18AIE339T-MATRIX THEORY FOR ARTIFICIAL INTELLIGENCE

Dr. Antony Sophia N
Assistant Professor
Department of Computational Intelligence
SRM Institute of Science and Technology

Course Objective

Unit I – Linear System
Understand the basic concepts of linear algebra through computer science and Engineering applications

Unit II – Matrix Calculus
Learn the basic concepts of matrix calculus

Unit III – Matrix Analysis
Perform matrix analysis for various optimization algorithms

Unit IV – Matrix Solutions
Apply the concepts of vector spaces, linear transformations, matrices and inner product spaces in engineering

Unit V – Optimization
Solve problems in computer vision using optimization algorithms with single and multi-variables for large datasets

UNIT-IV MATRIX SOLUTIONS

- Gauss Elimination
- Conjugate Gradient Methods
- Singular Value Decomposition
- Least Square Method
- Gradient Computation
- Gradient Descending
- Tikhonov Regularization
- Gauss-Seidel method
- Application: Gradient Explosion and Gradient Vanishing

What is the Gaussian elimination method of matrix?

- Using row operations to convert a matrix into reduced row echelon form is sometimes called Gauss–Jordan elimination.

- In this case, the term Gaussian elimination refers to the process until it has reached its upper triangular, or (unreduced) row echelon form.

Gauss Elimination

- In mathematics, the Gaussian elimination method is known as the row reduction algorithm for solving linear equations systems.
- It consists of a sequence of operations performed on the corresponding matrix of coefficient

Can also use this method to estimate either of the following:

- ➢ The rank of the given matrix
- ➢ The determinant of a square matrix
- ➢ The inverse of an invertible matrix
- To perform row reduction on a matrix, we have to complete a sequence of elementary row operations to transform the matrix till we get 0s (i.e., zeros) on the lower left-hand corner of the matrix as much as possible.

That means the obtained matrix should be an upper triangular matrix.

There are three types of elementary row operations; they are:

Swapping two rows and this can be expressed using the notation ↔, for example, R2 ↔ R3
Multiplying a row by a nonzero number, for example, R1 → kR2 where k is some nonzero number
Adding a multiple of one row to another row, for example, R2 → R2 + 3R1

Mathematical Objects in Linear Algebra

- The obtained matrix will be in row echelon form.
- The matrix is said to be in reduced row-echelon form when all of the leading coefficients equal 1, and every column containing a leading coefficient has zeros elsewhere.
- This final form is unique; that means it is independent of the sequence of row operations used.

Gauss Elimination Method with Example

Let’s have a look at the gauss elimination method example with a solution.

Question:

Solve the following system of equations:

$x + y + z = 2$

$x + 2y + 3z = 5$

$2x + 3y + 4z = 11$

Solution:

Given system of equations are:

$x + y + z = 2$

$x + 2y + 3z = 5$

$2x + 3y + 4z = 11$

Let us write these equations in matrix form.

$$\left[\begin{array}{ccc|c} 1 & 1 & 1 & 2 \\ 1 & 2 & 3 & 5 \\ 2 & 3 & 4 & 11 \end{array}\right]$$

- Subtracting R1 from R2 to get the new elements of R2, i.e. R2 → R2 – R1. From this we get,

$$= \left[\begin{array}{ccc|c} 1 & 1 & 1 & 2 \\ 0 & 1 & 2 & 3 \\ 2 & 3 & 4 & 11 \end{array}\right]$$

- Let us make another operation as $R_3 \rightarrow R_3 - 2R_1$

$$= \left[\begin{array}{ccc|c} 1 & 1 & 1 & 2 \\ 0 & 1 & 2 & 3 \\ 0 & 1 & 2 & 7 \end{array}\right]$$

- Subtract R2 from R1 to get the new elements of R1, i.e. R1 → R1 – R2.

$$= \left[\begin{array}{ccc|c} 1 & 0 & -1 & -1 \\ 0 & 1 & 2 & 3 \\ 0 & 1 & 2 & 7 \end{array}\right]$$

Now, subtract R_2 from R_3 to get the new elements of R_3, i.

$$= \left[\begin{array}{ccc|c} 1 & 0 & -1 & -1 \\ 0 & 1 & 2 & 3 \\ 0 & 0 & 0 & 4 \end{array}\right]$$

Here,

$x - z = -1$

$y + 2z = 3$

$0 = 4$

That means, there is no solution for the given system of equations.

Gauss Elimination Method Problems

1. Solve the following system of equations using Gauss elimination method.

$x + y + z = 9$

$2x + 5y + 7z = 52$

$2x + y - z = 0$

2. Solve the following linear system using the Gaussian elimination method.

$4x - 5y = -6$

$2x - 2y = 1$

3. Using Gauss elimination method, solve:

$2x - y + 3z = 9$

$x + y + z = 6$

$x - y + z = 2$

How Gauss Elimination used in AI & ML

Optimization Problems:

- Gaussian Elimination can be used in certain optimization algorithms, especially those that involve solving linear programming problems or systems of linear constraints.

Computer Graphics:

- In computer graphics, transformations, projections, and other operations often involve solving systems of linear equations. Gaussian Elimination can be used in these contexts.

Solving for Neural Network Weights:

- In the training process of neural networks, particularly in feedforward networks with multiple layers, the learning process involves solving a system of equations. Gaussian Elimination (or other numerical methods) can be employed to determine the weights that minimize the loss function.

Image Processing:

- In certain image processing tasks, Gaussian Elimination may be used for tasks such as image deblurring or image reconstruction from incomplete data.

Example: 2D Translation

Suppose we want to perform a 2D translation on a point represented by coordinates (x, y).

We want to translate it by a vector (dx, dy) resulting in the new coordinates (x', y').

The transformation equations for translation are:

$$x' = x + dx$$
$$y' = y + dy$$

This can be represented as a system of linear equations:

$$x' - x - dx = 0$$
$$y' - y - dy = 0$$

In matrix form, this system looks like:

$$\begin{bmatrix} 1 & 0 & -dx \\ 0 & 1 & -dy \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x' \\ y' \end{bmatrix}$$

Using Gaussian Elimination

Convert to Augmented Matrix:

$$\left[\begin{array}{ccc|c} 1 & 0 & -dx & x' \\ 0 & 1 & -dy & y' \end{array}\right]$$

Apply Gaussian Elimination:

Perform row operations to get the augmented matrix into row echelon form:

$$\left[\begin{array}{ccc|c} 1 & 0 & -dx & x' \\ 0 & 1 & -dy & y' \end{array}\right]$$

Back-Substitution:

From the first row, we get:

$$x - dx = x' \implies dx = x - x'$$

From the second row:

$$y - dy = y' \implies dy = y - y'$$

Example: Linear Programming

Consider the following linear programming problem:

Maximize: $Z = 2x + 3y + 4z$

Subject to:

1. $x + y + z \leq 5$
2. $2x + y - z \leq 8$
3. $x - y + 2z \leq 6$

And non-negativity constraints:

1. $x, y, z \geq 0$

Step 1: Convert Inequalities to Equations

Introduce slack variables for each inequality:

1. $x + y + z + s1 = 5$
2. $2x + y - z + s2 = 8$
3. $x - y + 2z + s3 = 6$

Step 2: Formulate the Augmented Matrix

Create the augmented matrix:

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & | & 5 \\ 2 & 1 & -1 & 0 & 1 & 0 & | & 8 \\ 1 & -1 & 2 & 0 & 0 & 1 & | & 6 \end{bmatrix}$$

Step 3: Apply Gaussian Elimination

1. R2=R2−2R1

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & -1 & -3 & -2 & 1 & 0 \\ 1 & -1 & 2 & 0 & 0 & 1 \end{bmatrix}$$

2. R3=R3−R1

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & -1 & -3 & -2 & 1 & 0 \\ 0 & -2 & 1 & -1 & 0 & 1 \end{bmatrix}$$

Conjugate Gradient Methods

What is the conjugate gradient method of a matrix?

The Conjugate Gradient (CG) method is an iterative optimization algorithm used for solving systems of linear equations and, more broadly, for minimizing quadratic functions. It's particularly effective for large, sparse, symmetric positive definite matrices.

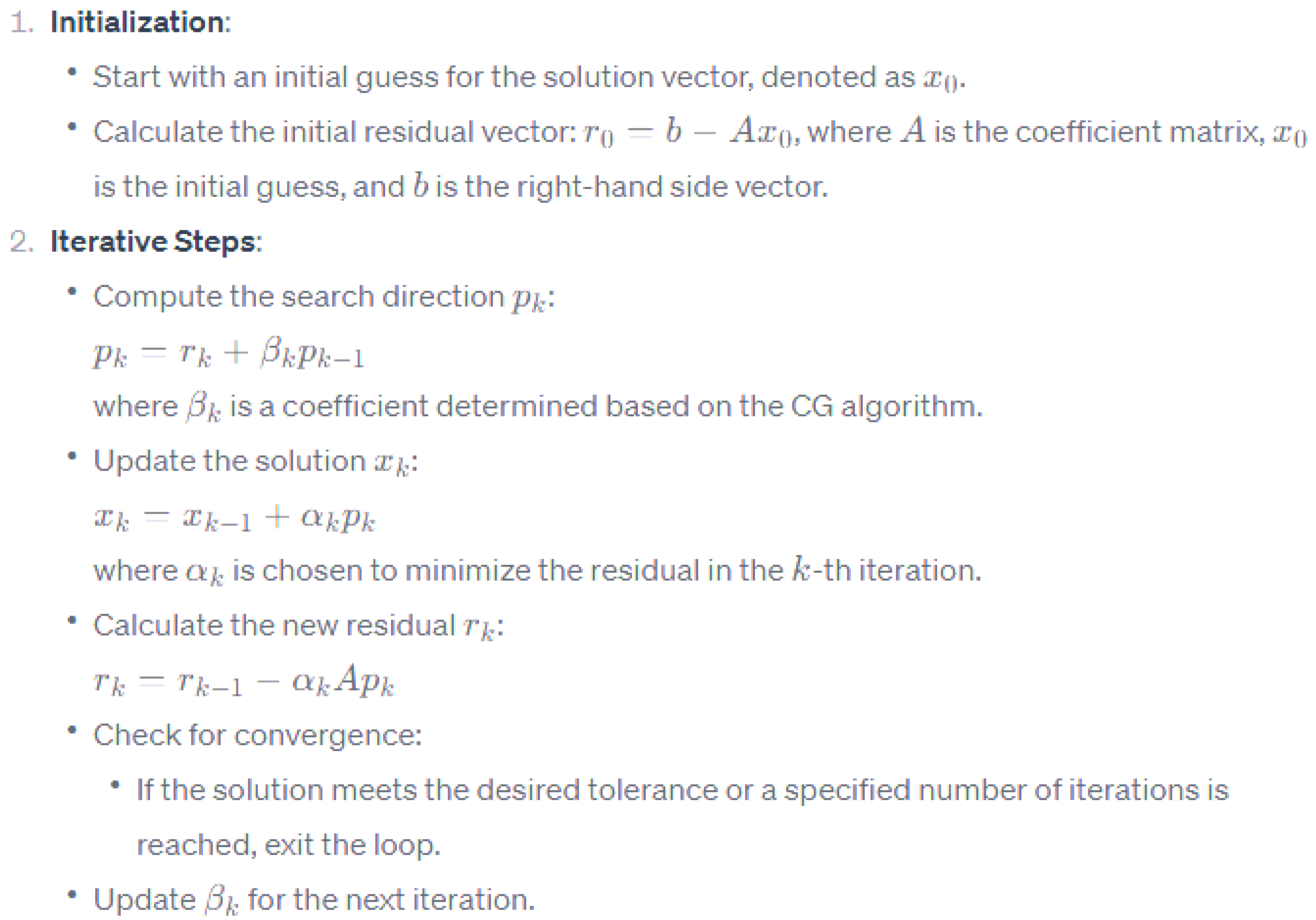
- Conjugate Gradient algorithm is used to solve a linear system, or equivalently, optimize a quadratic convex function.

- It sets the learning path direction such that they are conjugates with respect to the coefficient matrix A and hence the process is terminated after at most the dimension of A iterations.

- The conjugate gradient method can be applied to an arbitrary n-by-m matrix by applying it to normal equations A^TA and right-hand side vector A^Tb, since A^TA is a symmetric positive-semidefinite matrix for any A.

- The result is conjugate gradient on the normal equations (CGNR).

How the CG method works?



1. **Initialization**:
   - Start with an initial guess for the solution vector, denoted as x_0.
   - Calculate the initial residual vector: $r_0 = b - Ax_0$, where A is the coefficient matrix, x_0 is the initial guess, and b is the right-hand side vector.
2. **Iterative Steps**:
   - Compute the search direction p_k:
     $p_k = r_k + \beta_k p_{k-1}$
     where β_k is a coefficient determined based on the CG algorithm.
   - Update the solution x_k:
     $x_k = x_{k-1} + \alpha_k p_k$
     where α_k is chosen to minimize the residual in the k-th iteration.
   - Calculate the new residual r_k:
     $r_k = r_{k-1} - \alpha_k A p_k$
   - Check for convergence:
     - If the solution meets the desired tolerance or a specified number of iterations is reached, exit the loop.
   - Update β_k for the next iteration.

Example:

Consider the following system of linear equations:

$$3x + 2y = 9$$
$$2x + 6y = -4$$

This system can be represented in matrix form as:

$$Ax = b$$

Where:

$$A = \begin{bmatrix} 3 & 2 \\ 2 & 6 \end{bmatrix}$$

$$x = \begin{bmatrix} x \\ y \end{bmatrix}$$

$$b = \begin{bmatrix} 9 \\ -4 \end{bmatrix}$$

Step 1: Initialization

1. Initial guess: $x_0 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$
2. Calculate the initial residual vector:

$$r_0 = b - Ax_0 = \begin{bmatrix} 9 \\ -4 \end{bmatrix} - \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 9 \\ -4 \end{bmatrix}$$

Step 2: Iterative Steps

Next, we'll perform the iterative steps to find the solution using the Conjugate Gradient method.

Iteration 1:

1. Calculate the search direction $p_1 = r_0$:

$$p_1 = r_0 = \begin{bmatrix} 9 \\ -4 \end{bmatrix}$$

1. Calculate the step size α_1:

$$\alpha_1 = \frac{r_0^T \cdot r_0}{p_1^T \cdot A \cdot p_1}$$

$$\alpha_1 = \frac{\begin{bmatrix} 9 & -4 \end{bmatrix} \cdot \begin{bmatrix} 9 \\ -4 \end{bmatrix}}{\begin{bmatrix} 9 \\ -4 \end{bmatrix}^T \cdot \begin{bmatrix} 3 & 2 \\ 2 & 6 \end{bmatrix} \cdot \begin{bmatrix} 9 \\ -4 \end{bmatrix}}$$

$$\alpha_1 \approx 0.558$$

1. Update the solution $x_1 = x_0 + \alpha_1 p_1$:

$$x_1 = \begin{bmatrix} 0 \\ 0 \end{bmatrix} + 0.558 \cdot \begin{bmatrix} 9 \\ -4 \end{bmatrix} \approx \begin{bmatrix} 5.022 \\ -2.232 \end{bmatrix}$$

1. Calculate the new residual $r_1 = r_0 - \alpha_1 A p_1$:

$$r_1 = \begin{bmatrix} 9 \\ -4 \end{bmatrix} - 0.558 \cdot \begin{bmatrix} 3 & 2 \\ 2 & 6 \end{bmatrix} \cdot \begin{bmatrix} 9 \\ -4 \end{bmatrix} \approx \begin{bmatrix} 1.775 \\ 0.558 \end{bmatrix}$$

Iteration 2 and Subsequent Iterations:

Continue these steps until convergence or a specified number of iterations is reached.

Application

Graph-based Algorithms:

- In AI, especially in graph-based algorithms like PageRank (used by Google for web search), CG can be used to find the stationary distribution of the graph.

Optimization in Machine Learning:

- CG can be used to optimize the weights in machine learning models, particularly in cases where the objective function is quadratic (e.g., in Ridge Regression or Logistic Regression with a quadratic penalty term).

Image Processing and Computer Vision:

- In tasks such as image denoising, inpainting, or solving for optical flow, CG can be utilized.

Singular Value Decomposition

- The singular value decomposition of a matrix A is the factorization of A into the product of three matrices A = UDV T where the columns of U and V are orthonormal and the matrix D is diagonal with positive real entries. The SVD is useful in many tasks.
- Singular Value Decomposition is one of the important concepts in linear algebra.
- To understand the meaning of singular value decomposition (SVD), one must be aware of the related concepts such as matrix, types of matrices, transformations of a matrix, etc.
- As this concept is connected to various concepts of linear algebra, it’s become challenging to learn the singular value decomposition of a matrix.
- In this article, you will learn the definition of singular value decomposition, examples of 2×2 and 3×3 matrix decomposition in detail.

Singular Value Decomposition of a Matrix

- Mathematically, the singular value decomposition of a matrix can be explained as follows:
- Consider a matrix A of order mxn.
- This can be uniquely decomposed as:
- $A = UDV^T$
- U is mxn and column orthogonal (that means its columns are eigenvectors of AA^T)
- $(AA^T = UDV^T VDU^T = UD^2U^T)$
- V is nxn and orthogonal (that means its columns are eigenvectors of A^TA)
- $(A^TA = VDU^T UDV^T = VD^2V^T)$
- D is nxn diagonal, where non-negative real values are called singular values.
- Learn how to find eigenvalues and eigenvectors of a matrix here.
- Let $D = \text{diag}(\sigma_1, \sigma_2,\dots, \sigma_n)$ ordered such that $\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_n$.
- If σ is a singular value of A, its square is an eigenvalue of A^TA.
- Also, let $U = (u_1\ u_2 \dots u_n)$ and $V = (v_1\ v_2 \dots v_n)$.
- Therefore,
- Here, the sum can be given from 1 to r so that r is the rank of matrix A.
- Go through the example given below to understand the process of singular value decomposition of a matrix in a better way.

Singular Value Decomposition Example

- The process of finding the singular value decomposition for 3×3 matrix and 2×2 matrix is the same.
- Let's have a look at the example of 2×2 matrix decomposition.

Question:

Find the singular value decomposition of a matrix

$$A = \begin{bmatrix} -4 & -7 \\ 1 & 4 \end{bmatrix}$$

.

Solution:

Given,

$$A = \begin{bmatrix} -4 & -7 \\ 1 & 4 \end{bmatrix}$$

So,

$$A^T = \begin{bmatrix} -4 & 1 \\ -7 & 4 \end{bmatrix}$$

Now,

$$AA^T = \begin{bmatrix} -4 & -7 \\ 1 & 4 \end{bmatrix} \begin{bmatrix} -4 & 1 \\ -7 & 4 \end{bmatrix} = \begin{bmatrix} 65 & -32 \\ -32 & 17 \end{bmatrix}$$

Finding the eigenvector for AAT.

$$|A \cdot A' - \lambda I| = 0$$

$$\begin{vmatrix} (65 - \lambda) & -32 \\ -32 & (17 - \lambda) \end{vmatrix} = 0$$

$$\therefore (65 - \lambda) \times (17 - \lambda) - (-32) \times (-32) = 0$$

$$\therefore \left(1105 - 82\lambda + \lambda^2\right) - 1024 = 0$$

$$\therefore \left(\lambda^2 - 82\lambda + 81\right) = 0$$

$$\therefore (\lambda - 1)(\lambda - 81) = 0$$

$$\therefore (\lambda - 1) = 0 \text{ or } (\lambda - 81) = 0$$

∴ The eigenvalues of the matrix A·A′ are given by λ = 1, 81.

Now,

Eigenvectors for λ = 81 are:

$$v_1 = \begin{bmatrix} -2 \\ 1 \end{bmatrix}$$

Eigenvectors for λ = 1 are:

$$v_2 = \begin{bmatrix} 0.5 \\ 1 \end{bmatrix}$$

For Eigenvector-1 $(-2, 1)$, Length L = $\sqrt{(-2)^2 + 1^2} = 2.236$

So, normalizing gives $u_1 = \left(\frac{-2}{2.236}, \frac{1}{2.236}\right) = (-0.894, 0.447)$

For Eigenvector-2 $(0.5, 1)$, Length L = $\sqrt{0.5^2 + 1^2} = 1.118$

So, normalizing gives $u_2 = \left(\frac{0.5}{1.118}, \frac{1}{1.118}\right) = (0.447, 0.894)$

Similarly, we can find the eigenvectors for A′A as:

Eigenvectors for λ = 81 are:

$$v_1 = \begin{bmatrix} 0.5 \\ 1 \end{bmatrix}$$

Eigenvectors for λ = 1 are:

$$v_2 = \begin{bmatrix} -2 \\ 1 \end{bmatrix}$$

For Eigenvector-1 $(0.5, 1)$, Length L = $\sqrt{0.5^2 + 1^2} = 1.118$

So, normalizing gives $v_1 = \left(\frac{0.5}{1.118}, \frac{1}{1.118}\right) = (0.447, 0.894)$

For Eigenvector-2 $(-2, 1)$, Length L = $\sqrt{(-2)^2 + 1^2} = 2.236$

So, normalizing gives $v_2 = \left(\frac{-2}{2.236}, \frac{1}{2.236}\right) = (-0.894, 0.447)$

Using these values, we can write the solution.

$$\therefore \Sigma = \begin{bmatrix} \sqrt{81} & 0 \\ 0 & \sqrt{1} \end{bmatrix} = \begin{bmatrix} 9 & 0 \\ 0 & 1 \end{bmatrix}$$

$$\therefore U = [u_1, u_2] = \begin{bmatrix} -0.894 & 0.447 \\ 0.447 & 0.894 \end{bmatrix}$$

V is found using formula $v_i = \frac{1}{\sigma_i} A^T \cdot u_i$

$$\therefore V = \begin{bmatrix} 0.447 & -0.894 \\ 0.894 & 0.447 \end{bmatrix}$$

Or

$$\therefore \Sigma = \begin{bmatrix} \sqrt{81} & 0 \\ 0 & \sqrt{1} \end{bmatrix} = \begin{bmatrix} 9 & 0 \\ 0 & 1 \end{bmatrix}$$

$$\therefore V = [v_1, v_2] = \begin{bmatrix} 0.447 & -0.894 \\ 0.894 & 0.447 \end{bmatrix}$$

U is found using formula $u_i = \frac{1}{\sigma_i} A \cdot v_i$

$$\therefore U = \begin{bmatrix} -0.894 & 0.447 \\ 0.447 & 0.894 \end{bmatrix}$$

Least Square Method

Definition

The least squares method is a mathematical technique for finding the best-fitting curve to a set of data points. It minimizes the sum of the squared differences (the "errors") between observed and estimated values. This method is widely used in regression analysis and curve fitting.

- Let A be an m×n matrix and let b be a vector in Rm. A ***least-squares solution*** of the matrix equation Ax=b is a vector Kx in Rn such that

  dist(b,AKx)≤dist(b,Ax)

- for all other vectors x in Rn.

Compute a least-squares solution

Let A be an m×n matrix and let b be a vector in Rn. Here is a method for computing a least-squares solution of Ax=b:

1. Compute the matrix $A^T A$ and the vector $A^T b$.
2. Form the augmented matrix for the matrix equation $A^T Ax= A^T b$, and row reduce.
3. This equation is always consistent, and any solution Kx is a least-squares solution.

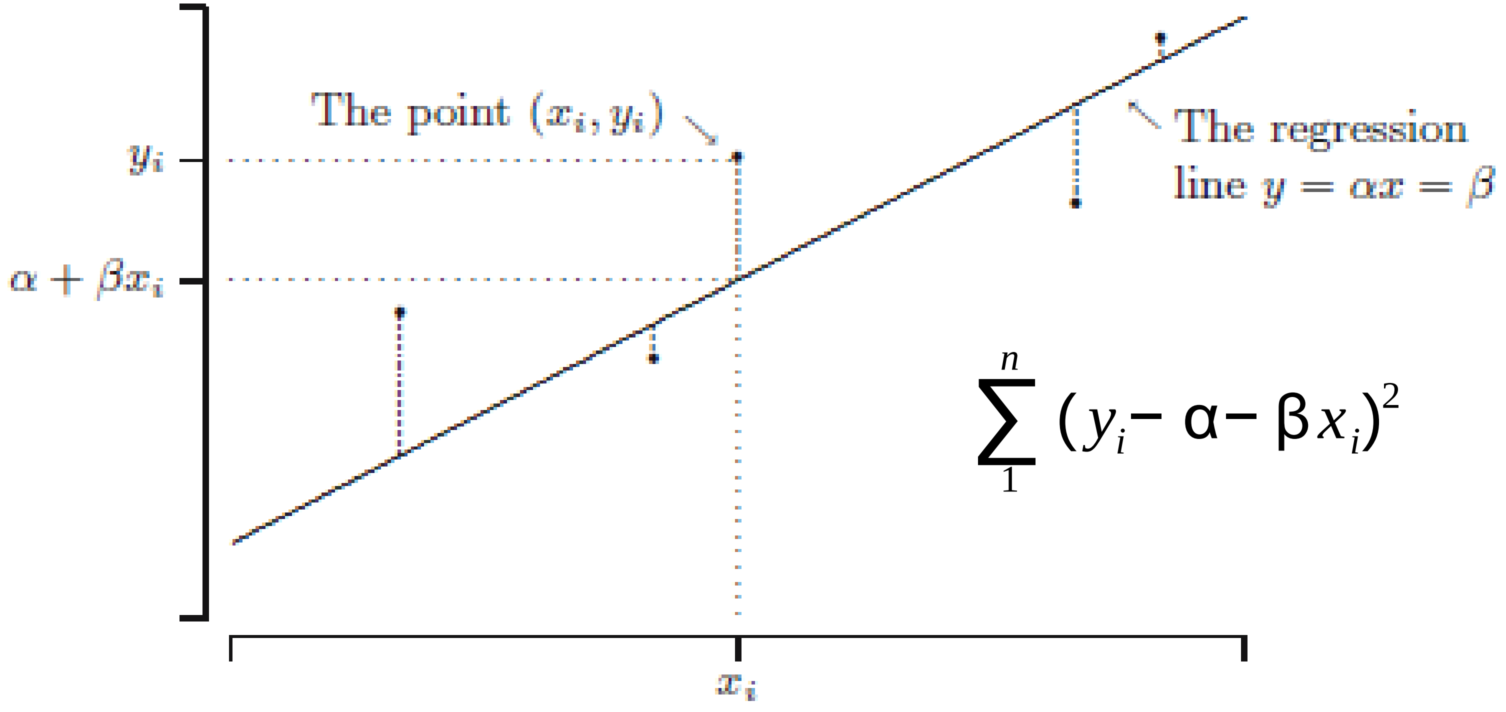
Least Squares

Given is a bivariate dataset (x1, y1), …, (xn, yn), where x1, …, xn are nonrandom and Yi = α + βxi + Ui are random variables for i = 1, 2, . . ., n. The random variables U1, U2, …, Un have zero expectation and variance σ^2

Method of Least Squares: Choose a value for α and β such that $S(\alpha,\beta)=(\sum_{1}^{n}(y_i-\alpha-\beta x_i)^2)$ is minimal.

Regression

The observed value y_i corresponding to x_i and the value $\alpha+\beta x_i$ on the regression line $y = \alpha + \beta x$.



$$\sum_{1}^{n} (y_i - \alpha - \beta x_i)^2$$

Estimation

Method of Least Squares: Choose a value for α and β such that $S(\alpha,\beta)=(\sum_{1}^{n}(y_i-\alpha-\beta x_i)^2)$ is minimal.

To find the *least squares estimates,* we differentiate *S(α, β)* with respect to α and β, and we set the derivatives equal to 0:

$$\frac{\partial}{\partial\alpha}S(\alpha,\beta)=0 \quad \Leftrightarrow \quad \sum_{i=1}^{n}(y_i-\alpha-\beta x_i)=0$$

$$\frac{\partial}{\partial\beta}S(\alpha,\beta)=0 \quad \Leftrightarrow \quad \sum_{i=1}^{n}(y_i-\alpha-\beta x_i)\,x_i=0.$$

- After some calculus magic, we get two equations to estimate α and β:

$$n\alpha+\beta\sum_{i=1}^{n}x_i=\sum_{i=1}^{n}y_i$$

$$\alpha\sum_{i=1}^{n}x_i+\beta\sum_{i=1}^{n}x_i^2=\sum_{i=1}^{n}x_iy_i.$$

Estimation

$$n\alpha + \beta\sum_{i=1}^{n} x_i = \sum_{i=1}^{n} y_i$$

$$\alpha\sum_{i=1}^{n} x_i + \beta\sum_{i=1}^{n} x_i^2 = \sum_{i=1}^{n} x_i y_i.$$

- After some simple algebraic rearranging, we obtain:

$$\hat{\beta} = \frac{n\sum x_i y_i - (\sum x_i)(\sum y_i)}{n\sum x_i^2 - (\sum x_i)^2}$$ (slope)

$$\hat{\alpha} = \bar{y}_n - \hat{\beta}\bar{x}_n.$$

(intercept)

Example

Find the least-squares solutions of $Ax = b$ where:

$$A = \begin{pmatrix} 0 & 1 \\ 1 & 1 \\ 2 & 1 \end{pmatrix} \qquad b = \begin{pmatrix} 6 \\ 0 \\ 0 \end{pmatrix}.$$

What quantity is being minimized?

Solution

We have

$$A^T A = \begin{pmatrix} 0 & 1 & 2 \\ 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & 1 \\ 2 & 1 \end{pmatrix} = \begin{pmatrix} 5 & 3 \\ 3 & 3 \end{pmatrix}$$

and

$$A^T b = \begin{pmatrix} 0 & 1 & 2 \\ 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} 6 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 6 \end{pmatrix}.$$

We form an augmented matrix and row reduce:

$$\left(\begin{array}{cc|c} 5 & 3 & 0 \\ 3 & 3 & 6 \end{array}\right) \xrightarrow{\text{RREF}} \left(\begin{array}{cc|c} 1 & 0 & -3 \\ 0 & 1 & 5 \end{array}\right).$$

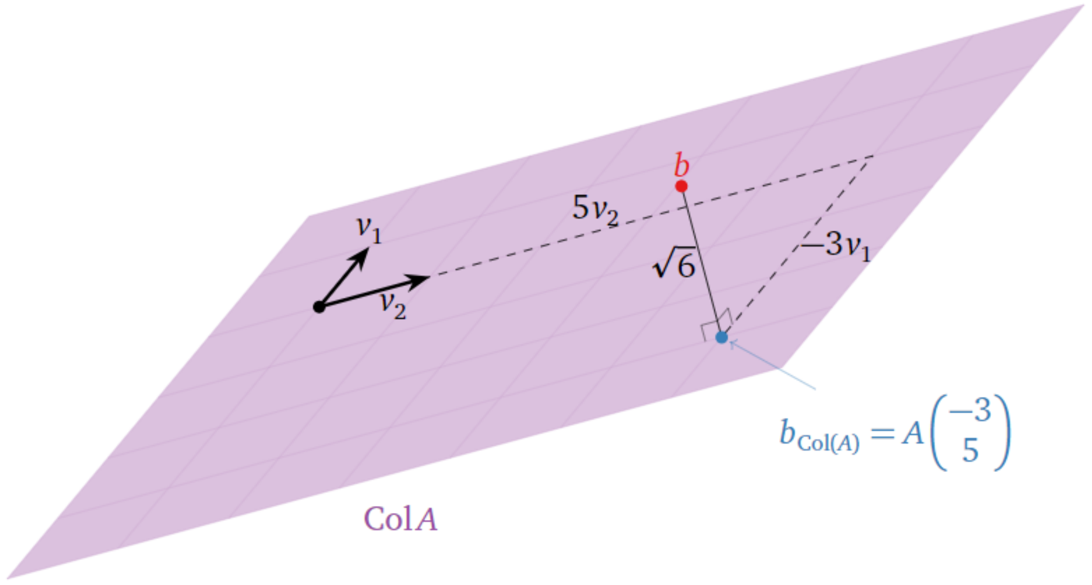
Therefore, the only least-squares solution is $\widehat{x} = \begin{pmatrix} -3 \\ 5 \end{pmatrix}$.

This solution minimizes the distance from $A\widehat{x}$ to b, i.e., the sum of the square of the entries of $b - A\widehat{x} = b - b_{\text{Col}(A)} = b_{\text{Col}(A)^\perp}$. In this case, we have

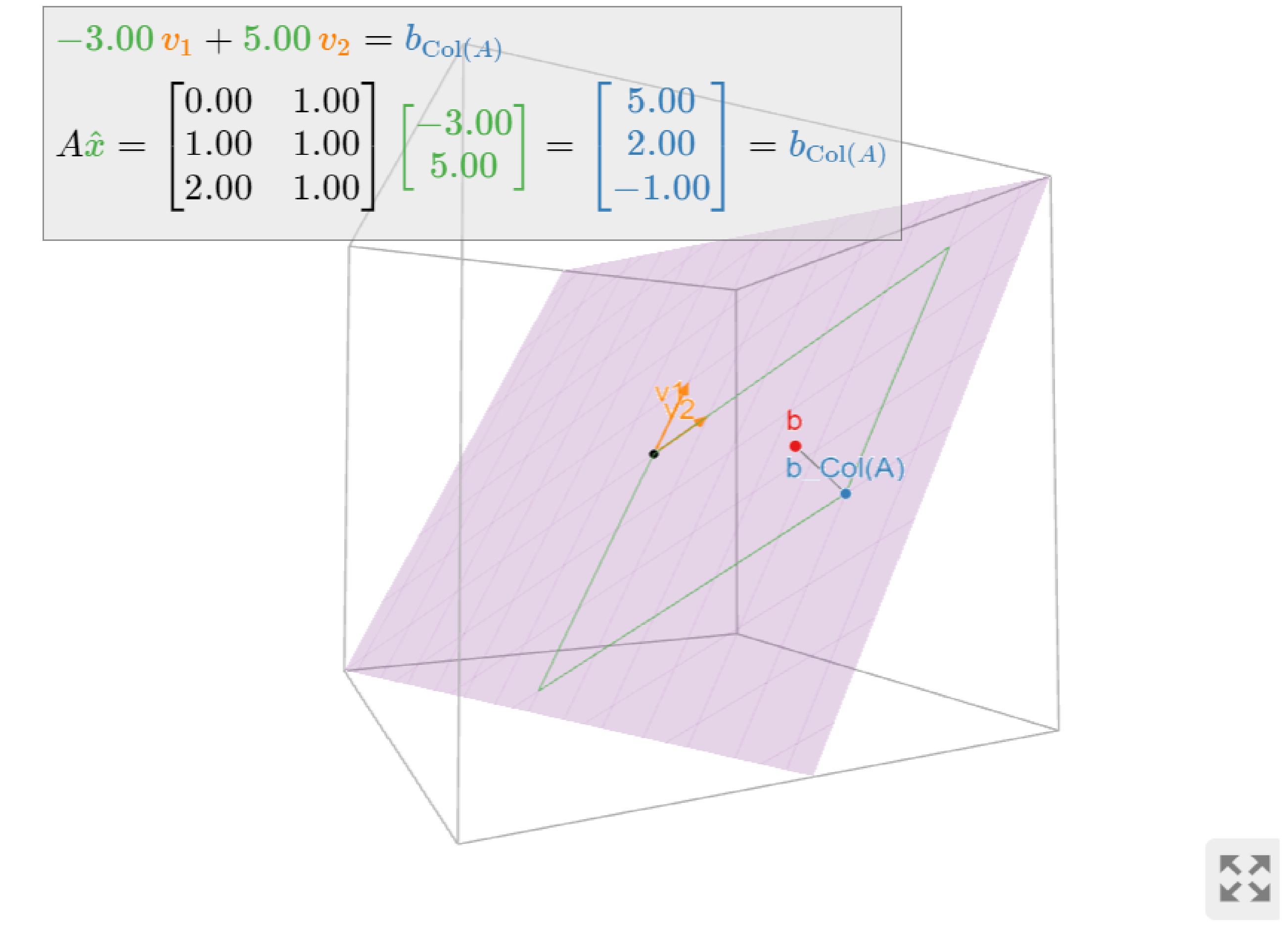
$$\|b - A\widehat{x}\| = \left\| \begin{pmatrix} 6 \\ 0 \\ 0 \end{pmatrix} - \begin{pmatrix} 5 \\ 2 \\ -1 \end{pmatrix} \right\| = \left\| \begin{pmatrix} 1 \\ -2 \\ 1 \end{pmatrix} \right\| = \sqrt{1^2 + (-2)^2 + 1^2} = \sqrt{6}.$$

Therefore, $b_{\text{Col}(A)} = A\widehat{x}$ is $\sqrt{6}$ units from b.

In the following picture, v_1, v_2 are the columns of A:



b
v_1
v_2
$5v_2$
$\sqrt{6}$
$-3v_1$
$b_{\text{Col}(A)} = A\begin{pmatrix} -3 \\ 5 \end{pmatrix}$
Col A

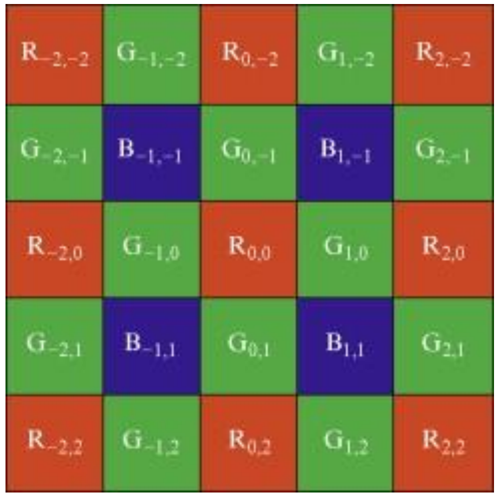

$$-3.00\,v_1 + 5.00\,v_2 = b_{\mathrm{Col}(A)}$$

$$A\hat{x} = \begin{bmatrix} 0.00 & 1.00 \\ 1.00 & 1.00 \\ 2.00 & 1.00 \end{bmatrix} \begin{bmatrix} -3.00 \\ 5.00 \end{bmatrix} = \begin{bmatrix} 5.00 \\ 2.00 \\ -1.00 \end{bmatrix} = b_{\mathrm{Col}(A)}$$

The violet plane is Col(A). *The closest that* Ax *can get to* b *is the closest vector on* Col(A) *to* b, *which is the orthogonal projection* $b_{\mathrm{Col}(A)}$ *(in blue). The vectors* v_1, v_2 *are the columns of* A, *and the coefficients of* $\widehat{x}$ *are the* $\mathcal{B}$*-coordinates of* $b_{\mathrm{Col}(A)}$, *where* $\mathcal{B} = \{v_1, v_2\}$.

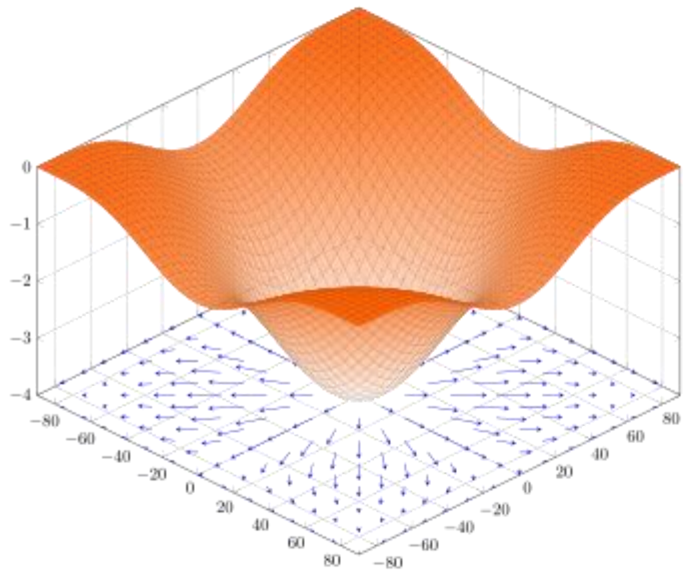
Gradient Computation

- In mathematical optimization, the gradient is a vector that points in the direction of the steepest increase of a function. Computation of the gradient involves finding the partial derivatives of the function with respect to each of its variables. The gradient is used in various optimization algorithms to find the minimum or maximum of a function.
- Gradient computation is a general solution to edge direction selection.
- Hibbard's method (1995) uses horizontal and vertical gradients, computed at each pixel where the G component must be estimated, in order to select the direction that provides the best green level estimation.

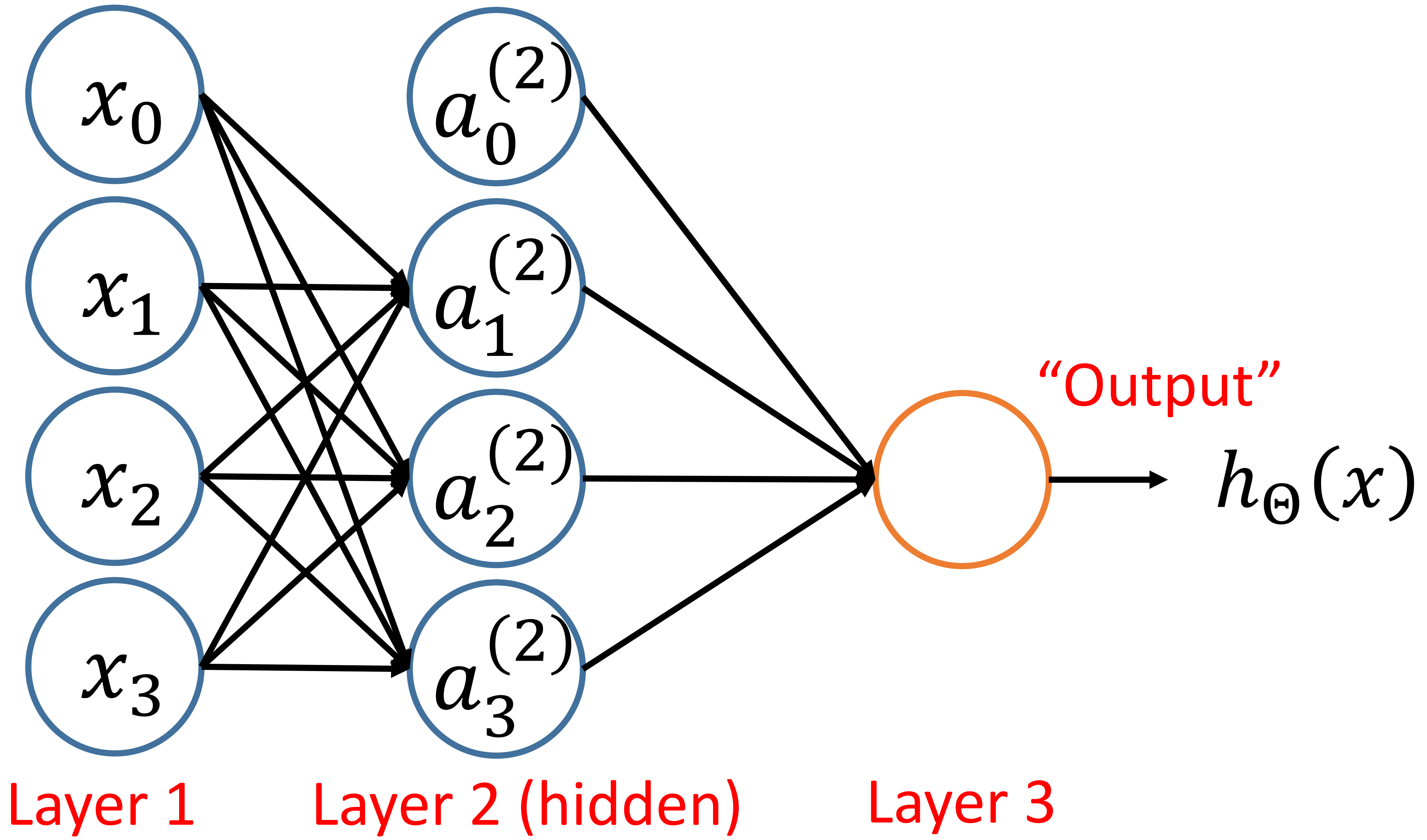

$R_{-2,-2}$ $G_{-1,-2}$ $R_{0,-2}$ $G_{1,-2}$ $R_{2,-2}$
$G_{-2,-1}$ $B_{-1,-1}$ $G_{0,-1}$ $B_{1,-1}$ $G_{2,-1}$
$R_{-2,0}$ $G_{-1,0}$ $R_{0,0}$ $G_{1,0}$ $R_{2,0}$
$G_{-2,1}$ $B_{-1,1}$ $G_{0,1}$ $B_{1,1}$ $G_{2,1}$
$R_{-2,2}$ $G_{-1,2}$ $R_{0,2}$ $G_{1,2}$ $R_{2,2}$

What is the purpose of gradient?

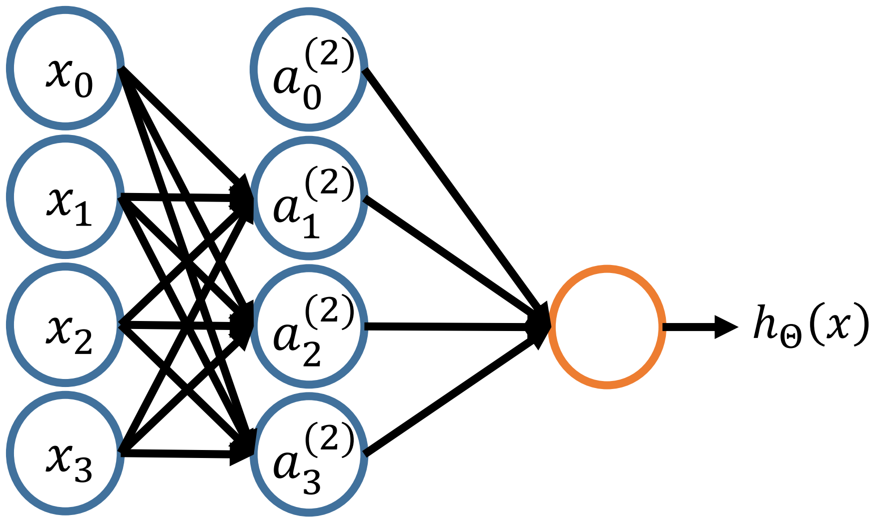
- The steepness of the slope at that point is given by the magnitude of the gradient vector.
- The gradient can also be used to measure how a scalar field changes in other directions, rather than just the direction of greatest change, by taking a dot product.



Neural network – Multilayer



Neural network



$a_i^{(j)}$ = "activation" of unit i in layer j
$\Theta^{(j)}$ = matrix of weights controlling function mapping from layer j to layer $j+1$

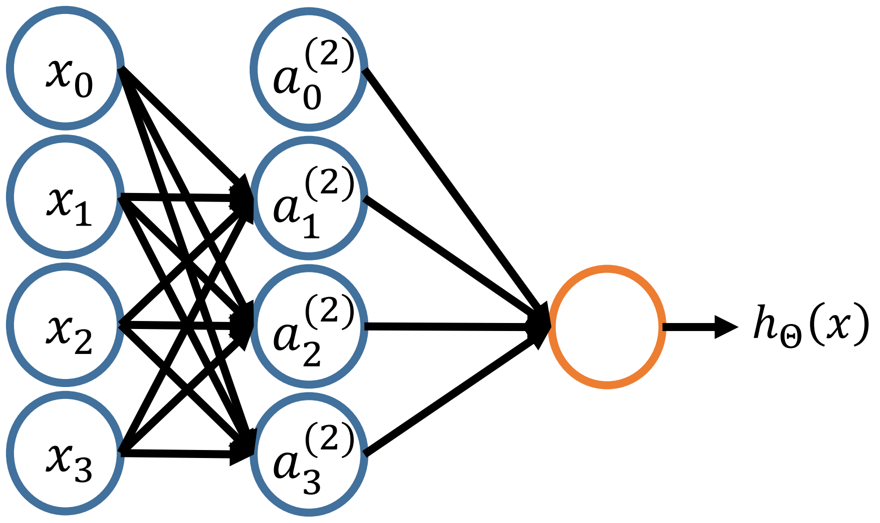
s_j unit in layer j
s_{j+1} units in layer $j+1$

$$a_1^{(2)} = g\left(\Theta_{10}^{(1)}x_0 + \Theta_{11}^{(1)}x_1 + \Theta_{12}^{(1)}x_2 + \Theta_{13}^{(1)}x_3\right)$$
$$a_2^{(2)} = g\left(\Theta_{20}^{(1)}x_0 + \Theta_{21}^{(1)}x_1 + \Theta_{22}^{(1)}x_2 + \Theta_{23}^{(1)}x_3\right)$$
$$a_3^{(2)} = g\left(\Theta_{30}^{(1)}x_0 + \Theta_{31}^{(1)}x_1 + \Theta_{32}^{(1)}x_2 + \Theta_{33}^{(1)}x_3\right)$$
$$h_\Theta(x) = g\left(\Theta_{10}^{(2)}a_0^{(2)} + \Theta_{11}^{(1)}a_1^{(2)} + \Theta_{12}^{(1)}a_2^{(2)} + \Theta_{13}^{(1)}a_3^{(2)}\right)$$

Size of $\Theta^{(j)}$?

$s_{j+1} \times (s_j + 1)$

Neural network



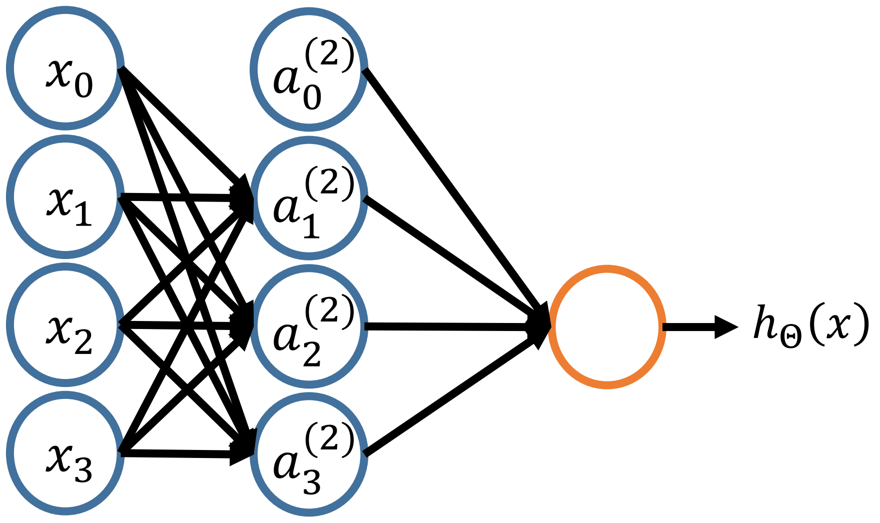
$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \qquad z^{(2)} = \begin{bmatrix} z_1^{(2)} \\ z_2^{(2)} \\ z_3^{(2)} \end{bmatrix}$$

"Pre-activation"

Why do we need g(.)?

$$a_1^{(2)} = g\left(\Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2 + \Theta_{13}^{(1)} x_3\right) = g(z_1^{(2)})$$
$$a_2^{(2)} = g\left(\Theta_{20}^{(1)} x_0 + \Theta_{21}^{(1)} x_1 + \Theta_{22}^{(1)} x_2 + \Theta_{23}^{(1)} x_3\right) = g(z_2^{(2)})$$
$$a_3^{(2)} = g\left(\Theta_{30}^{(1)} x_0 + \Theta_{31}^{(1)} x_1 + \Theta_{32}^{(1)} x_2 + \Theta_{33}^{(1)} x_3\right) = g(z_3^{(2)})$$
$$h_\Theta(x) = g\left(\Theta_{10}^{(2)} a_0^{(2)} + \Theta_{11}^{(1)} a_1^{(2)} + \Theta_{12}^{(1)} a_2^{(2)} + \Theta_{13}^{(1)} a_3^{(2)}\right) = g(z^{(3)})$$

Neural network



"Pre-activation"

$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \qquad z^{(2)} = \begin{bmatrix} z_1^{(2)} \\ z_2^{(2)} \\ z_3^{(2)} \end{bmatrix}$$

$$a_1^{(2)} = g(z_1^{(2)})$$
$$a_2^{(2)} = g(z_2^{(2)})$$
$$a_3^{(2)} = g(z_3^{(2)})$$
$$h_\Theta(x) = g(z^{(3)})$$

$$z^{(2)} = \Theta^{(1)}x = \Theta^{(1)}a^{(1)}$$
$$a^{(2)} = g(z^{(2)})$$
$$\text{Add } a_0^{(2)} = 1$$
$$z^{(3)} = \Theta^{(2)}a^{(2)}$$
$$h_\Theta(x) = a^{(3)} = g(z^{(3)})$$

Flow graph - Forward propagation

$$\begin{tikzcd}
X \arrow[rr] & & z^{(2)} \arrow[rr] & & a^{(2)} \arrow[rr] & & z^{(3)} \arrow[rr] & & a^{(3)} \arrow[r] & h_{\Theta}(x) \\
& W^{(1)} \arrow[ur] & & b^{(1)} \arrow[ul] & & W^{(2)} \arrow[ur] & & b^{(2)} \arrow[ul] & &
\end{tikzcd}$$

How do we evaluate our prediction?

$$z^{(2)} = \Theta^{(1)}x = \Theta^{(1)}a^{(1)}$$
$$a^{(2)} = g(z^{(2)})$$
$$\text{Add } a_0^{(2)} = 1$$
$$z^{(3)} = \Theta^{(2)}a^{(2)}$$
$$h_{\Theta}(x) = a^{(3)} = g(z^{(3)})$$

Cost function

Logistic regression:

$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1-y^{(i)})\log(1-h_\theta(x^{(i)}))\right] + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$$

Neural network:

$$h_\Theta(x) \in \mathbb{R}^K \quad (h_\Theta(x))_i = i^{th} \text{ output}$$

$$J(\Theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log(h_\Theta(x^{(i)}))_k + (1-y_k^{(i)})\log(1-(h_\Theta(x^{(i)}))_k)\right]$$
$$+\frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(\Theta_{ji}^{(l)})^2$$

Gradient computation

$$J(\Theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log(h_\Theta(x^{(i)}))_k + (1-y_k^{(i)})\log(1-(h_\Theta(x^{(i)}))_k)\right] + \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(\Theta_{ji}^{(l)})^2$$

$$\min_\Theta J(\Theta)$$

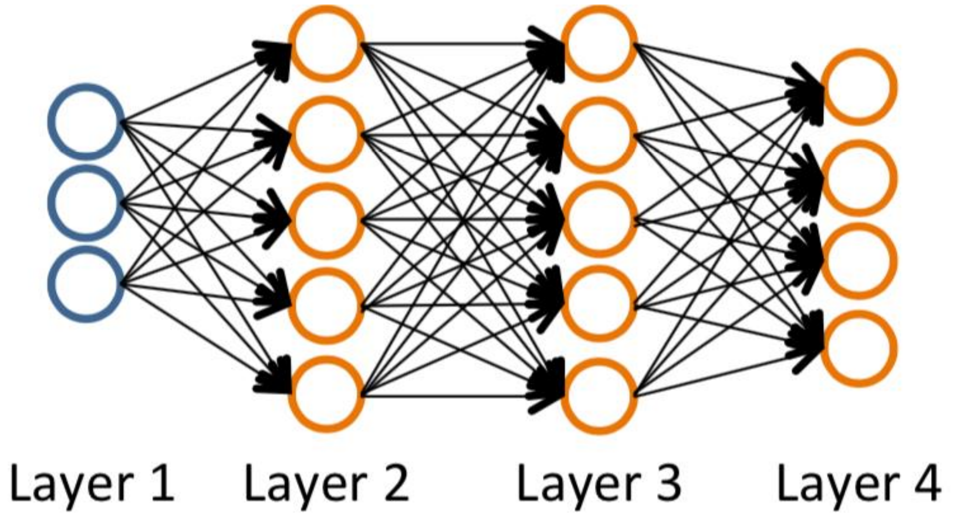
Need to compute:

$$\frac{\partial}{\partial\Theta_{ij}^{(l)}} J(\Theta)$$

Gradient computation

Given one training example (x, y)

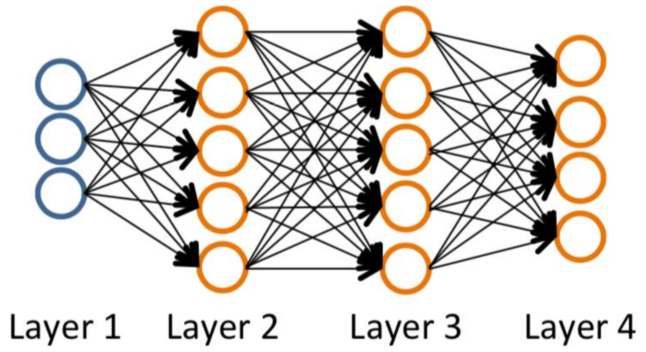
$$a^{(1)} = x$$
$$z^{(2)} = \Theta^{(1)} a^{(1)}$$
$$a^{(2)} = g(z^{(2)}) \text{ (add } \mathrm{a}_0^{(2)}\text{)}$$
$$z^{(3)} = \Theta^{(2)} a^{(2)}$$
$$a^{(3)} = g(z^{(3)}) \text{ (add } \mathrm{a}_0^{(3)}\text{)}$$
$$z^{(4)} = \Theta^{(3)} a^{(3)}$$
$$a^{(4)} = g\left(z^{(4)}\right) = h_\Theta(x)$$



Gradient computation: Backpropagation

Intuition: $\delta_j^{(l)}$ = "error" of node j in layer l

For each output unit (layer L = 4)



$$\delta^{(4)} = a^{(4)} - y$$

$$\delta^{(3)} = \delta^{(4)} \frac{\partial \delta^{(4)}}{\partial z^{(3)}} = \delta^{(4)} \frac{\partial \delta^{(4)}}{\partial a^{(4)}} \frac{\partial a^{(4)}}{\partial z^{(4)}} \frac{\partial z^{(4)}}{\partial a^{(3)}} \frac{\partial a^{(3)}}{\partial z^{(3)}}$$

$$= 1 * \Theta^{(3)^T} \delta^{(4)} .* \ g'\left(z^{(4)}\right) .* \ g'(z^{(3)})$$

$$z^{(3)} = \Theta^{(2)} a^{(2)}$$
$$a^{(3)} = g(z^{(3)})$$
$$z^{(4)} = \Theta^{(3)} a^{(3)}$$
$$a^{(4)} = g\left(z^{(4)}\right)$$

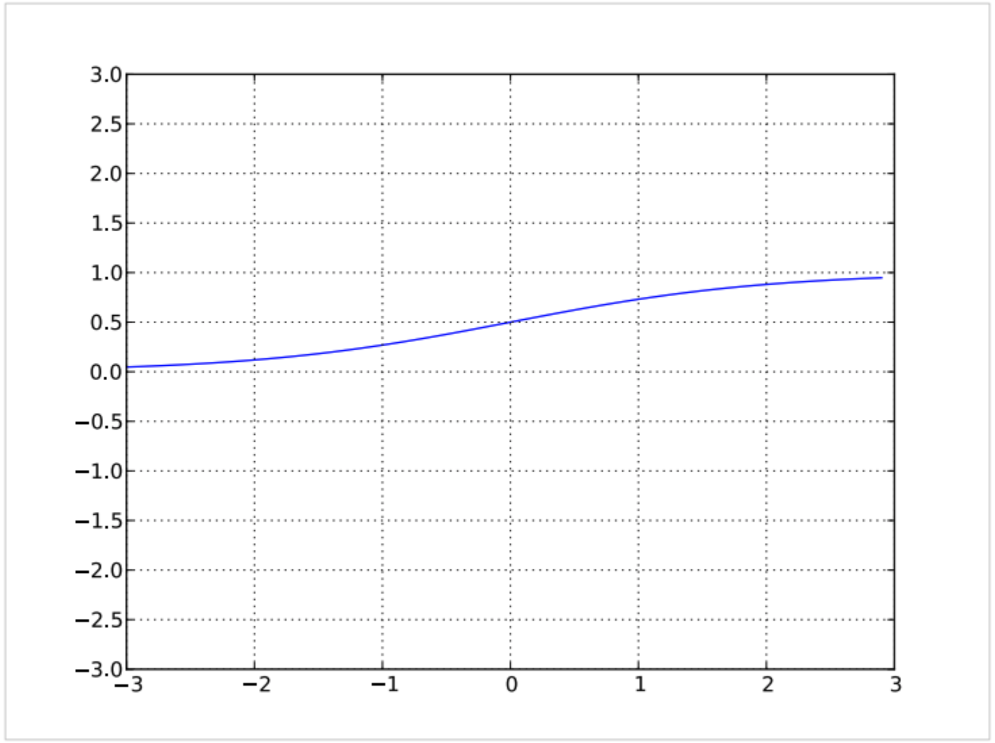
Backpropagation algorithm

```
Training set $\{(x^{(1)}, y^{(1)}) \dots (x^{(m)}, y^{(m)})\}$
Set $\Theta^{(1)} = 0$
For $i = 1$ to $m$
        Set $a^{(1)} = x$
        Perform forward propagation to compute $a^{(l)}$ for $l = 2..L$
        use $y^{(i)}$ to compute $\delta_0^{(L)} = a^{(L)} - y^{(i)}$
        Compute $\delta_0^{(L-1)}, \delta_0^{(L-2)} \dots \delta_0^{(2)}$
        $\Theta^{(l)} = \Theta^{(l)} - a^{(l)}\delta_0^{(l+1)}$
```

Activation - sigmoid

- Partial derivative

$$g'(x) = g(x)\left(1 - g(x)\right)$$

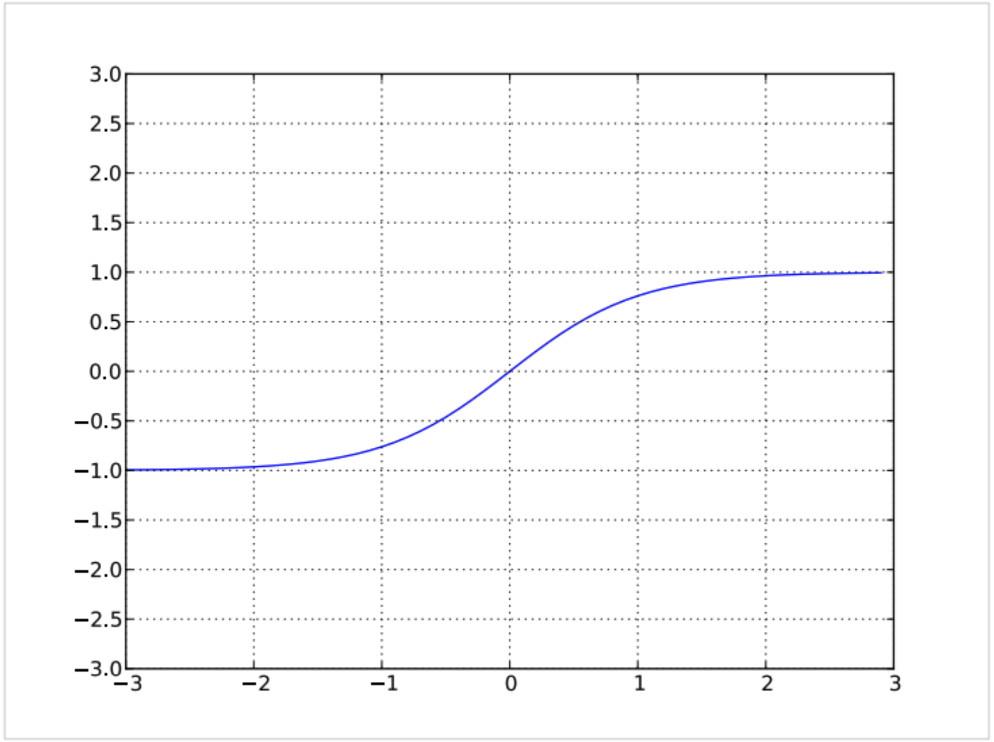


$$g(x) = \frac{1}{1 + e^{-x}}$$

Activation - hyperbolic tangent (tanh)

- Partial derivative

$$g'(x) = 1 - g(x)^2$$

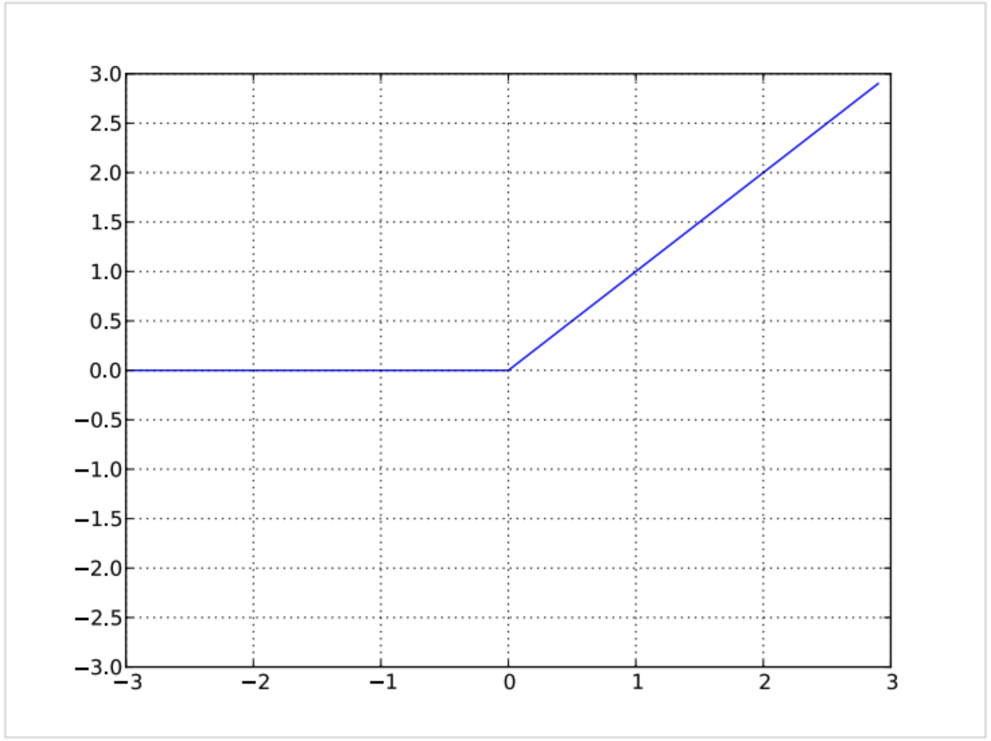


$$g(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

Activation - rectified linear(relu)

- Partial derivative

$$g'(x) = 1_{x>0}$$



$$g(x) = \mathrm{relu}(x) = \max(0, x)$$

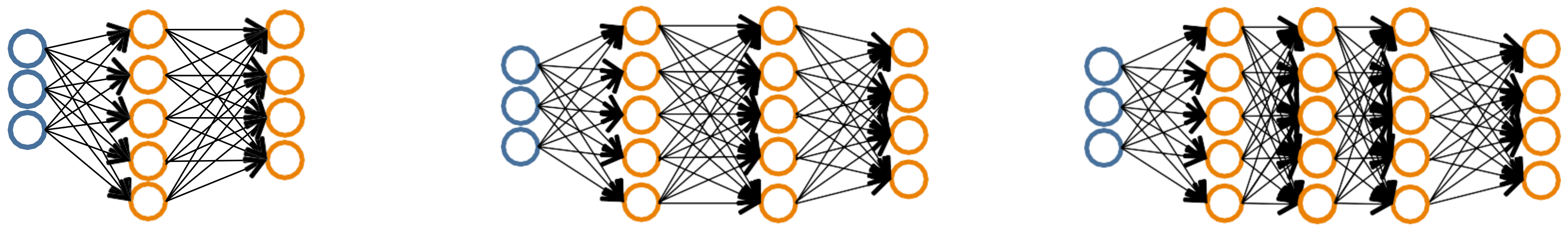
Slide credit: Hugo Larochelle

Initialization

- For bias
  - Initialize all to 0
- For weights
  - Can’t initialize all weights to the same value
    - we can show that all hidden units in a layer will always behave the same
    - need to break symmetry
  - Recipe: U[-b, b]
    - the idea is to sample around 0 but break symmetry

Putting it together

Pick a network architecture

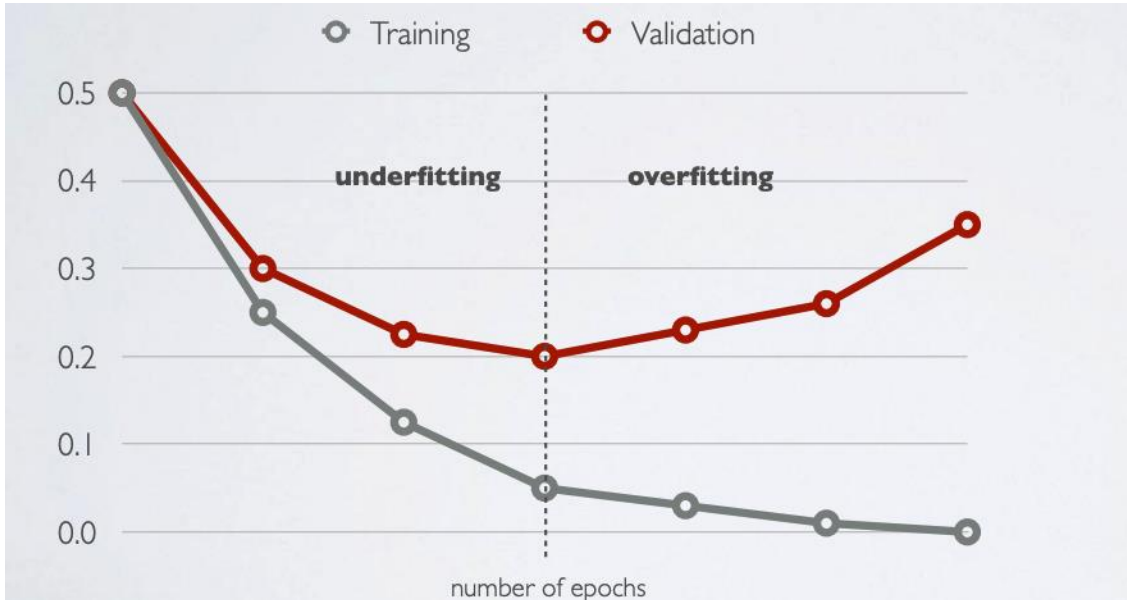


- No. of input units: Dimension of features
- No. output units: Number of classes
- Reasonable default: 1 hidden layer, or if >1 hidden layer, have same no. of hidden units in every layer (usually the more the better)
- Grid search

Putting it together

Early stopping

- Use a validation set performance to select the best configuration
- To select the number of epochs, stop training when validation set error increases



Gradient Descending

- Gradient descent works by iteratively adjusting the parameters of a model in the direction that reduces the loss function. This is done by taking the derivative (gradient) of the loss function with respect to the parameters. The gradient points in the direction of steepest ascent, so by moving in the opposite direction, we aim to reach a minimum point.
- There are three main variants of gradient descent:
- **Batch Gradient Descent (BGD):** It computes the gradient of the entire dataset before making a parameter update. This can be computationally expensive for large datasets.
- **Stochastic Gradient Descent (SGD):** It updates the parameters after each individual training example. While this is faster, it can be more noisy and may not converge as smoothly as BGD.
- **Mini-batch Gradient Descent:** This is a compromise between BGD and SGD. It computes the gradient on a small batch of data, striking a balance between computational efficiency and smoother convergence.

Application in AI

- Gradient descent is used extensively in training machine learning models, especially in deep learning. Let's take an example of training a simple linear regression model:

- **Example: Linear Regression**
- In linear regression, you have a set of data points (x, y) and you're trying to find the best-fit line (y = mx + b) that minimizes the error between the predicted (ŷ) and actual (y) values.

- **Initialize Parameters:** Randomly initialize the slope (m) and intercept (b).
- **Compute Predictions:** Use the current values of m and b to make predictions.
- **Compute Loss:** Calculate the loss using a suitable loss function (e.g., Mean Squared Error).
- **Compute Gradients:** Find the partial derivatives of the loss with respect to m and b.
- **Update Parameters:** Adjust m and b in the opposite direction of the gradients (scaled by a learning rate).
- **Repeat:** Keep repeating steps 3-5 until the loss converges to a minimum.

Tikhonov Regularization

- Remember the intuition: complicated hypotheses lead to overfitting
- Idea: change the error function to *penalize hypothesis complexity*:

$$J(\mathbf{w}) = J_D(\mathbf{w}) + \lambda J_{pen}(\mathbf{w})$$

  This is called *regularization* in machine learning and *shrinkage* in statistics
- λ is called *regularization coefficient* and controls how much we value fitting the data well, vs. a simple hypothesis

Tikhonov Regularization

- A squared penalty on the weights would make the math work nicely in our case:

$$\frac{1}{2}(\mathbf{\Phi w} - \mathbf{y})^T(\mathbf{\Phi w} - \mathbf{y}) + \frac{\lambda}{2}\mathbf{w}^T\mathbf{w}$$

- This is also known as L_2 *regularization*, or *weight decay* in neural networks
- By re-grouping terms, we get:

$$J_D(\mathbf{w}) = \frac{1}{2}(\mathbf{w}^T(\mathbf{\Phi}^T\mathbf{\Phi} + \lambda\mathbf{I})\mathbf{w} - \mathbf{w}^T\mathbf{\Phi}^T\mathbf{y} - \mathbf{y}^T\mathbf{\Phi w} + \mathbf{y}^T\mathbf{y})$$

- Optimal solution (obtained by solving $\nabla_{\mathbf{w}} J_D(\mathbf{w}) = 0$)

$$\mathbf{w} = (\mathbf{\Phi}^T\mathbf{\Phi} + \lambda I)^{-1}\mathbf{\Phi}^T\mathbf{y}$$

What L_2 regularization does

$$\arg\min_{\mathbf{w}} \frac{1}{2}(\mathbf{\Phi w} - \mathbf{y})^T(\mathbf{\Phi w} - \mathbf{y}) + \frac{\lambda}{2}\mathbf{w}^T\mathbf{w} = (\mathbf{\Phi}^T\mathbf{\Phi} + \lambda I)^{-1}\mathbf{\Phi}^T\mathbf{y}$$

- If $\lambda = 0$, the solution is the same as in regular least-squares linear regression
- If $\lambda \to \infty$, the solution $\mathbf{w} \to 0$
- Positive λ will cause the magnitude of the weights to be smaller than in the usual linear solution
- This is also called *ridge regression*, and it is a special case of Tikhonov regularization (more on that later)
- A different view of regularization: we want to optimize the error while keeping the L_2 norm of the weights, $\mathbf{w}^T\mathbf{w}$, bounded.

Gauss-Seidel Method

An *iterative* method.

Basic Procedure:

-Algebraically solve each linear equation for x_i

-Assume an initial guess solution array

-Solve for each x_i and repeat

-Use absolute relative approximate error after each iteration to check if error is within a pre-specified tolerance.

Gauss-Seidel Method

Why?

The Gauss-Seidel Method allows the user to control round-off error.

Elimination methods such as Gaussian Elimination and LU Decomposition are prone to prone to round-off error.

Also: If the physics of the problem are understood, a close initial guess can be made, decreasing the number of iterations needed.

Gauss-Seidel Method

Algorithm

A set of *n* equations and *n* unknowns:

$$a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \ldots + a_{1n}x_n = b_1$$

$$a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + \ldots + a_{2n}x_n = b_2$$

$$\vdots \qquad \vdots$$

$$a_{n1}x_1 + a_{n2}x_2 + a_{n3}x_3 + \ldots + a_{nn}x_n = b_n$$

If: the diagonal elements are non-zero

Rewrite each equation solving for the corresponding unknown

ex:

First equation, solve for x_1

Second equation, solve for x_2

Gauss-Seidel Method

Algorithm

Rewriting each equation

$$x_1 = \frac{c_1 - a_{12}x_2 - a_{13}x_3 \ldots\ldots - a_{1n}x_n}{a_{11}}$$

From Equation 1

$$x_2 = \frac{c_2 - a_{21}x_1 - a_{23}x_3 \ldots\ldots - a_{2n}x_n}{a_{22}}$$

From equation 2

$$\vdots \qquad \vdots \qquad \vdots$$

$$x_{n-1} = \frac{c_{n-1} - a_{n-1,1}x_1 - a_{n-1,2}x_2 \ldots\ldots - a_{n-1,n-2}x_{n-2} - a_{n-1,n}x_n}{a_{n-1,n-1}}$$

From equation n-1

$$x_n = \frac{c_n - a_{n1}x_1 - a_{n2}x_2 - \ldots\ldots - a_{n,n-1}x_{n-1}}{a_{nn}}$$

From equation n

Gauss-Seidel Method

Algorithm

General Form of each equation

$$x_1 = \frac{c_1 - \sum_{\substack{j=1 \\ j\neq 1}}^{n} a_{1j} x_j}{a_{11}}$$

$$x_{n-1} = \frac{c_{n-1} - \sum_{\substack{j=1 \\ j\neq n-1}}^{n} a_{n-1,j} x_j}{a_{n-1,n-1}}$$

$$x_2 = \frac{c_2 - \sum_{\substack{j=1 \\ j\neq 2}}^{n} a_{2j} x_j}{a_{22}}$$

$$x_n = \frac{c_n - \sum_{\substack{j=1 \\ j\neq n}}^{n} a_{nj} x_j}{a_{nn}}$$

Gauss-Seidel Method

Algorithm

General Form for any row ‘i’

$$x_i = \frac{c_i - \sum_{\substack{j=1 \\ j \neq i}}^{n} a_{ij} x_j}{a_{ii}}, i = 1, 2, \ldots, n.$$

How or where can this equation be used?

Gauss-Seidel Method

Solve for the unknowns

Assume an initial guess for [X]

$$\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n-1} \\ x_n \end{bmatrix}$$

Use rewritten equations to solve for each value of x_i.

Important: Remember to use the most recent value of x_i. Which means to apply values calculated to the calculations remaining in the **current** iteration.

Gauss-Seidel Method

Calculate the Absolute Relative Approximate Error

$$\left|\varepsilon_a\right|_i = \left|\frac{x_i^{new} - x_i^{old}}{x_i^{new}}\right| \times 100$$

So when has the answer been found?

The iterations are stopped when the absolute relative approximate error is less than a prespecified tolerance for all unknowns.

Gauss-Seidel Method: Example 2

Given the system of equations

$$12x_1 + 3x_2 - 5x_3 = 1$$

$$x_1 + 5x_2 + 3x_3 = 28$$

$$3x_1 + 7x_2 + 13x_3 = 76$$

With an initial guess of

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}$$

The coefficient matrix is:

$$[A] = \begin{bmatrix} 12 & 3 & -5 \\ 1 & 5 & 3 \\ 3 & 7 & 13 \end{bmatrix}$$

Will the solution converge using the Gauss-Siedel method?

Gauss-Seidel Method: Example 2

Checking if the coefficient matrix is diagonally dominant

$$[A]=\begin{bmatrix} 12 & 3 & -5 \\ 1 & 5 & 3 \\ 3 & 7 & 13 \end{bmatrix}$$

$$|a_{11}| = |12| = 12 \geq |a_{12}| + |a_{13}| = |3| + |-5| = 8$$

$$|a_{22}| = |5| = 5 \geq |a_{21}| + |a_{23}| = |1| + |3| = 4$$

$$|a_{33}| = |13| = 13 \geq |a_{31}| + |a_{32}| = |3| + |7| = 10$$

The inequalities are all true and at least one row is *strictly* greater than:

Therefore: The solution should converge using the Gauss-Siedel Method

Gauss-Seidel Method: Example 2

Rewriting each equation

$$\begin{bmatrix} 12 & 3 & -5 \\ 1 & 5 & 3 \\ 3 & 7 & 13 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 28 \\ 76 \end{bmatrix}$$

$$x_1 = \frac{1 - 3x_2 + 5x_3}{12}$$

$$x_2 = \frac{28 - x_1 - 3x_3}{5}$$

$$x_3 = \frac{76 - 3x_1 - 7x_2}{13}$$

With an initial guess of

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}$$

$$x_1 = \frac{1 - 3(0) + 5(1)}{12} = 0.50000$$

$$x_2 = \frac{28 - (0.5) - 3(1)}{5} = 4.9000$$

$$x_3 = \frac{76 - 3(0.50000) - 7(4.9000)}{13} = 3.0923$$

Gauss-Seidel Method: Example 2

The absolute relative approximate error

$$\left|\in_a\right|_1 = \left|\frac{0.50000-1.0000}{0.50000}\right|\times 100 = 67.662\%$$

$$\left|\in_a\right|_2 = \left|\frac{4.9000-0}{4.9000}\right|\times 100 = 100.00\%$$

$$\left|\in_a\right|_3 = \left|\frac{3.0923-1.0000}{3.0923}\right|\times 100 = 67.662\%$$

The maximum absolute relative error after the first iteration is 100%

Gauss-Seidel Method: Example 2

After Iteration #1

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 0.5000 \\ 4.9000 \\ 3.0923 \end{bmatrix}$$

Substituting the x values into the equations

$$x_1 = \frac{1 - 3(4.9000) + 5(3.0923)}{12} = 0.14679$$

$$x_2 = \frac{28 - (0.14679) - 3(3.0923)}{5} = 3.7153$$

$$x_3 = \frac{76 - 3(0.14679) - 7(4.900)}{13} = 3.8118$$

After Iteration #2

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 0.14679 \\ 3.7153 \\ 3.8118 \end{bmatrix}$$

Gauss-Seidel Method: Example 2

Iteration #2 absolute relative approximate error

$$\left|\in_a\right|_1 = \left|\frac{0.14679-0.50000}{0.14679}\right| \times 100 = 240.62\%$$

$$\left|\in_a\right|_2 = \left|\frac{3.7153-4.9000}{3.7153}\right| \times 100 = 31.887\%$$

$$\left|\in_a\right|_3 = \left|\frac{3.8118-3.0923}{3.8118}\right| \times 100 = 18.876\%$$

The maximum absolute relative error after the first iteration is 240.62%

This is much larger than the maximum absolute relative error obtained in iteration #1. Is this a problem?

Gauss-Seidel Method: Example 2

Repeating more iterations, the following values are obtained

Iteration	a_1	$\lvert\varepsilon_a\rvert_1$	a_2	$\lvert\varepsilon_a\rvert_2$	a_3	$\lvert\varepsilon_a\rvert_3$
1	0.50000	67.662	4.900	100.00	3.0923	67.662
2	0.14679	240.62	3.7153	31.887	3.8118	18.876
3	0.74275	80.23	3.1644	17.409	3.9708	4.0042
4	0.94675	21.547	3.0281	4.5012	3.9971	0.65798
5	0.99177	4.5394	3.0034	0.82240	4.0001	0.07499
6	0.99919	0.74260	3.0001	0.11000	4.0001	0.00000

The solution obtained $\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 0.99919 \\ 3.0001 \\ 4.0001 \end{bmatrix}$

is close to the exact solution of $\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 3 \\ 4 \end{bmatrix}$

Application: Gradient Explosion

- **Recurrent Neural Networks (RNNs) in Natural Language Processing (NLP)**
  - **Scenario**: When training RNNs on long sequences of text data, especially with deep architectures, the gradients can grow exponentially. This leads to numerical instability and makes learning difficult.
  - **Solution**: Techniques like gradient clipping can be used to bound the gradients and prevent them from exploding.
- **Training Very Deep Neural Networks**
  - **Scenario**: In deep neural networks, especially those with a large number of layers, the gradients can amplify as they propagate backward through the network.
  - **Solution**: Methods like weight initialization techniques (e.g., Xavier/Glorot initialization) and batch normalization can help mitigate gradient explosion.

Application: Gradient Vanishing

- **Deep Feedforward Neural Networks**
  - **Scenario**: In very deep feedforward networks (with many hidden layers), especially when using activation functions like sigmoid or tanh, the gradients can become extremely small as they propagate backward.
  - **Solution**: Activation functions like ReLU (Rectified Linear Unit) or Leaky ReLU can help mitigate the vanishing gradient problem by allowing a non-zero gradient for positive inputs.

- **Gradient-Based Optimization for Generative Adversarial Networks (GANs)**
- **Scenario**: In GANs, especially when training the generator network to produce realistic samples, the vanishing gradient problem can occur, leading to slow convergence or instability.
- **Solution**: Techniques like using appropriate activation functions (e.g., Leaky ReLU), careful weight initialization, and adjusting the learning rates for the generator and discriminator can help stabilize GAN training.

SRM
INSTITUTE OF SCIENCE & TECHNOLOGY
Deemed to be University u/s 3 of UGC Act, 1956

THANK YOU