On the Equivalence of Convolutional and Hadamard Networks using DFT

Marcel Crasmaru
October 30, 2018
crasmarum@gmail.com

Abstract
In this paper we introduce activation functions that move the entire computation of Convolutional Networks into the frequency domain, where they are actually Hadamard Networks. To achieve this result we employ the properties of Discrete Fourier Transform.

We present some implementation details and experimental results, as well as some insights into why convolutional networks perform well in learning use cases.

1 Introduction
Convolutional networks use real numbers and operate in the spatial domain, although their defining mathematical operation, that is, the convolution, is easier to compute and understand after applying the Discrete Fourier Transform (DFT), $F_N$, and moving the computation to the frequency domain. For example, as you can see in Figure 1, the first layer is convolving the input $X$ with three filters $W_1, W_2, W_3$. The output of the first layer $(X * W_1, X * W_2, X * W_3)$ is then forwarded to an activation layer, in this case ReLU.

In order to move the entire computation to the frequency domain we could apply the DFT $F_N$ to the input $X$ and to the filters. We can then use the fact that $F_N(X * W) = F_N(X) * F_N(W)$, that is, in the frequency domain convolutions become the entrywise product of some complex numbers, which is known as the Hadamard product (see Figure 2).

In addition to simplifying the convolution calculations, there are many good reasons for researching the convolutional network computation in the frequency domain and use complex numbers for weights [2]. One compelling reason is that real-valued networks cannot properly get close to the optimum on certain sets that are connected in the complex space but not in the real one [3]. Moreover, many researchers have observed that nets using complex values for weights have a more robust and stable behavior in training [4]. We could also mention some recent Computational Neuroscience studies [1], that suggest that the entorhinal cortex might employ some sort of Fourier transform in its functionality.

The first goal of our paper is to investigate whether we can create convolutional networks that operate entirely in the frequency domain and that have all their layers “commute” with the discrete Fourier transform in the same way as convolutions. For example, one of the issues we have to address is the fact that the currently used activation functions, for example, ReLU, do not get mapped nicely to the frequency domain. What we would like to find is a pair of easy-to-compute activation functions $A$ and $A'$ such that $F_N(A(x)) = A'(F_N(x))$.

In Section 3.5, we will introduce a pair $A_N, A'_N$ of activation functions that commute with the DFT. We will also show that all other widely used layers (fully connected, residual, batch normalization) are commutable with the DFT.

The second goal is to test that these convolutional networks converge for some known standard data sets, for example, MNIST, and then see if we can use Fourier analysis to get some insights on some hard open problems: why are
convolutional networks working well, is there a good reason why deep networks generalize better than the shallow ones, etc.

Experimental results show that a specific shallow network using $A(N)$, as an activation function converges fast in the frequency domain: we got 90% accuracy on MNIST after 70 epochs of Wirtinger gradient descent back-propagation. This experimental result confirms the findings in [7] and [12], that is, $A(N)$ is bounded almost everywhere but not holomorphic we should expect that feedforward networks using this activation function are capable of universally approximating any non-linear complex mapping.

In this paper we show that any convolutional network built with layers that commute with the DFT is equivalent to a Hadamard network. That is, the known technique of classifying patterns from input $x$ using convolutional networks is equivalent to the technique of extracting the recurring information from the input $x$ by applying DFT $F_N(x)$ then feeding it to a Hadamard network.

We propose that this equivalence might explain why convolutional networks work well in areas such as image recognition. This equivalence also suggests that predictive models using complex numbers for weights should perform better than the ones using only real numbers.

2 Preliminaries

2.1 The Discrete Fourier Transform

Let $N$ be an integer and suppose that $x = (x_0, \ldots, x_{N-1})$ is an $N$ dimensional complex vector. Let $\omega = \exp(2\pi i/N)$. Then the Discrete Fourier Transform (DFT) $F_N: \mathbb{C}^N \rightarrow \mathbb{C}^N$ is given by

$$X_k = F_N(x)_k = \sum_{j=0}^{N-1} x_j \omega^{jk}. \quad (1)$$

The inverse DFT $F_N^{-1}: \mathbb{C}^N \rightarrow \mathbb{C}^N$ is then

$$x_k = F_N^{-1}(X)_k = \frac{1}{N} \sum_{j=0}^{N-1} X_j \omega^{-jk}. \quad (2)$$

The DFT is a linear function, that is

$$F_N(\alpha x + \beta y) = \alpha F_N(x) + \beta F_N(y). \quad (3)$$

Parseval Theorem states that if $x, w \in \mathbb{C}^N$ then the following equality holds:

$$\sum_{j=0}^{N-1} x_j \bar{w}_j = \frac{1}{N} \sum_{j=0}^{N-1} F_N(x)_j \bar{F_N(w)}_j. \quad (4)$$

From this theorem one can easily prove the Plancherel Theorem that states that if $x, X \in \mathbb{C}^N$ and $X = F_N(x)$ then

$$\sum_{j=0}^{N-1} |x_j|^2 = \frac{1}{N} \sum_{j=0}^{N-1} |X_j|^2 \quad (5)$$

and this is equivalent to

$$\|x\|_2 = \frac{1}{\sqrt{N}} \|F_N(x)\|_2 \quad (6)$$

where $\|\cdot\|_2 = \sqrt{\sum_{j=0}^{N-1} x_j^2}$.

The circular convolution $x \star y \in \mathbb{C}^N$ of two vectors $x, y \in \mathbb{C}^N$ is defined by

$$(x \star y)_k = \sum_{j=0}^{N-1} x_j y_{k-j} \mod N. \quad (7)$$

The convolution property of the Fourier transform is

$$F_N(x \star y) = F_N(x) \cdot F_N(y) \quad (7)$$

where the multiplication indicated by the dot is element-wise.

2.2 Wirtinger Derivatives and Gradients

Given a function $g: \mathbb{C} \rightarrow \mathbb{C}$ differentiable in the real sense, its Wirtinger derivative (equivalent to the areolar derivative first introduced by Pompeiu in [8]) at $c \in \mathbb{C}$ is defined by:

$$\frac{\partial g}{\partial z}(c) = \frac{1}{2} \left( \frac{\partial g}{\partial x}(c) - i \frac{\partial g}{\partial y}(c) \right).$$

Its conjugate Wirtinger derivative is defined by:

$$\frac{\partial \bar{g}}{\partial \bar{z}}(c) = \frac{1}{2} \left( \frac{\partial \bar{g}}{\partial x}(c) + i \frac{\partial \bar{g}}{\partial y}(c) \right).$$

Note that a function $g(z) = u(x, y) + iv(x, y)$ is holomorphic (thus satisfying the Cauchy Riemann equation $u_x = v_y, u_y = -v_x$) if and only if its conjugate Wirtinger derivative vanishes, that is, $\frac{\partial \bar{g}}{\partial \bar{z}}(c) = 0$.

If we denote the conjugate of the complex number $z$ by $\bar{z}$ then the following relations hold:

$$\frac{\partial g}{\partial \bar{z}}(c) = \frac{\partial g}{\partial \bar{z}}(c) = \frac{\partial \bar{g}}{\partial z}(c).$$

For a function $f: \mathbb{C}^N \rightarrow \mathbb{C}$ that is differentiable with respect to the real and imaginary parts of all of its inputs,
the conjugate gradient $\nabla f(c)$ is defined similarly to its real counterpart:

$$\nabla f(c) = \left( \frac{\partial f}{\partial x_1}(c), \ldots, \frac{\partial f}{\partial x_N}(c) \right).$$

For any real-valued function (such as a cost function) knowing its $\nabla f(c)$ gradient is a sufficient condition to minimize it, see, for example [4].

The chain rules for two functions $f, g: \mathbb{C} \to \mathbb{C}$ that are differentiable in the real sense, are:

$$\frac{\partial (f \circ g)}{\partial z} = \left( \frac{\partial f}{\partial \overline{z}} \circ g \right) \cdot \frac{\partial g}{\partial z} + \left( \frac{\partial f}{\partial z} \circ g \right) \cdot \frac{\partial \overline{g}}{\partial \overline{z}},$$

$$\frac{\partial f \circ g}{\partial \overline{z}} = \left( \frac{\partial f}{\partial \overline{z}} \circ g \right) \cdot \frac{\partial g}{\partial z} + \left( \frac{\partial f}{\partial z} \circ g \right) \cdot \frac{\partial \overline{g}}{\partial z}.$$

The above rules extend easily to functions of several variables. Given two functions $f: \mathbb{C}^M \to \mathbb{C}$ and $g: \mathbb{C}^N \to \mathbb{C}^M$ differentiable in the real sense, the chain rules are:

$$\frac{\partial (f \circ g)}{\partial z_k} = \sum_{m=1}^M \left( \frac{\partial f}{\partial z_m} \circ g \right) \frac{\partial g_m}{\partial z_k} + \sum_{m=1}^M \left( \frac{\partial f}{\partial \overline{z}_m} \circ g \right) \frac{\partial \overline{g}_m}{\partial \overline{z}_k},$$

$$\frac{\partial f \circ g}{\partial \overline{z}_k} = \sum_{m=1}^M \left( \frac{\partial f}{\partial \overline{z}_m} \circ g \right) \frac{\partial g_m}{\partial z_k} + \sum_{m=1}^M \left( \frac{\partial f}{\partial z_m} \circ g \right) \frac{\partial \overline{g}_m}{\partial \overline{z}_k},$$

where $k \in [1, N]$.

3 Convolutional Networks as Hadamard Networks in the Frequency Domain

For reasons we will provide in the next subsection and also for the sake of simplicity, we will consider only complex valued networks operating on 2D tensors. For example, a layer $F$ with a 3-channels tensor input and a 2-channels tensor output of channel size 256 is a complex valued function:

$$F: \mathbb{C}^{256 \times 2} \to \mathbb{C}^{256 \times 2}. $$

In general we will define layers $G_N$ with $p$-channel tensor inputs and $q$-channel tensor output of channel size $N$ as complex valued functions mapping 2D tensors, for example,

$$G_N: \mathbb{C}^{N \times p} \to \mathbb{C}^{N \times q}.$$

Notation-wise, we usually use upper indices for channels and lower indices for vector/tensor components. We write a 3-channel tensor $x \in \mathbb{C}^{N \times 3}$ as $x = (x^1, x^2, x^3)$ and by $x^j$ we mean the $j$ scalar component of vector $x^j$, etc. We also use the notation $G_N$ for layers $G$ used in the spatial domain acting on tensors of channel size $N$ and $F_{(N)}$ for layers used in the frequency domain, etc.

Note also that Fourier transforms are always performed on the 1D components of the 2D tensors: for $x \in \mathbb{C}^{N \times m}$ and $x = (x^1, \ldots, x^m)$ then:

$$F_N(x) = (F_N(x)^1, \ldots, F_N(x)^m)$$

where, for each $1 \leq j \leq m$ we compute $F_N(x^j)$ as per definition [1], etc.

3.1 The Circular vs the Linear Convolution

Currently all convolutional layers used in the spatial domain use linear convolutions rather than circular ones. However, any linear convolution can be mapped to and computed from an 1D circular one. For example, given the following 2D tensor $X$ and the $3 \times 3$ kernel matrix $W$:

$$X = \begin{bmatrix}
    x_0 & x_1 & x_2 \\
    x_3 & x_4 & x_5 \\
    x_6 & x_7 & x_8
\end{bmatrix}, \quad W = \begin{bmatrix}
    w_0 & w_1 & w_2 \\
    w_3 & w_4 & w_5 \\
    w_6 & w_7 & w_8
\end{bmatrix}$$

their 2D linear convolution of stride 1 and same padding is the $3 \times 3$ matrix $X \ast W$ computed as $(X \ast W)_{11} = x_0 w_0 + x_1 w_5 + x_3 w_7 + x_4 w_8, \ldots$ and $(X \ast W)_{11} = \sum_i x_i w_i, \ldots$. One can get the same result by padding these matrices with zeros as shown here:

$$X = \begin{bmatrix}
    0 & 0 & 0 & 0 & 0 \\
    0 & x_0 & x_1 & x_2 & 0 \\
    0 & x_3 & x_4 & x_5 & 0 \\
    0 & x_6 & x_7 & x_8 & 0 \\
    0 & 0 & 0 & 0 & 0
\end{bmatrix}, \quad W = \begin{bmatrix}
    w_0 & w_1 & w_2 & 0 & 0 \\
    w_3 & w_4 & w_5 & 0 & 0 \\
    w_6 & w_7 & w_8 & 0 & 0 \\
    0 & 0 & 0 & 0 & 0 \\
    0 & 0 & 0 & 0 & 0
\end{bmatrix}$$

and then flattening them in row major order to 1D tensors $x, w$:

$$0, 0, 0, 0, 0, x_0, x_1, x_2, 0, 0, x_3, x_4, \ldots$$

and

$$(w_0, w_1, w_2, 0, 0, w_3, w_4, w_5, 0, 0, w_6, w_7, w_8, \ldots)$$

One can observe that the dot product of the above tensors is precisely $(X \ast W)_{00}$. Moreover, one can see from [6], that this value can be found by computing the circular convolution $x \ast w^*$, where $w^*$ is obtained from the tensor $w$ by performing 7 left rotations and then flipping the indices greater than 1 left to right.

Note also that if $x$ and $y$ are two 1D tensors, if $x$ has length $|x|$, and $y$ has length $|y|$, and both $x$ and $y$ are zero padded to length $|x| + |y| - 1$, then their circular convolution matches their linear convolution.

For these reasons henceforth we will only use 2D tensors and circular convolutions, that is, the $x \ast y$ notation will always mean the circular convolution.
3.2 The Convolution Layer

Given two \( m \)-channel tensors, that is, \( x, w \in \mathbb{C}^{N \times m} \), we define their convolution as the sum of convolutions of their 1D components:

\[
x \ast w = (x^1, \ldots, x^m) \ast (w^1, \ldots, w^m) = \sum_{j=1}^{m} x^j \ast w^j.
\]

By the linearity of DFT and the convolution property (7), we know \( \mathcal{F}_N(x \ast w) = \sum_{j=1}^{m} \mathcal{F}_N(x^j) \cdot \mathcal{F}_N(w^j) \). Using this observation it follows that the convolutional layer in the spatial domain:

\[
\text{Conv}_N(x, w_1, \ldots, w_n, b) = (x \ast w_1 + b^1, \ldots, x \ast w_n + b^n)
\]
commutes with the DFT:

\[
\mathcal{F}_N(\text{Conv}_N(x, w_1, \ldots, w_n, b)) = (\mathcal{F}_N(x \ast w_1) + \mathcal{F}_N(b^1), \ldots, \mathcal{F}_N(x \ast w_n) + \mathcal{F}_N(b^n))
\]

\[
= \left( \sum_{j=1}^{m} \mathcal{F}_N(x^j) \cdot \mathcal{F}_N(w_1^j) + \mathcal{F}_N(b^1), \ldots, \sum_{j=1}^{m} \mathcal{F}_N(x^j) \cdot \mathcal{F}_N(w_n^j) + \mathcal{F}_N(b^n) \right),
\]

where the input and the weights are \( m \)-channel tensors, that is, \( x, w_1, \ldots, w_n \in \mathbb{C}^{N \times m} \) and the bias \( b \) verifies that \( b \in \mathbb{C}^{N \times n} \).

Note that usually the tensors used as weights in the convolutional layer have low dimensionality and they are called “kernels”, for example \( 3 \times 3 \) matrices are widely used etc. As we have observed in the previous subsection, computing linear convolutions with these kernels is equivalent to performing 1D circular convolutions, after some suitable zero padding. Based on this observation we define a Pad function that takes a linear pattern, that is, an ordered set of indices \( K \subseteq \{0, \ldots, N-1\} \), and \( k = |K| \) tensor weights to create (via some zero padding) a length \( N \times m \) tensor of weights.

More precisely we define \( \text{Pad}_N : \mathbb{C}^k \times \mathcal{P}(\{0, N - 1\}) \rightarrow \mathbb{C}^{N \times m} \) by

\[
\text{Pad}_N(w, K)_j = \begin{cases} 
  w^j_i & \text{if } K[i] = j \\
  0 & \text{otherwise}
\end{cases}
\]

for \( 0 \leq j < N \) and \( 1 \leq l \leq m \).

For example, given a \( 3 \times 3 \) kernel of weights \( w \in \mathbb{C}^9 \times 1 \) and \( K = \{0, 1, 2, 5, 6, 7, 10, 11, 12\} \) then \( \text{Pad}_{25}(w, K) \) is equal to

\[
(w_0, w_1, w_2, 0, 0, w_3, w_4, 0, w_5, 0, 0, w_6, w_7, w_8, 0, \ldots).
\]

We can then extend convolutional layers in the spatial domain having a kernel \( K \subseteq \{0, \ldots, N-1\} \) as follows.

\[
\text{Conv}_N(x, K, w_1, \ldots, w_n, b) = \\
\text{Conv}_N(x, \text{Pad}_N(w_1, K), \ldots, \text{Pad}_N(w_n, K), b).
\]

3.3 The Hadamard Layer

The Hadamard layer \( H_{(N)} \) corresponding to the convolutional layer \( \text{Conv}_N \) is simply defined so that the DFT carries over from the spatial domain to the frequency domain:

\[
\mathcal{F}_N(\text{Conv}_N(z, K, w_1, \ldots, w_q, b)) = H_{(N)}(\mathcal{F}_N(z), K, w_1, \ldots, w_q, \mathcal{F}_N(b)).
\]

Formally, a Hadamard layer with a \( p \)-channels input tensor \( z = (z^1, \ldots, z^p) \in \mathbb{C}^{N \times q} \) and outputting a \( q \)-channels tensor \( y = (y^1, \ldots, y^q) \in \mathbb{C}^{N \times q} \) by using the kernel \( K \subseteq \{0, N - 1\} \), the weights \( w_1, \ldots, w_q \in \mathbb{C}^{K \times p} \) and the biases \( b \in \mathbb{C}^{N \times q} \) is defined as the complex multivariate function \( H_{(N)} : \mathbb{C}^{N \times p} \times \mathcal{P}(\{0, N - 1\}) \times \mathbb{C}^{K \times p} \times \mathbb{C}^{N \times q} \rightarrow \mathbb{C}^{N \times q} \) given by

\[
y^j = \sum_{i=1}^{p} z^i \cdot \mathcal{F}_N(\text{Pad}(w^j_i, K)) + b^j
\]

where \( 1 \leq j \leq q \), “\( \cdot \)” is the Hadamard product of two vectors, \( \mathcal{F}_N \) is the discrete Fourier transform and \( \text{Pad} \) is the function defined previously.

The Wirtinger derivatives of this layer with respect to input and weights can be computed by applying the chain rule (8), and the definition of the discrete Fourier transform (1). More precisely, assume we have back-propagated the Wirtinger gradients for the layer \( G \) where \( L : \mathbb{C}^{N \times q} \rightarrow \mathbb{R} \) and \( L = G \circ H_{(N)} \) is the loss function. The back-propagation rules for the layer \( H_{(N)} \) and the input variable \( z^i_k \) are:

\[
\frac{\partial G \circ H_{(N)}}{\partial z^i_k} = \sum_{j=1}^{q} \mathcal{F}_N(\text{Proj}(w^j_i, K)) \cdot \frac{\partial G}{\partial z^j_k} \circ H_{(N)}
\]

and

\[
\frac{\partial G \circ H_{(N)}}{\partial z^i_k} = \sum_{j=1}^{q} \mathcal{F}_N(\text{Proj}(w^j_i, K)) \cdot \frac{\partial G}{\partial z^j_k} \circ H_{(N)}
\]

where \( 1 \leq i \leq p \) and \( 0 \leq k < N \).

3.4 The Input Layer

In order to move the entire computation in the frequency domain we will simply apply the Fourier transform to the input tensors. That is, the input tensor \( z = (z^1, \ldots, z^q) \in \mathbb{C}^{N \times q} \) is mapped to \( Z = \mathcal{F}_N(z) = (\mathcal{F}_N(z^1), \ldots, \mathcal{F}_N(z^q)) \). For example, a CIFAR-10 input tensor \( x = (x^1, x^2, x^3) \in \mathbb{R}^{1024 \times 3} \) is
3.5 The Activation Layer

Let \( A_N \) be the function \( A_N: \mathbb{C}^N \rightarrow \mathbb{C}^N \) given by

\[
A_N(x) = \begin{cases} 
\frac{x}{\sqrt{N} \|x\|_2} & x \neq 0 \\
0 & x = 0.
\end{cases}
\] (11)

One can easily observe that \( A_N \) is bounded for all \( x \in \mathbb{C}^N \):

\[
\|A_N(x)\|_2 \leq \frac{1}{\sqrt{N}}.
\]

Moreover, for a given function \( g: \mathbb{C}^N \rightarrow \mathbb{C}^N \) and from equation (11), we know:

\[
\mathcal{F}_N(A_N(g(x)))_k = \sum_{j=0}^{N-1} A_N(g(x))_j \omega^{jk} = \frac{1}{\sqrt{N}} \frac{g(x)}{\|g(x)\|_2} \|g(x)\|_2 \omega^{jk} = \frac{\mathcal{F}_N(g(x))_k}{\|\mathcal{F}_N(g(x))\|_2},
\]

This means that for any function \( g: \mathbb{C}^N \rightarrow \mathbb{C}^N \) the following equation holds:

\[
\mathcal{F}_N(A_N(g(x))) = \begin{cases} 
\frac{\mathcal{F}_N(g(x))}{\|\mathcal{F}_N(g(x))\|_2} & g(x) \neq 0 \\
0 & g(x) = 0.
\end{cases}
\]

Based on the above observation we define the activation layer in the frequency domain \( A_{N}(z): \mathbb{C}^N \rightarrow \mathbb{C}^N \) to be

\[
A_{N}(z) = \begin{cases} 
\frac{z}{\|z\|_2} & z \neq 0 \\
0 & z = 0.
\end{cases}
\] (12)

and we know \( A_N \) and \( A_N \) commute with the DFT:

\[
\mathcal{F}_N(A_N(z)) = A_N(\mathcal{F}_N(z)).
\]

Let \( A_{N}(z) = (f_0(z), \ldots, f_{N-1}(z)) \) and \( z = (z_0, \ldots, z_{N-1}) \). For each \( k \in [0, N-1] \) we know \( f_k: \mathbb{C}^N \rightarrow \mathbb{C} \) and \( z_k = x_k + iy_k \). The partial derivatives of \( f_k(z) \) with respect to the variables \( x_k \) and \( y_k \) are:

\[
\frac{\partial f_k}{\partial x_k}(z) = \frac{\partial}{\partial x_k} x_k + iy_k = \frac{\|z\|_2^2 - x_k^2}{\|z\|_2^2} - \frac{x_k y_k}{\|z\|_2^2},
\]

\[
\frac{\partial f_k}{\partial y_k}(z) = \frac{\partial}{\partial y_k} x_k + iy_k = -\frac{x_k y_k}{\|z\|_2^2} + \frac{\|z\|_2^2 - y_k^2}{\|z\|_2^2}.
\]

The Wirtinger derivatives for \( f_k(z) \) follow from the above equations and their definitions:

\[
\frac{\partial f_k}{\partial \overline{z}_k}(z) = \frac{1}{2} \frac{\|z\|_2^2 - x_k^2}{\|z\|_2^2},
\]

\[
\frac{\partial f_k}{\partial z_k}(z) = -\frac{1}{2} \frac{\sqrt{N}}{\|z\|_2}, \text{when } j \neq k.
\]

Note that the activation function is bounded and differentiable in the real sense but it is not differentiable in the complex sense as its \( \frac{\partial f_j}{\partial \overline{z}_k} \) derivatives do not vanish, hence \( A_{N}(z) \) is not holomorphic.

Note also that Georgiou et al introduced the following activation function in (6):

\[
f(z) = \frac{z}{c + \sqrt{\frac{1}{r} \|z\|^2}},
\]

where \( c \) and \( r \) are real positive constants. This function maps a point \( z \) on the complex plane to a unique point \( f(z) \) on the open disc \( \{z: |z| < r\} \). One way to extend this activation function is to \( \mathbb{C}^N \), while making it commute with the DFT, is the following:

\[
A_{N,c,r}(z) = \frac{z}{c + \sqrt{\frac{N}{r} \|z\|^2}},
\]

and we can similarly show that \( A_{N,c,r} \) has the property that

\[
\mathcal{F}_N(A_{N,c,r}(z)) = \frac{\mathcal{F}_N(z)}{c + \sqrt{\frac{N}{r} \|\mathcal{F}_N(z)\|^2}}.
\]

Henceforth we will extend these activation functions to 2D tensors in the usual way, that is, if \( z = (z_1, \ldots, z_m) \in \mathbb{C}^{N \times m} \) then

\[
A_{N}(z) = (A_{N}(z_1), \ldots, A_{N}(z_m)).
\] (13)
3.6 The Fully Connected Layer

Parseval Theorem states that if \( z, w \in \mathbb{C}^N \) then the following equality holds:
\[
\frac{1}{N} \sum_{j=0}^{N-1} z_j \overline{w}_j = \frac{1}{N} \sum_{j=0}^{N-1} \mathcal{F}_N(z)_j \overline{\mathcal{F}_N(w)}_j
\]
that is, the complex dot product \( z \cdot w \) equals \( \frac{1}{N} \mathcal{F}_N(z) \cdot \mathcal{F}_N(w) \). Based on this equality it makes sense to define Fully Connected layers \( \text{FC}(N) \) in the frequency domain as follows:
\[
\text{FC}(N)(z, w_1, \ldots, w_k) = \left( \frac{1}{N} \sum_{j=0}^{mN-1} z_j \overline{w}_{1j}, \ldots, \frac{1}{N} \sum_{j=0}^{mN-1} z_j \overline{w}_{kj} \right)
\]
where \( \text{FC}(N) : \mathbb{C}^{N \times m} \times \mathbb{C}^{N \times m \times k} \rightarrow \mathbb{C}^k \), \( z \) is an \( m \)-channels input and \( w_1, \ldots, w_k \in \mathbb{C}^{N \times m} \) are the fully connected layer’s filters/weights.

This definition has the property that the well-known fully connected layers operating in the spatial domain are mapped naturally to the frequency domain. That is, if \( \mathbf{x} = (x^1, \ldots, x^m) \in \mathbb{R}^{N \times m} \) is an \( m \)-channels input tensor and \( w_1, \ldots, w_k \in \mathbb{R}^{N \times m} \) are the weights of a standard fully connected layer \( \text{FC}_N \), then the layer’s output verifies the following relation:
\[
\text{FC}_N(\mathbf{x}, w_1, \ldots, w_k) = \left( \sum_{j=0}^{mN-1} x_j w_{1j}, \ldots, \sum_{j=0}^{mN-1} x_j w_{kj} \right)
\]
\[
= \left( \sum_{i=1}^{m} x^i \cdot w^i_1, \ldots, \sum_{i=1}^{m} x^i \cdot w^i_k \right)
\]
\[
= \frac{1}{N} \left( \sum_{i=1}^{m} \mathcal{F}_N(x^i) \cdot \mathcal{F}_N(w^i_1), \ldots, \sum_{i=1}^{m} \mathcal{F}_N(x^i) \cdot \mathcal{F}_N(w^i_k) \right)
\]
\[
= \text{FC}(N)(\mathcal{F}_N(\mathbf{x}), \mathcal{F}_N(w_1), \ldots, \mathcal{F}_N(w_k)) = \text{FC}(N)(\mathbf{F}_N(\mathbf{x}, w_1, \ldots, w_k)).
\]

Note that formally we define \( \text{FC}_N \) as a complex valued function \( \text{FC}_N : \mathbb{C}^{N \times m} \times \mathbb{C}^{N \times m \times k} \rightarrow \mathbb{C}^k \) and
\[
\text{FC}_N(\mathbf{x}, w_1, \ldots, w_k) = \left( \sum_{i=1}^{m} x^i \cdot w^i_1, \ldots, \sum_{i=1}^{m} x^i \cdot w^i_k \right)
\]
and we have that
\[
\text{FC}_N(\mathbf{x}, w_1, \ldots, w_k) = \text{FC}(N)(\mathbf{F}_N(\mathbf{x}, w_1, \ldots, w_k)).
\]

The Wirtinger gradients of the \( \text{FC}(N) \) layer can be computed easily at back-propagation time by applying the chain rule \( [8] \) and the trivial equalities \( \partial(z\overline{w})/\partial w = z \) and \( \partial(z\overline{w})/\partial z = \overline{w} \).

3.7 The Output Layer and the Cross Entropy Loss

Let \( \text{Out}_N \) be the function \( \text{Out}_N : \mathbb{C}^N \rightarrow \mathbb{C}^N \) given by
\[
\text{Out}_N(\mathbf{z}) = \begin{cases} \frac{\mathbf{z}}{\| \mathbf{z} \|_2} & \mathbf{z} \neq \mathbf{0} \\ (\sqrt{1/N}, \ldots, \sqrt{1/N}) & \mathbf{z} = \mathbf{0}. \end{cases}
\]

For all \( \mathbf{z} = (z_0, \ldots, z_{N-1}) \in \mathbb{C}^N \) we have that \( 1 = \sum_{i=0}^{N-1} |\text{Out}_N(\mathbf{z})_i|^2 \). Thus the probability of a particular outcome \( j \in 0, \ldots, N - 1 \) is
\[
P(y = j \mid \mathbf{z}) = |\text{Out}_N(\mathbf{z})_j|^2 = z_j \cdot \bar{z}_j / \| \mathbf{z} \|_2^2.
\]

That is, the squared absolute value of the \( \text{Out}_N(\mathbf{z})_j \) amplitude. Incidentally, this way of defining probabilities is similar to the Born Rule in Quantum Mechanics: the probability of a particular outcome is the squared absolute value of a certain amplitude. Given a pair \( (\mathbf{z}^0, \mathbf{y}^0) \in \mathbb{C}^N \times \mathbb{R}^N \) with \( \sum_{j=0}^{N-1} y^0_j = 1 \), the cross entropy loss \( L : \mathbb{C}^N \times \mathbb{R}^N \rightarrow \mathbb{C}^N \) is
\[
L(\mathbf{z}^0, \mathbf{y}^0) = \sum_{k=0}^{N-1} -y^0_k \log(z^0_k \sqrt{\mathbf{z}} / \| \mathbf{z}^0 \|_2^2).
\]

We can show that the cost functions \( C_k : \mathbb{C}^N \rightarrow \mathbb{R} \), where \( 0 \leq k < N \), given by \( C_k(\mathbf{z}) = \log(z_k \sqrt{\mathbf{z}} / \| \mathbf{z} \|_2^2) \) have the following Wirtinger derivatives
\[
\frac{\partial C_k}{\partial z_k}(\mathbf{z}) = \frac{1}{z_k^2 - 1 / \| \mathbf{z} \|_2^2}
\]
\[
\frac{\partial C_k}{\partial \bar{z}_k}(\mathbf{z}) = \frac{1}{z_k^2 - 1 / \| \mathbf{z} \|_2^2}
\]
\[
\frac{\partial C_k}{\partial z_j}(\mathbf{z}) = -z_j / \| \mathbf{z} \|_2^2, \text{ when } k \neq j
\]
\[
\frac{\partial C_k}{\partial \bar{z}_j}(\mathbf{z}) = -\bar{z}_j / \| \mathbf{z} \|_2^2, \text{ when } k \neq j.
\]

The above equalities let us compute \( \nabla(L) \), the Wirtinger conjugate gradient of the cross entropy loss function, \( L \). As \( L \) is a real-valued function we can then use gradient descent algorithms to minimize it, see \([5]\), for more information.
Note that
\[ \text{Out}_N = \text{Out}_N, \]
that is, we will use the same output layer in both the spatial and
the frequency domains rather than having \( \text{Out}_N \equiv (1/\sqrt{N})F_N(\text{Out}_N) \). The reason for this is that while the
\( \text{Out}_N \) layer commutes with the DFT, the Hirschman enth-
tronic uncertainty principle tells us that we cannot minimize
\( L \), the cross entropy loss, in both domains at the same time.

More precisely, let \( z \in \mathbb{C}^N \) and for \( 0 \leq j < N \) let
\( p_j = |\text{Out}_N(z)_j|^2 \) and \( q_j = 1/\frac{1}{2}\|F_N(\text{Out}_N(z))_j\|^2 \). We
then know \( \sum_{j=0}^{N-1} p_j = 1 \) and by Parseval Theorem we also
know \( \sum_{j=0}^{N-1} q_j = 1 \), however the Hirschman uncertainty
principle tells us that
\[
-\sum_{j=0}^{N-1} \log(p_j) - \sum_{j=0}^{N-1} q_j \log(q_j) \geq \log(N).
\]
Roughly speaking, we cannot minimize \( L \), the cross entropy
loss, in both the space and the frequency domains, at the same
time.

3.8 Putting it All Together

Let’s start with an example by defining a minimal con-
volutional network \( \text{Net} \) in the space domain in order to create a
model for the MNIST dataset. Let this \( \text{Net} \) consist of a convolution layer, say with 64 filters of kernel
\( K = \{0, 1, 2, 28, 29, 30, 56, 57, 58\} \), an activation layer
\( A_{784} \) and a fully connected layer \( FC_{784} \). and biases. Another observation is that the only require-
mant for the final \( \text{Out}_N = \text{Out}_N \) layer is to be a map
from a complex valued vector to a probability vector.

The above result is easily generalized to any convolu-
tional network built with layers that commute with the DFT.
Therefore, we conclude that any convolutional network
built with layers that commute with the Discrete Fourier
Transform is equivalent to a Hadamard Network.

First consequence of this equivalence is that predictive
models using complex numbers for weights should perform
better as the space of solutions is larger.

Second consequence is that performing the computation
in the frequency domain should be faster as convolutions
become Hadamard products and we can precompute the
DFT of the inputs.

Third consequence is that the above convolutional net-
works are equivalent to multivariate rational expressions in
the frequency domain, and learning via gradient descent is
a technique for interpolation.

Last but not the least, the technique we present of ex-
tracting the “recurring” information in the input \( x \) by apply-
ing \( F_N(x) \) then feeding it in a Hadamard network is equivalent
to the known technique of extracting “patterns” from input \( x \) via convolutional networks.

4 Implementation and Experimental Results

In order to properly perform the back-propagation of gra-
dients in the frequency domain, we need support for com-
plex number operations and also a way to keep track of both
the Wirtinger and the Wirtinger conjugate gradients for all
inputs. For weights and biases, just keeping track of the Wirtinger conjugate gradient is enough. The current ma-
chine learning frameworks (Tensorflow, Torch, etc.) have
limited support for both complex differentiation and com-
puting Fourier Transforms on GPUs, therefore the author
has run experiments on custom software. The hardware
used was based on a NVIDIA GTX-1070-Ti GPU.

4.1 MNIST

We have obtained 90% accuracy on MNIST after 70
epochs using a mini-batch Wirtinger gradient descent of
batch size 100 and a shallow network in the frequency do-
main. This network has only one Hadamard layer with
50 out channels and a kernel \( K \) of size \( 7 \times 7 \) with \( K = \{0, 1, 2, 3, 4, 5, 6, 28, 29, 30, 31, 32, 33, 34, \ldots \} \)
\( \{168(=28*6), 169, 170, 171, 172, 173, 174\} \). This Hadamard layer is
directly connected to an activation layer, followed by a 10
out channels fully connected layer, followed by an output
layer:
\[
\text{In}_{(784)} \rightarrow H_{(784,50,7 \times 7)} \rightarrow A_{(784)}
\rightarrow FC_{(784,10)} \rightarrow \text{Out}_{(10)}.
\]
The above net has 394510 complex parameters that are initialized using the circularly symmetric complex Gaussian distribution, see Section 6.1 in the Annex for details.

We slightly improved the accuracy to 91.1% by increasing the number of Hadamard filters to 100 and training the net for 200 epochs.

5 Conclusions and Further Work

In this paper, we first investigated whether we can modify convolutional networks so that the entire computation moves from the space domain into the frequency domain and from convolutions to Hadamard products.

We found out that there are indeed certain activation functions and additional layers that commute with the DFT and make computation in the frequency domain possible.

Next, we did some experiments and found that there are shallow Hadamard networks in the frequency domain that converge fast to a model for the MNIST data.

Finally, we showed that any convolutional network built with layers that commute with the DFT is equivalent to a Hadamard Network. This result suggests that using complex numbers for weights creates better predictive models. Another consequence of this equivalence is that these convolutional networks are multivariate rational expressions in the frequency domain, and learning via gradient descent is a technique for interpolation.

This paper opens some interesting lines of research and here are some questions that might deserve some further theoretical and experimental investigation:

1. Could we build state of the art Hadamard predictive models for larger data sets, say CIFAR10? Can we do it with a shallow Hadamard network? Do we need data augmentation?

2. If \( b = (c_0, c_0, \ldots, c_0) \in \mathbb{C}^N \) then \( F_N(b) = (N \cdot c_0, 0, \ldots, 0) \), and \( F_N(x \ast w + b) = F_N(x) \cdot F_N(w) + (N \cdot c_0, 0, \ldots, 0) \). Does this mean that using a bias in the convolution layer is unnecessary as it only acts as a high pass filter for the stationary frequency? Or should the bias be multivariate?

3. Is the divider layer proposed in the Section 6.1 playing the same role for Hadamard networks as the drop out layer is playing for convolutional networks?

4. Batch normalization and residual layers are both commuting with the DFT, see the Sections 6.2 and 6.3 in the Appendix. Are these layers speeding up the learning in deep Hadamard networks?

5. Let \( c \in \mathbb{R}^+ \) be a positive constant. The cost functions \( C_k \) and their Wirtinger gradients introduced in the Section 3.7 are Lipschitz on the domain \( \{ z : \|z\|_2 > c \} \). Could we use this observation to analyse the gradient descent convergence?

References

[1] Jeff Orchard, Hao Yang and Xiang Ji, “Does the entorhinal cortex use the Fourier transform?”, Front. Comput. Neurosci., https://doi.org/10.3389/fncom.2013.00179, 2013, [1]

[2] Carlos E. Perez, “Should Deep Learning use Complex Numbers?”, medium.com, https://medium.com/intuitionmachine/should-deep-learning-use-complex-numbers-edbd3aac3fb8, 2017, [1]

[3] Cijo Jose, Moustapha Ciss and Franfois Fleuret, “Kronecker Recurrent Units”, https://arxiv.org/pdf/1705.10142v2.pdf, 2017, [1]

[4] Martin Arjovsky, Amar Shah and Yoshua Bengio, “Unitary Evolution Recurrent Neural Networks”, https://arxiv.org/pdf/1511.06445v4.pdf, 2016, [1]

[5] P Bouboulis, “Wirtingers calculus in general Hilbert Spaces”, arXiv preprint arXiv:1005.5170, https://arxiv.org/abs/1005.5170, 2010, [1]

[6] GM Georgiou and C Koutsougeras, “Complex Domain Backpropagation”, IEEE Transactions on Circuits and Systems-11: Analog and Digital Signal Processing, vol. 39, no. 5, 1992, [1]

[7] T Kim and T Adali, “Universal approximation of fully complex feed-forward neural networks”, Acoustics Speech and Signal Processing (ICASSP) 2002 IEEE International Conference on, vol. 1, pp. I-973-I-976, 2002, [2]

[8] Pompeiu, D., “Sur une classe de fonctions d’une variable complexe”, Rendiconti del Circolo Matematico di Palermo (in French), 33 (1): 108113, 1912. [2]

[9] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based Learning Applied to Document Recognition”, In Proceedings of the IEEE , pp. 22782324, 1998, [10]

[10] Chiheb Trabelsi, Olexa Bilaniuk, Ying Zhang, Dmitriy Serdyuk, Sandeep Subramanian, Joo Felipe Santos, Soroush Mehri, Negar Rostamzadeh, Yoshua Bengio and Christopher J Pal, “Deep Complex Networks”, ICLR 2018, https://arxiv.org/pdf/1705.09792v4.pdf, 2018, [9]
The above equalities are equivalent to the following relations:
\[ F_N(x)_k = \mathcal{F}_{N/2}(x_E)_k + \omega^k \mathcal{F}_{N/2}(x_O)_k \]
\[ F_N(x)_{k+N/2} = \mathcal{F}_{N/2}(x_E)_k - \omega^k \mathcal{F}_{N/2}(x_O)_k. \]

The above equalities are equivalent to the following relations:
\[ \mathcal{F}_{N/2}(x_E)_k = \frac{1}{2} (\mathcal{F}_N(x)_k + \mathcal{F}_N(x)_{k+N/2}) \]
\[ \mathcal{F}_{N/2}(x_O)_k = \frac{\omega^{-k}}{2} (\mathcal{F}_N(x)_k - \mathcal{F}_N(x)_{k+N/2}). \]

Then, we can define the Divider layer as the function
\[ \text{Div}_{\text{2N}}(x) = \begin{cases} \frac{1}{2} (x_k + x_{k+N/2}) & k < N \\ \omega^{-k} (x_k - x_{k+N/2}) & N \leq k < 2N \end{cases} \]

having the following property for all \( x \in \mathbb{C}^N \):
\[ \text{Div}_{\text{2N}}(F_{2N}(x)) = (\mathcal{F}_N(x_E), \mathcal{F}_N(x_O)). \] (16)

In words, the Divider layer halves the dimension of the channels by doubling their numbers while preserving the data under the discrete Fourier transform.

6.2 The Batch Normalization Layer Commutes with DFT

Rather than considering batch normalization in the complex domain to be equivalent to whitening 2D vectors, as Trabelsi et al have proposed in [10], we would like to analyse how the batch normalization layer is mapped into the frequency domain under the discrete Fourier transform.

Let \( z_1, \ldots, z_m \) be a batch of \( m \) input tensors with \( p \) channels / feature layers such that \( z_j = (z_{j1}, \ldots, z_{jp}) \in \mathbb{C}^{N \times p} \). For each layer \( 1 \leq k \leq p \) let the mean \( \mu_k \) be
\[ \mu_k = \frac{1}{m} \sum_{j=1}^{m} z_{jk}. \]

Obviously, \( \mu_k \in \mathbb{C}^N \) for all \( 1 \leq k \leq p \). Similarly, let the variance \( \sigma_k^2 \) be given by formula
\[ \sigma_k^2 = \frac{1}{mN} \sum_{j=1}^{m} (z_{jk} - \mu_k) \cdot (z_{jk} - \mu_k) \]

where \( \cdot \) is the inner product of complex vectors etc. Note that the variance \( \sigma_k^2 \) is a real number. By using the DFT linearity (2) and the Parseval Theorem (3) one can show that
\[ F_N(\mu_k) = \frac{1}{m} \sum_{j=1}^{m} F_N(z_{jk}^p), \]
\[ \sigma_k^2 = \frac{1}{mN^2} \sum_{j=1}^{m} (F_N(z_{jk}^p) - F_N(\mu_k))^2 \]
\[ \cdot (F_N(z_{jk}^p) - F_N(\mu_k)). \]

In other words, the mean \( \mu_k \) of \( F_N(z_{jk}^1), \ldots, F_N(z_{jk}^m) \) is equal to \( F_N(\mu_k) \) while their variance \( \sigma_k^2 \) is \( N \) times \( \sigma_k^2 \). The \( 1/N \) scaling factor appears again in the equality (17), due to the way we have chosen the normalization of the discrete Fourier transform. This fact is consistent with observation (20).

The batch normalization defined in [11], works with real numbers. However, we can extend it to complex ones. That is, we may define \( B_N \) be the multivariate function \( B_N: \mathbb{C}^{N \times p} \rightarrow \mathbb{C}^{N \times p} \) such that \( B_N(z) = (B_N^1(z), \ldots, B_N^p(z)) \), \( B_N^k: \mathbb{C}^{N \times p} \rightarrow \mathbb{C}^N \) given by:
\[ B_N^k(z) = \gamma_k \frac{z_k - \mu_k}{\sqrt{\sigma_k^2 + \epsilon}} + \beta_k \] (18)
where \( \gamma_k \in \mathbb{C} \) and \( \beta_k \in \mathbb{C}^N \) are some new weights and \( \epsilon \) is a constant added for numerical stability.

Using again the DFT linearity and the equality (17), we can show that:
\[ F_N(B_N^k(z)) = \frac{1}{N} \frac{\gamma_k}{\sigma_k^2 + \epsilon} \left( F_N(z^k) - F_N(\mu_k) \right) + F_N(\beta_k) \]
\[ = \sqrt{N} \gamma_k \frac{F_N(z^k) - \mu'_k}{\sqrt{\sigma_k^2 + \epsilon}} + F_N(\beta_k). \]

That is, the batch normalization layer is commuting with the DFT.

### 6.3 The Residual Layer Commutes with DFT

One can show that the residual layer \( R_N \) as introduced in \[14\], is commuting with the DFT when modified to use circular convolutions. That is, if
\[ R_N(x, w_1, w_2) = A_N(x * w_1) * w_2 + x \quad (19) \]
then
\[ F_N(R_N) = F_N(A_N(x * w_1) * w_2 + x) \]
\[ = A_N(F_N(x) \cdot F_N(w_1)) \cdot F_N(w_2) \]
\[ + F_N(x). \]

### 6.4 On Complex Weights Initialization

The usual way to initialize the weights in most deep learning frameworks is to sample them from the (multinomial) normal distribution with zero mean and a small standard deviation, see \[9\]. As we would like to initialize the weights of Hadamard layers in an equivalent way, we need to understand how is this distribution changing under the discrete Fourier transforms. More precisely, if \( x = (x_0, x_1, \ldots, x_{N-1}) \in \mathbb{R}^N \) is a white Gaussian noise signal of standard deviation \( \sigma \), that is, \( x_j \sim \mathcal{N}(0, \sigma^2) \) then the question is what is the distribution of \( X_k = F_N(x)_k \). If we note by \( \text{Re}(z) \) and \( \text{Im}(z) \) the real and respective the imaginary part of a complex number \( z \) then one can show that both \( \text{Re}(X_k) \) and \( \text{Im}(X_k) \) are normally distributed with standard deviation \( \sigma \sqrt{N/2} \), that is, \( \text{Re}(X_k) \) and \( \text{Im}(X_k) \sim \mathcal{N}(0, \sigma^2 N/2) \). This results suggests that we should initialize the complex weights \( w_j = u_j + iv_j \) such that
\[ u_j \sim \mathcal{N}(0, 2\sigma^2/N) \quad \text{and} \quad v_j \sim \mathcal{N}(0, 2\sigma^2/N). \quad (20) \]

Note that if the real parts \( u_j \) and the imaginary parts \( v_j \) are independent then \( w_j \) are circularly symmetric complex Gaussians as \( u_j \) and \( v_j \) have the same variance, see \[13\].

Note also that the term \( N/2 \) is due to the way we have chosen the normalization of the discrete Fourier transform. In general, the Fourier transform of the corresponding probability density of \( f(x \mid 0, \sigma^2) = \int_{-\infty}^{\infty} f(x \mid 0, \sigma^2) e^{-i\mu x} dx \)
\[ e^{-\frac{\mu^2}{2\sigma^2}}. \]