Nerva Library Specifications

Wieger Wesselink

October 1, 2024

Contents

1 Introduction **1**

2 Mathematical background **1**

3 Matrix operations **3**

4 Layer equations **4**
4.1 Derivations 8

5 Activation functions **11**
5.1 Softmax functions 11
5.2 Derivations 12

6 Loss functions **12**
6.1 Derivations 14

7 Weight initialization **14**

8 Optimization **15**

9 Learning Rate Schedulers **15**

1 Introduction

This document contains algorithm specifications for the Nerva Libraries `https://github.com/wiegerw/nerva`. The goal of this document is to provide precise mathematical specifications of key parts of the implementation. The code in the repository `https://github.com/wiegerw/nerva-rowwise` closely matches the equations in this document.

2 Mathematical background

In this section we provide notational conventions, we provide definitions for the gradient and the Jacobian, and a comprehensive table with matrix operations that are necessary for the execution of multilayer perceptrons.

In the rest of this document we use the following notation. Vectors are denoted using lowercase symbols x, y, z, and matrices using uppercase symbols X, Y, Z. The columns of a matrix $X \in \mathbb{R}^{m \times n}$ are denoted as $x^1, \ldots, x^n$, while the rows are denoted as $x_1, \ldots, x_m$. Occasionally, the subscript notation x_i is also used to denote element i of the vector x, but this will be made clear from the context. To distinguish between row

and column vectors, we denote their domains as $\mathbb{R}^{1\times n}$ and $\mathbb{R}^{m\times 1}$, respectively. We use the dot symbol $\cdot$ for matrix multiplication and not for the dot product. This notation is used primarily to enhance the readability of expressions.

In the context of neural networks, we typically use x, X for inputs, y, Y for outputs, z, Z for intermediate values, and t, T for targets. The number of inputs of a layer is denoted by D and the number of outputs by K. The number of examples in a mini-batch is denoted by N. We follow the convention of the major neural network frameworks where data is stored using a row layout. This means that by default x, y, and z are considered row vectors, and each row of an input matrix X represents an example of a data set. For example, if $X \in \mathbb{R}^{\mathrm{N}\times\mathrm{D}}$, then $x_i \in \mathbb{R}^{1\times\mathrm{D}}$ represents the i-th example.

Definition 1 (Gradient)**.** *Let $f : \mathbb{R}^{m\times n} \to \mathbb{R}$ be a function with input X that has elements x_{ij}, with $m, n \in \mathrm{N}^+$. Then the gradient $\nabla_X f$ is defined as*

$$\nabla_X f(X) = \begin{bmatrix} \frac{\partial f(X)}{\partial x_{11}} & \cdots & \frac{\partial f(X)}{\partial x_{1n}} \\ \vdots & \ddots & \vdots \\ \frac{\partial f(X)}{\partial x_{m1}} & \cdots & \frac{\partial f(X)}{\partial x_{mn}} \end{bmatrix}. \tag{1}$$

Definition 2 (Gradient of the loss function)**.** *We consider neural networks with output $Y \in \mathbb{R}^{\mathrm{N}\times\mathrm{K}}$ and target $T \in \mathbb{R}^{\mathrm{N}\times\mathrm{K}}$. We assume that there is a fixed loss function $\mathcal{L} : \mathbb{R}^{\mathrm{N}\times\mathrm{K}} \times \mathbb{R}^{\mathrm{N}\times\mathrm{K}} \to \mathbb{R}$, such that $\mathcal{L}(Y, T)$ is the loss corresponding to the output Y, given the target T. We use the following shorthand notation for the gradient of the loss:*

$$DY = \nabla_Y \mathcal{L}(Y, T). \tag{2}$$

If Y depends on a parameter Z, i.e. $Y = Y(Z)$, then we define

$$DZ = \nabla_Z \mathcal{L}(Y(Z), T). \tag{3}$$

Definition 3 (Jacobian)**.** *Let $f : \mathbb{R}^n \to \mathbb{R}^m$ be a function with input $x \in \mathbb{R}^n$. Then the Jacobian $\frac{\partial f}{\partial x}$ is defined as*[1]

$$\frac{\partial f}{\partial x}(x) = \begin{bmatrix} \frac{\partial f_1(x)}{\partial x_1} & \cdots & \frac{\partial f_1(x)}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m(x)}{\partial x_1} & \cdots & \frac{\partial f_m(x)}{\partial x_n} \end{bmatrix}. \tag{4}$$

The execution of MLPs depends on a small number of matrix operations. In Table 1 we provide a mathematical notation, a code representation, and a definition for the most important operations. These operations are used in the implementation of activation functions, loss functions, and the feedforward and backpropagation equations of neural network layers. Basic operations like matrix assignment, matrix indexing, and matrix dimensions are omitted, to keep the resulting code familiar to users of the respective Python frameworks. There is some redundancy in the table, to allow for more efficient, or numerically stable implementations. We refrain from using broadcasting notation, as commonly found in frameworks like NumPy, where arrays of different dimensions can be added, and use the standard notation of matrix calculus instead. This approach with explicit dimensions is needed to validate the correctness using SymPy. For example, to calculate the sum of elements in a column vector $x \in \mathbb{R}^{n\times 1}$, we express it as the dot product $1_n^\top \cdot x$, with $1_n = (1, \ldots, 1)^\top$ a column vector of ones.

[1] We use the convention that the Jacobian does not depend on whether x and $f(x)$ are row vectors or column vectors. This is consistent with SymPy, but other conventions are also in use.

3 Matrix operations

Table 1: An overview of matrix operations that are needed for the implementation of the class of MLPs described in this paper. We assume that $X, Y \in \mathbb{R}^{m \times n}$ and $Z \in \mathbb{R}^{n \times k}$, where $k, m, n \in \mathbb{N}^+$. Wherever possible, we use m to denote the number of rows and n to denote the number of columns of a matrix or vector. The function σ is the sigmoid function as defined in section 5.

OPERATION	CODE	DEFINITION
0_m	`zeros(m)`	$m \times 1$ column vector with elements equal to 0
0_{mn}	`zeros(m, n)`	$m \times n$ matrix with elements equal to 0
1_m	`ones(m)`	$m \times 1$ column vector with elements equal to 1
1_{mn}	`ones(m, n)`	$m \times n$ matrix with elements equal to 1
$\mathbb{I}_n$	`identity(n)`	$n \times n$ identity matrix
$X^\top$	`X.T`	transposition
cX	`c * X`	scalar multiplication, $c \in \mathbb{R}$
$X + Y$	`X + Y`	addition
$X - Y$	`X - Y`	subtraction
$X \cdot Z$	`X @ Z` or `X * Z`	matrix multiplication, also denoted as XZ
$x^\top y$ or $xy^\top$	`dot(x,y)`	dot product, $x, y \in \mathbb{R}^{m \times 1}$ or $x, y \in \mathbb{R}^{1 \times n}$
$X \odot Y$	`hadamard(X,Y)`	element-wise product of X and Y
$\mathsf{diag}(X)$	`diag(X)`	column vector that contains the diagonal of X
$\mathsf{Diag}(x)$	`Diag(x)`	diagonal matrix with x as diagonal, $x \in \mathbb{R}^{1 \times n}$ or $x \in \mathbb{R}^{m \times 1}$
$1_m^\top \cdot X \cdot 1_n$	`elements_sum(X)`	sum of the elements of X
$x \cdot 1_n^\top$	`column_repeat(x, n)`	n copies of column vector $x \in \mathbb{R}^{m \times 1}$
$1_m \cdot x$	`row_repeat(x, m)`	m copies of row vector $x \in \mathbb{R}^{1 \times n}$
$1_m^\top \cdot X$	`columns_sum(X)`	$1 \times n$ row vector with sums of the columns of X
$X \cdot 1_n$	`rows_sum(X)`	$m \times 1$ column vector with sums of the rows of X
$\max(X)_{\text{col}}$	`columns_max(X)`	$1 \times n$ row vector with maximum values of the columns of X
$\max(X)_{\text{row}}$	`rows_max(X)`	$m \times 1$ column vector with maximum values of the rows of X
$(1_m^\top \cdot X)/n$	`columns_mean(X)`	$1 \times n$ row vector with mean values of the columns of X
$(X \cdot 1_n)/m$	`rows_mean(X)`	$m \times 1$ column vector with mean values of the rows of X
$f(X)$	`apply(f, X)`	element-wise application of $f : \mathbb{R} \to \mathbb{R}$ to X
e^X	`exp(X)`	element-wise application of $f : x \to e^x$ to X
$\log(X)$	`log(X)`	element-wise application of the natural logarithm $f : x \to \ln(x)$ to X
$1/X$	`reciprocal(X)`	element-wise application of $f : x \to 1/x$ to X
$\sqrt{X}$	`sqrt(X)`	element-wise application of $f : x \to \sqrt{x}$ to X
$X^{-1/2}$	`inv_sqrt`(X)	element-wise application of $f : x \to x^{-1/2}$ to X
$\log(\sigma(X))$	`log_sigmoid`(X)	element-wise application of $f : x \to \log(\sigma(x))$ to X,

In Table 2 we provide implementations in several frameworks of the fundamental matrix-form operations for multilayer perceptrons.

Operation	NumPy + JAX	PyTorch	TensorFlow	Eigen
zeros(m,n)	zeros((m,n))	zeros(m,n)	zeros([m,n])	Zero(m,n)
ones(m,n)	ones((m,n))	ones(m,n)	ones([m,n])	Ones(m,n)
identity(n)	eye(n)	eye(n)	eye(n)	Identity(n,n)
X.T	X.T	X.T	X.T	X.transpose()
c * X	c * X	c * X	c * X	c * X
X + Y	X + Y	X + Y	X + Y	X + Y
X - Y	X - Y	X - Y	X - Y	X - Y
X * Z	X @ Z	X @ Z	X @ Z	X * Z
hadamard(X,Y)	X * Y	X * Y	multiply(X,Y)	X.array() * Y.array()
diag(X)	diag(X)	diag(X)	diag_part(X)	X.diagonal()
Diag(x)	diag(x)	diag(x.flatten())	diag(reshape(x,[-1]))	x.asDiagonal()
elements_sum(X)	sum(X)	sum(X)	reduce_sum(X)	X.sum()
column_repeat(x,n)	tile(x,(1,n))	x.repeat(1,n)	tile(x,[1,n])	x.replicate(1,n)
row_repeat(x,m)	tile(x,(m,1))	x.repeat(m,1)	tile(x,[m,1])	x.replicate(m,1)
columns_sum(X)	sum(X,axis=0)	sum(X,dim=0)	reduce_sum(X,axis=0)	X.colwise().sum()
rows_sum(X)	sum(X,axis=1)	sum(X,dim=1)	reduce_sum(X,axis=1)	X.rowwise().sum()
columns_max(X)	max(X,axis=0)	max(X,dim=0).values	reduce_max(X,axis=0)	X.colwise().maxCoeff()
rows_max(X)	max(X,axis=1)	max(X,dim=1).values	reduce_max(X,axis=1)	X.rowwise().maxCoeff()
columns_mean(X)	mean(X,axis=0)	mean(X,dim=0)	reduce_mean(X,axis=0)	X.colwise().mean()
rows_mean(X)	mean(X,axis=1)	mean(X,dim=1)	reduce_mean(X,axis=1)	X.rowwise().mean()
apply(f,X)	f(X)	f(X)	f(X)	X.unaryExpr(f)
exp(X)	exp(X)	exp(X)	exp(X)	X.array().exp()
log(X)	log(X)	log(X)	log(X)	X.array().log()
reciprocal(X)	1 / X	1 / X	inverse(X)	X.array().inverse()
sqrt(X)	sqrt(X)	sqrt(X)	sqrt(X)	X.array().sqrt()
inv_sqrt(X)	reciprocal(sqrt(X+e))	reciprocal(sqrt(X+e))	reciprocal(sqrt(X+e))	reciprocal(sqrt(X.array()+e))
log_sigmoid(X)	-logaddexp(0,-X)	-softplus(-X)	-softplus(-X)	-log1p(exp(-X.array()))

Table 2: Implementation of matrix operations in NumPy, JAX, PyTorch, TensorFlow and Eigen. We assume that `e` is a given small positive constant that is used to avoid division by zero. Note that some of the operations are located in Python submodules.

4 Layer equations

A major contribution of this paper is the following overview of the feedforward and backpropagation equations of layers **in matrix-form**. For each of the equations, the corresponding Python implementation is given. For several equations a derivation is included, while the correctness of the equations and derivations has been validated using SymPy. All layers are implemented in our Nerva Python packages, and a validation of all equations and derivations can be found in the `tests` directory of the `nerva-sympy` package. We consider input batches X in row layout, i.e. each row represents a single example. We use the following notations:

- $X \in \mathbb{R}^{\mathrm{N} \times \mathrm{D}}$ is the input batch, where N is the number of examples and D is the input dimension.
- $Y \in \mathbb{R}^{\mathrm{N} \times \mathrm{K}}$ is the output batch, where K is the output dimension.
- $W \in \mathbb{R}^{\mathrm{K} \times \mathrm{D}}$ is the weight matrix, which maps the input features to the output features.
- $b \in \mathbb{R}^{1 \times \mathrm{K}}$ is the bias vector.
- $Z \in \mathbb{R}^{\mathrm{N} \times \mathrm{K}}$ is a matrix with intermediate values.
- $\beta, \gamma, \Sigma \in \mathbb{R}^{1 \times \mathrm{K}}$ are the parameters of batch normalization.
- $a_l, t_l, a_r, t_r \in \mathbb{R}$ are the coefficients of an SReLU activation function.
- $R \in \mathbb{R}^{\mathrm{K} \times \mathrm{D}}$ is a dropout matrix.

Each of these matrices has a gradient with the same dimensions, denoted using the same symbol preceded by D, e.g. $\mathsf{D}X$ is the gradient corresponding to X. The implementation uses the same names. The input `X`, and layer parameters like `W`, `b` and their gradients `DX`, `DW`, `Db` are stored in the attributes of a layer. The only exception is the output `Y` and its gradient `DY`, which are stored in the `X` and `DX` attributes of the next layer.

linear layer

FEEDFORWARD EQUATIONS

$$Y = XW^\top + 1_\mathrm{N} \cdot b$$

```
Y = X * W.T + row_repeat(b, N)
```

BACKPROPAGATION EQUATIONS

$$\begin{aligned}\mathsf{D}W &= \mathsf{D}Y^\top \cdot X \\ \mathsf{D}b &= 1_\mathrm{N}^\top \cdot \mathsf{D}Y \\ \mathsf{D}X &= \mathsf{D}Y \cdot W\end{aligned}$$

```
DW = DY.T * X
Db = columns_sum(DY)
DX = DY * W
```

activation layer

Let $\mathsf{act} : \mathbb{R} \to \mathbb{R}$ be an activation function, for example relu.

FEEDFORWARD EQUATIONS

$$\begin{aligned}Z &= XW^\top + 1_\mathrm{N} \cdot b \\ Y &= \mathsf{act}(Z)\end{aligned}$$

```
Z = X * W.T + row_repeat(b, N)
Y = act(Z)
```

BACKPROPAGATION EQUATIONS

$$\begin{aligned}\mathsf{D}Z &= \mathsf{D}Y \odot \mathsf{act}'(Z) \\ \mathsf{D}W &= \mathsf{D}Z^\top \cdot X \\ \mathsf{D}b &= 1_\mathrm{N}^\top \cdot \mathsf{D}Z \\ \mathsf{D}X &= \mathsf{D}Z \cdot W\end{aligned}$$

```
DZ = hadamard(DY, act.gradient(Z))
DW = DZ.T * X
Db = columns_sum(DZ)
DX = DZ * W
```

srelu-layer

FEEDFORWARD EQUATIONS

$$Z = XW^\top + 1_N \cdot b$$
$$Y = \mathsf{srelu}(Z)$$

BACKPROPAGATION EQUATIONS

$$\mathsf{D}Z = \mathsf{D}Y \odot \mathsf{srelu}'(Z)$$
$$\mathsf{D}W = \mathsf{D}Z^\top \cdot X$$
$$\mathsf{D}b = 1_N^\top \cdot \mathsf{D}Z$$
$$\mathsf{D}X = \mathsf{D}Z \cdot W$$
$$\mathsf{D}a_l = 1_N^\top \cdot (\mathsf{D}Y \odot A^l) \cdot 1_K$$
$$\mathsf{D}a_r = 1_N^\top \cdot (\mathsf{D}Y \odot A^r) \cdot 1_K$$
$$\mathsf{D}t_l = 1_N^\top \cdot (\mathsf{D}Y \odot T^l) \cdot 1_K$$
$$\mathsf{D}t_r = 1_N^\top \cdot (\mathsf{D}Y \odot T^r) \cdot 1_K$$

```
Z = X * W.T + repeat_row(b, N)
Y = apply(act, Z)
```

```
DZ = hadamard(DY, apply(act_prime, Z))
DW = DZ.T * X
Db = sum_columns(DZ)
DX = DZ * W
Al = apply(al, Z)
Ar = apply(ar, Z)
Tl = apply(tl, Z)
Tr = apply(tr, Z)
Dal = sum_elements(hadamard(DY, Al))
Dar = sum_elements(hadamard(DY, Ar))
Dtl = sum_elements(hadamard(DY, Tl))
Dtr = sum_elements(hadamard(DY, Tr))
```

where

$$A^l_{ij} = \begin{cases} Z_{ij} - t_l & \text{if } Z_{ij} \le t_l \\ 0 & \text{otherwise} \end{cases}$$

$$A^r_{ij} = \begin{cases} 0 & \text{if } Z_{ij} \le t_l \vee Z_{ij} < t_r \\ Z_{ij} - t_r & \text{otherwise} \end{cases}$$

$$T^l_{ij} = \begin{cases} 1 - a_l & \text{if } Z_{ij} \le t_l \\ 0 & \text{otherwise} \end{cases}$$

$$T^r_{ij} = \begin{cases} 1 - a_r & \text{if } Z_{ij} \ge t_r \\ 0 & \text{otherwise} \end{cases}$$

softmax layer

FEEDFORWARD EQUATIONS

$$Z = XW^\top + 1_\mathrm{N} \cdot b$$
$$Y = \mathsf{softmax}(Z)$$

BACKPROPAGATION EQUATIONS

$$\mathsf{D}Z = Y \odot (\mathsf{D}Y - \mathsf{diag}(\mathsf{D}Y \cdot Y^\top) \cdot 1_\mathrm{K}^\top)$$
$$\mathsf{D}W = \mathsf{D}Z^\top \cdot X$$
$$\mathsf{D}b = 1_\mathrm{N}^\top \cdot \mathsf{D}Z$$
$$\mathsf{D}X = \mathsf{D}Z \cdot W$$

```
Z = X * W.T + row_repeat(b, N)
Y = softmax(Z)
```

```
DZ = hadamard(Y,
         DY - column_repeat(diag(DY * Y.T), K))
DW = DZ.T * X
Db = columns_sum(DZ)
DX = DZ * W
```

log-softmax layer

FEEDFORWARD EQUATIONS

$$Z = XW^\top + 1_\mathrm{N} \cdot b$$
$$Y = \mathsf{logsoftmax}(Z)$$

```
Z = X * W.T + row_repeat(b, N)
Y = log_softmax(Z)
```

BACKPROPAGATION EQUATIONS

$$\mathsf{D}Z = \mathsf{D}Y - \mathsf{softmax}(Z) \odot (\mathsf{D}Y \cdot 1_\mathrm{K} \cdot 1_\mathrm{K}^\top)$$
$$\mathsf{D}W = \mathsf{D}Z^\top \cdot X$$
$$\mathsf{D}b = 1_\mathrm{N}^\top \cdot \mathsf{D}Z$$
$$\mathsf{D}X = \mathsf{D}Z \cdot W$$

```
DZ = DY - hadamard(softmax(Z),
                   column_repeat(rows_sum(DY), K))
DW = DZ.T * X
Db = columns_sum(DZ)
DX = DZ * W
```

batch normalization layer

FEEDFORWARD EQUATIONS

$$R = X - \frac{1_\mathrm{N} \cdot 1_\mathrm{N}^\top}{\mathrm{N}} \cdot X$$
$$\Sigma = \frac{1}{\mathrm{N}} \cdot \mathsf{diag}(R^\top R)^\top$$
$$Z = (1_\mathrm{N} \cdot \Sigma^{-\frac{1}{2}}) \odot R$$
$$Y = (1_\mathrm{N} \cdot \gamma) \odot Z + 1_\mathrm{N} \cdot \beta$$

```
R = X - row_repeat(columns_mean(X), N)
Sigma = diag(R.T * R).T / N
inv_sqrt_Sigma = inv_sqrt(Sigma)
Z = hadamard(row_repeat(
             inv_sqrt_Sigma, N), R)
Y = hadamard(row_repeat(gamma, N), Z) +
             row_repeat(beta, N)
```

BACKPROPAGATION EQUATIONS

$$\mathsf{D}Z = (1_\mathrm{N} \cdot \gamma) \odot \mathsf{D}Y$$
$$\mathsf{D}\beta = 1_\mathrm{N}^\top \cdot \mathsf{D}Y$$
$$\mathsf{D}\gamma = 1_\mathrm{N}^\top \cdot (Z \odot \mathsf{D}Y)$$
$$\mathsf{D}X = \left(\frac{1}{\mathrm{N}} \cdot 1_\mathrm{N} \cdot \Sigma^{-\frac{1}{2}}\right) \odot \left((\mathrm{N} \cdot \mathbb{I}_\mathrm{N} - 1_\mathrm{N} \cdot 1_\mathrm{N}^\top) \cdot \mathsf{D}Z - Z \odot (1_\mathrm{N} \cdot \mathsf{diag}(Z^\top \cdot \mathsf{D}Z)^\top)\right)$$

```
DZ = hadamard(row_repeat(gamma, N), DY)
Dbeta = columns_sum(DY)
Dgamma = columns_sum(hadamard(Z, DY))
DX = hadamard(row_repeat(inv_sqrt_Sigma / N, N),
        (N * identity(N) - ones(N, N)) * DZ -
        hadamard(Z, row_repeat(diag(Z.T * DZ).T, N)))
```

linear dropout layer

FEEDFORWARD EQUATIONS

$$Y = X(W^\top \odot R) + 1_\mathrm{N} \cdot b$$

```
Y = X * hadamard(W.T, R) + row_repeat(b, N)
```

BACKPROPAGATION EQUATIONS

$$\mathsf{D}W = (\mathsf{D}Y^\top \cdot X) \odot R^\top$$
$$\mathsf{D}b = 1_\mathrm{N}^\top \cdot \mathsf{D}Y$$
$$\mathsf{D}X = \mathsf{D}Y(W \odot R^\top)$$

```
DW = hadamard(DY.T * X, R.T)
Db = columns_sum(DY)
DX = DY * hadamard(W, R.T)
```

activation dropout layer

FEEDFORWARD EQUATIONS

$$Z = X(W^\top \odot R) + 1_\mathrm{N} \cdot b$$
$$Y = \mathsf{act}(Z)$$

```
Z = X * hadamard(W.T, R) + row_repeat(b, N)
Y = act(Z)
```

BACKPROPAGATION EQUATIONS

$$\mathsf{D}Z = \mathsf{D}Y \odot \mathsf{act}'(Z)$$
$$\mathsf{D}W = (\mathsf{D}Z^\top \cdot X) \odot R^\top$$
$$\mathsf{D}b = 1_\mathrm{N}^\top \cdot \mathsf{D}Z$$
$$\mathsf{D}X = \mathsf{D}Z(W \odot R^\top)$$

```
DZ = hadamard(DY, act.gradient(Z))
DW = hadamard(DZ.T * X, R.T)
Db = columns_sum(DZ)
DX = DZ * hadamard(W, R.T)
```

4.1 Derivations

In this section we give some derivations of the backpropagation equations. We give some applications of the product rule and chain rule for vector functions, and show how some properties on the rows and columns of a matrix can be generalized into matrix-form.

Lemma 1 (Product Rule for vector functions)**.** *The product rule for scalar functions u and v is given by*

$$(u \cdot v)' = u' \cdot v + u \cdot v'.$$

It can be generalized to vector functions, but the result is sensitive to the orientation of the operands. Below we give four concrete applications of the product rule for vector functions. Let $x \in \mathbb{R}^p$, $A \in \mathbb{R}^{m \times n}$, and $h(x) = f(x)g(x)$ for $m, n, p \in \mathbb{N}^+$.

$$\text{Let } f(x) \in \mathbb{R}^{n \times 1}, \text{ and let } g(x) \in \mathbb{R}, \text{ then } \frac{\partial h}{\partial x} = \frac{\partial f}{\partial x} g + f \frac{\partial g}{\partial x}. \tag{5}$$

$$\text{Let } f(x) \in \mathbb{R}^{1 \times n}, \text{ and let } g(x) \in \mathbb{R}, \text{ then } \frac{\partial h}{\partial x} = \frac{\partial f}{\partial x} g + f^\top \frac{\partial g}{\partial x}. \tag{6}$$

$$\text{Let } f(x) = A, \text{ and let } g(x) \in \mathbb{R}^{n \times 1}, \text{ then } \frac{\partial h}{\partial x} = A \frac{\partial g}{\partial x}. \tag{7}$$

$$\text{Let } f(x) \in \mathbb{R}^{1 \times m}, \text{ and let } g(x) = A, \text{ then } \frac{\partial h}{\partial x} = A^\top \frac{\partial f}{\partial x}. \tag{8}$$

Lemma 2 (Chain rule for vector functions)**.** *Let $f : \mathbb{R}^n \to \mathbb{R}^m$, let $g : \mathbb{R}^m \to \mathbb{R}^p$, and let $h(x) = g(y)$ with $y = f(x)$. Then we have*

$$\frac{\partial h}{\partial x} = \frac{\partial g}{\partial y} \cdot \frac{\partial f}{\partial x} \tag{9}$$

Note that this equations holds irrespective of whether $f(x)$ is a row or a column vector.

Property 1 (Matrix properties)**.** *Let $X, Y, Z \in \mathbb{R}^{m \times n}$. We denote the i^{th} row of matrix A as a_i and the j^{th} column of matrix A as a^j. The element at position (i, j) of A is denoted as a_{ij}. Below we give a number of properties on the rows and columns of matrices, and the generalization of these properties into matrix-form.*

$$\textit{If } z^j = x^j \cdot (x^j)^\top \cdot y^j \quad (1 \le j \le n), \textit{ then } Z = X \odot (1_m \cdot \mathsf{diag}(X^\top \cdot Y)^\top). \tag{10}$$

$$\textit{If } z^j = 1_m \cdot (x^j)^\top \cdot y^j \quad (1 \le j \le n), \textit{ then } Z = 1_m \cdot \mathsf{diag}(X^\top Y)^\top. \tag{11}$$

$$\textit{If } z^j = x^j \cdot 1_m^\top \cdot y^j \quad (1 \le j \le n), \quad \textit{then } Z = X \odot (1_m \cdot 1_m^\top \cdot Y) \tag{12}$$

$$\textit{If } z_i = x_i \cdot y_i^\top \cdot y_i \quad (1 \le i \le m), \quad \textit{then } Z = (\mathsf{diag}(X \cdot Y^\top) \cdot 1_n^\top) \odot Y \tag{13}$$

$$\textit{If } z_i = x_i \cdot y_i^\top \cdot 1_n^\top \quad (1 \le i \le m), \quad \textit{then } Z = \mathsf{diag}(X \cdot Y^\top) \cdot 1_n^\top \tag{14}$$

$$\textit{If } z_i = x_i \cdot 1_n \cdot y_i \quad (1 \le i \le m), \quad \textit{then } Z = (X \cdot 1_n \cdot 1_n^\top) \odot Y \tag{15}$$

All properties have been validated with SymPy. Naturally there is a lot of symmetry between the row and column properties. By means of example we will prove equation (13). From $z_i = x_i \cdot y_i^\top \cdot y_i$ we derive that $z_{ij} = (x_i \cdot y_i^\top) y_{ij}$ for $1 \le j \le n$. Hence we can write $Z = R \odot Y$, where R is defined using $r_{ij} = (x_i \cdot y_i^\top)$. We observe that $\mathsf{diag}(X \cdot Y^\top) = (x_1 \cdot y_1^\top, \ldots, x_m \cdot y_m^\top)^\top$. From the definition of R we can see that it consists of n copies of the column vector $\mathsf{diag}(X \cdot Y^\top)$. Hence we have $R = \mathsf{diag}(X \cdot Y^\top) \cdot 1_n^\top$.

linear layer

We have $Y = X \cdot W^\top + 1_\mathrm{N} \cdot b$. Let x_i, y_i, b_i be the i^th row of X, Y, $1_\mathrm{N} \cdot b$ for $1 \le i \le \mathrm{N}$, hence $y_i = x_i W^\top + b_i$, where $b_i = b$. Furthermore, let w_j be the j^th row of W, and let e_i be the i^th row of the unit matrix $\mathbb{I}_\mathrm{N}$. Let $\mathcal{L}(Y) = \sum_{i=1}^\mathrm{N} \mathcal{L}(y_i)$ be the corresponding loss. We calculate

$$\frac{\partial \mathcal{L}}{\partial x_i} \overset{(9)}{=} \sum_{k=1}^\mathrm{N} \frac{\partial \mathcal{L}}{\partial y_k} \frac{\partial y_k}{\partial x_i} = \frac{\partial \mathcal{L}}{\partial y_i} \frac{\partial y_i}{\partial x_i} \overset{(8)}{=} \frac{\partial \mathcal{L}}{\partial y_i} \cdot W, \text{ hence } \mathsf{D}x_i = \mathsf{D}y_i \cdot W$$

$$\frac{\partial \mathcal{L}}{\partial b} \overset{(9)}{=} \sum_{k=1}^\mathrm{N} \frac{\partial \mathcal{L}}{\partial y_k} \frac{\partial y_k}{\partial b} \overset{(8)}{=} \sum_{k=1}^\mathrm{N} \frac{\partial \mathcal{L}}{\partial y_k} \cdot \mathbb{I}_K = \sum_{k=1}^\mathrm{N} \frac{\partial \mathcal{L}}{\partial y_k}, \text{ hence } \mathsf{D}b = 1_\mathrm{N}^\top \cdot \mathsf{D}Y$$

$$\text{We have } y_k = x_k W^\top + b \text{ and } \frac{\partial y_k}{\partial w_{ij}} = x_{kj} e_i, \text{ hence} \tag{16}$$

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial w_{ij}} &= \sum_{k=1}^\mathrm{N} \frac{\partial \mathcal{L}}{\partial y_k} \frac{\partial y_k}{\partial w_{ij}} = \sum_{k=1}^\mathrm{N} \frac{\partial \mathcal{L}}{\partial y_k} x_{kj} e_i = \sum_{k=1}^\mathrm{N} \sum_{l=1}^\mathrm{K} \frac{\partial \mathcal{L}}{\partial y_{kl}} x_{kj} (e_i)_l \\ &= \sum_{k=1}^\mathrm{N} (e_i)_l \sum_{l=1}^\mathrm{K} \frac{\partial \mathcal{L}}{\partial y_{kl}} x_{kj} = \sum_{k=1}^\mathrm{N} (e_i)_l (\mathsf{D}Y^\top \cdot X)_{lj} = (\mathsf{D}Y^\top \cdot X)_{ij} \end{aligned}$$

This can be generalized to matrix equations: $\mathsf{D}X = \mathsf{D}Y \cdot W$, $\mathsf{D}b = 1_\mathrm{N}^\top \cdot \mathsf{D}Y$, and $\mathsf{D}W = \mathsf{D}Y^\top \cdot X$.

log-softmax layer

We have $Y = \mathsf{log\text{-}softmax}(Z)$. Let y_i, z_i be the i^th row of Y, Z for $1 \le i \le \mathrm{N}$, hence $y_i = \mathsf{log\text{-}softmax}(z_i)$. We calculate

$$\begin{aligned} &\frac{\partial \mathcal{L}}{\partial z_i} \overset{(9)}{=} \frac{\partial \mathcal{L}}{\partial y_i} \frac{\partial}{\partial z_i} \mathsf{log\text{-}softmax}(z_i) \overset{(28)}{=} \frac{\partial \mathcal{L}}{\partial y_i} (\mathbb{I}_K - 1_\mathrm{K} \cdot \mathsf{softmax}(z_i)) \\ &\quad \iff \quad \mathsf{D}z_i = \mathsf{D}y_i \cdot (\mathbb{I}_K - 1_\mathrm{K} \cdot \mathsf{softmax}(z_i)) = \mathsf{D}y_i - \mathsf{D}y_i \cdot 1_\mathrm{K} \cdot \mathsf{softmax}(z_i). \end{aligned}$$

This can be generalized to matrices using property 15:

$$\mathsf{D}Z = \mathsf{D}Y - \mathsf{softmax}(Z) \odot (\mathsf{D}Y \cdot 1_\mathrm{K} \cdot 1_\mathrm{K}^\top) \tag{17}$$

batch normalization layer

We will derive the equation for $\mathsf{D}X$, which is the most complicated one. Following the approach of [Yeh17], we first derive the equations for a single column. Let $x^j, r^j, z^j \in \mathbb{R}^{\mathrm{N}\times 1}$ be the j^{th} column of X, R and Z, and let $\sigma \in \mathbb{R}$ be the j^{th} element of Σ, with $1 \leq j \leq \mathrm{D}$. Then we obtain the following equations:

$$r^j = x^j - \frac{1_\mathrm{N} \cdot 1_\mathrm{N}^\top}{\mathrm{N}} \cdot x^j = (\mathbb{I}_\mathrm{N} - \frac{1_\mathrm{N} \cdot 1_\mathrm{N}^\top}{\mathrm{N}}) \cdot x^j \tag{18}$$

$$\sigma = \frac{(r^j)^\top r^j}{\mathrm{N}} \tag{19}$$

$$z^j = r^j \cdot \sigma^{-\frac{1}{2}} \tag{20}$$

We calculate

$$\frac{\partial z^j}{\partial r^j} \overset{(5)}{=} \frac{\partial r^j}{\partial r^j} \cdot \sigma^{-\frac{1}{2}} + r^j \cdot \frac{\partial \sigma^{-\frac{1}{2}}}{\partial r^j} \overset{(9)}{=} \sigma^{-\frac{1}{2}} - r^j \cdot \frac{\sigma^{-\frac{3}{2}}}{2} \cdot \frac{\partial \sigma}{\partial r^j} = \sigma^{-\frac{1}{2}} - r^j \cdot \frac{\sigma^{-\frac{3}{2}}}{2} \cdot \left(\frac{2(r^j)^\top}{\mathrm{N}}\right) \tag{21}$$

$$= \frac{\sigma^{-\frac{1}{2}}}{\mathrm{N}} \left(\mathrm{N} \cdot \mathbb{I}_\mathrm{N} - \sigma^{-1} r^j (r^j)^\top\right) = \frac{\sigma^{-\frac{1}{2}}}{\mathrm{N}} \cdot \left(\mathrm{N} \cdot \mathbb{I}_\mathrm{N} - z^j (z^j)^\top\right). \tag{22}$$

Using the chain rule we find

$$\frac{\partial \mathcal{L}}{\partial r^j} \overset{(9)}{=} \frac{\partial \mathcal{L}}{\partial z^j} \frac{\partial z^j}{\partial r^j}, \text{ hence } \mathsf{D}r^j = \frac{\sigma^{-\frac{1}{2}}}{\mathrm{N}} \left(\mathrm{N} \cdot \mathbb{I}_\mathrm{N} - z^j (z^j)^\top\right) \cdot \mathsf{D}z^j \tag{23}$$

$$\frac{\partial \mathcal{L}}{\partial x^j} \overset{(9)}{=} \frac{\partial \mathcal{L}}{\partial r^j} \frac{\partial r^j}{\partial x^j}, \text{ hence } \mathsf{D}x^j = (\mathbb{I}_\mathrm{N} - \frac{1_\mathrm{N} \cdot 1_\mathrm{N}^\top}{\mathrm{N}}) \cdot \mathsf{D}r^j, \tag{24}$$

where one needs to take into account that for a column vector x we have $\mathsf{D}x = (\frac{\partial \mathcal{L}}{\partial x})^\top$. Hence

$$\begin{aligned}
\mathsf{D}x^j &= \frac{\sigma^{-\frac{1}{2}}}{\mathrm{N}} \cdot (\mathbb{I}_\mathrm{N} - \frac{1_\mathrm{N} \cdot 1_\mathrm{N}^\top}{\mathrm{N}}) \cdot \left(\mathrm{N} \cdot \mathbb{I}_\mathrm{N} - z^j \cdot (z^j)^\top\right) \cdot \mathsf{D}z^j \\
&= \frac{\sigma^{-\frac{1}{2}}}{\mathrm{N}} \cdot (\mathrm{N} \cdot \mathbb{I}_\mathrm{N} - z^j \cdot (z^j)^\top - 1_\mathrm{N} \cdot 1_\mathrm{N}^\top + \frac{1_\mathrm{N} \cdot 1_\mathrm{N}^\top \cdot z^j \cdot (z^j)^\top}{\mathrm{N}}) \cdot \mathsf{D}z^j \\
&\quad \{ \text{ In batch normalization the column sum } 1_\mathrm{N}^\top \cdot z^j \text{ evaluates to zero. } \} \\
&= \frac{\sigma^{-\frac{1}{2}}}{\mathrm{N}} \cdot ((\mathrm{N} \cdot \mathbb{I}_\mathrm{N} - 1_\mathrm{N} \cdot 1_\mathrm{N}^\top) \cdot \mathsf{D}z^j - z^j \cdot (z^j)^\top \cdot \mathsf{D}z^j)
\end{aligned}$$

Using property (10) we generalize this into matrix-form:

$$\mathsf{D}X = (\frac{1}{\mathrm{N}} \cdot 1_\mathrm{N} \cdot \Sigma^{-\frac{1}{2}}) \odot \left((\mathrm{N} \cdot \mathbb{I}_\mathrm{N} - 1_\mathrm{N} \cdot 1_\mathrm{N}^\top) \cdot \mathsf{D}Z - Z \odot (1_\mathrm{N} \cdot \mathsf{diag}(Z^\top \cdot \mathsf{D}Z)^\top)\right). \tag{25}$$

For the remaining equations, let $y_i, z_i, \beta_i, \gamma_i$ be the i^{th} row of Y, Z, $1_\mathrm{N} \cdot \beta$, and $1_\mathrm{N} \cdot \gamma$ for $1 \leq i \leq \mathrm{N}$, where $\beta_i = \beta$, and $\gamma_i = \gamma$. Hence $y_i = \gamma_i \odot z_i + \beta_i$. From this it follows

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial \beta_i} &= \frac{\partial \mathcal{L}}{\partial y_i} \\
\frac{\partial \mathcal{L}}{\partial \gamma_i} &= \frac{\partial \mathcal{L}}{\partial y_i} \odot z_i,
\end{aligned}$$

which we can generalize into matrix-form: $\mathsf{D}\beta = 1_\mathrm{N}^\top \cdot \mathsf{D}Y$ and $\mathsf{D}\gamma = 1_\mathrm{N}^\top \cdot (\mathsf{D}Y \odot Z)$.

5 Activation functions

In this section we give an overview of some commonly used univariate activation functions. These activation functions are typically applied element-wise to the output matrix Y of a neural network.

NAME	FUNCTION	DERIVATIVE
ReLU [Fuk75]	$\mathsf{relu}(x) = \max(0, x)$	$\mathsf{relu}'(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{otherwise} \end{cases}$
Leaky ReLU [Maa13]	$\mathsf{leaky\text{-}relu}(x) = \max(\alpha x, x)$	$\mathsf{leaky\text{-}relu}'(x) = \begin{cases} \alpha & \text{if } x < \alpha x \\ 1 & \text{otherwise} \end{cases}$
All-ReLU ([CMP21])	$\mathsf{all\text{-}relu}(x) = \begin{cases} \alpha x & \text{if } x < 0 \\ x & \text{otherwise} \end{cases}$	$\mathsf{all\text{-}relu}'(x) = \begin{cases} \alpha & \text{if } x < 0 \\ 1 & \text{otherwise} \end{cases}$
SReLU ([JXF+16])	$\mathsf{srelu}(x) = \begin{cases} t_l + a_l(x - t_l) & \text{if } x \leq t_l \\ x & \text{if } t_l < x < t_r \\ t_r + a_r(x - t_r) & \text{if } x \geq t_r \end{cases}$	$\mathsf{srelu}'(x) = \begin{cases} a_l & \text{if } x \leq t_l \\ 1 & \text{if } t_l < x < t_r \\ a_r & \text{if } x \geq t_r \end{cases}$
Hyperbolic tangent	$\mathsf{tanh}(x) = \dfrac{e^x - e^{-x}}{e^x + e^{-x}}$	$\mathsf{tanh}'(x) = 1 - \mathsf{tanh}(x)^2$
Sigmoid / logistic function [HDY+12]	$\sigma(x) = \dfrac{1}{1 + e^{-x}}$	$\sigma'(x) = \sigma(x)(1 - \sigma(x))$

Note that Leaky ReLU and All-ReLU depend on a parameter $\alpha \in \mathbb{R}$ and SReLU depends on four parameters $a_l, t_l, a_r, t_r \in \mathbb{R}$. N.B. The three cases of the SReLU function overlap if $t_l \geq t_r$. We use the convention that in such a case the first possible alternative must be chosen. For All-ReLU and SReLU layers the sign of the parameter α should be alternated. In a network with layers $1, 2, 3, \ldots$ the All-ReLU function should be applied to the layers $2, 3, 4, \ldots$ with parameters $-\alpha, \alpha, -\alpha, \ldots$.

The SReLU parameters a_l, t_l, a_r, t_r are learnable parameters, i.e. they should be updated periodically based on the values of their gradients with respect to the loss function. As initial values we take $a_l, t_l, a_r, t_r = 0, 0, 0, 1$. This corresponds to the ReLU function.

5.1 Softmax functions

In this section we give an overview of softmax functions and their derivatives, both for single inputs and for batches. These equations are present in the activation function of softmax layers and in some of the loss functions in section 6. We use the same notation as before, where $x \in \mathbb{R}^{1 \times \mathrm{D}}$ is a single input with dimension D, and $X \in \mathbb{R}^{\mathrm{N} \times \mathrm{D}}$ is an input batch, where N is the number of samples in the batch. We denote the rows of X as $x_1, \ldots, x_\mathrm{N}$.

Below an overview of softmax functions and their derivatives is given. Equations in matrix-form are only given if they are needed later on. Note that the function $\mathsf{log\text{-}softmax}$ is defined using

$$\mathsf{log\text{-}softmax}(x) = \log(\mathsf{softmax}(x)).$$

VECTOR FUNCTIONS	MATRIX FUNCTIONS
$\mathsf{softmax}(x) = \frac{e^x}{e^x \cdot 1_{\mathrm{D}}}$	$\mathsf{softmax}(X) = e^X \odot \left(\frac{1}{e^X \cdot 1_{\mathrm{D}}} \cdot 1_{\mathrm{D}}^\top\right)$
$\mathsf{stable\text{-}softmax}(x) = \frac{e^z}{e^z \cdot 1_{\mathrm{D}}}$	$\mathsf{stable\text{-}softmax}(X) = e^Z \odot \left(\frac{1}{e^Z \cdot 1_{\mathrm{D}}} \cdot 1_{\mathrm{D}}^\top\right)$
$\mathsf{log\text{-}softmax}(x) = x - \log(e^x \cdot 1_{\mathrm{D}}) \cdot 1_{\mathrm{D}}^\top$	$\mathsf{log\text{-}softmax}(X) = X - \log(e^X \cdot 1_{\mathrm{D}}) \cdot 1_{\mathrm{D}}^\top$
$\mathsf{stable\text{-}log\text{-}softmax}(x) = z - \log(e^z \cdot 1_{\mathrm{D}}) \cdot 1_{\mathrm{D}}^\top$	$\mathsf{stable\text{-}log\text{-}softmax}(X) = Z - \log(e^Z \cdot 1_{\mathrm{D}}) \cdot 1_{\mathrm{D}}^\top,$

where

$$\begin{cases} z = x - \max(x) \cdot 1_{\mathrm{D}}^\top \\ Z = X - \max(X)_{\mathrm{row}} \cdot 1_{\mathrm{D}}^\top. \end{cases}$$

The derivatives of the softmax functions are given by

$$\frac{\partial}{\partial x}\mathsf{softmax}(x) = \frac{\partial}{\partial x}\mathsf{stable\text{-}softmax}(x) = \mathsf{Diag}(\mathsf{softmax}(x)) - \mathsf{softmax}(x)^\top \cdot \mathsf{softmax}(x), \tag{26}$$

$$\frac{\partial}{\partial x}\mathsf{log\text{-}softmax}(x) = \frac{\partial}{\partial x}\mathsf{stable\text{-}log\text{-}softmax}(x) = \mathbb{I}_D - 1_{\mathrm{D}} \cdot \mathsf{softmax}(x) \tag{27}$$

5.2 Derivations

log-softmax

Let $y(x) = \mathsf{softmax}(x)$, then we have

$$\begin{aligned} \frac{\partial}{\partial x}\log\left(\mathsf{softmax}(x)\right) &\overset{(9)}{=} \frac{\partial}{\partial y}\log(y)\frac{\partial}{\partial x}\mathsf{softmax}(x) \\ &\overset{(??)}{=} \mathsf{Diag}(\frac{1}{y}) \cdot \left(\mathsf{Diag}(y) - y^\top y\right) \\ &= \mathbb{I}_D - \mathsf{Diag}(y)y^\top y \\ &= \mathbb{I}_D - 1_{\mathrm{D}} \cdot y \\ &= \mathbb{I}_D - 1_{\mathrm{D}} \cdot \mathsf{softmax}(x). \end{aligned} \tag{28}$$

6 Loss functions

In this section we give an overview of some loss functions and their gradients, both for single inputs and for batches. We use the following notations:

1. $y \in \mathbb{R}^{1\times \mathrm{K}}$ is a single output, where K is the output dimension.
2. $t \in \mathbb{R}^{1\times \mathrm{K}}$ is a target for a single output y.
3. $Z \in \mathbb{R}^{\mathrm{N}\times \mathrm{K}}$ is an output batch, where N is the number of examples in the batch.
4. $T \in \mathbb{R}^{\mathrm{N}\times \mathrm{K}}$ contains the targets for an output batch Y.

squared error loss

$$\begin{aligned}
\mathcal{L}_{\mathrm{SE}}(y,t) &= (y-t)(y-t)^\top \\
\nabla_y \mathcal{L}_{\mathrm{SE}}(y,t) &= 2(y-t) \\
\mathcal{L}_{\mathrm{SE}}(Y,T) &= 1_{\mathrm{N}}^\top \cdot ((Y-T) \odot (Y-T)) \cdot 1_{\mathrm{K}} \\
\nabla_Y \mathcal{L}_{\mathrm{SE}}(Y,T) &= 2(Y-T)
\end{aligned} \tag{29}$$

The mean squared error loss is a scaled version of the squared error loss.

$$\mathcal{L}_{\mathrm{MSE}}(Y,T) = \frac{\mathcal{L}_{\mathrm{SE}}(Y,T)}{\mathrm{K} \cdot \mathrm{N}}$$

cross-entropy loss

$$\begin{aligned}
\mathcal{L}_{\mathrm{CE}}(y,t) &= -t \cdot \log(y)^\top \\
\nabla_y \mathcal{L}_{\mathrm{CE}}(y,t) &= -t \odot \frac{1}{y} \\
\mathcal{L}_{\mathrm{CE}}(Y,T) &= -1_{\mathrm{N}}^\top \cdot (T \odot \log(Y)) \cdot 1_{\mathrm{K}} \\
\nabla_Y \mathcal{L}_{\mathrm{CE}}(Y,T) &= -T \odot \frac{1}{Y}
\end{aligned} \tag{30}$$

softmax cross-entropy loss

$$\begin{aligned}
\mathcal{L}_{\mathrm{SCE}}(y,t) &= -t \cdot \mathsf{log\text{-}softmax}(y)^\top \\
\nabla_y \mathcal{L}_{\mathrm{SCE}}(y,t) &= t \cdot 1_{\mathrm{K}} \cdot \mathsf{softmax}(y) - t \\
\mathcal{L}_{\mathrm{SCE}}(Y,T) &= -1_{\mathrm{N}}^\top \cdot (T \odot \mathsf{log\text{-}softmax}(Y)) \cdot 1_{\mathrm{K}} \\
\nabla_Y \mathcal{L}_{\mathrm{SCE}}(Y,T) &= \mathsf{softmax}(Y) \odot (T \cdot 1_{\mathrm{K}} \cdot 1_{\mathrm{K}}^\top) - T
\end{aligned} \tag{31}$$

Note that if t is a one-hot encoded target (e.g. in case of a classification problem), it is a vector consisting of one value 1 and all others values 0. In other words we have $t \cdot 1_{\mathrm{K}} = 1$, hence in this case the gradients can be simplified to

$$\begin{aligned}
\nabla_y \mathcal{L}_{\mathrm{SCE}}(y,t) &= \mathsf{softmax}(y) - t \\
\nabla_Y \mathcal{L}_{\mathrm{SCE}}(Y,T) &= \mathsf{softmax}(Y) - T
\end{aligned}$$

logistic cross-entropy loss

$$\begin{aligned}
\mathcal{L}_{\mathrm{LCE}}(y,t) &= -t \cdot \log(\sigma(y))^\top \\
\nabla_y \mathcal{L}_{\mathrm{LCE}}(y,t) &= t \odot \sigma(y) - t \\
\mathcal{L}_{\mathrm{LCE}}(Y,T) &= -1_{\mathrm{N}}^\top \cdot (T \odot (\log(\sigma(Y))) \cdot 1_{\mathrm{K}} \\
\nabla_Y \mathcal{L}_{\mathrm{LCE}}(Y,T) &= T \odot \sigma(Y) - T
\end{aligned} \tag{32}$$

negative log-likelihood loss

$$\begin{aligned}
\mathcal{L}_{\mathrm{NLL}}(y,t) &= -\log(y \cdot t^\top) \\
\nabla_y \mathcal{L}_{\mathrm{NLL}}(y,t) &= -\frac{1}{y \cdot t^\top} \cdot t \\
\mathcal{L}_{\mathrm{NLL}}(Y,T) &= -1_{\mathrm{N}}^\top \cdot (\log((Y \odot T) \cdot 1_{\mathrm{K}})) \cdot 1_{\mathrm{K}} \\
\nabla_Y \mathcal{L}_{\mathrm{NLL}}(Y,T) &= -\left(\frac{1}{(Y \odot T) \cdot 1_{\mathrm{K}}} \cdot 1_{\mathrm{K}}^\top\right) \odot T
\end{aligned} \tag{33}$$

6.1 Derivations

cross-entropy loss

$$\nabla_y \mathcal{L}_{\mathrm{CE}}(y,t) = -\frac{\partial}{\partial y}(t \cdot \log(y)^\top) \overset{(7)}{=} -t \cdot \mathsf{Diag}(\frac{1}{y}) = -t \odot \frac{1}{y} \tag{34}$$

softmax cross-entropy loss

$$\begin{aligned}
\nabla_y \mathcal{L}_{\mathrm{SCE}}(y,t) = -\frac{\partial}{\partial y}(t \cdot \mathsf{log\text{-}softmax}(y)^\top) &\overset{(7)}{=} -t \cdot \frac{\partial}{\partial y} \mathsf{log\text{-}softmax}(y) \\
&\overset{(28)}{=} -t \cdot (\mathbb{I}_K - 1_{\mathrm{K}} \cdot \mathsf{softmax}(y)) = t \cdot 1_{\mathrm{K}} \cdot \mathsf{softmax}(y) - t
\end{aligned} \tag{35}$$

If the target t is a one-hot encoded vector, we have $t \cdot 1_{\mathrm{K}} = 1$. In that case the gradient simplifies to

$$\nabla_y \mathcal{L}_{\text{SCE-one-hot}}(y,t) = \mathsf{softmax(y)} - t.$$

Using property (15) we can generalize equation (35) to

$$\nabla_Y \mathcal{L}_{\mathrm{SCE}}(Y,T) = \mathsf{softmax(Y)} \odot (T \cdot 1_{\mathrm{K}} \cdot 1_{\mathrm{K}}^\top) - T.$$

logistic cross-entropy loss

$$\begin{aligned}
\nabla_y \mathcal{L}_{\mathrm{LCE}}(y,t) = \frac{\partial}{\partial y}(-t \cdot \log(\sigma(y))^\top) &\overset{(7)}{=} -t \cdot \frac{\partial}{\partial y} \log(\sigma(y)) \\
&= -t \cdot \mathsf{Diag}(1_{\mathrm{K}}^\top - \sigma(y)) = t \odot \sigma(y) - t,
\end{aligned}$$

where we used the well known fact that in the univariate case $\frac{d}{dt}\log(\sigma(t)) = 1 - \log(\sigma(t))$.

7 Weight initialization

The initial values of the weights in linear layers must be carefully chosen, as they may have a large impact on the performance of a neural network [NBS22]. Typically, these values are randomly generated on the basis of specific probability distributions. In this section, we give a few commonly used distributions.

NAME	DISTRIBUTION
Uniform	$U(a, b)$
Xavier [GB10]	$U(-\frac{1}{\sqrt{\mathrm{D}}}, \frac{1}{\sqrt{\mathrm{D}}})$
Normalized Xavier [GB10]	$U(-\frac{\sqrt{6}}{\sqrt{\mathrm{D}+\mathrm{K}}}, \frac{\sqrt{6}}{\sqrt{\mathrm{D}+\mathrm{K}}})$,
He [HZRS15]	$\mathcal{N}(0, \sqrt{\frac{2}{\mathrm{D}}})$,

where D is the number of inputs and K the number of outputs of the layer to which the weight matrix belongs. Furthermore, $U(a, b)$ is the uniform distribution in a given interval $[a, b]$, and $\mathcal{N}(\mu, \sigma)$ is the normal distribution with mean μ and standard deviation σ.

Unlike weights, the initial values of bias vectors are typically set to a small constant or zero rather than drawn from probability distributions. However, the Xavier weight distribution can also be used.

8 Optimization

In the optimization step, the parameters θ of a layer are updated based on the value of their gradient $\mathsf{D}\theta$ with respect to a given loss function. The goal of this step is to decrease the value of the loss. In this section, we give three common choices for optimization methods: gradient descent, momentum, and Nesterov. All take a learning rate parameter η as input, which is used to control the size of the optimization step. Our equations are equivalent to the ones in Keras [C+15], but presented in matrix form. We use a prime symbol to denote updated values.

GRADIENT DESCENT	MOMENTUM	NESTEROV
$\theta' = \theta - \eta \cdot \mathsf{D}\theta$	$\Delta'_\theta = \mu \cdot \Delta_\theta - \eta \cdot \mathsf{D}\theta$	$\Delta'_\theta = \mu \cdot \Delta_\theta - \eta \cdot \mathsf{D}\theta$
	$\theta' = \theta + \Delta'_\theta$	$\theta' = \theta + \mu \cdot \Delta'_\theta - \eta \cdot \mathsf{D}\theta$

Both momentum and Nesterov depend on a parameter $0 < \mu < 1$. Furthermore, they store an additional parameter Δ_θ with the same dimensions as θ. This parameter is updated in each optimization call and initially contains only zeros.

9 Learning Rate Schedulers

We define a learning rate scheduler as a function $\eta : \mathbb{N} \to \mathbb{R}$ that returns the learning rate at optimization step i. We assume that η_0 is a given initial learning rate.

FORMULA	DESCRIPTION
$\eta_i = \eta_0$	A constant scheduler with initial value η_0.
$\eta_{i+1} = \frac{\eta_i}{1 + d \cdot i}$	A time-based scheduler with decay parameter d.
$\eta_i = \eta_0 \cdot d^{\lfloor \frac{1+i}{r} \rfloor}$	A step-based scheduler with change rate d and drop rate r.
$\eta_i = \eta_0 e^{-d \cdot i}$	An exponential scheduler with decay parameter d.
$\eta_i = \eta_0 \Gamma^{\sum_{j=1}^{k} \lfloor \frac{i}{m_j} \rfloor}$	A multi-step scheduler with decay parameter Γ and milestones $\{m_1, \dots m_k\}$.

References

[C+15] François Chollet et al. Keras. `https://keras.io`, 2015.

[CMP21] Selima Curci, Decebal Constantin Mocanu, and Mykola Pechenizkiy. Truly sparse neural networks at scale. *CoRR*, abs/2102.01732, 2021.

[Fuk75] K. Fukushima. Cognitron: a self-organizing multilayered neural network. *Biological Cybernetics*, 20:121–136, 1975.

[GB10] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In Yee Whye Teh and Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR.

[HDY+12] Geoffrey Hinton, Li Deng, Dong Yu, George E. Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N. Sainath, and Brian Kingsbury. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97, 2012.

[HZRS15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1026–1034, 2015.

[JXF+16] Xiaojie Jin, Chunyan Xu, Jiashi Feng, Yunchao Wei, Junjun Xiong, and Shuicheng Yan. Deep learning with s-shaped rectified linear activation units. In Dale Schuurmans and Michael P. Wellman, editors, *AAAI*, pages 1737–1743. AAAI Press, 2016.

[Maa13] Andrew L. Maas. Rectifier nonlinearities improve neural network acoustic models. 2013.

[NBS22] Meenal V. Narkhede, Prashant P. Bartakke, and Mukul S. Sutaone. A review on weight initialization strategies for neural networks. *Artif. Intell. Rev.*, 55(1):291–322, 2022.

[Yeh17] Chris Yeh. Deriving Batch-Norm Backprop Equations. `https://chrisyeh96.github.io/2017/08/28/deriving-batchnorm-backprop.html`, 2017.