Big Data and Artificial Intelligence

Lab Assignment (TP) 1: Introduction to Deep Learning and Neural Style Transfer

Supervisor: Montmaure Antoine

Written by : *Othmane Cherai & Hamza Houbbane*

Instructor: *Son Vu*
Department: Département Informatique
Date: March 22, 2023

Contents

1 Understanding Deep Learning with a Simple Task

1.1 A simple Neuron

This first section will be about the simple task that is inverting a boolean value *i.e*:

$$f : \{0, 1\} \to \{0, 1\}$$
$$x \mapsto 1 - x$$

A quick glimpse of the first algorithm given reveals that we will will be generating two main batches of data: **"data"** and **"label"**. Knowing this it is obvious that we will be using a **supervised** machine learning algorithm: **Neural Network**. *"Label"* is a common term that means in this context the value to which we will compare the model's output. The smaller is the gap when comparing the output to the label, the better is the model. The data size of the data is relatively small for a machine learning problem. As such, online learning is of no use since the data could be inputted in one go and is not updated continuously:Data is generated and used for the training once. Thus, we are going for **Batch Learning**. Finally, as clearly stated in the next section's title, we will be using a **Model-based ML**. This the common terminology for a system to which data is fed along with the parameters: *"hyper-parameters"*, that need to be tweaked in order to get better performance for a said task. A summary of this would go along the lines: **"The Model learns from Experience:E with respect to task: T and a performance measure:P ,where its performance on T was measured by P, improves with experience E."** with:

- E: Label
- T: function f
- P: to be explicated further in the report

1.1.1 Generating Data

The whole purpose of this code is generating **Train** and **Test** sets.

- **Train set**: is the set which the model will be using to adapt its parameters for better generalising purposes. This could be qualified as the learning phase.
- **Test Set**: is the set where the model will be testing its generalisation capacity. If the model performs well in the test set, but performs poorly in the Test set, we call that **Over-fitting**. It is indicative that your model does not generalise well to other cases.

As for the proportions of the aforementioned, a good rule of thumb is to take 80% of the data as the training set and the rest as the test set.

1.2 The Simple Training

The basic principle of supervised training is as follows:
Say you have a vector $X^{(k)} \in R_{(n,1)}$ a vector containing the features of the $k^{(th)}$ instance(data

line) and $W^{(k)} \in R_{(1,n)}$ a vector containing the weight for each feature. $X \times W = h(\theta)$ is the systems prediction function, also called **hypothesis**.
When the model is given an input Vector $X^{(k)}$, it calculates the distance between it and the label of the respective instance with a certain defined **Metric**. The choice of the metric is dependant of the problem at hand. A good model is one that minimises to a certain degree, the distance between the model output and the expected result. As such, the model will tweak the weights at each iteration so as to get a Loss as minimal as possible. The process which is used in order to do so in called **Gradient Descent** and it is done in the space(or hyperspace) generated by the hyper-parameters . For easier explaining purposes, we will explain the case of $Dim = 1$. Consider a one-variable multivariate function describing this space. The purpose here is the find the point that minimises this function. for visualising purposes, consider $f(x) = x^4 - 2x^3 + 2$

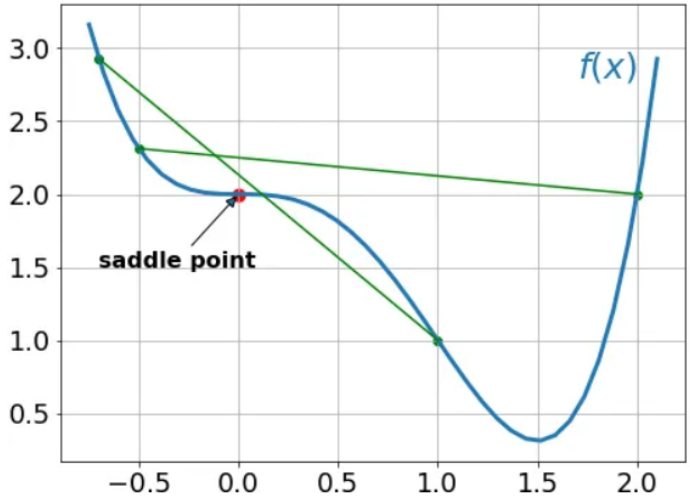


Figure 1: Semi-convex function with a saddle Point

consider now a random point on the curve. Then make a scaled step in the opposite direction of the gradient. Iterate the last step. There are two possible outcomes for this algorithm.

1. The point is located right to the saddle point: In this case the algorithm will result in the point ending in the saddle point and staying there ,since it is a null-variation-point($\frac{\partial f}{\partial x} = 0$
2. The point is in the region to the right of the aforementioned point and will end up in the absolute minimum of the function

In the first case, the absolute minimum is not reached and as a result so is the case for the Loss function. Generally, we do not know the weight of the parameters, so we choose them randomly. But we might then, end up in a local minimum. For such reason, we take randomly many points and retain the weight that minimise the best the Loss function. The only way to no have to do this, is to make sure that the space generated by the hyper parameters is **convex** and has no **saddle regions**.
With this explained, we now have to choose an **Optimzer and a Criterion**. The first handles the steps of the Gradient Descent Update and the second is the Loss function. Many other aspects are to be taken into account. For report-length report, they will not be explicitised but do note that they are so in the notebook.

1.3 The result

Below is the graph depicting the evolution of **train loss** and **test loss** over 1000 tries in 1000 different batches.

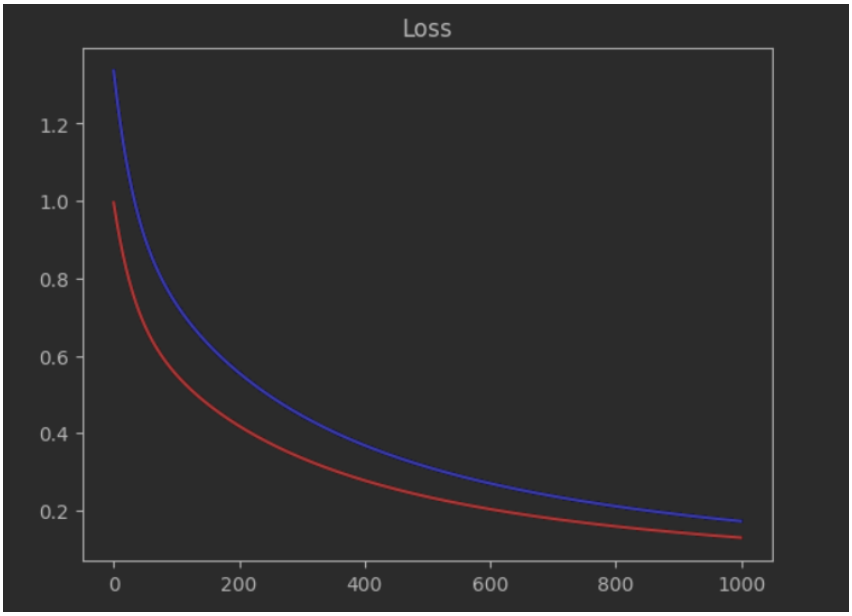


Figure 2: Test Loss: '–' Train Loss:'–'

We can see that as expected the train loss is as expected below the test loss. This is the correct outcome since initially, the model does not generalise well to other instances. The more the Model is trained the better it is expected to perform well, and this is in fact what we are observing. Both losses are converging to the same values. In light of this, we are sure that there is no over-fitting and that our model does a good job predicting the outcomes.

2 Neural Style Transfer

Now that the basic principles of machine learning are well explained, we will just go over the new features introduced by this ML algorithm and the inspiration behind some of its remarkable changes.

2.1 Main Aspects to build the NST network

2.1.1 variables

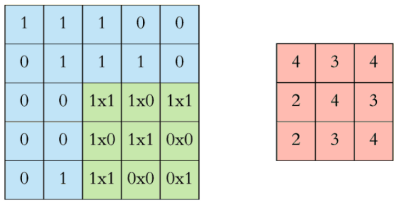
the specifics of this project lead to have the following variable:

- **Content Image**: Which will be the image that will implement the style introduced in the style image
- **Style Image**: Contains the blue print of the drawing technique to be used to draw the modified content image.

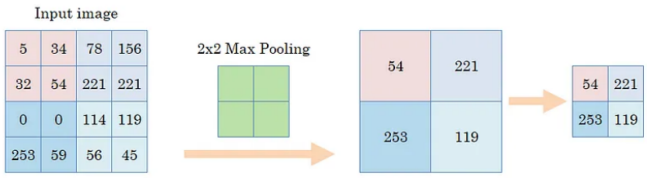
- **Generated Image**: Is the image outputted by the model. It is a representation of the content image drawn in the style image style.
- **pretrained weights and biases**

2.1.2 Pretrained Model

The code itself for each part is fairly well explained in the project notebook. The imported model here is **VGG19**. The latter has pretrained weights and is adapted to the task at hand. It uses a series of convolution and pooling layers . What the first means is: it basically calculate averages of features for entire blocks of pixels. The filters are convolution products. As for the latter, it is about performing aggregations over groups of pixels.



(a) Convolution



(b) Pooling

These two operations' purpose is to reduce the the data size and only keep the important feature and further intensify them.(watch https://www.youtube.com/watch?v=KuXjwB4LzSA) to better understand.)

2.1.3 Loss Function

Unlike the first project, here we will define two loss functions: **Content Loss** and **Style Loss**. The first ensures that the activation of the layers is similar between the generated and content image. Whereas the latter, makes sure the correlation of activation of all layers are similar between the style image and the generated image.

$$\mathcal{L}^l_{\text{style}}\left(\vec{a},\vec{x}\right) = \text{MSE}\left(G^l, A^l\right) = \frac{1}{N_l^2}\sum_{i,j}\left(G^l_{ij} - A^l_{ij}\right)^2$$

$$\mathcal{L}_{\text{style}}\left(\vec{a},\vec{x}\right) = \sum_{l=0}^{L} w^l_{\text{style}}\,\mathcal{L}^l_{\text{style}}\left(\vec{a},\vec{x}\right)$$

Essentially $\mathcal{L}^l_{style}$ which is a Mean Squared Error is capturing the error between activations produced by the generated image and content image Now consider:

$$\mathcal{L}^l_{\text{content}}\left(\vec{p},\vec{x}\right) = \text{MSE}\left(F^l, P^l\right) = \frac{1}{N_l M_l}\sum_{i,j}\left(F^l_{ij} - P^l_{ij}\right)^2$$

$$\mathcal{L}_{\text{content}}\left(\vec{p},\vec{x}\right) = \sum_{l=0}^{L} w^l_{\text{content}}\,\mathcal{L}^l_{\text{content}}\left(\vec{p},\vec{x}\right)$$

$\mathcal{L}^{l}_{content}$ is defined as the difference of correlation between the features computed by the generated image and style image. The **Gram Matrix** which defines style matrices for both content and style finds its roots in the fact that a style matrix is computed throughout an element wise multiplication of different features. Moreover, this matrix captures the distribution of features of a set of feature maps. By minimising the Loss, we are essentially matching the distribution of features between the two images.
From here on, we are doing the same as the first exercise *i.e* minimising the losses by tweaking the hyper-parameters and weights via a specific gradient descent.
N.B: Since these operation are resources consuming and luckily parallelizable, we are moving the calculation from the CPU to the GPU.The **Cuda** that you see in the notebook is a parallel computing platform developed by NVIDIA and which enables faster calculus using the GPU's resources rather than the CPU's.

Figure 4: Model Output

Similarly we observe the same variation pattern for the test loss as well as the train loss as observed in the the first example.

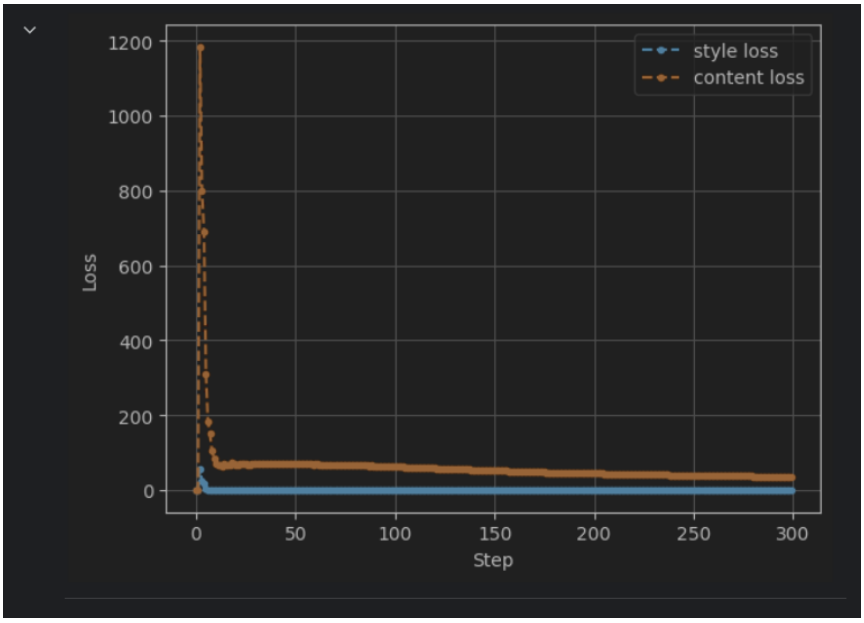


Figure 5: Model Output