

- Most machine learning problems are optimization problems:
  - define a *model* (a function) class (parameterized by a set of parameters  $w$ )
  - define an *objective* (or *loss, cost*) function  $J(w)$
  - find the weights (*model*) that minimize the objective function

$$w^* = \arg \min_w J(w)$$

- If the objective function is continuous and differentiable we can use derivative of the function to find the minimum
- If the loss function is *convex*, there is a single *global minimum*
- If we are lucky (e.g., as in regression), there is an analytic solution to  $J(w) = 0$
- In most cases we use a search procedure to find the minimum

## Estimating regression parameters (again)

analytic solution

$$\text{Data: } x = \begin{bmatrix} 4 \\ 2 \end{bmatrix}, y = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

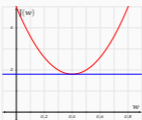
$$\text{Model: } \hat{y} = wx$$

- Squared errors

$$J(w) = (4w - 1)^2 + (2w - 2)^2 \\ = 20w^2 - 16w + 5$$

- Setting the derivative to zero:

$$\frac{dJ}{dw} = 40w - 16 = 0 \Rightarrow w = \frac{2}{5}$$



## Gradient descent for parameter estimation

- In many ML problems, we do not have a closed form solution for finding the minimum of the error function
- In these cases, we use a search strategy
- Gradient descent* is a search method for finding a minimum of a (error) function
- The general idea is to approach a minimum of the error function in small steps

$$w \leftarrow w - \eta \nabla J(w)$$

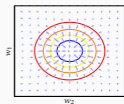
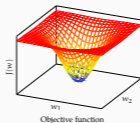
$\nabla J$  is the gradient of the loss function, it points to the direction of the maximum increase  
 $\eta$  is the *learning rate* (or *step size*)

## Gradient descent with single parameter

- For a single parameter, gradient is a one-dimensional vector
- The direction of gradient is towards the maximum increase
- We take steps proportional to  $-\nabla J(w)$
- Steeper the curve, the larger the parameter update

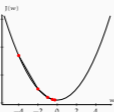


## Gradient descent with two/more parameters



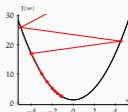
## Gradient descent: demonstration

- $J(w) = w^2 + 1, J'(w) = 2w$
- Initialize  $w = -4, \eta = 0.25$ :  
 $-0.25J'(-4) = 2$
- $w = -4 + 2 = -2$ :  
 $-0.25J'(-2) = 1$
- $w = -2 + 1 = -1$ :  
 $-0.25J'(-1) = 0.5$
- $w = -1 + 0.5 = -0.5$ :  
 $-0.25J'(-0.5) = 0.25$
- ...when do we stop?



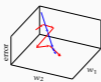
## The importance of the learning rate

- Small learning rate causes slow convergence
- Too large learning rate causes overshooting
- It is common to adapt the learning rate



## Stochastic gradient descent

- Standard (batch) gradient descent** is computationally expensive: it updates weight at every epoch
- Stochastic gradient descent (SGD)** updates weights for every training instance
- SGD may take more steps, but converges to the same solution
  - In practice a *mini-batch* is more common
  - Correct batch size is an important hyperparameter for neural networks



## Adapting learning rate

- The choice of learning rate  $\eta$  is important
  - too small: slow convergence
  - too big: overshooting - may fluctuate around the minimum, or even jump away
- The idea is to adapt the learning rate during learning
- A common trick is adding a momentum:
  - if we move in the same direction a long time, accelerate

$$\Delta w_t = \eta \nabla J(w) + \alpha \Delta w_{t-1}$$

- Other improvements include using a separate learning rate for each parameter
- There are many adaptive optimization algorithms: Adagrad, Adadelata, RMSprop, Adam, ...

## Summary

- Gradient descent is a general method for searching for the minima of a function
- We often use a mini-batch gradient descent: weights are based on a small number of training instances
- Reading: Jurafsky and Martin (2026, Section 5.6)
- Next:
  - More gradient descent & computation graphs

## References & further reading

- Jurafsky, Daniel and James H. Martin (2026). *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition with Language Models*. 3rd. Online manuscript released January 6, 2026. URL: <https://web.stanford.edu/~jurafsky/slp3/>.