# Feedforward Neural Networks

Michael Collins

## 1 Introduction

In the previous notes, we introduced an important class of models, *log-linear models*. In this note, we describe *feedforward neural networks*, which extend log-linear models in important and powerful ways.

Recall that a log-linear model takes the following form:

$$p(y|x;v) = \frac{\exp\left(v \cdot f(x,y)\right)}{\sum_{y' \in \mathcal{Y}} \exp\left(v \cdot f(x,y')\right)} \tag{1}$$

Here $x$ is an input, $y$ is a "label", $v \in \mathbb{R}^d$ is a parameter vector, and $f(x,y) \in \mathbb{R}^d$ is a feature vector that corresponds to a **representation** of the pair $(x,y)$.

Log-linear models have the advantage that the feature vector $f(x,y)$ can include essentially any features of the pair $(x,y)$. However, these features are generally designed by hand, and in practice this is a limitation. It can be laborious to define features by hand for complex problems such as language modeling, tagging, parsing, or machine translation.

Neural networks essentially allow the representation itself to be *learned*. In practice, this can significantly decrease the amount of human engineering required in various applications. More importantly, empirical results across a broad set of domains have shown that learned representations in neural networks can give very significant improvements in accuracy over hand-engineered features.

In the remainder of this note we first introduce *multi-class feedforward networks*. In a later note we will describe how these models can be trained, using stochastic gradient descent in conjunction with the *backpropagation algorithm* for calculation of gradients.

## 2 Multi-Class Feedforward Networks

Our first step in deriving multi-class feedforward networks is to introduce a variant of the log-linear models defined in Eq. 1. Consider a model where $f(x) \in \mathbb{R}^d$ is a

feature vector for an input $x$, $v(y) \in \mathbb{R}^d$ for each label $y$ is a parameter vector for label $y$, and $\gamma_y \in \mathbb{R}$ is a "bias" parameter for label $y$. We will use $v$ to refer to the set of all parameter vectors and bias values: that is, $v = \{(v(y), \gamma_y) : y \in \mathcal{Y}\}$. The distribution $p(y|x; v)$ is then defined as follows:

$$p(y|x; v) = \frac{\exp\left(v(y) \cdot f(x) + \gamma_y\right)}{\sum_{y' \in \mathcal{Y}} \exp\left(v(y') \cdot f(x) + \gamma_{y'}\right)} \tag{2}$$

Some remarks on this model:

- The feature vector $f(x)$ can capture any features of the input $x$.
- The score $v \cdot f(x, y)$ in Eq. 1 has essentially been replaced by $v(y) \cdot f(x) + \gamma_y$ in Eq. 2.

The two model forms are closely related. It can be shown that Eq. 2 is a special case of Eq. 1: more specifically, and model of the form in Eq. 2 can be easily converted into an equivalent model in the form of Eq. 1. However the model form in Eq. 2 is commonly used within feedforward neural networks, so we will work with this formulation.

The next step is to replace the feature vector $f(x)$ with a function $\phi(x; \theta)$ where $\theta$ are some additional parameters of the model. This gives a model of the following form:

$$p(y|x; \theta, v) = \frac{\exp\left(v(y) \cdot \phi(x; \theta) + \gamma_y\right)}{\sum_{y' \in \mathcal{Y}} \exp\left(v(y') \cdot \phi(x; \theta) + \gamma_{y'}\right)}$$

We now have two sets of parameters, $\theta$ and $v$. Both sets of parameters will be learned. The parameters $\theta$ define a representation $\phi(x; \theta)$. For now we leave the exact form of $\phi(x; \theta)$ unspecified; we will describe these extensively later in this note.[1]

This leads us to the following definition:

**Definition 1 (Multi-Class Feedforward Models)** *A multi-class feedforward model consists of the following components:*

- *A set $\mathcal{X}$ of possible inputs.*
- *A set $\mathcal{Y}$ of possible labels. The set $\mathcal{Y}$ is assumed to be finite.*

[1] We can now see one important advantage of using models of the form of Eq. 2 instead of Eq. 1 is that the representation $\phi(x; \theta)$ does not depend on the label $y$, so it can be computed once when calculating $p(y|x; \theta, v)$. Working with a representation of the form $\phi(x, y; \theta)$ would require recalculation of $\phi(\ldots)$ for each possible label $y$. In practice computing $\phi(\ldots)$ is computationally intensive for neural networks, so computing it once is a significant computational advantage.

- *A positive integer $d$ specifying the number of features and in the feedforward representation.*
- *A parameter vector $\theta$ defining the* feedforward parameters *of the network. We use $\Omega$ to refer to the set of possible values for $\theta$.*
- *A function $\phi : \mathcal{X} \times \Omega \to \mathbb{R}^d$ that maps any $(x, \theta)$ pair to a "feedforward representation" $\phi(x; \theta)$.*
- *For each label $y \in \mathcal{Y}$, a parameter vector $v(y) \in \mathbb{R}^d$, and a bias value $\gamma_y \in \mathbb{R}$.*

*For any $x \in \mathcal{X}$, $y \in \mathcal{Y}$, the model defines a conditional probability*

$$p(y|x; \theta, v) = \frac{\exp\left(v(y) \cdot \phi(x; \theta) + \gamma_y\right)}{\sum_{y' \in \mathcal{Y}} \exp\left(v(y') \cdot \phi(x; \theta) + \gamma_{y'}\right)}$$

*Here $\exp(x) = e^x$, and $v(y) \cdot \phi(x; \theta) = \sum_{k=1}^{d} v_k(y)\phi_k(x; \theta)$ is the inner product between $v(y)$ and $\phi(x; \theta)$. The term $p(y|x; \theta, v)$ is intended to be read as "the probability of $y$ conditioned on $x$, under parameter values $\theta$ and $v$".* □

This definition leads to the following questions:

- How can we define the feedforward representation $\phi(x; \theta)$?
- Given training examples $(x_i, y_i)$ for $i = 1 \ldots n$, how can we train the parameters $\theta$ and $v$?

Section 3 describes a generic solution to the second problem, *gradient-based learning*. Section 4 then gives a full description of how $\phi(x; \theta)$ is defined in single-layer feedforward networks.

# 3 Gradient-Based Learning

Figure 1 shows a simple generic algorithm for training the parameters of a multi-class feedforward network. The algorithm takes $T$ training steps. At each training step it selects a training example index $i$ uniformly at random from $\{1 \ldots n\}$ where $n$ is the number of training examples. It then defines the following function of the parameters $\theta$, $v$ in the model

$$L(\theta, v) = -\log p(y_i|x_i; \theta, v)$$

It can be seen that $L(\theta, v)$ is the negative log-likelihood of training example $(x_i, y_i)$ under the current parameters.

Finally, it updates the parameters $\theta$ and $v$ using the updates

$$\theta_j = \theta_j - \eta^t \times \frac{dL(\theta, v)}{d\theta_j}$$

and

$$v_k(y) = v_k(y) - \eta^t \times \frac{dL(\theta, v)}{dv_k(y)}$$

$$\gamma_y = \gamma_y - \eta^t \times \frac{dL(\theta, v)}{d\gamma_y}$$

Thus the parameter vectors move in the direction of the derivatives of $L(\theta, v)$. The parameter $\eta^t > 0$ is referred to as a *learning rate*; it governs how far the parameters move.

The algorithm is a simple variant of *stochastic gradient descent*. It is stochastic because it selects a random training example for each update. It is gradient descent because it can be interpreted as minimizing the log-likelihood function

$$-\sum_{i=1}^{n} \log p(y_i|x_i; \theta, v)$$

by following the gradients of this function.

The main computational step in the algorithm is to calculate the derivatives

$$\frac{dL(\theta, v)}{d\theta_j}$$

$$\frac{dL(\theta, v)}{d\gamma_y}$$

and

$$\frac{dL(\theta, v)}{dv_k(y)}$$

We will see that a method called back-propagation can be used to calculate derivatives of this form in feedforward networks.

# 4 Single-Layer Feedforward Representations

We now describe how to define the function $\phi(x; \theta)$ using single-layer feedforward networks.

**Inputs:** Training examples $(x_i, y_i)$ for $i = 1 \ldots n$. A feedforward representation $\phi(x; \theta)$. An integer $T$ specifying the number of updates. A sequence of learning rate values $\eta^1 \ldots \eta^T$ where each $\eta^t > 0$.
**Initialization:** Set $v$ and $\theta$ to random parameter values.
**Algorithm:**

- For $t = 1 \ldots T$
  - Select an integer $i$ uniformly at random from $\{1 \ldots n\}$
  - Define
$$L(\theta, v) = -\log p(y_i | x_i; \theta, v)$$
  - For each parameter $\theta_j$,
$$\theta_j = \theta_j - \eta^t \times \frac{dL(\theta, v)}{d\theta_j}$$
  - For each label $y$, for each parameter $v_k(y)$,
$$v_k(y) = v_k(y) - \eta^t \times \frac{dL(\theta, v)}{dv_k(y)}$$
  - For each label $y$,
$$\gamma_y = \gamma_y - \eta^t \times \frac{dL(\theta, v)}{d\gamma_y}$$

Figure 1: A simple variant of stochastic gradient descent (SGD), used to train a multi-class feedforward network.

## 4.1 Defining the input to a feedforward network

Our first assumption will be that we have a function that maps an input $x$ to a vector $f(x) \in \mathbb{R}^D$. This vector will be the input to the feedforward network. In general it is assumed that the representation $f(x)$ is "simple", not requiring careful hand-engineering. The neural network will take $f(x)$ as input, and will produce a representation $\phi(x; \theta)$ that depends on the input $x$ and the parameters $\theta$.

Note that we could build a log-linear model using $f(x)$ as the representation:

$$p(y|x; v) = \frac{\exp\{v(y) \cdot f(x)\}}{\sum_{y'} \exp\{v(y') \cdot f(x)\}} \tag{3}$$

This is a "linear" model, because the score $v(y) \cdot f(x)$ is linear in the input features $f(x)$. The general assumption is that a model of this form will perform poorly or at least non-optimally. Neural networks enable "non-linear" models that perform at much higher levels of accuracy.

We now give a few examples of how $f(x)$ can be constructed:

**Example 1: Acoustic modeling for speech recognition.** In this domain the goal is to map an acoustic waveform $a$ to a sentence $s$. A critical sub-problem in building a speech recognizer is estimating the probability distribution

$$p(y|x)$$

where $y$ is a phoneme label, which can take any value in some set $\mathcal{Y}$ of possible phonemes, and $x$ is a pair $(a, i)$ consisting of the acoustic waveform $a$ together with a "position" $i$ in the acoustic waveform. The position corresponds to a small segment of the acoustic waveform, typically around 10 milliseconds in length.

To be concrete, we will assume that $a$ is equal to a sequence of vectors $a = a^1 a^2 \dots a^m$ where $m$ is the length of the sequence, each $a^i \in \mathbb{R}^D$ for $i = 1 \dots m$ represents a 10 millisecond portion of the acoustic sequence. A common choice would be for $a^i$ to be a $D$-dimensional vector that represents the energy in the 10 millisecond section of speech at different frequencies. Thus each component $a^i_j$ for $j \in 1 \dots D$ would represent the energy in the $j$'th frequency band. A typical choice for $D$ would be approximately $D = 40$.

Once a model $p(y|x) = p(y|a, i)$ has been estimated, it can be integrated within a full speech recognition model that takes the entire sequence $a$ and produces a sentence $s$. Given that the set of possible positions in $a$ is $\{1 \dots m\}$, the probability distributions

$$p(y|a, i)$$

for $i = 1 \ldots m$ are taken into account when searching for the most likely sentence $s$.

In recent years, multi-class feedforward networks have been shown to be very successful methods for building a model of $p(y|a, i)$ in this domain, giving very significant improvements in accuracy over previous methods. In the simplest approach, the representation $f(x) = f(a, i)$ is simply defined to be $f(a, i) = a^i \in \mathbb{R}^{40}$.

In a slightly more complex approach, the representation might take into account the 40-dimensional representation of frames within some window of position $i$. For example we could construct $f(x) \in \mathbb{R}^{360}$ by simply concatenating the 40-dimensional vector representations of all 9 positions within the range $\{(i-4), (i-3), \ldots i, \ldots, (i+3), (i+4)\}$. Thus $f(a, i) = [a^{i-4}; a^{i-3}; \ldots a^i; \ldots a^{i+3}; a^{i+4}]$. This allows the model to take into account more context when modeling $p(y|a, i)$. □

**Example 2: Handwritten Digit Recognition** In this example our task is to map an image $x$ to a label $y$. Each image contains a hand-written digit in the set $\{0, 1, 2, \ldots 9\}$. The input representation $f(x) \in \mathbb{R}^D$ simply represents the pixel values in the image. For example if the image is $16 \times 16$ grey-scale pixels, where each pixel takes some value indicating how bright it is, we would have $D = 256$, with $f(x)$ just being the list of values for the 256 different pixels in the image.

For this task using the representation $f(x)$ in a linear model, as shown in Eq. 3, leads to poor performance. Neural networks, which effectively construct non-linear models from the input $f(x)$, give much improved performance. □

## 4.2 Neural Networks with a Single Hidden Layer

We now describe how feedforward representations $\phi(x; \theta)$ are defined. We will begin with networks with a single hidden layer, then describe the generalization to multiple hidden layers.

Throughout this section we assume $x = f(x)$: that is, the input $x$ is itself a vector used as input to the network. This will make notation less cumbersome. However it is important to remember that in general we will need to define some function $f(x)$ defining the input to the network, as described in the previous section.

### 4.2.1 Neurons

A key concept will be that of a *neuron*. A neuron is defined by a weight vector $w \in \mathbb{R}^D$, a bias $b \in \mathbb{R}$, and a transfer function $g : \mathbb{R} \to \mathbb{R}$. The neuron maps an

input vector $x$ to an output $h$ as follows:

$$h = g(w \cdot x + b)$$

The vector $w \in \mathbb{R}^d$ and scalar $b \in \mathbb{R}$ are parameters of the model, which are learned from training examples.

It is critical that the transfer function is non-linear. A linear function would be of the form

$$g(z) = \alpha z + \beta$$

for constants $\alpha \in \mathbb{R}$ and $\beta \in \mathbb{R}$. Some commonly used transfer functions—the "ReLU" and "tanh" functions—are shown in figure 2.

In addition to giving definitions of $g(z)$, figure 2 gives the derivatives

$$\frac{dg(z)}{dz}$$

for each transfer function. These derivatives will be important when deriving a gradient-based learning method for models that make use of these transfer functions. In particular, given that

$$h = g(w \cdot x + b)$$

it will be useful to calculate partial derivatives

$$\frac{\partial h}{\partial w_j}$$

for the parameters $w_1, w_2, \ldots w_d$, and also

$$\frac{\partial h}{\partial b}$$

for the bias parameter $b$. To do this we can use the *chain rule of differentiation*. First introduce an intermediate variable $z \in \mathbb{R}$:

$$z = w \cdot x + b$$

$$h = g(z)$$

Then by the chain rule we have

$$\frac{\partial h}{\partial w_j} = \frac{\partial h}{\partial z} \times \frac{\partial z}{\partial w_j} = \frac{\partial g(z)}{\partial z} \times x_j$$

and

$$\frac{\partial h}{\partial b} = \frac{\partial h}{\partial z} \times \frac{\partial z}{\partial b} = \frac{\partial g(z)}{\partial z} \times 1$$

Here we have used $\frac{\partial h}{\partial z} = \frac{\partial g(z)}{\partial z}$, $\frac{\partial z}{\partial w_j} = x_j$, and $\frac{\partial z}{\partial b} = 1$.

**Definition 2 (The ReLU (rectified linear unit) transfer function)** *The ReLU transfer function is defined as*

$$g(z) = \{z \text{ if } z \geq 0, \text{ or } 0 \text{ if } z < 0\}$$

*Or equivalently,*

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

*It follows that the derivative is*

$$\frac{dg(z)}{dz} = \{1 \text{ if } z > 0, \text{ or } 0 \text{ if } z < 0, \text{ or undefined if } z = 0\}$$

□

**Definition 3 (The tanh transfer function)** *The tanh transfer function is defined as*

$$g(z) = \frac{e^{2z} - 1}{e^{2z} + 1}$$

*It can be shown that the derivative is*

$$\frac{dg(z)}{dz} = 1 - (g(z))^2$$

□

Figure 2: Two commonly used transfer functions, the ReLU function, and the tanh function.

### 4.2.2 Single-Layer Neural Networks

We now describe how to construct a neural network with a single hidden layer. The key step will be to introduce $m$ neurons, where $m$ is some integer. The definition is then as follows:

**Definition 4 (Single-Layer Feedforward Representation)** *A single-layer feedforward representation consists of the following:*

- *An integer $d$ specifying the input dimension. Each input to the network is a vector $x \in \mathbb{R}^d$.*
- *An integer $m$ specifying the number of hidden units.*
- *A parameter matrix $W \in \mathbb{R}^{m \times d}$. We use the vector $W_k \in \mathbb{R}^d$ for each $k \in \{1, 2, \ldots m\}$ to refer to the $k$'th row of $W$.*
- *A vector $b \in \mathbb{R}^m$ of bias parameters.*
- *A transfer function $g : \mathbb{R} \to \mathbb{R}$. Common choices are $g(x) = ReLU(x)$ or $g(x) = tanh(x)$.*

*We then define the following:*

- *For $k = 1 \ldots m$, the input to the $k$'th neuron is $z_k = W_k \cdot x + b_k$.*
- *For $k = 1 \ldots m$, the output from the $k$'th neuron is $h_k = g(z_k)$.*
- *Finally, define the vector $\phi(x; \theta) \in \mathbb{R}^m$ as $\phi_k(x; \theta) = h_k$ for $k = 1 \ldots m$. Here $\theta$ denotes the parameters $W \in \mathbb{R}^{m \times d}$ and $b \in \mathbb{R}^m$. Hence $\theta$ contains $m \times (d + 1)$ parameters in total.*

We can then use the definition of multiclass feedforward models to define

$$p(y|x; \theta, v) = \frac{\exp\{v(y) \cdot \phi(x; \theta)\}}{\sum_{y'} \exp\{v(y') \cdot \phi(x; \theta)\}}$$

from which it follows that

$$\log p(y|x; \theta, v) = v(y) \cdot \phi(x; \theta) - \log \sum_{y'} \exp\{v(y') \cdot \phi(x; \theta)\}$$

It can be seen that the neural network employs $m$ units, each with their own parameters $W_k$ and $b_k$, and these neurons are used to construct a "hidden" representation $h \in \mathbb{R}^m$. The representation is referred to as being hidden because during

training we will only observe inputs $x^i$ together with labels $y^i$: the values for the hidden representation are unobserved, and will be learned through gradient descent on the parameters $W$, $b$, and $v$.

It will be convenient to write the above operations in matrix form, which can be considerably more compact. We can for example replace the operation

$$z_k = W_k \cdot x + b_k \quad \text{for } k = 1 \dots m$$

with

$$z = Wx + b$$

where the dimensions are as follows (note that an $m$-dimensional column vector is equivalent to a matrix of dimension $m \times 1$):

$$\underbrace{z}_{m\times 1} = \underbrace{\underbrace{W}_{m\times d}\underbrace{x}_{d\times 1}}_{m\times 1} + \underbrace{b}_{m\times 1}$$

This leads to the following definition:

**Definition 5 (Single-Layer Feedforward Representation (Matrix Form))** *A single-layer feedforward representation consists of the following:*

- *An integer $d$ specifying the input dimension. Each input to the network is a vector $x \in \mathbb{R}^d$.*
- *An integer $m$ specifying the number of hidden units.*
- *A matrix of parameters $W \in \mathbb{R}^{m\times d}$.*
- *A vector of bias parameters $b \in \mathbb{R}^m$*
- *A transfer function $g : \mathbb{R}^m \to \mathbb{R}^m$. Common choices would be to define $g(z)$ to be a vector with components $ReLU(z_1), ReLU(z_2), \dots, ReLU(z_m)$ or $tanh(z_1), tanh(z_2), \dots, tanh(z_m)$.*

*We then define the following:*

- *The vector of inputs to the hidden layer $z \in \mathbb{R}^m$ is defined as $z = Wx + b$.*
- *The vector of outputs from the hidden layer $h \in \mathbb{R}^m$ is defined as $h = g(z)$*
- *Finally, define $\phi(x;\theta) = h$. Here the parameters $\theta$ contain the matrix $W$ and the vector $b$.*
- *It follows that*

$$\phi(x;\theta) = g(Wx + b)$$

□

### 4.2.3 A Motivating Example: the XOR Problem

We now motivate the use of feedforward networks, using a classic problem, the XOR problem. We will show that a simple linear model fails on this problem, whereas a simple neural network can succeed in modeling that data.

We will assume a training set where each label is in the set $\mathcal{Y} = \{-1, +1\}$, and there are 4 training examples, as follows:

$$x^1 = [0, 0], \quad y^1 = -1$$
$$x^2 = [0, 1], \quad y^2 = 1$$
$$x^3 = [1, 0], \quad y^3 = 1$$
$$x^4 = [1, 1], \quad y^4 = -1$$

Note that for any $x = (x_1, x_2)$, the label $y$ is equal to XOR$(x_1, x_2)$ for a suitable definition of the XOR function.

We now analyze the behaviour of linear models, and neural networks, on this data. The following lemma will be useful:

**Lemma 1** *Assume we have a model of the form*

$$p(y|x; v) = \frac{\exp\{v(y) \cdot x + \gamma_y\}}{\sum_y \exp\{v(y) \cdot x + \gamma_y\}}$$

*and the set of possible labels is* $\mathcal{Y} = \{-1, +1\}$. *Then for any* $x$,

$$p(+1|x; v) > 0.5$$

*if and only if*

$$u \cdot x + \gamma > 0$$

*where* $u = v(+1) - v(-1)$ *and* $\gamma = \gamma_{+1} - \gamma_{-1}$. *Similarly for any* $x$,

$$p(-1|x; v) > 0.5$$

*if and only if*

$$u \cdot x + \gamma < 0$$

*Proof:* We have

$$\begin{aligned} p(+1|x; v) &= \frac{\exp\{v(+1) \cdot x + \gamma_{+1}\}}{\exp\{v(+1) \cdot x + \gamma_{+1}\} + \exp\{v(-1) \cdot x + \gamma_{-1}\}} \\ &= \frac{1}{1 + \exp\{-(u \cdot x + \gamma)\}} \end{aligned}$$

It follows that $p(+1|x;v) > 0.5$ if and only if $\exp\{-(u \cdot x + \gamma)\} < 1$ from which it follows that $u \cdot x + \gamma > 0$.

A similar proof applies to the condition $p(-1|x;v) > 0.5$. □

We can now state and prove a theorem concerning the failure of a simple linear model on the XOR problem:

**Theorem 1** *Assume we have examples $(x^i, y^i)$ for $i = 1 \ldots 4$ as defined above. Assume we have a model of the form*

$$p(y|x;v) = \frac{\exp\{v(y) \cdot x + \gamma_y\}}{\sum_y \exp\{v(y) \cdot x + \gamma_y\}}$$

*Then there are no parameter settings for $v(+1), v(-1), \gamma_{+1}, \gamma_{-1}$ such that*

$$p(y^i|x^i;v) > 0.5 \quad \textit{for } i = 1 \ldots 4$$

*Proof:* From lemma **??**, $p(y^i = 1|x^i;v) > 0.5$ if and only if

$$u \cdot x^i + \gamma > 0$$

where $u = v(+1) - v(-1)$ and $\gamma = \gamma_{+1} - \gamma_{-1}$.

Similarly $p(y^i = 0|x^i;v) > 0.5$ if and only if

$$v \cdot x^i + \gamma < 0$$

where $v = v(+1) - v(-1)$ and $\gamma = \gamma_{+1} - \gamma_{-1}$.

Hence to satisfy $p(y^i|x^i;v) > 0.5$ for $i = 1 \ldots 4$, there must exist parameters $u$ and $\gamma$ such that

$$u \cdot [0,0] + \gamma < 0 \tag{4}$$
$$u \cdot [0,1] + \gamma > 0 \tag{5}$$
$$u \cdot [1,0] + \gamma > 0 \tag{6}$$
$$u \cdot [1,1] + \gamma < 0 \tag{7}$$

Eqs. 5 and 6 imply

$$u \cdot [1,1] + 2\gamma > 0$$

whereas Eq. 4 implies

$$\gamma < 0$$

or equivalently

$$-\gamma > 0$$

hence Eqs. 4, 5, 6 together imply

$$u \cdot [1, 1] + \gamma > 0$$

This is in direct contradication to Eq. 7, so it follows that there is no parameter setting $u, \gamma$ that satisfies Eqs. 4, 5, 6, 7. $\square$

The constraints in Eqs. 4-7 above have a geometric interpretation. The points $x = [x_1, x_2]$ are in a two-dimensional space. The parameters $u$ and $\gamma$ define a hyperplane whose normal is in the direction of $u$, and which is at a distance of $\gamma/||u||$ from the origin. For Eqs. 4-7 to be satisfied, there must be a hyperplane defined by $u$ and $\gamma$ that "seperates" the points labeled $-1$ and $+1$: that is, all points labeled $+1$ are on one side of the hyperplane, all points labeled $-1$ are on the other side. The proof shows that this is not possible; the impossibility of such a hyperplane is also easily verified by plotting the points on a two-dimensional grid.

While a simple linear model fails, we now show that a simple neural network with a single hidden layer with $m = 2$ neurons can successfully model the data:

**Theorem 2** *Assume we have examples $(x^i, y^i)$ for $i = 1 \ldots 4$ as defined above. Assume we have a model of the form*

$$p(y|x; \theta, v) = \frac{\exp\{v(y) \cdot \phi(x; \theta) + \gamma_y\}}{\sum_y \exp\{v(y) \cdot \phi(x; \theta) + \gamma_y\}}$$

*where $\phi(x; \theta)$ is defined by a single layer neural network with $m = 2$ hidden units, and the ReLU$(z)$ activation function. Then there are parameter settings for $v(+1), v(-1), \gamma_{+1}, \gamma_{-1}, \theta$ such that*

$$p(y^i|x^i; v) > 0.5 \quad \textit{for } i = 1 \ldots 4$$

*Proof.* Define $W_1 = [1, 1]$, $W_2 = [1, 1]$, $b_1 = 0$, $b_2 = -1$. Then for each input $x$ we can calculate the value for the vectors $z$ and $h$ corresponding to the inputs and the outputs from the hidden layer:

$$\begin{array}{rcl} x = [0, 0] & \Rightarrow & z = [0, -1] \Rightarrow h = [0, 0] \\ x = [1, 0] & \Rightarrow & z = [1, 0] \Rightarrow h = [1, 0] \\ x = [0, 1] & \Rightarrow & z = [1, 0] \Rightarrow h = [1, 0] \\ x = [1, 1] & \Rightarrow & z = [2, 1] \Rightarrow h = [2, 1] \end{array}$$

Next, note that $\phi(x; \theta) = h$, so

$$p(y|x; \theta, v) = \frac{\exp\{v(y) \cdot h + \gamma_y\}}{\sum_y \exp\{v(y) \cdot h + \gamma_y\}}$$

Hence to satisfy $p(y^i|x^i; v) > 0.5$ for $i = 1 \ldots 4$, there must exist parameters $u = v(+1) - v(-1)$ and $\gamma = \gamma_{+1} - \gamma_{-1}$ such that

$$u \cdot [0, 0] + \gamma < 0 \quad (8)$$
$$u \cdot [1, 0] + \gamma > 0 \quad (9)$$
$$u \cdot [1, 0] + \gamma > 0 \quad (10)$$
$$u \cdot [2, 1] + \gamma < 0 \quad (11)$$

It can be verified that $u = [1, -2], \gamma = -0.5$ satisifies these contraints, because

$$[1, -2] \cdot [0, 0] - 0.5 = -0.5$$
$$[1, -2] \cdot [1, 0] - 0.5 = 0.5$$
$$[1, -2] \cdot [1, 0] - 0.5 = 0.5$$
$$[1, -2] \cdot [2, 1] - 0.5 = -0.5$$

It follows that choosing parameters such that $v(+1) - v(-1) = [1, -2]$ and $\gamma_{+1} - \gamma_{-1} = -0.5$ leads to

$$p(y^i|x^i; \theta, v) > 0.5$$

for $i = 1 \ldots 4$. □

It can be seen that the neural network effectively maps each input $x$ to a new representation $h$. The new representation leads to the constraints in Eqs. 8-11. We now have a set of constraints that can be satisfied by suitably chosen values for $u$ and $\gamma$.