Introduction to Deep Learning

Josh Lennon[1], Anirbit Mukherjee [2]

[1]Department of Computer Science, The University of Manchester, UK

*To whom correspondence should be addressed; E-mail joshlennon71@gmail.com:

Contents

1 Introduction **2**

2 Understanding a Basic Neural Regression Experiment **4**

A List of Some Notation **10**

B Rough Work **16**
B.1 Calculus exercises 16
B.2 Experimental Pseudocodes 17
B.3 Invalid work 19
B.4 Unnecessary Discussion 21
B.5 Matrices 25

1 Introduction

We choose n arbitrary training points $x_i \in \mathbb{R}$ indexed by i and labelled by the function $\mathbb{R} \ni x \mapsto f(x) = |x| \in [0, \infty)$.

We denote the training data thus obtained as,

$$\mathcal{S} \coloneqq \{(x_i, |x_i|) \mid i = 1, \ldots, n\}$$

Lets consider the neural network $\mathcal{N}$ with parameters w_1 and w_2 which is mapping as,

$$\mathbb{R} \ni x \mapsto \mathcal{N}(x) \coloneqq \max\{0, w_1 x\} + \max\{0, w_2 x\} \in \mathbb{R} \tag{1}$$

Now consider the ℓ_2-loss corresponding to the above training data and the predictor being the $\mathcal{N}$ above,

$$\mathcal{L}(\mathcal{S}, \mathcal{N}) \coloneqq \frac{1}{n} \cdot \sum_{i=1}^{n} (|x_i| - \mathcal{N}(x_i))^2 \in [0, \infty)$$

Writing the above explicitly in terms of the trainable weights of the neural net we have,

$$\mathcal{L}(w_1, w_2) \coloneqq \frac{1}{n} \cdot \sum_{i=1}^{n} (|x_i| - \max\{0, w_1 x_i\} - \max\{0, w_2 x_i\})^2$$

Its easy to observe that the above loss function as a global minima at $(w_1, w_2) = (1, -1)$ and also at $(w_1, w_2) = (-1, 1)$ since $\mathcal{L}(1, -1) = \mathcal{L}(-1, 1) = 0$. But note, that at both the above points the neural net function $\mathcal{N}$ as defined in equation 6 is the same!

Thus we have seen a simple example of how a neural loss function can have multiple global minima whilst all of them correspond to the same neural net function. In the studies we shall do going ahead it would be realized that this phenomenon is pervasive in deep-learning and occurs in increasingly more complicated forms.

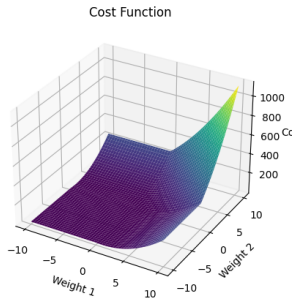


Figure 1: $n = 1$

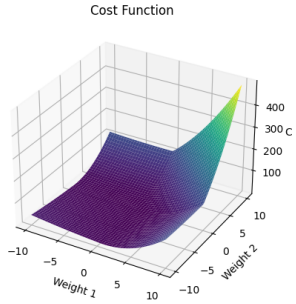


Figure 2: $n = 10$

When iterating over n values of training data scaling n in powers of 10, the following cost function plots are produced:

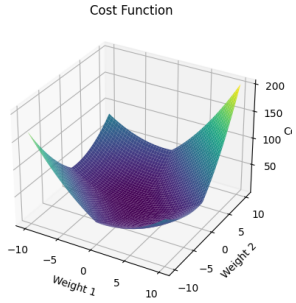


Figure 3: $n = 100$

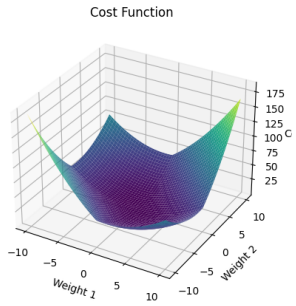


Figure 4: n = 1000

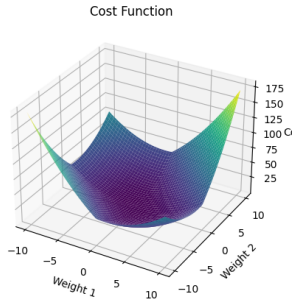


Figure 5: n = 10000

2 Understanding a Basic Neural Regression Experiment

Description of the training data. We generate two sets of n evenly spaced training points $x_i \in \mathbb{R}$ indexed by i,

$$\mathbb{X}_1 = \{x_{1i} \mid i = 1, \ldots, n\}$$

$$\mathbb{X}_2 = \{x_{2i} \mid i = 1, \ldots, n\}$$

where each value $x_1 i$ and $x_2 i \in \mathbb{X}_1$ and $\mathbb{X}_2$ respectively are evenly spaced generated in the following way,

$\left\{\beta + \dfrac{n-1}{\alpha-\beta} \cdot i \;\middle|\; i = 0, 1, 2, \ldots, n-1\right\}$ where α and β are equal to the upper and lower bound respectively.

When generating training data use α = 3, β = -3 and N_{Train} = 20 as the total number of training inputs where $n = N_{Train}$ such that x_{1i} and $x_{2i} \in [-3, 3]$. We can then define the matrix $\boldsymbol{X}_{Train} \in \mathbb{R}^{N^2_{Train} \times 2}$ of training data as the Cartesian product of the two sets $\mathbb{X}_1$ and $\mathbb{X}_2$, whose elements $\boldsymbol{X}_{ij}$ in the ith row and jth column contain each unique pairing of their elements:

$$\boldsymbol{X}_{Train} = \begin{bmatrix} x_{11} & x_{21} \\ x_{12} & x_{22} \\ \vdots & \vdots \\ x_{1N^2_{Train}} & x_{2N^2_{Train}} \end{bmatrix}$$

When generating the testing data, we generate evenly spaced data points in the same way this time using α = 5, β = -5 and N_{Test} = 10 where $n = N_{Test}$ such that x_{1i} and $x_{2i} \in [-5, 5]$ defining the test matrix containing every pair as $\boldsymbol{X}_{test} \in \mathbb{R}^{N^2_{Test} \times 2}$.

$$\boldsymbol{X}_{Test} = \begin{bmatrix} x_{11} & x_{21} \\ x_{12} & x_{22} \\ \vdots & \vdots \\ x_{1N^2_{Test}} & x_{2N^2_{Test}} \end{bmatrix}$$

Training points $\mathbf{x_{ij}} \in \boldsymbol{X}_{Train}$ indexed by ij are labelled by the function $\mathbb{R}^2 \ni (x_1, x_2) \mapsto f(x) = x_1^2 + x_2^2$ that maps x_{1i} and x_{2i} to obtain the label vector $\boldsymbol{y}_{Train}$,

$$\left(\boldsymbol{y}_{\text{Train}_i}\left(\boldsymbol{X}^2_{\text{Train}_{1,j}}, \boldsymbol{X}^2_{\text{Train}_{2,j}}\right)\right) \quad \forall j = 1, \ldots, N^2_{\text{Train}}$$

$$\boldsymbol{y}_{\text{Train}} = \begin{bmatrix} \left(X^2_{\text{Train}_{1,1}} + X^2_{\text{Train}_{2,1}}\right) \\ \left(X^2_{\text{Train}_{1,i}} + X^2_{\text{Train}_{2,i}}\right) \\ \vdots \\ \left(X^2_{\text{Train}_{1,N^2_{Train}}} + X^2_{\text{Train}_{2,N^2_{Train}}}\right) \end{bmatrix}$$

Test points $\mathbf{x_{ij}} \in \boldsymbol{X}_{Test}$ indexed by ij are also labelled by the function $\mathbb{R}^2 \ni (x_1, x_2) \mapsto f(x) = x_1^2 + x_2^2$ that maps x_{1i} and x_{2i} to obtain the label vector $\boldsymbol{y}_{Test}$,

$$\left(\boldsymbol{y}_{\text{Test}_i}\left(\boldsymbol{X}^2_{\text{Test}_{1,j}}, \boldsymbol{X}^2_{\text{Test}_{2,j}}\right)\right) \quad \forall j = 1, \ldots, N^2_{\text{Test}}$$

$$\boldsymbol{y}_{\text{Test}} = \begin{bmatrix} \left(X^2_{\text{Test}_{1,1}} + X^2_{\text{Test}_{2,1}}\right) \\ \left(X^2_{\text{Test}_{1,i}} + X^2_{\text{Test}_{2,i}}\right) \\ \vdots \\ \left(X^2_{\text{Test}_{1,N^2_{Test}}} + X^2_{\text{Test}_{2,N^2_{Test}}}\right) \end{bmatrix}$$

such that for $\boldsymbol{X}_{train}$ $f(x) = \mathbf{x_1^2} + \mathbf{x_2^2} \in [0, 18]$ and for $\boldsymbol{X}_{test}$ $f(x) = \mathbf{x_1^2} + \mathbf{x_2^2} \in [0, 50]$ due to the upper and lower bounds on the input data. Each label $\boldsymbol{y}_{\text{Train}_i}$ and $\boldsymbol{y}_{\text{Test}_i}$ is thus defined as $\mathbf{y_i} = x_{1i}^2 + x_{2i}^2$.

When selecting the value of n for training points and test points, we used N_{Train} = 20 and N_{Test} = 10 respectively. This number is important as it directly affect the quality of solutions output by the network. We can demonstrate this by observing changes in the cost function for different training values of n.

When generating data we used α = 3, β = -3 as the upper and lower bounds for training data and α = 5, β = -5 as the the upper and lower bounds for test data. The test data extends beyond values seen in training, the effect of which on the produced cost plot we will investigate later.

Description of the neural architecture to be used. We define the network function $\mathcal{N} : \mathbb{R}^2 \to \mathbb{R}$, where $\mathcal{A}_p$ represents an affine map function in the network and p denotes the layer index of the network. Each layer can be defined as follows:

$$\mathcal{A}_1 : \mathbb{R}^2 \to \mathbb{R}^{50}$$
$$\mathcal{A}_2 : \mathbb{R}^{50} \to \mathbb{R}^{50}$$
$$\mathcal{A}_3 : \mathbb{R}^{50} \to \mathbb{R}$$

The network $\mathcal{N}$,

$$\mathcal{N} : \mathbb{R}^{2\times 1} \to \mathbb{R}^{1\times 1}$$

Consider one row of the matrix $\boldsymbol{X}$ defined by $(\vec{x}) = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$ where each x_1 and x_2 are two corresponding input features. The network $\mathcal{N}$ can be defined explicitly in terms of each layer function as follows:

$$\mathcal{A}_1 : \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \mapsto \mathcal{A}_1(\boldsymbol{x}) = \max\{0, \boldsymbol{W}_1 \cdot \vec{x} + \boldsymbol{b}_1\}, \text{ where } \boldsymbol{W}_1 \in \mathbb{R}^{50\times 2}, \boldsymbol{b}_1 \in \mathbb{R}^{50}$$

$$\mathcal{A}_2 : \mathcal{A}_1(\vec{x}) \mapsto \mathcal{A}_2(\mathcal{A}_1(\vec{x})) = \max\{0, \mathbf{W}_2 \cdot \mathcal{A}_1(\vec{x}) + \vec{b}_2\}, \text{ where } \mathbf{W}_2 \in \mathbb{R}^{50\times 50}, \vec{b}_2 \in \mathbb{R}^{50}$$

$$\mathcal{A}_3 : \mathcal{A}_2(\mathcal{A}_1(\vec{x})) \mapsto \mathcal{A}_3(\mathcal{A}_2(\mathcal{A}_1(\vec{x}))) = \max\{0, \mathbf{W}_3 \cdot \mathcal{A}_2(\mathcal{A}_1(\vec{x})) + \vec{b}_3\}, \text{ where } \mathbf{W}_3 \in \mathbb{R}^{1\times 50}, \vec{b}_3 \in \mathbb{R}$$

Description of the loss function. The mean squared error (MSE) loss corresponding to the training data and the predictor being $\mathcal{N}$ is given by:

$$\mathcal{L}_{\text{Train}}(\mathcal{N}(\mathbf{X}_{\text{Train}}), \mathbf{y}_{\text{Train}}) = \frac{1}{n} \cdot \sum_{i=1}^{n} \left(\mathcal{N}(\mathbf{X}_{\text{Train}_{i,:}}) - \boldsymbol{y}_{\text{Train}_i}\right)^2 \in [0, \infty)$$

Substituting the explicit definition of the loss function and labels:

$$\mathcal{L}_{\text{Train}}(\mathcal{N}(\mathbf{X}_{\text{Train}}), \mathbf{y}_{\text{Train}}) = \frac{1}{n}\sum_{i=1}^{n}\left(\text{ReLU}\left(\max\left\{0, W_3 \cdot A_2(A_1(\mathbf{X}_{\text{Train}_{i,:}})) + \mathbf{b}_3\right\}\right) - (x_{1i}^2 + x_{2i}^2)\right)^2 \in [0, \infty)$$

The mean squared error (MSE) loss corresponding to the test data and the predictor being $\mathcal{N}$ is given by:

$$\mathcal{L}_{\text{Test}}(\mathcal{N}(\mathbf{X}_{\text{Test}}), \mathbf{y}_{\text{Test}}) = \frac{1}{n} \cdot \sum_{i=1}^{n}\left(\mathcal{N}(\mathbf{X}_{\text{Test}_{i,:}}) - \boldsymbol{y}_{\text{Test}_i}\right)^2 \in [0, \infty)$$

Substituting the explicit definition of the network and labels:

$$\mathcal{L}_{\text{Test}}(\mathcal{N}(\mathbf{X}_{\text{Test}}), \mathbf{y}_{\text{Test}}) = \frac{1}{n}\sum_{i=1}^{n}\left(\text{ReLU}\left(\max\left\{0, W_3 \cdot A_2(A_1(\mathbf{X}_{\text{Test}_{i,:}})) + \mathbf{b}_3\right\}\right) - (x_{1i}^2 + x_{2i}^2)\right)^2 \in [0, \infty)$$

Description of optimisation for the network. We optimise this network using the RMSprop algorithm:

Algorithm 1 RMSprop Algorithm for Multiple Weights

Input: Loss function - Mean Squared Error (MSE)
Input: Initialize $s_0 = 0$
Input: Initialize w_0
Input: learning rate η, decay rate γ, small constant ϵ
for $t \leftarrow 1$ **to** T **do**
    Compute gradient $\mathbf{g}_t$ using current data
    $\mathbf{s}_t \leftarrow \gamma \mathbf{s}_{t-1} + (1-\gamma) \cdot (\mathbf{g}_t \odot \mathbf{g}_t)$
    $\mathbf{w}_t \leftarrow \mathbf{w}_{t-1} - \frac{\eta}{\sqrt{\mathbf{s}_t + \epsilon}} \cdot \mathbf{g}_t$
end for
minimizing a cost function by optimizing the parameters $\mathbf{x}$ using gradients

$\mathbf{w}_t \leftarrow \mathbf{w}_{t-1} - \frac{\eta}{\epsilon} \cdot \mathbf{g}_t$

$\triangleright$ where s represents the moving average of squared gradients for each weight
$\triangleright$ where w_0 are the parameters to be optimized, a vector of weights (add into algorithm)
$\triangleright$ Where T represents the total number of iterations
$\triangleright$ $\mathbf{s}_t \leftarrow \gamma \mathbf{s}_{t-1} + (1-\gamma) \odot (\mathbf{g}_t \odot \mathbf{g}_t)$ is the element-wise update of the exponentially weighted moving average of squared gradients
$\triangleright$ $\odot$
is used to represent element-wise multiplication where each element in vector gt is multiplied by the corresponding element in itself to produce a vector $\triangleright$ $\mathbf{w}_t \leftarrow \mathbf{w}_{t-1} - \frac{\eta}{\sqrt{\mathbf{s}_t + \epsilon}} \odot \mathbf{g}_t$ adjusting the parameters based on the gradients and the moving average of squared gradients

Lets explore experimentally how the rate of decay affects RMSprop optimisation. What if we set the decay rate $\gamma = 1$ for our RMSprop algorithm? Let's start by substituting γ for 1 in s_t, where s_t represents the moving average of squared gradients for each weight:

$$\begin{aligned}\text{For } s_t \text{ with } \gamma = 1: \\ &= (1-\gamma)\cdot g_t^2 + \gamma \cdot s_{t-1} \\ &= (1-1)\cdot g_t^2 + 1 \cdot s_{t-1} \\ &= s_{t-1}\end{aligned}$$

The resulting formula $s_t = s_{t-1}$ no longer accounts for the gradient at t. This means there is no history of gradients and where s_0 is initialised with $s_0 = 0$, will prevent RMSprop from taking place. The denominator $\sqrt{s_t} + \epsilon$ becomes ϵ, causing the effective learning rate $\frac{\eta}{\sqrt{s_t}+\epsilon}$ to become extremely large, which will make convergence in optimization difficult.

$$\begin{aligned}\text{Alternatively For } s_t \text{ with } \gamma = 0: \\ &= (1-\gamma)\cdot g_t^2 + \gamma \cdot s_{t-1} \\ &= (1-0)\cdot g_t^2 + 0 \cdot s_{t-1} \\ &= g_t^2\end{aligned}$$

The resulting formula $s_t = g_t^2$ does not account for previous values s_{t-1} and thus the value of s_t is entirely dependant on the current gradient at t. When previous gradients are unaccounted for in RMSprop optimisation, the resultant learning will be inefficient.

we can see that for $0 < \gamma < 1$ will affect how much previous values of g_t (gradients) will contribute to the current value of s_t. We can observe that as γ approaches 1 the influence of s_{t-1} becomes more significant:

$$s_t = (1-\gamma)\cdot g_t^2 + \gamma \cdot s_{t-1}$$

Taking the limit as γ approaches 1:

$$\lim_{\gamma \to 1} s_t = \lim_{\gamma \to 1}\left((1-\gamma)\cdot g_t^2 + \gamma \cdot s_{t-1}\right)$$

Using limit properties:

$$\lim_{\gamma \to 1} s_t = \lim_{\gamma \to 1}(1-\gamma)\cdot g_t^2 + \lim_{\gamma \to 1}\gamma \cdot s_{t-1}$$

As γ tends to 1, the limit of $(1-\gamma)$ approaches 0:

$$\lim_{\gamma \to 1}(1 - \gamma) = 0$$

So, the expression becomes:

$$\lim_{\gamma \to 1} s_t = 0 \cdot g_t^2 + \lim_{\gamma \to 1} \gamma \cdot s_{t-1}$$

Now, applying the limit to $\gamma \cdot s_{t-1}$:

$$\lim_{\gamma \to 1} \gamma \cdot s_{t-1} = 1 \cdot s_{t-1} = s_{t-1}$$

So, the expression becomes:

$$\lim_{\gamma \to 1} s_t = 0 \cdot g_t^2 + s_{t-1}$$

Therefore:

$$\lim_{\gamma \to 1} s_t = s_{t-1}$$

This demonstrates that as γ tends to 1, the value of s_t becomes more reliant on s_{t-1} and less on the current gradient squared (g_t^2).

We've established how the parameter γ controls the decay or forgetting factor of past squared gradients.

A List of Some Notation

Theorem A.1. *The world would be a better place if ..*

Proof. (of Theorem A.1) ... □

$$\begin{aligned} x &= 2 + 2 \\ &= 3 + 1 \\ &= 4 \end{aligned} \tag{2}$$

$$\begin{aligned} y &= 4 + 4 \\ &= 7 + 1 \\ &= 8 \end{aligned} \tag{3}$$

$$\begin{aligned} z &= 6 + 6 \\ &= 11 + 1 \\ &= 12 \end{aligned} \tag{4}$$

Algorithm 2 Neuro-Tron (mini-batched, multi-gate, single-filter, stochastic algorithm)

Input: Sampling access to the marginal input distribution $\mathcal{D}_{\mathbf{x}}$
Input: Access to adversarial output $\mathbf{v} \in \mathbb{R}$ for any input $\mathbf{x} \in \mathbb{R}^n$
Input: Access to output $f_{\mathbf{w}}(\mathbf{x})$ of any $f_{\mathbf{w}} \in \mathcal{F}_{k,\alpha,\mathcal{A},\mathcal{W}}$ for any $\mathbf{w} \in \mathbb{R}^r$ and any input $\mathbf{x}$
Input: A sensing matrix $\boldsymbol{M} \in \mathbb{R}^{r \times n}$
Input: A starting point $\boldsymbol{w}_1$
Input: Number T of batches
Input: Batch size b
Input: Learning rate η
for $t = 1, \ldots,$ T **do**
    Sample batch $s_t \coloneqq (\mathbf{x}_{t_1}, \ldots, \mathbf{x}_{t_b})$, where $\mathbf{x}_{t_i} \sim \mathcal{D}_{\mathbf{x}},\ i = 1, \ldots, b$
    for $i = 1, \ldots, b$ **do**
        The oracle samples $\alpha_{t_i} \in \{0, 1\}$ with probability $\{1 - \beta(\mathbf{x}_{t_i}), \beta(\mathbf{x}_{t_i})\}$
        The oracle replies with $\mathbf{v}_{t_i} \coloneqq f_{\boldsymbol{w}^*}(\mathbf{x}_{t_i}) + \alpha_{t_i}\xi_{t_i}$
    end for
    Form the so-called Tron-gradient,

$$\mathbf{g}^{(t)} \coloneqq \boldsymbol{M}\left(\frac{1}{b}\sum_{i=1}^{b}\left(\left(\mathbf{v}_{t_i} - f_{\mathbf{w}^{(t)}}(\mathbf{x}_{t_i})\right)\mathbf{x}_{t_i}\right)\right)$$

    $\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} + \eta\mathbf{g}^{(t)}$
end for

The following come from the "physics" package except the inner-product

$\langle \boldsymbol{a}, \boldsymbol{b} \rangle, |c|, \|\boldsymbol{d}\|, \frac{\partial L}{\partial \boldsymbol{w}}, \frac{\partial^2 L}{\partial \boldsymbol{w}^2}, \frac{\mathrm{d}W}{\mathrm{d}t}, \mathrm{d}x, \mathrm{rank}(\boldsymbol{M}), \mathrm{tr}(\boldsymbol{A}), \mathrm{Tr}(\boldsymbol{B}), \delta W$

ReLU is our definition
We have also defined the following calligraphic letters, $\mathcal{F}, \mathcal{R}, \mathcal{N}, \mathcal{E}, \mathcal{L}, \mathcal{D}$

This is the default highlight! This is our customized highlight!

Everything below comes from the TMLR format
- which comes from this book, https://github.com/goodfeli/dlbook_notation/

Sets and Graphs

$\mathbb{A}$	A set
$\mathbb{R}$	The set of real numbers
$\{0, 1\}$	The set containing 0 and 1
$\{0, 1, \dots, n\}$	The set of all integers between 0 and n
$[a, b]$	The real interval including a and b
$(a, b]$	The real interval excluding a but including b
$\mathbb{A} \backslash \mathbb{B}$	Set subtraction, i.e., the set containing the elements of $\mathbb{A}$ that are not in $\mathbb{B}$
$\mathcal{G}$	A graph

Numbers and Arrays

a	A scalar (integer or real)
$\boldsymbol{a}$	A vector
$\boldsymbol{A}$	A matrix
$\mathsf{\mathbf{A}}$	A tensor
$\boldsymbol{I}_n$	Identity matrix with n rows and n columns
$\boldsymbol{I}$	Identity matrix with dimensionality implied by context
$\boldsymbol{e}^{(i)}$	Standard basis vector $[0, \dots, 0, 1, 0, \dots, 0]$ with a 1 at position i
$\text{diag}(\boldsymbol{a})$	A square, diagonal matrix with diagonal entries given by $\boldsymbol{a}$
a	A scalar random variable
$\mathbf{a}$	A vector-valued random variable
$\mathbf{A}$	A matrix-valued random variable

Indexing

a_i	Element i of vector $\boldsymbol{a}$, with indexing starting at 1
a_{-i}	All elements of vector $\boldsymbol{a}$ except for element i
$A_{i,j}$	Element i, j of matrix $\boldsymbol{A}$
$\boldsymbol{A}_{i,:}$	Row i of matrix $\boldsymbol{A}$
$\boldsymbol{A}_{:,i}$	Column i of matrix $\boldsymbol{A}$
$\mathsf{A}_{i,j,k}$	Element (i, j, k) of a 3-D tensor A
$\mathsf{A}_{:,:,i}$	2-D slice of a 3-D tensor
a_i	Element i of the random vector $\mathbf{a}$

Probability and Information Theory

$P(\mathrm{a})$	A probability distribution over a discrete variable
$p(\mathrm{a})$	A probability distribution over a continuous variable, or over a variable whose type has not been specified
$\mathrm{a} \sim P$	Random variable a has distribution P
$\mathbb{E}_{\mathrm{x}\sim P}[f(x)]$ or $\mathbb{E}f(x)$	Expectation of $f(x)$ with respect to $P(\mathrm{x})$
$\mathrm{Var}(f(x))$	Variance of $f(x)$ under $P(\mathrm{x})$
$\mathrm{Cov}(f(x), g(x))$	Covariance of $f(x)$ and $g(x)$ under $P(\mathrm{x})$
$H(\mathrm{x})$	Shannon entropy of the random variable x
$D_{\mathrm{KL}}(P\|Q)$	Kullback-Leibler divergence of P and Q
$\mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma})$	Gaussian distribution over $\boldsymbol{x}$ with mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}$

Functions

$f : \mathbb{A} \to \mathbb{B}$	The function f with domain $\mathbb{A}$ and range $\mathbb{B}$
$f \circ g$	Composition of the functions f and g
$f(\boldsymbol{x}; \boldsymbol{\theta})$	A function of $\boldsymbol{x}$ parametrized by $\boldsymbol{\theta}$. (Sometimes we write $f(\boldsymbol{x})$ and omit the argument $\boldsymbol{\theta}$ to lighten notation)
$\log x$	Natural logarithm of x
$\sigma(x)$	Logistic sigmoid, $\frac{1}{1+\exp(-x)}$
$\zeta(x)$	Softplus, $\log(1 + \exp(x))$
$\|\boldsymbol{x}\|_p$	L^p norm of $\boldsymbol{x}$
$\|\boldsymbol{x}\|$	L^2 norm of $\boldsymbol{x}$
x^+	Positive part of x, i.e., $\max(0, x)$
$\mathbf{1}_{\text{condition}}$	is 1 if the condition is true, 0 otherwise
$\operatorname{argmax}_{\mathbf{x} \in \mathbb{R}^n} f$	
$\operatorname{argmin}_{\mathbf{x} \in \mathbb{R}^n} f$	

Acknowledgments

The authors thank Professor Anirbit Mukherjee for their guidance. The thanks also extend to colleagues for their collaboration and to the authors family for their unwavering encouragement.

Supplementary materials

Materials and Methods
Supplementary Text
Figs. S1 to S3
Tables S1 to S4
References *(4-10)*

B Rough Work

B.1 Calculus exercises

If $f(x, y)$ is a function of the two variables x and y, then the partial derivative $\frac{\partial}{\partial x} f(x, y)$ is defined as the derivative of the function $g(x) = f(x, y)$ with respect to x, where y is kept constant. The partial derivative with respect to y is the derivative with respect to y, where x is fixed. We can use the notation $f_x(x, y) = \frac{\partial}{\partial x} f(x, y)$. When iterating derivatives, the notation is similar: we write, for example, $f_{xy} = \frac{\partial}{\partial x} \frac{\partial}{\partial y} f$.

Iterating derivatives can be satisfied as such:

Claim: The function $f(x, y) = x^3 - 3xy^2$ satisfies the Laplace equation $f_{xx}(x, y) + f_{yy}(x, y) = 0$.

Proof: Finding the iterated derivatives and evaluating whether the difference equates to 0

$$\begin{aligned} f_x(x, y) &= 3x^2 - 3y^2 \\ f_y(x, y) &= -6xy \\ f_{xx}(x, y) &= 6x \\ f_{yy}(x, y) &= -6x \end{aligned}$$

Now, substituting these derivatives into the Laplace equation:

$$f_{xx}(x, y) + f_{yy}(x, y) = 6x - 6x = 0$$

Since the result is zero, the Laplace equation is satisfied for the function $f(x, y) = x^3 - 3xy^2$.

Therefore, the claim is true, and the function $f(x, y) = x^3 - 3xy^2$ satisfies the Laplace equation.

Claim: The wave equation $f_{tt}(t, x) = f_{xx}(t, x)$ governs the motion of light or sound. The function $f(t, x) = \sin(x - t) + \sin(x + t)$ satisfies the wave equation.

Proof: Finding the iterated derivatives and evaluating whether the difference equates to 0

$$\begin{aligned} f_t(t, x) &= -\cos(x - t) + \cos(x + t), \\ f_{tt}(t, x) &= \sin(x - t) - \sin(x + t), \\ f_x(t, x) &= \cos(x - t) + \cos(x + t), \\ f_{xx}(t, x) &= -\sin(x - t) - \sin(x + t). \end{aligned}$$

Now, substituting these derivatives into the wave equation:

$$\begin{aligned} f_{tt}(t, x) - f_{xx}(t, x) &= (\sin(x - t) - \sin(x + t)) - (-\sin(x - t) - \sin(x + t)) \\ &= \sin(x - t) - \sin(x + t) + \sin(x - t) + \sin(x + t) \\ &= 2\sin(x - t) \neq 0. \end{aligned}$$

The wave equation is not satisfied, so the claim is false.

B.2 Experimental Pseudocodes

This first pseudocode is for visualising the cost function of the simple neural net for the function $|x|$.

Algorithm 3 Graphing Cost Function

Input: Random data list $data \in \mathbb{R}$ and corresponding labels $labels = |data|$
Input: Number of data values: n
Input: Range for weight values: $w1_{\text{min}}, w1_{\text{max}}, w2_{\text{min}}, w2_{\text{max}}$
Input: Number of cost values to calculate for each weight: c_{points}
Output: 3D plot of the cost function
Initialize an empty matrix $cost_values$ with dimensions $c_{\text{points}} \times c_{\text{points}}$
for $i = 1$ to c_{points} **do**
    for $j = 1$ to c_{points} **do**
        Calculate cost value for $w1 = w1_{\text{min}} + i \cdot \frac{w1_{\text{max}} - w1_{\text{min}}}{c_{\text{points}}}$ and $w2 = w2_{\text{min}} + j \cdot \frac{w2_{\text{max}} - w2_{\text{min}}}{c_{\text{points}}}$ using the cost function:
        $predictions = \max(0, w1 \cdot data) + \max(0, w2 \cdot data)$
        $error = labels - predictions$
        $cost_values[i][j] = \frac{1}{2} \cdot \text{mean}(error^2)$
    end for
end for
Create a 3D plot with the $w1$ values, $w2$ values, and $cost_values$ using matplotlib

Here is pseudo code for the second, more complex network visualising cost function experiment:

Algorithm 4 Testing and Visualization

Input: Testing data $test\ input$, Trained model $model\ depth\ 3$
Output: 3D plot of true and predicted paraboloids
$test\ prediction = model\ depth\ 3(test\ input)$
Create 3D plot
Plot true curve:
$plot(test\ input, test\ labels)$
Plot learned paraboloid:
$plot(test\ input, test\ prediction)$

Here is pseudo code for the same algorithm, but a more code based representation of training the model:

the general MSE loss function for this network looks like this (bad notation):

$$\mathcal{L}(\mathbf{y}_{\text{pred}}, \mathbf{y}_{\text{true}}) = \frac{1}{n} \cdot \sum_{i=1}^{n} (\mathbf{y}_{\text{pred}}[i] - \mathbf{y}_{\text{true}}[i])^2 \in [0, \infty)$$

Next is exploring how the experiment from before is optimised using RMSprop. Here is how the algorithm for optimisation works starting with a more general pseudocode and working to a pseudocode more specific to this experiment:

Algorithm 5 Training Neural Network on Data

Input: Training data $train\ input$, Training labels $train\ labels$
Input: Neural Network Model $model\ depth$ 3 with parameters, Loss Function $loss\ fn$
Input: Optimiser $optimiser$
Output: Trained model parameters
for t = 1 to 2×10^4 **do**
    $optimizer.zero\ gradient()$
    $y\ prediction$ = $model\ depth\ 3(train\ input)$
    $loss$ = $loss\ function(y\ prediction, train\ labels)$
    $loss.backward()$
    $optimiser.step()$
end for

Algorithm 6 RMSprop Algorithm

Input: Initialize $s_0 = 0$ (where s represents the moving average of squared gradients)
Input: Initialize x_0 (the parameter to be optimised)
Input: learning rate η, decay rate γ, small constant ϵ
for $t \leftarrow 1$ **to** T **do** (Where T represents the total number of iterations)
    Compute gradient g_t using current data
    $s_t \leftarrow \gamma s_{t-1} + (1-\gamma) g_t^2$ (exponentially weighted moving average of squared gradients is updated)
    $x_t \leftarrow x_{t-1} - \frac{\eta}{\sqrt{s_t+\epsilon}} g_t$ (adjusting the parameter based on the gradient and the moving average of squared gradients)
    Update parameters: $x_{t-1} \leftarrow x_t$
end for
minimizing a cost function by optimising the parameter x using gradients

Algorithm 7 RMSprop Algorithm for Multiple Weights

Input: Initialize $s_0 = 0$ (where s represents the moving average of squared gradients for each weight)
Input: Initialize x_0 (the parameters to be optimised, a vector of weights)
Input: learning rate η, decay rate γ, small constant ϵ
for $t \leftarrow 1$ **to** T **do** (Where T represents the total number of iterations)
    Compute gradient $\mathbf{g}_t$ using current data
    $\mathbf{s}_t \leftarrow \gamma \mathbf{s}_{t-1} + (1-\gamma) \odot (\mathbf{g}_t \odot \mathbf{g}_t)$ (element-wise update of exponentially weighted moving average of squared gradients)
    $\mathbf{x}_t \leftarrow \mathbf{x}_{t-1} - \frac{\eta}{\sqrt{\mathbf{s}_t+\epsilon}} \odot \mathbf{g}_t$ (adjusting the parameters based on the gradients and the moving average of squared gradients)
end for
minimizing a cost function by optimising the parameters $\mathbf{x}$ using gradients

B.3 Invalid work

The loss function quantifies the discrepancy between the absolute values of the training data and the predictions made by the neural network $\mathcal{N}$. Consider again the loss function,

$$\mathcal{L}(w_1, w_2) \coloneqq \frac{1}{n} \cdot \sum_{i=1}^{n} (|x_i| - \max\{0, w_1 x_i\} - \max\{0, w_2 x_i\})^2$$

when $\forall x_i \in \mathbb{R}, x_i \geq 0$ or $\forall x_i \in \mathbb{R}, x_i < 0$ we can observe that for $x_i w_1$ or $x_i w_2$ being certain values there is a constant region in the plot of the loss function. When:

$$\text{for } \{x_i \mid \forall x_i \in \mathbb{R}, \quad x_i \geq 0 \quad \text{where } w_1 \text{ and } w_2 < 0\} :$$

The loss function evaluates to:

$$\begin{aligned} \mathcal{L}(w_1, w_2) &\coloneqq \frac{1}{n} \cdot \sum_{i=1}^{n} (|x_i| - 0 - 0)^2 \\ &= \frac{1}{n} \cdot \sum_{i=1}^{n} |x_i|^2 \end{aligned}$$

It is clear for these values the loss function only depends on the value of x_i. And when:

$$\text{for } \{x_i \mid \forall x_i \in \mathbb{R}, \quad x_i \leq 0 \quad \text{where } w_1 \text{ and } w_2 < 0\} :$$

The loss function identically evaluates to:

$$\begin{aligned} \mathcal{L}(w_1, w_2) &\coloneqq \frac{1}{n} \cdot \sum_{i=1}^{n} (|x_i| - 0 - 0)^2 \\ &= \frac{1}{n} \cdot \sum_{i=1}^{n} |x_i|^2 \end{aligned}$$

And again it is clear for these values the loss function only depends on the value of x_i. We can thus observe that for certain values of $w_1 x_i$ and $w_2 x_i$ the graph of the loss function will produce a constant region. This demonstrates that the nature of the ReLU activation function introduces functions that can have flat spaces and as we will discover as we continue to explore the loss function, that this is something that is problematic in the process of applying gradient descent to obtain updated weights w_1' and w_2'.

Now consider the following partial derivatives with respect to w_1 and w_2 for $\mathcal{L}(w_1, w_2)$,

$$\frac{\partial}{\partial w_1} L = -\frac{n}{2} \sum_{i=1}^{n} \left(|x_i| - \max\{0, w_1 x_i\} - \max\{0, w_2 x_i\} \right) \cdot x_i \cdot (x_i w_1 + x_i w_2 > 0)$$
$$\frac{\partial}{\partial w_2} L = -\frac{n}{2} \sum_{i=1}^{n} \left(|x_i| - \max\{0, w_1 x_i\} - \max\{0, w_2 x_i\} \right) \cdot x_i \cdot (x_i w_1 + x_i w_2 > 0)$$

Next lets evaluate the derivatives at $(0, 0)$ using our data points x_i and corresponding labels $|x_i|$:

For $\frac{\partial w_1}{\partial L}$ with $w_1 = 0$:

$$\begin{aligned} \frac{\partial}{\partial w_1} L \Big|_{w_1=0} &= -2 \sum_{i=1}^{n} \left(|x_i| - \max\{0, 0 \cdot x_i\} - \max\{0, w_2 x_i\} \right) \cdot x_i \cdot (0 + x_i w_2 > 0) \\ &= -\frac{n}{2} \sum_{i=1}^{n} \left(|x_i| - \max\{0, w_2 x_i\} \right) \cdot 0 \cdot 0 \\ &= 0 \text{ for all } i. \end{aligned}$$

Similarly, for $\frac{\partial w_2}{\partial L}$ with $w_2 = 0$:

$$\begin{aligned} \frac{\partial}{\partial w_2} L \Big|_{w_2=0} &= -2 \sum_{i=1}^{n} \left(|x_i| - \max\{0, 0 \cdot x_i\} - \max\{0, 0 \cdot x_i\} \right) \cdot x_i \cdot (x_i \cdot 0 + 0 > 0) \\ &= -\frac{n}{2} \sum_{i=1}^{n} \left(|x_i| - 0 \right) \cdot 0 \cdot 0 \\ &= 0 \text{ for all } i. \end{aligned}$$

So, when you substitute $w_1 = 0$ and $w_2 = 0$, both $\frac{\partial \mathcal{L}}{\partial w_1}$ and $\frac{\partial \mathcal{L}}{\partial w_2}$ become 0 for all data points i demonstrating that the gradient is flat at this point.

The flat gradient at $(0, 0)$ suggests that the cost function is constant in the vicinity of this point, which implies a linear region in the cost function surface around $w_1 = 0$ and $w_2 = 0$.

ReLU introduces non-linearity, but it is a piecewise linear function, which can create flat regions in the cost function surface

When both w1 and w2 are zero or close to zero (within the flat region of the ReLU activation), the predictions are close to zero for all data points. Consequently, the error is close to zero, leading to a flat region in the cost function

When the number of training values is increased we observe that the 3d plot no longer produces a linear region however a local maxima forms at $(0, 0)$

It appears that the graph is variant with n which in our further studies we will see is a reoccurring behaviour in complicated networks.

As you increase the amount of data (e.g., from 10 to 30 data points), there's a higher likelihood that some of the data points will be within the flat region of the ReLU activation for a given combination of w1 and w2. When you have more data points, the cost function is less likely to have a flat region because it must account for the diverse data patterns

B.4 Unnecessary Discussion

Consider a more general version of network network N(x) where N(x) includes biases that maps a scalar input x to a scalar output y and has eight parameters

$$\phi = \phi_0, \phi_1, \phi_2, \phi_3, \phi_{10}, \phi_{11}, \phi_{20}, \phi_{21}$$

Where:

$$\mathcal{N}(x) := \phi_0 + \phi_1(max(0, \phi_{10}) + \phi_{11}x) + \phi_2(max(0, \phi_{20} + \phi_{21}x))$$

this can be broken down into 2 parts:

part 1 calculating each hidden neuron:

$$h_1 = \max\{0, [\phi_{10} + \phi_{11}x]\}$$

$$h_2 = \max\{0, [\phi_{20} + \phi_{21}x]\}$$

where each of h_1, h_2 are in the form y=mx+b and clipped below 0 by the ReLU activation function and such that:

$$\mathcal{N}(x) = p_0 + p_1h_1 + p_2h_2 + p_3h_3.$$

Generalising further where d is the number of hidden layers in the network:

$$h_d = \max\{0, [p_{d0} + p_{d1}x])\}$$

Rewriting $\mathcal{N}(x)$:

$$\mathcal{N}(x) = \phi_0 + \sum_{d=1}^{D} \phi_d h_d$$

With ReLU activation functions,it is thus clear to see that the output of a network with D hidden units has at most D joints and so is a piecewise linear function with at most D + 1 linear regions.

We can use the universal approximation theorem to prove that for any continuous function, there exists a shallow network that can approximate this function to any specified precision.

Thus we can see that if you continue to increase the value of D and add hidden units you will create more linear piecewise regions, meaning with enough units any continuous 1D function defined on a compact subset of the real line to arbitrary precision.

by adding more neurons, the network can get arbitrarily close to any continuous function within a given precision (ε).

To extend the network to multivariate outputs, we simply use a different linear function of the hidden units for each output.

So, the same network $\mathcal{N}(x)$ has a 2D multivariate output $N : x \mapsto (y_1, y_2)$

and is defined as,

$$h_1 = \max\{0, [\phi_{10} + \phi_{11}x]\}$$
$$h_2 = \max\{0, [\phi_{20} + \phi_{21}x]\}$$

with the same hidden units as before and,

$$y_1 = \phi_{10} + \phi_{11}h_1 + \phi_{12}h_2$$
$$y_2 = \phi_{20} + \phi_{21}h_1 + \phi_{22}h_2$$

Consider the network extended to two inputs $x = [x_1, x_2]$ with only one output. The hidden units will be defined in the same way as the outputs from 2 hidden layers, such that each hidden unit is weighted by both inputs:

$$h_1 = a[\phi_{10} + \phi_{11}x_1 + \phi_{12}x_2]$$
$$h_2 = a[\phi_{20} + \phi_{21}x_1 + \phi_{22}x_2]$$

Simplified where a is the activation function:

$$a[z] = \text{ReLU}[z] = \begin{cases} 0 & \text{if } z < 0 \\ z & \text{if } z \geq 0 \end{cases}$$

The final output where there is now one slope parameter for each input can be calculated,

$$y = \phi_0 + \phi_1 h_1 + \phi_2 h_2$$

.

Completing the multi-dimensional input network with a 2 dimensional input.

Using these cases we can see that it is easy to create a general case for the network $\mathcal{N}(x)$ that maps a multi-dimensional input to a multi-dimensional output,

$\mathbf{x} \in \mathbb{R}^{D_i} \mapsto \mathcal{N}(x) = \mathbf{y} \in \mathbb{R}^{D_o}$ using $h \in \mathbb{R}^D$ hidden units.

Where o is the number of outputs, i is the number of inputs and d is the number of hidden units in the network.

Each hidden unit is computed as:

$$h_d = a\left(\phi_{d0} + \sum_{i=1}^{D_i} \phi_{di} x_i\right)$$

and following on as in the previous network building methods we can apply this notation to create a general formula for the hidden layers in the network by combining them linearly,

$$\mathbf{x} \in \mathbb{R}^{D_i} \mapsto \mathcal{N}(x) = \phi_{o0} + \sum_{d=1}^{D} \phi_{od} h_d \in \mathbb{R}^{D_o}$$

I have demonstrated how shallow neural networks make predictions y based on inputs x by dividing the input space into a continuous surface of piecewise linear regions.

We can now see that the universal approximation theorem also holds for the more general case.

This means With enough hidden units (neurons), shallow neural networks can approximate any continuous function to arbitrary precision.

Consider two new neural networks,

$$\mathbb{R} \in x \mapsto \mathcal{N}(x) \coloneqq \phi_0 + max(0, [\phi_{10} + \phi_{11} x]) + \max\{0, [\phi_{20} + \phi_{21} x]\} \in \mathbb{R} \tag{5}$$

Where the output of the first is the input to the second,

$$\mathbb{R} \in x \mapsto \mathcal{N}'(x) \coloneqq \phi'_0 + max(0, [\phi'_{10} + \phi'_{11} x]) + \max\{0, [\phi'_{20} + \phi'_{21} x]\} \in \mathbb{R} \tag{6}$$

We now show that this is a special case of a deep network with two hidden layers.

Consider the substitution,

$$\begin{aligned} h'_1 &= a\left[\phi'_{10} + \phi'_{11} y\right] \\ &= a\left[\phi'_{10} + \phi'_{11}\phi_0 + \phi'_{11}\phi_1 h_1 + \phi'_{11}\phi_2 h_2\right] \\ h'_2 &= a\left[\phi'_{20} + \phi'_{21} y\right] \\ &= a\left[\phi'_{20} + \phi'_{21}\phi_0 + \phi'_{21}\phi_1 h_1 + \phi'_{21}\phi_2 h_2\right] \end{aligned}$$

Which we can rewrite as

$$\begin{aligned} h'_1 &= a\left[\phi_{10} + \phi_{11} h_1 + \phi_{12} h_2 + \phi_{13} h_3\right] \\ h'_2 &= a\left[\phi_{20} + \phi_{21} h_1 + \phi_{22} h_2 + \phi_{23} h_3\right] \end{aligned}$$

The result is a network with two hidden layers

Where

$$\begin{aligned} \phi_{10} &= \theta'_{10} + \theta'_{11}\phi_0 \\ \phi_{11} &= \theta'_{11}\phi_1 \\ \phi_{12} &= \theta'_{11}\phi_2 \end{aligned}$$

and so on.

B.5 Matrices

$$\boldsymbol{y}_{Train} = \begin{bmatrix} X^2_{\text{Train}_{1,i}} + X^2_{\text{Train}_{2,i}} \\ x^2_{1i} + x^2_{2i} \\ \vdots \\ x^2_{1n^2} + x^2_{2n^2} \end{bmatrix}$$