EngSci 721

Inverse Problems and Learning From Data

Oliver Maclaren (oliver.maclaren@auckland.ac.nz)

1. *Basic concepts* [5 lectures + 1 Tutorial]

   Forward vs inverse problems. Well-posed vs ill-posed problems. Algebra and calculus of inverse problems (left and right inverses, generalised and pseudo inverses, resolution operators, matrix calculus). Representing higher dimensional problems (image data etc).

2. *Instability and regularisation in linear and nonlinear problems* [6 lectures + 1 Tutorial]

   Instability and related issues for generalised inverses. Introduction to regularisation and trade-offs. Tikhonov regularisation. Higher-order Tikhonov regularisation. Sparsity and regularisation using different norms. Truncated singular value decomposition. Iterative regularisation, including stochastic/mini-batch gradient descent.

3. *Further topics* [3 lectures + 1 Tutorial]

   Regularisation parameter choice, including statistical and machine learning views of regularisation. Confidence sets for linear and nonlinear models. Physics-informed machine learning and neural networks.

Module overview

Inverse Problems and Learning From Data (*Oliver Maclaren*)
[~14 lectures/3 tutorials]

Lecture 14: Neural models (Nonlinear part II)

Topics:

- An attempt to learn about neural networks via matrix calculus!

Eng Sci 721 : Lecture 14: Neural models

ie Nonlinear II

– An attempt to apply matrix calculus & understand neural nets!

see also (loosely based on)

The Matrix Calculus You Need For Deep Learning

Terence Parr and Jeremy Howard

(Terence is a tech lead at Google and ex-Professor of computer/data science in University of San Francisco's MS in Data Science program. You might know Terence as the creator of the ANTLR parser generator. For more material, see Jeremy's fast.ai courses and University of San Francisco's Data Institute in-person version of the deep learning course.)

Please send comments, suggestions, or fixes to Terence.

Printable version (This HTML was generated from markup using bookish). A Chinese version is also available (content not verified by us).

Abstract

This paper is an attempt to explain all the matrix calculus you need in order to understand the training of deep neural networks. We assume no math knowledge beyond what you learned in calculus 1, and provide links to help you refresh the necessary math where needed. Note that you do **not** need to understand this material before you start learning to train and use deep learning in practice; rather, this material is for those who are already familiar with the basics of neural networks, and wish to deepen their understanding of the underlying math. Don't worry if you get stuck at some point along the way—just go back and reread the previous section, and try writing down and working through some examples. And if you're still stuck, we're happy to answer your questions in the Theory category at forums.fast.ai. **Note**: There is a reference section at the end of the paper summarizing all the key matrix calculus rules and terminology discussed here.

https://explained.ai/matrix-calculus/

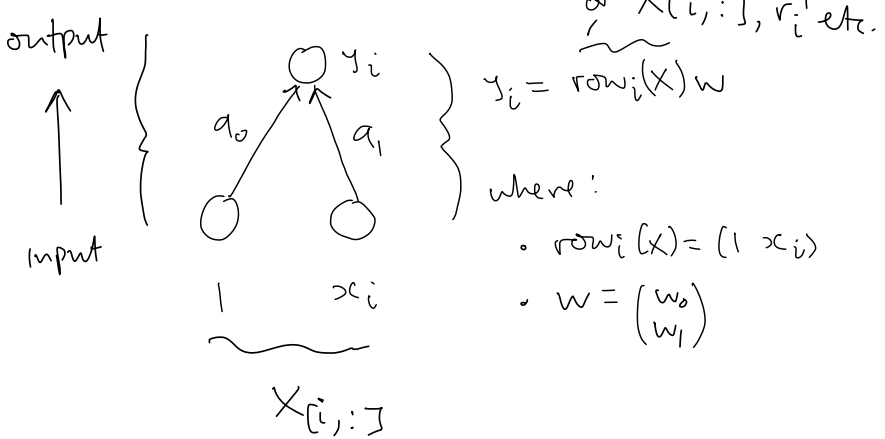
Linear models (again)

Consider

$y_i = w_0 + w_1 x_i$ for observations $i = 1, \ldots, m$

ie: $\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix} = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_m \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \end{bmatrix}$

We can write as a 'network diagram':



→

Constant or 'bias' terms

Given an $m \times n$ matrix X, let's introduce the notation

$$\bar{X} = [1_m \; X] = \text{hcat}((1_m, X))$$

$$= \begin{bmatrix} 1 & & \\ 1 & X & \\ \vdots & & \\ 1 & & \end{bmatrix}$$

(Think 'closure' or 'completion' of X by 1)

where 1_m is an $m \times 1$ column vector of ones. ie we have added an extra column of all ones to X giving an $m \times (n+1)$ matrix, or an $m \times \bar{n}$ matrix, where $\bar{n} = n+1$

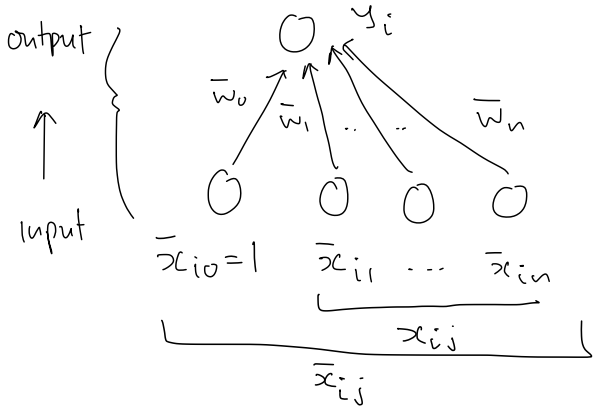
Also, given a coefficient or weight vector w of length n, define

$$\bar{w} = \begin{bmatrix} w_0 \\ w \end{bmatrix} \begin{matrix} \} 1 \\ \} n \end{matrix} \quad n+1 = \bar{n}$$

Thus, $y_i = w_0 + w_1 x_i$, $i = 1, \dots, m$ becomes

$m \times 1$ $\quad \boxed{y = [1 \; X] \begin{bmatrix} w_0 \\ w \end{bmatrix} = \bar{X}\bar{w}}$

More generally, write



for x_{ij} = i^{th} obs. of j^{th} feature, $j = 1, \dots n$

$\bar{x}_{ij}$ = " " , $j = 0, \dots, n$

ie $\bar{x}_{ij} = x_{ij}$, $j = 1, \dots n$

& $\bar{X} = [\bar{x}_{ij}]$ matrix

$\Rightarrow$ $\boxed{y = \bar{X}\bar{w}}$

m $\quad (m \times \bar{n}) \cdot (\bar{n})$

$\rightarrow$ So far just a linear model

Nonlinearity & element-wise functions

Next consider a nonlinear transformation of the output of the linear part instead:

$y_i = f(w_0 + w_1 x_i)$, for nonlinear f,

or $y_i = f(\text{row}_i(\bar{X})\bar{w})$.

For vector inputs, $z = \begin{pmatrix} z_1 \\ z_2 \\ \vdots \\ z_m \end{pmatrix}$ say, (or $z_0, \ldots z_m$ etc)

define f to act **element-wise**

f for vector input: $f(z) := \begin{pmatrix} f(z_1) \\ f(z_2) \\ \vdots \\ f(z_m) \end{pmatrix}$ (f for scalar input) (Similarly for row vectors & matrices)

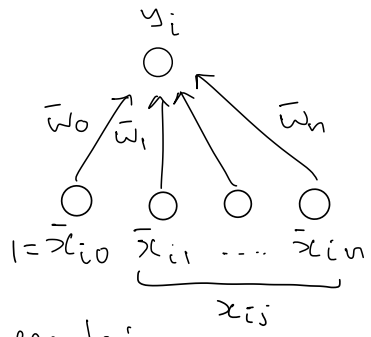
Then

$$\boxed{y = f(\bar{X}\bar{w})}$$

Defines a model which is part linear with a subsequent elementwise nonlinear transformation of the output

→ see later for more on element-wise f.

Neuron! (analogy to brain blah blah)



represents:

$$y_i = f(\text{row}_i(\bar{X})\bar{w})$$

ie $y = f(\bar{X}\bar{w})$ for vector y: dim m, matrix $\bar{X}$ $m \times \bar{n}$, vector $\bar{w}$ $\bar{n}$

$\bar{w}$: weights

$\bar{X}$: inputs/features

f: nonlinear activation function

Note: if f is Heaviside, then 'perceptron' (1958)

if f sigmoid, is 'logistic regression'

Nonlinearity

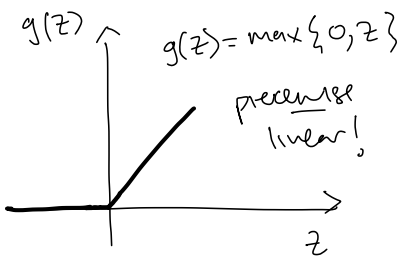
If f is linear then its just a linear model. This is also true for more complex models.

→ f should be nonlinear to be more exciting!

Typical:

ReLU ← common default

$g(z) = \max\{0, z\}$

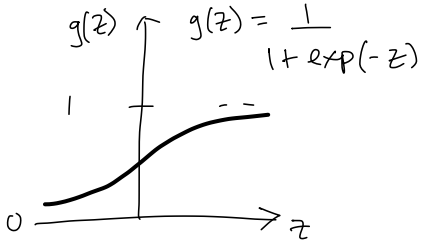


piecewise linear!

useful for binary output

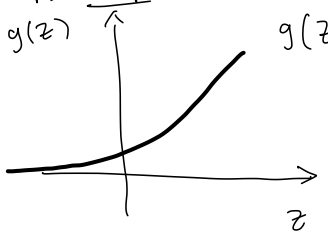
Sigmoid

$g(z) = \frac{1}{1+\exp(-z)}$



soft plus

$g(z) = \ln(1+\exp(z))$ ← smooth approx to ReLU



etc!

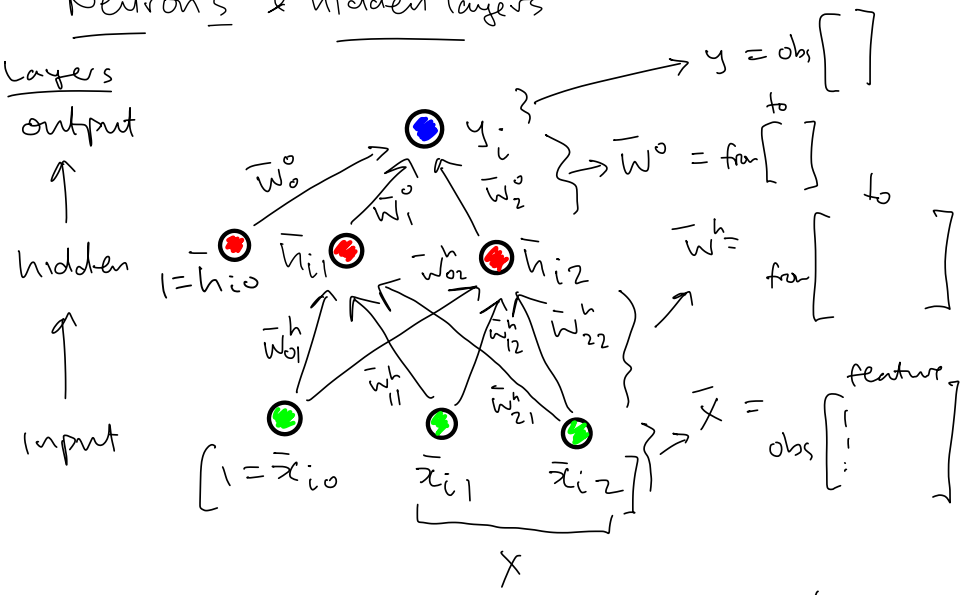
Neurons & hidden layers

A neural network typically contains many, many neurons ('regressions'!)

→ a feedforward neural network (also called multi-layer perceptrons) consists of a series of 'layers', each with multiple neurons:

- each 'layer' has multiple nonlinear (linear) regressions
- the 'next' layer uses the outputs of the previous layer as the inputs/features for its collection of regressions
- layers that aren't observable inputs or outputs are called 'hidden'
- 'deep': many hidden layers.

Neurons & hidden layers



First hidden layer : (or 'bias' term when weighted)

$\bar{h}_{i0} = 1 = x_{i0}$ (constant feature)

$\bar{h}_{i1} = f(\text{row}_i(X)\, w^h_{[:,1]}) = h_{i1}$

$\bar{h}_{i2} = f(\text{row}_i(X)\, w^h_{[:,2]}) = h_{i2}$

$\square\square = f\left(\begin{bmatrix}1 & \square & \square\end{bmatrix}\right)$

$\square\square\square = \square f\left(\begin{bmatrix}1 & \square & \square\end{bmatrix}\right)$

---

dim: $\begin{cases} \bar{X} : m \times \bar{n} \\ \bar{w}^h : \bar{n} \times p \end{cases}$

So :

$\bar{H}[i,0] = \bar{h}_{i0} = 1$

$\bar{H}[i,1:2] = [\bar{h}_{i1}\ \bar{h}_{i2}] = f(\text{row}_i(\bar{X})\,\bar{w}^h)$

$\Leftrightarrow \bar{H} = \begin{matrix} i \\ \downarrow \end{matrix} \begin{bmatrix} 1 & \cdots & 1 \end{bmatrix} \boxed{f(\bar{X}\bar{w}^h)}$

$= \begin{bmatrix} 1 & f\left(\begin{bmatrix}1\\1\\1\end{bmatrix}^{\bar{X}} \begin{bmatrix}\square & \square\end{bmatrix}^{\bar{w}^h}\right) \end{bmatrix}$

$= \begin{bmatrix} 1 & f\left(\begin{bmatrix}\square\end{bmatrix}\right) & f\left(\begin{bmatrix}\square\end{bmatrix}\right) \end{bmatrix}$

$\bar{X}\bar{w}^h_{[:,1]}$ $\quad$ $\bar{X}\bar{w}^h_{[:,2]}$

Since f element-wise (& applies to col element-wise too)

Also $H = f(\bar{X}\bar{w}^h)$

completed

not completed (need extra new 1 vector for $\bar{H}$)

Hidden layer cont'd

We've arranged as collection of individual regressions, one per hidden neuron.

Note though:

$$\bar{H} = [\mathbb{1}\ f(\bar{X}\bar{W}^h)] = [\mathbb{1}\ f(\bar{X}\bar{W}^h\mathbb{I})]$$

where $\mathbb{1}$ = vector of ones

$\mathbb{I}$ = identity matrix

$$\mathrm{vec}(\bar{H}) = \begin{bmatrix} \mathbb{1} \\ \mathrm{vec}(f(\bar{X}\bar{W}^h\mathbb{I})) \end{bmatrix}$$

$$\mathrm{vec}(\bar{X}\bar{W}^h\mathbb{I}) = (\mathbb{I}\otimes\bar{X})\,\mathrm{vec}(\bar{W}^h)$$

& $\mathrm{vec}(f(\cdot)) = f(\mathrm{vec}(\cdot))$ if f element-wise

$$(\mathbb{I}\otimes\bar{X})\,\mathrm{vec}(\bar{W}^h) = \begin{bmatrix} \bar{X} & 0 \\ 0 & \bar{X} \end{bmatrix} \begin{bmatrix} \begin{bmatrix} \bar{w}^h_{01} \\ \bar{w}^h_{11} \\ \bar{w}^h_{21} \end{bmatrix} \\ \begin{bmatrix} \bar{w}^h_{02} \\ \bar{w}^h_{12} \\ \bar{w}^h_{22} \end{bmatrix} \end{bmatrix}$$

$\Rightarrow$

So:

constant feature for use as input to next layer

$$\mathrm{vec}(\bar{H}) = \begin{bmatrix} \mathbb{1} \\ \bar{H}[:,1] \\ \bar{H}[:,2] \end{bmatrix} = \begin{bmatrix} \mathbb{1} \\ \bar{h}_1 \\ \bar{h}_2 \end{bmatrix}$$

where: (element-wise nonlinear; linear)

$$\begin{bmatrix} \bar{H}[:,1] \\ \bar{H}[:,2] \end{bmatrix} = f\left(\begin{bmatrix} \bar{X} & 0 \\ 0 & \bar{X} \end{bmatrix} \begin{bmatrix} \begin{bmatrix} \bar{w}^h_{01} \\ \bar{w}^h_{11} \\ \bar{w}^h_{21} \end{bmatrix} \\ \begin{bmatrix} \bar{w}^h_{02} \\ \bar{w}^h_{12} \\ \bar{w}^h_{22} \end{bmatrix} \end{bmatrix}\right)$$

($\Leftrightarrow$ $H = f((\mathbb{I}\otimes\bar{X})\mathrm{vec}(\bar{W}^h))$)

ie one big 'linear' regression for weights of prev. layer ... with a nonlinear transformation of output ... & ... an extra constant vector for later

(conceptually, if not computationally)

Output layer

For each observation i, we use the outputs of the hidden layer (here) as the inputs of the output layer

→ do same again! single output here though

→ possibly different activation function, g

→ include constant feature ('bias') from prev. layer × weight.

$$y_i = g(\bar{H}_{[i,:]}\bar{w}^o)$$

w^o : w^{output} : $\bar{P}\times 1$

$$y = g(\bar{H}\bar{w}^o)$$

$$\begin{bmatrix} \square \\ \square \\ \vdots \end{bmatrix} = g\left(\begin{bmatrix} 1 \\ 1 \\ \vdots \end{bmatrix} \begin{bmatrix} \square \end{bmatrix} \right)$$

} single 'regression' for single output node

Binary classification: $y \in [0,1]$, class 'probability'

→ use eg logistic, range = $(0,1)$

Composite

$$y = g(\bar{H}\bar{w}^o)$$
$$H = f(\bar{X}\bar{w}^h)$$

Note:
$$y = g([1\ H]\bar{w}^o)$$
$$H = f(\bar{X}\bar{w}^h)$$

So $$y = g([1\ f(\bar{X}\bar{w}^h)]\bar{w}^o)$$ } model prediction

unknowns $\bar{w}^h, \bar{w}^o$ } $\bar{n}p + \bar{p} = np+2p+1$ unknowns

givens $\bar{X}$

outputs y

---

want $D_{\bar{w}^h} y$ & $D_{\bar{w}^o} y$

ie $D_{\bar{w}} y = [D_{\bar{w}^h} y \quad D_{\bar{w}^o} y]$

→ derivative of prediction wrt param.

→ combine with ---

'Loss function' $l(y;d)$ (minimise)

compare $y(x)$ output to data d associated with x, ie (x,d) is full data.

└ call $d(x)$ & compare $y(x)$ & $d(x)$

→ sum of squares (even for classification!)

→ $-\log$ likelihood = $-\log$ prob. of data given param.

→ can use binary prob. dist. with y = prob. parameter

$$l(\bar{w};d) = l(y(\bar{w});d)$$

Want

$$D_{\bar{w}} l = D_y l \cdot D_{\bar{w}} y$$ ← hard part

$D_{\bar{w}} l$: vector; $D_y l$: vector; $D_{\bar{w}} y$: matrix (Jacobian)

$\bar{w}$ = array of all weights

Loss: simple case

$$l(y;d) = \frac{1}{2}\|y-d\|^2$$

$$= \frac{1}{2}(y-d)^T(y-d)$$

$$= \frac{1}{2}\left(y^T y - 2d^T y + d^T d\right)$$

$$D_y l = \frac{1}{2}\left(2y^T - 2d^T\right)$$

$$= y^T - d^T$$

$$= (y-d)^T \quad \text{[done]}$$

→

Hard part: $D_{\bar{W}} y$!

Warm-up

derivatives of

$$y = [1 \; \overbrace{H(\bar{W}^h)}^{\bar{H}}]\bar{W}^o$$
$$H = \bar{X}\bar{W}^h$$

$y : m \times 1$
$H : m \times p$
$\bar{H} : m \times \bar{p}$
$\bar{W}^h : \bar{n} \times p$
$\bar{X} : m \times \bar{n}$
$\bar{W}^o : \bar{p}$

if no activation functions

$\longrightarrow$

1. via differentials:

$$\bar{H} = [1 \; \bar{X}\bar{W}^h]$$
$$\Rightarrow d\bar{H} = [0 \; \bar{X} d\bar{W}^h]$$
$$= [0 \; dH]$$

$$dy = \bar{H} d\bar{W}^o + d\bar{H}\bar{W}^o$$
$$= \bar{H} d\bar{W}^o + [0 \; \bar{X} d\bar{W}^h]\bar{W}^o$$
$$= \bar{H} d\bar{W}^o + \bar{X} d\bar{W}^h w^o, \quad w^o = \begin{bmatrix} w_1^o \\ \vdots \\ w_p^o \end{bmatrix}$$
$$= \bar{H} d\bar{W}^o + (w^{oT} \otimes \bar{X}) \, d\mathrm{vec}(\bar{W}^h)$$
$$= \bar{H} \, d\mathrm{vec}(\bar{W}^o) + (w^{oT} \otimes \bar{X}) \, d\mathrm{vec}(\bar{W}^h)$$

$$\Rightarrow D_{\bar{W}^o} y = \bar{H}, \quad D_{\bar{W}^h} y = w^{oT} \otimes \bar{X}$$

$\bar{m} \times \bar{p}$ $\quad$ $m \times \bar{p}$ $\quad$ $m \times \bar{n}p$ $\quad$ $m \times \bar{n}p$ ✓

(no activation functions)

$\longrightarrow$

unknowns: $\begin{cases} \bar{p} = p+1 \\ \bar{n}p = (n+1)p \end{cases}$

2. Via vectorised chain rule

$\bar{W}^h : \bar{n} \times p$

$\bar{W}^o : \bar{p}$

$\bar{H} : m \times \bar{p}$

$$y = [1 \; H(\bar{W}^h)]\bar{W}^o = \bar{H}\bar{W}^o$$

$$H = \bar{X}\bar{W}^h, \quad \bar{H} = [1 \; \bar{X}\bar{W}^h]$$

$$D_{\bar{W}^o} y = \frac{\partial \text{vec}(y)}{\partial \text{vec}(\bar{W}^o)} = \bar{H} \qquad (\bar{W}^o \text{ already vec})$$

$$D_{\bar{W}^h} y = \frac{\partial \text{vec}(y)}{\partial \text{vec}(\bar{W}^h)} = \frac{\partial \text{vec}(y)}{\partial \text{vec} H} \frac{\partial \text{vec} H}{\partial \text{vec} \bar{W}^h}$$

$$y = 1W_o^o + H(\bar{W}^h)W^o$$

$$\text{vec}(y) = \text{vec}(1W_o^o) + \text{vec}\left(H(\bar{W}^h)W^o\right)$$

$$= 1W_o^o + (W^{oT} \otimes I_m)\text{vec}(H)$$

$$\frac{\partial \text{vec}(y)}{\partial \text{vec}(H)} = W^{oT} \otimes I_m \quad \Big\} m \times pm$$

$$\frac{\partial \text{vec}(H)}{\partial \bar{W}^h} = I_p \otimes \bar{X} \quad \Big\} pm \times p\bar{n}$$

$$D_{\bar{W}^h} y = (W^{oT} \otimes I_m)(I_p \otimes \bar{X})$$

$$= (W^{oT} I_p) \otimes (I_m \bar{X})$$

$$= W^{oT} \otimes \bar{X}$$

Kronecker: $(A \otimes B)(C \otimes D) = AC \otimes BD$

$\hookrightarrow (m \times p\bar{n})$ ✓

Activation functions ?

$$y = g(\bar{H}(\bar{W}^h)\bar{W}^o)$$

$$\bar{H} = [1 \; f(\bar{X}\bar{W}^h)] = [1 \; H(\bar{X}\bar{W}^h)]$$

Here f & g act element-wise

→ need to be careful when calc. derivatives →

Element-wise functions & derivatives

We have considered 'elementwise' functions eg defining for scalar input function $\phi(z)$,

$$\phi(x) = \begin{bmatrix} \phi(x_1) \\ \phi(x_2) \\ \phi(x_3) \end{bmatrix}$$

for vector input x.

More explicitly we could write

$$\tilde{\phi}(x) = \begin{bmatrix} \tilde{\phi}_1(x_1, x_2, x_3) \\ \tilde{\phi}_2(x_1, x_2, x_3) \\ \tilde{\phi}_3(x_1, x_2, x_3) \end{bmatrix}$$

$\tilde{\phi}$ is ϕ for vector input

where $\tilde{\phi}_1(x_1, x_2, x_3) := \phi(x_1)$
$\tilde{\phi}_2(x_1, x_2, x_3) := \phi(x_2)$
$\tilde{\phi}_3(x_1, x_2, x_3) := \phi(x_3)$ etc.

ϕ is our original function of a scalar var.

Element-wise functions & derivatives

Note then that the derivative of our elementwise function of a vector is

$$D_x \tilde{\phi}(x) = \frac{\partial \tilde{\phi}(x)}{\partial x^T}$$

$$= \begin{bmatrix} \frac{\partial \tilde{\phi}_1}{\partial x_1} & \frac{\partial \tilde{\phi}_1}{\partial x_2} & \frac{\partial \tilde{\phi}_1}{\partial x_3} \\ \frac{\partial \tilde{\phi}_2}{\partial x_1} & \frac{\partial \tilde{\phi}_2}{\partial x_2} & \frac{\partial \tilde{\phi}_2}{\partial x_3} \\ \frac{\partial \tilde{\phi}_3}{\partial x_1} & \frac{\partial \tilde{\phi}_3}{\partial x_2} & \frac{\partial \tilde{\phi}_3}{\partial x_3} \end{bmatrix}$$

$$= \begin{bmatrix} \phi'(x_1) & 0 & 0 \\ 0 & \phi'(x_2) & 0 \\ 0 & 0 & \phi'(x_3) \end{bmatrix}$$

Scalar derivative, ie $\frac{d\phi}{dz}$ of $\phi(z)$

Define $\phi'(x) = \begin{bmatrix} \phi'(x_1) \\ \phi'(x_2) \\ \phi'(x_3) \end{bmatrix}$ [vector arg.

scalar deriv

Then $\boxed{D_x \tilde{\phi}(x) = \text{diag}(\phi'(x))}$ & $\boxed{d\tilde{\phi}(x; dx) = \text{diag}(\phi'(x))\, dx}$

{ diag(x) = $\begin{bmatrix} x_1 & 0 & 0 \\ 0 & x_2 & 0 \\ 0 & 0 & x_3 \end{bmatrix}$ etc }

[usually just write $\phi(x)$ cf. $\tilde{\phi}(x)$]

Functions of a matrix ?

Element-wise function of a matrix A $(m \times n)$

$$\tilde{f}(A) := \begin{bmatrix} f(A_{11}) & f(A_{12}) & \cdots \\ \vdots & & \\ f(A_{m1}) & \cdots & f(A_{mn}) \end{bmatrix}$$

$$D_A \tilde{f}(A) = \frac{\partial \mathrm{vec}(\tilde{f}(A))}{\partial \mathrm{vec}(A)^T} \quad \left\} \text{ dim : } mn \times mn \right.$$

$$= \frac{\partial (\mathrm{vec}(\tilde{f}(A)))}{\partial \mathrm{vec}(A)^T}$$

$$= \mathrm{diag}\left(\mathrm{vec}(f'(A))\right)$$

where $f'(A)$ } matrix $\xrightarrow{\text{vec}}$ vector

- $f'(A)$: applied elementwise to A
- $f'(z) = \frac{df(z)}{dz}$: scalar

[$\mathrm{diag}(mn) \to mn \times mn$]

define: vectorised form of elementwise function of matrix input:

$$\boxed{\tilde{f}(\mathrm{vec}(A)) = \mathrm{vec}(\tilde{f}(A))}$$

$$= \begin{bmatrix} f(A_{11}) \\ f(A_{21}) \\ \vdots \\ f(A_{12}) \\ \vdots \\ f(A_{mn}) \end{bmatrix}$$

$$\boxed{= \tilde{f}(\mathrm{vec}(A))}$$ def for vector input

for element-wise f !

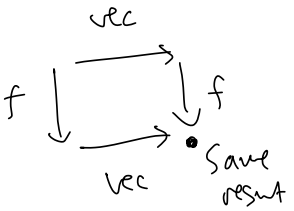
usually write $\tilde{f}(\mathrm{vec}(A))$

as $\boxed{f(\mathrm{vec}(A))}$

so $\boxed{\mathrm{vec}(f(A)) = f(\mathrm{vec}(A))} \neq$

for elementwise f !

[commutes with vec]



So, for element-wise, have:

$$d\tilde{f}(A) = d f(\mathrm{vec}(A)) = d\,\mathrm{vec}(f(A))$$

$$= \boxed{\mathrm{diag}\left(f'(\mathrm{vec}(A))\right) d\,\mathrm{vec}\,A}$$

$f'(\mathrm{vec}(A))$: scalar derivative; applied element-wise to $\mathrm{vec}(A)$

$\mathrm{vec}(A)$: vectorised; $f'(A)$: applied element-wise to A

& $D_A \tilde{f}(A) = \mathrm{diag}\left(\mathrm{vec}\left(f'(A)\right)\right)$

$= \mathrm{diag}\left(f'(\mathrm{vec}(A))\right)$

$f'(\mathrm{vec}(A))$: scalar derivative; applied element-wise to $\mathrm{vec}\,A$

$\bar{w}^o : [\bar{P}]$, $\bar{w}^h : = [\ P\ ]$

So... $y = g\left(\overbrace{\bar{H}(\bar{w}^h)}^{\text{vector}}\,\bar{w}^o\right) = g\left([1\ H]\,\bar{w}^o\right)$

$$\Rightarrow dy = \mathrm{diag}\left(g'\left(\bar{H}(\bar{w}^h)\,\bar{w}^o\right)\right)\left[d\bar{H}\,\bar{w}^o + \bar{H}\,d\bar{w}^o\right]$$

$$= \begin{cases} \mathrm{diag}\left(g'\left(\bar{H}(\bar{w}^h)\,\bar{w}^o\right)\right) d\bar{H}\,\bar{w}^o \\ + \\ \underbrace{\mathrm{diag}\left(g'\left(\bar{H}(\bar{w}^h)\,\bar{w}^o\right)\right)\bar{H}\,d\bar{w}^o}_{D_{\bar{w}^o} y} \end{cases} \quad ? \to D_{\bar{w}^h} y \ ?$$

$d\bar{w}^o = d\,\mathrm{vec}\,\bar{w}^o$ as $\bar{w}^o = \mathrm{vec}(\bar{w}^o)$

$\bar{H} : m\ \bar{P}$

Now:

$$\mathrm{vec}\left(\mathrm{diag}\left(g'\left(\bar{H}(\bar{w}^h)\bar{w}^o\right)\right) d\bar{H}\,\bar{w}^o\right)$$

$$= \mathrm{diag}\left(g'\left(\bar{H}(\bar{w}^h)\bar{w}^o\right)\right) d\bar{H}\,\bar{w}^o$$

[as already (m-dim) vector]

Also $d\bar{H} = [0 \quad dH]$, $\bar{w}^o = [w_0^o \ \ w^o]^T$

$$\Rightarrow d\bar{H}\,\bar{w}^o = dH\,w^o \qquad \Rightarrow$$

$\bar{H} = m\,[\ \bar{P}\ ]$

$\bar{w}^o = \bar{P}[\ ]$

$H = \bar{X}\bar{W}^h$

$\bar{W}^h = [\ ^p\ ]$

$\Rightarrow \operatorname{diag}\left(g'(\bar{H}(\bar{W}^h)\bar{W}^o)\right) d\bar{H}\bar{W}^o$

$= \operatorname{diag}\left(g'(\bar{H}(\bar{W}^h)\bar{W}^o)\right) dH\, W^o$

$= \left[(W^o)^T \otimes \operatorname{diag}\left(g'(\bar{H}(\bar{W}^h)\bar{W}^o)\right)\right] d\operatorname{vec}(H)$

[by Kronecker vec rule].

Now:

$\left[(W^o)^T \otimes \operatorname{diag}\left(g'(\bar{H}(\bar{W}^h)\bar{W}^o)\right)\right] d\operatorname{vec}(H)$

$d\operatorname{vec}(H) = d\operatorname{vec}\left(f(\bar{X}\bar{W}^h)\right)$

$= \operatorname{diag}\left(f'(\operatorname{vec}(\bar{X}\bar{W}^h)\right) d(\operatorname{vec}(\bar{X}\bar{W}^h))$

& $\operatorname{vec}(\bar{X}\bar{W}^h) = \operatorname{vec}(\bar{X}\bar{W}^h I_p)$

$= (I_p \otimes \bar{X})\, d\operatorname{vec}(\bar{W}^h)$

so $d\operatorname{vec}(H) =$

$\operatorname{diag}\left(f'(\operatorname{vec}(\bar{X}\bar{W}^h)\right)(I_p \otimes \bar{X})\, d\operatorname{vec}(\bar{W}^h)$ $\rightarrow$

So,

$\operatorname{diag}\left(g'(\bar{H}(\bar{W}^h)\bar{W}^o)\right) d\bar{H}\bar{W}^o$

$=$

$\left[(W^o)^T \otimes \operatorname{diag}\left(g'(\bar{H}(\bar{W}^h)\bar{W}^o)\right)\right] d\operatorname{vec}(H)$

$=$

$\left\{\left[(W^o)^T \otimes \operatorname{diag}\left(g'(\bar{H}(\bar{W}^h)\bar{W}^o)\right)\right]\operatorname{diag}\left(f'(\operatorname{vec}(\bar{X}\bar{W}^h)\right)(I_p \otimes \bar{X})\right\} \cdot d\operatorname{vec}(\bar{W}^h)$

$\Rightarrow$

$D_{\bar{W}^h}(y) = \left[(W^o)^T \otimes \operatorname{diag}\left(g'(\bar{H}(\bar{W}^h)\bar{W}^o)\right)\right]$ ---- (t times)

$\cdot \left[\operatorname{diag}\left(f'(\operatorname{vec}(\bar{X}\bar{W}^h)\right)(I_p \otimes \bar{X})\right]$

phew!

$\bar{X}$ $m \bar{n}$ $\bar{n}$ $\bar{p}$

sense check?

if $g(z) = z$
$g'(z) = 1$,

then eg

$$\text{diag}(g'(x)) = \begin{bmatrix} 1 & & \\ & 1 & \\ & & \ddots \\ & & & 1 \end{bmatrix} = I_{\text{len}(x)}$$

etc

So

$\bar{H}\bar{w}^0$

dim: $(m \times \bar{p})(\bar{p})$ $\}m$ $\Big\}$ $g'(H w^0) = \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} \}m$

$$\Rightarrow \text{diag}\left(g'(\bar{H}(\bar{w}^h)\bar{w}^0)\right) = I_m$$

$$\Rightarrow (w^0)^T \otimes \text{diag}\left(g'(\bar{H}(\bar{w}^h)\bar{w}^0)\right)$$

$$= (w^{0T}) \otimes I_m$$

Also if $f(z) = z$, $f'(z) = 1$

$$\Rightarrow f'(\overbrace{\text{vec}(\bar{X}\bar{w}^h)}^{mp}) = I_{mp}$$ } ones vector length $m \times p$

$$\Rightarrow \text{diag}\left(f'(\text{vec}(\bar{X}\bar{w}^h))\right)$$

$$= I_{mp}$$

---

so ...

$$D_{\bar{w}^h} = \left((w^0)^T \otimes I_m\right) I_{mp} \left(I_p \otimes \bar{X}\right)$$

$$= \left((w^0)^T \otimes I_m\right)\left(I_p \otimes \bar{X}\right)$$

$$= (w^0)^T I_p \otimes I_m \bar{X}$$

$$= (w^0)^T \otimes \bar{X} \quad \checkmark$$ (no activation function results).

Now we can compute

$$D_{\bar{W}} \ell(y(\bar{w}), d) \;!$$

& use (mini-batch) gradient descent

$$\boxed{\bar{W}(k+1) = \bar{W}(k) - \eta D_{\bar{W}} \ell}$$

to find weights!

First, note:

$$D_y \ell = (y-d)^T \quad \} (1 \times m)$$

$$D_{\bar{W}^0} y = \mathrm{diag}\left(g'\left(\bar{H}(\bar{W}^h)\bar{W}^0 \right) \right) \bar{H} \quad \} (m \times m)(m \times \bar{P})$$

$$D_{\bar{W}^h} = \Big[(W^0)^T \otimes \mathrm{diag}\left(g'(\bar{H}(\bar{W}^h)\bar{W}^0) \right) \Big] \text{..times..} \Big\} (m \times mp) \text{ times}$$

$$\Big[\mathrm{diag}\left(f'(\mathrm{vec}(\bar{X}\bar{W}^h)) \right) (\bar{I}_p \otimes \bar{X}) \Big] \Big\} (mp \times \bar{n}p) \text{ times}$$

Backpropagation & adjoint/reverse mode diff.

To compute the derivatives via the chain rule we end up needing to compute:

ABC or ABCD

for various size arrays (vectors & matrices)

→ while the final answer doesn't depend on the order we multiply these eg A(BC) or (AB)C, the computational cost does!

→ in general, want eg vector times matrix instead of matrix times matrix

Note eg (smaller)

$$D_{\bar{W}^0} \ell = \text{vector} \times \text{matrix} \times \text{matrix} \Big\} \text{vs} \; \text{vector} \times (\text{matrix} \times \text{matrix})$$

$$= \{ \text{vector} \times \text{matrix} \} \times \text{matrix}$$

vector × matrix } only vector × matrix ops.

⇒ If output is small, use 'backwards order'

→ "backpropagation"!

→ also called adjoint/reverse mode differentiation

Exercises (brave :)

- Implement what we derived (I might later ...)
- Compare to eg pytorch
  - ↳ does at 'lower' operation level!
  - ↳ automates for you!
- Read 'The matrix calculus you need for machine learning' by Parr & Howard (2018)

on Canvas &

'explained.ai/matrix-calculus/'

online

- other refs
  - Magnus & Neudecker (1999, 2019) 'Matrix differential calculus'
  - Goodfellow, Bengio, Courville (2016) 'Deep learning'

Physics-informed neural nets (PINNs)

→ Combine neural nets + eg differential eqn penalty } a la assignment! (NN instead of spline)

Eg:

→

updates

Scientific Machine Learning Through Physics–Informed Neural Networks: Where we are and What's Next

Salvatore Cuomo[1] · Vincenzo Schiano Di Cola[2] · Fabio Giampaolo[1] · Gianluigi Rozza[3] · Maziar Raissi[4] · Francesco Piccialli[1]

Received: 14 January 2022 / Revised: 6 July 2022 / Accepted: 6 July 2022 /
Published online: 26 July 2022

Abstract
Physics-Informed Neural Networks (PINN) are neural networks (NNs) that encode model equations, like Partial Differential Equations (PDE), as a component of the neural network itself. PINNs are nowadays used to solve PDEs, fractional equations, integral-differential equations, and stochastic PDEs. This novel methodology has arisen as a multi-task learning framework in which a NN must fit observed data while reducing a PDE residual. This article provides a comprehensive review of the literature on PINNs: while the primary goal of the study was to characterize these networks and their related advantages and disadvantages. The review also attempts to incorporate publications on a broader range of collocation-based physics informed neural networks, which stars form the vanilla PINN, as well as many other variants, such as physics-constrained neural networks (PCNN), variational hp-VPINN, and conservative PINN (CPINN). The study indicates that most research has focused on customizing the PINN through different activation functions, gradient optimization techniques, neural network structures, and loss function structures. Despite the wide range of applications for which PINNs have been used, by demonstrating their ability to be more feasible in some contexts than classical numerical techniques like Finite Element Method (FEM), advancements are still possible, most notably theoretical issues that remain unresolved.

Keywords Physics–Informed Neural Networks · Scientific Machine Learning · Deep Neural Networks · Nonlinear equations · Numerical methods · Partial Differential Equations · Uncertainty

→

Physics-informed neural networks (PINNs) for fluid mechanics: a review

Shengze Cai[1] · Zhiping Mao[2] · Zhicheng Wang[3] · Minglang Yin[4,5] · George Em Karniadakis[1,4]

Received: 21 May 2021 / Accepted: 10 September 2021 / Published online: 23 January 2022

Abstract
Despite the significant progress over the last 50 years in simulating flow problems using numerical discretization of the Navier–Stokes equations (NSE), we still cannot incorporate seamlessly noisy data into existing algorithms, mesh-generation is complex, and we cannot tackle high-dimensional problems governed by parametrized NSE. Moreover, solving *inverse flow problems* is often prohibitively expensive and requires complex and expensive formulations and new computer codes. Here, we review *flow physics-informed learning*, integrating seamlessly data and mathematical models, and implement them using physics-informed neural networks (PINNs). We demonstrate the effectiveness of PINNs for inverse problems related to three-dimensional wake flows, supersonic flows, and biomedical flows.

Keywords Physics-informed learning · PINNs · Inverse problems · Supersonic flows · Biomedical flows