Lecture 3

Logistic Regression

Outline

- **Logistic Regression**
- **Gradient descent for Logistic Regression**
- **Newton's Method for Logistic Regression**
- **Multinomial Logistic Regression**

Binary Classification

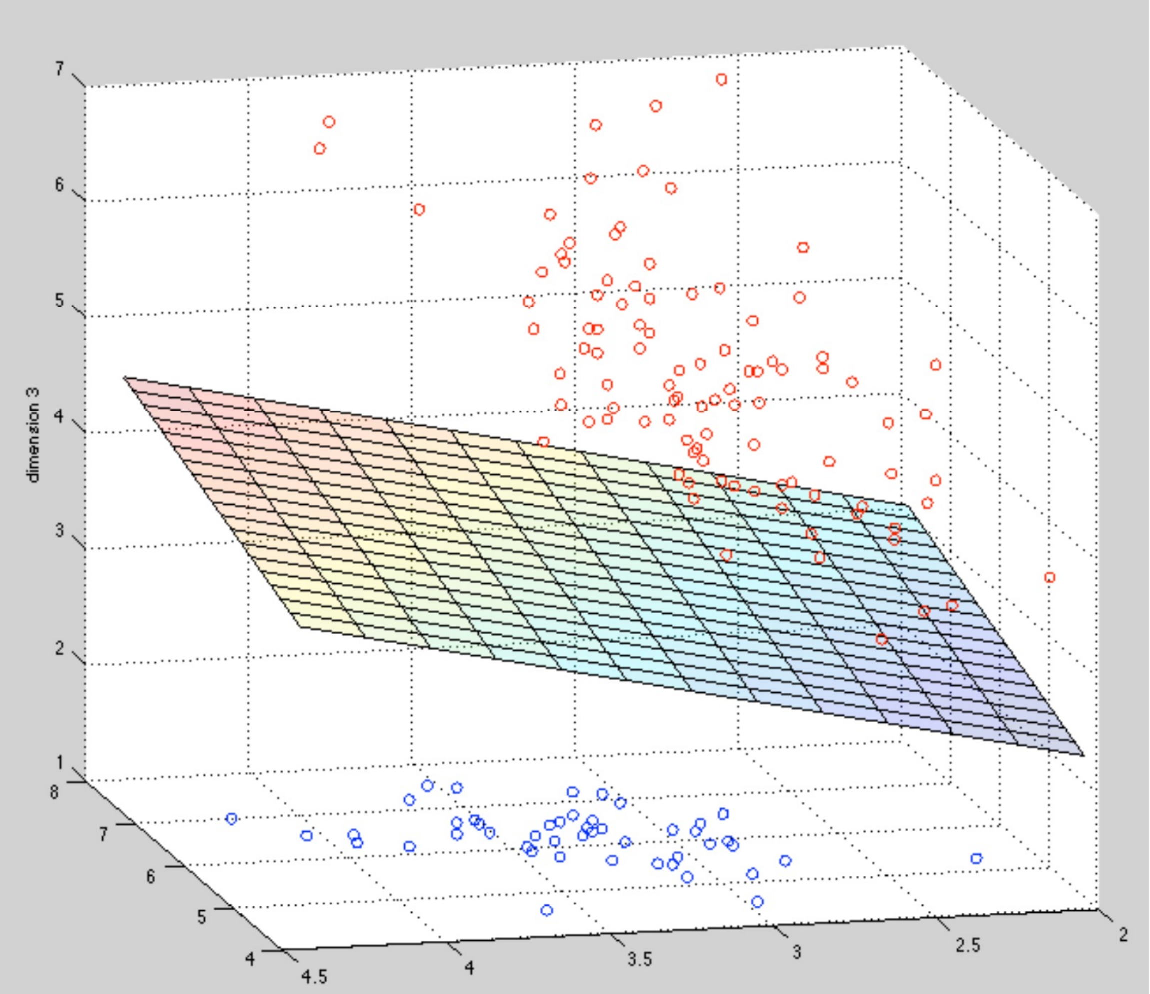
Suppose we're distinguishing cat from dog images

Two Phrases of Logistic Regression

- **Training**: we learn weights w and b using **stochastic gradient descent** and **cross-entropy loss**.

- **Test**: Given a test example x we compute $p(y|x)$ using learned weights w and b, and return whichever label ($y = 1$ *or* $y = 0$) is higher probability

Hyperplanes



Using gradient ascent for linear classifiers

Key idea behind today's lecture:

1. Define a linear classifier (logistic regression)
2. Define an objective function (likelihood)
3. Optimize it with gradient descent to learn parameters
4. Predict the class with highest probability under the model

Binary Classification

- The predictions and the output labels

$$z = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} + b \qquad\qquad y \in \{0, 1\}$$

- The real-valued predictions of the linear regression model need to be converted into **0/1**.

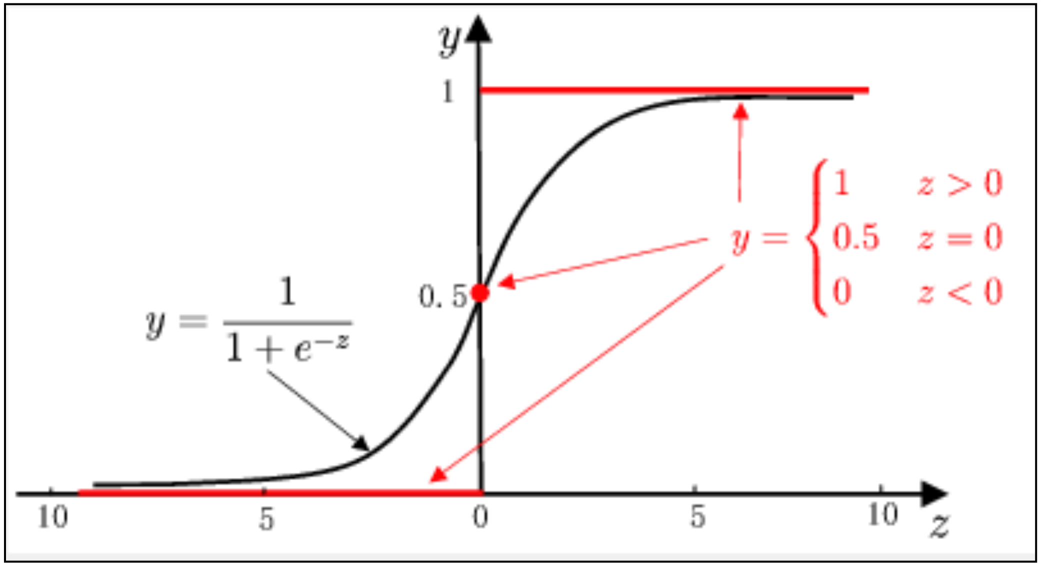
- Ideally, the unit-step function is desired

$$y = \begin{cases} 0, & z < 0; \\ 0.5, & z = 0; \\ 1, & z > 0, \end{cases}$$

  - which predicts positive for z greater than 0, negative for z smaller than 0, and an arbitrary output when z equals to 0.

Binary Classification

- Disadvantages of unit-step function
  - not continuous
- Logistic (sigmoid) function: a surrogate function to approximate the unit-step function
  - monotonic differentiable



Comparison between unit-step function and logistic function

$$\sigma(z) = \frac{1}{1+e^{-z}}$$

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

Logistic Regression

Data: Inputs are continuous vectors of length d. Outputs are discrete labels.

$$\mathcal{D} = \left\{ \boldsymbol{x}^{(i)}, y^{(i)} \right\}_{i=1}^{m} \text{ where } \boldsymbol{x} \in \mathbb{R}^d \text{ and } y \in \{0, 1\}$$

Model: Logistic function applied to dot product of parameters with input vector.

$$p_{\boldsymbol{\theta}}(y = 1 \mid \mathbf{x}) = \frac{1}{1 + \exp(-\boldsymbol{\theta}^T \mathbf{x})}$$

Learning: finds the parameters that minimize some objective function.

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$$

Prediction: Output is the most probable class.

$$\hat{y} = \underset{y \in \{0,1\}}{\operatorname{argmax}}\, p_{\boldsymbol{\theta}}(y|\mathbf{x})$$

Log odds

- Apply logistic function

$$y = \frac{1}{1+e^{-z}} \quad \text{transform into} \quad y = \frac{1}{1+e^{-(\boldsymbol{w}^T\boldsymbol{x}+b)}}$$

- Log odds
  - the logarithm of the relative likelihood of a sample being a positive sample

$$\ln\frac{y}{1-y} = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} + \mathrm{b}$$

- Logistic regression has several nice properties
  - without requiring any prior assumptions on the data distribution
  - it predicts labels together with associated probabilities
  - it is solvable with numerical optimization methods.

Logistic regression - maximum likelihood

*In statistics, **maximum likelihood estimation** (**MLE**) is a method of estimating the parameters of a statistical model given observations, by finding the parameter values that maximize the likelihood of making the observations given the parameters.*

MLE can be seen as a special case of the maximum a posteriori estimation (MAP) that assumes a uniform prior distribution of the parameters, or as a variant of the MAP that ignores the prior and which therefore is unregularized.

Logistic regression - maximum likelihood

- Maximum likelihood
  - Given the training dataset $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{m}$
  - Maximizing the probability of each sample being predicted as the ground-truth label
    - the log-likelihood to be maximized is:

$$\ell(\boldsymbol{w}, b) = \log \prod_{i=1}^{m} p(y_i \mid \boldsymbol{x}_i; \boldsymbol{w}, b)$$

    - assumption that the training examples are independent:

$$\ell(\boldsymbol{w}, b) = \sum_{i=1}^{m} \log p(y_i \mid \boldsymbol{x}_i; \boldsymbol{w}, b)$$

Logistic regression - maximum likelihood

- Log odds can be rewritten as

$$\ln \frac{p(y=1 \mid \boldsymbol{x})}{p(y=0 \mid \boldsymbol{x})} = \boldsymbol{w}^{\mathrm{T}} \boldsymbol{x} + b$$

and consequently,

$$p(y=1 \mid \boldsymbol{x}) = \frac{e^{\boldsymbol{w}^{\mathrm{T}} \boldsymbol{x} + b}}{1 + e^{\boldsymbol{w}^{\mathrm{T}} \boldsymbol{x} + b}} = \mathrm{sigmoid}(\boldsymbol{w}^{\mathrm{T}} \boldsymbol{x} + b)$$

$$\begin{aligned} p(y=0 \mid \boldsymbol{x}) = \frac{1}{1 + e^{\boldsymbol{w}^{\mathrm{T}} \boldsymbol{x} + b}} &= 1 - \mathrm{sigmoid}(\boldsymbol{w}^{\mathrm{T}} \boldsymbol{x} + b) \\ &= \mathrm{sigmoid}(-(\boldsymbol{w}^{T} \boldsymbol{x} + b)) \end{aligned}$$

Logistic regression - maximum likelihood

- Transform into minimize negative log-likelihood
  - Let $\boldsymbol{\beta} = (\boldsymbol{w}; b)$, $\hat{\boldsymbol{x}} = (\boldsymbol{x}; 1)$, $\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} + b$ can be rewritten as $\boldsymbol{\beta}^{\mathrm{T}}\hat{\boldsymbol{x}}$
  - Let $p_1(\hat{\boldsymbol{x}}_i; \boldsymbol{\beta}) = p(y = 1 \mid \hat{\boldsymbol{x}}; \boldsymbol{\beta})$

$$p_0(\hat{\boldsymbol{x}}_i; \boldsymbol{\beta}) = p(y = 0 \mid \hat{\boldsymbol{x}}; \boldsymbol{\beta}) = 1 - p_1(\hat{\boldsymbol{x}}_i; \boldsymbol{\beta})$$

the likelihood term in can be rewritten as

$$p(y_i \mid \boldsymbol{x}_i; \boldsymbol{w}_i, b) = y_i p_1(\hat{\boldsymbol{x}}_i; \boldsymbol{\beta}) + (1 - y_i) p_0(\hat{\boldsymbol{x}}_i; \boldsymbol{\beta})$$

  - maximizing log-likelihood is equivalent to minimizing

$$J(\boldsymbol{\beta}) = \sum_{i=1}^{m}\left(-y_i\boldsymbol{\beta}^{\mathrm{T}}\hat{\boldsymbol{x}}_i + \log\left(1 + e^{\boldsymbol{\beta}^{\mathrm{T}}\hat{\boldsymbol{x}}_i}\right)\right)$$

Logistic regression - maximum likelihood

- Transform into minimize negative log-likelihood
  - Let $\boldsymbol{\beta} = (\boldsymbol{w}; b)$, $\hat{\boldsymbol{x}} = (\boldsymbol{x}; 1)$, $\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} + b$ can be rewritten as $\boldsymbol{\beta}^{\mathrm{T}}\hat{\boldsymbol{x}}$
  - Let $p_1(\hat{\boldsymbol{x}}_i; \boldsymbol{\beta}) = p(y = 1 \mid \hat{\boldsymbol{x}}; \boldsymbol{\beta})$

$$p_0(\hat{\boldsymbol{x}}_i; \boldsymbol{\beta}) = p(y = 0 \mid \hat{\boldsymbol{x}}; \boldsymbol{\beta}) = 1 - p_1(\hat{\boldsymbol{x}}_i; \boldsymbol{\beta})$$

the likelihood term in can be rewritten as

$$p(y_i \mid \hat{\boldsymbol{x}}_i; \hat{\boldsymbol{w}}_i, b) = p_1(\hat{\boldsymbol{x}}_i; \boldsymbol{\beta})^{y_i} p_0(\hat{\boldsymbol{x}}_i; \boldsymbol{\beta})^{1-y_i}$$

  - maximizing log-likelihood is equivalent to minimizing

$$J(\boldsymbol{\beta}) = \sum_{i=1}^{m} -[y_i \log p_1(\hat{\boldsymbol{x}}_i; \boldsymbol{\beta}) + (1 - y_i) \log p_0(\hat{\boldsymbol{x}}_i; \boldsymbol{\beta})]$$

The Cross-Entropy loss!

Maximum Likelihood Estimation

Learning: Four approaches to solving $\boldsymbol{\beta}^* = \arg\min_{\boldsymbol{\beta}} J(\boldsymbol{\beta})$

- **Approach 1:** Gradient Descent
  (take larger – more certain – steps opposite the gradient)
- **Approach 2:** Stochastic Gradient Descent (SGD)
  (take many small steps opposite the gradient)
- **Approach 3:** Newton's Method
  (use second derivatives to better follow curvature)
- **Approach 4:** Closed Form???
  (set derivatives equal to zero and solve for parameters)

Maximum Likelihood Estimation

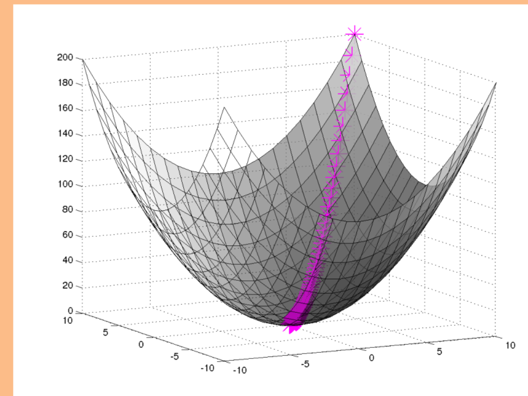
Learning: Four approaches to solving $\boldsymbol{\beta}^* = \arg\min_{\boldsymbol{\beta}} J(\boldsymbol{\beta})$

- **Approach 1:** Gradient Descent
  (take larger – more certain – steps opposite the gradient)
- **Approach 2:** Stochastic Gradient Descent (SGD)
  (take many small steps opposite the gradient)
- **Approach 3:** Newton's Method
  (use second derivatives to better follow curvature)
- ~~**Approach 4:** Closed Form???~~
  ~~(set derivatives equal to zero and solve for parameters)~~

Gradient Descent

Algorithm 1 Gradient Descent

procedure $\mathrm{GD}(\mathcal{D}, \boldsymbol{\theta}^{(0)})$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}^{(0)}$
    while not converged **do**
        $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$
    return $\boldsymbol{\theta}$



$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \begin{bmatrix} \frac{d}{d\theta_1} J(\boldsymbol{\theta}) \\ \frac{d}{d\theta_2} J(\boldsymbol{\theta}) \\ \vdots \\ \frac{d}{d\theta_N} J(\boldsymbol{\theta}) \end{bmatrix}$$

$$\boldsymbol{\theta}^{t+1} = \boldsymbol{\theta}^{t} - \eta \nabla J_{\boldsymbol{\theta}}(\boldsymbol{\theta}))$$

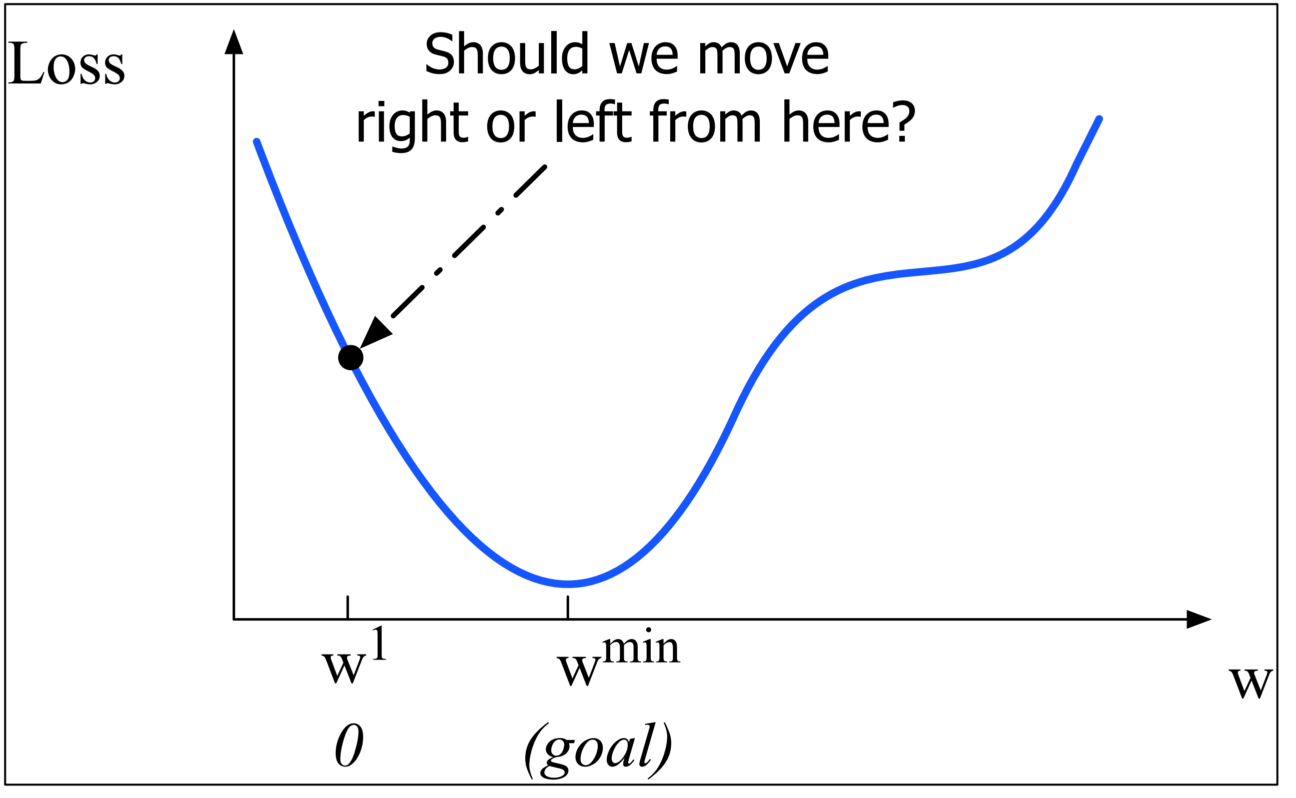
Review: Derivative of a Function

$\lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$ is called the derivative of f at x.

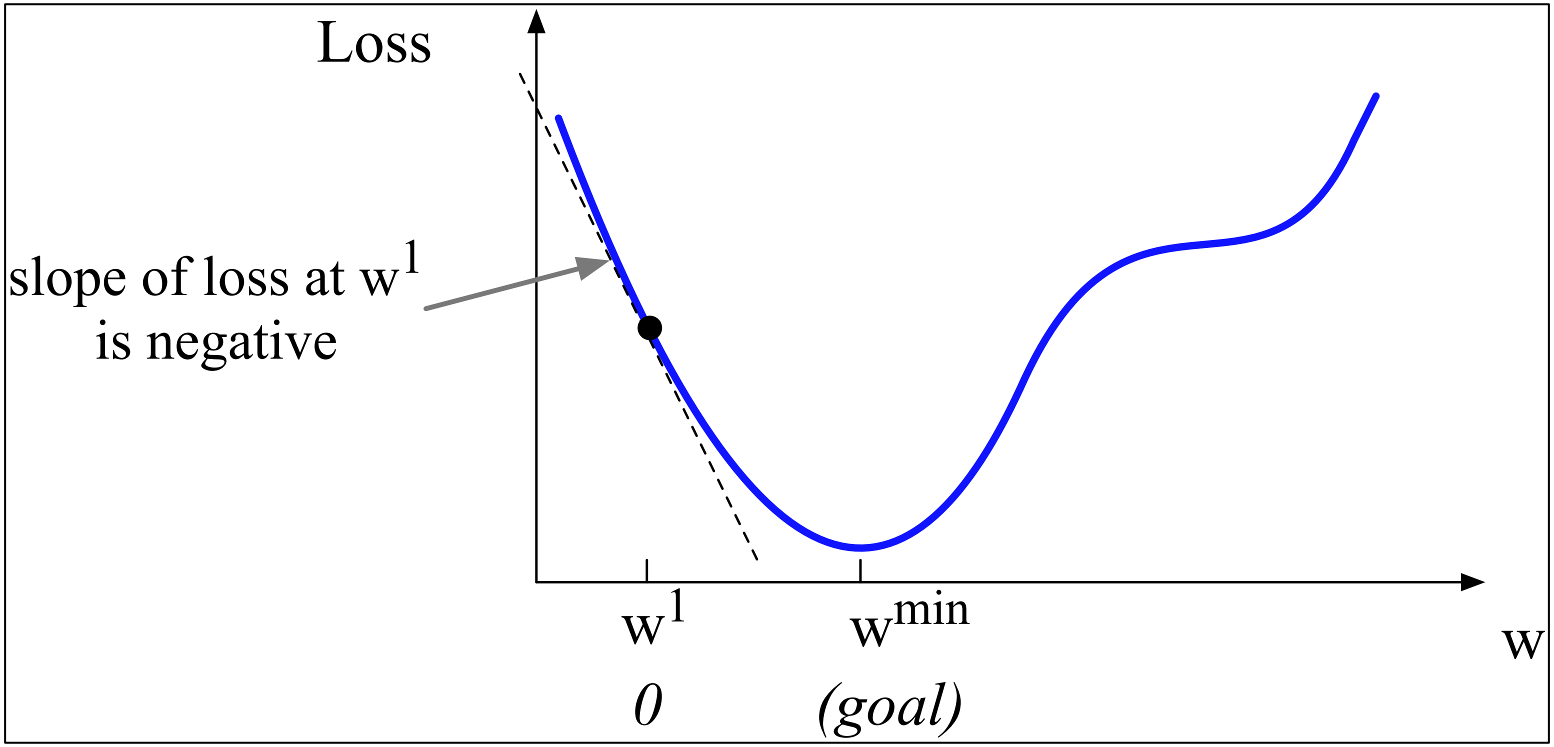
We write: $$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

"The derivative of f with respect to x is ..."

Gradient Descent

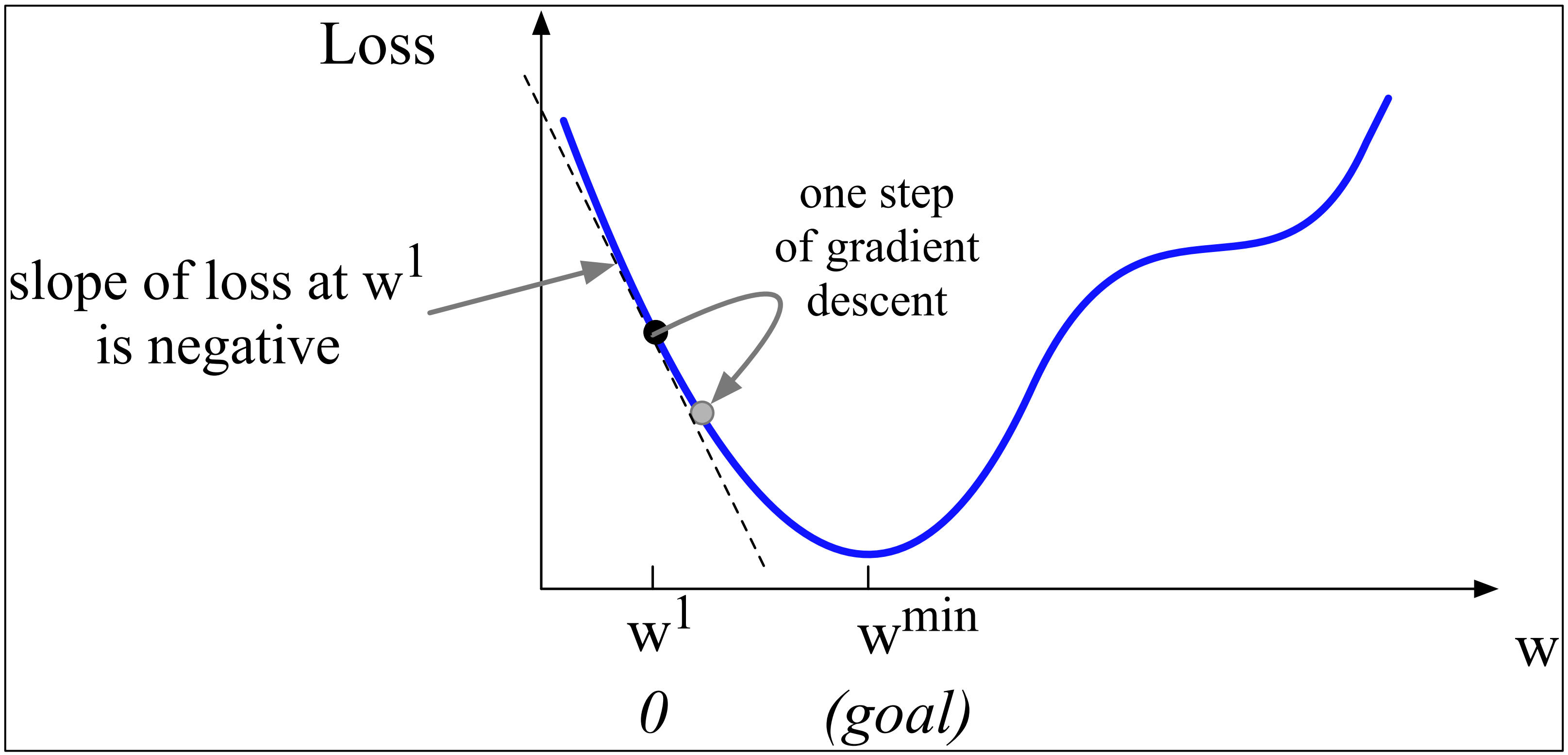


Gradient Descent



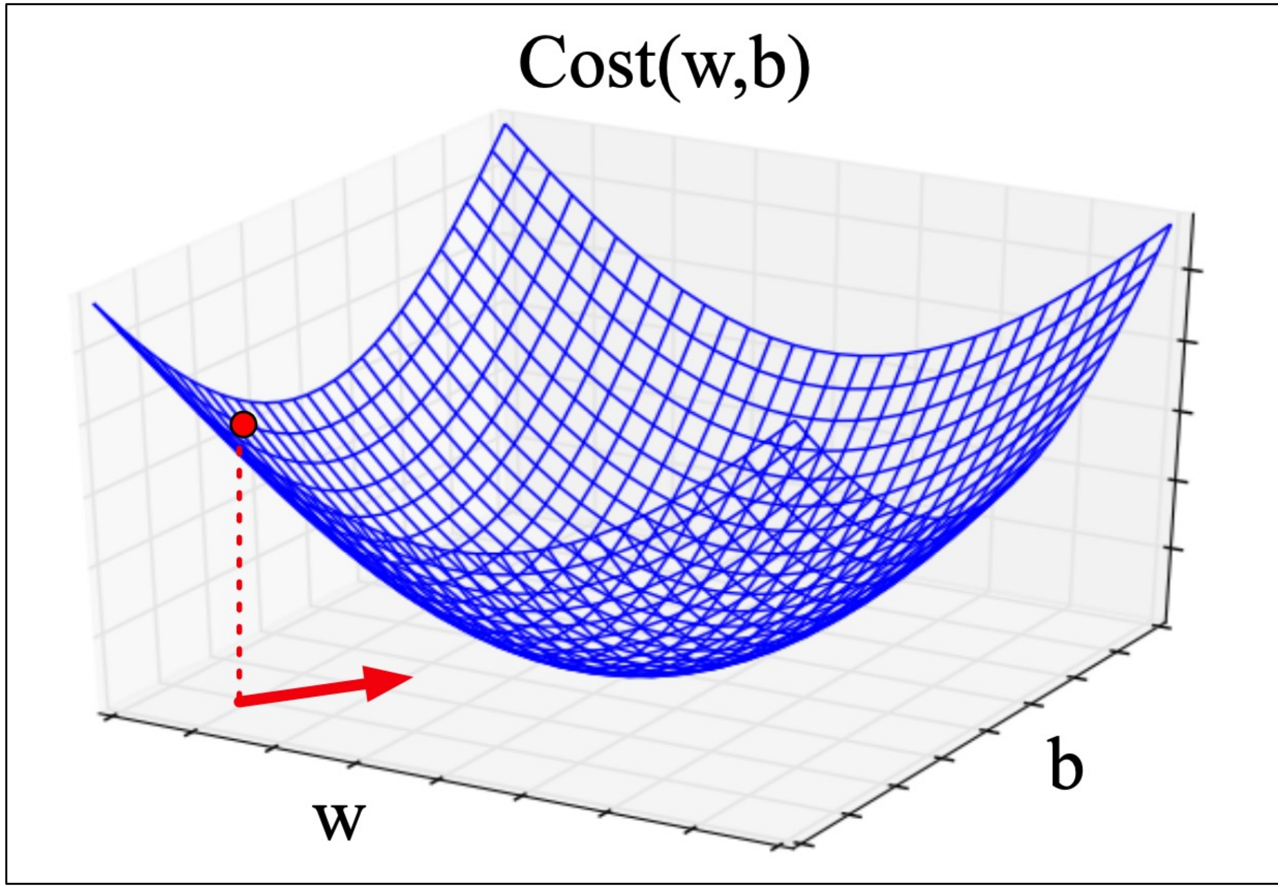
Gradient Descent

Q: Given current w, should we make it bigger or smaller?
A: Move *w* in the reverse direction from the slope of the function



Gradient Descent

- Visualizing the gradient vector at the red point
- It has two dimensions shown in the x-y plane



Online Resource

- Machine Learning Lecture 12 "Gradient Descent / Newton's Method"
- https://www.youtube.com/watch?v=o6FfdP2uYh4
- Instructor: Kilian Weinberger @ Cornell

Gradient for Logistic Regression

- The cross-entropy loss function

$$J(\boldsymbol{\beta}) = \sum_{i=1}^{m} -[y_i \log p_1(\hat{\boldsymbol{x}}_i; \boldsymbol{\beta}) + (1 - y_i) \log p_0(\hat{\boldsymbol{x}}_i; \boldsymbol{\beta})]$$

- The gradient

$$\frac{\partial J(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}} = - \sum_{i=1}^{m} \hat{\boldsymbol{x}}_i (y_i - p_1(\hat{\boldsymbol{x}}_i; \boldsymbol{\beta}))$$

- Instead of using the sum notation, we can more efficiently compute the gradient in its matrix form

$$\frac{\partial J(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}} = \mathbf{X}(\sigma(\mathbf{X}^T \boldsymbol{\beta}) - \mathbf{y})$$

$\mathbf{X} \in \mathbb{R}^{d \times m}$

σ : sigmoid

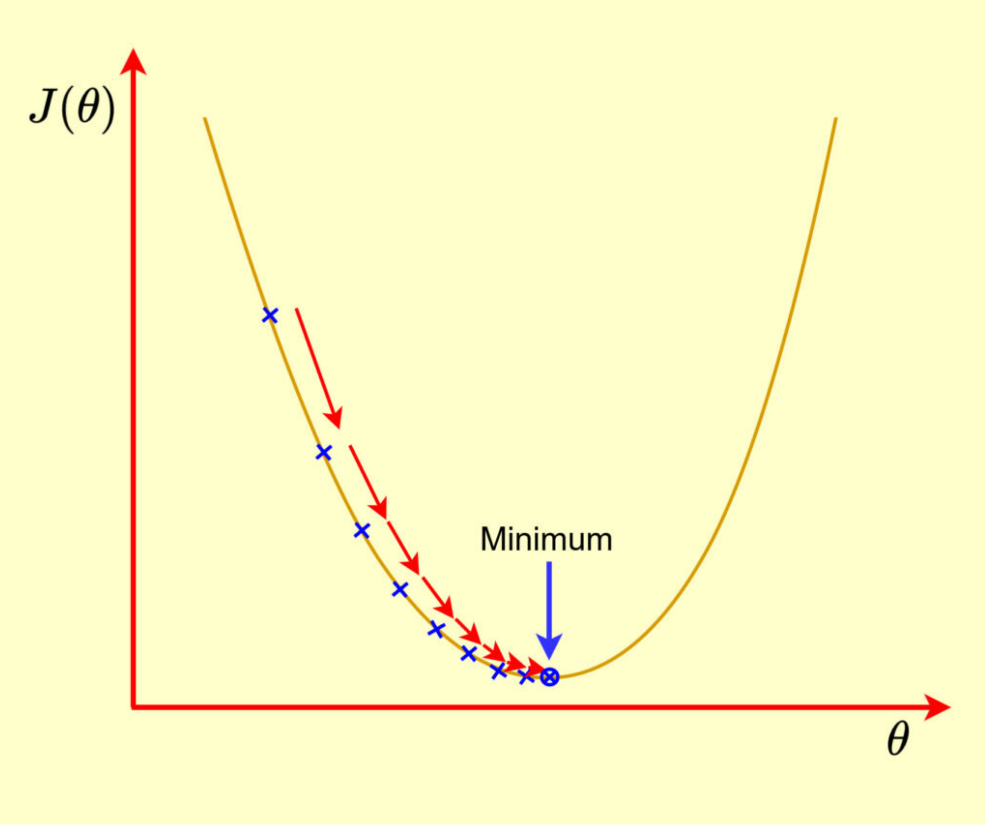
Picking learning rate

- Use grid-search in log-space over small values on a validation set:
  - e.g., 0.01, 0.001, …
- Sometimes, update after each pass:
  - e.g., decrease by a factor of 1/t
  - sometimes use cosine annealing
- Fancier techniques we won't talk about:
  - Adaptive gradient: scale gradient differently for each dimension (Adagrad, ADAM, ...)

Slide courtesy of Matt Gormley

Convexity and Logistic Regression

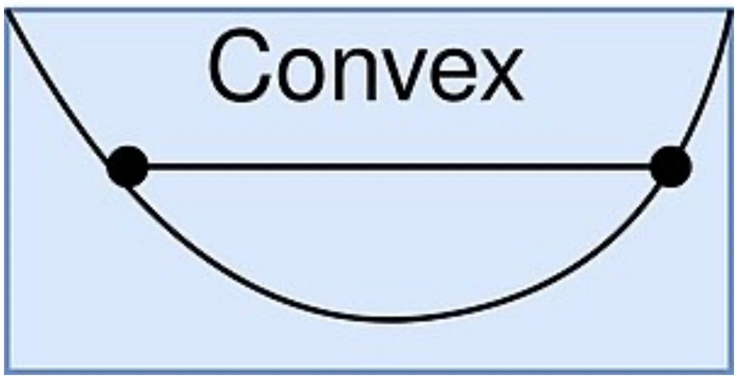
This loss function is convex: there is only one local minimum. So gradient descent will give the global minimum.

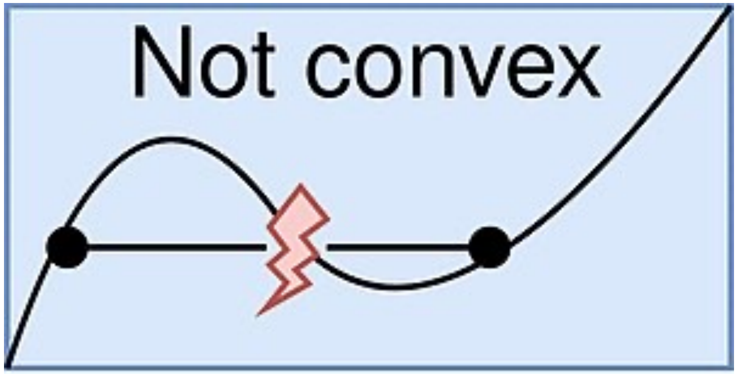


Convex function

Definition 1. *A function $f : \mathbb{R}^n \to \mathbb{R}$ is convex if its domain is a convex set and for all x, y in its domain, and all $\lambda \in [0, 1]$, we have*

$$f(\lambda x + (1 - \lambda)y) \leq \lambda f(x) + (1 - \lambda)f(y).$$





- e^{ax}
- $-\log(x)$

Strict and strong convexity

also known as
Jensen's Inequality

Definition 2. *A function* $f : \mathbb{R}^n \to \mathbb{R}$ *is*

- Strictly convex *if* $\forall x, y, x \neq y, \forall \lambda \in (0, 1)$

$$f(\lambda x + (1-\lambda)y) < \lambda f(x) + (1-\lambda)f(y).$$

- Strongly convex, *if* $\exists \alpha > 0$ *such that* $f(x) - \alpha ||x||^2$ *is convex.*

Lemma 1. *Strong convexity* $\Rightarrow$ *Strict convexity* $\Rightarrow$ *Convexity.*
(But the converse of neither implication is true.)

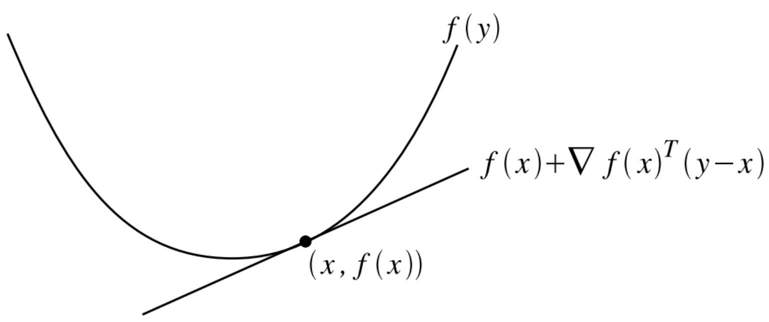
Convex function

Theorem 2. *Suppose $f : \mathbb{R}^n \to \mathbb{R}$ is twice differentiable over an open domain. Then, the following are equivalent:*

(i) f *is convex.*

(ii) $f(y) \geq f(x) + \nabla f(x)^T (y - x)$, *for all* $x, y \in dom(f)$.

(iii) $\nabla^2 f(x) \succeq 0$, *for all* $x \in dom(f)$.



Positive semidefinite
Hessian matrix

$$\nabla^2 f(x) \succeq 0$$

First Order Condition for Convexity

Second Order Condition for Convexity

Hessian Matrix

Definition: the **Hessian** of a K-dimensional function is the matrix of partial second derivatives with respect to each pair of dimensions.

$$H_f(\mathbf{x}) := \nabla^2 f(\boldsymbol{x}) = \begin{bmatrix} \frac{\partial^2 f(\boldsymbol{x})}{\partial x_1^2} & \frac{\partial^2 f(\boldsymbol{x})}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f(\boldsymbol{x})}{\partial x_1 \partial x_K} \\ \frac{\partial^2 f(\boldsymbol{x})}{\partial x_2 \partial x_1} & \frac{\partial^2 f(\boldsymbol{x})}{\partial x_2^2} & \cdots & \frac{\partial^2 f(\boldsymbol{x})}{\partial x_2 \partial x_K} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f(\boldsymbol{x})}{\partial x_K \partial x_1} & \frac{\partial^2 f(\boldsymbol{x})}{\partial x_K \partial x_2} & \cdots & \frac{\partial^2 f(\boldsymbol{x})}{\partial x_K^2} \end{bmatrix}$$

Slide courtesy of Matt Gormley

Hessian Matrix

- Let $f: \mathbb{R}^d \mapsto \mathbb{R}$ be a twice differentiable function. Then, the Hessian of f at $\mathbf{x} \in \mathbb{R}^d$ is a matrix in $\mathbb{R}^{d \times d}$ denoted by $\nabla^2 f(\mathbf{x})$ and defined by

$$\nabla^2 f(\mathbf{x}) = \left[\frac{\partial^2 f}{\partial x_i, x_j}(\mathbf{x}) \right]_{1 \le i,j \le d}$$

- Example: $f(\mathbf{x}) = -\sum_{i=1}^{d} x_i \ln x_i$

$$\nabla f(\mathbf{x}) = \begin{bmatrix} -(\ln x_1 + 1) \\ \vdots \\ -(\ln x_d + 1) \end{bmatrix} \Rightarrow \nabla^2 f(\mathbf{x}) = \mathrm{diag}(-\frac{1}{x_1}, \ldots, -\frac{1}{x_d})$$

Examples of convex functions

- Square Loss
  - $f(x, v) = (x - v)^2$
- Absolute Loss
  - $f(x, v) = |x - v|$
- Hinge Loss
  - $f(x, v) = \max(0, 1 - xv)$
- Regularization
  - $r(x) = \frac{\lambda}{2} \|x\|_2^2$
  - $r(x) = \lambda \|x\|_1$

The Newton's Method

- Gradient descent **may take many steps to converge** to that optimum.
- The motivation behind **Newton's method** is to use a **quadratic approximation** of our function to make a good guess where we should step next.
- From linear regression, we know that we can find the **minimizer** to a **quadratic function** analytically (i.e. **closed form**).

Taylor Series

How can we approximate a function in 1-dimension?

The **Taylor series expansion** for an infinitely differentiable function $f(x)$, $x \in \mathbb{R}$, about a point $v \in \mathbb{R}$ is:

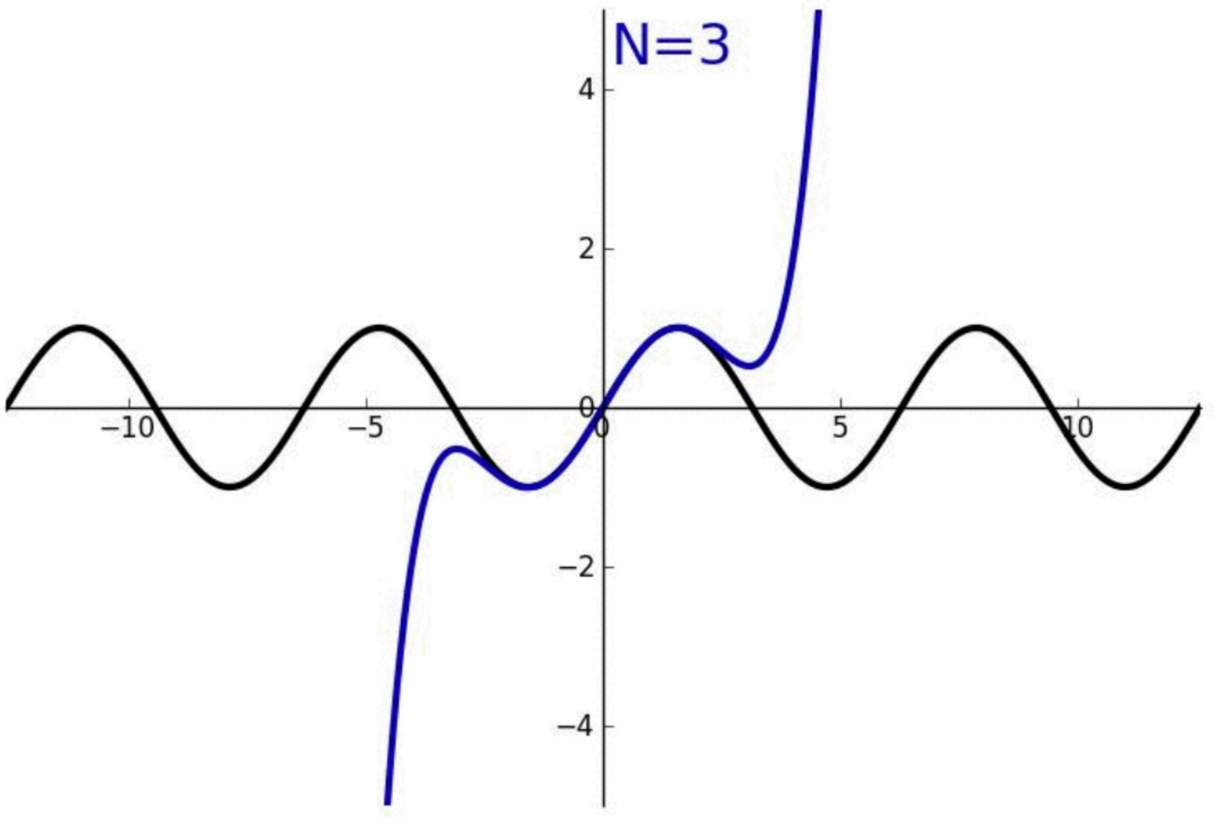
$$f(x) = f(v) + \frac{(x-v)f'(x)}{1!} + \frac{(x-v)^2 f''(x)}{2!} + \frac{(x-v)^3 f'''(x)}{3!} + \ldots$$

The **2nd-order Taylor series approximation** cuts off the expansion after the quadratic term:

$$f(x) \approx f(v) + \frac{(x-v)f'(x)}{1!} + \frac{(x-v)^2 f''(x)}{2!}$$

Slide courtesy of Matt Gormley

Taylor Series



Slide courtesy of Matt Gormley

The Newton's Method

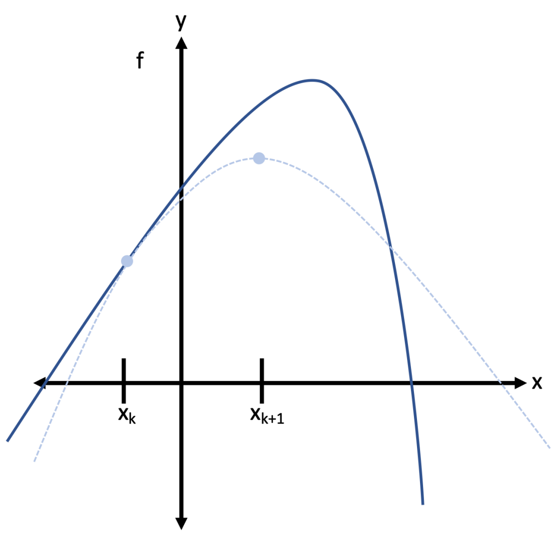
A Taylor expansion around the current point β

First order: $J(\beta + s) \approx J(\beta) + g(\beta)^\top s$

Second order: $J(\beta + s) \approx J(\beta) + g(\beta)^\top s + \frac{1}{2} s^\top H(\beta) s$

set $\nabla J(\beta + s) = 0$

$$s = -H^{-1} g(\beta)$$



Algorithm 1 Newton-Raphson Method

procedure NR($\mathcal{D}$, $\boldsymbol{\theta}^{(0)}$)
  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}^{(0)}$ ▷ Initialize parameters
  while not converged **do**
    $\mathbf{g} \leftarrow \nabla J(\boldsymbol{\theta})$ ▷ Compute gradient
    $\mathbf{H} \leftarrow \nabla^2 J(\boldsymbol{\theta})$ ▷ Compute Hessian
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \mathbf{H}^{-1}\mathbf{g}$ ▷ Update parameters
  return $\boldsymbol{\theta}$

The Newton's Method

- We have $$\boldsymbol{\beta}^* = \arg\min_{\boldsymbol{\beta}} \ell(\boldsymbol{\beta})$$

- Taking Newton's method as an example, the updating rule at the $(t+1)$-th iteration is

$$\boldsymbol{\beta}^{t+1} = \boldsymbol{\beta}^t - \left(\frac{\partial^2 \ell(\boldsymbol{\beta})}{\partial \boldsymbol{\beta} \partial \boldsymbol{\beta}^{\mathrm{T}}}\right)^{-1} \frac{\partial \ell(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}}$$

where the first- and second-order derivatives with respect to $\boldsymbol{\beta}$ are

$$\frac{\partial J(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}} = -\sum_{i=1}^{m} \hat{\boldsymbol{x}}_i (y_i - p_1(\hat{\boldsymbol{x}}_i; \boldsymbol{\beta}))$$

$$\frac{\partial^2 J(\boldsymbol{\beta})}{\partial \boldsymbol{\beta} \partial \boldsymbol{\beta}^{\mathrm{T}}} = \sum_{i=1}^{m} \hat{\boldsymbol{x}}_i \hat{\boldsymbol{x}}_i^{\mathrm{T}} p_1(\hat{\boldsymbol{x}}_i; \boldsymbol{\beta})(1 - p_1(\hat{\boldsymbol{x}}_i; \boldsymbol{\beta}))$$

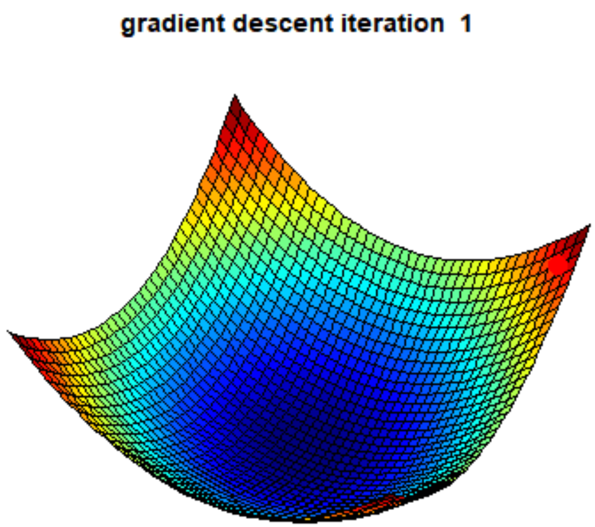
Newton's Method for Linear Regression

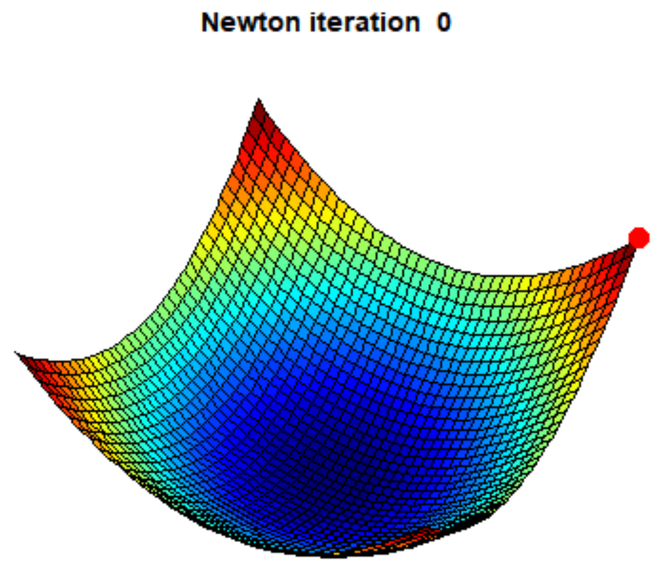
- Newton's method applied to Linear Regression (or any convex quadratic function) **converges in exactly 1-step** to the true optimum.

- This is **equivalent** to solving the Normal Equations

GD vs. Newton's Method

The Newton's method converges much faster often but it's computationally more expensive.





Multinomial logistic regression - Softmax

The multinomial logistic classifier uses a generalization of the sigmoid, called the **softmax** function, to compute $p(y_k = 1|\boldsymbol{x})$.

The **softmax** function takes a vector $\boldsymbol{z} = [z_1, z_2, \ldots, z_K]$ of K arbitrary values and maps them to a probability distribution, with each value in the range [0,1], and all the values summing to 1.

Like the sigmoid, it is an exponential function.

Multinomial logistic regression - Softmax

For a vector $\boldsymbol{z}$ of dimensionality K, the softmax is defined as:

$$\text{softmax}(\mathbf{z}_i) = \frac{\exp(\mathbf{z}_i)}{\sum_{j=1}^{K}\exp(\mathbf{z}_j)} \quad 1 \leq i \leq K$$

The softmax of an input vector $\boldsymbol{z} = [z_1, z_2, \dots, z_K]$ is thus a vector itself:

$$\text{softmax}(\mathbf{z}) = \left[\frac{\exp(\mathbf{z}_1)}{\sum_{i=1}^{K}\exp(\mathbf{z}_i)}, \frac{\exp(\mathbf{z}_2)}{\sum_{i=1}^{K}\exp(\mathbf{z}_i)}, \dots, \frac{\exp(\mathbf{z}_K)}{\sum_{i=1}^{K}\exp(\mathbf{z}_i)}\right]$$

An Example of Softmax

Given a vector:

$$\mathbf{z} = [0.6, 1.1, -1.5, 1.2, 3.2, -1.1]$$

the resulting (rounded) softmax($\mathbf{z}$) is

$$[0.055, 0.090, 0.006, 0.099, 0.74, 0.010]$$

Like the sigmoid, the softmax has the property of squashing values toward 0 or 1. Thus if one of the inputs is larger than the others, it will tend to push its probability toward 1, and suppress the probabilities of the smaller inputs.

Multinomial logistic regression

- When we apply softmax for logistic regression, we'll need separate weight vectors $\boldsymbol{w}_k$ and bias b_k for each of the K classes. The probability of each of our output classes $\hat{y}_k$ can thus be computed as:

$$p(y_k = 1 \mid \boldsymbol{x}) = \frac{\exp(\boldsymbol{w}_k \cdot \boldsymbol{x} + b_k)}{\sum_{j=1}^{K} \exp(\boldsymbol{w}_j \cdot \boldsymbol{x} + b_j)}$$

- If we represent the weights in matrix and bias in vectors, we can compute $\hat{\boldsymbol{y}}$, the vector of output probabilities for each of the K classes, by a single elegant equation:

$$\hat{\boldsymbol{y}} = \text{softmax}(\boldsymbol{W}^T \boldsymbol{x} + \boldsymbol{b})$$

Note: for more efficient computation by modern vector processing hardware

Multinomial logistic regression

The cross-entropy loss for a single example $\boldsymbol{x}$

$$\ell_{\text{CE}}(\hat{\boldsymbol{y}}, \boldsymbol{y}) = -\sum_{k=1}^{K} y_k \log \hat{y}_k \quad (y_c = 1 \text{ and} y_j = 0, \forall j \neq c)$$

$$= -\log \hat{y}_c \quad (\text{ where } c \text{ is the correct class})$$

negative log likelihood loss

$$= -\log \hat{p}(y_c = 1 \mid \boldsymbol{x})$$

$$= -\log \frac{\exp(\boldsymbol{w}_c \cdot \boldsymbol{x} + b_c)}{\sum_{j=1}^{K} \exp(\boldsymbol{w}_j \cdot \boldsymbol{x} + b_j)}$$

Gradient of the weight vector for class k

$$\frac{\partial \ell_{\text{CE}}}{\partial \boldsymbol{w}_k} = -(y_k - \hat{y}_k)\boldsymbol{x}$$

$$= -(y_k - p(y_k = 1 \mid \boldsymbol{x}))\boldsymbol{x}$$

$$= -\left(y_k - \frac{\exp(\boldsymbol{w}_k \cdot \boldsymbol{x} + b_k)}{\sum_{j=1}^{K} \exp(\boldsymbol{w}_j \cdot \boldsymbol{x} + b_j)}\right)\boldsymbol{x}$$

Summary

Data: Inputs are continuous vectors of length d. Outputs are discrete labels.

$$\mathcal{D} = \left\{ \boldsymbol{x}^{(i)}, y^{(i)} \right\}_{i=1}^{m} \text{ where } \boldsymbol{x} \in \mathbb{R}^d \text{ and } y \in \{0, 1\}$$

Model: Logistic function applied to dot product of parameters with input vector.

$$p(y = 1 \mid \boldsymbol{x}) = \frac{1}{1 + \exp(-\boldsymbol{\beta}^T \boldsymbol{x})}$$

sigmoid function

Learning: finds the parameters that minimize some objective function.

maximum likelihood estimation

$$\boldsymbol{\beta}^* = \arg\min_{\boldsymbol{\beta}} J(\boldsymbol{\beta})$$

iterative optimization

Prediction: Output is the most probable class.

$$\hat{y} = \arg\max_{y \in \{0,1\}} p(y \mid \boldsymbol{x})$$