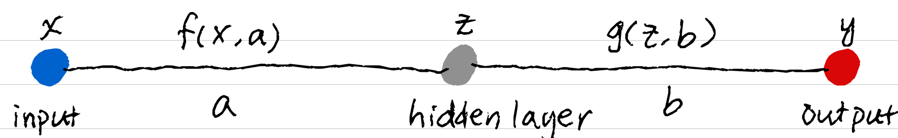
# ME 491 Lecture 27

## 6.3. The Backpropagation Algorithm

We need training data to determine the weights of the network, and we need an optimization routine and objective function to determine the weights.

Taking advantage of the compositional nature of NNs, backpropogation algorithm frame an optimization to determine the weights of the network. Specifically, it produces a formulation amenable to standard gradient descent.




$y = g(f(x,a), b)$

### * Formulation of backprop

Input-to-output relationship for single-node, one hidden-layer network is:

$$y = g(z,b) = g(f(x,a), b)$$

The output error computed against ground truth is

$$E = \frac{1}{2}(y_0 - y)^2$$

$y_0$: correct output

$y$: NN approximation

Goal: find parameter $a, b$ that minimize the error

$$\frac{\partial E}{\partial a} = -(y_0 - y)\frac{dy}{dz}\frac{dz}{da} = 0$$

Note: the terms $(dy/dz)(dz/da)$ show how the error backpropogates in the network.

Given functions $f(\cdot)$ and $g(\cdot)$, the chain rule can be explicitly computed.

Backprop results in an iterative, gradient descent update rule.

$$a_{k+1} = a_k - \delta \frac{\partial E}{\partial a_k}$$

$$b_{k+1} = b_k - \delta \frac{\partial E}{\partial b_k}$$

$\delta$: learning rate

General backprop algorithm

(i) an NN is specified along with a labeled training set

(ii) initialize weights of NN to random numbers.

(Note: it's important to have a random initialization to reduce chances of stuck in local minima)

(iii) run through the training data $x$ of the network to produce output $y$. compute the error based on ground truth, then compute the derivative of each weight using backprop.

(iv). for a given learning rate, update network weights

(v). repeat (iii) & (iv) till maximum iteration or convergence.

For a simple example, we consider a linear activation function

$$f(\xi, \alpha) = g(\xi, \alpha) = \alpha \xi$$

We have:

$$z = ax$$
$$y = bz$$

We can now explicitly compute the gradient:

$$\frac{\partial E}{\partial a} = -(y_0 - y)\frac{dy}{dz}\frac{dz}{da} = -(y_0 - y)\cdot b \cdot x$$

$$\frac{\partial E}{\partial b} = -(y_0 - y)\frac{dy}{db} = -(y_0 - y)z = -(y_0 - y)\cdot a \cdot x$$

With these two partial derivatives. we can write a gradient descent update rule:

$$a_{k+1} = a_k + \delta(y - y_0)\cdot b_k \cdot x$$

$$b_{k+1} = b_k + \delta(y - y_0)\cdot a_k \cdot x$$

For a network with M hidden layers labeled $z_1$ to $z_m$, with the first connection weight $a$ between $x$ and $z_1$, we have

$$\frac{\partial E}{\partial a} = -(y_0 - y)\frac{dy}{dz_m}\frac{dz_m}{dz_{m-1}} \cdots \frac{dz_2}{dz_1}\frac{dz_1}{da}$$

General update rule for multi-layer & multi-dimension NN.

$$W_{k+1} = W_k - \delta \nabla E$$

$$W_{k+1}^j = W_k^j - \delta \frac{\partial E}{\partial W_k^j} \quad (W_k^j : j^{th} \text{ component of vector } W_k)$$

## 6.4 The Stochastic Gradient Descent Algorithm

Two critical algorithms for training NN:
(1). backprop (calculate gradient)
(2). stochastic gradient descent (rapid evaluation of optimal network weights)

Optimization set up for NN:

$$f(x) = f(x, A_1, A_2, \ldots, A_M)$$

$A_j$: connectivity matrices from one layer to next
$A_1$: connects $1^{st}$ & $2^{nd}$ layer
M hidden layers

Goal: minimize error between the network and data.

Standard root-mean-square error:

$$\underset{A_j}{\operatorname{argmin}} E(A_1, A_2, \ldots, A_M) = \underset{A_j}{\operatorname{argmin}} \sum_{k=1}^{n} (f(x_k, A_1, A_2, \ldots, A_M) - y_k)^2$$

minimizing the error by setting $\partial E / \partial (a_{ij})_k = 0$

$(a_{ij})_k$: $i^{th}$ row & $j^{th}$ column of the $k^{th}$ matrix

Recall the gradient descent algorithm from 4.2.

$$x_{j+1}(\delta) = x_j - \delta \nabla f(x_j)$$

$\delta$ : learning rate in NN language

Though the gradients are not hard to compute, calculating all N gradients can be computationally heavy, so we need to use SGD instead.