UNIVERSITY of DELAWARE

Modern Machine Learning — Neural Networks & Backpropagation

Kenneth E. Barner
Department of Electrical and Computer Engineering
University of Delaware

Artificial neural networks are biologically-inspired networks of highly interconnected (simple) processing elements (neurons)

Optimization is performed through steps analogous to adjusting synaptic connections between neurons — adaptive feedback learning

Backpropagation is utilized in conjunction with an optimization technique, such as gradient descent

Optimization consists of two stages: propagation and weight update

The effect of an input is propagated forward through the network, the error determined, and the error is propagated back through the network

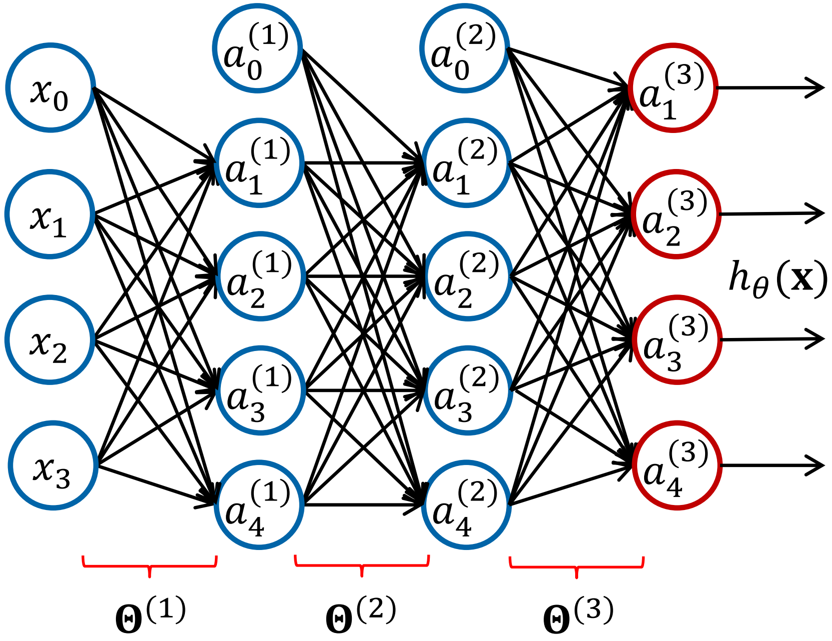
Weights are adjusted according to gradient descent and the back propagated error

Regularization, or shrinkage, can be applied to the optimization in order to minimize overfitting and reduce the number of non-zero weights

Neural Networks & Backpropagation

Methodology: Establish a cost function $J(\boldsymbol{\Theta})$ to be minimized as a function of $\boldsymbol{\Theta}$ by gradient descent, called backpropagation in the NN setting

Observations: The gradient is easily derived using the chain rule because of the NN structure



Gradient Computation

The gradient can be determined via the chain rule by a forward and backward sweep over the network

Assume a neural network consisting of L layers, and a training example $(\mathbf{x}, \mathbf{y})$ consisting of observation $\mathbf{x}$ and its corresponding label $\mathbf{y}$

The forward propagation equations are given by

$$
\begin{aligned}
\boldsymbol{z}^{(1)} &= \boldsymbol{\Theta}^{(1)}\mathbf{x} \\
\boldsymbol{a}^{(1)} &= g\left(\boldsymbol{z}^{(1)}\right) \\
\boldsymbol{z}^{(2)} &= \boldsymbol{\Theta}^{(2)}\boldsymbol{a}^{(1)} \\
\boldsymbol{a}^{(2)} &= g\left(\boldsymbol{z}^{(2)}\right) \\
&\vdots \\
\boldsymbol{z}^{(L)} &= \boldsymbol{\Theta}^{(L)}\boldsymbol{a}^{(L-1)} \\
\boldsymbol{a}^{(L)} &= h_\theta(\mathbf{x}) = g\left(\boldsymbol{z}^{(L)}\right)
\end{aligned}
$$

where

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

is the logistic function

Define the cost function $J(\mathbf{\Theta})$ as the squared output error:

$$J(\mathbf{\Theta}) = \frac{1}{2}\left\|\mathbf{y} - \boldsymbol{a}^{(L)}\right\|^2 = \frac{1}{2}\sum_i \left(a_i^L - y_i\right)^2$$

Using the chain rule we can express the error for layer L as

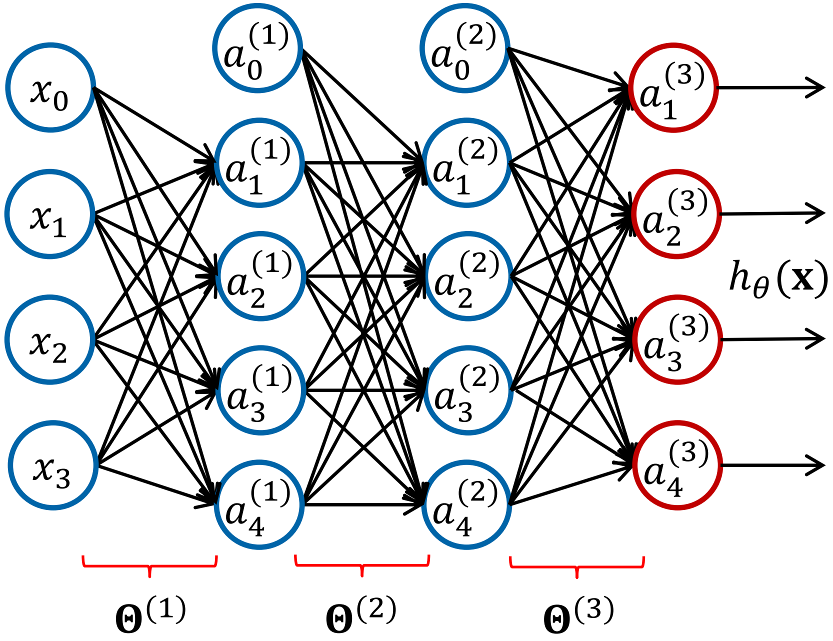
$$\frac{\partial J(\mathbf{\Theta})}{\partial \theta_{ij}^{(L)}} = \frac{\partial J(\mathbf{\Theta})}{\partial a_i^{(L)}} \frac{\partial a_i^{(L)}}{\partial z_i^{(L)}} \frac{\partial z_i^{(L)}}{\partial \theta_{ij}^{(L)}}$$

Observations:

$\frac{\partial J(\mathbf{\Theta})}{\partial a_i^{(L)}}$ — cost function derivative with respect to NN output

$\frac{\partial a_i^{(L)}}{\partial z_i^{(L)}}$ — NN output derivative with respect to output node inputs

$\frac{\partial z_i^{(L)}}{\partial \theta_{ij}^{(L)}}$ — output node input derivative with respect to weights (connecting outputs from prior layer nodes)



Neural Networks & Backpropagation

To determine

$$\frac{\partial J(\boldsymbol{\Theta})}{\partial \theta_{ij}^{(L)}} = \frac{\partial J(\boldsymbol{\Theta})}{\partial a_i^{(L)}} \frac{\partial a_i^{(L)}}{\partial z_i^{(L)}} \frac{\partial z_i^{(L)}}{\partial \theta_{ij}^{(L)}}$$

define define the error

$$\delta_i^{(L)} = \frac{\partial J(\boldsymbol{\Theta})}{\partial a_i^{(L)}} \frac{\partial a_i^{(L)}}{\partial z_i^{(L)}}$$

and solve for each derivative component

Since

$$J(\boldsymbol{\Theta}) = \frac{1}{2} \sum_j \left(a_j^L - y_j\right)^2$$

we have

$$\frac{\partial J(\boldsymbol{\Theta})}{\partial a_i^{(L)}} = \left(a_i^{(L)} - y_i\right)$$

Similarly,

$$\boldsymbol{a}^{(L)} = g\left(\boldsymbol{z}^{(L)}\right) \Longrightarrow \frac{\partial a_i^{(L)}}{\partial z_i^{(L)}} = g'\left(z_i^{(L)}\right)$$

Note: backpropagation relies on the fact that the activation function, $g(\cdot)$, is differentiable

Therefore

$$\delta_i^{(L)} = \frac{\partial J(\boldsymbol{\Theta})}{\partial a_i^{(L)}} \frac{\partial a_i^{(L)}}{\partial z_i^{(L)}} = \left(a_i^{(L)} - y_i\right) g'\left(z_i^{(L)}\right)$$

Finally, $z_i^{(L)} = \sum_k \theta_{ik}^{(L)} a_k^{(L-1)} \Longrightarrow$

$$\frac{\partial z_i^{(L)}}{\partial \theta_{ij}^{(L)}} = \frac{\partial}{\partial \theta_{ij}^{(L)}} \left(\sum_k \theta_{ik}^{(L)} a_k^{(L-1)} \right) = a_j^{(L-1)}$$

Putting all the components together,

$$\frac{\partial J(\boldsymbol{\Theta})}{\partial \theta_{ij}^{(L)}} = \frac{\partial J(\boldsymbol{\Theta})}{\partial a_i^{(L)}} \frac{\partial a_i^{(L)}}{\partial z_i^{(L)}} \frac{\partial z_i^{(L)}}{\partial \theta_{ij}^{(L)}} = \delta_i^{(L)} a_j^{(L-1)}$$

Note that this can be generalized to arbitrary level $l = 1, \ldots, L$

$$\frac{\partial J(\boldsymbol{\Theta})}{\partial \theta_{ij}^{(l)}} = \delta_i^{(l)} a_j^{(l-1)}$$

where it is understood $a_j^{(0)} = x_j$ (input sample)

Consider the error term $\delta_i^{(L-1)}$, first without the chain rule:

$$\delta_i^{(L-1)} = \frac{\partial J(\boldsymbol{\Theta})}{\partial z_i^{(L-1)}}$$

To utilize the chain rule in this case, all paths between $z_i^{(L-1)}$ and the output terms must be considered

$$\delta_i^{(L-1)} = \sum_j \left(\frac{\partial J(\boldsymbol{\Theta})}{\partial a_j^{(L)}} \frac{\partial a_j^{(L)}}{\partial z_j^{(L)}} \right) \left(\frac{\partial z_j^{(L)}}{\partial a_i^{(L-1)}} \right) \left(\frac{\partial a_i^{(L-1)}}{\partial z_i^{(L-1)}} \right)$$

The first and third terms have previously determined results:

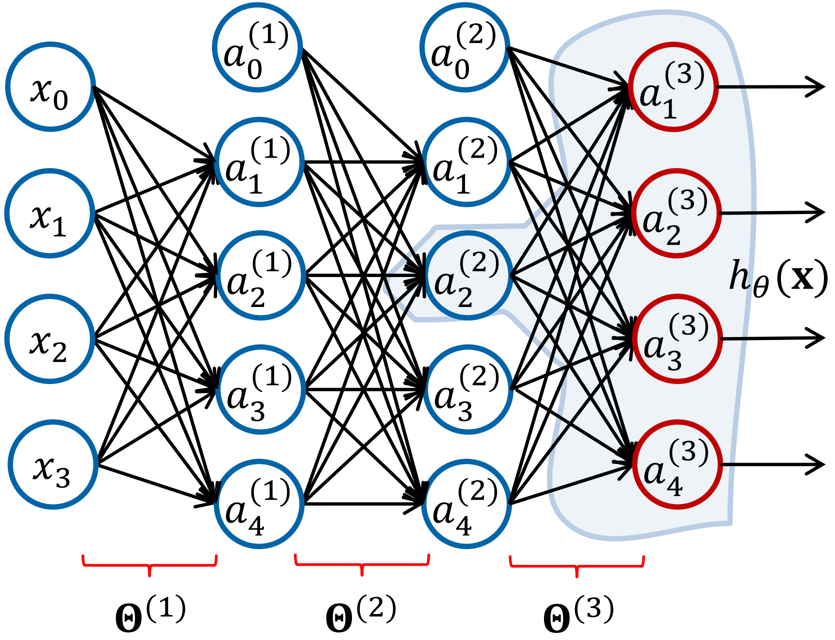
$$\delta_i^{(L-1)} = \sum_j \delta_j^{(L)} \left(\frac{\partial z_j^{(L)}}{\partial a_i^{(L-1)}} \right) g'\left(z_i^{(L-1)} \right)$$

Now consider the middle term:

$$\frac{\partial z_j^{(L)}}{\partial a_i^{(L-1)}} = \frac{\partial}{\partial a_i^{(L-1)}} \left(\sum_n \theta_{jn}^{(L)} a_n^{(L-1)} \right) = \theta_{ji}^{(L)}$$

Thus

$$\delta_i^{(L-1)} = \left(\sum_j \delta_j^{(L)} \theta_{ji}^{(L)} \right) g'\left(z_i^{(L-1)} \right)$$



The result

$$\delta_i^{(L-1)} = \left(\sum_j \delta_j^{(L)} \theta_{ji}^{(L)}\right) g'\left(z_i^{(L-1)}\right)$$

can be generalized as:

$$\delta_i^{(l)} = \left(\sum_j \theta_{ji}^{(l+1)} \delta_j^{(l+1)}\right) g'\left(z_i^{(l)}\right), \qquad l = 1, \dots, L-1$$

With the result then utilized in the cost function derivative with respect to arbitrary weight $\theta_{ij}^{(l)}$:

$$\frac{\partial J(\boldsymbol{\Theta})}{\partial \theta_{ij}^{(l)}} = \delta_i^{(l)} a_j^{(l-1)}$$

Observation: We back propagate the error $\delta^{(l+1)}$ from layer $l+1$ to layer l

It is generally more convenient to express the operations in matrix form:

$$\boldsymbol{\delta}^{(L)} = \left(\boldsymbol{a}^{(L)} - \mathbf{y}\right) \odot g'\left(\boldsymbol{z}^{(L)}\right),$$

and

$$\boldsymbol{\delta}^{(l)} = \left(\left(\boldsymbol{\Theta}^{(l+1)}\right)^T \boldsymbol{\delta}^{(l+1)}\right) \odot g'\left(\boldsymbol{z}^{(l)}\right)$$

for $l = 1, \dots, L-1$.

In this formulation, in this formulation, $\odot$ denotes component-wise multiplication

In the particular case when $g(\boldsymbol{z})$ is the logistic function,

$$\boldsymbol{a}^{(l)} = g\left(\boldsymbol{z}^{(l)}\right) = \frac{1}{1 + e^{-\boldsymbol{z}^{(l)}}},$$

we can express

$$g'\left(\boldsymbol{z}^{(l)}\right) = \boldsymbol{a}^{(l)} \odot \left(1 - \boldsymbol{a}^{(l)}\right)$$

To implement backpropagation, suppose we have a training set of the form

$$\left\{\left(\mathbf{x}^{(1)},\mathbf{y}^{(1)}\right),\left(\mathbf{x}^{(2)},\mathbf{y}^{(2)}\right),\ldots,\left(\mathbf{x}^{(n)},\mathbf{y}^{(n)}\right)\right\}$$

The error is thus defined as

$$J(\mathbf{\Theta})=\sum_{k=1}^{n}\frac{1}{2}\left\|\mathbf{y}^{(k)}-\boldsymbol{a}^{(k)(L)}\right\|^{2}$$

and the derivative is

$$\frac{\partial J(\mathbf{\Theta})}{\partial\theta_{ij}^{(l)}}=\sum_{k=1}^{n}\delta_i^{(k)(l)}a_j^{(k)(l-1)},$$

Note that the superscript (k) denotes that the quantity was computed using the training sample pair: $\left(\mathbf{x}^{(k)},\mathbf{y}^{(\mathrm{k})}\right)$

Indexing according to the training samples and utilizing vector representation:

$$\frac{\partial J(\boldsymbol{\Theta})}{\partial \boldsymbol{\Theta}^{(l)}} = \sum_{k=1}^{n} \boldsymbol{\delta}^{(k)(l)} \left(\boldsymbol{a}^{(k)(l-1)}\right)^T$$

Putting everything together yields the following backpropagation algorithm, where step #4 is the standard gradient descent update (other update methodologies can be utilized, given the gradient)

Backpropagation algorithm:

Suppose we have a training set of the form $\{(\mathbf{x}^{(1)}, \mathbf{y}^{(1)}), (\mathbf{x}^{(2)}, \mathbf{y}^{(2)}), \dots, (\mathbf{x}^{(n)}, \mathbf{y}^{(n)})\}$

1. Feedforward: compute $\boldsymbol{z}^{(k)(l)} = \boldsymbol{\Theta}^{(l)} \boldsymbol{a}^{(k)(l-1)}$ and $\boldsymbol{a}^{(k)(l)} = g\left(\boldsymbol{z}^{(k)(l)}\right)$
2. Feedback: Compute $\boldsymbol{\delta}^{(k)(L)} = \left(\boldsymbol{a}^{(k)(L)} - \mathbf{y}\right) \odot g'\left(\boldsymbol{z}^{(k)(L)}\right)$ and
$$\boldsymbol{\delta}^{(k)(l)} = \left(\left(\boldsymbol{\Theta}^{(l+1)}\right)^T \boldsymbol{\delta}^{(k)(l+1)} \right) \odot g'\left(\boldsymbol{z}^{(k)(l)}\right)$$
3. Compute $J(\boldsymbol{\Theta}) = \sum_{k=1}^{n} \frac{1}{2} \left\| \mathbf{y}^{(k)} - \boldsymbol{a}^{(k)(L)} \right\|^2$ and $\frac{\partial J(\boldsymbol{\Theta})}{\partial \boldsymbol{\Theta}^{(l)}} = \sum_{k=1}^{n} \boldsymbol{\delta}^{(k)(l)} \left(\boldsymbol{a}^{(k)(l-1)}\right)^T$
4. Update the weights: $\boldsymbol{\Theta}_{k+1}^{(l)} = \boldsymbol{\Theta}_{k}^{(l)} - \alpha \frac{\partial J(\boldsymbol{\Theta})}{\partial \boldsymbol{\Theta}^{(l)}}$

Observation: Regularization (shrinkage) can be included in the optimization to avoid over fitting and reduce the number of non-zero weights

For instance, including L_2 regularization yields:

$$J(\mathbf{\Theta}) = \sum_{k=1}^{n} \frac{1}{2} \left\| \mathbf{y}^{(k)} - \boldsymbol{a}^{(k)(L)} \right\|^2 + \frac{\lambda}{2} \sum_{l} \sum_{i} \sum_{j} \left(\theta_{ij}^{l}\right)^2$$

and the derivative becomes:

$$\frac{\partial J(\mathbf{\Theta})}{\partial \mathbf{\Theta}^{(l)}} = \sum_{k=1}^{n} \boldsymbol{\delta}^{(k)(l)} \left(\boldsymbol{a}^{(k)(l-1)}\right)^T + \lambda \mathbf{\Theta}^{(l)}$$

Summary:

- Neural networks can be optimized utilizing backpropagation in conjunction gradient descent
- Optimization consists of two stages: propagation and weight update
- The effect of an input is propagated forward through the network, the error determined, and the error is propagated back through the network
- Weights are adjusted according to gradient descent and the back propagated error
- Regularization, or shrinkage, can be applied to the optimization in order to minimize overfitting and reduce the number of non-zero weights