# Three layers AutoEncoder for piece-wise constant functions

January 1, 2022

## 1 Introduction

Let's recall that, for now, our main goal is to build the simplest autoencoder that can reconstruct step functions while having a latent code that is linear with respect to the characteristics of a step function such as the position of the transition and the non-zero amplitude. In other words, we wish to be able to, after training the autoencoder, move the transition of the step function linearly using the latent code corresponding to the position for example. Moreover, we want to mainly use the ReLU as an activation function which limits the set of possible architectures.

Although this task seems relatively easy compared to famous deep learning applications, our goal is not to empirically achieve high performance and accuracy leading to a so called 'black box' network. Instead, we want to address the problems that may occur by understanding the real cause behind each one of them and trying to find a solution that can be generalized for more complex data such as images.

In the previous section, we stated off with one amplitude caring only about linearly encoding the position of the transition. We proved that a linear encoding can be achieved using a single layered encoder without any activation i.e. by doing a simple linear projection. However, we also proved that reconstructing the step function with a scalar latent code as input while having a ReLU activation function on the output is not feasible with only one layer in the decoder. For that, we will explore increasing the architecture of the decoder in what follows.

## 2 Notations

Before starting, let's recall the notations.

- Scalars are written in normal font, vectors in bold and matrices are denoted by a capital letter.
- $\mathbf{x} \in \mathbb{R}^n$ is an input sample to the autoencoder. It represented a step function. In total, with a vector of dimension $n$ we can construct $n + 1$ possible step functions if we only vary the position of the transition while keeping the non-zero amplitude fixed.
- The autoencoder's output is denoted by $\mathbf{y}$. It is a vector of dimension $n$ as well. Ideally, after training, $\mathbf{y}$ should be exactly equal to $\mathbf{x}$ that is on the input.
- $\mathbf{z}$ denotes the latent code.
- The letter $i \in 1, 2, ..., n + 1$ is used to indicate a particular sample. The letter $j \in 1, 2, ..., n$ is used to indicate an entry of a vector of dimension $n$. For example $x_j^{(i)}$ represents the entry $j$ of the input sample $i$.
- Weights of a layer $l$ are represented by $W^{[l]}$. And biases are represented by $\mathbf{b}^{[l]}$.
- We denote a preactivation vector of a layer $l$ corresponding to a sample $i$ by $\mathbf{u}^{[l](i)}$. And we denote the corresponding activation vector by $\mathbf{a}^{[l](i)}$.
- An activation function or non-linearity of a layer $l$ is denoted by $g^{[l]}$. We'll have for example, $\mathbf{a}^{[l](i)} = g^{[l]}(\mathbf{u}^{[l](i)})$.

## 3 Two Layers Decoder

Having proved that a one layer decoder with a ReLU activation function can't correctly reconstruct all step functions when fed a scalar $z$. We will study whether by adding a hidden layer to the decoder, it will be able to reconstruct the step functions. The study will concern the number of neurons required in the added hidden layer and the necessity of a bias.

### 3.1 Architecture

Since we proved that a one layer encoder can encode a linear representation, we will only focus on the decoder and its ability to reconstruct all step functions each coded by a scalar $z^{(i)}$. We can suppose $z^{(i)}$ to be linear with the position of the transition of the step function $\mathbf{x}^{(i)}$, but as we'll see it won't be a necessary hypothesis, rather it is a particular solution.

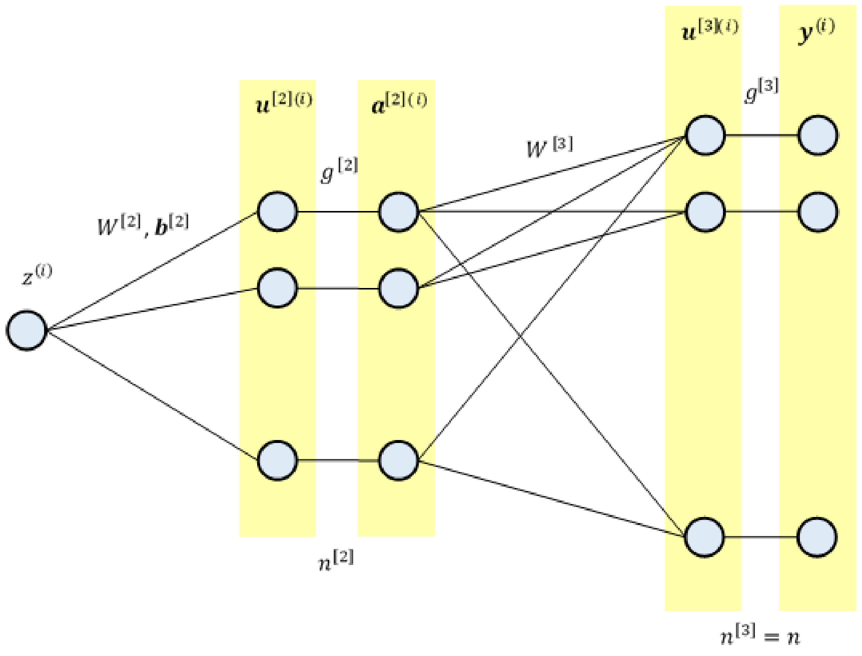

Figure 1: This is the decoder part of the autoencoder. The decoder is fed a latent code $z^{(i)}$ which is a scalar and outputs a vector $\mathbf{y}^{(i)}$ of dimension $n$. We added a hidden layer with $n^{[2]}$ neurons that takes the latent code as input and outputs the activation vector $\mathbf{a}^{[2](i)}$.

The decoder in fig.1 can be summarized in the following equations.

- Layer 2:
  - Input: $\underset{(1\times 1)}{z^{(i)}}$
  - Output: $\underset{(1\times n^{[2]})}{\mathbf{a}^{[2](i)}}$
  - Weight: $\underset{(n^{[2]}\times 1)}{W^{[2]}}$
  - Bias: $\underset{(1\times n^{[2]})}{\mathbf{b}^{[2]}}$
  - Equations:

$$\mathbf{u}^{[2](i)} = z^{(i)} {W^{[2]}}^T + \mathbf{b}^{[2]} \quad (1)$$
$$\mathbf{a}^{[2](i)} = g^{[2]}(\mathbf{u}^{[2](i)}) \quad (2)$$

- Layer 3:
  - Input: $\underset{(1\times n^{[2]})}{\mathbf{a}^{[2](i)}}$
  - Output: $\underset{(1\times n)}{\mathbf{y}^{(i)}}$
  - Weight: $\underset{(n\times 1)}{W^{[3]}}$
  - Bias: None
  - Equations:

$$\mathbf{u}^{[3](i)} = \mathbf{a}^{[2](i)} {W^{[3]}}^T \quad (3)$$
$$\mathbf{y}^{(i)} = g^{[3]}(\mathbf{u}^{[2](i)}) \quad (4)$$

The first question to be asked: How many neurons should have the added hidden layer? In other words what should $n^{[2]}$ be equal to?

Let's consider the output matrix where each row is a reconstruction $\mathbf{y}^{(i)}$. In case of a correct reconstruction, $Y$ will have the following form.

$$\underset{(n+1)\times n}{Y} = \begin{bmatrix} \dots & \mathbf{y}^{(1)} & \dots \\ \dots & \mathbf{y}^{(2)} & \dots \\ \dots & \mathbf{y}^{(3)} & \dots \\ & \vdots & \\ \dots & \mathbf{y}^{(n)} & \dots \\ \dots & \mathbf{y}^{(n+1)} & \dots \end{bmatrix} = \begin{bmatrix} 1 & 1 & \dots & 1 & 1 \\ 0 & 1 & \dots & 1 & 1 \\ 0 & 0 & \dots & 1 & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & 0 & 1 \\ 0 & 0 & \dots & 0 & 0 \end{bmatrix}$$

Let $U^{[3]}$ be the matrix of preactivations of the third layer where each row is a preactivation

$$\mathbf{u}^{[3](i)} = ReLU(\mathbf{y}^{(i)}) = max(0, \mathbf{y}^{(i)})$$

In order to obtain $Y$ from $U^{[3]}$, the nature of the nonlinearity ReLU imposes a necessary condition on the latter matrix. And $U^{[3]}$ should necessarly have the following form

$$\underset{(n+1)\times n}{U^{[3]}} = \begin{bmatrix} \dots & \mathbf{u}^{[3](1)} & \dots \\ \dots & \mathbf{u}^{[3](2)} & \dots \\ \dots & \mathbf{u}^{[3](3)} & \dots \\ & \vdots & \\ \dots & \mathbf{u}^{[3](n)} & \dots \\ \dots & \mathbf{u}^{[3](n+1)} & \dots \end{bmatrix} = \begin{bmatrix} 1 & 1 & \dots & 1 & 1 \\ u_1^{[3](2)} & 1 & \dots & 1 & 1 \\ u_1^{[3](3)} & u_2^{[3](3)} & \dots & 1 & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ u_1^{[3](n)} & u_2^{[3](n)} & \dots & u_{n-1}^{[3](n)} & 1 \\ u_1^{[3](n+1)} & u_2^{[3](n+1)} & \dots & u_{n-1}^{[3](n+1)} & u_n^{[3](n+1)} \end{bmatrix}$$

Where, $u_j^{[3](i)} \leq 0 \ \forall i, j$ such that $i = 2, ..., n+1$ and $j = 1, ..., i-1$.

Using lemma 1, we obtain the first condition: $rank(U^{[3]}) = n$. Having found the condition on $U^{[3]}$, we will use the above equations to extract the effect of this condition on the previous layer and hence deduce the the required $n^{[2]}$.

Indeed, let $A^{[2]}$ be a matrix of dimension $(n+1) \times n^{[2]}$ whose rows are the output of the second layer $\mathbf{a}^{[2](i)}$.

$$\underset{(n+1)\times n^{[2]}}{A^{[2]}} = \begin{bmatrix} \dots & \mathbf{a}^{[2](1)} & \dots \\ \dots & \mathbf{a}^{[2](2)} & \dots \\ \dots & \mathbf{a}^{[2](3)} & \dots \\ & \vdots & \\ \dots & \mathbf{a}^{[2](n)} & \dots \\ \dots & \mathbf{a}^{[2](n+1)} & \dots \end{bmatrix}$$

We have that

$$U^{[3]} = A^{[2]} {W^{[3]}}^T$$

And using the rank of a matrix product, we obtain

$$rank(U^{[3]}) \leq \min\left(rank(\underset{(n+1)\times n^{[2]}}{A^{[2]}}), rank(\underset{n\times n^{[2]}}{W^{[3]}})\right)$$

Thus, two more conditions arise. In order to achieve a full row rank matrix $U^{[3]}$, we first need $n^{[2]}$ to be equal to $n$. And second, we need $A^{[2]}$ and $W^{[3]}$ to have full rank.

The first condition is satisfied by fixing the number of the added hidden layer to $n$. Next, the weights $W^{[3]}$ are learnt. Thus, we will suppose that having all other conditions satisfied, the autoencoder will manage to learn a full rank $W^{[3]}$. But what about the rank of $A^{[2]}$? Having a dimension of $(n+1) \times n$, can $rank(A^{[2]}) = n$?

Let's recall how $A^{[2]}$ is computed.

$$A^{[2]} = \begin{bmatrix} \dots & \mathbf{a}^{[2](1)} & \dots \\ \dots & \mathbf{a}^{[2](2)} & \dots \\ & \vdots & \\ \dots & \mathbf{a}^{[2](n+1)} & \dots \end{bmatrix} = \begin{bmatrix} \dots & \mathbf{g}^{[2]}(z^{(1)}{W^{[2]}}^T + \mathbf{b}^{[2]}) & \dots \\ \dots & \mathbf{g}^{[2]}(z^{(2)}{W^{[2]}}^T + \mathbf{b}^{[2]}) & \dots \\ & \vdots & \\ \dots & \mathbf{g}^{[2]}(z^{(3)}{W^{[2]}}^T + \mathbf{b}^{[2]}) & \dots \end{bmatrix} \tag{5}$$

Looking at (5), we deduce another two conditions to have a full rank matrix. On one hand, the function $g^{[2]}$ should necessary be a non linearity. Otherwise, we'll have

$$A^{[2]} = \begin{bmatrix} z^{(1)} \\ z^{(2)} \\ \vdots \\ z^{(3)} \end{bmatrix} {W^{[2]}}^T + \begin{bmatrix} \dots & \mathbf{b}^{[2]} & \dots \\ \dots & \mathbf{b}^{[2]} & \dots \\ & \vdots & \\ \dots & \mathbf{b}^{[2]} & \dots \end{bmatrix}$$

Where the two terms are obviously rank 1 matrices. Using the rank of a matrix sum,

$$\begin{aligned} rank(A^{[2]}) &\leq rank\left( \begin{bmatrix} z^{(1)} \\ z^{(2)} \\ \vdots \\ z^{(3)} \end{bmatrix} {W^{[2]}}^T \right) + rank\left( \begin{bmatrix} \dots & \mathbf{b}^{[2]} & \dots \\ \dots & \mathbf{b}^{[2]} & \dots \\ & \vdots & \\ \dots & \mathbf{b}^{[2]} & \dots \end{bmatrix} \right) \\ rank(A^{[2]}) &\leq 2 \end{aligned}$$

On the other hand, since we favour a ReLU activation function, a bias is also crucial in order to achieve a full rank matrix. Indeed, having $g^{[2]} = ReLU$, without a bias, we'll have

$$g^{[2]}(z^{(i)}W^{[2]}) = \begin{cases} |z^{(i)}|g^{[2]}(W^{[2]}) & \text{if } z^{(i)} \geq 0 \\ |z^{(i)}|g^{[2]}(-W^{[2]}) & \text{if } z^{(i)} < 0 \end{cases} \tag{6}$$

Thus, without a bias and with a ReLU activation function, the matrix $A^{[2]}$ will have at most a rank equal to 2. But with a bias we can already think about one solution that can make $A^{[2]}$ have a full rank. This naive solution can be as follows:

- $z^{(i)} = i$
- $W^{[2]} = \begin{bmatrix}1, 1, \dots, 1\end{bmatrix}^T$
- $\mathbf{b}^{[2]} = \begin{bmatrix}0, -1, -2, \dots, -n+1\end{bmatrix}$

Thus, computing the full activation matrix,

$$\underset{(n+1)\times n}{A^{[2]}} = \begin{bmatrix} 1 & 0 & 0 & \dots & 0 \\ 2 & 1 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ n & n-1 & n-2 & \dots & 1 \\ n+1 & n & n-2 & \dots & 2 \end{bmatrix}$$

To conclude on the architecture part, in order to be able to fully reconstruct the step functions represented by a scalar latent code, the decoder should have two layers, both of dimension $n$ equal to the dimension of the input vector. Both layers have ReLU as activation functions. A bias is necessary in the second layer, but it is not in the output layer. Moreover, we didn't encounter nowhere the necessity of having a linear encoding, thus for now, we will suppose that it is a particular solution. In the following section we will put this supposition to test.

## 3.2 Decoder Training and Problems

In this section, we will test the above theoretical result by fixing the encoder to be linear and training only the decoder.

## 3.3 Understanding the solutions

This section is devoted to understand the possible set of solutions. For that we will look closely at each equation in the decoder and we'll try to break it down to make it more comprehensible. The goal behind this is to obtain insights that might help us improve the optimization.

Note: In what follows, I have used the term "vector" to indicated both vectors and points in the space of dimension d. This is not mathematically correct and needs to be fixed. I'll fix it later.

### 3.3.1 Layer 3

We tend to always start looking at the last layer because its outputs are the reconstructed step functions which we are sure about how we want them to be. Let's rewrite the equations (3) and (4) of the last layer in a different way.

The preactivation vector $\mathbf{u}^{[3](i)}$ can be written as

$$\mathbf{u}^{[3](i)} = \mathbf{a}^{[2](i)} W^{[3]^T} = \left[\mathbf{a}^{[2](i)}\mathbf{w}_1^{[3]}, \mathbf{a}^{[2](i)}\mathbf{w}_2^{[3]}, \dots, \mathbf{a}^{[2](i)}\mathbf{w}_n^{[3]}\right] \tag{7}$$

Where the set of vectors $\{\mathbf{w}_j^{[3]}\}_{(j=1,\dots,n)}$ constitute the rows of the matrix $W^{[3]}$.

The notation in (7) is of interest for us because it shows the dot products $\mathbf{a}^{[2](i)}\mathbf{w}_j^{[3]}$. Which can be understood as the orthogonal projection of the vector $\mathbf{a}^{[2](i)}$ onto the vectors $\mathbf{w}_j^{[3]}$. Hence, we can imagine $\mathbf{u}^{[3](i)}$ as an orthogonal projection of $\mathbf{a}^{[2](i)}$ on a fixed basis $\{\mathbf{w}_j^{[3]}\}_{(j=1,\dots,n)}$.

Moreover, we have

$$\begin{aligned}\mathbf{y}^{(i)} = ReLU(\mathbf{u}^{[3](i)}) &= \left[ReLU(\mathbf{a}^{[2](i)}\mathbf{w}_1^{[3]}), ReLU(\mathbf{a}^{[2](i)}\mathbf{w}_2^{[3]}), \dots, ReLU(\mathbf{a}^{[2](i)}\mathbf{w}_n^{[3]})\right] \\ &= \left[y_1^{(i)}, y_2^{(i)}, \dots, y_n^{(i)}\right]\end{aligned} \tag{8}$$

Considering one entry $j$, when $y_j^{(i)} = 1$, we necessarily have $\mathbf{a}^{[2](i)}\mathbf{w}_j^{[3]} = 1$ because of the linear part of the ReLU. Thus, keeping in mind the orthogonal projection, $y_j^{(i)} = 1$ will require $\mathbf{a}^{[2](i)}$ to belong to the hyperplane that we'll name $H_j^1$. $H_j^1$ is orthogonal to $\mathbf{w}_j^{[3]}$ at distance form the origin equal to the inverse of the modulus. This is depicted in fig.2, in a simplified 2D space, that is, when $n = 2$.

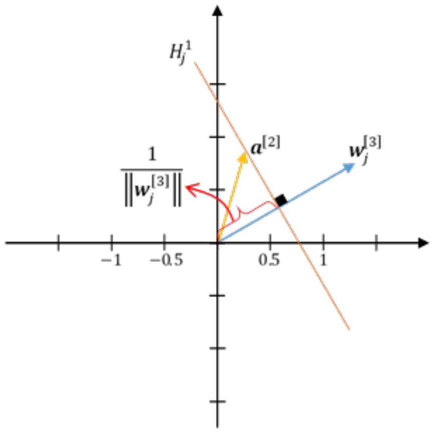

Figure 2: This is a representation of the hyperplane $H_j^1$ in a 2D space. $H_j^1$ must be at a distance from the orgin of the vector equal to $\frac{1}{||\mathbf{w}_j^{[3]}||}$. All vectors $\mathbf{a}^{[2]}$ belonging to $H_j^1$ will have an orthogonal projection onto $\mathbf{w}_j^{[3]}$ equal to 1.

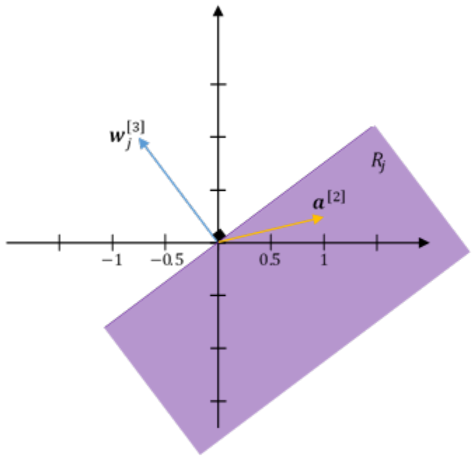

Figure 3: This is a representation of the region $R_j$ in a 2D space. All vectors $\mathbf{a}^{[2]}$ belonging to $R_j$ will have a negative orthogonal projection onto $\mathbf{w}_j^{[3]}$. $R_j$ takes half of the space, thus there's more freedom for $\mathbf{a}^{[2]}$.

We can deduce that for an entry $j$, $\mathbf{w}_j^{[3]}$ must be such that its hyperplane $H_j^1$ contain all points $\mathbf{a}^{[2](i)}$ that correspond to $y_j^{(i)} = 1$. For example, for $j = n$, we have $y^{(i)} = 1$ for $i = 1, ..., n$. This implies that for $i = 1, ..., n$, the $n$ activation $\mathbf{a}^{[2](i)}$ belong to $H_n^1$. And under the assumption that $A^{[2]}$ has full rank, when it is fixed, $H_n^1$ will have a unique solution.

Moreover, for a sample $i$ where $y_j^{(i)} = 1$ and $y_{j'}^{(i)} = 1$, the corresponding vector $\mathbf{a}^{[2](i)}$ must belong to the intersection of the two hyperplanes $H_j^1$ and $H_{j'}^1$. See fig.4. Thus, for $i = 0$, where $y_j^{(i)} = 1$ for $j = 1, ..., n$, the vector $\mathbf{a}^{[2](0)}$ belong to the intersection of $n$ hyperplanes $\{H_j^1\}_{j=1,...,n}$ in a space of dimension $n$. That is, $\mathbf{a}^{[2](0)}$ has a unique position in the space when the matrix $W^{[3]}$ becomes fixed.

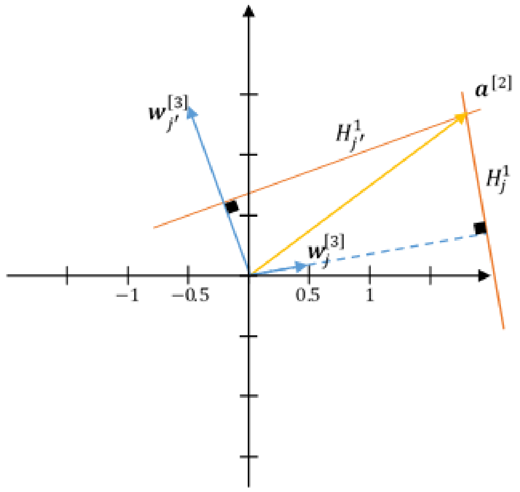

Figure 4: For the simple 2D case, the hyperplanes $H_j^1$ and $H_{j'}^1$ are straight lines. Thus the intersection is one point.

Again, let's consider one entry $j$, but when $y_j^{(i)} = 0$. Then, $\mathbf{a}^{[2](i)}\mathbf{w}_j^{[3]}$ must be negative. This implies that $\mathbf{a}^{[2](i)}$ lives in the half-space $R_j$ orthogonal to and opposite to the direction of the vector $\mathbf{w}_j^{[3]}$. We visualized the 2D case in fig.3.

Similarly to the previous case, $\mathbf{w}_j^{[3]}$ must be learnt such that $R_j$ contains all vectors $\mathbf{a}^{[2](i)}$ corresponding to $y_j^{(i)} = 0$. And, when $y_j^{(i)} = 0$ for $j \in J \subseteq \{1, ..., n\}$, the corresponding vector $\mathbf{a}^{[2](i)}$ must lie in the intersection $\bigcap_{j \in J} R_j$. However, for this case, the vectors $\mathbf{a}^{[2](i)}$ and $\mathbf{w}_j^{[3]}$ have more freedom. Thus, there will not be a unique solution for $R_j$.

In addition, there exists an important constraint that we shouldn't forget. The vectors $\mathbf{a}^{[2](i)}$ are non-negative. Thus for a 2D space, all $\mathbf{a}^{[2](i)}$ live in the first quadrant. Which decreases furthermore the degrees of freedom and forces the $\mathbf{w}_j^{[3]}$ to have directions such that both their associated hyperplane $H_j^1$ and half-space $R_j$ have a non-empty intersection with the first quadrant.

### 3.3.2 Layer 2

Now that we understand better what is going on in the third layer and the connection between the weights $W^{[3]}$ and the activations $\mathbf{a}^{[2](i)}$, we'll move back to the previous layer and relate the above information to the weights $W^{[2]}$ through $\mathbf{a}^{[2](i)}$.

Looking at the equation (1), the only constraint we have is the affine dependence between the vectors $\{\mathbf{u}^{[3](i)}\}_{(i=1,...,n+1)}$. For 2D and 3D vectors, the affine dependency is depicted in fig.5. Indeed, since the only free variable is $z^{(i)}$, once $W^{[2]}$ and $b^{[2]}$ are fixed, the vectors $\mathbf{u}^{[3](i)}$ can only vary on a straight line whatever the dimension of the space was.

Because of the ReLU in (1), this affine dependency breaks between the vectors $\mathbf{a}^{[2](i)}$. But their positions are still affected by the direction of the straight line where the vectors $\mathbf{u}^{[3](i)}$ live. And what adds more constraints and complexity is wanting a linear encoding. In fact, when the encoder is linear, the vectors $\mathbf{u}^{[3](i)}$ will be equally spaced on the straight line where they live, as in fig.6. This limits the freedom of the activation vectors even after applying the ReLU.

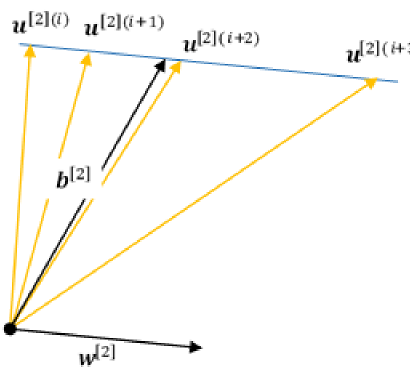

Figure 5: For all $i$, $\mathbf{u}^{[2](i)}$ lives on the blue line.

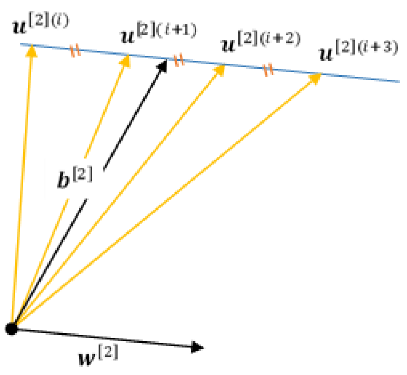

Figure 6: When $z^{(i)}$ are linear, the vectors will be equally spaced on the blue line.

### 3.3.3 Example of a solution for n=2

To make the previous notions more clear and to connect them together, we will take a very simple 2D example. Also, we will consider the case of a linear encoder.

Let the three possible output vectors, for $n = 2$ be:

- $\mathbf{y}^{(0)} = [1, 1]$
- $\mathbf{y}^{(1)} = [0, 1]$
- $\mathbf{y}^{(2)} = [0, 0]$

The solution is depicted in fig.7.

### 3.3.4 Key takeaways and Interpretations

# A Linear Algebra

**Lemma 1** *Let $M$ be a matrix of dimension $(n+1) \times n$ of the following form*

$$\underset{(n+1)\times n}{M} = \begin{bmatrix} 1 & 1 & \dots & 1 & 1 \\ m_{2,1} & 1 & \dots & 1 & 1 \\ m_{3,1} & m_{3,2} & \dots & 1 & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ m_{n,1} & m_{n,2} & \dots & m_{n,n-1} & 1 \\ m_{n+1,1} & m_{n+1,2} & \dots & m_{n+1,n-1} & m_{n+1,n} \end{bmatrix}$$

*Where $m_{i,j} \leq 0 \ \forall i,j$ such that $i = 2, ..., n+1$ and $j = 1, ..., i-1$. Then $M$ is a full row rank matrix.*

**Proof 1** *Let $\mathbf{m}_i$ be a row of the matrix $M$. The proof will consist in studying the linearly independence of the set of vectors $\mathbf{m}_i$ for $i = 1, ..., n$ (no need to consider the last null row).*

*Let's find a set of scalars $\{\alpha_i\}_{(i=1,...,n)}$ such that*

$$\sum_{i=1}^{n} \alpha_i \mathbf{m}_i = \mathbf{0} \tag{9}$$

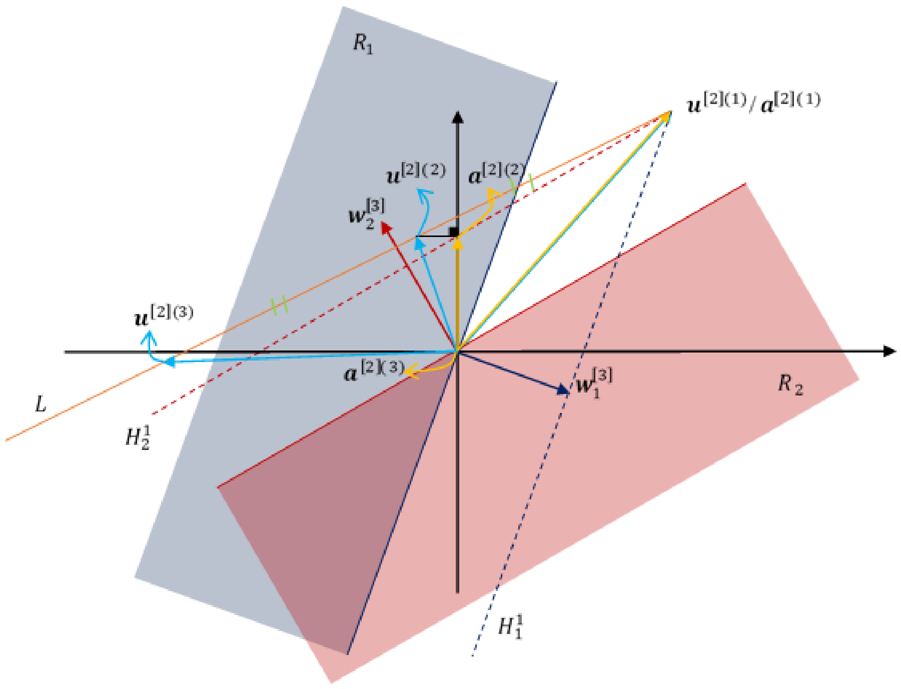
$R_1$
$\boldsymbol{u}^{[2](1)}/\boldsymbol{a}^{[2](1)}$
$\boldsymbol{u}^{[2](2)}$
$\boldsymbol{a}^{[2](2)}$
$\mathbf{w}_2^{[3]}$
$\boldsymbol{u}^{[2](3)}$
$\boldsymbol{a}^{[2](3)}$
$\mathbf{w}_1^{[3]}$
$R_2$
$L$
$H_2^1$
$H_1^1$

Figure 7: For the simple 2D case, the hyperplanes $H_j^1$ and $H_{j'}^1$ are straight lines. Thus the intersection is one point.

*All entries of the above combination should be null. Starting from entry $j = n$ and decreasing we'll have:*

- *for $j = n$:*

$$\sum_{i=1}^{n} \alpha_i m_{i,n} = \sum_{i=1}^{n} \alpha_i = 0 \tag{10}$$

- *for $j$ = n-1:*

$$\sum_{i=1}^{n} \alpha_i m_{i,n-1} = \sum_{i=1}^{n-1} \alpha_i + \alpha_n m_{n,n-1} = 0 \tag{11}$$

*Then using (10) in (11) will lead to,*

$$\alpha_n(m_{n,n-1} - 1) = 0$$

*Thus $\alpha_n = 0$ or $m_{n,n-1} = 1$. But $m_{n,n-1} \leq 0$, hence $\alpha_n = 0$. By recurrence, we can prove that $\forall i = 1, ..., n$ we'll have $\alpha_i = 0$.*

*This means that (9) can only be satisfied if all $\alpha_i = 0$. Hence, the set $\{\mathbf{m}_i\}_{(i=1,...,n)}$ is linearly independent and the matrix $M$ is a full row rank matrix.*

## B FCL

**Lemma 2** *Let's consider a layer $l$ with a weight matrix $W^{[l]}$, an input vector $\mathbf{a}^{[l-1]}$, an output vector $\mathbf{a}^{[l]}$ and a preactivation vector $\mathbf{u}^{[l]}$. The activation used is a ReLU. When $u_j^{[l]} < 0$, the weights $W_{j,:}^{[l]}$ (row $j$) are not updated during the optimization step. However, the weights $W^{[l-1]}$ are still going to be updated but only depending on the parameters of the layer $l$ that are not related to the entry $j$.*

**Proof 2** *Let $u_j^{[l]} < 0$. And let $MSE^{(i)} = \sum_{j=1}^{n}(y_j^{(i)} - x_j^{(i)})^2$. Using the chain rule, we have,*

$$\frac{\partial MSE^{(i)}}{\partial W_{j,k}^{[l]}} = \frac{\partial MSE^{(i)}}{\partial a_j^{[l]}} \frac{\partial a_j^{[l]}}{\partial u_j^{[l]}} \frac{\partial u_j^{[l]}}{\partial W_{j,k}^{[l]}} = 0 \tag{12}$$

*Because $a_j^{[l]} = ReLU(u_j^{[l]})$ and $u_j^{[l]}$ lives on the negative straight line where the ReLU is constant and null. In other words,*

$$\frac{\partial a_j^{[l]}}{\partial u_j^{[l]}} = 0 \tag{13}$$

*Thus the weights $W_{j,:}^{[l]}$ (row $j$) are not going to be updated.*
*Continuing with the derivations to the previous layer,*

$$\frac{\partial MSE^{(i)}}{\partial W_{r,s}^{[l-1]}} = \left(\sum_{k=1}^{n+1} \frac{\partial MSE^{(i)}}{\partial a_k^{[l](i)}} \frac{\partial a_k^{[l](i)}}{\partial u_k^{[l](i)}} \frac{\partial u_k^{[l](i)}}{\partial a_r^{[l-1](i)}}\right) \frac{\partial a_r^{[l-1](i)}}{\partial W_{r,s}^{[l-1]}} = \left(\sum_{k=1,k\neq j}^{n+1} \frac{\partial MSE^{(i)}}{\partial a_k^{[l](i)}} \frac{\partial a_k^{[l](i)}}{\partial a_r^{[l-1](i)}}\right) \frac{\partial a_r^{[l-1](i)}}{\partial W_{r,s}^{[l-1]}} \tag{14}$$

*The term inside the brackets will only depend on parameters not related to the entry $j$. Thus, negative preactivations will not prevent the update of the weights of previous layers during the back-propagation.*

## C Gradient computations

Let's compute the gradient of the MSE loss with respect to the parameters of our model: $\underset{n\times n}{\mathbf{W}^{[3]}}$, $\underset{n\times 1}{\mathbf{w}^{[2]}}$, $\underset{n\times 1}{\mathbf{b}^{[2]}}$, $\underset{1\times n}{\mathbf{w}^{[1]}}$ and $\underset{1\times 1}{b^{[1]}}$. We won't take into consideration the activations.

$$\begin{aligned} MSE &= \frac{1}{2}\sum_{i=1}^{n+1}\sum_{j=1}^{n}(y_j^{(i)} - x_j^{(i)})^2 & (15) \\ &= \frac{1}{2}\sum_{i=1}^{n+1}\sum_{j=1}^{n}(\sum_{k=1}^{n} W_{j,k}^{[3]}(w_k^{[2]}z^{(i)} + b_k^{[2]}) - x_j^{(i)})^2 & (16) \end{aligned}$$

<u>Layer 3</u>
The gradient of the MSE loss wrt to each entry $W_{u,v}^{[3]}$:

$$\frac{\partial MSE}{\partial W_{u,v}^{[3]}} = \sum_{i=1}^{n+1}(w_v^{[2]}z^{(i)} + b_v^{[2]})(y_u^{(i)} - x_u^{(i)}) \tag{17}$$

The gradient of the MSE loss wrt to $W^{[3]}$:

$$\frac{\partial MSE}{\partial W^{[3]}} = \sum_{i=1}^{n+1}(\mathbf{y}^{(i)} - \mathbf{x}^{(i)})(\mathbf{w}^{[2]}z^{(i)} + b^{[2]})^T \tag{18}$$

<u>Layer 2</u>
The gradient of the MSE loss wrt to each entry $w_r^{[2]}$:

$$\frac{\partial MSE}{\partial w_r^{[2]}} = \sum_{i=1}^{n+1}\sum_{j=1}^{n} z^{(i)}W_{r,j}^{[3]}(y_j^{(i)} - x_j^{(i)}) \tag{19}$$

The gradient of the MSE loss wrt to $\mathbf{w}^{[2]}$:

$$\frac{\partial MSE}{\partial \mathbf{w}^{[2]}} = \sum_{i=1}^{n+1} z^{(i)}W^{[3]}(\mathbf{y}^{(i)} - \mathbf{x}^{(i)}) \tag{20}$$

The gradient of the MSE loss wrt to each entry $b_r^{[2]}$:

$$\frac{\partial MSE}{\partial b_r^{[2]}} = \sum_{i=1}^{n+1} \sum_{j=1}^{n} W_{r,j}^{[3]} (y_j^{(i)} - x_j^{(i)}) \tag{21}$$

The gradient of the MSE loss wrt to $\mathbf{b}^{[2]}$:

$$\frac{\partial MSE}{\partial \mathbf{b}^{[2]}} = \sum_{i=1}^{n+1} W^{[3]} (\mathbf{y}^{(i)} - \mathbf{x}^{(i)}) \tag{22}$$

Layer 1

The gradient of the MSE loss wrt to each entry $w_s^{[1]}$:

$$\frac{\partial MSE}{\partial w_s^{[1]}} = \sum_{i=1}^{n+1} \sum_{j=1}^{n} \sum_{k=1}^{n} W_{j,k}^{[3]} w_k^{[2]} \frac{\partial z^{(i)}}{\partial w_s^{[1]}} (y_j^{(i)} - x_j^{(i)}) \tag{23}$$

$$= \sum_{i=1}^{n+1} \sum_{j=1}^{n} \sum_{k=1}^{n} W_{j,k}^{[3]} w_k^{[2]} x_s^{(i)} (y_j^{(i)} - x_j^{(i)}) \tag{24}$$

The gradient of the MSE loss wrt to $\mathbf{w}^{[1]}$:

$$\frac{\partial MSE}{\partial \mathbf{w}^{[1]}} = \sum_{i=1}^{n+1} \mathbf{x}^{(i)} * (\mathbf{y}^{(i)} - \mathbf{x}^{(i)})^T W^{[3]} \mathbf{w}^{[2]} \tag{25}$$