Lecture 6

Gradient Descent

Last Time:

- Machine learning, especially supervised learning
- Bias, variance, and overfitting
- Minimized an objective function, called error or cost or risk
- Did this on training set, showed test set was a good proxy for out of sample error
- Fit hyper-parameters on validation set

LLN: Expectations -> sample averages

$$E_f[R] = \int R(x) f(x) dx = \lim_{n \to \infty} \frac{1}{N} \sum_{x_i \sim f} R(x_i)$$

Empirical Risk Minimization:

$$R_{\mathcal{D}} = E_f[R] \sim \frac{1}{N} \sum_{x_i \sim f} R(x_i)$$

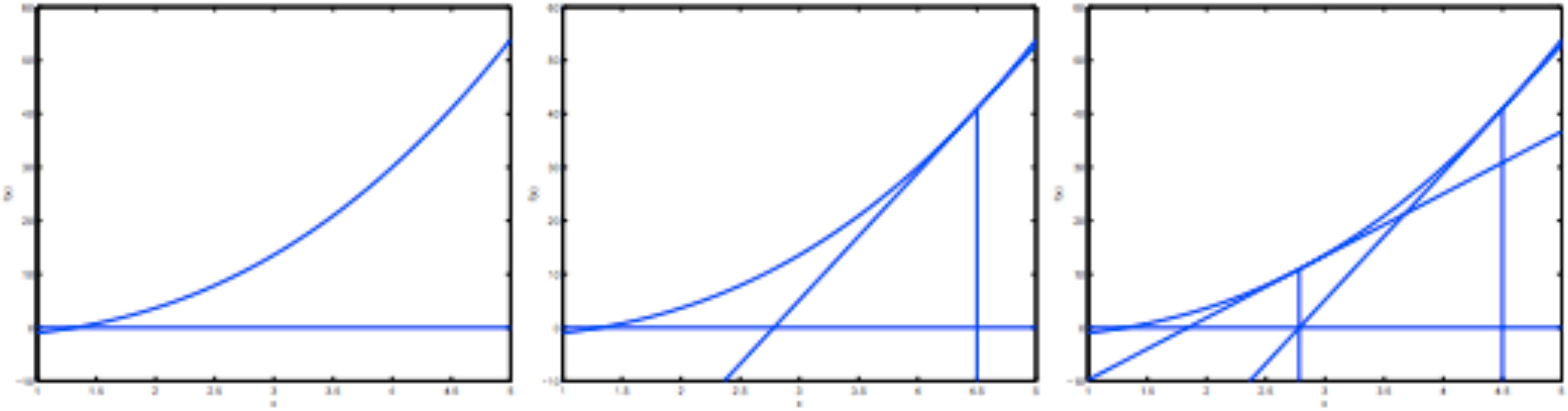
on training set(sample) $\mathcal{D}$.

Today: optimization using gradient descent

- gradient descent
- stochastic gradient descent

Remember Convex (bowl) like functions have 1 global minimum

Newton's Method



Find a zero of the first derivative.

Gradients and Hessians

$$J(\bar{\theta}) = \theta_1^2 + \theta_2^2$$

Gradient: $\nabla_{\bar{\theta}}(J) = \dfrac{\partial J}{\partial \bar{\theta}} = \begin{pmatrix} 2\theta_1 \\ 2\theta_2 \end{pmatrix}$

Hessian H = $\begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix}$

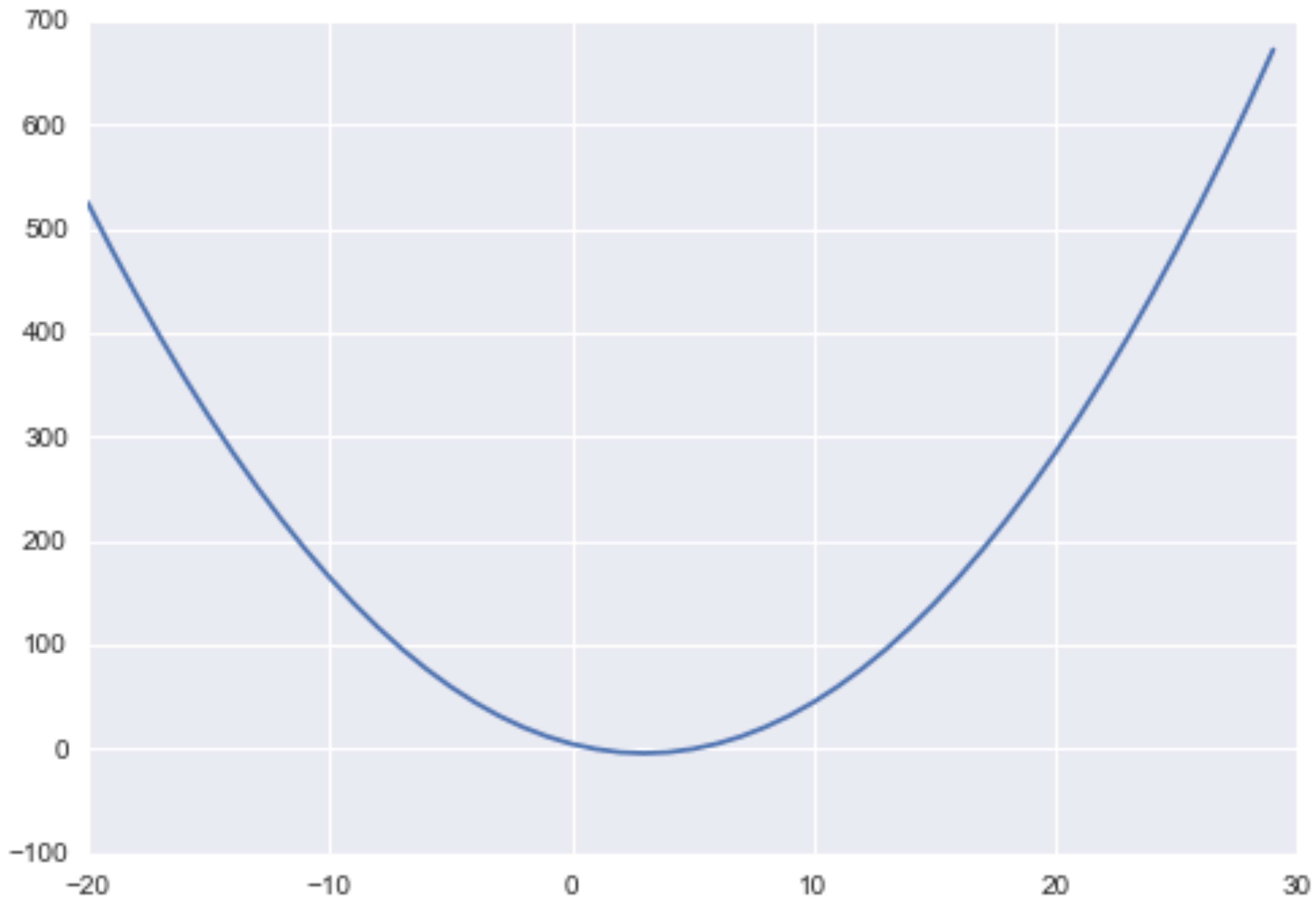
Hessian gives curvature. Why not use it?

Gradient ascent (descent)

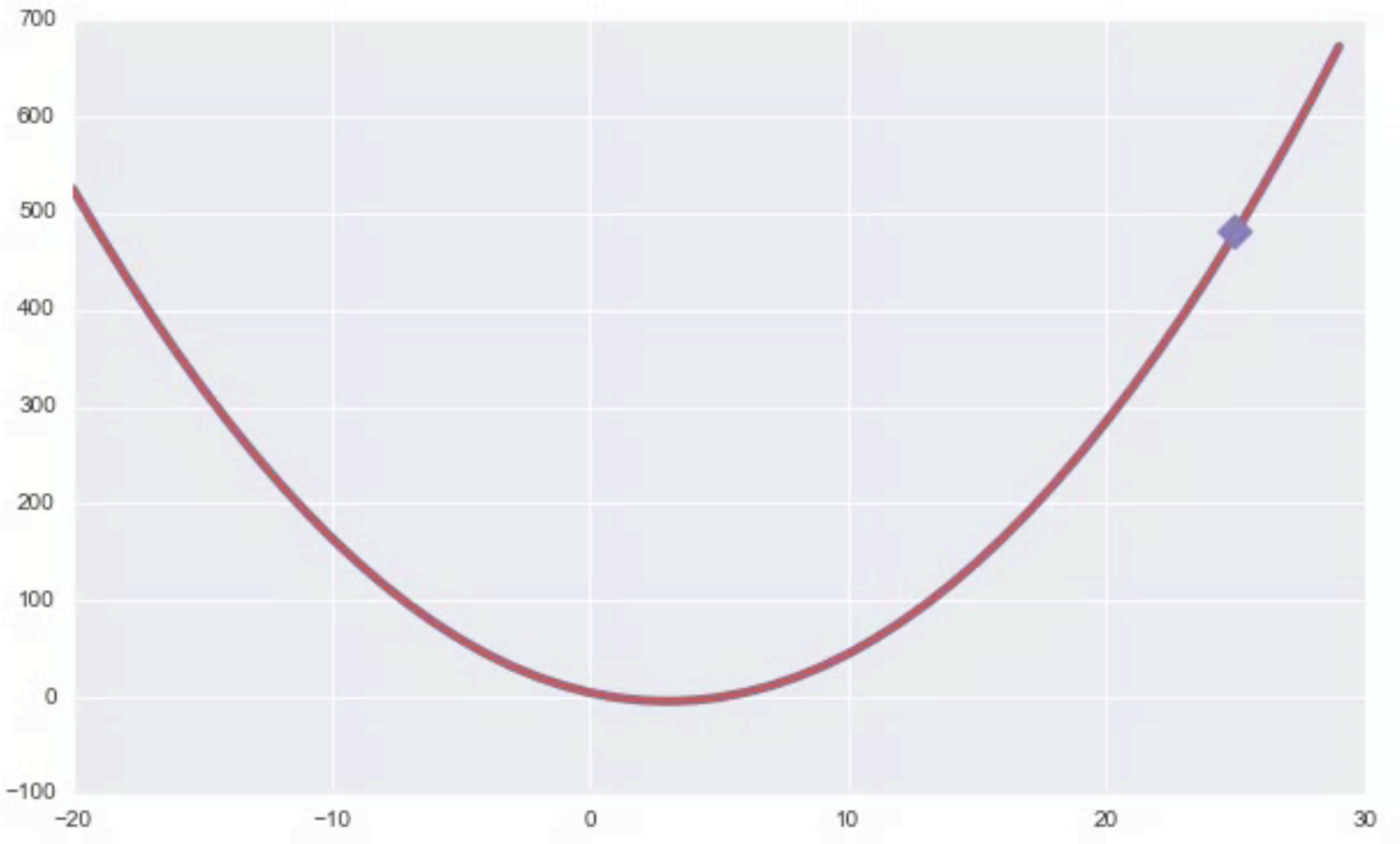
basically go opposite the direction of the derivative.

Consider the objective function:
$J(x) = x^2 - 6x + 5$

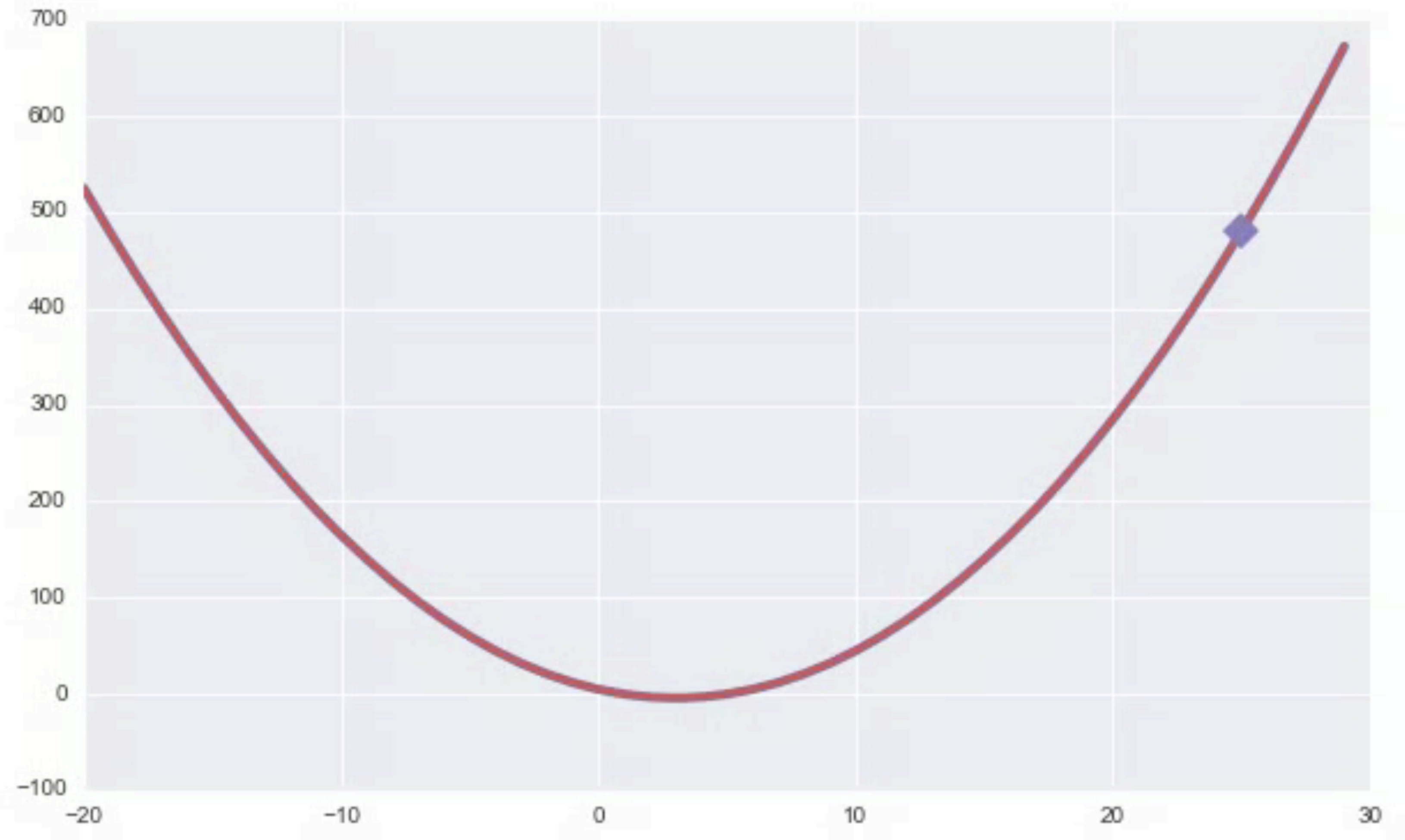
```
gradient = fprime(old_x)
move = gradient * step
current_x = old_x - move
```



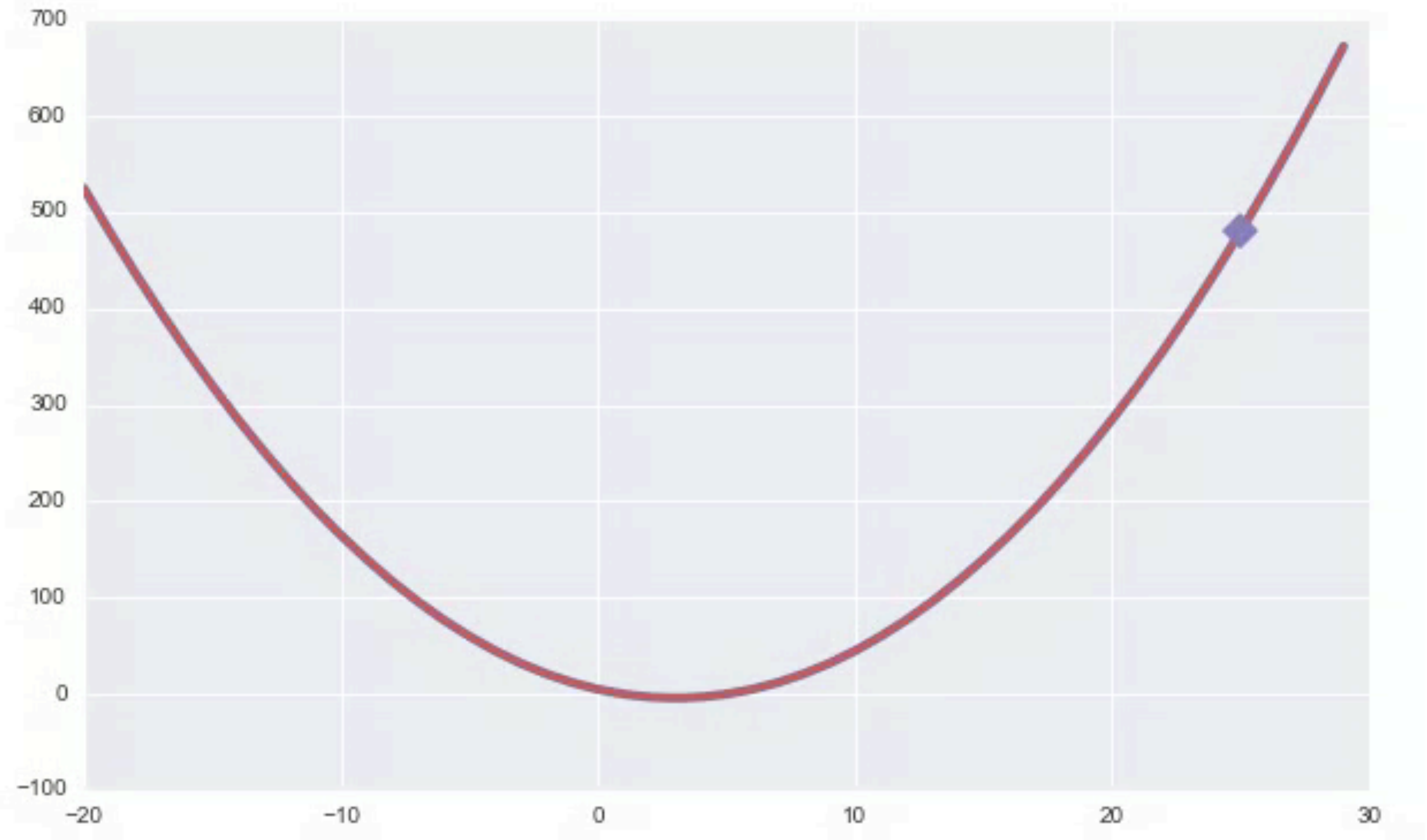
good step size



too big step size



too small step size



Example: Linear Regression

$$\hat(y) = f_\theta(x) = \theta^T x$$

Cost Function:

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{m}(f_\theta(x^{(i)} - y^{(i)})^2$$

Gradient Descent

$$\theta := \theta - \eta \nabla_{\theta} J(\theta) = \theta - \eta \sum_{i=1}^{m} \nabla J_i(\theta)$$

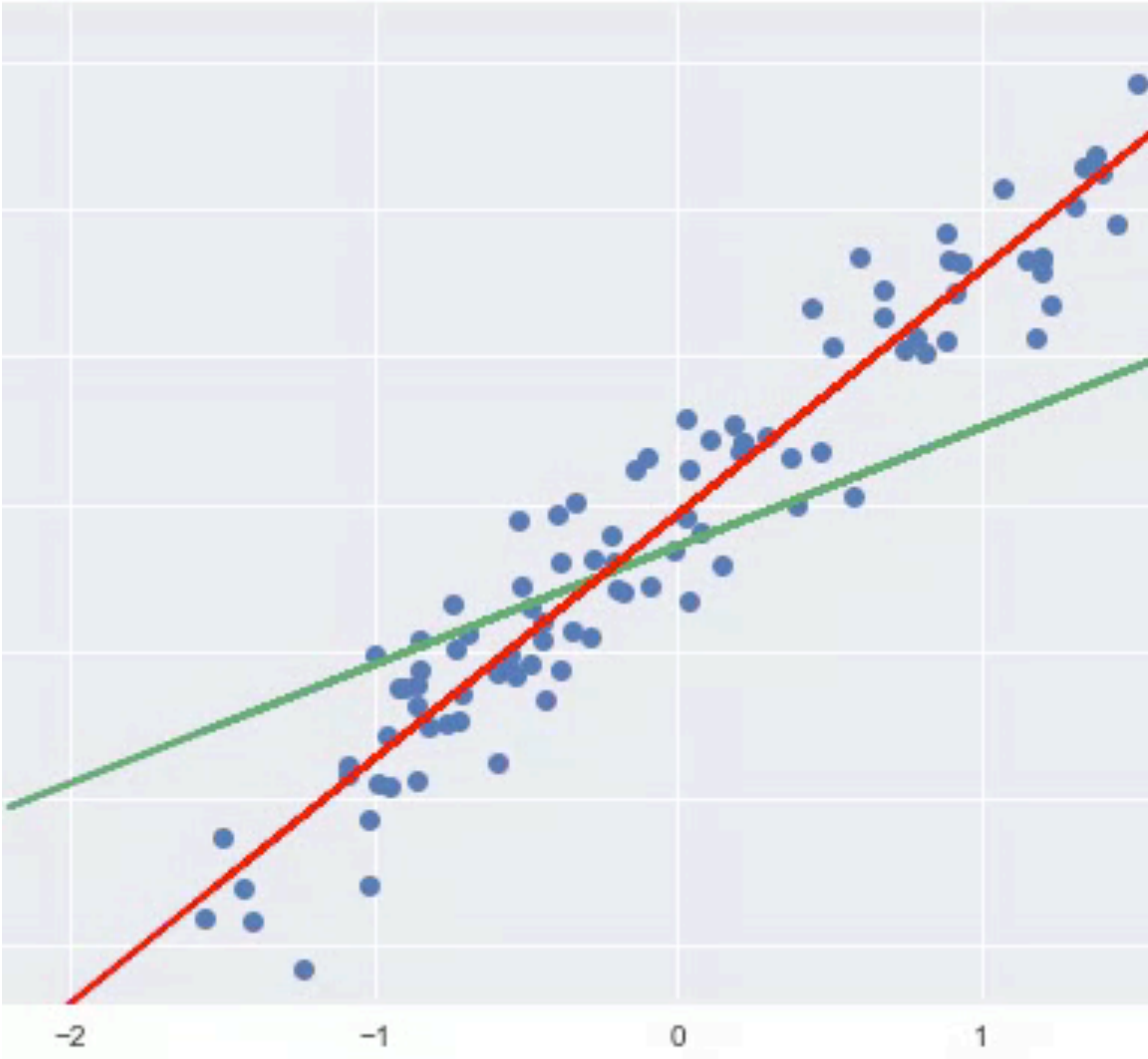
where η is the learning rate.

ENTIRE DATASET NEEDED

```
for i in range(n_epochs):
  params_grad = evaluate_gradient(loss_function, data, params)
  params = params - learning_rate * params_grad`
```

Linear Regression: Gradient Descent

$$\theta_j := \theta_j + \alpha \sum_{i=1}^{m} (y^{(i)} - f_\theta(x^{(i)}))x_j^{(i)}$$



Stochastic Gradient Descent

$$\theta := \theta - \alpha \nabla_\theta J_i(\theta)$$

ONE POINT AT A TIME

```
for i in range(nb_epochs):
  np.random.shuffle(data)
  for example in data:
    params_grad = evaluate_gradient(loss_function, example, params)
    params = params - learning_rate * params_grad
```

Mini-Batch: do some at a time

Linear Regression: SGD

$$\theta_j := \theta_j + \alpha(y^{(i)} - f_\theta(x^{(i)}))x_j^{(i)}$$