Background

This document will explore the technical details of a feed-forward neural network, and an application for recognizing hand-written digits.

In the simplest case, neural networks can be reduced to a generalized linear model (GLM), where the prediction is a linear combination of inputs but passed through a non-linear function:

$$f(\mathbf{x}, \mathbf{w}) = g\left(\sum_{i=1}^{N} w_i x_i\right) \tag{1}$$

Here $g(\cdot)$ is a non-linear function, with $\mathbf{x}$ is an N dimensional input vector, and $\mathbf{w}$ is the weight vector. Common choices of $g(\cdot)$ for neural networks include the logistic (sigmoid) function, the hyperbolic tangent function, and the rectified linear unit (ReLU):

$$\begin{aligned} g_{\text{logistic}}(z) &= \frac{1}{1+e^{-z}} \\ g_{\text{tanh}}(z) &= \tanh(z) = \frac{1-e^{-2x}}{1+e^{-2x}} \\ g_{\text{ReLU}}(z) &= \max(z, 0) \end{aligned} \tag{2}$$

Notice all of these functions have simple derivatives, and specifically logistic and hyperbolic tangent functions map to bounded ranges, which are great choices for classification problems.

Graphically, this can be represented by a series of input nodes $\{x_i\}$ connected to an output node f, with weights $\{w_i\}$ on the connections.

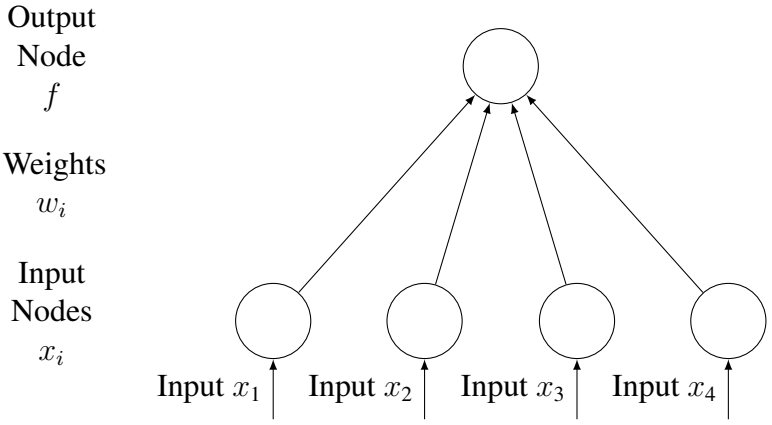

Figure 1: A generalized linear model represented in graphical form. In a neural network, this is also referred to as a single neuron.

A general feed-forward neural network is defined by recursive GLMs with different weights. For example, a neural network with two hidden layers (three layers of recursion) is defined as:

$$\begin{aligned} h_j^{(1)} &= g^{(1)}\left(\sum_{i=1}^{N^{(1)}} w_{ij}^{(1)} x_i\right) \\ h_k^{(2)} &= g^{(2)}\left(\sum_{j=1}^{N^{(2)}} w_{jk}^{(2)} h_j^{(1)}\right) \\ f_l &= g^{(3)}\left(\sum_{k=1}^{N^{(3)}} w_{kl}^{(3)} h_k^{(2)}\right) \end{aligned} \tag{3}$$

where $g(\cdot)^{(\alpha)}$ is some activation function, $h_j^{(\alpha)}$ denotes the j^{th} node of the α^{th} hidden layer, $w_{ij}^{(\alpha)}$ denotes the weight for the connection of the i^{th} node of the α^{th} layer to the j^{th} node of the $(\alpha+1)^{\text{th}}$ layer, and $N^{(\alpha)}$ denotes the number of nodes in the α^{th} layer.

Graphically, this structure has a very clear representation:

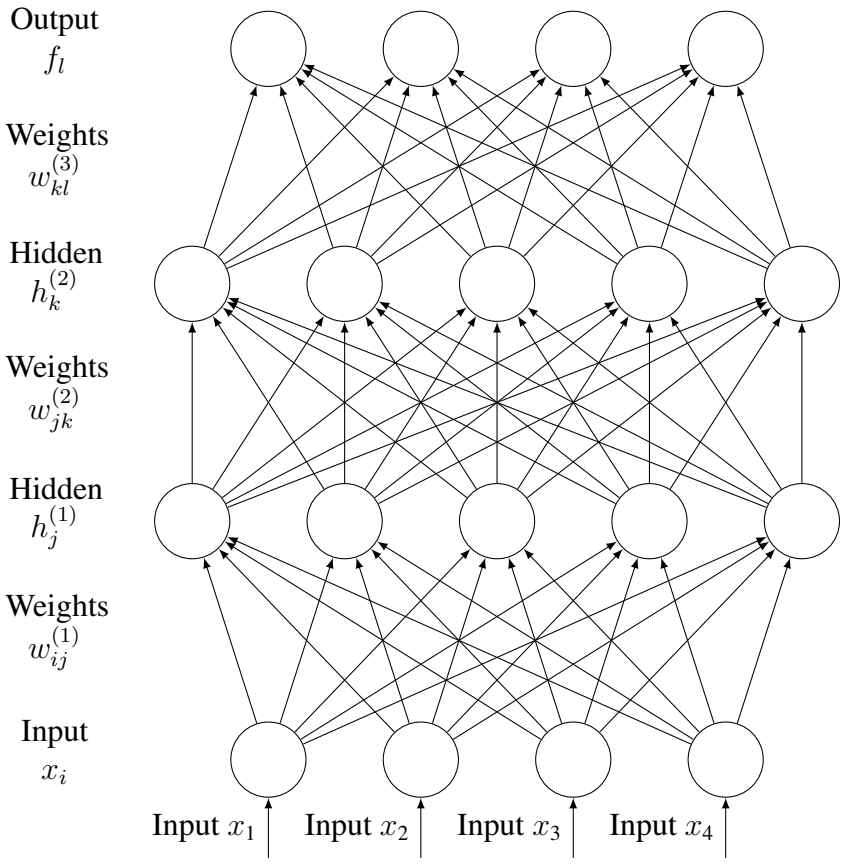

Figure 2: A generalized feed-forward neural network with two hidden layers.

While most GLMs do not admit a closed-form solution, a satisfactory optimization can be achieved by the gradient descent method. In the neural network case, the optimization becomes more difficult as the number of parameters increase with the number of nodes and layers. However, we can still apply the gradient descent method and find a local optimum.

Suppose we have a dataset $\mathcal{D} = \{\mathbf{x}^{[n]}, \mathbf{y}^{[n]}\}, n \in \mathbb{N}$, and we want to find a model $f(\mathbf{x}, \mathbf{w})$ such that it is the "closest" to $\mathbf{y}$. Given a differentiable error function $E(f, \mathbf{y})$ and model $f(\mathbf{x}, \mathbf{w})$ with respect to $\mathbf{w}$, the model can be optimized by gradient descent. In other words, for any randomly initialized $\mathbf{w}^0$, an improvement can be obtained by taking a step towards the direction of the gradient with respect to $\mathbf{w}^k$:

$$\mathbf{w}^{k+1} = \mathbf{w}^k - \eta \, \nabla_{\mathbf{w}^k} E\left(f(\mathbf{x}, \mathbf{w}^k), \mathbf{y}\right) \tag{4}$$

where η is a constant controlling the size of each optimization step, commonly called the learning rate.

In the the two hidden layer neural network previously, a derivative with respect to any $w_{ij}^{(\alpha)}$ can be found through chain rule of the derivatives. For example the derivative with respect to $w_{jk}^{(2)}$:

$$\begin{aligned} \text{let } z_j^{(\alpha)} &= \sum_{i=1}^{N^{(\alpha)}} w_{ij}^{(\alpha)} h_i^{(\alpha-1)} \\ \text{then } \frac{\partial E}{\partial w_{jk}^{(2)}} &= \sum_{l=1}^{N^{(4)}} \frac{\partial E}{\partial f_l} \frac{\partial f_l}{\partial z_l^{(3)}} \frac{\partial z_l^{(3)}}{\partial h_k^{(2)}} \frac{\partial h_k^{(2)}}{\partial z_k^{(2)}} \frac{\partial z_k^{(2)}}{\partial w_{jk}^{(2)}} \\ &= \sum_{l=1}^{N^{(4)}} \frac{\partial E}{\partial f_l} \frac{\partial g^{(3)}(z_l^{(3)})}{\partial z_l^{(3)}} w_{kl}^{(3)} \frac{\partial g^{(2)}(z_k^{(2)})}{\partial z_k^{(2)}} h_j^{(1)}. \end{aligned} \tag{5}$$

Recall $g^{(\alpha)}(\cdot)$ is selected to have a simple derivative, making the complex appearing expression above easy to compute.

A common technique to improve speed of convergence is by adding momentum. The idea is to let the gradient dictate the acceleration of the change in $\mathbf{w}^k$ instead of the velocity. This allows the optimization to move faster in the direction of the gradient, while making it harder to converge to a local minimum. The formulation starts with a velocity vector $\mathbf{v}^0$ initialized to zero, and the rest in similar:

$$\begin{aligned} \mathbf{v}^{k+1} &= \mathbf{v}^k - \eta \, \nabla_{\mathbf{w}^k} E\left(f(\mathbf{x}, \mathbf{w}^k), \mathbf{y}\right) \\ \mathbf{w}^{k+1} &= (1-\theta)\mathbf{w}^k + \theta \mathbf{v}^{k+1} \end{aligned} \tag{6}$$

where $\theta \in [0, 1]$ is the momentum constant.

MNIST Hand-Written Digits

The Mixed National Institute of Standards and Technology (MNIST) dataset is a collection of images of hand-written digits from various sources, with each image labeled the correct digit. The dataset contains 60,000 images for training (fitting), and 10,000 images for testing. The images are 28x28 in resolution, hence making 784 dimensions in input.

The data labels are changed to use the 1-of-K encoding scheme, where the label $\mathbf{y}$ is a binary vector of size K, with only one element take a value of one. In this case, given 10 possible digits, we have a vector

of size 10. To best model this type of vector, the softmax function is chosen for the output layer:

$$f_l = g^{(3)}(z_l^{(3)}) = \frac{\exp(z_l^{(3)})}{\sum_{k=1}^{N^{(4)}} \exp(z_k^{(3)})} \tag{7}$$

As a result the sum of f_l adds up to one. If f_l is modeled as the probability of the image being digit l, we can define the error function as negative log-likelihood:

$$E(\mathbf{f}, \mathbf{y}) = -\log \prod_{l=1}^{N^{(4)}} {f_l}^{y_l} = -\sum_{l=1}^{N^{(4)}} y_l \log f_l \tag{8}$$

where minimizing E is equivalent to maximizing likelihood.