Autoencoder study for piecewise constant functions

December 17, 2021

1 Encoder Linearity

Till now, we proved that for the architecture in fig. 1, there exists a solution for which the latent code is linear with respect to the position of the step function. We proved it by fixing the weights of the encoder to output a linear latent code and making the autoencoder learn the rest of the parameters by training. We obtained satisfiable results with a low MSE. However, the training was very slow (took several hours and we're still working on a small 1D signal). And we noticed that the speed of the training was decreasing enormously with epochs.

To increase the speed of the training, we tried removing all constraints. Indeed, we obtained a similar MSE to the one with a linear encoder but within no more than one hour. However, this time the latent representation was not linear.

We will devote this section to first understand why the autoencoder tends to have a non linear representation. Second, we will try to find a solution that makes the encoder linear while keeping the training fast.

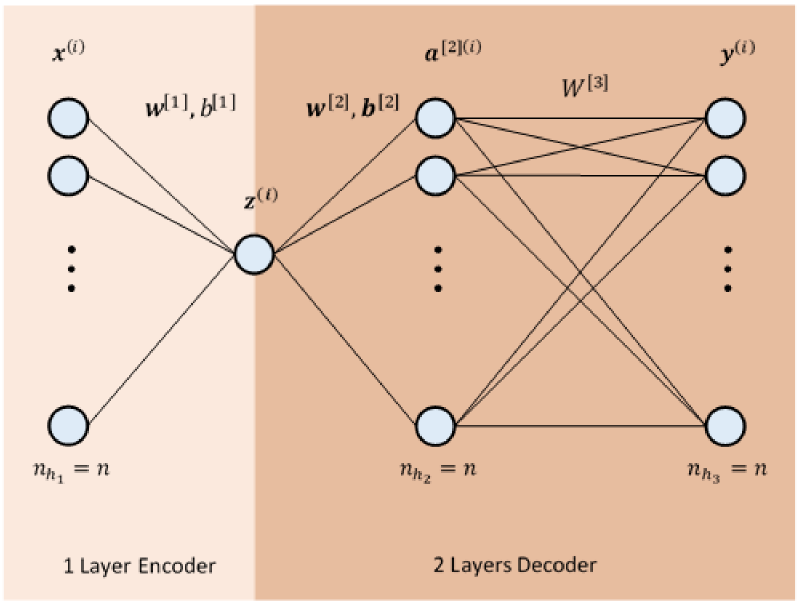

Figure 1: This three layers autoencoder architecture allows the reconstruction of all n possible step functions. There are no activations in the encoder. And ReLU are used for the decoder part.

1.1 Training with no constraints

The autoencoder is trained with no constraints on any weights. As before, the training set contains all the $n = 129$ step functions with an amplitude of 5. And we used an MSE loss.

The weights are initialized with close to zero random values using a uniform distribution. The optimization is done using RMSProps with a batch size equal to n. We fixed the learning rate to 10^{-4}

and decreased it to 10^{-5} if necessary during the tarining. The autoencoder is trained over 500000 epochs. You can check the notebook here.

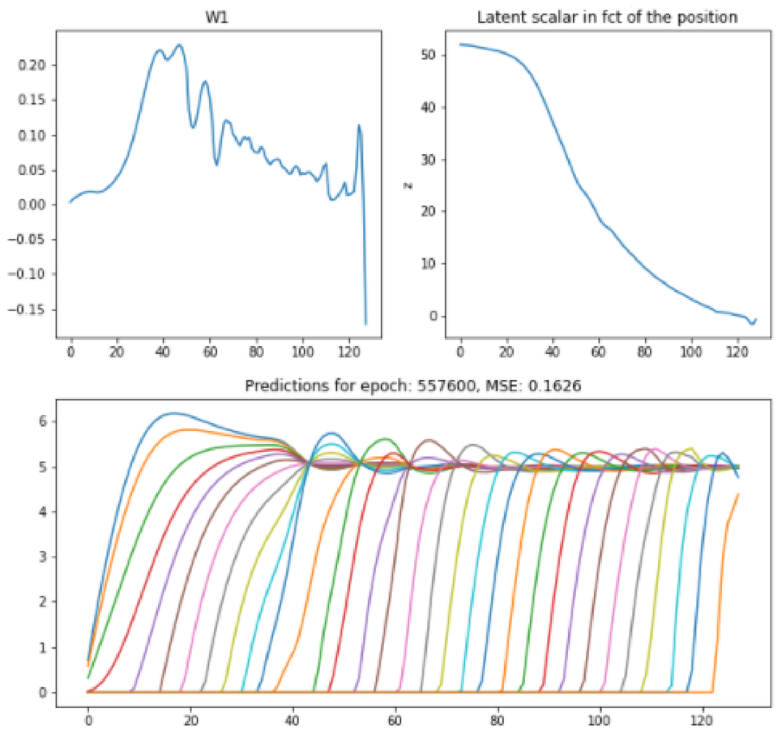

Figure 2: Results after training the autoencoder with no constraints over 557600 epochs. The top left plot $W1$ represents the weight vector $\mathbf{w}^{[1]}$ of the first layer. The top right plot is the plot of the latent code wrt the position of a step function. The last plot is the reconstructed step functions made by the autoencoder after the training. These plots are drawn in real-time while the autoencoder is training. You can check it here.

In fig. 2, we can observe the following problems:

1. Although the training time is reduced, it is still relatively slow. It took 557600 epochs and still did not manage to perfectly reconstruct all step functions.
2. It seems as the autoencoder favours the step functions with bigger positions. We can clearly see the degradation of the performance moving from right to left.
3. The latent code is not linear wtr to the positions.

There might be numerous explanations for these results. However, we found few of them and we are going to mention them next.

1.2 Understanding the results

Lemma 1 *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 1 *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,*

$$\frac{\partial MSE^{(i)}}{\partial W_{j,k}^{[l]}} = \frac{\partial MSE^{(i)}}{\partial a_j^{[l]}} \frac{\partial a_j^{[l]}}{\partial u_j^{[l]}} \frac{\partial u_j^{[l]}}{\partial W_{j,k}^{[l]}} = 0 \tag{1}$$

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

$$\frac{\partial a_j^{[l]}}{\partial u_j^{[l]}} = 0 \tag{2}$$

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

Continuing with the derivations to the previous layer,

$$\frac{\partial MSE^{(i)}}{\partial W_{r,s}^{[l-1]}} = \left(\sum_{k=1}^{n+1} \frac{\partial MSE^{(i)}}{\partial a_k^{[l](i)}} \frac{\partial a_k^{[l](i)}}{\partial u_k^{[l](i)}} \frac{\partial u_k^{[l](i)}}{\partial a_r^{[l-1](i)}}\right) \frac{\partial a_r^{[l-1](i)}}{\partial W_{r,s}^{[l-1]}} = \left(\sum_{k=1,k\neq j}^{n+1} \frac{\partial MSE^{(i)}}{\partial a_k^{[l](i)}} \frac{\partial a_k^{[l](i)}}{\partial a_r^{[l-1](i)}}\right) \frac{\partial a_r^{[l-1](i)}}{\partial W_{r,s}^{[l-1]}} \tag{3}$$

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.

In order to understand why the encoder tends to be non linear, we should look at the gradient of the loss wrt to the weights of the encoder $\mathbf{w}^{[1]}$. For that, We can rewrite equation (17) as follows,

$$\frac{\partial MSE}{\partial w_s^{[1]}} = \sum_{j=1}^{n} \left(\sum_{i=1}^{n+1} x_s^{(i)}(y_j^{(i)} - x_j^{(i)})\right) \left(\sum_{k=1}^{n} W_{j,k}^{[3]} w_k^{[2]}\right) \tag{4}$$

The equation (4) shows that the amplitude of the input vector $\mathbf{x}^{(i)}$ strongly affects the learning of the weights $\mathbf{w}^{[1]}$ because of the term $x_s^{(i)}$. This is a bad news given the natrure of our dataset. To make the idea clear, let's take two extreme cases:

For the entry $s = 1$, out of the $n + 1$ step functions only one is not null. Let $x^{(1)}$ be that step function such that $x_1^{(1)} = 1$. For all $i \neq 1$, $x_1^{(i)} = 0$. Thus, the gradient wrt to $w_1^{[1]}$ becomes

$$\frac{\partial MSE}{\partial w_1^{[1]}} = \sum_{j=1}^{n} (y_j^{(1)} - x_j^{(1)}) \left(\sum_{k=1}^{n} W_{j,k}^{[3]} w_k^{[2]}\right) \tag{5}$$

For the entry $s = n$, only one step function has a zero value. Let $x^{(n+1)}$ be the step function such that $x_n^{(n+1)} = 0$. For all $i \neq n + 1$, $x_1^{(i)} = 1$.Thus, the gradient wrt to $w_n^{[1]}$ becomes

$$\frac{\partial MSE}{\partial w_n^{[1]}} = \sum_{j=1}^{n} \left(\sum_{i=1}^{n} (y_j^{(i)} - x_j^{(i)})\right) \left(\sum_{k=1}^{n} W_{j,k}^{[3]} w_k^{[2]}\right) \tag{6}$$

$$= \frac{\partial MSE}{\partial w_1^{[1]}} + \sum_{j=1}^{n} \left(\sum_{i=2}^{n} (y_j^{(i)} - x_j^{(i)})\right) \left(\sum_{k=1}^{n} W_{j,k}^{[3]} w_k^{[2]}\right) \tag{7}$$

Since we initialize $\mathbf{w}^{[1]}$ with close to zero values, and since we want the learned values $w_s^{[1]}$ to be equal at the end of the learning, we expect that the accumulated gradients over all epochs $\frac{\partial MSE}{\partial w_s^{[1]}}$ will be approximately equal.

To be continued

A 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.

$$MSE = \frac{1}{2}\sum_{i=1}^{n+1}\sum_{j=1}^{n}(y_j^{(i)} - x_j^{(i)})^2 \tag{8}$$

$$= \frac{1}{2}\sum_{i=1}^{n+1}\sum_{j=1}^{n}(\sum_{k=1}^{n} W_{j,k}^{[3]}(w_k^{[2]}z^{(i)} + b_k^{[2]}) - x_j^{(i)})^2 \tag{9}$$

<u>Layer 3</u>

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

$$\frac{\partial MSE}{\partial W_{u,v}^{[3]}} = \sum_{i=1}^{n+1}(w_v^{[2]}z^{(i)} + b_v^{[2]})(y_u^{(i)} - x_u^{(i)}) \tag{10}$$

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

$$\frac{\partial MSE}{\partial W^{[3]}} = \sum_{i=1}^{n+1}(\mathbf{y}^{(i)} - \mathbf{x}^{(i)})(\mathbf{w}^{[2]}z^{(i)} + b^{[2]})^T \tag{11}$$

<u>Layer 2</u>

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

$$\frac{\partial MSE}{\partial w_r^{[2]}} = \sum_{i=1}^{n+1}\sum_{j=1}^{n} z^{(i)}W_{r,j}^{[3]}(y_j^{(i)} - x_j^{(i)}) \tag{12}$$

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

$$\frac{\partial MSE}{\partial \mathbf{w}^{[2]}} = \sum_{i=1}^{n+1} z^{(i)}W^{[3]}(\mathbf{y}^{(i)} - \mathbf{x}^{(i)}) \tag{13}$$

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

$$\frac{\partial MSE}{\partial b_r^{[2]}} = \sum_{i=1}^{n+1}\sum_{j=1}^{n} W_{r,j}^{[3]}(y_j^{(i)} - x_j^{(i)}) \tag{14}$$

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

$$\frac{\partial MSE}{\partial \mathbf{b}^{[2]}} = \sum_{i=1}^{n+1} W^{[3]}(\mathbf{y}^{(i)} - \mathbf{x}^{(i)}) \tag{15}$$

<u>Layer 1</u>

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

$$\frac{\partial MSE}{\partial w_s^{[1]}} = \sum_{i=1}^{n+1}\sum_{j=1}^{n}\sum_{k=1}^{n} W_{j,k}^{[3]}w_k^{[2]}\frac{\partial z^{(i)}}{\partial w_s^{[1]}}(y_j^{(i)} - x_j^{(i)}) \tag{16}$$

$$= \sum_{i=1}^{n+1}\sum_{j=1}^{n}\sum_{k=1}^{n} W_{j,k}^{[3]}w_k^{[2]}x_s^{(i)}(y_j^{(i)} - x_j^{(i)}) \tag{17}$$

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

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