REPORT

Mehdi REZZOUG

2022

Contents

Approach

To tackle this challenge, I choose to divide my approach into two part. First, I studied the approximation capacity of neural network, and the effect of different parameters (depth, breadth of the network, activation function ...), in order to identify the network's best configuration for every function. Then, I introduced the constraint of quantization and analyzed the performance of the quantized models.

I didn't have time to study the High dimensional Gaussian function.

You can find the results of all the experiments in the archive "experiments.zip".

Chapter 1

Neural Network Approximation

1.1 Introduction

The universal approximation theorem states that a feed-forward network with a single hidden layer containing a finite number of neurons can approximate continuous functions on compact subsets of $\mathbb{R}^n$.

The first versions of the arbitrary width case was proven by George Cybenko in 1989 for sigmoid activation functions[1]. Kurt Hornik showed in 1991[2] showed the same result for the activation function that is continuous, bounded and non-constant. Moshe Leshno et al in 1993[3] showed that the universal approximation property, is equivalent to having a non-polynomial activation function.

1.2 Experiments

We approximated 4 functions :

- $f(x) = sin(x)$
- $f(x) = exp(-\frac{(x-2)^2}{2/5})$
- $f(x) = xy * exp(-x^2 - y^2)$
- $f(x) = x^2 - y^2$

During our experiments, we studied the impact of different parameters (see appendix 12.3):

Number Of Training Data : We analysed the precision of fit depending on the number of training examples. We observed that by increasing the size of training set the precision of the fit increases. After a certain number, the precision no longer increases.
Loss Function : We analysed the precision of fit depending on the loss function. We noticed that there is no big difference between MSE and L1loss. However the MSE is better.

Neural Network Architecture : We analysed the precision of fit depending on the depth and the breadth of the network. We observed that with a more complex network we can achieve better results but we need more data to train the network.

Activation Function : We analysed the precision of fit depending on the activation function. We compared 4 activation functions : Relu, LeakyRelu, Tanh and sigmoid. For most of our functions, the best results were realised by Relu activation function.

We can mention two benefits of Relu :

It reduce likelihood of the gradient to vanish. This arises when the input is positive. In this case the gradient has a constant value. In contrast, the gradient of sigmoids becomes increasingly small as the absolute value of x increases. The constant gradient of ReLUs results in faster learning. The other benefit of ReLUs is sparsity. Sparsity arises when the input is negative. The more such units that exist in a layer the more sparse the resulting representation. Sigmoids on the other hand are always likely to generate some non-zero value resulting in dense representations. Sparse representations seem to be more beneficial than dense representations.

But for the sinus function the sigmoid and tanh functions are more efficient in learning than the Relu. We observed that with less data it performs better in learning. We can explain this by the fact that the sinus function is bounded so a bounded activation function will be more efficient.

Noise In Training-Set : We analysed the precision of fit depending on the noise in training-set. We noticed that adding noise to the training data doesn't impact our results

but in some cases it helps to generalize out of domain data.

Remark

We observed that neural networks can't approximate periodic functions, like sinus, it can approximate the periodic function within a interval but can't capture the periodic property no matter the length of the training interval.

1.3 Conclusion and Perspective

In this part, we studied the approximation capability of neural networks. We confirmed the universal approximation theorem. We observed that neural networks do not interpolate well outside the training domain and this remark is more apparent in functions that are not monotonic like sinus.

As perspective, to fit sinus function, I suggest to test custom cost function to oblige the neural network to behave like a periodic function perhaps by using fourier transform. Another suggestion is to use periodic function as activation function and architectures with memory like LSTM or recurrent neural net.

Chapter 2

Neural Network Quantization

2.1 Introduction

2.1.1 What is Quantization?

Quantization is the process of mapping input values from a large set (often a continuous set) to output values in a (countable) smaller set, often with a finite number of elements. Rounding and truncation are typical examples of quantization processes

2.1.2 Why Quantization ?

Such techniques are mainly used to improve neural network inference time on embedded hardware. The goal is to replace costly operations done on floats by much more efficient and relatively good approximations done on small integers.

Deep neural network consists of many parameters which are known as weights. In many scenarios, the bottleneck of running deep neural network is in transferring the weights and data between main memory and compute cores. Therefore, modern AI chips have a lot of local memories around the compute cores to minimize the data transfer latency. By using 8-bit integer rather than 32-bit, we instantly speed up the memory transfer and use less memory storage.

2.1.3 Basic Concepts Of Uniform Quantization :

We need first to define a function that can quantize weights and activations to a finite set of values. A popular choice for a quantization function is as follows:

$$Q(r) = Int(r/S) - Z$$

where Q is the quantization operator, r is a real valued input (activation or weight),S is a real valued scaling factor, and Z is an integer zero point.

It is possible to recover real values r from the quantized values $Q(r)$ through an operation that is often referred to as dequantization:

$$\hat{r} = S(Q(r) + Z)$$

One important factor in uniform quantization is the choice of the scaling factor S. This scaling factor essentially divides a given range of real values r into a number of partitions :

$$S = \frac{\beta - \alpha}{2^b - 1}$$

where $[\alpha, \beta]$ denotes the clipping range, a bounded range that we are clipping the real values with, and b is the quantization bit width.

2.1.4 Types of Quantization :

There are three types of quantization :

- dynamic quantization (weights quantized with activations read/stored in floating point and quantized for compute.)
- static quantization (weights quantized, activations quantized, calibration required post training)
- static quantization aware training (weights quantized, activations quantized, quantization numerics modeled during training)

2.2 Experiments

(see appendix 22.3)

General Observation :

To built a neural network using quantized weights and activations of a maximum of 8 bits and its accumulators do not exceed 19 bits, we have define architectures with maximum of 8 neurons per layer. So we chose to study a neural network of 3 layers of 8 neurons.

According to our experiments, we observe that for the same amount of data, quantized networks have worse results than 32bit networks. Therefore, to have the same results, we need to increase the amount of data for quantized networks.

We also notice that the performances of the quantization methods are not the same. The best performing technique is dynamic followed by static and finally QAT.

We observed that the Relu function has better performance.

Regarding the training time, the QAT take longer to train.

Remark: The quantization functions are not compatible with GPU.

2.3 Conclusion and Perspective

In this part, we studied the impact of different quantization techniques on the fit capacity of neural net. We tested different quantization methods with different constraints.

As perspectives, I suggest to test other type of quantization like non-uniform quantizations.

Bibliography

[1] George Cybenko. “Approximation by superpositions of a sigmoidal function”. In: *Mathematics of control, signals and systems* 2.4 (1989), pp. 303–314.

[2] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. “Multilayer feedforward networks are universal approximators”. In: *Neural networks* 2.5 (1989), pp. 359–366.

[3] Moshe Leshno et al. “Multilayer feedforward networks with a nonpolynomial activation function can approximate any function”. In: *Neural networks* 6.6 (1993), pp. 861–867.

Appendix 1

Figure 2.1: Training of EXP function with 500 Data / MSE loss / relu activation

Figure 2.2: Training of EXP function with 50000 Data / MSE loss / relu activation

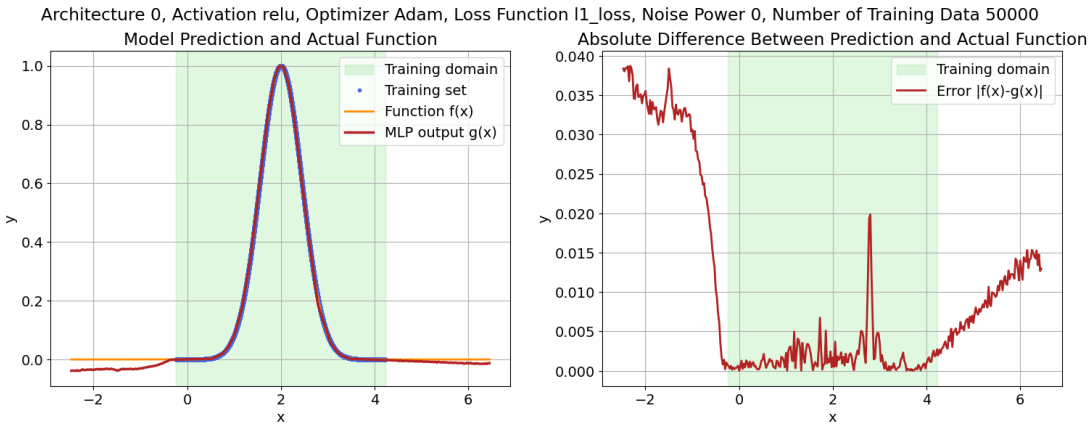

Figure 2.3: Training of EXP function with 50000 Data / L1 loss / relu activation

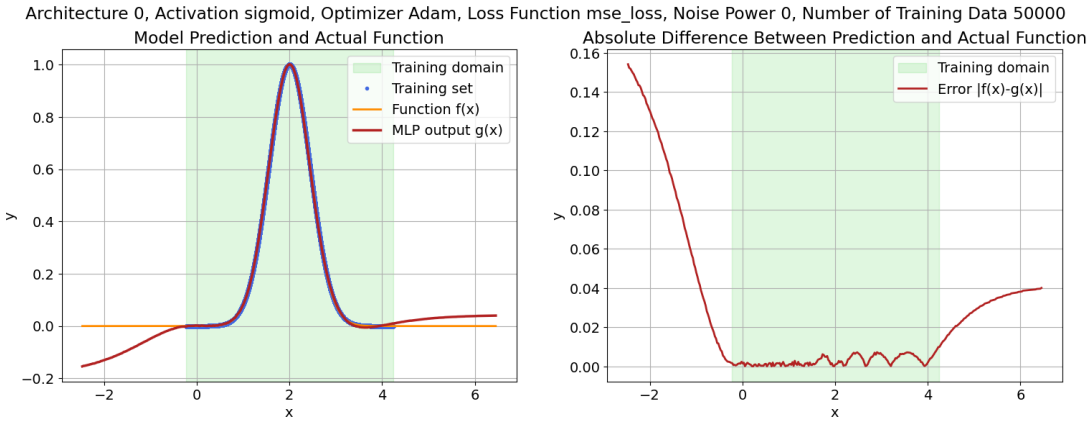

Figure 2.4: Training of EXP function with 50000 Data / MSE loss / sigmoid activation

Figure 2.5: Training of EXP function with 50000 Data / relu activation

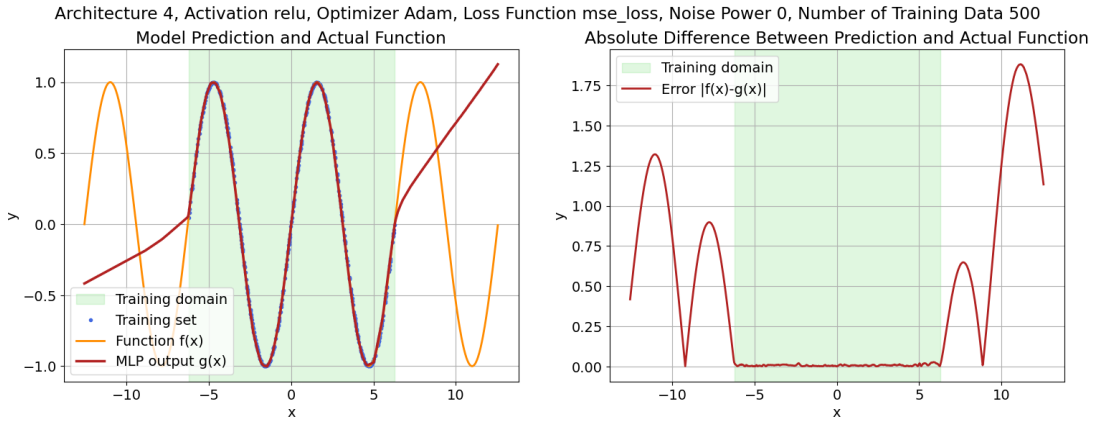

Figure 2.6: Training of SIN function with 500 Data / MSE loss / relu activation

Figure 2.7: Training of SIN function with 50000 Data / MSE loss / relu activation

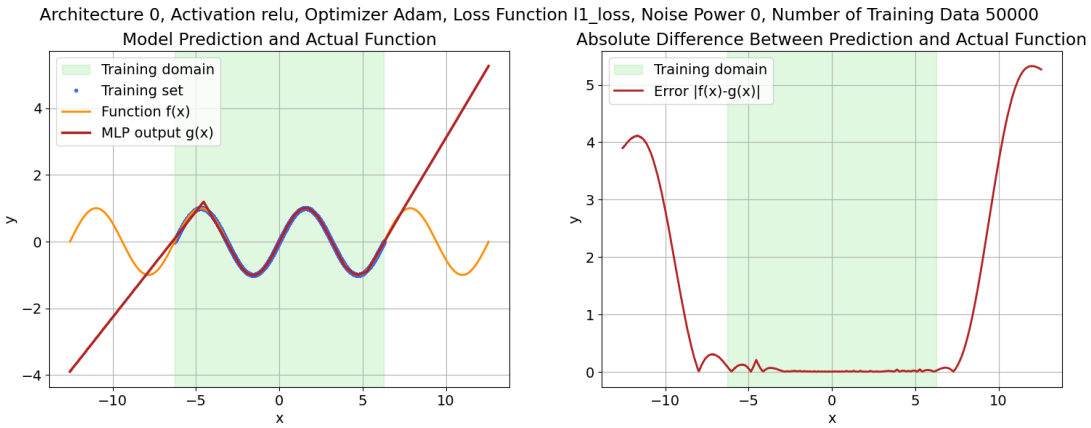

Figure 2.8: Training of SIN function with 50000 Data / L1 loss / relu activation

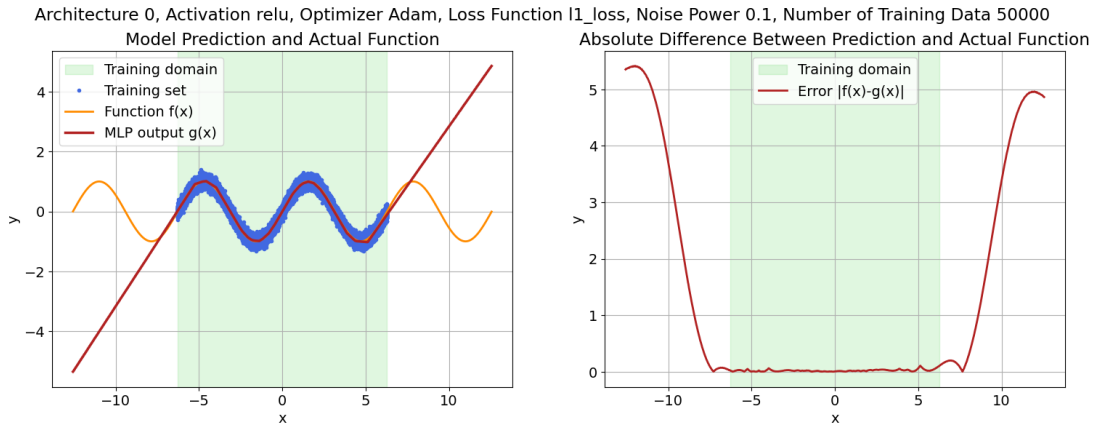

Figure 2.9: Training of SIN function with 50000 Data / L1 loss / relu activation with noise

Figure 2.10: Training of SIN function with 50000 Data / L1 loss / sigmoid activation

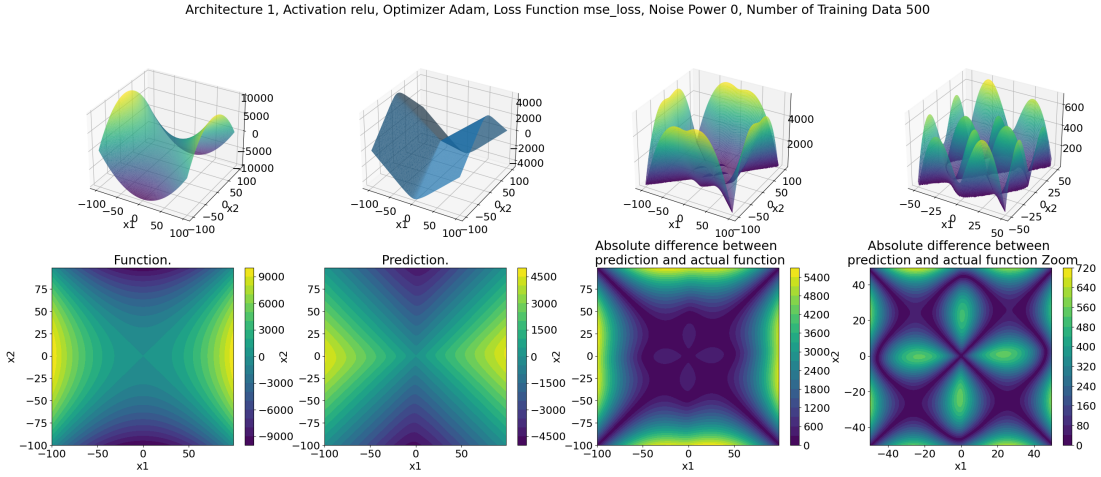

Figure 2.11: Training of SADDLE function with 500 Data / MSE loss / relu activation

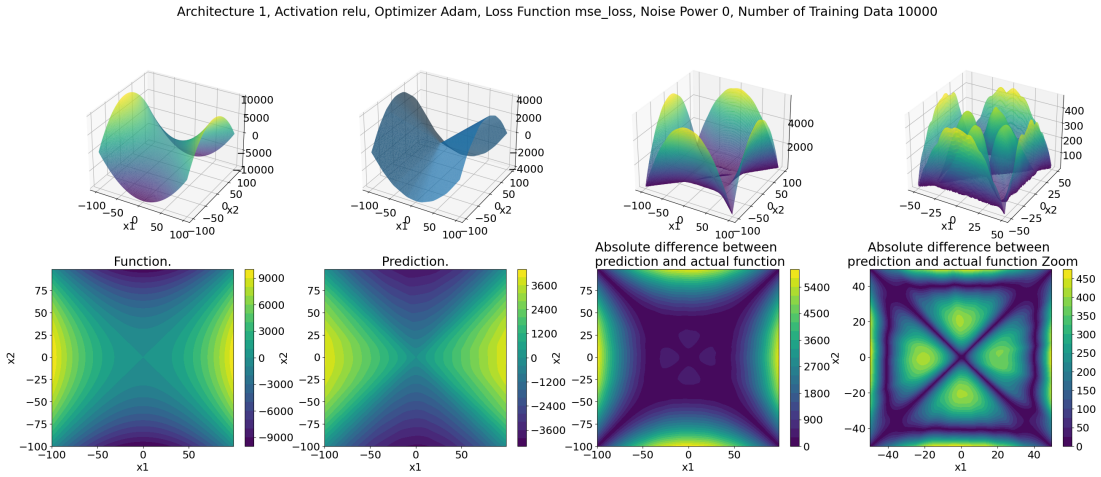

Figure 2.12: Training of SADDLE function with 10000 Data/ MSE loss / relu activation

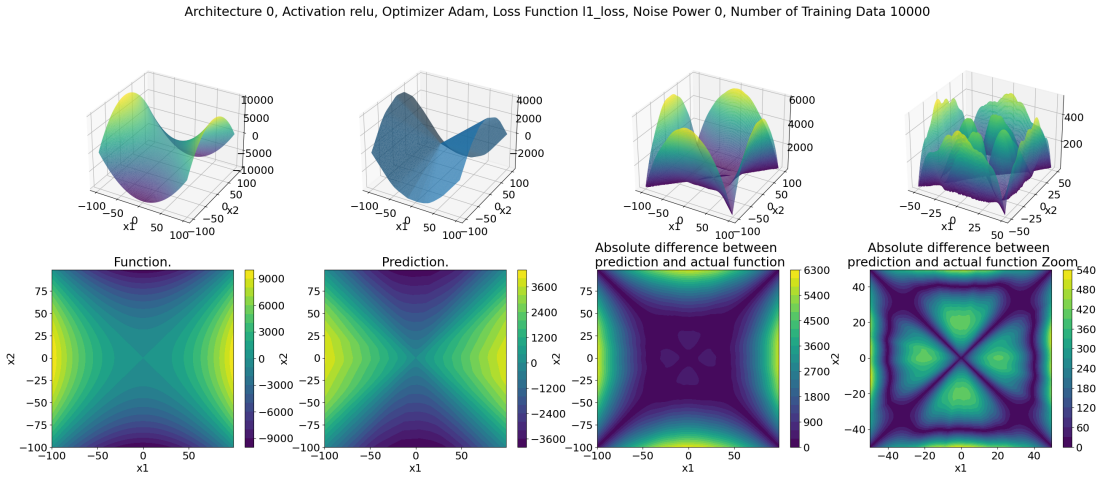

Figure 2.13: Training of SADDLE function with 10000 Data/ L1 loss / relu activation

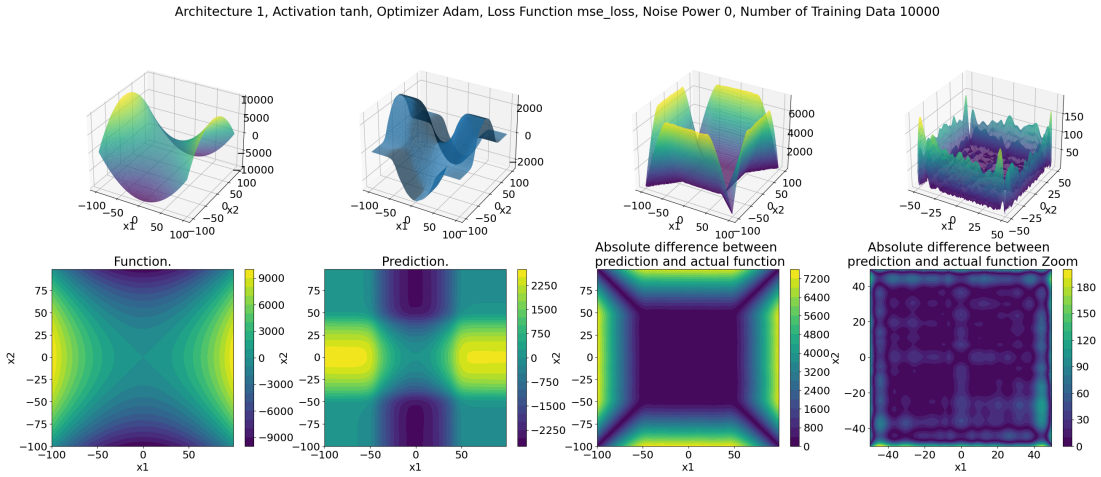

Figure 2.14: Training of SADDLE function with 10000 Data/ L1 loss / tanh activation

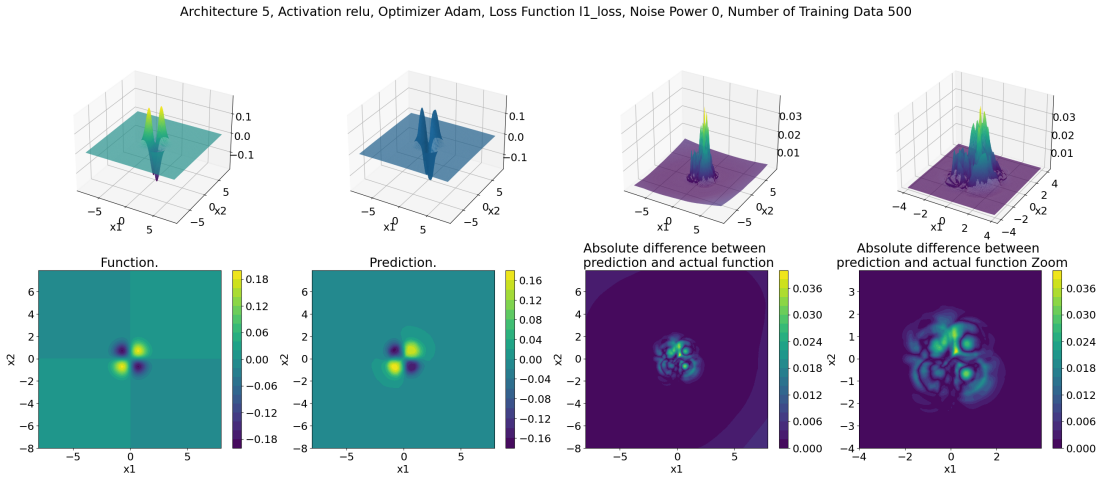

Figure 2.15: Training of EXP 2D function with 500 Data / L1 loss / relu activation

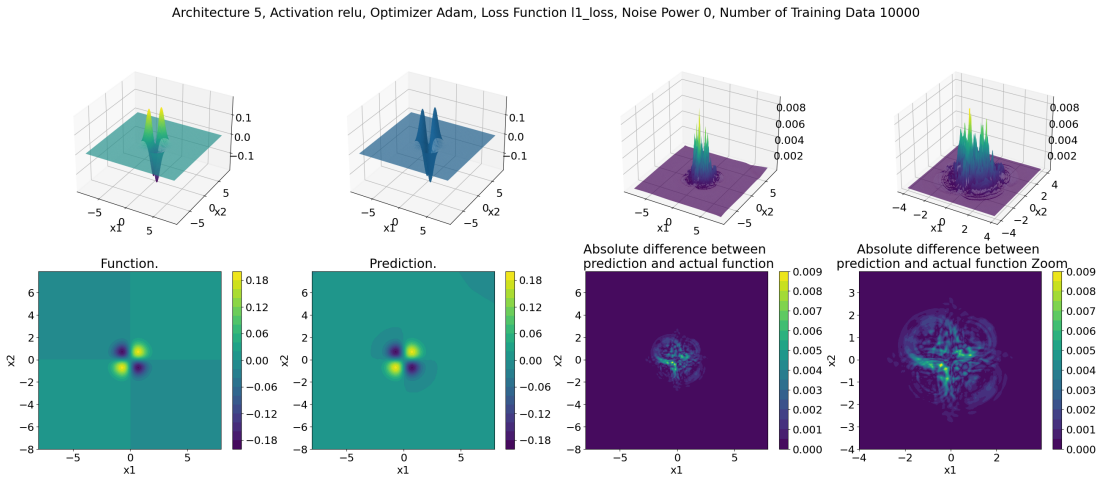

Figure 2.16: Training of EXP 2D function with 10000 Data / L1 loss / relu activation

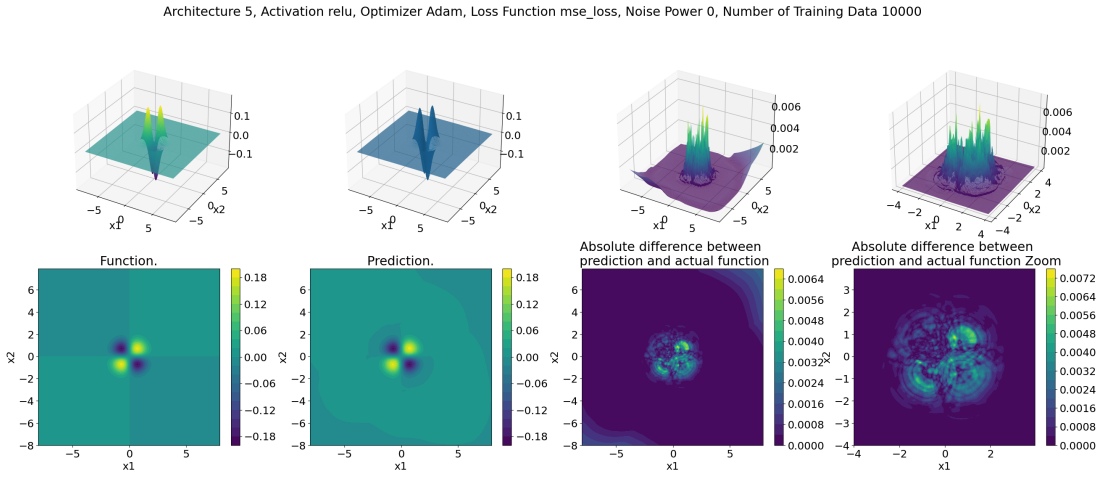

Figure 2.17: Training of EXP 2D function with 10000 Data / MSE loss / relu activation

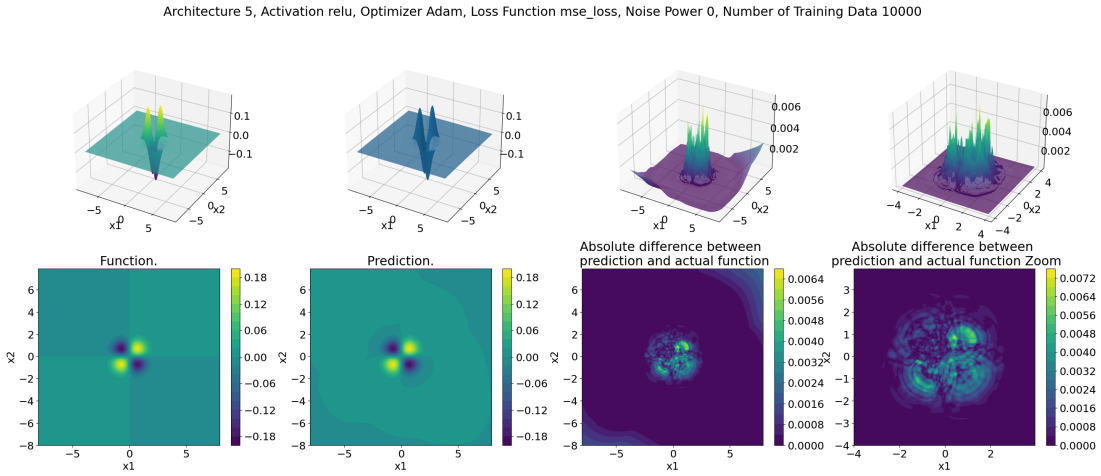

Figure 2.18: Training of EXP 2D function with 10000 Data / MSE loss / sigmoid activation

Appendix 2

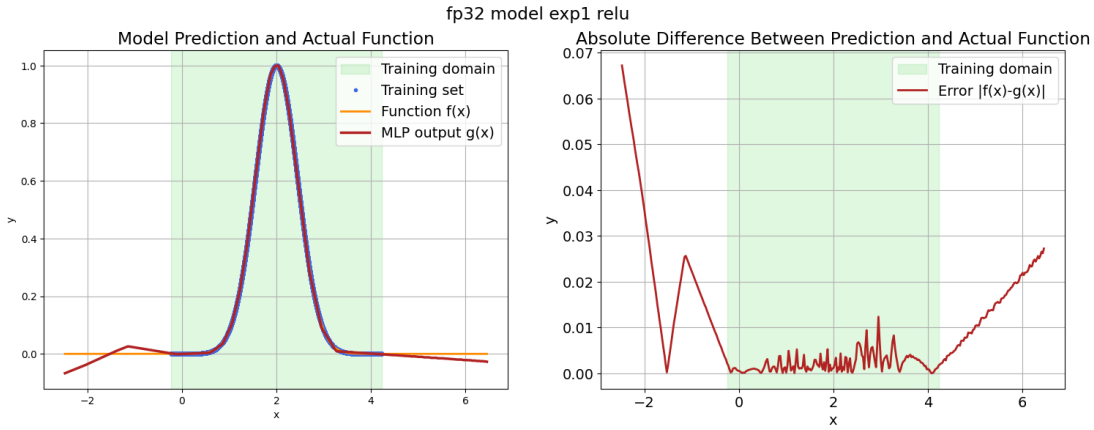

Figure 2.19: Training of EXP function with 50000 Data / relu activation / 3 layers of 8 neurones

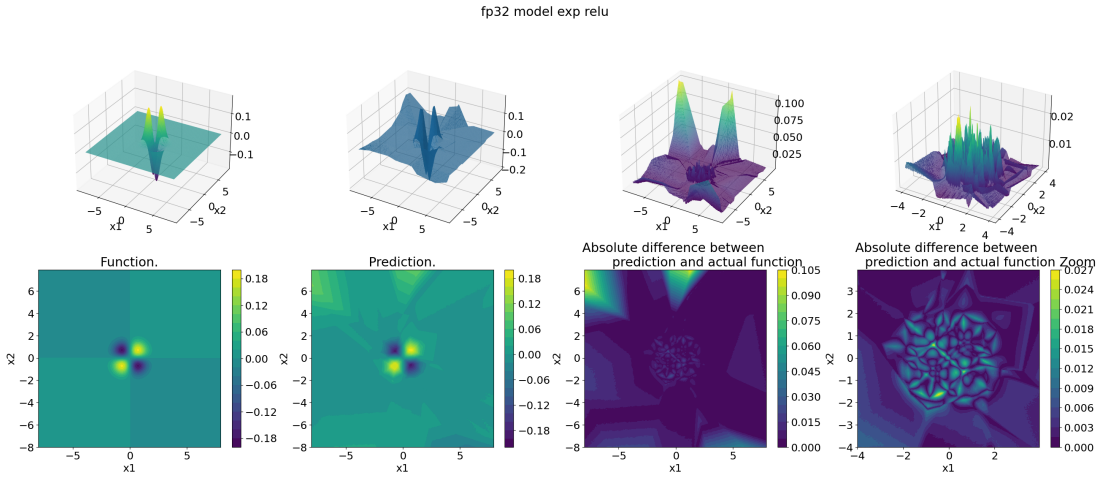

Figure 2.20: Training of EXP 2D function with 50000 Data / relu activation / 3 layers of 8 neurones

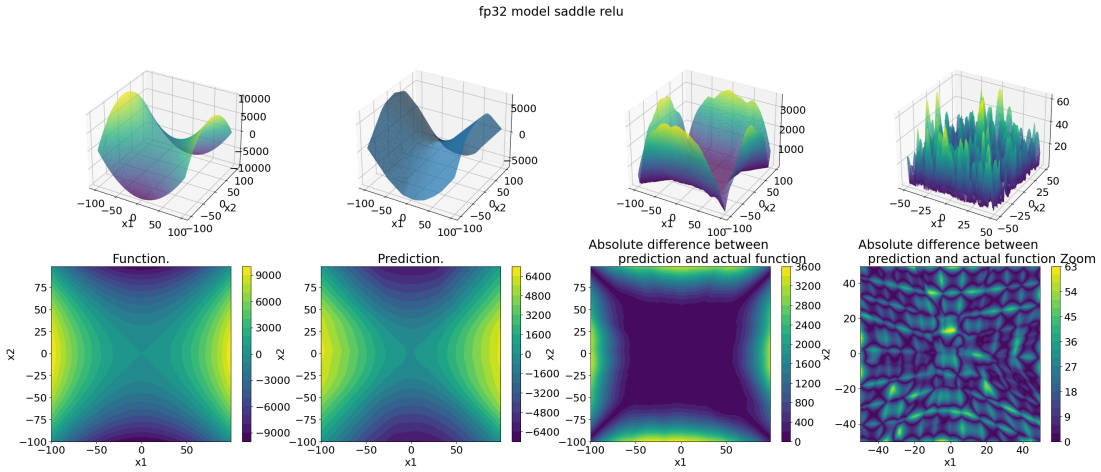

Figure 2.21: Training of SADDLE function with 50000 Data / relu activation / 3 layers of 8 neurones

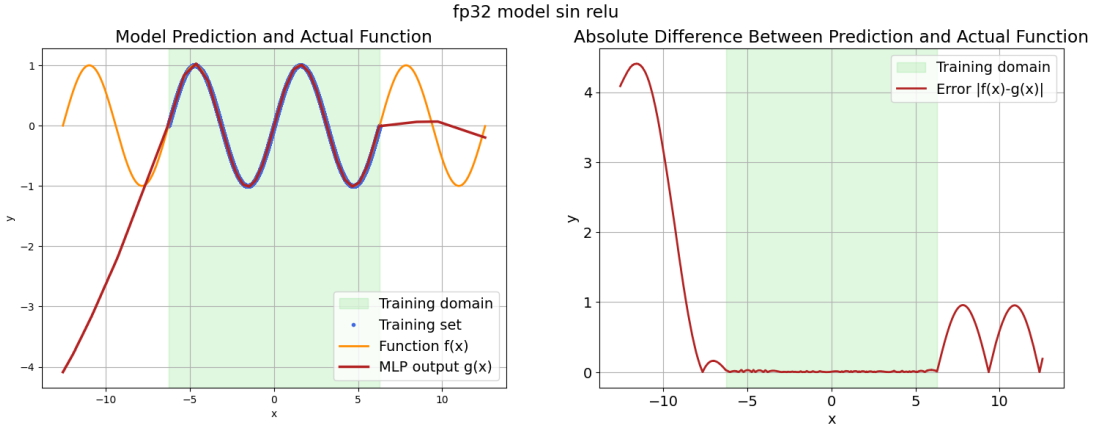

Figure 2.22: Training of SIN function with 50000 Data / relu activation / 3 layers of 8 neurones

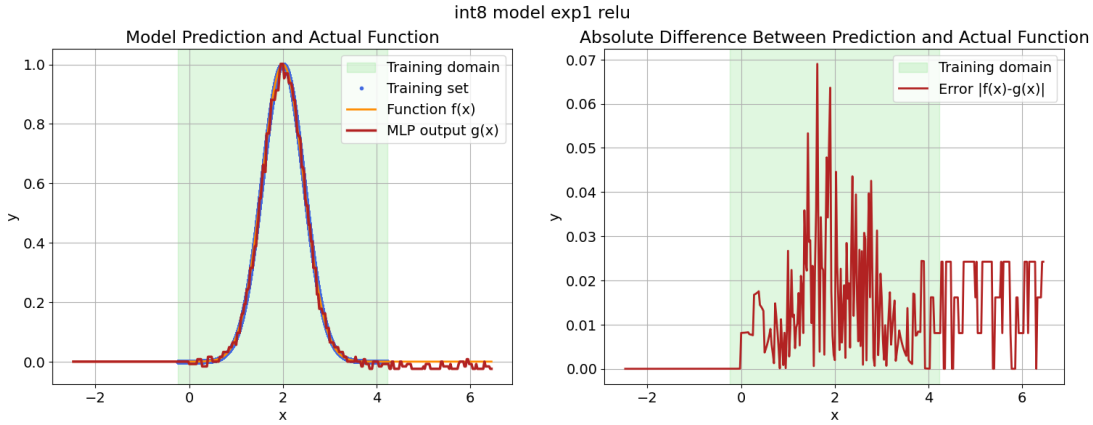

Figure 2.23: Training of EXP function with 50000 Data / relu activation / 3 layers of 8 neurones with QAT quantization

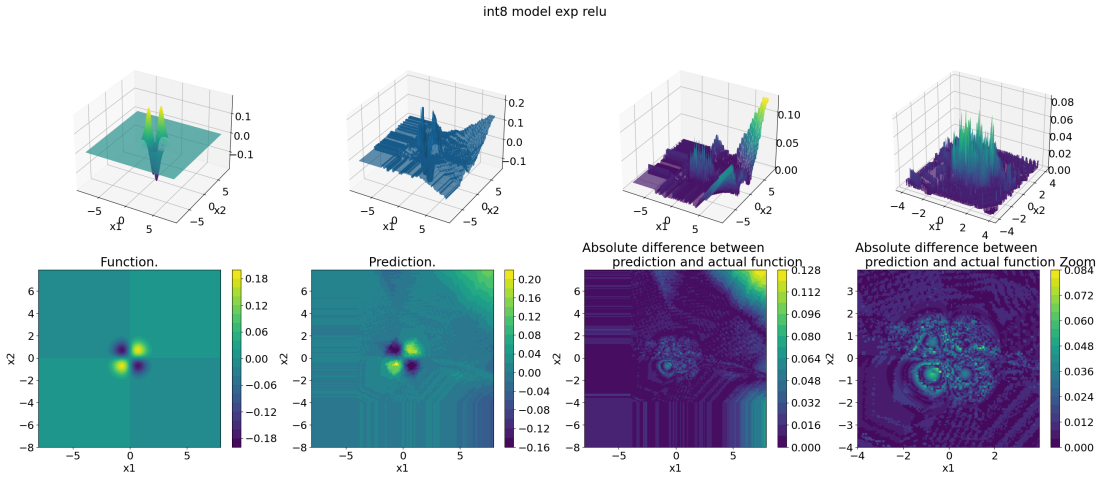

Figure 2.24: Training of EXP 2D function with 50000 Data / relu activation / 3 layers of 8 neurones with QAT quantization

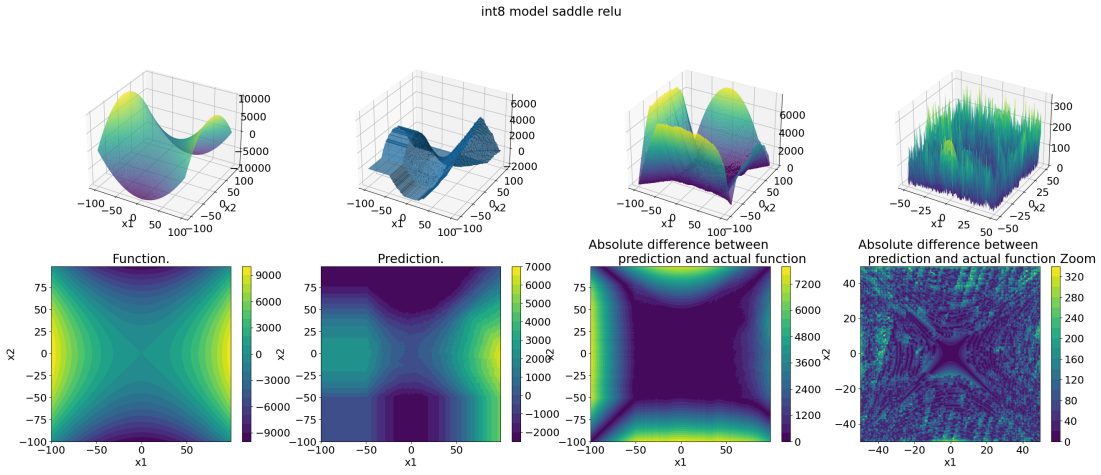

Figure 2.25: Training of SADDLE function with 50000 Data / relu activation / 3 layers of 8 neurones with QAT quantization

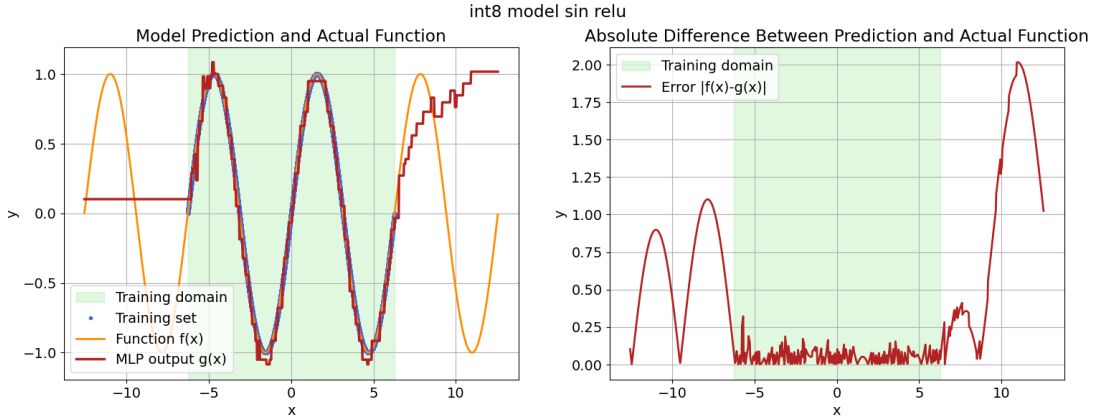

Figure 2.26: Training of SIN function with 50000 Data / relu activation / 3 layers of 8 neurones with QAT quantization

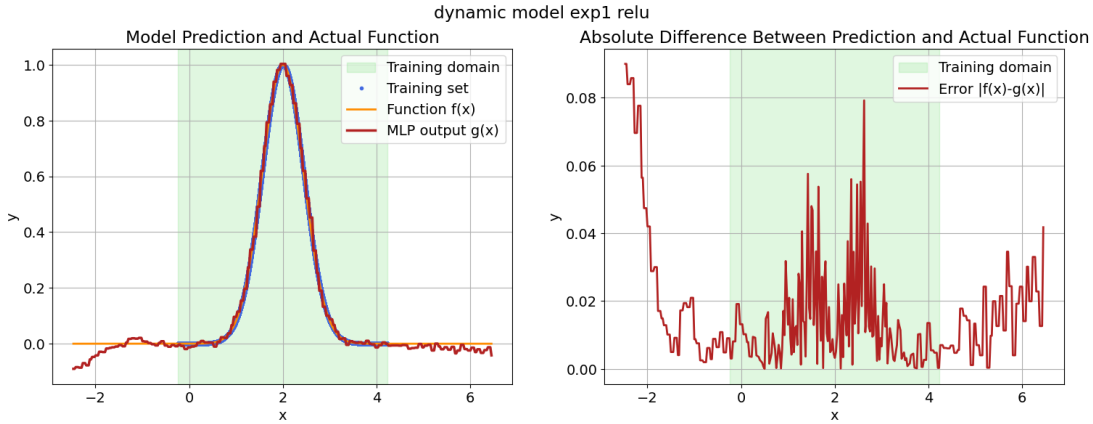

Figure 2.27: Training of EXP function with 50000 Data / relu activation / 3 layers of 8 neurones with dynamic quantization

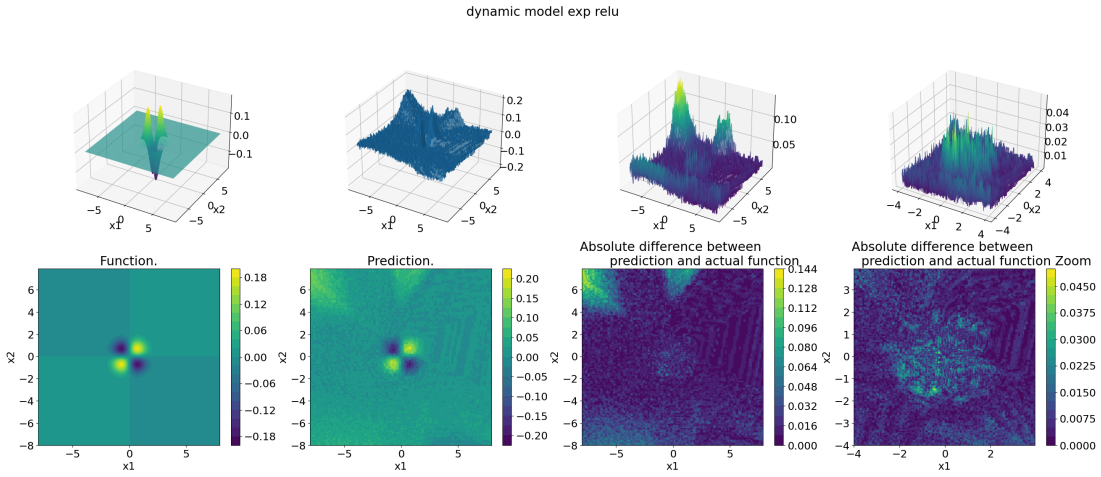

Figure 2.28: Training of EXP 2D function with 50000 Data / relu activation / 3 layers of 8 neurones with dynamic quantization

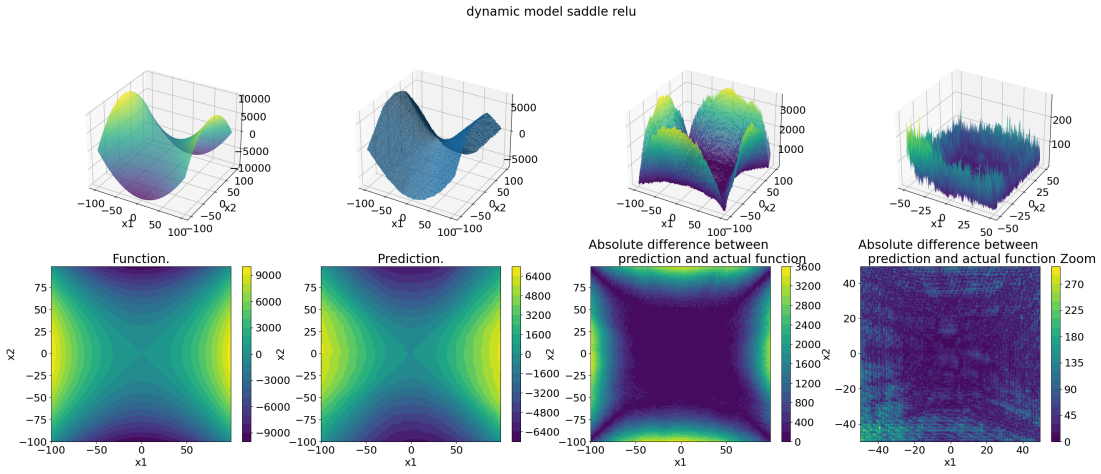

Figure 2.29: Training of SADDLE function with 50000 Data / relu activation / 3 layers of 8 neurones with dynamic quantization

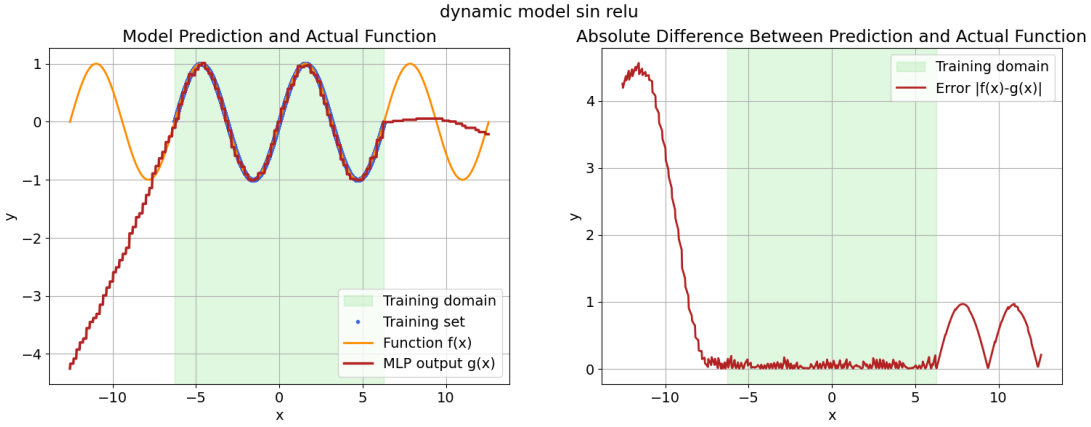

Figure 2.30: Training of SIN function with 50000 Data / relu activation / 3 layers of 8 neurones with dynamic quantization

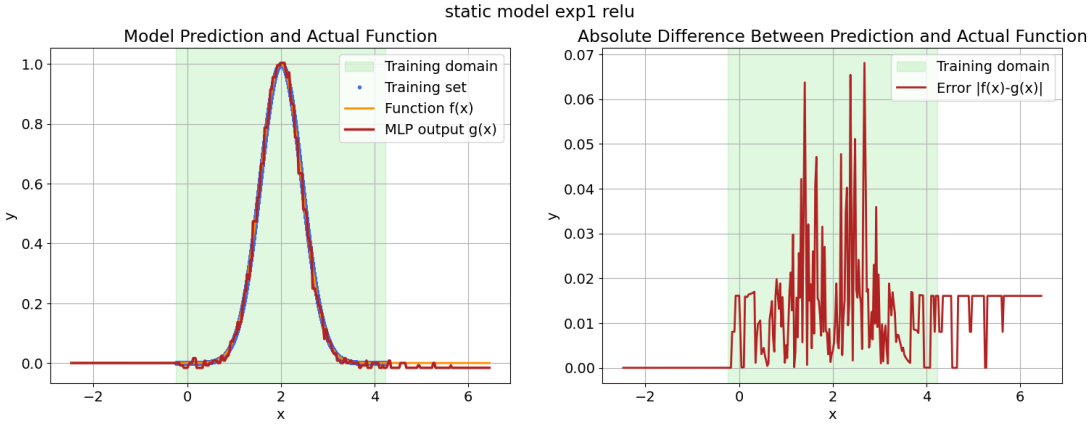

Figure 2.31: Training of EXP function with 50000 Data / relu activation / 3 layers of 8 neurones with static quantization

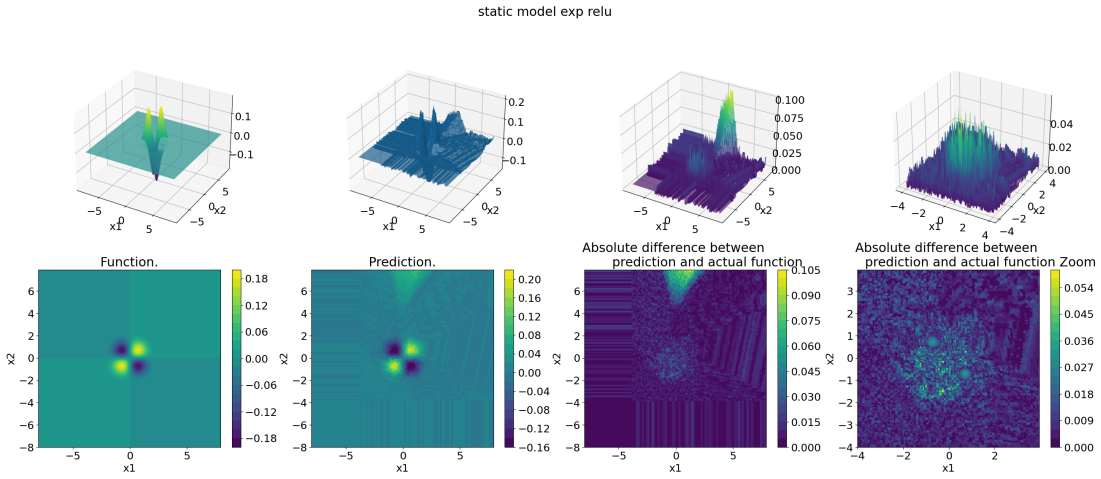

Figure 2.32: Training of EXP 2D function with 50000 Data / relu activation / 3 layers of 8 neurones with static quantization

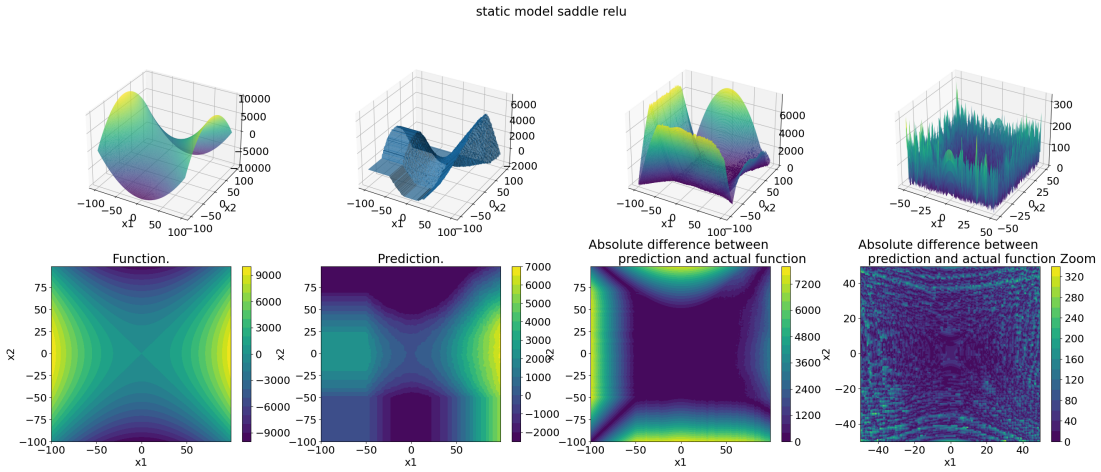

Figure 2.33: Training of SADDLE function with 50000 Data / relu activation / 3 layers of 8 neurones with static quantization

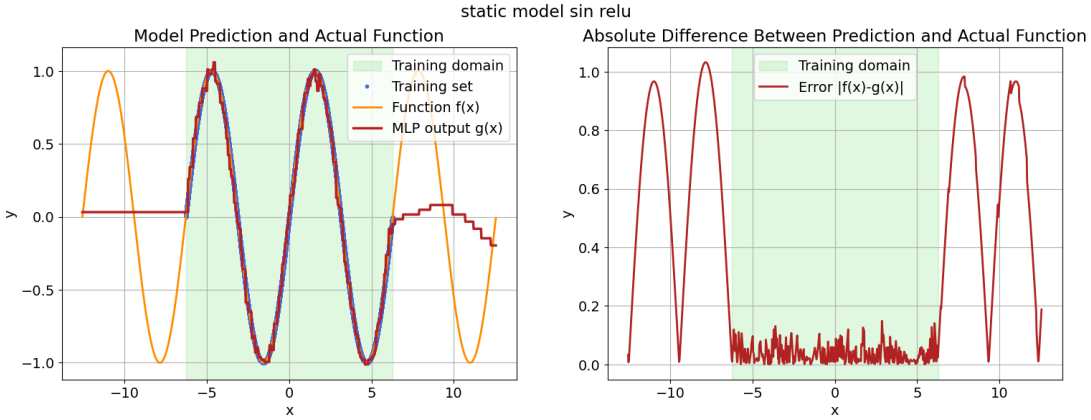

Figure 2.34: Training of SIN function with 50000 Data / relu activation / 3 layers of 8 neurones with static quantization

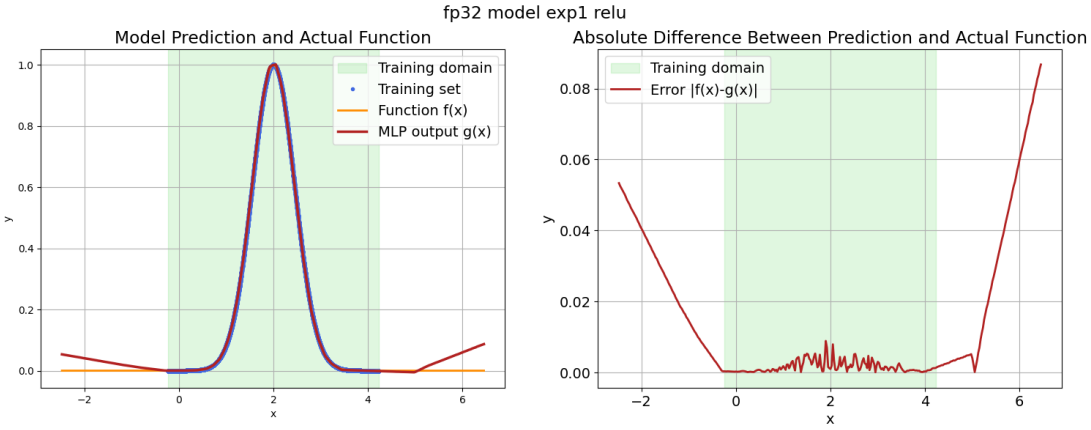

Figure 2.35: Training of EXP function with 10000 Data / relu activation / 5 layers of 8 neurones

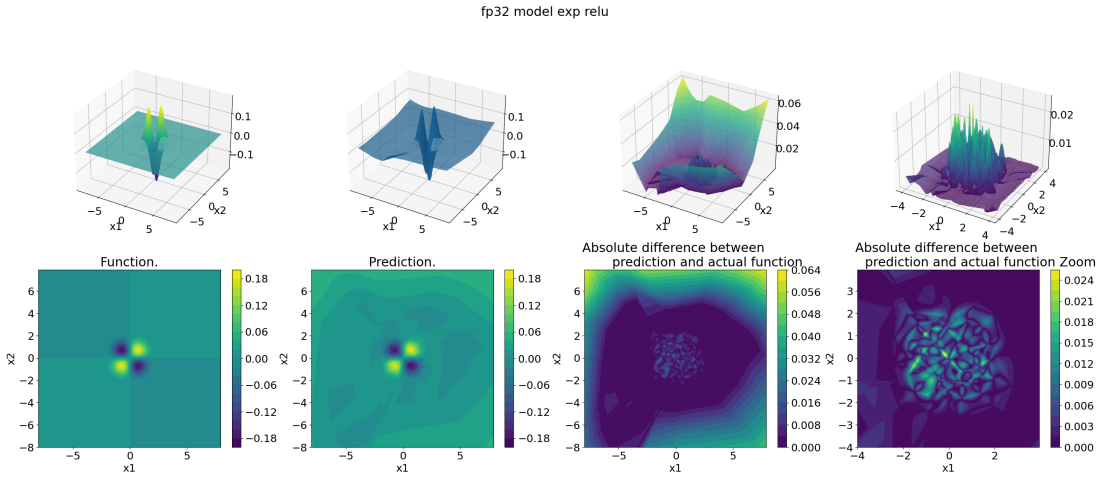

Figure 2.36: Training of EXP 2D function with 10000 Data / relu activation / 5 layers of 8 neurones

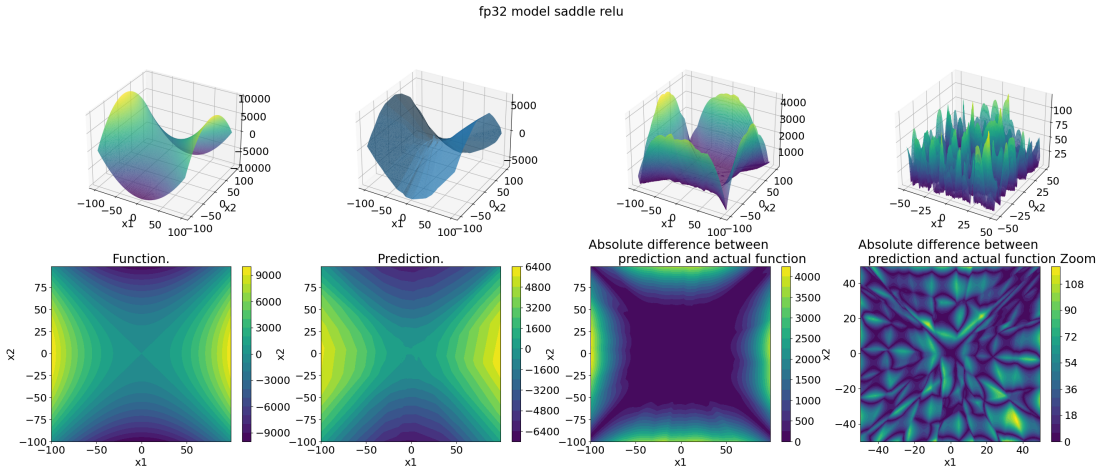

Figure 2.37: Training of SADDLE function with 10000 Data / relu activation / 5 layers of 8 neurones

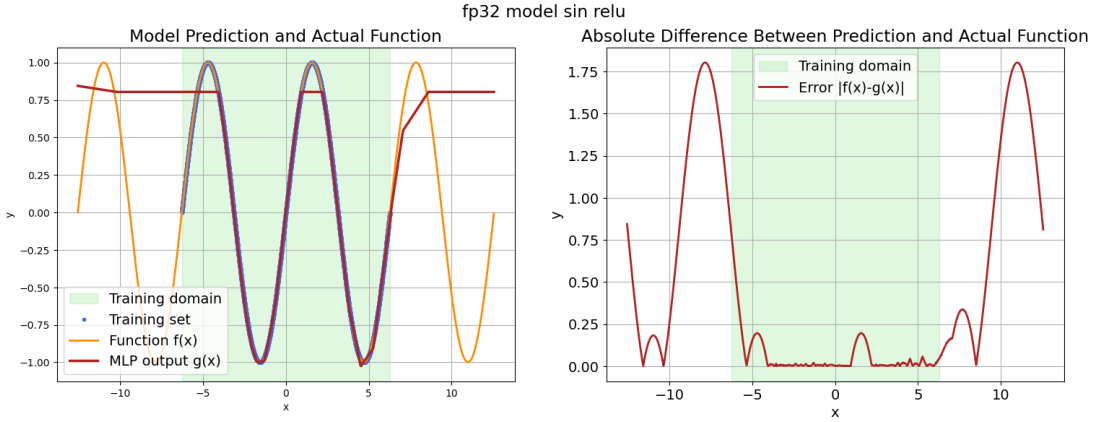

Figure 2.38: Training of SIN function with 10000 Data / relu activation / 5 layers of 8 neurones

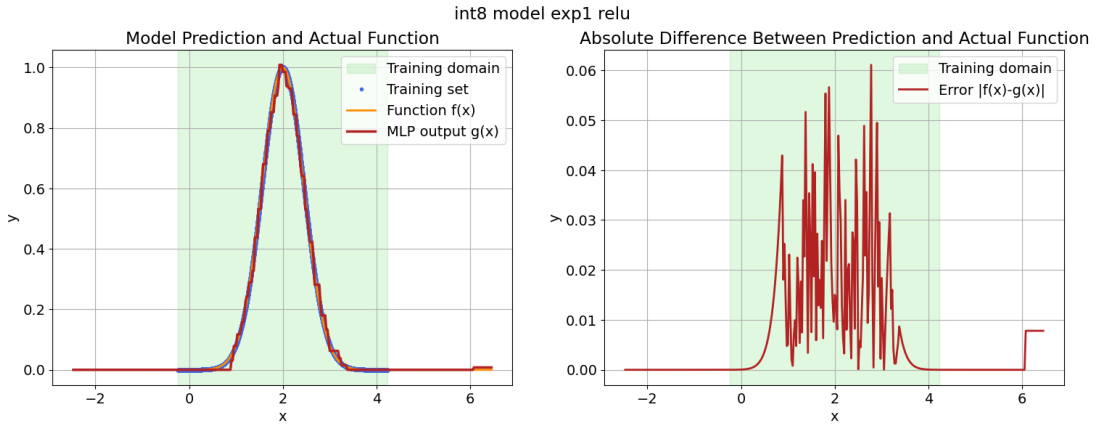

Figure 2.39: Training of EXP function with 10000 Data / relu activation / 5 layers of 8 neurones with QAT quantization

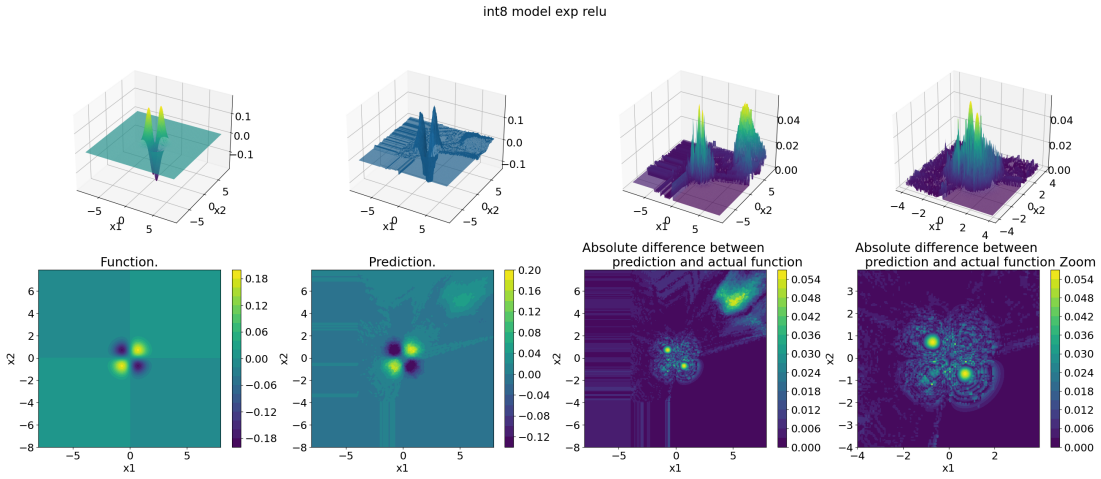

Figure 2.40: Training of EXP 2D function with 10000 Data / relu activation / 5 layers of 8 neurones with QAT quantization

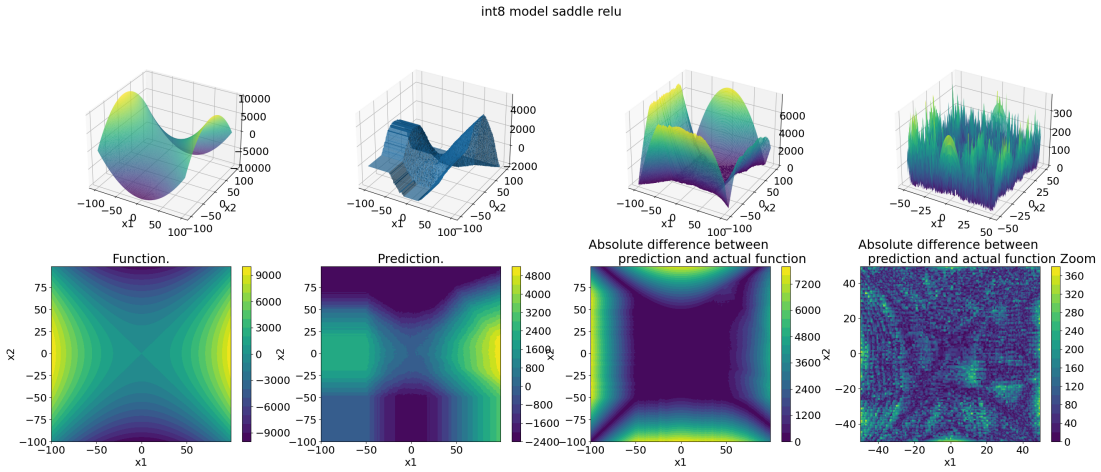

Figure 2.41: Training of SADDLE function with 10000 Data / relu activation / 5 layers of 8 neurones with QAT quantization

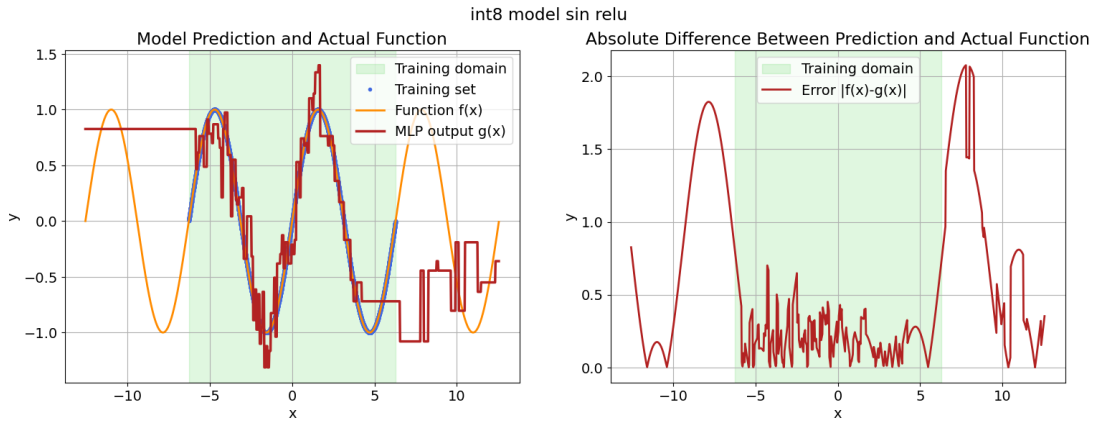

Figure 2.42: Training of SIN function with 10000 Data / relu activation / 5 layers of 8 neurones with QAT quantization

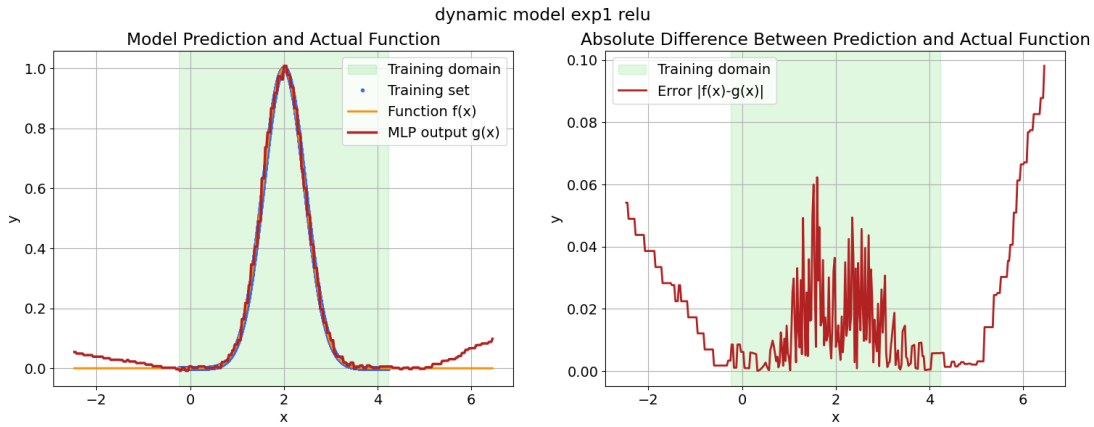

Figure 2.43: Training of EXP function with 10000 Data / relu activation / 5 layers of 8 neurones with dynamic quantization

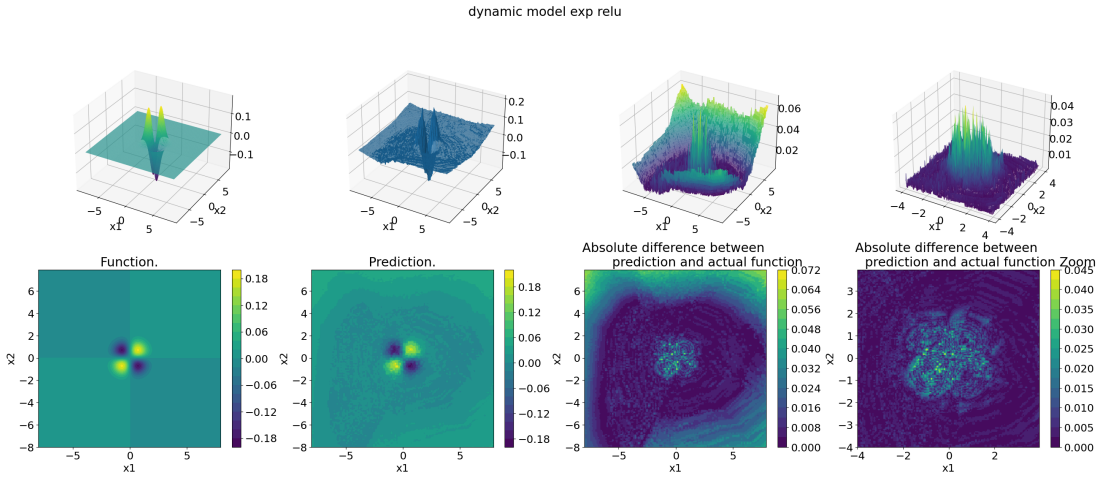

Figure 2.44: Training of EXP 2D function with 10000 Data / relu activation / 5 layers of 8 neurones with dynamic quantization

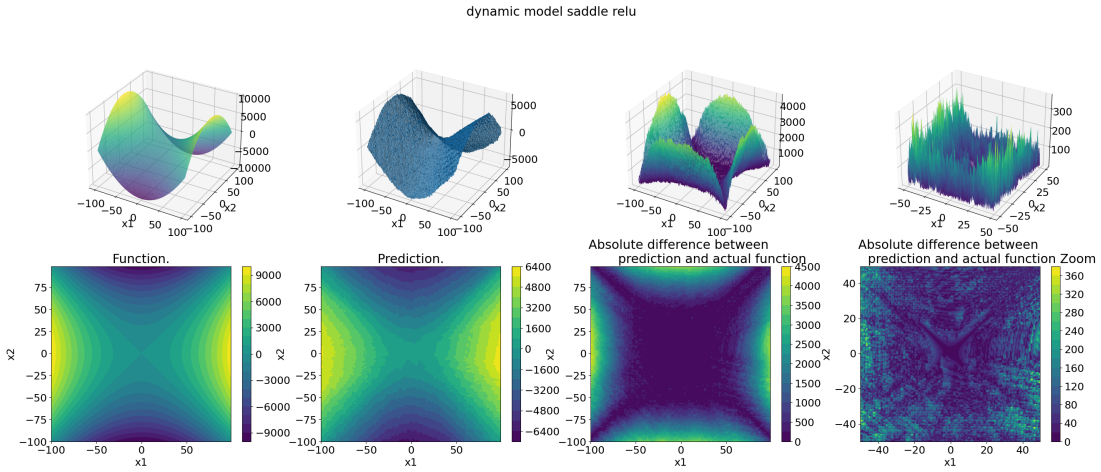

Figure 2.45: Training of SADDLE function with 10000 Data / relu activation / 5 layers of 8 neurones with dynamic quantization

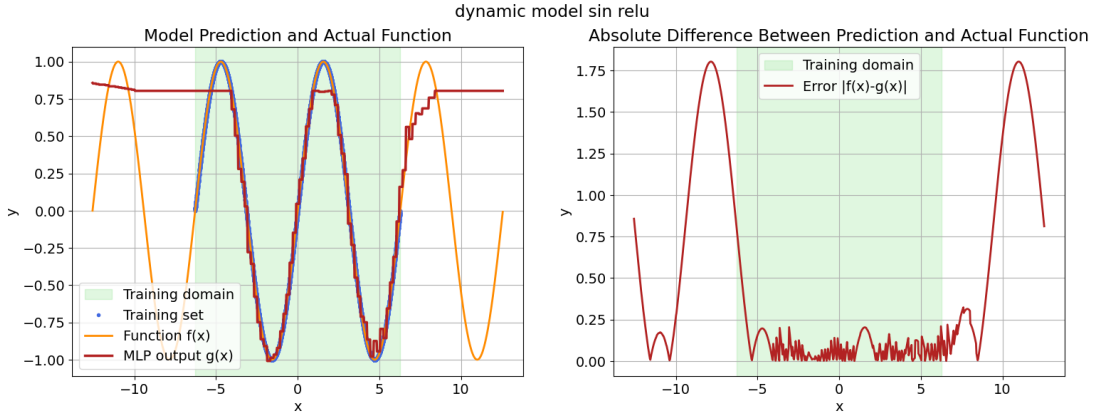

Figure 2.46: Training of SIN function with 10000 Data / relu activation / 5 layers of 8 neurones with dynamic quantization

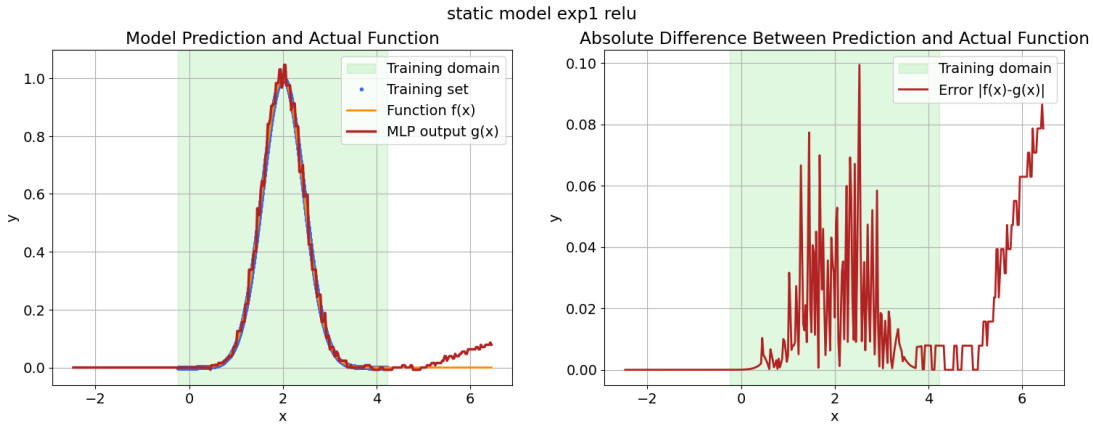

Figure 2.47: Training of EXP function with 10000 Data / relu activation / 5 layers of 8 neurones with static quantization

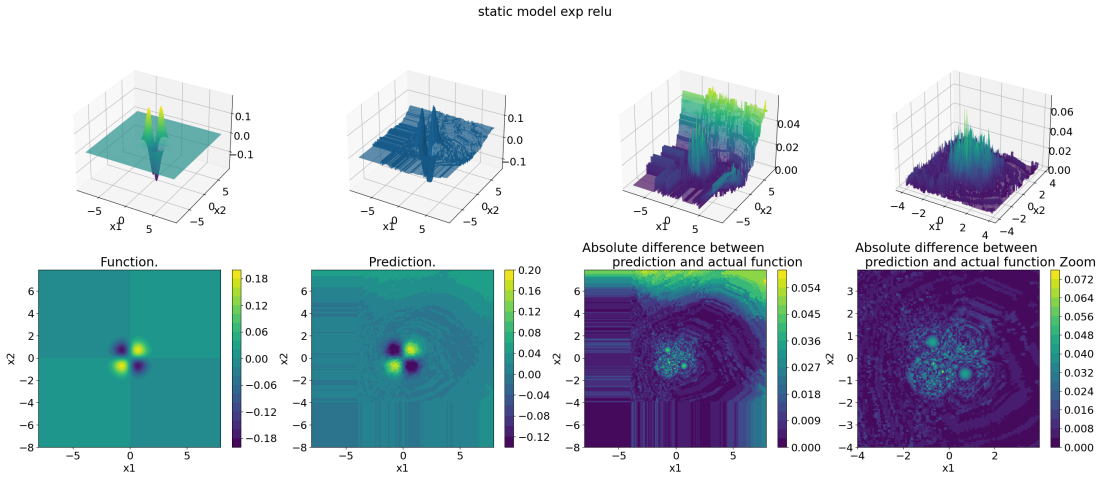

Figure 2.48: Training of EXP 2D function with 10000 Data / relu activation / 5 layers of 8 neurones with static quantization

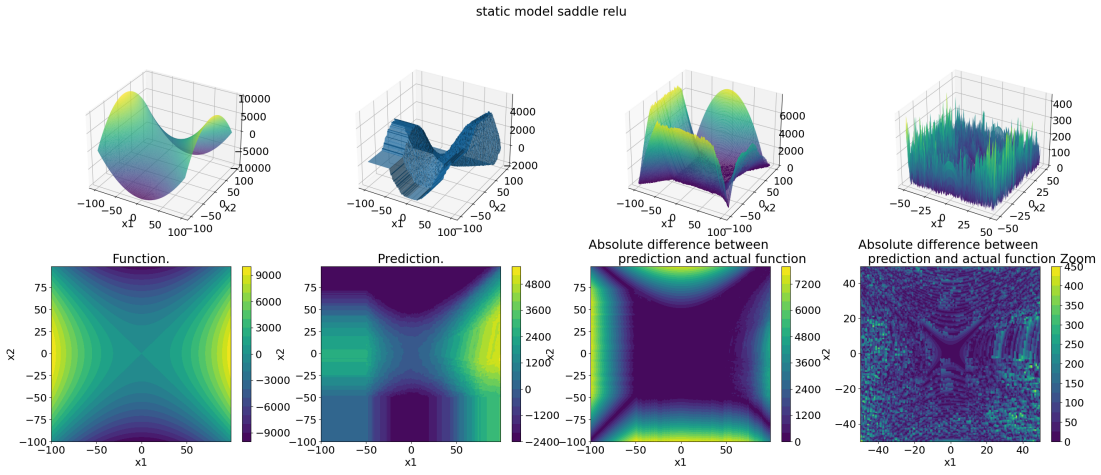

Figure 2.49: Training of SADDLE function with 10000 Data / relu activation / 5 layers of 8 neurones with static quantization

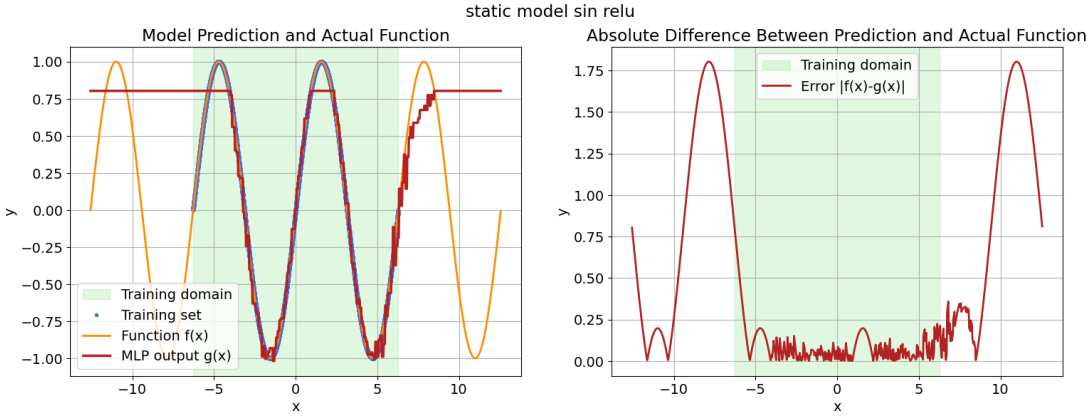

Figure 2.50: Training of SIN function with 10000 Data / relu activation / 5 layers of 8 neurones with static quantization