Business Analytics
Convex Optimization in Machine Learning

Prof. Bichler

Decision Sciences & Systems

Department of Informatics

Technical University of Munich

Course Content

- Introduction
- Regression Analysis
- Regression Diagnostics
- Logistic and Poisson Regression
- Naive Bayes and Bayesian Networks
- Decision Tree Classifiers
- Data Preparation and Causal Inference
- Model Selection and Learning Theory
- Ensemble Methods and Clustering
- High-Dimensional Problems
- Association Rules and Recommenders
- Neural Networks
- **Convex Optimization in Machine Learning**

Convex Optimization in Machine Learning

Many machine learning problems can be cast as minimizing a loss function:

- the OLS estimator for the linear regression (already invented by Gauss!)
- the use of regularization in lasso and the ridge regression
- the ML estimator of a logistic regression
- backpropagation in neural networks
- etc.

=> **Convex optimization** plays a crucial role, because it is often easier to "convexify" a problem (make it convex optimization friendly), rather than to use non-convex optimization.

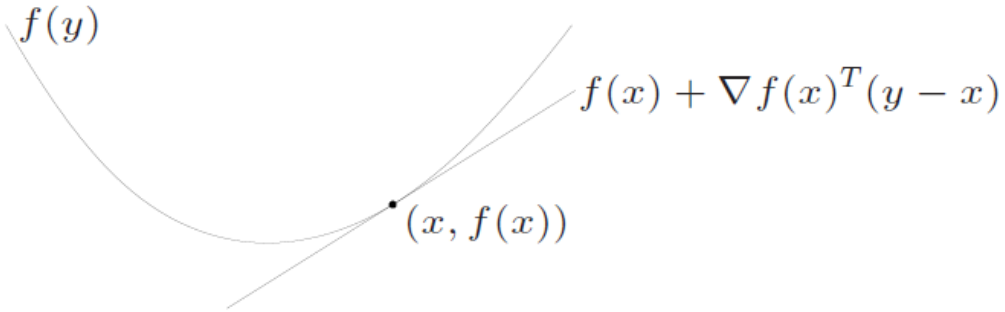
In this class, we **revisit** material from different classes in our course through the lense of convex optimization. This should aid your understanding of previous classes and how machine learning as a combination of statistics and optimization.

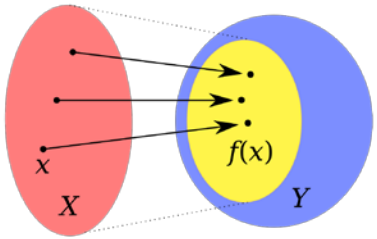
Differentiable Convex Functions

Definition of convexity via the gradient

A differentiable function f is convex, iff dom f is convex and

$$f(y) - f(x) \geq \nabla f(x)^T (y - x)\ \forall x, y \in dom\ f$$
$$\equiv f(x) - f(y) \leq \nabla f(x)^T (x - y)\ \forall x, y \in dom\ f$$





The red oval is dom f

The affine function is a tangent of f at $(x, f(x))$.

Definition via the Hessian matrix:

A twice differentiable function f is convex, iff dom f is convex and the Hessian matrix is positive semidefinite: $\nabla^2 f(x) \geq 0\ \forall x \in dom\ f$

Linear Regression

$$\begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} = \begin{pmatrix} 1 & x_{11} & x_{12} & \dots & x_{1p} \\ 1 & x_{21} & x_{22} & \dots & x_{2p} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_{n1} & x_{n2} & \dots & x_{np} \end{pmatrix} \begin{pmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_p \end{pmatrix} + \begin{pmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \varepsilon_n \end{pmatrix}$$

y	$\boldsymbol{X}$	β	$+ \varepsilon$
$(n \times 1)$	$(n \times (p+1))$	$((p+1) \times 1)$	$(n \times 1)$

OLS Estimation

- Sample-based counter part to population regression model:

$$y = \mathbf{X}\beta + \varepsilon$$
$$y = \mathbf{X}\hat{\beta} + e$$

- OLS requires choosing values of the estimated coefficients, such that Residual Sum of Squares (RSS) is as small as possible for the sample

$$RSS = e^T e = (y - \mathbf{X}\hat{\beta})^T (y - \mathbf{X}\hat{\beta})$$

- Need to differentiate with respect to the unknown coefficients

Least Squares Estimation

$\mathbf{X}$ is $n \times (p+1)$, y is the vector of outputs

RSS(β) = $(y - \mathbf{X}\beta)^T (y - \mathbf{X}\beta)$

If $\mathbf{X}$ is full rank, then $\mathbf{X}^T\mathbf{X}$ is positive definite

$$RSS = (y^T y - 2\beta^T \mathbf{X}^T \mathrm{y} + \beta^T \mathbf{X}^T \mathbf{X}\beta)$$

$$\frac{\partial RSS}{\partial \beta} = -2\mathbf{X}^T \mathrm{y} + 2\mathbf{X}^T \mathbf{X}\beta = 0$$ First-order condition

$$\beta = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T y$$

$$\hat{y} = \underbrace{\mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T}\, y$$

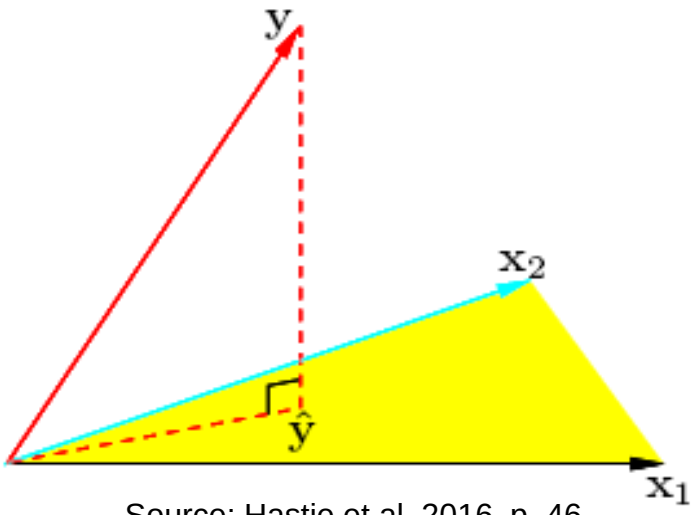
"Hat" or projection matrix H

Quadratic Optimization in Least Squares Estimation

- Least square estimates in $\mathbb{R}^n$
- We minimize the squared distance between observed and estimated values: RSS(β)=$||y - \mathbf{X}\beta||^2$, s.t. residual vector $y - \hat{y}$ is orthogonal to this subspace $\mathbf{X}$.
- We found an analytical solution: $\hat{y} = \mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T y$

Definition (Projection):
The set $X \subset \mathbb{R}^n$ is non-empty, closed and convex. For a fixed $y \in \mathbb{R}^n$ we search a point $\hat{y} \in X$, with the smallest distance to y (wrt. the Euclidean norm), i.e. we solve the minimization problem

$$P_X(y) = \min_{\hat{y} \in X} \|y - \hat{y}\|^2$$



Source: Hastie et al. 2016, p. 46

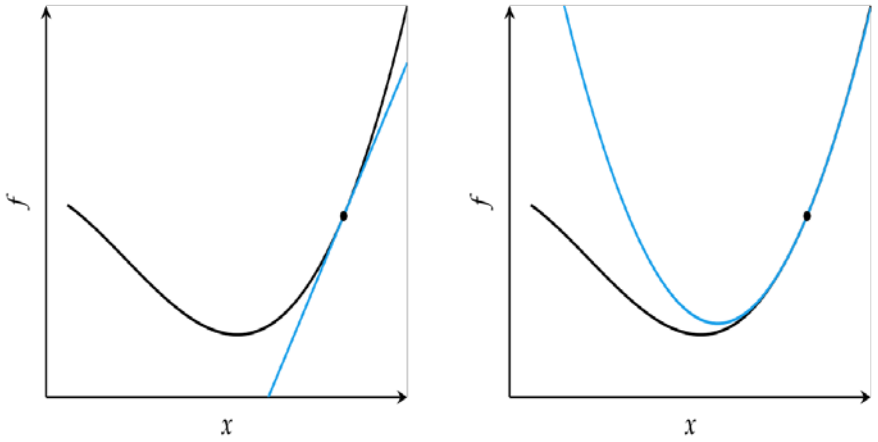
Recap: Taylor Polynomials for Approximation

Taylor polynomials are approximations of a function f at a point x.

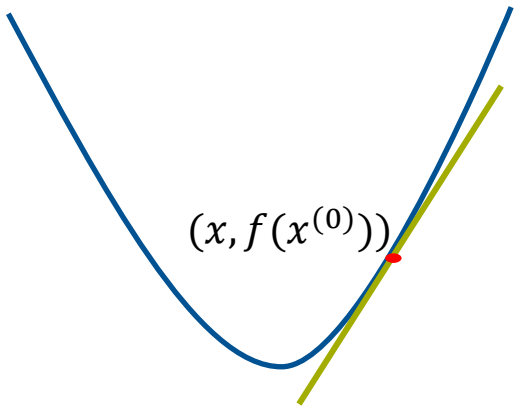
Univarate: $P(x) = f\left(x^{(0)}\right) + f'\left(x^{(0)}\right)\left(x - x^{(0)}\right) + \frac{f''\left(x^{(0)}\right)}{2!}\left(x - x^{(0)}\right)^2 + \cdots$

Multivariate: $P(x) = f\left(x^{(0)}\right) + \nabla f\left(x^{(0)}\right)^T\left(x - x^{(0)}\right) + \frac{1}{2!}\left(x - x^{(0)}\right)^T H_f\left(x^{(0)}\right)\left(x - x^{(0)}\right) + \cdots$

The partial sum formed by the first $k + 1$ terms of a Taylor series is a polynomial of degree k that is called the kth Taylor polynomial of the function.
The approxmation becomes generally better as k increases.



Gradient Descent



Unconstrained minimization: $\min f(x), s.t. x \in X$

Derivation via first-order Taylor approximation (i.e., a linear function)

$$f(x) \approx f\left(x^{(0)}\right) + \left\langle \nabla f\left(x^{(0)}\right), x - x^{(0)} \right\rangle$$

This way, we would follow the gradient very long.

The approximation is only close in the vicinity of $x^{(0)}$.

Therefore, we add a penalty term using the following formula starting with $k = 0$.

$$x^{(k+1)} = \operatorname{argmin}_x f(x^{(k)}) + \left\langle \nabla f\left(x^{(k)}\right), x - x^{(k)} \right\rangle + \frac{1}{2\alpha} \left\| x - x^{(k)} \right\|^2$$

The first-order condition wrt. x is:

$$0 + \nabla f\left(x^{(k)}\right) + \frac{1}{\alpha}\left(x - x^{(k)}\right) = 0$$

$$x^{(k+1)} = x^{(k)} - \alpha \nabla f\left(x^{(k)}\right)$$

OLS Estimation via Gradient Descent

Instead of the analytical solution, we can use gradient descent to minimize the sum of squared residuals.

$$\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x$$

We use the mean squared error as the *loss function*

$$L(\beta) = \frac{1}{n}\sum_i e_i^2 = \frac{1}{n}\sum_i (y_i - (\hat{\beta}_0 + \hat{\beta}_1 x_i))^2$$

Partial derivatives of the loss function wrt. $\hat{\beta}_1$ and $\hat{\beta}_0$ using the chain rule:

$$\frac{\partial L}{\partial \hat{\beta}_1} = \frac{1}{n}\sum_i 2(y_i - \hat{\theta}_0 - \hat{\beta}_1 x_i)(-x_i) = \frac{-2}{n}\sum_i x_i(y_i - \hat{y}_i)$$

$$\frac{\partial L}{\partial \hat{\beta}_0} = \frac{-2}{n}\sum_i (y_i - \hat{y}_i)$$

OLS Estimation via Gradient Descent

Pseudo code:

Start with $\hat{\beta}_0 = \hat{\beta}_1 = 0$

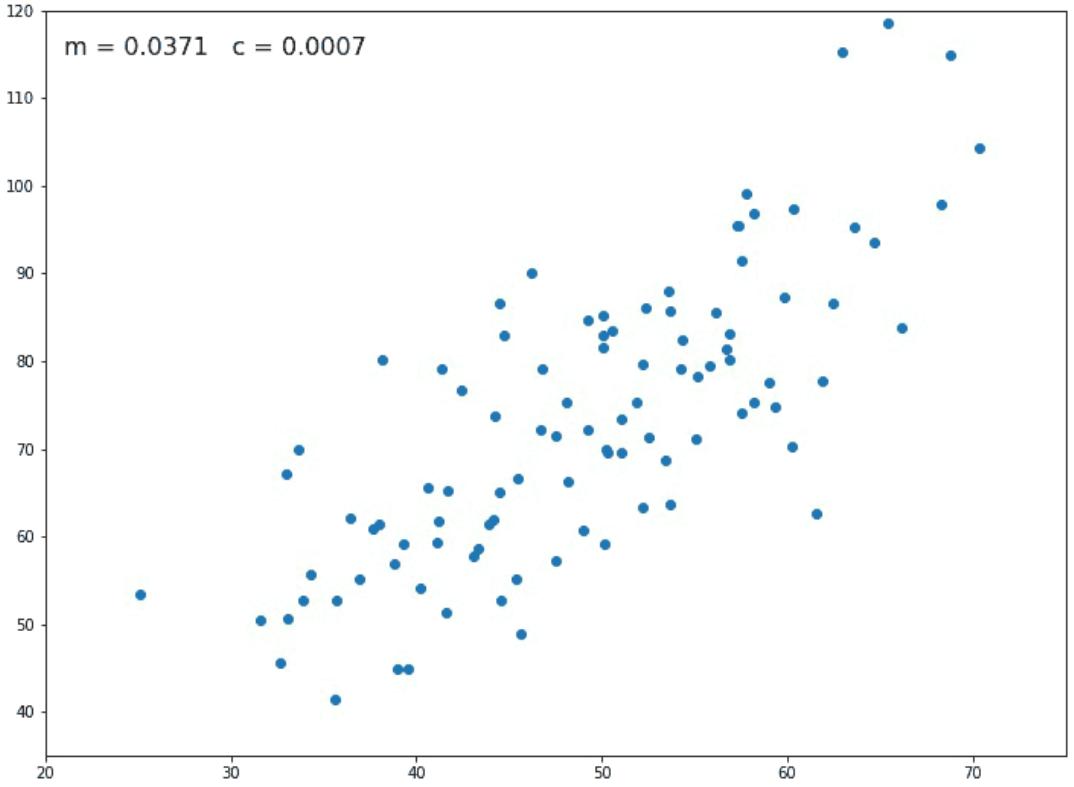
Repeat until the objective of the loss function is very small

$$\hat{\beta}_1 := \hat{\beta}_1 - \alpha \frac{\partial L}{\partial \hat{\beta}_1}; \quad \hat{\beta}_0 := \hat{\beta}_0 - \alpha \frac{\partial L}{\partial \hat{\beta}_0}$$

Python code snippet:

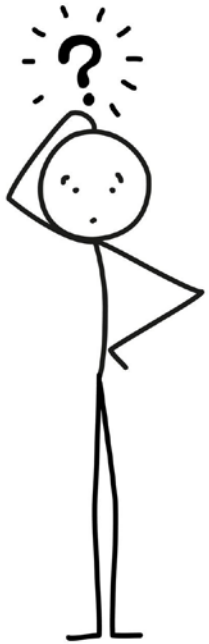
```
# Initialization of variables
b = 0
a = 0
L = 0.0001 # learning rate
epochs = 1000 # number of iterations
n = float(len(X)) # number of elements in X
# gradient descent
for i in range(epochs):
          Y_pred = b*X + a # The current predicted value of Y
          D_b = (-2/n) * sum(X * (Y - Y_pred)) # Derivative wrt b
          D_a = (-2/n) * sum(Y - Y_pred) # Derivative wrt a
          b = b - L * D_b # Update b
          a = a - L * D_a # Update a
print (b, a)
```

Progress of Gradient Descent



Note that there is no need to compute the minimum via gradient descent for the linear regression, because we have a closed-form solution (see slide 7).

What does Taylor approximation have to do with gradient descent?



Second Order Methods

- At a high level, gradient descent uses the first order Taylor expansion to approximate the function locally.
- This doesn't take into account the curvature of the function, i.e., how quickly the gradient is changing. Consequently, gradient descent can dramatically overshoot the optimum with a fixed step size.
- Instead of using only the first derivatives, *second order methods* use the first three terms of the multivariate Taylor series expansion

$$f(x) \approx f\left(x^{(0)}\right) + \nabla f\left(x^{(0)}\right)^T \left(x - x^{(0)}\right) + \frac{1}{2!}\left(x - x^{(0)}\right)^T H_f\left(x^{(0)}\right)\left(x - x^{(0)}\right)$$

$$\nabla^2 f(\hat{x}) = H_f(\hat{x}) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2}(\hat{x}) & \dots & \frac{\partial^2 f}{\partial x_1 \partial x_n}(\hat{x}) \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1}(\hat{x}) & \dots & \frac{\partial^2 f}{\partial x_n^2}(\hat{x}) \end{bmatrix}$$

Second Order Methods

Instead of using only the first derivatives, second order methods use the first three terms of the multivariate Taylor series expansion

Newton's method

$$f\left(x^{(0)}+\Delta x\right) \approx P(x) = f\left(x^{(0)}\right) + f'\left(x^{(0)}\right)\Delta x + \frac{1}{2}f''\left(x^{(0)}\right)(\Delta x)^2$$

$$\frac{d}{\Delta x}P(x) = f'\left(x^{(0)}\right) + f''\left(x^{(0)}\right)\Delta x = 0$$

$$\Delta x = -\frac{f'\left(x^{(0)}\right)}{f''\left(x^{(0)}\right)}$$

- Univariate update rule: $x^{(k+1)} = x^{(k)} - \frac{f'\left(x^{(k)}\right)}{f''\left(x^{(k)}\right)}$
- Multivariate update rule: $x^{(k+1)} = x^{(k)} - \left(H_f\left(x^{(k)}\right)\right)^{-1}\nabla f\left(x^{(k)}\right)$

Compare Newton's method to gradient descent: $x^{(k+1)} = x^{(k)} - \alpha\nabla f\left(x^{(k)}\right)$

Minimal Example of Newton‘s Method

min $f(x_1, x_2) = 3 + (x_1 - 1.5x_2)^2 + (x_2 - 2)^2$

Start point: $x^{(0)} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$

Newton step: $\Delta x = -\left[H\left(x^{(0)}\right)\right]^{-1} \nabla f(x^{(0)})$

Gradient

$$\nabla f\left(x^{(0)}\right) = \begin{pmatrix} -1 \\ -0.5 \end{pmatrix}$$

Inverse Hessian matrix

$$H\left(x^{(0)}\right) = \begin{bmatrix} 2 & -3 \\ -3 & 6.5 \end{bmatrix}$$

$$H^{-1}\left(x^{(0)}\right) = \begin{bmatrix} 1.625 & 0.75 \\ 0.75 & 0.5 \end{bmatrix}$$

Newton step:

$$\Delta x = -\begin{bmatrix} 1.625 & 0.75 \\ 0.75 & 0.5 \end{bmatrix} \begin{pmatrix} -1 \\ -0.5 \end{pmatrix} = \begin{pmatrix} 2 \\ 1 \end{pmatrix}$$

Next iteration:

$$x^{(1)} = x^{(0)} + \Delta x = \begin{pmatrix} 1 \\ 1 \end{pmatrix} + \begin{pmatrix} 2 \\ 1 \end{pmatrix} = \begin{pmatrix} 3 \\ 2 \end{pmatrix}$$

repeat until convergence

Newton's Method

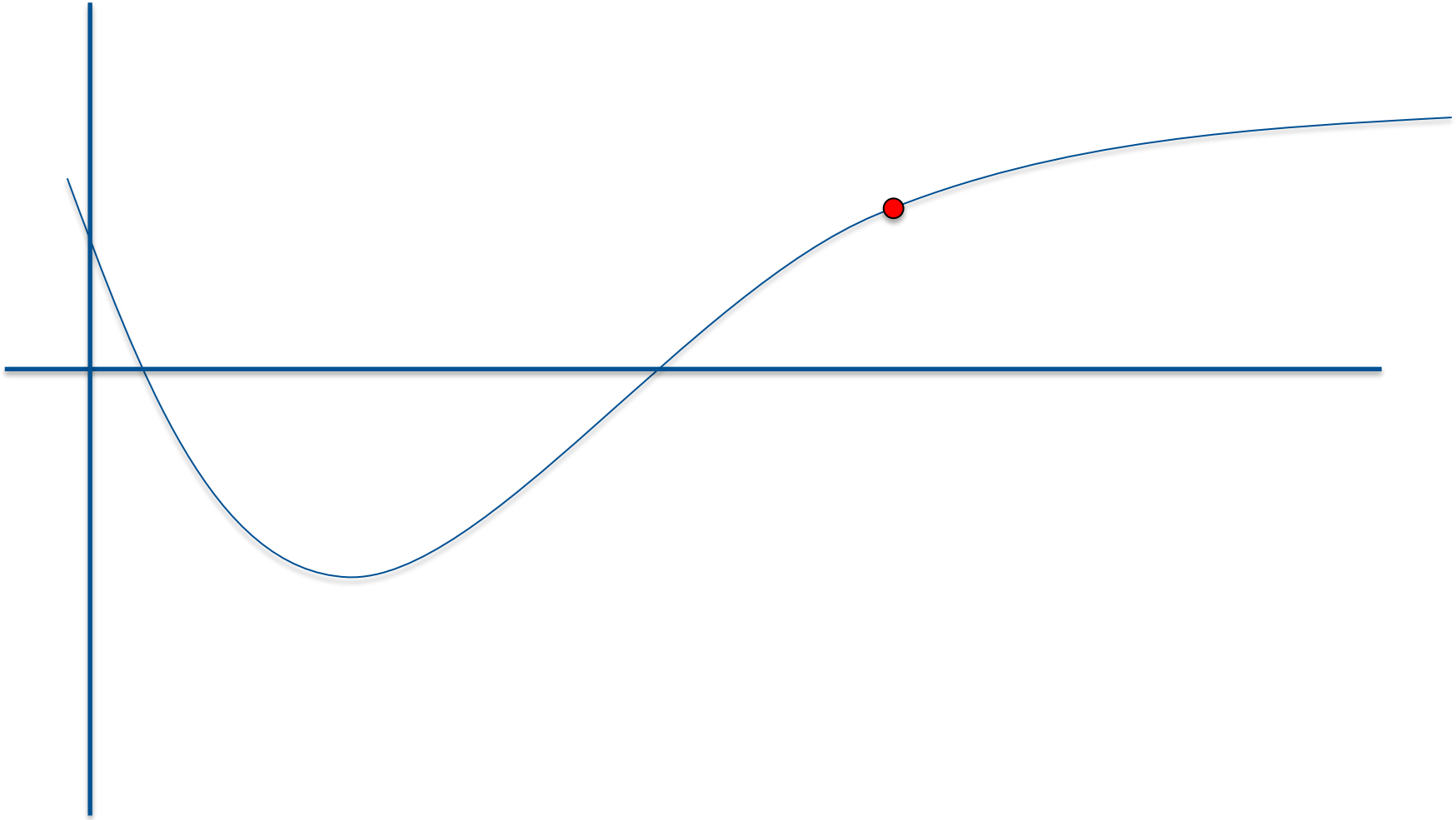
Newton's method is a *root-finding algorithm*, i.e., it seeks a solution to the equation $g(x) = 0$

Newton's method *for optimization* can be seen as a way to find the zeros of the first derivative.

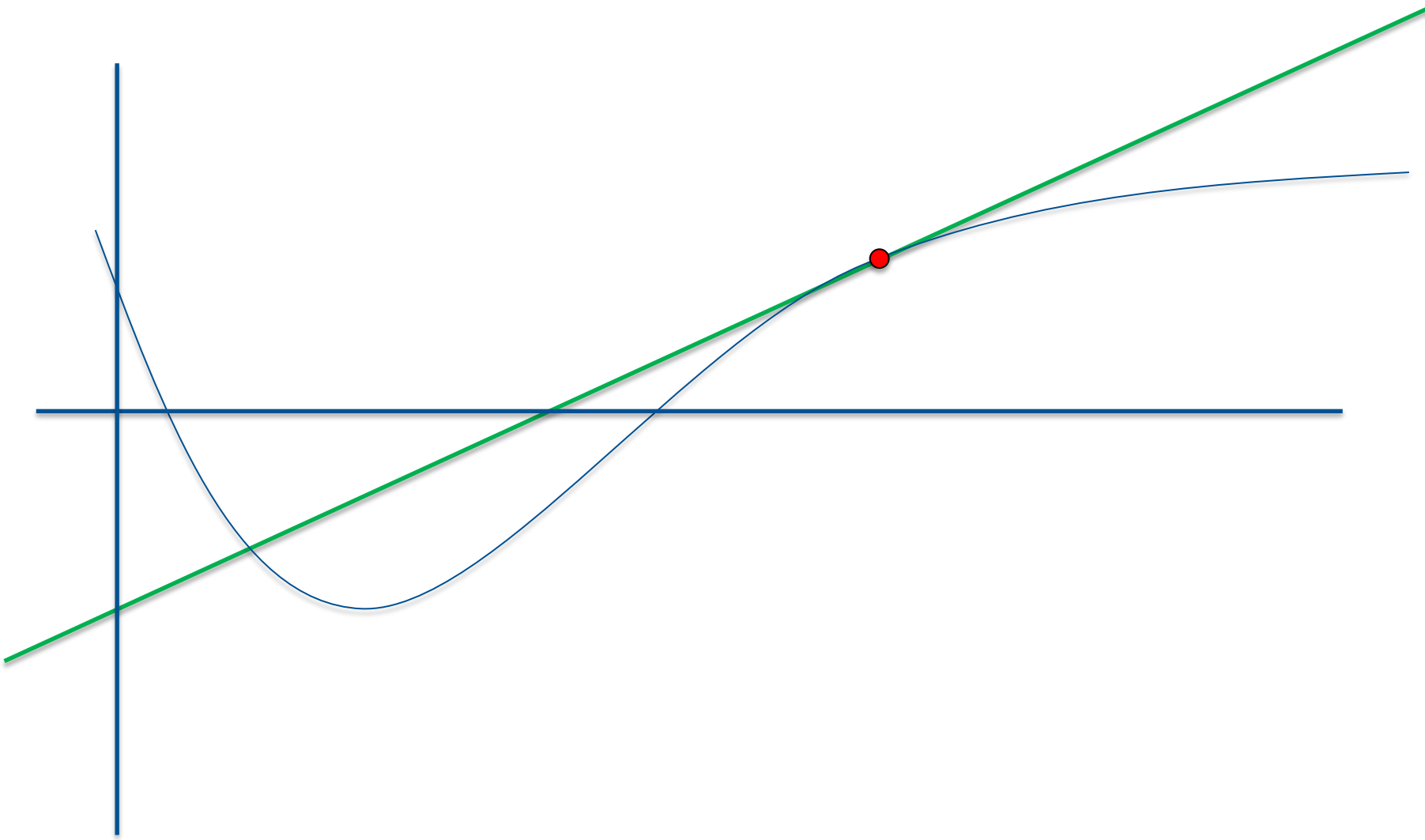
- Approximate: $f'(x) \approx f'(x_t) + f''(x_t)(x - x_t)$
- Update: $x_{t+1} = x_t - f'(x_t)/f''(x_t)$

This is equivalent to minimizing the second order approximation!

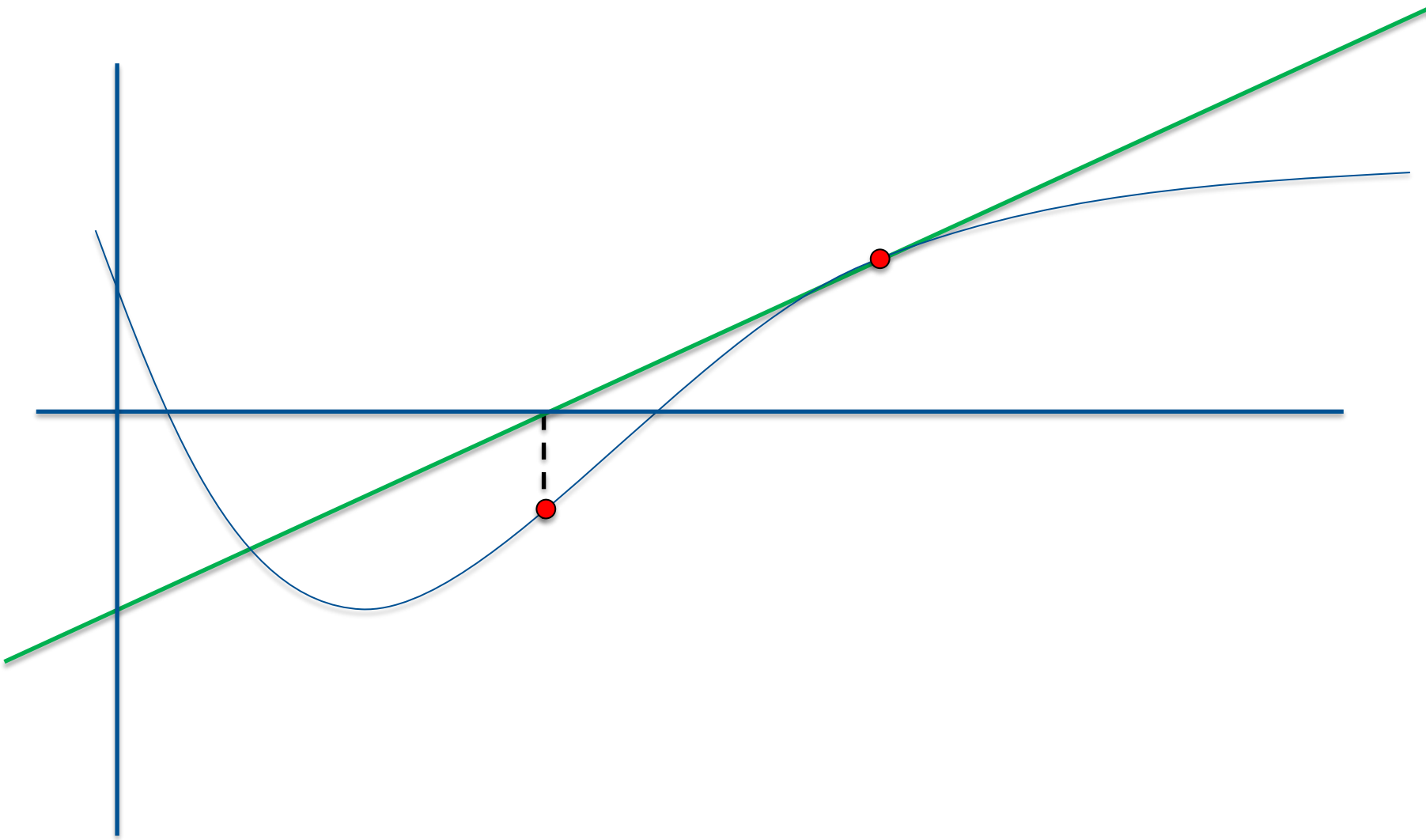
Newton’s Method



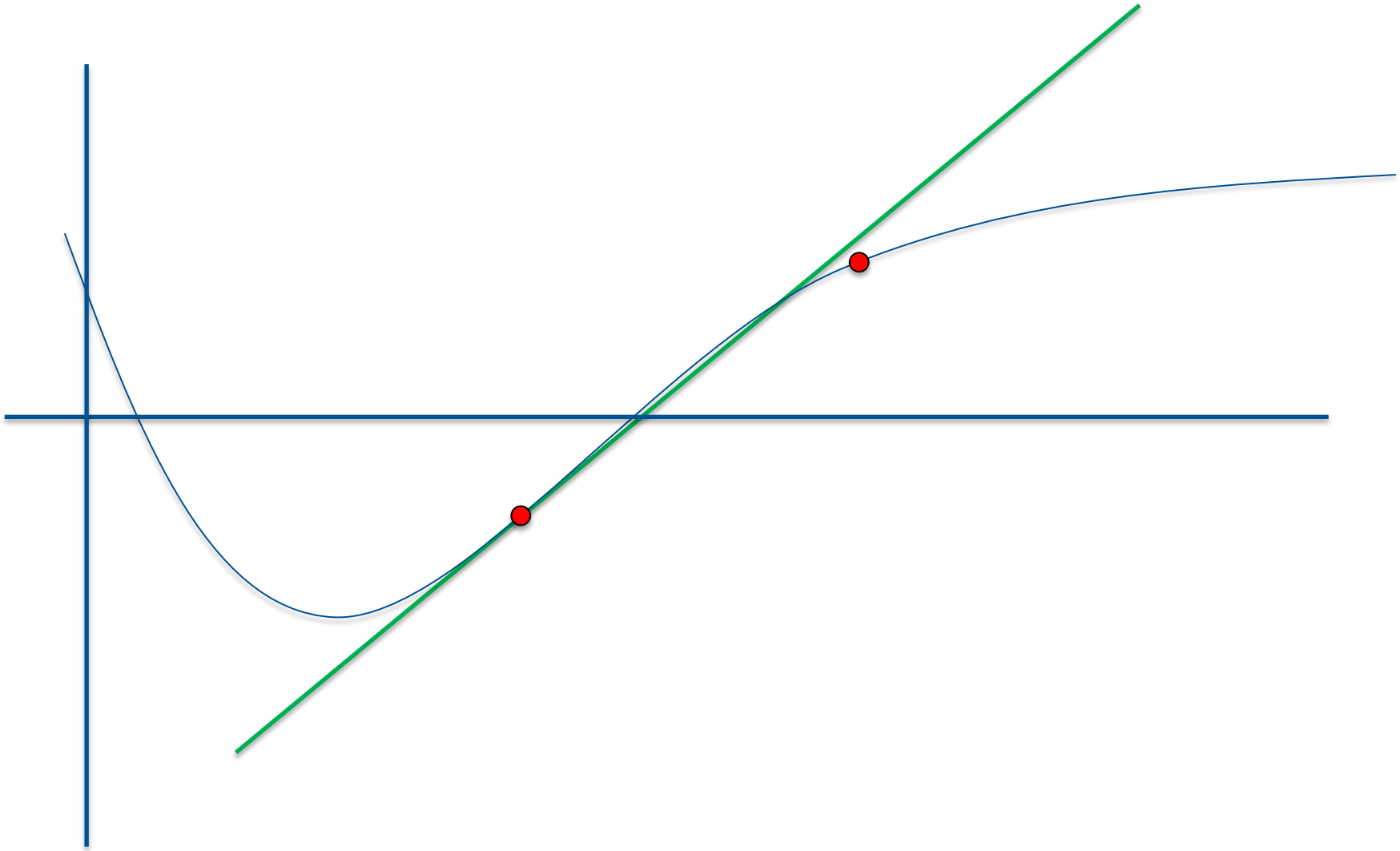
Newton’s Method



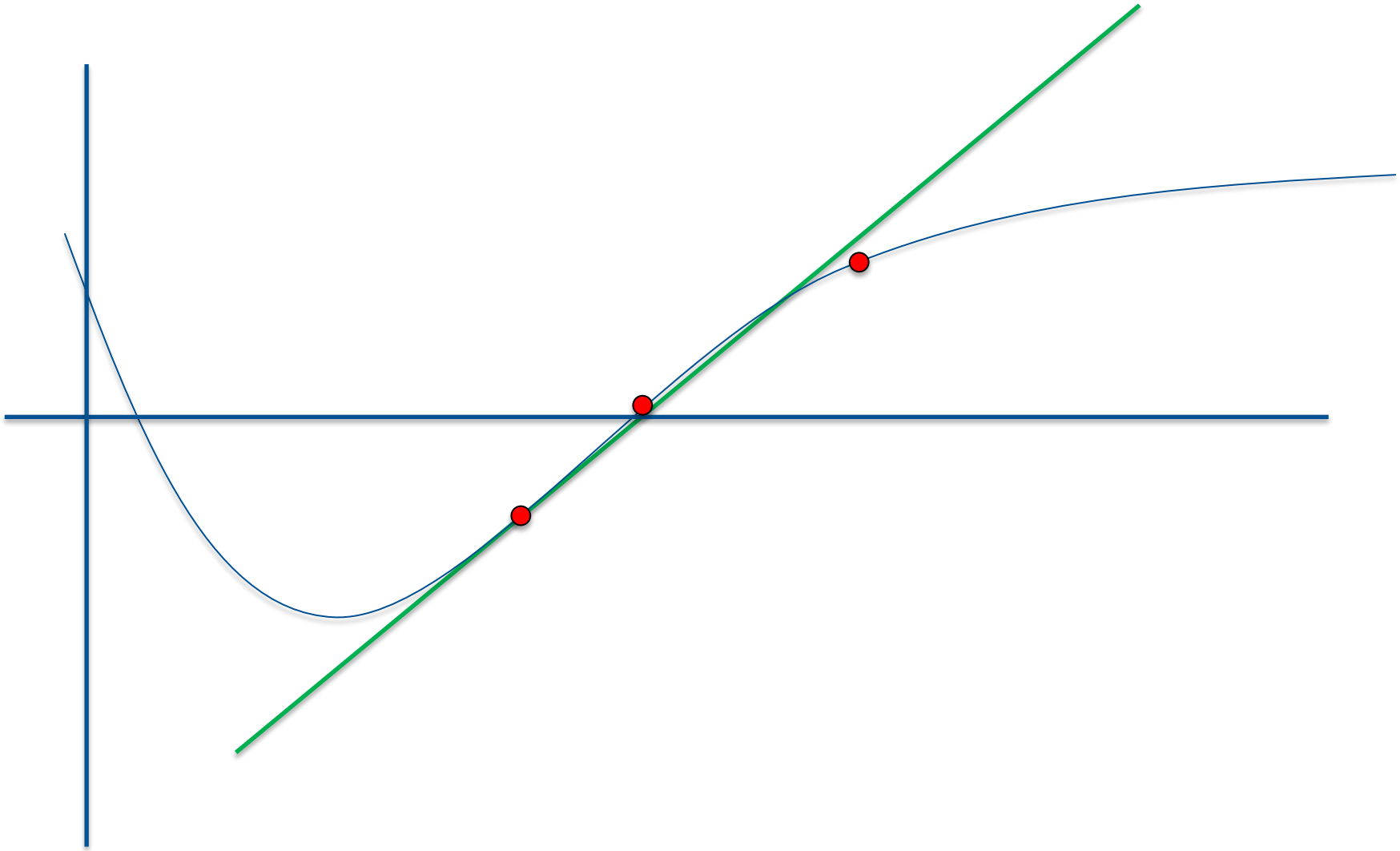
Newton’s Method



Newton’s Method

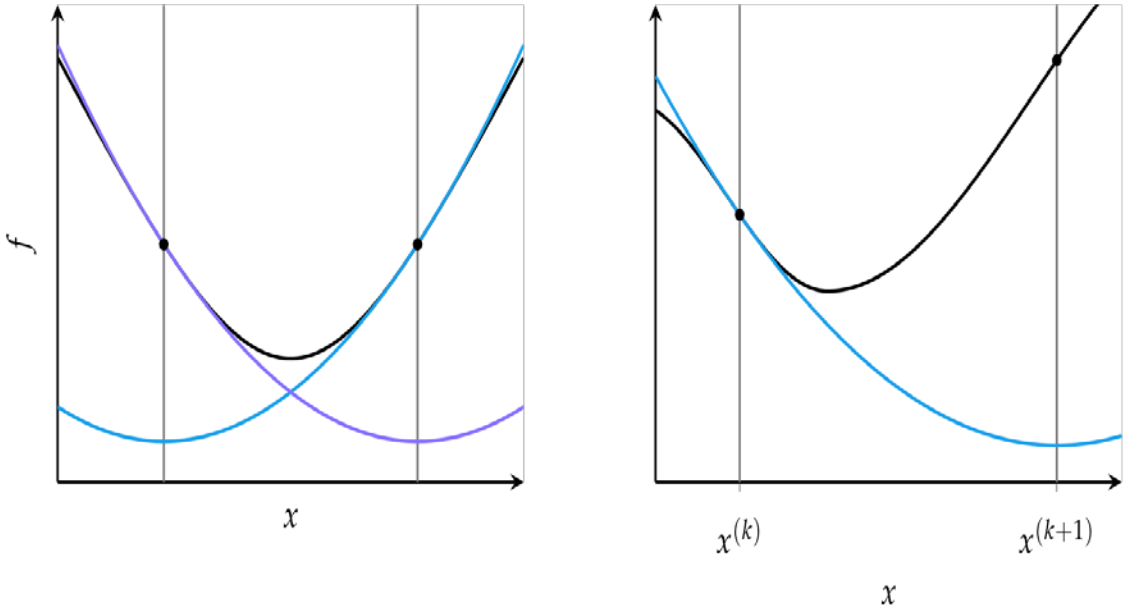


Newton’s Method



Newton’s Method

Common causes of error in Newton’s method

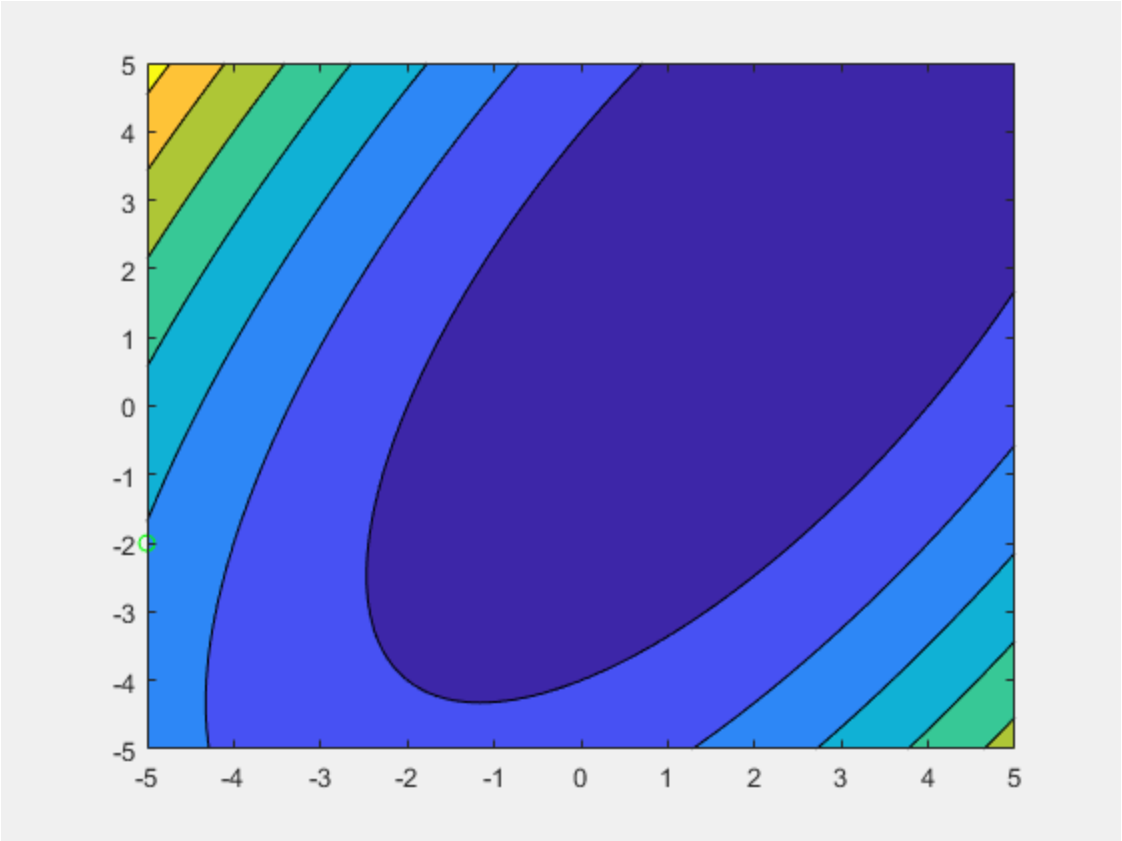


Multivariate Newton's Method

Update rule: $x^{(k+1)} = x^{(k)} - \left(H_f\left(x^{(k)}\right)\right)^{-1} \nabla f(x^{(k)})$

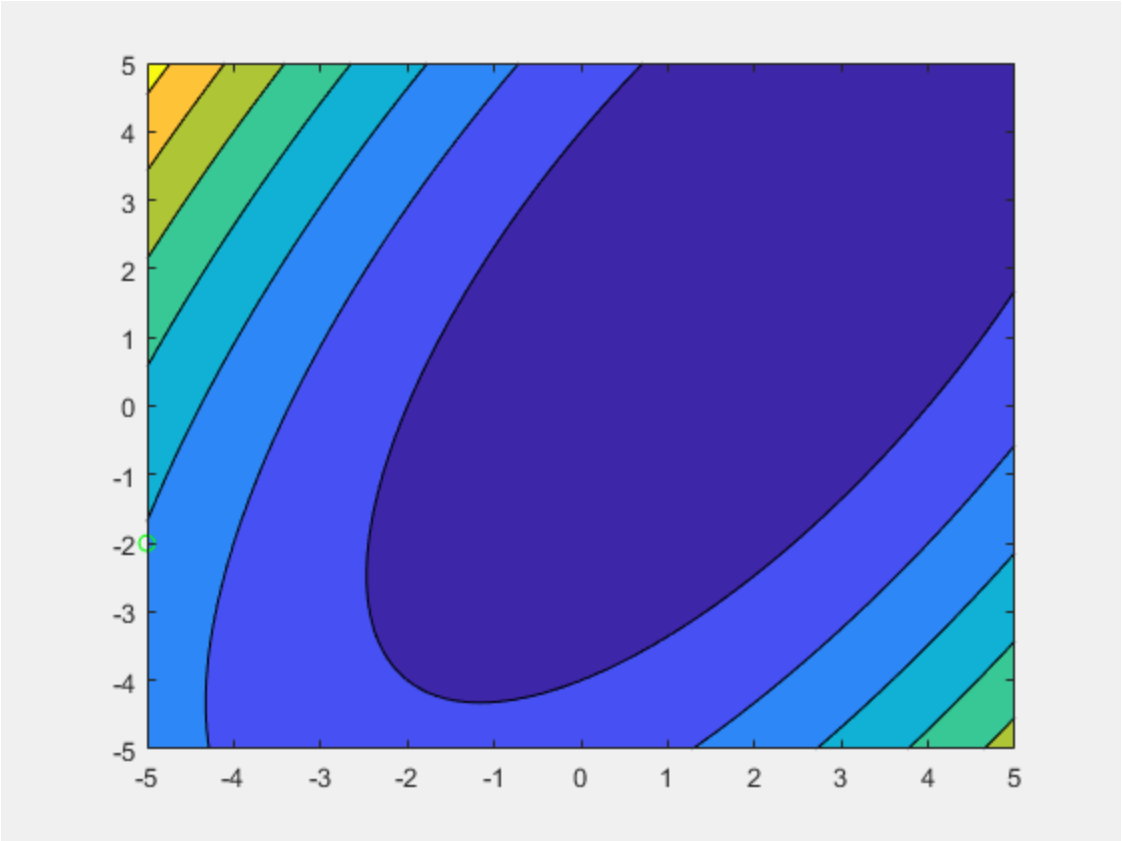
- Inverses do not always exist, if the inverse doesn't exist, then there may be no solution or infinitely many solutions.
- Computing the Hessian matrix is computationally expensive.
- Computing the inverse can be expensive: requires $O(n^3)$ operations for an $n \times n$ matrix.
- Overall, Newton's method requires more operations and more memory than gradient descent at each iteration. However, it converge in a lesser number of iterations.
- For multivariate functions, the inverse Hessian may not be available or infeasible to compute, so can be approximated using a variety of Quasi-Newton methods
  - Davidon-Fletcher-Powell (DFP) method
  - Broyden-Fletcher-Goldfarb-Shanno (BFGS) method
  - Limited-memory BFGS (L-BFGS) method

Newton’s Method



$$f(x_1, x_2) = 0.1x_1^4 + (x_1 - 2)^2 - (x_1 - 2)(x_2 - 2) + 0.5(x_2 - 2)^2$$

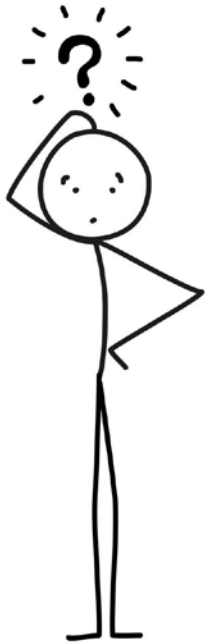
Gradient Descent



$$f(x_1, x_2) = 0.1x_1^4 + (x_1 - 2)^2 - (x_1 - 2)(x_2 - 2) + 0.5(x_2 - 2)^2$$

with diminishing step size rule

Where do the update steps of gradient descent and Newton‘s method differ?



Gauss-Markov Theorem

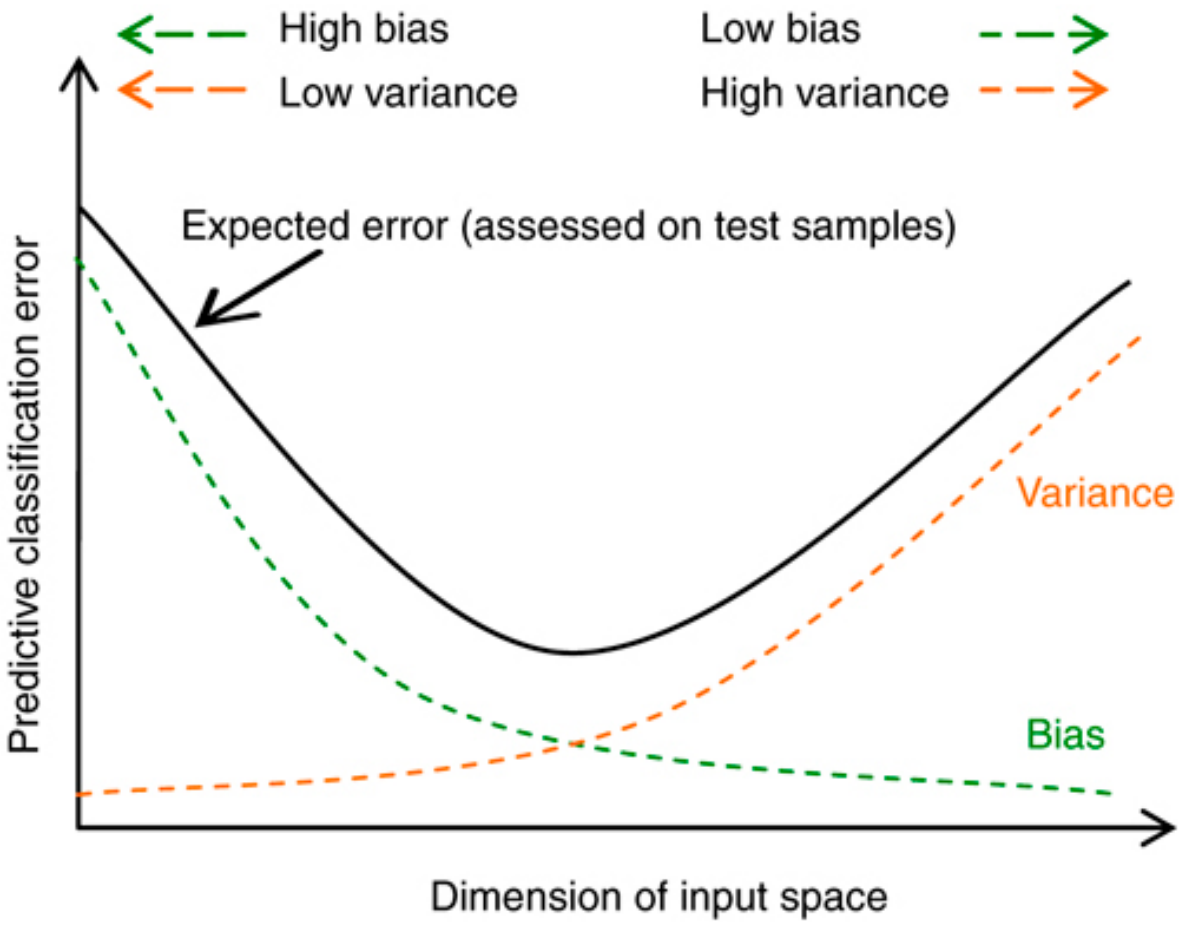
The Gauss-Markov theorem states that in a linear regression model in which the errors

- have expectation zero and
- are uncorrelated and
- have equal variances,

the *best linear unbiased estimator* (BLUE) of the coefficients is given by the ordinary least squares (OLS) estimator.

- „Unbiased“ means $E(\hat{\beta}_j) = \beta_j$
- „Best“ means giving the ***lowest variance*** of the estimate as compared to other linear unbiased estimators.
  - Restriction to unbiased (linear) estimation is not always the best (will be discussed in the context of the ridge regression later)

Bias-Variance Tradeoff



Regularization

Objective function:

$$J(\beta) = L(\beta) + \Omega(\beta)$$

- $L(\beta)$ is training loss: how well model fit on training data.
- $\Omega(\beta)$ is regularization, measures complexity of model.
- Lower training loss result in more predictive model.
- Lower regularization result in simpler (more biased) model.

Regularization methods introduce bias into the regression solution that can reduce variance considerably relative to the ordinary least squares (OLS) solution.

Ridge Regression

Ridge coefficient minimize a penalized RSS. The parameter $\lambda>0$ penalizes β_j proportional to its size β_j^2.

$$\hat{\beta}^{ridge} = \arg\min_{\beta}\{\sum_i (y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j)^2 + \lambda\sum_{j=1}^{p}\beta_j^2\}$$

Or

$$Minimize_{\beta}\{\sum_i (y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j)^2\}$$

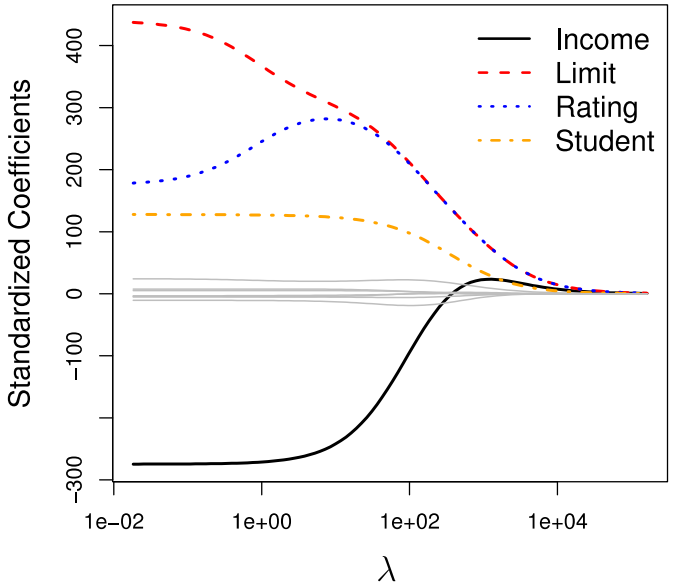
$$subject\ \text{to}\ \sum_{j=1}^{p}\beta_j^2 \le s$$

This is a biased estimator that for some value of $\lambda>0$ may have smaller mean squared error than the least squares estimator.

Ridge Regression (cont.)

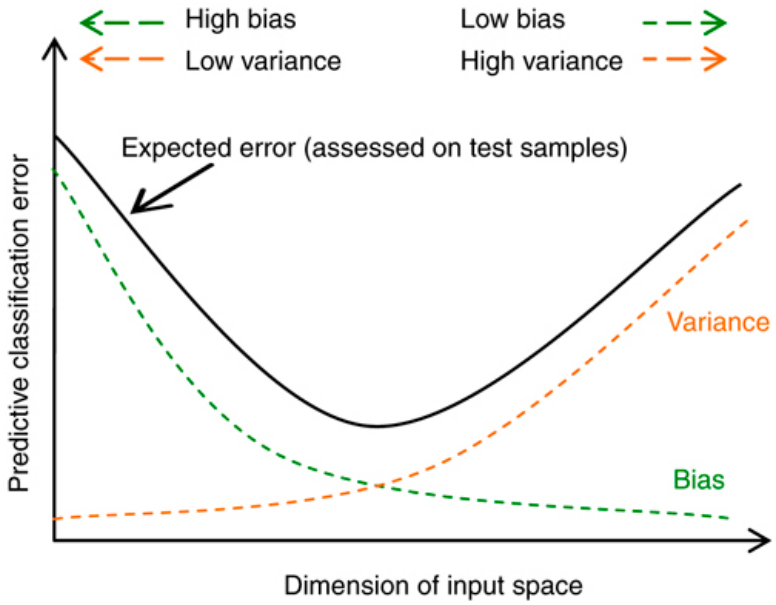
- As λ increases, the standardized ridge regression coefficients shrink to zero.
- Thus, when λ is extremely large, then all of the ridge coefficient estimates are basically zero; this corresponds to the *null model* that contains no predictors.

It is best to apply ridge regression after *standardizing the predictors*

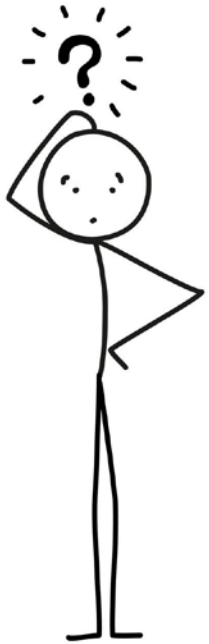


Ridge Regression

- In general, the ridge regression estimates will be *more biased* than the OLS ones but have *lower variance*.
- Ridge regression will work best in situations where the OLS estimates have high variance.
- The regularizer is convex such that we still have a convex loss function!



How did lasso differ from the ridge regression?



Logistic Regression

The logistic function is an example of a sigmoid function often used in feed-forward neural networks as activation function.

$$\Pr[Y_i = 1|X] = p(X) = \frac{e^{\beta_0+\beta_1 X}}{1 + e^{\beta_0+\beta_1 X}} = \sigma(\beta_0 + \beta_1 X_{1i})$$

- $\Pr(Y_i = 1) \rightarrow 0$ as $\beta_0 + \beta_1 X_{1i} \rightarrow -\infty$
- $\Pr(Y_i = 1) \rightarrow 1$ as $\beta_0 + \beta_1 X_{1i} \rightarrow \infty$

Likelihood function models a sequence of Bernoulli trials

$$L = \prod_{i=1} p^{y_i}(1-p)^{1-y_i} = \prod_{i=1} \sigma(\beta_0 + \beta_1 X_{1i})^{y_i} * [1 - \sigma(\beta_0 + \beta_1 X_{1i})]^{1-y_i}$$

The Likelihood Function for the Logit Model

Use $L = \prod_{i=1} p^{y_i}(1-p)^{1-y_i}$ and take the log to get the log likelihood

$$LL = \ln(L) = \sum_{i=1} y_i \ln p_i + (1-y_i)\ln(1-p_i)$$

We look for the vector β that maximizes LL (or minimizes $-LL$)

$$\beta = \text{argmax}_\beta LL(\beta) = \text{argmax}_\beta \left[\sum_{i=1} y_i \ln \sigma(\beta_0 + \beta_1 X_{1i}) + (1-y_i)\ln(1-\sigma(\beta_0 + \beta_1 X_{1i})) \right]$$

The LL function is twice differentiable and concave!

- For the OLS estimator, we set the FOC=0 and get an analytical solution.
- For the logistic regression, this does not get us a closed-form solution due to the nonlinearity of the logistic sogmoid function.
- We can use a numerical algorithm to find the maximum: gradient ascent!

The Negative LL Function is Convex

Is the negative LL function convex?

$$\sigma(\beta_0 + \beta_1 X_{1i}) = \frac{e^{\beta_0+\beta_1 X}}{1+e^{\beta_0+\beta_1 X}} = \frac{1}{1+e^{-(\beta_0+\beta_1 X_1)}} = \frac{1}{1+e^{-z}} = \sigma(z)$$

$$-LL(\beta) = \sum_{i=1} -y_i \ln \sigma(z) - (1-y_i)\ln(1-\sigma(z))$$

It is known that

- a convex function of an affine function is convex (i.e., $\sigma(\beta_0 + \beta_1 X_1)$).
- the sum of convex functions is convex (i.e., $\sum_{i=1} -y_i \ln \sigma(z) - (1-y_i)\ln(1-\sigma(z))$)

Let‘s define $f_1(z) = -\ln(\sigma(z))$ and $f_2(z) = -\ln\big(1-\sigma(z)\big)$ and $f(z) = y_i f_1(z) + (1-y_i) f_2(z)$

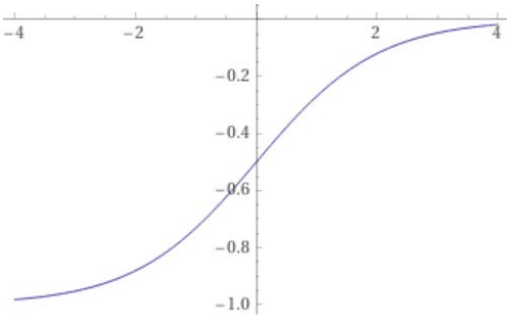
The key argument is to show that $f_1(z)$ and $f_2(z)$ are convex (next slide).

The Negative LL Function is Convex

1. $f_1(z)$ is convex:

$$f_1(z) = -\ln\left(\frac{1}{1+\exp(-z)}\right) = \ln\big(1+\exp(-z)\big)$$

The derivative is monotonically increasing



$$\frac{d}{dz}f_1(z) = -\frac{\exp(-z)}{1+\exp(-z)} = -1 + \frac{1}{1+\exp(-z)} = -1 + \sigma(z)$$

2. $f_2(z)$ is convex:

$$f_2(z) = -\ln\left(\frac{\exp(-z)}{1+\exp(-z)}\right) = \ln(1+\exp(-z)) + z = f_1(z) + z$$

The derivative of $f_2(z)$ is also monotonically increasing

$$\frac{\mathrm{d}}{dz}f_2(z) = \frac{d}{dz}f_1(z) + 1$$

Gradient for the LL Function

$$\beta = \text{argmax}_\beta LL(\beta) = \text{argmax}_\beta \left[\sum_{i=1} y_i \ln \sigma(\mathbf{X}\beta) + (1 - y_i) \ln(1 - \sigma(\mathbf{X}\beta)) \right]$$

$LL(\beta) = y \ln p + (1 - y) \ln(1 - p)$ for one sample

$p = \sigma(z), z = \mathbf{X}\beta$ short hands

$\frac{\partial LL(\beta)}{\partial \beta_j} = \frac{\partial LL(\beta)}{\partial p} * \frac{\partial p}{\partial z} * \frac{\partial z}{\partial \beta_j}$ chain rule

$$\frac{\partial p}{\partial z} = \frac{\partial \sigma(z)}{\partial z} = \frac{\partial}{\partial z} \frac{\exp(z)}{1 + \exp(z)} = \frac{\exp(z)(1 + \exp(z)) - \exp(2z)}{(1 + \exp(z))^2} =$$

$$\frac{\exp(z)}{1 + \exp(z)} - \left(\frac{\exp(z)}{1 + \exp(z)} \right)^2 = \sigma(z) - \big(\sigma(z)\big)^2 = \sigma(z)(1 - \sigma(z))$$

$$\equiv \frac{\partial p}{\partial z} = \sigma(z)[1 - \sigma(z)]$$

Gradient for the LL Function

$$\beta = \text{argmax}_\beta LL(\beta) = \text{argmax}_\beta \left[\sum_{i=1} y_i \ln \sigma(\mathbf{X}\beta) + (1 - y_i) \ln(1 - \sigma(\mathbf{X}\beta)) \right]$$

We can use the chain rule:

$$LL(\beta) = y \ln p + (1 - y) \ln(1 - p)$$

$$\frac{\partial LL(\beta)}{\partial p} = \frac{y}{p} - \frac{1 - y}{1 - p}$$

$$p = \sigma(z)$$

$$\frac{\partial p}{\partial z} = \sigma(z)[1 - \sigma(z)]$$

$$z = \mathbf{X}\beta$$

$$\frac{\partial z}{\partial \beta_j} = x_j$$

$$\frac{\partial LL(\beta)}{\partial \beta_j} = \frac{\partial LL(\beta)}{\partial p} * \frac{\partial p}{\partial z} * \frac{\partial z}{\partial \beta_j} =$$

$$\left[\frac{y}{p} - \frac{1 - y}{1 - p} \right] \sigma(z)[1 - \sigma(z)] x_j =$$

since $p = \sigma(z)$

$$\left[\frac{y}{p} - \frac{1 - y}{1 - p} \right] p[1 - p] x_j =$$

$$[y(1 - p) - p(1 - y)] x_j =$$

$$[y - p] x_j =$$

$$[y - \sigma(\mathbf{X}\beta)] x_j$$ => gradient

Gradient Ascent for the Likelihood Function

We want to choose parameters (β) that maximize the likelihood, and we know the partial derivative of the log likeliood (LL) with respect to each parameter.
The LL function is convex, but no closed-form solution exists for the derivative. So, we can use gradient ascent to maximize the log likelihood.

Repeat many times:
  For each training example $(\boldsymbol{x}_i, y_i)$ do:
    For each parameter $0 \leq j \leq p$ do:

$$\beta_j^{new} = \beta_j^{old} + \alpha * \frac{\partial LL(\beta^{old})}{\partial \beta_j^{old}}$$

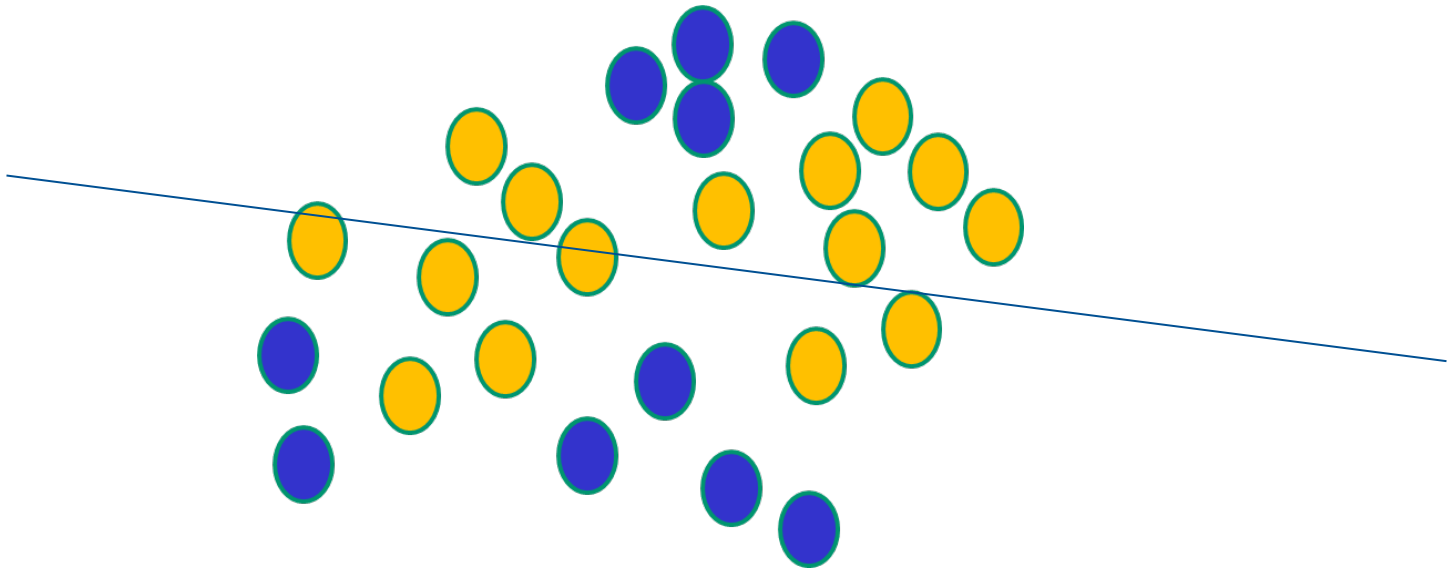
$$= \beta_j^{old} + \alpha * \sum_i [y_i - \sigma(\beta^T \boldsymbol{x}_i)] x_{ij}$$

$$= \beta_j^{old} + \alpha * \sum_i \left[y_i - \frac{\exp(\beta^T \boldsymbol{x}_i)}{1 + \exp(\beta^T \boldsymbol{x}_i)} \right] x_{ij}$$

Parameter β_0 is added for an additional feature x_0 that always takes the value 1.

Logistic Regression and Neural Networks

- Logistic regression does assume a linear relationship between the input variables with the output.
- Logistic regression is useful if we are working with a dataset where the classes are more or less “linearly separable.”
- We can think of logistic regression as a one layer neural network.

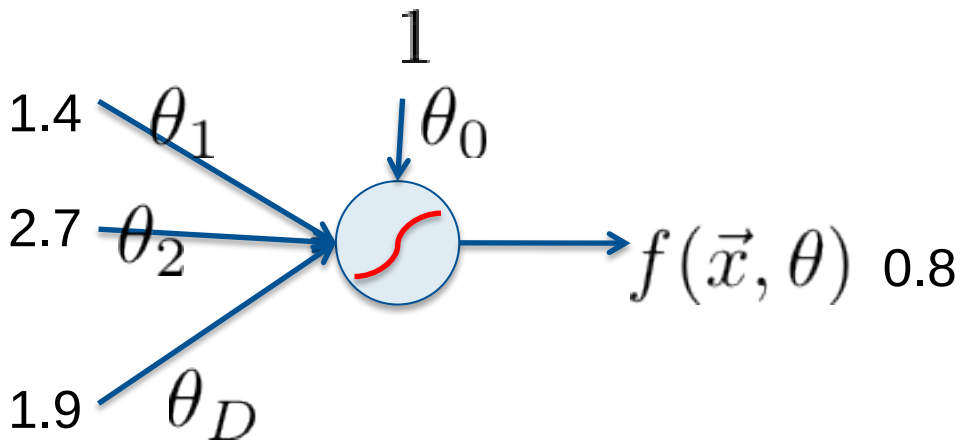


1-Layer Neural Network with Sigmoid Activation

Initialize with random weights
Present a training pattern
Feed it through to get output

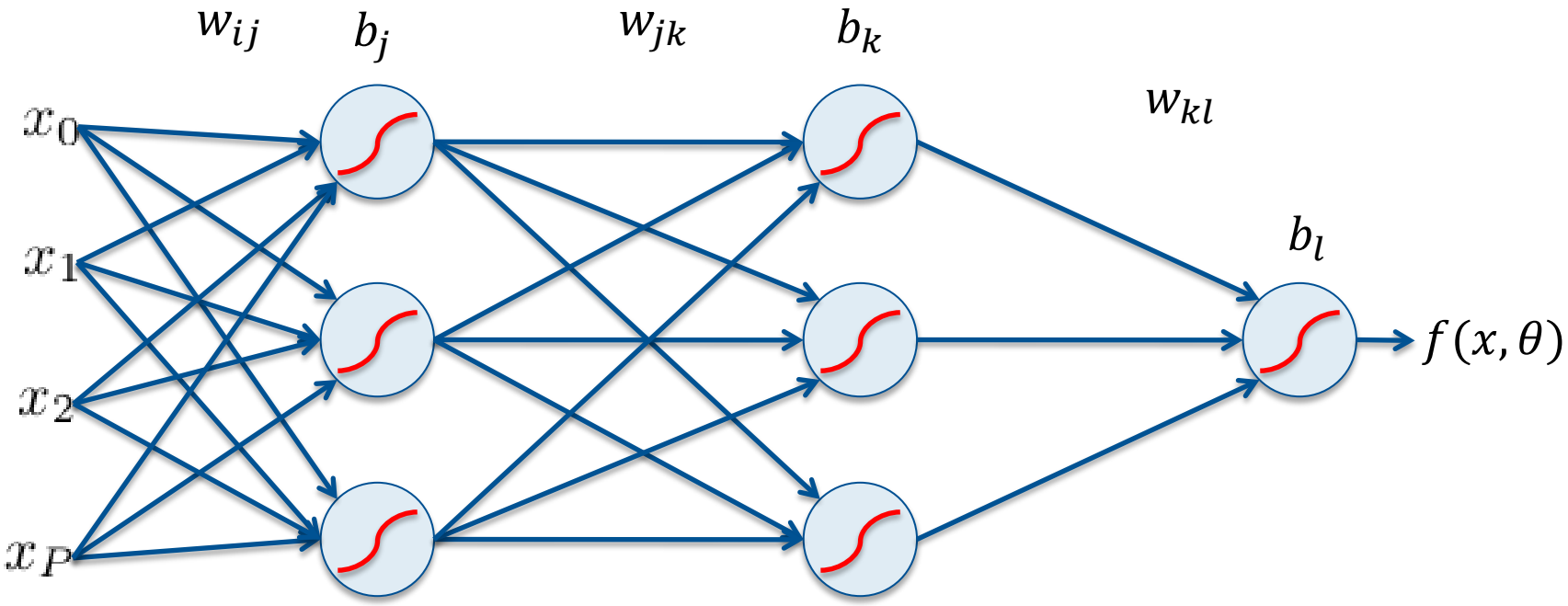
Training data set

Fields			*class*
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0
etc …			



Error Backpropagation

In backpropagation, we do gradient descent on the whole network.
Once we stack logistic activation functions in a multi-layer neural network, the loss function is not convex anymore!



Neural Networks and Gradient Descent

Minimizing the **empirical risk function** $\boldsymbol{R(\theta)}$, which is modeling ***expected loss*** (as we don't know the true distribution of data).
This means, the empirical risk $\boldsymbol{R(\theta)}$ is the average loss over the training data.

$$R(\theta) = \frac{1}{N}\sum_{n} L(y_n, g(\theta, x_n)) = \frac{1}{2N}\sum_{n} (y_n - g(\theta^T x_n))^2$$

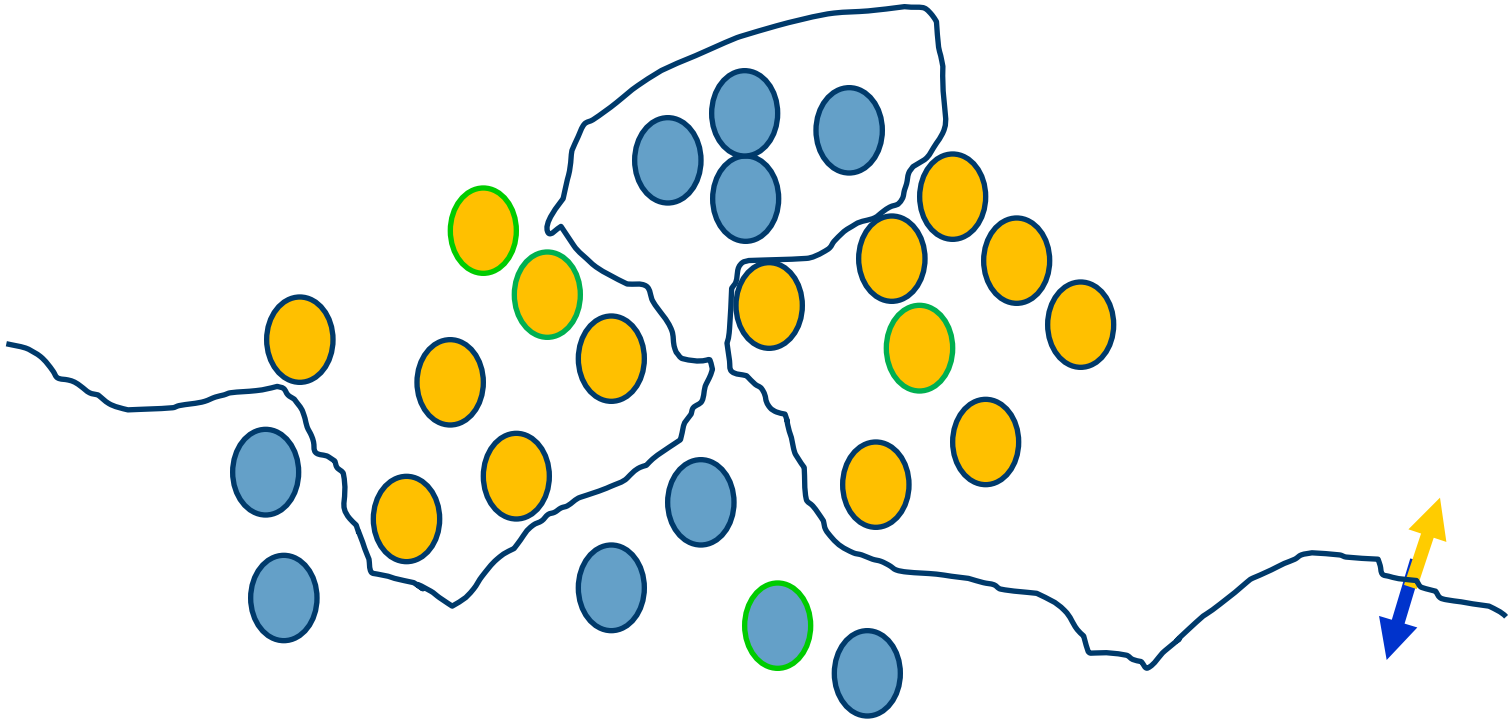
derivative of $f(z)^2 \Rightarrow 2f(z)f'(z)$ (chain rule)

$$\nabla_\theta R = \frac{1}{2N}\sum_{n} 2(y_n - g(\theta^T x_n))(-1)g'(\theta^T x_n)x_n = 0$$

$$g(z) = (1 + exp(-z))^{-1}$$

Again, there is no "closed-form" solution. We used gradient descent to backpropagate the error of training examples.

Neural Networks as a Non-Linear Classifier



Universal Approximation Theorem

A feed-forward neural network with a single hidden layer and continuous non-linear activation function can approximate any continuous function with arbitrary precision.*

1. Neural Network
2. One hidden layer
3. Non-linear activation function

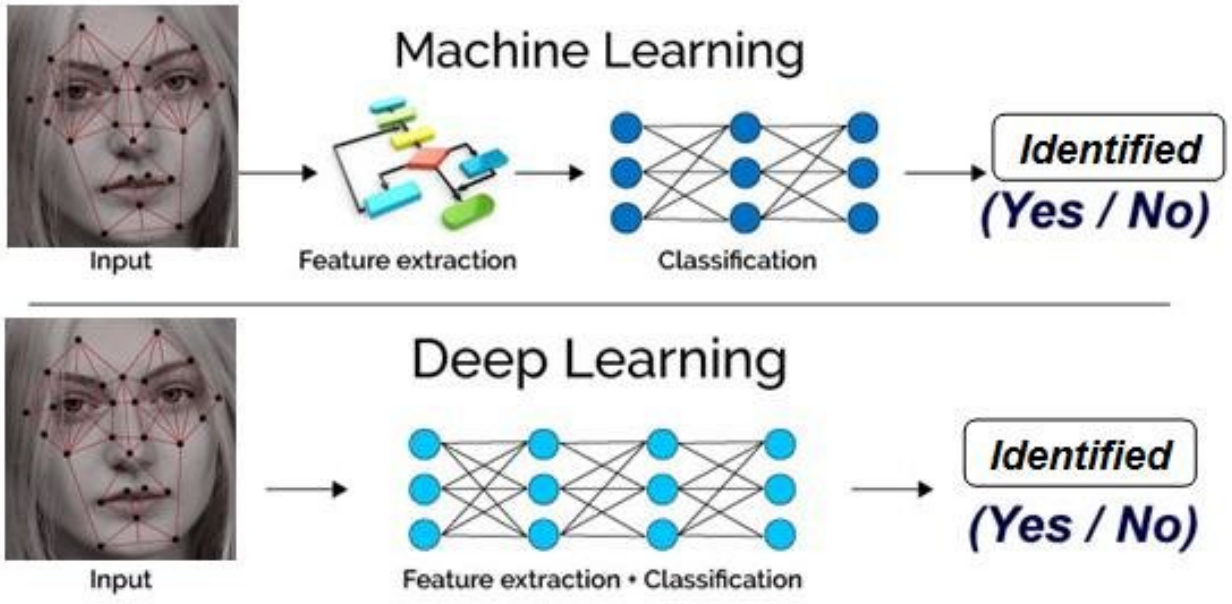
} Approximate any function with any precision

Deep Neural Networks $>_{\text{in practice}}$ Wide Neural Networks

*Hornik, Kurt; Tinchcombe, Maxwell; White, Halbert (1989). *Multilayer Feedforward Networks are Universal Approximators*. *Neural Networks*. 2. pp. 359–366.

Deep Learning

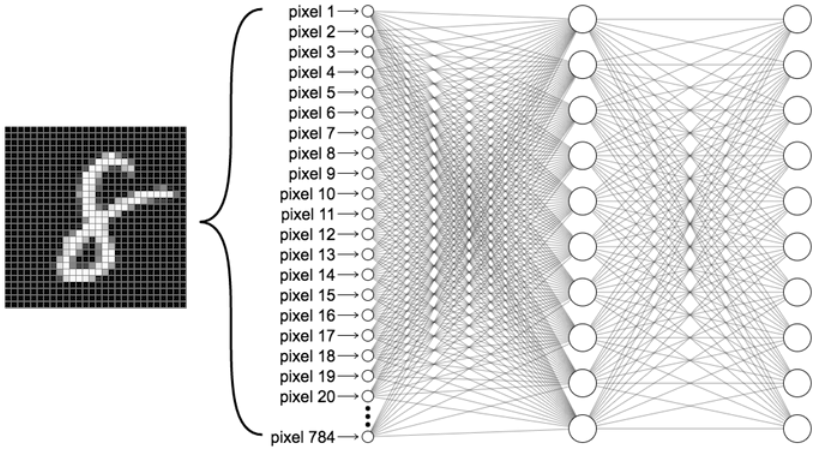
- Usually having *two or more* hidden layers counts as deep.
- Typically trained on GPUs.
- Seen as an end-to-end framework (no feature extraction steps necessary).

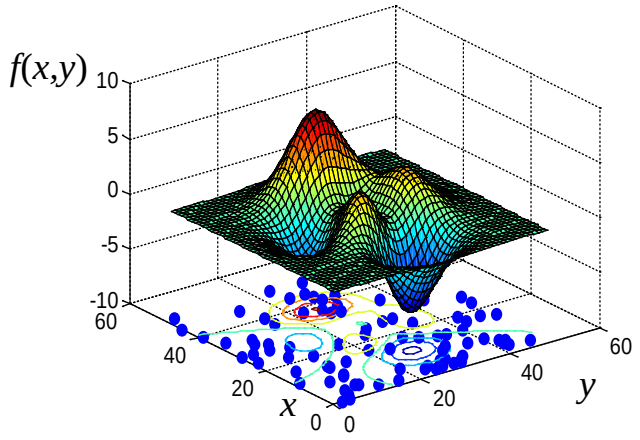


Source: Najm, Ansaf, Hassen (2019)

Machine Learning = Optimization over Data

Empirical risk minimization: Fitting the parameters of the model („training“) = optimization





$$R(\theta) = \frac{1}{N}\sum_{n} L(y_n, g(\theta, x_n)) + R(x_n)$$

N = # examples

y_n = labels

$\theta \in \mathbb{R}^d$ = features

$R(x_n)$ = regularization

Global optimization is NP-hard, convex optimization is not!

Course Content

- Introduction
- Regression Analysis
- Regression Diagnostics
- Logistic and Poisson Regression
- Naive Bayes and Bayesian Networks
- Decision Tree Classifiers
- Data Preparation and Causal Inference
- Model Selection and Learning Theory
- Ensemble Methods and Clustering
- High-Dimensional Problems
- Association Rules and Recommenders
- Neural Networks
- **Convex Optimization in Machine Learning**