University of
St Andrews

Freeing Neural Training Through Surfing

Georg Wölflein
170011885
Supervisor: Dr. Mike Weir

April 9, 2020

Word count: 2706 words

Abstract

TODO

Declaration

Georg Wölflein

Contents

Chapter 1

Introduction

Describe the problem you set out to solve and the extent of your success in solving it. You should include the aims and objectives of the project in order of importance and try to outline key aspects of your project for the reader to look for in the rest of your report. **TODO**

Chapter 2

Context survey

2.1 Neural networks

TODO

2.2 Implementation tools

- TensorFlow

- keras

TODO

Chapter 3

Requirements specification

Primary objectives:

1. Design a generic framework that can be used for various neural training algorithms with a clear set of inputs and outputs at each step. This framework should include benchmarking capabilities.

2. For a simple case of this framework (when the dimensionality of the control space and output space are suitably low), implement a visualisation tool that shows the algorithm's steps.

3. Implement a particular training algorithm for the framework that uses potential field techniques.

4. Evaluate the performance of this and other algorithms on tasks of differing complexity, especially with regard to the local minimum problem and similar issues.

Secondary objectives:

1. Investigate how this approach can be generalized to other numerical optimisation problems.

3.1   Ethics

There are no ethical considerations. All questions on the preliminary self-assessment form were answered with "NO" and hence no ethics form had to be completed.

Part I

Theory

Chapter 4

Neural network theory

4.1 Supervised learning

Definition 1 (Regression model). In machine learning, a regression model f is defined as a mathematical function of the form

$$f(\mathbf{x}) = \hat{y} = y + \epsilon \tag{4.1}$$

that models the relationship between a D-dimensional feature vector $\mathbf{x} \in \mathbb{R}^D$ of independent (*input*) variables and the dependent (*output*) variable $y \in \mathbb{R}$. Given a particular $\mathbf{x}$, the model will produce a *prediction* for y which we denote $\hat{y}$. Here, the additive error term ϵ represents the discrepancy between y and $\hat{y}$.

Definition 2 (Labelled dataset). A labelled dataset consists of N tuples of the form $\langle \mathbf{x}_i, y_i \rangle$ for $i = 1, \ldots, N$. For each feature vector $\mathbf{x}_i$ (a row vector), the corresponding y_i represents the observed output, or *label* [Burkov, 2019]. We use the vector

$$\mathbf{y} = \begin{bmatrix} y_1 & y_2 & \cdots & y_N \end{bmatrix}^\mathsf{T} \tag{4.2}$$

to denote all the labelled outputs in the dataset, and the $N \times D$ matrix

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1 & \mathbf{x}_2 & \cdots & \mathbf{x}_N \end{bmatrix}^\mathsf{T} \tag{4.3}$$

for representing the corresponding feature vectors.

Supervised learning A supervised learning algorithm for a regression task infers the function f given in (4.1) from a set of *labelled training data* of the form explained previously. We use the vector

$$\hat{\mathbf{y}} = \begin{bmatrix} \hat{y}_1 & \hat{y}_2 & \cdots & \hat{y}_N \end{bmatrix}^\mathsf{T} \tag{4.4}$$

to denote the prediction that f produces for each training sample.

4.2 Artifical neural networks

Artifical neural networks (ANNs) take inspiration from the human brain and can be regarded as a set of interconnected neurons. More formally, an ANN is a directed graph of n neurons (referred to as *nodes* or *units*) with weighted edges (*links*). Each link connecting two units i and j is directed and associated with a real-valued weight $w_{i,j}$.

A particular unit i's *excitation*, denoted z_i, is calculated as the weighted sum

$$z_i = \sum_{j=1}^{n} w_{j,i} a_j + b_i \tag{4.5}$$

where $a_j \in \mathbb{R}$ is another unit j's *activation* and $b_i \in \mathbb{R}$ is the ith unit's *bias*. Notice that in this model, if there exists no link between unit i and a particular j then simply $w_{i,j} = 0$ and therefore j will not contribute to i's excitation.

The unit i's activation is its excitation applied to a non-linear *activation function*, $g : \mathbb{R} \to \mathbb{R}$. We have

$$a_i = g(z_i) = g\left(\sum_{j=1}^{n} w_{j,i} a_j + b_i\right). \tag{4.6}$$

Activation functions In its original form, McCulloch and Pitts defined the neuron as having only binary activation [McCulloch and Pitts, 1943]. This means that in our model from (4.6), we would require $a_i \in \{0, 1\}$ and hence an activation function of the form $g_{\text{thres}} : \mathbb{R} \to \{0, 1\}$ which would be defined[1] as

$$g_{\text{thres}}(x) = \begin{cases} 0 & x < 0 \\ 1 & x \geq 0 \end{cases}. \tag{4.7}$$

Commonly used activation functions in modern neural networks include the sigmoid

$$g_{\text{sig}}(x) = \frac{1}{1 + e^{-x}} \tag{4.8}$$

and the rectified linear unit (ReLU)

$$g_{\text{ReLU}} = \begin{cases} 0 & x < 0 \\ x & x \geq 0 \end{cases} \tag{4.9}$$

which are depicted in Figure 4.1.

Rectified units do not suffer from the *vanishing gradient effect* [Glorot et al., 2011]. This phenomenon occurs with sigmoid activation functions when they reach high saturation, i.e. when the input is significantly far from zero such that the gradient is almost horizontal. However, the vanishing gradient problem is

---

[1] In fact, McCulloch and Pitts defined the activation to be zero when $x < \theta$ for a threshold parameter $\theta \in \mathbb{R}$ and one otherwise, but in our model the bias term b_i acts as the threshold.
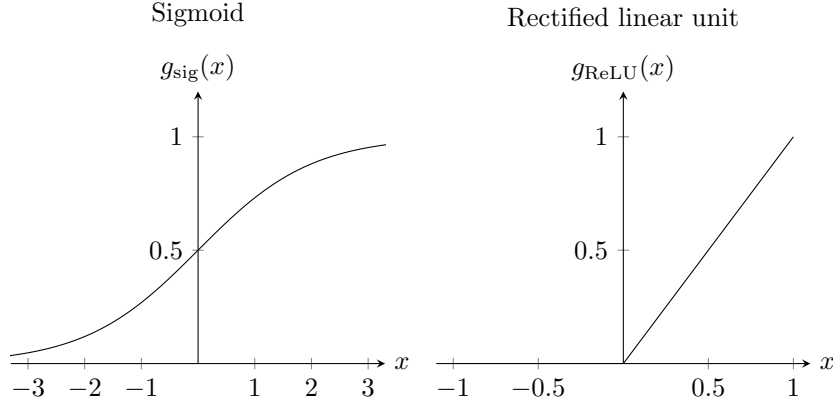
Figure 4.1: Plots of the the two most common activation functions.

usually not prevelant in shallow[2] networks so the sigmoid function still remains popular [Neal, 1992].

Particularly in deep neural networks, different neurons (grouped in *layers*, see Section 4.2.2) often have different activation functions [Burkov, 2019], but for the purposes of this report it is more convenient (in terms of notation) to have the activation function be the same for all neurons, so it does not need to be supplied as a parameter to the function describing the particular neural network. Much of the work in this report can easily be generalized to designs with multple activation functions. This is because the algorithms explained in this report do not concern themselves with the specifics of the activation functions, as long as they are non-linear.

ANNs as regression models We can employ an ANN to model a regression problem of the form given in (4.1). To do so, we need at least $D+1$ neurons in the network. We consider the first D units to be the *input* neurons, and the last neuron, n, is the output unit. Furthermore, we require $w_{j,k} = 0$ for $j, k \in \mathbb{Z}^+$ where $j \leq n$ and $k \leq D$ to ensure that there are no links feeding into the input neurons.

To obtain the prediction $\hat{y}$ given the D-dimensional feature vector $\mathbf{x}$, we set the activation of the ith unit to the value the ith element in $\mathbf{x}$ for $i = 1, \ldots, D$. Then, we propagate the activations using (4.6) until finally the prediction is the activation of the last neuron, $\hat{y} = a_n$. This process is often called *forward propagation* or *forward pass* [Russell and Norvig, 2010].

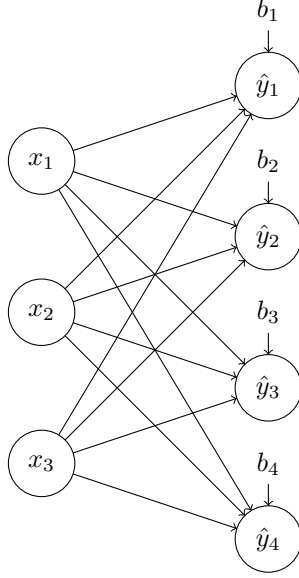[2]Shallow networks refer to ANNs with few layers.

Figure 4.2: A single-layer perceptron with three input and four output neurons.

4.2.1   Single-layer network

We introduce a single-layer network (SLN) as a type of ANN which consists of two conceptual layers, an input and an output layer. Every input node is connected to every output node, but there are no intra-layer links (i.e. there are no links between any two input nodes or any two output nodes), as shown in Figure 4.2. This is what we call a *fully-connected feedforward* architecture. SLN architectures will always form a *directed acyclic graph* (DAG) because there are no intra-layer or backwards connections.

We purposefully use the term SLN instead of single-layer perceptron (SLP) to avoid confusion. A SLP has only one output unit and uses the threshold activation function given in (4.7) [Rosenblatt, 1958]. In our definition of a SLN we allow more than one output and impose no restrictions on g, except that the same activation function is used for every output neuron. We still use the term 'single layer' because the input layer, lacking any incoming weight or bias connections, is not considered to be a 'proper' layer.

Let us consider a SLN with m inputs and n outputs. Since every output unit i only has connections from every input unit j, we can adapt (4.6) to give the activation of a particular output neuron i as

$$a_i = y_i = g\left(z_i\right) = g\left(\sum_{j=1}^{m} w_{j,i} x_j + b_i\right) = g\left(\mathbf{w}_i^{\mathsf{T}} \mathbf{x}_i + b_i\right) \qquad (4.10)$$

12

where $\mathbf{w}_i = \begin{bmatrix} w_{1,i} & w_{2,i} & \cdots & w_{m,i} \end{bmatrix}^\mathsf{T}$ represents the weights of all the edges that connect to output unit i. This is all we need to formally define a SLN.

Definition 3 (Single-layer network). A SLN with m inputs and m outputs is the vector-valued function $\mathbf{S} : \mathbb{R}^m \to \mathbb{R}^n$ defined as

$$\mathbf{S}(\mathbf{x}; \mathbf{W}, \mathbf{b}) = \mathbf{g}\left(\mathbf{W}^\mathsf{T}\mathbf{x} + \mathbf{b}\right) \tag{4.11}$$

where the $m \times n$ matrix

$$\mathbf{W} = \begin{bmatrix} \mathbf{w}_1 & \mathbf{w}_2 & \cdots & \mathbf{w}_n \end{bmatrix} = \begin{bmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,n} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{m,1} & w_{m,2} & \cdots & w_{m,n} \end{bmatrix} \tag{4.12}$$

captures all weights and the vector

$$\mathbf{b} = \begin{bmatrix} b_1 & b_2 & \cdots & b_n \end{bmatrix}^\mathsf{T} \tag{4.13}$$

represents the biases. The vector-valued activation function $\mathbf{g} : \mathbb{R}^n \to \mathbb{R}^n$ is simply the activation function $g : \mathbb{R} \to \mathbb{R}$ applied pointwise to a vector, i.e.

$$\mathbf{g}(\mathbf{z}) = \begin{bmatrix} g(z_1) & g(z_2) & \cdots & g(z_n) \end{bmatrix}^\mathsf{T}$$

for the vector of excitations $\mathbf{z} = \begin{bmatrix} z_1 & z_2 & \cdots & z_n \end{bmatrix}^\mathsf{T}$.

Unlike the formula for a regression model, a SLN is a vector-valued function, due to the fact that there are multiple outputs. Note that when $n = 1$, we reach the same form as in (4.1). Moreover, if we additionally use the threshold activation function from (4.7), we arrive at the SLP model given by Rosenblatt [1958].

4.2.2 Multi-layer perceptron

A multi-layer perceptron[3] (MLP) is a fully-connected feedforward ANN architecture with multiple layers which we will define in terms of multiple nested functions as in Burkov [2019].

Definition 4 (Multi-layer perceptron). A MLP M with m inputs and L layers is the mathematical function $M : \mathbb{R}^m \to \mathbb{R}$ defined as the nested function

$$M(\mathbf{x}; \mathscr{W}, \mathscr{B}) = \hat{y} = f_L\left(\mathbf{f}_{L-1}\left(\ldots\left(\mathbf{f}_1\left(\mathbf{x}\right)\right)\right)\right) \tag{4.14}$$

where $\mathscr{W} = \mathbf{W}_1, \mathbf{W}_2, \ldots, \mathbf{W}_L$ are the weight matrices and $\mathscr{B} = \mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_L$ the bias vectors such that the nested functions are given by $\mathbf{f}_l(\mathbf{x}) = \mathbf{S}(\mathbf{x}; \mathbf{W}_l, \mathbf{b}_l)$ for $l = 1, \ldots, L-1$. The outermost function f_L represents a SLN with only one output unit and is hence the scalar-valued function $f_L(\mathbf{x}) = S(\mathbf{x}; \mathbf{W}_L, \mathbf{b}_L)$.

---

[3]Unlike SLPs, the activation function in a MLP as defined in literature does not necessarily need to be the binary threshold function g_{thres}; in fact, it is often one of the more modern activation functions explained in Section ?? [Hastie et al., 2017; Burkov, 2019]. Hence we can use the term 'multi-layer perceptron'.
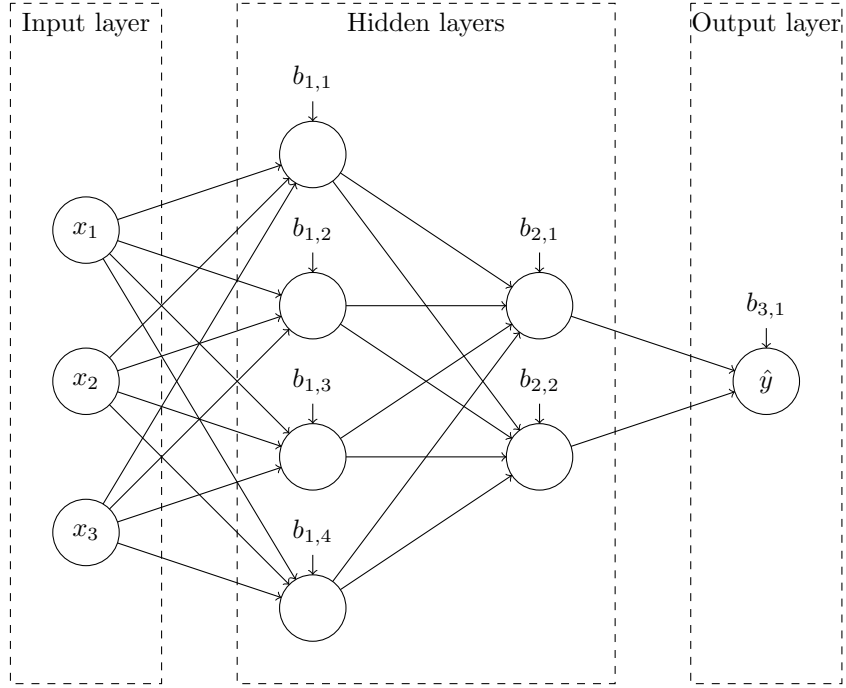
Figure 4.3: A multi-layer perceptron with three inputs and two hidden layers.

Notice that for every $l < L$, $\mathbf{W}_l$ is a $n_l \times m_l$ matrix such that $n_l = m_{l+1}$ to esnure that the number of outputs of layer l is the number of inputs to layer $l+1$. This means that the MLP has m_1 input neurons. Since the final layer has only one output unit, $\mathbf{W}_L$ has only one row, and finally $n_L = 1$

The graph representing this type of network consists of connecting the outputs of the SLN representing layer l with the inputs of the SLN representing layer $l+1$, as shown in Figure 4.3. The layers between the input and output layers are referred to as *hidden* layers.

Since MLPs are simply nested SLNs, it follows that MLPs retain the DAG property and are therefore *feedforward* networks as well. In the forward pass, the activations are propagated from layer to layer (i.e. nested function to nested function) as in (4.11).

Chapter 5

Neural network learning

Chapter 6

Neural surfing theory

6.1 Weight and output spaces

In Definition 4 we established that the tuple $\langle \mathscr{W}, \mathscr{B} \rangle$ along with the activation function is sufficient to fully define a MLP. Most importantly, we have $\mathscr{W} = \mathbf{W}_1, \mathbf{W}_2, \ldots, \mathbf{W}_L$ and $\mathscr{B} = \mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_L$ representing each layer's weight matrices and bias vectors, respectively.

Definition 5 (Weight space). The weight space $\mathcal{W}_A$ of an artificial neural network A is the set of all possible assignments to its *trainable parameters*. The trainable parameters are its weights $\mathscr{W}$ and biases $\mathscr{B}$. If A has P trainable parameters then its weight space is defined as

$$\mathcal{W}_A = \mathbb{R}^P. \tag{6.1}$$

Definition 6 (Output space). The output space $\mathcal{O}_A$ of an artificial neural network A with one output neuron spans the space of all possible output predictions on the training set. From (4.4), the vector $\hat{\mathbf{y}}$ represents the prediction $\hat{y}$ for all N training samples. The output space spans all possible assignments of $\hat{\mathbf{y}}$, so

$$\mathcal{O}_\mathcal{A} = \mathbb{R}^N. \tag{6.2}$$

Lemma 1. *The weight space for a SLN S with m inputs and n outputs is $\mathcal{W}_S = \mathbb{R}^{n(m+1)}$.*

Proof. S's trainable parameters are the weight matrix $\mathbf{W} \in \mathbb{R}^{m \times n}$ from (4.12) and bias vector $\mathbf{b} \in \mathbb{R}^n$ from (4.13). By Definition 5, the weight space encompasses all values of $\mathbf{W}$ and $\mathbf{b}$, so

$$\mathcal{W}_S = \mathbb{R}^{m \times n} \times \mathbb{R}^n = \mathbb{R}^{mn+n} = \mathbb{R}^{(m+1)n}.$$

$\square$

Lemma 2. *A MLP M with L layers where the number of inputs to layer l is given as m_l will have the weight space $\mathcal{W}_M = \mathbb{R}^P$ where $P = \sum_{l=1}^{L-1} (m_{l+1}(m_l + 1)) + m_L + 1$.*

Proof. By Definition 4, M is comprised of L SLNs which we will denote $\mathbf{S}_1, \mathbf{S}_2, \ldots, S_L$. This allows us to express the weight space of M as the product of the weight spaces of each of the SLNs,

$$\mathcal{W}_M = \prod_{l=1}^{L} \mathcal{W}_{S_l}.$$

For every layer l, the number of inputs to S_l will be the number of inputs to the lth layer, m_l. Let n_l denote the number outputs for each layer l. Then, by Lemma 1,

$$\mathcal{W}_{S_l} = \mathbb{R}^{n_l(m_l+1)}.$$

By splitting of the last factor in the product of weight spaces, we obtain

$$\mathcal{W}_M = \prod_{l=1}^{L-1} \mathcal{W}_{S_l} \times \mathcal{W}_{S_L} = \prod_{l=1}^{L-1} \mathbb{R}^{n_l(m_l+1)} \times \mathbb{R}^{n_L(m_L+1)}.$$

Notice that for any layer l, the number of outputs is equal to the number of inputs to the next layer, so $n_l = m_{l+1}$ except for the last layer where there is only one output unit leaving $n_L = 1$. This leaves

$$\mathcal{W}_M = \prod_{l=1}^{L-1} \mathbb{R}^{m_{l+1}(m_l+1)} \times \mathbb{R}^{m_L+1}$$

$$= \mathbb{R}^{\sum_{l=1}^{L-1} m_{l+1}(m_l+1)} \times \mathbb{R}^{m_L+1}$$

$$= \mathbb{R}^{\sum_{l=1}^{L-1} m_{l+1}(m_l+1)+m_L+1},$$

so $\mathcal{W}_M = \mathbb{R}^P$ with

$$P = \sum_{l=1}^{L-1} m_{l+1}(m_l + 1) + m_L + 1.$$

$\square$

6.1.1 Relationship between weight and output space

The significance of Lemma 2 is that we obtain a formula for the number of trainable parameters P in a MLP. By Definition 5, P determines the dimensionality of the weight space. On other other hand, Definition 6 states that the number of samples in the training set N determines the dimensionality of the output space. There is no relationship between P and N since the number of samples in the training set can be arbitrarily chosen. It follows that there is no relationship between the dimensionalities of $\mathcal{W}$ and $\mathcal{O}$.

We will now examine the nature of the mapping between the two spaces, and whether there exists a linear mapping. Note that linear mappings can exist between spaces of different dimensionalities [Rudin, 2006].
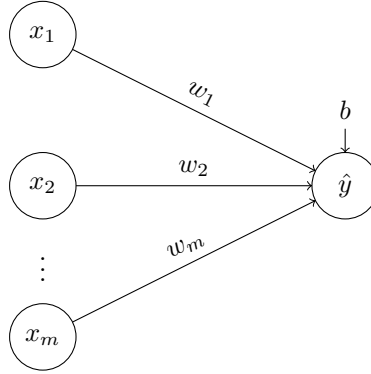
Figure 6.1: The DAG representing a SLN with m inputs and one output unit.

Definition 7 (Weight-output mapping). Given an artifical neural network $A : \mathbb{R}^m \to \mathbb{R}^n$ with m inputs and n outputs parameterized by a set of trainable parameters $\mathbf{w} \in \mathcal{W}_{\mathcal{A}}$, the weight-to-output-space mapping $h_A : \mathcal{W}_{\mathcal{A}} \to \mathcal{O}_A$ for a dataset with N m-dimensional feature vectors given by the matrix $\mathbf{X} = \begin{bmatrix} \mathbf{x}_1 & \mathbf{x}_2 & \cdots & \mathbf{x}_N \end{bmatrix}^{\mathsf{T}} \in \mathbb{R}^{N \times m}$ is

$$h_A (\mathbf{w}) = \begin{bmatrix} A(\mathbf{x}_1; \mathbf{w}) \\ A(\mathbf{x}_2; \mathbf{w}) \\ \vdots \\ A(\mathbf{x}_N; \mathbf{w}) \end{bmatrix}.$$

TODO : explain confusion of the term weight-output mapping

Theorem 3. *For a SLN S with one output unit, the function $h_S : \mathcal{W}_S \to \mathcal{O}_S$ is not a linear mapping.*

Proof. Let S have m inputs, as depicted in Figure 6.1. Modifying the formula for a SLN given in Definition 3 (4.11) for the case where there is only one output unit, we obtain $\hat{y} = f_{\mathrm{SLP}}(\mathbf{x}; \mathbf{w}^{\mathsf{T}}, b) = g\left(\mathbf{w}\mathbf{x} + b\right)$ where $\mathbf{x} = \begin{bmatrix} x_1 & x_2 & \cdots & x_m \end{bmatrix}^{\mathsf{T}}$ is the input feature vector.

We will consider a dataset with N samples where the input is given by the $N \times m$ matrix $\mathbf{X} = \begin{bmatrix} \mathbf{x}_1 & \mathbf{x}_2 & \cdots & \mathbf{x}_N \end{bmatrix}^{\mathsf{T}}$ as in (4.3). By Definition 7, the mapping from weight to output space $h_S : \mathcal{W}_S \to \mathcal{O}_S$ is

$$h_S (\mathbf{w}) = \begin{bmatrix} g\left(\mathbf{w}^{\mathsf{T}}\mathbf{x}_1 + b\right) \\ g\left(\mathbf{w}^{\mathsf{T}}\mathbf{x}_2 + b\right) \\ \vdots \\ g\left(\mathbf{w}^{\mathsf{T}}\mathbf{x}_N + b\right) \end{bmatrix}.$$

We will assume that h is a linear mapping. From the definition of linear mappings, it must be true that $h(\mathbf{u} + \mathbf{v}) = h(\mathbf{u}) + h(\mathbf{v})$ for $\mathbf{u}, \mathbf{v} \in \mathcal{W}_S$ [Rudin,

2006]. On the LHS we have

$$h\left(\mathbf{u}+\mathbf{v}\right)=\begin{bmatrix}g((\mathbf{u}+\mathbf{v})^{\mathsf{T}}\mathbf{x}_1+b)\\g((\mathbf{u}+\mathbf{v})^{\mathsf{T}}\mathbf{x}_2+b)\\\vdots\\g((\mathbf{u}+\mathbf{v})^{\mathsf{T}}\mathbf{x}_N+b)\end{bmatrix}=\begin{bmatrix}g(\mathbf{u}^{\mathsf{T}}\mathbf{x}_1+\mathbf{v}^{\mathsf{T}}\mathbf{x}_1+b)\\g(\mathbf{u}^{\mathsf{T}}\mathbf{x}_2+\mathbf{v}^{\mathsf{T}}\mathbf{x}_2+b)\\\vdots\\g(\mathbf{u}^{\mathsf{T}}\mathbf{x}_N+\mathbf{v}^{\mathsf{T}}\mathbf{x}_N+b)\end{bmatrix}$$

and on the RHS we get

$$h\left(\mathbf{u}\right)+h\left(\mathbf{v}\right)=\begin{bmatrix}g(\mathbf{u}^{\mathsf{T}}\mathbf{x}_1+b)\\g(\mathbf{u}^{\mathsf{T}}\mathbf{x}_2+b)\\\vdots\\g(\mathbf{u}^{\mathsf{T}}\mathbf{x}_N+b)\end{bmatrix}+\begin{bmatrix}g(\mathbf{v}^{\mathsf{T}}\mathbf{x}_1+b)\\g(\mathbf{v}^{\mathsf{T}}\mathbf{x}_2+b)\\\vdots\\g(\mathbf{v}^{\mathsf{T}}\mathbf{x}_N+b)\end{bmatrix}=\begin{bmatrix}g(\mathbf{u}^{\mathsf{T}}\mathbf{x}_1+b)+g(\mathbf{v}^{\mathsf{T}}\mathbf{x}_1+b)\\g(\mathbf{u}^{\mathsf{T}}\mathbf{x}_2+b)+g(\mathbf{v}^{\mathsf{T}}\mathbf{x}_2+b)\\\vdots\\g(\mathbf{u}^{\mathsf{T}}\mathbf{x}_N+b)+g(\mathbf{v}^{\mathsf{T}}\mathbf{x}_N+b)\end{bmatrix}.$$

We obtain

$$\begin{bmatrix}g(\mathbf{u}^{\mathsf{T}}\mathbf{x}_1+\mathbf{v}^{\mathsf{T}}\mathbf{x}_1+b)\\g(\mathbf{u}^{\mathsf{T}}\mathbf{x}_2+\mathbf{v}^{\mathsf{T}}\mathbf{x}_2+b)\\\vdots\\g(\mathbf{u}^{\mathsf{T}}\mathbf{x}_N+\mathbf{v}^{\mathsf{T}}\mathbf{x}_N+b)\end{bmatrix}=\begin{bmatrix}g(\mathbf{u}^{\mathsf{T}}\mathbf{x}_1+b)+g(\mathbf{v}^{\mathsf{T}}\mathbf{x}_1+b)\\g(\mathbf{u}^{\mathsf{T}}\mathbf{x}_2+b)+g(\mathbf{v}^{\mathsf{T}}\mathbf{x}_2+b)\\\vdots\\g(\mathbf{u}^{\mathsf{T}}\mathbf{x}_N+b)+g(\mathbf{v}^{\mathsf{T}}\mathbf{x}_N+b)\end{bmatrix}.$$

Let $\alpha_i=\mathbf{u}^{\mathsf{T}}\mathbf{x}_i$ and $\beta_i=\mathbf{v}^{\mathsf{T}}\mathbf{x}_i$ for all i. Since $\mathbf{u},\mathbf{v}\in\mathbb{R}^m$ and all $\mathbf{x}_i\in\mathbb{R}^m$, it follows that $\alpha_i,\beta_i\in\mathbb{R}$ for all i. Hence $g(\alpha+\beta+b)=g(\alpha+b)+g(\beta+b)$.

The only functions that satisfy g are functions that satisfy Cauchy's functional equation[1], but these solutions only apply when $b=0$ and furthermore are linear, whereas the activation function g is non-linear. We arrived at a contradiction, thus disproving our initial assumption that h is a linear mapping, so it must be a non-linear mapping. □

Corollary 3.1. *For any SLN S, the function $h_S:\mathcal{W}_S\to\mathcal{O}_S$ is not a linear mapping.*

Proof. We will generalize the results from Theorem 3 to SLNs with multiple outputs. Let S have m inputs and n outputs. We construct n smaller SLNs, $S_1,S_2,\ldots,S_n$ where each S_i has all m input units, but only the ith output unit. The DAG representing S_i will only contain links from the input nodes to output node $\hat{y}_i$ (and, of course, the associated bias term b_i) as depicted in Figure 6.2.

Now, we can simulate the function of S by the construction

$$S(\mathbf{x})=\begin{bmatrix}\hat{y}_1\\\hat{y}_2\\\vdots\\\hat{y}_n\end{bmatrix}=\begin{bmatrix}S_1(\mathbf{x})\\S_2(\mathbf{x})\\\vdots\\S_n(\mathbf{x})\end{bmatrix}.$$

By Theorem 3, each S_i does not have a linear mapping from weight space to output space, so S cannot have a linear mapping either. □

---

[1] Cauchy's functional equation is $f(a+b)=f(a)+f(b)$. For $a,b\in\mathbb{Q}$, the only solutions are linear functions of the form $f(x)=cx$ for some $c\in\mathbb{Q}$ [Reem, 2017].
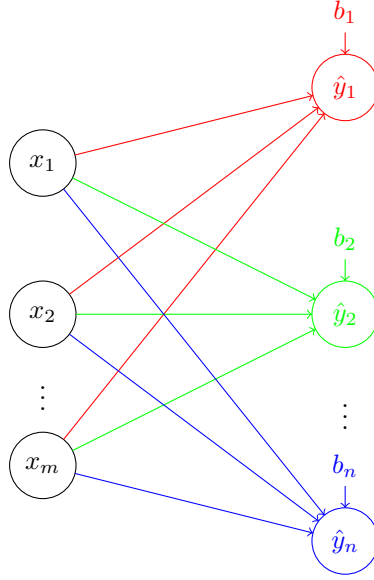
Figure 6.2: The DAGs representing the constructions of n SLNs with one output from a SLN with m inputs and n outputs. Each color represents one of the constructed smaller SLNs.

Corollary 3.2 (Weight-output mapping in general). *For any MLP, $h : \mathcal{W} \to \mathcal{O}$ is not a linear mapping.*

Proof. Consider a MLP M with L layers. By Definition 4, that MLP is a nested function of L SLNs. Corollary 3.1 states that each of these SLNs does not have a linear mapping from weight to output space. Hence the composition of L SLNs that forms M does not have a linear mapping from weight to output space. $\qquad \square$

TODO : Significance of this proof

6.2 Ideal goal line

Chapter 7

Problems

7.1   Stripe problem

Chapter 8

Generalising neural surfing

Generalize to classification as regression with multiple output variables

Part II

Framework

Chapter 9

Design

Chapter 10

Implementation

TODO

Bibliography

Burkov, A. (2019). *The Hundred-Page Machine Learning Book*. Andriy Burkov.

Glorot, X., Bordes, A., and Bengio, Y. (2011). Deep sparse rectifier neural networks. In Gordon, G., Dunson, D., and Dudík, M., editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pages 315–323, Fort Lauderdale, FL, USA. PMLR.

Hastie, T., Friedman, J., and Tisbshirani, R. (2017). *The Elements of statistical learning: data mining, inference, and prediction*. Springer, 2nd edition.

McCulloch, W. S. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, 5(4):115–133.

Neal, R. M. (1992). Connectionist learning of belief networks. *Artificial Intelligence*, 56(1):71–113.

Reem, D. (2017). Remarks on the cauchy functional equation and variations of it. *Aequationes mathematicae*, 91(2):237–264.

Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408.

Rudin, W. (2006). *Functional Analysis*. International series in pure and applied mathematics. McGraw-Hill.

Russell, S. and Norvig, P. (2010). *Artificial Intelligence: A Modern Approach*. Pearson, 3rd edition.