Linear Algebra for AI: Deep Learning

Thuan L Nguyen, PhD

Slide 2: Linear Algebra for AI: Deep Learning

1. AI Deep Learning: Linear Algebra and Deep Learning
2. AI Deep Learning: Scalars, Vectors, and Matrices
3. AI Deep Learning: Matrices: Operations: Addition, Subtraction, and Multiplication
4. AI Deep Learning: Matrices and Python Numpy N-Dimensional Arrays
5. AI Deep Learning: Matrices: A Vector of Vectors
6. AI Deep Learning: Matrices: The Concept of Axis
7. AI Deep Learning: Tensors: N-Dimensional Arrays
8. AI Deep Learning: Multi-Class Classifications: One-Hot Coding
9. AI Deep Learning: Activation Functions: ReLU (Rectified Linear Unit) and Softmax

Slide 3: Linear Algebra for AI: Deep Learning

Linear Algebra:

- Linear algebra is a key branch of mathematics.
- An understanding of linear algebra is crucial for deep learning, that is, neural networks.
- Linear Algebra deals with linear systems of equations.
- The focus will be on vectors, matrices, and tensors.
- Using linear algebra, we can describe complicated operations in deep learning.

Slide 4: Linear Algebra for AI: Deep Learning

Scalars, Vectors, and Matrices

- Scalars, vectors, and matrices are the fundamental objects of mathematics:
  - Scalar is represented by a single number or numerical value called magnitude.
  - Vector is an array of numbers assembled in order.
    - A unique index identifies each number.
    - Vector represents a point in space, with each element giving the coordinate along a different axis.
  - Matrices is a two-dimensional array of numbers where each number is identified using two indices (i, j).

Slide 5: Linear Algebra for AI: Deep Learning

Scalars, Vectors, and Matrices: Examples

- Scalars, vectors, and matrices are the fundamental objects of mathematics:
  - Scalar is represented by a single number or numerical value called magnitude.

- For example:
  - 5
  - 16.32
  - -0.128
  - …

Slide 6: Linear Algebra for AI: Deep Learning

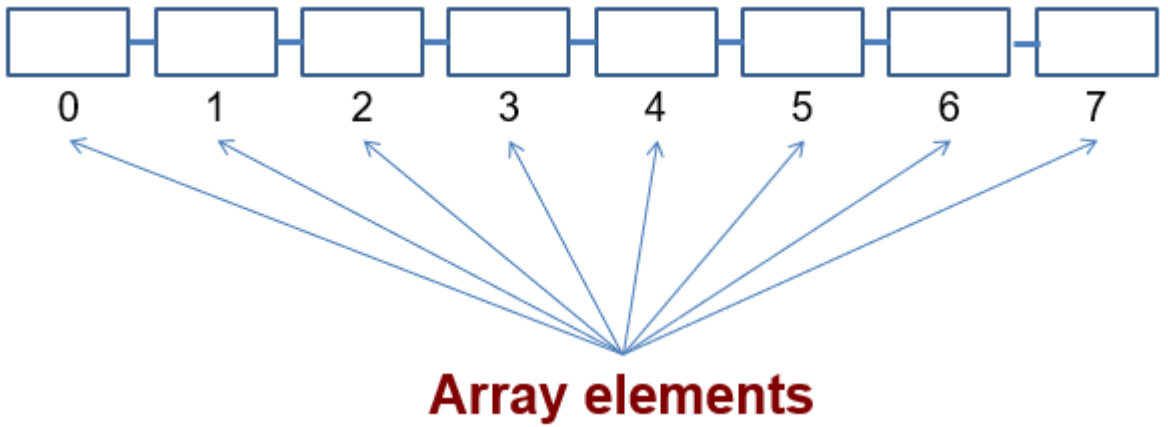
Scalars, Vectors, and Matrices: Examples

- **Scalars, vectors,** and **matrices** are the fundamental objects of mathematics:
  - **Vector** is an array of values assembled in order.
    - A unique index identifies each value.

- For example:
  - [1, 3, 5, 7, 9]
  - [‘a’, ‘b’, ‘c’]
  - [“abc”, “def”]

- A **vector** can be implemented with a Python Numpy 1D-Array.

Slide 7: Linear Algebra for AI: Deep Learning



Vector: Index of Elements

- A vector can be implemented with a Python Numpy 1D-Array.
- Usually, the **index** of the elements of a vector **starts at 0**.

Slide 8: Linear Algebra for AI: Deep Learning

Scalars, Vectors, and Matrices: Examples

- Scalars, vectors, and matrices are the fundamental objects of mathematics:
  - **Matrices** is a two-dimensional array of values where each value is identified using **two indices (i, j)**.

- For example:

$$\begin{bmatrix} 1 & 1 & 1 & -1 \\ 1 & 2 & 4 & 3 \\ 1 & 3 & 9 & 3 \end{bmatrix}$$

$$\begin{bmatrix} 2 & 1 & 3 & 4 & 5 \\ 0 & 0 & 1 & 4 & 2 \\ 4 & 2 & 6 & 8 & 10 \\ 6 & 3 & 14 & 35 & 33 \end{bmatrix}$$

Slide 9: Linear Algebra for AI: Deep Learning

Scalars, Vectors, and Matrices: Examples

- Scalars, vectors, and matrices are the fundamental objects of mathematics:
  - **Matrices** is a two-dimensional array of values where each value is identified using **two indices (i, j)**.

- Other examples (Not using square brackets):

$$\begin{pmatrix} 2 & 1 & 2 \\ 1 & -1 & 4 \end{pmatrix}$$

$$B = \begin{pmatrix} 2 & 5 & 14 \\ 1 & 3 & 8 \\ -1 & -2 & -6 \end{pmatrix}$$

Slide 10: Linear Algebra for AI: Deep Learning

Matrices: Operations: Addition & Subtraction

- Two matrices A and B can be added or subtracted if and only if their **dimensions** are the **same** (i.e. both matrices have the same number of rows and columns).

- For example:

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 1 & 0 & 2 \end{pmatrix} \quad and \quad B = \begin{pmatrix} 2 & 1 & 2 \\ 1 & 0 & 3 \end{pmatrix}$$

$$A + B = \begin{pmatrix} 1 & 2 & 3 \\ 1 & 0 & 2 \end{pmatrix} + \begin{pmatrix} 2 & 1 & 2 \\ 1 & 0 & 3 \end{pmatrix} = \begin{pmatrix} 3 & 3 & 5 \\ 2 & 0 & 5 \end{pmatrix}$$

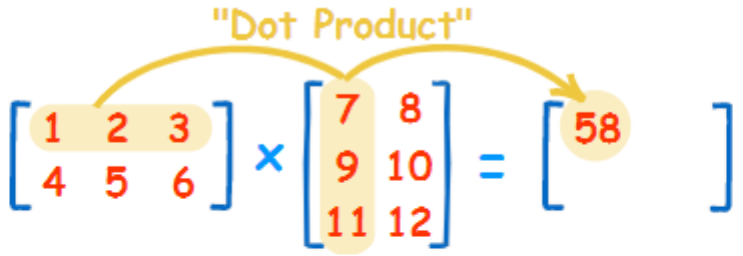
$$A - B = \begin{pmatrix} 1 & 2 & 3 \\ 1 & 0 & 2 \end{pmatrix} - \begin{pmatrix} 2 & 1 & 2 \\ 1 & 0 & 3 \end{pmatrix} = \begin{pmatrix} -1 & 1 & 1 \\ 0 & 0 & -1 \end{pmatrix}$$

Slide 11: Linear Algebra for AI: Deep Learning

RNN: LSTM: Linear Algebra for Deep Learning:

Vector and Matrix Multiplication: Dot Multiplication

$$(m \times \mathbf{n}) * (\mathbf{n} \times p) \rightarrow (m \times p)$$



The "Dot Product" is where we **multiply matching members**, then sum up:

$$(1, 2, 3) \bullet (7, 9, 11) = 1\times7 + 2\times9 + 3\times11$$
$$= 58$$

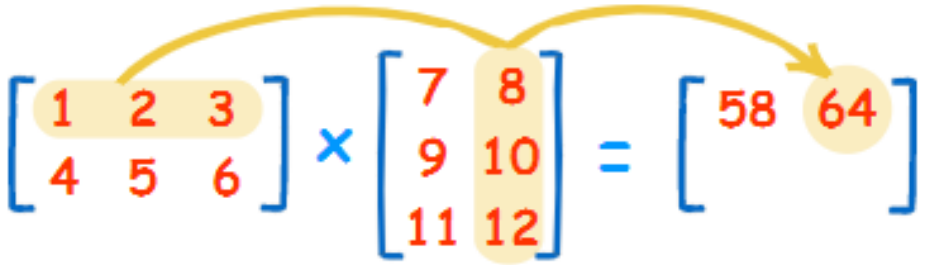
Dot Product of Matrices (Sources: mathsisfun.com)

Slide 12: Linear Algebra for AI: Deep Learning

RNN: LSTM: Linear Algebra for Deep Learning:

Vector and Matrix Multiplication: Dot Multiplication

$$(m \times \mathbf{n}) * (\mathbf{n} \times p) \rightarrow (m \times p)$$



$$(1, 2, 3) \bullet (8, 10, 12) = 1\times8 + 2\times10 + 3\times12 = 64$$

Dot Product of Matrices (Sources: mathsisfun.com)

Slide 13: Linear Algebra for AI: Deep Learning

RNN: LSTM: Linear Algebra for Deep Learning:

Vector and Matrix Multiplication: Dot Multiplication

$(m \times \mathbf{n}) * (\mathbf{n} \times p) \rightarrow (m \times p)$

We can do the same thing for the **2nd row** and **1st column**:

$$(4, 5, 6) \bullet (7, 9, 11) = 4\times7 + 5\times9 + 6\times11 = 139$$

And for the **2nd row** and **2nd column**:

$$(4, 5, 6) \bullet (8, 10, 12) = 4\times8 + 5\times10 + 6\times12 = 154$$

And we get:

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \times \begin{bmatrix} 7 & 8 \\ 9 & 10 \\ 11 & 12 \end{bmatrix} = \begin{bmatrix} 58 & 64 \\ 139 & 154 \end{bmatrix}$$ ✓

Dot Product of Matrices (Sources: mathsisfun.com)

Slide 14: Linear Algebra for AI: Deep Learning

RNN: LSTM: Linear Algebra for Deep Learning:

Vector and Matrix Multiplication: Dot Multiplication

$$(m \times \mathbf{n}) * (\mathbf{n} \times p) \rightarrow (m \times p)$$

$$A = \begin{bmatrix} 1 & 3 \\ -1 & 0 \end{bmatrix} \quad B = \begin{bmatrix} 2 & 1 & 1 \\ -1 & 2 & 4 \end{bmatrix}$$

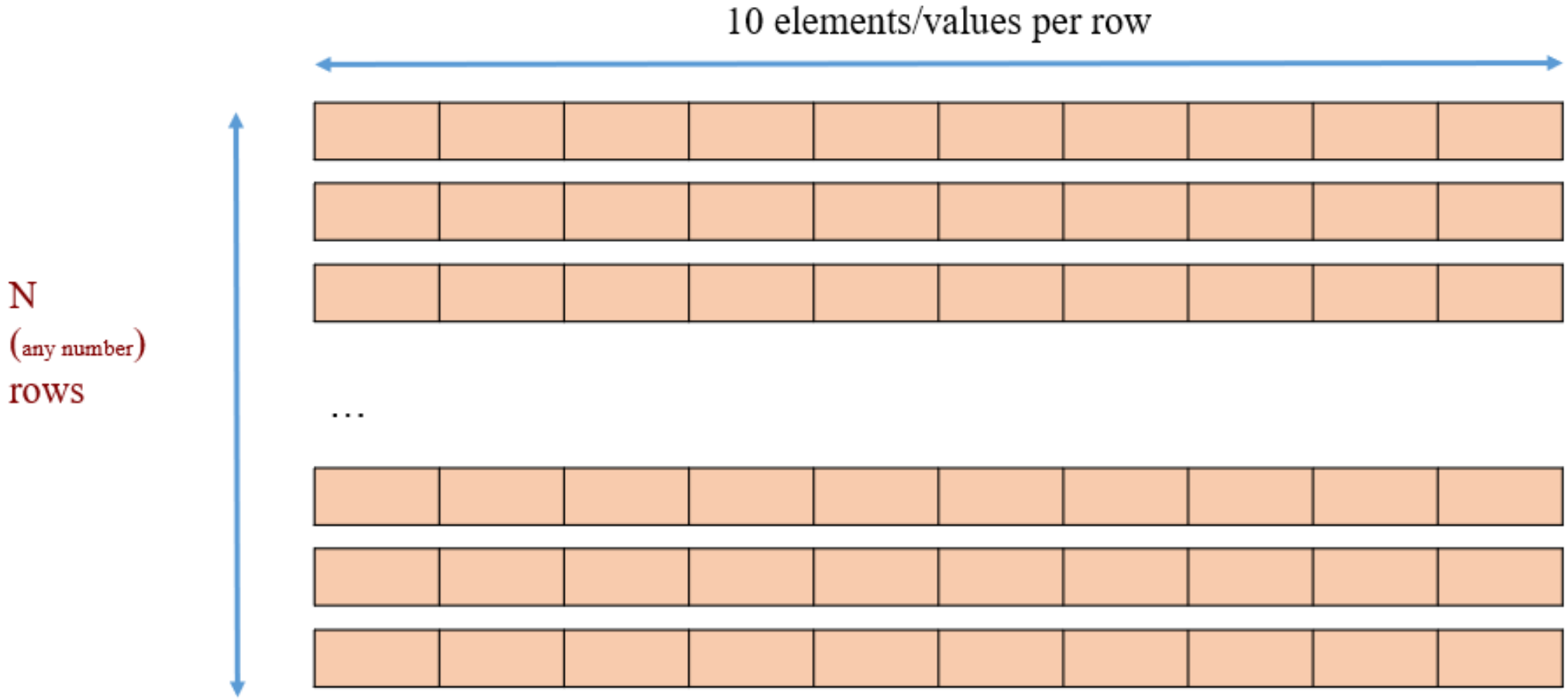
$$(2 \times \mathbf{2}) \quad * \quad (\mathbf{2} \times 3) \quad \rightarrow \quad \mathbf{(2 \times 3)}$$

$$AB = \begin{bmatrix} 1 \cdot 2 + 3 \cdot -1 & 1 \cdot 1 + 3 \cdot 2 & 1 \cdot 1 + 3 \cdot 4 \\ -1 \cdot 2 + 0 \cdot -1 & -1 \cdot 1 + 0 \cdot 2 & -1 \cdot 1 + 0 \cdot 4 \end{bmatrix} = \begin{bmatrix} -1 & 7 & 13 \\ -2 & -1 & -1 \end{bmatrix}$$

Slide 15: Linear Algebra for AI: Deep Learning

Matrices & Python Numpy N-Dimensional Arrays

- A **matrix** can be viewed as a **vector of vectors;** each **vector** is one **row.**



Slide 16: Linear Algebra for AI: Deep Learning

Matrices & Python Numpy N-Dimensional Arrays

- A **matrix** can be viewed as a **vector of vectors;** each **vector** is one **row.**

$$\begin{bmatrix} 1 & 1 & 1 & -1 \\ 1 & 2 & 4 & 3 \\ 1 & 3 & 9 & 3 \end{bmatrix}$$

- The matrix (a vector of vectors) has three vector elements:
  - Index 0: [1, 1, 1, -1]
  - Index 1: [1, 2, 4, 3]
  - Index 2: [1, 3, 9, 3]

- The matrix can be displayed in another format:

[[1, 1, 1, -1], [1, 2, 4, 3], [1, 3, 9, 3]]

Slide 17: Linear Algebra for AI: Deep Learning

Matrices & Python Numpy N-Dimensional Arrays: The Concept of Axis

IMPORTANT NOTES: Axis = 0

- Matrix: A vector of vectors

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15

- **Axis 0**:
  - First axis
  - Direction: Moving from one vector to another vector
  - For example: Element-wise along Axis 0
    - 1 → 6 → 11
    - 2 → 7 → 12
    - …
    - 5 → 10 → 15

Slide 18: Linear Algebra for AI: Deep Learning

Matrices & Python Numpy N-Dimensional Arrays: The Concept of Axis

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15

- **Axis 1**:
  - Second axis
  - Direction: Moving from one element to another element in the same vector
  - For example: Element-wise along Axis 1
    - 1 → 2 → 3 → …
    - 6 → 7 → 8 → …
    - …
    - 11 → 12 → …

Slide 19: Linear Algebra for AI: Deep Learning

Matrices & Python Numpy N-Dimensional Arrays

- A **matrix** can be viewed as a **vector of vectors**.
- Instead of displaying a matrix in the form of rows x columns, it is possible to show the **matrix** as a vector of which **each element** is one **row** of the matrix.

$$A = \begin{bmatrix} 1 & 3 \\ -1 & 0 \end{bmatrix} \quad = \quad [\,[1, 3], [-1, 0]\,]$$

$$B = \begin{bmatrix} 2 & 1 & 1 \\ -1 & 2 & 4 \end{bmatrix} \quad = \quad [\,[2, 1, 1], [-1, 2, 4]\,]$$

Slide 20: Linear Algebra for AI: Deep Learning

Flattening a Numpy N-Dimensional Array into a Vector (an 1D-Array)

IMPORTANT NOTES:

- [[1, 3], [-1, 0]] **=/=** [1, 3, -1, 0]
  - **[1, 3, -1, 0]** is a vector, i.e., a 1D-array, that has 4 elements; each element is a scalar
  - **[[1, 3], [-1, 0]]** is a matrix, i.e., a 2D-array. That has 2 elements; each element is a vector.

To **flatten** a Numpy N-Dimensional array, say **ND-array**, is to collapse all the elements of ND-array into a vector, or a one-dimensional array, **1D-array**.

Slide 21: Linear Algebra for AI: Deep Learning

Flattening a Numpy N-Dimensional Array into a Vector (an 1D-Array)

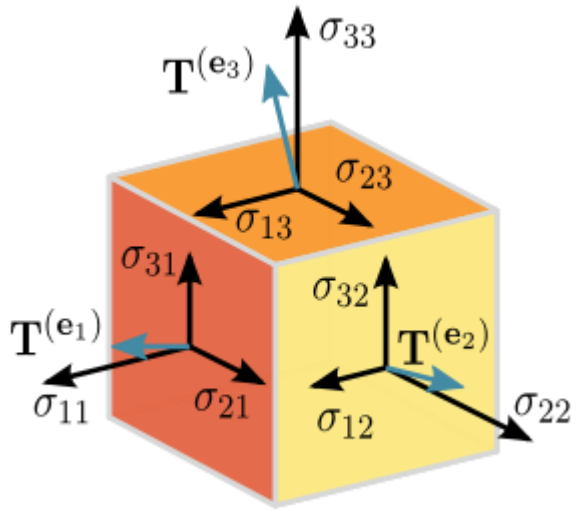
For example:

- [[1, 3], [-1, 0]] ------ flattening → **[1, 3, -1, 0]**
- [[2, 1, 1], [-1, 2, 4], [1, 3, 1]] ------ flattening → **[2, 1, 1, -1, 2, 4, 1, 3, 1]**

Slide 22: Linear Algebra for AI: Deep Learning

Tensors: N-Dimensional Arrays

- In mathematics, a tensor is a geometric object:
  - This object can map geometric vectors, scalars, and other tensors to a resulting tensor in a multi-linear manner.
- Geometric vectors and scalars are considered as the simplest tensors.
- The dimension of a tensor is called its rank.
- The image to the left describes a tensor of rank 2, i.e., 2nd-order tensor, in a 3D Cartesian coordinate system:



Tensor (Source: Wikipedia)

Slide 23: Linear Algebra for AI: Deep Learning

Tensors: N-Dimensional Arrays

- To be simple, a tensor is often thought of as a generalized matrix. It could be:
  - A 0-D matrix: a single number, i.e., a scalar
  - A 1-D matrix: a vector
  - A 2-D matrix
  - A higher dimensional array, i.e., a N-Dimensional array

- A tensor can be implemented as a Python Numpy N-Dimensional array.

Slide 24: Linear Algebra for AI: Deep Learning

Tensors: N-Dimensional Arrays

- A scalar is a tensor ($f : \mathbb{R} \to \mathbb{R},\ f(e_1) = c$)
- A vector is a tensor ($f : \mathbb{R}^n \to \mathbb{R},\ f(e_i) = v_i$)
- A matrix is a tensor ($f : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R},\ f(e_i, e_j) = A_{ij}$)
- Common to have fixed basis, **so a tensor can be represented as a multidimensional array of numbers.**

Slide 25: Linear Algebra for AI: Deep Learning

Multi-Class Classification: Encoding Classes with One-Hot Coding

- One of the major problems with Machine Learning:
  - It is a daunting task to work directly with categorical data.

- Computers are powerful and efficient when dealing with numbers.
  - So, it would be much better to convert our input data (if it is not in numeric formats) to numbers.

Slide 26: Linear Algebra for AI: Deep Learning

Multi-Class Classification: Encoding Classes with One-Hot Coding

- What is category data?
  - Categorical data are variables that contain label values rather than numeric values.
  - The number of possible values is often limited to a fixed set.
  - Categorical variables are often called nominal.

- Some examples include:
  - A “pet” variable with the values: “dog” and “cat“.
  - A “color” variable with the values: “red“, “green” and “blue“.
  - A “place” variable with the values: “first”, “second” and “third“.

- Each value represents a different category.

- Some categories may have a natural relationship to each other, such as a natural ordering.
  - The “place” variable above does have a natural ordering of values.
  - This type of categorical variable is called an ordinal variable.

Slide 27: Linear Algebra for AI: Deep Learning

Multi-Class Classification: Encoding Classes with One-Hot Coding

What is the problem with category data?

- Some algorithms can work with categorical data directly.
  - For example, a decision tree can be learned directly from categorical data with no data transform required (this depends on the specific implementation).

- Many machine learning algorithms cannot operate on label data directly.
  - They require all input variables and output variables to be numeric.
  - In general, this is mostly a constraint of the efficient implementation of machine learning algorithms rather than hard limitations on the algorithms themselves.

- This means that categorical data must be converted to a numerical form.
  - If the categorical variable is an output variable, it is desirable to convert predictions by the model back into a categorical form in order to present them or use them in some application.

Slide 28: Linear Algebra for AI: Deep Learning

Multi-Class Classification: Encoding Classes with One-Hot Coding

How to Convert Categorical Data to Numerical Data?

- The conversion is done in two steps:
  - Integer coding
  - One-hot coding

First Step: **Integer Coding**

- First, each unique category value is assigned an integer value.
  - For example:
    - "Yes" = 1, "No" = 0
    - "*red*" = 0, "*green*" = 1, "*blue*" = 2.
    - "sedan" = 1, "truck" = 2; "sport-utility" = 3.

- This is called a label encoding or an integer encoding and is easily reversible.
- The integers have a natural ordered relationship that is good for ordinal and boolean variables
  - In this case, integer coding is good enough, e.g., "Yes" = 1; "No" = 0.

Slide 29: Linear Algebra for AI: Deep Learning

Multi-Class Classification: Encoding Classes with One-Hot Coding

How to Convert Categorical Data to Numerical Data?

Second Step: **One-Hot Coding**

- For categorical variables where **no** such ordinal relationship exists:
  - The integer encoding is **not** enough.
- In fact, using the integer encoding and allowing the model to assume a natural ordering between categories may result in poor performance or unexpected results.
  - Like predictions halfway between categories
- In this case, a **one-hot encoding** can be applied to the integer representation.
  - A vector (1D array) of binary values is used to represent each unique integer value.
  - For example, for three colors, there are 3 categories that needs 3 vectors of binary values:
    - "Red" = 0: [1, 0, 0] → the value at the index 0 = 1; all the values at other indices = 0
    - "Green" = 1: [0, 1, 0] → the value at the index 1 = 1; all the values at other indices = 0
    - "Blue" = 2: [0, 0, 1] → the value at the index 2 = 1; all the values at other indices = 0

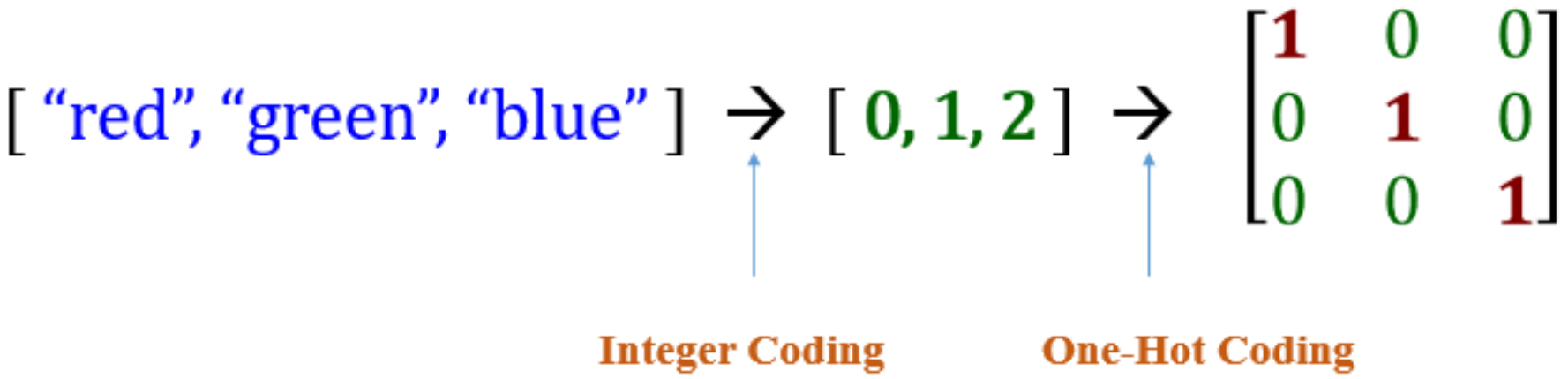
Slide 30: Linear Algebra for AI: Deep Learning

Multi-Class Classification: Encoding Classes with One-Hot Coding

How to Convert Categorical Data to Numerical Data?

Second Step: **One-Hot Coding**

- With the one-hot coding, the set of categorical values (“red”, “green”, “blue”) is transformed from a **vector** (1D array) into a **matrix** (2D array = vector of vectors):



Slide 31: Linear Algebra for AI: Deep Learning

Multi-Class Classification: Encoding Classes with One-Hot Coding

How to Convert Categorical Data to Numerical Data?

Second Step: **One-Hot Coding**

- With the one-hot coding, the set of categorical values ("red", "green", "blue") is transformed from a **vector** (1D array) into a **matrix** (2D array = vector of vectors).

IMPORTANT NOTES:

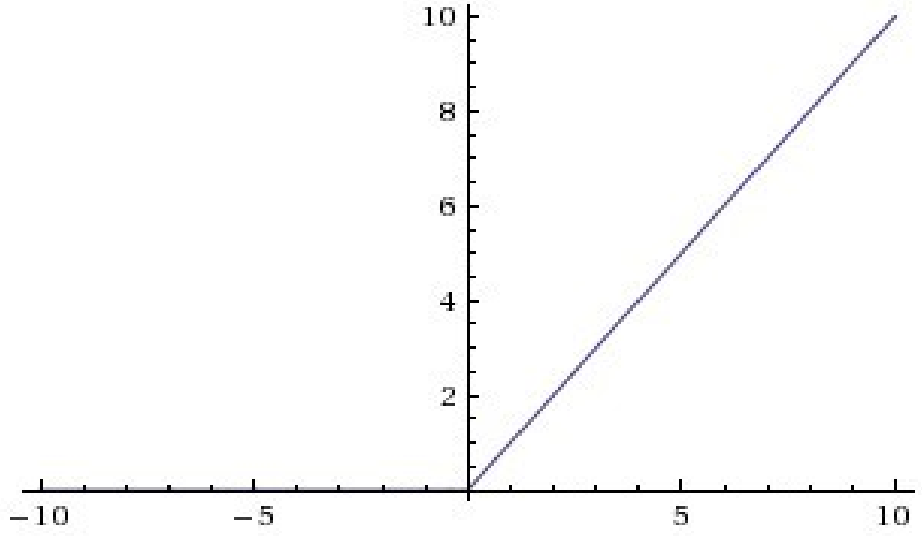
*--) If the class attribute of a dataset has only two categorical values, e.g., "**Yes**" and "**No**", "Approved" and "Rejected", etc., in most cases, the **integer coding is enough**.*

*--) If the class attribute of a dataset has more than two categorical values, **even if** that they are already in numeric formats like **integers**, it should be better to **encode them using one-hot coding** to get better performance in AI machine learning and deep learning.*

Slide 32: Linear Algebra for AI: Deep Learning

AI: Deep Learning: Activation Functions

- Many activation functions can be used in the layers of a neural networks.
  - Two most popular ones: **ReLU** and **Softmax**
- ReLU (Rectified Linear Unit) function:
  - The Rectified Linear Unit (ReLU) has become very popular in the last few years.
  - It computes the function **f(x)=max(0,x)**.
    - In other words, the activation is simply thresholded at zero (see image to the right).



Activation Function ReLU (Source: Stanford)

Slide 33: Linear Algebra for AI: Deep Learning

AI: Deep Learning: Activation Functions

- **Softmax** function:
  - In mathematics, the **softmax** function, also known as softargmax or normalized exponential function is a function that takes as **input** a vector of K real numbers, and **normalizes** it into a probability distribution consisting of K probabilities. (Source: Wikipedia)
    - It means that prior to applying softmax, some vector components could be negative, or greater than one; and might not sum to 1; but after applying softmax, each component will be in the interval (0,1), and the components will add up to 1, so that they can be interpreted as probabilities.
    - Furthermore, the larger input components will correspond to larger probabilities. Softmax is often used in neural networks, to map the non-normalized output of a network to a probability distribution over predicted output classes.

The standard (unit) softmax function $\sigma : \mathbb{R}^K \to \mathbb{R}^K$ is defined by the formula

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad \text{for } j = 1, \ldots, K \text{ and } \mathbf{z} = (z_1, \ldots, z_K) \in \mathbb{R}^K$$

  - The **softmax** function is often used in the final layer of a **neural network**-based classifier.
    - Such networks are commonly trained under a log loss (or cross-entropy) regime, giving a non-linear variant of multinomial logistic regression. (Source: Wikipedia)