

# **Machine Learning**

Instructor: Chandra Sekhar V

## **1. Introduction to Matrices**

- Definition and Notation of Matrices
- Types of Matrices (Identity, Diagonal, Zero, etc.)

## **2. Basic Matrix Operations**

- Addition and Subtraction of Matrices
- Scalar Multiplication
- Matrix Transposition

## **3. Advanced Matrix Operations**

- Matrix Multiplication
- Dot Product and Its Applications

## **4. Solving Systems of Linear Equations**

- Representing Linear Systems in Matrix Form
- Gaussian Elimination

## **6. Determinants**

- Finding Determinants of Matrices
- Cofactors and Minors

## 7. Matrix Inverses

- Definition of an Inverse Matrix
- Finding Inverses Using Row Reduction
- Finding Inverses Using the Adjoint Method

## 8. Linear Dependence and Independence

- Definitions and Testing for Dependence
- Applications in Machine Learning

## 9. Eigenvalues and Eigenvectors

- Definitions and Basic Properties
- Applications in PCA and Dimensionality Reduction

## 10. Applications

- Projections and Vector Transformations
- Finding Angles Between Vectors
- Applications in Machine Learning and Data Science (e.g., PCA, Regression)

## 11. Practice Problems

- Basic Matrix Calculations
- Solving Systems of Equations
- Real-world Applications (Optimization, Machine Learning)

# 1. Introduction to Matrices

## Definition and Notation

A 3x3 matrix is a square matrix with 3 rows and 3 columns:  $A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$

For example:  $A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$

## Types of Matrices

1. Identity Matrix:  $I = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

2. Diagonal Matrix:  $D = \begin{bmatrix} 3 & 0 & 0 \\ 0 & 7 & 0 \\ 0 & 0 & 2 \end{bmatrix}$

3. Zero Matrix:  $Z = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$

## Addition and Subtraction

- Addition:

$$A + B = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} + \begin{bmatrix} 9 & 8 & 7 \\ 6 & 5 & 4 \\ 3 & 2 & 1 \end{bmatrix} = \begin{bmatrix} 10 & 10 & 10 \\ 10 & 10 & 10 \\ 10 & 10 & 10 \end{bmatrix}$$

- Subtraction:

$$A - B = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} - \begin{bmatrix} 9 & 8 & 7 \\ 6 & 5 & 4 \\ 3 & 2 & 1 \end{bmatrix} = \begin{bmatrix} -8 & -6 & -4 \\ -2 & 0 & 2 \\ 4 & 6 & 8 \end{bmatrix}$$

## Scalar Multiplication

Multiply each element by a scalar  $k$ :

$$3 \cdot A = 3 \cdot \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} = \begin{bmatrix} 3 & 6 & 9 \\ 12 & 15 & 18 \\ 21 & 24 & 27 \end{bmatrix}$$

## Transpose

Flip rows and columns:

$$A^T = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} = \begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{bmatrix}$$

### 3. Advanced Matrix Operations

#### Matrix Multiplication

Multiply rows of the first matrix by columns of the second matrix:

$$A \cdot B = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \cdot \begin{bmatrix} 9 & 8 & 7 \\ 6 & 5 & 4 \\ 3 & 2 & 1 \end{bmatrix} = \begin{bmatrix} 30 & 24 & 18 \\ 84 & 69 & 54 \\ 138 & 114 & 90 \end{bmatrix}$$

#### Dot Product

For vectors  $u = [1, 2, 3]$  and  $v = [4, 5, 6]$ :

$$u \cdot v = 1 \cdot 4 + 2 \cdot 5 + 3 \cdot 6 = 32$$

#### Addition and Subtraction

- Addition:

$$A + B = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} + \begin{bmatrix} 9 & 8 & 7 \\ 6 & 5 & 4 \\ 3 & 2 & 1 \end{bmatrix} = \begin{bmatrix} 10 & 10 & 10 \\ 10 & 10 & 10 \\ 10 & 10 & 10 \end{bmatrix}$$

- Subtraction:

$$A - B = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} - \begin{bmatrix} 9 & 8 & 7 \\ 6 & 5 & 4 \\ 3 & 2 & 1 \end{bmatrix} = \begin{bmatrix} -8 & -6 & -4 \\ -2 & 0 & 2 \\ 4 & 6 & 8 \end{bmatrix}$$

## Scalar Multiplication

Multiply each element by a scalar  $k$ :

$$3 \cdot A = 3 \cdot \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} = \begin{bmatrix} 3 & 6 & 9 \\ 12 & 15 & 18 \\ 21 & 24 & 27 \end{bmatrix}$$

## Transpose

Flip rows and columns:

$$A^T = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} = \begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{bmatrix}$$



### 3. Advanced Matrix Operations

#### Matrix Multiplication

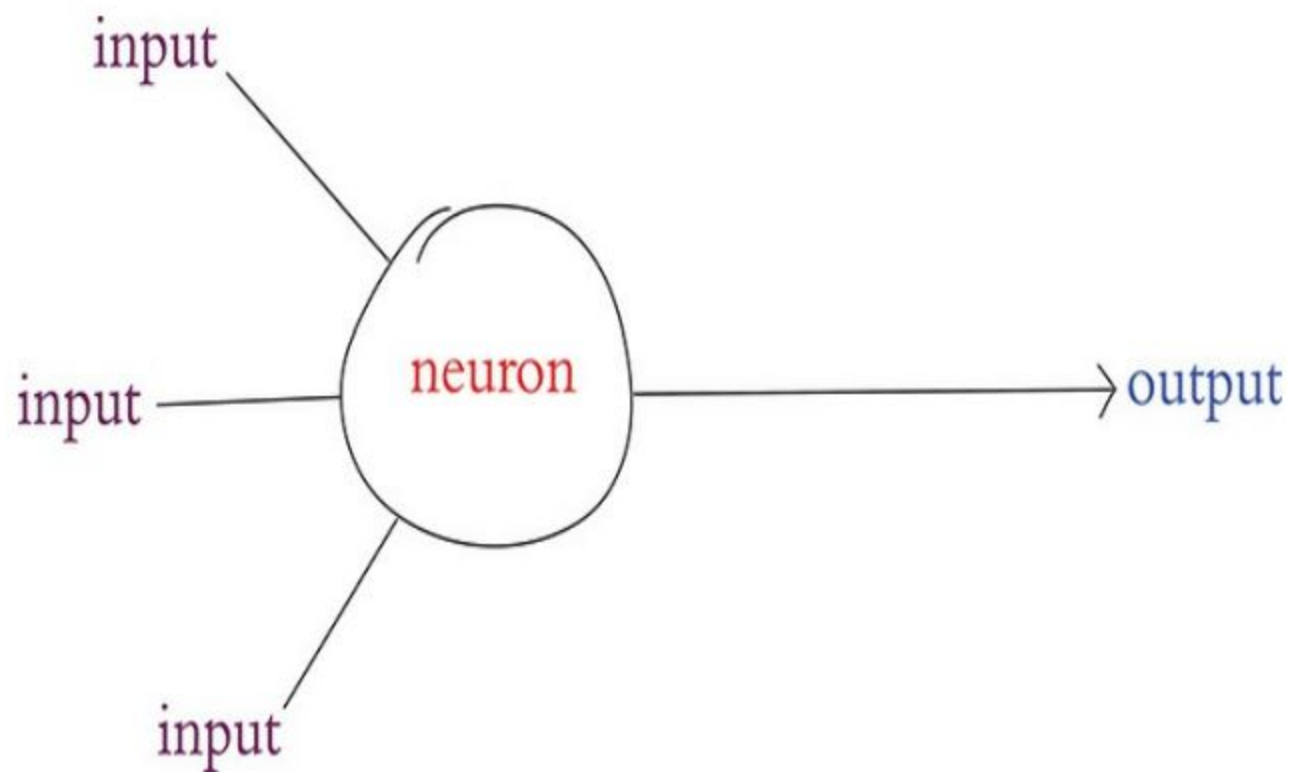
Multiply rows of the first matrix by columns of the second matrix:

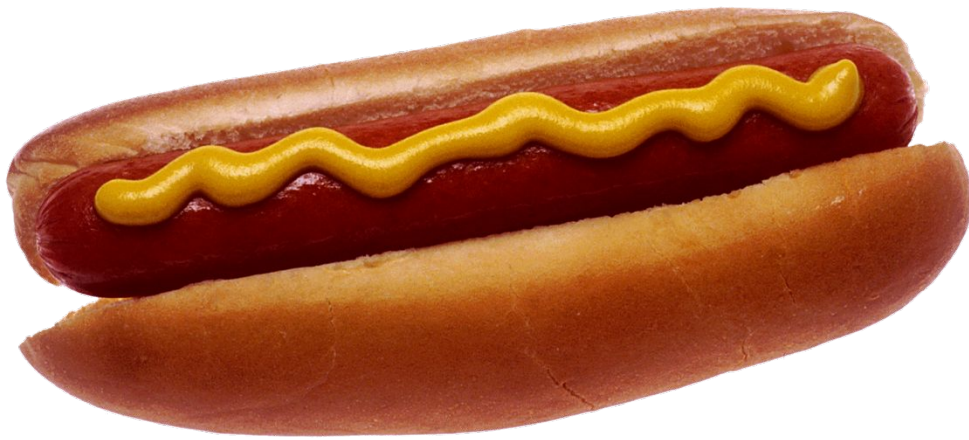
$$A \cdot B = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \cdot \begin{bmatrix} 9 & 8 & 7 \\ 6 & 5 & 4 \\ 3 & 2 & 1 \end{bmatrix} = \begin{bmatrix} 30 & 24 & 18 \\ 84 & 69 & 54 \\ 138 & 114 & 90 \end{bmatrix}$$

#### Dot Product

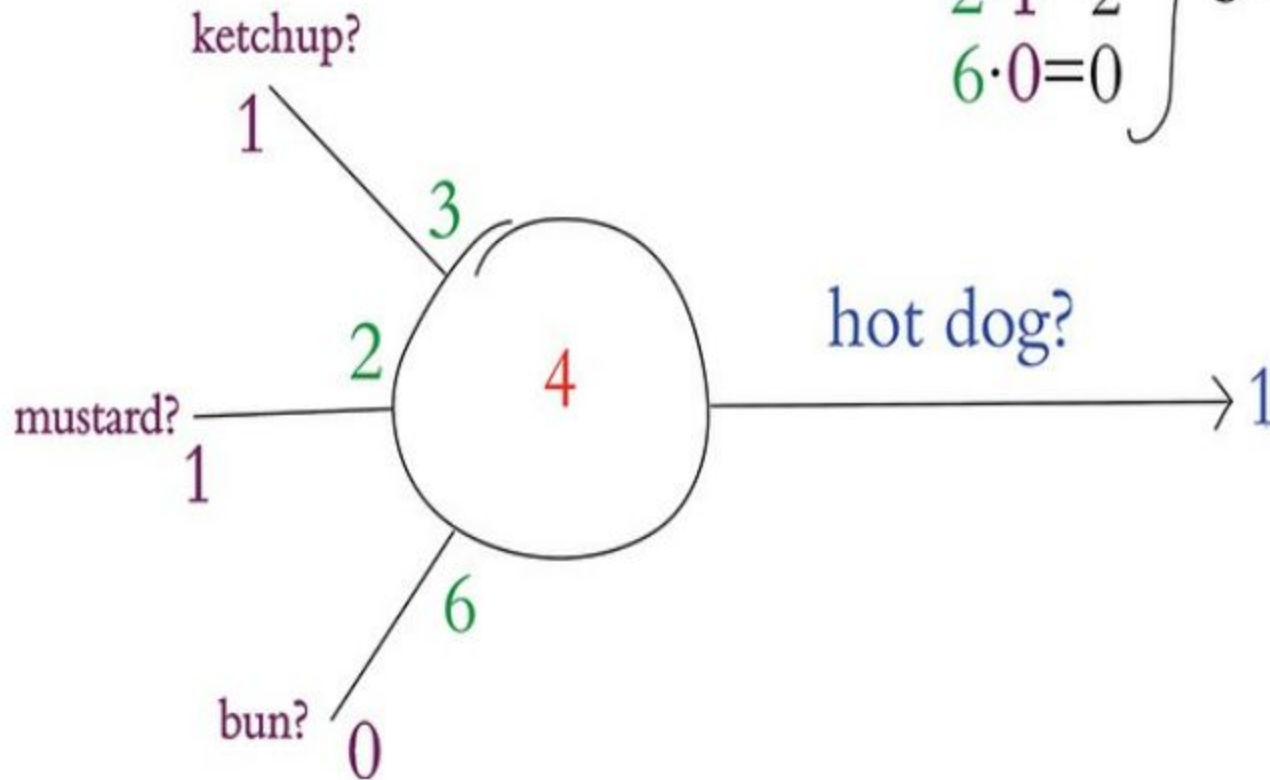
For vectors  $u = [1, 2, 3]$  and  $v = [4, 5, 6]$ :

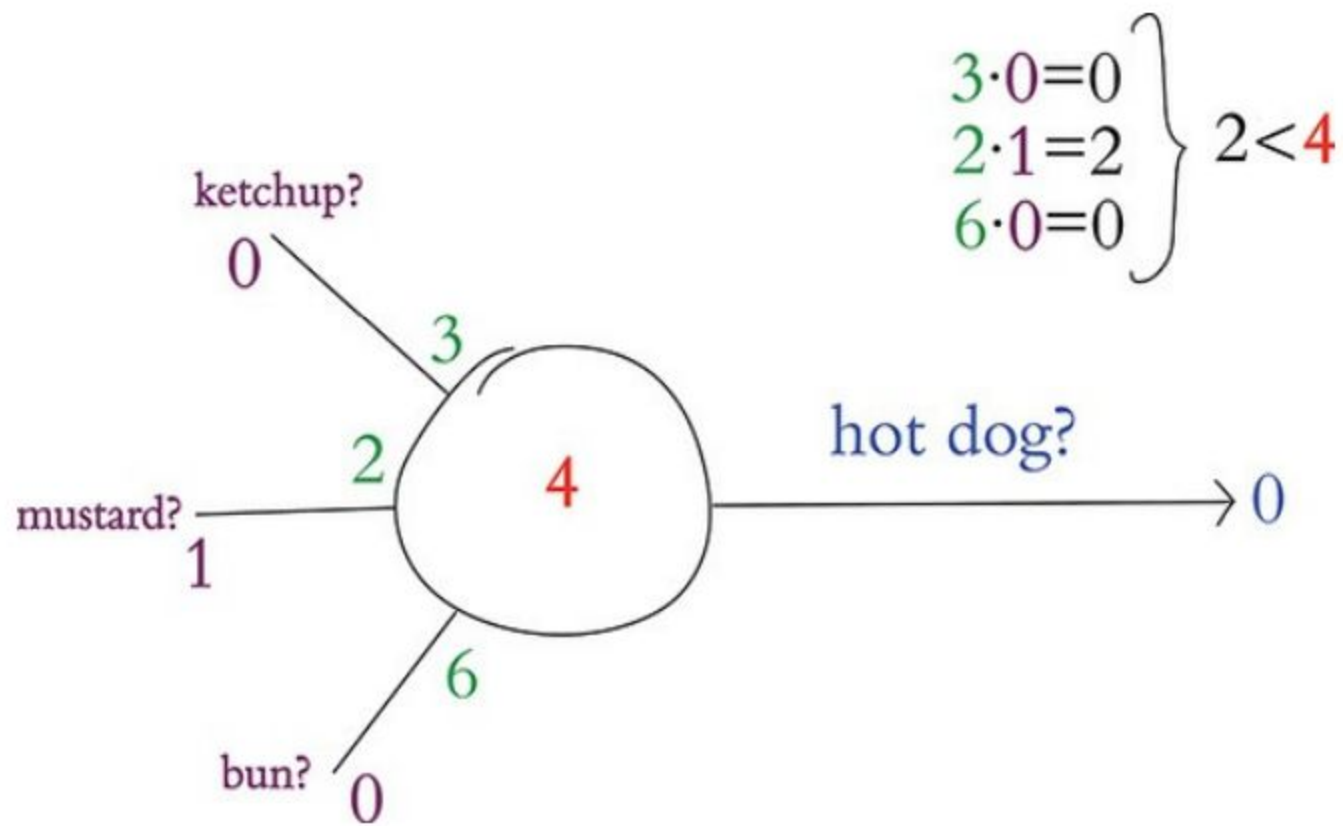
$$u \cdot v = 1 \cdot 4 + 2 \cdot 5 + 3 \cdot 6 = 32$$





$$\left. \begin{array}{l} 3 \cdot 1 = 3 \\ 2 \cdot 1 = 2 \\ 6 \cdot 0 = 0 \end{array} \right\} 5 > 4$$

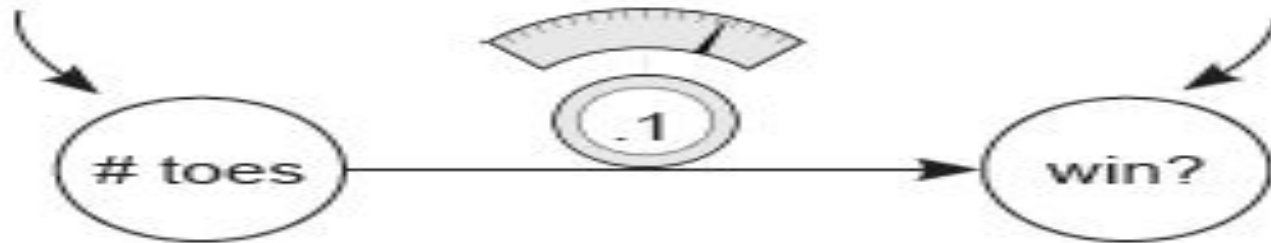




# Neuron

## ① An empty network

Input data  
enters here.



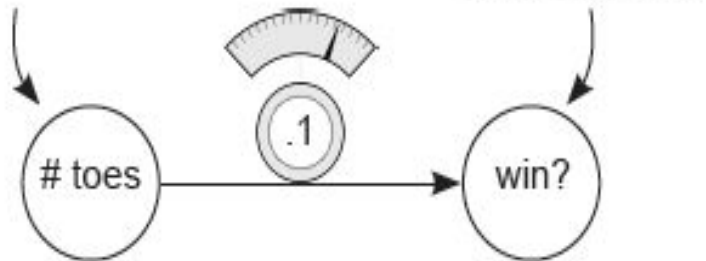
Predictions  
come out here.

As you can see, with one weight, this network takes in one datapoint at a time (average number of toes per player on the baseball team) and outputs a single prediction (whether it thinks the team will win).

# Neuron

## ❶ An empty network

Input data  
enters here.

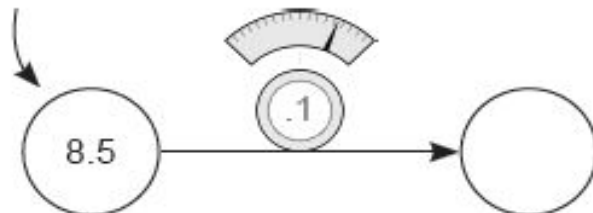


```
weight = 0.1
```

```
def neural_network(input, weight):  
    prediction = input * weight  
    return prediction
```

## ❷ Inserting one input datapoint

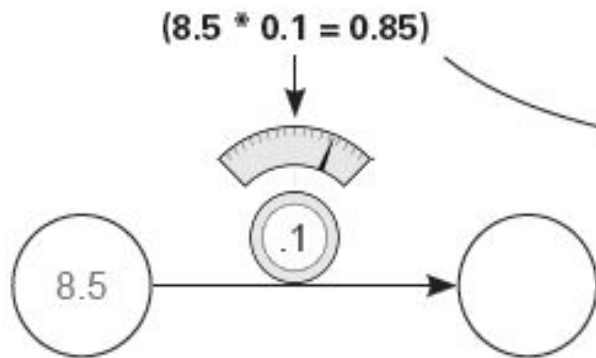
Input data  
(# toes)



```
number_of_toes = [8.5, 9.5, 10, 9]  
input = number_of_toes[0]  
pred = neural_network(input, weight)  
print(pred)
```

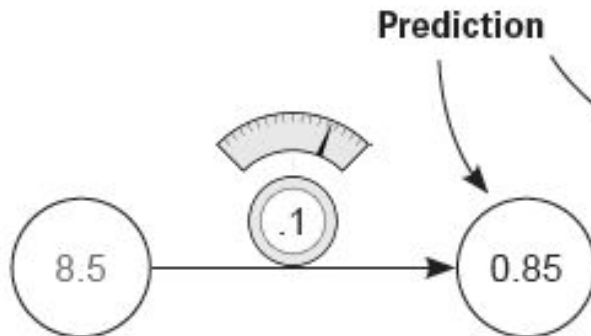
# Neuron

## 3 Multiplying input by weight



```
def neural_network(input, weight):  
    prediction = input * weight  
    return prediction
```

## 4 Depositing the prediction



```
number_of_toes = [8.5, 9.5, 10, 9]  
input = number_of_toes[0]  
pred = neural_network(input, weight)
```

# What is a neural network?

```
weight = 0.1
```

```
def neural_network(input, weight):
```

```
    prediction = input * weight
```

```
    return prediction
```

**The network**



# What is a neural network?

```
weight = 0.1
```

```
def neural_network(input, weight):  
    prediction = input * weight  
    return prediction
```

The network

```
number_of_toes = [8.5, 9.5, 10, 9]
```

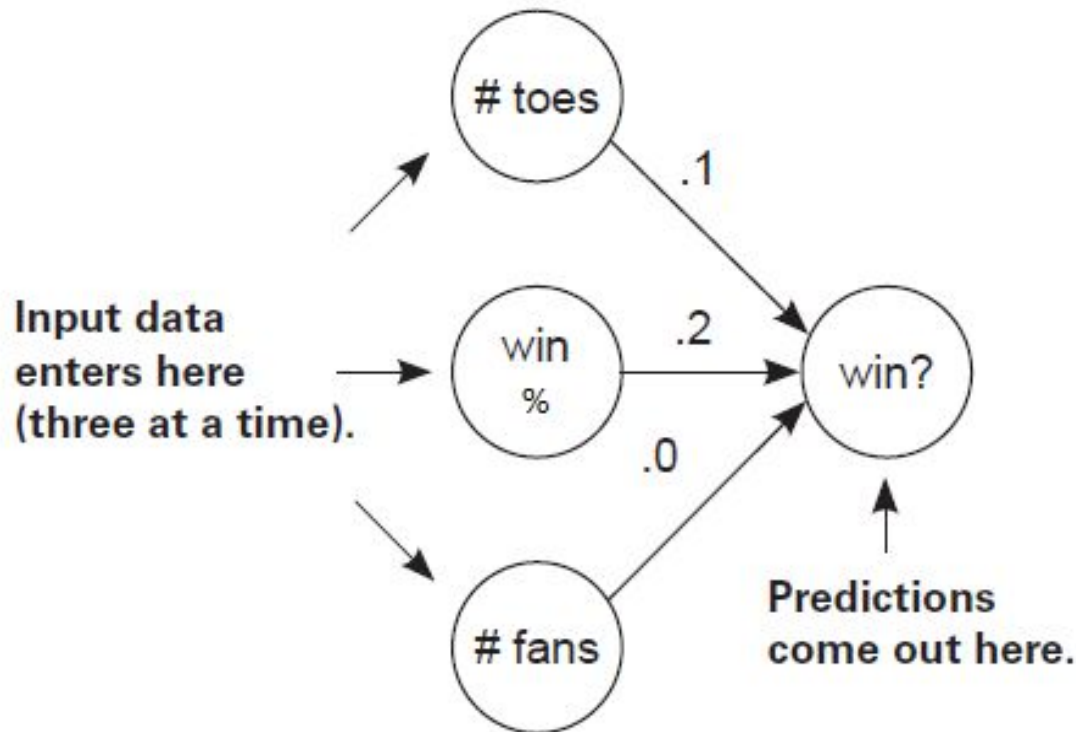
```
input = number_of_toes[0]
```

```
pred = neural_network(input, weight)  
print(pred)
```

How you use the  
network to predict  
something

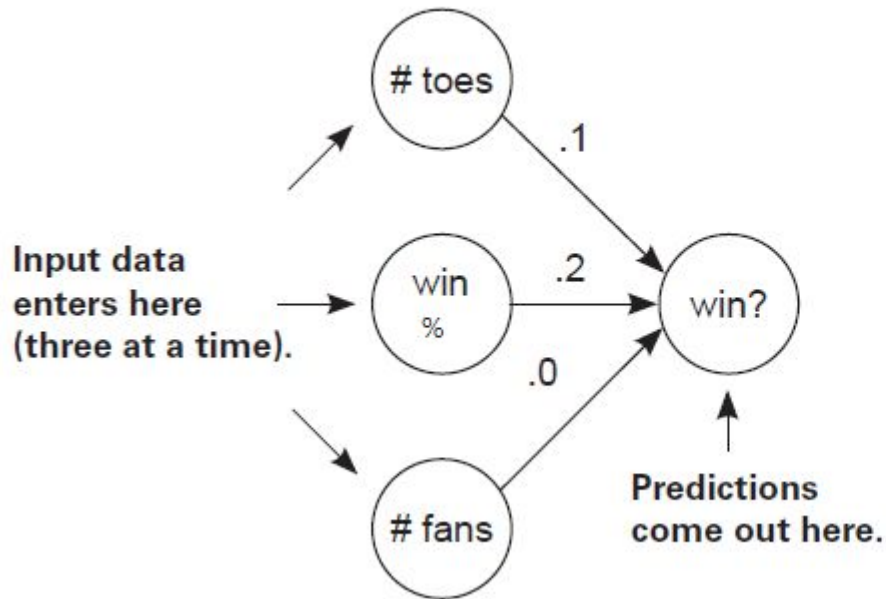
# Making a prediction with multiple inputs

What if you could give the network **more information** (at one time) than just the average number of toes per player? In that case, the network should, in theory, be able to make **more-accurate** predictions.



# Making a prediction with multiple inputs

## 1 An empty network with multiple inputs



```
weights = [0.1, 0.2, 0]
```

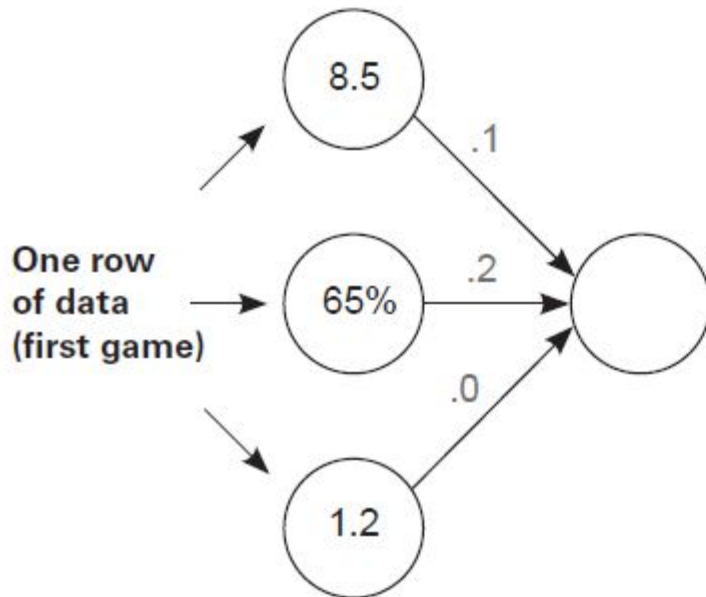
```
def neural_network(input, weights):
```

```
    pred = w_sum(input, weights)
```

```
    return pred
```

# Making a prediction with multiple inputs

## ② Inserting one input datapoint



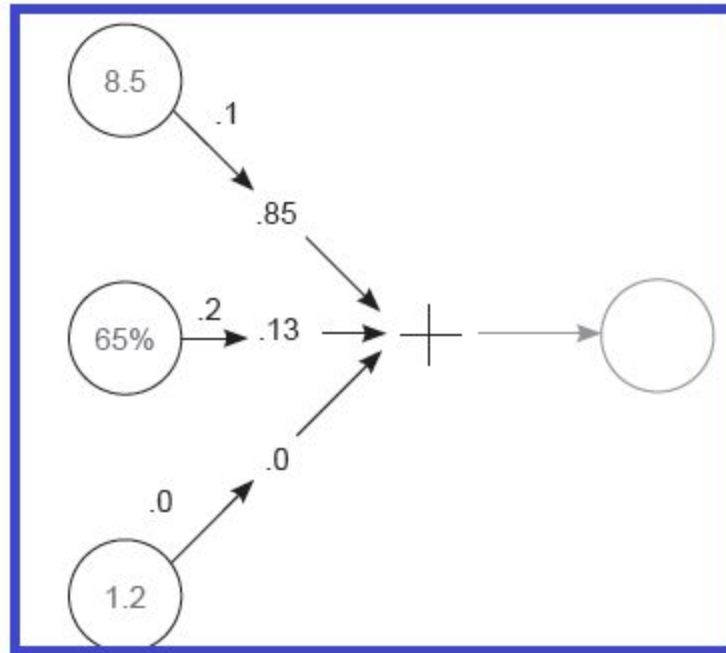
This dataset is the current status at the beginning of each game for the first four games in a season:  
toes = current average number of toes per player  
wlrec = current games won (percent)  
nfans = fan count (in millions).

```
toes = [8.5, 9.5, 9.9, 9.0]  
wlrec = [0.65, 0.8, 0.8, 0.9]  
nfans = [1.2, 1.3, 0.5, 1.0]
```

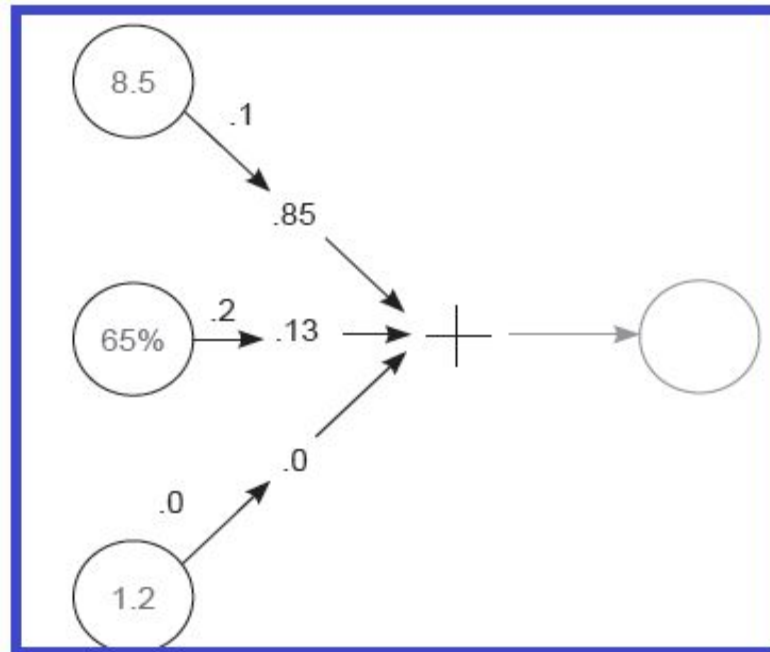
```
input = [toes[0], wlrec[0], nfans[0]]  
pred = neural_network(input, weights)
```

Input corresponds to every entry for the first game of the season.

# Making a prediction with multiple inputs

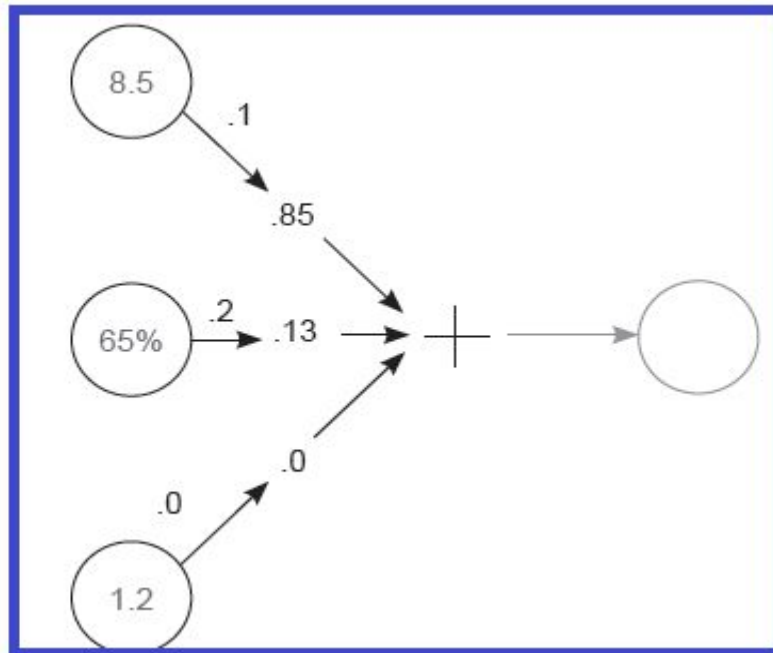


# Making a prediction with multiple inputs



Inputs	Weights	Local predictions	
(8.50 * 0.1)	=	0.85	= toes prediction
(0.65 * 0.2)	=	0.13	= wlrec prediction
(1.20 * 0.0)	=	0.00	= fans prediction
toes prediction + wlrec prediction + fans prediction = final prediction			
0.85	+	0.13	+ 0.00 = 0.98

# Making a prediction with multiple inputs

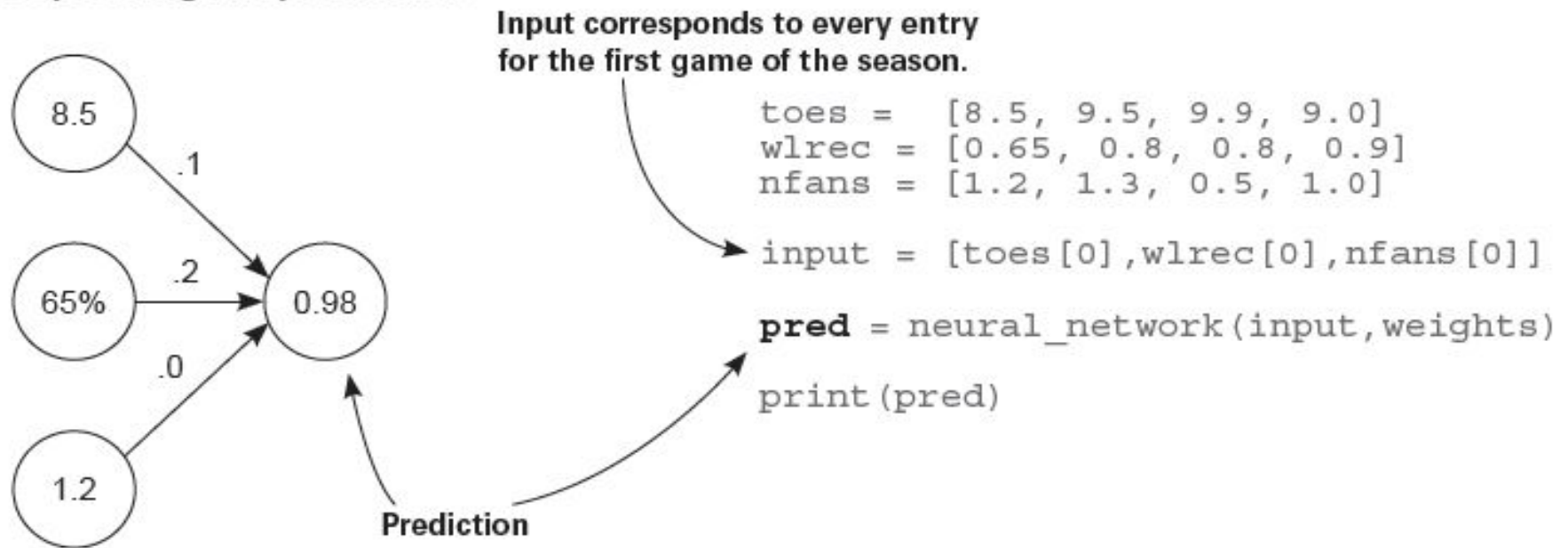


```
def w_sum(a,b):  
    assert(len(a) == len(b))  
    output = 0  
    for i in range(len(a)):  
        output += (a[i] * b[i])  
    return output  
  
def neural_network(input, weights):  
    pred = w_sum(input,weights)  
    return pred
```

Inputs	Weights	Local predictions	
(8.50 * 0.1)	=	0.85	= toes prediction
(0.65 * 0.2)	=	0.13	= wlrec prediction
(1.20 * 0.0)	=	0.00	= fans prediction
toes prediction + wlrec prediction + fans prediction = final prediction			
0.85	+	0.13	+ 0.00 = 0.98

# Making a prediction with multiple inputs

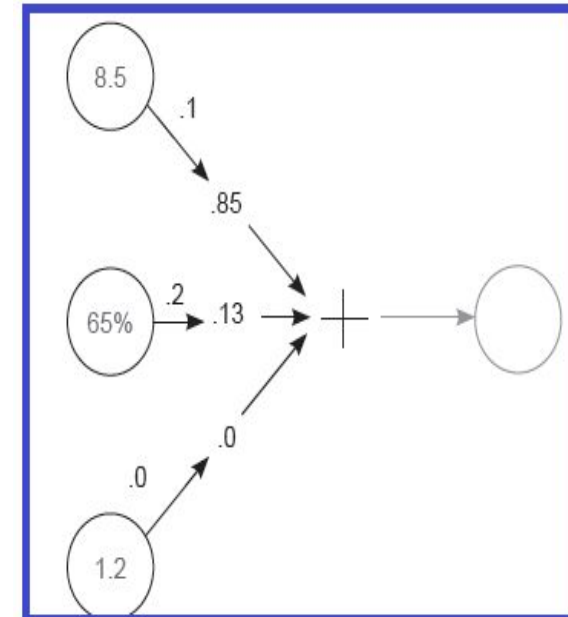
## ④ Depositing the prediction





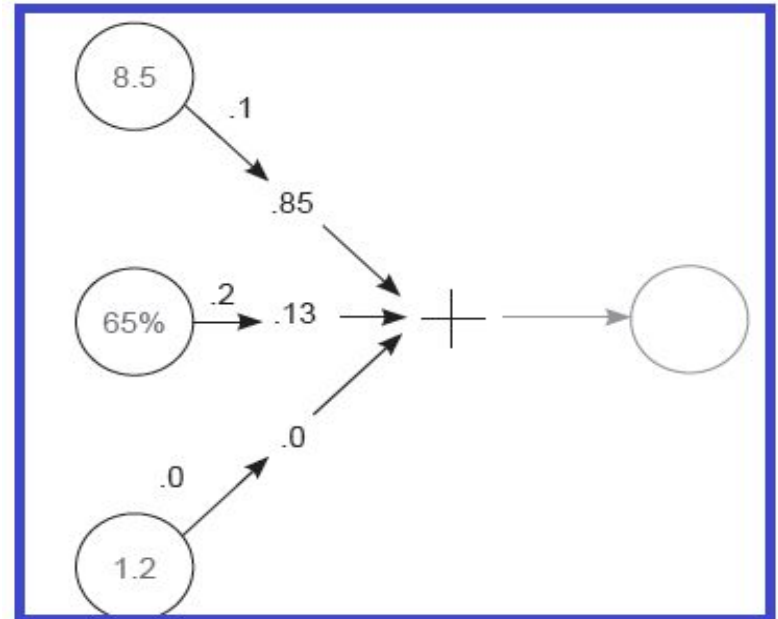
# Making a prediction with multiple inputs

- This new neural network can accept **multiple inputs** at a time per prediction.
- This allows the network to **combine various forms of information** to make **better-informed** decisions.
- But the **fundamental mechanism for using weights hasn't changed**.
- You still take each input and run it through its own volume knob.
- In other words, **you multiply each input by its own weight**.



# Making a prediction with multiple inputs

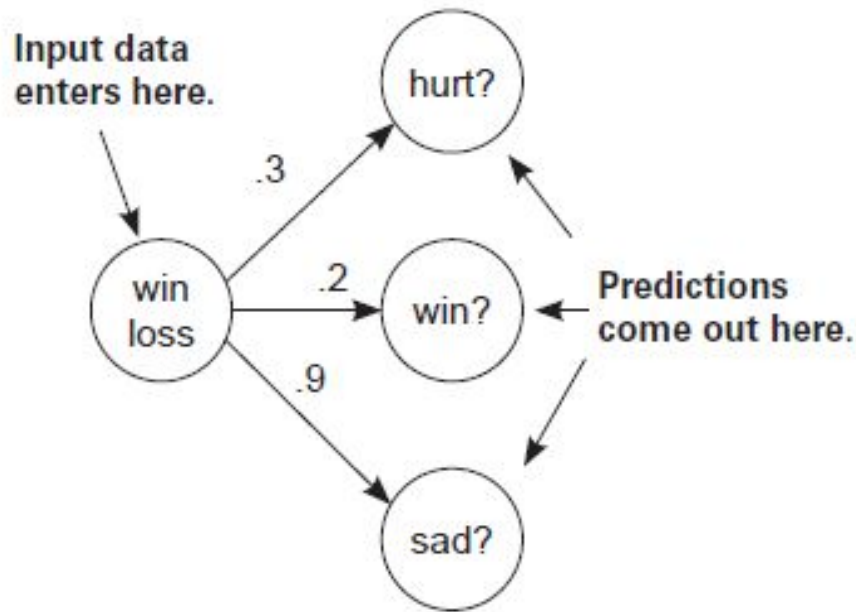
- The new property here is that, because you have multiple inputs, you have to sum their respective predictions.
- Thus, you multiply each input by its respective weight and then sum all the local predictions together.
- This is called a weighted sum of the input, or a weighted sum for short. Some also refer to the weighted sum as a dot product, as you'll see.



# Making a prediction with multiple outputs

Neural networks can also make multiple predictions using only a single input.

## ① An empty network with multiple outputs



Instead of predicting just whether the team won or lost, you're also predicting whether the players are happy or sad and the percentage of team members who are hurt. You make this prediction using only the current win/loss record.

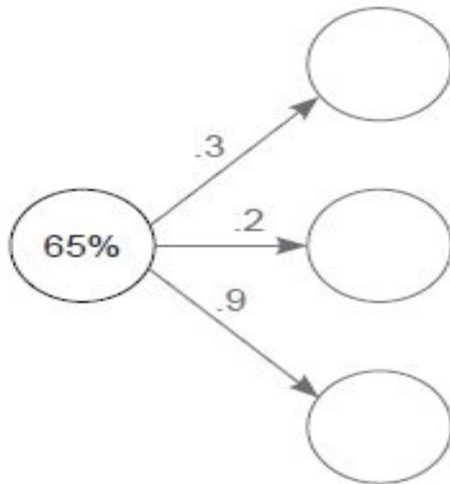
```
weights = [0.3, 0.2, 0.9]
```

```
def neural_network(input, weights):  
    pred = ele_mul(input, weights)  
    return pred
```

Observation?

# Making a prediction with multiple outputs

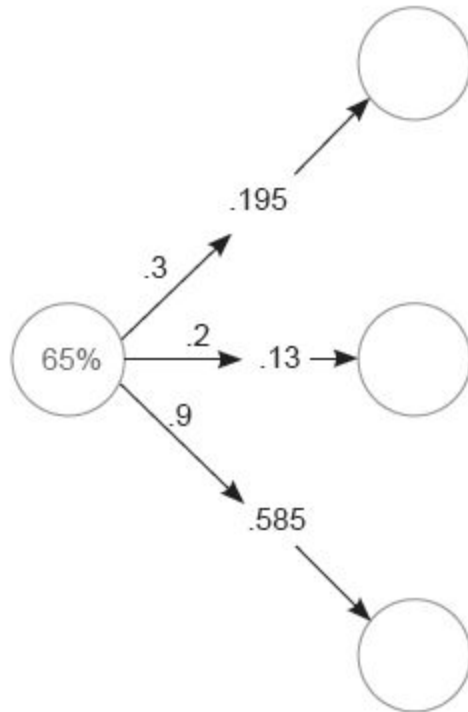
## ② Inserting one input datapoint



```
wlrec = [0.65, 0.8, 0.8, 0.9]  
input = wlrec[0]  
pred = neural_network(input, weights)
```

# Making a prediction with multiple outputs

## ③ Performing elementwise multiplication



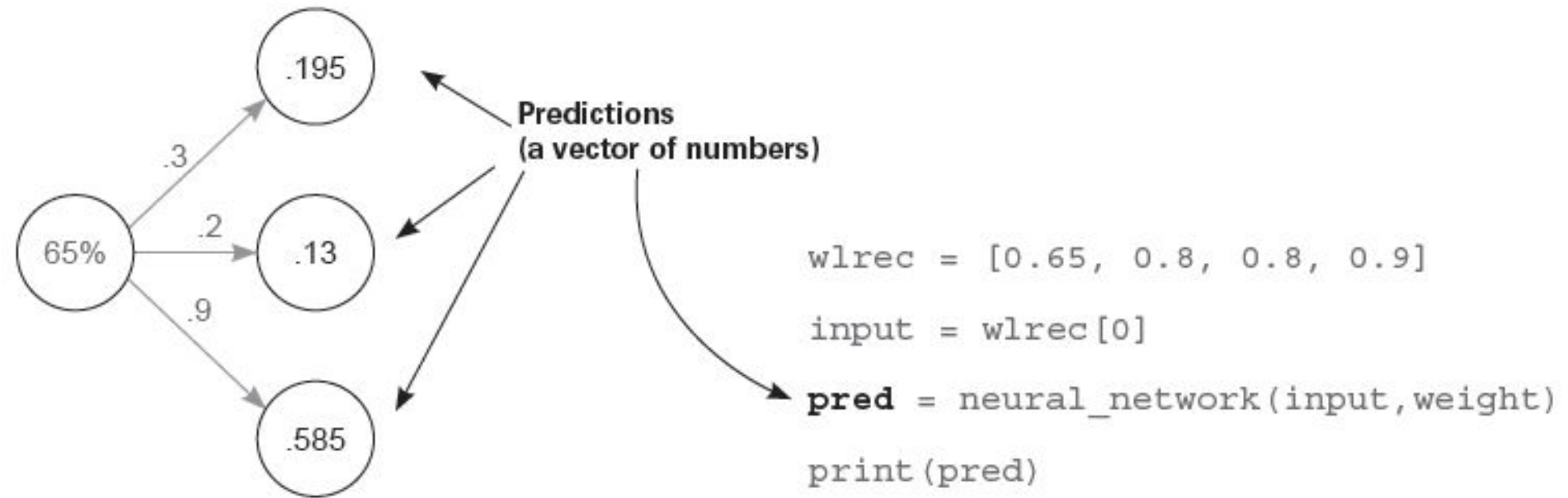
```
def ele_mul(number, vector):  
    output = [0,0,0]  
    assert(len(output) == len(vector))  
    for i in range(len(vector)):  
        output[i] = number * vector[i]  
    return output
```

```
def neural_network(input, weights):  
    pred = ele_mul(input, weights)  
    return pred
```

Inputs	Weights	Final predictions
(0.65 * 0.3)	=	0.195 = hurt prediction
(0.65 * 0.2)	=	0.13 = win prediction
(0.65 * 0.9)	=	0.585 = sad prediction

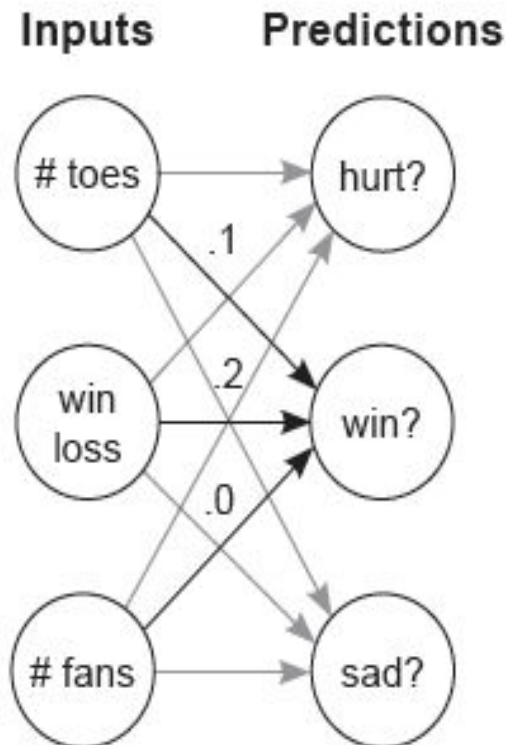
# Making a prediction with multiple outputs

## 4 Depositing predictions



# Predicting with multiple inputs and outputs

## 1 An empty network with multiple inputs and outputs

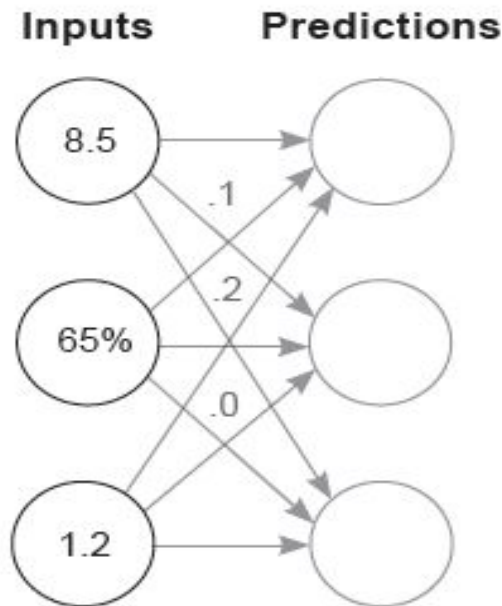


```
# toes % win % fans
weights = [ [0.1, 0.1, -0.3], # hurt?
            [0.1, 0.2, 0.0], # win?
            [0.0, 1.3, 0.1] ] # sad?
```

```
def neural_network(input, weights):
    pred = vect_mat_mul(input, weights)
    return pred
```

# Predicting with multiple inputs and outputs

## ② Inserting one input datapoint



This dataset is the current status at the beginning of each game for the first four games in a season:  
toes = current average number of toes per player  
wlrec = current games won (percent)  
fans = fan count (in millions)

```
toes = [8.5, 9.5, 9.9, 9.0]  
wlrec = [0.65, 0.8, 0.8, 0.9]  
nfans = [1.2, 1.3, 0.5, 1.0]
```

```
input = [toes[0], wlrec[0], nfans[0]]  
pred = neural_network(input, weights)
```

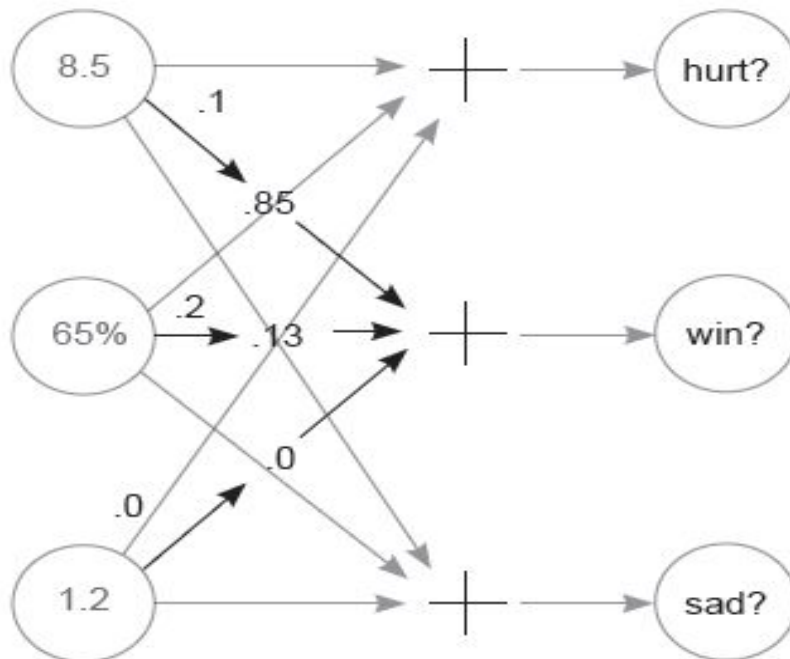
Input corresponds to every entry for the first game of the season.



# Predicting with mul inputs and outputs

```
# toes % win # fans
weights = [ [0.1, 0.1, -0.3], # hurt?
             [0.1, 0.2, 0.0], # win?
             [0.0, 1.3, 0.1] ] # sad?
```

③ For each output, performing a weighted sum of inputs



```
def w_sum(a,b):
    assert(len(a) == len(b))
    output = 0
    for i in range(len(a)):
        output += (a[i] * b[i])
    return output
```

```
def vect_mat_mul(vect,matrix):
    assert(len(vect) == len(matrix))
    output = [0,0,0]

    for i in range(len(vect)):
        output[i] = w_sum(vect,matrix[i])

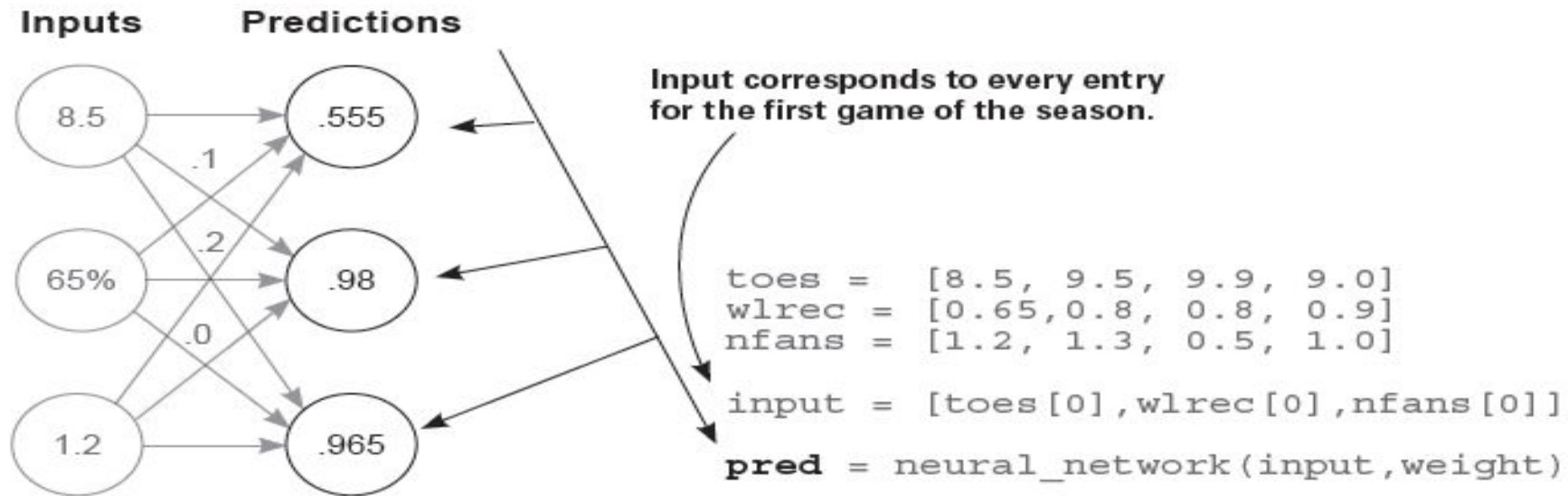
    return output
```

```
def neural_network(input, weights):
    pred = vect_mat_mul(input,weights)
    return pred
```

# toes	% win	# fans			
(8.5 * 0.1)	+	(0.65 * 0.1)	+	(1.2 * -0.3)	= 0.555 = hurt prediction
(8.5 * 0.1)	+	(0.65 * 0.2)	+	(1.2 * 0.0)	= 0.98 = win prediction
(8.5 * 0.0)	+	(0.65 * 1.3)	+	(1.2 * 0.1)	= 0.965 = sad prediction

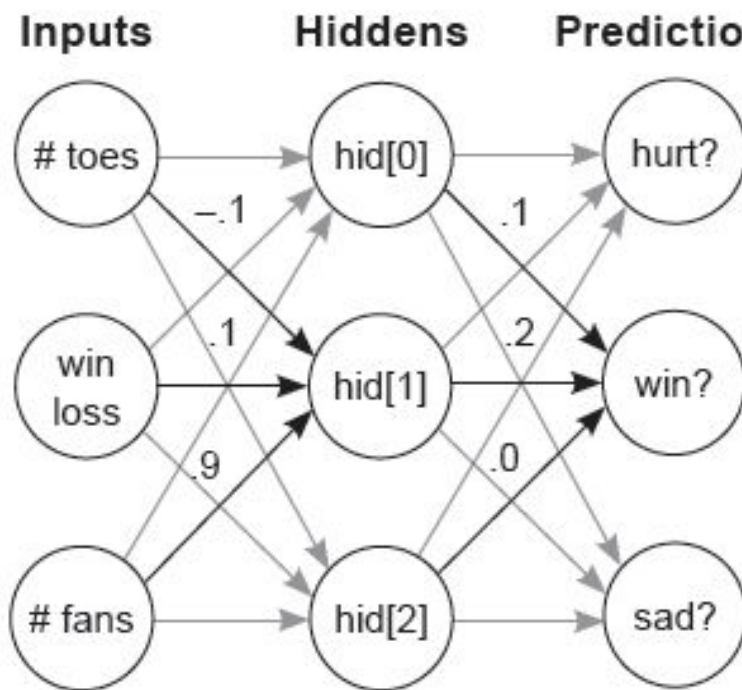
# Predicting with multiple inputs and outputs

## 4 Depositing predictions



# Predicting on predictions

## 1 An empty network with multiple inputs and outputs



```
ih_wgt = [ # toes % win # fans
            [0.1, 0.2, -0.1], # hid[0]
            [-0.1, 0.1, 0.9], # hid[1]
            [0.1, 0.4, 0.1] ] # hid[2]
```

```
hp_wgt = [ #hid[0] hid[1] hid[2]
            [0.3, 1.1, -0.3], # hurt?
            [0.1, 0.2, 0.0], # win?
            [0.0, 1.3, 0.1] ] # sad?
```

```
weights = [ih_wgt, hp_wgt]
```

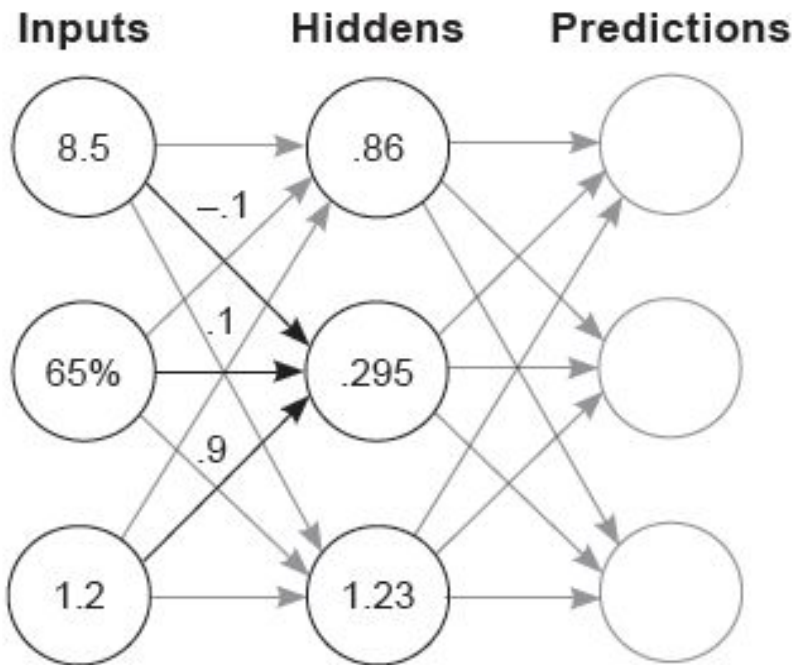
```
def neural_network(input, weights):
    hid = vect_mat_mul(input, weights[0])
    pred = vect_mat_mul(hid, weights[1])
    return pred
```

# Predicting on predictions

```
ih_wgt = [ # toes % win # fans
           [0.1, 0.2, -0.1], # hid[0]
           [-0.1, 0.1, 0.9], # hid[1]
           [0.1, 0.4, 0.1] ] # hid[2]
```

```
hp_wgt = [ #hid[0] hid[1] hid[2]
           [0.3, 1.1, -0.3], # hurt?
           [0.1, 0.2, 0.0], # win?
           [0.0, 1.3, 0.1] ] # sad?
```

## ② Predicting the hidden layer



Input corresponds to every entry  
for the first game of the season.

```
toes = [8.5, 9.5, 9.9, 9.0]
wlrec = [0.65, 0.8, 0.8, 0.9]
nfans = [1.2, 1.3, 0.5, 1.0]
```

```
input = [toes[0], wlrec[0], nfans[0]]
```

```
pred = neural_network(input, weights)
```

```
def neural_network(input, weights):
```

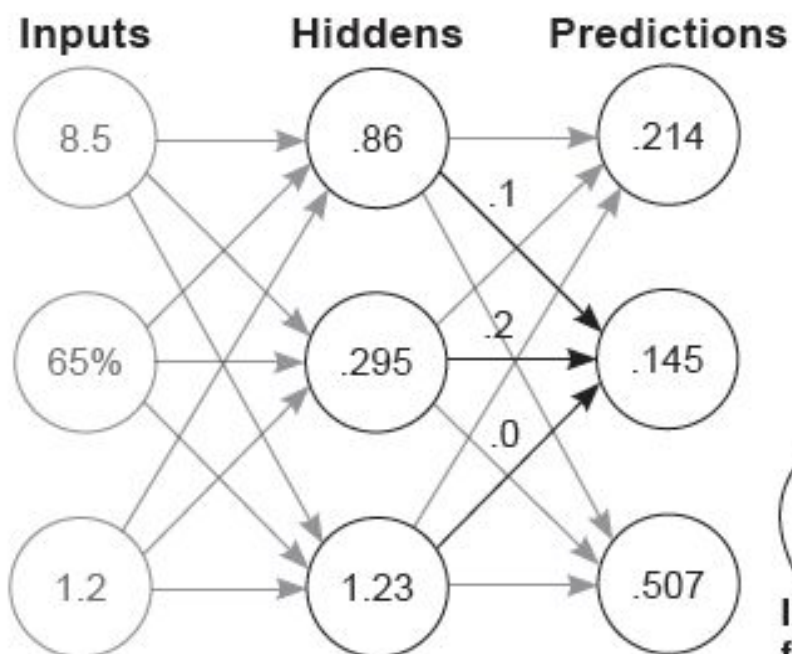
```
    hid = vect_mat_mul(input, weights[0])
    pred = vect_mat_mul(hid, weights[1])
    return pred
```

# Predicting on predictions

```
ih_wgt = [ [0.1, 0.2, -0.1], # hid[0]
            [-0.1, 0.1, 0.9], # hid[1]
            [0.1, 0.4, 0.1] ] # hid[2]

hp_wgt = [ #hid[0] hid[1] hid[2]
            [0.3, 1.1, -0.3], # hurt?
            [0.1, 0.2, 0.0], # win?
            [0.0, 1.3, 0.1] ] # sad?
```

## 3 Predicting the output layer (and depositing the prediction)



```
def neural_network(input, weights):
    hid=vect_mat_mul(input,weights[0])
    pred=vect_mat_mul(hid,weights[1])
    return pred

toes = [8.5, 9.5, 9.9, 9.0]
wlrec = [0.65, 0.8, 0.8, 0.9]
nfans = [1.2, 1.3, 0.5, 1.0]

input = [toes[0],wlrec[0],nfans[0]]

pred = neural_network(input,weights)
print(pred)
```

Input corresponds to every entry  
for the first game of the season.

# Predicting on predictions

## Summary

- To predict, neural networks perform repeated **weighted sums of the input**.
- You've seen an increasingly complex variety of neural networks in this chapter.
- I hope it's clear that a relatively **small number of simple rules** are **used repeatedly to create larger, more advanced neural networks**.
- The network's **intelligence** depends on the **weight** values you give it.



## 1. Variance and Covariance: The Foundations

### Variance

- **Definition:** Variance quantifies the spread of a single variable around its mean.

$$\text{Var}(X) = \frac{1}{n} \sum_{i=1}^n (x_i - \mu_X)^2$$

Where:

- $x_i$ : Individual values of  $X$
- $\mu_X$ : Mean of  $X$
- $n$ : Number of samples
- **Interpretation:**
  - High variance: The data is spread out from the mean.
  - Low variance: The data is clustered closely around the mean.

## Covariance

- **Definition:** Covariance measures how two variables vary together.

$$\text{Cov}(X, Y) = \frac{1}{n} \sum_{i=1}^n (x_i - \mu_X)(y_i - \mu_Y)$$

Where:

- $X$  and  $Y$ : Two variables
- $\mu_X$  and  $\mu_Y$ : Their means
- **Interpretation:**
  - Positive covariance: When  $X$  increases,  $Y$  tends to increase.
  - Negative covariance: When  $X$  increases,  $Y$  tends to decrease.
  - Near-zero covariance: No linear relationship between  $X$  and  $Y$ .



# 1. Understanding Covariance

## What does a covariance number indicate?

- **High covariance:** Two features are strongly related (change together).  
For example: If covariance between Open and Close is high, they provide **overlapping** information.
- **Low covariance:** Two features are less related, indicating they provide **distinct** information.

## Why is high covariance problematic?

- High **covariance** among features leads to **redundancy**.
- Models trained on highly correlated features might \_\_\_\_\_ and become less interpretable.

# 1. What is Covariance?

- Covariance measures the degree to which two features change together.
- A positive covariance indicates that as one feature increases, the other tends to increase.
- The magnitude of covariance depends on the scale of the features.

## Explanation of the Pairs

a. tenure with tenure :

- Covariance: 603.168
- Explanation:
  - This is the covariance of tenure with itself.
  - The value represents the **variance** of tenure because:

$$\text{Cov}(X, X) = \text{Var}(X)$$

- High variance indicates that tenure has a wide spread of values, meaning customers have varied lengths of tenure with the service.

## **Tenure with MonthlyCharges:**

**Covariance:** 183.197

### **Explanation:**

- This measures how tenure and MonthlyCharges vary together.
- A positive covariance indicates that customers with longer tenure tend to have higher monthly charges.

## **Insights from Covariance Values:**

### **Magnitude of Covariance:**

The higher the covariance, the stronger the linear relationship between the features. tenure and MonthlyCharges have a moderate covariance (183.197), indicating some correlation between these features.

### **Feature Redundancy:**

High covariance between features like tenure and MonthlyCharges suggests potential redundancy.

Both features may contribute overlapping information to the model.

### **Feature Selection:**

Removing one of the features in a high-covariance pair reduces redundancy, simplifies the model, and can improve generalization.

If height ( $X$ ) and weight ( $Y$ ) are highly covariant,

---

If height ( $X$ ) and weight ( $Y$ ) are highly covariant, taller individuals tend to weigh more.

## 1. Scaling Makes the Relationship Comparable

- **Covariance** measures the relationship between two variables in raw units (e.g., height in cm and weight in kg).
- **Correlation** standardizes (scales) covariance by dividing it by the product of the standard deviations of the two variables. This removes the effect of units and scales.

**Formula:** 
$$\rho_{X,Y} = \frac{\text{Cov}(X, Y)}{\sigma_X \cdot \sigma_Y}$$

Where:

- $\text{Cov}(X, Y)$ : Covariance of  $X$  and  $Y$ .
- $\sigma_X$ : Standard deviation of  $X$ .
- $\sigma_Y$ : Standard deviation of  $Y$ .

## 2. Correlation Lies Within a Defined Range

Unlike covariance, which can take any value depending on the magnitude of the variables, **correlation is always bounded between  $-1$  and  $+1$ :**

- $+1$ : Perfect positive linear relationship.
- $0$ : No linear relationship.
- $-1$ : Perfect negative linear relationship.

This bounded range makes correlation easier to interpret and compare across datasets or variable pairs.

## 3. Correlation Removes Scale Dependence

- Covariance depends on the **units and scale** of the variables:
  - For example, if you change the height unit from cm to meters, the covariance value will change.
- Correlation is **unit-free** because it uses standardized values (z-scores) of the variables:

$$Z_X = \frac{X - \bar{X}}{\sigma_X}, \quad Z_Y = \frac{Y - \bar{Y}}{\sigma_Y}$$

This means correlation is unaffected by the units or magnitude of the variables.

Height (cm)	Weight (kg)
150	50
160	55
170	60
180	65

**Covariance:**

$$\text{Cov}(X, Y) = 83.33$$

- Positive covariance indicates that height and weight increase together.
- The value 83.33 depends on the scales of the variables (cm and kg).

**Correlation:**

$$\rho_{X,Y} = \frac{\text{Cov}(X, Y)}{\sigma_X \cdot \sigma_Y} = 1.0$$

- A correlation of 1.0 indicates a perfect positive linear relationship.



## **6. Practical Use Cases**

### **Covariance:**

- Principal Component Analysis (PCA):
- Covariance matrix helps in identifying directions of maximum variance.

### **Finance:**

- Covariance of stock returns indicates how two stocks move together.

### **Correlation:**

- Feature Selection:
- Identify redundant features in datasets.

### **Regression Analysis:**

- Correlation between predictors and target variable helps determine importance.

# Principal Component Analysis (PCA)

- PCA is a dimensionality reduction technique used to transform a dataset with many features into a smaller set of components while retaining as **much variance** (**information**) as possible.
- It achieves this by finding **new axes**, called **principal components**, that maximize **variance**.

# Steps of PCA

## 1. Standardize the Data:

- Convert all features to have a mean of 0 and a standard deviation of 1. This ensures that variables with larger scales don't dominate.

## 2. Compute the Covariance Matrix:

- The covariance matrix summarizes the pairwise covariances between features in the dataset.
- If the dataset  $X$  has  $n$  features, the covariance matrix  $\Sigma$  will be an  $n \times n$  matrix where:

$$\Sigma_{i,j} = \text{Cov}(X_i, X_j)$$

## 3. Find Eigenvalues and Eigenvectors:

- **Eigenvalues** represent the amount of variance explained by each principal component.
- **Eigenvectors** represent the direction of the principal components (the axes along which variance is maximized).

## 4. Select Principal Components:

- Rank the eigenvalues in descending order.
- Choose the top  $k$  eigenvalues and their corresponding eigenvectors to form the new feature space.

## 5. Transform the Data:

- Project the original data onto the new feature space using the selected eigenvectors.

# Importance of Covariance in PCA

## Identifying Variance:

- The covariance matrix captures the relationships and variance between features. Higher covariance between features indicates redundancy, which PCA aims to reduce.

## Directions of Maximum Variance:

- PCA identifies the directions (principal components) where the data varies the most. These directions correspond to the eigenvectors of the covariance matrix, and the magnitude of variance is given by the eigenvalues.

## Feature Relationships:

- Features with high covariance contribute significantly to the same principal component. PCA exploits these relationships to reduce redundancy.

# How the Covariance Matrix Identifies Maximum Variance

## 1. Covariance Matrix Calculation:

- For a dataset with features  $X_1, X_2, \dots, X_n$ , the covariance matrix is:

$$\Sigma = \begin{bmatrix} \text{Var}(X_1) & \text{Cov}(X_1, X_2) & \dots & \text{Cov}(X_1, X_n) \\ \text{Cov}(X_2, X_1) & \text{Var}(X_2) & \dots & \text{Cov}(X_2, X_n) \\ \vdots & \vdots & \ddots & \vdots \\ \text{Cov}(X_n, X_1) & \text{Cov}(X_n, X_2) & \dots & \text{Var}(X_n) \end{bmatrix}$$

- Diagonal elements represent the variance of individual features.
- Off-diagonal elements represent the covariance between feature pairs.

## 2. Eigenvectors and Eigenvalues:

- Solving the eigenvalue equation for the covariance matrix:

$$\Sigma v = \lambda v$$

- $\lambda$ : Eigenvalues (representing variance explained by each principal component).
- $v$ : Eigenvectors (directions of maximum variance).

### 3. Selecting Principal Components:

- The eigenvector corresponding to the largest eigenvalue points in the direction of maximum variance (first principal component).
- Subsequent eigenvectors represent directions of decreasing variance.

Observation	X 1	X2
1	2	3
2	4	5
3	6	6
4	8	8

## Step 1: Standardize the Data

To ensure that all features contribute equally to the PCA, we standardize the dataset.

**Formula:** 
$$Z = \frac{X - \bar{X}}{\sigma_X}$$

Where:

- $\bar{X}$ : Mean of the feature.
- $\sigma_X$ : Standard deviation of the feature.

**Standardize  $X_1$ :**

- Mean:  $\bar{X}_1 = \frac{2+4+6+8}{4} = 5$
- Standard deviation:  $\sigma_{X_1} = \sqrt{\frac{(2-5)^2+(4-5)^2+(6-5)^2+(8-5)^2}{4}} = 2.236$

$$Z_1 = \frac{X_1 - 5}{2.236} = \begin{bmatrix} \frac{2-5}{2.236} \\ \frac{4-5}{2.236} \\ \frac{6-5}{2.236} \\ \frac{8-5}{2.236} \end{bmatrix} = \begin{bmatrix} -1.34 \\ -0.45 \\ 0.45 \\ 1.34 \end{bmatrix}$$

**Standardize  $X_2$ :**

- Mean:  $\bar{X}_2 = \frac{3+5+6+8}{4} = 5.5$
- Standard deviation:  $\sigma_{X_2} = \sqrt{\frac{(3-5.5)^2+(5-5.5)^2+(6-5.5)^2+(8-5.5)^2}{4}} = 1.87$

$$Z_2 = \frac{X_2 - 5.5}{1.87} = \begin{bmatrix} \frac{3-5.5}{1.87} \\ \frac{5-5.5}{1.87} \\ \frac{6-5.5}{1.87} \\ \frac{8-5.5}{1.87} \end{bmatrix} = \begin{bmatrix} -1.34 \\ -0.27 \\ 0.27 \\ 1.34 \end{bmatrix}$$



**Standardized Dataset:**  $Z = \begin{bmatrix} -1.34 & -1.34 \\ -0.45 & -0.27 \\ 0.45 & 0.27 \\ 1.34 & 1.34 \end{bmatrix}$

## Step 2: Compute the Covariance Matrix

The covariance matrix captures how the features vary together.

**Covariance Formula:**

$$\text{Cov}(X_1, X_2) = \frac{\sum (X_{1,i} - \bar{X}_1)(X_{2,i} - \bar{X}_2)}{n - 1}$$

Using the standardized data:

- Variance of  $Z_1$ :  $\text{Var}(Z_1) = \frac{(-1.34)^2 + (-0.45)^2 + (0.45)^2 + (1.34)^2}{4-1} = 1$
- Variance of  $Z_2$ :  $\text{Var}(Z_2) = 1$
- Covariance of  $Z_1$  and  $Z_2$ :

$$\text{Cov}(Z_1, Z_2) = \frac{(-1.34 \cdot -1.34) + (-0.45 \cdot -0.27) + (0.45 \cdot 0.27) + (1.34 \cdot 1.34)}{3} = 0.95$$

**Covariance Matrix:**  $\Sigma = \begin{bmatrix} 1 & 0.95 \\ 0.95 & 1 \end{bmatrix}$

$$\Sigma u = \lambda u$$

$$(\Sigma - \lambda I)u = 0$$

Here:

- $\lambda I$  is the identity matrix scaled by  $\lambda$ , ensuring the dimensions match  $\Sigma$ .
- $(\Sigma - \lambda I)$ : A matrix resulting from subtracting  $\lambda I$  from  $\Sigma$ .

## 2. Non-Trivial Solution Condition:

- For  $u$  to be a non-zero vector (non-trivial solution),  $(\Sigma - \lambda I)$  must be singular.
- A matrix is singular if its determinant is zero:

$$\text{Det}(\Sigma - \lambda I) = 0$$

3. **Result:** Solving  $\text{Det}(\Sigma - \lambda I) = 0$  gives the **eigenvalues** ( $\lambda$ ) of  $\Sigma$ .

### Solve the Eigenvalue Equation:

The eigenvalue equation is:

$$\text{Det}(\Sigma - \lambda I) = 0$$

Where  $I$  is the identity matrix.

For:

$$\Sigma = \begin{bmatrix} 1 & 0.95 \\ 0.95 & 1 \end{bmatrix}$$

We compute:

$$\text{Det} \begin{bmatrix} 1 - \lambda & 0.95 \\ 0.95 & 1 - \lambda \end{bmatrix} = (1 - \lambda)^2 - 0.95^2 = 0$$

$$\text{Det} \begin{bmatrix} 1 - \lambda & 0.95 \\ 0.95 & 1 - \lambda \end{bmatrix} = (1 - \lambda)^2 - 0.95^2 = 0$$

$$(1 - \lambda)^2 - 0.9025 = 0 \implies (1 - \lambda) = \pm 0.95$$

$$\lambda_1 = 1.95, \quad \lambda_2 = 0.05$$

**Eigenvectors:**

$$\text{For } \lambda_1 = 1.95: \quad (\Sigma - 1.95I)v = 0 \implies \begin{bmatrix} 1 - 1.95 & 0.95 \\ 0.95 & 1 - 1.95 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = 0$$

$$\text{Solve: } \begin{bmatrix} -0.95 & 0.95 \\ 0.95 & -0.95 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = 0$$

$$\text{Eigenvector: } v_1 = \begin{bmatrix} 0.707 \\ 0.707 \end{bmatrix}$$

$$\text{For } \lambda_2 = 0.05: \quad v_2 = \begin{bmatrix} -0.707 \\ 0.707 \end{bmatrix}$$

## Step 4: Select Principal Components

1. The eigenvalue  $\lambda_1 = 1.95$  explains most of the variance (~97.5% of the total variance:  $\frac{1.95}{1.95+0.05}$ ).
2. The eigenvalue  $\lambda_2 = 0.05$  explains only ~2.5% of the variance.

We select the first principal component (PC1).

### 1. Understanding the Formula

- $Z$ : Matrix of standardized data.
- $v_1$ : The first eigenvector (principal component direction).
- $Z'$ : The projection of the original data onto the first principal component.

This formula represents the transformation of the original data  $Z$  into a new coordinate system defined by the eigenvector  $v_1$ .

## Step 5: Transform the Data

Project the standardized data  $Z$  onto the first principal component:  $Z' = Z \cdot v_1$

Where:  $v_1 = \begin{bmatrix} 0.707 \\ 0.707 \end{bmatrix}$

And:  $Z = \begin{bmatrix} -1.34 & -1.34 \\ -0.45 & -0.27 \\ 0.45 & 0.27 \\ 1.34 & 1.34 \end{bmatrix}$

Compute:  $Z' = \begin{bmatrix} -1.34 & -1.34 \\ -0.45 & -0.27 \\ 0.45 & 0.27 \\ 1.34 & 1.34 \end{bmatrix} \begin{bmatrix} 0.707 \\ 0.707 \end{bmatrix} = \begin{bmatrix} -1.90 \\ -0.51 \\ 0.51 \\ 1.90 \end{bmatrix}$

### Final Result

The transformed data (projected onto PC1) is:  $Z' = \begin{bmatrix} -1.90 \\ -0.51 \\ 0.51 \\ 1.90 \end{bmatrix}$

This is the dataset reduced to **one dimension** while retaining 97.5% of the variance.



### Step 3: Intuition for the Covariance Matrix

The **covariance matrix** ( $\Sigma$ ) summarizes the relationships between features. Here's a breakdown of its diagonal and off-diagonal elements:

#### Diagonal Elements ( $\Sigma_{i,i}$ )

- **What They Represent:** Variance of each feature.
  - Variance measures the spread of a feature's values around its mean.
  - Since the features were standardized, the variances are 1 (unit variance).

For our example:  $\Sigma_{1,1} = \text{Var}(Z_1) = 1$ ,  $\Sigma_{2,2} = \text{Var}(Z_2) = 1$

#### Off-Diagonal Elements ( $\Sigma_{i,j}$ )

- **What They Represent:** Covariance between pairs of features ( $Z_1$  and  $Z_2$ ).
  - Positive covariance indicates that both features tend to increase together.
  - Negative covariance indicates that one feature increases while the other decreases.

For our example:

$$\Sigma_{1,2} = \text{Cov}(Z_1, Z_2) = 0.8, \quad \Sigma_{2,1} = \text{Cov}(Z_2, Z_1) = 0.8$$


GDP SOCIAL LIFE FREEDOM GENEROSITY CORRUPTION


FR  1.1 1.1 1.3 0.3 -0.9 -0.9

DE  1.2 0.8 1.1 0.7 0.2 -1.5

IN  -0.6 -1.8 -0.6 0.9 0.7 0.3

MA  -0.5 -2.2 0.2 -0.2 -1.5 0.4

TR  0.7 0.1 0.3 -1.9 -0.8 0.3

US  1.4 0.9 0.5 0.4 0.8 -0.2

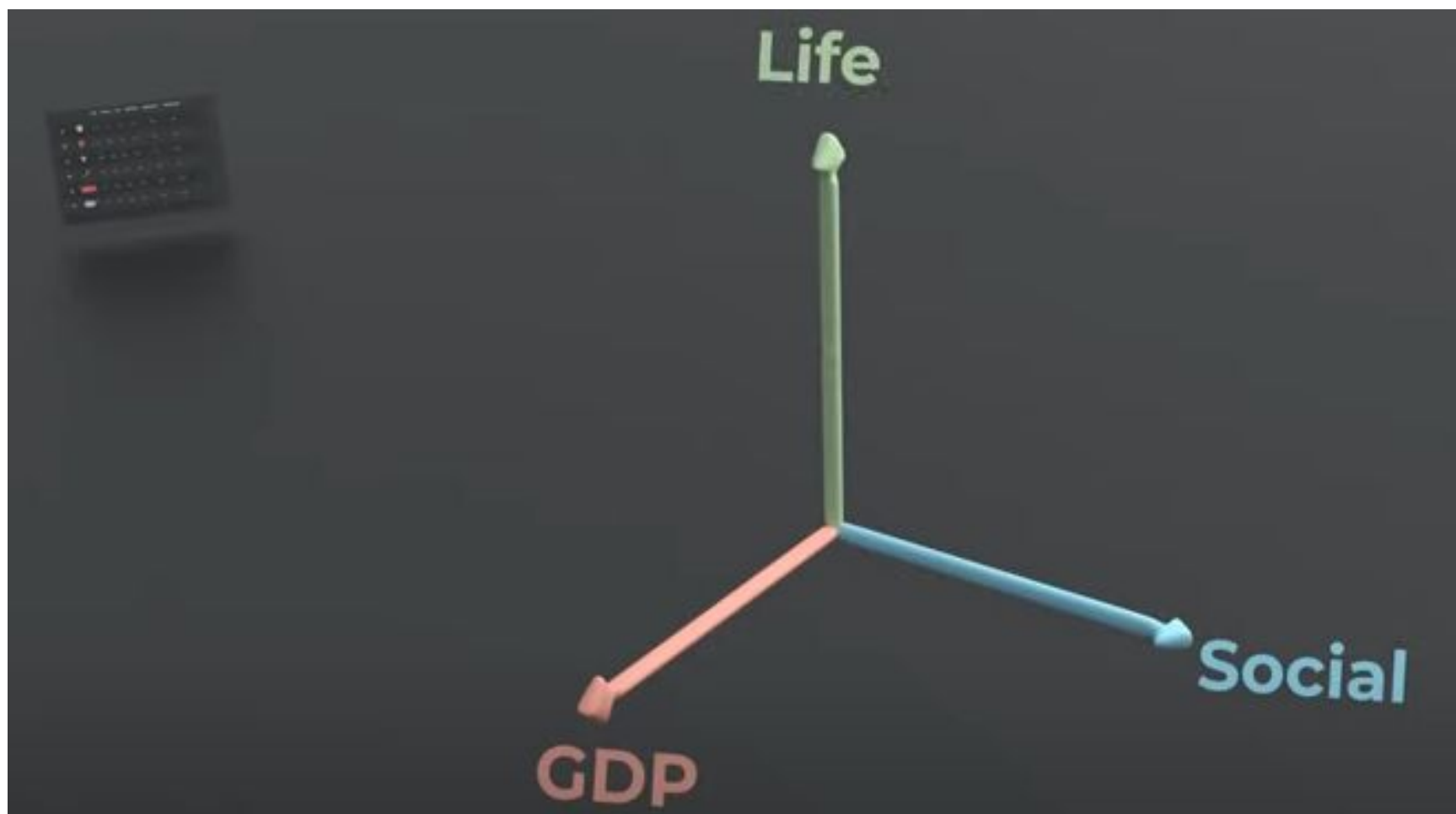
--

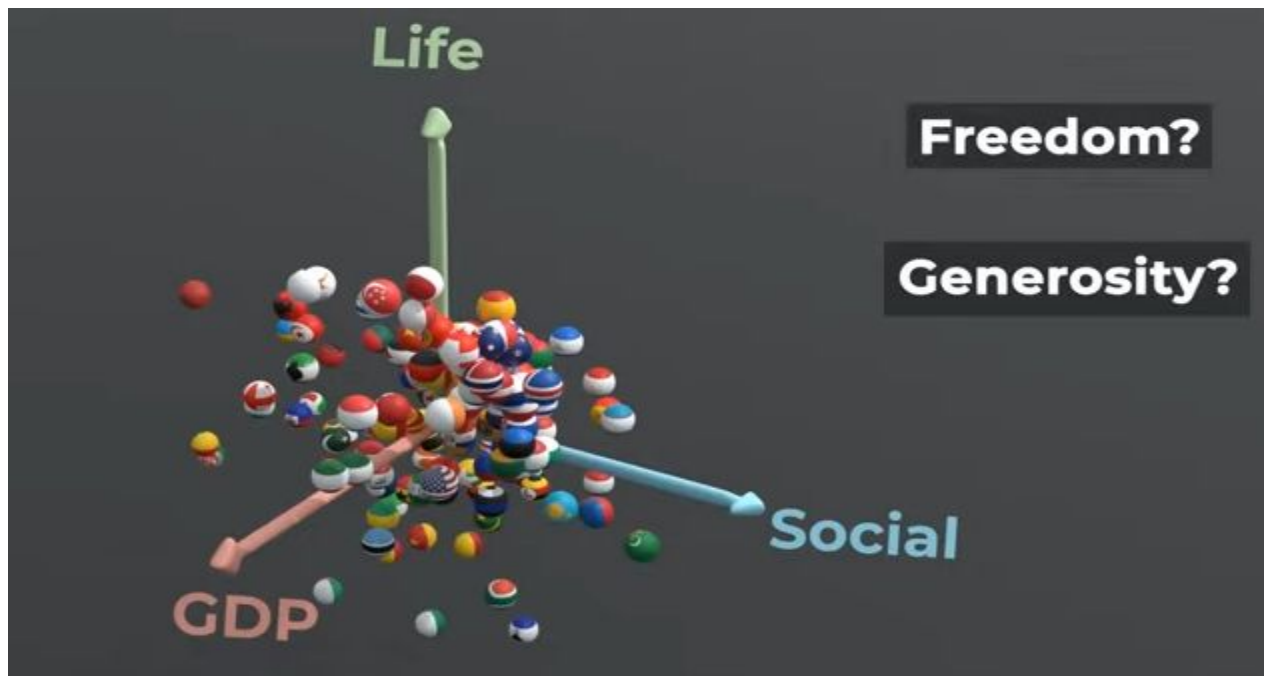
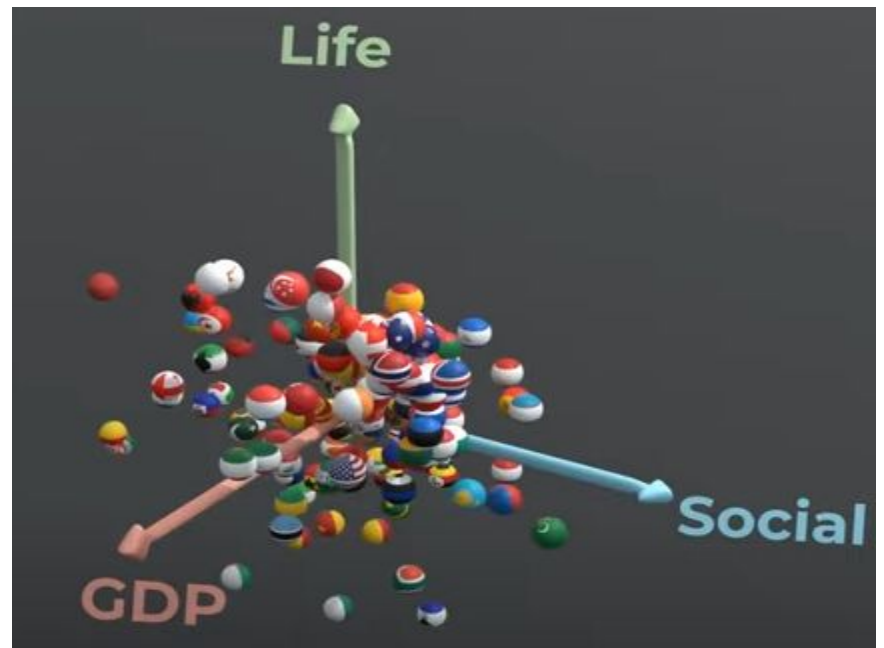
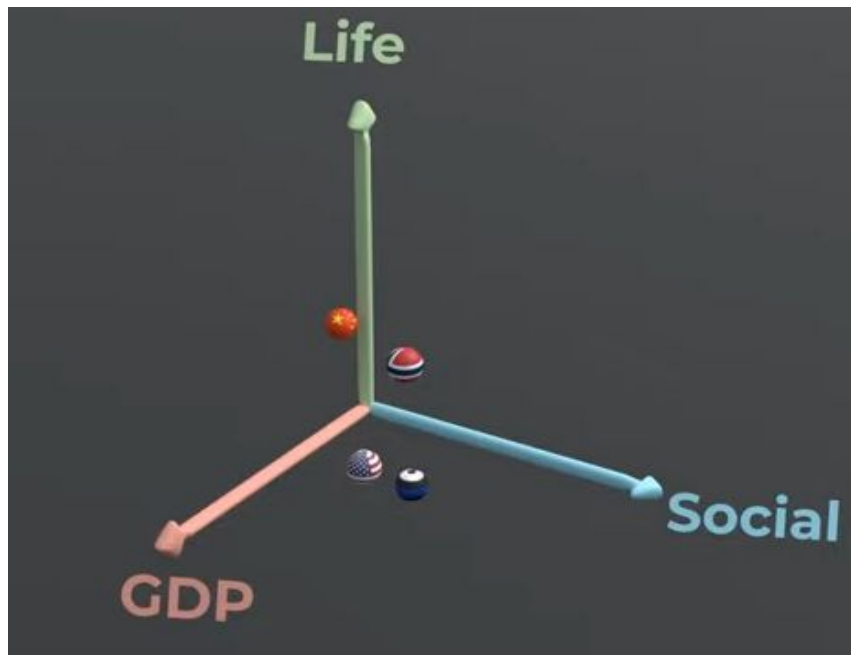
Analyze

Understand

Visualize

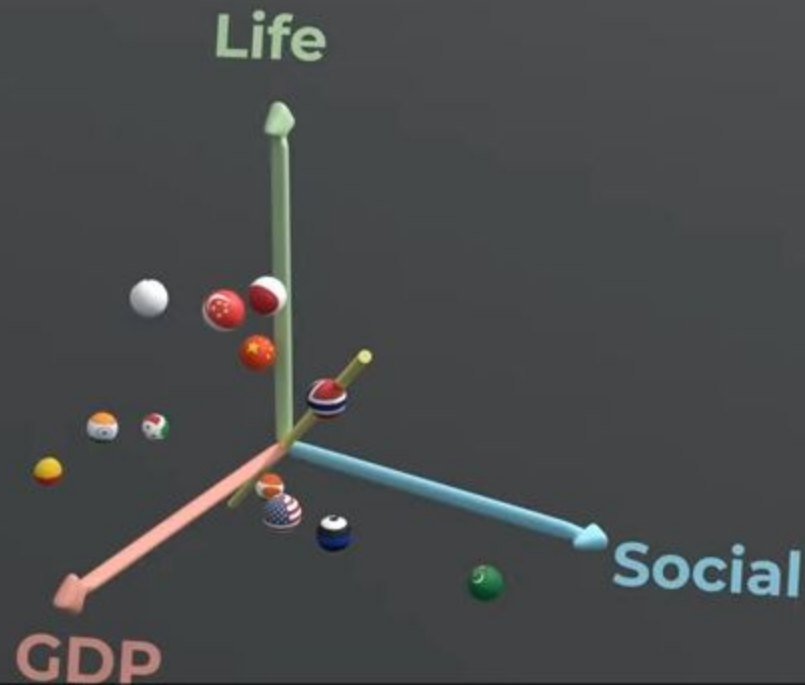




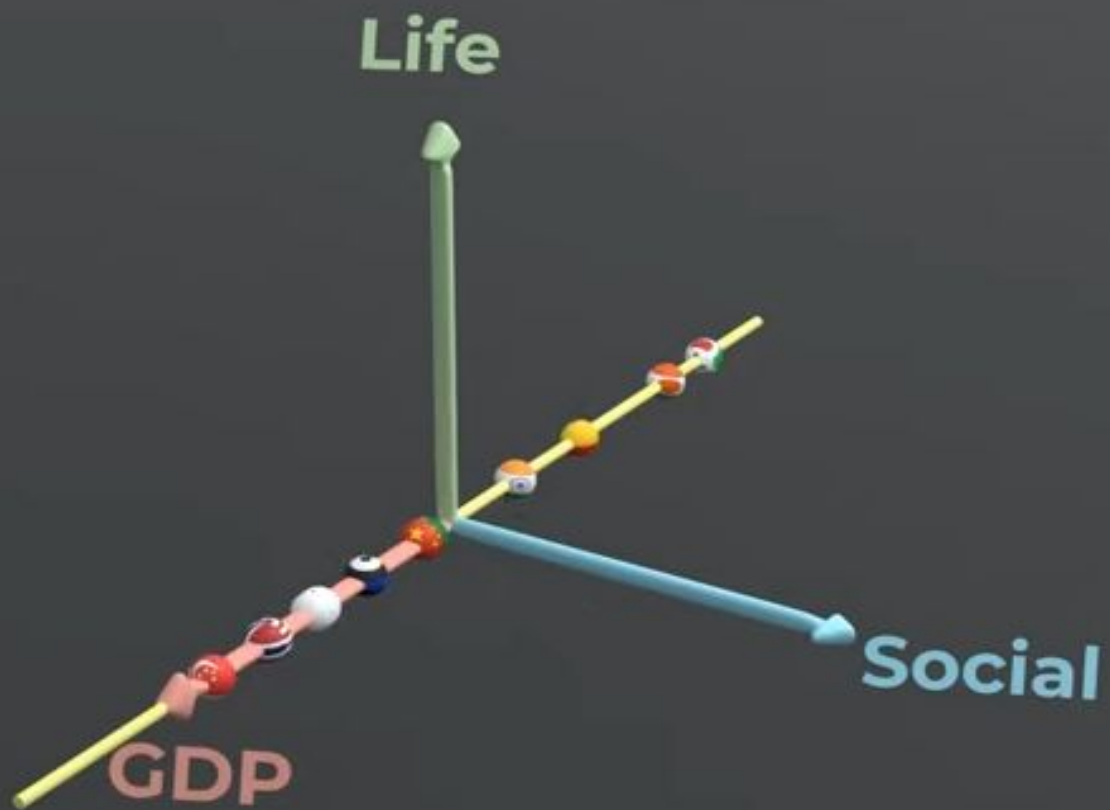




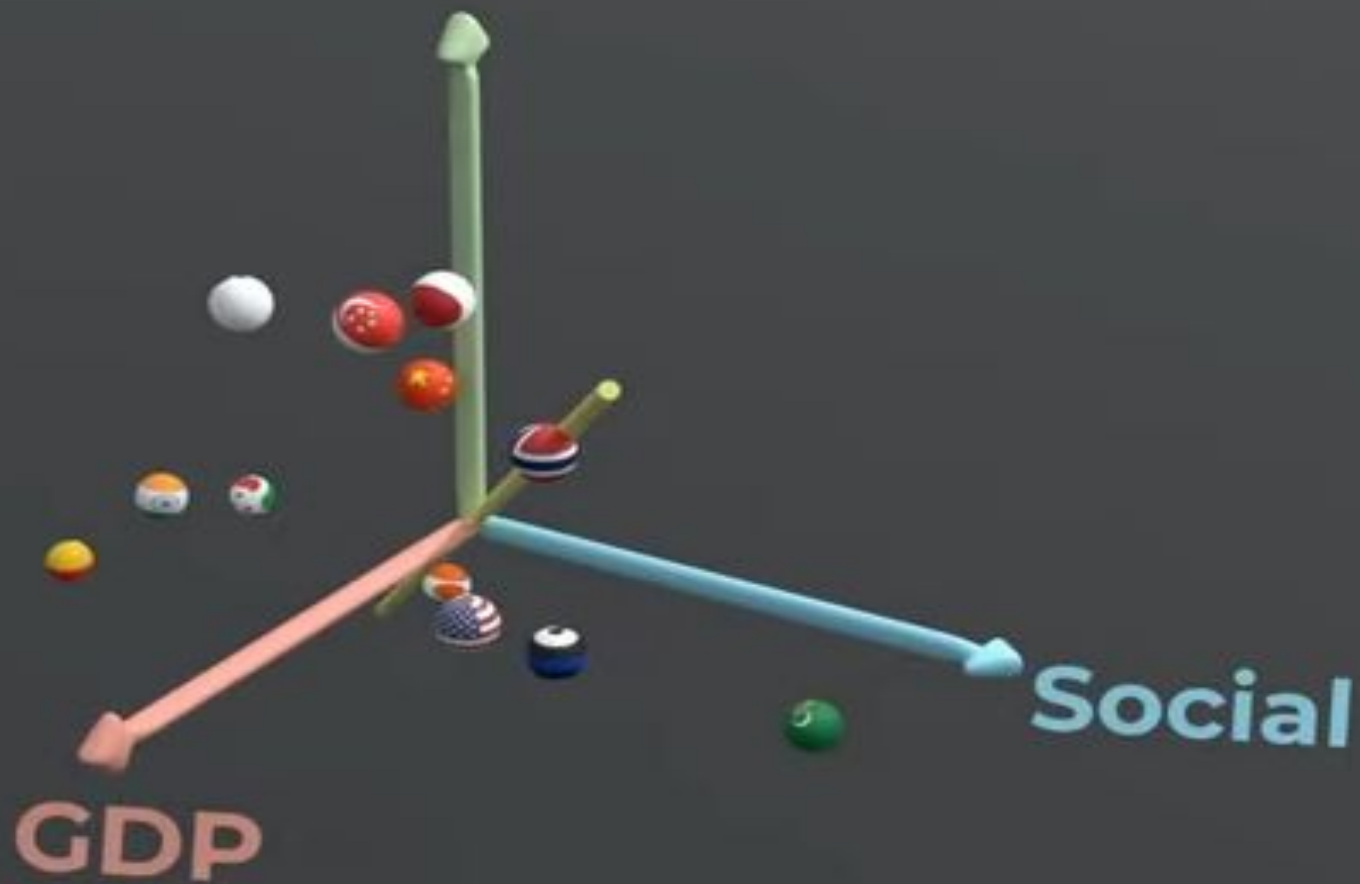
# Principal Components



**Arrange these points **in a line**  
and **preserve** as much info as possible?**



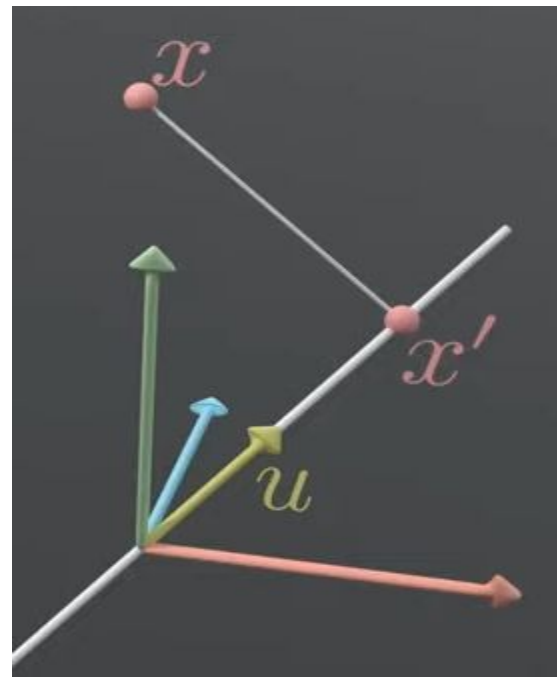
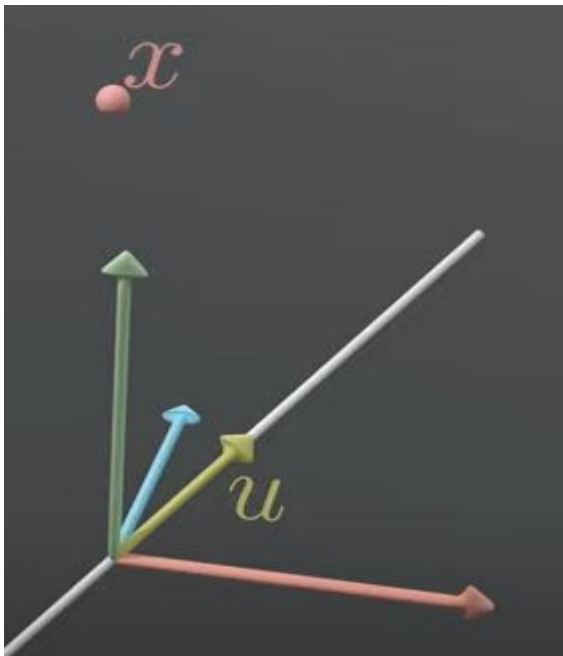
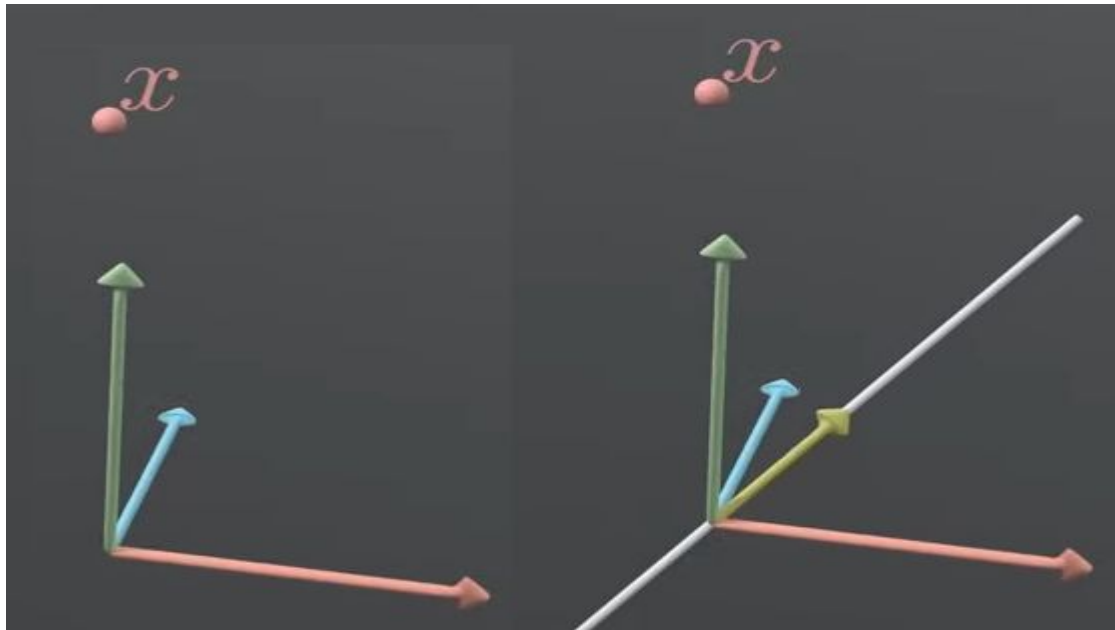
Life



Social

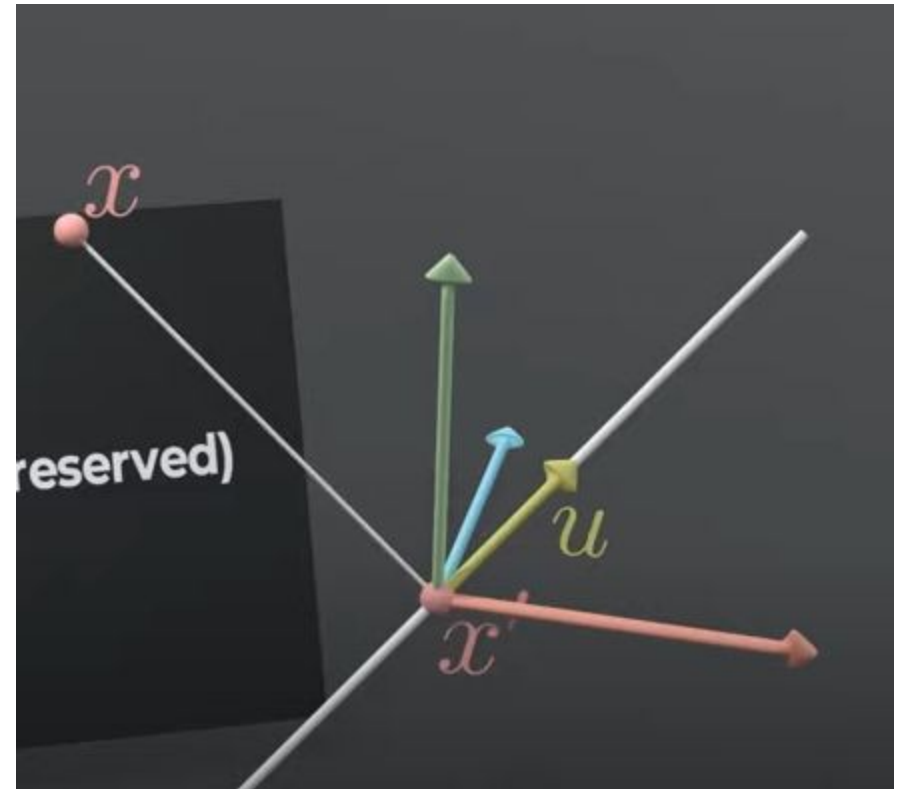
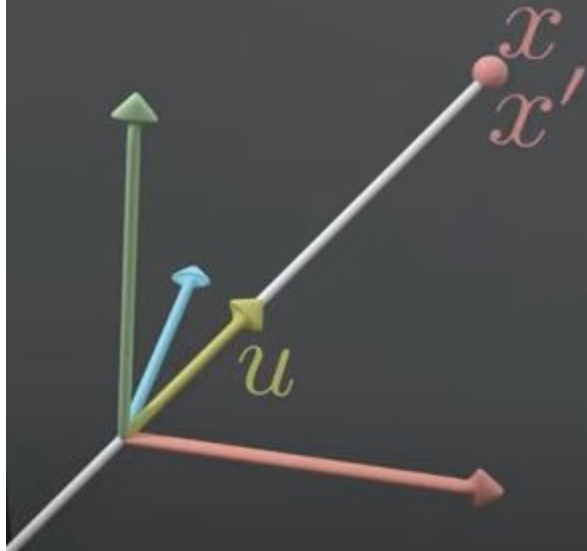
GDP

projecting a data point  $x$  onto a principal component  $u$ .



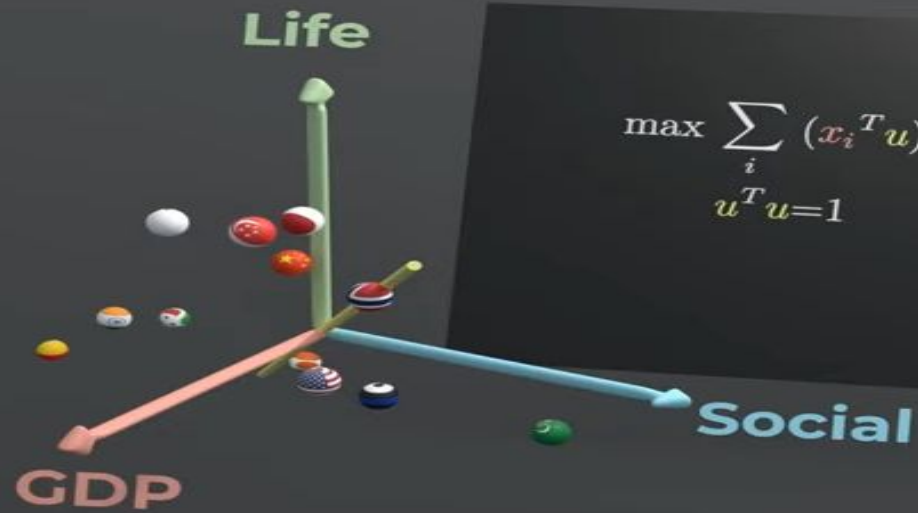
$$x' = (x^T u) u$$

$$(x^T u)^2 \text{ (info preserved)}$$



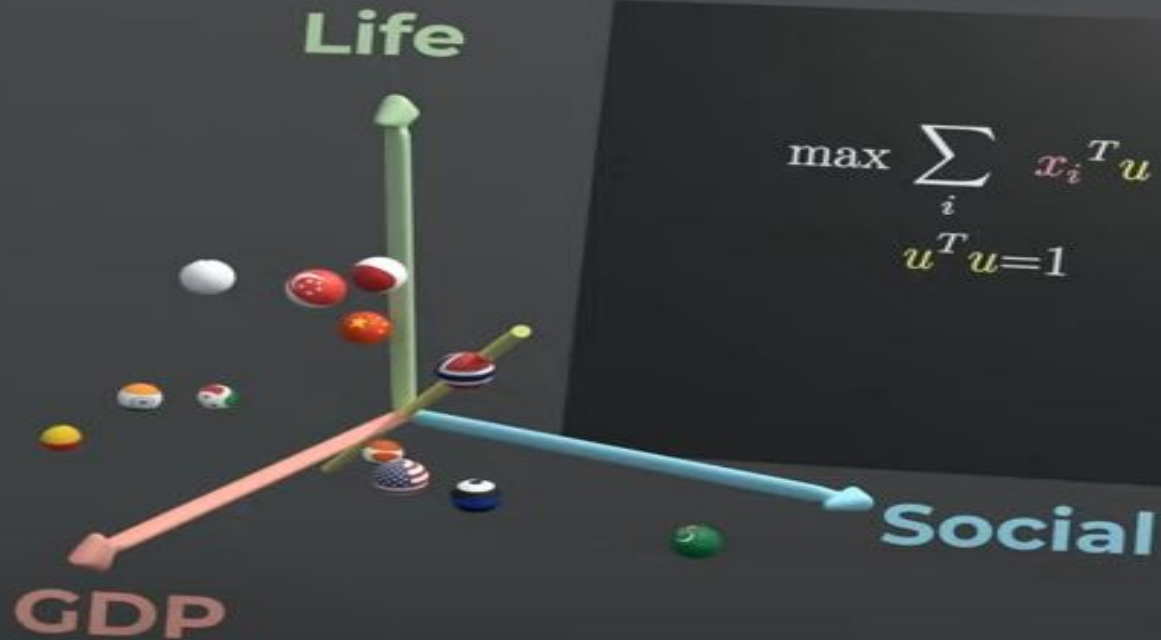


**Optimization**

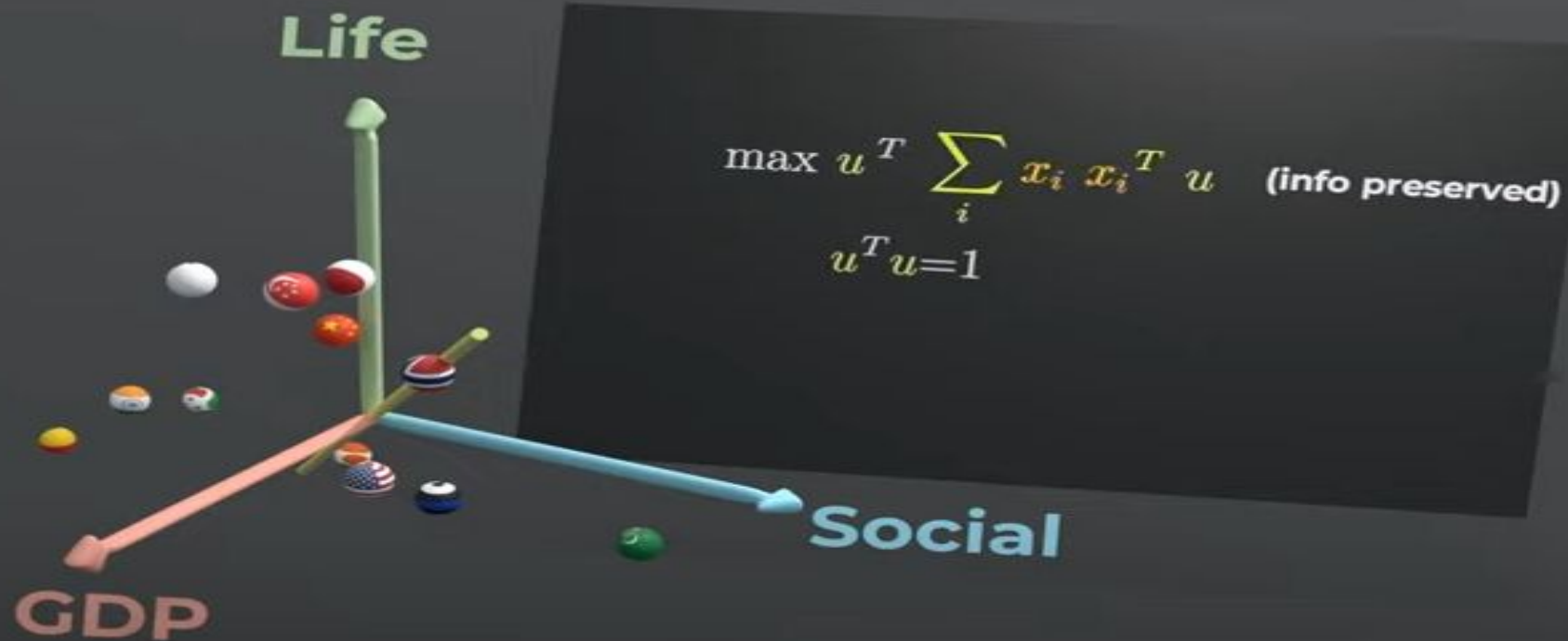
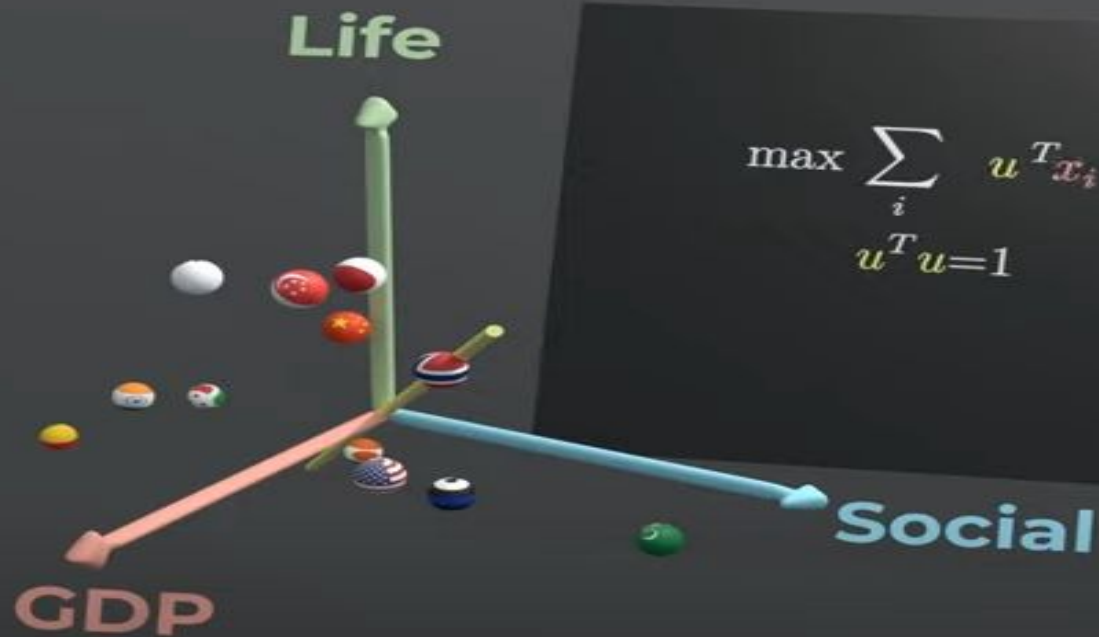


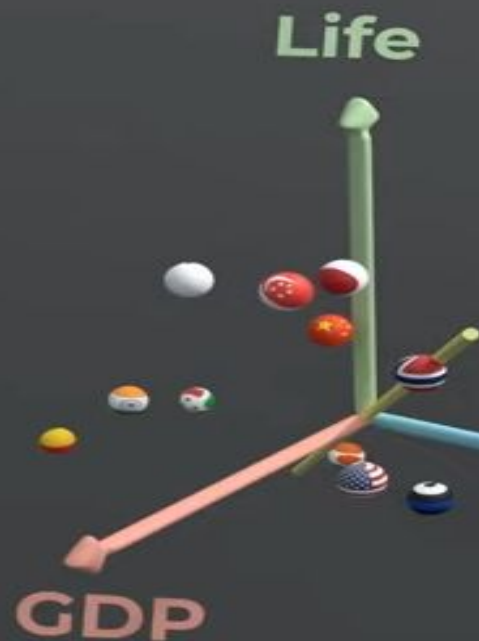
$$\max \sum_i (x_i^T u)^2 \quad (\text{info preserved})$$
$$u^T u = 1$$

**Lagrange  
Multipliers**



$$\max \sum_i x_i^T u \quad x_i^T u \quad (\text{info preserved})$$
$$u^T u = 1$$





$$\max \quad u^T C u \quad (\text{info preserved})$$

$$u^T u = 1$$

$$C = \frac{1}{n} \sum_i x_i x_i^T \quad (\text{cov matrix})$$

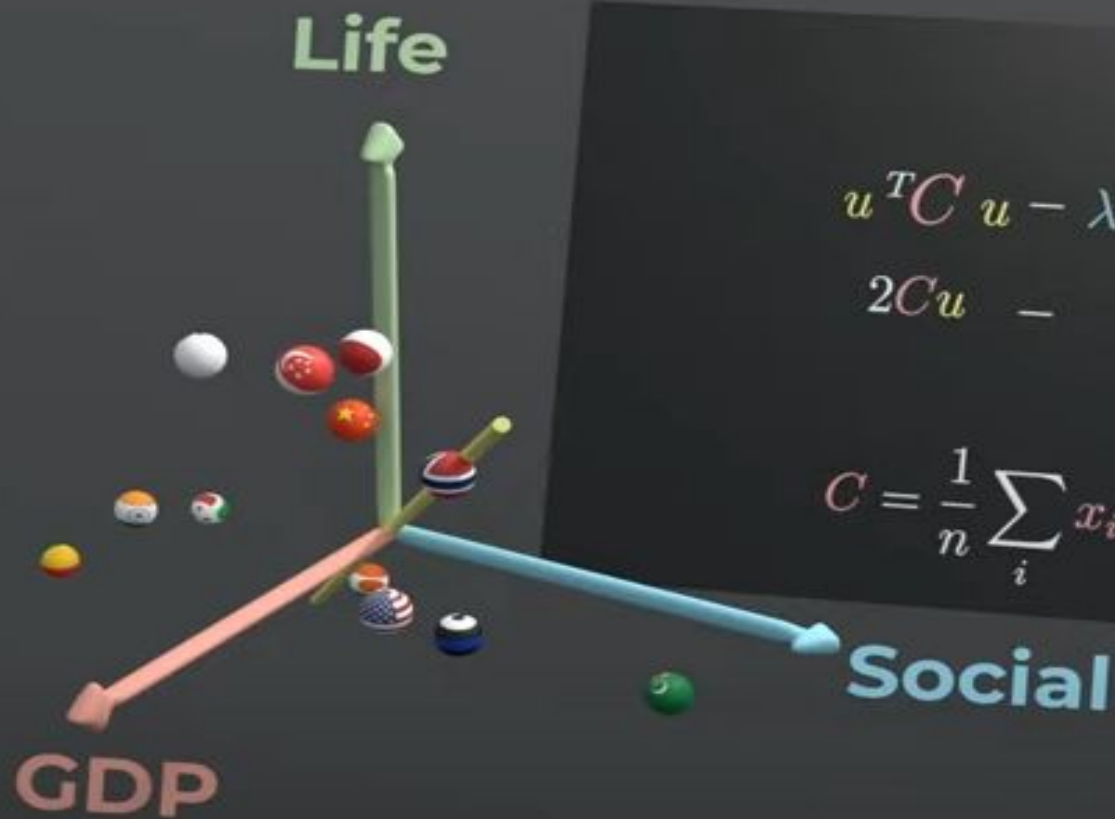
**Social**



$$u^T C u - \lambda (u^T u - 1)$$

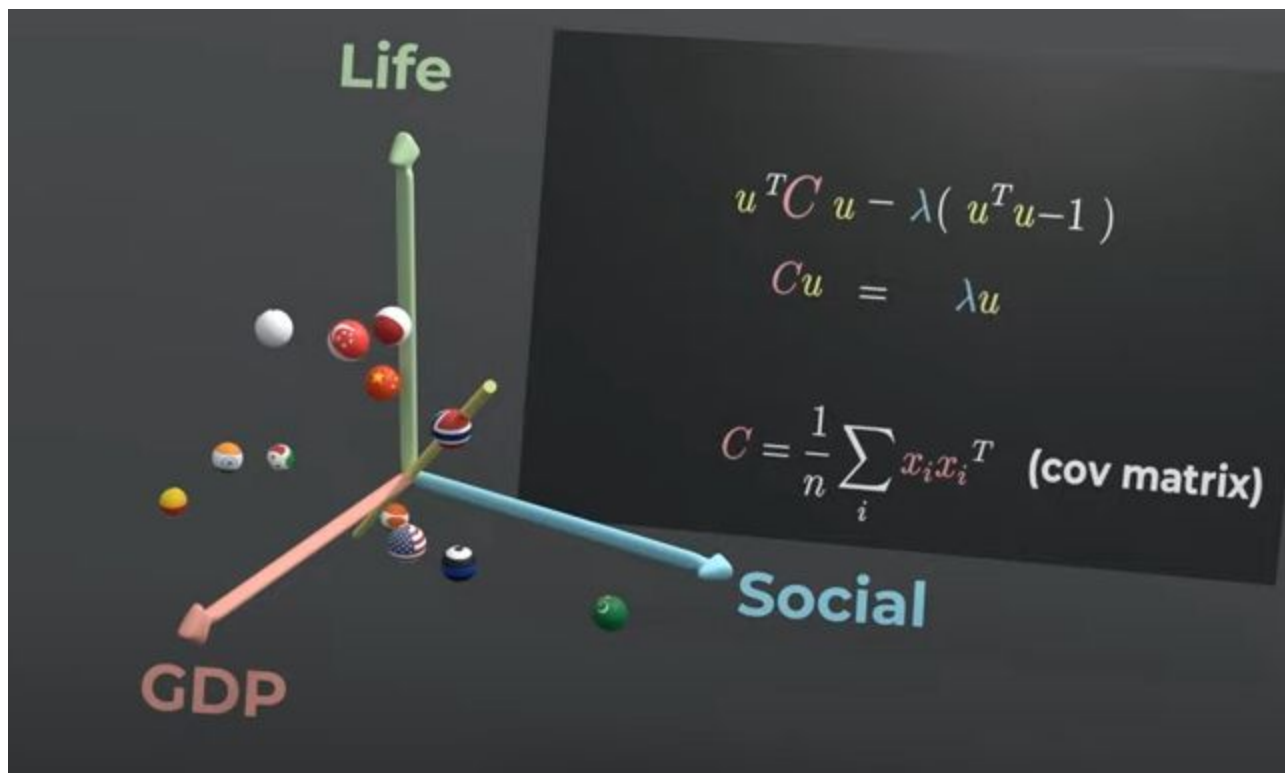
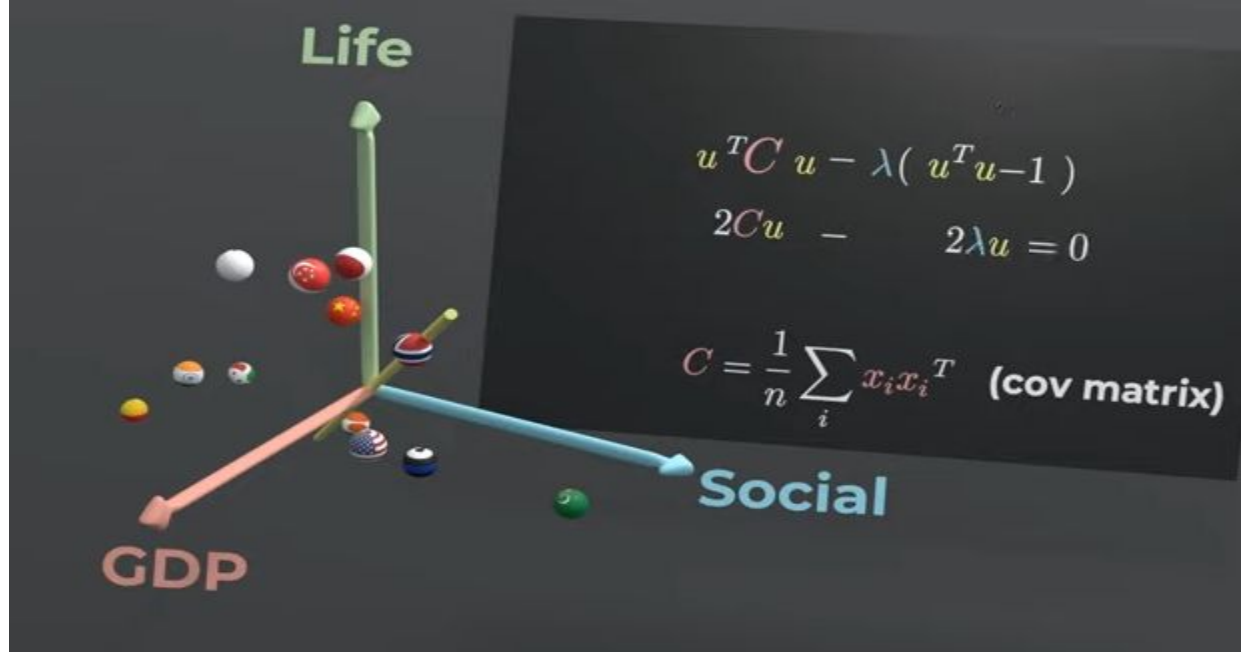
$$C = \frac{1}{n} \sum_i x_i x_i^T \quad (\text{cov matrix})$$

**Social**



$$\nabla$$
$$u^T C u - \lambda (u^T u - 1)$$
$$2Cu - 2\lambda u$$

$$C = \frac{1}{n} \sum_i x_i x_i^T \quad (\text{cov matrix})$$



## Eigenvalue Decomposition

For the covariance matrix  $\Sigma$ :

$$\Sigma v = \lambda v$$

Where:

- $v$ : Eigenvector (principal component direction).
- $\lambda$ : Eigenvalue (variance explained by the eigenvector).

### What It Means:

- The eigenvectors of  $\Sigma$  define the directions in the feature space (principal components) along which the data has the **maximum variance**.
- The eigenvalues represent the **amount of variance** captured along these directions.

### **3. Role in Dimensionality Reduction**

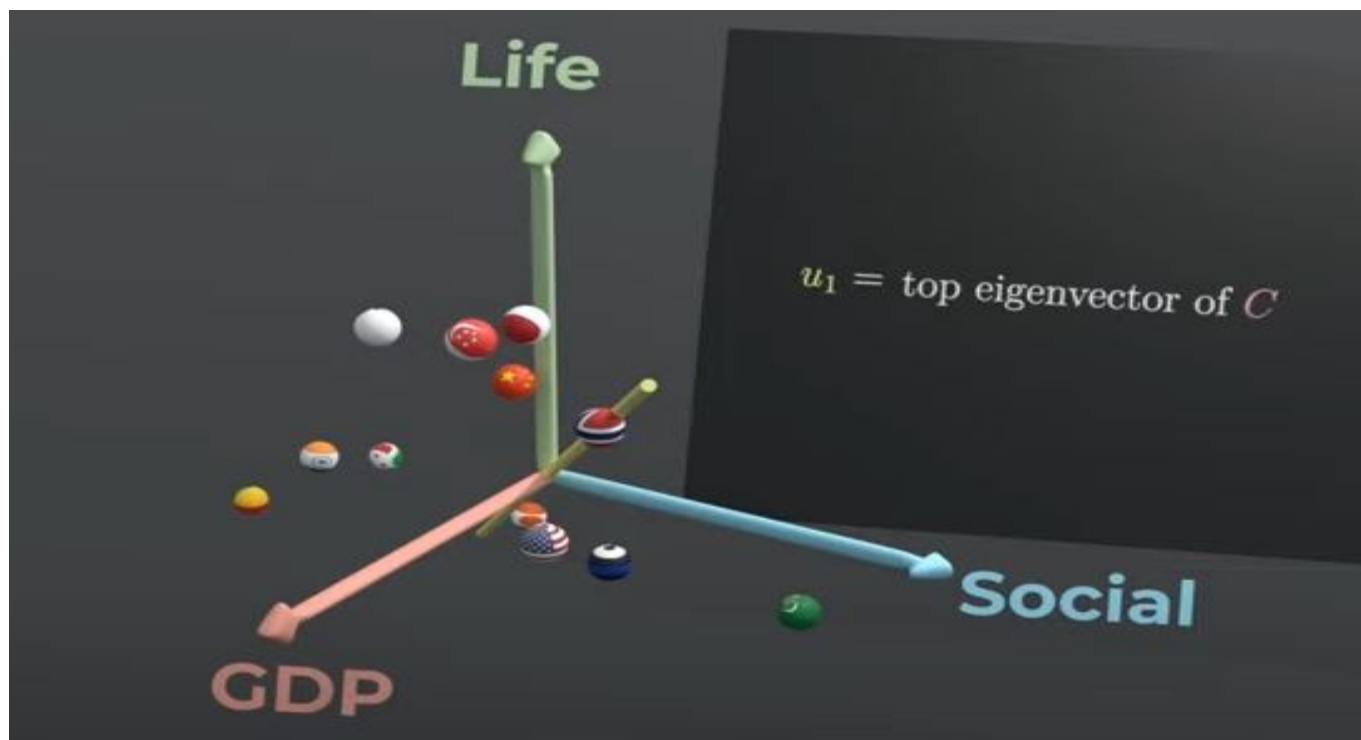
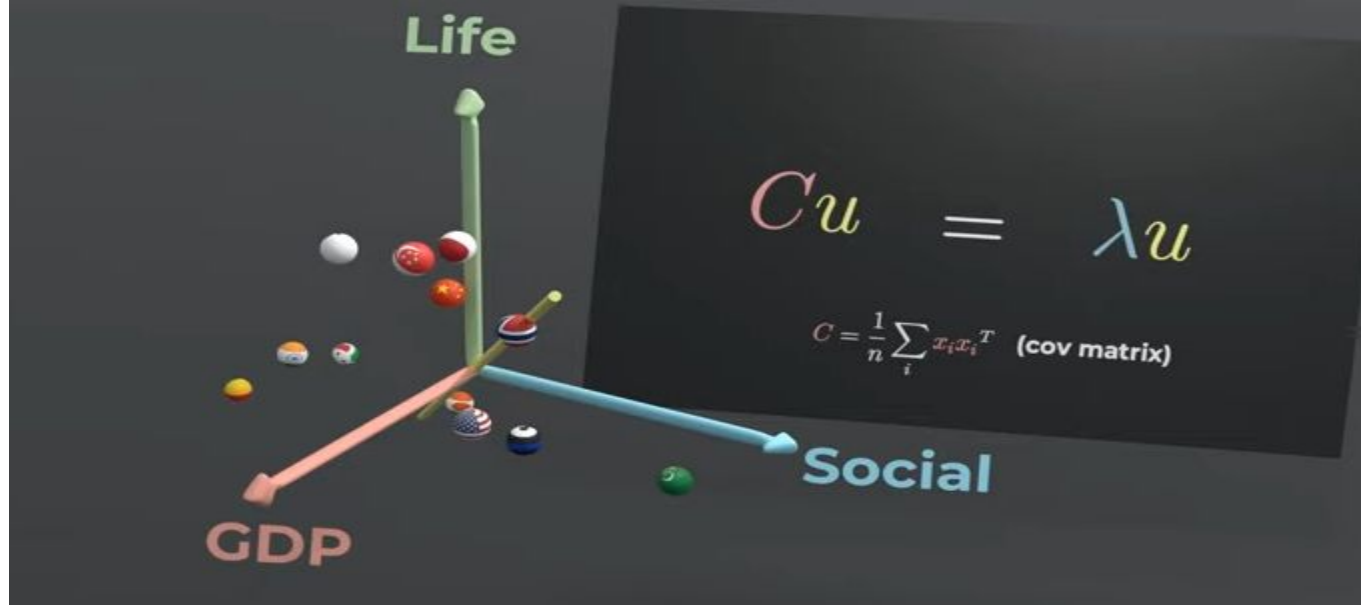
The covariance matrix helps PCA achieve dimensionality reduction by:

#### **Highlighting Redundant Features:**

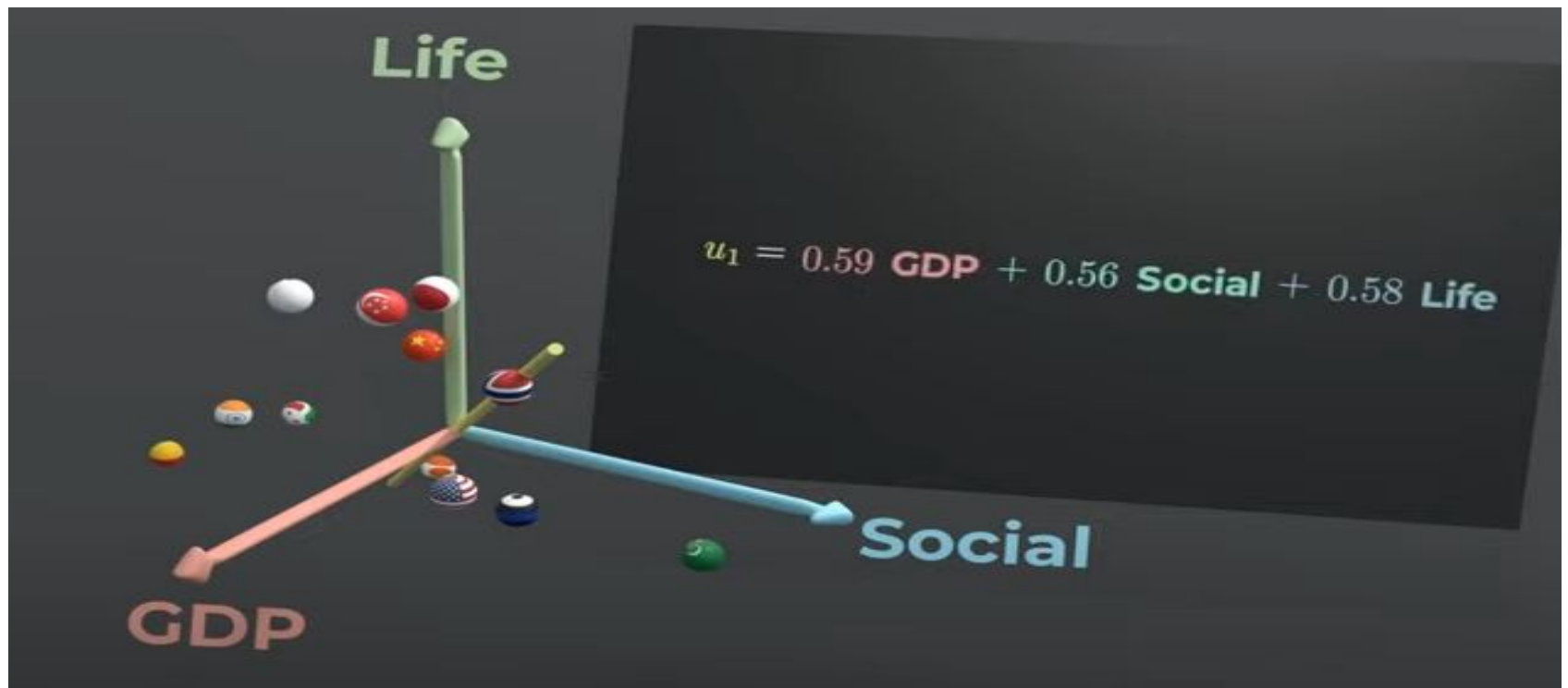
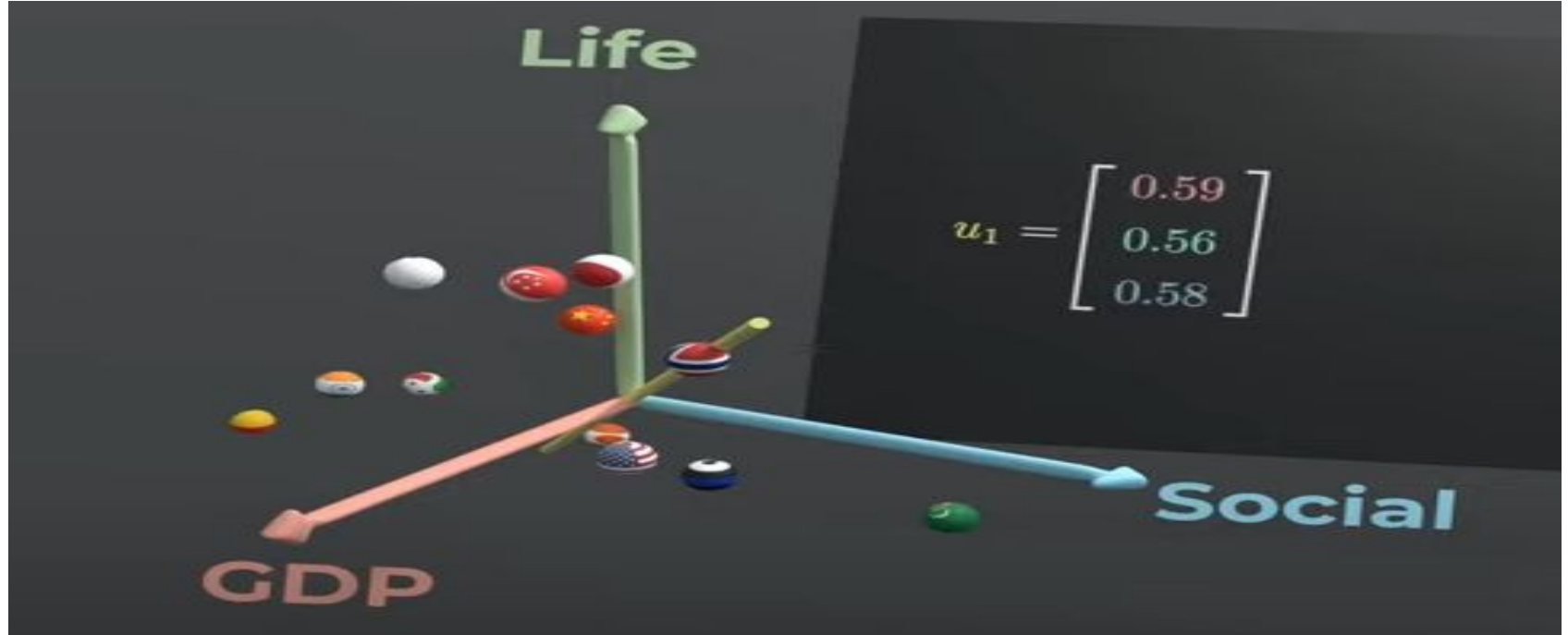
- Features with high covariance indicate redundancy, as they contain similar information.
- PCA combines such features into a single principal component, reducing dimensionality.

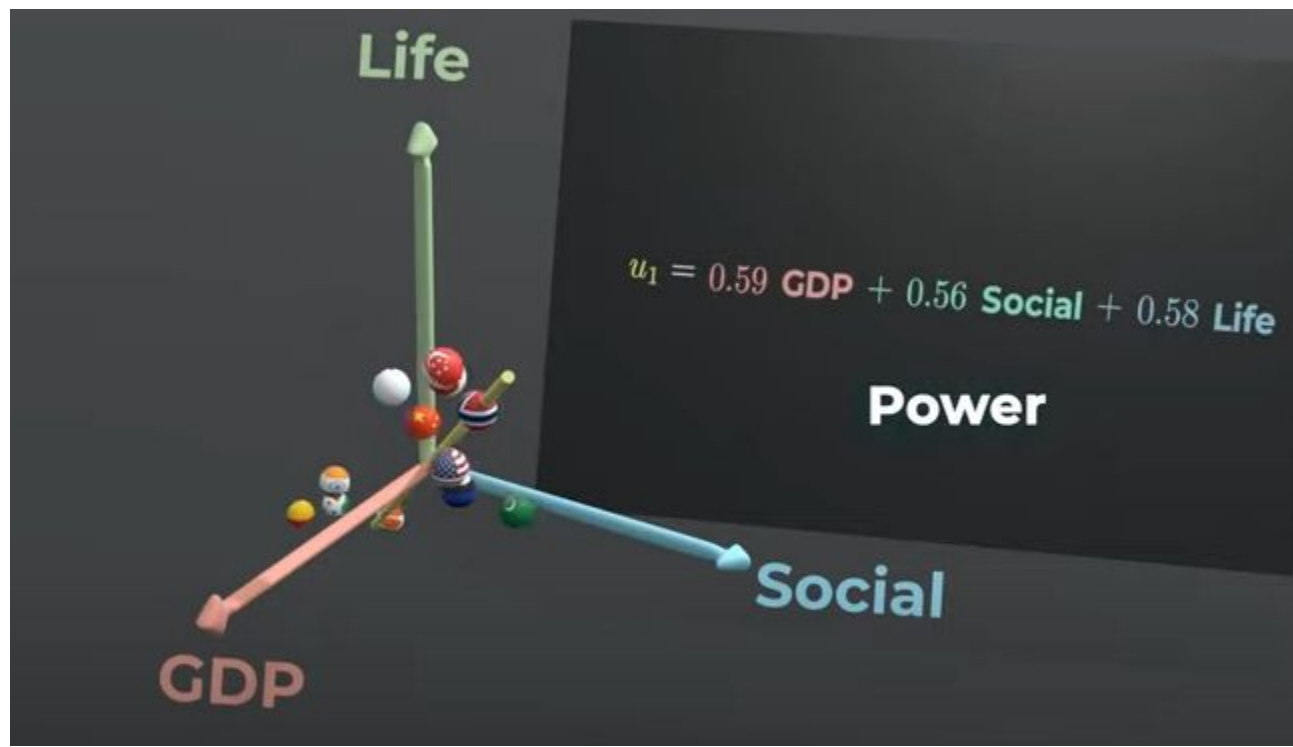
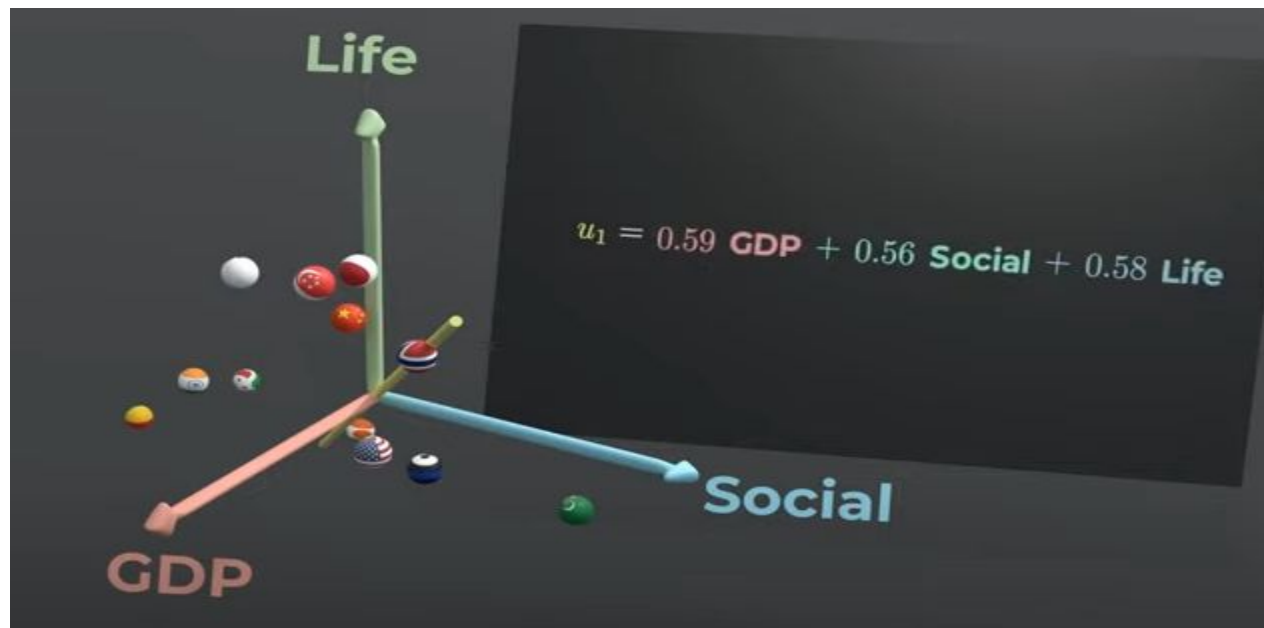
#### **Selecting Key Components:**

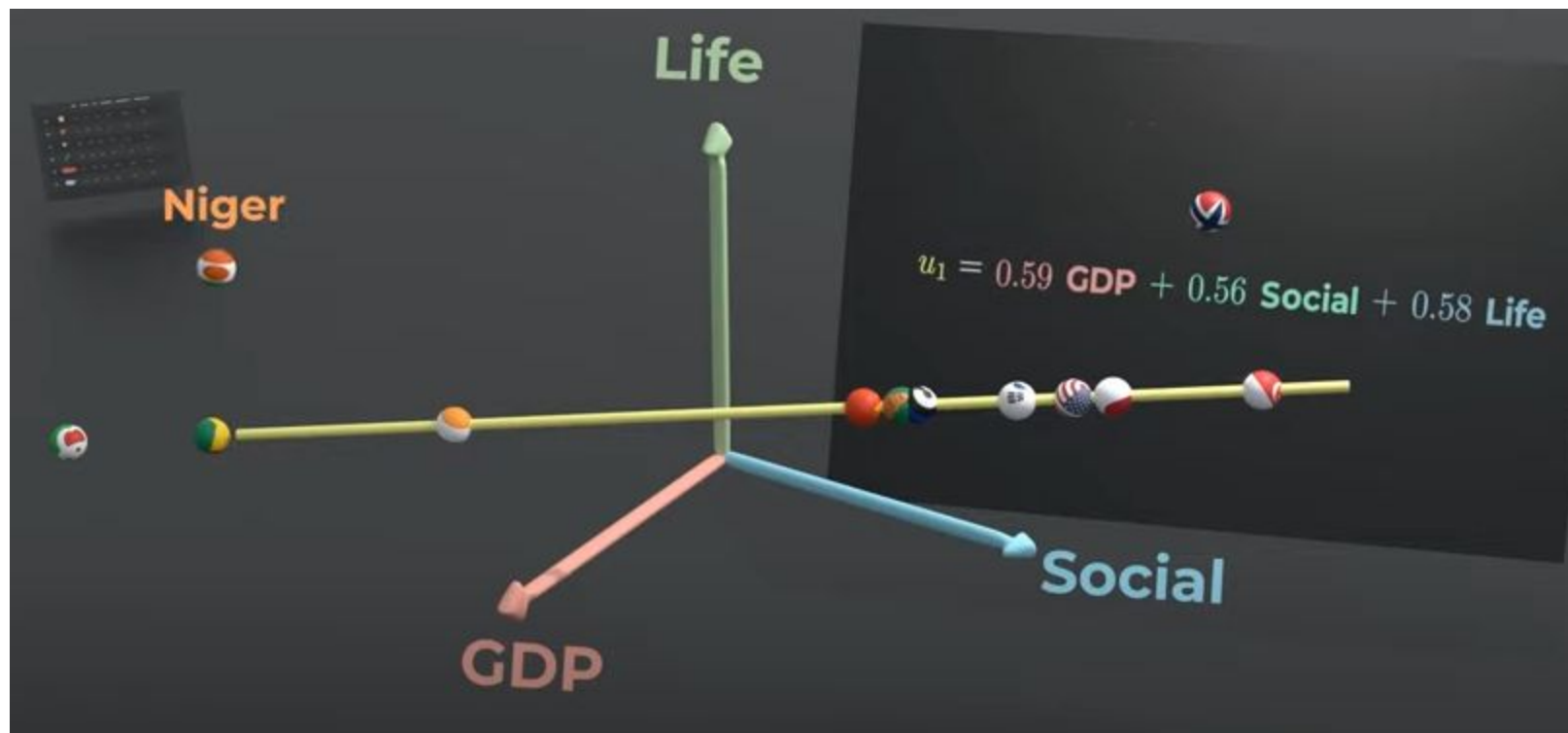
- The eigenvalues (derived from the covariance matrix) determine the importance of each principal component.
- PCA ranks components by variance and retains only the most important one



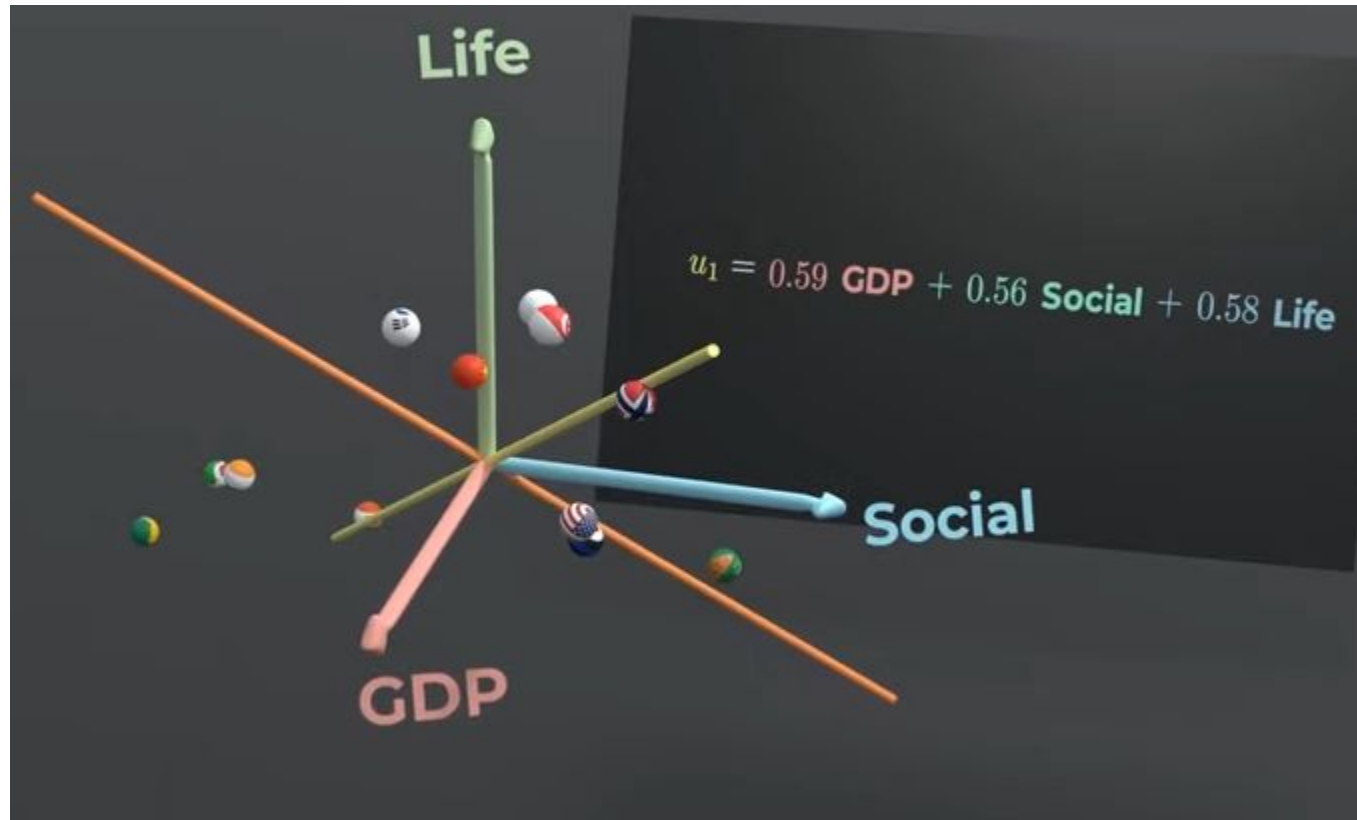




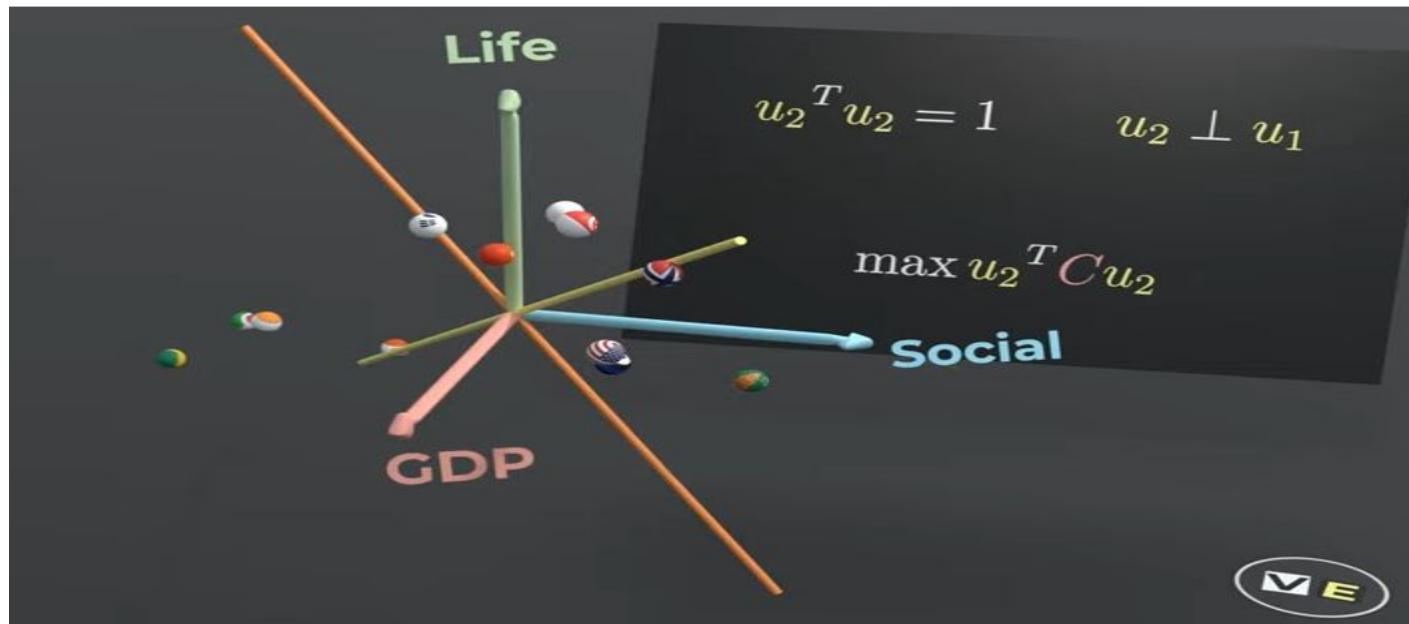
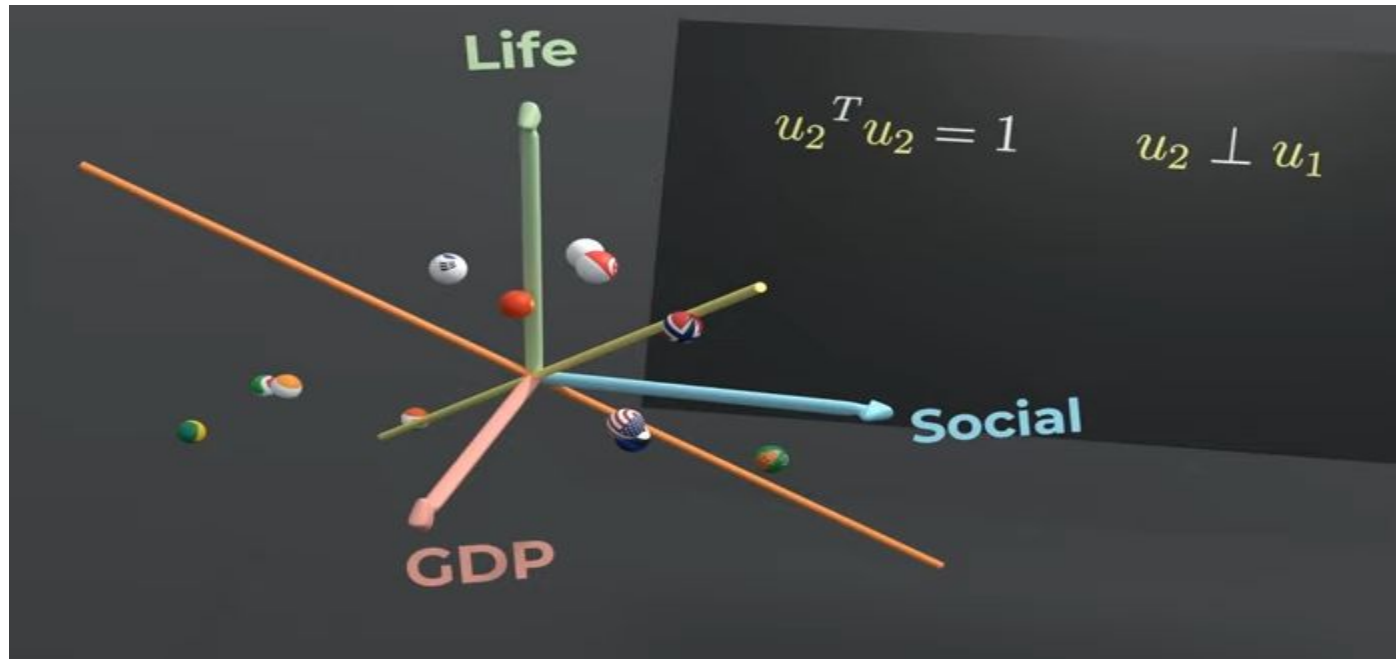




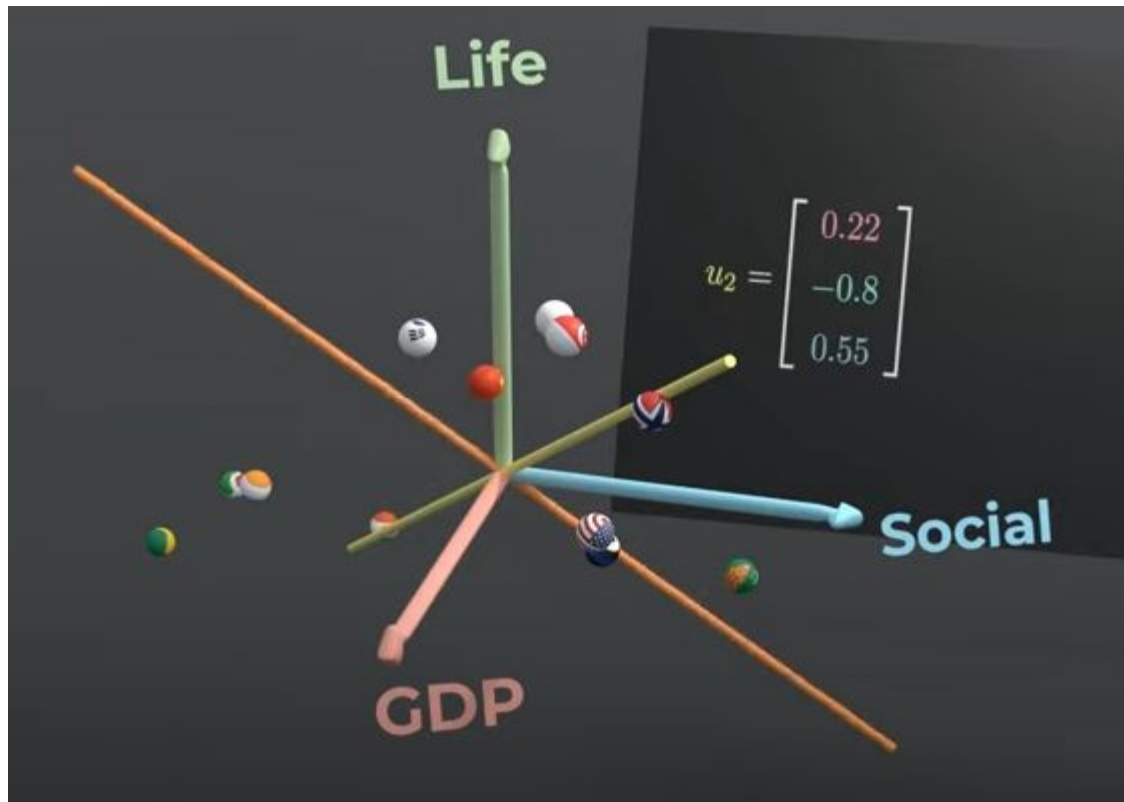
## Second Component

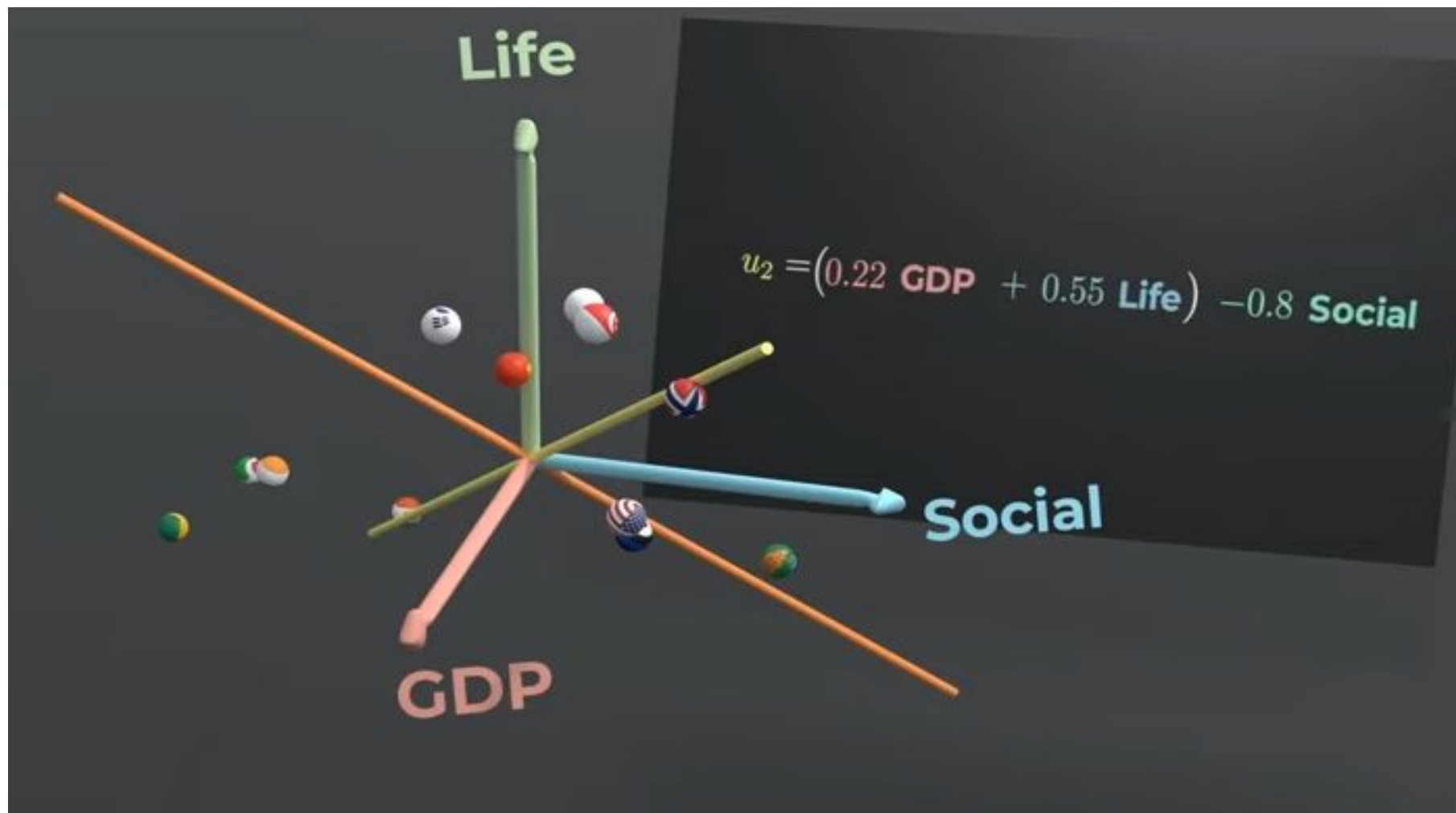


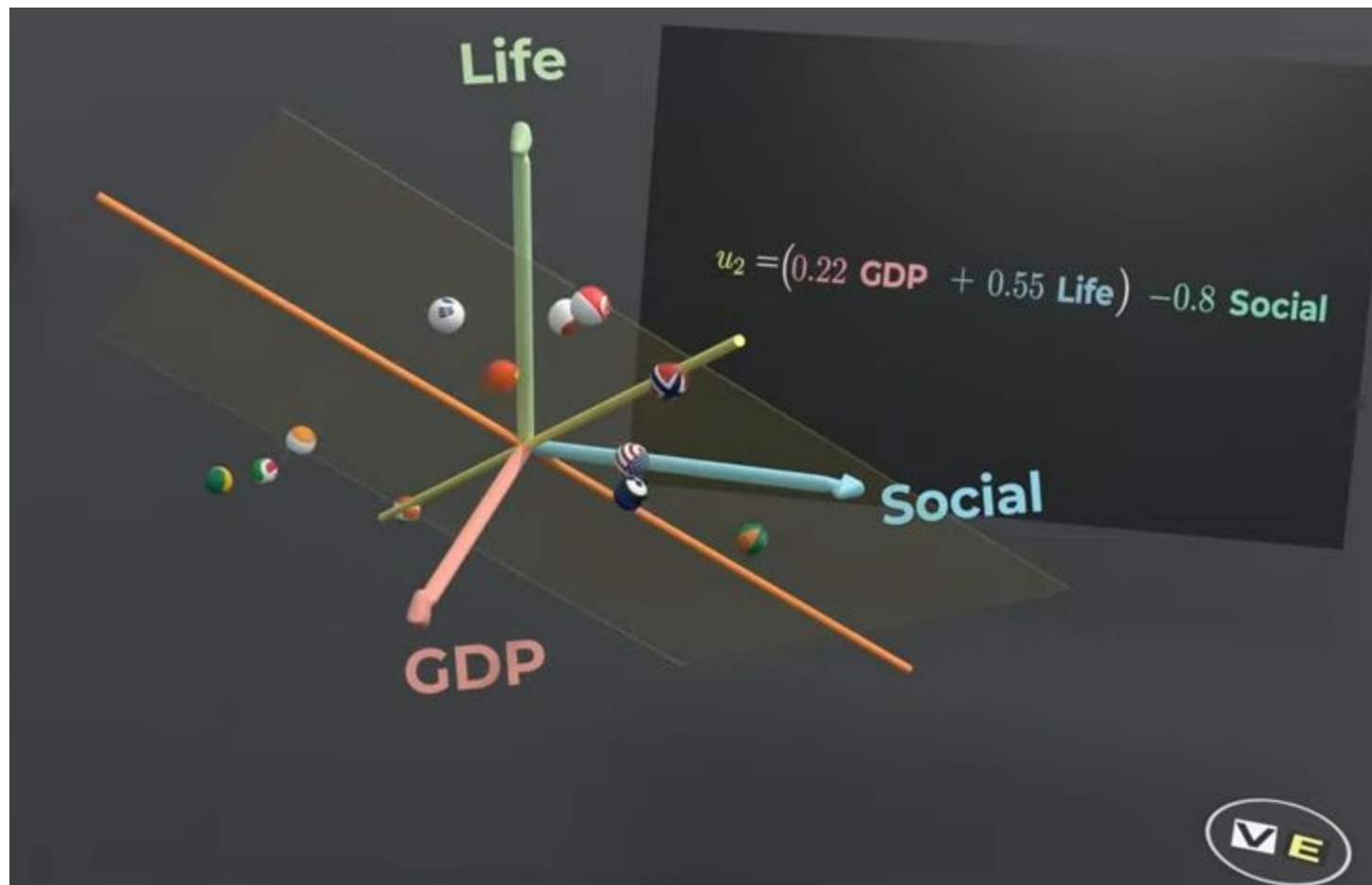
## Second Component



## Second Component

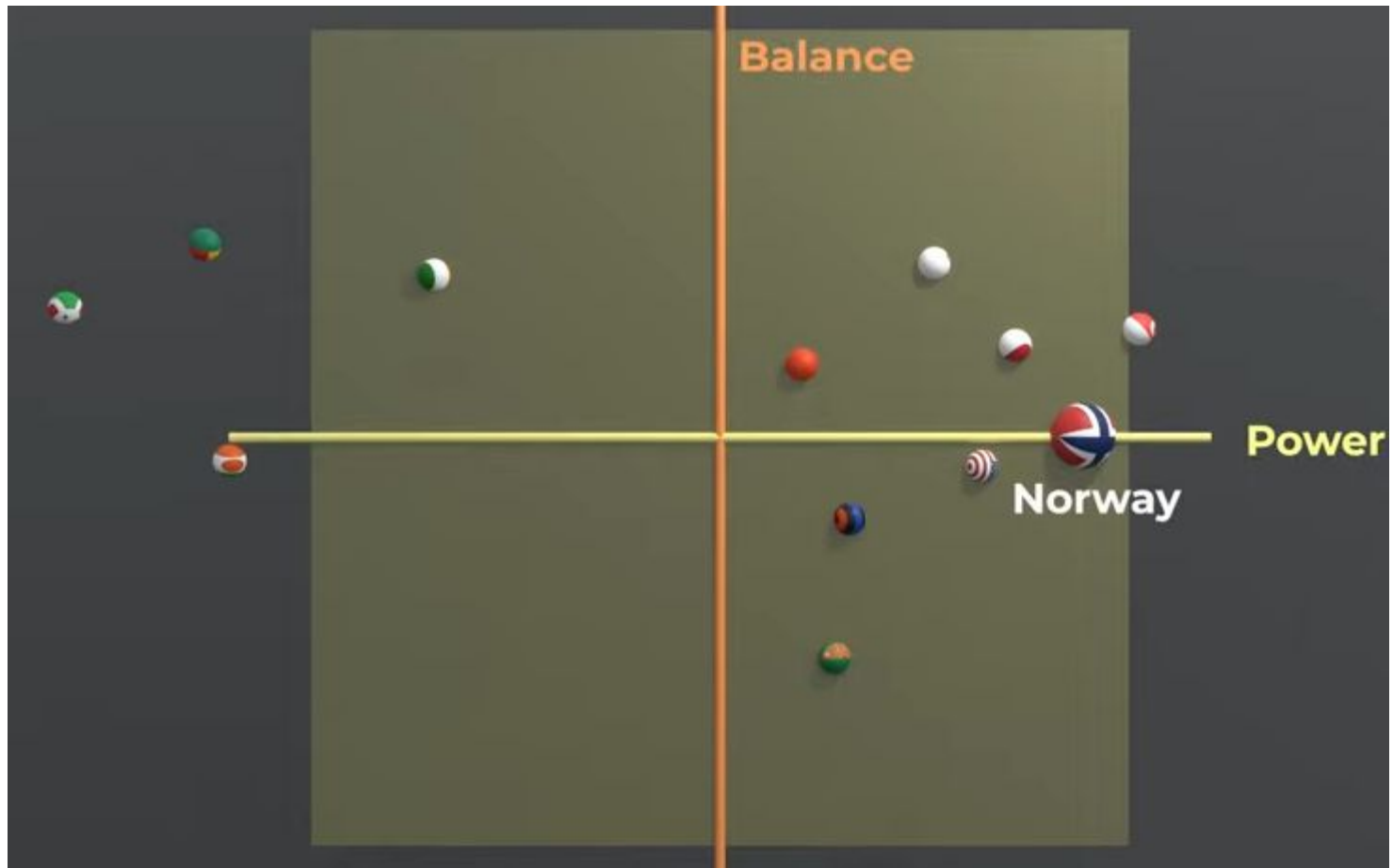






**PCA's Components** **=** **Eigenvectors of C**





**C**

**Eigenvectors**  
(PCA's components)

$u_1 \perp u_2 \perp \dots \perp u_n$

**Eigenvalues**

$\lambda_1 \quad \lambda_2 \quad \dots \quad \lambda_n$

Component	1 (Power)	2 (Balance)	3 (Other)
Eigenvalue	2.58	0.29	0.13

Component	1 (Power)	2 (Balance)	3 (Other)
Eigenvalue	2.58 85%	0.29 10%	0.13 5%

# Introduction to Solving Linear Equations with Matrices:

- Linear equations appear frequently in data science and machine learning, often as systems of equations that describe relationships between variables.
- Matrices provide a compact and efficient way to represent and solve such systems.

## What Are Linear Equations?

A **system of linear equations** consists of multiple equations, where each equation is linear in its variables. For example:

$$\begin{aligned}2x + y &= 5 \\4x - y &= 2\end{aligned}$$

In matrix form, this is represented as:  $AX = B$

Where:

- $A$  is the **coefficient matrix**:  $\begin{bmatrix} 2 & 1 \\ 4 & -1 \end{bmatrix}$
- $X$  is the **variable vector**:  $\begin{bmatrix} x \\ y \end{bmatrix}$
- $B$  is the **constant vector**:  $\begin{bmatrix} 5 \\ 2 \end{bmatrix}$

## 2. Inverse of Matrix $A$

To solve  $AX = B$ , we compute:  $X = A^{-1}B$

Where  $A^{-1}$  is the **inverse of matrix  $A$** .

The formula for the inverse of a  $2 \times 2$  matrix is:  $A^{-1} = \frac{1}{\text{Det}(A)} \cdot \text{Adj}(A)$

### Step 2.1: Determinant of $A$

The determinant ( $\text{Det}(A)$ ) of  $A = \begin{bmatrix} 2 & 1 \\ 4 & -1 \end{bmatrix}$  is computed as:

$$\text{Det}(A) = (2)(-1) - (1)(4)$$

$$\text{Det}(A) = -2 - 4 = -6$$

### Step 2.2: Adjoint of $A$

The adjoint ( $\text{Adj}(A)$ ) of a  $2 \times 2$  matrix is computed by:

1. Swapping the diagonal elements of  $A$ .
2. Changing the sign of the off-diagonal elements.

$$\text{For } A = \begin{bmatrix} 2 & 1 \\ 4 & -1 \end{bmatrix}: \quad \text{Adj}(A) = \begin{bmatrix} -1 & -1 \\ -4 & 2 \end{bmatrix}$$

### Step 2.3: Inverse of $A$

Substitute  $\text{Det}(A) = -6$  and  $\text{Adj}(A) = \begin{bmatrix} -1 & -1 \\ -4 & 2 \end{bmatrix}$  into the formula:

$$A^{-1} = \frac{1}{-6} \cdot \begin{bmatrix} -1 & -1 \\ -4 & 2 \end{bmatrix}$$

$$A^{-1} = \begin{bmatrix} \frac{1}{6} & \frac{1}{6} \\ \frac{2}{3} & -\frac{1}{3} \end{bmatrix}$$

### 3. Solve for $X$

Now compute:

$$X = A^{-1}B$$

Substitute  $A^{-1} = \begin{bmatrix} \frac{1}{6} & \frac{1}{6} \\ \frac{2}{3} & -\frac{1}{3} \end{bmatrix}$  and  $B = \begin{bmatrix} 5 \\ 2 \end{bmatrix}$ :

#### Step 3.1: Matrix Multiplication

Multiply  $A^{-1}$  and  $B$ :

$$X = \begin{bmatrix} \frac{1}{6} & \frac{1}{6} \\ \frac{2}{3} & -\frac{1}{3} \end{bmatrix} \cdot \begin{bmatrix} 5 \\ 2 \end{bmatrix}$$

$$X = \begin{bmatrix} \frac{1}{6} & \frac{1}{6} \\ \frac{2}{3} & -\frac{1}{3} \end{bmatrix} \cdot \begin{bmatrix} 5 \\ 2 \end{bmatrix}$$

Compute each element of  $X$ :

1. First element ( $x$ ):  $x = \frac{1}{6}(5) + \frac{1}{6}(2) = \frac{5}{6} + \frac{2}{6} = \frac{7}{6}$        $X = \begin{bmatrix} \frac{7}{6} \\ \frac{8}{3} \end{bmatrix}$

2. Second element ( $y$ ):  $y = \frac{2}{3}(5) - \frac{1}{3}(2) = \frac{10}{3} - \frac{2}{3} = \frac{8}{3}$

# Solving Matrix Equations

## Matrix Representation

The system can be written in **matrix form** as:

$$AX = B$$

Where:

- $A = \begin{bmatrix} 1 & 0 & 1 \\ 1 & -1 & 1 \\ 1 & 1 & -1 \end{bmatrix}$  (Coefficient matrix),
- $X = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$  (Unknown vector),
- $B = \begin{bmatrix} 1 \\ 3 \\ 2 \end{bmatrix}$  (Constant vector).

$$\begin{bmatrix} 1 & 0 & 1 \\ 1 & -1 & 1 \\ 1 & 1 & -1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 1 \\ 3 \\ 2 \end{bmatrix}$$



Expand along the first row:

$$\text{Det}(A) = 1 \cdot \begin{vmatrix} -1 & 1 \\ 1 & -1 \end{vmatrix} - 0 \cdot \begin{vmatrix} 1 & 1 \\ 1 & -1 \end{vmatrix} + 1 \cdot \begin{vmatrix} 1 & -1 \\ 1 & 1 \end{vmatrix}$$

$$\text{Simplify: } \begin{vmatrix} -1 & 1 \\ 1 & -1 \end{vmatrix} = (-1)(-1) - (1)(1) = 1 - 1 = 0$$

$$\begin{vmatrix} 1 & -1 \\ 1 & 1 \end{vmatrix} = (1)(1) - (-1)(1) = 1 + 1 = 2$$

$$\text{Det}(A) = 1 \cdot 0 + 1 \cdot 2 = 2$$

## Step 2: Compute the Adjoint of $A$

The adjoint ( $\text{Adj}(A)$ ) is the transpose of the cofactor matrix.

### Cofactor Matrix

The cofactor of each element is calculated by removing the corresponding row and column and computing the determinant of the minor matrix.

The cofactor of each element is calculated by removing the corresponding row and column and computing the determinant of the minor matrix.

$$\text{Cofactor Matrix} = \begin{bmatrix} \begin{vmatrix} -1 & 1 \\ 1 & -1 \end{vmatrix} & \begin{vmatrix} 1 & 1 \\ 1 & -1 \end{vmatrix} & \begin{vmatrix} 1 & -1 \\ 1 & 1 \end{vmatrix} \\ \begin{vmatrix} 0 & 1 \\ 1 & -1 \end{vmatrix} & \begin{vmatrix} 1 & 1 \\ 1 & 1 \end{vmatrix} & \begin{vmatrix} 1 & 0 \\ 1 & 1 \end{vmatrix} \\ \begin{vmatrix} 0 & 1 \\ -1 & 1 \end{vmatrix} & \begin{vmatrix} 1 & 1 \\ 1 & -1 \end{vmatrix} & \begin{vmatrix} 1 & 0 \\ 1 & -1 \end{vmatrix} \end{bmatrix}$$

Compute each element:

1.  $\text{Cofactor}(1, 1) = 0$ ,  $\text{Cofactor}(1, 2) = 0$ ,  $\text{Cofactor}(1, 3) = 2$
2.  $\text{Cofactor}(2, 1) = 0$ ,  $\text{Cofactor}(2, 2) = -2$ ,  $\text{Cofactor}(2, 3) = 1$
3.  $\text{Cofactor}(3, 1) = 1$ ,  $\text{Cofactor}(3, 2) = 0$ ,  $\text{Cofactor}(3, 3) = -1$

The cofactor matrix is:  $\begin{bmatrix} 0 & 0 & 2 \\ 0 & -2 & 1 \\ 1 & 0 & -1 \end{bmatrix}$

**Adjoint of A**

Transpose the cofactor matrix:  $\text{Adj}(A) = \begin{bmatrix} 0 & 0 & 1 \\ 0 & -2 & 0 \\ 2 & 1 & -1 \end{bmatrix}$

**Step 3: Compute  $A^{-1}$**   $A^{-1} = \frac{1}{\text{Det}(A)} \cdot \text{Adj}(A)$

Substitute  $\text{Det}(A) = 2$  and  $\text{Adj}(A)$ :  $A^{-1} = \frac{1}{2} \cdot \begin{bmatrix} 0 & 0 & 1 \\ 0 & -2 & 0 \\ 2 & 1 & -1 \end{bmatrix}$

$$A^{-1} = \begin{bmatrix} 0 & 0 & \frac{1}{2} \\ 0 & -1 & 0 \\ 1 & \frac{1}{2} & -\frac{1}{2} \end{bmatrix}$$

**Step 4: Solve for  $X$**

Now compute:  $X = A^{-1}B$  Substitute  $B = \begin{bmatrix} 1 \\ 3 \\ 2 \end{bmatrix}$ :

$$X = \begin{bmatrix} 0 & 0 & \frac{1}{2} \\ 0 & -1 & 0 \\ 1 & \frac{1}{2} & -\frac{1}{2} \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 3 \\ 2 \end{bmatrix}$$

Perform the multiplication:

1. First row:  $0 \cdot 1 + 0 \cdot 3 + \frac{1}{2} \cdot 2 = 1$

2. Second row:  $0 \cdot 1 + (-1) \cdot 3 + 0 \cdot 2 = -3$

3. Third row:  $1 \cdot 1 + \frac{1}{2} \cdot 3 + (-\frac{1}{2}) \cdot 2 = 1 + 1.5 - 1 = 1.5$

$$X = \begin{bmatrix} 1 \\ -3 \\ 1.5 \end{bmatrix}$$

## Solve Using Row Reduction (Gaussian Elimination)

$$\begin{bmatrix} 1 & 2 & 3 \\ 0 & 4 & 5 \\ 0 & 0 & 6 \end{bmatrix}$$

- Write the Augmented Matrix: Represent the system in matrix form.
- **Forward Elimination:**
  - Identify the pivot element in each column.
  - Use row operations to create zeros below the pivot.
- **Back Substitution:**
  - Solve for the variables starting from the last row of the transformed matrix.

# Solve Using Row Reduction (Gaussian Elimination)

## What Is a Pivot?

- A pivot is the leading (non-zero) element in a row of a matrix that is used as a reference to eliminate elements below or above it in the same column.

## Pivot Rules:

- The pivot is typically chosen as the first non-zero element in a row.
- In Gaussian elimination, the pivot must be non-zero to ensure valid row operations.
- For numerical stability, the largest absolute value in the column is often chosen as the pivot (partial pivoting).

## Why Do We Eliminate Below the Pivot?

The goal of elimination is to simplify the matrix into row echelon form or reduced row echelon form:

## Row Echelon Form:

- All non-zero rows are above rows of all zeros.
- The pivot of each row is to the right of the pivot in the row above.
- All entries below each pivot are zero.

$$\begin{bmatrix} 1 & 2 & 3 \\ 0 & 4 & 5 \\ 0 & 0 & 6 \end{bmatrix}$$

# Solve Using Row Reduction (Gaussian Elimination)

## Linear Regression: Solving for Coefficients Using the Normal Equation

Linear regression involves finding a relationship between independent variables ( $X$ ) and a dependent variable ( $Y$ ) by minimizing the sum of squared errors (residuals). The **Normal Equation** provides a direct solution for the regression coefficients ( $\beta$ ).

### Mathematical Representation

The linear regression model is:  $Y = X\beta + \epsilon$

Where:

- $Y$  is the dependent variable (target),
- $X$  is the feature matrix (independent variables),

Where:

- $Y$  is the dependent variable (target),
- $X$  is the feature matrix (independent variables),
- $\beta$  is the vector of regression coefficients,
- $\epsilon$  is the error term.

The objective is to minimize the residual sum of squares (RSS):  $RSS = \|Y - X\beta\|^2$

By minimizing RSS, we derive the **Normal Equation**:  $\beta = (X^T X)^{-1} X^T Y$

## Solve Using Row Reduction (Gaussian

Step 1: Write the Augmented Matrix  $\left[ \begin{array}{ccc|c} 1 & 0 & 1 & 1 \\ 1 & -1 & 1 & 3 \\ 1 & 1 & -1 & 2 \end{array} \right]$

Step 2: Perform Row Operations

1. Eliminate below the pivot in column 1:

$$R_2 = R_2 - R_1, \quad R_3 = R_3 - R_1$$

$$R_2 = R_2 - R_1, \quad R_3 = R_3 - R_1$$

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

### What Is a Pivot?

• A pivot is the leading (non-zero) element in a row of a matrix that is used as a reference to eliminate elements below or above it in the same column.

### Pivot Rules:

- The pivot is typically chosen as the first non-zero element in a row.
- In Gaussian elimination, the pivot must be non-zero to ensure valid row operations.
- For numerical stability, the largest absolute value in the column is often chosen as the pivot (partial pivoting).

### Step 3: Back-Substitution

1. From row 3:  $-2z = 3 \implies z = -\frac{3}{2}$

2. From row 2:  $-y = 2 \implies y = -2$

3. From row 1:  $x + z = 1 \implies x - \frac{3}{2} = 1 \implies x = \frac{5}{2}$

$$x = \frac{5}{2}, \quad y = -2, \quad z = -\frac{3}{2}$$



## Example: Linear Regression

### Dataset

Consider the dataset:  $X = \begin{bmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \end{bmatrix}$ ,  $Y = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$

Here,  $X$  has an intercept column (first column) and one feature column (second column).

### Step 2: Augment the Matrix

Create the augmented matrix by appending the identity matrix:

$$[X^T X | I] = \left[ \begin{array}{cc|cc} 3 & 6 & 1 & 0 \\ 6 & 14 & 0 & 1 \end{array} \right]$$

### Step 3: Apply Gaussian Elimination

Perform row operations to reduce  $X^T X$  (left side) to  $I$  (identity matrix):

1. Scale  $R_1$  (Row 1) to make the pivot element 1:  $R_1 \rightarrow \frac{1}{3}R_1$

Result:  $\left[ \begin{array}{cc|cc} 1 & 2 & \frac{1}{3} & 0 \\ 6 & 14 & 0 & 1 \end{array} \right] \quad \left[ \begin{array}{cc|cc} 1 & 2 & \frac{1}{3} & 0 \\ 0 & 2 & -2 & 1 \end{array} \right]$

2. Eliminate the first element in  $R_2$  (Row 2) using  $R_1$ :  $R_2 \rightarrow R_2 - 6R_1$

3. Scale  $R_2$  to make the pivot element 1:  $R_2 \rightarrow \frac{1}{2}R_2$

$$\text{Result: } \left[ \begin{array}{cc|cc} 1 & 2 & \frac{1}{3} & 0 \\ 0 & 1 & -1 & \frac{1}{2} \end{array} \right]$$

4. Eliminate the second element in  $R_1$  using  $R_2$ :

$$R_1 \rightarrow R_1 - 2R_2$$

Result:

$$\left[ \begin{array}{cc|cc} 1 & 0 & \frac{5}{3} & -1 \\ 0 & 1 & -1 & \frac{1}{2} \end{array} \right]$$

The left side is now  $I$ , and the right side is  $(X^T X)^{-1}$ :

$$(X^T X)^{-1} = \begin{bmatrix} \frac{5}{3} & -1 \\ -1 & \frac{1}{2} \end{bmatrix}$$

Result:  $\left[ \begin{array}{cc|cc} 1 & 0 & \frac{5}{3} & -1 \\ 0 & 1 & -1 & \frac{1}{2} \end{array} \right]$

The left side is now  $I$ , and the right side is  $(X^T X)^{-1}$ :  $(X^T X)^{-1} = \begin{bmatrix} \frac{5}{3} & -1 \\ -1 & \frac{1}{2} \end{bmatrix}$

#### Step 4: Compute $\beta$

Now calculate  $\beta = (X^T X)^{-1} X^T Y$ :

1. Compute  $X^T Y$ :  $X^T Y = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 3 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} = \begin{bmatrix} 6 \\ 14 \end{bmatrix}$

2. Multiply  $(X^T X)^{-1}$  with  $X^T Y$ :  $\beta = \begin{bmatrix} \frac{5}{3} & -1 \\ -1 & \frac{1}{2} \end{bmatrix} \begin{bmatrix} 6 \\ 14 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$

**Final Coefficients**  $\beta_0 = 0, \quad \beta_1 = 1$

The regression model is:  $Y = 0 + 1 \cdot X_1$

**End of Algebra Part**

# **Start of Calculus Part**

Consider the quadratic function  $f(t) = 3t^2 - 2t + 1$ :

- **Domain:** Since  $f(t)$  is a polynomial, there are no restrictions on the input values  $t$ . Thus, the domain is all real numbers  $(-\infty, \infty)$ .
- **Range:** The range depends on the vertex of the parabola. This function opens upwards (as the coefficient of  $t^2$  is positive), so the range is  $[f(t_{\text{vertex}}), \infty)$ , where  $t_{\text{vertex}} = -b/(2a)$ .

## 2. Function Composition

- **Basics:**
  - Composition combines two functions  $f(x)$  and  $g(x)$ :  $(f \circ g)(x) = f(g(x))$  or  $(g \circ f)(x) = g(f(x))$ .
- **Deeper Concepts:**
  - **Order matters** in composition:  $(f \circ g)(x) \neq (g \circ f)(x)$  in general.
  - Domain considerations are critical. For example, if  $g(x)$  outputs a value outside  $f(x)$ 's domain, the composition is undefined.

## Function Composition Example

Let's consider two functions:

1.  $f(x) = x^2 + 1$ : Squares the input and adds 1.
2.  $g(x) = \sqrt{x}$ : Computes the square root of the input (only defined for  $x \geq 0$ ).

### Compositions:

1.  $(f \circ g)(x) = f(g(x)) = f(\sqrt{x}) = (\sqrt{x})^2 + 1 = x + 1$ , defined for  $x \geq 0$ .
2.  $(g \circ f)(x) = g(f(x)) = g(x^2 + 1) = \sqrt{x^2 + 1}$ , defined for all  $x$  since  $x^2 + 1 \geq 0$ .

### Key Points:

- The **domain** of  $f \circ g(x)$  is restricted to  $x \geq 0$  because  $g(x)$  (square root) is undefined for  $x < 0$ .
- The **domain** of  $g \circ f(x)$  is all real numbers because  $x^2 + 1$  is always non-negative.

# The Derivative: A Deeper Explanation

The derivative of a function  $f(x)$  is a fundamental concept in calculus that describes how the function  $f(x)$  changes as  $x$  changes. It has both **conceptual** and **geometrical** interpretations that make it incredibly useful in mathematics, physics, engineering, and beyond.

## 1. Formal Definition

The derivative of  $f(x)$  at a point  $x$  is defined as:

$$f'(x) = \lim_{h \rightarrow 0} \frac{f(x + h) - f(x)}{h}$$

- **Numerator:**  $f(x + h) - f(x)$  measures the change in the output of the function as  $x$  increases by a small amount  $h$ .
- **Denominator:** Dividing by  $h$  scales this change per unit increase in  $x$ , giving the rate of change.

### Intuition:

- The derivative calculates the **instantaneous rate of change** of  $f(x)$  at a specific  $x$ .
- It approximates the slope of the **tangent line** to the curve  $y = f(x)$  at that point.



## Role of the Denominator $h$ :

### 1. Represents the Change in Input (Step Size):

- The denominator  $h$  measures how much the input  $x$  is incremented.
- For example, if  $h = 1$ , we are calculating the average rate of change over an interval of width 1.

### 2. Scaling the Change in Output:

- The numerator  $f(x + h) - f(x)$  gives the **change in the function's output** over the interval.
- Dividing by  $h$  gives the **rate of change per unit of  $x$** . It answers the question: "How much does  $f(x)$  change for every unit increase in  $x$ ?"

Example:

- If  $h = 2$  and  $f(x + h) - f(x) = 6$ , then the rate of change is  $\frac{6}{2} = 3$ , meaning  $f(x)$  increases by 3 units for every unit increase in  $x$ .

## Numerical Example

Consider  $f(x) = x^2$  at  $x = 1$ . Let's compute  $\frac{f(1+h)-f(1)}{h}$  for decreasing values of  $h$ :

1. Formula:

$$\frac{f(1+h) - f(1)}{h} = \frac{(1+h)^2 - 1^2}{h} = \frac{1 + 2h + h^2 - 1}{h} = 2 + h.$$

2. Values for Different  $h$ :

- $h = 1$ :  $\frac{f(1+1)-f(1)}{1} = 2 + 1 = 3.$
- $h = 0.1$ :  $\frac{f(1+0.1)-f(1)}{0.1} = 2 + 0.1 = 2.1.$
- $h = 0.01$ :  $\frac{f(1+0.01)-f(1)}{0.01} = 2 + 0.01 = 2.01.$
- $h = 0.001$ :  $\frac{f(1+0.001)-f(1)}{0.001} = 2 + 0.001 = 2.001.$

3. As  $h \rightarrow 0$ :

- The slope approaches 2, the value of the derivative  $f'(1) = 2.$

## 1. Tangent Line Approximations:

- Tangent lines provide linear approximations to non-linear functions near a specific point.
- For small changes around  $x = a$ ,  $f(x) \approx f'(a)(x - a) + f(a)$ .

## 2. Generalization:

- This method works for any differentiable function  $f(x)$  at any point  $x = a$ .
- The steeper the curve at  $a$ , the larger the slope  $f'(a)$ .

The function is:  $f(x) = \sqrt{x}$

## 2. Tangent Line Equation

The tangent line to  $f(x)$  at  $x = 4$  is given by the formula:

$$y = f'(4)(x - 4) + f(4)$$

Here:

- $f'(4)$ : Slope of the tangent line at  $x = 4$ .
- $f(4)$ : Value of the function at  $x = 4$ .

## 3. Calculate $f(4)$ and $f'(4)$

- $f(4) = \sqrt{4} = 2$
- $f'(4) = \frac{1}{2\sqrt{4}} = \frac{1}{4}$

## 4. Write the Tangent Line Equation

Substitute these values into the tangent line equation:

$$y = \frac{1}{4}(x - 4) + 2$$

$$y = \frac{1}{4}x - 1 + 2$$

$$y = \frac{1}{4}x + 1$$

## 5. Approximation for $x = 4.1$

To estimate  $\sqrt{4.1}$ , substitute  $x = 4.1$  into the tangent line equation:

$$y = \frac{1}{4}(4.1) + 1$$

$$y = 1.025 + 1 = 2.025$$

Thus,  $\sqrt{4.1} \approx 2.025$ .

# Tangent Line Applications in Machine Learning:

The concept of tangent lines is fundamental in machine learning, particularly when working with **optimization** algorithms like **gradient descent**.

In the context of gradient descent and optimization, tangent lines play a crucial role in providing guidance, magnitude, and efficiency for parameter updates.

## I. Gradient Descent: Optimization in Machine Learning Overview:

- In machine learning, the goal is often to minimize a loss function  $L(w)$ , which measures how well a model's predictions match the true values.
- **Tangent lines are essential for calculating the slope of the loss function at any point**
- The slope (or gradient) is used to adjust the model's parameters to reduce the loss.

## How Tangent Lines Are Used

- The slope of the tangent line to  $L(w)$  at a specific point indicates the direction and rate of change of  $L(w)$ .
- Gradient descent updates parameters  $w$  iteratively:

$$w_{\text{new}} = w_{\text{old}} - \eta \cdot L'(w)$$

Here:

- $L'(w)$ : Slope of the tangent line (derivative of  $L(w)$ ).
- $\eta$ : Learning rate (step size).

## 1. Goal:

- The aim in machine learning is to minimize a **loss function**  $L(w)$ , which quantifies how far off a model's predictions are from the actual values.
- $w$ : Represents model parameters (e.g., weights).

## 2. Role of Tangent Lines:

- The tangent line to the loss function  $L(w)$  at a point gives the **slope** or **gradient** ( $L'(w)$ ).
- This slope indicates:
  - **Direction**: Should we increase or decrease  $w$  to minimize  $L(w)$ ?
  - **Magnitude**: How much should  $w$  change to effectively reduce  $L(w)$ ?

## 3. Update Rule:

- In **gradient descent**, the model's parameters  $w$  are updated iteratively:

$$w_{\text{new}} = w_{\text{old}} - \eta \cdot L'(w_{\text{old}})$$

- Here:
  - $\eta$ : Learning rate (controls step size).
  - $L'(w)$ : Slope of the loss function (from the tangent line).

## 4. Why It Works:

- The slope (gradient) tells us how steep the curve is and helps us move toward the direction of the minimum  $w$ .

# 1. Guidance

- What it means:
  - The slope of the tangent line at a point  $w$  (computed as  $L'(w)$ ) tells us whether to **increase** or **decrease**  $w$  to minimize the loss function  $L(w)$ .
- How it works:
  - If  $L'(w) > 0$ : The tangent line slopes **upward**, meaning  $w$  is too large, and we need to **decrease**  $w$ .
  - If  $L'(w) < 0$ : The tangent line slopes **downward**, meaning  $w$  is too small, and we need to **increase**  $w$ .
  - If  $L'(w) = 0$ : The slope is flat, meaning we are at the minimum, and no further updates are needed.
  - Example: For the loss function  $L(w) = (w - 2)^2$ :
    - At  $w = 0$ :  $L'(w) = 2(0 - 2) = -4 \rightarrow$  Negative slope  $\rightarrow$  Increase  $w$ .
    - At  $w = 3$ :  $L'(w) = 2(3 - 2) = 2 \rightarrow$  Positive slope  $\rightarrow$  Decrease  $w$ .



- Deeper Concepts:
  - Rules of differentiation:
    - Power Rule:  $\frac{d}{dx}(x^n) = nx^{n-1}$ .
    - Chain Rule:  $\frac{d}{dx}[g(f(x))] = g'(f(x))f'(x)$ .
    - Product Rule:  $(uv)' = u'v + uv'$ .
    - Quotient Rule:  $\left(\frac{u}{v}\right)' = \frac{u'v - uv'}{v^2}$ .

- **Norms of Vectors:**
- In machine learning, vector norms are used to measure the magnitude (or length) of a vector.
- Norms are crucial for optimization, regularization, distance measurement, and assessing the scale of vectors.

$$\|x\|_1 = \sum_{i=1}^n |x_i|$$

- **Definition:**

- The  $L_1$  norm is the **sum of the absolute values** of all elements in the vector.
- For a vector  $x = [x_1, x_2, \dots, x_n]$ :

$$\|x\|_1 = |x_1| + |x_2| + \dots + |x_n|$$

- **Geometric Interpretation:**

- It represents the "Manhattan distance" (or taxicab geometry) between the vector and the origin in the vector space.

# L1 norm

- **Machine Learning Applications:**

1. **Feature Sparsity:**

- $L_1$ -based regularization (e.g., Lasso regression) promotes sparsity in the model by driving some coefficients to zero, effectively selecting features.

2. **Robustness:**

- The  $L_1$  norm is robust to outliers, as it minimizes absolute differences instead of squared differences.

- **Example:**

- Given  $x = [1, -2, 3]$ :

$$\|x\|_1 = |1| + |-2| + |3| = 1 + 2 + 3 = 6$$

# L2 norm

$$\|x\|_2 = \sqrt{\sum_{i=1}^n x_i^2}$$

- Definition:
  - The  $L_2$  norm is the **Euclidean norm**, representing the **straight-line distance** between the vector and the origin.
  - For a vector  $x = [x_1, x_2, \dots, x_n]$ :

$$\|x\|_2 = \sqrt{x_1^2 + x_2^2 + \dots + x_n^2}$$

- **Geometric Interpretation:**

- It measures the length (or magnitude) of the vector in the Euclidean space.

- **Machine Learning Applications:**

1. **Smoothness:**

- $L_2$ -based regularization (e.g., Ridge regression) penalizes large coefficients without driving them to zero, ensuring smooth and less complex models.

2. **Gradient Descent:**

- The  $L_2$  norm is commonly used to compute distances and gradients in optimization problems.

- **Example:**

- Given  $x = [1, -2, 3]$ :

$$\|x\|_2 = \sqrt{1^2 + (-2)^2 + 3^2} = \sqrt{1 + 4 + 9} = \sqrt{14} \approx 3.74$$

# L-Infinity norm

$$\|x\|_{\infty} = \max(|x_i|)$$

- **Definition:**

- The  $L_{\infty}$  norm is the **maximum absolute value** among the elements of the vector.
- For a vector  $x = [x_1, x_2, \dots, x_n]$ :

$$\|x\|_{\infty} = \max(|x_1|, |x_2|, \dots, |x_n|)$$

- **Geometric Interpretation:**

- It represents the distance in "Chebyshev geometry," where the metric considers the largest coordinate displacement.

- **Example:**

- Given  $x = [1, -2, 3]$ :

$$\|x\|_{\infty} = \max(|1|, |-2|, |3|) = 3$$

# Matrices



## **Topic**

## **Machine Learning Applications**

- Matrix Multiplication Neural networks (forward propagation), data transformation
- Matrix Operations Covariance matrices, solving systems of equations
- Dot Product Cosine similarity, weighted sums in neural networks
- Orthogonal Vectors PCA, feature independence
- Gaussian Elimination Solving linear regression problems
- Linear Dependence Feature selection, eliminating redundant features
- Rank of a Matrix Dimensionality reduction, low-rank approximations

# Why Orthogonality Matters in PCA

- **Redundancy Removal:**

- Orthogonal components ensure no redundancy (correlation) in the transformed data.

- **Feature Selection:**

- The first few components explain most of the variance, allowing us to reduce dimensions while retaining information.

- **Model Stability:**

- Using orthogonal features improves numerical stability in machine learning algorithms.

- Orthogonal vectors ensure that features or components are independent and contribute unique information.
- In advanced scenarios like PCA, orthogonality simplifies the data representation, removes redundancy, and improves the efficiency of machine learning models.
- This approach is widely used in dimensionality reduction, feature engineering, and unsupervised learning tasks.

# Orthogonal Vectors

## Definition:

Two vectors **a** and **b** are **orthogonal** if their dot product is zero:

$$\mathbf{a} \cdot \mathbf{b} = 0$$

Geometrically, this means that the two vectors are **perpendicular** to each other (their angle is  $90^\circ$ ).

## Importance of Orthogonal Vectors

Feature Independence in Machine Learning:

### Interpretation:

If two feature vectors (columns in a dataset) are orthogonal, they are completely **independent** of each other.

This ensures that one feature does not contribute redundant information, leading to better models and easier interpretation.

### Applications:

Orthogonality ensures no correlation between features.

Orthogonal vectors simplify computations in linear models and reduce multicollinearity.

# Why Keep Both Features?

## Unique Contribution:

- Orthogonal features are completely independent, meaning they describe different aspects of the data.
- Removing one feature would result in a loss of information.

### Dataset Example:

Suppose we have two features:

1. Feature  $x_1 = [1, 0]$  (e.g., horizontal movement).
2. Feature  $x_2 = [0, 1]$  (e.g., vertical movement).

These features are orthogonal, meaning they are independent:

- $x_1 \cdot x_2 = 0$ .

### Retaining Both Features:

- $x_1$ : Captures information about horizontal variability.
- $x_2$ : Captures information about vertical variability.
- If you remove one feature, you lose the ability to describe one axis entirely, reducing the descriptive power of your model.

## Gaussian Elimination

- Gaussian elimination is a systematic method for solving systems of linear equations.
- It transforms a given system into an equivalent triangular form (row echelon form) using elementary row operations.
- Once in this form, the solution can be easily obtained by back-substitution

# Key Steps in Gaussian Elimination

## 1. Augmented Matrix Formation:

- Write the system of equations in matrix form:

$$A\mathbf{x} = \mathbf{b}$$

Combine  $A$  (the coefficient matrix) and  $\mathbf{b}$  (the right-hand side vector) into an **augmented matrix**:

$$[A|\mathbf{b}]$$

## 2. Forward Elimination:

- Eliminate variables below the pivot (diagonal) element by performing row operations:
  - Row Swapping:** Swap rows to ensure a non-zero pivot.
  - Scaling:** Multiply a row by a scalar.
  - Row Replacement:** Replace a row by subtracting a multiple of another row.

## 3. Back Substitution:

- Once the augmented matrix is in **row echelon form** (upper triangular matrix), solve for each variable starting from the last equation.

$$\begin{array}{l} \text{System of Linear Equations:} \\ 2x + y - z = 8 \\ -3x - y + 2z = -11 \\ -2x + y + 2z = -3 \end{array}$$

### Step 1: Augmented Matrix

Write the system as an augmented matrix:

$$\left[ \begin{array}{ccc|c} 2 & 1 & -1 & 8 \\ -3 & -1 & 2 & -11 \\ -2 & 1 & 2 & -3 \end{array} \right]$$

### Step 2: Forward Elimination

#### 1. First Pivot (Row 1):

- Divide the first row by 2 to make the pivot element 1:

$$\left[ \begin{array}{ccc|c} 1 & 0.5 & -0.5 & 4 \\ -3 & -1 & 2 & -11 \\ -2 & 1 & 2 & -3 \end{array} \right]$$

- Eliminate the first variable ( $x$ ) from Rows 2 and 3:
  - Row 2:  $R_2 = R_2 + 3 \cdot R_1$
  - Row 3:  $R_3 = R_3 + 2 \cdot R_1$

$$\left[ \begin{array}{ccc|c} 1 & 0.5 & -0.5 & 4 \\ 0 & 0.5 & 0.5 & 1 \\ 0 & 2 & 1 & 5 \end{array} \right]$$

## 2. Second Pivot (Row 2):

- Make the second pivot element 1 by dividing Row 2 by 0.5:

$$\begin{bmatrix} 1 & 0.5 & -0.5 & 4 \\ 0 & 1 & 1 & 2 \\ 0 & 2 & 1 & 5 \end{bmatrix}$$

- Eliminate the second variable ( $y$ ) from Row 3:

- Row 3:  $R_3 = R_3 - 2 \cdot R_2$

$$\begin{bmatrix} 1 & 0.5 & -0.5 & 4 \\ 0 & 1 & 1 & 2 \\ 0 & 0 & -1 & 1 \end{bmatrix}$$

## 3. Third Pivot (Row 3):

- Make the third pivot element 1 by dividing Row 3 by -1:

$$\begin{bmatrix} 1 & 0.5 & -0.5 & 4 \\ 0 & 1 & 1 & 2 \\ 0 & 0 & 1 & -1 \end{bmatrix}$$