Equations in the matrix form for implementing simple artificial neural networks

Ajit R. Jadhav, Ph.D.
https://JadhavResearch.wordpress.com

Release Status: Alpha
Major Version: 1.0
Minor Version: Wednesday 05 February 2020 16:57:35 IST

Abstract

This document presents the basic equations in reference to which artificial neural networks are designed and implemented. The scope is restricted to the simpler feedforward networks, including those having hidden layers. Convolutional and recurrent networks are out of the scope.

Equations are often initially noted using an index-basd notation for the typical element. However, all the equations are eventually cast in the direct matrix form, using a consistent set of notation. Some of the minor aspects of notation were invented to make the presentation as simple and direct as possible.

The presentation here regards a layer as the basic unit. The term "layer" is understood in the same sense in which APIs of modern libraries like TensorFlow-Keras 2.x take it. The presentation here is detailed enough that neural networks with hidden layers could be implemented, starting from the scratch.

Preface

Raison d'être

I wrote this document mainly for myself, to straighten out the different notations and formulae used in different sources and contexts.

In particular, I wanted to have a document that better matches the design themes used in today's libraries (like TensorFlow-Keras 2.x) than the description in the text-books.

For instance, in many sources, the input layer is presented as consisting of both a fully connected layer and its corresponding activation layer. However, for flexibility, libraries like TF-Keras 2.x treat them as separate layers.

Also, some sources uniformly treat the input of any layer as $\vec{X}$ and output of any layer as activation, $\vec{a}$, but such usage overloads the term "activation". Confusions also creep in because different conventions exist: treating the bias by expanding the input vector with 1 and the weights matrix with w_0; the "to-from" vs "from-to" convention for the weights matrix, etc.

I wanted to have a consistent notation that dealt with all such issues with a uniform, matrix-based notation that came as close to the `numpy` `ndarray` interface as possible.

Level of coverage

The scope here is restricted to the simplest ANNs, including the simplest DL networks. Convolutional neural networks and recurrent neural networks are out of the scope.

Yet, this document wouldn't make for a good tutorial for a complete beginner; it is likely to confuse him more than explaining anything to him.

So, if you are completely new to ANNs, it is advisable to go through sources like Nielsen's online book [1] to learn the theory of ANNs. Mazur's fully worked out example of the back-propagation algorithm [2] should also prove to be very helpful, before returning back to this document.

If you already know ANNs, and don't want to see equations in the fully expanded forms—or, plain dislike the notation used here—then a good reference, roughly at the same level as this document, is the set of write-ups/notes by Mallya [3].

Feedback

Any feedback, especially that regarding errors, typos, inconsistencies in notation, suggestions for improvements, etc., will be thankfully received.

email: aj175tp@yahoo.co.in.

How to cite this document

Cite this document as:

Insert citation info. at the time of the Release version **TBD**

ARJ
Pune, India

Contents

1 Layers as building blocks of ANNs

1.1 Data

The data for training the network consists a table having μ number of rows.

Each row has one tuple for the independent variables, $\{\mathrm{X}\}$, and another tuple for the dependent variable $\{\mathrm{Y}\}$. The independent variables may be regarded as "features" of the dataset, and the dependent variables form the "targets" of the dataset (which act as the "ground truth" for training of the ANN).

In this document, the input vector $\{\mathrm{X}\}$ is assumed to have M number of elements, and the target vector $\{\mathrm{Y}\}$ is assumed to have N number of elements.

In case of the MNIST dataset, the tuple for the independent variables consists of a flattened vector of floating point values for the pixel intensities; its size is $\mathrm{M} = 28 \times 28 = 784$. The tuple for the dependent variables consists of a one-hot encoded vector of size $\mathrm{N} = 10$; it indicates the class of digits to which the input picture belongs.

1.2 Layers involved

In this document, the phrase "simplest ANN" is taken to mean an artificial neural network that has *no hidden layer*.

Accordingly, the simplest ANN may be regarded as composed from three layers which appear in the following sequence:

- L_1: A fully connected layer: It contains the weights and biases, i.e., the parameters of the model
- L_2: An activation layer: It determines how each neuron fires
- L_3: A loss layer: It determines the measure for the error between the parameters- and activation-based predictions, and the ground truth contained in the target data.

Many textbooks present the material as if L_1 and L_2 were a single layer.

2 Matrix-notation for vectors, matrices, and operations on them

Before turning to the ANNs, it is important to develop a consistent notation for denoting vectors, matrices and their transposes, and the operations involving them—algebraic operations as well as differentiations. Only those cases are considered which are directly relevant to the equations of ANNs.

2.1 Basic notation for vectors, matrices, and indices for the typical elements

Vectors are always column-matrices

We always represent a *vector* using a *column* matrix, and never using a row matrix. A column matrix is denoted using the braces (curly brackets); e.g. $\{\mathrm{V}\}$.

We always represent the transpose of a vector, i.e. a row-vector, using the floor symbols, as in $\lfloor \mathrm{V}^{\mathrm{T}} \rfloor = \{\mathrm{V}\}^{\mathrm{T}}$.

We always represent a matrix (whether a square matrix or a general-rectangular matrix) using square brackets, as in $[\mathrm{A}]$.

To avoid confusion, in this article, we never use braces, floor symbols, or square brackets for grouping of terms in an equation. We only use parentheses "( )" for this purpose, even if they get nested.

Sizes appear in subscripts, sometimes

Sometimes, the dimensions (i.e. the "size" or the number of rows and columns) are explicitly noted using square-brackets in the subscripts, as in

$$\{\mathrm{V}\}_{[\mathrm{R}\,\times\,1]}\,, \qquad \text{or} \qquad [\mathrm{W}]_{[\mathrm{R}\,\times\,\mathrm{C}]}\,,$$

where R is the number of rows and C is the number of columns.

Transposed matrices and vectors: basic form

The transpose of a rectangular matrix is denoted as:

$$[\mathrm{W}^{\mathrm{T}}]_{[\mathrm{C}\,\times\,\mathrm{R}]} \;=\; \left([\mathrm{W}]_{[\mathrm{R}\,\times\,\mathrm{C}]}\right)^{\mathrm{T}}$$

where R and C refer to the number of rows and columns of the original, untransposed matrix. Similarly, a row-matrix is denoted as

$$\lfloor \mathrm{v}^{\mathrm{T}} \rfloor_{[1\,\times\,\mathrm{C}]} \;=\; \left(\{\mathrm{v}\}_{[\mathrm{R}\,\times\,1]}\right)^{\mathrm{T}}\;; \qquad \mathrm{C} \;=\; \mathrm{R}.$$

Notation for vectors using the typical elements

Vectors are sometimes also denoted using a notation involving its typical symbol and its shape (whether a column- or a row-matrix):

$$\{\mathrm{X}\} \;\overset{\text{def}}{=}\; \begin{Bmatrix} \mathrm{x}_1 \\ \mathrm{x}_2 \\ \vdots \\ \mathrm{x}_\mathrm{r} \\ \vdots \\ \mathrm{x}_\mathrm{M} \end{Bmatrix} \;\overset{\text{denoted}}{=}\; \{\mathrm{x}_\mathrm{r}\}$$

and

$$\lfloor \mathrm{A} \rfloor \overset{\text{def}}{=} \lfloor \mathrm{a}_1 \quad \mathrm{a}_2 \quad \dots \quad \mathrm{a}_\mathrm{c} \quad \dots \quad \mathrm{a}_\mathrm{N} \rfloor \overset{\text{denoted}}{=} \lfloor \mathrm{a}_\mathrm{c} \rfloor$$

Notation for matrices using the typical elements

Similarly, matrices are sometimes denoted using a notation that involves two indices for its typical symbol, put inside the rectangular matrix symbol. Thus:

$$[\mathrm{W}]_{[\mathrm{M} \times \mathrm{N}]} \overset{\text{def}}{=} \begin{bmatrix} \mathrm{w}_{11} & \mathrm{w}_{12} & \dots & \mathrm{w}_{1\mathrm{c}} & \dots & \mathrm{w}_{1\mathrm{N}} \\ \mathrm{w}_{21} & \mathrm{w}_{22} & \dots & \mathrm{w}_{2\mathrm{c}} & \dots & \mathrm{w}_{2\mathrm{N}} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \mathrm{w}_{\mathrm{r}1} & \mathrm{w}_{\mathrm{r}2} & \dots & \mathrm{w}_{\mathrm{rc}} & \dots & \mathrm{w}_{\mathrm{rN}} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \mathrm{w}_{\mathrm{M}1} & \mathrm{w}_{\mathrm{M}2} & \dots & \mathrm{w}_{\mathrm{Mc}} & \dots & \mathrm{w}_{\mathrm{MN}} \end{bmatrix}_{[\mathrm{M} \times \mathrm{N}]}$$

$$\overset{\text{denoted}}{=} [\mathrm{w}_{\mathrm{rc}}]_{[\mathrm{M} \times \mathrm{N}]}$$

where M is the number of rows and N is the number of columns. The transpose of this matrix would be denoted as:

$$[\mathrm{W}^\mathrm{T}]_{[\mathrm{N} \times \mathrm{M}]} \overset{\text{def}}{=} \left([\mathrm{W}]_{[\mathrm{M} \times \mathrm{N}]} \right)^\mathrm{T}$$

$$\overset{\text{denoted}}{=} \begin{bmatrix} \mathrm{wt}_{11} & \mathrm{wt}_{12} & \dots & \mathrm{wt}_{1\mathrm{c}} & \dots & \mathrm{wt}_{1\mathrm{M}} \\ \mathrm{wt}_{21} & \mathrm{wt}_{22} & \dots & \mathrm{wt}_{2\mathrm{c}} & \dots & \mathrm{wt}_{2\mathrm{M}} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \mathrm{wt}_{\mathrm{r}1} & \mathrm{wt}_{\mathrm{r}2} & \dots & \mathrm{wt}_{\mathrm{rc}} & \dots & \mathrm{wt}_{\mathrm{rM}} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \mathrm{wt}_{\mathrm{N}1} & \mathrm{wt}_{\mathrm{N}2} & \dots & \mathrm{wt}_{\mathrm{Nc}} & \dots & \mathrm{wt}_{\mathrm{NM}} \end{bmatrix}_{[\mathrm{N} \times \mathrm{M}]}$$

$$\overset{\text{denoted}}{=} [\mathrm{wt}_{\mathrm{rc}}]_{[\mathrm{N} \times \mathrm{M}]}.$$

Note carefully that the typical element is here listed as wt and not as w. The number of rows now is N and the number of columns M. Since wt is the typical element and not w, the first subscript refers to the row of *transposed* matrix and the second subscript to the column of the *transposed* matrix.

Sometimes, we also note the transpose of $[\mathrm{W}]$ using the indices of the *original* *matrix*, as in:

$$\begin{aligned}
\left[W^T\right]_{[N \times M]} &\overset{\text{def}}{=} \left([W]_{[M \times N]} \right)^T \\
&\overset{\text{denoted}}{=} \begin{bmatrix} w_{11} & w_{21} & \cdots & w_{c1} & \cdots & w_{N1} \\ w_{12} & w_{22} & \cdots & w_{c2} & \cdots & w_{N2} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ w_{1r} & w_{2r} & \cdots & w_{cr} & \cdots & w_{Nr} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ w_{1M} & w_{2M} & \cdots & w_{cM} & \cdots & w_{NM} \end{bmatrix}_{[N \times M]} \\
&\overset{\text{denoted}}{=} \left[w_{cr}\right]_{[N \times M]}.
\end{aligned}$$

Note carefully that the typical element listed in the last equation was as w. However, the notation still indicates a *transpose*, because the *order of the indices is now flipped*. The first subscript now refers to the *column* of the *original* matrix, and the second subscript now refers to the *row* of the *original* matrix. Thus w_{cr}.

Summary: Typical elements-based notation

In short, in the typical element-based notation, we denote a matrix and its transpose as:

$$\begin{aligned}
[W] &\overset{\text{denoted}}{=} \left[w_{rc}\right] \; ; \qquad r \in [1, M], c \in [1, N] \\
\left[W^T\right] &\overset{\text{denoted}}{=} \left[wt_{rc}\right] \; ; \qquad r \in [1, N], c \in [1, M] \\
&\overset{\text{denoted}}{=} \left[w_{cr}\right] \; ; \qquad r \in [1, M], c \in [1, N]
\end{aligned}$$

Similarly, a vector and its transpose are denoted as:

$$\begin{aligned}
\{V\} &\overset{\text{denoted}}{=} \{v_r\} \; ; \qquad r \in [1, M] \\
\lfloor V^T \rfloor &\overset{\text{denoted}}{=} \lfloor vt_r \rfloor \; ; \qquad r \in [1, M] \\
&\overset{\text{denoted}}{=} \lfloor v_c \rfloor \; ; \qquad c \in [1, N] \; ; \qquad \text{but for a vector, } N = M
\end{aligned}$$

2.2 A rectangular matrix as one column of many rows, or as one row of many columns

A rectangular matrix can also be seen as a column matrix having row-matrices for its elements. It can also be seen as a row matrix having column-matrices for its elements. We adopt the following notation to simplify noting these representations:

$$\left[\mathrm{W}\right]_{[\mathrm{M}\times\mathrm{N}]} = \left\{ \left\lfloor \mathrm{w}_{\mathrm{r}*} \right\rfloor_{[1\times\mathrm{N}]} \right\}_{[\mathrm{M}\times 1]} = \left\{ \begin{matrix} \lfloor \mathrm{w}_{1*} \rfloor \\ \lfloor \mathrm{w}_{2*} \rfloor \\ \vdots \\ \lfloor \mathrm{w}_{\mathrm{r}*} \rfloor \\ \vdots \\ \lfloor \mathrm{w}_{\mathrm{M}*} \rfloor \end{matrix} \right\}$$

where $*$ denotes expanding over the entire range of that index.

Similarly,

$$\begin{aligned} \left[\mathrm{W}\right]_{[\mathrm{M}\times\mathrm{N}]} &= \left\lfloor \left\{\mathrm{w}_{*\mathrm{c}}\right\}_{[\mathrm{M}\times 1]} \right\rfloor_{[1\times\mathrm{N}]} \\ &= \left\lfloor \{\mathrm{w}_{*1}\} \quad \{\mathrm{w}_{*2}\} \quad \ldots \quad \{\mathrm{w}_{*\mathrm{c}}\} \quad \ldots \quad \{\mathrm{w}_{*\mathrm{N}}\} \right\rfloor . \end{aligned}$$

This notation denotes a matrix using its typical elements.

3 Matrix-notation for algebraic operations between vectors and matrices

3.1 Initialization of vectors and matrices with numbers

When all elements of a vector or a matrix are initialized to some number, say 0.0, we use the following notation:

$$\{\mathrm{A}\} \leftarrow \{0.0\} .$$

where it is understood that $\{0.0\}$ is a vector of all zeros, and of the same size as that of $\{\mathrm{A}\}$.

Similarly, for a matrix:

$$[\mathrm{W}] \leftarrow [0.0] .$$

Initialization with random values is noted as:

$$\{\mathrm{A}\} \leftarrow \text{random values}$$

where it is understood that each element is initialized to a different random value drawn from a certain distribution (normal or uniform) over a certain numerical range, e.g., $[-1.0, 1.0]$.

3.2 Summing all the elements of a vector

Let $\{\mathrm{A}\}$ be a vector. The sum of all its elements can be noted, in the index notation, as:

$$\mathrm{S} \;=\; \sum_{\mathrm{r}=1}^{\mathrm{M}} \mathrm{a_r}$$

We note this operation, in our notation, as:

$$\mathrm{S} \overset{\text{denoted}}{=} \Sigma\,(\;\{\mathrm{A}\}\;) \overset{\text{denoted}}{=} \Sigma_{\mathrm{r}}\,(\;\{\mathrm{a_r}\}\;)$$

Similarly, for summing all elements of a row-vector $\lfloor \mathrm{B} \rfloor$

$$\mathrm{S} \;=\; \sum_{\mathrm{c}=1}^{\mathrm{N}} \mathrm{b_c},$$

we use the notations:

$$\mathrm{S} \overset{\text{denoted}}{=} \Sigma\,(\;\lfloor \mathrm{B} \rfloor\;) \overset{\text{denoted}}{=} \Sigma_{\mathrm{c}}\,(\;\lfloor \mathrm{b_c} \rfloor\;)$$

3.3 Multiplication of a vector by a scalar

We denote the multiplication of each element of a vector with a scalar as:

$$\mathrm{s}\,\{\mathrm{A}\} \overset{\text{denoted}}{=} \{\mathrm{s\,a_r}\}$$

Similarly,

$$\mathrm{s}\,\lfloor \mathrm{B} \rfloor \overset{\text{denoted}}{=} \lfloor \mathrm{s\,a_c} \rfloor$$

Similar operations can be defined for element-wise divisions, as also for element-wise additions and subtractions.

3.4 Functions acting element-wise on one vector to produce another vector

Let $\{\mathrm{X}\}$ and $\{\mathrm{Y}\}$ be two vectors of equal lengths.

If each r-th element of $\{\mathrm{Y}\}$, $\mathrm{y_r}$, can be obtained by applying the same scalar function $\mathcal{F}$ on the corresponding elements of $\{\mathrm{X}\}$, $\mathrm{x_r}$, i.e., if

$$\mathrm{y_r} \;=\; \mathcal{F}_{\mathrm{r}}\,(\;\mathrm{x_r}\;) \qquad \forall\; 1 \le \mathrm{r} \le \mathrm{M},$$

then we denote this fact as:

$$\begin{aligned} \{\mathrm{y_r}\} &= \mathcal{F}_{\mathrm{r}}\,(\;\{\mathrm{x_r}\}\;) \\ \{\mathrm{Y}\} &= \mathcal{F}_{\mathrm{r}}\,(\;\{\mathrm{X}\}\;), \end{aligned}$$

where it is understood that there can be a set of different row-wise acting functions $\mathcal{F}_r$.

Similarly, we can define a function relationship between two transposed vectors:

$$\begin{aligned} \lfloor y_c \rfloor &\overset{\text{def}}{=} \mathcal{F}_c\left(\lfloor x_c \rfloor \right) \\ \lfloor Y \rfloor &\overset{\text{def}}{=} \mathcal{F}_c\left(\lfloor X \rfloor \right) \end{aligned}$$

Further, we can also define functions that send some column-matrix to a row-matrix, and vice-versa, as:

$$\begin{aligned} \lfloor y_c \rfloor &= \mathcal{T}_{r\to c}\left(\{x_r\} \right) \\ \{y_r\} &= \mathcal{T}_{c\to r}\left(\lfloor x_c \rfloor \right) \end{aligned}$$

and

$$\begin{aligned} \lfloor Y \rfloor &= \mathcal{T}_{r\to c}\left(\{X\} \right) \\ \{Y\} &= \mathcal{T}_{c\to r}\left(\lfloor X \rfloor \right) \end{aligned}$$

Seen in this light, the transpose operation is a special function that converts a column-matrix to a row-matrix while keeping the same elements, and vice versa:

$$\begin{aligned} \lfloor X^T \rfloor &\overset{\text{def}}{=} \mathcal{T}_{r\to c}\left(\{X\} \right) \overset{\text{def}}{=} \left(\{X\} \right)^T \\ \{X\} &\overset{\text{def}}{=} \mathcal{T}_{c\to r}\left(\{X^T\} \right) \overset{\text{def}}{=} \left(\{X^T\} \right)^T \end{aligned}$$

3.5 Algebraic binary operators between two vectors

We use the following symbols for operations on vectors (i.e. column matrices):

- Element-wise addition of $\{X\}$ and $\{Y\}$:

  $$\begin{aligned} \{A\} &= \{X\} \oplus \{Y\} \\ \{a_r\} &= \{x_r\} \oplus \{y_r\} \\ &\overset{\text{def}}{=} \{x_r + y_r\} \end{aligned}$$

  Similarly for the following operators.

- Element-wise subtraction of $\{Y\}$ from $\{X\}$ is denoted as:

  $$\{S\} = \{X\} \ominus \{Y\} \qquad \Leftrightarrow \qquad \{s_r\} \overset{\text{def}}{=} \{x_r - y_r\}$$

- The Hadamard product: Element-wise multiplication of $\{X\}$ and $\{Y\}$ is denoted as:

  $$\{H\} = \{X\} \circledast \{Y\} \qquad \Leftrightarrow \qquad \{h_r\} \overset{\text{def}}{=} \{a_r b_r\}$$

- Element-wise division of $\{\mathrm{A}\}$ by $\{\mathrm{B}\}$ is denoted as:

$$\{\mathrm{D}\} = \{\mathrm{A}\} \oslash \{\mathrm{B}\} \qquad \Leftrightarrow \qquad \{\mathrm{d_r}\} \overset{\text{def}}{=} \left\{\frac{\mathrm{a_r}}{\mathrm{b_r}}\right\}$$

The last two operations are not found defined in the usual undergraduate courses on vector algebra. However, the `ndarrays` from the Python library `numpy` come with intuitive operator overloadings for all these four operations.

3.6 The outer- and inner-products in the vector and matrix notations

It is to be realized that when writing equations using an explicit matrix form, there is no need to have separate symbols for the outer- or inner-products. To appreciate this point, consider the following vector equations:

$$\vec{\mathrm{A}} \cdot \vec{\mathrm{B}} = \mathrm{s} \qquad \text{(Inner, i.e. dot, product)}$$
$$\vec{\mathrm{A}} \otimes \vec{\mathrm{B}} = \overline{\overline{\sigma}} \qquad \text{(Outer product)}$$

where s is a single number (a scalar), and the double-overbars denotes a tensor.

Notice that the symbols for the operators, viz. $\cdot$ and $\otimes$, explicitly appeared in the above equations. If the same equations are expressed using the matrix notation, appropriate transposes have to be taken, but there is no need to insert the explicit symbols for the inner or outer products. The rules of the transpose-taking and order-based matrix-multiplications together take care of the appropriate semantics. Thus:

$$\lfloor \mathrm{A^T} \rfloor_{[1\,\times\,\mathrm{M}]} \{\mathrm{B}\}_{[\mathrm{M}\,\times\,1]} = \mathrm{s} \qquad \text{(Inner, i.e. dot, product)}$$
$$\{\mathrm{A}\}_{[\mathrm{M}\,\times\,1]} \lfloor \mathrm{B^T} \rfloor_{[1\,\times\,\mathrm{N}]} = [\sigma]_{[\mathrm{M}\,\times\,\mathrm{N}]} \qquad \text{(Outer product)}$$

Further, using our typical element-based notation, the same equations will now be put as:

$$\lfloor \mathrm{a_c} \rfloor_{[1\,\times\,\mathrm{M}]} \{\mathrm{b_r}\}_{[\mathrm{M}\,\times\,1]} = \sum_{\mathrm{i}=1}^{\mathrm{M}} \mathrm{a_i b_i} = \mathrm{c} \qquad \text{(Inner, i.e. dot, product)}$$
$$\{\mathrm{a_r}\}_{[\mathrm{M}\,\times\,1]} \lfloor \mathrm{b_c} \rfloor_{[1\,\times\,\mathrm{N}]} = [\mathrm{t_{rc}}]_{[\mathrm{M}\,\times\,\mathrm{N}]} \;; \mathrm{t_{rc}} = \mathrm{a_r b_c} \qquad \text{(Outer product)}$$

In this document, we use mostly the matrix notation. Hence, the $\otimes$ and the $\cdot$ operators will not appear in the equations as stated. In contrast, we will often use the four element-wise operators noted earlier.

To avoid confusion, make sure to note again the difference between the $\otimes$ operator (the outer product) which will *not* appear in our matrix-based equations, and the $\circledast$ operator (the Hadamard product) which will.

3.7 Algebraic binary operators between the same rows of a vector and a matrix

If the number of rows of a vector and a matrix is the same, then, all the aforementioned element-wise operations can be conducted, in a row-by-row manner, between the two.

For instance, the element-wise multiplication of the elements of a column vector and the rows of a matrix can be denoted as:

$$\left[\mathrm{P}\right]_{[\mathrm{M}\times\mathrm{N}]} = \left\{\mathrm{V}\right\}_{[\mathrm{M}\times 1]} \circledast \left[\mathrm{D}\right]_{[\mathrm{M}\times\mathrm{N}]},$$

which means:

$$\begin{aligned}
\left[\mathrm{p_{rc}}\right]_{[\mathrm{M}\times\mathrm{N}]} &= \left\{\mathrm{v_r}\right\} \circledast \left[\mathrm{d_{rc}}\right] \\
&= \left\{\mathrm{v_r}\right\} \circledast \left\{\left\lfloor \mathrm{d_{r*}} \right\rfloor\right\} \\
&= \left\{\mathrm{v_r}\right\}_{[\mathrm{M}\times 1]} \circledast \left\{\left\lfloor \mathrm{d_{r1}} \quad \mathrm{d_{r2}} \quad \ldots \quad \mathrm{d_{rN}} \right\rfloor_{[1\times\mathrm{N}]}\right\}_{[\mathrm{M}\times 1]} \\
&= \left\{\left\lfloor \mathrm{v_r\,d_{r1}} \quad \mathrm{v_r\,d_{r2}} \quad \ldots \quad \mathrm{v_r\,d_{rN}} \right\rfloor_{[1\times\mathrm{N}]}\right\}_{[\mathrm{M}\times 1]} \\
&= \begin{bmatrix} \mathrm{v_1\,d_{11}} & \mathrm{v_1\,d_{12}} & \ldots & \mathrm{v_1\,d_{1N}} \\ \mathrm{v_2\,d_{21}} & \mathrm{v_2\,d_{22}} & \ldots & \mathrm{v_2\,d_{2N}} \\ \vdots & \vdots & \ddots & \vdots \\ \mathrm{v_M\,d_{M1}} & \mathrm{v_r\,d_{M2}} & \ldots & \mathrm{v_M\,d_{MN}} \end{bmatrix}_{[\mathrm{M}\times\mathrm{N}]} .
\end{aligned}$$

Of course, a similar operation can be defined also for a row-vector and a matrix, provided that the number of columns of the row-vector is equal to the number of columns of the matrix:

$$\left[\mathrm{P}\right]_{[\mathrm{M}\times\mathrm{N}]} = \left\lfloor\mathrm{V}\right\rfloor_{[1\times\mathrm{N}]} \circledast \left[\mathrm{D}\right]_{[\mathrm{M}\times\mathrm{N}]},$$

because the number of columns N is the same for both the operands. In the typical element-notation:

$$\begin{aligned}
\left[\mathrm{p_{rc}}\right]_{[\mathrm{M}\times\mathrm{N}]} &= \left\lfloor\mathrm{v_c}\right\rfloor_{[1\times\mathrm{N}]} \circledast \left[\mathrm{d_{rc}}\right]_{[\mathrm{M}\times\mathrm{N}]} \\
&= \left\lfloor\mathrm{v_c}\right\rfloor_{[1\times\mathrm{N}]} \circledast \left\lfloor\left\{\mathrm{d_{*c}}\right\}_{[\mathrm{M}\times 1]}\right\rfloor_{[1\times\mathrm{N}]} \\
&= \begin{bmatrix} \mathrm{v_1\,d_{11}} & \mathrm{v_2\,d_{12}} & \ldots & \mathrm{v_N\,d_{1N}} \\ \mathrm{v_1\,d_{21}} & \mathrm{v_2\,d_{22}} & \ldots & \mathrm{v_N\,d_{2N}} \\ \vdots & \vdots & \ddots & \vdots \\ \mathrm{v_1\,d_{M1}} & \mathrm{v_2\,d_{M2}} & \ldots & \mathrm{v_N\,d_{MN}} \end{bmatrix}
\end{aligned}$$

Other binary operators can be defined on similar lines.

3.8 Converting a vector into its diagonal matrix form

Our notation

Let's take a concrete example. Consider

$$\left\{ \mathrm{A} \right\}_{[3 \times 1]} \overset{\text{def}}{=} \begin{Bmatrix} \mathrm{a}_1 \\ \mathrm{a}_2 \\ \mathrm{a}_3 \end{Bmatrix}_{[3 \times 1]}$$

Its corresponding diagonal matrix is given by:

$$\begin{bmatrix} \mathrm{a}_1 & 0 & 0 \\ 0 & \mathrm{a}_2 & 0 \\ 0 & 0 & \mathrm{a}_3 \end{bmatrix}_{[3 \times 3]} .$$

In our notation, we denote the above matrix as:

$$\left[\mathbf{diag} \left\{ \mathrm{A} \right\} \right]_{[3 \times 3]}$$

Identity matrix

The identity matrix is defined as a matrix with all its diagonal elements 1, and all other elements 0:

$$[\mathcal{I}] \overset{\text{def}}{=} \begin{bmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 \end{bmatrix}_{[\mathrm{N} \times \mathrm{N}]}$$

The identity matrix may be seen as embedding the operation of converting the identity vector to its diagonal matrix form.

Diagonal form for an arbitrary vector

Let $\left\{ \mathrm{A} \right\}$ be a $[\mathrm{N} \times 1]$-sized column vector. Then, perform:

$$[\mathrm{O}]_{[\mathrm{N} \times \mathrm{N}]} \overset{\text{def}}{=} \left\{ \mathrm{A} \right\}_{[\mathrm{N} \times 1]} \otimes \lfloor 1 \rfloor_{[1 \times \mathrm{N}]} \tag{1}$$

where, remember, $\otimes$ represents the outer product. Realize, the result of the outer product here looks like:

$$[\mathrm{O}]_{[\mathrm{N} \times \mathrm{N}]} = \begin{bmatrix} \mathrm{a}_1 & \mathrm{a}_1 & \dots & \mathrm{a}_1 \\ \mathrm{a}_2 & \mathrm{a}_2 & \dots & \mathrm{a}_2 \\ \vdots & \vdots & \ddots & \vdots \\ \mathrm{a}_\mathrm{N} & \mathrm{a}_\mathrm{N} & \dots & \mathrm{a}_\mathrm{N} \end{bmatrix}_{[\mathrm{N} \times \mathrm{N}]}$$

Now, the diagonal form for $\left\{\mathrm{A}\right\}$ can be easily obtained as:

$$\begin{aligned}\left[\mathbf{diag}\left\{\mathrm{A}\right\}\right]_{[\mathrm{N}\,\times\,\mathrm{N}]} &= \left[\mathrm{O}\right]_{[\mathrm{N}\,\times\,\mathrm{N}]} \circledast \left[\mathcal{I}\right]_{[\mathrm{N}\,\times\,\mathrm{N}]} \\ &= \left(\ \left\{\mathrm{A}\right\}_{[\mathrm{N}\,\times\,1]} \otimes \left\lfloor 1 \right\rfloor_{[1\,\times\,\mathrm{N}]}\ \right) \circledast \left[\mathcal{I}\right]_{[\mathrm{N}\,\times\,\mathrm{N}]}\end{aligned}$$

where $\circledast$ implies multiplications of the corresponding elements.

The diagonal matrix-form of the identity vector $\left\lfloor 1 \right\rfloor$ is the identity matrix. It *can* be seen as row-wise or column-wise additions of unit vectors to the zero-matrix. Similarly, the vector $\left\{\mathrm{A}\right\}$ could also be put into the diagonal matrix form using row-wise matrix additions its component-vectors. However, we will not pursue these matters because our focus here is on ANNs.

4 Matrix-notation for element-wise differentiations of vectors and matrices

The importance of the indices of the typical element

The most "unintuitive" part of our notation appears in denoting differentiations. The best strategy here is to

- first focus on the typical elements (like the value at r-th row and c-th column),
- then express the desired differentiations among these typical elements,
- then put the typical differential expression inside the symbol that indicates the shape of the resulting matrix (a column matrix, a row matrix, or a rectangular matrix).

The subsection sub-sections will make clear what exactly we mean.

The notation for functions relating vectors and matrices

In general, a given matrix $\left[\mathrm{B}\right]$ could be the result of applying a *different* function $\mathrm{f_{ij}}$ to each element of another matrix $\left[\mathrm{A}\right]$. Thus,

$$\left[\mathrm{b_{ij}}\right] = \left[\mathrm{f_{ij}}\left(\mathrm{a_{ij}}\right)\right],$$

which expands to

$$\begin{bmatrix} b_{11} & b_{12} & \dots & b_{1N} \\ b_{21} & b_{22} & \dots & b_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ b_{M1} & b_{M2} & \dots & b_{MN} \end{bmatrix}$$
$$= \begin{bmatrix} f_{11}(a_{11}) & f_{12}(a_{12}) & \dots & f_{1N}(a_{1N}) \\ f_{21}(a_{21}) & f_{22}(a_{22}) & \dots & f_{2N}(a_{2N}) \\ \vdots & \vdots & \ddots & \vdots \\ f_{M1}(a_{M1}) & f_{M2}(a_{M2}) & \dots & f_{MN}(a_{MN}) \end{bmatrix}.$$

In this document, we denote even the above kind of a very general functional relationship using only the simple notation:

$$[B]_{[M \times N]} = \mathcal{F}\Big([A]_{[M \times N]} \Big).$$

Even if a single symbol $\mathcal{F}$ appears on the outside of the parentheses, it is understood that there in fact are $M \times N$ number of different fuctions acting here.

So, when it comes to equations involving functions of vectors and matrices, particular attention will have to be paid to the context in which they arise.

4.1 Differentiation of a single variable with respect to the rows of a vector

Suppose that

$$\mathcal{E} = \mathcal{L}\Big(\{A\}_{[N \times 1]}, \{Y\}_{[N \times 1]} \Big)$$

Now, if the single variable $\mathcal{E}$ is differentiated with respect to each element of $\{A\}$, then we use the following notation to capture the resulting column of the respective differential quantities:

$$\begin{Bmatrix} \partial\mathcal{E}/\partial a_1 \\ \partial\mathcal{E}/\partial a_2 \\ \vdots \\ \partial\mathcal{E}/\partial a_r \\ \vdots \\ \partial\mathcal{E}/\partial a_N \end{Bmatrix}_{[N \times 1]} \overset{\text{denoted}}{=} \left\{ \frac{\partial\mathcal{E}}{\partial a_r} \right\}_{[N \times 1]}$$

Notice that there is no subscript on $\mathcal{E}$ because it is a single variable, not a vector.

4.2 Row-wise differentiation of a vector with respect to another vector

Suppose that:

$$\{B\}_{[N \times 1]} = \mathcal{F}\Big(\{A\}_{[N \times 1]}, \dots \Big).$$

where $\dots$ indicates some more, unspecified, independent variables. In terms of typical elements,

$$\{\mathrm{b_r}\} = \mathcal{F}\left(\{\mathrm{a_r}\} \, , \dots \right) .$$

Now, if we partially differentiate each $\mathrm{b_r}$ element with respect to the corresponding $\mathrm{a_r}$ element, then we denote the resulting set in our matrix notation as:

$$\left\{ \begin{matrix} \partial \mathrm{b_1}/\partial \mathrm{a_1} \\ \partial \mathrm{b_2}/\partial \mathrm{a_2} \\ \vdots \\ \partial \mathrm{b_r}/\partial \mathrm{a_r} \\ \vdots \\ \partial \mathrm{b_N}/\partial \mathrm{a_N} \end{matrix} \right\} \overset{\text{denoted}}{=} \left\{ \frac{\partial \mathrm{b_r}}{\partial \mathrm{a_r}} \right\}_{[\mathrm{N} \times 1]}$$

4.3 Column-wise differentiation of a column-matrix with respect to another column-matrix

Similar results can be had also for transposed vectors. For instance, if

$$\lfloor \mathrm{B} \rfloor_{[1 \times \mathrm{N}]} = \mathcal{F}\left(\lfloor \mathrm{A} \rfloor_{[1 \times \mathrm{N}]} , \dots \right),$$

i.e., in terms of typical elements, if,

$$\lfloor \mathrm{b_r} \rfloor = \mathcal{F}\left(\lfloor \mathrm{a_r} \rfloor \, , \dots \right),$$

then,

$$\left\lfloor \frac{\partial \mathrm{b_1}}{\partial \mathrm{a_1}} \quad \frac{\partial \mathrm{b_2}}{\partial \mathrm{a_2}} \quad \dots \quad \frac{\partial \mathrm{b_r}}{\partial \mathrm{a_r}} \quad \dots \quad \frac{\partial \mathrm{b_N}}{\partial \mathrm{a_N}} \right\rfloor_{[1 \times \mathrm{N}]} \overset{\text{denoted}}{=} \left\lfloor \frac{\partial \mathrm{b_r}}{\partial \mathrm{a_r}} \right\rfloor_{[1 \times \mathrm{N}]}$$

4.4 Differentiation of each element of a vector with respect to all the elements of another vector

Suppose that each element $\mathrm{a_r}$ of a vector $\{\mathrm{A}\}$ is a function of *all* the elements of another vector $\{\mathrm{Z}\}$. Such a relationship is best expressed by considering the transpose of the second vector. Thus, in terms of typical elements,

$$\mathrm{a_r} = \mathcal{F}\left(\lfloor \mathrm{z_1} \quad \mathrm{z_2} \quad \dots \quad \mathrm{z_c} \quad \dots \quad \mathrm{z_N} \rfloor \, , \dots \right) .$$

Now, suppose we differentiate each $\mathrm{a_r}$ with respect not just the corresponding $\mathrm{z_r}$, but *all* the rest of them too, taken separately. Then, they naturally form an

$\mathrm{[N \times N]}$ matrix:

$$\begin{bmatrix} \partial a_1/\partial z_1 & \partial a_1/\partial z_2 & \dots & \partial a_1/\partial z_c & \dots & \partial a_1/\partial z_N \\ \partial a_2/\partial z_1 & \partial a_2/\partial z_2 & \dots & \partial a_2/\partial z_c & \dots & \partial a_2/\partial z_N \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \partial a_r/\partial z_1 & \partial a_r/\partial z_2 & \dots & \partial a_r/\partial z_c & \dots & \partial a_r/\partial z_N \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \partial a_N/\partial z_1 & \partial a_N/\partial z_2 & \dots & \partial a_N/\partial z_c & \dots & \partial a_N/\partial z_N \end{bmatrix}_{[N \times N]} \tag{2}$$

$$\overset{\text{denoted}}{=} \left[\frac{\partial a_r}{\partial z_c}\right] \overset{\text{denoted}}{=} \mathbf{J}_{rc},$$

where $\mathbf{J}_{rc}$ stands for the Jacobian matrix. The pnemonic for the $[\bullet_{rc}]$ order for the indices is this:

Hold the the row $_r$ fixed, and unroll all the $_c$ columns!. Then repeat for the next rows.

Note that, some times, in literature, the Jacobian matrix is defined as the transpose of the matrix we gave. Be careful about what index occupies what place.

A further generalization of the Jacobian matrix for the case when $\{B\}$ has N elements, $\{A\}$ has M elements, and $M \neq N$, is possible. In such a case, the Jacobian matrix has $[M \times N]$ size. However, we will not need it in this document.

4.5 Differentiation of each element of a vector with respect to the corresponding row of a matrix

Suppose that an $[N \times 1]$-sized vector $\{Z\}$ is a function of a matrix $[W^T]$ of size $[N \times M]$. (Yes, there does exist another matrix $[W]$of size $[M \times N]$. However, for the fully connected layer, the function of interest is better captured with respect to its transpose.) Thus, suppose

$$\{Z\}_{[N \times 1]} = \mathcal{F}\Big([W^T]_{[N \times M]} \ , \dots \Big) .$$

Now, express $[W^T]$ as a column matrix of rows:

$$\{Z\}_{[N \times 1]} = \mathcal{F}\left(\Big\{ \lfloor W_r^T \rfloor_{[1 \times M]} \Big\}_{[N \times 1]} \ , \dots \right)$$

where $\lfloor W_r^T \rfloor$ denotes the entire r-th row of the rectangular matrix $[W^T]$.

In terms of typical elements,

$$z_r = \mathcal{F}\big(\lfloor wt_{r*} \rfloor \ , \dots \big)$$

where $\lfloor wt_{r*} \rfloor$ denotes the entire r-th row of the rectangular matrix $[W^T]$.

Differentiation with respect to a single typical row

Now, consider the differentiation of $\mathrm{z_r}$ with respect to each element of the r-th row of $[\mathrm{W^T}]$. Thus, the quantities of interest are:

$$\left\lfloor \frac{\partial \mathrm{z_r}}{\partial \mathrm{wt_{r1}}} \quad \frac{\partial \mathrm{a_r}}{\partial \mathrm{wt_{r2}}} \quad \ldots \quad \frac{\partial \mathrm{z_r}}{\partial \mathrm{wt_{rc}}} \quad \ldots \quad \frac{\partial \mathrm{z_r}}{\partial \mathrm{wt_{rM}}} \right\rfloor .$$

Note how the differential quantities naturally form a row-matrix. Using the $*$-based notation introduced earlier, we can condense down the above long expression to:

$$\left\lfloor \frac{\partial \mathrm{z_r}}{\partial \mathrm{wt_{r1}}} \quad \frac{\partial \mathrm{z_r}}{\partial \mathrm{wt_{r2}}} \quad \ldots \quad \frac{\partial \mathrm{z_r}}{\partial \mathrm{wt_{rc}}} \quad \ldots \quad \frac{\partial \mathrm{z_r}}{\partial \mathrm{wt_{rM}}} \right\rfloor \overset{\text{denoted}}{=} \left\lfloor \frac{\partial \mathrm{z_r}}{\partial \mathrm{wt_{r*}}} \right\rfloor_{[1 \times \mathrm{M}]},$$

where it is understood that the expression refers to a single row-vector of a rectangular matrix.

Differentiation with respect to a matrix

Repeating the above procedure for every row of $[\mathrm{W^T}]$, we can say that we have arrived at an expression for the partial differentiation of a vector with respect to a matrix:

$$\begin{bmatrix} \partial \mathrm{z_1}/\partial \mathrm{wt_{11}} & \partial \mathrm{z_1}/\partial \mathrm{wt_{12}} & \ldots & \partial \mathrm{z_1}/\partial \mathrm{wt_{1c}} & \ldots & \partial \mathrm{z_1}/\partial \mathrm{wt_{1M}} \\ \partial \mathrm{z_2}/\partial \mathrm{a_{21}} & \partial \mathrm{z_2}/\partial \mathrm{wt_{22}} & \ldots & \partial \mathrm{z_2}/\partial \mathrm{a_{2c}} & \ldots & \partial \mathrm{z_2}/\partial \mathrm{wt_{2M}} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \partial \mathrm{z_r}/\partial \mathrm{wt_{r1}} & \partial \mathrm{z_r}/\partial \mathrm{wt_{r2}} & \ldots & \partial \mathrm{z_r}/\partial \mathrm{wt_{rc}} & \ldots & \partial \mathrm{z_r}/\partial \mathrm{wt_{rM}} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \partial \mathrm{z_N}/\partial \mathrm{wt_{N1}} & \partial \mathrm{z_N}/\partial \mathrm{wt_{N2}} & \ldots & \partial \mathrm{z_N}/\partial \mathrm{wt_{Nc}} & \ldots & \partial \mathrm{z_N}/\partial \mathrm{wt_{NM}} \end{bmatrix} \tag{3}$$

$$\overset{\text{denoted}}{=} \left[\frac{\partial \mathrm{z_r}}{\partial \mathrm{wt_{rc}}} \right]_{[\mathrm{N} \times \mathrm{M}]}$$

The logic behind adopting such unusual notation is simply this much: The actual operations happen, at the most basic level, only with respect to sets of linear equations. If the operation involved is differentiation, that form of matrix notation should be adopted which reduces the cognitive load stating the equations.

5 Training a network using the mini-batch method: main steps

The description here refers to the mini-batch gradient descent. We will take the example of the MNIST dataset.

Suppose that the number of independent variables is M. In the MNIST example, $\mathrm{M} = 28 \times 28 = 784$.

Suppose that the number of dependent variables is N. In the MNIST example, $\mathrm{N} = 10$.

Suppose that there are μ number of rows in the training dataset. In the MNIST example, $\mu = 60,000$.

An epoch refers to processing all the rows of the training dataset (here, $60,000$ of them) once. Training is conducted for many such epochs, may be up to a few hundred, until the errors drop down sufficiently. We will not look into the criteria to stop training.

Now, the processing flow is the following.

5.1 Initialize the parameters of the model

Convention for denoting the weights matrix

In this document, we adopt the convention that the weights matrix for the fully connected layer is arranged in the "from–to" order. Thus, if you take the typical element w_{ij}, then it refers to the weight of the edge *from* i-th *input* neuron *to* the j-th *output* neuron.

With this convention, all our matrix equations come to refer to only the transpose of the weight matrix, i.e., to $[W^T]$, which can be seen as being listed in the "to–from" order. Thus:

$$\{Z\} = [W^T]\{X\} \oplus \{B\}$$

In index notation:

$$z_i = \sum_{j=1}^{M} (\, wt_{ij}\, x_j\,) + b_j, \tag{4}$$

where the index i runs over all rows of $\{Z\}$ and the index j runs over all columns of $[W^T]$ (or all rows of $[W]$) and also over all rows of $\{X\}$.

In matrix notation using the typical elements:

$$\begin{aligned}\{z_i\} &= [wt_{ij}]\{x_j\} \oplus \{b_j\} \\ &= [w_{ji}]\{x_j\} \oplus \{b_j\}\end{aligned} \tag{5}$$

We follow the "from–to" convention for two reasons: (i) the ordering is in the natural forward sequence, i.e., from input to output, and not from output to input, and (ii) the equations in the matrix form remain in direct correspondence with those in the vector-tensor form, as in:

$$\vec{Z} = \overline{\overline{W}} \cdot \vec{X} + \vec{B}$$

Initialization

First of all, before any training begins, initialize the parameters (the weights and biases) with some random values.

$$\begin{aligned}\left[\mathrm{W}^{\mathrm{T}}\right]_{[\mathrm{N}\times\mathrm{M}]} &\leftarrow \left[\text{random values}\right] \qquad \text{range: } -1.0 \text{ to } 1.0\\ \left[\mathrm{B}\right]_{[\mathrm{N}\times 1]} &\leftarrow \left\{\text{random values}\right\} \qquad \text{range: } -1.0 \text{ to } 1.0\end{aligned}$$

During training, these values will get *updated* at the end of each mini-batch to reflect learning, but they will never be *initialized* at any step during the training.

Training is conducted continuously for a number of epochs (say, between a few tens to a few hundreds).

5.2 Processing during each epoch

Processing during each epoch consists of the following steps.

Randomly select rows for all the mini-batches of an epoch

At the beginning of each epoch, row-wise shuffle the dataset.

Then divide the datasets into a number of mini-batches. For instance, in case of the MNIST dataset, mini-batches of $2,000$ rows might be created. Denote the number of rows in a mini-batch as ν.

Initialize the accumulated gradients

Let $\left[\mathcal{D}_{\mathcal{E},\mathrm{W}^{\mathrm{T}}}\right]$ and $\left\{\mathcal{D}_{\mathcal{E},\mathrm{B}}\right\}$ denote the gradients accumulated at the end of processing of a mini-batch.

At the beginning of each mini-batch, initialize these matrices to zero:

$$\begin{aligned}\left[\mathcal{D}_{\mathcal{E},\mathrm{W}^{\mathrm{T}}}\right] &\leftarrow [0.0]\\ \left\{\mathcal{D}_{\mathcal{E},\mathrm{B}}\right\} &\leftarrow \{0.0\}\end{aligned}$$

Process all the rows of a mini-batch

For each row of the current mini-batch, make one forward pass, and one backward pass. Gradients of errors with respect to parameters will be obtained at the end of the backward pass. Accumulate these gradients for all rows of the current mini-batch:

$$\begin{aligned}\left[\mathcal{D}_{\mathcal{E},\mathrm{W}^{\mathrm{T}}}\right] &\leftarrow \left[\mathcal{D}_{\mathcal{E},\mathrm{W}^{\mathrm{T}}}\right] \oplus \left[\mathrm{D}_{\mathcal{E},\mathrm{W}^{\mathrm{T}}}\right]_{\mathrm{r}}\\ \left\{\mathcal{D}_{\mathcal{E},\mathrm{B}}\right\} &\leftarrow \left\{\mathcal{D}_{\mathcal{E},\mathrm{B}}\right\} \oplus \left\{\mathrm{D}_{\mathcal{E},\mathrm{B}}\right\}_{\mathrm{r}}\end{aligned}$$

where the quantities in black, i.e. $\left[\mathrm{D}_{\mathcal{E},\mathrm{W}^{\mathrm{T}}}\right]_{\mathrm{r}}$ and $\left\{\mathrm{D}_{\mathcal{E},\mathrm{B}}\right\}$, represent the gradients computed at the end of the r-th row, and the quantities in blue represent the accumulated gradients.

Update the weights and biases

At the end of a mini-batch, update all the weights and biases, using the accumulated gradients:

$$\begin{aligned}
\left[\mathrm{W}^{\mathrm{T}}\right]_{[\mathrm{M}\times\mathrm{N}]} &\leftarrow \left[\mathrm{W}^{\mathrm{T}}\right]_{[\mathrm{M}\times\mathrm{N}]} \ominus \rho \left[\mathrm{D}_{\mathcal{E},\mathrm{W}^{\mathrm{T}}}\right]_{[\mathrm{N}\times\mathrm{M}]} \\
\left[\mathrm{B}\right]_{[\mathrm{N}\times 1]} &\leftarrow \left[\mathrm{B}\right]_{[\mathrm{N}\times 1]} \ominus \rho \left\{\mathcal{D}_{\mathcal{E},\mathrm{B}}\right\}
\end{aligned}$$

where ρ is the learning rate, a hyper-parameter of the training process (i.e., it is not a parameter denoting learning). The minus sign appears because the gradient by definition points "uphill", whereas the objective to reach the global minimum, not the maximum.

5.3 Variations for processing the rows of a dataset

The stochastic gradient descent and the batch-gradient descent are two other variations used during the training phase.

Further, instead of accumulating gradients as we have done here, it is possible to increase performance using certain matrix-based techniques. Consult sources like Nielsen [1] for such variations.

6 Forward pass through each layer

In this section and the next, we will develop the equations used in the forward and backward passes through all the three layers (the fully connected, activation, and loss layers). At this point, it is assumed that the reader is familiar with the ideas of forward and backward passes and mini-batches.

The forward pass is conducted for each r-th row of a mini-batch (which has in all ν number of rows).

Remember, the sizes of the vectors:

- The input vector $\{\mathrm{X}\}$ has the size of $[\mathrm{M}\times 1]$.
- The output vector $\{\mathrm{Y}\}$ has the size of $[1\times\mathrm{N}]$.
- Therefore, in a simple ANN without any hidden layer, all the following vectors have the size of $[1\times\mathrm{N}]$:
  - The bias vector of the fully connected layer, $\{\mathrm{B}\}$.
  - The output of the fully connected layer $\{\mathrm{Z}\}$, which also is the input of the activation layer.
  - The output of the activation layer $\{\mathrm{A}\}$, which also is one of the inputs of the loss layer (the other input for the loss layer being, the target

vector $\{\mathrm{Y}\}$).

Also, remember, we define the weights matrix using the "from input $\{\mathrm{X}\}$ to $\{\mathrm{Z}\}$" convention. Hence,

- The size of the weights matrix, $[\mathrm{W}]$, is $[\mathrm{M} \times \mathrm{N}]$, whereas,
- the size of its transpose, $[\mathrm{W}^{\mathrm{T}}]$ is $[\mathrm{N} \times \mathrm{M}]$.

6.1 Fully connected layer

$$\{\mathrm{z_i}\}_{\mathrm{r}} \;\leftarrow\; [\mathrm{W}^{\mathrm{T}}]\{\mathrm{x_i}\}_{\mathrm{r}} + \{\mathrm{b_i}\}\;; \qquad \forall\; \mathrm{i} \in [1, \mathrm{N}]$$

where the subscript r denotes the index of the current input data-row in the current mini-batch. The parameters, i.e., the weights and biases (printed in the blue color) remain the same for all rows of a mini-batch.

6.2 Activation layer

A detailed treatment in respect of the various activation functions is given in a later section. Here, we note the activation equation in generic terms.

When each element $\mathrm{a_i}$ of the activation vector is a function of only the row-wise corresponding $\mathrm{z_i}$ elements

$$\{\mathrm{a_i}\}_{\mathrm{r}} \;\leftarrow\; \mathcal{A}\Big(\;\{\mathrm{z_i}\}_{\mathrm{r}}\;\Big)\;; \qquad \forall\; \mathrm{i} \in [1, \mathrm{N}]$$

The exact matrix operation depends on the activation function $\mathcal{A}$ chosen. Typical examples include the sigmoid, $\tanh$ or ReLU functions for classification problems, and the identify function (i.e. no processing) for regression problems.

When each element $\mathrm{a_i}$ of the activation vector is a function of all the $\mathrm{z_k}$ elements

This case arises when we use the softmax activation function. Each r-th element of the activation vector produced by the softmax layer, $\mathrm{a_r}$, is a function of not only the row-wise corresponding element $\mathrm{z_r}$ but also all the other $\mathrm{z_k}$ elements of the $\{\mathrm{Z}\}$ vector:

$$\{\mathrm{a_j}\}_{\mathrm{r}} \;\leftarrow\; \mathcal{A}\Big(\;[\mathrm{z_{i \to j}}]_{\mathrm{r}}\;\Big)\;; \qquad \forall\; \mathrm{i} \in [1, \mathrm{N}],\; \forall\; \mathrm{k} \in [1, \mathrm{N}]$$

6.3 Loss layer

$$\mathcal{E}_{\mathrm{r}} \;\leftarrow\; \mathcal{L}\Big(\;\{\mathrm{a_i}\}_{\mathrm{r}},\; \{\mathrm{y_i}\}_{\mathrm{r}}\;\Big)$$

The exact matrix operation depends on the loss function $\mathcal{L}$ chosen. Typical examples include the SSE (sum of squared errors) for regression, and the log-loss i.e. the cross-entropy loss for classification. A detailed treatment of these loss functions is given in a later section.

Realize that $\mathcal{E}_r$ is a single number.

7 Backward pass through each layer

The backward pass is conducted once per row of the mini-batch.

7.1 Loss layer

Backward pass through the loss layer:

$$\left\{ D_{\mathcal{E},A} \right\}_{[N \times 1]} \leftarrow \left\{ \frac{\partial \mathcal{E}}{\partial a_i} \right\}_{[N \times 1]} .$$

The exact expression for the right hand-side will depend on the particular loss layer actually being used.

All the vectors here are of the dimensions $[N \times 1]$.

7.2 Activation layer

Case 1: Each a_i is a function of the corresponding z_i only

In such a case, a backward pass through the activation layer can be written as:

$$\left\{ D_{A,Z} \right\}_{[N \times 1]} \leftarrow \left\{ \frac{\partial a_i}{\partial z_i} \right\}_{[N \times 1]}$$

The exact expression for the right hand-side will depend on the particular activation layer actually being used.

Then, by chain rule,

$$\begin{aligned} \left\{ \frac{\partial \mathcal{E}}{\partial z_i} \right\}_{[N \times 1]} &= \left\{ \left(\frac{\partial \mathcal{E}}{\partial a_i} \right) \left(\frac{\partial a_i}{\partial z_i} \right) \right\}_{[N \times 1]} \\ &= \left\{ \frac{\partial \mathcal{E}}{\partial a_i} \right\}_{[N \times 1]} \circledast \left\{ \frac{\partial a_i}{\partial z_i} \right\}_{[N \times 1]} ; \qquad \forall\, i \in [1, N]. \end{aligned}$$

In other words,

$$\left\{ D_{\mathcal{E},Z} \right\}_{[N \times 1]} \leftarrow \left\{ D_{\mathcal{E},A} \right\}_{[N \times 1]} \circledast \left\{ D_{A,Z} \right\}_{[N \times 1]}$$

All the vectors here are of the dimensions $[N \times 1]$.

Case 2: Each $\mathrm{a_i}$ is a function of all $\mathrm{z_j}$'s i.e., of $\mathrm{z_1, z_2, \ldots, z_N}$

$$\mathrm{a_i} \;=\; \mathcal{F}\left(\; \lfloor \mathrm{z_1} \quad \mathrm{z_2} \quad \ldots \quad \mathrm{z_j} \quad \ldots \quad \mathrm{z_N} \rfloor \; \right) \; ; \qquad \forall \; \mathrm{j} \in [1, \mathrm{N}].$$

The Jacobian matrix involved here is:

$$[\mathrm{D_{A,Z}}]_{[\mathrm{N_A} \times \mathrm{N_Z}]} \;\overset{\text{def}}{=}\; \mathbf{J}_{\mathrm{ij}} \;=\; \left[\frac{\partial \mathrm{a_i}}{\partial \mathrm{z_j}} \right]_{[\mathrm{N_A} \times \mathrm{N_Z}]} ,$$

where $\mathrm{N_Z} = \mathrm{N_A} = \mathrm{N}$, but we have noted them separately only for convenience in reading, in understanding the vector from where a given range comes. See eq. (2) for the full expansion of the Jacobian matrix.

Now, by chain rule,

$$\begin{aligned}
\left\{ \frac{\partial \mathcal{E}}{\partial \mathrm{z_i}} \right\}_{[\mathrm{N_Z} \times 1]} &= \left\{ \sum_{\mathrm{j}=1}^{\mathrm{N_A}} \left(\frac{\partial \mathcal{E}}{\partial \mathrm{a_j}} \right) \left(\frac{\partial \mathrm{a_j}}{\partial \mathrm{z_i}} \right) \right\}_{[\mathrm{N_Z} \times 1]} \\
&= \left[\frac{\partial \mathrm{a_j}}{\partial \mathrm{z_i}} \right]_{[\mathrm{N_Z} \times \mathrm{N_A}]} \left\{ \frac{\partial \mathcal{E}}{\partial \mathrm{a_j}} \right\}_{[\mathrm{N_A} \times 1]} \\
&= (\mathbf{J}_{\mathrm{ij}})^{\mathrm{T}}_{[\mathrm{N_Z} \times \mathrm{N_A}]} \left\{ \frac{\partial \mathcal{E}}{\partial \mathrm{a_j}} \right\}_{[\mathrm{N_A} \times 1]} ,
\end{aligned}$$

Thus, using our matrix notation:

$$\{\mathrm{D}_{\mathcal{E},\mathrm{Z}}\}_{[\mathrm{N_Z} \times 1]} \;\leftarrow\; \left(\; [\mathrm{D_{A,Z}}] \; \right)^{\mathrm{T}}_{[\mathrm{N_Z} \times \mathrm{N_A}]} \; \{\mathrm{D}_{\mathcal{E},\mathrm{A}}\}_{[\mathrm{N_A} \times 1]} . \tag{6}$$

7.3 Fully connected layer

Derivation

The equation for forward pass through the fully connected layer is eq. (5), which we reproduce below with color-coding:

$$\{\mathrm{z_i}\} \;\leftarrow\; [\mathrm{wt_{ij}}]\{\mathrm{x_j}\} \;\oplus\; \{\mathrm{b_j}\}.$$

Of course, at this point in the backward pass,

$$\mathcal{E} \;=\; \mathcal{F}\left(\; \{\mathrm{Z}\} \; \right)$$

So, for the backward pass, by chain rule,

$$\begin{aligned}
\left[\frac{\partial \mathcal{E}}{\partial \mathrm{wt_{ij}}} \right]_{[\mathrm{N} \times \mathrm{M}]} &= \left\{ \frac{\partial \mathcal{E}}{\partial \mathrm{z_i}} \right\}_{[\mathrm{N} \times 1]} \circledast \left[\frac{\partial \mathrm{z_i}}{\partial \mathrm{wt_{ij}}} \right]_{[\mathrm{N} \times \mathrm{M}]} \\
&= \left\{ \frac{\partial \mathcal{E}}{\partial \mathrm{z_i}} \right\}_{[\mathrm{N} \times 1]} \circledast \left\{ \left\lfloor \frac{\partial \mathrm{z_i}}{\partial \mathrm{wt_{i*}}} \right\rfloor_{[1 \times \mathrm{M}]} \right\}_{[\mathrm{N} \times 1]}
\end{aligned}$$

Now, given eq. (5), if we conduct the partial differentiations for the $\mathrm{z_r}$ with respect to the r-th row of $\left[\mathrm{W^T}\right]$, the result turns out to be:

$$\left\lfloor \frac{\partial \mathrm{z_i}}{\partial \mathrm{wt_{i*}}} \right\rfloor_{[1 \times \mathrm{M}]} = \left\lfloor \mathrm{xt_i} \right\rfloor_{[1 \times \mathrm{M}]}$$

where $\mathrm{xt_i}$ is the i-th element of $\left\lfloor \mathrm{X^T} \right\rfloor$, i.e., of the transpose of $\left\{\mathrm{X}\right\}$. Therefore,

$$\left[\frac{\partial \mathcal{E}}{\partial \mathrm{wt_{ij}}} \right]_{[\mathrm{N} \times \mathrm{M}]} = \left\{ \frac{\partial \mathcal{E}}{\partial \mathrm{z_i}} \right\}_{[\mathrm{N} \times 1]} \circledast \left\{ \left\lfloor \mathrm{xt_i} \right\rfloor_{[1 \times \mathrm{M}]} \right\}_{[\mathrm{N} \times 1]}$$

Gradients of errors with respect to weights

Let

$$\left[\mathrm{D_{Z,W^T}}\right]_{[\mathrm{N} \times \mathrm{M}]} \overset{\text{denoted}}{=} \left[\frac{\partial \mathrm{z_i}}{\partial \mathrm{wt_{ij}}} \right]_{[\mathrm{N} \times \mathrm{M}]}.$$

For full expansion of $\left[\frac{\partial \mathrm{z_i}}{\partial \mathrm{wt_{ij}}} \right]$, see eq. (3).

In the light of the immediately preceding paragraph, the gradients of $\left\{\mathrm{Z}\right\}$ with respect to $\left[\mathrm{W^T}\right]$ turn out to be N number of copies of the row-vectors $\left\lfloor \mathrm{X^T} \right\rfloor$ stacked in a column:

$$\left[\mathrm{D_{Z,W^T}}\right]_{[\mathrm{N} \times \mathrm{M}]} \leftarrow \left\{ \begin{array}{c} \left\lfloor \mathrm{X^T} \right\rfloor_{[1 \times \mathrm{M}]} \\ \left\lfloor \mathrm{X^T} \right\rfloor_{[1 \times \mathrm{M}]} \\ \vdots \\ \left\lfloor \mathrm{X^T} \right\rfloor_{[1 \times \mathrm{M}]} \end{array} \right\}_{[\mathrm{N} \times 1]}.$$

Then

$$\left[\mathrm{D_{\mathcal{E},W^T}}\right]_{[\mathrm{N} \times \mathrm{M}]} \leftarrow \left\{\mathrm{D_{\mathcal{E},Z}}\right\}_{[\mathrm{N} \times 1]} \circledast \left[\mathrm{D_{Z,W^T}}\right]_{[\mathrm{N} \times \mathrm{M}]}$$

Gradients of errors with respect to biases

Let

$$\left\{\mathrm{D_{Z,B}}\right\} \overset{\text{denoted}}{=} \left\{ \frac{\partial \mathrm{z_i}}{\partial \mathrm{b_i}} \right\}.$$

Then, it should be the case that

$$\left\{\mathrm{D_{\mathcal{E},B}}\right\} \leftarrow \left\{\mathrm{D_{\mathcal{E},Z}}\right\} \circledast \left\{\mathrm{D_{Z,B}}\right\}$$

However, given eq. (5), it so turns out that

$$\left\{ \frac{\partial z_i}{\partial b_i} \right\} \;=\; \{1.0\}\,.$$

Therefore,

$$\{D_{\mathcal{E},B}\} \leftarrow \{D_{\mathcal{E},Z}\}$$

Gradients of errors with respect to input

In case of networks with hidden layers, there are many pairs of (fully connected layer + activation layers). The output of the activation layer part of the previous pair forms the input for the next hidden pair's fully connected layer part. Hence, errors have to backpropagated to the activation layer part of the previous pair.

Thus, gradients have to be propagated not only to the weights and biases of the current fully connected layer, but also to its input.

Let the index j run over $\{X\}$ and i over $\{Z\}$. Then, reproducing eq. (4), we have

$$z_i \;=\; b_i + \sum_{j=1}^{M} (\; wt_{ij}\; x_j \;)\,.$$

Therefore, for each pair of i and j,

$$\frac{\partial z_i}{\partial x_j} \;=\; wt_{ij} \;=\; w_{ji}$$

So, in our matrix notation:

$$\left[\frac{\partial z_i}{\partial x_j} \right]_{[N \times M]} \;=\; [W^T]_{[N \times M]}$$

Then, by chain rule, for each typical element,

$$\frac{\partial \mathcal{E}}{\partial x_j} \;=\; \frac{\partial z_i}{\partial x_j}\frac{\partial \mathcal{E}}{\partial z_i}$$

So, for all the elements,

$$\begin{aligned}
\left\{ \frac{\partial \mathcal{E}}{\partial x_j} \right\}_{[M \times 1]} &= \left(\left[\frac{\partial z_i}{\partial x_j} \right]_{[N \times M]} \right)^T \left\{ \frac{\partial \mathcal{E}}{\partial z_i} \right\}_{[N \times 1]} \\
&= \left(\; [wt_{ij}]_{[N \times M]} \; \right)^T \left\{ \frac{\partial \mathcal{E}}{\partial z_j} \right\}_{[N \times 1]} \\
&= \left(\; [W^T]_{[N \times M]} \; \right)^T \left\{ \frac{\partial \mathcal{E}}{\partial z_j} \right\}_{[N \times 1]} \\
&= [W]_{[M \times N]} \left\{ \frac{\partial \mathcal{E}}{\partial z_j} \right\}_{[N \times 1]}
\end{aligned}$$

Hence, using our matrix notation:

$$\{D_{\mathcal{E},X}\} \;\leftarrow\; [W]\,\{D_{\mathcal{E},Z}\}$$

8 Activation functions and their gradients

A discussion of the relative advantages and disadvantages of the different activation functions used, is out of the scope of this document.

8.1 ReLU

Forward pass: Activation function

The ReLU (rectified linear unit) is defined as:

$$\mathrm{a_i} \;=\; \begin{cases} 0.0 & \forall\ \mathrm{z_i} < 0.0 \\ \mathrm{z_i} & \forall\ \mathrm{z_i} \geq 0.0 \end{cases}$$

See [4] for a discussion on the fastest way to implement this method in `numpy`. For instance, the multiplication method could be put, using our matrix notation, as:

$$\begin{aligned} \{\mathrm{t}\} &\leftarrow \{\mathrm{Z}\} > \{0.0\} \\ \{\mathrm{A}\} &\leftarrow \{\mathrm{Z}\} \circledast \{\mathrm{t}\} \end{aligned}$$

where $>$ denotes the element-wise binary comparison operator, and its result, the truth-hood vector $\{\mathrm{t}\}$, has either 1.0 or 0.0 as its elements.

Backward pass: Gradient of $\{\mathrm{A}\}$ with respect to $\{\mathrm{Z}\}$

$$\frac{\partial \mathrm{a_i}}{\partial \mathrm{z_i}} \;=\; \begin{cases} 0.0 & \forall\ \mathrm{z_i} < 0.0 \\ 1.0 & \forall\ \mathrm{z_i} \geq 0.0 \end{cases}$$

See the preceding paragraph for examples of how the case-wise assignment of element-values can be realized in matrix notation (or `numpy`).

We thus get the vector $\{\mathrm{D_{A,Z}}\}$ for the ReLU function.

8.2 Sigmoid

Forward pass: Activation function

The sigmoid function is defined as:

$$\mathrm{a_i} \overset{\text{def}}{=} \sigma\left(\mathrm{z_i}\right) \;=\; \frac{1}{1 + \mathrm{e}^{-\mathrm{z_i}}}$$

where i goes over the rows of the respective vectors. The domain of the sigmoid function is $[-\infty, \infty]$ and its range is $(0, 1.0)$ for any finite input.

Hence, in our matrix notation, the sigmoid function becomes:

$$\{\mathrm{A}\} \;\leftarrow\; \{1.0\} \oslash \left(\; \{1.0\} \;\oplus\; \exp\left(\; \{-1.0\} \otimes \{\mathrm{Z}\}\; \right)\right),$$

where it is understood that the $\exp$ function is applied element-wise.

Backward pass: Gradient of $\{A\}$ with respect to $\{Z\}$

The partial derivative of a_i with respect to z_i can be worked out as:

$$\frac{\partial a_i}{\partial z_i} \;=\; \frac{e^{-z_i}}{(1+e^{-z_i})^2} \;=\; a_i\,(1-a_i)$$

Hence, in our matrix notation,

$$\left\{\frac{\partial a_i}{\partial z_i}\right\} \overset{\text{denoted}}{=} \{D_{A,Z}\} \;\leftarrow\; \{A\} \circledast \left(\; \{1.0\} \ominus \{A\} \;\right)$$

8.3 tanh

Forward pass: Activation function

The tanh function is defined as:

$$a_i \overset{\text{def}}{=} \tanh(z_i) \;=\; \frac{1-e^{-2\,z_i}}{1+e^{-2\,z_i}} \;=\; \frac{2}{1+e^{-2\,z_i}} - 1$$

where i goes over the rows of the respective vectors. The domain of the tanh function is $[-\infty, \infty]$ and its range is $(-1.0, 1.0)$ for any finite input.

Hence, in our matrix notation, the tanh function becomes:

$$\{A\} \;\leftarrow\; \left(\; \{2.0\} \oslash \left(\; \{1.0\} \;\oplus\; \exp\left(\; \{-2.0\} \circledast \{Z\} \;\right)\;\right)\;\right) \;\ominus\; \{1.0\}\,,$$

where it is understood that the $\exp$ function is applied element-wise.

Backward pass: Gradient of $\{A\}$ with respect to $\{Z\}$

The partial derivative of a_i with respect to z_i can be derived as:

$$\frac{\partial a_i}{\partial z_i} \;=\; 1 - a_i^2$$

Hence, in our matrix notation,

$$\left\{\frac{\partial a_i}{\partial z_i}\right\} \overset{\text{denoted}}{=} \{D_{A,Z}\} \;\leftarrow\; \{1.0\} \;\ominus\; \left(\; \{a_i\} \circledast \{a_i\} \;\right)$$

Remarks on tanh vs. sigmoid

Andrew Ng reportedly [5] says that using tanh is almost always preferable to using the sigmoid.

8.4 Softmax

The distinctive feature of the softmax function is that the a_i values it produces are such that they all add up to 1.0. Thus:

$$\sum_{i=1}^{N} a_i = 1.0.$$

Hence, we can think of the softmax function as directly providing probabilities for different output classes in the multi-class classification problem that uses logistic regression.

It can be shown [1] that the softmax function is optimal for the maximum-likelihood estimation of the model's parameters (weights and biases).

Forward pass: Activation

The softmax function is defined as:

$$a_i \overset{\text{def}}{=} \frac{e^{z_i}}{\sum\limits_{j=1}^{N} e^{z_j}} \qquad \forall\ i \in [1, N],\ \forall\ j \in [1, N].$$

The domain of the softmax function is $[-\infty, \infty]$ and its range is $(0.0, 1.0)$ for any finite input.

Using our matrix notation,

$$\begin{aligned} \{Q\} &\leftarrow \exp\left(\ \{Z\}\ \right) \\ s &\leftarrow \Sigma_r\left(\ \{Q\}\ \right) \\ \{A\} &\leftarrow \{Q\} \oslash \{s\} \end{aligned}$$

Backward pass: Gradients of $\{A\}$ with respect to $\{Z\}$

Since each a_i is a function of all z_j's, there are N number of partial derivatives for each i-th row, and so, the partial derivatives form an $N \times N$ matrix, i.e. the square-Jacobian.

The specific steps are the following:

$$\begin{aligned} a_i &\overset{\text{def}}{=} \frac{e^{z_i}}{s} & \forall\ i \in [1, N] \\ s &\overset{\text{def}}{=} \sum_{j=1}^{N} e^{z_j} & \forall\ j \in [1, N] \end{aligned}$$

Given this definition, when $i \neq j$, then by product and quotient rules of differentiation,

$$\begin{aligned} \frac{\partial a_i}{\partial z_j} &= e^{z_i} \frac{\partial}{\partial z_j}\left(\frac{1}{s}\right) + \left(\frac{1}{s}\right)\frac{\partial e^{z_i}}{\partial z_j} \\ &= e^{z_i}\left(-\frac{1}{s^2}e^{z_j}\right) + \left(\frac{1}{s}\right)(0) \\ &= -\frac{e^{z_i}}{s}\frac{e^{z_j}}{s} \\ &= -a_i\, a_j \\ &= a_i\left(0 - a_j\right) \\ &= a_i\left(\delta_{ij} - a_j\right), \end{aligned}$$

where δ_{ij} is Kronecker's delta, defined as:

$$\delta_{ij} \overset{\text{def}}{=} \begin{cases} 1 & \forall\ i = j \\ 0 & \forall\ i \neq j \end{cases} .$$

When $i = j$, by product and quotient rules of differentiation,

$$\begin{aligned}
\frac{\partial a_i}{\partial z_i} &= e^{z_i} \frac{\partial}{\partial z_i}\left(\frac{1}{s}\right) + \left(\frac{1}{s}\right)\frac{\partial e^{z_i}}{\partial z_i} \\
&= e^{z_i}\left(-\frac{1}{s^2} e^{z_i} \right) + \frac{1}{s} e^{z_i} \\
&= -\frac{e^{z_i}}{s}\frac{e^{z_i}}{s} + \frac{e^{z_i}}{s} \\
&= -a_i\, a_i + a_i \\
&= a_i\,(\,1 - a_i\,) \\
&= a_i\,(\,\delta_{ij} - a_i\,) \\
&= a_i\,(\,\delta_{ij} - a_j\,) \qquad \because i = j
\end{aligned}$$

To express the above component-wise equations to the matrix form, we use eq. (1), which is repeated here:

$$[O]_{[N \times N]} \overset{\text{def}}{=} \{A\}_{[N \times 1]} \otimes \lfloor 1 \rfloor_{[1 \times N]}$$

To repeat,

$$[O]_{[N \times N]} = \begin{bmatrix} a_1 & a_1 & \dots & a_1 \\ a_2 & a_2 & \dots & a_2 \\ \vdots & \vdots & \ddots & \vdots \\ a_N & a_N & \dots & a_N \end{bmatrix}_{[N \times N]} ,$$

and so,

$$\begin{aligned}
[O^T]_{[N \times N]} &\overset{\text{def}}{=} \left([O]_{[N \times N]} \right)^T \\
&= \begin{bmatrix} a_1 & a_2 & \dots & a_N \\ a_1 & a_2 & \dots & a_N \\ \vdots & \vdots & \ddots & \vdots \\ a_1 & a_2 & \dots & a_N \end{bmatrix}_{[N \times N]}
\end{aligned}$$

Then, the necessary equation is given by:

$$\begin{aligned}
[D_{A,Z}]_{[N_A \times N_Z]} &\overset{\text{def}}{=} \left[\frac{\partial a_i}{\partial z_j}\right] \\
&= [O] \circledast \left([\mathcal{I}]_{[N \times N]} \ominus [O^T] \right)
\end{aligned} \tag{7}$$

Note carefully the use of the outer product operator $\otimes$ and the element-wise multiplication operator $\circledast$.

We can then substitute eq. (7) into eq. (6).

8.5 Identity

In the forward pass, the identify function simply passes $\{\mathrm{Z}\}$ unmodified as $\{\mathrm{A}\}$. It thus is a mere placeholder. Its only utility is so that the linear regression problems (simple, polynomial, or multivariate) can be handled within the framework of ANNs. The loss function used in such a case is the SSE (sum of squared errors).

In the backward pass, the identity function simply passes the gradient of the loss vector $\{\mathcal{E}\}$ with respect to $\{\mathrm{Z}\}$ unmodified.

9 Loss functions and their gradients

The terms "loss", "error", and "cost" are used interchangeably in the context of gradient descent algorithm for artificial neural networks.

9.1 Euclidean loss (SSE)

Note that SSE (or HSSE) produces a convex loss-function surface only when the activation function is linear. Thus, SSE is used only for regression problems, not for classification problems.

Forward pass: Loss function

The half-sum of squared error for the i-th element is given by:

$$\mathcal{E} \overset{\text{def}}{=} \frac{1}{2} \sum_{i=1}^{M} (y_i - a_i)^2$$

Therefore, in our matrix notation,

$$\begin{aligned}
\{d\}_{[N \times 1]} &\leftarrow \{Y\}_{[N \times 1]} \ominus \{A\}_{[N \times 1]} \\
\{\epsilon\}_{[N \times 1]} &\leftarrow \{0.5\}_{[N \times 1]} \circledast \{d\}_{[N \times 1]} \circledast \{d\}_{[N \times 1]} \\
\mathcal{E} &\leftarrow \Sigma_r \left(\{\epsilon\}_{[N \times 1]} \right)
\end{aligned}$$

Backward pass: Gradient of errors with respect to activations

Given the above-mentioned loss function, the gradient can be worked out as:

$$\frac{\partial \epsilon_i}{\partial a_i} = \left(\frac{1}{2}\right)(2)\,(y_i - a_i) = y_i - a_i = d_i$$

Therefore, in our matrix notation,

$$\{D_{\mathcal{E},A}\}_{[N \times 1]} \leftarrow \{Y\}_{[N \times 1]} \ominus \{A\}_{[N \times 1]} .$$

9.2 Log loss/Cross entropy loss

The log-loss function is used when the activation function is the sigmoid, as in the simple binary classification problems (i.e. the logistic regression).

Forward pass: Loss function

$$\mathcal{E} \overset{\text{def}}{=} - \sum_{i=1}^{N} (y_i \log a_i) \, .$$

So, define

$$\epsilon_i \overset{\text{def}}{=} (y_i \log a_i)$$

Then, in our matrix notation,

$$\{\epsilon\}_{[N \times 1]} \leftarrow \{Y\}_{[N \times 1]} \circledast \log\left(\{A\}_{[N \times 1]} \right),$$

where it is understood that the $\log$ function is applied element-wise. Then,

$$\mathcal{E} \leftarrow - \Sigma_r \left(\{\epsilon\}_{[N \times 1]} \right),$$

Backward pass: Gradient of errors with respect to activations

Given the above-mentioned loss function, the gradient can be worked out as:

$$\frac{\partial \epsilon_i}{\partial a_i} = - \frac{y_i}{a_i}$$

Therefore, in our matrix notation,

$$\{D_{\mathcal{E},A}\}_{[N \times 1]} \leftarrow \left(\{-1.0\} \circledast \{Y\} \right) \oslash \{A\}$$

10 Combining layers for simpler tasks

The three layers presented so far form the basic building blocks to construct different kinds of feedforward networks that are designed to perform various tasks. Given below is a brief indication of some of the various combinations that are possible.

10.1 Linear regressions

Performing linear regressions with ANNs is, really speaking, only an issue of terminology.

Linear regression involves only the fully connected layer and the loss layer.

Simple linear regression

The design to follow is:

- Fully connected layer of size $\mathrm{M} = 1$ for the input and $\mathrm{N} = 1$ for its output
- No activation layer (or one with the identity function)
- Loss layer using the SSE loss function.

Multivariable linear regression

The design to follow is:

- Fully connected layer of size $\mathrm{M} > 1$ for the input and $\mathrm{N} = 1$ for its output
- No activation layer (or one with the identity function)
- Loss layer using the SSE loss function.

Multivariate linear regression

The design to follow is:

- Fully connected layer of the size $\mathrm{M} \geq 1$ for the input and of the size $\mathrm{N} > 1$ for its output
- No activation layer (or one with the identity function)
- Loss layer using the SSE loss function.

Polynomial regression

The model here is:

$$\hat{y} \;=\; \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_1 x_2 + \beta_4 x_1^2 + \beta_5 x_2^2$$

Realize that this is an instance of *linear* regressions, and hence, in principle, it can be handled in ANNs.

To make it work in the framework of ANNs, perform some pre-processing by defining new independent variables:

$$\begin{aligned}
\chi_1 &\overset{\text{def}}{=} x_1 \\
\chi_2 &\overset{\text{def}}{=} x_2 \\
\chi_3 &\overset{\text{def}}{=} x_1\, x_2 \\
\chi_4 &\overset{\text{def}}{=} x_1^2 \\
\chi_5 &\overset{\text{def}}{=} x_2^2
\end{aligned}$$

Now, treat $\{\chi_i\}$ as the input vector for any of the previously mentioned linear regressions.

10.2 Nonlinear regression having β^n terms

It is not possible to directly realize an ANN that corresponds to the model:

$$\hat{y} = \beta_0 + \beta_1^2 x$$

The reason is that the fully connected layer, as presented, performs only linear combinations. Therefore, implementing a nonlinear model of this kind would involve using a special-purpose layer in place of the fully connected layer. The loss layer can be used as is.

10.3 Regression for classification

This, again, is an issue only of terminology.

Logistic regression (binary classification)

We take the term "logistic regression" to mean binary classification using the sigmoid function. Accordingly, the design to follow is:

- Fully connected layer of size $M = 1$ for the input, and $N = 1$ for its output
- Activation layer of size $N = 1$ using the sigmoid function
- Loss layer of size $N = 1$ using the binary log-loss function.

Multinomial regression (multi-class classification)

The design to follow is:

- Fully connected layer of size $M \geq 1$ for the input, and $N > 1$ for its output
- Activation layer of size N using the sigmoid function
- Loss layer of size N using the multi-class log-loss function.

11 Simple deep learning networks

The term "deep learning" is typically used for denoting advanced architectures like the convolutional neural networks (ConvNNs or CNNS), and the recurrent neural networks (RNNs). The layers presented in this document are not sufficient to build such advanced architectures. Mind you, they do form parts of CNNs and RNNs too. However, only these layers, by themselves, are not sufficient.

However, technically speaking, any neural network that has hidden layers also is called a deep network.

Using the terminology of layers we introduced, a hidden layer of a deep network consists of a *pair* of a fully connected and an activation layer.

Accordingly, the design to follow is:

- Input stage ("layer")
  - One fully connected layer of the size M
  - Its corresponding activation layer of some arbitrary size H_1^I
- One or more hidden stages ("layers"), *each* consisting of
  - One fully connected layer with size of its input vector the same as the activation vector of the previous stage (which, for the first hidden stage is H_1^I)
  - One activation layer of some arbitrary size H_1^O.

  When multiple hidden layers are used, the size of the input for the next hidden layer $\mathrm{H}_{\mathrm{n+1}}^\mathrm{I}$ is, of course, the same as the size of the output for the previous hidden layer output $\mathrm{H}_\mathrm{n}^\mathrm{O}$.
- Output stage ("layer")
  - Just one loss layer. It takes two inputs: (i) the output of the last hidden layer $\mathrm{H}_{\text{last}}^\mathrm{O}$, and (ii) the target variables vector $\{\mathrm{Y}\}$

During mini-batch processing, for each r-th row of a mini-batch, all the layers are traversed through during the forward and backward passes. The gradients are accumulated in each fully connected layer, until the mini-batch gets over. The weights and biases in each fully connected layer get updated at the end of the mini-batch.

References

[1] Nielsen, Michael (2015) "Neural networks and deep learning", Determination Press. https://neuralnetworksanddeeplearning.com/

[2] Mazur, Matt (2015) "A step by step backpropagation example". https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/

[3] Mallya, Arun (2017) "Writeups/Notes". https://arunmallya.github.io

[4] Online discussion (2015) "How to implement the ReLU function in Numpy", StackOverFlow, https://stackoverflow.com/questions/32109319/how-to-implement-the-relu-function-in-numpy.

[5] Online discussion (2018) "Why is tanh almost always better than sigmoid as an activation function?", Stats.StackExchange, https://stats.stackexchange.com/questions/330559/why-is-tanh-almost-always-better-than-sigmoid-as-an-activation-function