Miniproject 2: Report

Luca Bracone, Omid Karimi, Gianni Lodetti

May 2021

1 Introduction

In order to gain a better appreciation of the comfort granted to us by libraries such as pytorch, for the second mini-project we implement some of the modules on our own. Using only pre-made functions for matrix operations and data helpers.
Overall, we have implemented a module to combine layers as well as different activation functions and loss criteria, which apply forward and backward passes seamlessly. We test our architecture by generating $n = 1000$ points in $[0, 1] \subset \mathbb{R}^2$, labelling each of them 1 if they are inside the circle centered at $(0.5, 0.5)$ with radius $1/\sqrt{2}$ and 0 outside.

2 Modules

Each module is equipped with a `forward`, `backward` and `param` method. The `forward` method takes an input x and is tasked of calculating and returning the result of $f(x)$, where f is a function that is defined for each module. The `backward` method takes as input the gradient of the layer in front, and is tasked of calculating the gradient with respect to the current layer. Finally, the `param` method returns a list of the parameters of the module with their gradient (if there are parameters at all).

2.1 Sequential

The Sequential module is the one that combines all the others. It is initialised with a list of modules. Its forward method takes any input of correct size and calls forward on each of the modules inside Sequential, taking as input the output of the previous iteration. It outputs the final output obtained in this way. Its backward method takes as input the gradient of the loss, then calls backward on each of the modules in reverse order, taking as input the output of the previous iteration. It outputs the final output obtained in this way. The param method returns the concatenated list of the parameters of each module inside Sequential.

2.2 Linear

The Linear module is initialised by specifying an input size n and output size m. The weights and biases are initialised using a Gaussian distribution of mean zero. The variance can be specified by the user.

- Default: 1
- He: $\sqrt{\frac{2}{n}}$
- Xavier: $\sqrt{\frac{2}{n+m}}$

The forward method of the Linear module computes a matrix-vector product followed by adding a bias. Where the vector x is the input from the previous layer, the matrix W is a parameter of the layer, and the bias b a vector whose length is equal to the output layer's, is also a parameter. So the output is a vector

$$y_i = b_i + \sum_{j=1}^{n} w_{i,j} x_j.$$

The backward method takes as input the gradient of the loss w.r.t. the parameters of the following layer

∇L and calculates the following quantities

$$\left(\frac{\partial L}{\partial w_{i,j}}\right) = x^t \cdot \nabla L$$
$$\frac{\partial L}{\partial b_i} = \nabla L$$

The zero_grad method sets the gradient of the parameters to zero and the param method returns the parameters with their gradient.

2.3 Activations

2.3.1 ReLU

The forward method returns

$$f(x) = \begin{cases} x & \text{if } x \geq 0 \\ 0 & \text{otherwise.} \end{cases}$$

The backward method is then

$$f'(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

2.3.2 Tanh

The forward method returns

$$f(x) = \tanh(x).$$

The backward method is then

$$f'(x) = \cosh(x)^{-2}.$$

2.3.3 Sigmoid

The forward method returns

$$f(x) = \frac{1}{1 + \exp -x}.$$

The backward method is then

$$f'(x) = f(x) \cdot (1 - f(x)).$$

2.4 Losses

2.4.1 Mean Square Error

Given an ouput vector $\hat{y}$ whose target is y the loss is

$$L(\hat{y}) = \frac{1}{n} \sum_{i=1}^{n} (y - \hat{y})^2.$$

So the gradient is

$$\nabla L(\hat{y}) = \frac{2}{n}(y_i - \hat{y}_i)$$

2.4.2 Cross Entropy

For binary classification and given an output vector $\hat{y}$ whose target is y, the cross entropy loss is

$$L(\hat{y}) = y(-\log(p(\hat{y}))) - (1 - y)\log(1 - p(\hat{y})).$$

We choose $p(x)$ as the sigmoid function. The gradient is

$$\nabla L(\hat{y}) = -\frac{y}{p(\hat{y})} + \frac{1 - y}{1 - p(\hat{y})}$$

3 Experimentation

We test two models. Each of them has two input nodes, followed by three fully connected layers of size 25, which feed into a single output node. The only difference is that one uses tanh as activation while the other uses ReLU. Although we implemented other activations and loss functions, we decided to only showcase those two in this report because they are the ones who gave the best results. We talk about why we think this is the case in the conclusion.

Figure 1 on page 3 shows the average over 30 models of the evolution of the loss with respect to training epoch.

We see that on average for this model tanh performs better.

Furthermore we see on figure 2 that the errors were only produced for points that were close to the boundary.

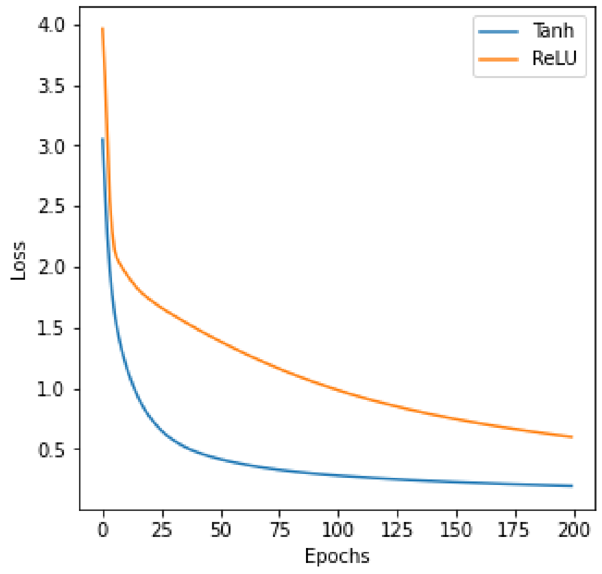
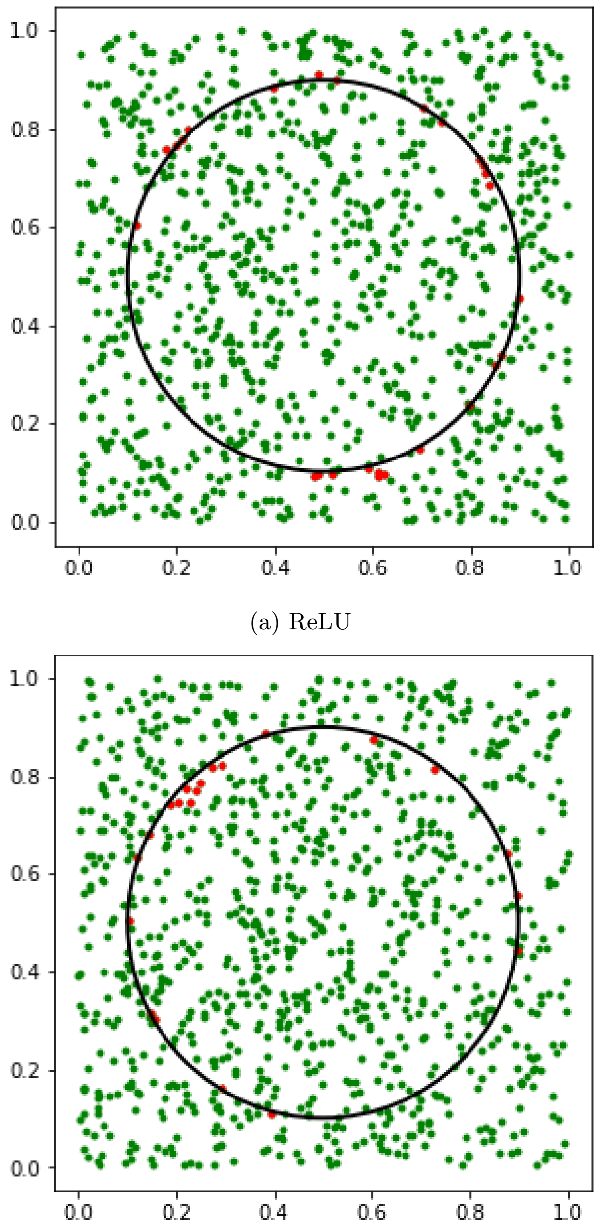


Figure 1: Evolution of loss (in logarithm) with respect to training epoch

4 Conclusion

Writing the framework from scratch was an interesting exercise that allowed us to familiarize ourselves with the mathematical theory of the course. Some of the functions were obtained effortlessly while others were built only after a considerable amount of time and effort by tinkering with every imaginable parameter. Some funcitons never came to fruition, such as the cross entropy loss or the sigmoid activation. Although we believe the code to be mathematically correct, it may be possible that unforseen issues such as parameter initialisation or using a different optimiser may cause it to give unsatisfactory results. Nonetheless it is unsurprising that MSE gives better results. Indeed, the geometry of the problem (the circle shape) is perfect to use MSE – in fact MSE defines the labelling. Though in a real context we would have no way of knowing what the underlying shape of data is. For a future project it would be instersting to implement more functions from scratch and test them on a realistic data set.



(a) ReLU

(b) Tanh

Figure 2: The predicitons for some of the two models among the 30. Green means a correct prediciton, red means incorrect. The theoretical circle is drawn for clarity.