Lecture Notes for

# Machine Learning in Python

⟦👨‍🏫, 👨‍💻, 🐍, 👨‍🔬⟧

Professor Eric Larson

## Optimization Techniques for Logistic Regression

# Class Logistics and Agenda

- Logistics: Guest lecture next time, NVIDIA
- Agenda
  - Numerical Optimization Techniques
    - Quasi Newton Methods
  - Town Hall, Lab 3
- **Last Time:**
  - Logistic regression update equations
  - Line Searches
  - Stochastic small batches
  - Hessian-based methods

# Class Overview, by topic

**Table Data Visualization**
Numpy, Pandas, Seaborn
Overviews with some in-depth discussion

**Dimension Reduction and Image Processing**
Scikit-learn, Scikit Image,
Intuition only, Some mathematics

**Linear and Logistic Regression**
Numpy, Recreate API for Scikit-learn
Detailed mathematics for simple optimization
***intuition for advanced optimization***

**Neural Networks and Back Prop.**
Numpy
Detailed mathematics for NN operations

**Wide and Deep Networks** | **Convolutional Networks** | **Recurrent Networks**
Keras, Tensorflow
Intuition, Detailed implement.

**Ethics in Language Models**
ConceptNet
Case studies

$$\mathbf{H}_{j,k}(\mathbf{w}) = \frac{\partial}{\partial w_k} \boxed{\frac{\partial}{\partial w_j} l(\mathbf{w})} \longrightarrow \frac{\partial}{\partial w_j} l(\mathbf{w}) = \sum_i \left( y^{(i)} - g(\mathbf{x}^{(i)} \cdot \mathbf{w}) \right) x_j^{(i)}$$

$$\mathbf{H}_{j,k}(\mathbf{w}) = \frac{\partial}{\partial w_k} \sum_i \left( y^{(i)} - g(\mathbf{x}^{(i)} \cdot \mathbf{w}) \right) x_j^{(i)}$$

**Self Test:**
Hessian assumptions (on website)

$$= \cancel{\sum_i \frac{\partial}{\partial w_k} y^{(i)} x_j^{(i)}} - \sum_i \frac{\partial}{\partial w_k} g(\mathbf{x}^{(i)} \cdot \mathbf{w}) x_j^{(i)}$$

no dependence on $w_k$, zero

$$= - \sum_i x_j^{(i)} \boxed{\frac{\partial}{\partial w_k} g(\mathbf{x}^{(i)} \cdot \mathbf{w})}$$

already know this as $g(1-g)x_k$

$$\mathbf{H}_{j,k}(\mathbf{w}) = - \sum_{i=1}^{M} \left[ g(\mathbf{x}^{(i)} \cdot \mathbf{w})(1 - g(\mathbf{x}^{(i)} \cdot \mathbf{w})) \right] \cdot x_k^{(i)} x_j^{(i)}$$

for each $j,k$ pair

# Review: Beyond the Hessian


**Solution:**

**Approximate** the Hessian and iteratively **update** our guess, try to **eliminate** the need to find an **inverse**

# Approximation of Hessian, Rank One

$$\underbrace{\mathbf{H}_{k+1}}_{\text{approx. Hessian}} \cdot \underbrace{(\mathbf{w}_{k+1} - \mathbf{w}_k)}_{\text{Change in } w} = \underbrace{\nabla l(\mathbf{w}_{k+1}) - \nabla l(\mathbf{w}_k)}_{\text{Change in gradient}}$$

**Secant Property**

new guess guess update, low rank

$$\mathbf{H}_{k+1} = \mathbf{H}_k + \alpha_k \mathbf{u}_k \cdot \mathbf{u}_k^T$$

Rank One Hessian Guess

Each new guess $k \rightarrow k+1$ of $\mathbf{w}_{k+1}$ allows for new Hessian Approx.

$$\mathbf{H}_{k+1}\mathbf{s}_k = \mathbf{v}_k \quad \rightarrow \quad (\mathbf{H}_k + \alpha_k \mathbf{u}_k \cdot \mathbf{u}_k^T)\mathbf{s}_k = \mathbf{v}_k$$

secant property

plug in and solve for $\mathbf{u}_k$ and $\alpha_k$

$$\mathbf{H}_{k+1}^{-1} = \mathbf{H}_k^{-1} - \frac{\mathbf{H}_k^{-1}\mathbf{u}_k \cdot \mathbf{u}_k^T \mathbf{H}_k^{-1}}{1 + \mathbf{u}_k^T \mathbf{H}_k^{-1} \mathbf{u}_k}$$

Rank One Hessian Inverse

**Develop Iterative Algorithm Keep Updating Quasi-Hessian**

# Approximation of Hessian, Rank Two

$$\underbrace{\mathbf{H}_{k+1}}_{\text{approx.}} \cdot \underbrace{(\mathbf{w}_{k+1} - \mathbf{w}_k)}_{\mathbf{s}_k} = \underbrace{\nabla l(\mathbf{w}_{k+1}) - \nabla l(\mathbf{w}_k)}_{\mathbf{v}_k}$$

**Secant Property**

$$\mathbf{H}_{k+1} = \mathbf{H}_k + \alpha_k \mathbf{u}_k \cdot \mathbf{u}_k^T - \beta_k \mathbf{z}_k \cdot \mathbf{z}_k^T$$

Rank Two Hessian

$$\mathbf{H}_{k+1}\mathbf{s}_k = \mathbf{v}_k \quad \rightarrow \quad (\mathbf{H}_k + \alpha_k \mathbf{u}_k \cdot \mathbf{u}_k^T - \beta_k \mathbf{z}_k \cdot \mathbf{z}_k^T)\mathbf{s}_k = \mathbf{v}_k$$

$$\mathbf{u}_k = \mathbf{v}_k \qquad \mathbf{z}_k = \mathbf{H}_k \mathbf{s}_k \qquad \alpha_k = \frac{1}{\mathbf{u}_k^T \mathbf{s}_k} \qquad \beta_k = \frac{1}{\mathbf{z}_k^T \mathbf{s}_k}$$

solve for $\mathbf{u}_k, \mathbf{z}_k, \alpha_k, \beta_k$

$$\mathbf{H}_{k+1}^{-1} = \mathbf{H}_k^{-1} + \frac{(\mathbf{s}_k^T \mathbf{v}_k + \mathbf{H}_k^{-1})(\mathbf{s}_k \mathbf{s}_k^T)}{(\mathbf{s}_k^T \mathbf{v}_k)^2} - \frac{\mathbf{H}_k^{-1} \mathbf{v}_k \mathbf{s}_k^T + \mathbf{s}_k \mathbf{v}_k^T \mathbf{H}_k^{-1}}{\mathbf{s}_k^T \mathbf{v}_k}$$

Rank Two Inverse

**Develop Iterative Algorithm Keep Updating Quasi-Hessian**

# Demo

`06. Optimization`

Quasi-Newton Methods

- BFGS Hessian Approximation
- Practical Example: Titanic

# David F. Shanno

- Started out as Engineer in Gulf Oil, but wasn't super motivated by profits …
    - leaves for academic research
- 1967: Got into a new field, "math programming"
- Went to a lecture and thought he could improve on algorithm (conditioning on approximation)
- Wrote with Broyden, Fletcher, Goldfarb and they all thought what they were doing was different
- Turns out it was same, release a joint paper (before meeting yet), Shared the credit
- Left Chicago because good friend was murdered, wanted a safer place for kids
- Offered Presidency of Toronto University ($$$$)
- Turned it down to continue math research

1938 - 2019

# Town Hall

$$L_2 = C\sum_j w_j^2$$

```
penalty = '12'
```

$$L_1 = C\sum_j |w_j|$$

```
penalty = '11'
```

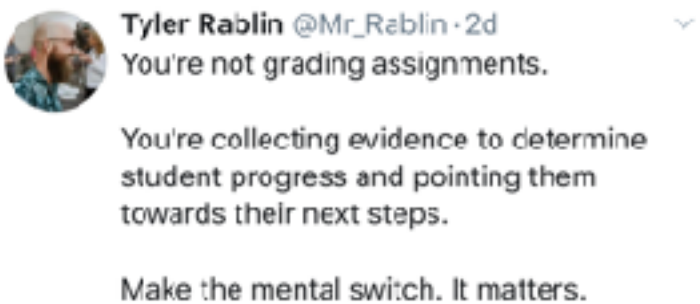
$$L_{12} = C_1\sum_j |w_j| + C_2\sum_j w_j^2$$

```
penalty = 'elasticnet'
```

**Warning:** The choice of the algorithm depends on the penalty chosen. Supported penalties by solver:

- 'lbfgs' - ['l2', None]
- 'liblinear' - ['l1', 'l2']
- 'newton-cg' - ['l2', None]
- 'newton-cholesky' - ['l2', None]
- 'sag' - ['l2', None]
- 'saga' - ['elasticnet', 'l1', 'l2', None]

# Lab 3, Town Hall


Tyler Rablin @Mr_Rablin · 2d
You're not grading assignments.

You're collecting evidence to determine student progress and pointing them towards their next steps.

Make the mental switch. It matters.

# Back Up Slides

# Last time

$$p(y^{(i)} = 1 \mid x^{(i)}, w) = \frac{1}{1 + \exp\left(w^T x^{(i)}\right)}$$

$$l(w) = \sum_i \left( y^{(i)} \ln[g(w^T x^{(i)})] + (1 - y^{(i)})(\ln[1 - g(w^T x^{(i)})]) \right)$$

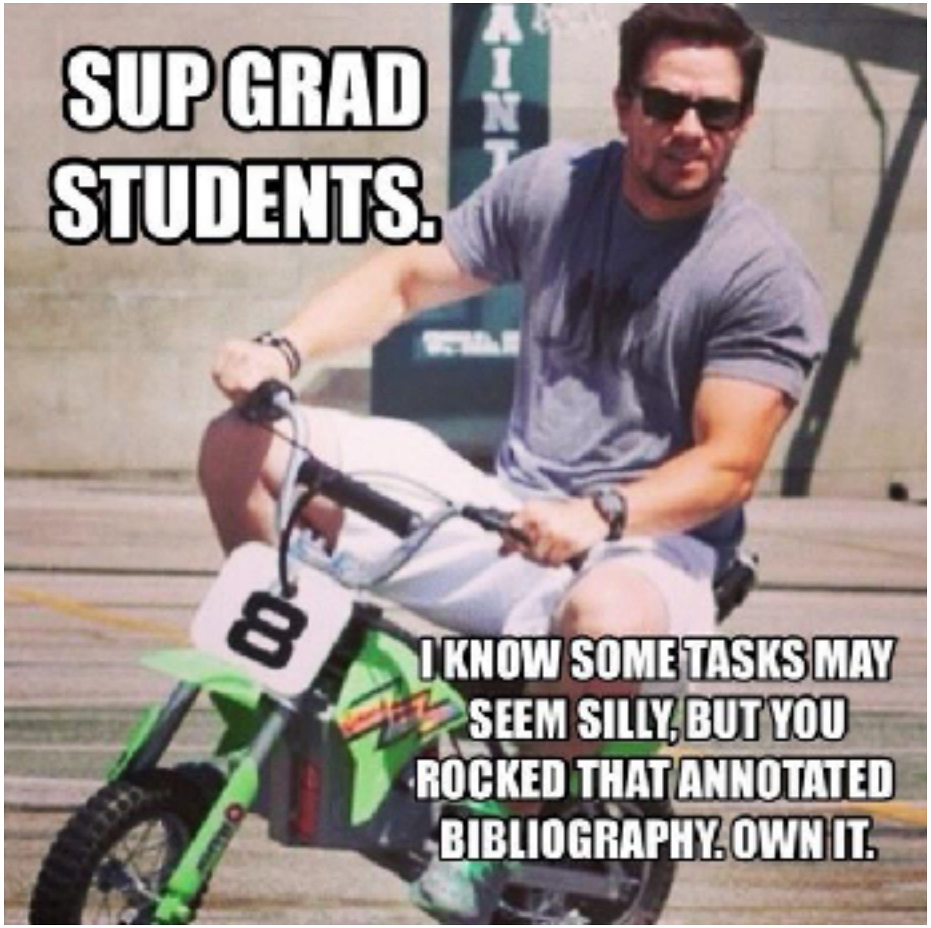
$$\underbrace{w_j}_{\text{new value}} \leftarrow \underbrace{w_j}_{\text{old value}} + \underbrace{\eta \sum_{i=1}^{M} (y^{(i)} - g(x^{(i)}))x_j^{(i)}}_{\text{gradient}}$$

$$w \leftarrow w + \eta \sum_{i=1}^{M} (y^{(i)} - g(x^{(i)}))x^{(i)}$$

$$w \leftarrow w + \eta \left[ \underbrace{\nabla l(w)_{old}}_{\text{old gradient}} - C \cdot 2w \right]$$

```python
def _get_gradient(self,X,y):
    # programming \sum_i (yi-g(xi))xi
    gradient = np.zeros(self.w_.shape) # set
    for (xi,yi) in zip(X,y):
        # the actual update inside of sum
        gradi = (yi - self.predict_proba(xi,
        # reshape to be column vector and ad
        gradient += gradi.reshape(self.w_.sh

    return gradient/float(len(y))
```

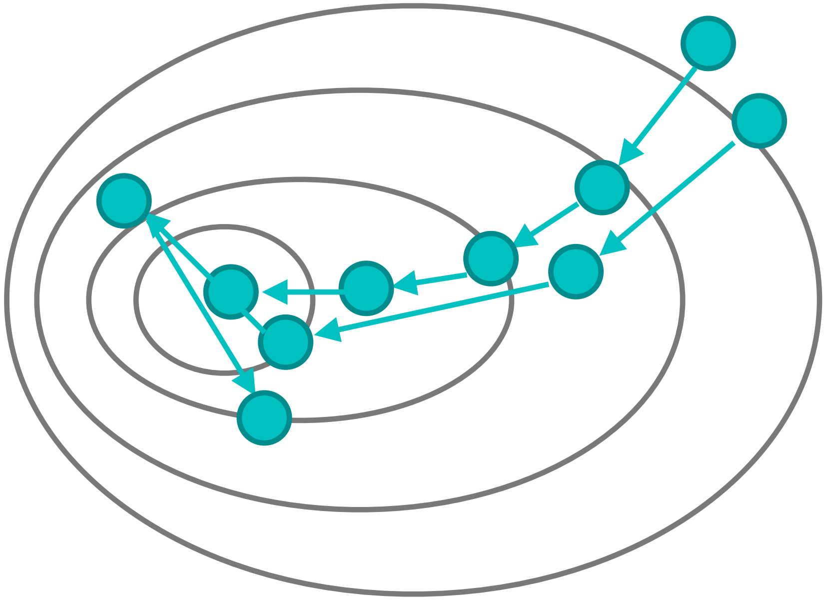

SUP GRAD STUDENTS.
I KNOW SOME TASKS MAY SEEM SILLY, BUT YOU ROCKED THAT ANNOTATED BIBLIOGRAPHY. OWN IT.

# Optimization: gradient descent

- What we know thus far:

$$\underbrace{w_j}_{\text{new value}} \leftarrow \underbrace{w_j}_{\text{old value}} + \eta \underbrace{\left[ \left( \sum_{i=1}^{M} (y^{(i)} - g(x^{(i)}))x_j^{(i)} \right) - C \cdot 2w_j \right]}_{\nabla l(w)}$$

$$w \leftarrow w + \eta \nabla l(w)$$

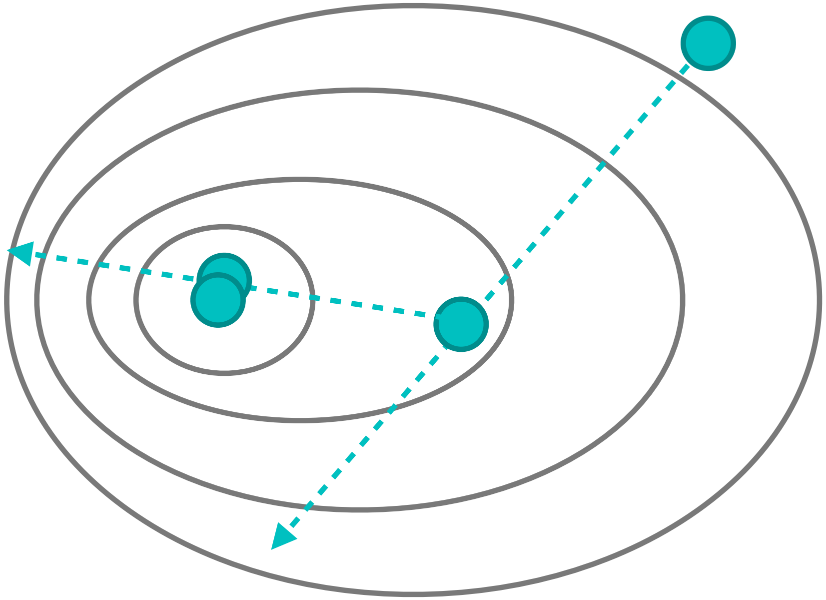


# Line Search: a better method

- Line search in direction of gradient:

$$w \leftarrow w + \eta \nabla l(w)$$

$$w \leftarrow w + \underbrace{\eta}_{\text{best step?}} \nabla l(w)$$



# Revisiting the Gradient

- How much computation is required (for gradient)?

$$\sum_{i=1}^{M}(y^{(i)} - \hat{y}^{(i)})x^{(i)} - 2C \cdot w$$

M = number of instances
N = number of features

**Self Test: How many multiplies per gradient calculation?**

A. M*N+1 multiplications
B. (M+1)*N multiplications
C. 2N multiplications
D. 2N-M multiplications

# Stochastic Methods

- How much computation is required (for gradient)?

$$\sum_{i=1}^{M}(y^{(i)} - \hat{y}^{(i)})x^{(i)} - 2C \cdot w$$

**Per iteration:**

(M+1)*N multiplications

2M add/subtract

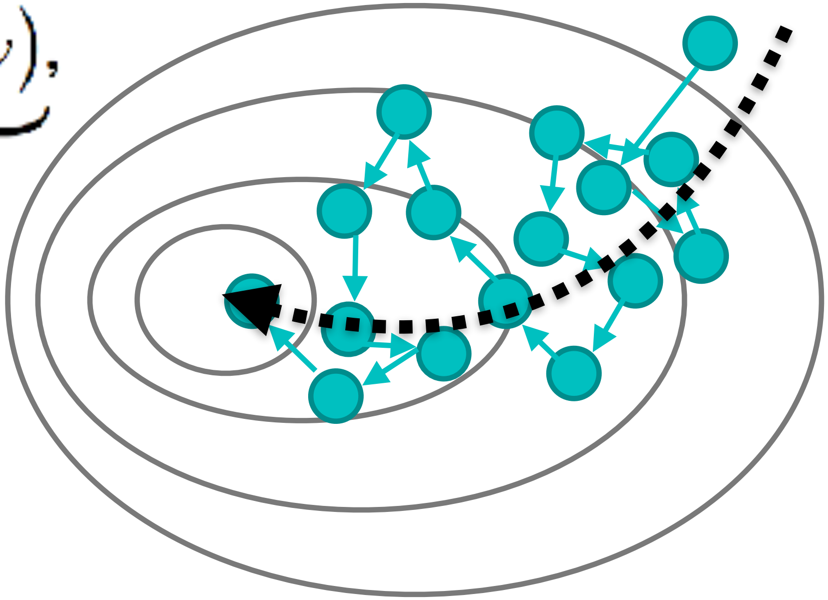
$$w \leftarrow w + \eta\underbrace{\left((y^{(i)} - \hat{y}^{(i)})x^{(i)} - 2C \cdot w\right)}_{\text{approx. gradient}},$$

$i$ chosen at random

**Per iteration:**

N+1 multiplications

1 add/subtract



# Demo

Gradient Descent (with line search)

Stochastic Gradient Descent

Hessian

Quasi-Newton Methods

Multi-processing

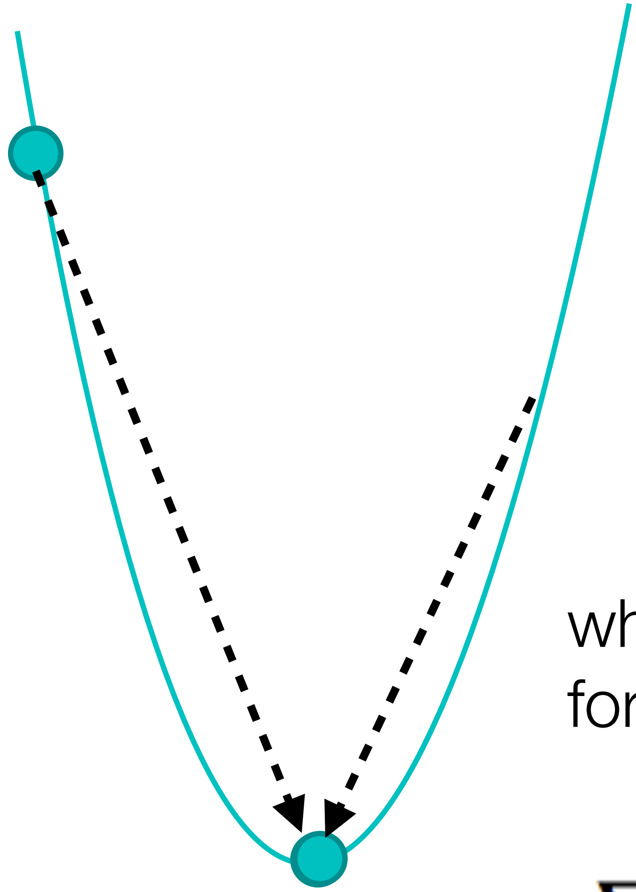
# For Next Lecture

- **Next time**: SVMs via in class assignment
- **Next Next time**: Neural Networks


SUP. HAVE YOU HEARD THE LATEST STATS JOKE?
PROBABLY.

# Can we do better than the gradient?

- Assume function is quadratic:



function of one variable:

$$w \leftarrow w - \underbrace{[\frac{\partial^2}{\partial w} l(w)]^{-1}}_{\text{inverse 2nd deriv}} \underbrace{\frac{\partial}{\partial w} l(w)}_{\text{derivative}}$$

will solve in one step!

what is the second order derivative for a multivariate function?

$$\nabla^2 l(w) = \mathbf{H}[l(w)]$$

# The Hessian

- Assume function is quadratic:

function of one variable:

$$\mathbf{H}[l(w)] = \begin{bmatrix} \frac{\partial^2}{\partial w_1} l(w) & \frac{\partial}{\partial w_1}\frac{\partial}{\partial w_2} l(w) & \cdots & \frac{\partial}{\partial w_1}\frac{\partial}{\partial w_N} l(w) \\ \frac{\partial}{\partial w_2}\frac{\partial}{\partial w_1} l(w) & \frac{\partial^2}{\partial w_2} l(w) & \cdots & \frac{\partial}{\partial w_2}\frac{\partial}{\partial w_N} l(w) \\ & \vdots & & \\ \frac{\partial}{\partial w_N}\frac{\partial}{\partial w_1} l(w) & \frac{\partial}{\partial w_N}\frac{\partial}{\partial w_2} l(w) & \cdots & \frac{\partial^2}{\partial w_N} l(w) \end{bmatrix}$$

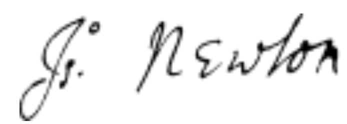
$$\nabla^2 l(w) = \mathbf{H}[l(w)]$$

# The Newton Update Method

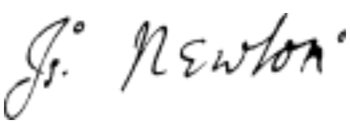
- Assume function is quadratic (in high dimensions):

$$w \leftarrow w - \underbrace{[\frac{\partial^2}{\partial w} l(w)]^{-1}}_{\text{inverse 2nd deriv}} \underbrace{\frac{\partial}{\partial w} l(w)}_{\text{derivative}}$$

$$w \leftarrow w + \eta \cdot \underbrace{\mathbf{H}[l(w)]^{-1}}_{\text{inverse Hessian}} \cdot \underbrace{\nabla l(w)}_{\text{gradient}}$$



I do not know what I may appear to the world, but to myself I seem to have been only like a boy playing on the sea-shore, and diverting myself in now and then finding a smoother pebble or a prettier shell than ordinary, whilst the great ocean of truth lay all undiscovered before me.



I do not know what I may appear to the world, but to myself I seem to have been only like a boy playing on the sea-shore, and diverting myself in now and then finding a smoother pebble or a prettier shell than ordinary, whilst the great ocean of truth lay all undiscovered before me.

$$\frac{\partial}{\partial w_j} l(w) = \sum_i \left(y^{(i)} - g(x^{(i)})\right) x_j^{(i)}$$

PLUG IN

$$H[k,j] = \frac{\partial}{\partial w_k} \frac{\partial}{\partial w_j} l = \frac{\partial}{\partial w_k} \left( \sum_i \left(y^{(i)} - g(x^{(i)})\right) x_j^{(i)} \right)$$

$$= \sum_i \frac{\partial}{\partial w_k} y^{(i)} x_j^{(i)} - \frac{\partial}{\partial w_k} g(x^{(i)}) x_j^{(i)}$$

LEFT OVER TERM

$$= - \sum_i \frac{\partial}{\partial w_k} g(x^{(i)}) x_j^{(i)}$$

Already know $\frac{\partial}{\partial w_k} g(w^T x^{(i)})$ side calculation

$$= g(x^{(i)})\left(1 - g(x^{(i)})\right) \frac{\partial}{\partial w_k}\left(w^T x^{(i)}\right)$$

$$= g(x^{(i)})\left(1 - g(x^{(i)})\right) x_k^{(i)}$$

PLUG IN

$$\therefore = - \sum_i g(x^{(i)})\left(1 - g(x^{(i)})\right) x_k^{(i)} x_j^{(i)}$$

THAT'S THE HESSIAN!

$H(k,j) =$ This is a valid equation for the Hessian, but we want to represent it using linear algebra

$$\boxed{= - \sum_i g(x^{(i)})\left(1 - g(x^{(i)})\right) x_k^{(i)} x_j^{(i)}}$$

SIGMOID

LINEAR ALG

# The Hessian for Logistic Regression

- The hessian is easy to calculate from the gradient for logistic regression

$$w \leftarrow w + \eta \cdot \underbrace{\mathbf{H}[l(w)]^{-1}}_{\text{inverse Hessian}} \cdot \underbrace{\nabla l(w)}_{\text{gradient}}$$

$$\mathbf{H}_{j,k}[l(w)] = -\sum_{i=1}^{M} g(x^{(i)})(1 - g(x^{(i)})x_k{}^{(i)}x_j{}^{(i)}$$

$$\underbrace{\sum_{i=1}^{M}(y^{(i)} - \hat{y}^{(i)})x_j^{(i)}}_{\text{gradient}}$$

$$\mathbf{H}[l(w)] = X^T \cdot \text{diag}[g(x^{(i)})(1 - g(x^{(i)}))] \cdot X$$

$$X * y_{diff}$$

$$w \leftarrow w + \eta[X^T \cdot \text{diag}[g(x^{(i)})(1 - g(x^{(i)}))] \cdot X]^{-1} \cdot X * y_{diff}$$

# Demo

Newton's method

# Problems with Newton's Method

- **Quadratic** isn't always a great assumption:
  - highly dependent on starting point
    - jumps can get **really random**!
  - near saddle points, inverse hessian **unstable**
  - hessian **not** always **invertible**…
    - or invertible with correct numerical precision

# The solution: quasi Newton methods

- In general:
  - **approximate** the **Hessian** with something numerically sound and efficiently invertible
  - **back off to gradient** descent when the approximate hessian is **not stable**
  - use **momentum** to update approximate hessian
- **A popular approach**: use Broyden-Fletcher-Goldfarb-Shanno (BFGS)
  - which you can look up if you are interested …

https://en.wikipedia.org/wiki/Broyden–Fletcher–Goldfarb–Shanno_algorithm

# BFGS

$$\mathbf{H}_0 = \mathbf{I}$$ init

$$p_k = -\mathbf{H}_k^{-1} \nabla l(w_k)$$ get update direction

find next w $$w_{k+1} \leftarrow w_k + \eta \cdot p_k$$

get scaled direction $$s_k = \eta \cdot p_k$$

$$v_k = \nabla l(w_{k+1}) - \nabla l(w_k)$$ approx gradient change

$$\mathbf{H}_{k+1} = \mathbf{H}_k + \underbrace{\frac{v_k v_k^T}{v_k^T s_k}}_{\text{approx. Hessian}} - \underbrace{\frac{\mathbf{H}_k s_k s_k^T \mathbf{H}_k}{s_k^T \mathbf{H}_k s_k}}_{\text{momentum}}$$

update Hessian and inverse Hessian approx

$$\mathbf{H}_{k+1}^{-1} = \mathbf{H}_k^{-1} + \frac{(s_k^T v_k + \mathbf{H}_k^{-1})(s_k s_k^T)}{(s_k^T v_k)^2} - \frac{\mathbf{H}_k^{-1} v_k s_k^T + s_k v_k^T \mathbf{H}_k^{-1}}{s_k^T v_k}$$

$$k = k + 1$$ increment k and repeat

invertibility of H well defined / only matrix operations