Neural networks can learn any low complexity pattern

Sourav Chatterjee

Based on joint work with Tim Sudijono

Statistical prediction

Suppose we have data $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)$, where $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^p$ and $y_1, \ldots, y_n \in \mathbb{R}$.

The goal is to predict y given $\mathbf{x}$, as some function $f(\mathbf{x})$ of $\mathbf{x}$.

The statistical approach is to find a function $f : \mathbb{R}^p \to \mathbb{R}$ from some function class $\mathcal{F}$ that minimizes a loss such as

$$\sum_{i=1}^{n} (y_i - f(\mathbf{x}_i))^2,$$

and declare the minimizer $\widehat{f} \in \mathcal{F}$ as the predictor.

For example, if $\mathcal{F}$ is the class of affine maps, then this is linear regression.

Feedforward neural networks

In feedforward neural networks, we consider a larger class of functions.

First, we fix an activation function $\sigma : \mathbb{R} \to \mathbb{R}$. The most commonly used is the ReLU activation function $\sigma(x) = \max\{x, 0\}$.

The activation function acts on $\mathbb{R}^d$ (for any d) coordinatewise. That is, $\sigma(x_1, \ldots, x_d) = (\sigma(x_1), \ldots, \sigma(x_d))$.

Functions considered in a feedforward neural network with activation σ and depth D are of the form

$$g_D \circ \sigma \circ g_{D-1} \circ \sigma \circ \cdots \circ g_2 \circ \sigma \circ g_1$$

where each g_i is an affine map from $\mathbb{R}^{d_i}$ into $\mathbb{R}^{d_{i+1}}$, with $d_1 = p$ and $d_{D+1} = 1$.

Example

Consider the sigmoid activation function

$$\sigma(x) = \frac{e^x}{1 + e^x}.$$

Suppose that the input dimension p is 1.

Then, a depth 2 feedforward neural network with activation σ consists of functions of the form

$$\gamma + \delta \frac{e^{\alpha+\beta x}}{1 + e^{\alpha+\beta x}}.$$

Note that this is an enhancement of ordinary logistic regression with two additional parameters γ and δ. Here $g_1(x) = \alpha + \beta x$ and $g_2(x) = \gamma + \delta x$.

Mysteries

Framed in the way we did in the preceding slides, it seems like feedforward neural networks are just a class of parametric models similar to the ones that we consider in statistics.

So why are they so much more effective in many situations?

What kinds of patterns can they learn?

Classical theories from statistics and machine learning do not seem to be adequate for studying neural networks.

There has been considerable theoretical progress in recent years, but we are still far from understanding why they work as well as they do.

Our setting

We consider the following simple class of problems.

Suppose $\mathbf{x}_1, \ldots, \mathbf{x}_n$ are i.i.d. from some (unknown) probability distribution on $\{1, \ldots, N\}^d$, where N and d are (known) positive integers.

$y_i = P(\mathbf{x}_i)$ for some function P that has low complexity (to be defined later).

We estimate P by finding the feedforward neural network f with ReLU activation that has minimum description length (to be defined later), among all f that satisfy $y_i = f(\mathbf{x}_i)$, $i = 1, \ldots, n$.

Low complexity functions

A function P has low complexity if can be expressed as the output of a short program in a programming language that we define.

Our programming language has a very small number of commands, consisting of:

- Value assignment to variables, which must be nonnegative integers.
- For loops.
- If statements.
- Return statement.
- Addition and multiplication.
- Testing equality, greater than, less than.

The complexity of the program is just the number of lines.

Example: Primality testing

```
input n
int i = 2
int j = 2
int res = 0
int prod = 0
int t = 0
bool output = 0
bool prod_equals = 0
for i = 2,...,n:
    for j = 2,...,n:
        prod = i*j
        prod_equals = (prod == n)
        res = res + prod_equals
output = (res > 0)
return output
```

Description length of a neural network

Recall that a feedforward neural network with activation σ and depth D has the form

$$g_D \circ \sigma \circ g_{D-1} \circ \sigma \circ \cdots \circ g_2 \circ \sigma \circ g_1$$

where each g_i is an affine map from $\mathbb{R}^{d_i}$ into $\mathbb{R}^{d_{i+1}}$, with $d_1 = p$ and $d_{D+1} = 1$.

Then $g_i(\mathbf{x}) = W_i\mathbf{x} + b_i$ for some $d_{i+1} \times d_i$ matrix W_i and some vector $b_i \in \mathbb{R}^{d_{i+1}}$.

We will only work with networks where the entries of the W_i's and b_i's can be expressed by finitely many binary digits.

We define description length to be the total number of binary digits required to express entries of the W_i's, b_i's, d_i's and D, subject to further compressibility (next slide).

Further reduction

Sometimes, the sequence $W_1, b_1, \ldots, W_D, b_D$ can be described more succinctly, e.g., if some W_i, b_i pair is repeated several times.

For example, we can write a sequence like $W_1, b_1, W_1, b_1, W_2, b_2$ as $(W_1, b_1)^{*2}, W_2, b_2$.

Going further, the sequence

$$W_1, b_1, W_1, b_1, W_2, b_2, W_1, b_1, W_1, b_1, W_2, b_2, W_3, b_3$$

can be compressed as

$$((W_1, b_1)^{*2}, W_2, b_2)^{*2}, W_3, b_3.$$

We define description length as the number of binary digits required to write down the sequence in the most economical way. (This includes the binary digits for the exponents as well.)

Main result

Theorem (C. & Sudijono, 2025)
There is an absolute constant C such that the following is true. Let P be a program of length L which outputs a result $P(x) \in \{0,1\}$ for each input $x \in \{1,\dots,N\}^d$. Let B be the maximum possible value attained by a variable within the program. Let V be the number of variables in P. Our data is $(x_i, y_i), i = 1,\dots,n$ where x_i is drawn i.i.d. from a distribution μ on $\{1,\dots,N\}^d$ and $y_i = P(x_i)$. Let $\widehat{f}$ be a minimum description length ReLU network that exactly fits the data. Then for a new sample $x \sim \mu$,

$$\mathbb{P}(\widehat{f}(x) \neq P(x)) \leq \frac{CL^3V^2 \ln B}{n}.$$

Example: Primality testing

Sample x uniformly from $1, \ldots, N$. Let $P(x) = 1$ if x is prime and 0 if not.

The program satisfies $L = 11, V = 9, B = N^2$.

Suppose we train $\widehat{f}$ on a sample of size n.

Then for the new sample x, the theorem shows that

$$\mathbb{P}(\widehat{f}(x) \neq P(x)) \leq \frac{C \ln N}{n}.$$

Recall that the density of the primes among the first N natural numbers is $(\ln N)^{-1}$ by the prime number theorem. Therefore $\widehat{f}$ would classify both primes and non-primes correctly with high accuracy if $n \gg (\ln N)^2$.

Example: Sum of two squares

Sample x uniformly from $1, \ldots, N$. Let $P(x) = 1$ if x is a sum of two squares and 0 if not.

The program satisfies $L = 13, V = 11, B = 2N^2$.

Suppose we train $\widehat{f}$ on a sample of size n.

Then for the new sample x, the theorem shows that

$$\mathbb{P}(\widehat{f}(x) \neq P(x)) \leq \frac{C \ln N}{n}.$$

A classical result says that among the first N natural numbers, the proportion of numbers that can be expressed as a sum of two squares is $\sim K(\ln N)^{-1/2}$ for an explicit constant K. Therefore $\widehat{f}$ would classify correctly with high accuracy if $n \gg (\ln N)^{3/2}$.

Example: Sides of triangles

Sample $x = (u, v, w)$ uniformly from $\{1, \ldots, N\}^3$. Let $P(x) = 1$ if u, v, w are the side-lengths of a triangle and 0 if not.

The program satisfies $L = 7, V = 11, B = 2N$.

Suppose we train $\widehat{f}$ on a sample of size n.

Then for the new sample x, the theorem shows that

$$\mathbb{P}(\widehat{f}(x) \neq P(x)) \leq \frac{C \ln N}{n}.$$

As $N \to \infty$, approximately $1/2$ of triples satisfy the condition. Therefore $\widehat{f}$ would classify correctly with high accuracy if $n \gg \ln N$.

Remarks

It is a standard idea in statistics that if we find an estimator $\widehat{f}$ by minimizing

$$\sum_{i=1}^{n}(y_i - f(\mathbf{x}_i))^2$$

over f in a 'small' class of functions $\mathcal{F}$, and the 'true' function $f_0(\mathbf{x}) = \mathbb{E}(y|\mathbf{x})$ is also in this class, then $\widehat{f}$ is close to f_0 with high probability.

But usually $\mathcal{F}$ is a relatively simple class, such as a subclass of linear functions, or monotone functions, or convex functions, etc.

But complicated patterns, which may nevertheless be of low complexity in the sense of Kolmogorov complexity (i.e., expressible as short program), are not captured by standard approaches in statistics. Neural networks can do the job.

A further remark

Our theorem says that a neural network that has minimum description length among all networks that exactly fit the data is likely to be a good predictor. Actually, 'minimum' is not important; 'small' is enough.

But how do we find such a network?

Current methods for fitting neural networks to data typically use gradient descent, which often converges to an exact fit.

A possible idea is that if gradient descent is started from an initial network that has small description length, and it converges fast to an exact fit, the fitted network also has small description length.

A rigorous proof of this claim will show why neural networks work so well in detecting patterns.

About the literature

Several previous works show that certain neural network architectures, particularly transformers, can model basic programs.

Furthermore, folklore says that feedforward neural networks are equivalent to boolean circuits.

The main novelty of our theorem is that it provides an interpretable conversion between a simple programming language and deep feedforward neural networks.

For the details of the conversion scheme and a survey of the literature, we refer to our preprint, which is available on arXiv.

Towards the proof

The following theorem is the first step towards the proof of the main theorem. It shows that if a function P can be encoded by a short program, then it can also be encode by a ReLU network with a small description length.

Theorem (C. & Sudijono, 2025)
There is an absolute constant C such that the following is true. Let P be a program of length L which outputs a result $P(x) \in \{0, 1\}$ for each input $x \in \{1, \dots, N\}^d$. Let B be the maximum possible value attained by a variable within the program. Let V be the number of variables in P. Then P can be expressed as a ReLU network with description length at most $CL^3V^2 \ln B$.

Remarks

The proof of this result is by an explicit construction.

Given a program P, each line of the code is transformed into a series of compositions of affine functions and ReLU activations.

The input vector contains the values of all input variables.

Each application of g_i followed by σ updates the values of the variables, or adds or deletes one or more variables.

For loops are encoded by a single neural network representing the body of the loop that is repeated the maximum possible number of times, together with an expansion that encodes the number of iterations and exits the loop when the required condition is met.

An example

Consider the following program, which adds two numbers

```
input m
input n
int s = 0
s = m + n
return s
```

It is represented by a network of depth $D = 2$, dimensions $d_1 = 3$, $d_2 = 3$,

$$W_1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 1 & 0 \end{bmatrix}, \; W_2 = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix},$$

and $b_1, b_2 =$ zero vectors of dimensions 3 and 1.

Example, contd.

Note that

$$W_1 \begin{bmatrix} m \\ n \\ 0 \end{bmatrix} + b_1 = \begin{bmatrix} m \\ n \\ m+n \end{bmatrix}.$$

Thus,

$$\textsc{ReLU}\left(W_1 \begin{bmatrix} m \\ n \\ 0 \end{bmatrix} + b_1 \right) = \begin{bmatrix} m \\ n \\ m+n \end{bmatrix}.$$

Finally,

$$W_2\, \textsc{ReLU}\left(W_1 \begin{bmatrix} m \\ n \\ 0 \end{bmatrix} + b_1 \right) + b_2 = m + n,$$

which is the required output.

Program Conversion: Example 2

```
input c
input x
input y
int z = 0
if c:
    z = x
else:
    z = y
return z
```

(Here c is boolean, i.e., 1 if true and 0 if false.)

Program Conversion: Example 2

The idea is to notice that when $x, y \leq B$ (for some given upper bound B), the required output can be written as

$$\textsc{ReLU}(2Bc + x - B) + \textsc{ReLU}(-2Bc + y + B) - B.$$

To see this, note that if $c = 1$, then

- $2Bc + x - B = B + x \geq 0$, and so, $\textsc{ReLU}(2Bc + x - B) = B + x$.
- $-2Bc + y + B = y - B \leq 0$, and so, $\textsc{ReLU}(-2Bc + y + B) = 0$.
- Thus, $\textsc{ReLU}(2Bc + x - B) + \textsc{ReLU}(-2Bc + y + B) - B = x$.

A similar calculation shows that the result is y if $c = 0$.

Program Conversion: Example 2

Take

$$W_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 2B & 1 & 0 & 0 \\ -2B & 0 & 1 & 0 \end{bmatrix}, \quad b_1 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ -B \\ B \end{bmatrix}.$$

Then

$$W_1 \begin{bmatrix} c \\ x \\ y \\ 0 \end{bmatrix} + b_1 = \begin{bmatrix} c \\ x \\ y \\ 0 \\ 2Bc + x - B \\ -2Bc + y + B \end{bmatrix}$$

Program Conversion: Example 2

Next, let

$$W_2 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}, \quad b_2 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ -B \end{bmatrix}.$$

Then

$$W_2 \, \mathrm{ReLU} \left(W_1 \begin{bmatrix} c \\ x \\ y \\ 0 \end{bmatrix} + b_1 \right) + b_2$$

$$= \begin{bmatrix} c \\ x \\ y \\ \mathrm{ReLU}(2Bc + x - B) + \mathrm{ReLU}(-2Bc + y + B) - B \end{bmatrix}.$$

Program Conversion: Example 2

Finally, let

$$W_3 = \begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix}, \; b_3 = 0.$$

Then

$$W_3 \, \textsc{ReLU} \left(W_2 \, \textsc{ReLU} \left(W_1 \begin{bmatrix} c \\ x \\ y \\ 0 \end{bmatrix} + b_1 \right) + b_2 \right) + b_3$$

produces the required output, provided that $x, y \leq B$.

Sketch of the proof of the main theorem

Let P be a program of length L which outputs a result $P(x) \in \{0, 1\}$ for each input $x \in \{1, \ldots, N\}^d$. Let B be the maximum possible value attained by a variable within the program. Let V be the number of variables in P.

By the previous theorem, there exists a neural network f of description length $\leq C_1 L^3 V^2 \ln B$ which encodes the program P.

Let $\mathcal{N}_s$ be the set of all networks with description length $\leq s$,

One can show by a counting argument that $|\mathcal{N}_s| \leq e^{C_2 s}$.

Thus, if s is the description length of f, then $|\mathcal{N}_s| \leq e^{C_3 L^3 V^2 \ln B}$.

Proof sketch, contd.

Take any two networks $f_1, f_2 \in \mathcal{N}_s$ which disagree on a subset of $\{1, \ldots, N\}^d$ with μ-measure $\geq \epsilon$.

The chance that f_1, f_2 agree on the data is $\leq (1-\epsilon)^n$, where n is the number of data points.

Let A be the event that there exist $f_1, f_2 \in \mathcal{N}_s$ which disagree on a subset μ-measure $\geq \epsilon$ but agree on the data. By the above observation,

$$\mathbb{P}(A) \leq \binom{|\mathcal{N}_s|}{2}(1-\epsilon)^n \leq C_4 e^{C_5 L^3 V^2 \ln B - n\epsilon}.$$

Proof sketch, contd.

Let $\widehat{f}$ be a minimum description length network which exactly fits the data.

Then $\widehat{f} \in \mathcal{N}_s$, since $f \in \mathcal{N}_s$ and $\widehat{f}$ has description length $\leq$ the description length of f. Also, f and $\widehat{f}$ agree on the observed data.

On the event A^c, $\widehat{f}$ and f will agree on a subset S with $\mu(S) \geq 1 - \epsilon$, so they will agree on a new test point x with probability $\geq 1 - \epsilon$. Thus,

$$\mathbb{P}(\mathbb{P}(\widehat{f}(x) \neq f(x)|\text{data}) > \epsilon) \leq \mathbb{P}(A) \leq \min\{C_4 e^{C_5 L^3 V^2 \ln B - n\epsilon}, 1\}.$$

$\mathbb{P}(\widehat{f}(x) \neq f(x))$ is the integral of the left side over ϵ from 0 to 1. This completes the proof.

Limitations of our approach

Our notion of programs is somewhat restricted.

Although it accommodates many interesting examples, notice that the number of variables cannot scale with the inputs.

Moreover, arrays and accessing arrays with variable locations are not allowed.

Other natural expressions are disallowed, such as while loops.

Furthermore, all variables must be positive integers, and must be bounded by a constant B.

The way the main result depends on B precludes programs that do an exponential amount of computation in N.

Possible extensions

Many of the limitations can be overcome by increasing the expressivity of the programming language, while considering more expressive description measures.

As long as there is a conversion between short programs and neural networks of low complexity, the idea behind the proof of our theorem carries through.

By extending the programming language, other neural network architectures beyond feedforward networks may be considered.

For example, can generalization guarantees be obtained for convolutional neural network architectures on structured image data?

Possible extensions, contd.

Can similar guarantees be obtained for recurrent architectures on structured sequence data?

In particular, there has been much recent interest in the transformer architecture, in an attempt to explain various phenomena in large language models such as in-context learning, out-of-distribution generalization, and length generalization.

Specializing our argument to transformers and minimum description learning would be of interest.

Possible extensions, contd.

In some cases, the interpretability of $\widehat{f}$ can be of interest. This relates to the mechanistic interpretability literature.

For example, if the program P were unknown, it would be interesting to investigate to what extent $\widehat{f}$ describes the program P. Our results do not speak to this; we only provide conversions from simple neural programs to neural networks and not vice-versa.

Finally, as mentioned earlier, it would be of great interest to show that exactly fitting networks found by gradient descent or stochastic gradient descent have small description length with high probability.

Reference

The results presented in this talk are from:

Chatterjee, S. and Sudijono, T. (2025). Neural Networks Generalize on Low Complexity Data. arXiv:2409.12446. To appear in the Annals of Statistics.

Thank you!