Applications of Fourier analytic Barron Space theory

Applications

Jonathan Miller, April 24, 2024

Applications of Fourier analytic Barron Space Theory

Applications: Outline

- Barron-E Canonical representation
- Model Compression
- Barron-E Weight initialization
  - Sum of Sines
  - Digits
- Barron neural network
- Conclusions

Barron-E Canonical representation: Sum of Product of Sines

Barron-E Canonical representation

Sum of Product of Sines

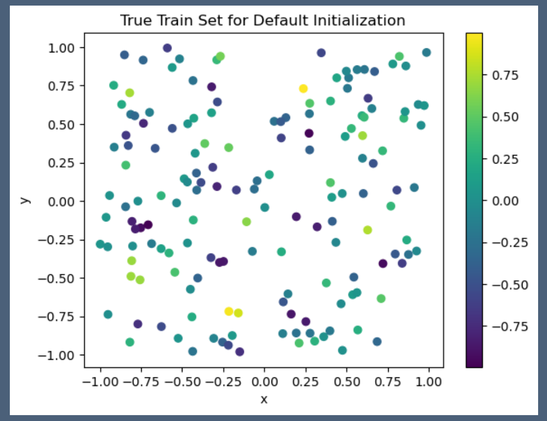
- The standard shallow neural neural network is $w_{ij}^1\sigma\left(w_{jk}^0x_k+b_j^0\right)+b_i^1$ where the superscripts $1=$ outer and $0=$ inner and we are doing Einstein summation over k up to d, the dimension, and over j up to M, the number of nodes, and the i is per target.
- In the Barron-E connoical form, we have $\sum_j^M c(\omega_j,\alpha_j)\sigma\left(\hat{\omega}_{jk}x_k+t_j\right)$ where $\hat{\omega}_{jk}=\omega_{jk}/\|\omega_{jk}\|_1$ where $\|\omega_{jk}\|_1=\left(\sum_k^d|\omega_{jk}|\right)$.
- In Barron-E Canonical form, we see that $c(\omega_j,\alpha_j)\lesssim\tilde{\gamma}_2(f^*)w_{ij}^1\|w_{jk}^0\|_1$, $\hat{\omega}_{jk}\simeq\hat{w}_{jk}^0$ and $t_j\simeq b_j^0/\|w_{jk}^0\|_1$.
  - There is that term $\tilde{\gamma}_2(f^*)$ is a bound, and is a very high bound. We can correct it by, for example, matching effective function examples maximum and minimum with approximation maximum and minimum.
- Obviously we also need to find b_i^1 , recall that the theory is for $f^*(0)\simeq 0$. We can just find it from the effective function examples (maybe use median and not mean).

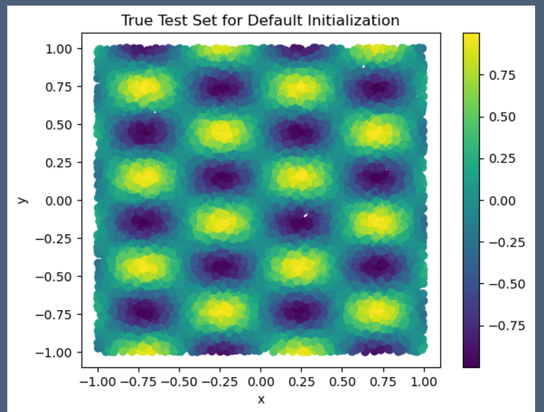
Model Compression: Sum of Product of Sines

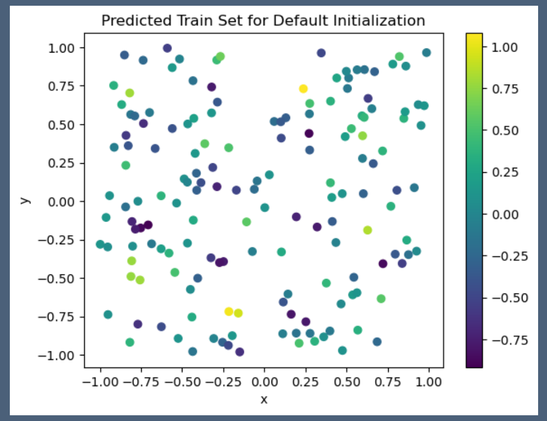
Model Compression

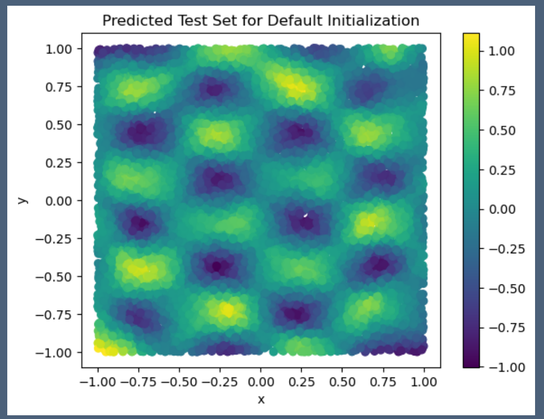
Sum of Product of Sines

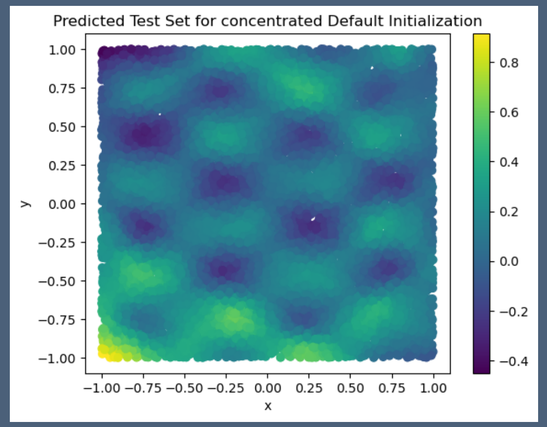
- Experiment: 160 samples, $(x, y) \in [-1,1]^2$, $\sin(2.1\pi x)\sin(3.4\pi y)$
- It is pretty easy to train a $M = 45000$ network on this problem.
- We could select 15000 nodes randomly, and get some approximation.
- But we could also transform to canonical form and then do analysis.

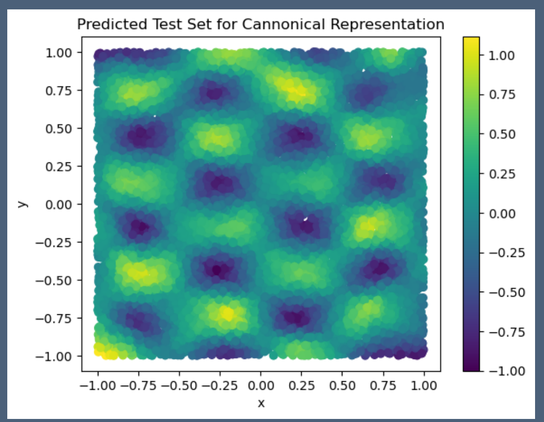








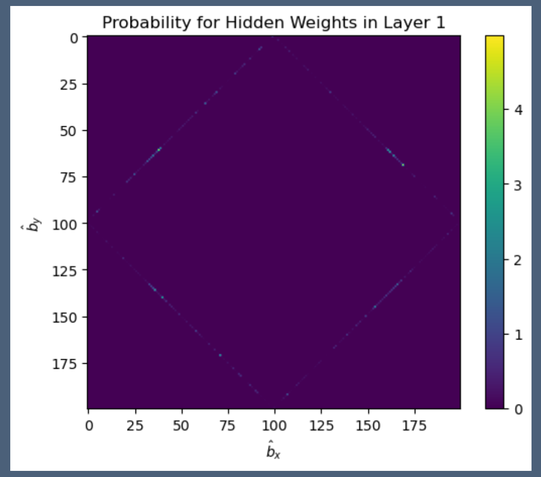


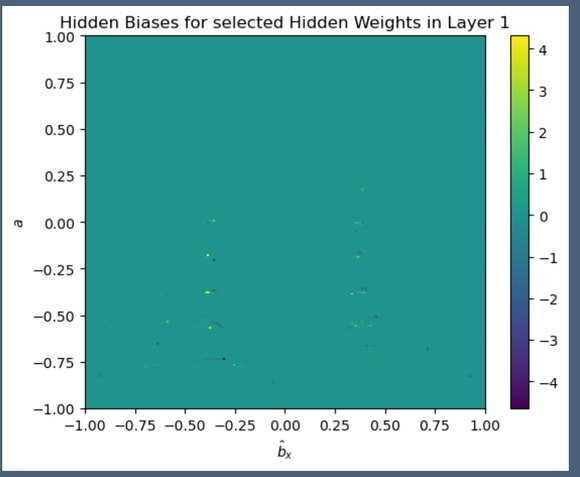


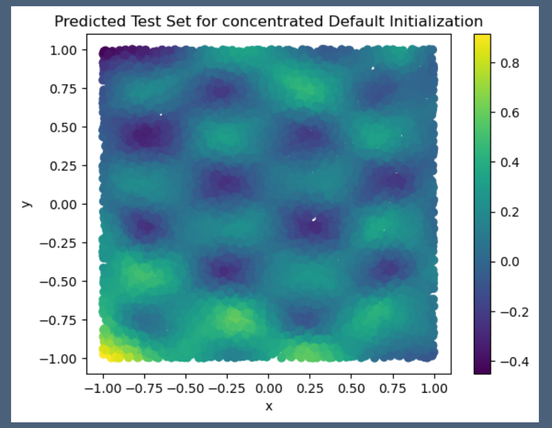
Model Compression

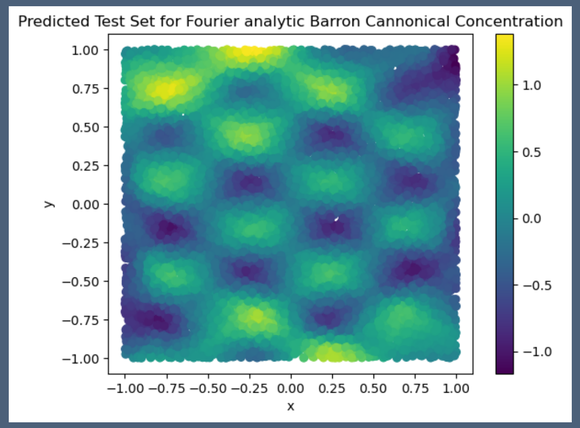
Sum of Product of Sines

- We can look the peaks of probability.
  - A possible way to do this is to quantize or to bin the parameters. This was done here, [-1,1]->[0,255]
- We select the peaks.
- Then we reduce from $M = 45000$ to 21264 nodes.
- We get pretty close to the correct scale, and could expect to have a pretty small increase in our loss.
- We see the structure was found in the bias, and we could select some $\|w_{jk}^0\|_1$ and remove nodes that aren't consistent - This wasn't done in this example.









Barron-E Weight Initialization: Sum of Product of Sines

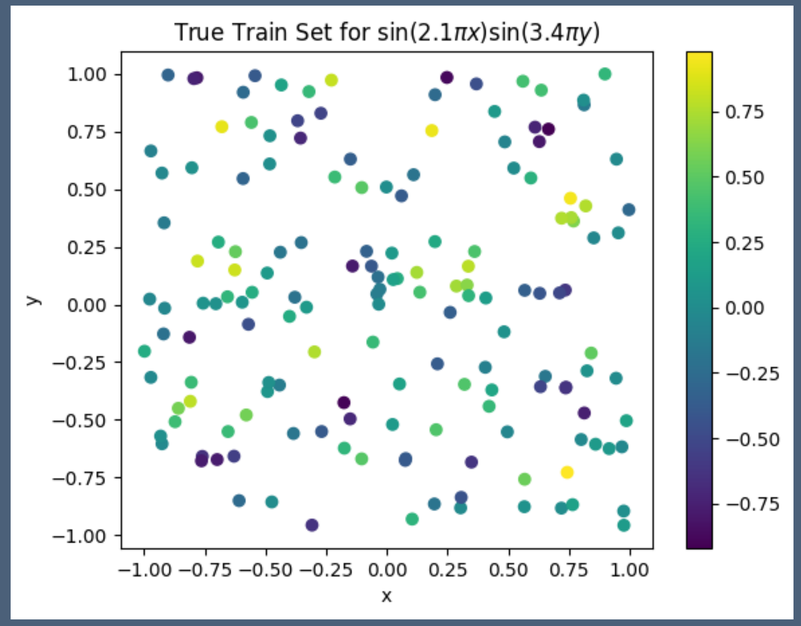
Simple ‘Analytic’ Problem: Sum of Product of Sines

Number of Instances: 160

Attribute Information: $(x, y) \in [-1,1]^2$

The target function is known
$\sin(2.1\pi x)\sin(3.4\pi y)$

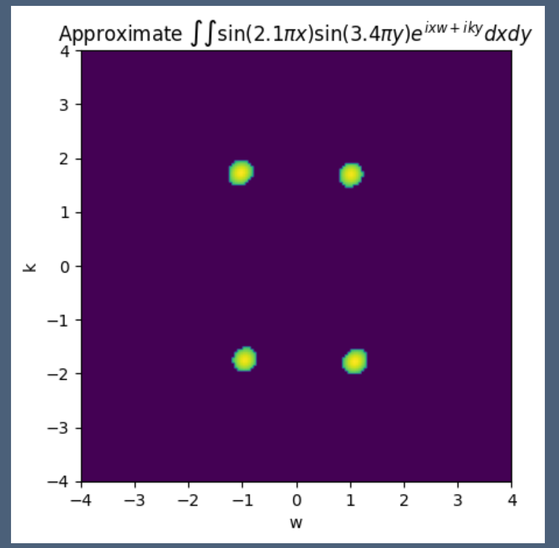
This is at about the minimal size for the theory with $d = 2$ (although the theory extends to $d = 1$).

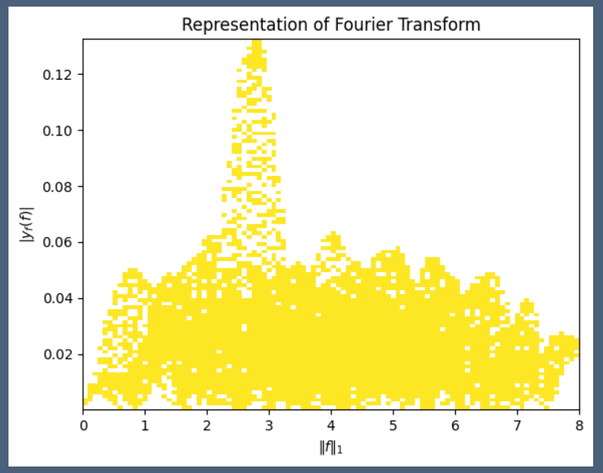


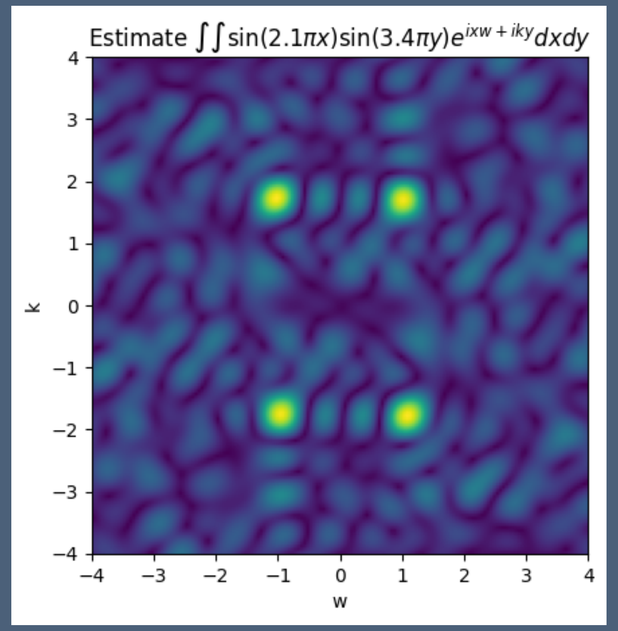
Sum of Product of Sines

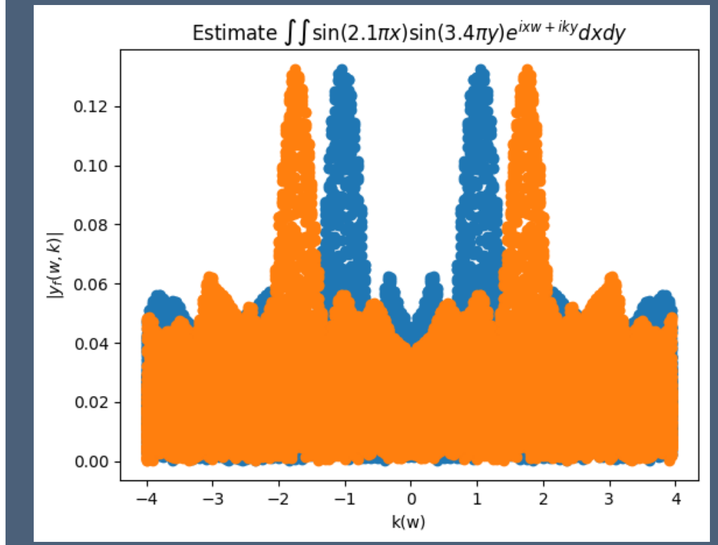
Fourier analysis

- For an analytical problem like this, it is easy to know what the ‘true’ value is.
- While we get an approximation from the MC estimate, we can improve our use case (as a PDF and as a spectral norm) by applying a threshold on the MC estimate of the Fourier transform.
- We initially sample sparsely, but then sample more densely in interesting region.
- Due to random sampling, we can get a better or worse estimate of the Fourier transform.









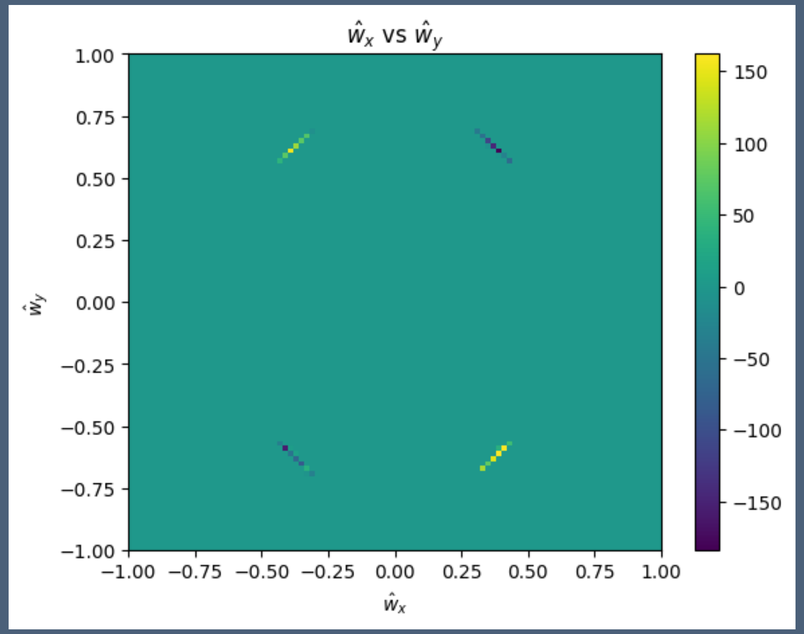
Sum of Product of Sines: PDF

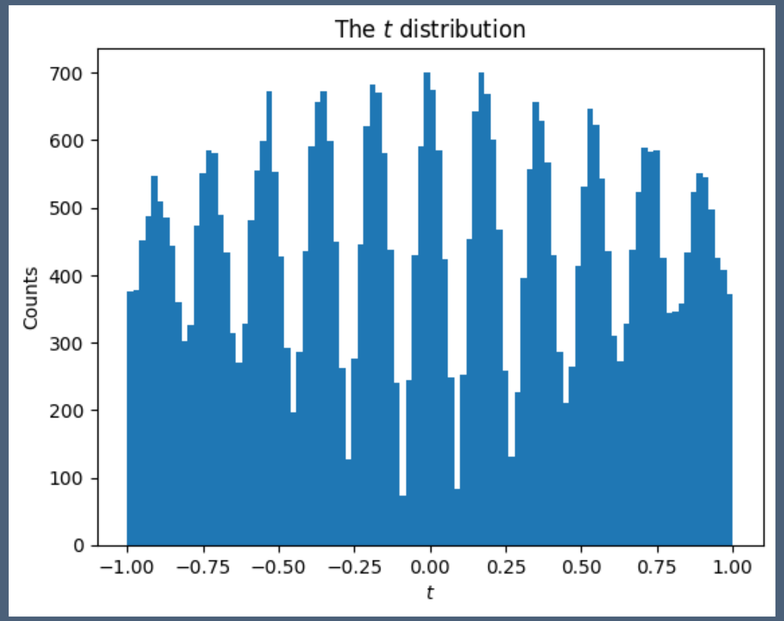
We pull 45000 times from our PDF.

We easily see the peaks in $(\hat{w}_x, \hat{w}_y)$ at

$$(\pm\frac{2.1}{2.1+3.4}, \pm\frac{3.4}{2.1+3.4}) = (\pm 0.38, \pm 0.62)$$

Since we have some width (if we have a great sample, we could place a very high threshold), the peaks in t are noisier, but we see the expected $2 \times (2.1 + 3.4) = 11$ peaks.

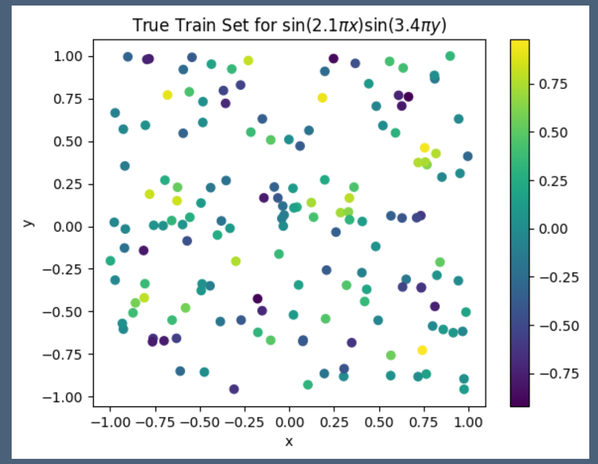


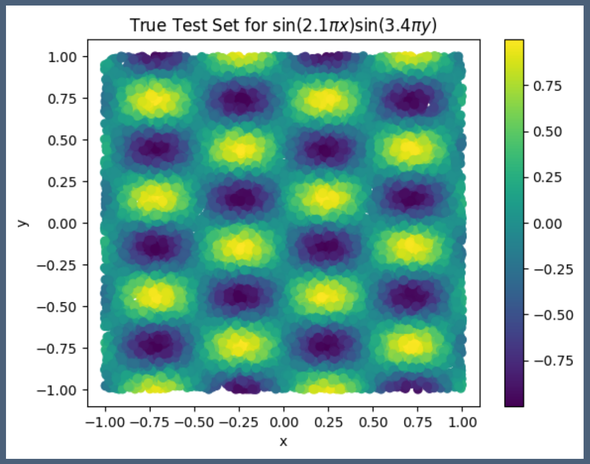


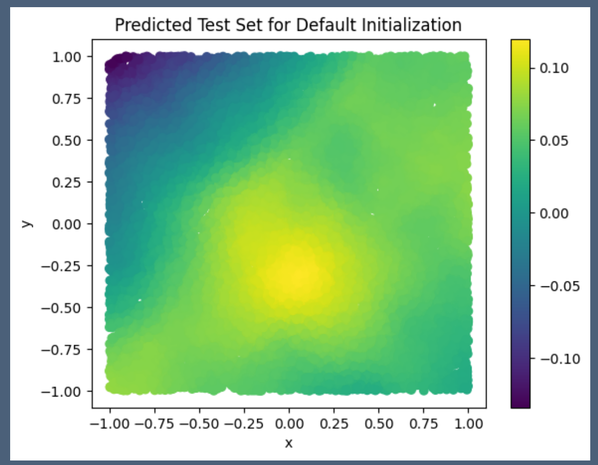
Sum of Product of Sines

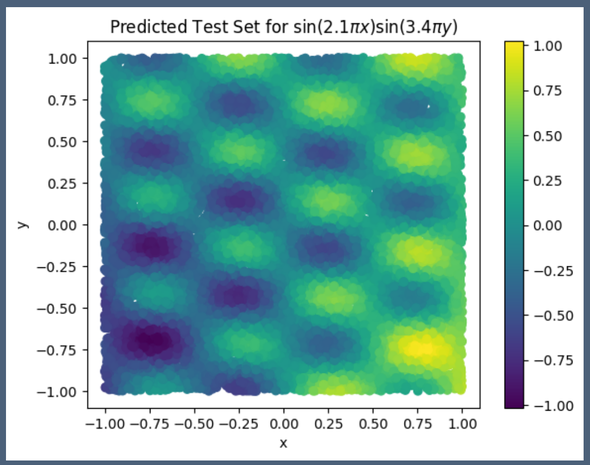
Improved Initialization

- In this case, we actually do better than we are likely able to by training our network using standard practices.
- Current best practices is to rescale the outer weights and use median of initialization for the outer bias.
  - This in general should be from Train set.
- We used $n = 12800$ for the test set.
- It seems that the error, just from initialization, will be small.





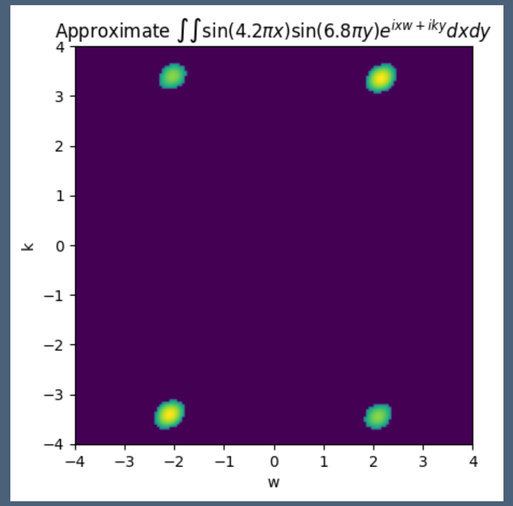


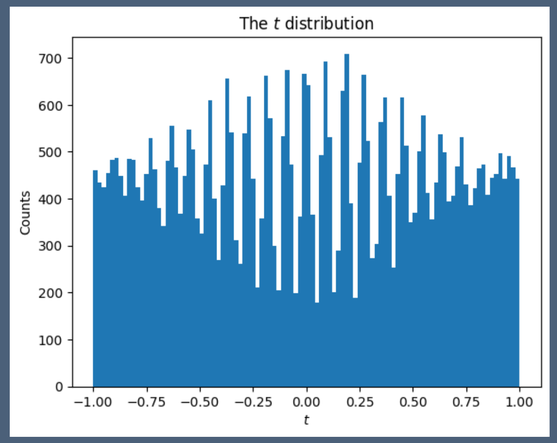


Sum of Product of Sines

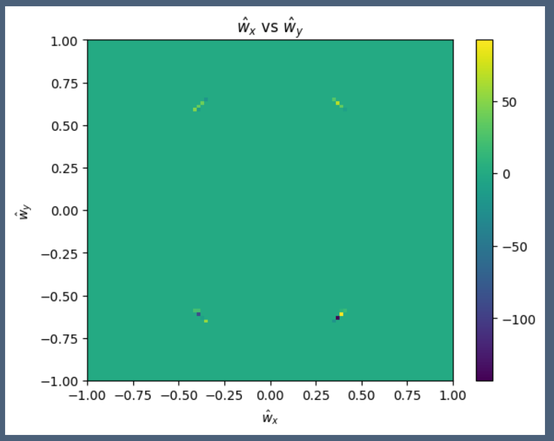
Improved Initialization

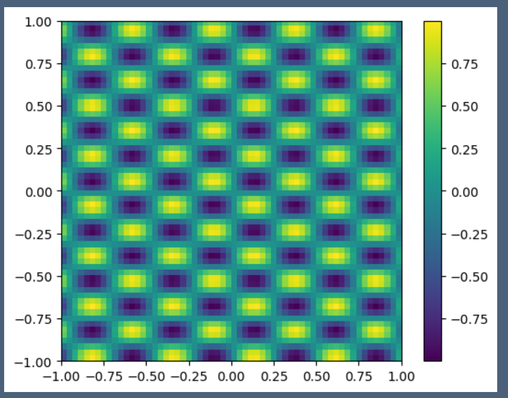
- Here we consider $\sin(4.2\pi x)\sin(6.8\pi y)$ and 160 points.
- We pull 45000 times from our PDF. We easily see the peaks in $(\hat{w}_x, \hat{w}_y)$ at $(\pm 0.38, \pm 0.62)$
- Since we have some width (if we have a great sample, we could place a very high threshold), the peaks in are noisier, but we see the approximately $2 \times (4.2 + 6.8) = 22$ peaks.
- With these few of data points, it is unlikely that a neural network in trained in a standard way would provide a good approximation.



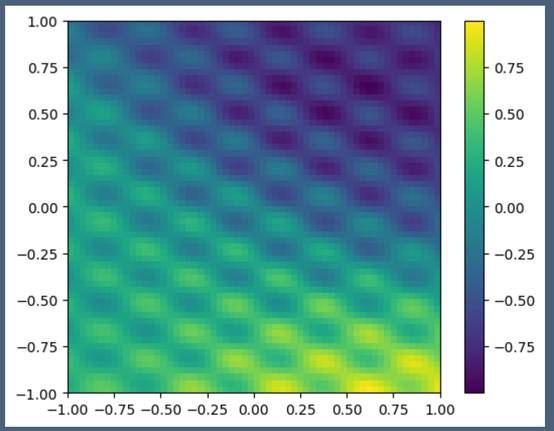


Higher $\|\omega\|_1$ dependence requires both more nodes and is more sensitive to data.





Prediction is scaled.



Barron-E Weight Initialization: Digits

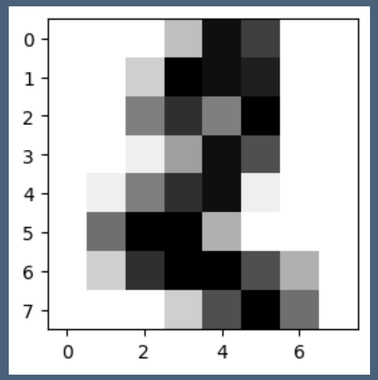
Simple ‘Real’ Problem: Digits

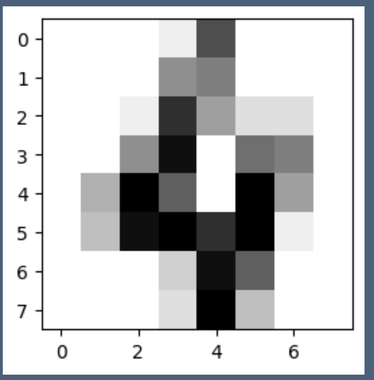
Number of Instances: 1797

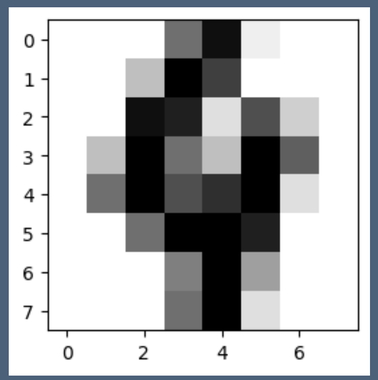
Attribute Information: 8x8 image of integer pixels in the range 0..16

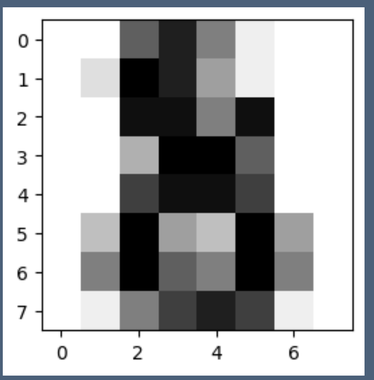
The data set contains images of hand-written digits: 10 classes where each class refers to a digit.

Note that Digits is a small problem, with a space of size $2^{3\times64=192}$, however, still very sparse









From sklearn datasets

Digits

Processing and Problem Statement

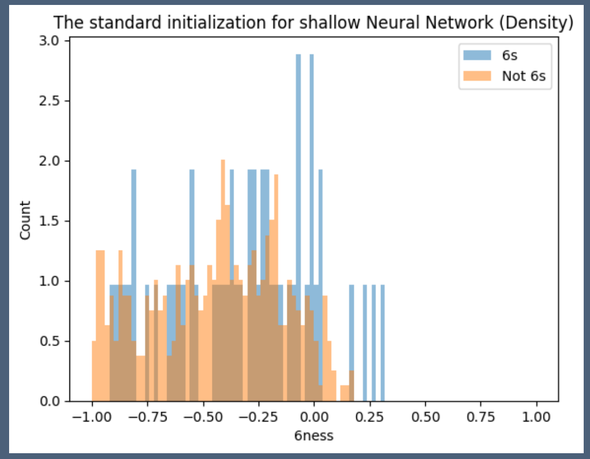
- Our effective target function is if the image is a 6 or not. Instead of using $1,0$ as the target function, we use 6ness as the target function (and so a range of values are possible for any image). The labeled data are given values of $10, -10$.
- We run PCA to reduce down to $d = 6$ features, this makes it easy to calculate and to interpret the features/weights.
- We scale the input by the maximum value of the full dataset. This is so that we fulfill the requirement $x \in [-1,1]^d$.
- We divide into validation (20%), test (30%) and train (50%).
- We use an initialization where the inner biases are set from a uniform distribution scaled by $\pm\sqrt{2/d}$, the outer bias is set to 0, the inner weights are set according to the a normal distribution scaled by $\pm\sqrt{2/d}$ and the outer weights are set by a uniform distribution between $\pm\sqrt{3/M}$ where M is the number of nodes.
- We use SGD with a learning rate of 0.01 and a batch size of 64. We sometimes adjust the number of nodes, unless noted $M = 8000$.

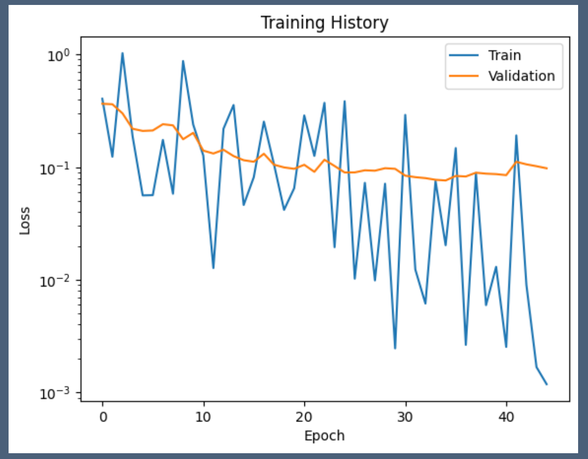
Digits

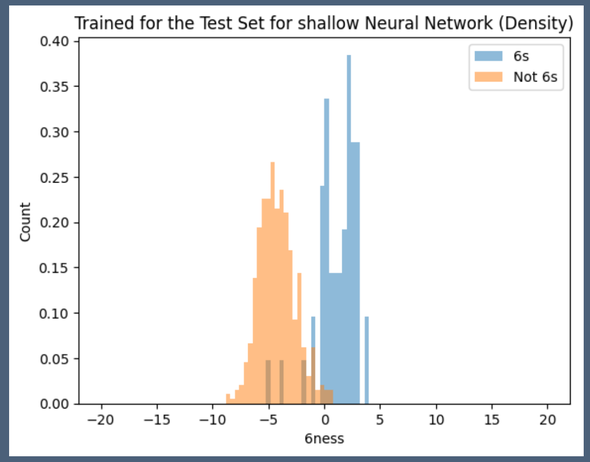
Standard Shallow Neural Network

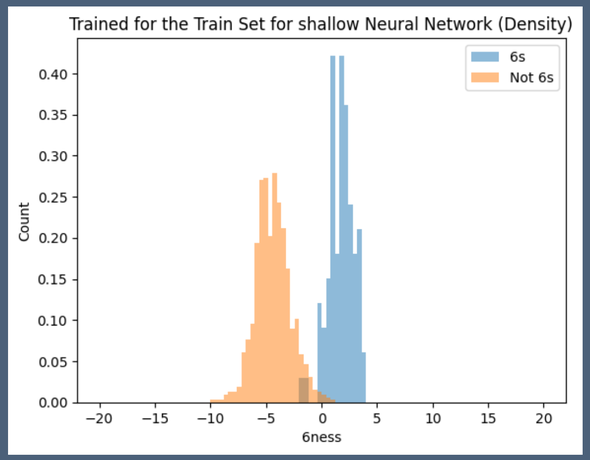
- We only allowed the network to train for 4000 epochs with early stop (if we used GD instead of SGD it would continue to improve slowly to 4000 epochs).
- This is an easy problem, we don't see much sign of overtraining (especially for GD).

	Accuracy	Precision	Sensitivity
Test (Initial)	87.6%	24.1%	13.5%
Train (Final)	98.7%	92.8%	92.8%
Test (Final)	97.0%	87.5%	80.9%









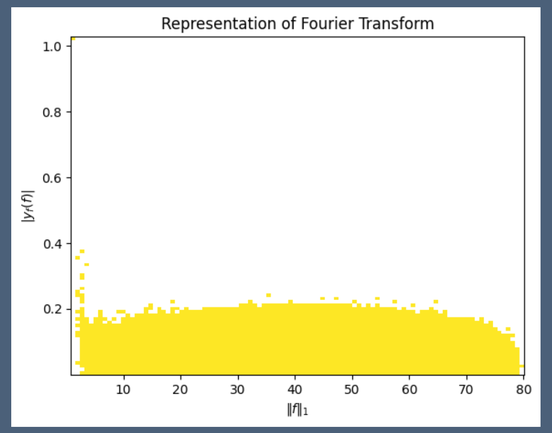
The empirically estimated generalization error is then 1.7% (accuracy), 5.3% (precision) and 12.1% (sensitivity).

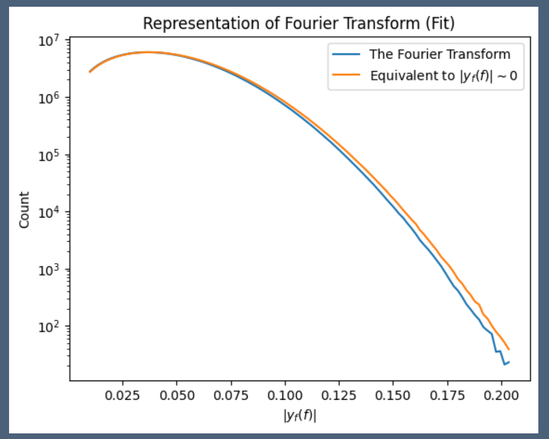
Digits is very simple, very much a toy dataset.

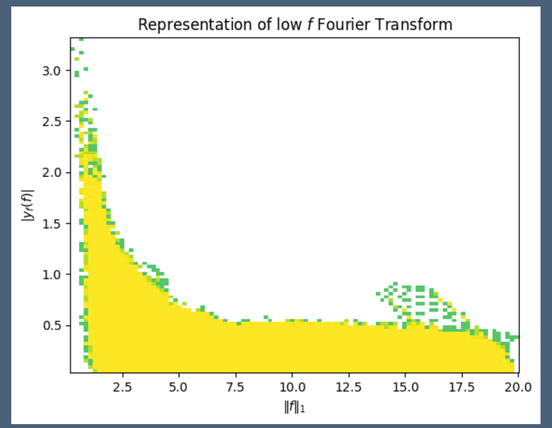
Digits

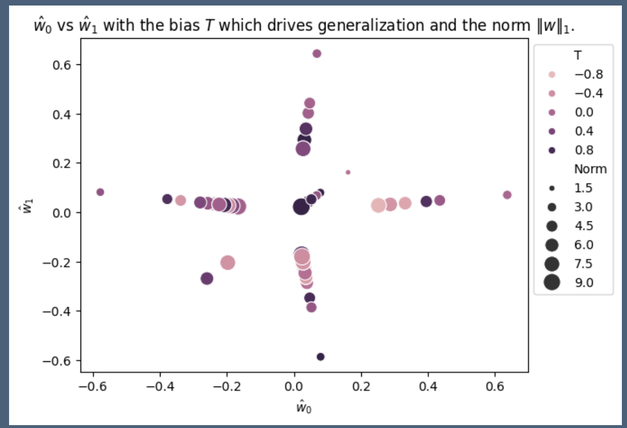
Approximate Fourier transforms

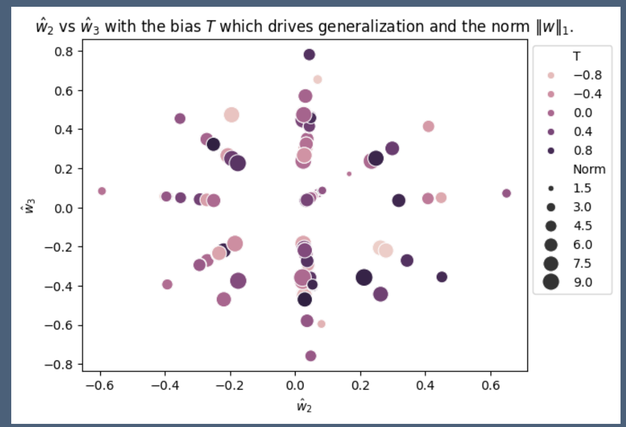
- We use MC estimation to approximate the Fourier transform (Rather than a non-equispaced to equispaced/non-equispaced Fourier transform: NFFT/ NNFFT).
- We find as expected that if we take what we think as the full frequency space, that most of it is consistent with 0 (we can estimate the σ due to the estimation). We can then look at the space and reduce what part of the space we use.
- We can then draw out $\hat{w}_i$, t, z from the PDF defined by the approximate Fourier transform to use as an initialization weight in our network.
- Interestingly, the neural network does well with low freq.

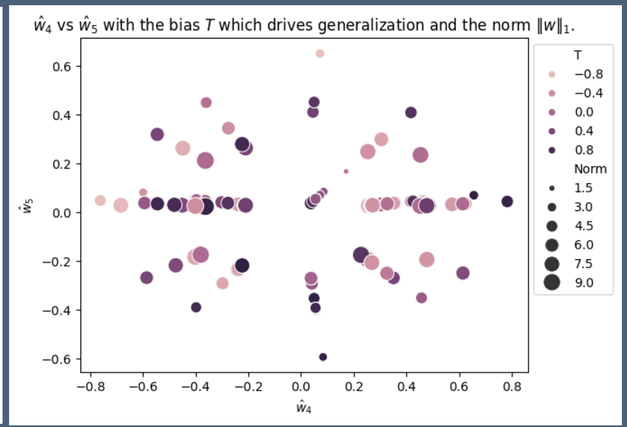




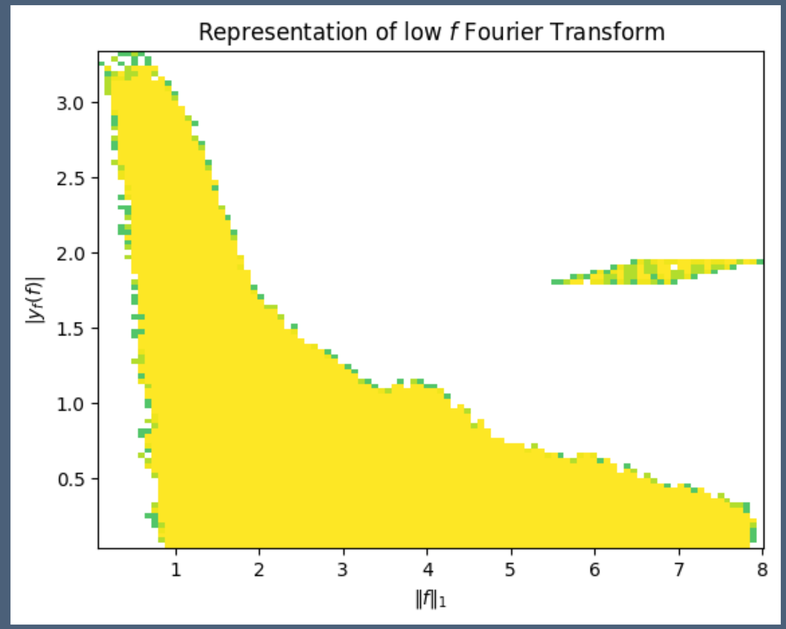






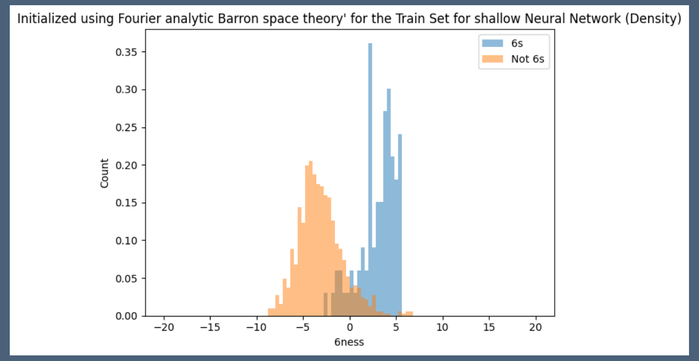


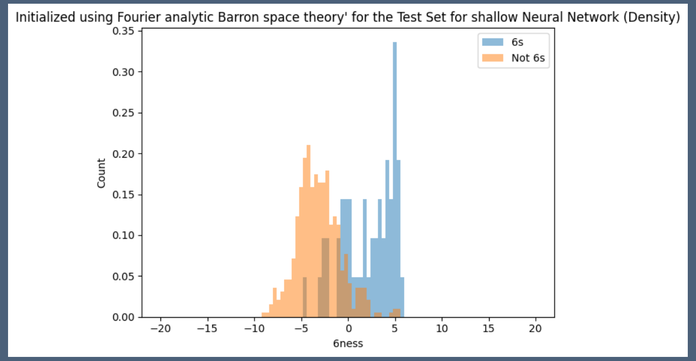
We focus on the lower frequencies

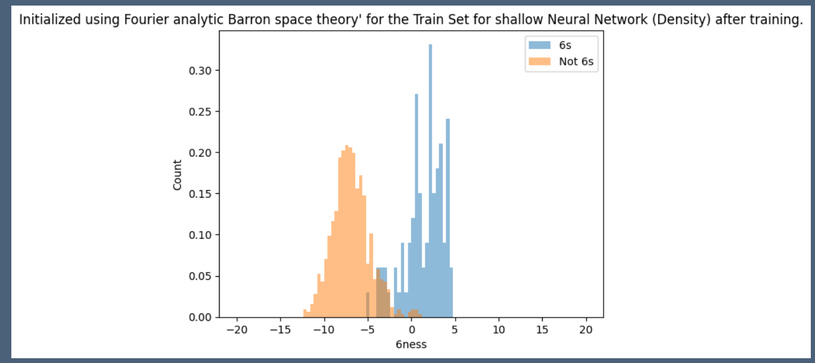


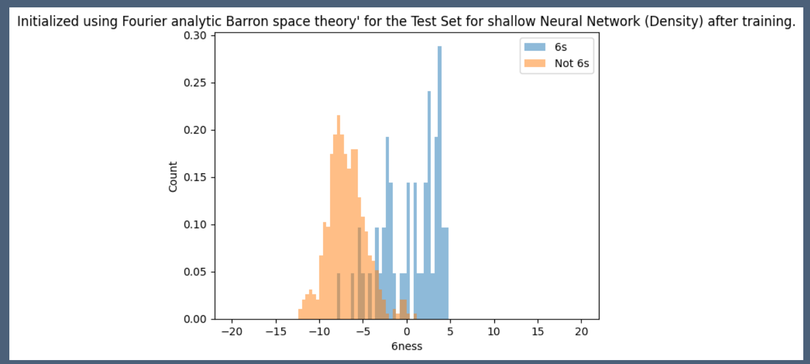
Digits

Barron Initialized Shallow Neural Network









- We only allowed the network to train for 4000 epochs, but we include an early stop condition based on improvement of the validation sample.
- We include a threshold $|\tilde{y}| > 1.7$ here, this doesn't get the high frequency behavior.
- We start close to minimal loss. But our generalization doesn't improve if we train.

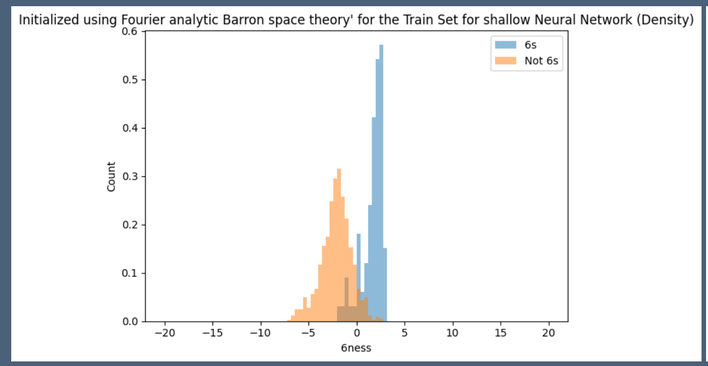
	Accuracy	Precision	Sensitivity
Train (Initial)	90.4%	49.0%	90.4%
Test (Initial)	89.6%	47.5%	73.1%
Train (Final)	97.2%	90.3%	78.3%
Test (Final)	95.7%	93.9%	59.6%

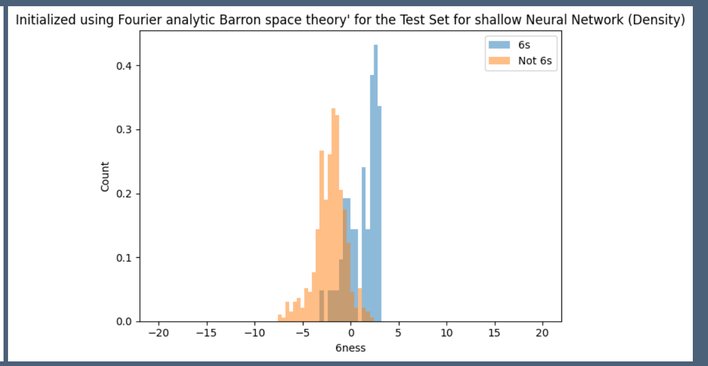
The empirically estimated generalization error is then 1.5% (accuracy), 3.6% (precision) and 18.7% (sensitivity).

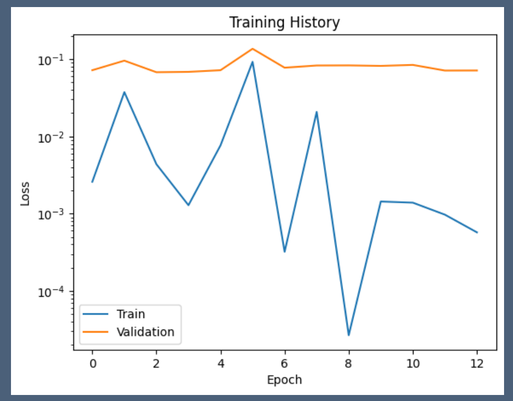
Initialization is much better, but final is worse.

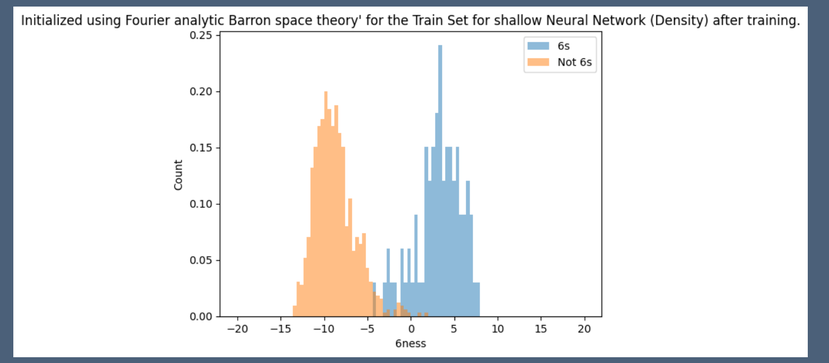
Digits

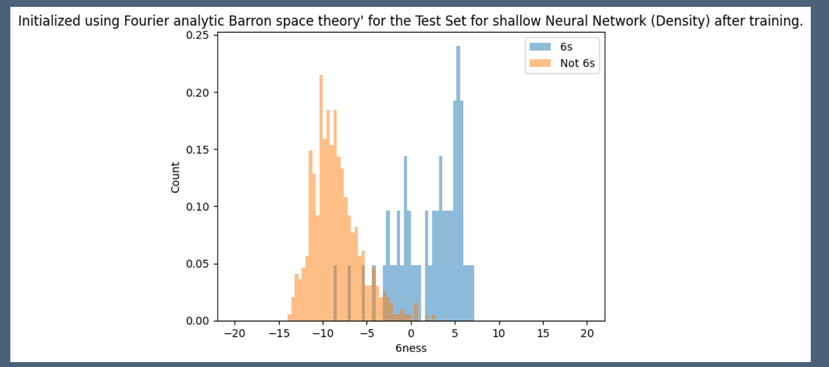
Barron Initialized Shallow Neural Network











- We only allowed the network to train for 4000 epochs with early stop (if we used GD instead of SGD it would continue to improve slowly to 4000 epochs).
- We include a threshold $|\tilde{y}| > 1.2$ here, this doesn't get all the high frequency behavior.
- We need to increase $M = 80000$ since we have higher high frequency contribution.

	Accuracy	Precision	Sensitivity
Train (Initial)	92.3%	55.1%	91.6%
Test (Initial)	91.7%	55.1%	73.1%
Train (Final)	98.6%	97.3%	86.7%
Test (Final)	95.9%	87.5%	67.3%

The empirically estimated generalization error is then 2.7% (accuracy), 9.8% (precision) and 19.4% (sensitivity).

Not highly optimized, but similar. This is not a great training example.

We know (part of) the reason Neural Networks generalize.

We know how (in one paradigm) the weights connect to the target function.

We can use our understanding of the Fourier representation of our target function to provide a prior on the initialization.

Applications of Fourier analytic Barron Space theory

Discussion of the Future

- Prototype exists that extends approximate Fourier transform to $d \sim 100$.
- Improving Fourier analysis is very important for application.
- Deep Neural Networks are similar but require more careful calculation.
  - There is a difference in the calculation for deep (2+ hidden layers) versus shallow. Only simple deep neural networks have been analyzed.
- Future theory step is extension to Recurrent Neural Networks (possibly easier to calculate than DNN).

Publications in preparation.