An Exploration of Neural Network Expressivity and Approximation Dynamics

Dylan Hu

May 11, 2023

Contents

1 Introduction

In this report, we explore the expressivity and approximation dynamics of neural networks. In particular, we aim to investigate the relationship between a handful of specific architectural decisions when building multi-layer perceptrons and relate these findings to the theoretical bases of expressivity

and universal approximation. Furthermore, we explore the effects of additional techniques, such as positional encoding, to facilitate more efficient approximations of complex functions.

The expressivity of a neural network refers to the diversity and complexity of functions it can represent. This complexity is primarily determined by the architecture of the network, especially its depth and width. The depth of a neural network, represented by the number of layers it possesses, and the width, indicated by the number of neurons in each layer, play crucial roles in determining the network's expressive power. By manipulating these parameters, one can significantly alter the kind of functions a network can efficiently approximate.

One particularly fascinating notion with these types of neural networks is their universal approximation property, as posited in proofs by Cybenko and Hanin and Sellke, among others. According to Cybenko, a sufficiently wide feed-forward neural network with a single hidden layer can approximate any continuous function to an arbitrary degree of accuracy, given that the network's activation function is a non-constant, bounded, and monotonically-increasing continuous function. This theorem posits that neural networks are universally capable of learning and representing a broad array of functions, given enough neurons in the hidden layer. In practice, however, we often find that deeper networks with multiple layers can learn more efficiently and generalize better to unseen data.

In this project, we put these theoretical assertions to the practical test. We explore how varying the depth and width of a neural network affects its ability to approximate given functions. We specifically focus on a class of functions that are generated using random Fourier series, which are increasingly oscillatory, mirroring real-world functions that neural networks often grapple with. By observing how these networks learn to fit these functions over time, we gain insights into the dynamics of the training process.

In addition, we touch upon the concept of positional encoding, a technique often used in various types of applications including language modeling, where it helps to provide the model with a sense of order of the input data. Positional encoding is especially beneficial for high-frequency fits, enabling the model to handle complex, oscillatory functions more effectively. By incorporating positional encoding in our neural networks, we demonstrate enhanced approximation efficiency.

As we delve into these intriguing aspects of neural networks, we keep our focus on understanding the interplay of theory and practice in neural network design and training. Through this project, we strive to bridge the gap between what is theoretically possible and what is practically achievable, shedding light on how to more efficiently and effectively leverage the power of neural networks in solving complex problems.

2 Method

We consider the task of approximating a function given by a set of values on a regular grid of values $0 = x_0, \ldots, x_M = 1$. As such, the function to approximate takes the form $f : [0, 1] \to \mathbb{R}$, and we refer to our model's approximation as $g : [0, 1] \to \mathbb{R}$. We consider the following class of functions:

$$f(x) = \sum_{i=1}^{N} a_i \sin(2\pi i x) + b_i \cos(2\pi i x) \tag{1}$$

where a_i, b_i are randomly chosen. This class of functions is generated using random Fourier series, which are increasingly oscillatory and mirror real-world functions that neural networks often grapple with. We use this class of functions to explore the expressivity and approximation dynamics of neural networks.

We implement our experiments in PyTorch 2. Unless otherwise specified, we use the Adam optimizer with a learning rate of 0.001 and train for 1000 epochs.

2.1 Neural Network Architecture

This project utilizes a multi-layer perceptron (MLP), a type of feed-forward deep neural network. The MLP is composed of multiple layers, each containing a number of neurons or nodes, and utilizes an activation function for introducing non-linearities.

The network is characterized by a depth, N, indicating the total number of layers, and widths, represented by $d_0, d_1, ..., d_N$. The conditions $d_0 = d_N = 1$ are imposed to denote a single neuron in the input and output layers, a typical configuration for function approximation tasks. The widths of the hidden layers, $d_1, ..., d_{N-1}$, are varied throughout the project.

The MLP follows a fully-connected architecture, where each neuron in a layer is connected to all neurons in the following layer. Various activation functions, including sigmoid, ELU (Exponential Linear Unit), and ReLU (Rectified Linear Unit), are applied to add non-linear transformations at each neuron.

2.2 Optimization Objective

The parameters of the network, including the weights and biases (collectively referred to as w), are learned during the training process. The goal is to minimize the difference between the network's predictions and the actual function values by iteratively adjusting these parameters through gradient descent in order to minimize a mean squared error (MSE) loss function. The loss function is defined as follows:

$$L(g) = \frac{1}{M} \sum_{j=0}^{M} (g(x_j; w) - f(x_j))^2 \tag{2}$$

2.3 Investigations

2.3.1 Time Dynamics

We study the time dynamics of the training process by observing how the network's approximation changes over time for increasingly oscillatory functions. In this initial investigation, we fix the network size at a depth of 3 and a width of 16 for each hidden layer. We train the network for a fixed number of epochs and continuously plot the network's approximation at various points in time in comparison to the ground truth. We also plot the loss over time to observe how the loss changes as the network trains.

2.3.2 Expressivity

We explore the expressivity of the network by varying both the width of the hidden layers and the depth of the network and observe how the character and efficiency of the network's approximation changes. We train the network for a fixed number of epochs and plot the network's approximation at various widths and depths. We also plot the loss over time to observe how the loss changes as the network trains.

In particular, in investigating width, we fix $N = 2$ and choose $d = 32, 64, 128$.

In investigating depth, we fix $d = 2$ and choose $N = 4, 16, 64$.

We also explore the dynamics for combinations of N and d such that the total parameter count is the same (that is, parameter count $p = N \times d^2$ is constant). We experiment with $p = 512$ using the combinations $N = 8, d = 8$ and $N = 2, d = 16$.

2.3.3 Positional Encoding

We explore the effects of sine-cosine positional encoding. Before sending the input x into the network, we modulate it with a series of sine and cosine functions. Specifically:

$$x \to \begin{bmatrix} \sin(2\pi x) \\ \cos(2\pi x) \\ \sin(4\pi x) \\ \cos(4\pi x) \\ \vdots \\ \sin(2^k \pi x) \\ \cos(2^k \pi x) \end{bmatrix} \tag{3}$$

where k is a hyperparameter set to 6. We fix the size of the network and observe how the network's approximation changes with and without positional encoding. We also plot the loss over time to observe how the loss changes as the network trains.

3 Results

In this section, we present in prose analyses corresponding to each of the investigations outlined above. As much of the analysis concerns the qualitative characteristics of the approximation dynamics, we find that animated visualizations are much more salient. As such, we refer the reader to the GitHub repository which includes the animated visualizations and an accompanying web page for easy viewing. Instructions for viewing the visualizations are included in the README file.

3.1 Time Dynamics

We observe that from an initialization of near-zero random weights which result in near-zero initial approximation values across the domain of input,

the model consistently begins training by elevating or lower the very horizontal approximation to quickly match the average value (height) of the target function. In cases where the function's value changes more significantly from one side of the domain (x close to 0) to the other side (x close to 1), we notice that the model tilts the approximation to match this trend as it changes the height of the approximation.

We note that in all cases the early approximation stays very linear and does not bend significantly during this stage, regardless of the amplitude of low-frequency oscillation. That is, despite a target function having large hills and valleys, the model first matches the average height and overall tilt while maintaining a linear approximation.

After this early stage which generally converges within the first 150 epochs, we observe that the approximation gradually begins to learn non-linear, oscillatory behavior. Improvement past the early stage is much slower, as evidenced by the plot of the loss over time.

3.2 Expressivity

3.2.1 Width

Given a constrained model depth of $N = 2$, we observe that increasing the width of the hidden layer assists the model in learning more complex functions; the model with greater width exhibits a tighter fit. A particularly salient observation is that with greater width, the early phase in which the model learns the average height and tilt of the function is much shorter, the approximation snapping into place immediately.

3.2.2 Depth

We observe interesting results constraining width to $d = 2$ and varying depth. We notice that for small depth like $N = 4$, the model learns smooth, fairly linear approximations. As we increase the depth to 16, we notice that it now begins to attempt to learn sharper features, but it is not able to represent many large jumps. With a depth of 64, the model seems to only be able to translate its initial approximation to match the average value.

3.2.3 Equiparameter

We notice in comparing the two cases that $N = 8, d = 8$ outperforms $N = 2, d = 16$, with the former reaching a loss of 0.2400 and the latter reaching 0.3178.

3.3 Positional Encoding

Clearly, the addition of positional encoding allows the model to learn more complex functions, especially demonstrating its effectiveness in learning higher frequency fits. We also notice that as a result of the sine-cosine positional encoding that the approximation begins as a sinusoidal wave, but also exhibits the same early stage behavior as the traditional model, in which it learns the average height and tilt of the function before learning the oscillatory behavior. Indeed, we still observe refinement into higher frequencies later in the training process, but the model is almost instantly able to learn the broad features along with medium-frequency oscillations.

4 Discussion

Through our investigations, we have observed that these types of MLPs, despite having non-linear activation functions, require scale in order to learn higher frequency fits. Without positional encoding, no combination of hyperparameters was able to capture any high-frequency detail.

We do observe that a balance of width and depth is highly beneficial to model performance; with width or depth highly constrained, we witness poor approximations. Interestingly, in the case of $d = 2$ and large width mimicing the theory of Cybenko, we actually observe a sort of collapse in which the model only represents a constant linear function. The intuition behind this behavior is unknown and warrants further investigation, but this at least demonstrates that despite the theoretical guarantees of Cybenko, the practical performance of MLPs is not guaranteed and is heavily reliant on choice of architecture.

In introducing positional encoding, we demonstrate a solution to a rather intuitive problem: that MLPs are not able to learn high-frequency fits. This behavior of traditional MLPs is well-documented and is often referred to as "spectral bias" [2] – the network is biased towards learning lower-frequency parts of the spectrum before higher-frequency parts, as learning broad-stroke

features leads to greater decreases in the loss. By introducing positional encoding, we are able to elevate our input into a higher-dimensional frequency space, and the network is encouraged towards higher frequencies.

Outside of language modeling, positional encoding is often used in reconstruction of spatial information, such as image reconstruction [3] and scene representation. For instance, the use of positional encoding in neural radiance fields [1] is essential in allowing the model to capture the fine detail in 3D scenes.

Future investigation may include the same nature of experiments with larger networks.

References

[1] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis, 2020.

[2] Nasim Rahaman, Aristide Baratin, Devansh Arpit, Felix Draxler, Min Lin, Fred A. Hamprecht, Yoshua Bengio, and Aaron Courville. On the spectral bias of neural networks, 2019.

[3] Matthew Tancik, Pratul P. Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Nithin Raghavan, Utkarsh Singhal, Ravi Ramamoorthi, Jonathan T. Barron, and Ren Ng. Fourier features let networks learn high frequency functions in low dimensional domains, 2020.