


on Polynomial Approximation of Activation Function

 John Chiang

liyue.sun@mail.nankai.edu.cn, john.chiang.smith@gmail.com

1 Introduction

When it comes to applying homomorphic encryption technique to machine learning such as neural network, there is a technique problem that non-polynomial of activation function could not be calculated directly in the HE domain. The common way to deal with it is to approximate the activation function using the least-square method, to a polynomial that can be calculated in a HE-based environment[1]. Being widely adapted in recent work related to HE[2], however, the least-square method might not be ideal for this task. In this essay, we propose a neat method to approximate the activation function based on the least squared method.

Note that the idea of this essay did come to the present author, but we can not guarantee that other researchers haven't found it (we should give some time to read the book carefully before trying to develop some own method). In conclusion, this is a simple idea that can properly work for approximating activation function of HE. //

2 Least Square Method

For a continuous real-value function $f(x)$, a simple version of least square method to approximate $f(x)$ over range $[a, b]$ by a polynomial of a given degree n is to find a polynomial of at most degree n , $p_n(x)$, such that minimise the

$$\int_a^b (f(x) - p_n(x))^2 dx.$$

For approximating a continuous real-value function $f(x)$ over the range $[a, b]$ by a polynomial of a given degree n , a simple version of least square method is to find a polynomial of at most degree n , $p_n(x)$, such that maximise the

$$\int_a^b (f(x) - p_n(x))^2 dx,$$

which has a unique solution (the best approximation). Least square method is such a common method that many softwares implement this method, such as the function `polyfit` in Python, Matlab or Octave. For example, only 3 lines of Octave codes is all it needs to fit the activation function $ReLU(x) = \max(0, x)$ over the range $[-8, +8]$ by a polynomial of degree 2, resulting in the figure 1:

```
x = [-8:0.000001:+8];  
y = max(0,x);  
polyfit(x,y,2)
```

and we get the polynomial approximation $p_2^{ls}(x) = 0.058594 + 0.500000 \cdot x + 0.750000 \cdot x^2$

Even though least square method has many advantages, it might not be the best choice in the sense of a HE-based environment. In the sense of a HE-based environment, we would like to choose a low-degree polynomial due to the expensive HE operations, and also prefer the similar curve shape between the approximation polynomial and the function to approximate, even at the cost of losing best approximation. Thus, we propose a simple method to consider the gradient (the slope) into the optimal function.

and its limitation in the polynomial approximation of activation function To minimise the gray area of the is not enough, we perhaps would like a polynomial approximation that has a similar shape to the activation function, even at the cost of some larger gray area.

3 Our Method

In this section, we present a simple method based on the least-square method, which includes the graduals (shape) of the Polynomial and the activation into the least-square method, in order to get the desired polynomial approximation that we described above. such that minimise the

$$\int_a^b (f(x) - p_n(x))^2 + (f'(x) - p'_n(x))^2 dx.$$

For a toy example, we use this method for the same task above, that is to fit the activation function $ReLU$ over the range $[-8, +8]$ by a polynomial of degree 2 denoted by $p_2^{lg}(x) = c + b \cdot x + a \cdot x^2$. The question is to minimise the observation loss function

$$\begin{aligned} F(a, b, c) &= \int_{-8}^0 (f(x) - p_2^{lg}(x))^2 dx + \int_0^{+8} (f(x) - p_2^{lg}(x))^2 dx + \int_{-8}^0 (f'(x) - p_2^{lg'}(x))^2 dx + \int_0^{+8} (f'(x) - p_2^{lg'}(x))^2 dx \\ &= \left[\frac{4}{3}a^2x^3 + (b^2 + 1 - 2b)x + 2a(b-1)x^2 \right]_0^{+8} + \\ &\quad \left[\frac{1}{5}a^2x^5 + \frac{1}{3}b^2x^3 + bcx^2 - \frac{2}{3}bx^3 + c^2x + \frac{1}{3}x^3 - cx^2 + \frac{1}{2}abx^4 - \frac{1}{2}ax^4 + \frac{2}{3}acx^3 \right]_0^{+8} + \\ &\quad \left[\frac{4}{3}a^2x^3 + b^2x + 2abx^2 \right]_{-8}^0 + \\ &\quad \left[\frac{1}{5}a^2x^5 + \frac{1}{3}b^2x^3 + c^2x + bcx^2 + \frac{1}{2}abx^4 + \frac{2}{3}acx^3 \right]_{-8}^0 + \\ &= 14472.5\dot{3}a^2 + 357.\dot{3}b^2 + 16c^2 + 682.\dot{6}ac - 357.\dot{3}b + 178.\dot{6} - 64c - 2176a \end{aligned}$$

Minimising $F(a, b, c)$ is a problem of unconstrained optimization. To do that, we need to calculate the first-order partial derivatives of $F(a, b, c)$ and have them equal to 0 :

$$\begin{aligned} F_a(a, b, c) &= 28945.0\dot{6}a + 682.\dot{6}c - 2176 = 0 \\ F_b(a, b, c) &= 714.\dot{6}b - 357.\dot{3} = 0 \\ F_c(a, b, c) &= 32c + 682.\dot{6}a - 64 = 0 \end{aligned}$$

Solving these equations, we obtain the unique solution and the polynomial approximation, respectively:

$$\begin{aligned} a &= 0.0563686709, b = 0.5, c = 0.7974683544, \\ p_n^{lg}(x) &= 0.797468 + 0.500000 \cdot x + 0.056369 \cdot x^2 \end{aligned}$$

We compare the polynomial generated by least square, $p_n^{ls}(x)$, with that of our method $p_n^{lg}(x)$.

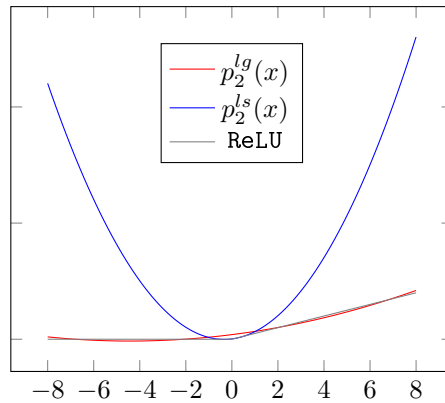


Figure 1: Results in the

From the figure 1, we can see that our polynomial are much the same as ReLU in the term of shape(slope), and that the least square polynomial doesn't even look like the ReLU function.

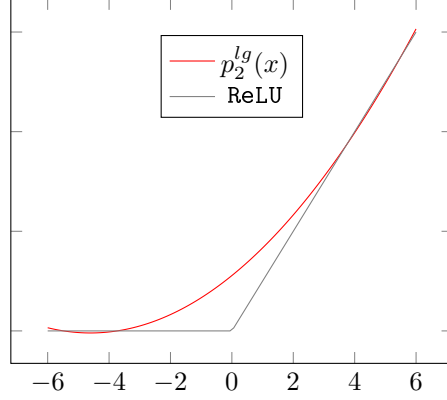


Figure 2: Results in the

A more flexible way to generate the polynomial approximation is to set add more parameters to control the weight or proportion of the least square to slope gradient resulting polynomial. To take the *ReLU* within the domain $[-L, +L]$ as an example, we could optimise the following function:

$$F(a, b, c) = \lambda_0 \cdot \int_{-L}^0 (0 - p_2^{lg}(x))^2 dx + \lambda_1 \cdot \int_0^{+L} (x - p_2^{lg}(x))^2 dx \\ + \lambda_2 \cdot \int_{-L}^0 (0 - p_2^{lg'}(x))^2 dx + \lambda_3 \cdot \int_0^{+L} (1 - p_2^{lg'}(x))^2 dx,$$

where $\lambda_0, \lambda_1, \lambda_2$ and λ_3 are four real numbers greater than zero to control the various weight proportion.

Another example is, supposing that we want to find a polynomial to approximate the *relu* over the domain $[-6, +6]$ such that it fits the *ReLU* very well at the both ends at the expense of losing some precision around the center. In this case, we need to minimise the function (we just set $\lambda_0 = \lambda_1 = \lambda_2 = \lambda_3 = 1$):

$$F(a, b, c) = \int_{-6}^{-3} (0 - p_2^{lg}(x))^2 dx + \int_{+3}^{+6} (x - p_2^{lg}(x))^2 dx + \int_{-6}^{-3} (0 - p_2^{lg'}(x))^2 dx + \int_{+3}^{+6} (1 - p_2^{lg'}(x))^2 dx \\ = 3517.2a^2 + 132b^2 + 6c^2 + 54bc + 1917ab + 252ac - 607.5a - 78b - 27c + 12.$$

Solving this unconstrained optimization, we get the polynomial $p_2^{(1)} = 1.1110537229 + 0.5 \cdot x + 0.054235537 \cdot x^2$. Figure 2 shows that $p_2^{(1)}$ is exactly what polynomial we want.

4 Conclusion

Your conclusion here Our method to approximate activation function is much flexible compared to least square method in the sense of ml. Vouriac setting of parameters should result in polynomials of different shapes.

References

- [1] Guy Hadash, Einat Kermany, Boaz Carmeli, Ofer Lavi, George Kour, and Alon Jacovi. Estimate and replace: A novel approach to integrating deep neural networks with existing applications. *arXiv preprint arXiv:1804.09028*, 2018.
- [2] George Kour and Raid Saabne. Real-time segmentation of on-line handwritten arabic script. In *Frontiers in Handwriting Recognition (ICFHR), 2014 14th International Conference on*, pages 417–422. IEEE, 2014.