Intelligent Systems: Reasoning and Recognition

James L. Crowley

MoSIG M1 Winter Semester 2021
Lesson 8 4 March 2021

Artificial Neural Networks

Outline

Notation

x_d	A feature. An observed or measured value.
$\vec{X}$	A vector of D features.
D	The number of dimensions for the vector $\vec{X}$
$\{\vec{x}_m\}$ $\{y_m\}$	Training samples for learning.
M	The number of training samples.
$a_j^{(l)}$	the activation output of the j^{th} neuron of the l^{th} layer.
$w_{ij}^{(l)}$	the weight from unit i of layer l–1 to the unit j of layer l.
b_j^l	bias for unit j of layer l.
η	A learning rate. Typically very small (0.01). Can be variable.
L	The number of layers in the network.
$\delta_m^{out} = \left(a_m^{(L)} - y_m\right)$	Output Error of the network for the m^{th} training sample
$\delta_{j,m}^{(l)}$	Error for the j^{th} neuron of layer l, for the m^{th} training sample.
$\Delta w_{ij,m}^{(l)} = a_i^{(l-1)}\delta_{j,m}^{(l)}$	Update for weight from unit i of layer l–1 to the unit j of layer l.
$\Delta b_{j,m}^{(l)} = \delta_{j,m}^{(l)}$	Update for bias for unit j of layer l.

Introduction

Key Equations

Feed Forward from Layer i to j:

$$a_j^{(l)} = f\left(\sum_{i=1}^{N^{(l-1)}} w_{ij}^{(l)} a_i^{(l-1)} + b_j^{(l)}\right)$$

Feed Forward from Layer j to k:

$$a_k^{(l+1)} = f\left(\sum_{j=1}^{N^{(l)}} w_{jk}^{(l+1)} a_j^{(l)} + b_k^{(l+1)}\right)$$

Back Propagation from Layer j to i:

$$\delta_{i,m}^{(l-1)} = \frac{\partial f(z_i^{(l-1)})}{\partial z_i^{(l-1)}} \sum_{j=1}^{N^{(l)}} w_{ij}^{(l)} \delta_{j,m}^{(l)}$$

Back Propagation from Layer k to j:

$$\delta_{j,m}^{(l)} = \frac{\partial f(z_j^{(l)})}{\partial z_j^{(l)}} \sum_{k=1}^{N^{(l+1)}} w_{jk}^{(l+1)} \delta_{k,m}^{(l+1)}$$

Weight and Bias Corrections for layer j:

$$\Delta w_{ij,m}^{(l)} = a_i^{(l-1)} \delta_{j,m}^{(l)}$$

$$\Delta b_{j,m}^{(l)} = \delta_{j,m}^{(l)}$$

Network Update Formulas:

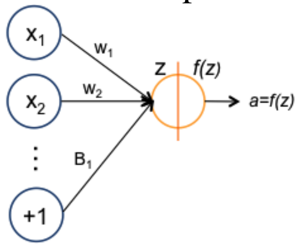
$$w_{ij}^{(l)} \leftarrow w_{ij}^{(l)} - \eta \cdot \Delta w_{ij,m}^{(l)}$$

$$b_j^{(l)} \leftarrow b_j^{(l)} - \eta \cdot \Delta b_{j,m}^{(l)}$$

Artificial Neural Networks

Artificial Neural Networks, also referred to as "Multi-layer Perceptrons", are computational structures composed a weighted sums of "neural" units. Each neural unit is composed of a weighted sum of input units, followed by a non-linear decision function.

The simplest possible neural network is composed of a single neuron.

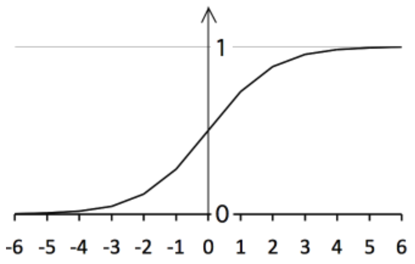


A "neuron" is a computational unit that integrates information from a vector of features, $\vec{X}$, to compute the likelihood of an activation, a. The neuron is composed of a weighted sum of input values $z = w_1x_1 + w_2x_2 + ... + w_Dx_D + b$ followed by a non-linear "activation" function, $f(z)$

$$a = f(\vec{w}^T\vec{X} + b)$$

Many different activation functions may be used. Historically, the classic activation function is the sigmoid (or Logistic) activation function:

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

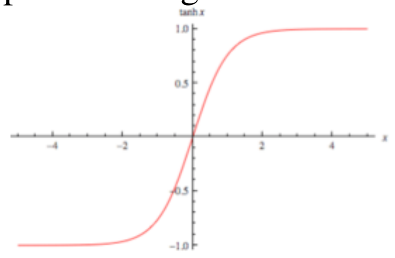


The sigmoid has long been used in biology and in economics to model processes that grow exponentially to a point of saturation. For example, the population of bacteria during fermentation, or the growth in performance of a new technology.

The sigmoid is useful because the derivative is: $\frac{d\sigma(z)}{dz} = \sigma(z)(1-\sigma(z))$

Another classic decision functions is the hyperbolic tangent:

$$f(z) = \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$



For multiple classes, we can use the Softmax activation function.

$$f(z_k) = \frac{e^{z_k}}{\sum_{k=1}^{K} e^{z_k}}$$

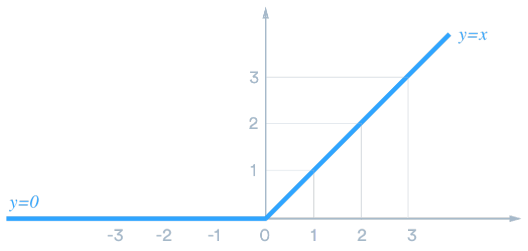
The softmax function takes as input a vector $\vec{z}$ of K real numbers, and normalizes it into a probability distribution consisting of K probabilities.

The softmax function is used to select the maximum from a vector of activations for K classes. Before applying softmax, the vector components of $\vec{z}$ will generally not sum to 1, and some of the components may be negative, or greater than one. After applying softmax, each component will be in the interval [0, 1] and the components will sum to 1. Thus the output can be interpreted as a probability distribution indicating the likelihood of each component.

Softmax is used as the last activation function of a neural network to normalize the output of a network to a probability distribution over predicted output classes.

The rectified linear function is popular for deep learning because of a trivial derivative:

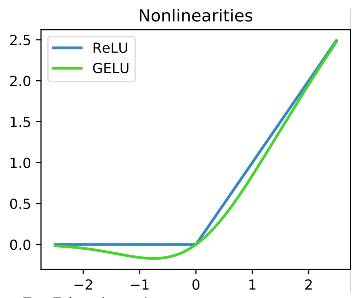
$$relu(z) = \max(0, z)$$



For $z \leq 0$ $\frac{d(relu(z))}{dz} = 0$ for $z > 0$: $\frac{d(relu(z))}{dz} = 1$

Recently, a variation of RELU called GELU (Gaussian Error Linear Unit) has gained popularity.

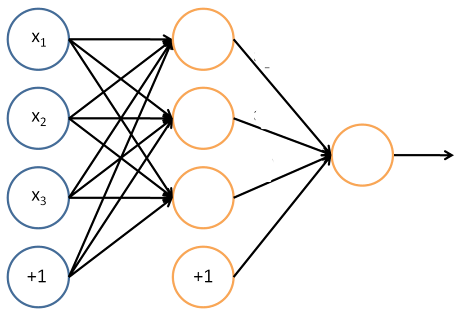
$$gelu(z) = 0.5z\left(1 + \frac{2}{\sqrt{\pi}} \int_0^{\frac{z}{\sqrt{2}}} e^{-x^2} dx\right)$$



From Wikipedia: By Ringdongdang - https://commons.wikimedia.org/w/index.php?curid=95947821

The Multilayer Neural Network model

A neural network is a multi-layer assembly of neurons. For example, this is a 2-layer network:



The circles labeled +1 are the bias terms.
The circles on the left are the input terms. Some authors, notably in the Stanford tutorials, refer to this as Level 1.

We will NOT refer to this as a level (or, if necessary, level L=0).
The rightmost circle is the output layer, also called L.
The circles in the middle are referred to as a "hidden layer". In this example there is a single hidden layer and the total number of layers is L=2.

The parameters carry a superscript, referring to their layer.
We will use the following notation:

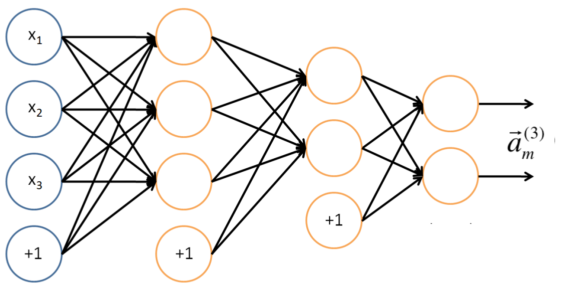
L	The number of layers (Layers of non-linear activations).
l	The layer index. l ranges from 0 (input layer) to L (output layer)
$N^{(l)}$	The number of units in layer l. $N^{(0)}=D$
$a_j^{(l)}$	The activation output of the jth neuron of the lth layer.
$w_{ij}^{(l)}$	The weight from the unit i of layer l-1 for the unit j of layer l.
$b_j^{(l)}$	The bias term for j^{th} unit of the l^{th} layer
$f(z)$	A non-linear activation function, such as a sigmoid, relu or tanh.

For example: $a_1^{(2)}$ is the activation output of the first neuron of the second layer.
$W_{13}^{(2)}$ is the weight for neuron 1 from the first level to neuron 3 in the second level.

The above network would be described by:

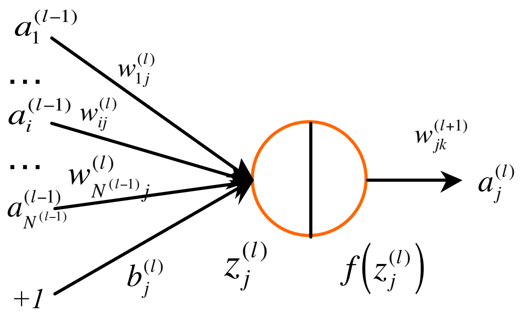
$$a_1^{(1)} = f(w_{11}^{(1)}X_1 + w_{21}^{(1)}X_2 + w_{31}^{(1)}X_3 + b_1^{(1)})$$
$$a_2^{(1)} = f(w_{12}^{(1)}X_1 + w_{22}^{(1)}X_2 + w_{32}^{(1)}X_3 + b_2^{(1)})$$
$$a_3^{(1)} = f(w_{13}^{(1)}X_1 + w_{23}^{(1)}X_2 + w_{33}^{(1)}X_3 + b_3^{(1)})$$
$$a_1^{(2)} = f(w_{11}^{(2)}a_1^{(1)} + w_{21}^{(2)}a_2^{(1)} + w_{31}^{(2)}a_3^{(1)} + b_1^{(2)})$$

This can be generalized to multiple layers. For example:



$\vec{a}_m^{(3)}$ is the vector of network outputs (one for each class) at the third layer.

Each unit is defined as follows:



The notation for a multi-layer network is

$\vec{a}^{(0)} = \vec{X}$ is the input layer. $a_i^{(0)} = X_d$

l is the current layer under discussion.

$N^{(l)}$ is the number of activation units in layer l. $N^{(0)} = D$

i,j,k Unit indices for layers l-1, l and l+1: $i \rightarrow j \rightarrow k$

$w_{ij}^{(l)}$ is the weight for the unit i of layer l-1 feeding to unit j of layer l.

$a_j^{(l)}$ is the activation output of the jth unit of the layer l

$b_j^{(l)}$ the bias term feeding to unit j of layer l.

$z_j^{(l)} = \sum_{i=1}^{N^{(l-1)}} w_{ij}^{(l)} a_i^{(l-1)} + b_j^{(l)}$ is the weighted input to j^{th} unit of layer l

$f(z)$ is a non-linear decision function, such as a sigmoid, tanh(), or soft-max

$a_j^{(l)} = f(z_j^{(l)})$ is the activation output for the j^{th} unit of layer l

For layer l this gives:

$$z_j^{(l)} = \sum_{i=1}^{N^{(l-1)}} w_{ij}^{(l)} a_i^{(l-1)} + b_j^{(l)} \qquad a_j^{(l)} = f\left(\sum_{i=1}^{N^{(l-1)}} w_{ij}^{(l)} a_i^{(l-1)} + b_j^{(l)}\right)$$

and then for l+1 :

$$z_k^{(l+1)} = \sum_{j=1}^{N^{(l)}} w_{jk}^{(l+1)} a_j^{(l)} + b_k^{(l+1)} \qquad a_k^{(l+1)} = f\left(\sum_{j=1}^{N^{(l)}} w_{jk}^{(l+1)} a_j^{(l)} + b_k^{(l+1)}\right)$$

It can be more convenient to represent the network using vectors:

$$\vec{z}^{(l)} = \begin{bmatrix} z_1^{(l)} \\ z_2^{(l)} \\ \vdots \\ z_{N_l}^{(l)} \end{bmatrix} \qquad \vec{a}^{(l)} = \begin{bmatrix} a_1^{(l)} \\ a_2^{(l)} \\ \vdots \\ a_{N_l}^{(l)} \end{bmatrix}$$

and to write the weights and bias at each level l as a k by j Matrix,

$$W^{(l)} = \begin{pmatrix} w_{11}^{(l)} & \cdots & w_{1i}^{(l)} & \cdots & w_{1N^{(l-1)}}^{(l)} \\ \vdots & \ddots & \vdots & \iddots & \vdots \\ w_{j1}^{(l)} & \cdots & w_{ji}^{(l)} & \cdots & w_{jN^{(l-1)}}^{(l)} \\ \vdots & \iddots & \vdots & \ddots & \vdots \\ w_{N^{(l)}1}^{(l)} & \cdots & w_{N^{(l)}i}^{(l)} & \cdots & w_{N^{(l)}N^{(l-1)}}^{(l)} \end{pmatrix} \qquad \vec{b}^{(l)} = \begin{pmatrix} b_1^l \\ \vdots \\ b_i^l \\ \vdots \\ b_{N^{(l-1)}}^l \end{pmatrix}$$

(note: To respect matrix notation, we have reversed the order of i and j in the subscripts.)

We can see that the weights are a 3rd order Tensor or vector of matrices, with one matrix for each layer, The biases are a matrix (vector of vectors) with a vector for each level.

$$\vec{z}^{(l)} = W^{(l)}\vec{a}^{(l-1)} + \vec{b}^{(l)} \quad \text{and} \quad \vec{a}^{(l)} = f(\vec{z}^{(l)}) = f(W^{(l)}\vec{a}^{(l-1)} + \vec{b}^{(l)})$$

We can assemble the set of matrices $W^{(l)}$ into an 3rd order Tensor (Vector of matrices), W, and represent $\vec{a}^{(l)}$, $\vec{z}^{(l)}$ and $\vec{b}^{(l)}$ as matrices (vectors of vectors): A, Z, B.

So how to do we learn the weights W and biases B?

We could train a 2-class detector from a labeled training set $\{\vec{X}_m\},\{y_m\}$ using gradient descent. For more than two layers, we will need to use the more general "back-propagation" algorithm.

Back-propagation adjusts the network the weights $w_{ij}^{(l)}$ and biases $b_j^{(l)}$ so as to minimize an error function between the network output $\vec{a}_m^L$ and the target value $\vec{y}_m$ for the M training samples $\{\vec{X}_m\}$, $\{\vec{y}_m\}$.

This is an iterative algorithm that propagates an error term back through the hidden layers and computes a correction for the weights at each layer so as to minimize the error term.

This raises two questions:
1) How do we initialize the weights?
2) How do we compute the error term for hidden layers?

Initializing the weights
How do we initialize the weights?
The obvious answer is to initialize all the weights to 0.
However, this causes problems.

If the parameters all start with identical values, then the algorithm will end up learning the same value for all parameters. To avoid this, the parameters should be initialized with small random variables that are near 0, for example computed with a normal density with variance ε (typically 0.01).

$\underset{i,j,l}{\forall} w_{ji}^{(l)} = \mathcal{N}(X;0,\varepsilon)$ and $\underset{j,l}{\forall} b_j^{(l)} = \mathcal{N}(X;0,\varepsilon)$ where $\mathcal{N}$ is a sample from a normal density.

An even better solution is provided by Xavier GLOROT's technique (see course web site for a paper on Xavier normalization).

Backpropagation

Back propagation is a distributed parallel algorithm for computing gradient descent. Back-propagation propagates the error term back through the layers, using the weights. We will present this for individual training samples. The algorithm can easily be generalized to learning from sets of training samples (Batch mode).

Given a training sample, $\vec{X}_m$, we first propagate the $\vec{X}_m$ through the L layers of the network (Forward propagation) to obtain an output activation $\vec{a}^{(L)}$.

We then compute an error term. In the case, of a multi-class network, this is a vector, with k components, one output for each hypothesis. In this case the indicator vector would be a vector, with one component for each possible class:

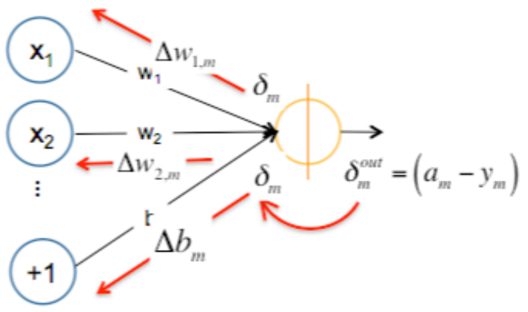
$$\vec{\delta}_m^{(out)} = \left(\vec{a}_m^{(L)} - \vec{y}_m\right) \qquad \text{or for each class k:} \quad \delta_{k,m}^{(out)} = \left(a_{k,m}^{(L)} - y_{k,m}\right)$$

To keep things simple, let us consider the case of a two class network, so that δ_m^{out}, $h(\vec{X}_m)$, $a_m^{(L)}$, and y_m are scalars. The results are easily generalized to vectors for multi-class networks.

For a single neuron, at the output layer, the "error" for each training sample is:

$$\delta_m^{out} = \left(a_m^{(L)} - y_m\right)$$

The error term $\vec{\delta}_m^{out}$ is the total error for the whole network for sample m. This error is used to compute an error for the weights that activate the neuron:



$$\delta_m = \frac{\partial f(z)}{\partial z}\delta_m^{out}$$

This correction is then used to determine a correction term for the weights:

$$\Delta w_{d,m} = x_d \delta_m$$
$$\Delta b_m = \delta_m$$

Backpropagation can be generalized for multiple neurons at multiple layers (*l=1* to *L*). The error term for unit *k* at layer *L* is:

$$\delta_{k,m}^{(L)} = \frac{\partial f(z_k^{(L)})}{\partial z_k^{(L)}} \delta_m^{out}$$

For the hidden units in layers $l < L$ the error $\delta_j^{(l)}$ is based on a weighted average of the error terms for $\delta_k^{(l+1)}$.

$$\delta_{j,m}^{(l)} = \frac{\partial f(z_j^{(l)})}{\partial z_j^{(l)}} \sum_{k=1}^{N^{l+1}} w_{jk}^{(l+1)} \delta_{k,m}^{(l+1)}$$

We compute error terms, $\delta_j^{(l)}$ for each unit *j* in layer *l* back to layer *l–1* using the sum of errors times the corresponding weights times the derivative of the activation function. This error term tells how much the unit j was responsible for differences between the activation of the network $\vec{h}(\vec{x}_m; w_{jk}^{(l)}, b_k^{(l)})$ and the target value $\vec{y}_m$.

For the sigmoid activation function, $\sigma(z) = \frac{1}{1+e^{-z}}$ the derivative is:

$$\frac{d\sigma(z)}{dz} = \sigma(z)(1-\sigma(z))$$

For $a_j^{(l)} = f(z_j^{(l)})$ this gives: $\delta_{j,m}^{(l)} = a_{j,m}^{(l)}(1-a_{j,m}^{(l)}) \cdot \sum_{k=1}^{N^{(l+1)}} w_{jk}^{(l+1)} \delta_{k,m}^{(l+1)}$

This error term can then used to correct the weights and bias terms leading from layer *j* to layer *i*.

$$\Delta w_{ij,m}^{(l)} = a_i^{(l-1)} \delta_{j,m}^{(l)}$$
$$\Delta b_{j,m}^{(l)} = \delta_{j,m}^{(l)}$$

Note that the corrections $\Delta w_{ij,m}^{(l)}$ and $\Delta b_{j,m}^{(l)}$ are NOT applied until after the error has propagated all the way back to layer *l=1*, and that when *l=1*, $a_i^{(0)} = x_i$.

For "batch learning", the corrections terms, $\Delta w_{ji,m}^{(l)}$ and $\Delta b_{j,m}^{(l)}$are averaged over M samples of the training data and then only an average correction is applied to the weights.

$$\Delta w_{ij}^{(l)} = \frac{1}{M}\sum_{m=1}^{M}\Delta w_{ij,m}^{(l)} \qquad \Delta b_{j}^{(l)} = \frac{1}{M}\sum_{m=1}^{M}\Delta b_{j,m}^{(l)}$$

then

$$w_{ij}^{(l)} \leftarrow w_{ij}^{(l)} - \eta\cdot\Delta w_{ij}^{(l)} \qquad b_{j}^{(l)} \leftarrow b_{j}^{(l)} - \eta\cdot\Delta b_{j}^{(l)}$$

where η is the learning rate.

Back-propagation is equivalent to computing the gradient of the loss function for each layer of the network. A common problem with gradient descent is that the loss function can have local minimum. This problem can be minimized by regularization. A popular regularization technique for back propagation is to use "momentum"

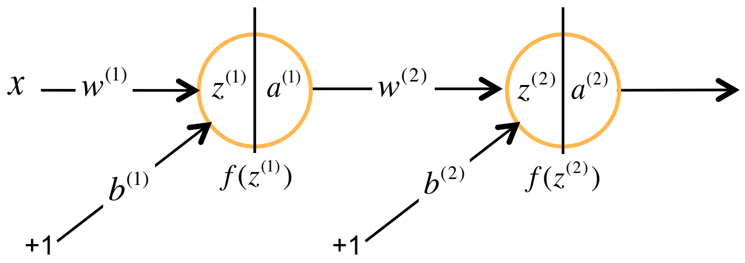
$$w_{ij}^{(l)} \leftarrow w_{ij}^{(l)} - \eta\cdot\Delta w_{ij}^{(l)} + \mu\cdot w_{ij}^{(l)}$$
$$b_{j}^{(l)} \leftarrow b_{j}^{(l)} - \eta\cdot\Delta b_{j}^{(l)} + \mu\cdot b_{j}^{(l)}$$

where the terms $\mu\cdot w_{j}^{(l)}$ and $\mu\cdot b_{j}^{(l)}$ serves to stabilize the estimation.

The back-propagation algorithm may be continued until all training data has been used. For batch training, the algorithm may be repeated until all error terms, $\delta_{j,m}^{(l)}$, are a less than a threshold.

Derivation of Backpropagation as gradient Descent.

To derive the backpropagation equations, consider a simple 2 layer network with 1 neuron at each level that maps a scalar feature, x, to a activation $a^{(2)}$.



The network equations are

$$
\begin{aligned}
z^{(1)} &= w^{(1)}x + b^{(1)} \\
a^{(1)} &= f(z^{(1)}) = f(w^{(1)}x + b^{(1)}) \\
z^{(2)} &= w^{(2)}a^{(1)} + b^{(2)} \\
a^{(2)} &= f(z^{(2)}) = f(w^{(2)}a^{(1)} + b^{(2)})
\end{aligned}
$$

The network has 4 parameters

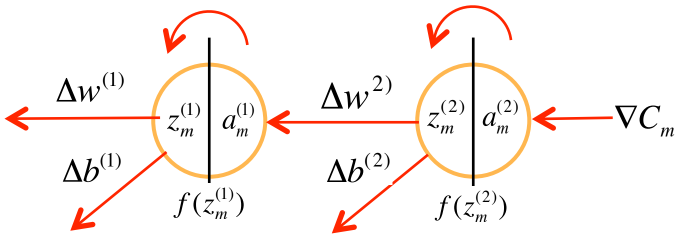
$$\vec{w} = \begin{pmatrix} w^{(1)} \\ b^{(1)} \\ w^{(2)} \\ b^{(2)} \end{pmatrix}$$

The “cost”, C, of the error of the network for using the parameters to discriminate the input, x, with ground truth, y, is:

$$C = \frac{1}{2}\left(a^{(2)} - y\right)^2$$

Where we have multiplied by "1/2" to simplify the algebra.

The gradient of the cost with respect to each of the parameters in $\vec{w}$ tells us how much each parameter contributed to the error.



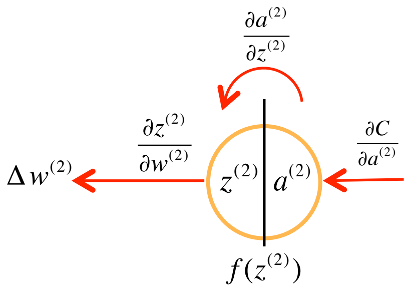
For our 2 layer network.

$$\nabla C = \frac{\partial C}{\partial \vec{w}} = \begin{pmatrix} \frac{\partial C}{\partial w^{(1)}} \\ \frac{\partial C}{\partial b^{(1)}} \\ \frac{\partial C}{\partial w^{(2)}} \\ \frac{\partial C}{\partial b^{(2)}} \end{pmatrix} = \begin{pmatrix} \Delta w^{(1)} \\ \Delta b^{(1)} \\ \Delta w^{(2)} \\ \Delta b^{(2)} \end{pmatrix}$$

To evaluate these derivatives we use the chain rule. For example the derivative with of the cost with respect to the weight of the second neuron, $w^{(2)}$ is

$$\frac{\partial C}{\partial w^{(2)}} = \frac{\partial C}{\partial a^{(2)}} \cdot \frac{\partial a^{(2)}}{\partial z^{(2)}} \cdot \frac{\partial z^{(2)}}{\partial w^{(2)}} = \Delta w^{(2)}$$

This can be seen graphically as:

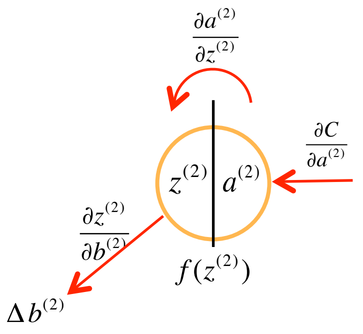


$\frac{\partial a^{(2)}}{\partial z^{(2)}}$
$\frac{\partial z^{(2)}}{\partial w^{(2)}}$
$\frac{\partial C}{\partial a^{(2)}}$
$\Delta w^{(2)}$
$z^{(2)}$
$a^{(2)}$
$f(z^{(2)})$

The derivative with respect to $b^{(2)}$ is:

$$\frac{\partial C}{\partial b^{(2)}} = \frac{\partial C}{\partial a^{(2)}} \cdot \frac{\partial a^{(2)}}{\partial z^{(2)}} \cdot \frac{\partial z^{(2)}}{\partial b^{(2)}} = \Delta b^{(2)}$$

This can be seen graphically as:



$\frac{\partial a^{(2)}}{\partial z^{(2)}}$
$\frac{\partial C}{\partial a^{(2)}}$
$z^{(2)}$
$a^{(2)}$
$f(z^{(2)})$
$\frac{\partial z^{(2)}}{\partial b^{(2)}}$
$\Delta b^{(2)}$

We can simplify the notation by defining an error term for each neuron.

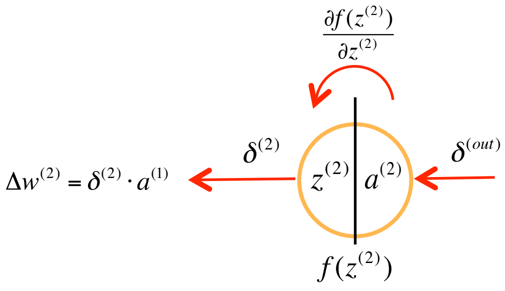
Let $\delta^{(out)} = \left(a^{(2)} - y\right) = \frac{\partial C}{\partial a^{(2)}}$ be the error for the error for the network for training sample x with ground truth indicator y.

The error term for 2nd neural unit is $\delta^{(2)} = \left(\frac{\partial a^{(2)}}{\partial z^{(2)}} \cdot \delta^{(out)}\right) = \left(\frac{\partial f(z^{(2)})}{\partial z^{(2)}} \cdot \delta^{(out)}\right)$

with this notation $\Delta w^{(2)} = \left(\frac{\partial C}{\partial a^{(2)}} \cdot \frac{\partial a^{(2)}}{\partial z^{(2)}}\right) \cdot \frac{\partial z^{(2)}}{\partial w^{(2)}} = \frac{\partial z^{(2)}}{\partial w^{(2)}} \cdot \delta^{(2)}$

Reordering the terms and noting that $\frac{\partial z^{(2)}}{\partial w^{(2)}} = \frac{\partial (w^{(2)} a^{(1)} + b^{(2)})}{\partial w^{(2)}} = a^{(1)}$

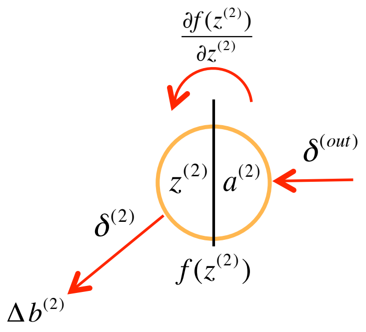
gives: $\Delta w^{(2)} = a^{(1)} \cdot \delta^{(2)}$


$\frac{\partial f(z^{(2)})}{\partial z^{(2)}}$
$\delta^{(2)}$
$\delta^{(out)}$
$\Delta w^{(2)} = \delta^{(2)} \cdot a^{(1)}$
$z^{(2)}$
$a^{(2)}$
$f(z^{(2)})$

Similarly for the bias term for the 2nd neural unit: $\Delta b^{(2)} = \left(\frac{\partial C}{\partial a^{(2)}} \cdot \frac{\partial a^{(2)}}{\partial z^{(2)}}\right) \cdot \frac{\partial z^{(2)}}{\partial b^{(2)}} = \delta^{(2)} \cdot \frac{\partial z^{(2)}}{\partial b^{(2)}}$

Noting that $\frac{\partial z^{(2)}}{\partial b^{(2)}} = \frac{\partial (w^{(2)} a^{(1)} + b^{(2)})}{\partial b^{(2)}} = 1$

We obtain: $\Delta b^{(2)} = \delta^{(2)}$



For the next layer we continue the same process recursively

The derivative of the cost with respect to $w^{(1)}$ is:

$$\Delta w^{(1)} = \frac{\partial C}{\partial w^{(1)}} = \left(\frac{\partial C}{\partial a^{(2)}} \cdot \frac{\partial a^{(2)}}{\partial z^{(2)}}\right) \cdot \left(\frac{\partial z^{(2)}}{\partial a^{(1)}}\right) \cdot \left(\frac{\partial a^{(1)}}{\partial z^{(1)}}\right) \cdot \frac{\partial z^{(1)}}{\partial w^{(1)}}$$

Substituting $\delta^{(2)} = \left(\frac{\partial C}{\partial a^{(2)}} \cdot \frac{\partial a^{(2)}}{\partial z^{(2)}}\right)$ gives $\Delta w^{(1)} = \delta^{(2)} \cdot \left(\frac{\partial z^{(2)}}{\partial a^{(1)}}\right) \cdot \left(\frac{\partial a^{(1)}}{\partial z^{(1)}}\right) \cdot \frac{\partial z^{(1)}}{\partial w^{(1)}}$

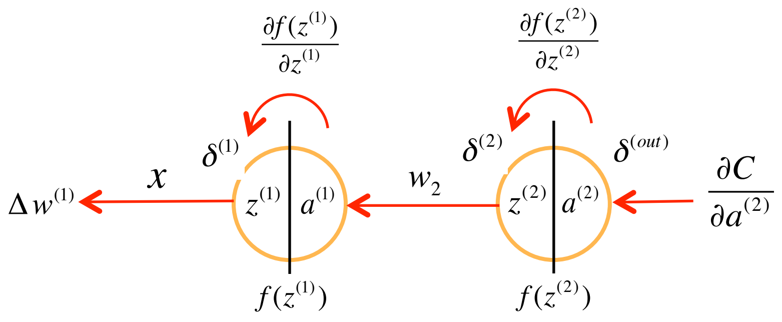
Substituting $w^{(2)} = \frac{\partial (w^{(2)} a^{(1)} + b^{(2)})}{\partial a^{(1)}} = \left(\frac{\partial z^{(2)}}{\partial a^{(1)}}\right)$ gives $\Delta w^{(1)} = \delta^{(2)} \cdot w^{(2)} \cdot \left(\frac{\partial a^{(1)}}{\partial z^{(1)}}\right) \cdot \frac{\partial z^{(1)}}{\partial w^{(1)}}$

Substituting $\frac{\partial f(z^{(1)})}{\partial z^{(1)}} = \left(\frac{\partial a^{(1)}}{\partial z^{(1)}}\right)$ gives $\Delta w^{(1)} = \left(\delta^{(2)} \cdot w^{(2)} \cdot \frac{\partial f(z^{(1)})}{\partial z^{(1)}}\right) \cdot \left(\frac{\partial z^{(1)}}{\partial w^{(1)}}\right)$

Substituting $x = \frac{\partial (w^{(1)} x + b^{(1)})}{\partial w^{(1)}} = \left(\frac{\partial z^{(1)}}{\partial w^{(1)}}\right)$ gives $\Delta w^{(1)} = \left(\delta^{(2)} \cdot w^{(2)} \cdot \frac{\partial f(z^{(1)})}{\partial z^{(1)}}\right) \cdot x$

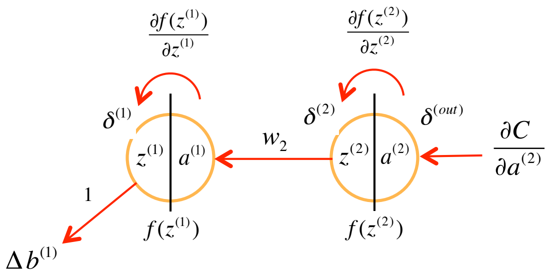
We define the error term for level 1 as $\delta^{(1)} = \left(\delta^{(2)} \cdot w^{(2)} \cdot \frac{\partial f(z^{(1)})}{\partial z^{(1)}}\right)$

Rearranging the terms gives: $\Delta w^{(1)} = x \cdot \delta^{(1)}$


$\frac{\partial f(z^{(1)})}{\partial z^{(1)}}$
$\frac{\partial f(z^{(2)})}{\partial z^{(2)}}$
$\delta^{(1)}$
$\delta^{(2)}$
$\delta^{(out)}$
$\frac{\partial C}{\partial a^{(2)}}$
$\Delta w^{(1)}$
x
$z^{(1)}$
$a^{(1)}$
w_2
$z^{(2)}$
$a^{(2)}$
$f(z^{(1)})$
$f(z^{(2)})$

Similarly for the correction factor of $b^{(1)}$

$$\Delta b^{(1)} = \frac{\partial C}{\partial b^{(1)}} = \left(\frac{\partial C}{\partial a^{(2)}} \cdot \frac{\partial a^{(2)}}{\partial z^{(2)}}\right) \cdot \left(\frac{\partial z^{(2)}}{\partial a^{(1)}}\right) \cdot \left(\frac{\partial a^{(1)}}{\partial z}\right) \cdot \frac{\partial z^{(1)}}{\partial b^{(1)}}$$


$\frac{\partial f(z^{(1)})}{\partial z^{(1)}}$
$\frac{\partial f(z^{(2)})}{\partial z^{(2)}}$
$\delta^{(1)}$
$\delta^{(2)}$
$\delta^{(out)}$
$\frac{\partial C}{\partial a^{(2)}}$
$z^{(1)}$
$a^{(1)}$
w_2
$z^{(2)}$
$a^{(2)}$
1
$f(z^{(1)})$
$f(z^{(2)})$
$\Delta b^{(1)}$

Substituting $\delta^{(2)} = \left(\frac{\partial C}{\partial a^{(2)}} \cdot \frac{\partial a^{(2)}}{\partial z^{(2)}}\right)$ gives $\Delta b^{(1)} = \delta^{(2)} \cdot \left(\frac{\partial z^{(2)}}{\partial a^{(1)}}\right) \cdot \left(\frac{\partial a^{(1)}}{\partial z^{(1)}}\right) \cdot \frac{\partial z^{(1)}}{\partial b^{(1)}}$

Substituting $w^{(2)} = \frac{\partial (w^{(2)} a^{(1)} + b^{(2)})}{\partial a^{(1)}} = \left(\frac{\partial z^{(2)}}{\partial a^{(1)}}\right)$ gives $\Delta b^{(1)} = \delta^{(2)} \cdot w^{(2)} \cdot \left(\frac{\partial a^{(1)}}{\partial z^{(1)}}\right) \cdot \frac{\partial z^{(1)}}{\partial b^{(1)}}$

Substituting $\left(\frac{\partial a^{(1)}}{\partial z^{(1)}}\right)=\frac{\partial f(z^{(1)})}{\partial z^{(1)}}$ gives $\Delta b^{(1)}=\left(\delta^{(2)}\cdot w^{(2)}\cdot\frac{\partial f(z^{(1)})}{\partial z^{(1)}}\right)\cdot\left(\frac{\partial z^{(1)}}{\partial b^{(1)}}\right)$

noting $\left(\frac{\partial z^{(1)}}{\partial b^{(1)}}\right)=\frac{\partial(w^{(1)}x\ +b^{(1)})}{\partial w^{(1)}}=1$ and substituting $\delta^{(1)}=\left(\delta^{(2)}\cdot w^{(2)}\cdot\frac{\partial f(z^{(1)})}{\partial z^{(1)}}\right)$

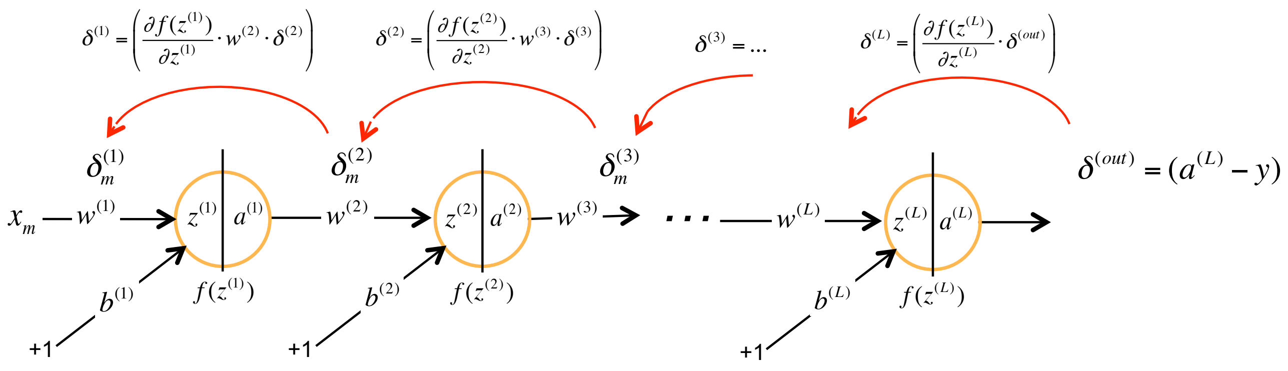
Gives: $\Delta b^{(1)}=\delta^{(1)}$

General formula for the error term

In general, the chain rule $\frac{\partial C}{\partial w^{(l)}}=\frac{\partial C}{\partial a^{(L)}}\cdot\frac{\partial a^{(L)}}{\partial z^{(L)}}\cdot\frac{\partial z^{(L)}}{\partial a^{(L-1)}}\cdot\ldots\cdot\frac{\partial z^{(l+1)}}{\partial a^{(l)}}\cdot\frac{\partial a^{(l)}}{\partial w^{(l)}}$

Provides a recursive formula for each neural unit:

$$\delta^{(l)}=\left(\frac{\partial f(z^{(l)})}{\partial z^{(l)}}\cdot w^{(l+1)}\cdot\left(\frac{\partial f(z^{(l+1)})}{\partial z^{(l+1)}}\cdot w^{(l+2)}\cdot\left(\cdots\left(\frac{\partial f(z^{(L)})}{\partial z^{(L)}}\cdot\delta^{(out)}\right)\right)\right)\right)$$


$\delta^{(1)}=\left(\frac{\partial f(z^{(1)})}{\partial z^{(1)}}\cdot w^{(2)}\cdot\delta^{(2)}\right)$
$\delta^{(2)}=\left(\frac{\partial f(z^{(2)})}{\partial z^{(2)}}\cdot w^{(3)}\cdot\delta^{(3)}\right)$
$\delta^{(3)}=\ldots$
$\delta^{(L)}=\left(\frac{\partial f(z^{(L)})}{\partial z^{(L)}}\cdot\delta^{(out)}\right)$
$\delta_m^{(1)}$
$\delta_m^{(2)}$
$\delta_m^{(3)}$
$\delta^{(out)}=(a^{(L)}-y)$
x_m
$w^{(1)}$
$z^{(1)}$
$a^{(1)}$
$w^{(2)}$
$z^{(2)}$
$a^{(2)}$
$w^{(3)}$
$w^{(L)}$
$z^{(L)}$
$a^{(L)}$
$b^{(1)}$
$f(z^{(1)})$
$b^{(2)}$
$f(z^{(2)})$
$b^{(L)}$
$f(z^{(L)})$
+1
+1
+1

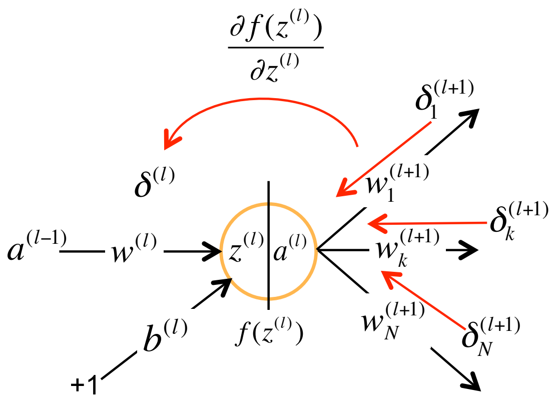
Giving a simple formula for adjusting the values of weights and biases

$\Delta w^{(l)}=a^{(l-1)}\delta^{(l)}$ and $\Delta b^{(l)}=\delta^{(l)}$

Formula for multiple activations

In the case where there are N neural units at level *l+1*,
the error at level *l* is the weighted sum of the errors at level *l+1*.

$$\delta^{(l)} = \left(\frac{\partial f(z^{(l)})}{\partial z^{(l)}} \cdot \sum_{k=1}^{N} w_k^{l+1} \cdot \delta_k^{(l+1)} \right)$$



Summary of Backpropagation

The Back-propagation algorithm can be summarized as:

1) Initialize the network and a set of correction vectors:

$$\begin{aligned}
&\underset{i,j,l}{\forall}\, w_{ji}^{(l)} = \mathcal{N}(X;0,\varepsilon)\\
&\underset{i,l}{\forall}\, b_{j}^{(l)} = \mathcal{N}(X;0,\varepsilon)\\
&\underset{i,j,l}{\forall}\, \Delta w_{ji}^{(l)} = 0\\
&\underset{i,l}{\forall}\, \Delta b_{j}^{(l)} = 0
\end{aligned}$$

where $\mathcal{N}$ is a sample from a normal density, and ε is a small value.

2) For each training sample, $\vec{x}_m$, propagate $\vec{x}_m$ through the network (forward propagation) to obtain a network activation $a_m^{(L)}$. Compute the error and propagate this back through the network:

a) Compute the network error term: $\delta_m^{out} = \left(a_m^{(L)} - y_m\right)$

b) Compute the error term at Layer L: $\delta_m^{(L)} = \frac{\partial f(z_j^{(l)})}{\partial z_j^{(l)}} \delta_m^{out}$

c) Propagate the error back from *l=L-1* to *l=1*: $\delta_{j,m}^{(l)} = \frac{\partial f(z_j^{(l)})}{\partial z_j^{(l)}} \sum_{k=1}^{N^{(l+1)}} w_{jk}^{(l+1)} \delta_{k,m}^{(l+1)}$

d) Use the error at each layer to set a vector of correction weights.

$$\Delta w_{ij,m}^{(l)} = a_i^{(l-1)} \delta_{j,m}^{(l)} \qquad \Delta b_{j,m}^{(l)} = \delta_{j,m}^{(l)}$$

3) For all layers, *l=1 to L*, update the weights and bias using a learning rate, η

$$\begin{aligned}
w_{ij}^{(l)} &\leftarrow w_{ij}^{(l)} - \eta \cdot \Delta w_{ij,m}^{(l)} + \mu \cdot w_{ij}^{(l)}\\
b_{j}^{(l)} &\leftarrow b_{j}^{(l)} - \eta \cdot \Delta b_{j,m}^{(l)} + \mu \cdot b_{j}^{(l)}
\end{aligned}$$

Note that this last step can be done with an average correction matrix obtained from many training samples (Batch mode), providing a more efficient algorithm.