# Estimating the Discrete Fourier Transform using Deep Learning

Jonathan Tuck
Department of Electrical Engineering
Stanford University
jtuck1@stanford.edu

March 20, 2018

**Abstract**

Throughout a wide variety of fields, the Discrete Fourier Transform (DFT) is used as a method of analysis of a signal's frequency components. In this paper, we show that a simple three-layer neural network with linear activation functions can estimate the DFT to high accuracy. We test our architecture on a variety of signals, and gauge both its accuracy and its speed against the ground truth and its fast, state-of-the-art implementation. We find that our neural network architecture achieves results very close to that of the ground truth, and that our implementation for our problem instance provides extremely similar accuracy, and is faster than both the naive implementation and the state-of-the-art implementation.

## 1 Introduction

Throughout a wide variety of fields – from signal processing [OSB99], to medical imaging [Ste00], to optics [Goo96] – the Fourier Transform has been used as an important tool for signal analysis, allowing one to decompose a signal in space or time into its frequency components. Typically, when one wants to analyze a signal, it requires the knowledge of the Fourier Transform (sometimes referred to as the *spectrum*) of the signal.

The goal of this project is to estimate the DFT of a signal using a neural network architecture. This neural network should be able to estimate the Discrete Fourier Transform of the signal without having to explicitly compute the Discrete Fourier Transform. Ultimately, the results of this project may lead to a robust deep learning architecture able to compute the DFT of a signal faster than the current state-of-the-art algorithm, the FFT [CT65].

## 2 The Discrete Fourier Transform

The context and background for this project is abundant. Although there exist many formulations of the Fourier transform [Osg17], the scope of this project will only deal with the

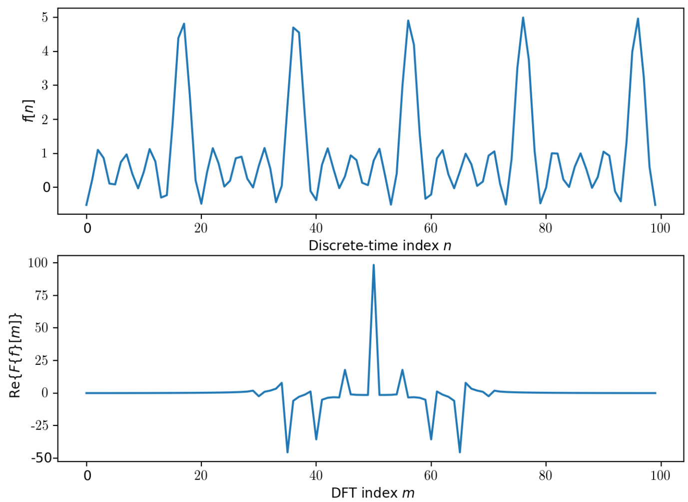


**Figure 1:** Discrete-time signal $f[n] = \sum_{i=1}^{5} \cos(2\pi in + i)$ and the real part of its DFT, $\mathrm{Re}\{F\{f\}[m]\}$.

(real part of the) DFT, the version that maps from discrete-time to discrete-time, as that is what computers mathematically can handle. The $N$*-point DFT* $F : \mathbf{R}^N \to \mathbf{R}^N$ for an $N$-dimensional discrete-time signal [Bra78, Osg17] $f = (f[0], f[1], \ldots, f[N-1])$ is defined as

$$F\{f\}[m] = \sum_{n=0}^{N-1} f[n]e^{2\pi imn/N}, \quad m = 0, \ldots, N-1. \tag{1}$$

Figure 1 illustrates a simple example of a discrete-time signal $f[n]$ and the real part of its DFT, $\mathrm{Re}\{F\{f\}[m]\}$.

The DFT in equation (1) can be written as a dense (complex-valued) matrix multiplication, *i.e.*, $F\{f\} = Df$, where $D \in \mathbf{C}^{N\times N}$ is a complex matrix with values $e^{2\pi imn/N}$ in the $(m,n)$-index. As the DFT can be computed via a dense matrix multiplication, one can compute the DFT naively in $O(N^2)$ time. In addition, there exists a method to compute the DFT in $O(N \log N)$ time, the FFT. The details of the FFT is beyond the scope of this project, and we refer the interested reader to [CT65].

# 3 Approach

**Evaluation metrics.** The evaluation of this project shall be based on how close the neural network estimates for a signal's DFT are to the actual signal's DFT. Specifically, the cost function used is

$$\mathcal{J} = (1/m) \sum_{i=1}^{m} \|F\{f_i\} - \hat{F}\{f_i\}\|_2^2,$$

where $m$ is the number of examples in the set, $F\{f_i\} \in \mathbf{R}^N$ is the vector of actual DFT for the $i$-th example, and $\hat{F}\{f_i\} \in \mathbf{R}^N$ is the DFT estimate for the $i$-th example. In signal processing literature, this particular cost function is referred to as the *mean squared error* (MSE) of the estimates [GD10].

### 3.1 Neural network architecture

We recognize that since the DFT is simply a matrix multiplication, it is trivial to learn the DFT matrix with a one-layer neural network with 100 nodes and a linear activation function. However, we specifically look to learn the DFT matrix with a neural network architecture that emperically shows faster DFT computation times.

Our neural network architecture is three layers of fully connected layers, with 17, nodes per (hidden) layer. Intuitively, as the DFT of a matrix can be described as a matrix multiplication, we pick all of the layers' activation functions to be linear activation functions (and indeed, linear activation functions yielded the greatest performance.) In addition, we used a learning rate of 0.001, a minibatch size of 250, and a drop-out probability of 0.9 (other forms of regularization, such as L2 and L1 regularization, did not increase training accuracy.) We pick these values because our architecture with these hyperparameter values yielded the lowest values of the cost function $\mathcal{J}$ on both the training and test data.

## 4 Data

In order to keep the scope of this project manageable, we shall consider only real, one-dimensional signals (*e.g.*, $f[n] = \cos(n)$), although Fourier transforms and DFTs can be extended to two-dimensional signals (*e.g.*, $g[n,k] = \cos(n)\cos(k)$) and so on [Osg17]. We *a priori* fix the maximum bandwidth to 10 Hz, so that aliasing does not occur for our results [OSB99]. In addition, the time series data is discretized into 100 elements which correspond to evenly spaced indices $t \in [0,1]$. That is, the input signals are vectors in $\mathbf{R}^{100}$. For the remainder of this paper, when we refer to the DFT, we mean the real part of the 100-point DFT.

### 4.1 Training and test data

Discrete-time time series data is readily abundant and training data can be synthetically generated efficiently. Since all signals are simply sums of sinusoids at different frequencies and amplitudes, it is appropriate to synthetically generate data by simply adding randomly generated sinusoids together, some with various types of noise (*e.g.*, white noise, pink noise, *etc.*.) For each example, we shall randomly choose the number of sinusoids to be added, and their frequencies. The test data shall be generated in the same way that the training data is generated. In this way, the training data and the test data shall be drawn from the same distributions, so as to minimize variance.

**Data splits.** In this project, we split our $m = 30000$ examples into training, development, and test sets. We have chosen that 90% of the data is dedicated to the training set, and the remaining 10% is dedicated to the test set. We found that we did not need to allocate any data to include a development set.

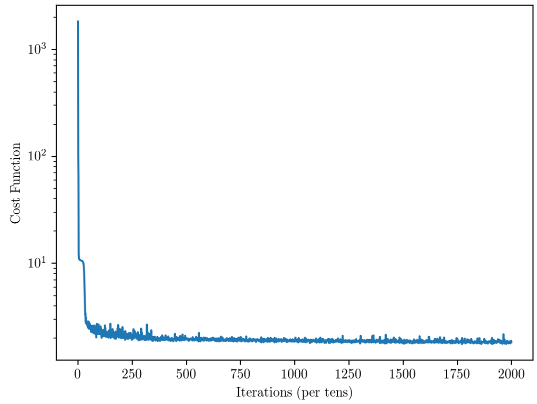


**Figure 2:** Cost versus epoch on the training data.

# 5 Results

We examine the accuracy of our neural network architecture compared to the ground truth, as well as the timing performance of the naive implementation of the DFT, the FFT, and our neural network architecture on the data. We utilize the Python packages, NumPy and SciPy, for the Fourier transform implementations [JOP+ , Oli ]. We implement the neural network architecture with TensorFlow [AAB+15].

## 5.1 Accuracy

Figure 2 is a plot of the cost $\mathcal{J}$ versus epoch on the training data for this particular problem instance and initialization. For the hyperparameters specified in §3, our neural network architecture achieves an MSE of $8.1 \times 10^{-4}$ on the training data and an MSE of $2.1 \times 10^{-2}$ on the test data after 10000 epochs.

Figure 3 is a plot comparing the real part of the DFT of an example in the test set versus its neural network estimate. The two graphs are very similar, but not identical; we find that for the particular example in Figure 3, $\|F\{f\} - \hat{F}\{f\}\|_2 / \|F\{f\}\|_2 = 6.0 \times 10^{-4}$. This result suggests that this particular neural network architecutre for estimating the DFT of a vector is valid and can be used successfully.

## 5.2 Timing

Over the 3000 data examples in the test set, the naive implementation of the DFT averaged a time of 4.1 $\mu$s, the FFT averaged a time of 3.5 $\mu$s, while our neural network implementation maintained a time of 1.9 $\mu$s, a time faster than both the naive implementation and the FFT. These results show that there exists situations where the neural network architecture is faster than the current state-of-the-art methods, at the cost of a negligible loss in accuracy.

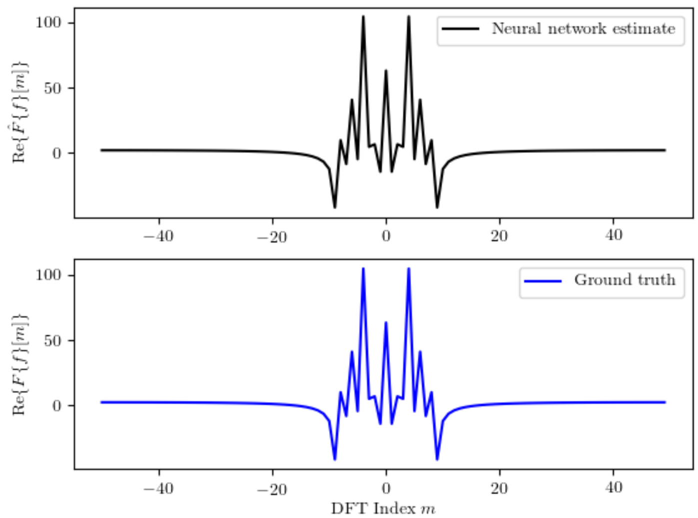


**Figure 3:** The real part of the DFT of an example (bottom) and the estimate from the output of the neural network.

# 6 Future work

**Exploiting structure.** It is well known that many signal processing problems become easier to solve if structure is exploited, such as in compressed sensing, where sparsity is exploited to reduce the minimum sampling rate for perfect signal reconstruction [Don06]. It is possible that these efficients can further be explained or exploited using neural networks with a potentially sparse amount of weights, compared to the neural network of a DFT.

**Other transforms.** Although the DFT is the most widely known basis transformations, there exists a wide variety of basis transforms that could be exploited using deep learning and with wide-ranging applications, such as the Discrete Cosine transform in image compression [ANR74, YL95], the Radon transform in tomography [Dea07], and the Continuous Wavelet transform [Mal08]. The creation of such a neural network would potentially allow for a much speedier implementation of a particular transform, designed for one particular task (*e.g.*, image reconstruction from a fixed image size.)

# 7 Contributions and acknowledgements

This project was designed, written, and programmed by Jonathan Tuck. The code for this project can be viewed at `https://github.com/jonathantuck/CS230-project`.

The author would like to thank Xingyu Liu for his insightful comments on how to improve this project, and for suggesting neural network architectures to use for this particular application.

## References


[AAB+15] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[ANR74] N. Ahmed, T. Natarajan, and K. R. Rao. Discrete cosine transform. *IEEE Transactions on Computers*, C-23(1):90–93, Jan 1974.

[Bra78] R.N. Bracewell. *The Fourier Transform and its Applications*. McGraw-Hill, Tokyo, second edition, 1978.

[CT65] J. Cooley and J. Tukey. An algorithm for the machine calculation of complex fourier series. *Mathematics of Computation*, 19(90):297–301, 1965.

[Dea07] S. R. Deans. *The Radon Transform and Some of Its Applications*. Dover Publications, Mineola, N.Y, 2007.

[Don06] D. L. Donoho. Compressed sensing. *IEEE Transactions on Information Theory*, 52:1289–1306, 2006.

[GD10] R. Gray and L. Davisson. *An Introduction to Statistical Signal Processing*. Cambridge University Press, New York, NY, USA, 1st edition, 2010.

[Goo96] J.W. Goodman. *Introduction to Fourier Optics*. McGraw-Hill Series in Electrical and Computer Engineering: Communications and Signal Processing. McGraw-Hill, 1996.

[JOP+ ] Eric Jones, Travis Oliphant, Pearu Peterson, et al. SciPy: Open source scientific tools for Python, 2001–. [Online; `https://www.scipy.org`].

[Mal08] S. Mallat. *A Wavelet Tour of Signal Processing, Third Edition: The Sparse Way*. Academic Press, 3rd edition, 2008.

[Oli ] T. Oliphant. NumPy: Open source scientific computing for Python, 2006–. [Online. `www.numpy.org`].

[OSB99] A. V. Oppenheim, R. W. Schafer, and J. R. Buck. *Discrete-time Signal Processing*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1999.

[Osg17] B. Osgood. *Lectures on the Fourier Transform and its Applications*. McGraw Hill, first edition, 2017.

[Ste00] S. Stergiopoulos. *Advanced Signal Processing Handbook: Theory and Implementation for Radar, Sonar, and Medical Imaging Real-Time Systems*. CRC Press, Inc., Boca Raton, FL, USA, 1st edition, 2000.

[YL95] B. Yeo and B. Liu. Volume rendering of DCT-based compressed 3d scalar data. *IEEE Transactions on Visualization and Computer Graphics*, 1(1):29–43, Mar 1995.