Week 3

```
fd = open('file.txt', 'r')

# do something

fd.close()
```

```
with open('file.txt', 'r') as
fd:
    data = fd.read()
    # do something exciting
```

Announcements

- HW due today
  - pdf
- HW2 will posted on the Blackboard on Wednesday

Regression Line

- Equation : $Y = \beta_0 + \beta_1 X$

- β_0: intercept → value of Y when X = 0
- β_1: slope → change in Y for each unit increase in X

- How to learn those parameters?

$$\hat{\beta}_1 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2},$$

$$\hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x},$$

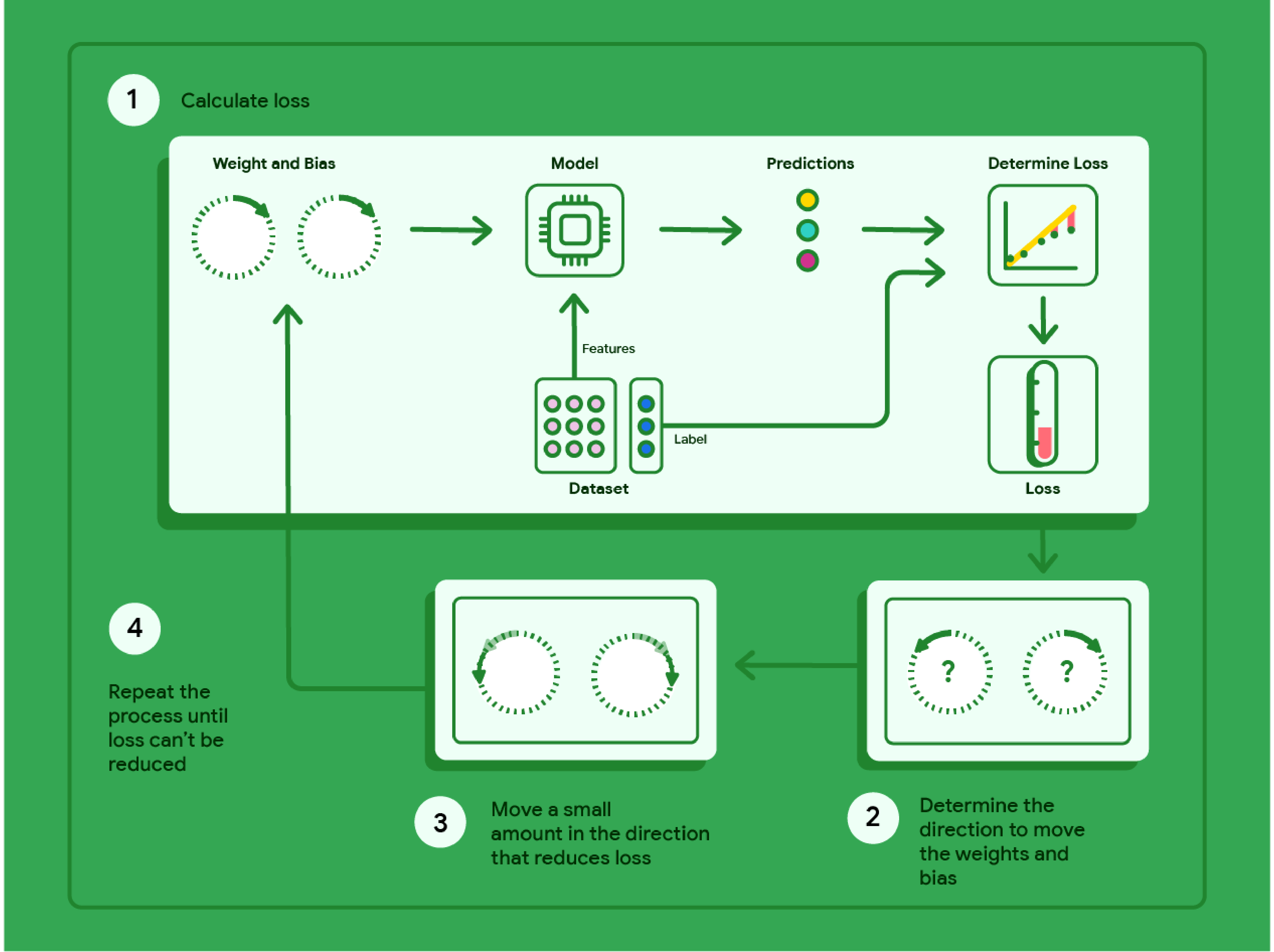
$\bar{y} \equiv \frac{1}{n}\sum_{i=1}^{n} y_i$ and $\bar{x} \equiv \frac{1}{n}\sum_{i=1}^{n} x_i$ are the sample means.

Gradient Descent

- Finding the coefficients for the line of best fit using a closed form equation works well for single variable regression.

- When we have more than one feature, there is a closed form equation called the normal equation, but it is slow to compute the coefficients.

- Gradient descent is an algorithm used to find the coefficients for multiple linear regression and many other machine learning models like logistic regression and neural networks.


1
Calculate loss
Weight and Bias
Model
Predictions
Determine Loss
Features
Label
Dataset
Loss
4
Repeat the process until loss can't be reduced
3
Move a small amount in the direction that reduces loss
2
Determine the direction to move the weights and bias
?
?

Linear Regression Setup

- Model:

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

- Loss function (Mean Squared Error, MSE):

$$L(w, b) = \frac{1}{n} \sum_{i=1}^{n} (y_i - (wx_i + b))^2$$

- Need to find w, b that minimize L

Gradient Descent Idea

- Start with random w, b
- Iteratively update:

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

- In gradient descent, we don't just *define* a variable, we *update* it.

$$w := w - \alpha \frac{\partial L}{\partial w}$$

$$b := b - \alpha \frac{\partial L}{\partial b}$$

The symbol **":="** means **"is updated to"** (assignment)

$$w := w - \alpha \frac{\partial L}{\partial w}$$

$$w_{\text{new}} = w_{\text{old}} - \alpha \frac{\partial L}{\partial w}$$

Update rule: Mathematical Justification

- Gradient descent is based on a first-order approximation (Taylor expansion):

$$f(x) = f(a) + f'(a)\frac{(x-a)}{1!} + f''(a)\frac{(x-a)^2}{2!} + f'''(a)\frac{(x-a)^3}{3!} + \dots$$
$$= \sum_{n=0}^{\infty} f^{(n)}(a)\frac{(x-a)^n}{n!}$$

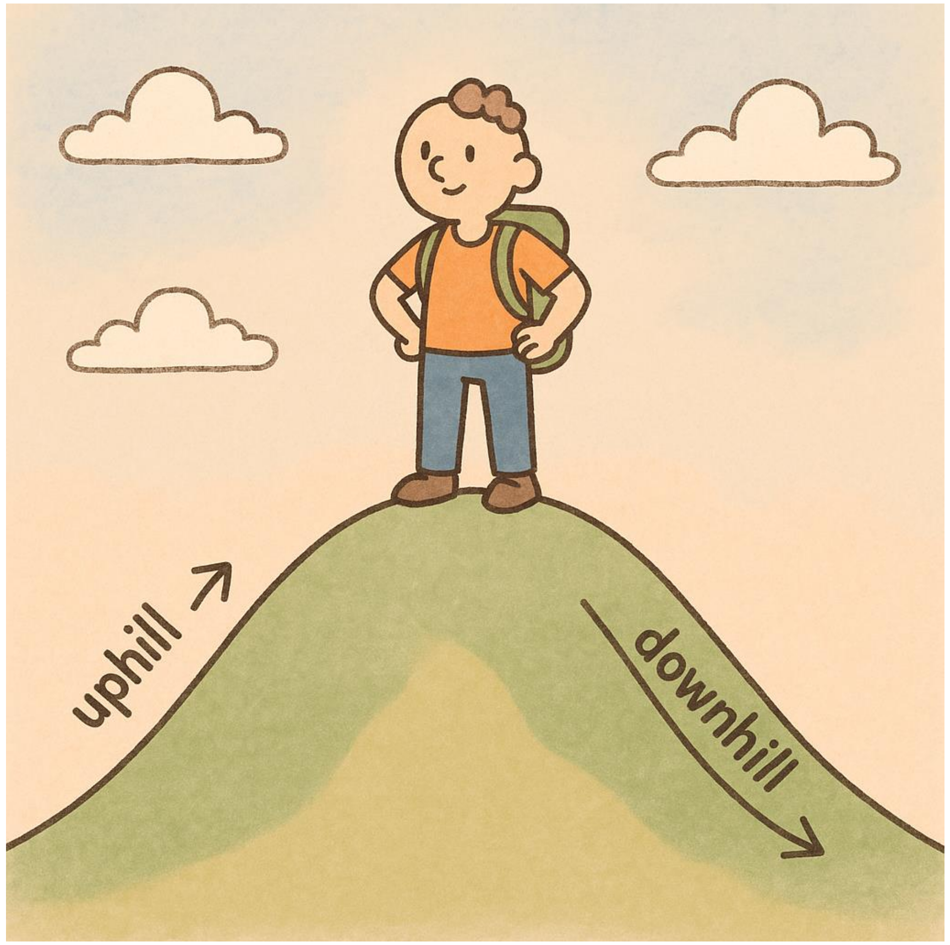
$$L(w + \Delta w) \approx L(w) + \frac{\partial L}{\partial w} \cdot \Delta w$$

- if you move in the **same direction** as the gradient:
  Δw has the same sign -> the term is positive -> L increases.
- If you move in the **opposite direction**:
  Δw has the opposite sign -> the term is negative -> L decreases.

- The **height** of the hill is like the *loss function* $L(w)$.
- Your **position** on the hill is like the *weights* w.
- The **slope of the hill** at your feet is like the *gradient* $\frac{\partial L}{\partial w}$

If you walk in the same direction as the slope points (uphill), you’ll go higher (the loss increases).

If you walk in the opposite direction (downhill), you’ll go lower (the loss decreases).



$$L(w,b) = \frac{1}{n}\sum_{i=1}^{n}(y_i - (wx_i + b))^2$$

Derivation for Linear Regression

- We want to minimize: $$L(w,b) = \frac{1}{n}\sum_{i=1}^{n}(y_i - (wx_i + b))^2$$
- Step 1: Gradient wrt *w*

$$\frac{\partial L}{\partial w} = \frac{\partial}{\partial w}\left[\frac{1}{n}\sum_{i=1}^{n}(y_i - (wx_i + b))^2\right]$$

Apply chain rule:

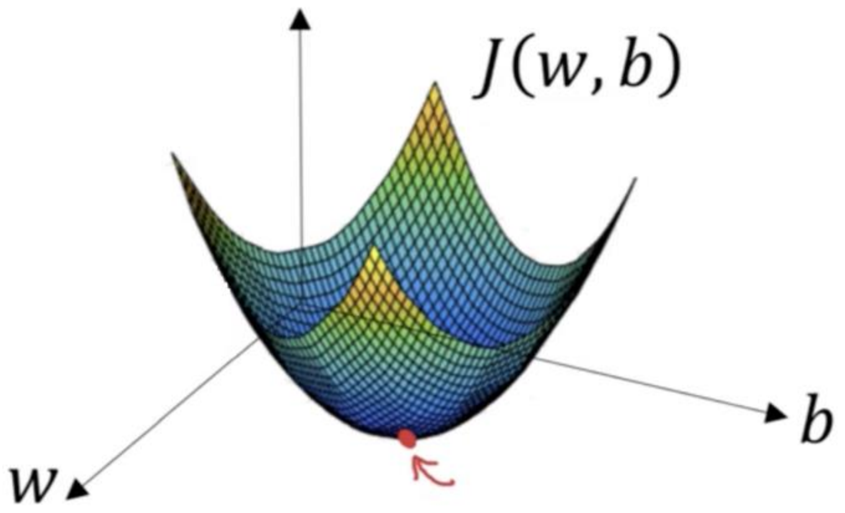
$$= \frac{1}{n}\sum_{i=1}^{n} 2(y_i - (wx_i + b)) \cdot \frac{\partial}{\partial w}(y_i - (wx_i + b))$$

Derivative inside:

$$\frac{\partial}{\partial w}(y_i - (wx_i + b)) = -x_i$$

So:

$$\frac{\partial L}{\partial w} = -\frac{2}{n}\sum_{i=1}^{n} x_i\,(y_i - (wx_i + b))$$



Derivation for Linear Regression

- Step 2: Gradient wrt b

$$\frac{\partial L}{\partial b} = \frac{\partial}{\partial b}\left[\frac{1}{n}\sum_{i=1}^{n}(y_i - (wx_i + b))^2\right]$$

Again, apply chain rule:

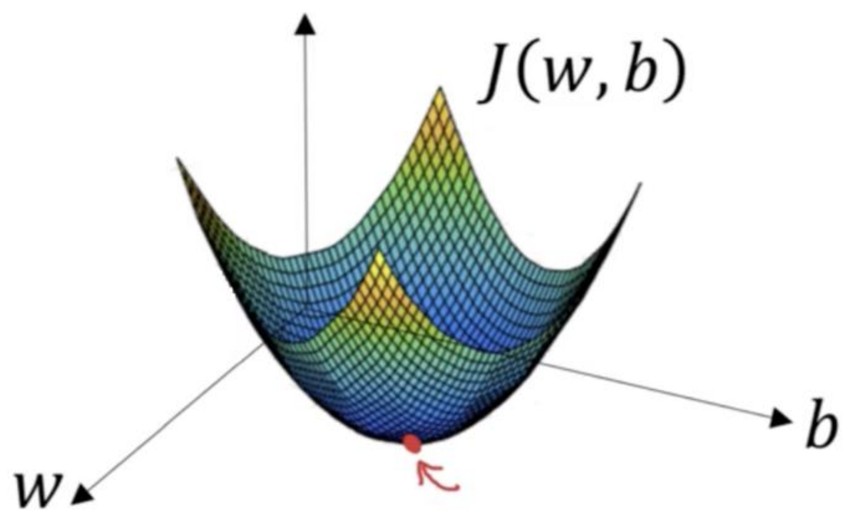
$$= \frac{1}{n}\sum_{i=1}^{n} 2(y_i - (wx_i + b)) \cdot \frac{\partial}{\partial b}(y_i - (wx_i + b))$$

Derivative inside:

$$\frac{\partial}{\partial b}(y_i - (wx_i + b)) = -1$$

So:

$$\frac{\partial L}{\partial b} = -\frac{2}{n}\sum_{i=1}^{n}(y_i - (wx_i + b))$$



Update Rules

- Now plug these into gradient descent:

$$w_{\text{new}} = w_{\text{old}} - \alpha\left(-\frac{2}{n}\sum_{i=1}^{n} x_i(y_i - (w_{\text{old}}x_i + b_{\text{old}}))\right)$$

$$b_{\text{new}} = b_{\text{old}} - \alpha\left(-\frac{2}{n}\sum_{i=1}^{n}(y_i - (w_{\text{old}}x_i + b_{\text{old}}))\right)$$

Each update moves parameters in the direction of steepest decrease of MSE

Simplify:

$$w_{\text{new}} = w_{\text{old}} + \frac{2\alpha}{n}\sum_{i=1}^{n} x_i\left(y_i - (w_{\text{old}}x_i + b_{\text{old}})\right)$$

$$b_{\text{new}} = b_{\text{old}} + \frac{2\alpha}{n}\sum_{i=1}^{n}(y_i - (w_{\text{old}}x_i + b_{\text{old}}))$$

$$w_{\text{new}} = w_{\text{old}} + \frac{2\alpha}{n}\sum x_i \cdot \text{error}_i$$

$$b_{\text{new}} = b_{\text{old}} + \frac{2\alpha}{n}\sum \text{error}_i$$

Example Dataset

x	y
1	2
2	4
3	6

Goal: fit line $y = wx + b$

Start: $w = 0, b = 0, \alpha = 0.1$
Compute predictions: $\hat{y} = 0$

$$w_{\text{new}} = w_{\text{old}} + \frac{2\alpha}{n} \sum_{i=1}^{n} x_i \left(y_i - (w_{\text{old}} x_i + b_{\text{old}}) \right)$$

$$b_{\text{new}} = b_{\text{old}} + \frac{2\alpha}{n} \sum_{i=1}^{n} (y_i - (w_{\text{old}} x_i + b_{\text{old}}))$$

Example Dataset

x	y
1	2
2	4
3	6

Goal: fit line $y = wx + b$

$$w_{\text{new}} = w_{\text{old}} + \frac{2\alpha}{n}\sum_{i=1}^{n} x_i \left(y_i - (w_{\text{old}}x_i + b_{\text{old}})\right)$$

$$b_{\text{new}} = b_{\text{old}} + \frac{2\alpha}{n}\sum_{i=1}^{n}(y_i - (w_{\text{old}}x_i + b_{\text{old}}))$$

t	w	b	dL/dw	dL/db	MSE
0	0.000000	0.000000	-18.666667	-8.000000	18.666667
1	1.866667	0.800000	1.955556	1.066667	0.296296
2	1.671111	0.693333	-0.296296	0.071111	0.073376
3	1.700741	0.686222	-0.048198	0.175407	0.067396

$$w := w - \alpha \frac{\partial L}{\partial w}$$

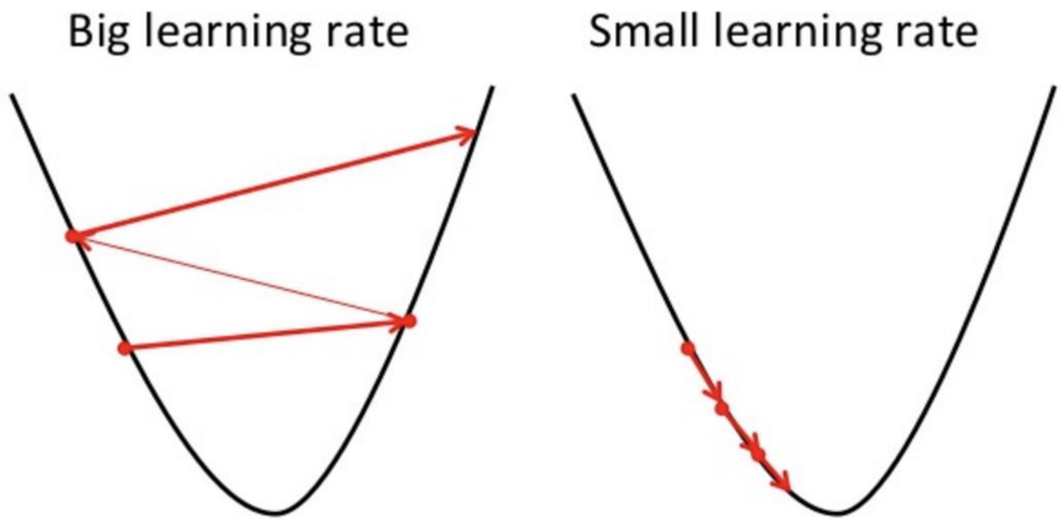
Learning Rate

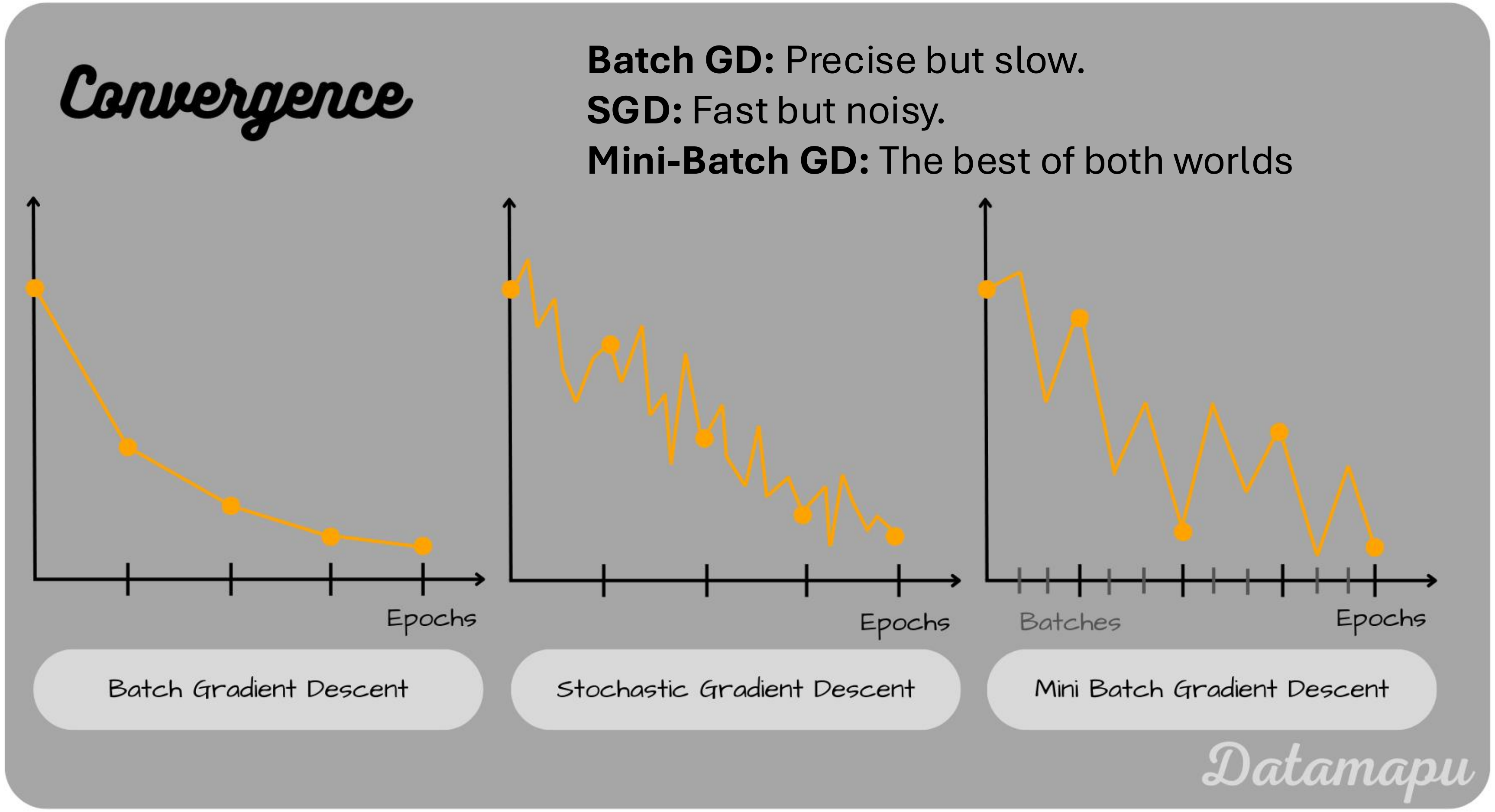
$$b := b - \alpha \frac{\partial L}{\partial b}$$

- The factor α controls how big a correction is made per iteration.
- Too large -> updates overshoot and diverge.
- Too small -> very slow convergence.
- α is something you tune during training

$$w_{\text{new}} = w_{\text{old}} + \frac{2\alpha}{n} \sum x_i \cdot \text{error}_i$$

$$b_{\text{new}} = b_{\text{old}} + \frac{2\alpha}{n} \sum \text{error}_i$$



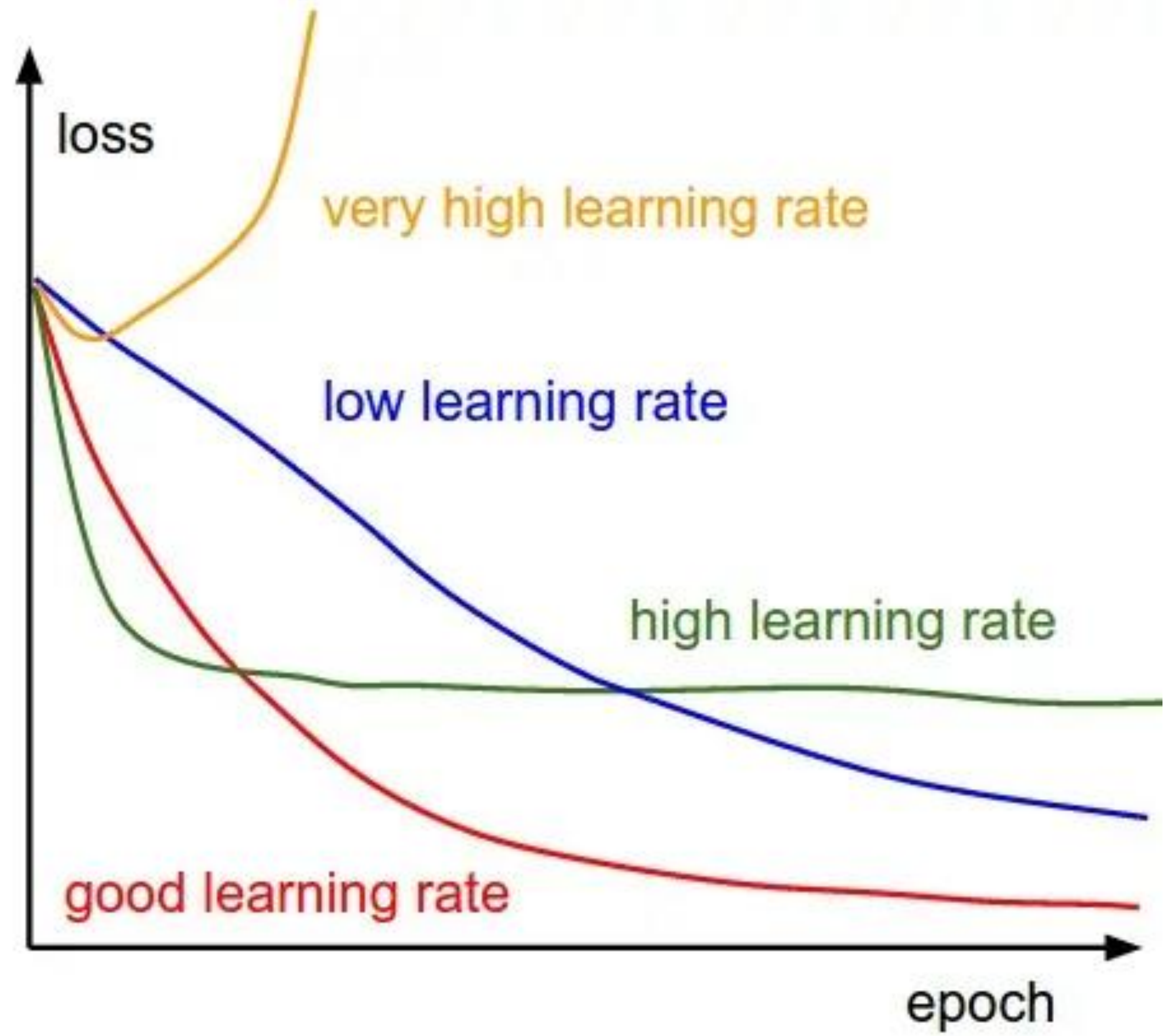


Definition: Uses the entire dataset to compute the gradient before updating parameters.
Good for: Small datasets where computation is manageable.

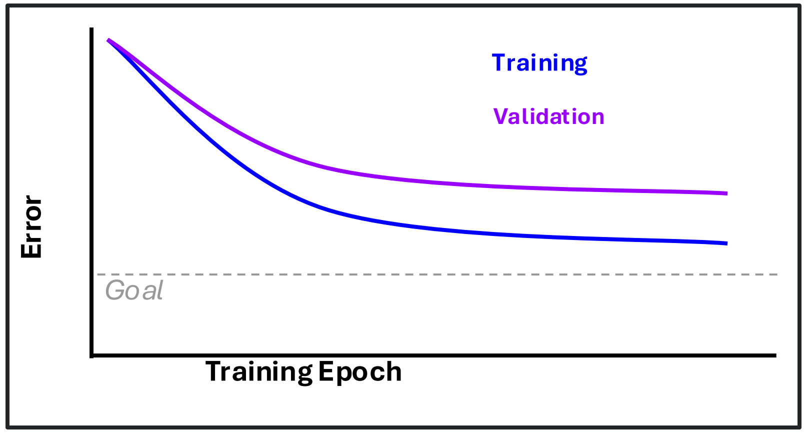
Definition: Uses only one random data point to compute the gradient and update parameters.
Good for: Very large datasets where full-batch would be too expensive

Definition: Splits data into small batches (e.g., 32, 64 samples) and uses each batch to compute gradients.
Good for: Large datasets, GPU optimization, and real-world ML training

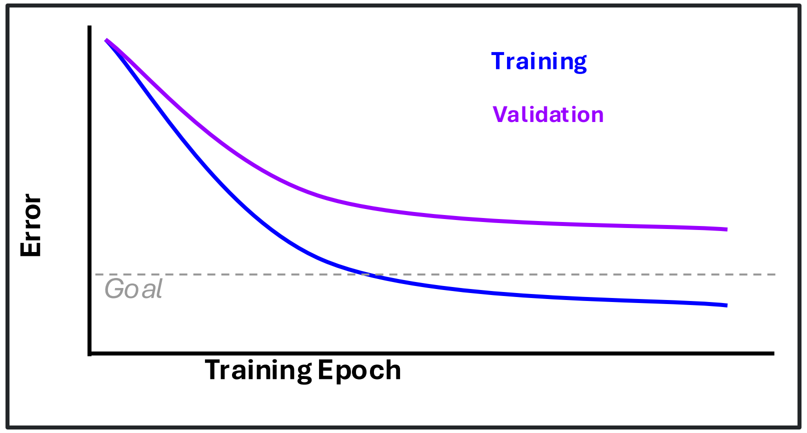
Make sure gradient descent is working correctly



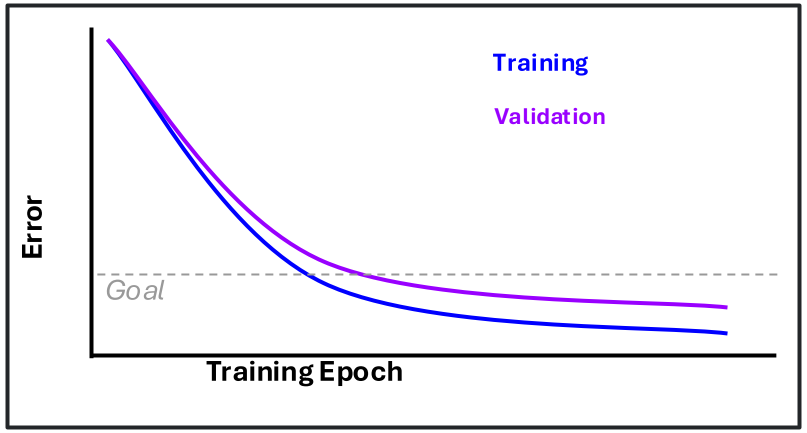
Make sure gradient descent is working correctly



Underfitting



Overfitting



Good Fit

Stop when the **training loss** stops decreasing
(i.e., changes less than some threshold)

Polynomial Regression

What if the data isn't linear?

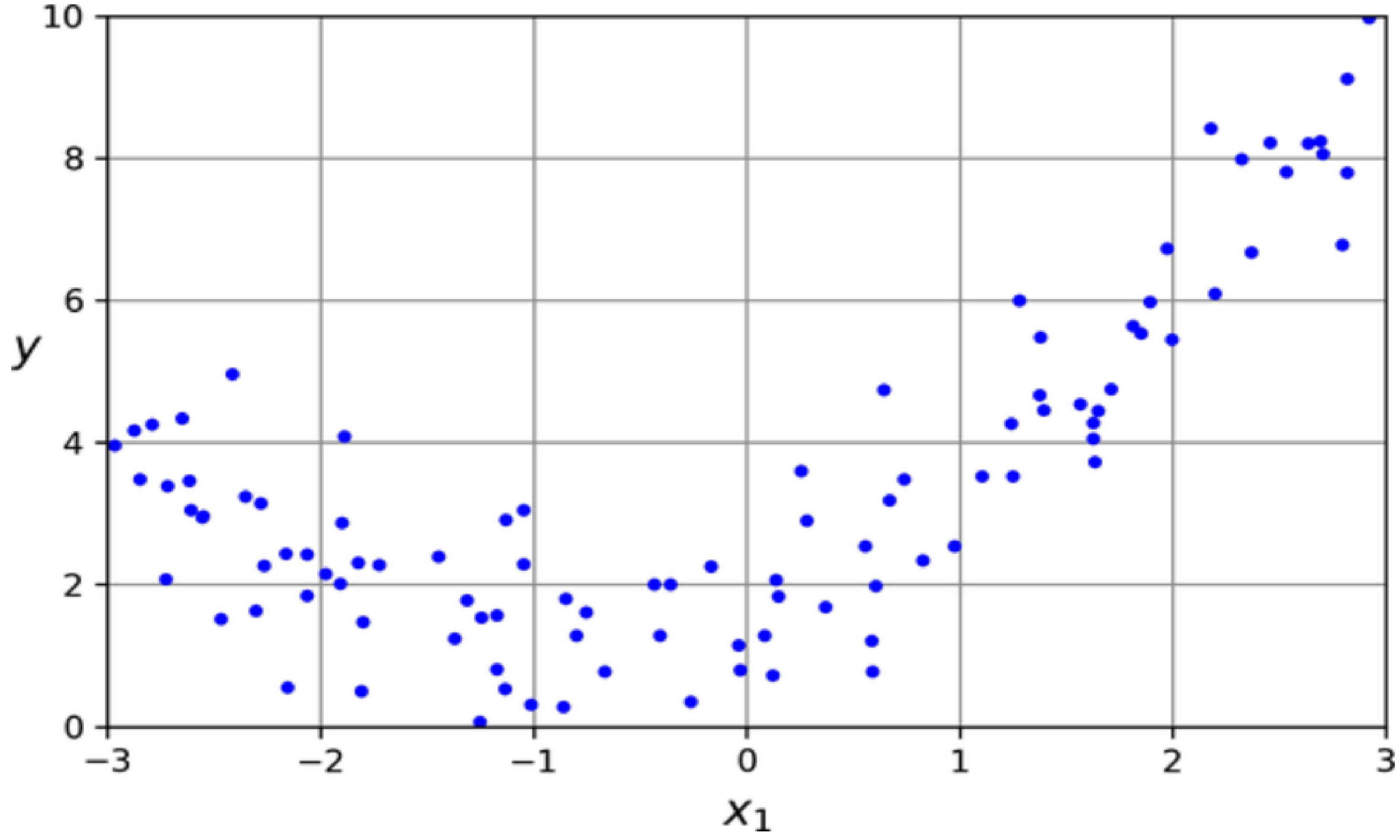


Figure 4-12. Generated nonlinear and noisy dataset

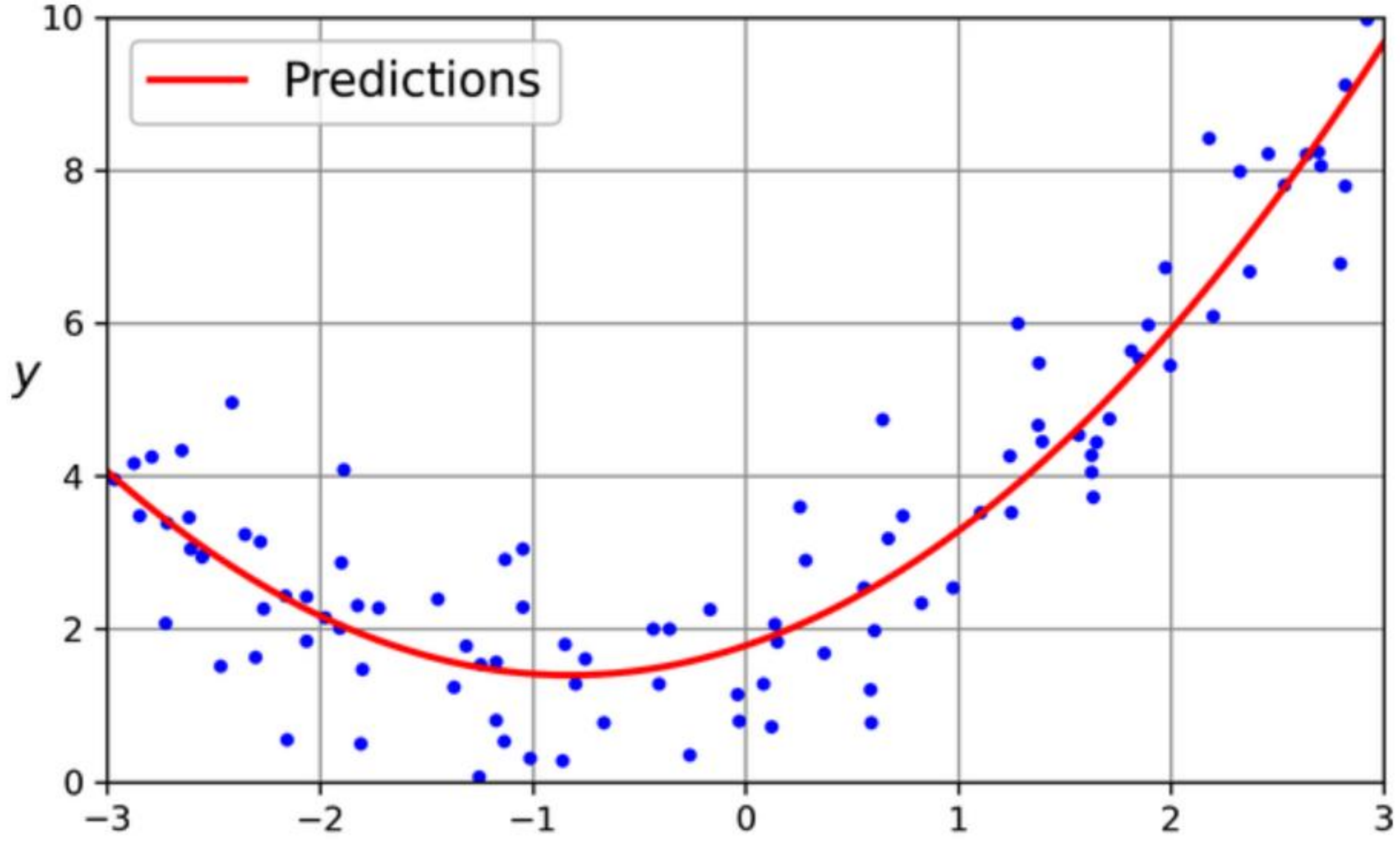
Polynomial Regression

- Polynomial regression adds powers to each feature to create new features.

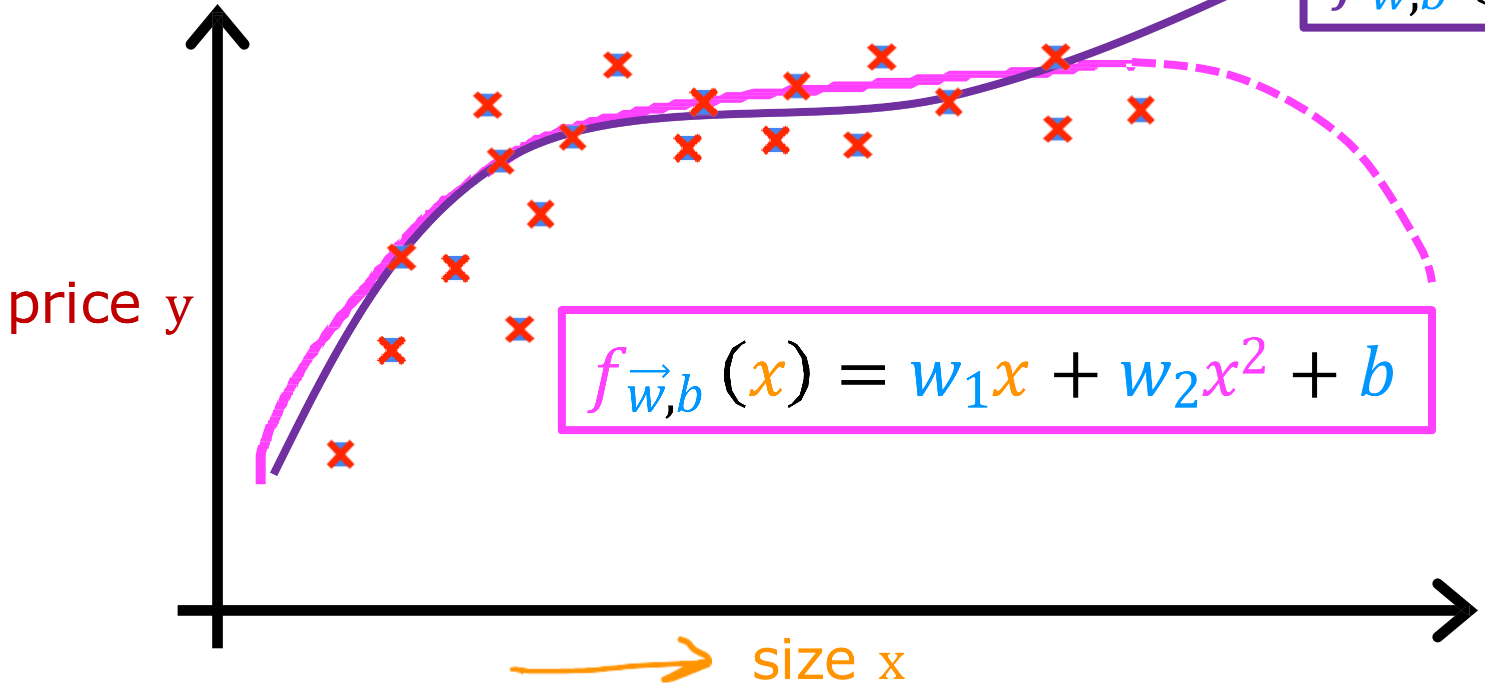
- A quadratic function would fit the data on the previous slide better than a linear function.

- $y = w_1x_1 + w_2x_2^2 + b$

Much better fit!



Polynomial regression



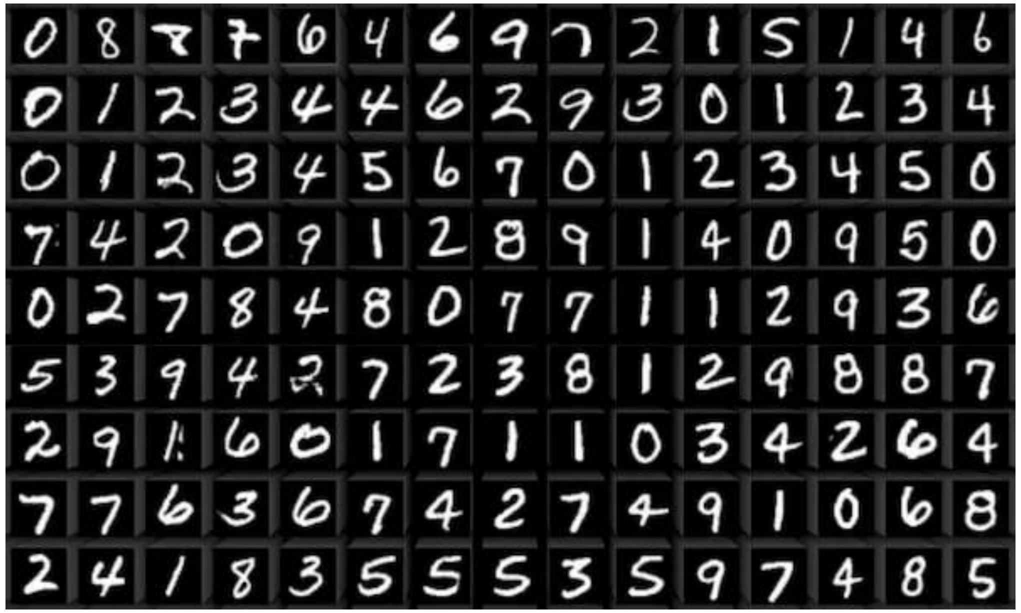
$$f_{\vec{w},b}(x) = w_1x + w_2x^2 + w_3x^3 + b$$

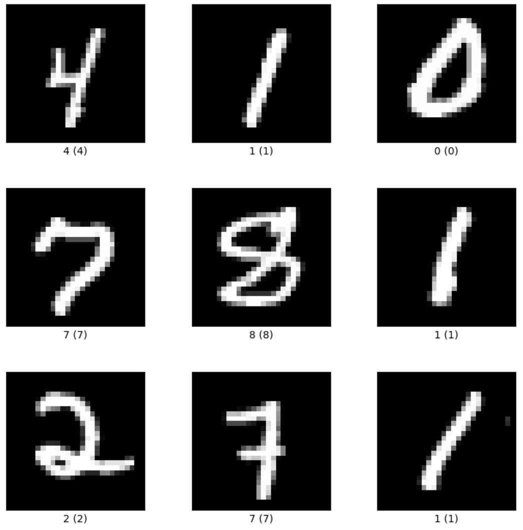
$$f_{\vec{w},b}(x) = w_1x + w_2x^2 + b$$

Intro to Neural Networks

Recognizing Digits: MNIST Dataset

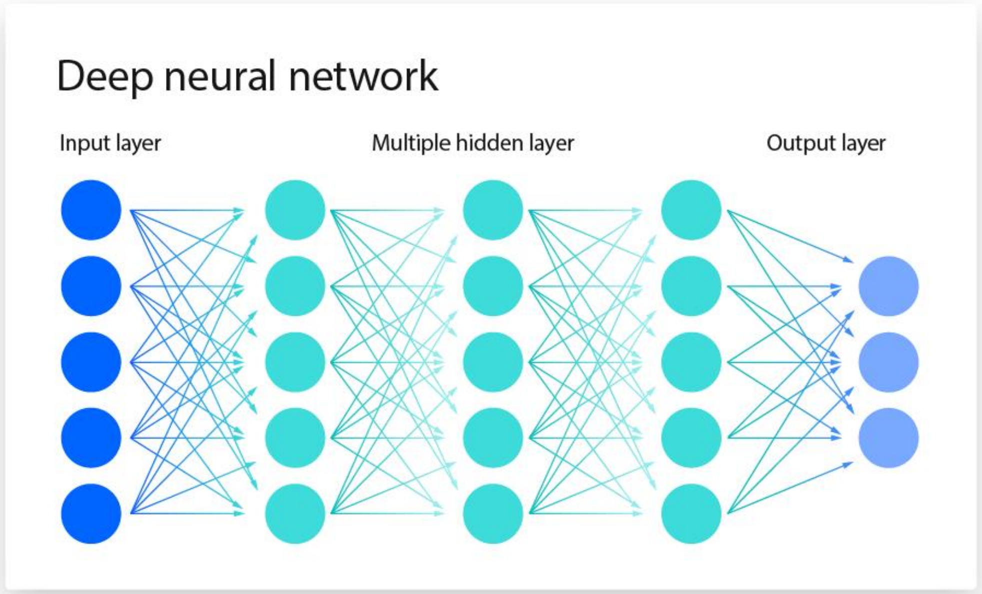
- Each image is 28x28 pixels -> flatten 784 values
- Programming this task explicitly = very difficult





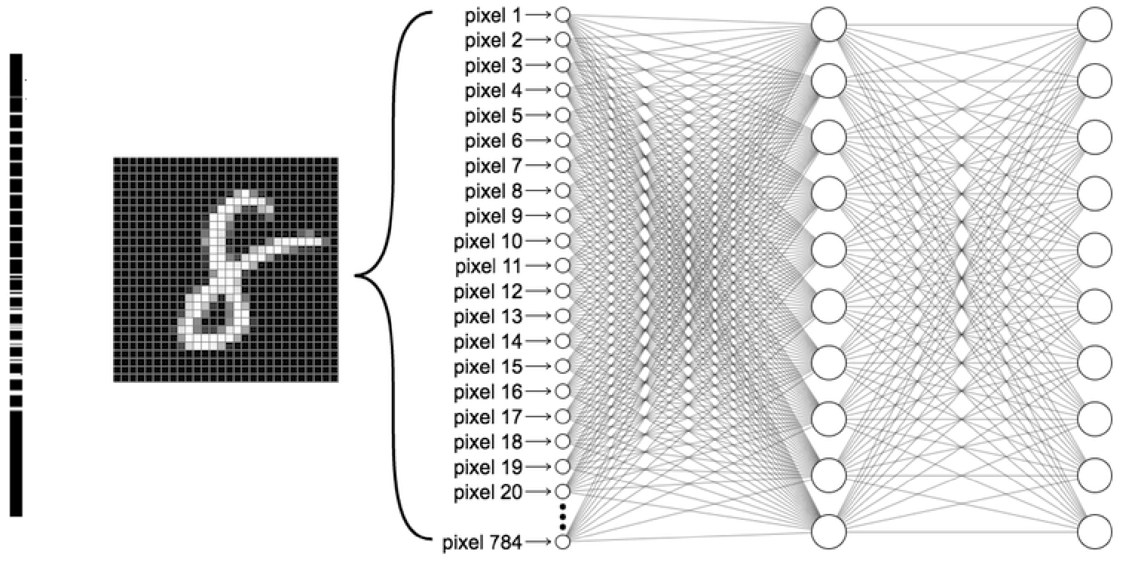
Neural Network Basics:

- Input layer -> Receives data (features)
- Hidden layer(s) -> Processes information through weighted connections
- Output layer -> Produces the final prediction

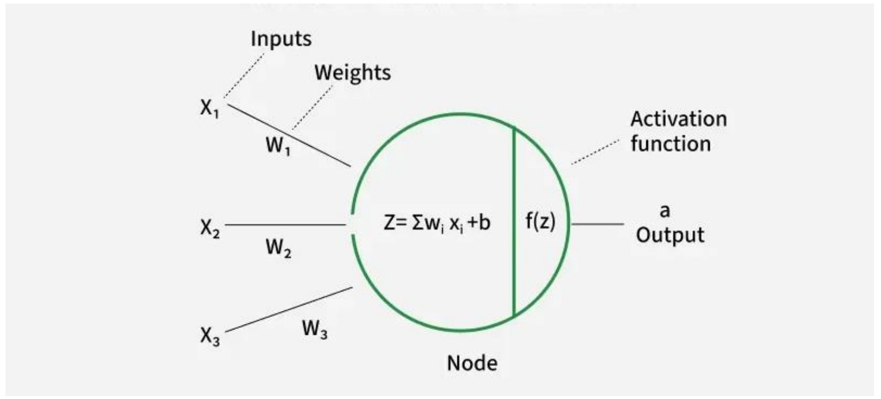


Neural Network Basics: example

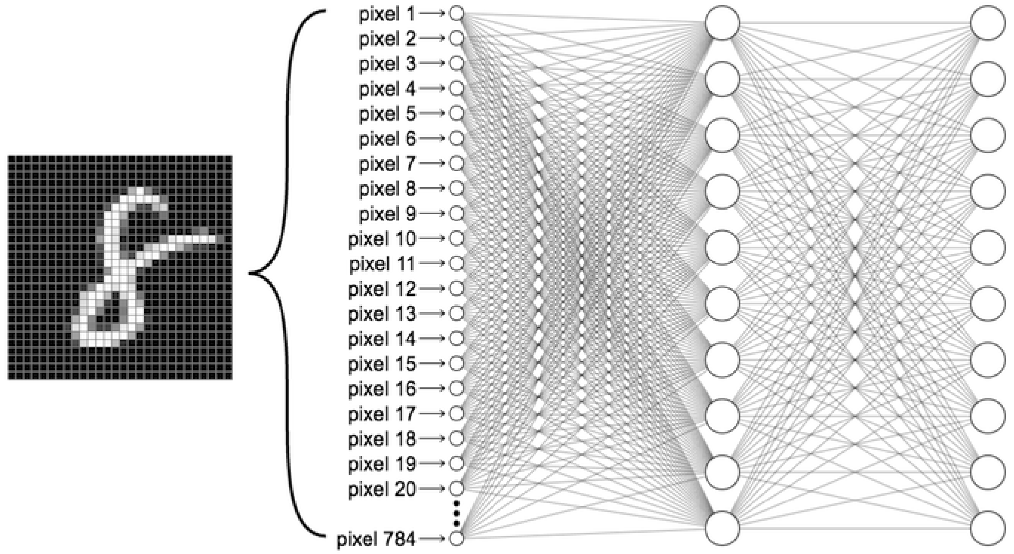
- Neuron = holds a number between 0 and 1 (activation)
- Input layer: 28x28 pixels → 784 neurons
- Output layer: 10 neurons (digits 0–9)
- Hidden layers: middle layers, structure is flexible
- Each neuron: use its own linear regression model



Weights and Biases



- Connections between neurons have weights
- Weighted sum of inputs + bias -> activation
- Sigmoid function squishes result to [0,1]
- Bias shifts activation threshold

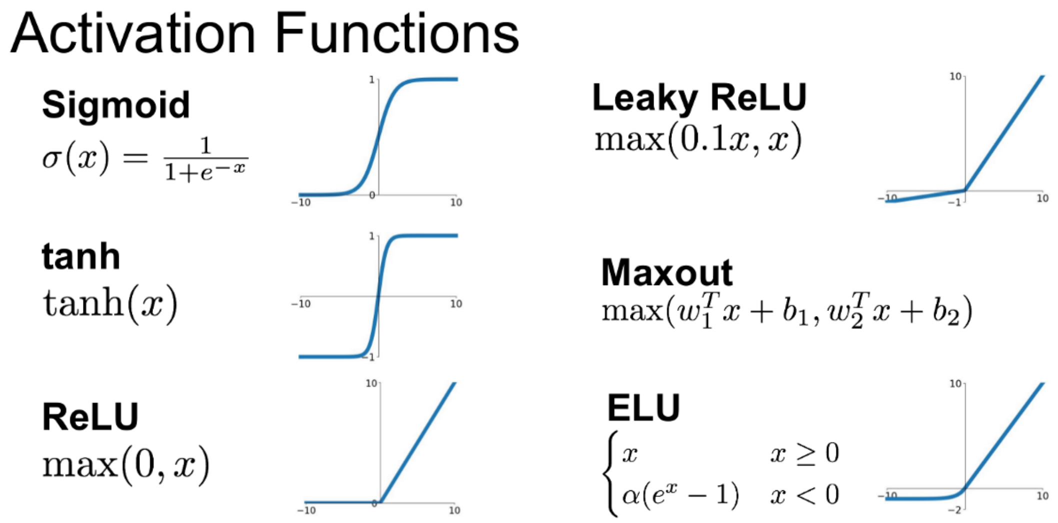


Total parameters:
~13,000 (weights + biases)

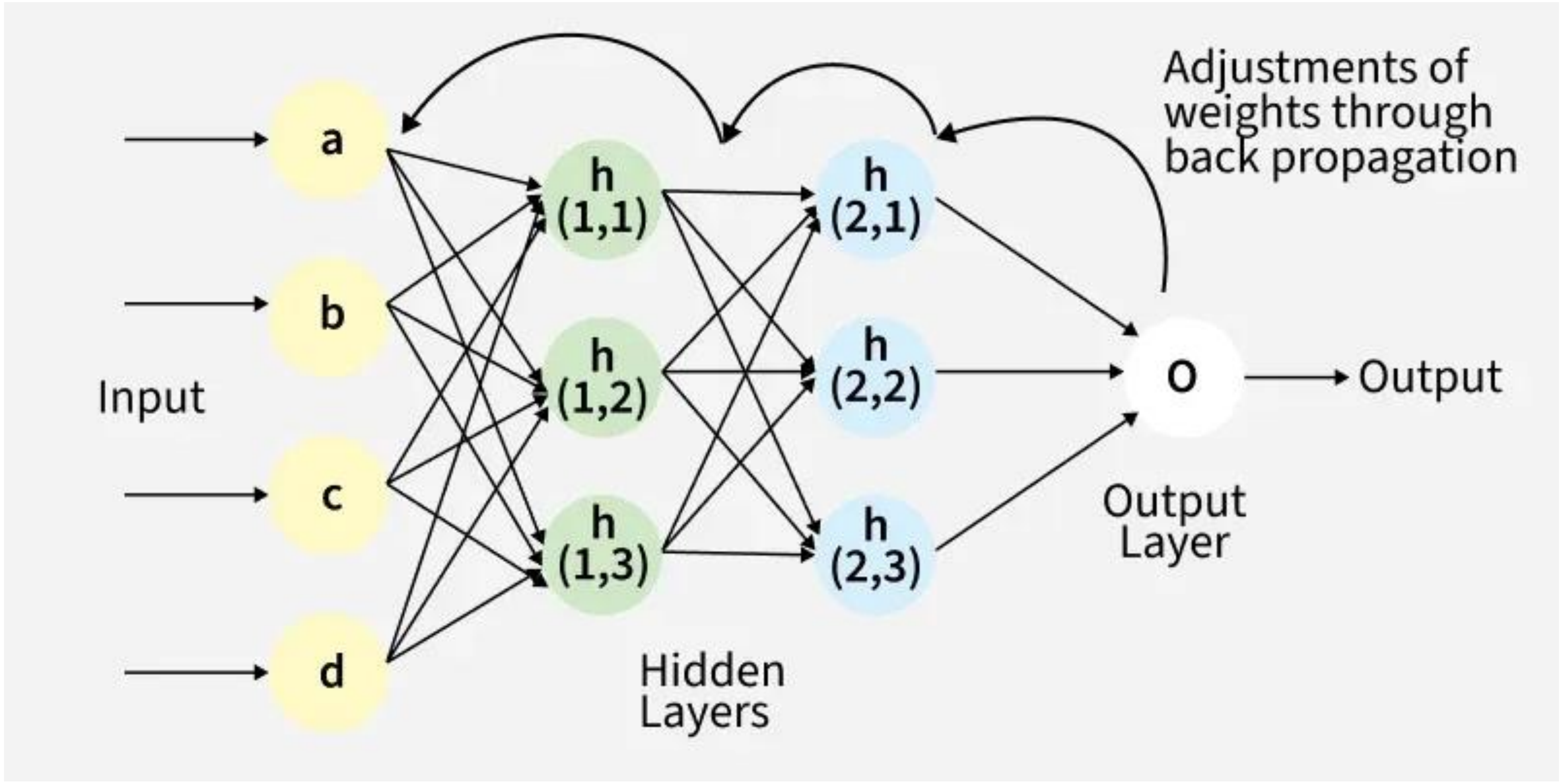
Activation Functions

- Sigmoid: squishes input into [0,1]
- Motivated by biological analogy (active/inactive)

- Modern networks often use ReLU
- ReLU = max(0, a), easier to train deep networks



Back propagation


Adjustments of weights through back propagation
a
b
c
d
Input
h (1,1)
h (1,2)
h (1,3)
h (2,1)
h (2,2)
h (2,3)
Hidden Layers
O
Output
Output Layer