A GENERAL APPROACH FOR LEARNING CONSTITUTIVE RELATIONSHIPS (PHYSICAL LAWS) FROM NEURAL NETWORKS

ANDREW TEMPLE, AARON STEBNER, PETER COLLINS, AND BRANDEN KAPPES

1. Introduction

In the main, a disconnect exists between physically-based equations that describe observations of complex phenomena and models derived from machine learning methods, including neural networks. This disconnect results in misunderstandings (at a minimum) and philosophical gulfs (worse) between the data scientists and those scientists deeply rooted in disciplines which traditionally study the various complex phenomena. On the one hand, many academics and scientists have called machine learning, data analytics, and neural networks "black boxes", implying that they are either not understood or that they are simply another, perhaps crude, tool. On the other hand, there are those

2. Methods

Fully dense neural network (NN) architectures, such as the one shown in Figure 1, perform a sequence of affine transformations, $\mathbf{z}_i \leftarrow \boldsymbol{\theta}_i \mathbf{x}^{(i)}$, followed by element-wise functional operations, $\sigma(\mathbf{z}_i)$ to introduce non-linearity at each layer; that is, each layer stretches and distorts the underlying space.

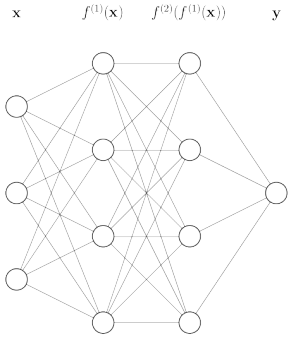

Figure 1. Schematic view of a fully dense neural network. Each sequence of affine and non-linear transformations are captured in the function, $f_i(\mathbf{x})$: $\mathbf{x}^{(i+1)} \leftarrow \sigma(\boldsymbol{\theta}_i \mathbf{x}^{(i)})$

The resulting network,

$$f(x) = \sigma(\boldsymbol{\theta}_n \sigma(\boldsymbol{\theta}_{n-1} \sigma(\dots \boldsymbol{\theta}_2 \sigma(\boldsymbol{\theta}_1 \mathbf{x})))) \tag{1}$$

is an arbitrary function generator, but at present, the network weights $\boldsymbol{\theta}_i$ can not map back to analytic forms that capture and describe the underlying physics. There are, however, many such mappings through polynomial series expansions,

$$f(x) = \sum_{n=0}^{\infty} a_n x^n \tag{2}$$

We hypothesize that the physics of a process can be extracted by fitting the polynomial expansions of known physical relationships to the polynomial coefficients of a polynomial series expansion of Equation (1).

The ReLU function is given below in Equation (3):

$$f(x) = \begin{cases} x, & \text{if } x > 0 \\ 0, & \text{otherwise} \end{cases} \tag{3}$$

$$f(x) = \sigma(\theta_n \sigma(\theta_{n-1} \sigma(\dots \theta_2 \sigma(\theta_1 x)))) \tag{4}$$

The variable $\boldsymbol{x}$ represents a column vector containing $\boldsymbol{n}$ elements, and thus, the output from each node will be multplied by the respective weight, θ, before the application of the subsequent acitivation function.

e.g.,

$$\sigma_2 \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \Rightarrow \begin{bmatrix} \sigma_{11} x_1 & \sigma_{12} x_2 & \dots & \sigma_{1n} x_n \\ \sigma_{21} x_1 & \sigma_{22} x_2 & & \\ \vdots & & \ddots & \\ \sigma_{n1} x_1 & & & \end{bmatrix} \tag{5}$$

Although ReLU (rectified linear units) have become a more common activation function, its discontinuity at $x = 0$ requires an infinite series to fully capture the behavior at this transition. However, the softplus function,

$$f(x) = log(1 + e^x) \tag{6}$$

The series expansion for the exponential

$$e^x = \sum_{k=0}^{\infty} \frac{x^k}{k!} \tag{7}$$

where, $a_k = \frac{1}{k!}$, can be expressed as

$$e^x = \sum_{k=0}^{\infty} x^k a_k \tag{8}$$

A similar series expansion for the logarithm

$$log(1+x) = \sum_{n=1}^{\infty} (-1)^{n+1} \frac{x^n}{n} \tag{9}$$

where, $x = e^x$, allows for the softplus function to be represented as

$$f(x) = \sum_{n=1}^{\infty} \frac{(-1)^{n+1}}{n} (\sum_{k=0}^{\infty} x^k a_k)^n \tag{10}$$

If the expansion of e^x is performed, then it can be seen that

$$(\sum_{k=0}^{\infty} x^k a_k)^n = (a_0 + x(a_1 + x(a_2 + xa_3...)))^n \tag{11}$$

Using the binomial theorem

$$(a+b)^n = \sum_{m=0}^{n} \binom{n}{m} a^{n-m} b^m \tag{12}$$

where, $a = a_0$ and $b = (x(a_1 + x(a_2 + xa_3...)))^n$, the series expansion of the exponential can be represented as

$$(\sum_{k=0}^{\infty} x^k a_k)^n = \sum_{m=0}^{n} \binom{n}{m} a_0^{n-m} (a_1 + x(a_2 + xa_3...))^n x^n \tag{13}$$

Continuing with this approach, it can be seen that ...

The analytical form, combining Equations (1) and (**??**), the estimated output of a two-layer NN can be written as an expansion:

$$
\begin{aligned}
\mathbf{y}_1 &= \sum_{k=0}^{\infty} a_k (\boldsymbol{\theta}_1^T \mathbf{x})^{\circ k} \\
\mathbf{y}_2 &= \sum_{k=0}^{\infty} b_k (\boldsymbol{\theta}_2^T \mathbf{y}_1)^{\circ k} \\
&= b_0 \mathbf{1} + \\
&\quad + b_1(\tilde{a}_0 + (\tilde{a}_1 + (\tilde{a}_2 + (\tilde{a}_3 + (\ldots)\tilde{\mathbf{x}})\tilde{\mathbf{x}})\tilde{\mathbf{x}})\tilde{\mathbf{x}}) \\
&\quad + b_2(\tilde{a}_0 + (\tilde{a}_1 + (\tilde{a}_2 + (\tilde{a}_3 + (\ldots)\tilde{\mathbf{x}})\tilde{\mathbf{x}})\tilde{\mathbf{x}})\tilde{\mathbf{x}})^2 \\
&\quad \vdots \\
&\quad + b_k(\tilde{a}_0 + (\tilde{a}_1 + (\tilde{a}_2 + (\tilde{a}_3 + (\ldots)\tilde{\mathbf{x}})\tilde{\mathbf{x}})\tilde{\mathbf{x}})\tilde{\mathbf{x}})^k \\
&\quad \vdots
\end{aligned} \tag{14}
$$

where $\tilde{a}_i = \boldsymbol{\theta}_2^T a_i$ and $\tilde{\mathbf{x}} = \boldsymbol{\theta}_1^T \mathbf{x}$. All $\boldsymbol{\theta}_i$, $\mathbf{x}$, and $\mathbf{y}$ are augmented to include the bias, $\mathbf{b}_i$, that is,

$$
\mathbf{x} \;:\; \mathbf{x} \leftarrow \begin{pmatrix} 1 \\ \mathbf{x} \end{pmatrix} = \begin{pmatrix} 1 \\ x_0 \\ x_1 \\ \vdots \\ x_n \end{pmatrix} \tag{15}
$$

$$
\mathbf{y} \;:\; \mathbf{y} \leftarrow \begin{pmatrix} 1 \\ \mathbf{y} \end{pmatrix} \tag{16}
$$

$$
\boldsymbol{\theta}_i \;:\; \boldsymbol{\theta}_i \leftarrow \begin{pmatrix} \mathbf{b}_i & \boldsymbol{\theta}_i \end{pmatrix} \tag{17}
$$

The element-wise exponent operator, $\mathbf{x}^{\circ m} = (x_1^m \; x_2^m \; \cdots \; x_n^m)^T$, raises each element of a vector or matrix to the specified power, m.

From Equation (**??**), $a_i = 0$ for $i = 2, 4, 6, \ldots$, and therefore,

$$
\begin{aligned}
\mathbf{y}_1 =& \sum_{k=0}^{\infty} a_k (\boldsymbol{\theta}_1^T \mathbf{x})^{\circ k} \\
\mathbf{y}_2 =& \sum_{k=0}^{\infty} b_k (\boldsymbol{\theta}_2^T \mathbf{y}_1)^{\circ k} \\
=& b_0 \mathbf{1} + \\
&+ b_1(\tilde{a}_0 + (\tilde{a}_1 + (\tilde{a}_3 + (\tilde{a}_5 + (\ldots)\tilde{\mathbf{x}}^{\circ 2})\tilde{\mathbf{x}}^{\circ 2})\tilde{\mathbf{x}}^{\circ 2})\tilde{\mathbf{x}}) \\
&+ b_2(\tilde{a}_0 + (\tilde{a}_1 + (\tilde{a}_3 + (\tilde{a}_5 + (\ldots)\tilde{\mathbf{x}}^{\circ 2})\tilde{\mathbf{x}}^{\circ 2})\tilde{\mathbf{x}}^{\circ 2})\tilde{\mathbf{x}})^{\circ 2} \\
&\vdots \\
&+ b_k(\tilde{a}_0 + (\tilde{a}_1 + (\tilde{a}_3 + (\tilde{a}_5 + (\ldots)\tilde{\mathbf{x}}^{\circ 2})\tilde{\mathbf{x}}^{\circ 2})\tilde{\mathbf{x}}^{\circ 2})\tilde{\mathbf{x}})^{\circ k} \\
&\vdots \\
\mathbf{y}_2 =& \sum_{N=0}^{\infty} \sum_{k=0}^{N} \sum_{l=0}^{k} \sum_{m=0}^{l} \ldots b_N \binom{N}{k, l, m, \ldots} \tilde{a}_0^k \tilde{a}_1^l \tilde{a}_3^m \ldots \tilde{\mathbf{x}}^{\circ(N-k\ldots)} (\tilde{\mathbf{x}}^2)^{\circ(N-k-l\ldots)} (\tilde{\mathbf{x}}^2)^{\circ(N-k-l-m\ldots)} \\
& \qquad (18) \\
=& \sum_{N=0}^{\infty} \sum_{k=0}^{N} \sum_{l=0}^{k} \sum_{m=0}^{l} \ldots b_N \binom{N}{k, l, m, \ldots} \tilde{a}_0^k \tilde{a}_1^l \tilde{a}_3^m \ldots \tilde{\mathbf{x}}^{\circ(l+m+n+\ldots)} (\tilde{\mathbf{x}}^{\circ 2})^{\circ(m+n+\ldots)} (\tilde{\mathbf{x}}^{\circ 2})^{\circ(n+\ldots)}
\end{aligned}
$$

where $k + l + m + n + \ldots = N$. Collecting coefficients and terms of power k,

$$\mathbf{y}_2 = \sum_{k=0}^{\infty} c_k \tilde{\mathbf{x}}^{\circ k}$$

that, having the same form as Equation (**??**) creates a sequential process for determining the coefficients of the power series expansion of each layer in an ANN. Importantly, the output layer in a ANN regression is a single node with a linear activation, so the final layer, y_f, working from the last hidden layer, $\mathbf{y}_n$, is simply,

$$y_f = \boldsymbol{\theta}_n^T \mathbf{y}_n \qquad (19)$$

3. Discussion

Many non-trivial problems in materials science, and in science more broadly, are explained not through a single constitutive relationship, but through a superposition of contributing physics.

The goal of this approach is to identify the coefficients of an hypothesized constitutive relationship, coefficients that capture the specific physics of a process, through a least-squares fit between the covector space of a neural network series expansion, $\boldsymbol{\alpha}(\boldsymbol{\theta})$, which is

a function of the model parameters, and the covector space of the constitutive relationship, $\boldsymbol{\beta}(C_k)$, which is a function of the physical constants of the model.

Having fit the model parameters, $\boldsymbol{\theta}$, on a vector space spanned by the column vectors of $\mathbf{x}$, the coefficients (the covector basis) of the neural network expansion, $\boldsymbol{\alpha}(\boldsymbol{\theta})$, capture the functional relationship between the input space and the response space, both affine and non-linear contributions, introduced through those parameters, $\boldsymbol{\theta}$, and the coefficient generating functions for the activation, e.g. Equations **??** (Rectified Linear Unit, ReLU) and **??** (softmax), respectively.

Naturally, the activation generating functions must match the activation function chosen in the neural network model architecture. Equation **??** is derived for ReLU activation, the most common hidden layer activation. (Generating functions for other activations are provided in the appendix.) In addition to the hidden layers, activation functions must also be chosen for the output layer. The two most common output activations are linear (Eq. **??**) and softmax (Eq. **??**) for regression and classification, respectively.

ReLU activation,

$$\sigma(z) = \begin{cases} z & \text{if } z > 0, \\ 0 & \text{otherwise} \end{cases}$$

is discontinuous in the first derivative at $z_i = 0$. Therefore, the coefficient generating function of this activation must either be either a function of the input data, $\mathbf{z}$ or a small modification must be made to the softplus,

$$\sigma(z;\alpha) = \frac{1}{\alpha}\ln\left(1 + e^{\alpha x}\right). \tag{20}$$

In the limit as α approaches infinity, this converges to the ReLU. Practically, though, α can be assigned a large value and the coefficient generating function no longer depends on the input data, see Equation **??** in the appendix. However, because of the high computational cost of expressing the coefficients using the modified softplus, and the relative low cost of forward evaluation of the trained neural network in order to apply Equation **??**, expressing the coefficient generating function of ReLU in terms of the input data, as in Equation **??**, rather than the training-data-agnostic approach in Equation **??** would seem more practical.

Condition 1. *Both the neural network and the constitutive relationship must depend on the same independent variables.*

That is, they must be described on the same basis vectors. The fit between the coefficients of the neural network expansion–$\boldsymbol{\alpha}$, the covector space of the neural network's basis vectors–and the coefficients of the series expansion of the constitutive relationship (its covector space, $\boldsymbol{\beta}$) is only possible because both span the same subspace and share a common description of the solution within that subspace, that is, the covector spaces are colinear. Help! I don't have the mathematical background to say this correctly. That is, choose A such that A maps between the vector and covector spaces. If $A : A(Y) \to Y^*$ and $A : A(X) \to X^*$, then $Y^* = X^*$ if, and only if, $Y = X$.

No neural network layer operation that leaves the vector space intact, such as dropout or explicitly setting specific θ_{ij}, would compromise the comparability between the covector

spaces of the neural network and constitutive relationship series expansions. However, any neural network layer operation that changes the vector space, such as layer normalization, must be explicitly included in the neural network series expansion. In order to maintain a consistent framework, any such change should be expressed as an activation function and incorporated through a coefficient generating function. For example, the coefficient generating function for data standardization would be

$$\alpha_n = \begin{cases} -\overline{\mathbf{z}}/\sigma & \text{if } n = 0, \\ 1/\sigma & \text{if } n = 1, \\ 0 & \text{otherwise} \end{cases} \tag{21}$$

where

$\overline{\mathbf{z}}$ = Arithmetic mean of the layer input values.

σ = Standard deviation of the layer input values.

There is no guarantee that the input vector space is orthogonal, so cross-term interactions will not vanish and must be included explicitly. Therefore, this method is based on a fully dense neural network to introduce all cross-terms. Furthermore when introduced into a polynomial expansion, element-wise exponentiation also explicitly captures all combinations of powers of all cross-terms. Equation 18 shows the polynomial series expansion for the first layer of a neural network,

$$y^{(1)} = \sum_k \alpha_k^{(1)} \left(\boldsymbol{\theta}^{(1)} \mathbf{x} \right)^{\circ k},$$

which relies on the element-wise exponential, $(\bullet)^{\circ n}$, as does the expansion of all layers. Unlike scalar exponentiation, element-wise exponentiation does not distribute, as seen in Equation 22, and because element-wise exponentiation does not distribute, this equation explicitly captures all possible (second, $x_i^m x_j^n$; third, $x_i^m x_j^n x_k^p$; fourth, $x_i^m x_j^n x_k^p x_l^r$; etc.) cross-interactions of each term in $(\boldsymbol{\theta}\mathbf{x})$ at all polynomial orders. This is equivalent, then, of expanding over the basis set that includes all cross-interactions in the input vector space, and therefore, does not assume the column vectors of the input spare are orthogonal.

This should go in the methods section.

For two matrices, $A : A \in \mathbb{R}^{m \times n}$ and $B : B \in \mathbb{R}^{n \times q}$,

$$
\begin{aligned}
(AB)^n &= \left(\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1p} \\ a_{21} & a_{22} & & \vdots \\ \vdots & & \ddots & \\ a_{m1} & & \cdots & a_{mp} \end{pmatrix} \begin{pmatrix} b_{11} & b_{12} & \cdots & b_{1q} \\ b_{21} & b_{22} & & \vdots \\ \vdots & & \ddots & \\ b_{p1} & & \cdots & b_{pq} \end{pmatrix} \right)^{\circ p} \\
&= \begin{pmatrix} \mathbf{a}_{1k}\mathbf{b}_{k1} & \mathbf{a}_{1k}\mathbf{b}_{k2} & \cdots & \mathbf{a}_{1k}\mathbf{b}_{kq} \\ \mathbf{a}_{2k}\mathbf{b}_{k1} & \mathbf{a}_{2k}\mathbf{b}_{k2} & & \vdots \\ \vdots & & \ddots & \\ \mathbf{a}_{mk}\mathbf{b}_{k1} & & \cdots & \mathbf{a}_{mk}\mathbf{b}_{kq} \end{pmatrix}^{\circ p} \\
&= \begin{pmatrix} (\mathbf{a}_{1k}\mathbf{b}_{k1})^p & (\mathbf{a}_{1k}\mathbf{b}_{k2})^p & \cdots & (\mathbf{a}_{1k}\mathbf{b}_{kq})^p \\ (\mathbf{a}_{2k}\mathbf{b}_{k1})^p & (\mathbf{a}_{2k}\mathbf{b}_{k2})^p & & \vdots \\ \vdots & & \ddots & \\ (\mathbf{a}_{mk}\mathbf{b}_{k1})^p & & \cdots & (\mathbf{a}_{mk}\mathbf{b}_{kq})^p \end{pmatrix} \qquad \text{(22a)} \\
(\mathbf{a}_{ik}\mathbf{b}_{kj})^p &= (a_{i1}b_{1j} + a_{i2}b_{2j} + \cdots + a_{in}b_{nj})^p \\
&= \sum_{k_1=0}^{p} \sum_{k_2=0}^{p-k_1} \cdots \sum_{k_{n-1}=0}^{p-k_1-k_2-\ldots-k_{n-2}} \binom{p}{k_1, k_2, \ldots, k_n} \prod_{m=1}^{n} (a_{im}b_{mj})^{k_m} \qquad \text{(22b)}
\end{aligned}
$$

where $p = \sum_i k_i$, i.e. $k_n = p - k_1 - k_2 - \cdots - k_{n-1}$ and summation over a repeated index is assumed. That is, Equation 22b is the expansion of the element-wise exponent of a vector product, $(\mathbf{ab})^{\circ p} = (\mathbf{ab})^p \neq \mathbf{a}^{\circ p}\mathbf{b}^{\circ p}$.

To avoid implicit bias, data preprocessing is an important first step in training a neural network. Commonly, data is whitened, also known as scaling or standardization, $\mathbf{z}_s^{(k)}$: $\mathbf{z}_s^{(k)} = \frac{\mathbf{z}^{(k)} - \overline{\mathbf{z}^{(k)}}}{\sigma}$, where $\overline{\mathbf{z}^{(k)}}$ is the arithmetic mean and σ the standard deviation of data in layer, k. However, a model trained on such scaled data would no longer share the primal space of the constitutive relationship (unscaled).

To accommodate whitened data, which is necessary to remove implicit bias while training a neural network, the constitutive relationship must also operate in the scaled primal space. After model training and fit of the physical constants in the constitutive relationship from $\boldsymbol{\alpha}$, the dual space of the neural network expansion, and $\boldsymbol{\beta}$, the dual space of the constitutive relationship expansion, the inverse transformation should be applied to closed-form constitutive relationship to extract the physical constant, e.g.

$$
\begin{aligned}
f(x) &= C_0 + C_1 x_s \\
&= C_0 + C_1 \left(\frac{x - \overline{x}}{\sigma} \right) \\
&= C_0' + C_1' x
\end{aligned}
$$

where

$$C_0' = C_0 - C_1\overline{x}/\sigma$$
$$C_1' = C_1/\sigma$$

Note that this accommodation may introduce a bias/offset even in constitutive relationships where no such offset was present before.

TBW. This must answer the question: how do we know if we've measured the right things–not the number of measurements, but that we have enough information? How do we know that the solution has converged? Example: A model is to be fit to the number of cakes produced by a bakery. If we are given weights of flour and sugar and number of eggs, our model can accurately tell us the *volume* of cakes produced, but not the number. If this bakery makes cupcakes, but the model is trained across a spectrum of bakers, such as purveyors of wedding cakes and catering companies who work with large sheet cakes, then our dimensions (flour, sugar, eggs) are insufficient to fit the number of cakes produced. If, however, we also include number of orders and revenue, some information about the *quanta* of cakes is baked into those two additional dimensions (sorry, I couldn't help myself). Therefore, a model based only on (flour, sugar, eggs) is dimensionally insufficient, but a model based on (flour, sugar, eggs, order size, revenue) is dimensionally sufficient.

This is a broader question that may be beyond the scope of this paper. Let's return to this if, after completing the first pass, we feel that this can be addressed by what we've done.

This leads to a seven-step process for systematically and incrementally extracting physics information from an ANN:

(1) Collect data–features and targets–for which relationships are expected to exist.
(2) Design and train a fully dense multi-layer perceptron network (ANN).
(3) Build a power series expansion from the architecture of this ANN, using Equations (**??**) and (18) to populate the coefficients using the trained weights from the neural network.
(4) Hypothesize a constitutive relationship between the feature space and the target space.
(5) Recast the terms in the hypothesis function from #4 as power series expansions, creating power series coefficient generating functions that are functions of the constitutive model fitting parameters. An example of this process is provided below, and a table of select power series expansions relevant to materials research are provided in Table (**??**).
(6) Perform an optimization, *e.g.* least squares, fit to find the fitting parameters from #5
(7) Calculate the residuals of the ANN power series expansion coefficient vector, and from this residual vector, the error in the model. If the accuracy is sufficient for the application, stop; otherwise, expand the constitutive relationship from step #4 and repeat.

TABLE 1. Examples of coefficient generating functions for functional forms commonly found in materials physics.

k	Ca^x	Cx^n	$Ce^{-\beta x}$	$Cx^{-1/2}$
0	1	–	C	C
1	$C\ln a$	–	$-\beta C$	$-\frac{1}{2}C$
2	$\frac{(\ln a)^2}{2}C$	–	$\frac{\beta^2}{2}C$	$\frac{3}{8}C$
$\vdots$			$\vdots$	
n	$\frac{(\ln a)^n}{n!}C$	$\begin{cases} C & \text{if } k=n \\ 0 & \text{otherwise} \end{cases}$	$(-1)^n\frac{\beta^n}{n!}C$	$C\prod_{i=1}^{n}(-1)\frac{2i-1}{2i}$