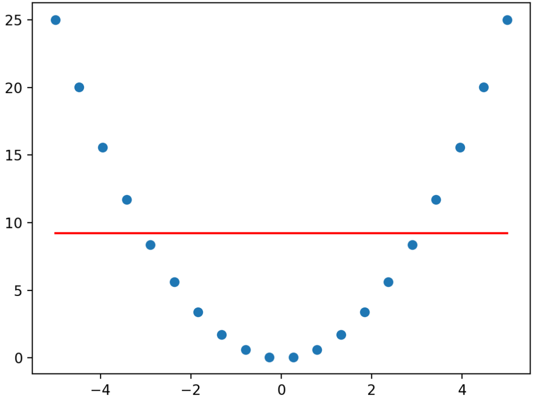
Regularization, Non-linear models & Gradient Descent

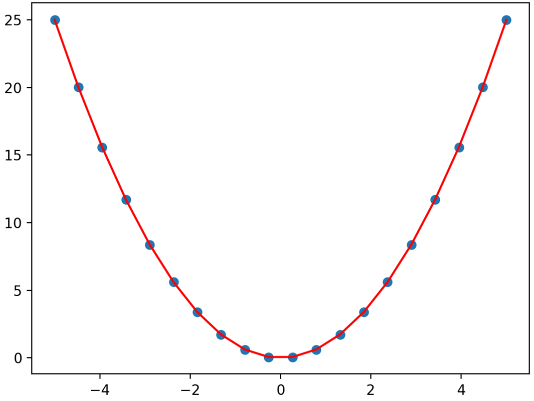
CS 480

Intro to Artificial Intelligence

Linear regression for nonlinear functions



$$X = \begin{bmatrix} 1 & x^{(1)} \\ 1 & x^{(2)} \\ 1 & x^{(3)} \\ & \vdots \\ 1 & x^{(N)} \end{bmatrix}$$



$$X = \begin{bmatrix} 1 & x^{(1)} & (x^{(1)})^2 \\ 1 & x^{(2)} & (x^{(2)})^2 \\ 1 & x^{(3)} & (x^{(3)})^2 \\ & \vdots & \\ 1 & x^{(N)} & (x^{(N)})^2 \end{bmatrix}$$

Basis function expansion

This generalizes to any set of transforming functions we care to use.

Our hypothesis class is now functions that are linear combinations of a set of **basis functions**.

The θ which minimizes the loss can be found the same way, just by replacing X with $\boldsymbol{\Phi}$

$$\mathcal{H} = \left\{ h_\theta : h_\theta(\mathbf{x}) = \sum_{i=1}^{K} \theta_k \phi_k(\mathbf{x}) \right\}$$

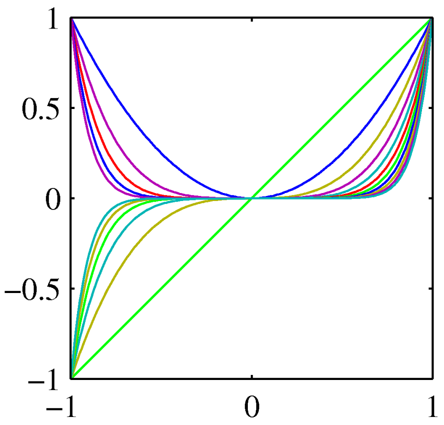
$$\phi_k : \mathcal{X} \to \mathbb{R}, \qquad \{\phi_k\}_{k=1}^{K}$$

$$\Phi = \begin{bmatrix} — & \phi^{(1)} & — \\ — & \phi^{(2)} & — \\ & \vdots & \\ — & \phi^{(N)} & — \end{bmatrix}$$

$$\phi^{(i)} = \begin{bmatrix} 1 & \phi_1(\mathbf{x}^{(i)}) & \phi_2(\mathbf{x}^{(i)}) & \cdots & \phi_k(\mathbf{x}^{(i)}) \end{bmatrix}$$

$$\theta = \left(\Phi^\top \Phi\right)^{-1} \Phi^\top Y$$

Choices for ϕ

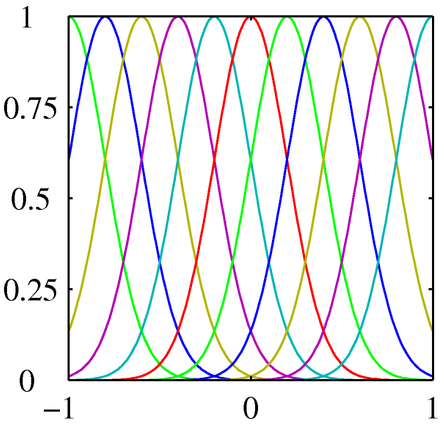


$$\phi_k(x) = x^k$$

$$\phi_{kj}(\mathbf{x}) = x_j^k$$

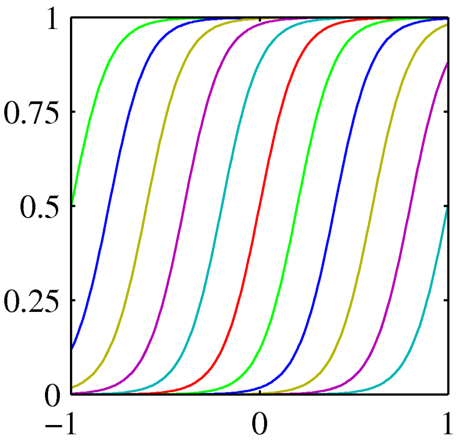
$$\phi_{kjl}(\mathbf{x}) = (x_j \cdot x_l)^k$$

$$\vdots$$



$$\phi_k(x) = \exp\left\{-\frac{(x-\mu_k)^2}{2s^2}\right\}$$

$$\phi_k(\mathbf{x}) = \exp\left\{-\frac{1}{2}(\mathbf{x}-\mu_k)^\top \Sigma^{-1}(\mathbf{x}-\mu_k)\right\}$$



$$\phi_k(x) = \left(1 + \exp\left\{-\frac{x-\mu_k}{s}\right\}\right)^{-1}$$

$$\phi_k(x) = \tanh(x)$$

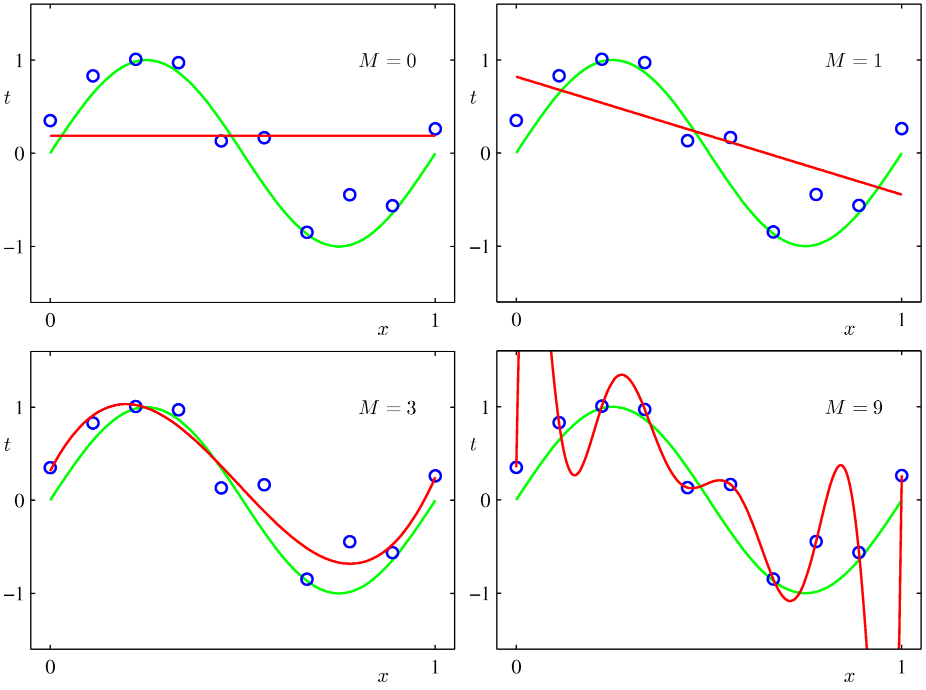
A problem - knowing when to stop

Adding more and more complicated functions to ϕ lets the model get more **complex**.

In general:
The more complex the model, the better at fitting the training data, but the more likely it is to **not generalize**

We will return to this, but for the moment we need a way to penalize the model for being too complex.

Enter regularization



Regularization

AKA: weight decay, ridge regression, Tikhonov regularization

If a component of θ is small, that feature has a small impact on the output

If all the components of θ are large, we're letting individual features have a big impact on the output.

Add a term to the loss that penalizes θ for having large magnitude

$\sum \theta_j^2 = \theta^\top \theta$

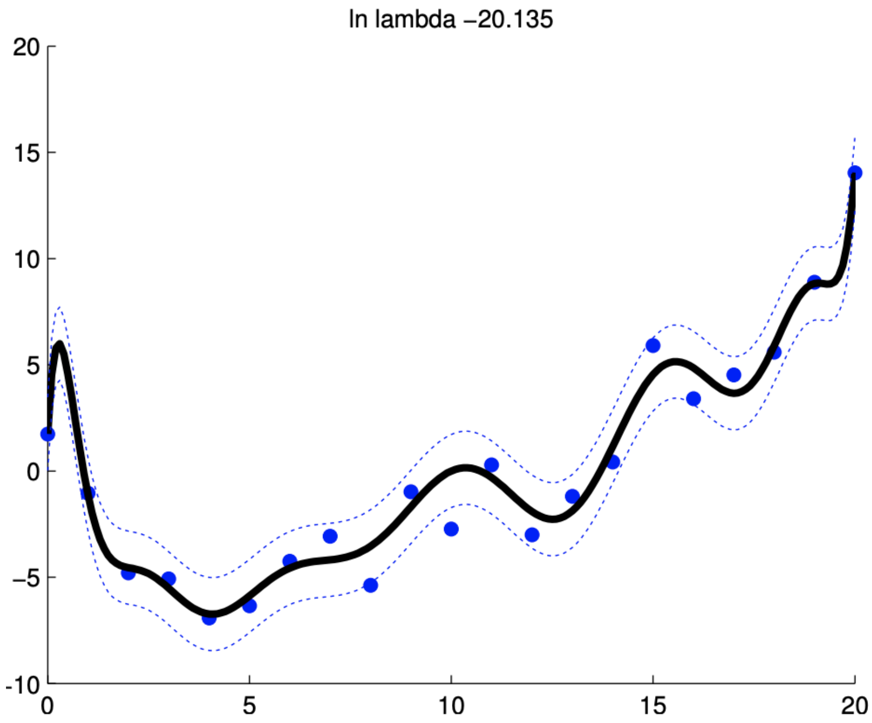
$$\mathcal{J}_S(h_\theta) = \frac{1}{2N}(\Phi\theta - Y)^\top(\Phi\theta - Y) + \underline{\frac{\lambda}{2}\|\theta\|^2}$$

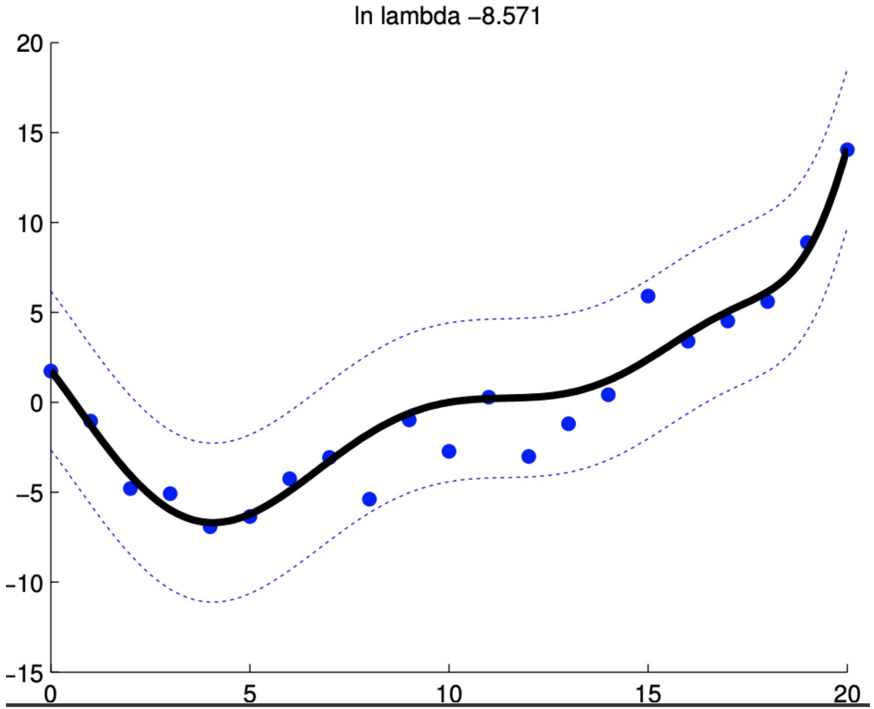
Regularization - solving for θ

$$
\begin{aligned}
0 &= \nabla_\theta \mathcal{J}_S(h_\theta) \\
&= \nabla_\theta \left[\frac{1}{2N} (\Phi\theta - Y)^\top (\Phi\theta - Y) + \frac{\lambda}{2} \theta^\top \theta \right] \\
&= \frac{1}{N} [\Phi^\top \Phi\theta - \Phi^\top Y] + \lambda\theta \\
&= \Phi^\top \Phi\theta + N\lambda I\theta - \Phi^\top Y \\
&= (\Phi^\top \Phi + N\lambda I)\theta - \Phi^\top Y \\
\theta &= (\Phi^\top \Phi + N\lambda I)^{-1} \Phi^\top Y
\end{aligned}
$$

Regularization - example

Fitting a degree 14 polynomial





Regularization - notes

Alternatives?

- Instead of making most weights small, make some of them actually = 0 (L_0, aka **model selection**)
- No closed form solution, but efficient optimizations exist (LASSO, L_1 approximation $\sum|\theta_i| = ||\theta||_1$)

A general approach to keeping models from becoming too complex: add terms to the loss function that measure the complexity of the model

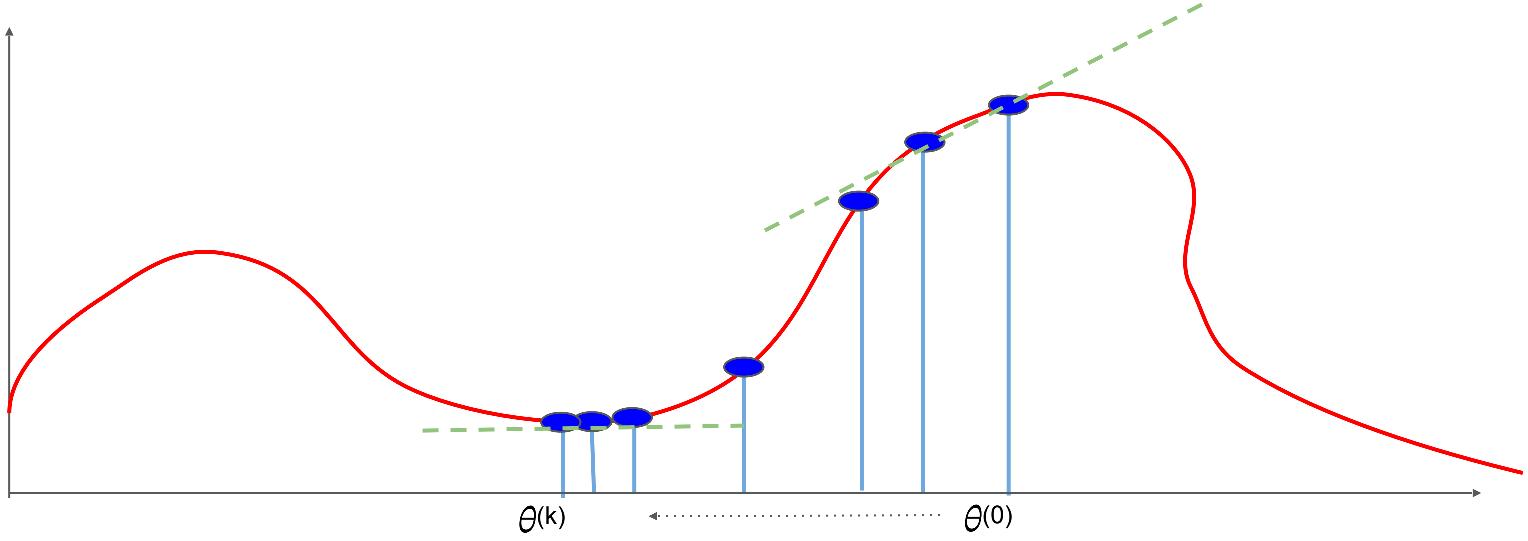
Question: How big should λ be?

Transformed the question from “How should we design $\{\phi\}$ to not be *too* complex, but complex *enough*” to “what’s the right value for this one parameter”

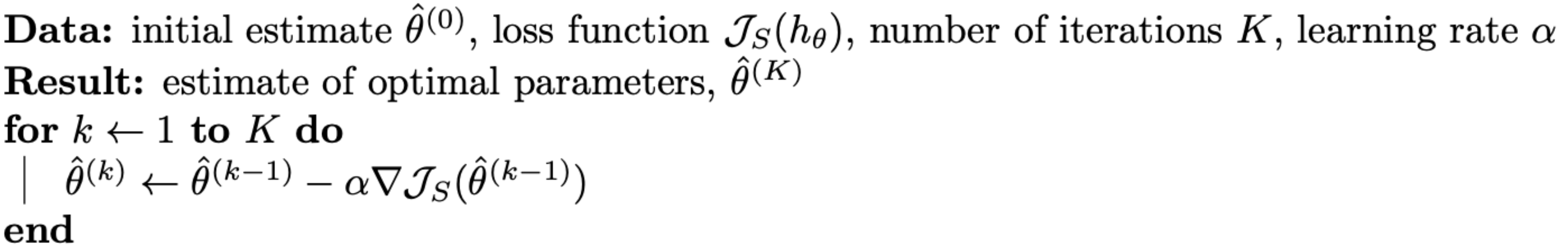
Gradient Descent

We needed an iterative method when the exact solution isn't tractable

Algorithm Idea:
Take a small step in the **direction** down the gradient, loop until done



Gradient Descent - Algorithm



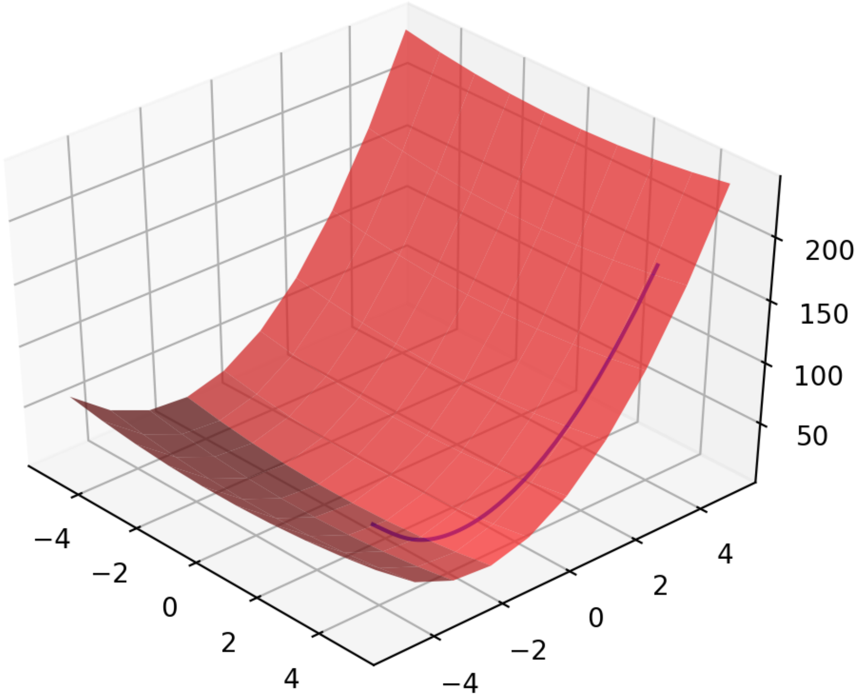
```
**Data:** initial estimate $\hat{\theta}^{(0)}$, loss function $\mathcal{J}_S(h_\theta)$, number of iterations $K$, learning rate $\alpha$
**Result:** estimate of optimal parameters, $\hat{\theta}^{(K)}$
**for** $k \leftarrow 1$ **to** $K$ **do**
    $\hat{\theta}^{(k)} \leftarrow \hat{\theta}^{(k-1)} - \alpha \nabla \mathcal{J}_S(\hat{\theta}^{(k-1)})$
**end**
```

The **learning rate** α controls how far along the gradient to move at each step

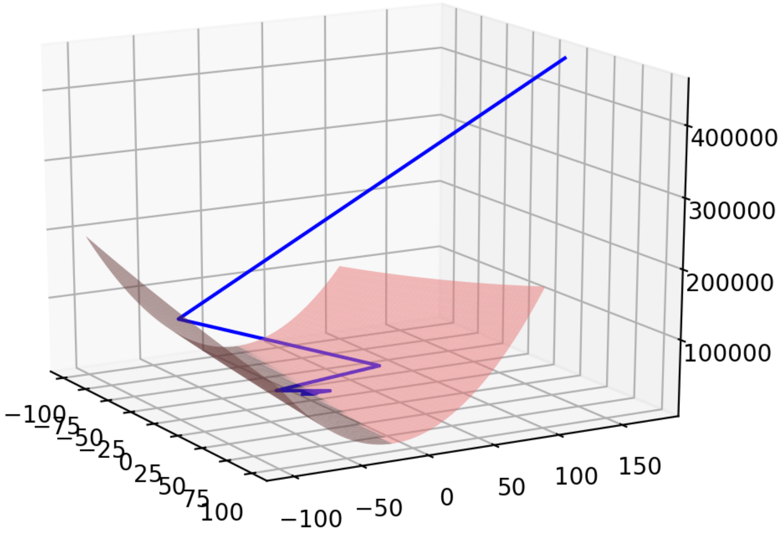
Works for just about any objective function as long as we can find the gradient

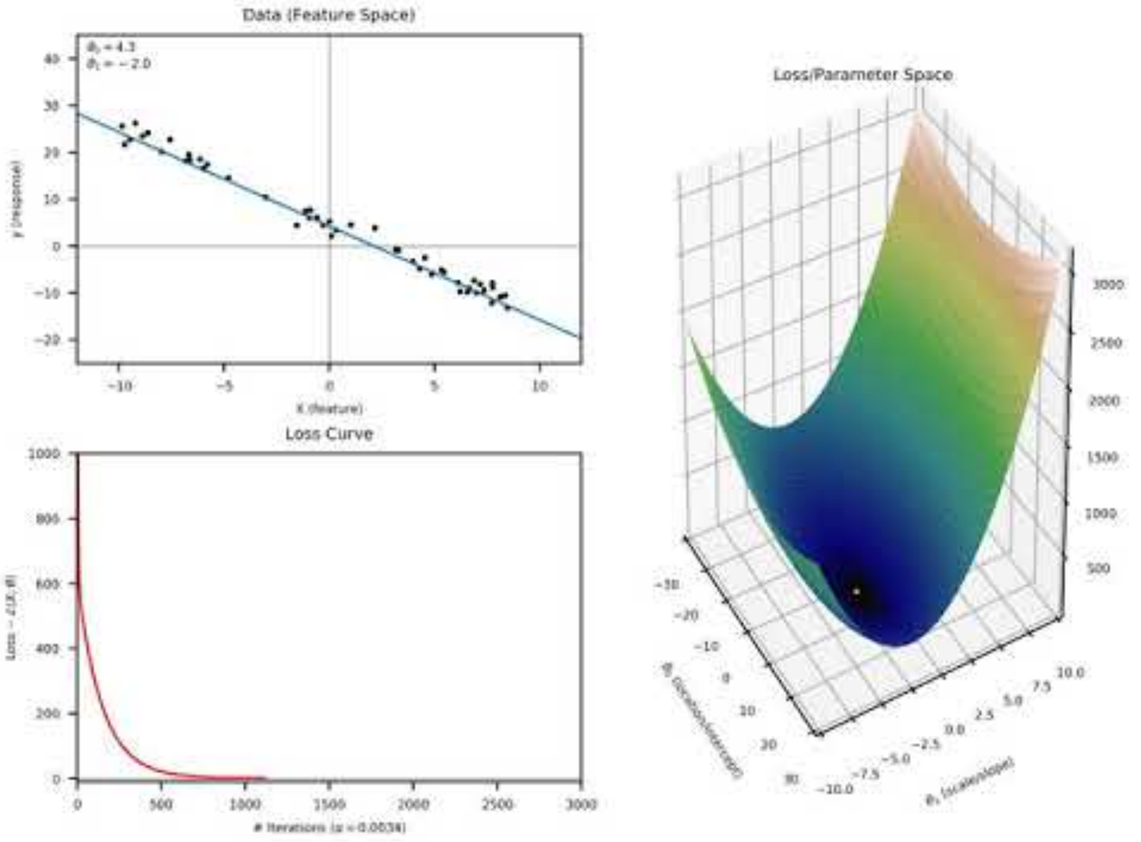
Since the loss for Linear Regression is **convex**, gradient descent will find the **global optima**

Gradient Descent converging



Gradient Descent diverging





(thanks to Kanad Khanna for the visualization)

Gradient Descent - notes

Learning rate is important!

- Need to be able to compute the gradient! (LASSO?)
- Slows down once close to minima. Fix: adaptive learning rate
- Can get caught in local optima. Fix: momentum
- Variant - Stochastic Gradient Descent: update using single data point sampled from the training dataset
  - Can escape local optima
  - Can speed up convergence if the gradient is “noisy”
  - Each iteration is faster to compute
- “Faster” optimization methods exist (for some classes of objective functions)
  - Newton’s, BFGS (second order methods)
  - Conjugate Gradient (requires sym. pos. def. matrix)

Summary and preview

Wrapping up

- We can use linear regression to fit nonlinear data with **basis function expansion**
- We can use **regularization** to combat overfitting when the hypothesis class is complex
- **Gradient Descent** is an iterative **optimization** method that takes a small step in the direction of the gradient each iteration

For next time

- A new model class: Neural Networks