Math 147: Applications of Fourier Analysis in Artificial Intelligence

Brandon Choi*, Declan McGahan, Anthony Padilla, Ryan Sanchez

March 2025

1 Introduction

Spectral bias is a phenomenon observed within the Neural Network during the training process. Specifically, it describes the tendency for deep neural networks to fit lower frequencies faster than higher frequencies. For a function f(x) with a Fourier decomposition, this means a neural network will first learn the components for small k values (low frequency) before going onto the components for larger k values (high frequency).

$$f(x) = \sum_k \hat{f}(k) e^{ikx} \tag{1}$$

Deep Neural Networks are a type of artificial intelligence model that mimics how the human brain processes information. A Deep Neural Network consists of multiple layers of artificial neurons in between the input and output layers. These artificial neurons can learn patterns from data allowing the model to compute complex tasks such as image processing and speech recognition. Traditional machine learning relies on manual extraction of features while deep learning learns features automatically from the raw input data. The input layers of Deep Neural Networks receive the raw data and pass this to the hidden layers which perform any transformations needed and extract features from the data. Each neuron in the hidden layers will receive an input and apply a bias, or weighted sum, that determines the strength of the connection. The weight in each layer determines how much each frequency contributes to the final output, and each neuron has a bias that will shift the activation threshold.

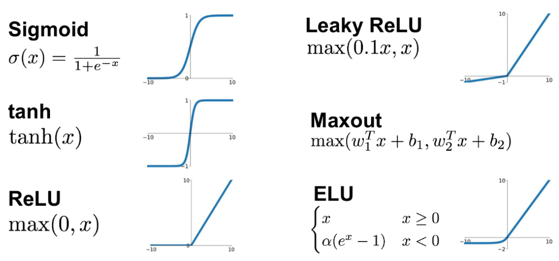


Figure 1: Common Activation Functions in Neural Networks

Activation functions introduce non-linearity to the data, allowing the network to learn more complex patterns from the data. Activation functions determine whether or not the artificial neuron produces an output signal to be passed onto the next layer. The Deep Neural Network is able to learn by adjusting these weights through a process called backpropagation. This is a supervised learning technique which requires labeled data and known desired outputs to compare with the actual output and adjust the weights accordingly.

1.1 What is Machine Learning?

Machine Learning (ML) is a field of study in artificial intelligence. What Machine Learning does is it allows a program to learn patterns from data and make independent decisions about that data without any explicit programming. The training that the artificial intelligence goes through can have guidance and reinforcement for correct evaluations of the data, or have no supervision at all. Emails being sorted between spam and non-spam and a self-driving car learning how to navigate in the safest manner are some examples of supervised learning. Unsupervised machine learning can be found in speech and facial recognition software and when summarizing large amounts of text to find key details within it.

2 Pre-Requisites

2.1 Mathematical framework of Machine Learning

Definitions

Hyperparameter

A **hyperparameter** is a parameter whose value is set before the learning process begins. Hyperparameters are typically set manually or through automated techniques, but not an optimized value such as weights in a machine learning model. They are not learned from the data but rather influence the learning process. Examples of hyperparameters in machine learning include learning rate, number of hidden layers in a neural network, and the regularization strength.

Loss Function

A **loss function** (or cost function) measures how well the model's predictions match the actual data. The goal of most machine learning models is to minimize the loss function, which in turn leads to better performance. Some common loss functions include Mean Squared Error (MSE) for regression tasks and Cross-Entropy Loss for classification tasks. The choice of loss function depends on the type of problem being solved.

Common Machine Learning Models

Below is a table summarizing some common machine learning models:

Model	**Description**
Logistic Regression	A linear model for binary classification.
K-Nearest Neighbors (KNN)	A non-parametric model that classifies based on proximity to neighbors.
Support Vector Machine (SVM)	A model that finds the hyperplane that best separates different classes.
Neural Network (NN)	A model inspired by the human brain, composed of layers of neurons.

Table 1: Common Machine Learning Models

2.1.1 Deep Neural Networks

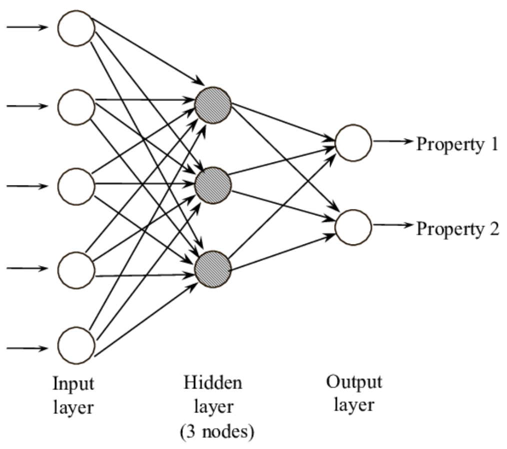


Figure 2: Neural Network Architecture

Neural Networks are one of the most powerful machine learning models, due to their intuition being the replication of human thought. A Neural Network consists of neurons, which take in some input x

and produce an output based on an activation function (as mentioned above). A neural network usually consists of many hidden layers, where these neurons take in their respective inputs and output values based on the layer's activation function. This is typically represented as:

$$z = Wh + b$$

where W is the weight of the neuron, h is the activation function output, and b is the bias.

One of the most common uses for Neural Networks is multi-class classification, where a Softmax Neural Network is used. The output of said model is a vector $\mathbf{y}$ containing the probability distribution over K classes. The softmax function is defined as:

$$y_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$$

where z_j is the raw score of a class j, and K is the total number of classes.

2.1.2 Loss Functions

Loss Functions are crucial in order to evaluate the performance of a model. During training, typically data is partitioned into training and validation data. Training data has the correct "labels", which are used to compare our models predicted label vs. the true label of the input. Here are common loss functions used throughout a myriad of models.

- **Mean Squared Error (MSE):**

  $$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

  Used in regression tasks.

- **Mean Absolute Error (MAE):**

  $$\text{MAE} = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i|$$

  Another common regression loss, less sensitive to outliers.

- **Binary Cross-Entropy:**

  $$L = -\frac{1}{N}\sum_{i=1}^{N}\left[y_i \log \hat{y}_i + (1 - y_i)\log(1 - \hat{y}_i)\right]$$

  Used for binary classification problems.

- **Categorical Cross-Entropy:**

  $$L = -\sum_{i=1}^{K} y_i \log \hat{y}_i$$

  Used for multi-class classification.

- **Hinge Loss:**

  $$L = \sum_{i=1}^{N} \max(0, 1 - y_i\hat{y}_i)$$

  Commonly used in Support Vector Machines (SVMs).

where y_i is the prediction label of our model, and $\hat{y}_i$ is the correct label. This function works for both discrete and continuous labels. For example, if classifying a dog as 1 and a cat as 0, we could have $1 - 1 = 0$, indicating an accurate decision, and $1 - 0 = 1$, incurring a loss. Our choice of loss-function constitutes as a hyperparameter.

2.1.3 Gradient Descent

Gradient descent is generally defined as:

$$W_{t+1} = W_t - \alpha \cdot \nabla L(W_t)$$

where $\nabla L(W_t)$ is the partial derivative of the loss function with respect to W_t, our weights. In machine learning, when optimizing models, what we optimize over is our weights. The loss function assists us with this optimization. By taking the partial derivative in respect to w, we are able to find the direction of greatest change, and take a "step" towards this direction, where the size of the step is controlled by the hyperparameter α. A larger alpha corresponds with a large step, and hence faster learning.

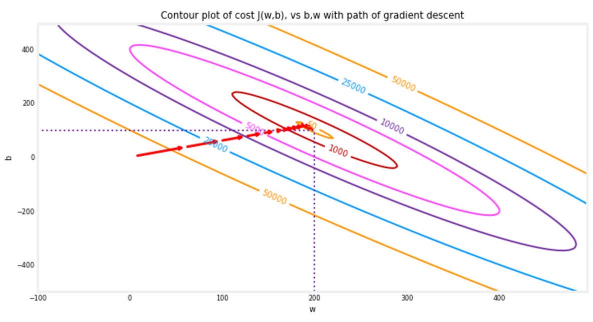


Figure 3: Contour Plot of A Loss Function Showing Gradient Descent

In the above figure, the red line traces the path of gradient descent towards an optimum, graphed on some an arbitrary loss function.

2.1.4 Activation Functions

Activation Function	**Formula**	**Use Case**
Sigmoid	$\sigma(x) = \frac{1}{1+e^{-x}}$	Binary Classification
Tanh	$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$	Hidden Layers, NLP
ReLU	$f(x) = \max(0, x)$	Deep Neural Networks
Leaky ReLU	$f(x) = \max(0.01x, x)$	Avoids dying ReLU problem
Softmax	$\sigma(x)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$	Multi-Class Classification

Table 2: Common Activation Functions in Neural Networks

3 Fourier Analysis of a ReLU Network

3.1 ReLU Networks are Piecewise Linear Functions

A function $f : \mathbb{R}^n \to \mathbb{R}$ is said to be a **Piecewise Linear (PWL) function** if there exists a finite collection of convex polyhedral regions $P_1, P_2, ..., P_k$ covering the domain such that for each region P_i, the function is affine. That is, for each i^{th} region, we can have $f_i(x) = A_i \cdot x + b_i$, for all $x \in P_i$ and $1 \leq i \leq k$ where $b_i \in \mathbb{R}$ and $A_i \in \mathbb{R}^n$. The definition of affine functions are

The representation of PWL functions in higher dimensions is rather complicated. Furthermore, a canonical representation of all higher dimensional CPWL functions is a recent result from works of Xingye et. al [12] from 2001. We can just observe a simple example in the $n = 1$ case:

$$f(x) = \begin{cases} x + 4, & \text{if } x < 1 \\ -x + 2, & \text{if } 1 \leq x < 3 \\ \frac{1}{2}x + \frac{3}{2}, & \text{if } x \geq 3 \end{cases}$$

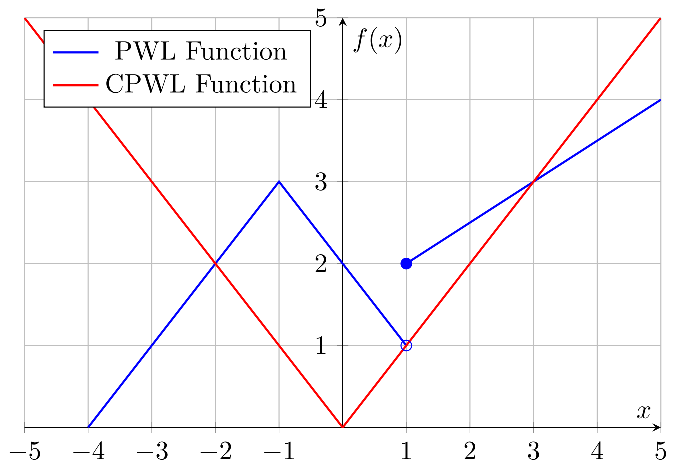


Figure 4: Example of a PWL function, in blue, and a CPWL function, in red.

The graph of this PWL is shown in Figure 4 along with an example **Continuous Piecewise Linear (CPWL)** function in red, $f(x) = |x|$. For a function to be CPWL, it simply requires that the boundaries of every adjacent regions be equal. In the $n = 1$ case, we need that

$$\lim_{x \to a_i^-} f_{i-1}(x) = \lim_{x \to a_i^+} f_i(x)$$

,where a_i represents the left boundary of the i^{th} region and right boundary of the $(i-1)^{th}$ region.

We follow the exact notation present in [14]. Consider the 'ReLU network' function $f : \mathbb{R}^n \to \mathbb{R}$ with L hidden layers of corresponding widths $d_1, d_2, ..., d_L$ and a single output neuron:

$$f(x) = (T^{(L+1)} \circ \sigma \circ T^{(L)} \circ \cdots \circ \sigma \circ T^{(1)})(x) \tag{2}$$

where each $T^{(k)} : \mathbb{R}^{d_{k-1}} \to \mathbb{R}^{d_k}$ is an affine function, $d_0 = d$ and $d_{L+1} = 1$, and $\sigma(\mathbf{u})_i = max(0, u_i)$ denotes the ReLU activation function acting component-wise on a vector $\mathbf{u} = (u_1, ..., u_n)$. In other words, $T^{(k)}(x) = W^{(k)}x + b^{(k)}$ for some weight matrix $W^{(k)} \in \mathbb{R}^{d_k \times d_{k-1}}$ and bias vector $b^{(k)} \in \mathbb{R}^{d_k}$.

Theorem 1 (Theorem 2.1 of [6])**.** *Every $\mathbb{R}^n \to \mathbb{R}$ ReLU DNN represents a piecewise linear function, and every piecewise linear function $\mathbb{R}^n \to \mathbb{R}$ can be represented by a ReLU DNN with at most $\lceil \log_2(n+1) \rceil + 1$ depth.*

Proof. For any ReLU network $f : \mathbb{R}^n \to \mathbb{R}$ with L hidden layers of corresponding widths $d_1, ..., d_L$ given by equation (2), it suffices to show that the ReLU network can be represented as a composition of piecewise affine functions, each with its respective partition of their domains. Then with the final output layer mapping $\mathbb{R}^{d_L}$ to $\mathbb{R}$ and the accumulation of partitions from the layers, we will see that the ReLU network is a piecewise linear function.

For any hidden layer k, let the function $h : \mathbb{R}^{d_{k-1}} \to \mathbb{R}^{d_k}$ be given by:

$$h^{(k)}(x) = (\sigma \circ T^{(k)})(x) = \sigma(W^{(k)}x + b^{(k)}) \tag{3}$$

Recall that the activation function is applied component-wise. We can define a activation pattern function (i.e. the indicator function extended to $\mathbb{R}^{d_k}$) $\mathbf{1}^{(k)}(x) = (\mathbf{1}_1^{(k)}(x), \mathbf{1}_2^{(k)}(x), \ldots, \mathbf{1}_{d_k}^{(k)}) \in \{0,1\}^{d_k}$ following familiar conditions of the activation function:

$$\mathbf{1}_i^{(k)}(x) = \begin{cases} 1, & \text{if } T_i^{(k)}(x) \geq 0 \\ 0, & \text{if } T_i^{(k)}(x) < 0 \end{cases} \tag{4}$$

Now for each binary vector $\beta \in \{0,1\}^{d_k}$, we define a partition of the domain given by:

$$P_\beta^{(k)} = \{x \in \mathbb{R}^{d_{k-1}} : \mathbf{1}^{(k)}(x) = \beta\} \tag{5}$$

This produces at most 2^{d_k} disjoint regions that cover $\mathbb{R}^{d_{k-1}}$. We can reformulate equation (3) as follows:

$$h_\beta^{(k)}(x) = D_\beta^{(k)}(W^{(k)}x + b^{(k)}) = D_\beta^{(k)}W^{(k)}x + D_\beta^{(k)}b^{(k)} \tag{6}$$

where $D_\beta^{(k)} : \mathbb{R}^{d_k} \to \mathbb{R}^{d_k}$ is a diagonal matrix with entries given by $D_\beta^{(k)} = diag(\beta_1, \beta_2, \dots, \beta_{d_k})$ i.e.

$$D_\beta = \begin{bmatrix} \beta_1 & & & \\ & \beta_2 & & \\ & & \ddots & \\ & & & \beta_{d_k} \end{bmatrix}$$

D_β masks the affine transformation, so that for any $x \in P_\beta^{(k)}$, $h^{(k)}(x)$ is precisely equal to $h_\beta^{(k)}(x)$. This allows us to define $h^{(k)}$ by:

$$h^{(k)}(x) = \sum_{\beta \in \{0,1\}^{d_k}} D_\beta^{(k)}W^{(k)}x + D_\beta^{(k)}b^{(k)} \tag{7}$$

Consider the base case $L = 1$. The network, with omission of the final layer, is given by:

$$h^{(1)}(x) = \sigma(W^{(1)}x + b^{(1)}), x \in \mathbb{R}^n$$

Since $\mathbf{1}^{(1)}(x) \in \{0,1\}^{d_1}$ and for every $\beta \in \{0,1\}^{d_1}$, we have the partition $P_\beta^{(1)} = \{x \in \mathbb{R}^n : \mathbf{1}^{(1)}(x) = \beta\}$, where for each partition an affine function

$$h_\beta^{(1)}(x) = D_\beta^{(1)}W^{(1)}x + D_\beta^{(1)}b^{(1)}$$

is defined and thus $h^{(1)} : \mathbb{R}^n \to \mathbb{R}^{d_1}$ is a piecewise affine function.

For the inductive step, we want to show that the composition of consecutive hidden layers is piecewise affine. Assume that a function $g : \mathbb{R}^n \to \mathbb{R}^{d_k}$ with k hidden layers is piecewise affine. Then for the $(k+1)$ layer, we have

$$(h^{(k+1)} \circ g)(x) = \sum_{\beta \in \{0,1\}^{d_{k+1}}} D_\beta^{(k+1)}W^{(k+1)}g^{(k)}(x) + D_\beta^{(k+1)}b^{(k+1)}$$

The corresponding activation pattern function $\mathbf{1}^{(k+1)} \in \{0,1\}^{d_{k+1}}$ for every $\beta \in \{0,1\}^{d_{k+1}}$ gives the partition $P_\beta^{(k+1)} = \{x \in \mathbb{R}^{d_k} : \mathbf{1}^{(k+1)}(x) = \beta\}$. By construction, the $(k+1)$ layer partitions the range of g, $\mathbb{R}^{d_k}$ into d_{k+1} disjoint regions that cover $\mathbb{R}^{d_k}$ where for each an affine function is defined. Thus, $h^{(k+1)}$ is an affine piecewise function, as desired.

By induction, the function computed by the network with L hidden layers is piecewise affine. Finally, since the final output layer $T^{(L+1)} : \mathbb{R}^{d_L} \to \mathbb{R}$ is affine, the entire network $f : \mathbb{R}^n \to \mathbb{R}$ is piecewise affine and hence piecewise linear.

The second part of the proof is given as a sketch in Theorem 2.1 of [6]. □

3.1.1 Additional Discussion of ReLU Networks

1. **Compositional Structure and Hierarchical Partitioning**

   In the proof of **Theorem 1**, we show that each layer introduces at most 2^{d_k} new partitions. So each subsequent composition of layers introduces an exponential number of hierarchal partitioning of the domain. This accurately reflects the phenomenon of how deeper networks can be more expressive than shallower ones, allowing even networks of moderate width to represent highly complex functions by increasing the depth.

   More explicitly, Montúfar et al. [13] show that multiple regions of, say, layer k are mapped to a common region in layer $k+1$. These repetitions show that the maximal number of linear regions of a ReLU network can be given by recursively counting the number of repetitive sets of each subsequent layer for every region stemming from the domain. The analogy given by Montúfar relates to folding a piece of paper along the hyperplane where the overlapping regions represent the repetitive regions.

2. **Universal Approximation of ReLU Networks**
   It has been known that ReLU networks can memorize (i.e., perfectly fit) any arbitrary dataset given enough width and depth. In particular, Hornik et al. [9] established that a Neural Network with depth even as low as 1, equipped with a squashing activation function (e.g., a function that limits the range like $\mathbb{R} \to [0,1]$), is capable of approximating any Borel-measurable function given enough width. Note that a ReLU activation function can be adjusted to replicate a squashing function like $max(0,x) - max(0,x-1)$, although this particular shape may not occur naturally through training. Nevertheless, Hornik shows that Neural Networks are a class of universal approximators.

   Current studies in this line of research seek to answer what the bounds are for the width of a 2 layer deep network while retaining universal approximation properties. More generally, what are the bounds of the width in relation to the depth and vice versa. A particular study by Kim et al. [11] shows that, in a compact domain, the minimum width for the approximation L^p of L^p functions from $[0,1]^{d_x}$ to $\mathbb{R}^{d_y}$ is exactly $max\{d_x, d_y, 2\}$. In other words, ReLU networks with the mentioned minimum widths are dense in $L^p([0,1]^{d_x}, \mathbb{R}^{d_y})$.

3.2 Derivation for Fourier Transforms of ReLU Networks

A particular derivation by Rahaman et al. [14] follows a construction similar to the proof of Theorem 1. They state that each linear region of the network corresponds to a unique activation pattern (i.e., the fact that all elements from a particular region corresponds to only one binary vector where the components are 1 if it is active (positive) and 0 if inactive (negative)). Then the original equation (2) can be rewritten as the sum over all possible activation patterns, which is given in Lemma 3 in [14]. Their construction of the function is distinct from ours in the way that instead of *explicitly* using a masking matrix, they keep the original weight matrix and set the inactive column to all 0.

Rahaman et al. considers two cases as to whether the function f has or does not have compact support. Recall that the Fourier transform is given by:

$$\hat{f}(\mathbf{k}) = \int_{\mathbb{R}^d} f(x) e^{-i\mathbf{k}\cdot\mathbf{x}} \mathbf{dx}$$

For the first case, integration by parts is directly applied which utilizes Stokes' Theorem on the derivative of the exponential factor. Since we assume that f has compact support, the boundary term vanishes yielding:

$$\int \nabla_{\mathbf{x}} \cdot [\mathbf{k} f(x) e^{-i\mathbf{k}\cdot\mathbf{x}}] \mathbf{dx} = 0 \implies \hat{f}(\mathbf{k}) = \frac{1}{-ik^2} \mathbf{k} \cdot \int (\nabla_{\mathbf{x}})(\mathbf{x}) e^{-i\mathbf{k}\cdot\mathbf{x}}$$

Since f is affine, the gradient is a constant vector, $\nabla_{\mathbf{x}} f_\epsilon = W_\epsilon^T$, where ϵ is how the author denotes a particular region.

For the second case without compact support, f is not square integrable and so the Fourier transform only exists in the sense of tempered distributions. A tempered distribution T_f is a continuous linear functional on S given by:

$$T_f : S \to \mathbb{R}, \varphi \mapsto \langle f, \varphi \rangle = \int_{\mathbb{R}^d} f(\mathbf{x}) \varphi(\mathbf{x}) \mathbf{dx}$$

Author then identifies T_f with f and defines the Fourier transform $\tilde{f}$ as:

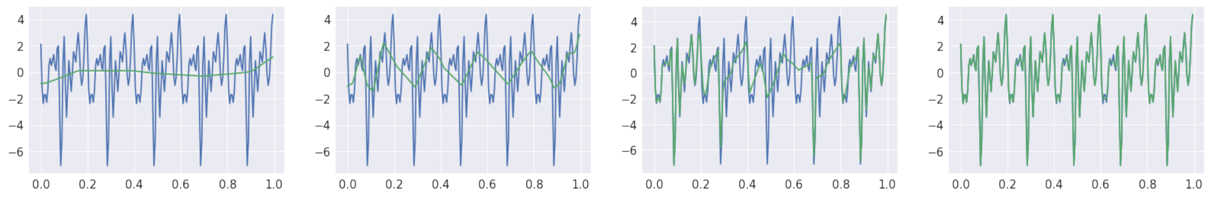
$$\langle \tilde{f}, \varphi \rangle := \langle f, \tilde{\varphi} \rangle$$

where $\tilde{\varphi}$ is the Fourier transform of φ. Since the ReLU network can grow at most linearly, the author interprets it as a tempered distribution on $\mathbb{R}^d$. The distributional derivative of f where f_ϵ is the restriction of f to a particular region enumerated by ϵ is then given as:

$$\nabla_{\mathbf{x}} f = \sum_{\epsilon} \nabla_{\mathbf{x}} f_\epsilon \cdot 1_{P_\epsilon} = \sum_{\epsilon} W_\epsilon^T \mathbf{x}$$

In particular, the boundary terms did not vanish in the classical sense, but vanish when interpreted by the above relation. The same integration by parts is carried out for these elements of Schwartz spaces:

$$[\widetilde{\nabla_{\mathbf{x}} f}](\mathbf{k}) = -i\mathbf{k}\tilde{f}(\mathbf{k}) \implies \tilde{f}(\mathbf{k}) = \frac{1}{-ik^2} \mathbf{k} \cdot [\widetilde{\nabla_{\mathbf{x}} f}](\mathbf{k})$$



(a) Iteration 100 (b) Iteration 1000 (c) Iteration 10000 (d) Iteration 80000

Figure 5: From [14]. The learnt function (green) overlayed on the target function (blue) as the training progresses. The target function is a superposition of sinusoids of frequencies $\kappa = (5, 10, ..., 45, 50)$, equal amplitudes and randomly sampled phases.

, as desired.

With case 1 and 2, the Fourier transform of ReLU networks decomposes as,

$$\tilde{f}(\mathbf{k}) = i \sum_{\epsilon} \frac{W_\epsilon \mathbf{k}}{k^2} \tilde{1}_{P_\epsilon}(\mathbf{k}) \tag{8}$$

where $k = ||\mathbf{k}||$ and the Fourier transform of the indicator function of P is $\tilde{1}_P(\mathbf{k}) = \int_P e^{-i\mathbf{k}\cdot\mathbf{x}}\mathbf{dx}$.

3.3 Learning Low Frequencies First

Several papers [14], [18], [8], provide empirical and theoretical evidence supporting the phenomenon of how neural networks have a tendency to learn higher frequencies *exponentially slower than* lower frequencies. In other words, neural networks are biased towards smooth function. In particular, Rahaman et al. demonstrated spectral bias by having a 6-layer deep and 256-unit wide ReLU network trained to learn $\lambda : [0, 1] \to \mathbb{R}$ given by:

$$\lambda(z) = \sum_i A_i sin(2\pi k_i z + \varphi)$$

with frequencies $K = (k_1, k_2, \dots)$, amplitudes $\alpha = (A_1, A_2, \dots)$, and phases $\phi = (\varphi_1, \varphi_2, \dots)$. Figure 5 shows that lower frequencies are learned regardless of amplitude. Notice that the network needed an exponential number of iterations for it to effectively learn higher frequencies.

This behavior has been observed across many architectures and datasets. One example that more resembles a real-life case is work done by Xu et al. [18]. They work with a image dataset, specifically MNIST/CIFAR10, employing the VGG16 architecture, a type of a convolutional neural network designed for image classification tasks, and several loss functions. The phenomenon is consistent throughout their experiments. Additionally, the learned high-frequency components of the target function are found to be less robust, in the sense of performance, than lower-frequency counterparts. Meaning, small random perturbations in network parameters destroy high-frequency output while barely affecting low-frequency outputs.

3.4 Motivations for Understanding Spectral Bias

Spectral bias reveals fundamental relationships between the structure of the neural network and its learning process. In particular, it helps explain why over-parameterized networks tend to *generalize* well despite having enough capacity to memorize the entire dataset. That is to say, the bias of the network towards learning a smooth function produces a consequent difficulty to learn complicated functions. The spectral bias acts as a double edged sword in the case of adversarial attacks. Carefully crafting high frequency noises can easily degrade performance and fool the network. This phenomenon has been largely a open question [2], which spectral bias is a promising step towards understanding this.

Cao et al. [7], in the Neural Tangent Kernel (NTK) regime, provide a rigorous explanation for spectral bias. They showed that the error terms from different frequencies are controlled by the eigenvalues of the NTK. In particular, given enough width and sample size, the neural network trained through gradient descent learns the target function along the eigendirections of the NTK with larger eigenvalues,and then the components corresponding to smaller eigenvalues. Additionally, lower-frequency components (larger eigenvalues) can learn better than higher-frequency components (smaller eigenvalues) with less data and smaller widths. As such, Cao et al. believe that the difficulty to learn can be characterized in the eigenspace of the NTK.

3.5 Computing the Spectral Bias

Works done by Kiessling and Thor [10] focus on two computations. First method is given by estimating the Fourier transforms of the difference between the target and output of the neural network with discrete Fourier transforms. The quotient is given as $SB = (\mathcal{E}_{high} - \mathcal{E}_{low})/(\mathcal{E}_{high} + \mathcal{E}_{low})$ where $\mathcal{E}_{low}$ and $\mathcal{E}_{high}$ are the frequency errors. The frequency errors are then estimated using a simple quadrature rule. The second method avoids direct computation of the Fourier transform and relies on convolution with the Fourier transform of the indicator function and Monte Carlo quadrature for the estimation of the frequency errors. Calculating the spectral bias by splitting errors of low and high frequencies allows for a particular numerical analysis. Their results indicate that a spectral bias of 0 implies that the neural network predicts low and high frequencies with equal accuracy. A positive spectral bias implies that the error for low frequencies is smaller than high frequencies. Some immediate limitations of this perspective is that the spectrum of the reconstructed function is unknown since it only depends on relative sizes of the errors of high and low frequencies.

Both direct and indirect methods come with their respective pros and cons. The direct computation via FFT has computational complexity which increases exponentially with dimension as given by $\mathcal{O}(N^d \log(N^d))$. This makes method 1 efficient for lower dimensions but unfeasible for larger dimensions. The complexity of method 2 is $\mathcal{O}(dN^2)$, which in dimension 1 is substantially larger than method 1. However, the complexity of Monte Carlo integration scales linearly with d, avoiding the curse of dimensionality seen in method 1.

4 Extended Topics and Related Works

4.1 Neural Tangent Kernel (NTK)

In machine learning, a **kernel** is a function that helps transform data into a higher-dimensional space to make it easier to analyze. Some datasets will not be linearly separable in their original space, which is where kernels come in to play. Instead of the kernels explicitly transforming the data, which can be a lengthy process, the kernels can take a shortcut with these transformations. This kernel trick will compute the dot product between two transformed points without actually transforming them. Normally, mapping data into higher dimensions is a tedious process with many steps, but the kernel trick makes this computational process much more efficient.

There are different types of kernel formulas depending on whether or not the dataset is linearly separable and what degree the space is in. When the space is of a small degree and the data is linearly separable, the equation is relatively simple. The input vectors, x and y, are used by the kernel to compute the dot product between the two and measure their similarity. This formula is used for simple tasks such as text classification.

$$K(x, y) = x^T y \tag{9}$$

When the kernels begin to map datasets into a higher dimension, it becomes a polynomial kernel. The same equation is used except with an added constant and it is raised to the degree of the polynomial. This formula is used when trying to find and measure complex relationships in images.

$$K(x, y) = (x^T y + c)^d \tag{10}$$

A Gaussian kernel, or radial basis function, is one used for infinite-dimensional spaces. Within an infinite-dimensional space, the similarity between data points is measured with distance where close points have a value near one and far apart points have a value near zero. The numerator of this formula represents the squared Euclidean distance between input vectors x and y. The denominator of this formula is a parameter which controls how much influence a single data point has, with larger sigma values resulting in smoother decision boundaries. Gaussian Kernels are used in programs that carry out image and speech recognition.

$$K(x, y) = e^{-\frac{\|x-y\|^2}{2\sigma^2}} \tag{11}$$

A sigmoid kernel, or hyperbolic tangent kernel, is a kernel that behaves like an activation function. This type of kernel is similar to the Gaussian kernels in terms of being in an infinite-dimensional space. The difference with this type of kernel is the fact that this formula applies the hyperbolic tangent function to the dot product of the input vectors which maps the data into a non-linear space with a sigmoid-shaped curve. Alpha and beta are parameters which control the shape of the sigmoid-shaped curve. This type of kernel function can be used for many different tasks and is useful for classifying binary data with clear patterns. Although, due to the sensitivity of the parameters requiring careful tuning, it is often times less preferred when compared to other kernels like the radial bias function.

$$K(x, y) = \tanh(\alpha x^T y + c) \tag{12}$$

A **Neural Tangent Kernel** is a unique type of kernel which helps describe how infinitely wide neural networks behave and change during training with gradient descent. While deep neural networks are usually non-linear, neural tangent kernels use kernel methods to make their training behavior predictable and analyzable.

4.2 From Vin(t) to Vin(f): Passive Components and Neural Signal Processing

4.2.1 Role of Components in Analog Neural Networks

In analog neural networks, the behavior of signals such as charge or voltage is shaped by passive components; resistors (R), capacitors (C), and inductors (L). These components filter or modify the frequency of an input signal (Vin(t)) based on their impedance, which depends on frequency. This frequency filtering is key to processing signals, mimicking how neurons might process inputs. This section will delve deeper into the circuit theory behind these components, exploring how their frequency response can be analyzed using the Fourier transform to bridge the time domain (Vin(t)) to the frequency domain (Vin(f)), and how this process enhances signal processing in analog neural networks.

- Capacitors: $Z_C = 1/(j2\pi fC)$
- Inductors: $Z_L = j2\pi fL$, with cutoff frequency $f_c = R/(2\pi L)$
- Resistors: $Z_R = R$

4.2.2 Frequency Filtering and Neural Mimicry

Paper [16] demonstrates how capacitors in a neuro-transistor-based network integrate and fire signals, effectively filtering Vin(t) to mimic neuron behavior, while Paper [4] shows how RL circuits act as low-pass filters, passing low frequencies and blocking high ones, further supporting this neural mimicry [16, 4]. These properties show that analog neural networks can replicate the selective signal processing observed in biological systems. In a series RLC configuration, the circuit acts as a band pass filter, allowing only a specific range of frequencies to pass while minimizing others. The parallel RLC setup might function as a band reject filter, blocking certain frequencies; simpler RC or RL circuits can serve as low pass or high pass filters, based on their impedance characteristics.

For example, from [3] an RL series circuit is driven by a voltage source, such as a battery or an AC power supply, which supplies the electrical potential required for its operation. When the output is measured across the resistor, the circuit functions as a low-pass filter.

4.2.3 Fourier Transform Analysis: Vin(t) to V

The Fourier transform maps a time domain signal $V_{in}(t)$ to its frequency domain spectrum $V_{in}(f)$ revealing how RLC circuits filter specific frequencies. RLC circuits filter signals based on their impedance, which varies with frequency. For a resistor, impedance is $Z_R = R$, for a capacitor, $Z_C = \frac{1}{j2\pi fC}$, and for an inductor, $Z_L = j2\pi fL$, where j is the imaginary, f is frequency, R is resistance, C is capacitance, and L is inductance [3]. In a series RLC circuit (band-pass filter), the total impedance is:

$$Z = R + j\left(2\pi fL - \frac{1}{2\pi fC}\right) \tag{13}$$

This impedance minimizes at the resonant frequency, $f_0 = \frac{1}{2\pi\sqrt{LC}}$, allowing frequencies near f_0 to pass while minimizing others. In a parallel RLC circuit (band-reject filter), the impedance peaks at f_0, blocking that frequency [5].

An RC low pass filter, with a cutoff frequency $f_c = \frac{1}{2\pi RC}$ minimizes high frequencies like 1,000 Hz while passing 100 Hz. Conversely, an RL high-pass filter, with $f_c = \frac{R}{2\pi L}$, minimizes low frequencies like 100 Hz while passing 1,000 Hz. For $V_{in}(t)$, these filters reshape $V_{in}(f)$, reducing peak amplitudes beyond f_c based on the transfer function $H(f) = \frac{V_{out}(f)}{V_{in}(f)}$ [3].

This behavior underpins signal processing, where impedance dictates frequency selectivity. Research like in [5] quantifies amplitude minimization and phase shifts in $V_{in}(f)$, generalizing RLC filtering.

5 Fourier Transforms in Audio Classification

5.1 Introduction

The Short-Time Fourier Transform (STFT) [1] is a widely used technique for feature extraction in audio signal classification. The motivation behind using STFT stems from the limitation of the traditional Fourier Transform (FT), which answers the question of *what frequencies are present* but not *when* they occur. By introducing a window function, we introduce hyperparameters that help obtain a more informative decomposition of the signal.

Unlike the standard Fourier Transform, which outputs a one-dimensional spectral vector, the STFT produces a *spectral matrix*, consisting of frequency bins and indexed time frames.

5.2 Definition of STFT

Let’s define the Discrete Fourier Transform (DFT) as [15]:

$$\hat{x}(k) = \sum_{n=0}^{N-1} x(n) e^{-i2\pi n \frac{k}{N}} \tag{14}$$

Then, the Short-Time Fourier Transform (STFT) is given by:

$$STFT(m,k) = \sum_{n=0}^{N-1} x(n+mH) w(n) e^{-i2\pi n \frac{k}{N}} \tag{15}$$

where:

- H represents the Hop Size,
- m represents the frame index,
- $w(n)$ is the window function.

The windowing function is typically defined as:

$$w(k) = 0.5\left(1 - \cos\left(\frac{2\pi k}{K-1}\right)\right), \quad \text{for } k = 1, \ldots, K. \tag{16}$$

The Hop Size is the distance the window "shifts" every iteration, which can be altered to overlap with a previous window, or be disjoint. m, the frame index, is simply the current frame we are analyzing at a given value m. Finally, $w(n)$ is the window function that modulates the rest of the signal outside of the frame of interest. As mentioned before, the output of STFT(m,k) is a two-dimensional matrix. Typically, the squared magnitude of the output matrix is taken, to obtain a $Y(m,k)$, a matrix that will now consist of real-valued coefficients that can be used in itself, as a feature. This looks like:

$$Y(m,k) = |\text{STFT}(m,k)|^2$$

Typically, $Y(m,k)$ is used in its entirety as an input feature into neural networks, spanning the entire matrix. The input matrix X is defined as $X = \{S(m,k)\}_{m=1}^{M}, k = 1, \ldots, K$

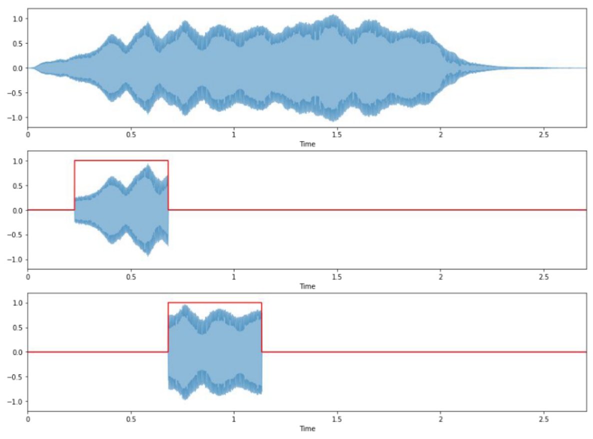


Figure 6: Capturing Frames of a signal x(n)

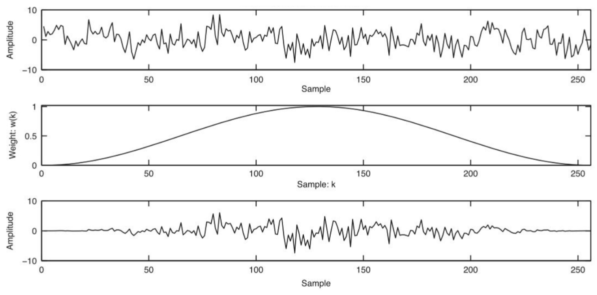


Figure 7: A window function w(k) being applied to some signal function for frame capture

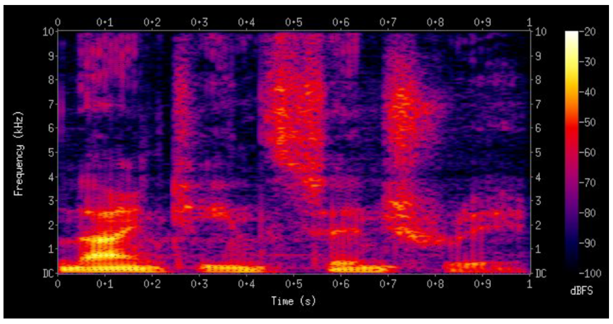


Figure 8: Spectrogram Output of STFT for use as a input

5.3 Parameters of STFT

The STFT takes in three key parameters:

- **Original Signal** $x(n)$: The input audio signal.
- **Frame Size**: Defines the length of each segment of the signal that undergoes Fourier Transform.
- **Hop Size**: Determines the step size for shifting the window across the signal.

The number of frequency bins is defined as:

$$\frac{\text{Frame Size}}{2} + 1 \tag{17}$$

The number of frames is defined as:

$$\frac{\text{Samples} - \text{Frame Size}}{\text{Hop Size}} + 1 \tag{18}$$

Thus, these two aspects are determined by our hyperparameters, being the choice of frame size and hop size.

5.4 Time-Frequency Tradeoff

However, the STFT introduces a fundamental limitation known as the *time-frequency tradeoff*. This arises because the frame size acts as a proxy for time resolution:

- Increasing the frequency resolution results in a decrease in time resolution.
- Conversely, decreasing the frequency resolution improves time resolution.

The time-frequency tradeoff comes from the fact that time and frequency are inversely proportional to the frame size/window duration. For example, let:

$$\Delta t \cdot \Delta f \geq T \tag{19}$$

where T represents some window duration. If we decrease Δt, then Δf (the width of frequency bins) will increase, and vice versa. This motivates optimizing the choice of window duration.

This tradeoff necessitates a balance between time and frequency resolution, depending on the specific requirements of the classification task.

5.5 STFT Parameter Optimization

As mentioned before, it is possible to use Gradient Descent with the goal of finding the optimal frame size and hop size. It would be similar to optimizing two parameters in typical gradient descent, motivated by the Time-Frequency Tradeoff. To perform gradient descent, [17] introduces a slightly different notation for the STFT, which is:

$$\text{STFT}(m,k) = \sum_{n=-\frac{N}{2}}^{\frac{N}{2}} x[m+n] \cdot w(n) \cdot e^{-i2\pi \frac{N}{k} n}$$

In this case, we have a constant-sized window.

This opens up the ability to consider two cases:

1. Optimizing assuming chosen STFT parameters are constant
2. Adapting parameters over time

Each has its own use case. However, Gradient Descent uses the idea of partial differentiation in order to take "steps" in a direction to minimize loss. In our case, the parameters for optimization (frame size and hop size) are *not* continuous, therefore the window function must be altered. In [17], they propose defining the window function as:

$$w(n) = \exp\left(-\left(\frac{n}{2\delta}\right)^2\right),$$

using δ as a proxy for the window length.

Using this new window function in substitution in the STFT function, a classifier model K can be used to perform a prediction based on the output of $\mathrm{STFT}(m, K)$, for class i and time m. If we define

$$z_i[m] = K(\mathrm{STFT}[m]),$$

then the new loss function that could be used for optimization would be defined as:

$$L = -\sum_m \sum_i t_i[m] \log(z_i[m]) + \text{Regularization Term.}$$

6 Conclusion

In the age where there is a strong desire to make huge advances in machine learning, and artificial intelligence, it is more important than ever to understand at its core, its identity lies within mathematics. Therefore, there is value in tackling the concept with multiple fields of mathematics, and as we have discussed here, extreme value in viewing machine learning through the lens of Fourier analysis. The field as it stands produces great results, but also brings with it an equal number of questions that, if answered, can push the field even further. Fourier analysis can shine light into the "black-box" that is machine learning, explaining biases in models, being used as a data pre-processing technique, or to explore kernels, a pivotal pillar when it comes to any classification task. It reminds us that, while all these fields may be separate, they can come together in substantial ways.

References

[1] Jont B. Allen. A unified approach to short-time fourier analysis and synthesis. *Proceedings of the IEEE*, 65(11):1558–1564, 1977.

[2] Zeyuan Allen-Zhu, Yuanzhi Li, and Yingyu Liang. Learning and generalization in overparameterized neural networks, going beyond two layers, 2020.

[3] Anonymous. Cct1 book (9th edition), 2025. Accessed: 2025-03-16.

[4] Anonymous. Rl circuit, 2025. Accessed: 2025-03-16.

[5] Anonymous. Unknown book title, 2025. Accessed: 2025-03-16.

[6] Raman Arora, Amitabh Basu, Poorya Mianjy, and Anirbit Mukherjee. Understanding deep neural networks with rectified linear units. *CoRR*, abs/1611.01491, 2016.

[7] Yuan Cao, Zhiying Fang, Yue Wu, Ding-Xuan Zhou, and Quanquan Gu. Towards understanding the spectral bias of deep learning, 2020.

[8] Sara Fridovich-Keil, Raphael Gontijo-Lopes, and Rebecca Roelofs. Spectral bias in practice: The role of function frequency in generalization, 2022.

[9] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

[10] Jonas Kiessling and Filip Thor. A computable definition of the spectral bias. *Proceedings of the AAAI Conference on Artificial Intelligence*, 36(7):7168–7175, Jun. 2022.

[11] Namjun Kim, Chanho Min, and Sejun Park. Minimum width for universal approximation using relu networks on compact domain, 2024.

[12] Xingye Li, Shuning Wang, and Wenjun Yin. A canonical representation of high-dimensional continuous piecewise-linear functions. *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, 48(11):1347–1351, 2001.

[13] Guido Montúfar, Razvan Pascanu, Kyunghyun Cho, and Yoshua Bengio. On the number of linear regions of deep neural networks, 2014.

[14] 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.

[15] Valerio Velardo. Short-time fourier transform explained easily, 2020.

[16] Zhongrui Wang, Mingyi Rao, Jin-Woo Han, Jiaming Zhang, Peng Lin, Yunning Li, Can Li, Wenhao Song, Shiva Asapu, Rivu Midya, Ye Zhuo, Hao Jiang, Jung Ho Yoon, Navnidhi Upadhyay, Saumil Joshi, Miao Hu, John Paul Strachan, Mark Barnell, Qing wu, and Jianhua Joshua Yang. Capacitive neural network with neuro-transistors. *Nature Communications*, 9, 08 2018.

[17] An Zhao, Krishna Subramani, and Paris Smaragdis. Optimizing short-time fourier transform parameters via gradient descent. In *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 311–315. IEEE, 2020.

[18] Zhi-Qin John Xu Zhi-Qin John Xu, Yaoyu Zhang Yaoyu Zhang, Tao Luo Tao Luo, Yanyang Xiao Yanyang Xiao, and Zheng Ma Zheng Ma. Frequency principle: Fourier analysis sheds light on deep neural networks. *Communications in Computational Physics*, 28(5):1746–1767, January 2020.