**Antoine CAROSSIO – X2016 – INF554 Machine Learning I – Assignment 1 – October 2018**

**Main goal**: Minimize *Hamming Loss* according to the following parameters: type of optimizer, cost function, number of hidden layers, learning rate, batch size, weight decay, number of nodes in each layer, threshold.

The default *Hamming Loss* (with an empty Network class) is 0.17895833… I tried to make better!

It would take a really long time to as all the possible combinations, so I adopted the following strategy.

**1)** After some research on the Internet and some manual tests, I found that AdamOptimizer() and sigmoid\_cross\_entropy\_with\_logits and were supposedly the best optimizer and cost function for this multi-label multi-class classification exercise.

**2)** Then, I assumed that the most influent parameters were the number of hidden layers & the learning rate and I tried to find the best combination, by fixing other parameters to reasonable values.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| LR \ Layer | **2** | 3 | 4 | 5 |
| 0.0001 | 0.082 | 0.084 | 0.088 | 0.087 |
| **0.001** | **0.081** | 0.083 | 0.088 | 0.081 |
| 0.01 | 0.094 | 0.083 | 0.109 | 0.118 |
| 0.1 | 0.195 | 0.179 | 0.179 | 0.179 |

🡪 I choose learning\_rate=0.001. Since 2 layers give the same result as 5, I fix num\_hidden\_layers=2

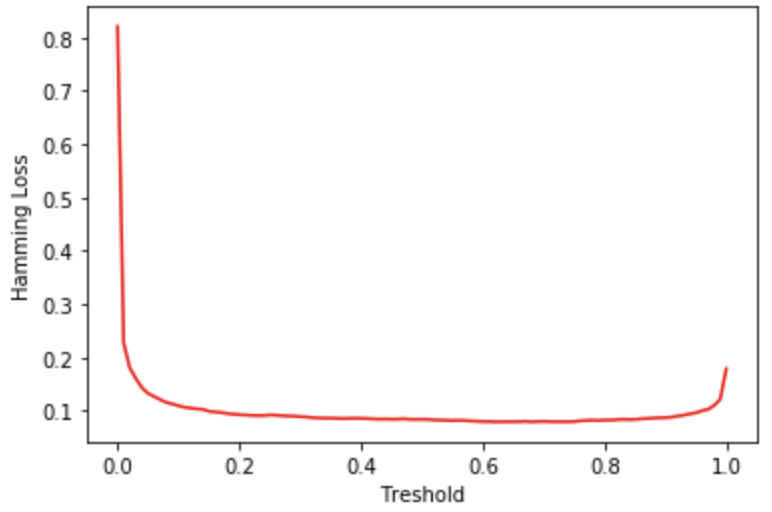
**3)** With these new values fixed, I tried to optimize the best batch size and weight decay combination.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Batch \ Decay | 0 | 0.00001 | **0.0001** | 0.001 |
| 8 | 0.089 | 0.089 | 0.092 | 0.096 |
| 16 | 0.089 | 0.083 | 0.088 | 0.098 |
| **32** | 0.088 | 0.086 | **0.079** | 0.100 |
| 64 | 0.088 | 0.083 | 0.095 | 0.095 |

🡪 Fix batch\_size=32 and weight\_decay=0.0001

**4)** After, I looked for the most relevant number of nodes in each of the two hidden layers, with n\_1 > n\_2 (especially so that the test does not last too long…)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| n2 \ n1 | 16 | 32 | 64 | **128** | 256 |
| 16 | 0.093 | 0.085 | 0.080 | 0.080 | 0.082 |
| **32** |  | 0.084 | 0.084 | **0.079** | 0.081 |
| 64 |  |  | 0.088 | 0.085 | 0.085 |
| 128 |  |  |  | 0.084 | 0.085 |
| 256 |  |  |  |  | 0.082 |

🡪 Thus, the optimal number of nodes are n\_1=128 and n\_2=32

**5)** Finally, find optimal threshold in the predict() function, which turns out to be threshold=0.6, where I actually get a final hamming loss of **0.076**.

**Comparison with other methods**:

Once trained during 120 seconds, the neural network we created is way faster and gives better results than any other method as shown on the chats below.

