DEEP LEARNING AND NEURAL NETWORKS

MINH QUANG LE

An average is a value that is typical, or representative, of a set of data. Since such typical values tend to lie centrally within a set of data arranged according to magnitude, averages are also called measures of central tendency. Several types of measures of central tendency can be defined, such as the arithmetic mean, median, mode, geometric mean, harmonic mean, root mean square, trimmed mean, winsorized mean, quartiles, deciles, and percentiles. Each has advantages and disadvantages, depending on the data and the intended purpose.

Definition (The Arithmetic Mean): The arithmetic mean, or briefly the mean, of a set of N numbers $v_1, v_2, v_3, \ldots, v_N$ is denoted by $\bar{v}$ and is defined as [4]:

$$\bar{v} = \frac{v_1 + v_2 + \cdots + v_N}{N} = \frac{\sum_{j=1}^{N} v_j}{N}. \tag{1.4}$$

For instance, the arithmetic mean of the numbers in the set

$$\begin{aligned} h = \{&133, 136, 149, 133, 123, 121, 140, 139, 117, 117, 136, 108, 126, \\ &104, 116, 147, 140, 148, 150, 122, 135, 146, 133, 144, 117, 124, 135, \\ &117, 120, 121, 110, 124, 103, 137, 101, 119, 104, 113, 139, 133\}, \end{aligned} \tag{1.5}$$

is $\bar{v} = 127$. The mean is useful because it shows where the "center of gravity" exists for an observed set of values (see Figure 1.2).

If the numbers $v_1, v_2, v_3, \ldots, v_K$ occur with frequencies $f_1, f_2, f_3, \ldots, f_K$, (grouped data), the arithmetic mean is [5]

$$\begin{aligned} \bar{v} &= \frac{f_1 v_1 + f_2 v_2 + \cdots + f_K v_K}{f_1 + f_2 + \cdots + f_K} \\ &= \frac{\sum_{j=1}^{K} f_j v_j}{\sum_{j=1}^{K} f_j} = \frac{\sum_{j=1}^{K} f_j v_j}{N}, \end{aligned} \tag{1.6}$$

where $\sum_{j=1}^{K} f_j = N$ is the total frequency. For instance, if 5, 8, 6, and 2 occur with frequencies 3, 2, 4, and 1, respectively, the arithmetic mean is $\bar{v} = \frac{(5)(3)+(8)(2)+(6)(4)+(2)(1)}{3+2+4+1} = 5.7$.

Median

Definition (The Median): The median of a set of numbers arranged in order of magnitude (i.e., in an array) is either the middle value or the arithmetic mean of the two middle values. Sometimes, the median denotes by $\tilde{v}$.

Loosely speaking, order the values of a data set of size n from smallest to largest. If n is odd, the sample median is the value in position $(n+1)/2$; if n is even, it is the average of the values in positions $n/2$ and $n/2+1$ (see Figure 1.4). For instance, the set of numbers 2, 5, 6, 9, and 11 has a median 6, (rank the $n = 5$ measurements from smallest to largest: 2, 5, 6, 9, 11), however, the set of numbers 2, 9, 11, 5, 6 and 27 has a median $\frac{1}{2}(6+9) = 7.5$, (rank the measurements from smallest to largest: 2, 5, 6, 9, 11, 27) .

Mode

Definition (The Mode): The mode of a set of numbers is that value which occurs with the greatest frequency. For instance, the set of numbers 2, 4, 7, 8, 8, 8, 10, 10, 11, 12 and 18 has mode 8, however, the set of numbers 2, 3, 8, 11, 12, 14, and 16 has no mode. Set 1, 2, 3, 3, 3, 5, 5, 8, 8, 8, and 9 has two modes, 3 and 8.

Definition (The Empirical Relation): For unimodal frequency curves that are moderately skewed (asymmetrical), we have the empirical relation [5],

$$\text{Mean} - \text{mode} = 3(\text{mean} - \text{median}). \tag{1.8}$$

Geometric Mean

The geometric mean is a mean or average that uses the product of the values of a finite set of real numbers to indicate a central tendency (as opposed to the arithmetic mean, which uses the sum of the values).

Definition (The Geometric Mean G): The geometric mean G of a set of N positive numbers $v_1, v_2, v_3, \ldots, v_N$ is the Nth root of the product of the numbers [7]:

$$G = \sqrt[N]{v_1 v_2 v_3 \ldots v_N}. \tag{1.9}$$

Harmonic Mean

Definition (The Harmonic Mean *H*): The harmonic mean H of a set of N numbers $v_1, v_2, v_3, \ldots, v_N$ is the reciprocal of the arithmetic mean of the reciprocals of the numbers [7]:

$$H = \frac{1}{\frac{1}{N}\sum_{j=1}^{N}\frac{1}{v_j}}. \tag{1.11}$$

For instance, the harmonic mean of the numbers 2, 4, and 8 is $H = \frac{3}{\frac{1}{2}+\frac{1}{4}+\frac{1}{8}} = 3.43$.

Root Mean Square

Definition (The Root Mean Square (RMS) or Quadratic Mean): The RMS is calculated by taking the square root of the average of the squares of a set of values. The RMS of a set of numbers $v_1, v_2, v_3, \ldots, v_N$ is defined by [6]:

$$\text{RMS} = \sqrt{\overline{v^2}} = \sqrt{\frac{\sum_{j=1}^{N} v_j^2}{N}}. \tag{1.12}$$

For instance, the RMS of the set 1, 3, 4, 5, and 7 is $\text{RMS} = \sqrt{\frac{1^2+3^2+4^2+5^2+7^2}{5}} = 4.47$.

Standard Deviation

In contrast to the range, the standard deviation considers all the observations. Roughly speaking, the standard deviation measures variation by indicating how far, on average, the observations are from the mean. For a data set with a large amount of variation, the observations will, on average, be far from the mean; so the standard deviation will be large. For a data set with a small amount of variation, the observations will, on average, be close to the mean; so the standard deviation will be small.

Definition (The Standard Deviation of a Population): The standard deviation of a set of N numbers $v_1, v_2, v_3, \ldots, v_N$ is denoted by S and is defined by [5]:

$$S = \sqrt{\frac{\sum_{j=1}^{N}(v_j - \bar{v})^2}{N}}. \tag{1.16}$$

Thus S is sometimes called, the root-mean-square deviation (see Figure 1.10).

Sometimes the standard deviation of a sample's data is defined with $N - 1$ replacing N in the denominators of the expression in (1.16) because the resulting value represents a better estimate of the standard deviation of a population from which the sample is taken. For large values of N (certainly $N > 30$), there is practically no difference between the two definitions.

Definition (The Standard Deviation for Sample): The standard deviation of a set of N numbers $v_1, v_2, v_3, \ldots, v_N$ is denoted by S and is defined by [10,12]:

$$S = \sqrt{\frac{\sum_{j=1}^{N}(v_j - \bar{v})^2}{N - 1}}. \tag{1.17}$$

Thus S is sometimes called, the root-mean-square deviation.

Variance

Variance is a measure of the spread of data points around the mean. Variance is a useful tool for understanding and comparing data sets. For example, if you have two data sets with the same mean, but different variances, you can tell that the data points in one data set are more spread out than the data points in the other data set.

Definition (The Variance): The variance of a set of data is defined as the square of the standard deviation and is thus given by S^2 in (1.17).

For instance, the variances of the data set 3, 4, 6, 7, 10 is 6 (the mean is $(3+4+6+7+10)/5 = 6$ and $S^2 = ((-3)^2 + (-2)^2 + (0)^2 + (1)^2 + (4)^2))/5 = 6$).

When it is necessary to distinguish the standard deviation of a population from the standard deviation of a sample drawn from this population, we often use the symbol S for the latter and σ for the former. Thus S^2 and σ^2 would represent the sample variance and population variance, respectively.

Theorem 1.2: The standard deviation of a set of N numbers $v_1, v_2, v_3, \ldots, v_N$ is defined by [6]:

$$S = \sqrt{\frac{\sum_{j=1}^{N} v_j^2}{N} - \left(\frac{\sum_{j=1}^{N} v_j}{N}\right)^2} = \sqrt{\overline{v^2} - \bar{v}^2}, \tag{1.18}$$

where $\overline{v^2}$ denotes the mean of the squares of the various values of v, while $\bar{v}^2$ denotes the square of the mean of the various values of v.

If an experiment is repeated n times and an event A is observed f times where f is the frequency, then, according to the relative frequency concept of probability:

$$P(A) = \frac{f}{n} = \frac{\text{Frequency of A}}{\text{Sample size}}. \tag{1.46}$$

Axioms of Probability

Axioms are the foundational principles upon which probability theory is constructed. The axiomatic approach defines the properties that probabilities must satisfy to be considered valid measures of uncertainty. The Kolmogorov axioms are introduced by Russian mathematician Andrey Kolmogorov in 1933.

Definition (Kolmogorov Axioms, Axioms of Probability): Probability is a number that is assigned to each member of a collection of events from a random experiment that satisfies the following properties:
(1) $P(S) = 1$ where S is the sample space
(2) $0 \leq P(E) \leq 1$ for any event E
(3) For two events E_1 and E_2 with $E_1 \cap E_2 = \emptyset$

$$P(E_1 \cup E_2) = P(E_1) + P(E_2). \tag{1.47}$$

Remark:

$$P(\emptyset) = 0, \tag{1.48}$$

and for any event E,

$$P(E') = 1 - P(E). \tag{1.49}$$

Example 1.7

A coin is tossed twice. What is the probability that at least 1 head occurs?

Solution

The sample space for this experiment is

$$S = \{HH, HT, TH, TT\}.$$

If the coin is balanced, each of these outcomes is equally likely to occur. Therefore, we assign a probability of ω to each sample point. Then $4\omega = 1$, or $\omega = 1/4$. If A represents the event of at least 1 head occurring, then

$$A = \{HH, HT, TH\} \text{ and } P(A) = \frac{1}{4} + \frac{1}{4} + \frac{1}{4} = \frac{3}{4}.$$

Hence, if an experiment can result in any one of N different equally likely outcomes, and if exactly n of these outcomes correspond to event A, then the probability of event A is

$$P(A) = \frac{n}{N}. \tag{1.50}$$

Unions of Events and Addition Rules

Joint events are generated by applying basic set operations to individual events. Unions of events, such as $A \cup B$; intersections of events, such as $A \cap B$; and complements of events, such as A'— are common of interest. The probability of a joint event can often be determined from the probabilities of the individual events that it comprises. Basic set operations are also sometimes helpful in determining the probability of a joint event.

Theorem 1.5 (Probability of a Union):

$$P(A \cup B) = P(A) + P(B) - P(A \cap B). \tag{1.51}$$

If A and B are mutually exclusive events, ($A \cap B = \emptyset$ and $P(A \cap B) = 0$), then

$$P(A \cup B) = P(A) + P(B). \tag{1.52}$$

For three events, we have

$$P(A \cup B \cup C) = P(A) + P(B) + P(C) - P(A \cap B) - P(A \cap C) - P(B \cap C) + P(A \cap B \cap C). \tag{1.53}$$

Moreover, a collection of events, $E_1, E_2, \ldots, E_k$, is said to be mutually exclusive if for all pairs,

$$E_i \cap E_j = \emptyset. \tag{1.54}$$

For a collection of mutually exclusive events,

$$P(E_1 \cup E_2 \cup \ldots \cup E_k) = P(E_1) + P(E_2) + \cdots + P(E_k). \tag{1.55}$$

1.8 Conditional Probability

One very important concept in probability theory is conditional probability. In some applications, the practitioner is interested in the probability structure under certain restrictions. The probability of an event B under the knowledge that the outcome will be in event A is denoted as $P(B|A)$ and this is called the conditional probability of B given A.

Definition (Conditional Probability): The conditional probability of an event B given that an event A has occurred, $P(A) > 0$, is

$$P(B|A) = \frac{P(A \cap B)}{P(A)}. \tag{1.56}$$

This definition can be understood in a special case in which all outcomes of a random experiment are equally likely. If there are N total outcomes,

$$P(A) = (\text{number of outcomes in } A)/N. \tag{1.57}$$

Also,

$$P(A \cap B) = (\text{number of outcomes in } A \cap B)/N. \tag{1.58}$$

Consequently,

$$\frac{P(A \cap B)}{P(A)} = \frac{\text{number of outcomes in } A \cap B}{\text{number of outcomes in } A}. \tag{1.59}$$

Therefore, $P(B|A)$ can be interpreted as the relative frequency of event B among the trials that produce an outcome in event A.

Definition (Multiplication Rule [15]):

$$P(A \cap B) = P(B|A)P(A) = P(A|B)P(B). \tag{1.60}$$

Definition (Multiplication Rule [15]):

$$P(A \cap B) = P(B|A)P(A) = P(A|B)P(B). \tag{1.60}$$

Thus, the probability that both A and B occur is equal to the probability that A occurs multiplied by the conditional probability that B occurs, given that A occurs.

For any event B, we can write B as the union of the part of B in A and the part of B in A'. That is,

$$B = (A \cap B) \cup (A' \cap B). \tag{1.61}$$

This result is shown in the Venn diagram in Figure 1.12. Because A and A' are mutually exclusive, $A \cap B$ and $A' \cap B$ are mutually exclusive.

Therefore, the following total probability rule is obtained.

Definition (Total Probability Rule [15]): For any events A and B,

$$\begin{aligned} P(B) &= P(B \cap A) + P(B \cap A') \\ &= P(B|A)P(A) + P(B|A')P(A'). \end{aligned} \tag{1.62}$$

Moreover, assume $E_1, E_2, \ldots, E_k$ are k mutually exclusive and exhaustive sets. Then

$$\begin{aligned} P(B) &= P(B \cap E_1) + P(B \cap E_2) + \cdots + P(B \cap E_k) \\ &= P(B|E_1)P(E_1) + P(B|E_2)P(E_2) + \cdots + P(B|E_k)P(E_k). \end{aligned} \tag{1.63}$$

Independence

Two events are said to be independent if the occurrence (or non-occurrence) of one event does not affect the probability that the other event will occur. In this case, the conditional probability of $P(B|A)$ might equal $P(B)$, i.e., the knowledge that the outcome of the experiment is in event A does not affect the probability that the outcome is in event B. So that, we obtain

$$P(A \cap B) = P(B|A)P(A) = P(B)P(A), \tag{1.64}$$

and

$$P(A|B) = \frac{P(A \cap B)}{P(B)} = \frac{P(A)P(B)}{P(B)} = P(A). \tag{1.65}$$

These conclusions lead to an important definition.

Definition (Independence, Two Events): Two events are independent if any one of the following equivalent statements is true:
(1) $P(A|B) = P(A)$,
(2) $P(B|A) = P(B)$,
(3) $P(A \cap B) = P(A)P(B)$.

It is simple to show that independence implies related results such as

$$P(A' \cap B') = P(A')P(B'). \tag{1.66}$$

Some Types of Matrices [32, 35]

Definition (Identity Matrix): The identity matrix of size n is the $n \times n$ square matrix with ones on the main diagonal and zeros elsewhere.

$$\mathbf{I} = \begin{pmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & \vdots \\ \vdots & \vdots & \ddots & 0 \\ 0 & 0 & \dots & 1 \end{pmatrix}. \tag{2.19}$$

Definition (Real Matrix): The conjugate complex of a matrix $\mathbf{M}$ is written as $\mathbf{M}^*$ and the elements of $\mathbf{M}^*$ are the conjugate complexes of the elements of $\mathbf{M}$ i.e.

$$(M^*)_{ij} = \left(M_{ij}\right)^*. \tag{2.20}$$

For a real matrix, all the elements are real and, therefore

$$\mathbf{M} = \mathbf{M}^*, \qquad (M)_{ij} = M^*_{ij}. \tag{2.21}$$

Definition (Symmetric Matrix): The transpose of a matrix $\mathbf{M}$, denoted as $\mathbf{M}^T$, is obtained by changing rows into columns (or vice versa). For a symmetric matrix

$$\mathbf{M}^T = \mathbf{M}, \; M_{ij} = M_{ji}. \tag{2.22}$$

Definition (Skew-Symmetric Matrix): The skew-symmetric matrix satisfies

$$\mathbf{M}^T = -\mathbf{M}, \; M_{ij} = -M_{ji}. \tag{2.23}$$

Definition (Orthogonal Matrix): An orthogonal matrix satisfies

$$\mathbf{M}\mathbf{M}^T = \mathbf{M}^T\mathbf{M} = \mathbf{I}. \tag{2.24}$$

This leads to the equivalent characterization: a matrix $\mathbf{M}$ is orthogonal if its transpose is equal to its inverse:

$$\mathbf{M}^T = \mathbf{M}^{-1}. \tag{2.25}$$

Definition (Conjugate Transpose or Hermitian Transpose): Hermitian transpose of an $m \times n$ complex matrix $\mathbf{M}$ is an $n \times m$ matrix obtained by transposing $\mathbf{M}$ and applying complex conjugate on each entry.

$$\mathbf{M}^H = \mathbf{M}^\dagger = (\mathbf{M}^T)^*. \tag{2.26}$$

For real matrices, the conjugate transpose is just the transpose, $\mathbf{M}^H = \mathbf{M}^\dagger = \mathbf{M}^T$.

Definition (Unitary Matrix): A complex square matrix $\mathbf{M}$ is unitary if its conjugate transpose $\mathbf{M}^\dagger$ is also its inverse, that is, if

$$\mathbf{M}^\dagger\mathbf{M} = \mathbf{M}\mathbf{M}^\dagger = \mathbf{M}\mathbf{M}^{-1} = \mathbf{I}. \tag{2.27}$$

Definition (Lower and Upper Triangular Matrix): A matrix of the form

$$\begin{pmatrix} l_{11} & 0 & \dots & 0 \\ l_{21} & l_{22} & \ddots & \vdots \\ \vdots & \vdots & \ddots & 0 \\ l_{n1} & l_{n2} & \dots & l_{nn} \end{pmatrix}, \tag{2.28}$$

is called a lower triangular matrix or left triangular matrix, and analogously a matrix of the form

$$\begin{pmatrix} u_{11} & u_{12} & \dots & u_{1n} \\ 0 & u_{22} & \dots & u_{2n} \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & u_{nn} \end{pmatrix}, \tag{2.29}$$

is called an upper triangular matrix or right triangular matrix. In the lower triangular matrix, all elements above the diagonal are zeros, in the upper triangular matrix, all the elements below the diagonal are zeros.

The entities that Dirac called "kets" and "bras" are simply column vectors and row vectors, respectively. In Dirac notation, "braket", ⟨⬚⟩, refers to the combination of "bra" and "ket" elements. Of course, the elements of these vectors are generally complex numbers. In this book, for convenience, we will express ourselves in terms of vectors and matrices of real numbers. Hence, in the language of matrices, these two vectors are related by simply taking the transpose. In summary, Dirac refers to a "bra," which he denoted as $\langle \mathbf{a}|$, a "ket," which he denoted as $|\mathbf{b}\rangle$, and a square matrix $\mathbf{M}$, we can associate these with vectors and matrices (in 3 dimensions) as follows;

$$\langle \mathbf{a}| = \mathbf{a}^T = (a_1, a_2, a_3), \quad |\mathbf{b}\rangle = \mathbf{b} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix}, \quad \mathbf{M} = \begin{pmatrix} M_{11} & M_{12} & M_{13} \\ M_{21} & M_{22} & M_{23} \\ M_{31} & M_{32} & M_{33} \end{pmatrix}. \tag{2.30}$$

The product of a bra and a ket, denoted by Dirac as $\langle \mathbf{a}||\mathbf{b}\rangle$ or, more commonly, by omitting one of the middle lines, as $\langle \mathbf{a}|\mathbf{b}\rangle$, is simply a number given by inner products of a row vector and a column vector in the usual way, i.e.,

$$\langle \mathbf{a}|\mathbf{b}\rangle = \mathbf{a}^T\mathbf{b} = (a_1, a_2, a_3)\begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} = a_1b_1 + a_2b_2 + a_3b_3 = \sum_{i=1}^{3} a_ib_i. \tag{2.31}$$

We can also form the product of ket times a bra, which gives a square matrix, as shown below,

$$|\mathbf{b}\rangle\langle\mathbf{a}| = \mathbf{b}\mathbf{a}^T = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix}(a_1, a_2, a_3) = \begin{pmatrix} b_1a_1 & b_1a_2 & b_1a_3 \\ b_2a_1 & b_2a_2 & b_2a_3 \\ b_3a_1 & b_3a_2 & b_3a_3 \end{pmatrix}. \tag{2.32}$$

The product of square matrix times a ket corresponds to the product of square matrix times a column vector, yielding another column vector (i.e., a ket). Mathematically, if we have a square matrix $\mathbf{M}$ of size $n \times n$ and a ket vector $|\mathbf{b}\rangle$ of size $n \times 1$, their product $\mathbf{M}|\mathbf{b}\rangle = \mathbf{M}\mathbf{b}$ results in another column vector (ket) of size $n \times 1$. The multiplication is performed by taking the dot product of each row of the matrix $\mathbf{M}$ with the column vector $|\mathbf{b}\rangle$ (i.e., using row-column multiplication or inner products of the rows with column). Each element of the resulting vector is obtained by multiplying corresponding elements of the row and the column vectors and summing up the products.

In summation notation, we have

$$\mathbf{M}|\mathbf{b}\rangle = \mathbf{M}\mathbf{b} = \begin{pmatrix} (M_{11} & M_{12} & M_{13}) \\ (M_{21} & M_{22} & M_{23}) \\ (M_{31} & M_{32} & M_{33}) \end{pmatrix}\begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} = \begin{pmatrix} M_{11}b_1 + M_{12}b_2 + M_{13}b_3 \\ M_{21}b_1 + M_{22}b_2 + M_{23}b_3 \\ M_{31}b_1 + M_{32}b_2 + M_{33}b_3 \end{pmatrix} = \begin{pmatrix} \sum_{i=1}^{3} M_{1i}b_i \\ \sum_{i=1}^{3} M_{2i}b_i \\ \sum_{i=1}^{3} M_{3i}b_i \end{pmatrix}. \tag{2.33.1}$$

In Dirac matrix notation (2.33.1) becomes,

$$\mathbf{M}|\mathbf{b}\rangle = \mathbf{M}\mathbf{b} = \begin{pmatrix} \langle \mathbf{m}_{(1)}|\mathbf{b}^{(1)}\rangle \\ \langle \mathbf{m}_{(2)}|\mathbf{b}^{(1)}\rangle \\ \langle \mathbf{m}_{(3)}|\mathbf{b}^{(1)}\rangle \end{pmatrix}, \tag{2.33.2}$$

where,

$$\langle \mathbf{m}_{(1)}| = (M_{11} \quad M_{12} \quad M_{13}), \qquad \langle \mathbf{m}_{(2)}| = (M_{21} \quad M_{22} \quad M_{23}), \qquad \langle \mathbf{m}_{(3)}| = (M_{31} \quad M_{32} \quad M_{33}), \tag{2.34}$$

$$|\mathbf{b}^{(1)}\rangle = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix}. \tag{2.35}$$

However, we also have column-row multiplication (i.e., a linear combination of the columns using b_i). In summation notation, we have

$$\begin{aligned} \mathbf{M}|\mathbf{b}\rangle = \mathbf{M}\mathbf{b} &= \left(\begin{pmatrix} M_{11} \\ M_{21} \\ M_{31} \end{pmatrix} \begin{pmatrix} M_{12} \\ M_{22} \\ M_{32} \end{pmatrix} \begin{pmatrix} M_{13} \\ M_{23} \\ M_{33} \end{pmatrix} \right) \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} \\ &= \begin{pmatrix} M_{11} \\ M_{21} \\ M_{31} \end{pmatrix} b_1 + \begin{pmatrix} M_{12} \\ M_{22} \\ M_{32} \end{pmatrix} b_2 + \begin{pmatrix} M_{13} \\ M_{23} \\ M_{33} \end{pmatrix} b_3 \\ &= \begin{pmatrix} M_{11}b_1 \\ M_{21}b_1 \\ M_{31}b_1 \end{pmatrix} + \begin{pmatrix} M_{12}b_2 \\ M_{22}b_2 \\ M_{32}b_2 \end{pmatrix} + \begin{pmatrix} M_{13}b_3 \\ M_{23}b_3 \\ M_{33}b_3 \end{pmatrix} \\ &= \begin{pmatrix} M_{11}b_1 + M_{12}b_2 + M_{13}b_3 \\ M_{21}b_1 + M_{22}b_2 + M_{23}b_3 \\ M_{31}b_1 + M_{32}b_2 + M_{33}b_3 \end{pmatrix} = \begin{pmatrix} \sum_{i=1}^{3} M_{1i}b_i \\ \sum_{i=1}^{3} M_{2i}b_i \\ \sum_{i=1}^{3} M_{3i}b_i \end{pmatrix}. \end{aligned} \tag{2.36.1}$$

In Dirac matrix notation (2.36.1) becomes,

$$\begin{aligned}\mathbf{M}|\mathbf{b}\rangle &= \mathbf{M}\mathbf{b} \\ &= |\mathbf{m}^{(1)}\rangle\langle\mathbf{b}_{(1)}| + |\mathbf{m}^{(2)}\rangle\langle\mathbf{b}_{(2)}| + |\mathbf{m}^{(3)}\rangle\langle\mathbf{b}_{(3)}| \\ &= \sum_{i=1}^{3} |\mathbf{m}^{(i)}\rangle\langle\mathbf{b}_{(i)}|,\end{aligned} \tag{2.36.2}$$

where,

$$|\mathbf{m}^{(1)}\rangle = \begin{pmatrix} M_{11} \\ M_{21} \\ M_{31} \end{pmatrix}, \qquad |\mathbf{m}^{(2)}\rangle = \begin{pmatrix} M_{12} \\ M_{22} \\ M_{32} \end{pmatrix}, \qquad |\mathbf{m}^{(3)}\rangle = \begin{pmatrix} M_{13} \\ M_{23} \\ M_{33} \end{pmatrix}, \tag{2.37}$$

$$\langle\mathbf{b}_{(1)}| = (b_1), \qquad \langle\mathbf{b}_{(2)}| = (b_2), \qquad \langle\mathbf{b}_{(3)}| = (b_3). \tag{2.38}$$

Thus, $\mathbf{M}|\mathbf{b}\rangle$ is a linear combination of the columns of $\mathbf{M}$. This is fundamental. This thinking leads us to the column space of the matrix $\mathbf{M}$ and the idea of the rank of the matrix. The key idea is to take all combinations of the columns. All real numbers b_i are allowed - the space includes $\mathbf{M}|\mathbf{b}\rangle$ for all vectors $|\mathbf{b}\rangle$. In this way, we get infinitely many output vectors $\mathbf{M}|\mathbf{b}\rangle$.

The product of bra times a square matrix corresponds to the product of a row vector times a square matrix, which is again a row vector (i.e., a "bra"), written as

$$\begin{aligned}\langle\mathbf{a}|\mathbf{M} &= \mathbf{a}^T\mathbf{M} \\ &= (a_1, a_2, a_3)\begin{pmatrix} M_{11} & M_{12} & M_{13} \\ M_{21} & M_{22} & M_{23} \\ M_{31} & M_{32} & M_{33} \end{pmatrix} \\ &= \left(\sum\nolimits_{1=1}^{3} a_i M_{i1}\,, \sum\nolimits_{1=1}^{3} a_i M_{i2}\,, \sum\nolimits_{1=1}^{3} a_i M_{i3}\right),\end{aligned} \tag{2.39}$$

or

$$
\begin{aligned}
\langle \mathbf{a}|\mathbf{M} = \mathbf{a}^T\mathbf{M} &= (a_1, a_2, a_3)\begin{pmatrix}(M_{11} & M_{12} & M_{13})\\ (M_{21} & M_{22} & M_{23})\\ (M_{31} & M_{32} & M_{33})\end{pmatrix}\\
&= a_1(M_{11} \quad M_{12} \quad M_{13}) + a_2(M_{21} \quad M_{22} \quad M_{23}) + a_3(M_{31} \quad M_{32} \quad M_{33})\\
&= (a_1M_{11} \quad a_1M_{12} \quad a_1M_{13}) + (a_2M_{21} \quad a_2M_{22} \quad a_2M_{23}) + (a_3M_{31} \quad a_3M_{32} \quad a_3M_{33})\\
&= \left(\sum_{1=1}^{3} a_iM_{i1}\,, \sum_{1=1}^{3} a_iM_{i2}\,, \sum_{1=1}^{3} a_iM_{i3}\right)\\
&= \sum_{i=1}^{3} |\mathbf{a}^{(i)}\rangle\langle \mathbf{m}_{(i)}|,
\end{aligned}
\tag{2.40}
$$

where

$$
|\mathbf{a}^{(1)}\rangle = (a_1), \qquad |\mathbf{a}^{(2)}\rangle = (a_2), \quad |\mathbf{a}^{(3)}\rangle = (a_3). \tag{2.41}
$$

Thus, $\langle \mathbf{a}|\mathbf{M}$ is a linear combination of the rows of $\mathbf{M}$. This thinking leads us to the row space of the matrix $\mathbf{M}$ and the second definition of the rank of a matrix. The key idea is to take all combinations of the rows. All real numbers a_i are allowed - the space includes $\langle \mathbf{a}|\mathbf{M}$ for all vectors $\langle \mathbf{a}|$. In this way, we get infinitely many output vectors $\langle \mathbf{a}|\mathbf{M}$.

MULTILAYER FEED-FORWARD NEURAL NETWORKS

In the rapidly evolving landscape of AI and machine learning [49-70], Feed-Forward Neural Networks (FFNNs) stand out as foundational structures, powering a multitude of applications across various domains. This chapter serves as a comprehensive introduction to the inner workings of FFNNs, delving into essential concepts and mechanisms crucial for understanding their functionality and effectiveness. We will start by looking at the structure of a FFNN, followed by how they are trained and used for making predictions. We will also take a brief look at the loss functions that should be used in different settings, the Activation Functions (AFs) used within a neuron, and the different types of optimizers that could be used for training.

The training process is a crucial phase in the development of FFNNs, enabling them to learn from data and improve their performance over time. The training procedure can be broken down into two main components, each playing a distinct role in the network's learning process (forward propagation and back propagation).

We begin with an exploration of forward propagation in NNs, elucidating how input data traverse through the network's layers to produce output predictions. Through a step-by-step examination, readers will grasp the fundamental principles underlying the propagation of information within these intricate systems.

Subsequently, attention shifts towards understanding multilayer networks as computational graphs, offering insights into the structural organization of NNs and their representation as interconnected nodes and edges. This graphical depiction lays the groundwork for comprehending the computational processes that drive NN operations.

A pivotal aspect of NN training is automatic differentiation and its main modes [71-74]. Automatic differentiation illustrates the mechanisms through which gradients are computed efficiently, enabling the network to adapt and optimize its parameters during the learning process. Readers will gain a nuanced understanding of automatic differentiation's role in facilitating gradient-based optimization techniques.

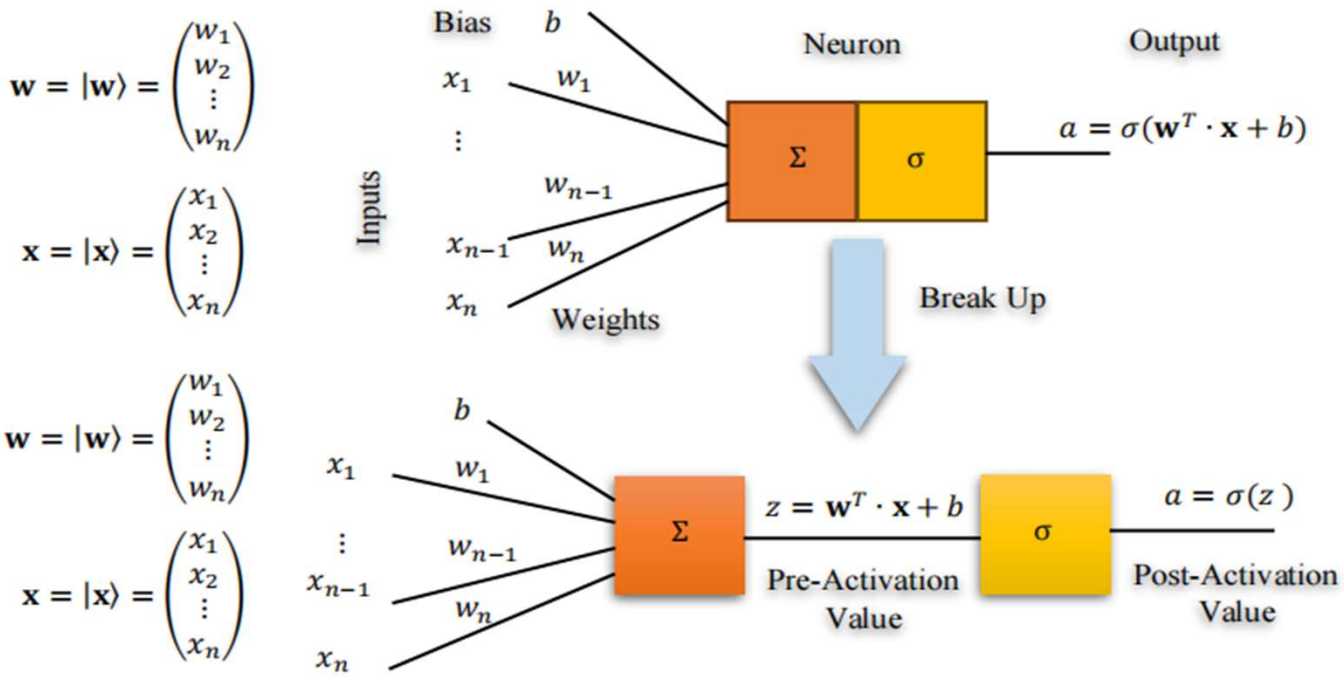

Figure 3.1. pre- and post- activation values within a neuron.

Unit/ Node/Neuron Computations

NNs are computational models inspired by the structure and functioning of the human brain. They consist of interconnected units (also known as nodes or neurons) organized into layers, see Figure 3.1 and Figure 3.2. Each node, or neuron, Figure 3.1, in a NN is a basic processing unit that receives one or more inputs, performs mathematical operations on these inputs, and produces an output. Each connection between nodes has an associated weight, which determines the strength of the connection. Additionally, each node has a bias term. The weights and biases are adjusted during the training process to minimize the difference between the predicted output and the actual output.

One single neuron has an n-dimensional input vector and has one single output signal. Mathematically, a neuron is a function that takes as input a vector $\mathbf{x} \in \mathbb{R}^n$, $\mathbf{x} = |\mathbf{x}\rangle = (x_1, x_2, \dots, x_n)^T$ and produces a scalar. A unit is parameterized by a weight vector $\mathbf{w} \in \mathbb{R}^n$, $\mathbf{w} = |\mathbf{w}\rangle = (w_1, w_2, \dots, w_n)^T$, where w_i is the weight associated with input x_i, and a bias term denoted by b, see Figure 3.1. Neurons apply an AF, σ, to the weighted sum of their inputs. The basic operations of a neuron can be represented mathematically as follows:

$$z = \sum_{i=1}^{n} w_i x_i + b = \mathbf{w}^T \cdot \mathbf{x} + b = \langle \mathbf{w}|\mathbf{x}\rangle + b, \tag{3.1.1}$$

$$a = \sigma(z). \tag{3.1.2}$$

The break-up of the neuron computations into two separate values is shown in Figure 3.1. A neuron really computes two functions within the node, which is why we have incorporated the summation symbol Σ as well as the activation symbol σ within a neuron.

- Before applying the AF, σ, the node computes a value referred to as the pre-activation value (z). This value is calculated as the weighted sum of the inputs (x_i) plus the bias (b), (3.1.1).
- The pre-activation value is then passed through an AF σ, denoted as $\sigma(z)$. Common AFs include Sigmoid $(\sigma(z) = 1/(1 + e^{-z}))$, and hyperbolic tangent $(\tanh(z))$. The result of applying the AF is the post-activation value a. This is the final output of the node. The AF allows NNs to model complex relationships in data that may not be captured by a simple linear transformation.

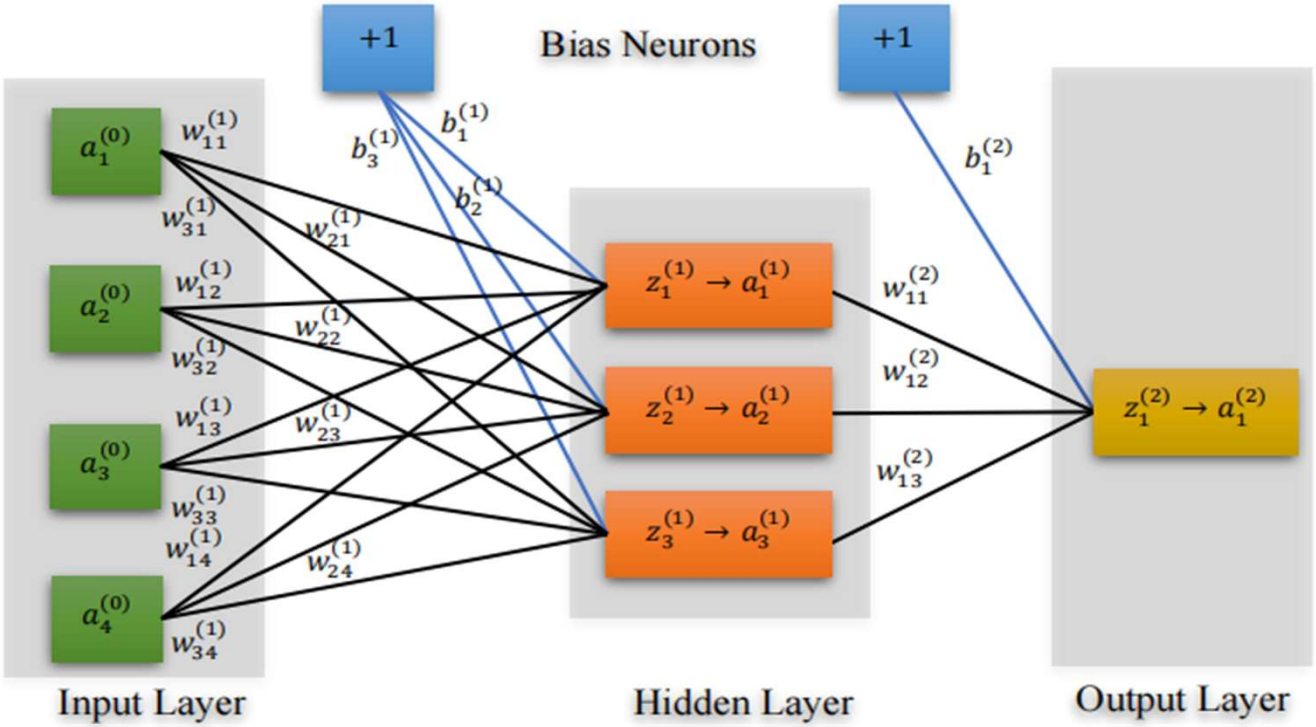

Figure 3.2. FFNN having one input layer (4 neurons), only one hidden layer (3 neurons) and one output layer (1 neuron).

An important point that emerges from Figure 3.1 is that one could treat a node with a nonlinear activation as two separate computational nodes, one for the linear transformation $z = \mathbf{w}^T \cdot \mathbf{x} + b$ and the other for the nonlinear transformation $\sigma(z)$. This conceptual separation can simplify analytical results and make it easier to understand and analyze the behavior of NNs in certain scenarios.

Feed-Forward Neural Networks

In artificial NNs, a layer is a fundamental building block that helps organize and structure the network. Each layer typically consists of a set of neurons or nodes that perform specific operations on the data. The layers work together to process input data and produce output. The basic architecture of a FFNN consists of three main types of layers: the input layer, one or more hidden layers, and the output layer. A FFNN is a type of artificial NN where the information moves in one direction— the input data is fed into the input layer, and it passes through the network layer by layer until it reaches the output layer, producing the final output. It is one of the simplest NNs. There are no cycles or loops in the network, the data flows in a forward direction. Figure 3.2 shows a FFNN having only one hidden layer. A default architecture of FFNNs assumes fully connected, also known as a fully connected layer or dense layer. It is a type of NN architecture where each node in one layer is connected to every node in the next layer. The connections are represented by weights, and each connection has its own weight parameter.

One way of representing a network of neural interconnections is as a Directed Acyclic Graph (DAG). A graph is a set of vertices or nodes (representing basic computing elements) and a set of edges (representing the connections between the nodes), where we assume that both sets are of finite size. In a directed graph (or digraph), the edges are assigned an orientation so that numerical information flows along each edge in a particular direction. An acyclic graph is one in which no loops or feedback are allowed.

Each layer type has its specific purpose and contributes to the overall functionality and performance of the NN:

- Input Layer:
  - The input layer is responsible for receiving the initial raw input data.
  - Each node in the input layer represents a feature or attribute of the input data.
  - The number of nodes in the input layer is determined by the dimensionality of the input data.

    - Each neuron in a hidden layer takes inputs from the previous layer, applies a transformation (usually a weighted sum followed by an AF), and passes the result to the next layer.
    - The number of hidden layers and the number of nodes in each hidden layer are design choices that depend on the complexity of the problem.
- Output Layer:
    - The output layer produces the final results of the NN's computation.
    - The number of nodes in the output layer depends on the nature of the task.

Remarks:

- The number of units in a layer is referred to as the width of the layer. The width of each layer need not be the same, but the dimension should be aligned, as we shall see later.
- The number of layers is referred to as the depth of the network. This is where the notion of "deep" (as in "deep learning") comes from.
- The input vector of the NN is the input vector of the neurons in the first layer. In this input vector of the network, there are $(n-1)$ externally applied signals and one bias input signal. The input vector of the neurons in the second layer consists of all outputs of the neurons in the first layer and of one bias input signal. Other layers in the multi-layer network receive their inputs only from all neurons in the previous layer and from one bias signal source.
- The output of the last layer is the output of the network and is the prediction generated based on the input.
- Bias neurons can be used both in the hidden layers and in the output layers.

Therefore, the architecture of a FFNN is defined by specifying the number of layers, the number of nodes in each layer, and the AFs.

Activation Functions

AFs play a crucial role in artificial NNs by introducing non-linearity, allowing them to model complex relationships between inputs and outputs [50] and [63]. Here are some common AFs used in NNs:

Sigmoid Function:

$$\sigma_{\text{Sigmoid}}(z) = \frac{1}{1 + e^{-z}}. \tag{3.2}$$

Hyperbolic Tangent (Tanh):

$$\sigma_{\text{Tanh}}(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}. \tag{3.3}$$

Rectified Linear Unit (ReLU):

$$\sigma_{\text{ReLU}}(z) = \max(z, 0). \tag{3.4}$$

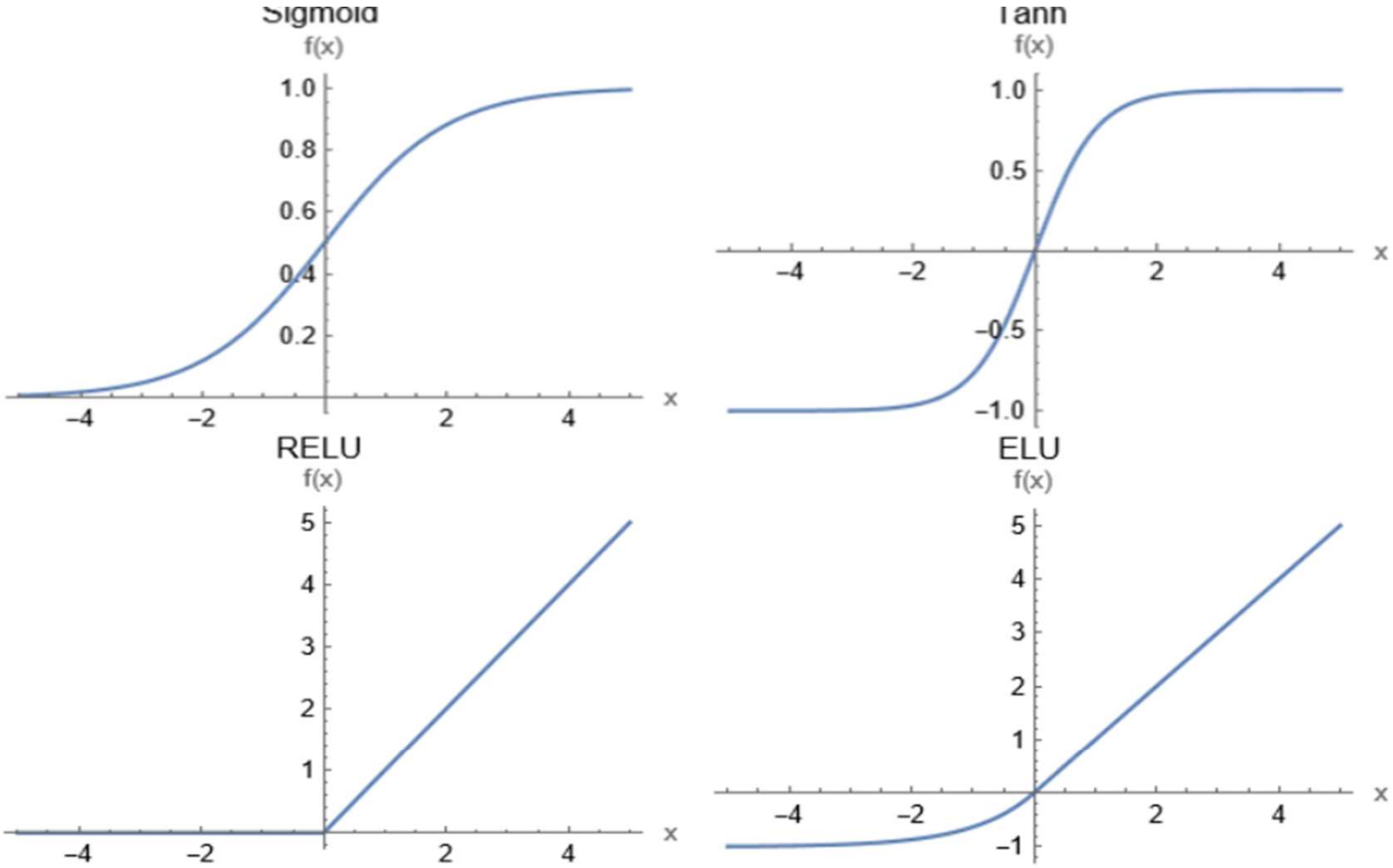

Figure 3.3. Comparison of popular AFs: Sigmoid, Tanh, ReLU, and ELU. Each function exhibits distinct characteristics essential for shaping the behavior of neurons within NNs.

Exponential Linear Unit (ELU):

$$\sigma_{\mathrm{ELU}}(z) = \begin{cases} z, & z > 0 \\ \alpha(e^{z} - 1), & z \leq 0 \end{cases}. \tag{3.5}$$

Softmax Function:

$$\sigma_{\text{Softmax}}(z_i) = a_i = \frac{e^{z_i}}{\sum_j e^{z_j}} \quad \text{for } i = 1,2,\ldots,n. \tag{3.6}$$

These functions play pivotal roles in shaping the activation patterns of neurons, influencing the learning dynamics within NNs, see Figure 3.3.

- Most NN architectures maintain uniformity in the AF used within a layer. Throughout the layers of the network, the same AF is typically applied, except for the output layer. This consistency facilitates the training process and allows for a smooth flow of information through the network.
- While this assumption holds for many traditional architectures, it is essential to note that there are variations and advancements in NN architectures, such as skip connections, attention mechanisms, and architectures like transformers, where different parts of the network might use different AFs or have more complex interactions. These advancements aim to improve the learning capabilities of NNs, especially in handling complex tasks and capturing long-range dependencies in data.

The choice of AFs is crucial and is often guided by the characteristics of the data and the underlying assumptions about the distribution of target variables. Different types of problems and data may require different AFs to ensure that the network can effectively learn and represent the patterns in the data. This adaptability in choosing AFs contributes to the flexibility and applicability of NNs across various tasks and domains. The choice of AF at the output layer depends on the nature of the task.

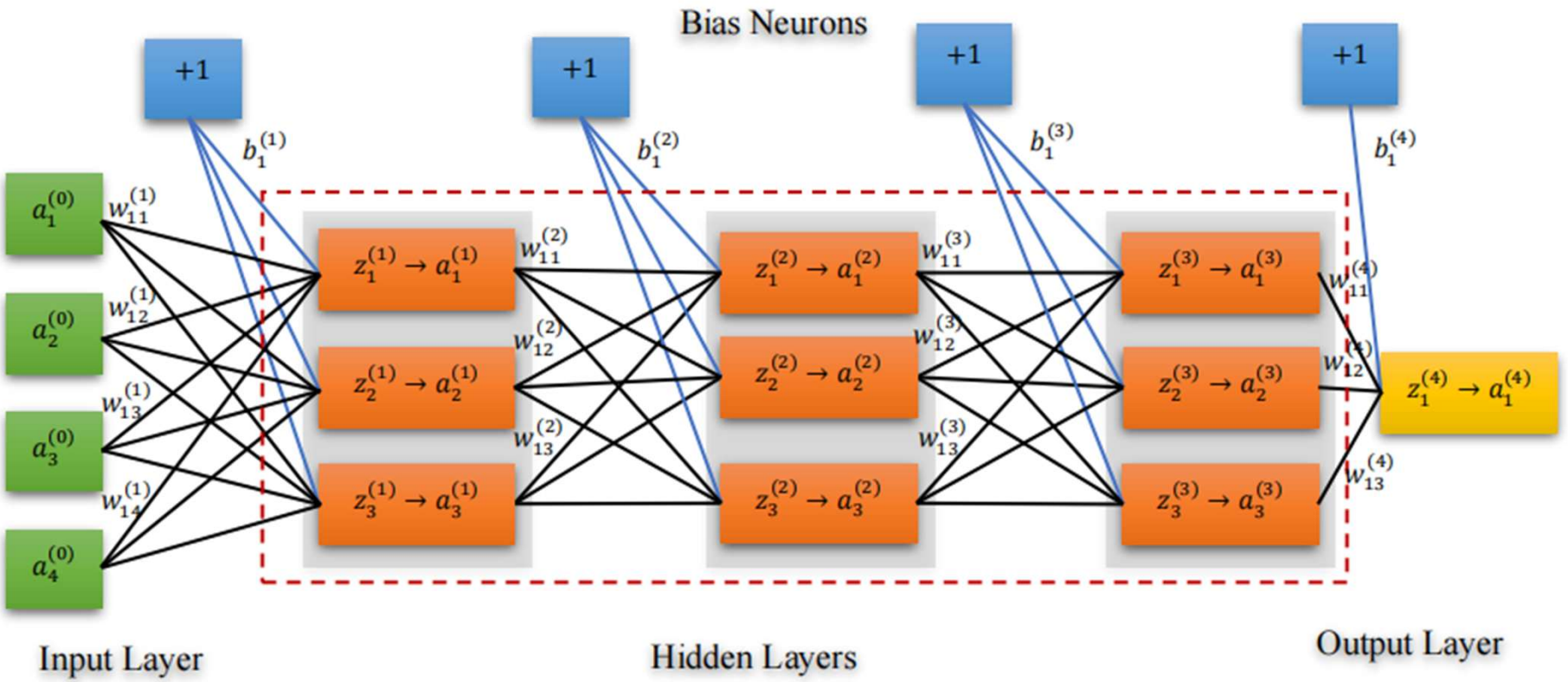

Figure 3.4. Scalar notations and architecture (the basic architecture of a FFNN with input layer, three hidden layers and output layer.) Schematic of a simple FFNN with information flowing from the left to right. The inputs $a_i^{(0)}$ are multiplied by weights, added together with the bias, and passed to the function σ to get the intermediate values $z_k^{(1)}$. The $z_k^{(1)}$ are then combined with a bias and fed to the next layer.

Deep Neural Networks

The real power of NNs arises when the network has more than a single hidden layer. NNs with multiple hidden layers are called Deep Neural Network (DNN). For example, a NN with 3 hidden layers is shown in Figure 3.4. This network has 3 hidden units per hidden layer. The term "deep" in DNNs refers to the depth of the network, which is determined by the number of hidden layers. Deeper networks can capture more complex patterns and representations in the data, allowing them to learn hierarchical features. This ability to learn intricate and abstract features makes DNNs powerful for tasks such as image recognition, natural language processing, and more.

The increasing number of connections between nodes in deeper networks also means that there are more weights and biases to be optimized during the training process. While this may pose challenges in terms of computational complexity and training time, advancements in hardware and optimization algorithms have enabled the successful training of DNNs.

Training the FFNN consists of two parts: working out what the outputs are for the given inputs and the current weights, and then updating the weights according to the error, which is a function of the difference between the outputs and the targets. These are generally known as going forwards and backwards through the network.

3.2 Forward Propagation in Neural Networks

Forward propagation is the process of passing input data through the NN to calculate an output, and it sets the stage for the subsequent steps in the training process. The term "forward" signifies the direction in which information flows through the network, from the input layer to the output layer. The following is a step-by-step explanation of the forward propagation process:

1. The process begins with the input layer, where the network receives the raw input data.
2. The weighted sum of inputs and biases is calculated for each node in the first hidden layer.
3. This value is then passed through an AF.
4. The process is repeated for each hidden layer in the network. The output from the previous layer serves as the input to the next layer.
5. The final hidden layer's output is passed through output layer to produce the network's final output. The type of AF used in the output layer depends on the nature of the problem.
6. The output is then compared to the true target values using a loss function, which measures the difference between the predicted output and the actual output.

The forward propagation process is integral to both training and making predictions with NNs. During training, the weights and biases are adjusted using optimization algorithms (e.g., GD) to minimize the loss function, enabling the network to learn from the data. Once the network is trained, it can be used to make predictions on new, unseen data by simply performing forward propagation with the learned weights and biases.

Before we delve into the intricacies of the mathematics, let us begin with a useful observation: the layers of a fully connected NN can be thought of as vector functions:

$$\mathbf{a} = \sigma(\mathbf{z}), \tag{3.7}$$

where the input to the layer is $\mathbf{z}$ and the output is $\mathbf{a}$. These are both vectors; each node in a layer produces a single scalar output, which, when grouped, becomes $\mathbf{a}$, a vector representing the output of the layer.

AFs, such as the Sigmoid function, can be applied element-wise to vector arguments. This means that the AF is independently applied to each element of the vector, producing a new vector of the same length where each element is the result of applying the AF to the corresponding element of the input vector. For example, let us say you have a vector $\mathbf{z} \in \mathbb{R}^n$, $\mathbf{z} = |\mathbf{z}\rangle = (z_1, z_2, \dots, z_n)^T$ and you want to apply AF σ element-wise to each element of the vector. The resulting vector $\mathbf{a}$ would be:

$$\mathbf{a} = |\mathbf{a}\rangle = \sigma(\mathbf{z}) = \big(\sigma(z_1), \sigma(z_2), \dots, \sigma(z_n)\big)^T. \tag{3.8}$$

This element-wise application is a common operation in NNs, where the AFs are typically applied independently to the elements of the input vectors in each layer.

Hence, one can represent NN architectures with vector variables. In vector-based representations, instead of representing individual units in a layer separately, the entire layer is treated as a vector which is represented as one rectangle. The use of rectangles is a simplification and abstraction to convey the idea of a layer with multiple units without explicitly drawing individual circles or squares for each unit, which can become impractical and cluttered in diagrams for large networks. This can make it more convenient for mathematical operations and expressions. For example, the architectural diagram in Figure 3.4 (with scalar units) has been transformed into a vector-based neural architecture in Figure 3.5.

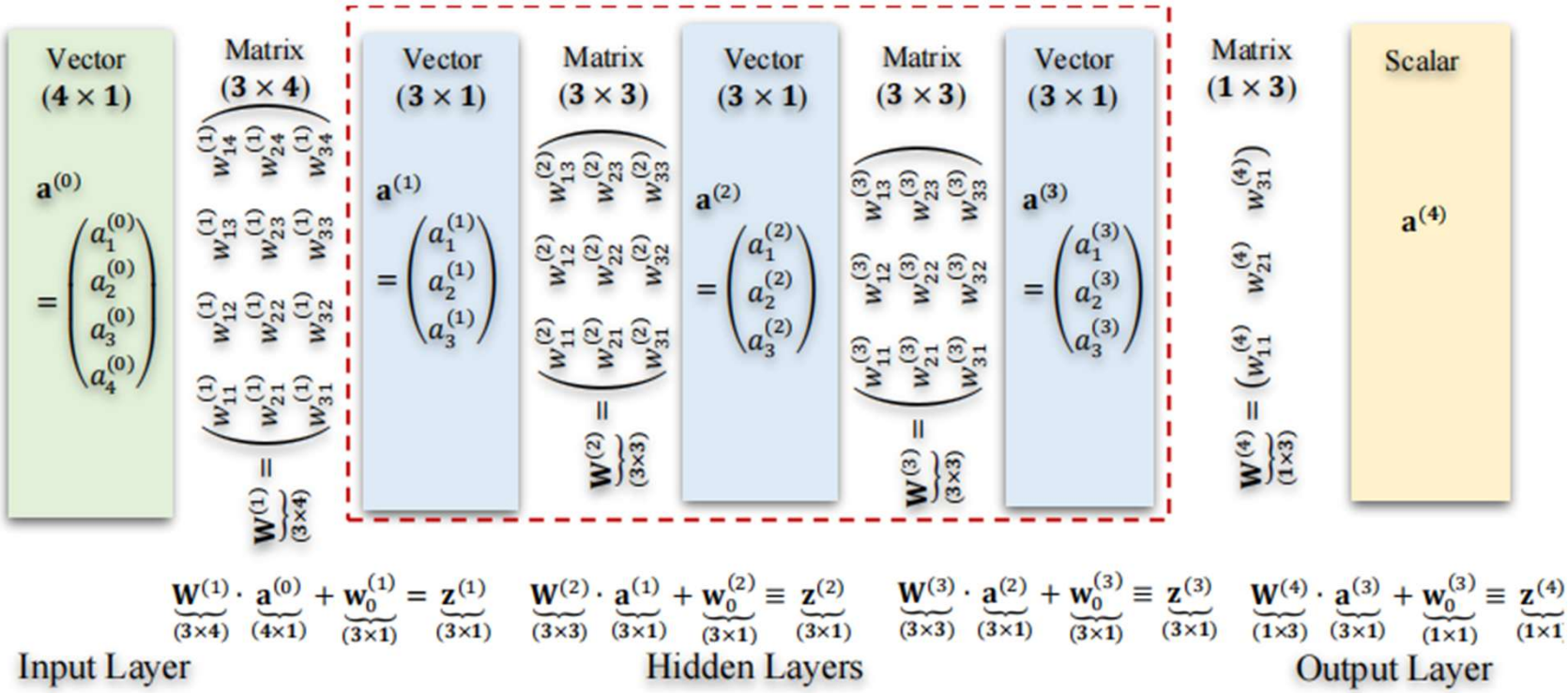

Figure 3.5. Vector notations and architecture (the basic architecture of a FFNN with input layer, three hidden layers and output layer.)

To describe forward propagation in a DNN, we need to define some notation for the weights and biases. The input layer has n_0 neurons. In this work, we adopt a systematic notation to enhance clarity and cohesion in our mathematical expressions. To maintain a consistent representation throughout the model, we denote the vector input data at the input layer as $\mathbf{a}^{(0)}$. This choice is made to unify the notation with $\mathbf{a}^{(l)}$, which represents the output of hidden layers (l). By aligning the input layer notation with that of hidden layers, we aim to streamline the understanding of our mathematical formulations. This notation strategy serves to clearly convey the layer index, providing a cohesive framework for both input and hidden layers in our models. We consider the input layer as layer number 0 of length n_0. Explicitly, the input values are denoted as $\mathbf{a}^{(0)}$, where $\mathbf{a}^{(0)}$ is an n_0-dimensional vector:

$$\mathbf{a}^{(0)} = |\mathbf{a}^{(0)}\rangle = \begin{pmatrix} a_1^{(0)} \\ a_2^{(0)} \\ \vdots \\ a_{n_0}^{(0)} \end{pmatrix}. \tag{3.9}$$

Each hidden layer (l) has n_l units, and the output layer (L) has n_L units. For each layer l, the activation values $\mathbf{a}^{(l)}$ are calculated using the weighted sum of the inputs followed by an AF:

$$\mathbf{z}^{(l)} = \mathbf{W}^{(l)} \cdot \mathbf{a}^{(l-1)} + \mathbf{b}^{(l)}, \quad \text{or} \quad |\mathbf{z}^{(l)}\rangle = \mathbf{W}^{(l)}|\mathbf{a}^{(l-1)}\rangle + |\mathbf{b}^{(l)}\rangle, \tag{3.10}$$

$$\mathbf{a}^{(l)} = \sigma^{(l)}(\mathbf{z}^{(l)}), \quad \text{or} \quad |\mathbf{a}^{(l)}\rangle = \sigma^{(l)}|\mathbf{z}^{(l)}\rangle, \tag{3.11}$$

where: $\mathbf{W}^{(l)}$ is the weight matrix for layer l, $\mathbf{b}^{(l)}$ is the bias vector for layer l, and $\sigma^{(l)}$ is the AF for layer l. The output of the NN is the activation of the last layer: $\mathbf{a}^{(L)}$. The equations above now explicitly include the bias term $\mathbf{b}^{(l)}$ in the calculation of the weighted sum $\mathbf{z}^{(l)}$. Let

$$\mathbf{W}^{(l)} = \begin{pmatrix} w_{11}^{(l)} & w_{12}^{(l)} & \cdots & w_{1n_{l-1}}^{(l)} \\ w_{21}^{(l)} & w_{22}^{(l)} & \cdots & w_{2n_{l-1}}^{(l)} \\ \vdots & \vdots & \ddots & \vdots \\ w_{n_l1}^{(l)} & w_{n_l2}^{(l)} & \cdots & w_{n_ln_{l-1}}^{(l)} \end{pmatrix}, \quad \mathbf{a}^{(l)} = |\mathbf{a}^{(l)}\rangle = \begin{pmatrix} a_1^{(l)} \\ a_2^{(l)} \\ \vdots \\ a_{n_l}^{(l)} \end{pmatrix}, \text{and} \quad \mathbf{b}^{(l)} = |\mathbf{b}^{(l)}\rangle = \begin{pmatrix} b_1^{(l)} \\ b_2^{(l)} \\ \vdots \\ b_{n_l}^{(l)} \end{pmatrix}, \tag{3.12}$$

$$\mathbf{W}^{(l)} = \begin{pmatrix} w_{11}^{(l)} & w_{12}^{(l)} & \cdots & w_{1n_{l-1}}^{(l)} \\ w_{21}^{(l)} & w_{22}^{(l)} & \cdots & w_{2n_{l-1}}^{(l)} \\ \vdots & \vdots & \ddots & \vdots \\ w_{n_l1}^{(l)} & w_{n_l2}^{(l)} & \cdots & w_{n_ln_{l-1}}^{(l)} \end{pmatrix}, \qquad \mathbf{a}^{(l)} = \left|\mathbf{a}^{(l)}\right\rangle = \begin{pmatrix} a_1^{(l)} \\ a_2^{(l)} \\ \vdots \\ a_{n_l}^{(l)} \end{pmatrix}, \text{and} \qquad \mathbf{b}^{(l)} = \left|\mathbf{b}^{(l)}\right\rangle = \begin{pmatrix} b_1^{(l)} \\ b_2^{(l)} \\ \vdots \\ b_{n_l}^{(l)} \end{pmatrix}, \tag{3.12}$$

where $w_{ij}^{(l)}$ represents the weight connecting the j-th neuron in layer $l-1$ to the i-th neuron in layer l. The dimension of $\mathbf{W}^{(l)}$ would be $n_l \times n_{l-1}$. When indexing weights, we will typically use i to indicate the destination and j to indicate source - remember weights are indexed (destination, source). We have

$$\begin{aligned} \mathbf{z}^{(l)} &= \mathbf{W}^{(l)} \cdot \mathbf{a}^{(l-1)} + \mathbf{b}^{(l)} \\ &= \mathbf{W}^{(l)}\left|\mathbf{a}^{(l-1)}\right\rangle + \left|\mathbf{b}^{(l)}\right\rangle \\ &= \begin{pmatrix} w_{11}^{(l)} & w_{12}^{(l)} & \cdots & w_{1n_{l-1}}^{(l)} \\ w_{21}^{(l)} & w_{22}^{(l)} & \cdots & w_{2n_{l-1}}^{(l)} \\ \vdots & \vdots & \ddots & \vdots \\ w_{n_l1}^{(l)} & w_{n_l2}^{(l)} & \cdots & w_{n_ln_{l-1}}^{(l)} \end{pmatrix} \begin{pmatrix} a_1^{(l-1)} \\ a_2^{(l-1)} \\ \vdots \\ a_{n_{l-1}}^{(l-1)} \end{pmatrix} + \begin{pmatrix} b_1^{(l)} \\ b_2^{(l)} \\ \vdots \\ b_{n_l}^{(l)} \end{pmatrix} \\ &= \begin{pmatrix} w_{11}^{(l)} a_1^{(l-1)} + \cdots + w_{1n_{l-1}}^{(l)} a_{n_{l-1}}^{(l-1)} + b_1^{(l)} \\ w_{21}^{(l)} a_1^{(l-1)} + \cdots + w_{2n_{l-1}}^{(l)} a_{n_{l-1}}^{(l-1)} + b_2^{(l)} \\ \vdots \\ w_{n_l1}^{(l)} a_1^{(l-1)} + \cdots + w_{n_ln_{l-1}}^{(l)} a_{n_{l-1}}^{(l-1)} + b_{n_l}^{(l)} \end{pmatrix} \end{aligned} \tag{3.13.1}$$

$$\mathbf{z}^{(l)} = \begin{pmatrix} \sum_{j=1}^{n_{l-1}} w_{1j}^{(l)} a_j^{(l-1)} + b_1^{(l)} \\ \sum_{j=1}^{n_{l-1}} w_{2j}^{(l)} a_j^{(l-1)} + b_2^{(l)} \\ \vdots \\ \sum_{j=1}^{n_{l-1}} w_{n_l j}^{(l)} a_j^{(l-1)} + b_{n_l}^{(l)} \end{pmatrix} = \begin{pmatrix} z_1^{(l)} \\ z_2^{(l)} \\ \vdots \\ z_{n_l}^{(l)} \end{pmatrix}, \tag{3.13.2}$$

where

$$z_i^{(l)} = \sum_{j=1}^{n_{l-1}} w_{ij}^{(l)} a_j^{(l-1)} + b_i^{(l)}, \tag{3.13.3}$$

and

$$\mathbf{a}^{(l)} = \left|\mathbf{a}^{(l)}\right\rangle = \sigma^{(l)}\left|\mathbf{z}^{(l)}\right\rangle = \sigma^{(l)} \begin{pmatrix} z_1^{(l)} \\ z_2^{(l)} \\ \vdots \\ z_{n_l}^{(l)} \end{pmatrix} = \begin{pmatrix} \sigma^{(l)}\left(z_1^{(l)}\right) \\ \sigma^{(l)}\left(z_2^{(l)}\right) \\ \vdots \\ \sigma^{(l)}\left(z_{n_l}^{(l)}\right) \end{pmatrix} = \begin{pmatrix} a_1^{(l)} \\ a_2^{(l)} \\ \vdots \\ a_{n_l}^{(l)} \end{pmatrix}. \tag{3.13.4}$$

The output of the NN is the activation of the last layer: $\mathbf{a}^{(L)}$.

In addition to our systematic notation for vector input data, we introduce a consistent representation for bias terms within our model architecture. Traditionally, bias terms are denoted as $b_i^{(l)}$, where (i) is the index of the neuron in

In addition to our systematic notation for vector input data, we introduce a consistent representation for bias terms within our model architecture. Traditionally, bias terms are denoted as $b_i^{(l)}$, where (i) is the index of the neuron in layer (l). However, for the sake of clarity and uniformity in our notation, we adopt the convention of using $(w_{i0}^{(l)})$ to represent the bias term associated with the neuron (i) in layer (l) and using $(\mathbf{w}_0^{(l)})$ to represent the bias vector associated with the neurons in layer (l). This choice aligns with our overarching goal of establishing a cohesive and intuitive notation system. Now, both weights and bias terms are seamlessly integrated into our notation, promoting a clearer understanding of the mathematical expressions within each layer of our model. Hence,

$$\mathbf{b}^{(l)} = \mathbf{w}_0^{(l)} = \begin{pmatrix} w_{10}^{(l)} \\ w_{20}^{(l)} \\ \vdots \\ w_{n_l 0}^{(l)} \end{pmatrix}, \tag{3.14}$$

and

$$\begin{aligned} \mathbf{z}^{(l)} &= \mathbf{W}^{(l)} \cdot \mathbf{a}^{(l-1)} + \mathbf{b}^{(l)} \\ &= \mathbf{W}^{(l)} \cdot \mathbf{a}^{(l-1)} + \mathbf{w}_0^{(l)}. \end{aligned} \tag{3.15}$$

Now that you have an idea of how each neuron is computed in the NN, you might have realized that explicitly writing out the computation on each node in each layer can be a daunting task. We generally do not express NNs in terms of the computation that happens on each node. We instead express them in terms of layers and because each layer has multiple nodes, we can write the equations in terms of vectors and matrices. Let us examine the preceding forward propagation computations in their complete matrix form for the NN illustrated in Figures 3.4 and 3.5. For the sake of simplicity, we will analyze it layer by layer. To simplify the view and to properly understand what is happening, we will now denote $\mathbf{z}^{(1)} = \mathbf{W}^{(1)} \cdot \mathbf{a}^{(0)} + \mathbf{w}_0^{(1)}$ and $\mathbf{a}^{(1)} = \sigma(\mathbf{z}^{(1)})$.

$$\mathbf{W}^{(l)} = \begin{pmatrix} w_{11}^{(l)} & w_{12}^{(l)} & \cdots & w_{1n_{l-1}}^{(l)} \\ w_{21}^{(l)} & w_{22}^{(l)} & \cdots & w_{2n_{l-1}}^{(l)} \\ \vdots & \vdots & \ddots & \vdots \\ w_{n_l1}^{(l)} & w_{n_l2}^{(l)} & \cdots & w_{n_ln_{l-1}}^{(l)} \end{pmatrix}, \quad \mathbf{a}^{(l)} = \left|\mathbf{a}^{(l)}\right\rangle = \begin{pmatrix} a_1^{(l)} \\ a_2^{(l)} \\ \vdots \\ a_{n_l}^{(l)} \end{pmatrix}, \text{ and } \quad \mathbf{b}^{(l)} = \left|\mathbf{b}^{(l)}\right\rangle = \begin{pmatrix} b_1^{(l)} \\ b_2^{(l)} \\ \vdots \\ b_{n_l}^{(l)} \end{pmatrix}, \tag{3.12}$$

where $w_{ij}^{(l)}$ represents the weight connecting the j-th neuron in layer $l-1$ to the i-th neuron in layer l. The dimension of $\mathbf{W}^{(l)}$ would be $n_l \times n_{l-1}$. When indexing weights, we will typically use i to indicate the destination and j to indicate source - remember weights are indexed (destination, source). We have

$$\begin{aligned} \mathbf{z}^{(l)} &= \mathbf{W}^{(l)} \cdot \mathbf{a}^{(l-1)} + \mathbf{b}^{(l)} \\ &= \mathbf{W}^{(l)}\left|\mathbf{a}^{(l-1)}\right\rangle + \left|\mathbf{b}^{(l)}\right\rangle \\ &= \begin{pmatrix} w_{11}^{(l)} & w_{12}^{(l)} & \cdots & w_{1n_{l-1}}^{(l)} \\ w_{21}^{(l)} & w_{22}^{(l)} & \cdots & w_{2n_{l-1}}^{(l)} \\ \vdots & \vdots & \ddots & \vdots \\ w_{n_l1}^{(l)} & w_{n_l2}^{(l)} & \cdots & w_{n_ln_{l-1}}^{(l)} \end{pmatrix} \begin{pmatrix} a_1^{(l-1)} \\ a_2^{(l-1)} \\ \vdots \\ a_{n_{l-1}}^{(l-1)} \end{pmatrix} + \begin{pmatrix} b_1^{(l)} \\ b_2^{(l)} \\ \vdots \\ b_{n_l}^{(l)} \end{pmatrix} \\ &= \begin{pmatrix} w_{11}^{(l)} a_1^{(l-1)} + \cdots + w_{1n_{l-1}}^{(l)} a_{n_{l-1}}^{(l-1)} + b_1^{(l)} \\ w_{21}^{(l)} a_1^{(l-1)} + \cdots + w_{2n_{l-1}}^{(l)} a_{n_{l-1}}^{(l-1)} + b_2^{(l)} \\ \vdots \\ w_{n_l1}^{(l)} a_1^{(l-1)} + \cdots + w_{n_ln_{l-1}}^{(l)} a_{n_{l-1}}^{(l-1)} + b_{n_l}^{(l)} \end{pmatrix} \end{aligned} \tag{3.13.1}$$

$$\mathbf{a}^{(1)} = |\mathbf{a}^{(1)}\rangle = \underbrace{\begin{pmatrix} a_1^{(1)} \\ a_2^{(1)} \\ a_3^{(1)} \end{pmatrix}}_{3\times 1} = \sigma(\mathbf{z}^{(1)}) = \underbrace{\begin{pmatrix} \sigma(z_1^{(1)}) \\ \sigma(z_2^{(1)}) \\ \sigma(z_3^{(1)}) \end{pmatrix}}_{3\times 1}. \tag{3.16.2}$$

Calculate $\mathbf{z}^{(2)}$ and $\mathbf{a}^{(2)}$ as follows:

$$\mathbf{z}^{(2)} = |\mathbf{z}^{(2)}\rangle = \underbrace{\begin{pmatrix} z_1^{(2)} \\ z_2^{(2)} \\ z_3^{(2)} \end{pmatrix}}_{3\times 1} = \underbrace{\begin{pmatrix} w_{11}^{(2)} & w_{12}^{(2)} & w_{13}^{(2)} \\ w_{21}^{(2)} & w_{22}^{(2)} & w_{23}^{(2)} \\ w_{31}^{(2)} & w_{32}^{(2)} & w_{33}^{(2)} \end{pmatrix}}_{3\times 3} \underbrace{\begin{pmatrix} a_1^{(1)} \\ a_2^{(1)} \\ a_3^{(1)} \end{pmatrix}}_{3\times 1} + \underbrace{\begin{pmatrix} w_{10}^{(2)} \\ w_{20}^{(2)} \\ w_{30}^{(2)} \end{pmatrix}}_{3\times 1}, \tag{3.17.1}$$

$$\mathbf{a}^{(2)} = |\mathbf{a}^{(2)}\rangle = \underbrace{\begin{pmatrix} a_1^{(2)} \\ a_2^{(1)} \\ a_3^{(2)} \end{pmatrix}}_{3\times 1} = \sigma(\mathbf{z}^{(2)}) = \underbrace{\begin{pmatrix} \sigma(z_1^{(2)}) \\ \sigma(z_2^{(2)}) \\ \sigma(z_3^{(2)}) \end{pmatrix}}_{3\times 1}. \tag{3.17.2}$$

Calculate $\mathbf{z}^{(3)}$ and $\mathbf{a}^{(3)}$ as follows:

$$\mathbf{z}^{(3)} = |\mathbf{z}^{(3)}\rangle = \underbrace{\begin{pmatrix} z_1^{(3)} \\ z_2^{(3)} \\ z_3^{(3)} \end{pmatrix}}_{3\times 1} = \underbrace{\begin{pmatrix} w_{11}^{(3)} & w_{12}^{(3)} & w_{13}^{(3)} \\ w_{21}^{(3)} & w_{22}^{(3)} & w_{23}^{(3)} \\ w_{31}^{(3)} & w_{32}^{(3)} & w_{33}^{(3)} \end{pmatrix}}_{3\times 3} \underbrace{\begin{pmatrix} a_1^{(2)} \\ a_2^{(2)} \\ a_3^{(2)} \end{pmatrix}}_{3\times 1} + \underbrace{\begin{pmatrix} w_{10}^{(3)} \\ w_{20}^{(3)} \\ w_{30}^{(3)} \end{pmatrix}}_{3\times 1}, \tag{3.18.1}$$

$$\mathbf{a}^{(3)} = |\mathbf{a}^{(3)}\rangle = \underbrace{\begin{pmatrix} a_1^{(3)} \\ a_2^{(3)} \\ a_3^{(3)} \end{pmatrix}}_{3\times 1} = \sigma(\mathbf{z}^{(3)}) = \underbrace{\begin{pmatrix} \sigma(z_1^{(3)}) \\ \sigma(z_2^{(3)}) \\ \sigma(z_3^{(3)}) \end{pmatrix}}_{3\times 1}. \tag{3.18.2}$$

Calculate $\mathbf{z}^{(4)}$ and $\mathbf{a}^{(4)}$ as follows:

$$\mathbf{z}^{(4)} = \left|\mathbf{z}^{(4)}\right\rangle = \underbrace{\left(z_1^{(4)}\right)}_{1\times1} = \underbrace{\begin{pmatrix} w_{11}^{(4)} & w_{21}^{(4)} & w_{31}^{(4)} \end{pmatrix}}_{1\times3} \underbrace{\begin{pmatrix} a_1^{(3)} \\ a_2^{(3)} \\ a_3^{(3)} \end{pmatrix}}_{3\times1} + \underbrace{w_{10}^{(4)}}_{1\times1}, \tag{3.19.1}$$

$$\mathbf{a}^{(4)} = \left|\mathbf{a}^{(4)}\right\rangle = \underbrace{\left(a_1^{(4)}\right)}_{1\times1} = \sigma\left(\mathbf{z}^{(4)}\right) = \underbrace{\sigma\left(z_1^{(4)}\right)}_{1\times1}. \tag{3.19.2}$$

Those are all the operations that take place in our FFNN illustrated in Figures 3.4 and 3.5. we have slightly tweaked the preceding notation by putting in brackets and writing $\mathbf{a}^{(4)}$ as a vector, even though it is clearly a scalar. This was only done to keep the flow and to avoid changing the notation.

In linear algebra, when a matrix or vector is multiplied by another matrix, the resulting matrix or vector may have different dimensions. This multiplication operation can be seen as a mapping, where points in one space are transformed to points in another space. In NN, various operations are carried out on input vectors. These operations involve matrix multiplications, activations functions, and possibly biases. The composition of these operations essentially represents a series of transformations on the input data. The input vector is transformed through the layers of the network, involving matrix multiplications and non-linear activations, ultimately mapping the input from one Euclidean space to the output in another Euclidean space. Hence, NNs can be conceptualized as mappings between different Euclidean spaces, where the layers of the network act as transformation functions. Matrix multiplication is a key operation in this process, contributing to the mapping of input data to output predictions.

Using this observation, we can generalize and write the following:

$$\mathcal{N}: \mathbb{R}^{n_0} \to \mathbb{R}^{n_L}, \tag{3.20}$$

where $\mathcal{N}$ represents the NN, which is a function composed of multiple layers. $\mathbb{R}^{n_0}$ denotes the input space, where n_0 is the dimensionality of the input layer. This is the space of all possible input vectors. $\mathbb{R}^{n_L}$ represents the output space, where n_L is the dimensionality of the output layer. This is the space of all possible output vectors. The function $\mathcal{N}$ encapsulates the entire architecture of the NN, which consists of an input layer, hidden layers, and an output layer. Each layer involves matrix multiplication with weights, addition of biases, and AFs. Now, we can summarize our NN of Figure 3.4 in the following equations:

$$\mathcal{N}: \mathbb{R}^4 \rightarrow \mathbb{R}^1. \tag{3.21.1}$$

$$\mathcal{N}(\mathbf{a}^{(0)}) = \sigma^{(4)}\left(\overbrace{\mathbf{W}^{(4)}}^{1\times 3}\left[\sigma^{(3)}\left(\overbrace{\mathbf{W}^{(3)}}^{3\times 3}\left[\sigma^{(2)}\left(\overbrace{\mathbf{W}^{(2)}}^{3\times 3}\left[\sigma^{(1)}\left(\underbrace{\overbrace{\mathbf{W}^{(1)}}^{3\times 4}\overbrace{\mathbf{a}^{(0)}}^{4\times 1}+\overbrace{\mathbf{w}_0^{(1)}}^{3\times 1}}_{\mathbf{z}^{(1)}}\right)\right]}^{\mathbf{a}^{(1)}}+\overbrace{\mathbf{w}_0^{(2)}}^{3\times 1}\right)\right]}^{\mathbf{z}^{(2)}}+\overbrace{\mathbf{w}_0^{(3)}}^{3\times 1}\right)\right]}^{\mathbf{a}^{(2)}}+\overbrace{\mathbf{w}_0^{(4)}}^{3\times 1}\right)}^{\mathbf{z}^{(3)}}\Bigg)}^{\mathbf{a}^{(3)}}\Bigg)}^{\mathbf{z}^{(4)}}\Bigg)}^{\mathbf{a}^{(4)}}\Bigg)}^{1\times 1}. \tag{3.21.2}$$

Forward Propagation for All Training Examples

Let $\mathbf{X}$ be the matrix representing the input data for all training examples, where each column corresponds to a different example:

$$\mathbf{X} = \begin{pmatrix} | & | & & | \\ \mathbf{x}^{(1)} & \mathbf{x}^{(2)} & \cdots & \mathbf{x}^{(m)} \\ | & | & & | \end{pmatrix}. \tag{3.22}$$

Here, $\mathbf{x}^{(i)}$ represents the input vector for the i-th training example, and m is the number of training examples, see Figure 3.7. $\mathbf{X}$ is $(n_0 \times m)$ matrix, where n_0 is the number of input features, and m is the number of training examples. The entire forward propagation process can be expressed in vectorized form for all training examples (if you are working with a batch of examples):

$$\mathbf{Z}^{(l)} = \mathbf{W}^{(l)} \cdot \mathbf{A}^{(l-1)} + \mathbf{W}_0^{(l)}, \tag{3.23}$$

$$\mathbf{A}^{(l)} = \sigma\left(\mathbf{Z}^{(l)}\right), \tag{3.24}$$

where: $\mathbf{A}^{(0)} = \mathbf{X}$ is the input data, $\mathbf{A}^{(l)}$ is the $(n_l \times m)$ activation matrix for layer l, and $\mathbf{Z}^{(l)}$ is the $(n_l \times m)$ weighted sum matrix for layer l. For each layer l, $\mathbf{W}^{(l)}$ is $(n_l \times n_{l-1})$ weight matrix. $\mathbf{W}_0^{(l)}$ is $(n_l \times m)$ bias matrix for layer l. The final output of the network is given by $\mathbf{A}^{(L)}$, and its dimensions depend on the number of output units: $(n_L \times m)$ activation matrix for the output layer.

Forward Propagation and Bias Term (for AF $\sigma(1) \neq 1$)

To incorporate the bias term into the weight matrix and the activation vector, we can introduce a dummy input neuron with a fixed value of 1. This way, the bias term becomes a part of the weight matrix, and the activation vector includes the bias term implicitly. Let us denote the augmented weight matrix for layer l as $\widehat{\mathbf{W}}^{(l)}$ and the augmented activation vector as $\hat{\mathbf{a}}^{(l)}$. The equations are as follows. The augmented weight matrix becomes

$$\widehat{\mathbf{W}}^{(l)} = \begin{pmatrix} w_{10}^{(l)} & w_{11}^{(l)} & w_{12}^{(l)} & \cdots & w_{1n_{l-1}}^{(l)} \\ w_{20}^{(l)} & w_{21}^{(l)} & w_{22}^{(l)} & \cdots & w_{2n_{l-1}}^{(l)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ w_{n_l0}^{(l)} & w_{n_l1}^{(l)} & w_{n_l2}^{(l)} & \cdots & w_{n_ln_{l-1}}^{(l)} \end{pmatrix}, \tag{3.25}$$

where $w_{i0}^{(l)}$ is the bias term for the i-th neuron in layer l, and $w_{ij}^{(l)}$ represents the weight connecting the j-th neuron in layer $l-1$ to the i-th neuron in layer l. The augmented activation vector becomes

$$\hat{\mathbf{a}}^{(l)} = \begin{pmatrix} 1 \\ a_1^{(l)} \\ a_2^{(l)} \\ \vdots \\ a_{n_l}^{(l)} \end{pmatrix}. \tag{3.26}$$

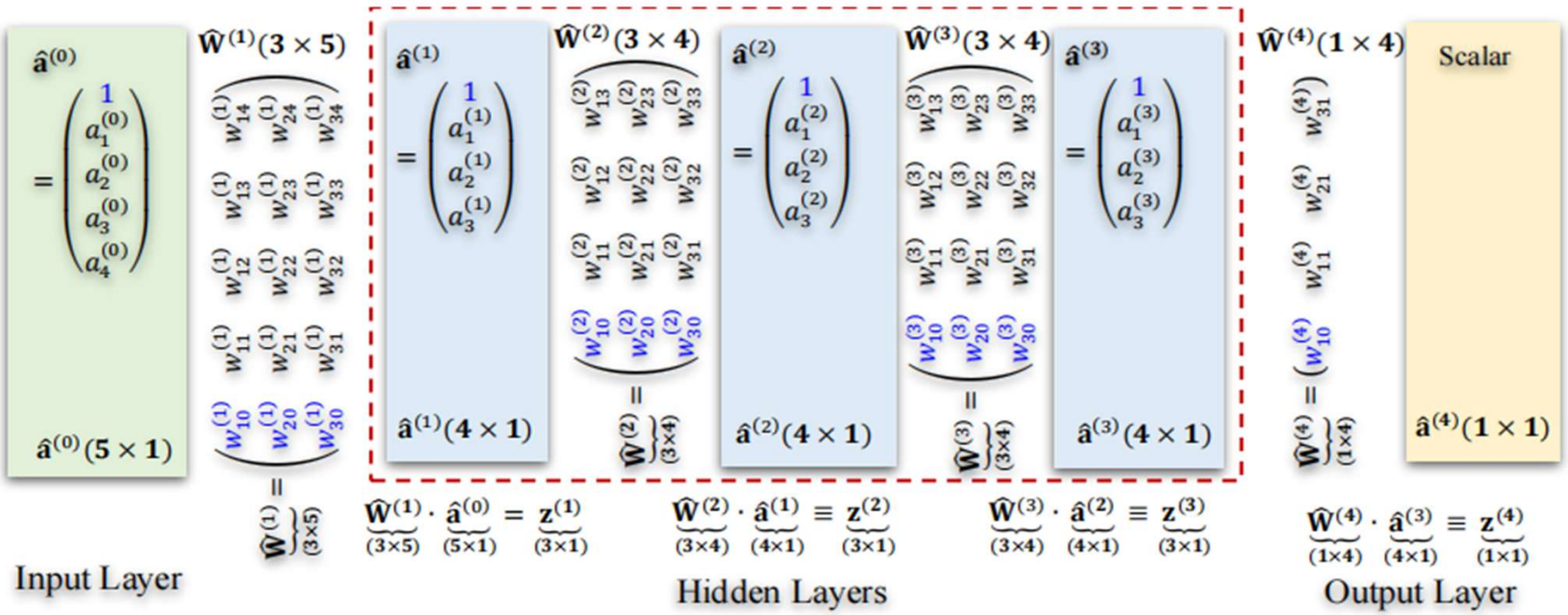

Figure 3.8. Vector notations and architecture (the basic architecture of a FFNN with input layer, three hidden layers and output layer.) (without bias term in the case $\sigma(1) \neq 1$)

Here, the first element is fixed at 1, representing the bias term. With these augmentations, the equations for forward propagation become:

$$\mathbf{z}^{(l)} = \widehat{\mathbf{W}}^{(l)} \cdot \hat{\mathbf{a}}^{(l-1)}, \tag{3.27}$$

$$\mathbf{a}^{(l)} = \sigma\left(\mathbf{z}^{(l)}\right). \tag{3.28}$$

This way, the bias term is treated as just another weight in the weight matrix, and the activation vector includes the bias implicitly, see Figure 3.8. Now, it is clear, why we use the notation $w_{i0}^{(l)}$ with bias term. This formulation simplifies the implementation and helps maintain consistency in the mathematical expressions.

Moreover, let us express the entire forward propagation process in vectorized form for all training examples. Introduce a row of ones to represent the bias term in the input layer:

$$\widehat{\mathbf{X}} = \begin{pmatrix} 1 & 1 & \dots & 1 \\ \mathbf{x}^{(1)} & \mathbf{x}^{(2)} & \dots & \mathbf{x}^{(m)} \end{pmatrix}. \tag{3.29}$$

Now, $\widehat{\mathbf{X}}$ includes the bias term. $\widehat{\mathbf{X}}$ is $(n_0 + 1) \times m$ matrix (the input layer with an additional row for the bias term).

Simillary,

$$\widehat{\mathbf{A}}^{(l)} = \begin{pmatrix} 1 & 1 & \dots & 1 \\ \mathbf{a}_1^{(l)} & \mathbf{a}_2^{(l)} & \dots & \mathbf{a}_m^{(l)} \end{pmatrix}. \tag{3.30}$$

Here, $\mathbf{a}_i^{(l)}$ represents the activation vector of layer (l) for the i-th training example, and m is the number of training examples. For each layer l, the weighted sum $\mathbf{Z}^{(l)}$ and activation $\mathbf{A}^{(l)}$ can be calculated as follows:

$$\mathbf{Z}^{(l)} = \widehat{\mathbf{W}}^{(l)} \cdot \widehat{\mathbf{A}}^{(l-1)}, \tag{3.31}$$

$$\mathbf{A}^{(l)} = \sigma\left(\mathbf{Z}^{(l)}\right). \tag{3.32}$$

$\widehat{\mathbf{W}}^{(l)}$ is $n_l \times (n_{l-1} + 1)$ weight matrix for layer l, including bias weights. $\mathbf{Z}^{(l)}$ is $n_l \times m$ weighted sum matrix for layer l. $\widehat{\mathbf{A}}^{(l)}$ is $(n_l + 1) \times m$ activation matrix for layer l, including the bias term. The final output of the network is given by $\widehat{\mathbf{A}}^{(L)}$, and its dimensions depend on the number of output units: $\widehat{\mathbf{A}}^{(L)}$ is $(n_L + 1) \times m$ activation matrix for the output layer.

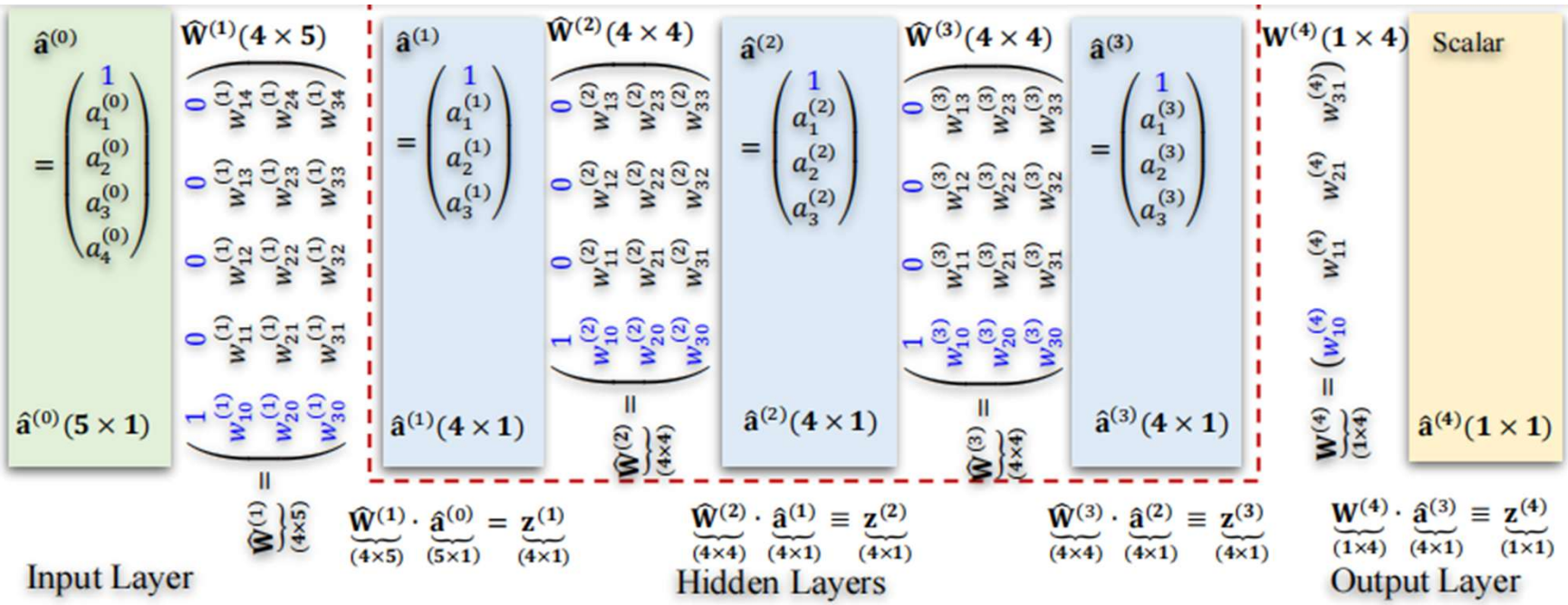

Figure 3.9. Vector notations and architecture (the basic architecture of a FFNN with input layer, three hidden layers and output layer.) (without bias term in the case $\sigma(1) = 1$)

$$\bar{\mathbf{W}}^{(l)} = \begin{pmatrix} 1 & 0 & 0 & \dots & 0 \\ w_{10}^{(l)} & w_{11}^{(l)} & w_{12}^{(l)} & \dots & w_{1n_{l-1}}^{(l)} \\ w_{20}^{(l)} & w_{21}^{(l)} & w_{22}^{(l)} & \dots & w_{2n_{l-1}}^{(l)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ w_{n_l0}^{(l)} & w_{n_l1}^{(l)} & w_{n_l2}^{(l)} & \dots & w_{n_ln_{l-1}}^{(l)} \end{pmatrix} = \begin{pmatrix} 1 & \mathbf{0}^T \\ \mathbf{w}_0^{(l)} & \mathbf{W}^{(l)} \end{pmatrix}, \tag{3.33}$$

where $w_{i0}^{(l)}$ is the bias term for the i-th neuron in layer l, $w_{ij}^{(l)}$ represents the weight connecting the j-th neuron in layer $l-1$ to the i-th neuron in layer l, $\mathbf{0}^T$ is the transpose of zero vector with dimension $1 \times n_{l-1}$, $\mathbf{w}_0^{(l)}$ represent the bias vector associated with the neurons in layer l and $\mathbf{W}^{(l)}$ is the weight matrix for layer l with $n_l \times n_{l-1}$ dimension. The augmented activation vector becomes

$$\hat{\mathbf{a}}^{(l)} = \begin{pmatrix} 1 \\ a_1^{(l)} \\ a_2^{(l)} \\ \vdots \\ a_{n_l}^{(l)} \end{pmatrix} = \begin{pmatrix} 1 \\ \mathbf{a}^{(l)} \end{pmatrix}. \tag{3.34}$$

Here, the first element is fixed at 1, representing the bias term. For each layer of the neural net, the top entry of activation vector for layer l is now fixed at 1. After multiplying that activation vector by $\bar{\mathbf{W}}^{(l)}$, the top entry of activation vector on the next layer is still 1, because $\sigma(1) = \sigma_{\mathrm{ReLU}}(1) = 1$. With these augmentations, the equations for forward propagation become:

$$\hat{\mathbf{z}}^{(l)} = \bar{\mathbf{W}}^{(l)} \cdot \hat{\mathbf{a}}^{(l-1)}, \tag{3.35}$$

$$\hat{\mathbf{a}}^{(l)} = \sigma\left(\hat{\mathbf{z}}^{(l)}\right), \tag{3.36}$$

and final output of the network is given by

$$\hat{\mathbf{z}}^{(L)} = \widehat{\mathbf{W}}^{(L)} \cdot \hat{\mathbf{a}}^{(L-1)}, \tag{3.37}$$

$$\hat{\mathbf{a}}^{(L)} = \sigma\left(\hat{\mathbf{z}}^{(L)}\right), \tag{3.38}$$

where $\bar{\mathbf{W}}^{(l)}$ is $(n_l + 1) \times (n_{l-1} + 1)$ weight matrix for layer l, including bias weights. $\hat{\mathbf{z}}^{(l)}$ is $(n_l + 1) \times 1$ weighted sum vector for layer l. $\hat{\mathbf{a}}^{(l)}$ is $(n_l + 1) \times 1$ activation vector for layer l, including the bias term. $\widehat{\mathbf{W}}^{(L)}$ is the $n_L \times (n_{L-1} + 1)$ weight matrix for layer L i.e., without the row (1 0 ... 0), because there is no existence of next layer. This way, the bias term is treated as just another weight in the weight matrix, and the activation vector includes the bias implicitly, see Figure 3.9. Finally, let us express the entire forward propagation process in vectorized form for all training examples. Introduce a row of ones to represent the bias term:

$$\widehat{\mathbf{A}}^{(l)} = \begin{pmatrix} 1 & 1 & \dots & 1 \\ \mathbf{a}_1^{(l)} & \mathbf{a}_2^{(l)} & \dots & \mathbf{a}_m^{(l)} \end{pmatrix}. \tag{3.39}$$

Now, $\widehat{\mathbf{A}}^{(l)}$ includes the bias term. $\widehat{\mathbf{A}}^{(l)}$ is $(n_l + 1) \times m$ matrix (the layer with an additional row for the bias term). Here, $\mathbf{a}_i^{(l)}$ represents the activation vector of layer (l) for the i-th training example, and m is the number of training examples. For each layer l, the weighted sum $\widehat{\mathbf{Z}}^{(l)}$ and activation $\widehat{\mathbf{A}}^{(l)}$ can be calculated as follows:

$$\widehat{\mathbf{Z}}^{(l)} = \overline{\mathbf{W}}^{(l)} \cdot \widehat{\mathbf{A}}^{(l-1)}, \qquad \widehat{\mathbf{A}}^{(l)} = \sigma\left(\widehat{\mathbf{Z}}^{(l)}\right), \qquad \widehat{\mathbf{Z}}^{(L)} = \overline{\mathbf{W}}^{(L)} \cdot \widehat{\mathbf{A}}^{(L-1)}, \qquad \widehat{\mathbf{A}}^{(L)} = \sigma\left(\widehat{\mathbf{Z}}^{(L)}\right). \tag{3.40}$$

$\overline{\mathbf{W}}^{(l)}$ is $(n_l + 1) \times (n_{l-1} + 1)$ weight matrix for layer l, including bias weights. $\widehat{\mathbf{Z}}^{(l)}$ is $(n_l + 1) \times m$ weighted sum matrix for layer l. $\widehat{\mathbf{A}}^{(l)}$ is $(n_l + 1) \times m$ activation matrix for layer l, including the bias term.

Now, we can summarize our NN of Figure 3.9 in the following equations:

$$\mathcal{N}: \mathbb{R}^4 \to \mathbb{R}^1, \tag{3.41.1}$$

$$\mathcal{N}\left(\mathbf{a}^{(0)}\right) = \sigma^{(4)}\left(\overbrace{\mathbf{W}^{(4)}}^{1\times4} \cdot \left[\sigma^{(3)}\left(\overbrace{\mathbf{W}^{(3)}}^{4\times4} \cdot \left[\sigma^{(2)}\left(\overbrace{\mathbf{W}^{(2)}}^{4\times4} \cdot \left[\sigma^{(1)}\left(\overbrace{\mathbf{W}^{(1)}}^{4\times5} \cdot \overbrace{\mathbf{a}^{(0)}}^{5\times1}\right)\right]_{\mathbf{z}^{(1)}}\right)\right]_{\mathbf{a}^{(1)}}\right)\right]_{\mathbf{z}^{(2)}}\right)\right]_{\mathbf{a}^{(2)}}\right)\Bigg)_{\mathbf{z}^{(3)}}\Bigg)_{\mathbf{a}^{(3)}}\Bigg)_{\mathbf{z}^{(4)}}\Bigg)_{\mathbf{a}^{(4)}}\Bigg)_{1\times1}. \tag{3.41.2}$$

3.3 Multilayer Network as a Computational Graph

As you increase the depth and width of a NN, the composition of functions becomes more complex, leading to an exponentially growing expression for the overall function. This is due to the fact that each additional node or edge in the NN introduces new terms and dependencies, leading to an exponential increase in the mathematical representation. Writing out this function in closed form quickly becomes impractical, and even if you could write it down, the resulting expression would be incredibly long and difficult to manage. For example, the global function evaluated by the NN of Figure 3.4 is as follows:

$$
\begin{aligned}
&\mathcal{N}\left(\mathbf{a}^{(0)}\right) \\
&= \sigma^{(4)}\left(\mathbf{W}^{(4)} \cdot \left[\sigma^{(3)}\left(\mathbf{W}^{(3)} \cdot \left[\sigma^{(2)}\left(\mathbf{W}^{(2)} \cdot \left[\sigma^{(1)}\left(\mathbf{W}^{(1)} \cdot \mathbf{a}^{(0)} + \mathbf{w}_0^{(1)}\right)\right] + \mathbf{w}_0^{(2)}\right)\right] + \mathbf{w}_0^{(3)}\right)\right] + \mathbf{w}_0^{(4)}\right) \\
&= \sigma^{(4)}\left(\underbrace{\begin{pmatrix} w_{11}^{(4)} & w_{21}^{(4)} & w_{31}^{(4)} \end{pmatrix}}_{1\times 3} \cdot \left[\sigma^{(3)}\left(\underbrace{\begin{pmatrix} w_{11}^{(3)} & w_{12}^{(3)} & w_{13}^{(3)} \\ w_{21}^{(3)} & w_{22}^{(3)} & w_{23}^{(3)} \\ w_{31}^{(3)} & w_{32}^{(3)} & w_{33}^{(3)} \end{pmatrix}}_{3\times 3} \cdot \left[\sigma^{(2)}\left(\underbrace{\begin{pmatrix} w_{11}^{(2)} & w_{12}^{(2)} & w_{13}^{(2)} \\ w_{21}^{(2)} & w_{22}^{(2)} & w_{23}^{(2)} \\ w_{31}^{(2)} & w_{32}^{(2)} & w_{33}^{(2)} \end{pmatrix}}_{3\times 3}\right.\right.\right. \right. \\
&\left.\left.\left. \cdot \left[\sigma^{(1)}\left(\underbrace{\begin{pmatrix} w_{11}^{(1)} & w_{12}^{(1)} & w_{13}^{(1)} & w_{14}^{(1)} \\ w_{21}^{(1)} & w_{22}^{(1)} & w_{23}^{(1)} & w_{24}^{(1)} \\ w_{31}^{(1)} & w_{32}^{(1)} & w_{33}^{(1)} & w_{34}^{(1)} \end{pmatrix}}_{3\times 4} \underbrace{\begin{pmatrix} a_1^{(0)} \\ a_2^{(0)} \\ a_3^{(0)} \\ a_4^{(0)} \end{pmatrix}}_{4\times 1} + \underbrace{\begin{pmatrix} w_{10}^{(1)} \\ w_{20}^{(1)} \\ w_{30}^{(1)} \end{pmatrix}}_{3\times 1}\right)\right] + \underbrace{\begin{pmatrix} w_{10}^{(2)} \\ w_{20}^{(2)} \\ w_{30}^{(2)} \end{pmatrix}}_{3\times 1}\right)\right] + \underbrace{\begin{pmatrix} w_{10}^{(3)} \\ w_{20}^{(3)} \\ w_{30}^{(3)} \end{pmatrix}}_{3\times 1}\right)\right] + \underbrace{w_{10}^{(4)}}_{1\times 1}\Bigg) \\
&= \mathbf{a}^{(4)},
\end{aligned}
\tag{3.42.1}
$$

where, for example $\sigma^{(1)}(z) = \sigma^{(2)}(z) = \sigma^{(3)}(z) = \sigma^{(4)}(z) = 1/(1+\exp(-z))$ are Sigmoid functions. Since, the AFs must be applied element-wise to vector arguments, the inner term of (3.42.1) becomes

$$\sigma^{(1)}\left(\underbrace{\begin{pmatrix} w_{11}^{(1)} & w_{12}^{(1)} & w_{13}^{(1)} & w_{14}^{(1)} \\ w_{21}^{(1)} & w_{22}^{(1)} & w_{23}^{(1)} & w_{24}^{(1)} \\ w_{31}^{(1)} & w_{32}^{(1)} & w_{33}^{(1)} & w_{34}^{(1)} \end{pmatrix}}_{3\times 4}\underbrace{\begin{pmatrix} a_1^{(0)} \\ a_2^{(0)} \\ a_3^{(0)} \\ a_4^{(0)} \end{pmatrix}}_{4\times 1} + \underbrace{\begin{pmatrix} w_{10}^{(1)} \\ w_{20}^{(1)} \\ w_{30}^{(1)} \end{pmatrix}}_{3\times 1}\right)$$

$$= \begin{pmatrix} \sigma^{(1)}\left(w_{11}^{(1)}a_1^{(0)} + w_{12}^{(1)}a_2^{(0)} + w_{13}^{(1)}a_3^{(0)} + w_{14}^{(1)}a_4^{(0)} + w_{10}^{(1)}\right) \\ \sigma^{(1)}\left(w_{21}^{(1)}a_1^{(0)} + w_{22}^{(1)}a_2^{(0)} + w_{23}^{(1)}a_3^{(0)} + w_{24}^{(1)}a_4^{(0)} + w_{20}^{(1)}\right) \\ \sigma^{(1)}\left(w_{31}^{(1)}a_1^{(0)} + w_{32}^{(1)}a_2^{(0)} + w_{33}^{(1)}a_3^{(0)} + w_{34}^{(1)}a_4^{(0)} + w_{30}^{(1)}\right) \end{pmatrix}$$

$$= \begin{pmatrix} \dfrac{1}{1+e^{-\left(w_{11}^{(1)}a_1^{(0)} + w_{12}^{(1)}a_2^{(0)} + w_{13}^{(1)}a_3^{(0)} + w_{14}^{(1)}a_4^{(0)} + w_{10}^{(1)}\right)}} \\ \dfrac{1}{1+e^{-\left(w_{21}^{(1)}a_1^{(0)} + w_{22}^{(1)}a_2^{(0)} + w_{23}^{(1)}a_3^{(0)} + w_{24}^{(1)}a_4^{(0)} + w_{20}^{(1)}\right)}} \\ \dfrac{1}{1+e^{-\left(w_{31}^{(1)}a_1^{(0)} + w_{32}^{(1)}a_2^{(0)} + w_{33}^{(1)}a_3^{(0)} + w_{34}^{(1)}a_4^{(0)} + w_{30}^{(1)}\right)}} \end{pmatrix}. \quad (3.42.2)$$

Trying to find the derivative of complete composition function $\mathcal{N}\left(\mathbf{a}^{(0)}\right)$ algebraically becomes increasingly tedious as we increase the complexity of the NN.

Viewing a NN as a computational graph is a useful abstraction that helps in understanding and implementing NNs. The computational graph provides a clear and structured representation of the NN's operations, making it easier to apply the chain rule and perform efficient calculations for finding the derivative of the composition functions by using BP and automatic differentiation, as we will see in the next sections.

Computational graph separates the big computation into small steps, and we can find the derivative of each step (each computation) on the graph. Then the chain rule gives the derivatives of the final output. This is an incredibly efficient improvement on the computation of the derivative. At first it seems unbelievable, that reorganizing the computations can make such an enormous difference. In the end, you have to compute derivatives for each step and multiply by the chain rule. But the method does work.

Definition (Directed Acyclic Computational Graph): A directed acyclic computational graph is a directed acyclic graph of nodes, where each node contains a variable. Edges might be associated with learnable parameters. A variable in a node is either fixed externally (for input nodes with no incoming edges), or it is a computed as a function of the variables in the tail ends of edges incoming into the node and the learnable parameters on the incoming edges.

Remarks:

- We have been using graphs all along to represent NNs. In the following, we will use graphs to represent expressions instead.
- A graph is a collection of nodes (vertices) and edges connecting these nodes. In a directed graph, each edge has a specific direction, indicating a relationship between the connected nodes.
- Acyclic graph means that there are no cycles or loops in the graph. In other words, you cannot start at a node and follow a sequence of edges to return to the same node.
- Each node in the graph contains a variable. This variable can represent various quantities or data, depending on the context of the computational graph.
- For input nodes with no incoming edges, variables are fixed externally. This means that the values of these variables are provided from an external source rather than being computed within the graph.