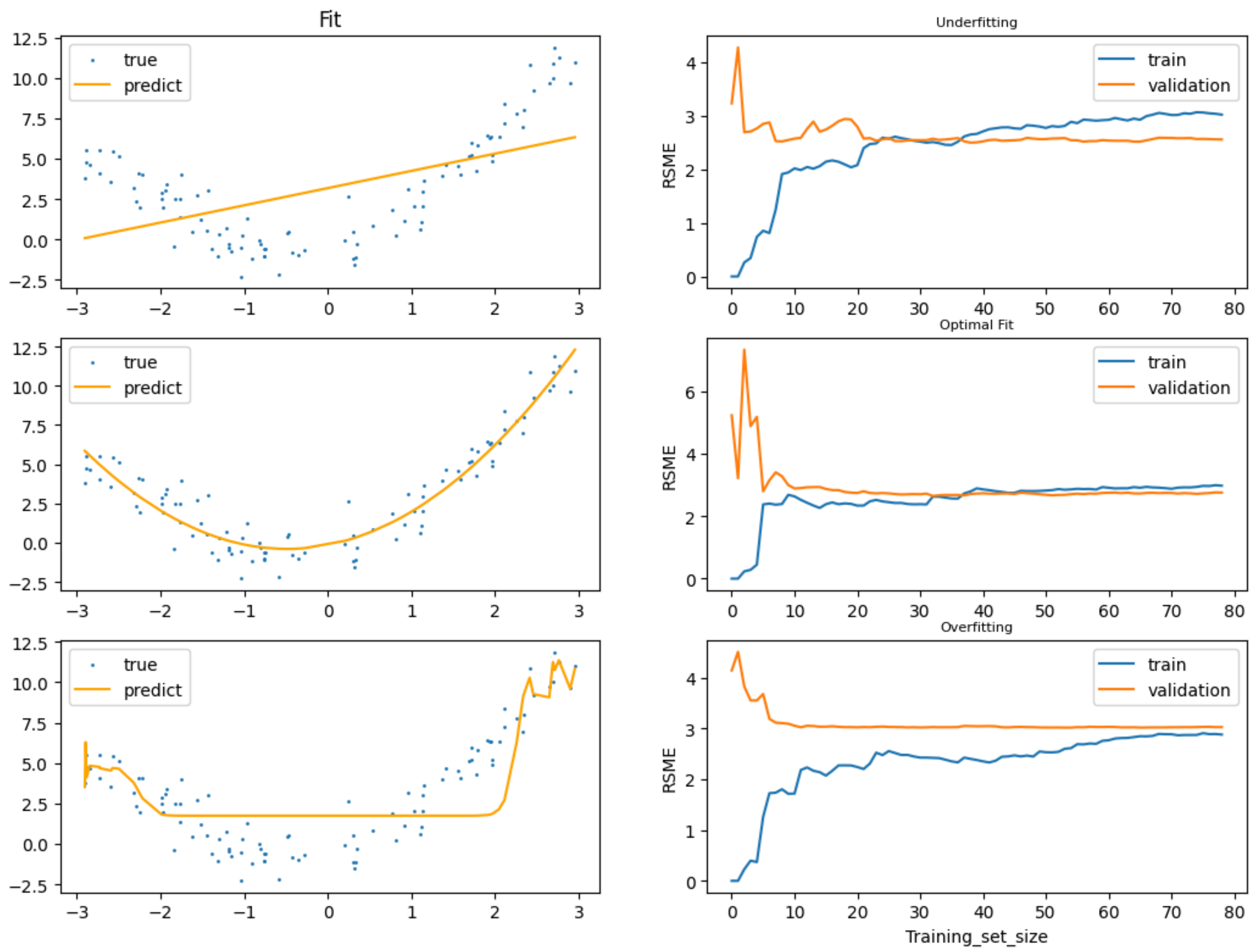
Measures overfitting and underfitting?

Old Way to measure model

- Training performs well, Test set does not
  - Overfitting
- Training and Test performs bad
  - Underfitting

New Way to measure model

- Use Learning Curves
  - Using Training Set
  - Using Validation Set



Mode Generalization Error expressed as the sum of three errors

Bias
- High bias due to underfitting (wrong fit assumptions)

Variance
- sensitivity of data
- High variance model due to high degrees polynomial

Irreducible error
- Noise of data, must be clean up (pre-processed to remove outliers)

Model Complexity

overfitting

High Variance, Low Bias

Optimal Search Region

Low Variance, High Bias

undefitting

Linear based model: $\theta_0 + \theta_1 x_1 + \theta_2 x_2 \ldots$

Constrain the weight of the model

X1 (Instance)	X2 (Instance)	h(X) (Prediction)	Y (Target)	Error (h(X) - Y)	Error^2 (h(X) - Y)^2

$MSE(\theta_i)$ + Ridge-Regularizer

$$Cost = MSE(\theta_i) + \alpha \frac{1}{2}(\theta_1^{\,2} + \theta_2^{\,2})$$

Use **ONLY** during training (updated model parameters)

$$Cost = MSE(\theta_i)$$

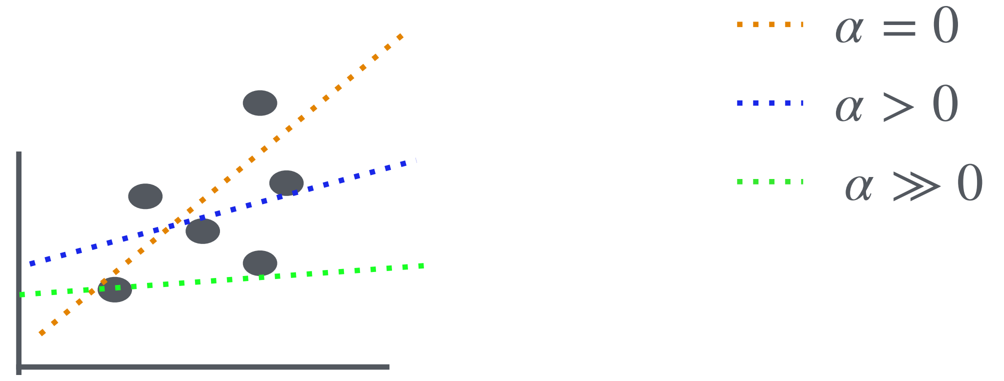
Use during evaluation

$$Cost = MSE(\theta_i) + \alpha\frac{1}{2}(\theta_1^2 + \theta_2^2)$$

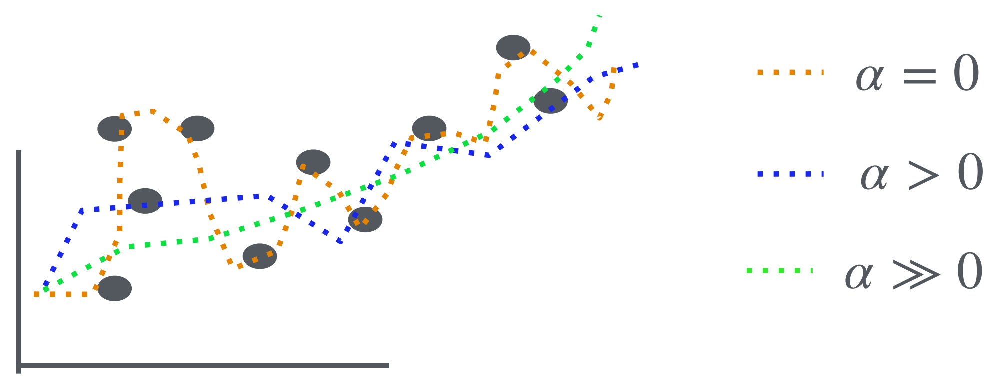
Training step works to reduce MSE

Training step works to reduce model weights

Reduces slope of 1 degree fit



Reduces sensitivity of 2 degree fit



Regularization reduces generalization error's variance(increases bias)

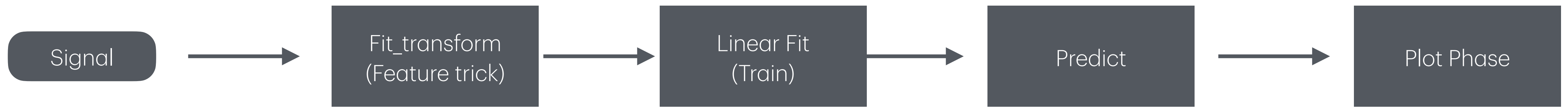
$$Cost = MSE(\theta_i) + \alpha\frac{1}{2}(\theta_1^2 + \theta_2^2)$$

Training step
works to
reduce MSE

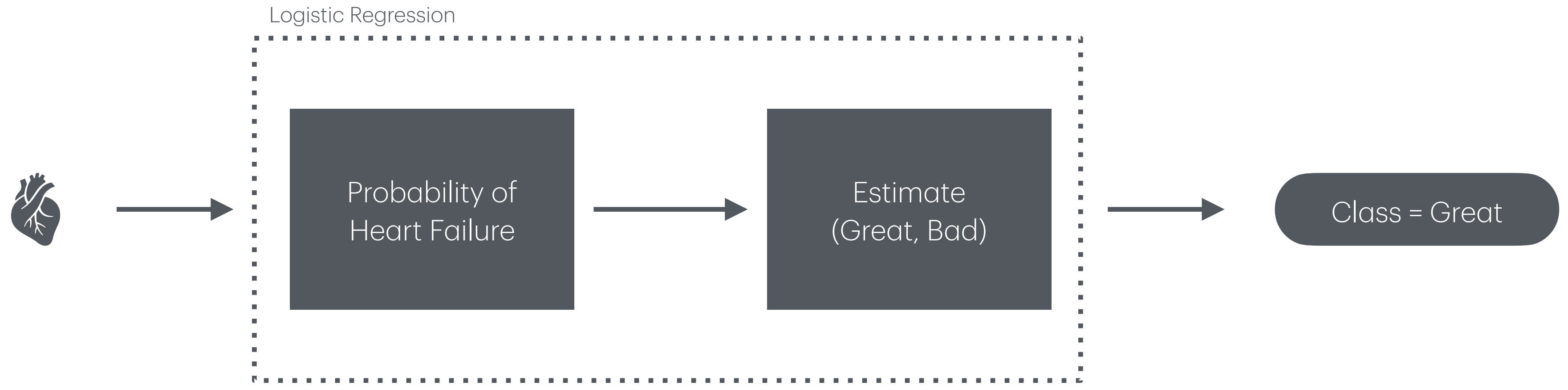
Ridge
Regularization

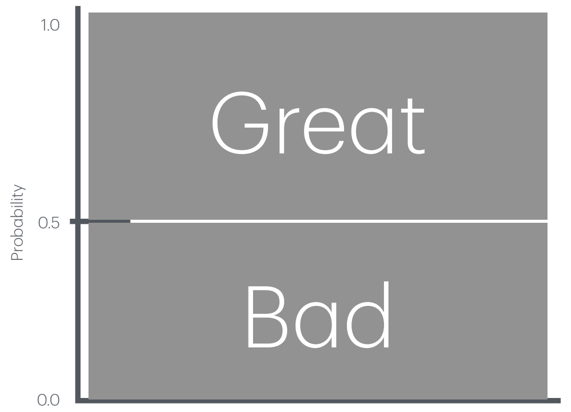
Lasso Regularization
another regularizer which
eliminates the weights(i.e. θ_i)
of least important features

Pipeline Poly

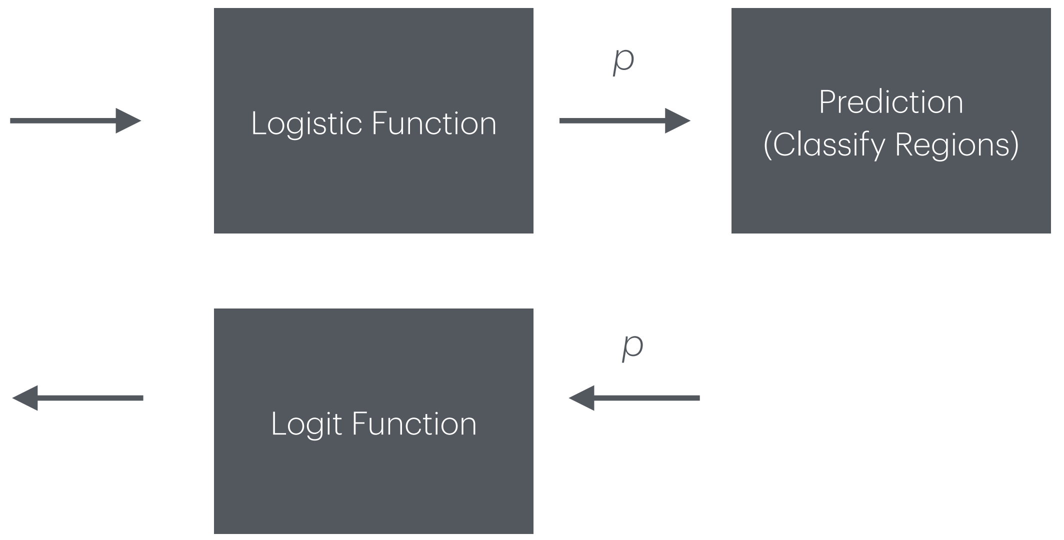


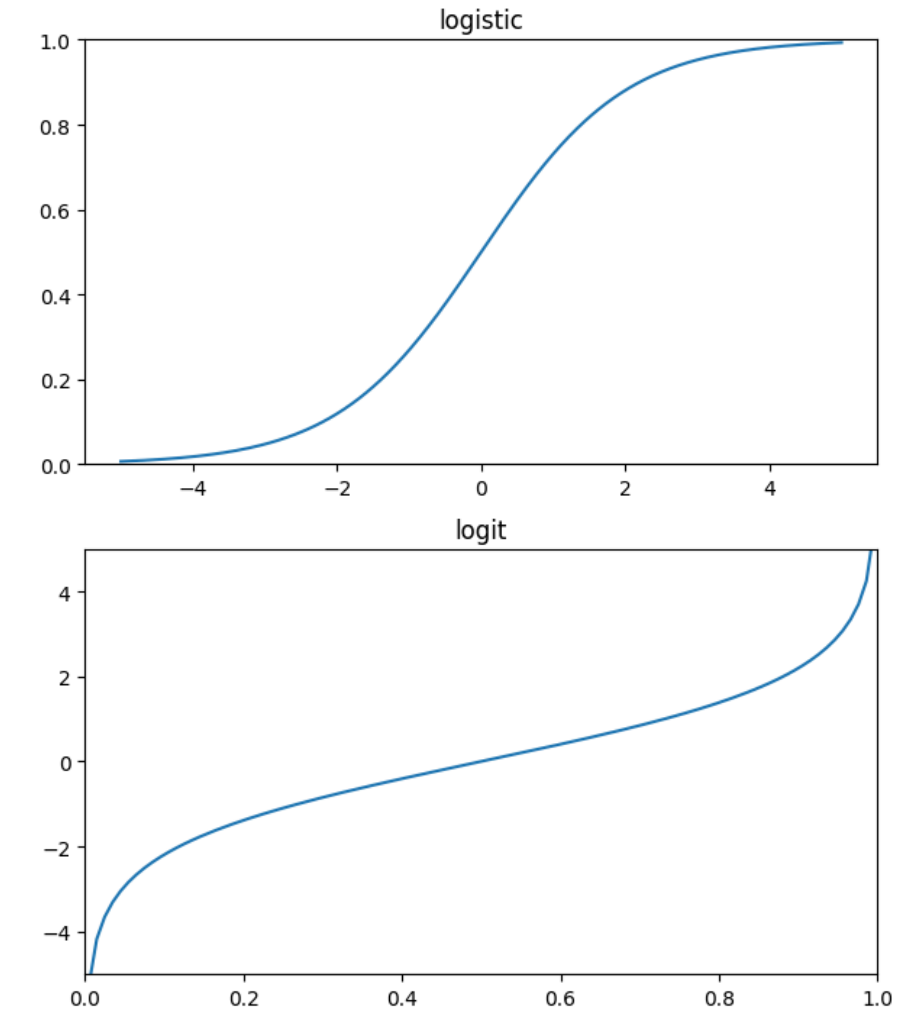
Logistic Regression





Weighted Sum: $\theta_0 + \theta_1 x_1 + \theta_2 x_2 \ldots$



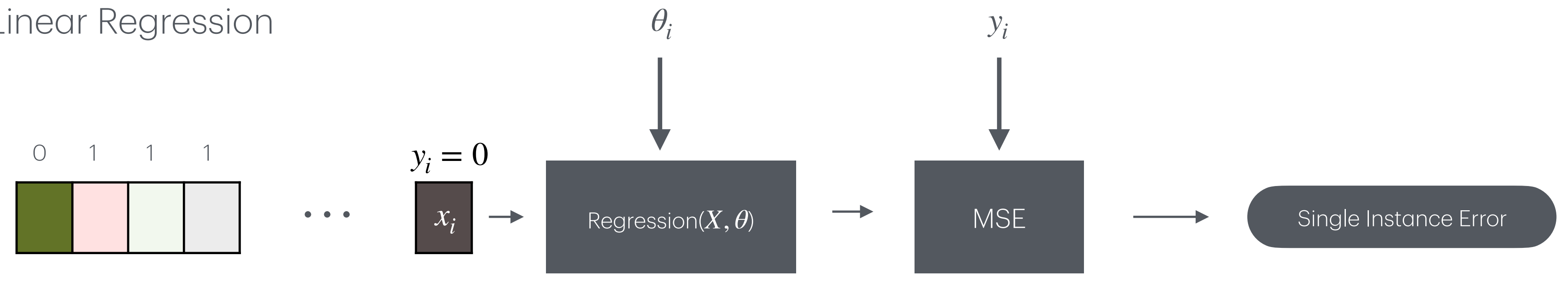


Logistic Regression: How to Train?

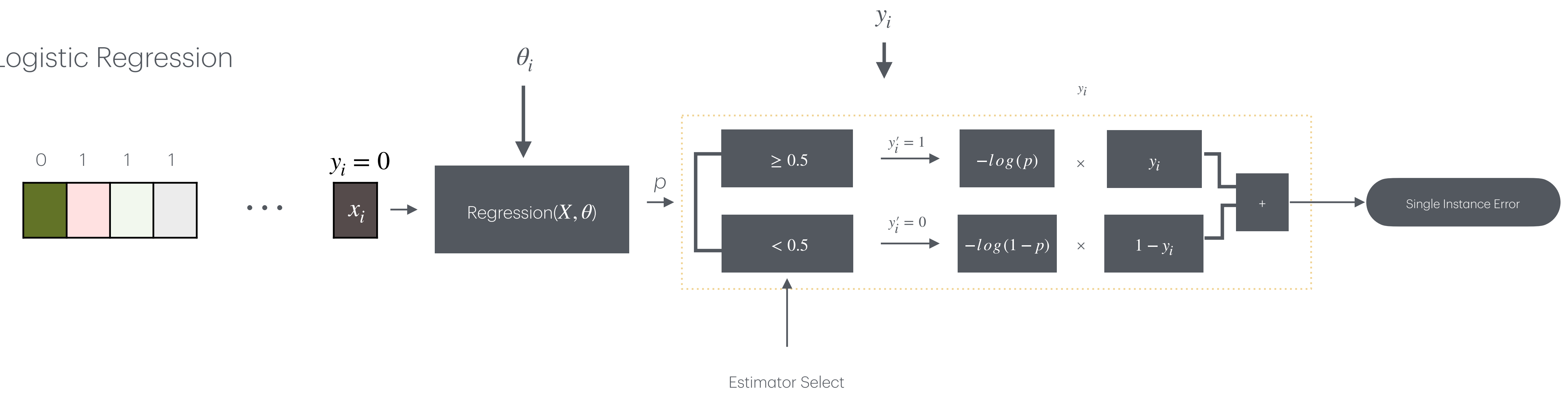
Positive Class → Parameters(θ_i) estimates high probability for positive instance

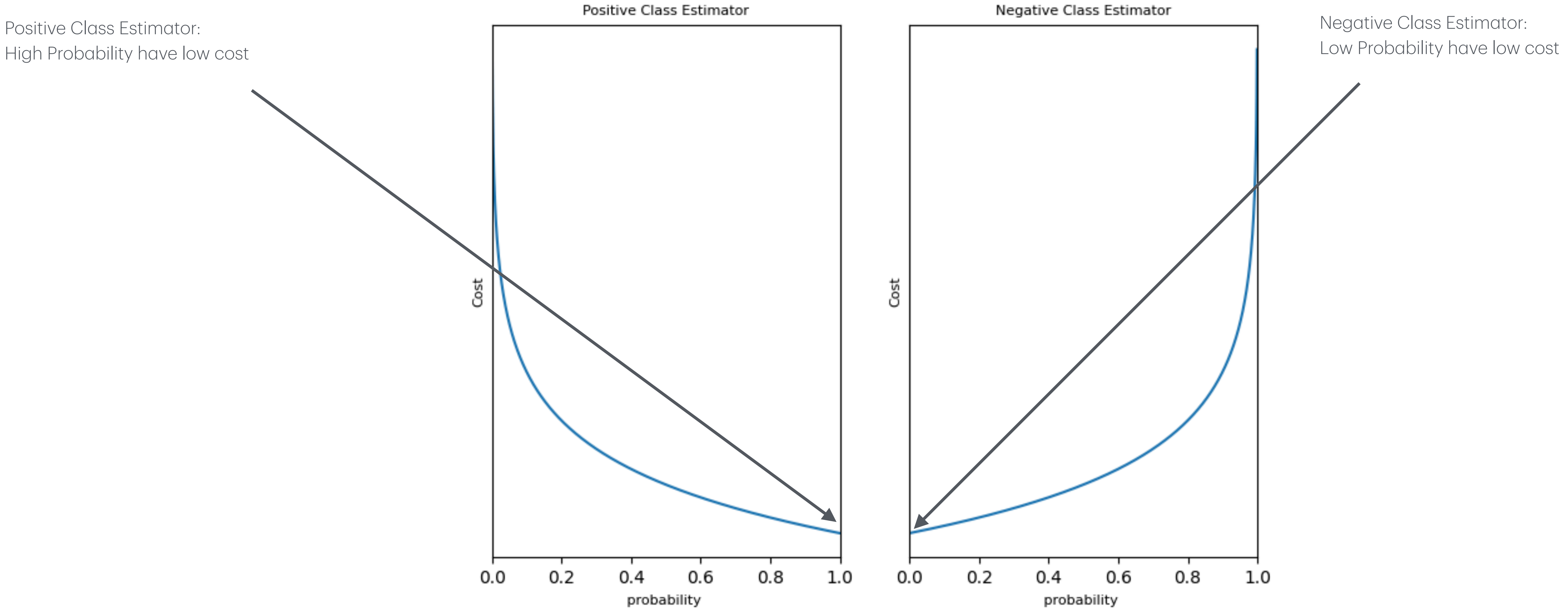
Negative Class → Parameters estimates low probability for negative instance

Linear Regression



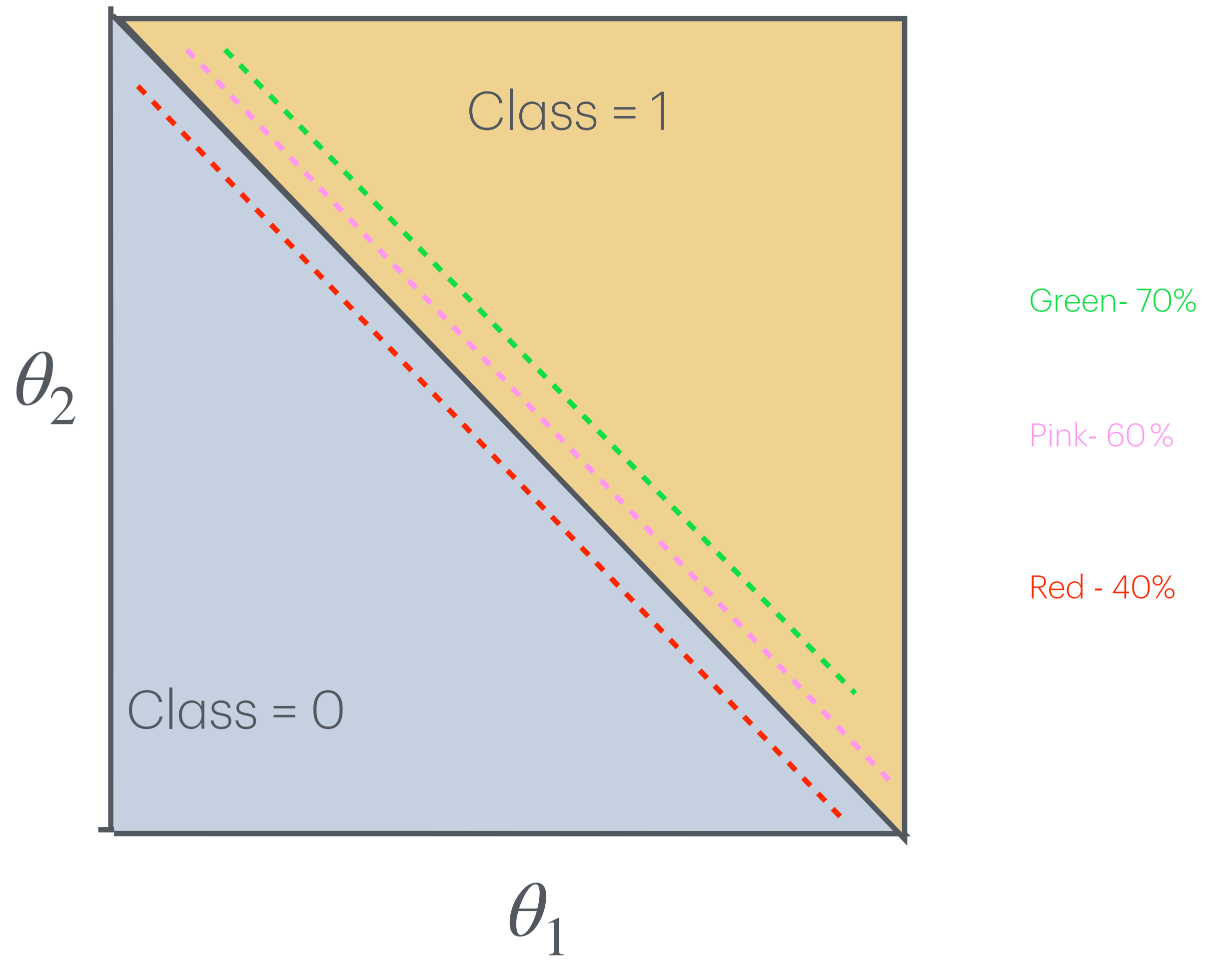
Logistic Regression





Logistic Regression: How to Train?

No closed form for Logistic Regression. Find partial derivatives of cost function and update parameters at each step (Batch, stochastic, or mini-batch)



The positive class would be the probability of instance being class=1 (e.g. prob of being male given features). All other classes are NOT male.

Softmax Regression: How to Train?

u_1	u_2	u_3	k_1	k_2	k_3	k_4
X1	X2	X3	Y1	Y2	Y3	Y4

Parameter Matrix

θ

Feature u_1

Feature k_1

Q11	Q12	Q13
Q21	Q22	Q23
Q31	Q32	Q33
Q41	Q42	Q43

<k,u>

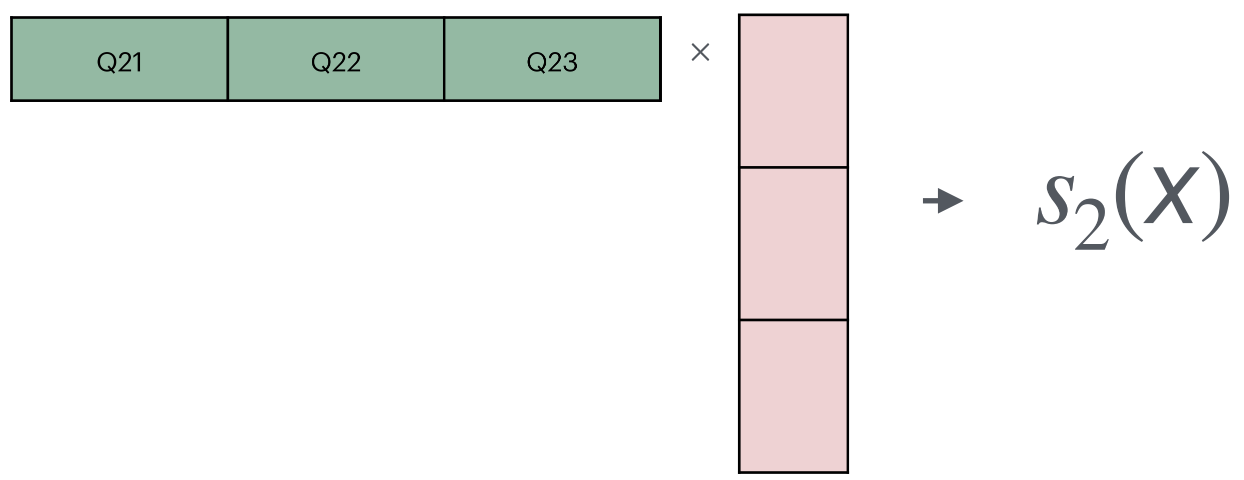
θ

Feature u_1

Feature k_1

Q11	Q12	Q13
Q21	Q22	Q23
Q31	Q32	Q33
Q41	Q42	Q43

<k,u>



$\hat{P}_k$ Estimated Probability

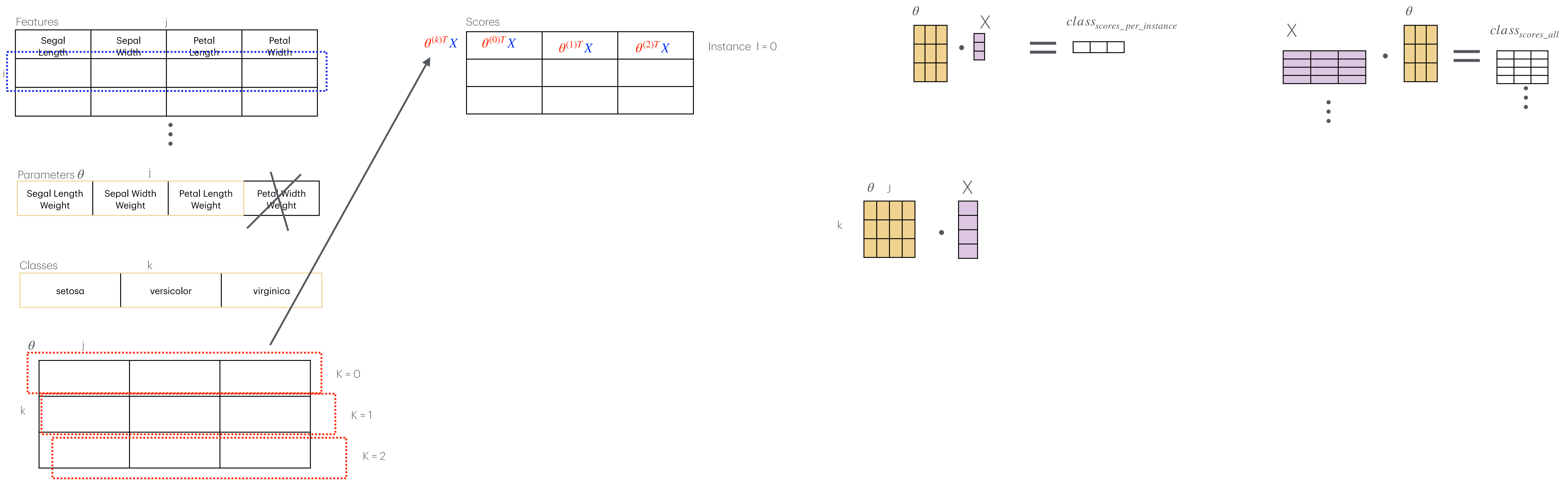
$s_1(X)$	$s_2(X)$	$s_3(X)$	$s_4(X)$
$\frac{exp(s_1(X))}{\sum_{j=1}^{K} exp(s_j(X))}$	$\frac{exp(s_2(X))}{\sum_{j=1}^{K} exp(s_j(X))}$	$\frac{exp(s_3(X))}{\sum_{j=1}^{K} exp(s_j(X))}$	$\frac{exp(s_4(X))}{\sum_{j=1}^{K} exp(s_j(X))}$
Estimated probability	Estimated probability	Estimated probability	Estimated probability

Softmax predicts only one class

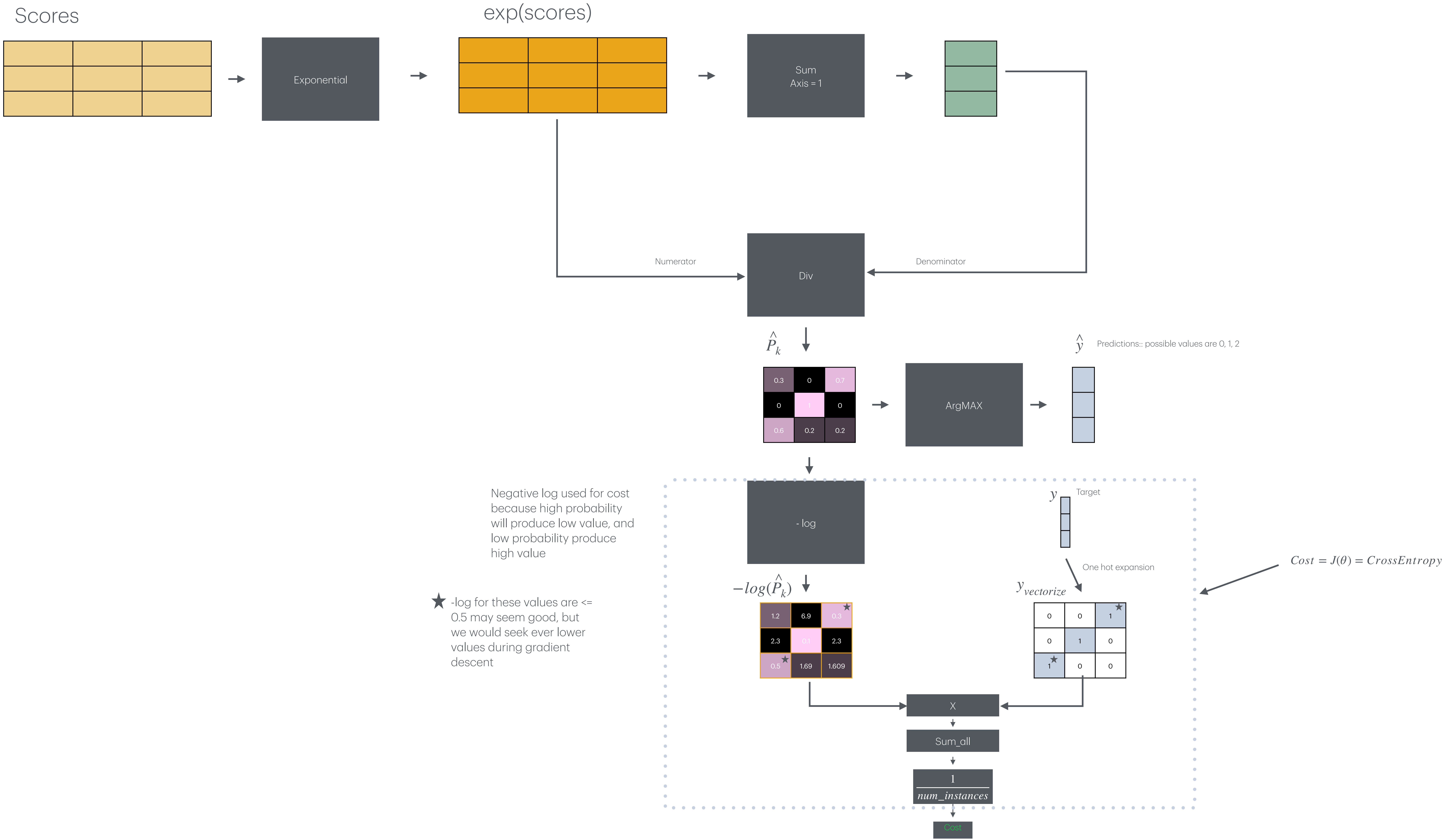
Parameter Matrix is updated during training, updating matrix during each training step that minimizes the cost function

Prediction $y' = argmax(estimated_{probabilities}) \longrightarrow$ k index

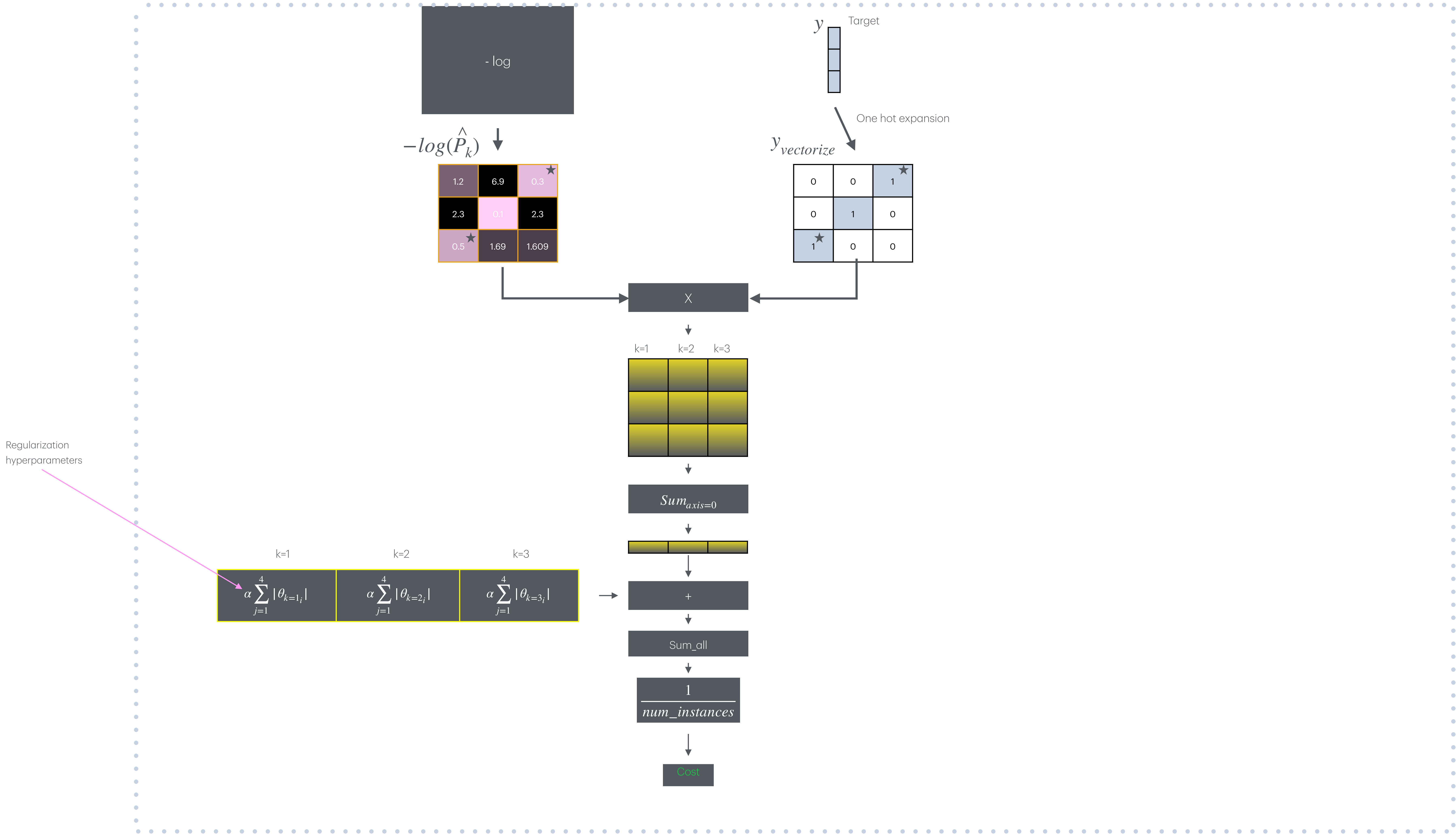
Batch Gradient Descent Example : Softmax



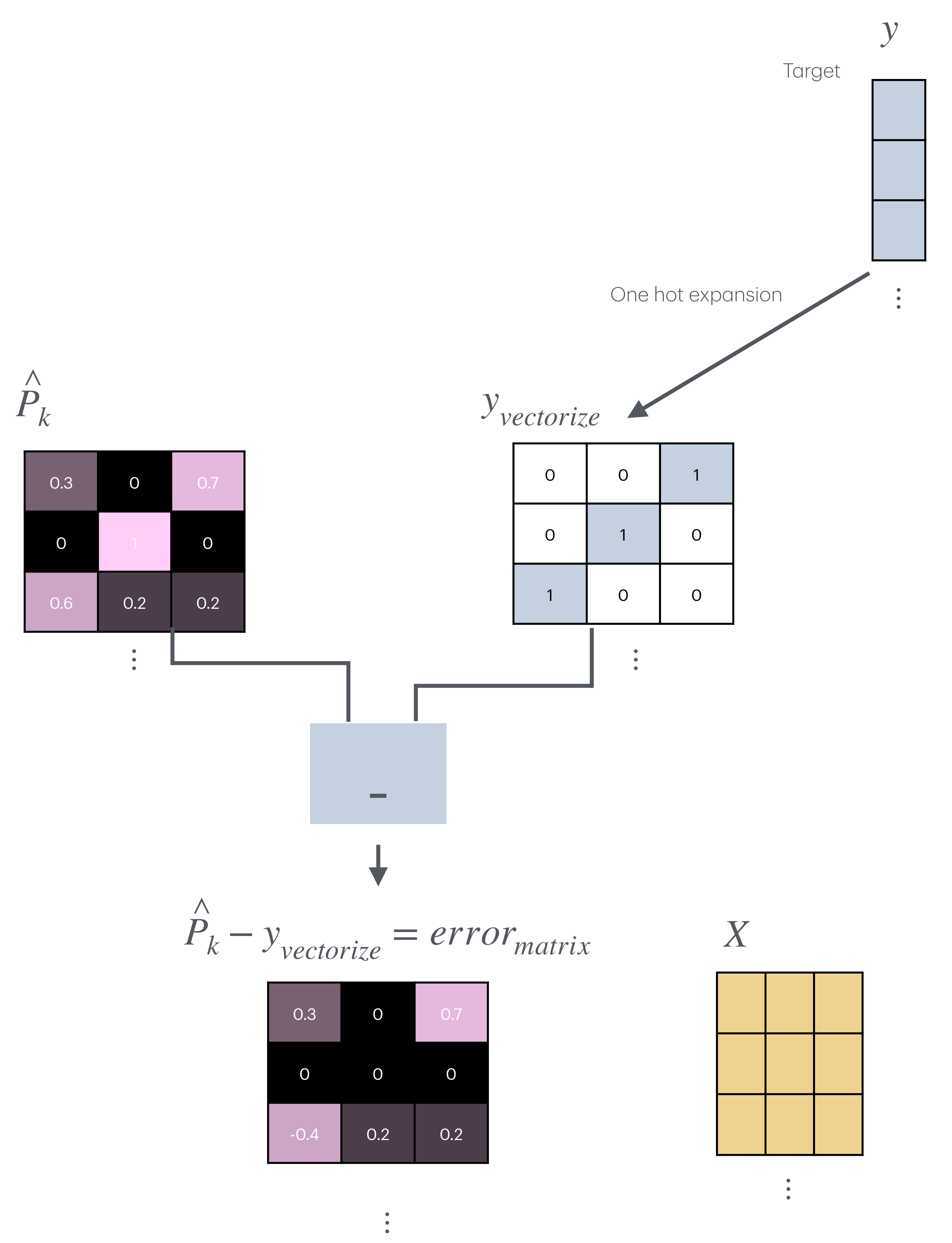
Batch Gradient Descent Example : Softmax : Scores



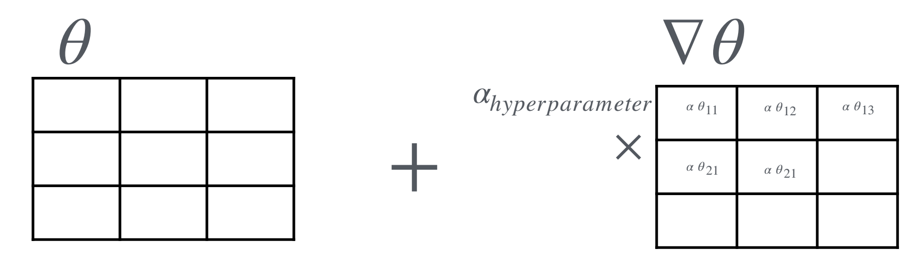
Batch Gradient Descent Example : Softmax : Scores : Regularization



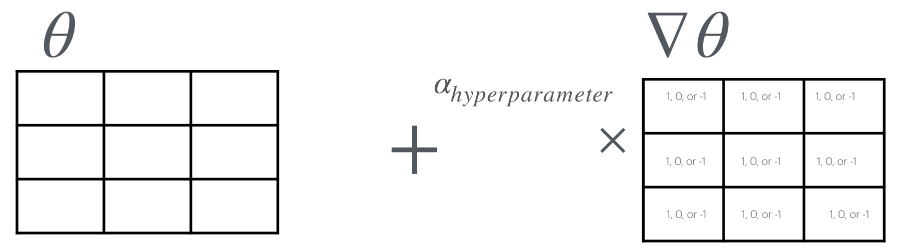
Gradient Vector Cost

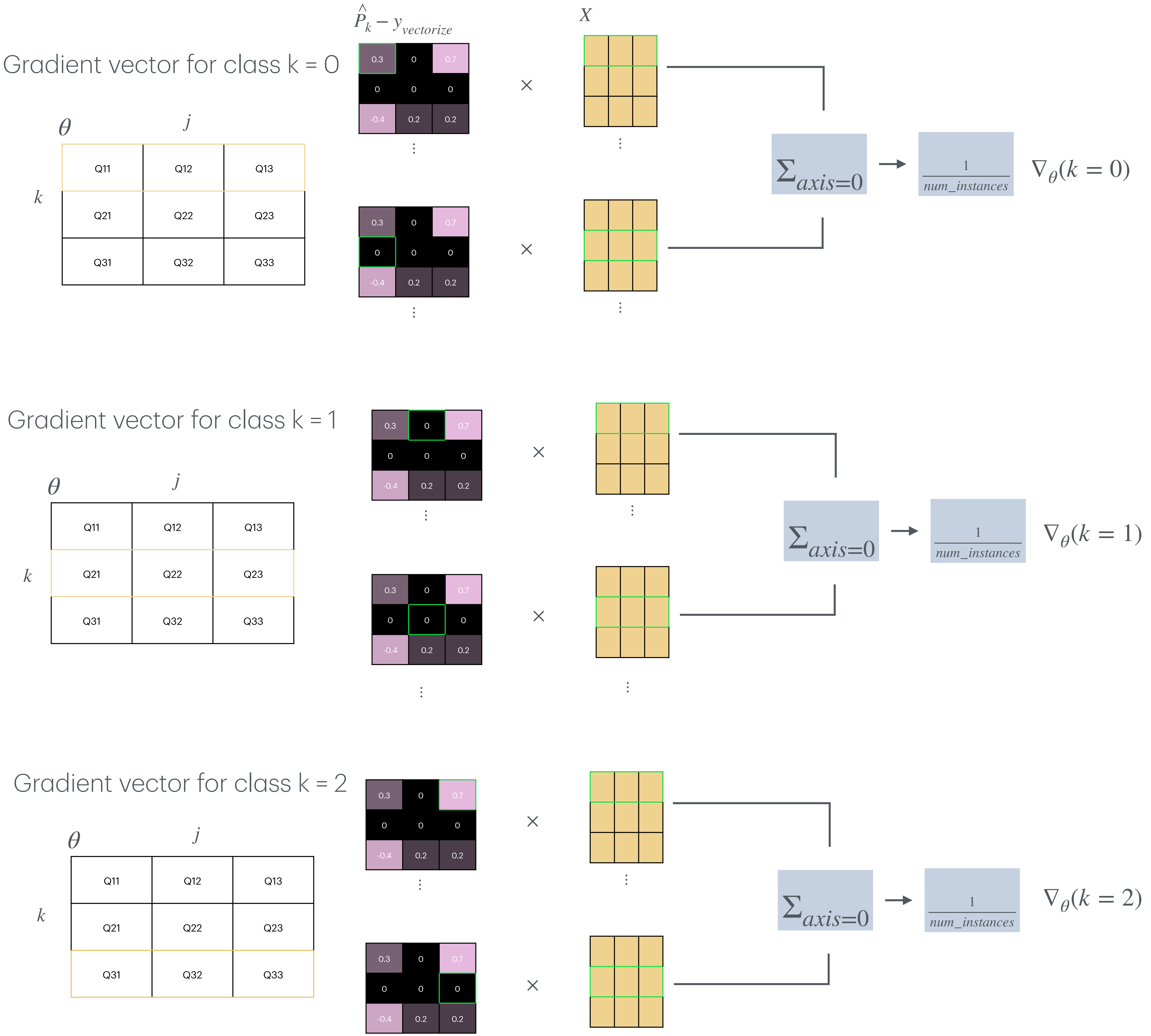


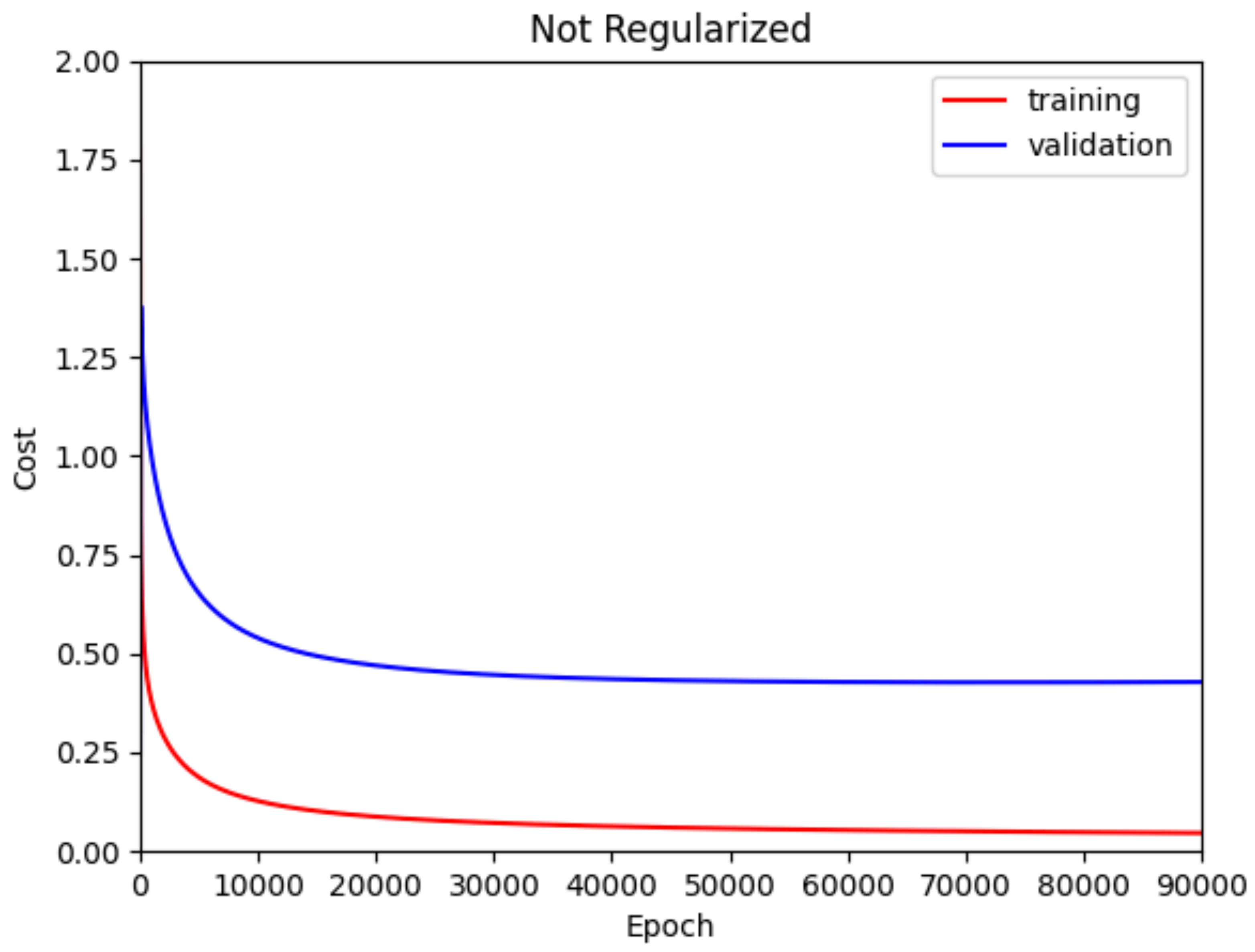
Ridge Gradient update

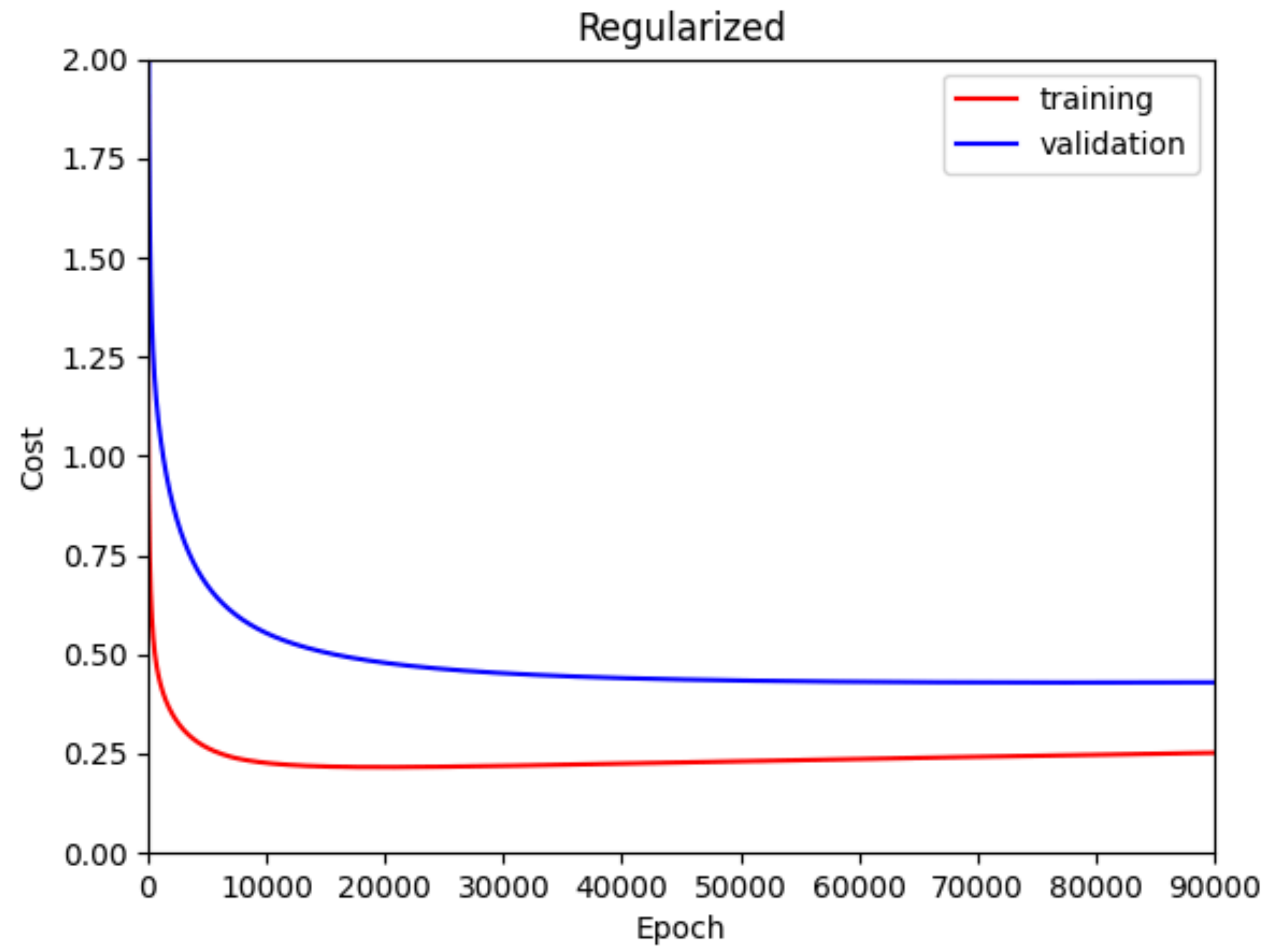


Lasso Gradient update

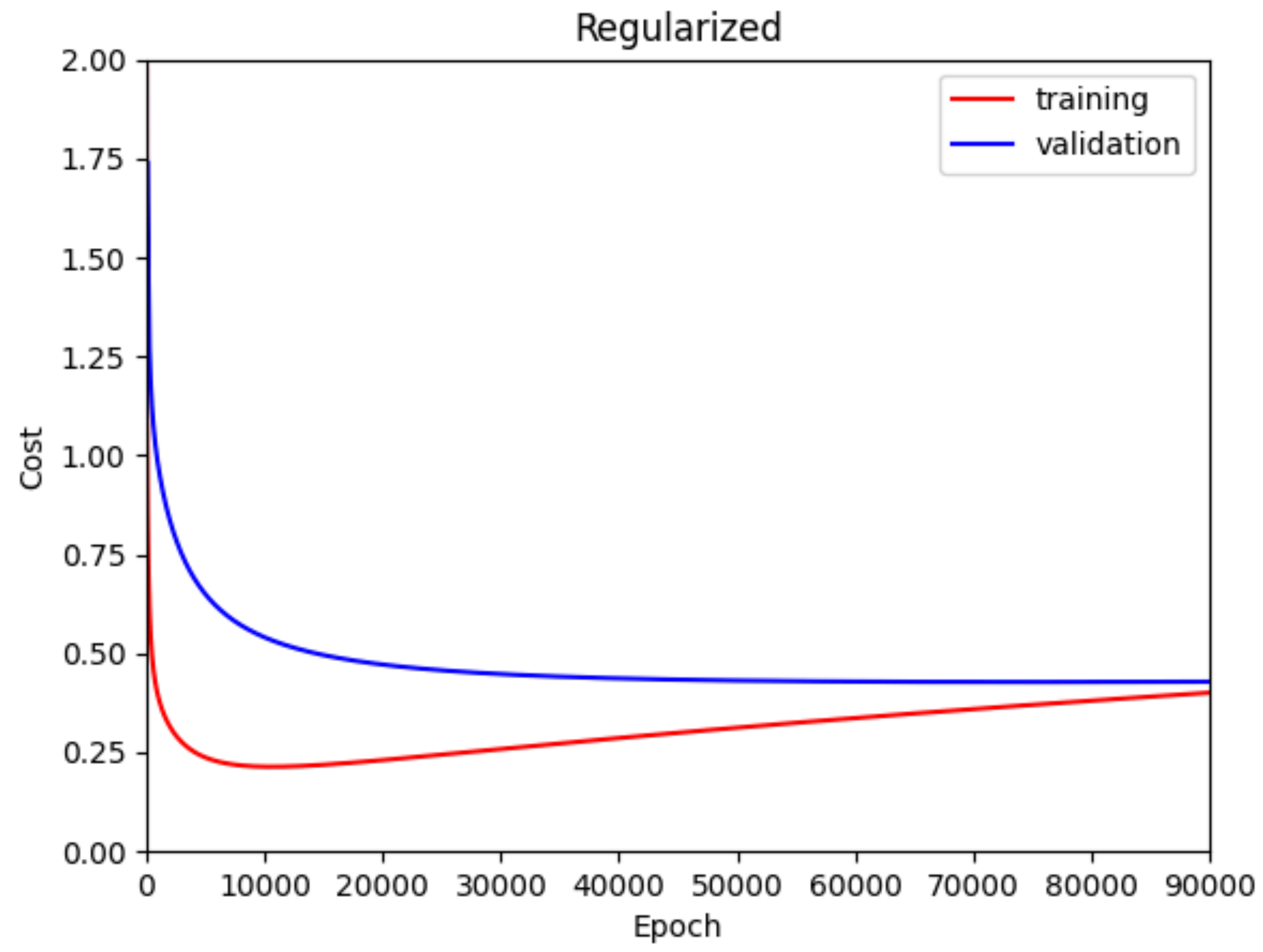





Not Regularized
training
validation
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
Cost
0
10000
20000
30000
40000
50000
60000
70000
80000
90000
Epoch


Regularized
training
validation
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
Cost
0
10000
20000
30000
40000
50000
60000
70000
80000
90000
Epoch

Lasso


Regularized
training
validation
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
Cost
0
10000
20000
30000
40000
50000
60000
70000
80000
90000
Epoch

Ridge