MA 591 - HOMEWORK 02

Hayden Outlaw

NCSU Applied Mathematics, 03 Oct 2025

---

1 Representative Density of Exponential Neural Networks

Let $C(X;\mathbb{R})$ be the set of continuous functions on a compact set $X \subset \mathbb{R}^d$. Let $\mathcal{NN}^\phi$ be the collection of two-layer neural networks with activation function ϕ. Let $\phi(x) = e^x$. We can show that $\mathcal{NN}$ is dense in $C(X;\mathbb{R})$.

We can express $\mathcal{NN}^\phi$ (of any width, depth 2) as:

$$\mathcal{NN}^\phi = \text{span}\left\{\sum_{i=1}^{k} c_i\phi(w_i^\top x + b_i) + c_0 : c_i, b_i \in \mathbb{R}, w_i \in \mathbb{R}^d, n \in \mathbb{Z}\right\}$$

We use the real valued Stone- Weierstrass Approximation theorem, which states for a compact Hausdorff space X, and L a subalgebra of $C(X,\mathbb{R})$ which contains a nonzero constant function, L is dense in $C(X,\mathbb{R})$ if and only if it separates points.

First, $\mathcal{NN}^\phi$ clearly contains a nonzero constant function. For example if $c_i = 0, i = 1,\dots,n$, and $c_0 = 1$, the corresponding network $f \in \mathcal{NN}^\phi$ is the constant function $f(x) = 1$.

Next, we show $\mathcal{NN}^\phi$ is a subalgebra of $C(X,\mathbb{R})$. By the continuity of $\phi(x) = e^x$ and matrix-vector addition and multiplication, we have $\mathcal{NN}^\phi \subset C(X,\mathbb{R})$. Additionally, X is defined to be compact, and inherits Hausdorff properties from $\mathbb{R}^d$. Furthermore, for $f \in \mathcal{NN}^\phi$ of width m, $g \in \mathcal{NN}^\phi$ of depth n, since our model space includes models of any arbitrary width, we clearly have $(f+g)$ is a model of width $(m+n-1)$ that is also a member of $\mathcal{NN}^\phi$.
To show multiplicative closure, define two models:

$$f(x) = \sum_{i=1}^{m} c_i^{(0)}\phi\left(\left[w^{(0)}\right]_i^\top x + b_i^{(0)}\right) + c_0^{(0)}$$
$$g(x) = \sum_{j=1}^{m} c_j^{(1)}\phi\left(\left[w_j^{(1)}\right]^\top x + b_j^{(1)}\right) + c_0^{(1)}$$

Then,

$$\begin{aligned} fg(x) = &\sum_{ij} c_i^{(0)}c_j^{(1)}\phi\left(\left[w^{(0)}\right]_i^\top x + b_i^{(0)}\right)\phi\left(\left[w_j^{(1)}\right]^\top x + b_j^{(1)}\right) \\ &+ c_0^{(0)}\sum_{j=1}^{m} c_j^{(1)}\phi\left(\left[w_j^{(1)}\right]^\top x + b_j^{(1)}\right) + c_0^{(1)}\sum_{i=1}^{m} c_i^{(0)}\phi\left(\left[w^{(0)}\right]_i^\top x + b_i^{(0)}\right) + c_0^{(0)}c_0^{(1)} \end{aligned}$$

Because $\phi(x) = e^x$, $\phi(x)\phi(y) = \phi(x+y)$, so we can conclude $fg(x) \in \mathcal{NN}^\phi$ as well. Therefore, $\mathcal{NN}^\phi$ is a subalgebra of $C(X,\mathbb{R})$.

Finally, to show $\mathcal{NN}^\phi$ separates points, let p, q be distinct in X. Since $p \neq q$, there is an index i such that $p_i \neq q_i$. We can define the Lagrange basis polynomial $p(z) := \frac{z_i - q_i}{p_i - q_i}$. Consider the simplest case of a width 1 model:

$$f(x) = \phi\left(w^\top x + b\right)$$

If we define $w = \frac{z_i}{x_i - y_i}\boldsymbol{e}_i$, and $b = -\frac{y_i}{x_i - y_i}$, then:

$$\begin{aligned} f(z) &= \phi(w^\top z + b) \\ &= \phi(p(z)) \\ &= e^{p(z)} \end{aligned}$$

Since $p(x) = 0, p(y) = 1$, $f(x) = 1, f(y) = e$, so there is a model f that separates arbitrary points.

Therefore, we can conclude that since $\mathcal{NN}^\phi$ is a subalgebra of $C(X, \mathbb{R})$ with a nonzero constant that separates points, it is therefore dense in $C(X, \mathbb{R})$ by the Stone-Weierstrass Approximation theorem.

2 Arbitrarily Dense ReLU Networks are Universal for Fixed Width

It is well known that $\mathcal{NN}^{\text{ReLU}}$ is dense in $C([0,1]^d; \mathbb{R})$. The objective is to show that arbitrarily deep ReLU networks of width at most $d + 2$ are universal approximators on the d dimension unit cube.

For any $f \in C([0,1]^d, \mathbb{R})$, it follows from the assumption that there exists a two layer RELU network

$$u^{\text{two}}(x) = \sum_{i=1}^{n} c_i \text{ReLU}(w_i^\top x + b_i) + c_0$$

such that $\|f - u^{\text{two}}\|_{C^0} < \varepsilon$. For notational convenience, let $\kappa_i(x) := c_i \text{ReLU}(w_i^\top x + b_i)$ for $i = 1, \ldots, n-1$ and $\kappa_n(x) := c_n \text{ReLU}(w_n^\top x + b_n) + c_0$. Then, $u^{\text{two}} = \sum_{i=1}^{n} \kappa_i(x)$.

2.1 ReLU Identity Map

We can easily define a two layer neural network that maps $x \to x$ in $[0,1]^d$. If:

$$u_{NN}(x) := W^{(2)} \left(\text{ReLU} \left(W^{(1)} x + b^{(1)} \right) \right) + b^{(2)}$$

Let $b^{(1)}, b^{(2)} = \mathbf{0} \in \mathbb{R}^d$, and $W^{(1)} = W^{(2)} = I \in \mathbb{R}^{d \times d}$. Then:

$$\begin{aligned} u_{NN}(x) &= W^{(2)} \left(\text{ReLU} \left(W^{(1)} x + b^{(1)} \right) \right) + b^{(2)} \\ &= \text{ReLU}(x) \end{aligned}$$

However, since $x \in [0,1]^d$, we have $\text{ReLU}(x_i) = x_i \ \forall i = 1, \ldots, d$ which implies $u_{NN}(x) = x$, or that $u_{NN} = \text{Id}$

2.2 Extending ReLU Identity Map

We know $\kappa_1(x) = c_1 \text{ReLU}(w_1^\top x + b_1) \in \mathbb{R}$ where $w_1 \in \mathbb{R}^d, c_1, w_1 \in \mathbb{R}$.

If we define $W^{(1)} := \begin{bmatrix} I_d \\ w_1^\top \end{bmatrix} \in \mathbb{R}^{d+1 \times d}, b^{(1)} = \begin{bmatrix} \mathbf{0}_d \\ b_1 \end{bmatrix} \in \mathbb{R}^{d+1}$, then:

$$\begin{aligned} \text{ReLU} \left(W^{(1)} x + b^{(1)} \right) &= \text{ReLU} \left(\begin{bmatrix} I_d \\ w_1^\top \end{bmatrix} x + \begin{bmatrix} \mathbf{0}_d \\ b_1 \end{bmatrix} \right) \\ &= \text{ReLU} \left(\begin{bmatrix} x \\ w_1^\top x + b_1 \end{bmatrix} \right) \\ &= \begin{bmatrix} x \\ \text{ReLU}(w_1^\top x + b_1) \end{bmatrix} \\ &= \begin{bmatrix} x \\ \kappa_1(x) \end{bmatrix} \end{aligned}$$

If we then define $W^{(2)} = I \in \mathbb{R}^{d+1 \times d+1}, b^{(2)} = \mathbf{0} \in \mathbb{R}^{d+1}$, then we have the two layer neural network of width $d + 1$ such that:

$$\begin{aligned} u_{NN}(x) &= W_2^{(2)} \left(\text{ReLU} \left(W^{(1)} x + b^{(1)} \right) \right) + b^{(2)} \\ &= \text{ReLU} \left(W^{(1)} x + b^{(1)} \right) \\ &= \begin{bmatrix} x \\ \kappa_1(x) \end{bmatrix} \end{aligned}$$

2.3 Generalizing ReLU Identity Map, Base Case

Using the definition of $\kappa_2(x)$, maintain $W^{(1)} := \begin{bmatrix} I_d \\ w_1^\top \end{bmatrix} \in \mathbb{R}^{d+1\times d}, b^{(1)} = \begin{bmatrix} \mathbf{0}_d \\ b_1 \end{bmatrix} \in \mathbb{R}^{d+1}$. However, now define the new weight matrix $W^{(2)} := \begin{bmatrix} I_{d+1} \\ w_2^\top \end{bmatrix} \in \mathbb{R}^{d+2\times d+1}$ and bias vector $b^{(2)} = \begin{bmatrix} \mathbf{0}_{d+1} \\ b_2 \end{bmatrix} \in \mathbb{R}^{d+2}$, and let the outer weight and bias matrices $W^{(3)} = I \in R^{d+2\times d+2}, b^{(3)} = \mathbf{0} \in \mathbb{R}^{d+2}$. Then:

$$\begin{aligned}
u_{NN}(x) &= W^{(3)}\text{ReLU}\left(W^{(2)}\text{ReLU}\left(W^{(1)}x + b^{(1)}\right) + b^{(2)}\right) + b^{(3)} \\
&= \text{ReLU}\left(W^{(2)}\text{ReLU}\left(W^{(1)}x + b^{(1)}\right) + b^{(2)}\right) \\
&= \text{ReLU}\left(W^{(2)}\begin{bmatrix} x \\ \kappa_1(x) \end{bmatrix} + b^{(2)}\right) \\
&= \text{ReLU}\left(\begin{bmatrix} x \\ \kappa_1(x) \\ w_2^\top x \end{bmatrix} + \begin{bmatrix} \mathbf{0}_{d+1} \\ b_2 \end{bmatrix}\right) \\
&= \begin{bmatrix} \text{ReLU}(x) \\ \text{ReLU}(\kappa_1(x)) \\ \text{ReLU}(w_2^\top x + b_2) \end{bmatrix}
\end{aligned}$$

We established $\text{ReLU}(x) = x \;\forall\; x \in [0,1]^d$. But now:

$$\begin{aligned}
\text{ReLU}(\kappa_1(x)) &= \text{ReLU}\left(\text{ReLU}(w_1^\top x + b_1)\right) \\
&= \text{ReLU}\left(w_1^\top x + b_1\right) \\
&= \kappa_1(x)
\end{aligned}$$

Since $\text{ReLU}(w_2^\top x + b_2) = \kappa_2(x)$, we can conclude:

$$u_{NN}(x) = \begin{bmatrix} x \\ \kappa_1(x) \\ \kappa_2(x) \end{bmatrix}$$

where u_{NN} is a three layer ReLU network of maximum width $d+2$.

2.4 Generalizing RELU Identity Map, Inductive Case

Maintain that $W^{(1)} := \begin{bmatrix} I_d \\ w_1^\top \end{bmatrix} \in \mathbb{R}^{d+1\times d}, b^{(1)} = \begin{bmatrix} \mathbf{0}_d \\ b_1 \end{bmatrix} \in \mathbb{R}^{d+1}$, and that $W^{(2)} := \begin{bmatrix} I_{d+1} \\ w_2^\top \end{bmatrix} \in \mathbb{R}^{d+2\times d+1}$, $b^{(2)} = \begin{bmatrix} \mathbf{0}_{d+1} \\ b_2 \end{bmatrix} \in \mathbb{R}^{d+2}$, define $W^{(3)} = \begin{bmatrix} I_d & 0 & 0 \\ \mathbf{0}_d^\top & 1 & 1 \\ w_3^\top & 0 & 0 \end{bmatrix} \in \mathbb{R}^{d+2\times d+2}$, and $b^{(3)} = \begin{bmatrix} \mathbf{0}_{d+1} \\ b_3 \end{bmatrix} \in \mathbb{R}^{d+2}$, and finally that $W^{(4)} = I \in \mathbb{R}^{d+2\times d+2}$, $b^{(4)} = \mathbf{0} \in \mathbb{R}^{d+2}$.

$$\begin{aligned}
u_{NN}(x) &= W^{(4)}\text{ReLU}\left(W^{(3)}\text{ReLU}\left(W^{(2)}\text{ReLU}\left(W^{(1)}x + b^{(1)}\right) + b^{(2)}\right) + b^{(3)}\right) + b^{(4)} \\
&= W^{(4)}\text{ReLU}\left(W^{(3)}\begin{bmatrix} x \\ \kappa_1(x) \\ \kappa_2(x) \end{bmatrix} + b^{(3)}\right) + b^{(4)} \\
&= \text{ReLU}\left(\begin{bmatrix} I_d & 0 & 0 \\ \mathbf{0}_d^\top & 1 & 1 \\ w_3^\top & 0 & 0 \end{bmatrix}\begin{bmatrix} x \\ \kappa_1(x) \\ \kappa_2(x) \end{bmatrix} + \begin{bmatrix} \mathbf{0}_{d+1} \\ b_3 \end{bmatrix}\right) \\
&= \text{ReLU}\left(\begin{bmatrix} x \\ \kappa_1(x) + \kappa_2(x) \\ w_3^\top x + b_3 \end{bmatrix}\right) \\
&= \begin{bmatrix} x \\ \text{ReLU}(\kappa_1(x) + \kappa_2(x)) \\ \text{ReLU}(w_3^\top x + b_3) \end{bmatrix} \\
&= \begin{bmatrix} x \\ \kappa_1(x) + \kappa_2(x) \\ \kappa_3(x) \end{bmatrix} \quad \text{since } \kappa_1(x), \kappa_2(x) \geq 0 \text{ by definition}
\end{aligned}$$

2.5 Approximating u^{two} using ReLU Networks

Using this framework, we can construct a ReLU network such that:

$W^{(k)} = \begin{bmatrix} I_d & 0 & 0 \\ \mathbf{0}_d ß^\top & 1 & 1 \\ w_k^\top & 0 & 0 \end{bmatrix} \in \mathbb{R}^{d+2 \times d+2}, b^{(k)} = \begin{bmatrix} 0_{d+1} \\ b_k \end{bmatrix} \in \mathbb{R}^{d+1}$. Since $\text{ReLU}\left(\sum_{i=1}^k \kappa_i(x)\right) = \sum_{i=1}^k \kappa_i(x)$, repeated composition, i.e., creating a network of depth $n+1$ creates a ReLU newtwork of fixed maximum width $d+2$ such that for $x \in [0,1]^d$:

$$u_{NN}(x) = \begin{bmatrix} x \\ \sum_{i=1}^{n-1} \kappa_i(x) \\ \kappa_n(x) \end{bmatrix}$$

This network exactly represents u^{two} on the unit cube in $\mathbb{R}^d$, which is known to be dense. Therefore, since the model class of ReLU networks of fixed width, arbitrary depth, completely include a dense model class on $[0,1]^d$, fixed width networks are also a dense model class, and therefore universal approximator models.

3 JAX: Training of Deep Neural Networks

Source Code Available on Github: https://github.com/outlawhayden/ncsu-sciml/tree/main/hw2

We utilize the provided dataset on Moodle (`hw2_p3_data.npz`) from the Moodle page, comprising 500×2 training samples X, and 500×1 label samples y. The objective is to train a neural network to effectively model the relationship between X and y.

Across all experiments, the following hyperparameters remain fixed. We utilize the `adam` optimizer, with constant learning rate, as well as maximum of $100,000$ training iterations. The neural networks are fully connected, with variable depth, but all layers are width 50. We aim to minimize the discrete $L2$ loss function between the model's output, and the known true sample values in y. We additionally utilize partition the provided dataset into a training and test dataset, with the test dataset comprising a 0.33 fraction holdout of all provided data.

The all models are trained on an Apple M1 Pro processor, utilizing the Apple Metal ARM processor API. The models are constructed in the JAX processing library, with Equinox PyTree structuring, and Optax optimization functions. For a model of depth 5 specified in this way, the corresponding training wall time rate was around 250 epoch iterations per second, for a overall training time per individual model generally < 7 minutes.

3.1 Optimizing Model Depth

First, we examine the effect of model depth, using anywhere between 2 and 5 hidden layers. Depicted in Figure 1 is the $L2$ loss of a fully connected neural network on the testing holdout dataset, with all other parameters held constant. We use a learning rate of 10^{-3}, and a ReLU activation function.

Figure 1: Validation Performance vs Model Depth

While there is less of a dramatic difference in smaller models, this empirically suggests that depths $4, 5$ are relatively overfit to the underlying complexity of the problem - the smaller model of depth 2 performs the best, as it's not as overparameterized. This empirically also suggests that performance does not improve past around 4000 training iterations, and that the high initial loss values hint at potentially more reasonable initialization schemes.

3.2 Optimizing Model Learning Rate

Based on 3.1, we fix the model depth at 2 hidden layers, and maintain the use of the ReLU activation function. Now, we search for an improved learning rate for use with the Adam optimizer, testing the values $[10^{-1}, 10^{-2}, 10^{-3}10^{-4}10^{-5}]$ as candidates.

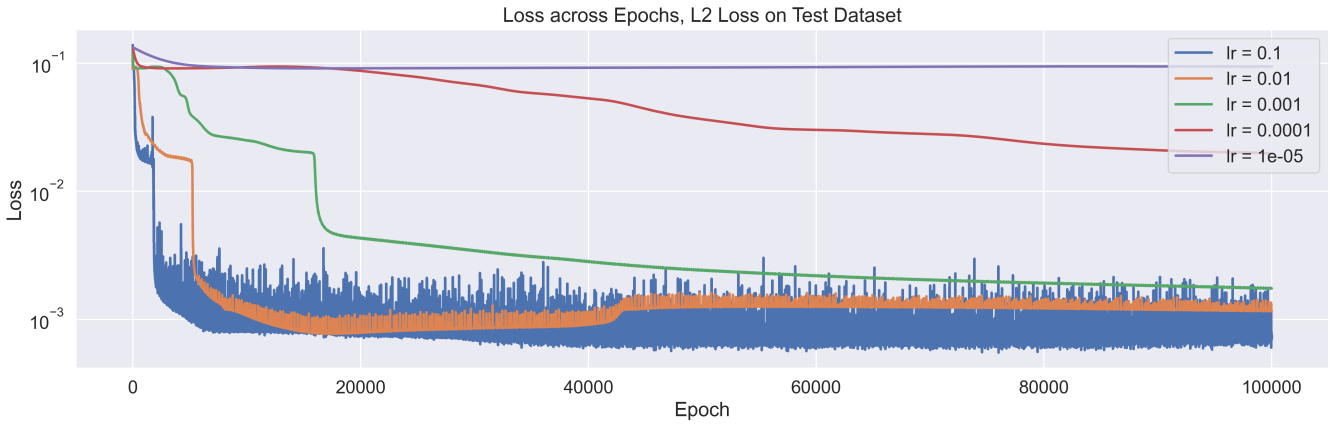

Figure 2: Validation Performance vs Learning Rate

Depicted in Figure 2 is the validation $L2$ loss of the model equipped with each potential learning rate. Clearly, the smallest learning rate does not learn the problem at all - the best performance is lr $= 0.1$, although as clearly shown larger learning rates produce much noisier learning trajectories, as they are more sensitive to concavities within the optimization surface for the model.

The stabilization of the performance past 5000 epochs also supports the need for early stopping logic. While searching the hyperparameters in a linear ordered manner, we fixed the epoch count across all studies so that we can see the full progression of the optimization curves, but this clearly is increasing the computational cost for the model, while at best only marginally improving the model, while at worst overfititng it in some configurations. Implementing early stopping logic, and its consequences, is addressed in Section 3.4.

3.3 Optimizing Activation Function

Fixing the learning rate at 0.01, and depth at 2, we examine performance across different activation functions, listed in Table 3.3. These functions, and their gradients, are all implemented natively in the `jax.nn.functional` package.

Activation	**Definition**
Sigmoid	$\sigma(x) = \dfrac{1}{1+e^{-x}}$
ReLU	$\text{ReLU}(x) = \max(0, x)$
ReLU6	$\text{ReLU6}(x) = \min(\max(0, x), 6)$
GELU	$\text{GELU}(x) = \frac{1}{2}x\left(1 + \tanh\left(\sqrt{\frac{2}{\pi}}\left(x + 0.044715x^3\right)\right)\right)$
Softmax	$\text{softmax}(x_i) = \dfrac{e^{x_i}}{\sum_j e^{x_j}}$
Softplus	$\text{softplus}(x) = \ln\left(1 + e^{x}\right)$
Leaky ReLU	$\text{LeakyReLU}(x) = \begin{cases} x, & x \geq 0 \\ \alpha x, & x < 0 \end{cases}$

In the same way, we can examine holdout L2 performance, as shown in Figure 3. This plot shows a clear performance improvement in selecting the 'softmax' function:

$$softmax(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

While the best learning trajectory clearly stops improving at around 20000 epochs, the longer trajecotry time reveals the instability of some of the activation functions over time - specifically the 'Leaky ReLU' function degrades past a certain point, as the reintroduction of negative values in gradient calculation can make optimization difficult. We can also see that the classic 'ReLU' function also degrades, albeit in a less dramatic manner, probably from inducing gradient collapse as the number of epochs grows.

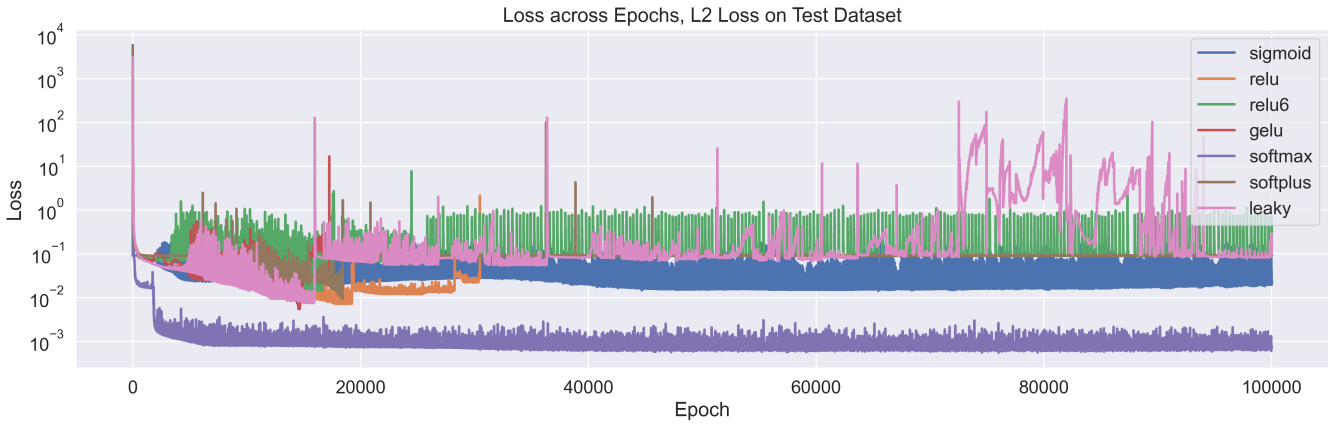

Figure 3: Validation Performance vs Activation Function

3.4 Adaptive Hyperparameter Space Search

Searching the hyperparameter space in this order is not an exhaustive search. As configured so far, there are $(4 * 5 * 7) = 140$ different potential configurations, whereas so far we have only examined $(4 + 5 + 7) = 16$ possible options. An ordered search also makes it difficult to implement early stopping, as the ideal end point for each model is dependent on the specific combination of each hyperparameter together.

To this end, we can use the Optuna hyperparameter optimization package to more efficiently search the hyperparameter space for the model class without introducing an ordering bias. This package can search all possible permutations in a greedy manner based on a defined objective function, instead of in a predefined order. With this, we also implement early stopping logic, halting training if the test loss value does not improve by at least 10^{-3} for at least 100 epochs. Adding this criteria should both improve overall performance and mitigate computational requirements, although it makes comparing learning trajectories more difficult. Specific implementation details, especially with regards to integration with JAX, are available on Github here. Note that the hyperparameter search space will not include the maximum number of epochs, the model layer widths, or any of the early stopping logic.

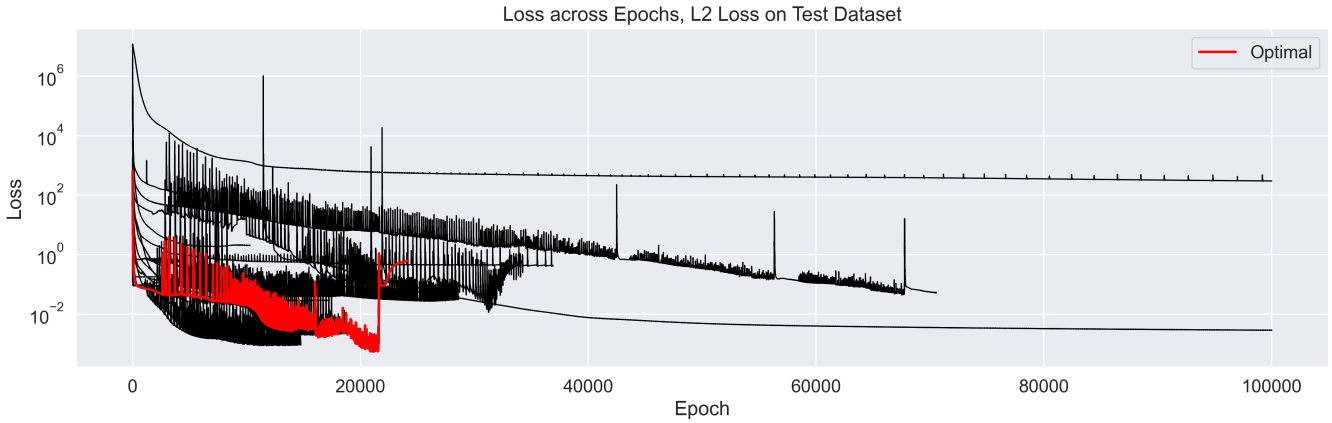

Figure 4: Validation Performance across Configurations

The optimal hyperparameter configuration, out of 20 tested options, was a 'GeLU' activation function, a network depth of 2, and a learning rate of 0.061. The results on the testing dataset of the hyperparameter space search, as well as the learning trajectory of the ideal model, are depicted in Figure 4. Early stopping logic greatly improved training time. As demonstrated in Figure 5, the ability to abort clearly unproductive configurations, as well as the ability to halt converged models, significantly reduces computational load. Note that while certain configurations depicted in Figure 4 could perform similarly on a testing dataset with respect to the L2 norm, they required much more training time before reaching that level of accuracy.

The total time to run a study in Optuna of 20 configurations, with the early stopping logic, was 2275 seconds, or around 37 minutes - a vast improvement over the linear search without early stopping logic.

How did the optimal model actually perform? Figure 6 shows the NN model approximation to the known solution surface, which seems to be some kind of elliptic system. While we were able to reconstruct the circular symmetry, and some of the concentric rings,

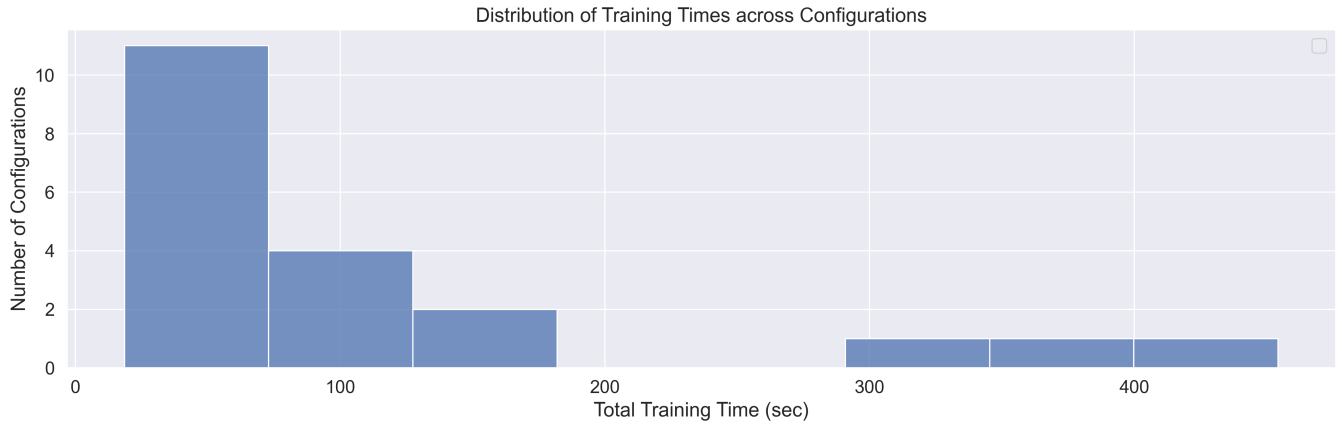

Figure 5: Optuna Trial Times

we were not able to reconstruct the innermost divot. Smaller details in the solution surface would probably require a finer sample mesh in (x_1, x_2), or some additional regularization in the loss function (ie., L1 regularization) that penalizes the model away from predicting the mean value by default, which in this case is $y = 0.0$.

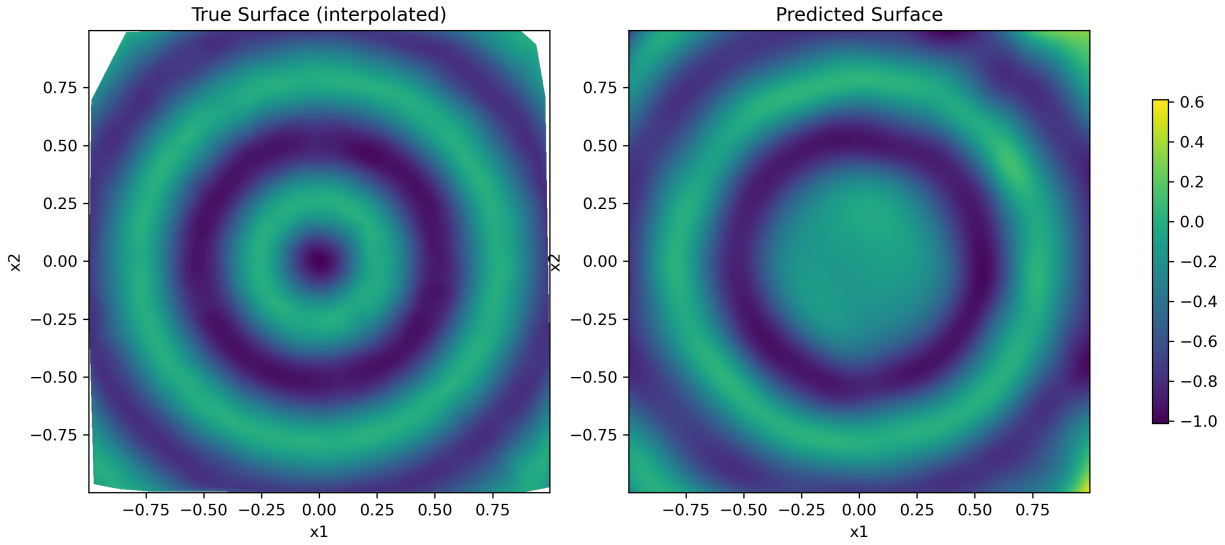

Figure 6: Predicted Solution Surface

Interestingly, note the empty corners in the true surface, which are artifacts of the cubic interpolation methods used to render a smooth image. The network is actually able to correctly fill in the corners with elliptic rings, even though there is no technically provided training data in that area, demonstrating (albeit briefly) an ability to generalize.

There are a few potential ways which this model could still be improved. First, there is most likely a better weight initialization scheme for the training process. Given that this is usually decided based on activation function and other hyperparameters, it wasn't implemented in the Optuna study, but as seen in the individual loss trajectories there is probably some training time to be saved with non-uniform initialization schemes. As mentioned before, we could also add regularization terms to the loss function $L2$ loss function, to avoid overfitting to the mean value in the domain. Finally, we kept all layers the same width, and did not use varied topology. With additional time to conduct the study, we could vary the network topology and see if that could either increase expressivity or produce a lower-complexity model.