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:

$$f(\mathbf{x}, \mathbf{w}) = g\left(\sum_{i=1}^{N} w_i x_i + w_0\right) \tag{1}$$

Here $g(\cdot)$ is a non-linear function, with $\mathbf{x}$ is an N dimensional input vector, and $\mathbf{w}$ is the weight vector, which includes w_0 as the bias. Common choices of $g(\cdot)$, also known as activation functions, for neural networks include the logistic (sigmoid) function, the hyperbolic tangent function, and the rectified linear unit (ReLU):

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

where z denotes the linear combination $\left(\sum_{i=1}^{N} w_i x_i + w_0\right)$ for GLMs. Notice all of these functions have simple derivatives, and specifically logistic and hyperbolic tangent functions are monotonic and bounded by their horizontal asymptotes at infinities, which makes them great choices for binary 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. Note bias is omitted from the graph but remains a parameter.

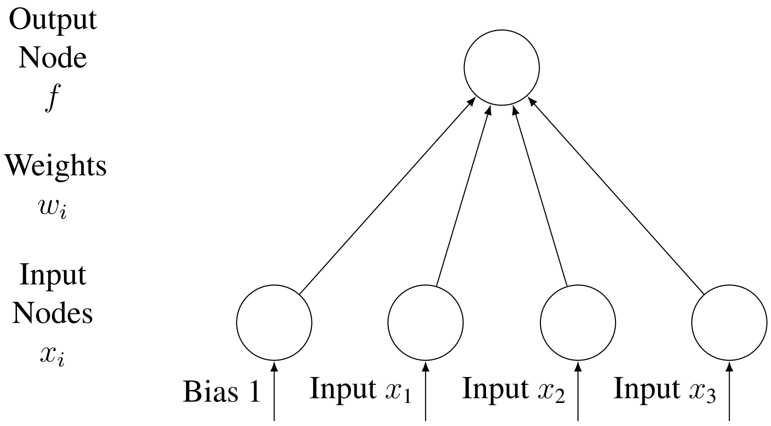


Figure 1: A generalized linear model represented in graphical form. In a neural network, this is also referred to as a single neuron.

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:

$$\begin{aligned} h_j^{(1)} &= g^{(1)}\left(\sum_{i=1}^{N^{(1)}} w_{ij}^{(1)} x_i + w_{0j}^{(1)}\right) \\ h_k^{(2)} &= g^{(2)}\left(\sum_{j=1}^{N^{(2)}} w_{jk}^{(2)} h_j^{(1)} + w_{0k}^{(2)}\right) \\ f_l &= g^{(3)}\left(\sum_{k=1}^{N^{(3)}} w_{kl}^{(3)} h_k^{(2)} + w_{0l}^{(3)}\right) \end{aligned} \tag{3}$$

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. Additionally, let $N^{(4)}$ be the number of output nodes f_l, and $\mathbf{w} = \left[\mathbf{w}^{(1)}\mathbf{w}^{(2)}\mathbf{w}^{(3)}\right]$. Here we also note that $N^{(1)}$ is the number of input nodes.

Graphically, this structure has a very clear representation:

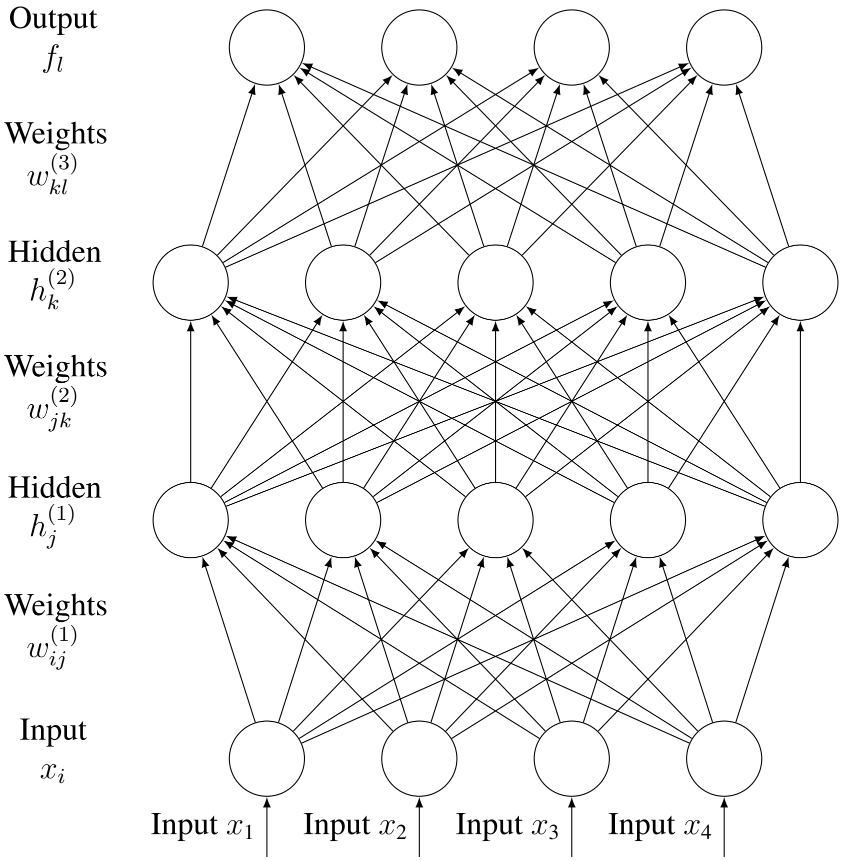


Figure 2: A generalized feed-forward neural network with two hidden layers. Bias parameters are not drawn for compactness, although they are present in all forward passing nodes.

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 for the simpler neural networks.

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 $\mathbf{y}$. If the error function $E(f, \mathbf{y})$ and the model $f(\mathbf{x}, \mathbf{w})$ are differentiable 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 $\mathbf{w}^{k+1}$ can be obtained by making a small modification in 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 $\eta > 0$ is hyper-parameter controlling the change of each optimization iteration, commonly called the learning rate. Note η is not part of the final model $f(\mathbf{x}, \mathbf{w})$, but it will significantly influence optimization.

In the two hidden layer neural network previously, a derivative with respect to any weight $w_{ij}^{(\alpha)}$ can be found by applying the chain rule to the derivatives. For example the derivative with respect to $w_{jk}^{(2)}$ where $j \neq 0$:

$$\begin{aligned} \text{let } z_j^{(\alpha)} &= \sum_{i=1}^{N^{(\alpha)}} w_{ij}^{(\alpha)} h_i^{(\alpha-1)} + w_{0j}^{(\alpha)} \\ \text{then } \frac{\partial E}{\partial w_{jk}^{(2)}} &= \sum_{l=1}^{N^{(4)}} \frac{\partial E}{\partial f_l} \frac{\partial f_l}{\partial z_l^{(3)}} \frac{\partial z_l^{(3)}}{\partial h_k^{(2)}} \frac{\partial h_k^{(2)}}{\partial z_k^{(2)}} \frac{\partial z_k^{(2)}}{\partial w_{jk}^{(2)}} \\ &= \sum_{l=1}^{N^{(4)}} \frac{\partial E}{\partial f_l} \frac{\partial g^{(3)}(z_l^{(3)})}{\partial z_l^{(3)}} w_{kl}^{(3)} \frac{\partial g^{(2)}(z_k^{(2)})}{\partial z_k^{(2)}} h_j^{(1)}. \end{aligned} \tag{5}$$

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

A common technique to improve speed of convergence is by adding momentum. Instead of letting the gradient dictate the change in $\mathbf{w}^k$, the idea is to let the gradient dictate the rate of change. If $\mathbf{w}^k$ is interpreted as a coordinate, and each optimization iteration as velocity, momentum can be seen as using the gradient as acceleration instead of velocity. This allows the optimization to accumulate speed in a consistent direction of the gradient, while making it harder to slow down and converge to a poor local minimum. The formulation starts with a velocity vector $\mathbf{v}^0$ initialized to zero, and the rest in similar:

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

where $\theta \in [0, 1]$ is the hyper-parameter deciding the preservation of momentum. Here choosing a larger θ would result in a stronger preservation of the velocity vector $\mathbf{v}$, which then retains more momentum.

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 $N^{(1)} = 784$ 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 taking a value of one. In this case, given 10 possible digits, we have a vector of size $N^{(4)} = 10$. For example, a possible scheme can label the digit "3" using the vector $[0, 0, 1, 0, \ldots]$ where only the 3rd index is a "1".

To best model this type of label 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, and the most optimal output is exactly the 1-of-K encoded label. If f_l is modeled as the probability of the image being digit l, suppose the correct digit is m, then the likelihood of making the correct prediction is:

$$L(\mathbf{f}, \mathbf{y}) = f_m = \prod_{l=1}^{N^{(4)}} {f_l}^{y_l} \tag{8}$$

since $y_m = 1$ is the only non-zero term in the label vector. We can then define the error function as negative log-likelihood:

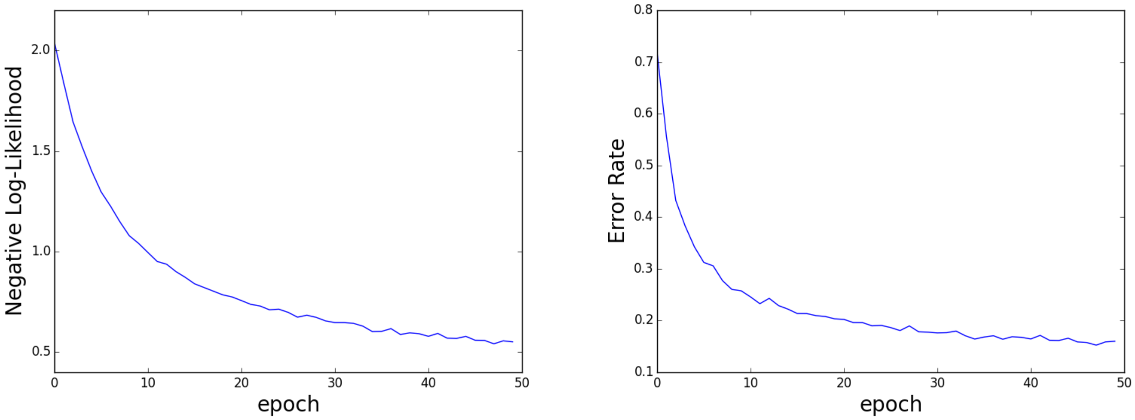
$$E(\mathbf{f}, \mathbf{y}) = -\log \prod_{l=1}^{N^{(4)}} {f_l}^{y_l} = -\sum_{l=1}^{N^{(4)}} y_l \log f_l \tag{9}$$

where $N^{(4)}$ is the number of output nodes, and minimizing E is equivalent to maximizing likelihood. Note taking the logarithm creates an error function with much simpler derivative, hence simplifying the gradient descent method.

In the following experiment, a two hidden layer neural network is used to model the MNIST digits. We used $N^{(2)} = N^{(3)} = 1000$ nodes in the hidden layers, creating a structure of 784-1000-1000-10 $\big(N^{(1)} - N^{(2)} - N^{(3)} - N^{(4)}\big)$ nodes in each layer. We also chose $g^{(1)}(\cdot) = g^{(2)}(\cdot) = g_{\text{ReLU}}(\cdot)$ in the hidden layers, and softmax for the output layer. The hyper parameters were chosen as $\eta = 10^{-5}$ and $\theta = 0.9$. We also chose to update the weight vector $\mathbf{w}^k$ once for every 100 samples of digits, also known as a mini-batch.

After training (optimizing) for 50 epochs, with each epoch denoting one complete run through of the training dataset, we reach a test error rate of 16% (Figure 3b). By increasing the number of epochs and a few minor modifications, this model could potentially reach error rates as low as 2%. However, this will not be explored since it is not the main purpose of this document, and optimization can be very time consuming due to computation.

While the technique is more than a sufficient solution for recognizing hand-written digits, naively applying gradient descent to more complex neural networks tend to have poor results. Alternative methods will be discussed in the next section in order to address this problem.



(a) The negative log-likelihood

(b) The classification error rate

Figure 3: MNIST hand-written digits modeled using a two hidden layer neural network, with negative log-likelihood and classification error rate computed after each epoch.

Also note this type of neural networks is feed-forward, which mean it is limited to only supervised type problems where the data structure is consistent and a prediction target (label) is provided for each sample. For a collaborative filtering type problem, the inference is often made within the data structure itself, which makes an unsupervised learning problem. Feed-forward neural networks also fail to fully utilize the datasets that are partly labeled, known as semi-supervised problems. These problems would require other variations of neural networks with different methods for inference.

Restricted Boltzmann Machines

Motivated by optimization of a more complex neural network and prediction of an unsupervised or semi-supervised learning problem such as collaborative filtering, it turns out that restricted Boltzmann machines can address all of these problems. A RBM is a Markov random field in a bipartite graph structure, where there are two layers of nodes, no internal connections within each layer. One layer of nodes contain the input $\mathbf{x}$, while the other contain hidden values $\mathbf{h}$.

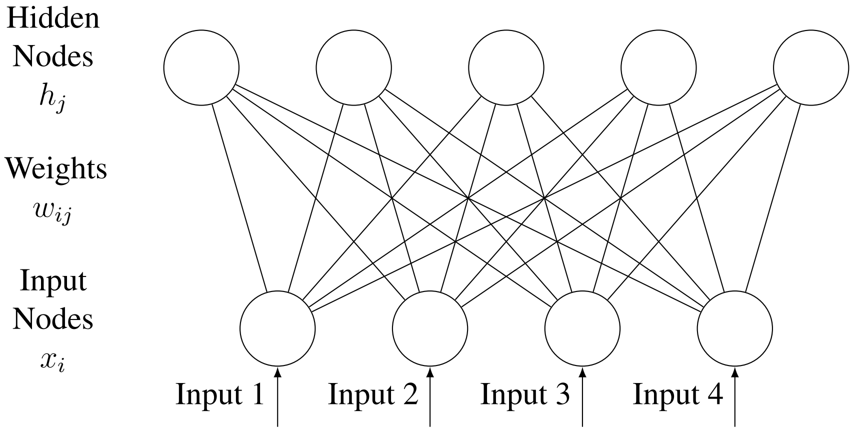


Figure 4: A restricted Boltzmann machine (RBM) with 4 input and 5 hidden nodes.

Suppose the graph have $N^{(1)}$ input nodes and $N^{(2)}$ hidden nodes, with weights w_{ij} defined in the same way as the neural network the bias parameters w_{0j} and additionally w_{i0}. Here we consider the case where h_j can only take binary values, and σ_i as the standard deviation for each input dimension. Finally, we can define an energy function and joint Boltzmann distribution for the graph:

$$\begin{aligned} E(\mathbf{x}, \mathbf{h}|\theta) &= \sum_{i=1}^{N} \frac{(w_{i0} - x_i)^2}{2\sigma_i} - \sum_{i=1}^{N} \sum_{j=1}^{M} w_{ij} h_j \frac{x_i}{\sigma_i} - \sum_{i=j}^{M} w_{0j} h_j \\ P(\mathbf{x}, \mathbf{h}|\theta) &= \frac{\exp\left[-E(\mathbf{x}, \mathbf{h}|\theta)\right]}{\mathcal{Z}} \end{aligned} \tag{10}$$