Solving EMNIST using CNN

Miguel Angel De la Vega Rodriguez[1]

January 05, 2026

Abstract

This article tries to solve EMNIST digit recognition problem using Convolutional Neural Networks which are written in Rust from scratch using State of the Art techniques such as FFT. (Fast Fourier Transform)

1 Introduction

EMNIST [1] is an extension of the classical MNIST dataset for hand written digits (which in turn was an extension for the NIST extension database). This dataset includes more demanding digits which allow for better benchmark for the CNN.

Convolutional Neural Networks [2] which were firstly mentioned when studying the sight of some animals, have proven to be the go-to approach when dealing with image processing.

This article provides a way to understand the whole process involved, from the mathematical foundations to the implementation, showing the intuitive approach and refining solutions to achieve the best possible solution.

2 Convolutions

A convolution is a mathematical operation on two functions that produces a third function as the integral of the product of both after one is shifted and reflected over the y-axis.

3 Cooley-Tukey FFT algorithm

The Cooley-Tukey FFT algorithm is the most common fast Fourier transform algorithm. To understand its importance, we must first look at the computational cost of the operations it optimizes.

The core idea is to perform a convolution over two polynomials, let's call them p_1 and p_2 with degree d. The product of these two polynomials will have at most degree $2d$.

If we represent these polynomials by their coefficients (the standard approach), calculating the product corresponds to the Cauchy product of the coefficients. For two polynomials $A(x) = \sum a_i x^i$ and $B(x) = \sum b_i x^i$, the coefficient c_k of the product $C(x) = A(x)B(x)$ is given by:

$$c_k = \sum_{j=0}^{k} a_j b_{k-j}$$

***Example* 3.1.** Consider the following polynomials:

$$p_1 = 4 + 2x + 6x^2$$
$$p_2 = 2 + 5x + 3x^2$$

You can think of the convolution of these polynomials in different ways, for example, the most straight-forward way would be simply multiplying both polynomials:

$$(4 + 2x + 6x^2) \cdot (2 + 5x + 3x^2) = 4 \cdot (2 + 5x + 3x^2) + 2x \cdot (2 + 5x + 3x^2) + 6x^2 \cdot (2 + 5x + 3x^2)$$

Which results in $p = 8 + 24x + 34x^2 + 36x^3 + 18x^4$

Calculating every coefficient c_k requires iterating through the sums, resulting in a complexity of $O(N^2)$. This is too slow for large inputs, which is where FFT comes to help. Before describing the process lets first introduce some basis.

Theorem 3.2. (Existence and Uniqueness of the Interpolating Polynomial): *For any $n+1$ points $(x_0, y_0)...(x_n, y_n) \in \mathbb{R}^2$, where $x_i \neq x_j \forall i, j \in [0, n]$ there exists a unique polynomial $p(x)$ of degree at most n that interpolates these points.*

Proof. If we write the interpolation polynomial in the form:

$$p(x) = a_n x^n + a_{n-1} x^{n-1} + ... + a_1 x + a_0$$

then if we substitute this into each of the interpolation equations $p(x_i) = y_j$ we get a system of linear equations:

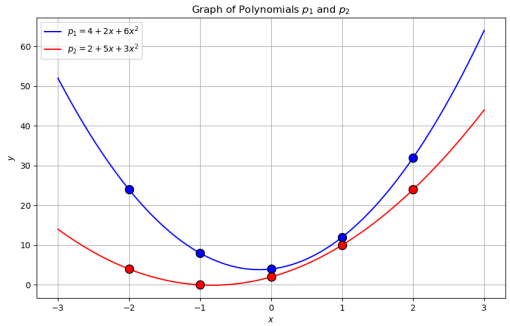
$$\begin{aligned} y_0 = p(x_0) &= a_n x_0^n + a_{n-1} x_0^{n-1} + +... + a_1 x_0 + a_0 \\ y_1 = p(x_1) &= a_n x_1^n + a_{n-1} x_1^{n-1} + +... + a_1 x_1 + a_0 \\ &\vdots \\ y_n = p(x_n) &= a_n x_n^n + a_{n-1} x_n^{n-1} + +... + a_1 x_n + a_0 \end{aligned}$$

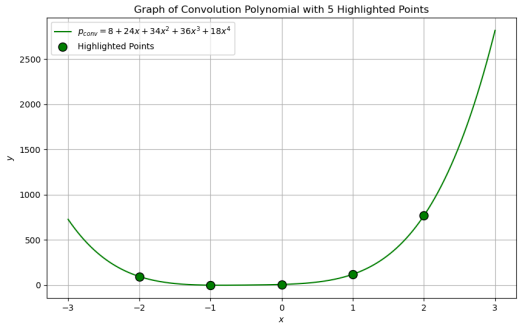
which can be expressed in matrix form:

$$\begin{pmatrix} x_0^n & x_0^{n-1} & ... & x_0 & 1 \\ x_1^n & x_1^{n-1} & ... & x_1 & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ x_n^n & x_n^{n-1} & ... & x_n & 1 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{pmatrix} = \begin{pmatrix} p(x_0) \\ p(x_1) \\ \vdots \\ p(x_n) \end{pmatrix}$$

If we express the above ecuation as $X \cdot A = Y$. Then, we know there is a solution $\Leftrightarrow \det(X) \neq 0$. But X is a Vandermore matrix whose determinant is known to be non-zero as each point is distinct so the system has a unique solution. □

Now, from the previous example we could take $2d$ points from each polynomial which identifies them and multiply point-wise like both of them, getting $2d$ new points which identify the convolution on the right





Now we know that multiplying d points from both polynomials is equivalent to calculating the convolution, note that we have reduced $O(n^2)$ operations to $O(d)$ operations, however there are two things left:

- Calculating $n+1$ points from a polynomial
- Obtaining the expression of the polynomial given the points

The first problem is just an evaluation of the polynomial at some points, however, note that this evaluation translates to the old $O(d^2)$ computational cost, this is where FFT comes to the rescue, the idea is deduced from the odd and even functions, which we know verify $f(-x) = -f(x) \wedge f(-x) = f(x)$ respectively. This property allows us to reduce the amount of points evaluated in a factor of 2, however, we know not all polynomials are odd or even.

For the solution (FFT) we will rewrite the polynomial separating odd terms from even terms, also we will express the polynomial at x^2, so, let $P(x) = a_0 + a_1 x + ... + a_{n-1}x^{n-1} + a_n x^n$. Then we can separate odd and even degree terms, lets call them $P_{odd}(x)$ and $P_{even}(x)$, then we can express:

$$P(x) = P_{even}(x) + P_{odd}(x)$$

If we factor out x from $P_{odd}(x)$ we get even powers for both, lets see an example:

***Example* 3.3.** Let $P(x) = 1 + 4x + 5x^2 + 3x^3 + x^4 + 7x^5$, then we can define:

$$P(x) = \underbrace{(1 + 5x^2 + x^4)}_{P_{\text{even}}} + \underbrace{x(4 + 3x^2 + 7x^4)}_{P_{\text{odd}}}$$

The discrete Fourier transform is defined by the formula:

$$X_k = \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi i}{N} nk}$$

So we can find (at least in $O(N^2)$ operations) a polynomial that interpolates p_1, p_2, if we multiplied these two new polynomials in n points we would get a new Fast Fourier Transform

Bibliography

[1] G. Cohen, S. Afshar, J. Tapson, and A. van Schaik, "EMNIST: An extension of MNIST to handwritten letters," in *2017 International Joint Conference on Neural Networks (IJCNN)*, 2017, pp. 2921–2926. doi: 10.1109/IJCNN.2017.7966217.

[2] K. O'Shea and R. Nash, "An Introduction to Convolutional Neural Networks," *ArXiv*, 2015, [Online]. Available: https://api.semanticscholar.org/CorpusID:9398408