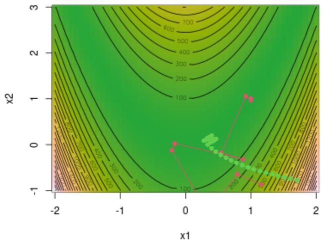
# Optimization in Machine Learning

## Second order methods
## Newton-Raphson




**Learning goals**

- Newton-Raphson
- Limitations

# FROM FIRST TO SECOND ORDER METHODS

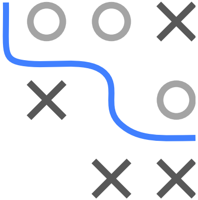
- So far: **First order methods**
  $\Rightarrow$ *Gradient* information, i.e., first derivatives
- Now: **Second order methods**
  $\Rightarrow$ *Hessian* information, i.e., second derivatives

# NEWTON-RAPHSON

**Assumption:** $f \in \mathcal{C}^2$

**Aim:** Find stationary point $\mathbf{x}^*$, i.e., $\nabla f(\mathbf{x}^*) = \mathbf{0}$

we now work with the gradient



**Idea:** Find root of first order Taylor approximation of $\nabla f(\mathbf{x})$:

$$
\begin{aligned}
\nabla f(\mathbf{x}) \approx \nabla f(\mathbf{x}^{[t]}) + \nabla^2 f(\mathbf{x}^{[t]})(\mathbf{x} - \mathbf{x}^{[t]}) &= \mathbf{0} \\
\nabla^2 f(\mathbf{x}^{[t]})(\mathbf{x} - \mathbf{x}^{[t]}) &= -\nabla f(\mathbf{x}^{[t]}) \\
\mathbf{x}^{[t+1]} &= \mathbf{x}^{[t]} - \left(\nabla^2 f(\mathbf{x}^{[t]})\right)^{-1} \nabla f(\mathbf{x}^{[t]})
\end{aligned}
$$

if approximation is valid we'll jump to the optimum in one step

**Update scheme:**

$$\mathbf{x}^{[t+1]} = \mathbf{x}^{[t]} + \mathbf{d}^{[t]}$$

with $\mathbf{d}^{[t]} = -\left(\nabla^2 f(\mathbf{x}^{[t]})\right)^{-1} \nabla f(\mathbf{x}^{[t]})$

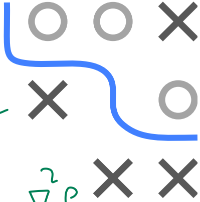
# NEWTON-RAPHSON

**Note:** In practice, we get $\mathbf{d}^{[t]}$ by solving the linear system

$$\nabla^2 f(\mathbf{x}^{[t]})\mathbf{d}^{[t]} = -\nabla f(\mathbf{x}^{[t]})$$

with direct (matrix decompositions) or iterative methods.

check out Cholesky decomposition (because $\nabla^2 f$ is symmetric)



**Relaxed/Damped Newton-Raphson:** Use step size $\alpha > 0$ with

$$\mathbf{x}^{[t+1]} = \mathbf{x}^{[t]} + \alpha\mathbf{d}^{[t]}$$

(closer we are to quadratic the less useful this is)

to satisfy Wolfe conditions (or just Armijo rule)

# ANALYTICAL EXAMPLE WITH QUADRATIC FORM

$$f(x_1, x_2) = x_1^2 + \frac{x_2^2}{2}$$

This is a seperable func

Update direction: $\mathbf{d}^{[t]} = -\left(\nabla^2 f(x_1^{[t]}, x_2^{[t]})\right)^{-1} \nabla f(x_1^{[t]}, x_2^{[t]})$

$$\nabla f(x_1, x_2) = \begin{pmatrix} 2x_1 \\ x_2 \end{pmatrix}, \quad \nabla^2 f(x_1, x_2) = \begin{pmatrix} 2 & 0 \\ 0 & 1 \end{pmatrix}$$

Hessian will be symetric

(Understand this)

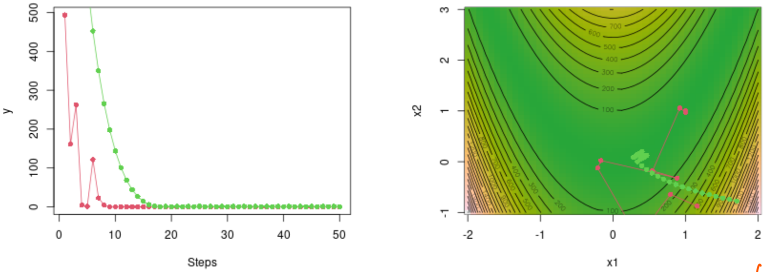
First step:

$$\begin{pmatrix} x_1^{[1]} \\ x_2^{[1]} \end{pmatrix} = \begin{pmatrix} x_1^{[0]} \\ x_2^{[0]} \end{pmatrix} + \mathbf{d}^{[0]} = \begin{pmatrix} x_1^{[0]} \\ x_2^{[0]} \end{pmatrix} - \begin{pmatrix} 1/2 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 2x_1^{[0]} \\ x_2^{[0]} \end{pmatrix}$$

$$= \begin{pmatrix} x_1^{[0]} \\ x_2^{[0]} \end{pmatrix} + \begin{pmatrix} -x_1^{[0]} \\ -x_2^{[0]} \end{pmatrix} = \mathbf{0}$$

**Note:** Newton-Raphson only needs one iteration for quadratic forms

# NEWTON-RAPHSON VS. GD ON BRANIN FUNCTION




Red: Newton-Raphson. Green: Gradient descent.
Newton-Raphson has much better convergence speed here.

what was the step size here? How do we set it for comparing?

# DISCUSSION

**Advantage:**

- For $f$ sufficiently smooth:

Newton-Raphson converges *locally* quadratically
(i.e., for starting points close enough to stationary point)

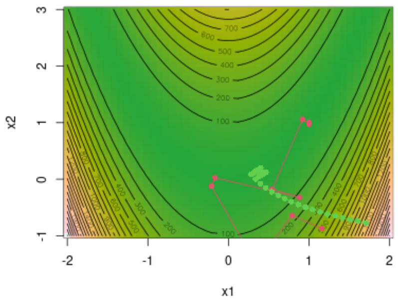
**Disadvantage:**

- For “bad” starting points:

Newton-Raphson may diverge

# LIMITATIONS

**Problem 1:** In general, $\mathbf{d}^{[t]}$ is not a descent direction




**But**: If Hessian is positive definite, $\mathbf{d}^{[t]}$ is descent direction:

~~Definition~~ + A p.d. ⇒ $A^{-1}$ p.d.

$$\nabla f(\mathbf{x}^{[t]})^\top \mathbf{d}^{[t]} = -\nabla f(\mathbf{x}^{[t]})^\top \left(\nabla^2 f(\mathbf{x}^{[t]})\right)^{-1} \nabla f(\mathbf{x}^{[t]}) < 0$$

Near minimum, Hessian is positive definite. For initial steps, Hessian is often not positive definite and Newton-Raphson may give non-descending update directions

# LIMITATIONS

**Problem 2:** Hessian can be **computationally expensive** to calculate, since descent direction $\mathbf{d}^{[t]}$ is the solution of the linear system

$$\nabla^2 f(\mathbf{x}^{[t]})\mathbf{d}^{[t]} = -\nabla f(\mathbf{x}^{[t]}).$$

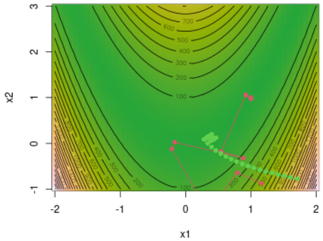
**Aim**: Find quasi-second order methods not relying on exact Hessians

- Quasi-Newton method
- Gauss-Newton algorithm (for least squares)

# Optimization in Machine Learning

## Second order methods

## Quasi-Newton




**Learning goals**

- Newton-Raphson vs. Quasi-Newton
- SR1
- BFGS

# QUASI-NEWTON: IDEA

Start point of **QN method** is (as with NR) a Taylor approximation of the gradient, except that H is replaced by a **pd** matrix $\boldsymbol{A}^{[t]}$:

$$
\begin{aligned}
\nabla f(\mathbf{x}) &\approx \nabla f(\mathbf{x}^{[t]}) + \nabla^2 f(\mathbf{x}^{[t]})(\mathbf{x} - \mathbf{x}^{[t]}) = \mathbf{0} && \text{NR} \\
\nabla f(\mathbf{x}) &\approx \nabla f(\mathbf{x}^{[t]}) + \boldsymbol{A}^{[t]} \quad (\mathbf{x} - \mathbf{x}^{[t]}) = \mathbf{0} && \text{QN}
\end{aligned}
$$

The update direction:

$$
\begin{aligned}
\boldsymbol{d}^{[t]} &= -\nabla^2 f(\mathbf{x}^{[t]})^{-1} \nabla f(\mathbf{x}^{[t]}) && \text{NR} \\
\boldsymbol{d}^{[t]} &= -(\boldsymbol{A}^{[t]})^{-1} \quad \nabla f(\mathbf{x}^{[t]}) && \text{QN}
\end{aligned}
$$

# QUASI-NEWTON: IDEA

1. Select a starting point $\mathbf{x}^{[0]}$ and initialize pd matrix $\mathbf{A}^{[0]}$ (can also be a diagonal matrix - a very rough approximation of Hessian).
2. Calculate update direction by solving

$$\boldsymbol{A}^{[t]}\boldsymbol{d}^{[t]} = -\nabla f(\mathbf{x}^{[t]})$$

   and set $\boldsymbol{x}^{[t+1]} = \boldsymbol{x}^{[t]} + \alpha^{[t]}\boldsymbol{d}^{[t]}$ (Step size through backtracking)
3. Calculate an efficient update $\mathbf{A}^{[t+1]}$,
   based on $\mathbf{x}^{[t]}$, $\mathbf{x}^{[t+1]}$, $\nabla f(\mathbf{x}^{[t]})$, $\nabla f(\mathbf{x}^{[t+1]})$ and $\mathbf{A}^{[t]}$.

# QUASI-NEWTON: IDEA

Usually the matrices $\boldsymbol{A}^{[t]}$ are calculated recursively by performing an additive update

$$\boldsymbol{A}^{[t+1]} = \boldsymbol{A}^{[t]} + \boldsymbol{B}^{[t]}.$$

How $\boldsymbol{B}^{[t]}$ is constructed is shown on the next slides.

**Requirements** for the matrix sequence $\boldsymbol{A}^{[t]}$:

1. Symmetric pd, so that $\boldsymbol{d}^{[t]}$ are descent directions.
2. Low computational effort when solving LES

$$\boldsymbol{A}^{[t]}\boldsymbol{d}^{[t]} = -\nabla f(\mathbf{x}^{[t]})$$

3. Good approximation of Hessian: The “modified” Taylor series for $\nabla f(\boldsymbol{x})$ (especially for $t \to \infty$) should provide a good approximation

$$\nabla f(\mathbf{x}) \approx \nabla f(\mathbf{x}^{[t]}) + \boldsymbol{A}^{[t]}(\mathbf{x} - \mathbf{x}^{[t]})$$

# SYMMETRIC RANK 1 UPDATE (SR1)

Simplest approach: symmetric rank 1 updates (**SR1**) of form

$$\boldsymbol{A}^{[t+1]} \leftarrow \boldsymbol{A}^{[t]} + \boldsymbol{B}^{[t]} = \boldsymbol{A}^{[t]} + \beta \boldsymbol{u}^{[t]} (\boldsymbol{u}^{[t]})^\top$$

with appropriate vector $\boldsymbol{u}^{[t]} \in \mathbb{R}^n$, $\beta \in \mathbb{R}$.

# SYMMETRIC RANK 1 UPDATE (SR1)

**Choice of $\boldsymbol{u}^{[t]}$:**
Vectors should be chosen so that the “modified” Taylor series corresponds to the gradient:

$$
\begin{aligned}
\nabla f(\mathbf{x}) &\overset{!}{=} \nabla f(\mathbf{x}^{[t+1]}) + \boldsymbol{A}^{[t+1]}(\mathbf{x} - \mathbf{x}^{[t+1]}) \\
\nabla f(\mathbf{x}) &= \nabla f(\mathbf{x}^{[t+1]}) + \left(\boldsymbol{A}^{[t]} + \beta \boldsymbol{u}^{[t]}(\boldsymbol{u}^{[t]})^\top\right) \underbrace{(\mathbf{x} - \mathbf{x}^{[t+1]})}_{:=\boldsymbol{s}^{[t+1]}} \\
\underbrace{\nabla f(\mathbf{x}) - \nabla f(\mathbf{x}^{[t+1]})}_{\boldsymbol{y}^{[t+1]}} &= \left(\boldsymbol{A}^{[t]} + \beta \boldsymbol{u}^{[t]}(\boldsymbol{u}^{[t]})^\top\right) \boldsymbol{s}^{[t+1]} \\
\boldsymbol{y}^{[t+1]} - \boldsymbol{A}^{[t]}\boldsymbol{s}^{[t+1]} &= \left(\beta(\boldsymbol{u}^{[t]})^\top \boldsymbol{s}^{[t+1]}\right) \boldsymbol{u}^{[t]}
\end{aligned}
$$

Push hessian into dir of largest error. like CMA ES

For $\boldsymbol{u}^{[t]} = \boldsymbol{y}^{[t+1]} - \boldsymbol{A}^{[t]}\boldsymbol{s}^{[t+1]}$ and $\beta = \frac{1}{\left(\boldsymbol{y}^{[t+1]} - \boldsymbol{A}^{[t]}\boldsymbol{s}^{[t+1]}\right)^\top \boldsymbol{s}^{[t+1]}}$ the equation is satisfied.

# SYMMETRIC RANK 1 UPDATE (SR1)

## Advantage

- Provides a sequence of **symmetric pd** matrices
- Matrices can be inverted efficiently and stable using Sherman-Morrison:

$$(\boldsymbol{A} + \beta \boldsymbol{u}\boldsymbol{u}^\top)^{-1} = \boldsymbol{A} + \beta \frac{\boldsymbol{u}\boldsymbol{u}^\top}{1 + \beta \boldsymbol{u}^\top \boldsymbol{u}}.$$

## Disadvantage

- The constructed matrices are not necessarily pd, and the update directions $\boldsymbol{d}^{[t]}$ are therefore not necessarily descent directions

# BFGS ALGORITHM

Instead of Rank 1 updates, the **BFGS** procedure (published simultaneously in 1970 by Broyden, Fletcher, Goldfarb and Shanno) uses rank 2 modifications of the form

$$\mathbf{A}^{[t]} + \beta \mathbf{u}^{[t]}(\mathbf{u}^{[t]})^\top + \beta \mathbf{v}^{[t]}(\mathbf{v}^{[t]})^\top$$

with $\mathbf{s}^{[t]} := \mathbf{x}^{[t+1]} - \mathbf{x}^{[t]}$

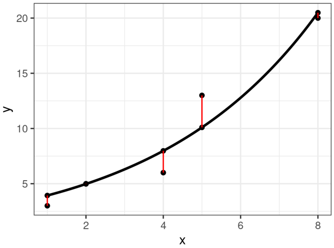
- $\mathbf{u}^{[t]} = \nabla f(\mathbf{x}^{[t+1]}) - \nabla f(\mathbf{x}^{[t]})$
- $\mathbf{v}^{[t]} = \mathbf{A}^{[t]}\mathbf{s}^{[t]}$
- $\beta = \frac{1}{(\mathbf{u}^{[t]})^\top (\mathbf{s}^{[t]})}$
- $\beta = -\frac{1}{(\mathbf{s}^{[t]})^\top \mathbf{A}^{[t]}\mathbf{s}^{[t]}}$

The resulting matrices $\mathbf{A}^{[t]}$ are positive definite and the corresponding quasi-newton update directions $\mathbf{d}^{[t]}$ are actual descent directions.

# Optimization in Machine Learning

## Second order methods
## Gauss-Newton

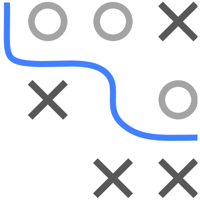
17.12.24
by Toby




### Learning goals

- Least squares
- Gauss-Newton
- Levenberg-Marquardt

# LEAST SQUARES PROBLEM



Consider the problem of minimizing a sum of squares

$$\min_{\boldsymbol{\theta}} g(\boldsymbol{\theta}),$$

where

$$g(\boldsymbol{\theta}) = r(\boldsymbol{\theta})^\top r(\boldsymbol{\theta}) = \sum_{i=1}^{n} r_i(\boldsymbol{\theta})^2$$

and

$$\begin{aligned} r : \mathbb{R}^d &\to \mathbb{R}^n \\ \boldsymbol{\theta} &\mapsto (r_1(\boldsymbol{\theta}), \ldots, r_n(\boldsymbol{\theta}))^\top \end{aligned}$$

maps parameters $\boldsymbol{\theta}$ to residuals $r(\boldsymbol{\theta})$

# LEAST SQUARES PROBLEM

**Risk minimization with squared loss** $L\left(y, f(\mathbf{x})\right) = \left(y - f(\mathbf{x})\right)^2$

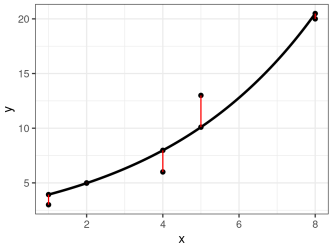
**Least squares regression:**

$$\mathcal{R}_{\text{emp}}(\boldsymbol{\theta}) = \sum_{i=1}^{n} L\left(y^{(i)}, f\left(\mathbf{x}^{(i)} \mid \boldsymbol{\theta}\right)\right) = \sum_{i=1}^{n} \underbrace{\left(y^{(i)} - f\left(\mathbf{x}^{(i)} \mid \boldsymbol{\theta}\right)\right)^2}_{r_i(\boldsymbol{\theta})^2}$$

- $f\left(\mathbf{x}^{(i)} \mid \boldsymbol{\theta}\right)$ might be a function that is **nonlinear in $\boldsymbol{\theta}$**
- Residuals: $r_i = y^{(i)} - f(\mathbf{x}^{(i)} \mid \boldsymbol{\theta})$

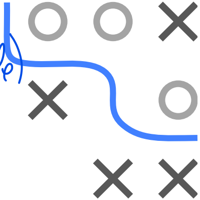
**Example:**

$$\begin{aligned}\mathcal{D} &= \left(\left(\mathbf{x}^{(i)}, y^{(i)}\right)\right)_{i=1,\dots,5} \\ &= ((1,3),(2,7),(4,12),(5,13),(7,20))\end{aligned}$$



# LEAST SQUARES PROBLEM

Suppose, we suspect an *exponential* relationship between $x \in \mathbb{R}$ and $y$



$$f(x \mid \boldsymbol{\theta}) = \theta_1 \cdot \exp(\theta_2 \cdot x), \quad \theta_1, \theta_2 \in \mathbb{R}$$ (just an example)

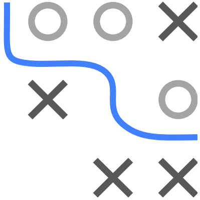
**Residuals:**

$$r(\boldsymbol{\theta}) = \begin{pmatrix} \theta_1 \exp(\theta_2 x^{(1)}) - y^{(1)} \\ \theta_1 \exp(\theta_2 x^{(2)}) - y^{(2)} \\ \theta_1 \exp(\theta_2 x^{(3)}) - y^{(3)} \\ \theta_1 \exp(\theta_2 x^{(4)}) - y^{(4)} \\ \theta_1 \exp(\theta_2 x^{(5)}) - y^{(5)} \end{pmatrix} = \begin{pmatrix} \theta_1 \exp(1\theta_2) - 3 \\ \theta_1 \exp(2\theta_2) - 7 \\ \theta_1 \exp(4\theta_2) - 12 \\ \theta_1 \exp(5\theta_2) - 13 \\ \theta_1 \exp(7\theta_2) - 20 \end{pmatrix}$$

**Least squares problem:**

$$\min_{\boldsymbol{\theta}} g(\boldsymbol{\theta}) = \min_{\boldsymbol{\theta}} \sum_{i=1}^{5} \left( y^{(i)} - \theta_1 \exp\left(\theta_2 x^{(i)}\right)\right)^2$$

# NEWTON-RAPHSON IDEA



**Approach:** Calculate Newton-Raphson update direction by solving:

$$\nabla^2 g(\boldsymbol{\theta}^{[t]})\mathbf{d}^{[t]} = -\nabla g(\boldsymbol{\theta}^{[t]}).$$

(better than inverse calc. approach) (like in normal eq.)

Gradient is calculated via chain rule

$$\nabla g(\boldsymbol{\theta}) = \nabla(r(\boldsymbol{\theta})^\top r(\boldsymbol{\theta})) = 2 \cdot J_r(\boldsymbol{\theta})^\top r(\boldsymbol{\theta}),$$

where $J_r(\boldsymbol{\theta})$ is Jacobian of $r(\boldsymbol{\theta})$.

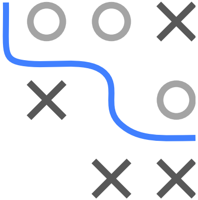
In our example:

$$J_r(\boldsymbol{\theta}) = \begin{pmatrix} \frac{\partial r_1(\boldsymbol{\theta})}{\partial \theta_1} & \frac{\partial r_1(\boldsymbol{\theta})}{\partial \theta_2} \\ \frac{\partial r_2(\boldsymbol{\theta})}{\partial \theta_1} & \frac{\partial r_2(\boldsymbol{\theta})}{\partial \theta_2} \\ \vdots & \vdots \\ \frac{\partial r_5(\boldsymbol{\theta})}{\partial \theta_1} & \frac{\partial r_5(\boldsymbol{\theta})}{\partial \theta_2} \end{pmatrix} = \begin{pmatrix} \exp(\theta_2 x^{(1)}) & x^{(1)}\theta_1 \exp(\theta_2 x^{(1)}) \\ \exp(\theta_2 x^{(2)}) & x^{(2)}\theta_1 \exp(\theta_2 x^{(2)}) \\ \exp(\theta_2 x^{(3)}) & x^{(3)}\theta_1 \exp(\theta_2 x^{(3)}) \\ \exp(\theta_2 x^{(4)}) & x^{(4)}\theta_1 \exp(\theta_2 x^{(4)}) \\ \exp(\theta_2 x^{(5)}) & x^{(5)}\theta_1 \exp(\theta_2 x^{(5)}) \end{pmatrix}$$

# NEWTON-RAPHSON IDEA

Hessian of $g$, $\mathbf{H}_g = (H_{jk})_{jk}$, is obtained via product rule:

$\nabla^2 J_y(\theta)^T y\, \theta =$



$$= \quad H_{jk} = 2 \sum_{i=1}^{n} \left( \frac{\partial r_i}{\partial \theta_j} \frac{\partial r_i}{\partial \theta_k} + r_i \frac{\partial^2 r_i}{\partial \theta_j \partial \theta_k} \right)$$

**But:**

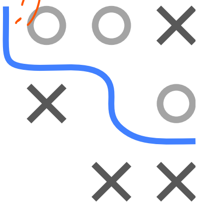
**Main problem with Newton-Raphson:**

Second derivatives can be computationally expensive.

# GAUSS-NEWTON FOR LEAST SQUARES

(why specific for least squares?)

Gauss-Newton approximates $\mathbf{H}_g$ by dropping its second order part:



$$H_{jk} = 2\sum_{i=1}^{n}\left(\frac{\partial r_i}{\partial \theta_j}\frac{\partial r_i}{\partial \theta_k} + r_i \frac{\partial^2 r_i}{\partial \theta_j \partial \theta_k}\right)$$

$$\approx 2\sum_{i=1}^{n}\frac{\partial r_i}{\partial \theta_j}\frac{\partial r_i}{\partial \theta_k}$$

$$= 2J_r(\boldsymbol{\theta})^\top J_r(\boldsymbol{\theta})$$

just drop this. crazy!

**Note**: We assume that

$$\left|\frac{\partial r_i}{\partial \theta_j}\frac{\partial r_i}{\partial \theta_k}\right| \gg \left|r_i \frac{\partial^2 r_i}{\partial \theta_j \partial \theta_k}\right|.$$

This assumption may be valid if:

- Residuals $r_i$ are small in magnitude **or**
- Functions are only "mildly" nonlinear s.t. $\frac{\partial^2 r_i}{\partial \theta_j \partial \theta_k}$ is small.

this will be a lot smaller but no rigorous explanation, more like rule of thumb

# GAUSS-NEWTON FOR LEAST SQUARES

If $J_r(\theta)^\top J_r(\theta)$ is invertible, Gauss-Newton update direction is

$$
\begin{aligned}
\mathbf{d}^{[t]} &= -\left[\nabla^2 g(\theta^{[t]})\right]^{-1} \nabla g(\theta^{[t]}) \\
&\approx -\left[J_r(\theta^{[t]})^\top J_r(\theta^{[t]})\right]^{-1} J_r(\theta^{[t]})^\top r(\theta) \\
&= -(J_r^\top J_r)^{-1} J_r^\top r(\theta)
\end{aligned}
$$

Pos. semi def.

$\triangleright$ $A^\top A$ $\quad x^\top A^\top A x = (Ax)^\top Ax = \|Ax\|_2^2 \geq 0$

there can still be 0 eigenvalues

**Advantage**:

| |
|---|
| Reduced computational complexity since no Hessian necessary. |

**Note:** Gauss-Newton can also be derived by starting with

This way we would not have noticed what we dropped (and added assump. that r is small)

$$
r(\theta) \approx r(\theta^{[t]}) + J_r(\theta^{[t]})^\top(\theta - \theta^{[t]}) = \tilde{r}(\theta)
$$

(I order Taylor for r, around $\theta^{[t]}$)

(For Newton-R. we used II order)

and $\tilde{g}(\theta) = \tilde{r}(\theta)^\top \tilde{r}(\theta)$. Then, set $\nabla \tilde{g}(\theta)$ to zero.

# LEVENBERG-MARQUARDT ALGORITHM

- **Problem:** Gauss-Newton may not decrease $g$ in every iteration but may diverge, especially if starting point is far from minimum
- **Solution:** Choose step size $\alpha > 0$ s.t.

$$\mathbf{x}^{[t+1]} = \mathbf{x}^{[t]} + \alpha \mathbf{d}^{[t]}$$

decreases $g$ (e.g., by satisfying Wolfe conditions)

- However, if $\alpha$ gets too small, an **alternative** method is the

will this satisfy Wolfe though? Do we need to recheck?

2?

**Levenberg-Marquardt algorithm**

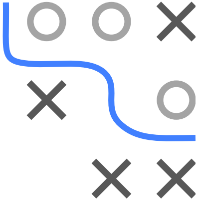
$$(J_r^\top J_r + \lambda D)\mathbf{d}^{[t]} = -J_r^\top r(\theta)$$

↳ just adding vals to diagonal

- $D$ is a positive diagonal matrix
- $\lambda = \lambda^{[t]} > 0$ is the *Marquardt parameter* and chosen at each step

# LEVENBERG-MARQUARDT ALGORITHM

?4 Do we use this right from the start or we do some of the alg. than in the end this?



- **Interpretation:** Levenberg-Marquardt *rotates* Gauss-Newton update directions towards direction of *steepest descent*

Let $D = I$ for simplicity. Then:

invertable because psd + pd = pd

mult by $\lambda$ and then separate

$$
\begin{aligned}
\lambda \mathbf{d}^{[t]} &= \lambda (J_r^\top J_r + \lambda I)^{-1} (-J_r^\top r(\boldsymbol{\theta})) \\
&= (I - J_r^\top J_r/\lambda + (J_r^\top J_r)^2/\lambda^2 \mp \cdots)(-J_r^\top r(\boldsymbol{\theta})) \\
&\to -J_r^\top r(\boldsymbol{\theta}) = -\nabla g(\boldsymbol{\theta})/2
\end{aligned}
$$

$\lambda$ disappears

for $\lambda \to \infty$

Boo also $(\lambda I)^{-1} = \frac{1}{\lambda} I$

we set $A = \lambda I$ and only compute it's inverse

we just hope this is right

**Note:** $(\mathbf{A} + \mathbf{B})^{-1} = \sum_{k=0}^{\infty} (-\mathbf{A}^{-1}\mathbf{B})^k \mathbf{A}^{-1}$ if $\|\mathbf{A}^{-1}\mathbf{B}\| < 1$

TODO

How many terms do we need in practice?

- Therefore: $\mathbf{d}^{[t]}$ approaches direction of negative gradient of $g$
- Often: $D = \text{diag}(J_r^\top J_r)$ to get scale invariance (**Recall:** $J_r^\top J_r$ is positive semi-definite $\Rightarrow$ non-negative diagonal)

why? $e_i A^\top A e_i$

# Optimization in Machine Learning

## Second order methods
## Optimization in R

**Learning goals**

- `optim()`

# OPTIMIZATION IN R

Function **optim()** from base R provides algorithms for general optimization problems:

- **Brent:** Only for one-dimensional functions. Use the function **optimize()**. Can be useful if **optim()** is called within another function.
- **CG:** conjugated Gradient Methods
- **BFGS, Quasi-Newton**

# OPTIMIZATION IN R

General Call:

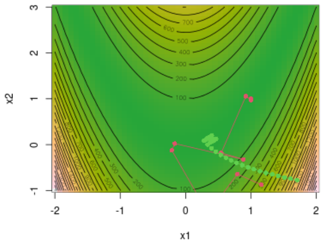
```
optim(par, fn, gr, method, lower, upper, control)
```

- **par** starting values of the parameters to be optimized
- **fn** (objective) function, to be optimized (default: minimized)
- **gr** gradient / derivative with corresponding method
- **method** optimization method (see above)
- **lower/upper** boundaries for optimization (L-BFGS-B)
- **control** List of control parameters

# Optimization in Machine Learning

## Second order methods
## Newton-Raphson vs Gradient Descent

18.12.24




### Learning goals

- Comparison of Newton-Raphson and Gradient Descent
- Pure Newton vs relaxed Newton with step size

# NEWTON-RAPHSON AND GD (RECAP)

- Gradient Descent: **first order method**
  $\Rightarrow$ *Gradient* information, i.e., first derivatives
- Newton-Raphson: **second order method**
  $\Rightarrow$ *Hessian* information, i.e., second derivatives

**Gradient Descent:**

$$\mathbf{x}^{[t+1]} = \mathbf{x}^{[t]} - \alpha \nabla f(\mathbf{x}^{[t]})$$

**Pure Newton-Raphson:**

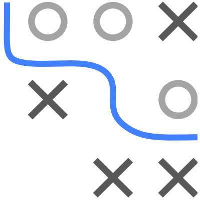
$$\mathbf{x}^{[t+1]} = \mathbf{x}^{[t]} - \left(\nabla^2 f(\mathbf{x}^{[t]})\right)^{-1} \nabla f(\mathbf{x}^{[t]})$$

**Relaxed/Damped Newton-Raphson:**

$$\mathbf{x}^{[t+1]} = \mathbf{x}^{[t]} - \alpha \left(\nabla^2 f(\mathbf{x}^{[t]})\right)^{-1} \nabla f(\mathbf{x}^{[t]})$$

# COMPARISON SIMULATION SET-UP

Comparison of Newton-Raphson, relaxed NR and GD+momentum:



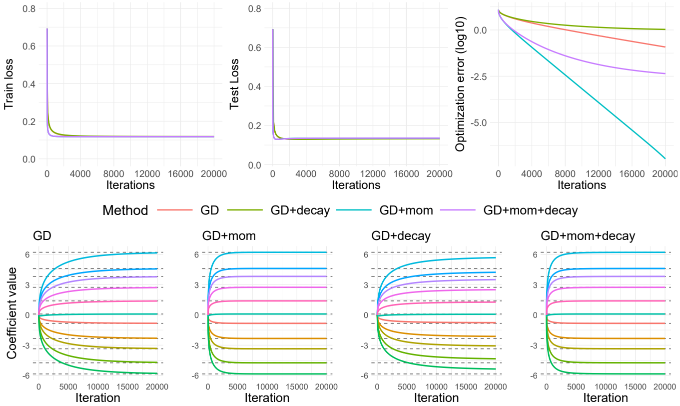
- **Logistic regression** (log loss) simulation with $n = 500$ samples and $p = 11$ features, where $\boldsymbol{\theta}^* = (-5, -4, \dots 0, \dots, 4, 5)^\top$, and $\boldsymbol{X} \sim \mathcal{N}(\mathbf{0}, \Sigma)$ for $\Sigma = \boldsymbol{I}$ (i.i.d.) or $\Sigma_{i,j} = 0.99^{|i-j|}$ (corr. features)
- To simulate response, we set $y^{(i)} \sim \mathcal{B}(\pi^{(i)}), \pi^{(i)} = \frac{1}{1+e^{-\left(\boldsymbol{x}^{(i)}\right)^\top \boldsymbol{\theta}^*}}$
- Indep. features result in a condition number of $\approx 2.9$ while corr. features yield (moderately) bad condition number $\approx 600$

(High $\kappa$ => Big changes in output when we change input slightly. Not "robust/stable")

- ERM has unique global minimizer (convexity) but no closed-form solution. We can approximate $\hat{\boldsymbol{\theta}}$ using `glm` solution
- We als track the optimization error $\|\boldsymbol{\theta} - \hat{\boldsymbol{\theta}}\|_2$
- For relaxed NR we use $\alpha = 0.7$ and for GD we set $\alpha = 1$, momentum to 0.8 and use no step size control

# LOGISTIC REGRESSION (GD VARIANTS RECAP)

Recall comparison of GD variants on log. reg. in last chapter:



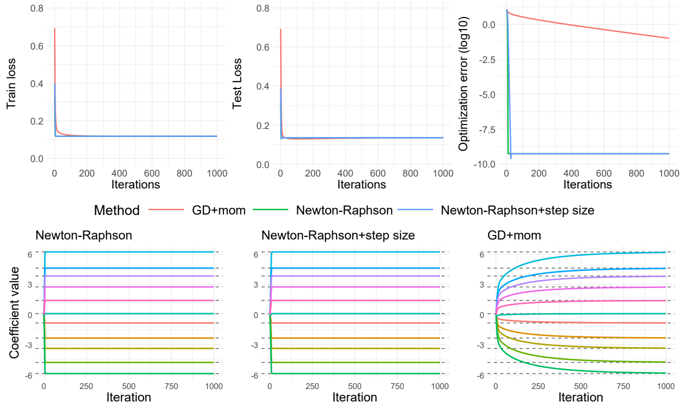

Dotted lines indicate global minimizers.

**GD+momentum** was fastest $\Rightarrow$ now compare w/ Newton-Raphson.

**NB**: GD+momentum only converges after several thousand steps

# LOGISTIC REGRESSION (GD VS. NR)

Let's run GD vs. NR for 1000 **steps** (independent features):



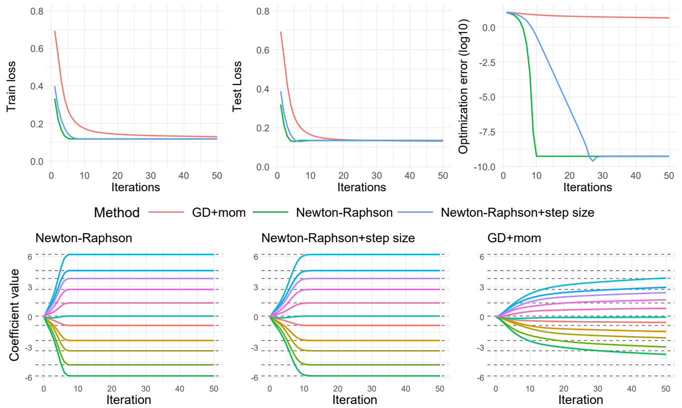

Dotted lines indicate global minimizers.

**NR** and **relaxed NR** $\Rightarrow$ almost instantaneous convergence (see optimization error). Using $\alpha < 1$ slightly slows down **relaxed NR**.

**GD+mom** several orders of magnitude slower than NR.

# LOGISTIC REGRESSION (GD VS. NR)

Let's run the same configuration only for 50 **steps** to see clearer picture:




Dotted lines indicate global minimizers.

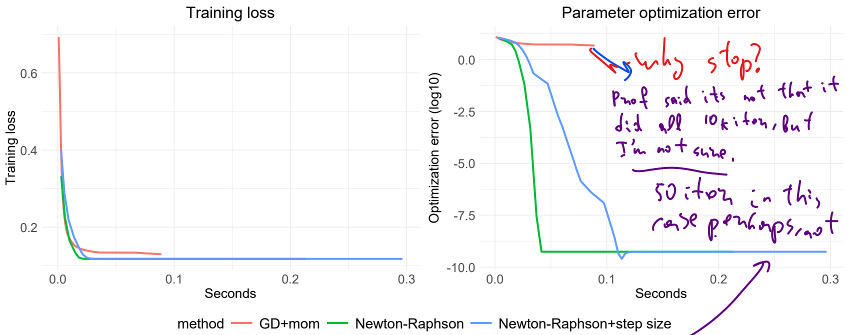
**NR** takes $\approx$ 10 steps to reach same optimization error as **GD+mom** after $20,000$ steps! **Relaxed NR** with $\alpha < 1$ shows no advantage here.

# RUNTIME COMPARISON (INDEP.)

on the other hand

Clearly, NR makes more progress than GD per iteration. OTOH Newton steps are much more expensive than GD updates

$\Rightarrow$ How do NR and GD compare wrt runtime instead of iterations (50 steps)?




why stop?

Prof said its not that it did all 10k iter, but I'm not sure.

50 iter in this case perhaps, not 10k

We're more interested in runtime, rather iter num.

**Observations**:

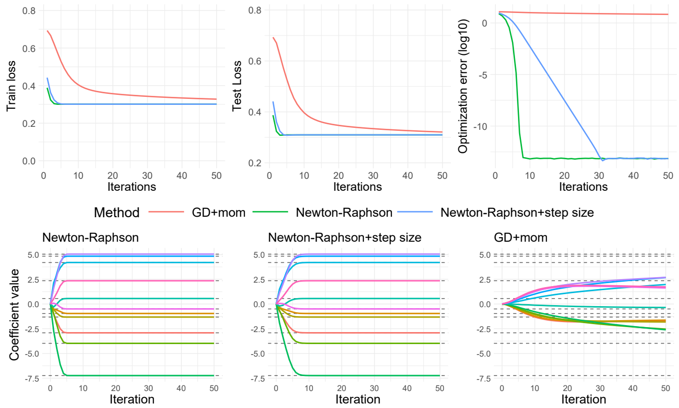
1. **NR** steps are indeed slower than **GD** steps ($\approx 3\times$ here)
2. But each step NR step is so much better than GD ($\approx 2000\times$) that per-iteration runtime advantage of GD becomes **irrelevant**

How is measured?

maybe average improvement in terms of loss, but prof doesn't know for sure

# LOGISTIC REGRESSION (CORR.)

In case of correlated features the results are very similar:

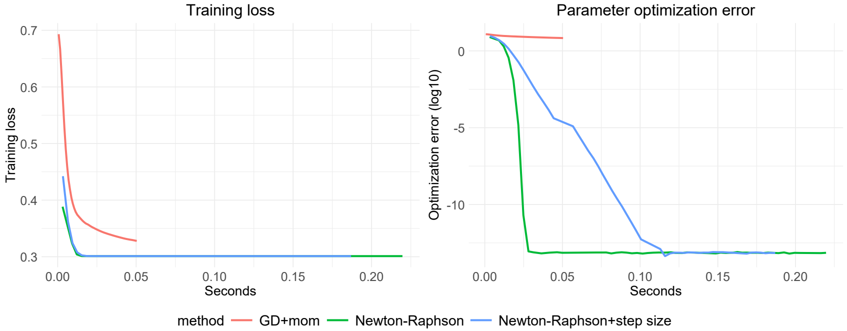



Dotted lines indicate global minimizers.

It can be noted that **NR**'s performance is unaffected by feature correlation while **GD** iterates become “warped“ compared to before

# RUNTIME COMPARISON (CORR.)

Previous conclusions on runtime comparison for independent features carry over to correlated feature case:

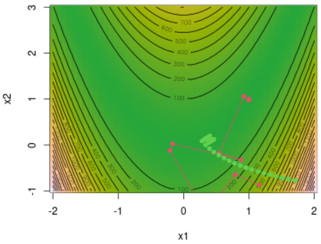



**Observations**:

1. **NR** steps are indeed slower than **GD** steps ($\approx 4\times$ here)
2. Overall **NR** is strongly superior to **GD** wrt optim error and speed

# Optimization in Machine Learning

## Second order methods
## Fisher Scoring




**Learning goals**

- Fisher Scoring
- Newton-Raphson vs. Fisher scoring
- Logistic regression

# RECAP OF NEWTON'S METHOD

Second-order Taylor expansion of log-likelihood around the current iterate $\boldsymbol{\theta}^{(t)}$:

$$\ell(\boldsymbol{\theta}) \approx \ell(\boldsymbol{\theta}^{(t)}) + \nabla \ell(\boldsymbol{\theta}^{(t)})^\top (\boldsymbol{\theta} - \boldsymbol{\theta}^{(t)}) + \frac{1}{2} (\boldsymbol{\theta} - \boldsymbol{\theta}^{(t)})^\top [\nabla^2 \ell(\boldsymbol{\theta}^{(t)})] (\boldsymbol{\theta} - \boldsymbol{\theta}^{(t)})$$

We then differentiate w.r.t. $\boldsymbol{\theta}$ and set the gradient to zero:

$$\nabla \ell(\boldsymbol{\theta}^{(t)}) + [\nabla^2 \ell(\boldsymbol{\theta}^{(t)})](\boldsymbol{\theta} - \boldsymbol{\theta}^{(t)}) = \mathbf{0}$$

Solving for $\boldsymbol{\theta}^{(t)}$ yields the pure Newton-Raphson update:

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} + [-\nabla^2 \ell(\boldsymbol{\theta}^{(t)})]^{-1} \nabla \ell(\boldsymbol{\theta}^{(t)})$$

**Potential stability issue**: pure Newton-Raphson updates do not why?
always converge. Its quadratic convergence rate is "local" in the sense that it requires starting close to a solution.

# FISHER SCORING

Fisher's scoring method replaces the negative *observed Hessian* $-\nabla^2 \ell(\boldsymbol{\theta})$ by the Fisher information matrix, i.e., the variance of $\nabla \ell(\boldsymbol{\theta})$, which, under weak regularity conditions, equals the negative *expected Hessian*

check this later,
but it's about how well behaved our func is and we can do tricks with exp and $\nabla$

$$\mathbb{E}[\nabla \ell(\boldsymbol{\theta}) \nabla \ell(\boldsymbol{\theta})^\top] = \mathbb{E}[-\nabla^2 \ell(\boldsymbol{\theta})],$$

and is positive semi-definite under exchangeability of expectation and differentiation.

Exp. over what?

$\ell$ - is log likelihood

$\prod_{i=1}^{n} p(y^{(i)} | \theta, x)$

exp over $y$-s

Therefore the Fisher scoring iterates are given by

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} + \mathbb{E}[-\nabla^2 \ell(\boldsymbol{\theta}^{(t)})]^{-1} \nabla \ell(\boldsymbol{\theta}^{(t)})$$

$E_y \nabla \log p(y)$

$\int p(y) \nabla_\theta \log p(y|\theta) =$

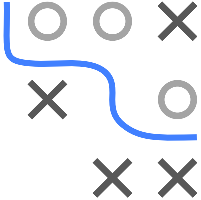
$= \int p(y) \frac{1}{p(y)} \nabla_\theta p(y|\theta) = \nabla_\theta \underbrace{\int p(y|\theta)}_{1}$

# NEWTON-RAPHSON VS. FISHER SCORING

| **Aspect** | **Newton-Raphson** | **Fisher scoring** |
|---|---|---|
| Second-order Matrix | Exact negative Hessian matrix | Fisher information matrix |
| Curvature | Exact | Approximated |
| Computational Cost | Higher | Lower (often has a simpler structure) |
| Convergence | Fast but potentially unstable (bad starting point is dangerous) | Slower but more stable |
| Positive Definite | Not guaranteed | Yes with Fisher information (Exp. of outher product) |
| Use Case | General non-linear optimization | Likelihood-based models, especially GLMs |

In many cases Newton-Raphson and Fisher scoring are equivalent (see below).

# LOGISTIC REGRESSION



The goal of logistic regression is to predict a binary event. Given $n$ observations $\left(\mathbf{x}^{(i)}, y^{(i)}\right) \in \mathbb{R}^{p+1} \times \{0, 1\}$, $y^{(i)} | \, \mathbf{x}^{(i)} \sim$ *Bernoulli*$(\pi^{(i)})$.

We want to minimize the following risk

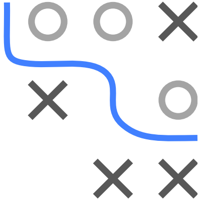
$$\mathcal{R}_{\text{emp}}(\boldsymbol{\theta}) \quad = \quad -\sum_{i=1}^{n} y^{(i)} \log(\pi^{(i)}) + \left(1 - y^{(i)} \log(1 - \pi^{(i)})\right)$$

with respect to $\boldsymbol{\theta}$, where the probabilistic classifier $\pi^{(i)} = \pi\left(\mathbf{x}^{(i)} \mid \boldsymbol{\theta}\right) = s\left(f\left(\mathbf{x}^{(i)} \mid \boldsymbol{\theta}\right)\right)$, the sigmoid function $s(f) = \frac{1}{1+\exp(-f)}$ and the score $f\left(\mathbf{x}^{(i)} \mid \boldsymbol{\theta}\right) = \boldsymbol{\theta}^\top \mathbf{x}$.

**NB**: Note that $\frac{\partial}{\partial f} s(f) = s(f)(1 - s(f))$ and $\frac{\partial f\left(\mathbf{x}^{(i)} \mid \boldsymbol{\theta}\right)}{\partial \boldsymbol{\theta}} = \left(\mathbf{x}^{(i)}\right)^\top$.

For more details we refer to the i2ml lecture.

# LOGISTIC REGRESSION



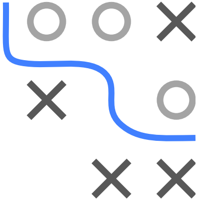
Partial derivative of empirical risk using chain rule:

$$
\begin{aligned}
\frac{\partial}{\partial \boldsymbol{\theta}} \mathcal{R}_{\text{emp}}(\boldsymbol{\theta}) &= -\sum_{i=1}^{n} \frac{\partial}{\partial \pi^{(i)}} \left(y^{(i)} \log(\pi^{(i)}) + (1 - y^{(i)}) \log(1 - \pi^{(i)})\right) \frac{\partial \pi^{(i)}}{\partial \boldsymbol{\theta}} \\
&= -\sum_{i=1}^{n} \left( \frac{y^{(i)}}{\pi^{(i)}} - \frac{1 - y^{(i)}}{1 - \pi^{(i)}} \right) \frac{\partial s(f\left(\mathbf{x}^{(i)} \mid \boldsymbol{\theta}\right))}{\partial f\left(\mathbf{x}^{(i)} \mid \boldsymbol{\theta}\right)} \frac{\partial f\left(\mathbf{x}^{(i)} \mid \boldsymbol{\theta}\right)}{\partial \boldsymbol{\theta}} \\
&= \sum_{i=1}^{n} \left(\pi^{(i)} - y^{(i)}\right) \left(\mathbf{x}^{(i)}\right)^{\top} \\
&= (\pi(\mathbf{X} \mid \boldsymbol{\theta}) - \mathbf{y})^{\top} \mathbf{X}
\end{aligned}
$$

where $\mathbf{X} = \left(\mathbf{x}^{(1)\top}, \dots, \mathbf{x}^{(n)\top}\right)^{\top} \in \mathbb{R}^{n \times (p+1)}, \mathbf{y} = \left(y^{(1)}, \dots, y^{(n)}\right)^{\top},$
$\pi(\mathbf{X} \mid \boldsymbol{\theta}) = \left(\pi^{(1)}, \dots, \pi^{(n)}\right)^{\top} \in \mathbb{R}^{n}.$
$\nabla_{\boldsymbol{\theta}} \mathcal{R}_{\text{emp}} = \left(\frac{\partial}{\partial \boldsymbol{\theta}} \mathcal{R}_{\text{emp}}\right)^{\top}$

# LOGISTIC REGRESSION

The Hessian of logistic regression:



$$\nabla^2_{\boldsymbol{\theta}} \mathcal{R}_{\text{emp}} = \frac{\partial^2}{\partial \boldsymbol{\theta}^\top \partial \boldsymbol{\theta}} \mathcal{R}_{\text{emp}} = \frac{\partial}{\partial \boldsymbol{\theta}^\top} \sum_{i=1}^{n} \left(\pi^{(i)} - y^{(i)}\right) \left(\mathbf{x}^{(i)}\right)^\top$$

$$= \sum_{i=1}^{n} \mathbf{x}^{(i)} \left(\pi^{(i)} \left(1 - \pi^{(i)}\right)\right) \left(\mathbf{x}^{(i)}\right)^\top$$

$$= \mathbf{X}^\top \mathbf{D} \mathbf{X}$$

where $\mathbf{D} \in \mathbb{R}^{n \times n}$ is a diagonal matrix containing the variances of $y^{(i)}$ on the diagonals

$$\mathbf{D} = \text{diag}\left(\pi^{(1)}(1 - \pi^{(1)}), \ldots, \pi^{(n)}(1 - \pi^{(n)})\right).$$

y y does not appear here, so the exp is the same

For log reg NR is equivalent to Fisher scoring

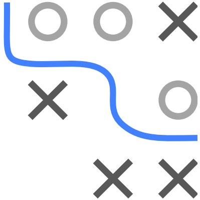
That's why we don't worry much about initialization for Log.Reg.

# LOGISTIC REGRESSION

We now have

$$\nabla_{\boldsymbol{\theta}} \mathcal{R}_{\text{emp}} = \mathbf{X}^\top (\pi(\mathbf{X}|\, \boldsymbol{\theta}) - \mathbf{y})$$

$$\nabla^2_{\boldsymbol{\theta}} \mathcal{R}_{\text{emp}} = \mathbf{X}^\top \mathbf{D} \mathbf{X}$$



Newton-Raphson:

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - [\mathbf{X}^\top \mathbf{D} \mathbf{X}]^{-1} \nabla_{\boldsymbol{\theta}^{(t)}} \mathcal{R}_{\text{emp}}$$

Fisher scoring:

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \mathbb{E}[\mathbf{X}^\top \mathbf{D} \mathbf{X}]^{-1} \nabla_{\boldsymbol{\theta}^{(t)}} \mathcal{R}_{\text{emp}}$$

# GENERALIZED LINEAR MODELS

$y|\mathbf{x} = \mathbf{x}$ belongs to an **exponential family** with density:

$$p(y|\delta, \phi) = exp\left\{\frac{y\delta - b(\delta)}{a(\phi)} + c(y, \phi)\right\},$$

where $\delta$ is the natural parameter and $\phi > 0$ is the dispersion parameter. We often take $a_i(\phi) = \frac{\phi}{w_i}$, where $\phi$ is a positive constant, and $w_i$ is the weight.

Generalized linear models (GLMs) relate the conditional mean $\mu(x) = \mathbb{E}[y|\mathbf{x}]$ of $Y$ to a linear predictor $\eta$ via a strictly increasing link function $g(\mu) = \eta = \mathbf{x}^\top\theta$.

Notice that mean $\mu = b'(\delta) = g^{-1}(\eta)$, variance $Var(Y|\mathbf{x}) = a(\phi)b''(\delta)$,

$$\frac{\partial b(\delta)}{\partial\theta} = \frac{\partial b(\delta)}{\partial\delta}\frac{\partial\delta}{\partial\mu}\frac{\partial\mu}{\partial\eta}\frac{\partial\eta}{\partial\theta} = \mu\frac{1}{b''(\delta)}\frac{1}{g'(\mu)}\mathbf{x}$$

# GENERALIZED LINEAR MODELS

We can estimate $\delta$ using MLE with sample $(\mathbf{x}^{(i)}, y^{(i)})$ for $i = 1, \ldots, n$.
Take $a^{(i)}(\phi) = \frac{\phi}{w^{(i)}}$, $\phi$ is a positive constant, we could ignore it since the goal is to maximize the function:

$$
\begin{aligned}
\nabla \ell_\theta(\delta, \phi) &= \sum_{i=1}^{n} \frac{w_i(y^{(i)} - \mu^{(i)})}{b''(\delta) g'(\mu^{(i)})} \mathbf{x}^{(i)} \\
&= \sum_{i=1}^{n} \frac{w^{(i)}(y^{(i)} - \mu^{(i)}) g'(\mu^{(i)})}{b''(\delta)[g'(\mu^{(i)})]^2} \mathbf{x}^{(i)} \\
&= \mathbf{X}^\top \mathbf{W} \mathbf{G} (\mathbf{Y} - \boldsymbol{\mu})
\end{aligned}
$$

Here mistake $b''(\delta)$ depends on $\theta$ which was not taken into account

**W** is a diagonal matrix with element $\frac{w^{(i)}}{b''(\delta)[g'(\mu^{(i)})]^2}$.

**G** is a diagonal matrix with element $g'(\mu^{(i)})$.

# GENERALIZED LINEAR MODELS

$$-\nabla^2 \ell_\theta(\delta, \phi) = \sum_{i=1}^{n} \frac{w^{(i)}}{b''(\delta)[g'(\mu^{(i)})]^2} \mathbf{x}^{(i)} \mathbf{x}^{(i)\top}$$

$$- \sum_{i=1}^{n} \frac{w^{(i)}(y^{(i)} - \mu^{(i)})(g''(\mu^{(i)})/g'(\mu^{(i)}))}{b''(\delta)[g'(\mu^{(i)})]^2} \mathbf{x}^{(i)} \mathbf{x}^{(i)\top}$$

$$+ \sum_{i=1}^{n} \frac{w^{(i)}(y^{(i)} - \mu^{(i)})(\partial b''(\delta)[g'(\mu^{(i)})]^2/\partial \mu^{(i)})}{[b''(\delta)]^2[g'(\mu^{(i)})]^4} \mathbf{x}^{(i)} \mathbf{x}^{(i)\top}$$

$E\, y = \mu$

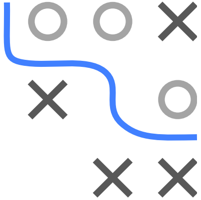
so we have

$\mu - \mu = 0$

This contains a mistake prof will fix later

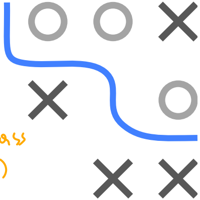
$$\mathbb{E}[-\nabla^2 \ell_\theta(\delta, \phi)] = \sum_{i=1}^{n} \frac{w^{(i)}}{b''(\delta)[g'(\mu^{(i)})]^2} \mathbf{x}^{(i)} \mathbf{x}^{(i)\top} = \mathbf{X}^\top \mathbf{W} \mathbf{X}$$

Iteratively Reweighted Least Squares (IRLS) with weights $\frac{w^{(i)}}{b''(\delta)[g'(\mu^{(i)})]^2}$



# GENERALIZED LINEAR MODELS

Equivalence also works for NNs, prof. didn't say an example of a case where equiv doesn't hold, but something that falls out of GLM class (not sure what)



Fisher scoring:

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} + (\mathbf{X}^\top \mathbf{W} \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{W} \mathbf{G} (\mathbf{Y} - \boldsymbol{\mu})$$

$$= (\mathbf{X}^\top \mathbf{W} \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{W} \left( \mathbf{G}(\mathbf{Y} - \boldsymbol{\mu}) + \mathbf{X} \boldsymbol{\theta}^{(t)} \right)$$

Kind of like normal equation (OLS)

$$(X^T X)^{-1} X^T y$$

projected space of $g$ $y$ (don't know what's this)

For canonical link where $\eta = g(\mu) = \mathbf{x}^\top \theta$, the second and third term of Hessian vanishes and Hessian coincides with Fisher information matrix.

This will now be a convex problem with Fisher scoring equal to Newton's method. There are also hybrid algorithms that start out with

IRLS which is easier to initialize, and switch over to Newton-Raphson after some iterations.