

# Using Neural Networks to Control QR Code Design

Emily Haigh

Computer Science Department, Penn State Harrisburg, 777 West  
Harrisburg Pike, Middletown, 17057, PA, USA.

## Abstract

Personalizing QR codes has become increasingly popular, and QR codes that store a URL can be altered by appending a query string to the URL. The goal is to create a neural network that can learn the pattern between a URL with a query string and the corresponding QR code matrix. This experiment uses a query string of 30 alphanumeric characters, which results in a 33 by 33 QR code matrix. The final accuracy achieved was 78.89%.

GitHub link: <https://github.com/emilyh-301/QRCodeResearch>

**Keywords:** QR Codes, Machine Learning, Neural Networks

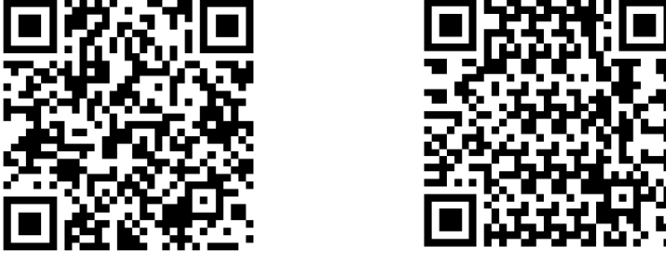
## 1 Introduction

In our fast-paced world, technology changes everyday. One tool that has become really important is the Quick Response (QR) code. They are versatile and have become widely used by many.

QR codes surpass traditional barcodes in terms of information capacity, accessibility, error correction, and interactive functionality. Their versatility and user-friendly nature have contributed to their popularity across different industries and applications. With QR codes' rise in popularity comes the want to customize and personalize the codes.

The goal of this paper is to create a neural network that can learn the pattern between a URL with a query string and the corresponding QR code matrix. This experiment will use a query string of 30 alphanumeric characters, which results in a 33 by 33 QR code matrix. Adding a query string to the URL will change the QR code matrix, and this idea can be used to personalize QR codes. The URL with the appended query string will still lead the user to the

same destination as the URL without the query string. If you scan the codes in figure 1 you can see that both codes take you to the same destination.

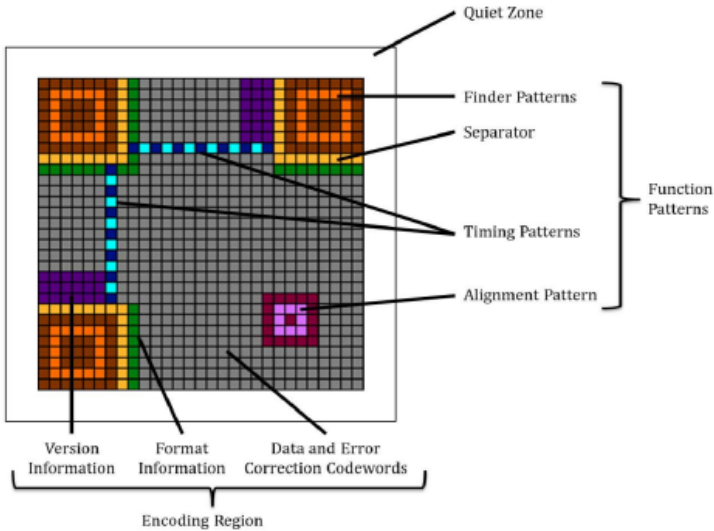


**Fig. 1:** Left: QR Code with Query String. Right: QR Code without Query String.

There are 62 alphanumeric characters, and with strings of length 30, there are  $62^{30}$  possible query strings to use. Moreover, with 33 by 33 QR codes, there are  $2^{33 \times 33}$  possible QR codes to consider. Since  $2^{33 \times 33} \gg 62^{30}$ , it may be difficult for the neural network to find a corresponding query string, if one exists at all.

## 2 Related Work

### 2.1 QR Code Generation



**Fig. 2:** Diagram of the QR code matrix (1)

QR codes are generated by following a specific algorithm which is explained by Sumit Tiwari (1). First, the data is encoded. UTF-8 is commonly used. Second, error correction codes are added to the data, and Reed-Solomon error correction is typically used. The error correction codes make the QR code more robust and allow for successful scanning even when a portion of the code is destroyed. Third, the data and error correction codes are arranged in the correct order. The data is split into smaller chunks, then each chunk is put into a part of the QR code that is normally about 4 or 5 squares, depending on the size of the matrix. There will be an error correction block for every data block. Fourth, the QR code matrix is created. Every single matrix follows a pattern as shown in figure 2. The error correction blocks and data blocks are evenly distributed across the matrix in such a way to improve the integrity of the data. Black squares are 1 and white squares are 0. Fifth, a mask is applied to the matrix. Sixth, the format information which includes the error correction level and the mask pattern, is added to the matrix. The seventh and final step is converting the matrix to an image.

## 2.2 Reed-Solomon Encoding Algorithm

Here is an example of the Reed-Solomon encoding algorithm (6) where we want to encode the message  $m$ .  $m = (m_1, \dots, m_k) \in F^k$ .  $F$  is a finite field.  $m$  will be mapped to  $p$  such that  $p$  is a polynomial over  $F$  of degree less than  $k$ . The characters in the message are treated as the coefficients of  $p$ . Hence, the message *dog* maps to a polynomial like  $d + ox + gx^2$ .

Next an integer  $n$  is selected based on the error correction level and  $n$  must be less than the length of  $m$ .  $n$  unique points  $\in F$  are selected:  $a_1, \dots, a_n$ . Those  $n$  points are used to create the set  $C$  of codewords:  $C = \{p(a_1), p(a_2), \dots, p(a_n)\}$ . These codewords are used as the error correction information.

Note, there are different ways to interpret this algorithm, but I explained the original idea presented by Reed and Solomon in 1960.

## 2.3 Personalizing and Beautifying QR Codes

Beautifying QR codes means to enhance its visual appearance by applying various design techniques, while maintaining the functionality of the code. M. Xu et al. (2) explain some existing ways to “beautify” QR codes, and then they go on to explain their new beautifying approach. The existing methods include embedding images, deformation, manually changing the matrix, and lastly blending the QR code with an image.

They took into account personalization, artistry, and robustness, when creating their new approach to this problem. Personalization means they want the user to be able to customize the QR code as much as possible. Artistry means that the output image will be as close as possible to the input image and selected style. Robustness means that we do not want to exceed the QR code error correction capability.

They do not mention changing the input URL, so there is potential to combine my method with theirs to further enhance the QR codes. Additionally, my method will ensure robustness.

## 2.4 Neural Network Properties

The paper “Intriguing Properties of Neural Networks” (3) published in 2014 by Szegedy et al, suggests that a neural network’s learned input-output

mappings are significantly discontinuous. Their study consisted of them training a neural network to classify images. Then, they edited the pixels of the input images in a way that is not perceivable to humans. The edits cause the pre-trained network to misclassify those images. Their conclusion was that “deep neural networks that are learned by back-propagation have non-intuitive characteristics” and “structure is connected to the data distribution in a non-obvious way”.

## 2.5 Hyperparameters

The following related works are about neural network hyperparameters. The paper “Activation Functions in Neural Networks” (4) explained that ReLU (rectified linear unit) is a widely used and strong performing activation function when used in hidden layers. It can be advantageous to use different functions in the hidden and output layers (4).

A loss function compares the target and predicted output values during training to measure the accuracy of the network. Mean Squared Error (MSE) is a convex loss function, therefore it works well with gradient descent to learn the weights. One disadvantage of mean squared error is that it is sensitive to outliers (5).

## 3 Dataset

The input to the neural network is a valid QR code matrix: a 33 by 33 2-dimensional list of 0's and 1's. To create this data, I generated 100,000 unique 30 character alphanumeric sequences. Next, I needed to generate the QR codes. I configured my QR code to use the error correction level L. Then, I took the base URL <https://h3turing.vmhost.psu.edu> plus “?” and appended each sequence individually. Resulting in <https://h3turing.vmhost.psu.edu?5qp0MJ4x794kWSabn8lKj74zZm1Gis>, for example. The URL with the appended query string will still lead the user to the same destination as the URL without the query string. The last step of generating the data is creating the QR code matrix, and that was done with the “qrcode” Python library. 80% of this data was used for training and the remaining 20% was used for testing. 20% of the training data was used for validation during training.

## 4 Methodology

The goal was to create a neural network that could take a QR code matrix as input, and output a 30 character query string. The input is a 33 by 33 2-dimensional binary list. However, the neural network cannot output characters. To solve that issue, the neural network will output 180 binary values. 6 binary values will map to 1 character using the binary Base64 alphabet.

A very important factor in this experiment was finding the appropriate architecture for the neural network. I considered trying a CNN, but my thought was that the convolutional layers would not make sense with this data. I used the pre-made CNN Efficient Net B0 as an early test to see if it could learn the pattern. The results were not promising, so that lead me to design my own architecture.

### 4.1 Hyperparameters

I chose ReLU as the hidden layer activation function because it is known to be strong performing. For the activation function in the output layer, I needed

Hidden Layer Activation	ReLU
Output Layer Activation	Sigmoid
Loss	Mean Squared Error

**Fig. 3:** Hyperparameters

a function whose output is between 0 and 1. Therefore, I chose sigmoid for its range of (0,1). Knowing that it can be advantageous to use different functions in the hidden and output layers further confirmed my decision to try ReLU and sigmoid.

$$ReLU(x) = \max(0, x)$$

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

For the loss function, I selected Mean Squared Error (MSE). Since the input to this network will be valid QR matrices, no significant outliers are expected. The network's output will be a list of numbers between 0 and 1, So, a random output will have a difference of .5 from its target value on average. Then, .5 squared is .25, so I expect the untrained network to start with a loss near .25 and that loss should decrease with training.

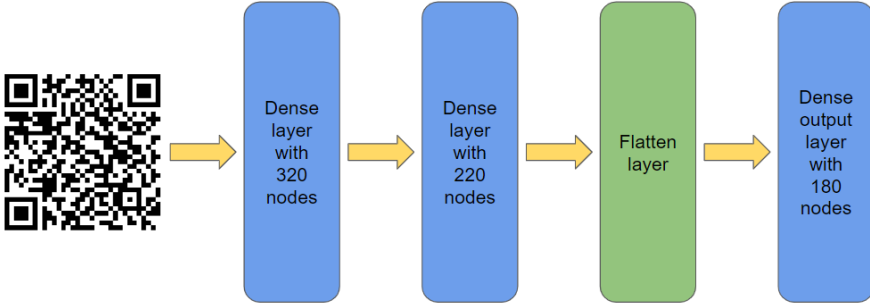
$$MSE = \sum_{i=1}^n (x_i - y_i)^2$$

Next, I decided how to measure accuracy. The output of the network will be a list of 180 numbers in the range (0,1), and the target output is a list of 180 binary values. Before these lists can be compared, each number in the output list is rounded to the nearest integer. Next, the 2 lists are compared and the accuracy is determined by the amount of matching bits.

I chose to use dense layers in the following networks because the information from every single node is sent to each node in the next layer. That is important because the data and error correction information is spread throughout the matrix.

## 4.2 Network 1

Figure 4 is a diagram of this network's architecture.



**Fig. 4:** Architecture of Network 1

Here is an explanation of why there are approximately 43.2 million parameters in this network: For each data point  $x$ , it is multiplied by a weight  $w$  and a bias  $b$  is added. For the first layer, there are 320 weights and 320 biases.

$$320 + 320 = 640$$

$$320 * 200 + 200 = 70620$$

The flatten layer has no weights, but the output shape will be  $33 * 33 * 220 = 239580$ .

$$239580 * 180 = 43124580$$

Therefore, the total number of parameters is 43,195,840.

### 4.3 Network 2

My idea for network 2 was to flatten the input first because the 2d input can easily be converted to a 1d list. I tried to keep the number of parameters close to that of network 1, so this network has about 40.7 million parameters.

Layer (type)	Output Shape	Param #
flatten (Flatten)	(None, 1089)	0
dense (Dense)	(None, 6000)	6540000
dense_1 (Dense)	(None, 4000)	24004000
dense_2 (Dense)	(None, 2000)	8002000
dense_3 (Dense)	(None, 1000)	2001000
dense_4 (Dense)	(None, 180)	180180
=====		
Total params: 40,727,180		
Trainable params: 40,727,180		
Non-trainable params: 0		

**Fig. 5:** Architecture of Network 2

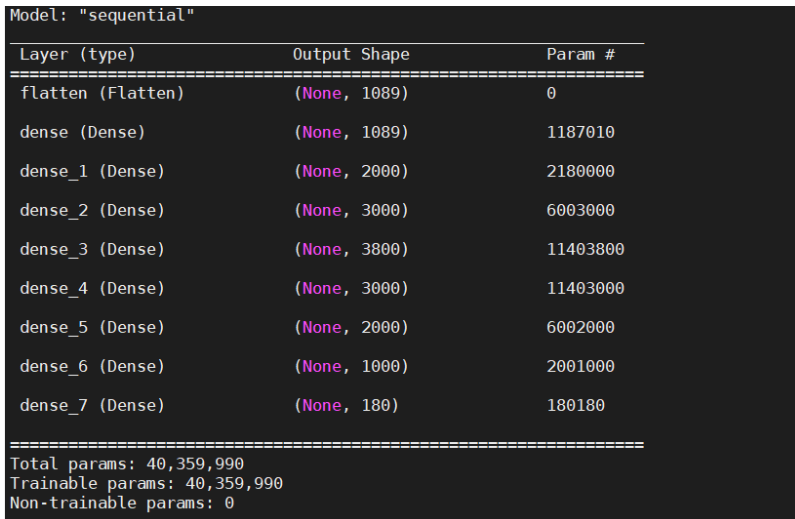
Figure 5 shows network 2's architecture. Here is an explanation for the number of parameters in network 2:

There are zero parameters in the first layer.

$$\begin{aligned}
 1089 * 6000 + 6000 &= 6540000 \\
 6000 * 4000 + 4000 &= 24004000 \\
 4000 * 2000 + 2000 &= 8002000 \\
 2000 * 1000 + 1000 &= 2001000 \\
 1000 * 180 + 180 &= 180180 \\
 6540000 + 24004000 + 8002000 + 2001000 + 180180 &= 40,727,180
 \end{aligned}$$

## 4.4 Network 3

Network 3 has about 40.3 million trainable parameters.



```

Model: "sequential"
Layer (type)                Output Shape                Param #
=====
flatten (Flatten)           (None, 1089)                0
dense (Dense)                (None, 1089)                1187010
dense_1 (Dense)              (None, 2000)                2180000
dense_2 (Dense)              (None, 3000)                6003000
dense_3 (Dense)              (None, 3800)                11403800
dense_4 (Dense)              (None, 3000)                11403000
dense_5 (Dense)              (None, 2000)                6002000
dense_6 (Dense)              (None, 1000)                2001000
dense_7 (Dense)              (None, 180)                 180180
=====
Total params: 40,359,990
Trainable params: 40,359,990
Non-trainable params: 0
  
```

**Fig. 6:** Architecture of Network 3

Figure 6 shows network 3's architecture. The following is an explanation of the weights for network 3:

$$\begin{aligned}
 1089 * 1089 + 1089 &= 1187010 \\
 1089 * 2000 + 2000 &= 2180000 \\
 2000 * 3000 + 3000 &= 6003000 \\
 3000 * 3800 + 3800 &= 11403800 \\
 3800 * 3000 + 3000 &= 11403000 \\
 3000 * 2000 + 2000 &= 6002000 \\
 2000 * 1000 + 1000 &= 2001000 \\
 1000 * 180 + 180 &= 180180
 \end{aligned}$$

The sum of all of these parameters is 40,359,990.

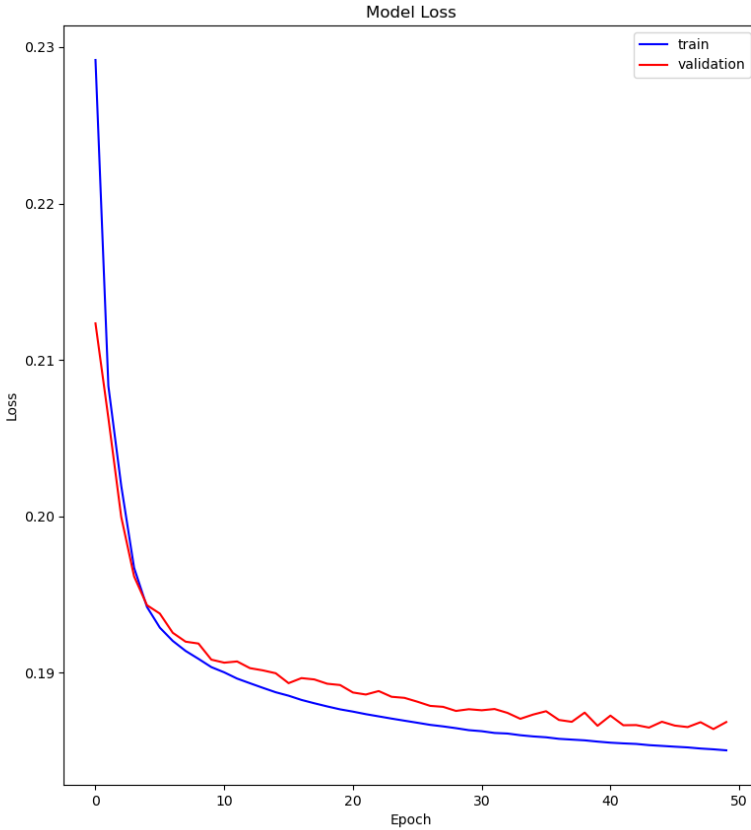
## 5 Results and Discussion

Note that the graphs in this section are not all on the same scale.

Figure 8 shows network 1's loss during training for 50 epochs. After 50 epochs, the loss was .188 and the accuracy was 72.22%. After training for an additional 100 epochs, network 1 was able to achieve the loss and accuracy shown in figure 7.

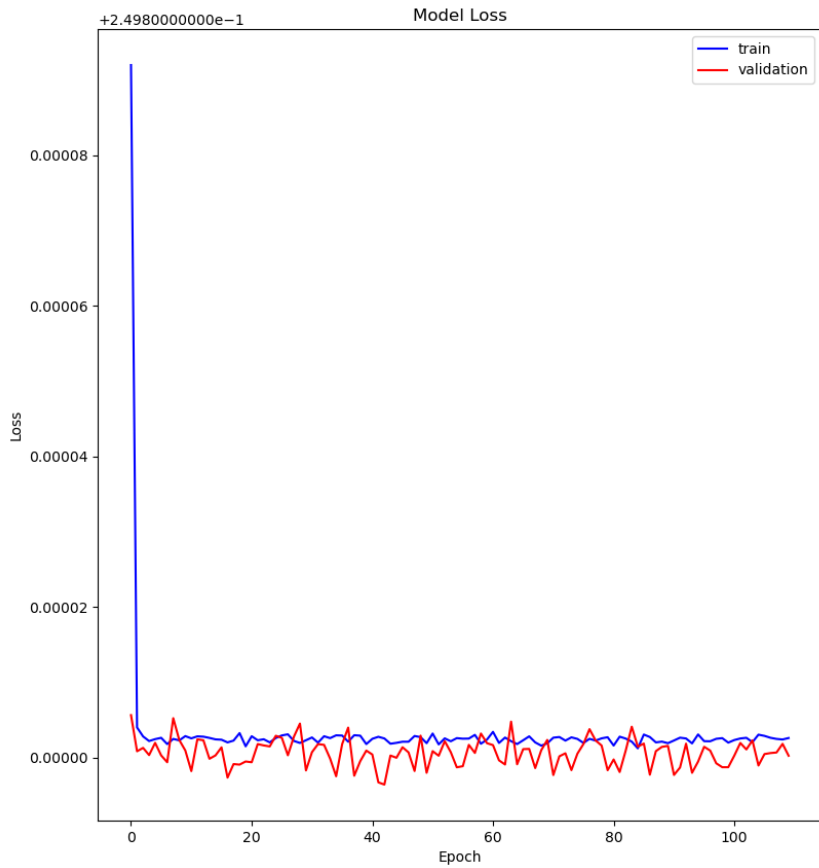
Network	Loss	Accuracy
1	0.185	78.89%
2	0.249	51.36%
3	0.233	54.32%

**Fig. 7:** Results



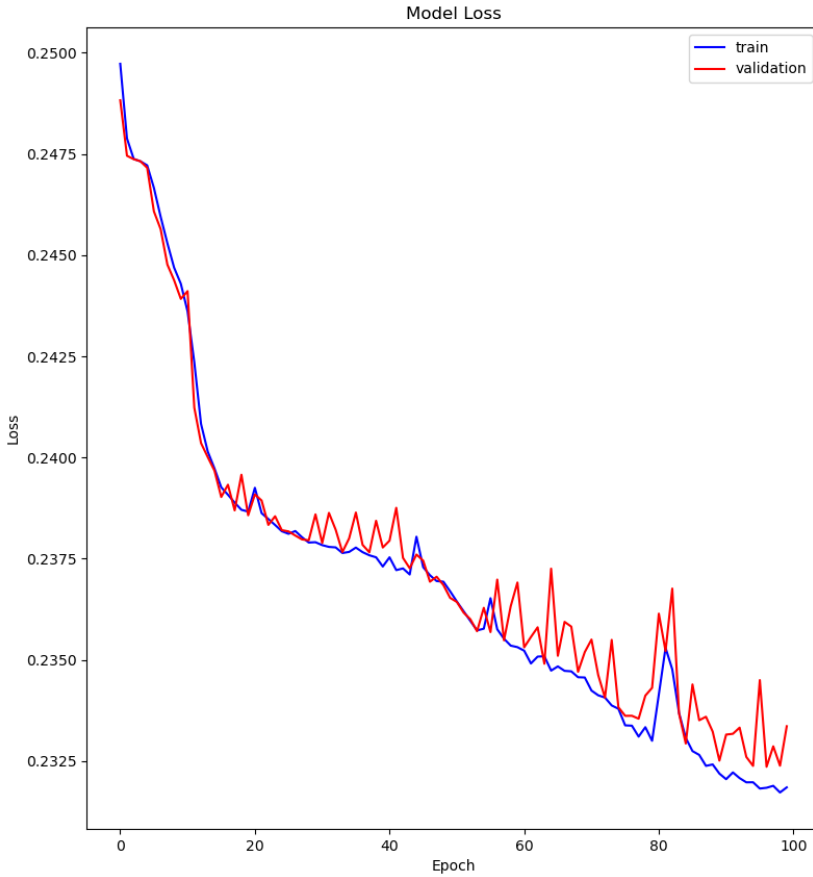
**Fig. 8:** Network 1 Training MSE Loss





**Fig. 9:** Network 2 Training MSE Loss

Figure 9 shows network 2's loss after 110 epochs. Network 2 was able to achieve the loss and accuracy shown in figure 7.



**Fig. 10:** Network 3 Training MSE Loss

Figure 10 shows network 3's loss after 100 epochs. Network 3 was able to achieve the loss and accuracy shown in figure 7.

## 6 Conclusion

Quick response codes are widely used in society and the ability to personalize them is highly desired. From my results, it is clear that we are able to learn the pattern between QR codes and the corresponding query string. I believe this research is a strong foundation for the future of this topic. With more experiments using different architectures and more training, I believe accuracy can increase even more.

## References

- [1] Tiwari, S.: "An introduction to QR code technology". In: 2016 International Conference on Information Technology (ICIT), pp. 39–44 (2016). doi: 10.1109/ICIT.2016.021. <https://ieeexplore.ieee.org/document/7966807>

- [2] M. Xu et al., “Stylized Aesthetic QR Code,” in *IEEE Transactions on Multimedia*, vol. 21, no. 8, pp. 1960-1970, Aug. 2019, doi: 10.1109/TMM.2019.2891420. <https://ieeexplore.ieee.org/abstract/document/8604076>
- [3] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow and Rob Fergus, “Intriguing Properties of Neural Networks,” in *arXiv*, 2014, <https://doi.org/10.48550/arXiv.1312.6199>
- [4] Sagar, Simone Sharma, and Anidhya Athaiya. “Activation functions in neural networks.” *International Journal of Engineering Applied Sciences and Technology* (2020): 310-316. <https://www.ijeast.com/papers/310-316,Tesma412,IJEAST.pdf>
- [5] Yathish, Vishal. “Loss Functions and Their Use In Neural Networks.” *Towards Data Science*, 2022, [towardsdatascience.com/loss-functions-and-their-use-in-neural-networks-a470e703f1e9](https://towardsdatascience.com/loss-functions-and-their-use-in-neural-networks-a470e703f1e9).
- [6] “Reed–Solomon Error Correction.” *Wikipedia*, 27 Jan. 2023, <https://en.wikipedia.org/wiki/Reed%E2>
- [7] Neal, Brady, and Et al. “A Modern Take on the Bias-Variance Tradeoff in Neural Networks.” *arXiv*, 2019, <https://arxiv.org/abs/1810.08591>.