Pattern Recognition and Machine Learning

James L. Crowley

ENSIMAG 3 - MMIS Fall Semester 2016/2017
Lessons 8 4 Jan 2017

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.
L	The number of layers (number of non-linear activation layers)
l	The layer index. l ranges from 1 (input layer) to L+1 (output)
$D^{(l)}$	The number of units in layer l
$a_j^{(l)}$	is the activation output of the j^{th} activation unit of the l^{th} layer.
$w_{kj}^{(l)}$	the weight for the unit j of layer l and the unit k of layer $l+1$.
$b_k^{(l)}$	the bias term feeding to unit k of layer $l+1$.
$f(z)$	A non-linear activation function, such as a sigmoid, tanh, or soft-max

Key Equations:

Feed Forward from Layer j to k:
$$a_k^{(l+1)} = f\left(\sum_{j=1}^{D^{(l)}} w_{kj}^{(l)} a_j^{(l)} + b_k^{(l)}\right)$$

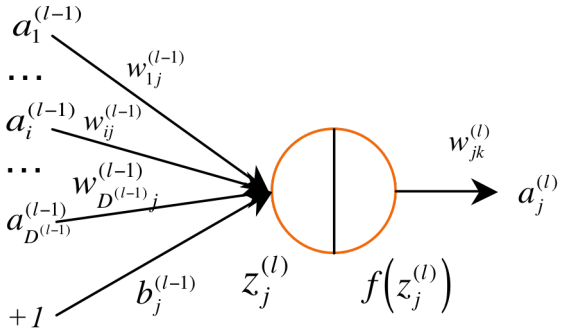
Back Propagation from Layer k to j:
$$\delta_{j,m}^{(l)} = \frac{\partial f(z_j^{(l)})}{\partial z_j^{(l)}} \sum_{k=1}^{D^{(l+1)}} w_{kj}^{(l)} \delta_{k,m}^{(l+1)}$$

Weight and Bias Corrections for layer j:
$$\Delta w_{ji,m}^{(l-1)} = a_i^{(l-1)} \delta_{j,m}^{(l)}$$
$$\Delta b_{j,m}^{(l-1)} = \delta_{j,m}^{(l)}$$

Network Update Formulas:
$$w_{ji}^{(l-1)} \leftarrow w_{ji}^{(l-1)} - \eta \cdot \Delta w_{ji,m}^{(l-1)}$$
$$b_j^{(l-1)} \leftarrow b_j^{(l-1)} - \eta \cdot \Delta b_{j,m}^{(l-1)}$$

Back Propagation for multi-layer networks.

Notation and Terminology



Recall that a "neural unit" j at level l computes a non-linear function for a weighted sum of activations from the previous level.

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

and feeds this forward to the $D^{(L+1)}$ units at level $l+1$. In a multi-layer network with more than 2 layers, this model continues recursively:

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

In the last lecture we introduced the notation:

$\vec{a}^{(1)} = \vec{X}$ is the input layer. $a_i^{(1)} = X_d$ and $D^{(1)} = D$

l The current layer under discussion.

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

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

$w_{kj}^{(l)}$ is the weight for the unit j of layer l feeding to unit k of layer $l+1$.
(This is $w_{kj}^{(l)}$ to respect matrix notation convention)

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

$b_j^{(l-1)}$ the bias term feeding to unit j of layer l.

$z_j^{(l)} = \sum_{i=1}^{D^{(l-1)}} w_{ji}^{(l-1)} a_i^{(l-1)} + b_j^{(l-1)}$ 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

$\vec{h}(\vec{X}_m; w_{kj}^{(l)}, b_k^{(l)}) = \vec{a}^{L+1}$ is the vector of network outputs (one for each class).

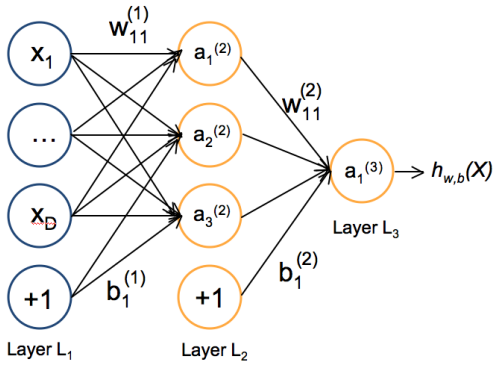
In our last lecture, we used the sigmoid activation function. $f(z) = \frac{1}{1+e^{-z}}$

This is convenient because the derivative is:

$$\frac{df(z)}{dz} = f(z)(1 - f(z))$$

and tends to give good results in training for 2 layer networks.

Using this notation, we saw that a simple 2-layer network had the form:

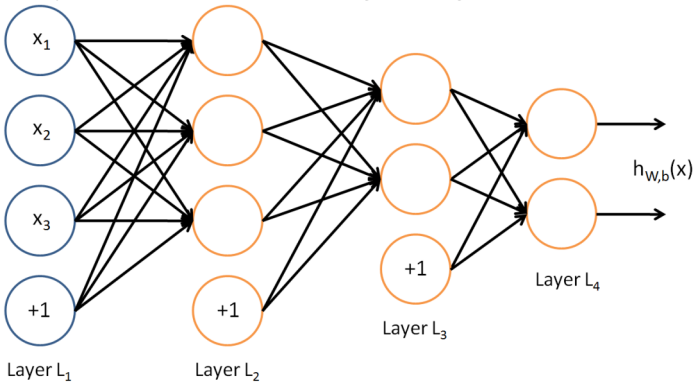


This network was be described by:

$$a_1^{(2)} = f(w_{11}^{(1)}X_1 + w_{12}^{(1)}X_2 + w_{13}^{(1)}X_3 + b_1^{(1)})$$
$$a_2^{(2)} = f(w_{21}^{(1)}X_1 + w_{22}^{(1)}X_2 + w_{23}^{(1)}X_3 + b_2^{(1)})$$
$$a_3^{(2)} = f(w_{31}^{(1)}X_1 + w_{32}^{(1)}X_2 + w_{33}^{(1)}X_3 + b_3^{(1)})$$
$$z_1^{(3)} = w_{11}^{(2)}a_1^{(2)} + w_{12}^{(2)}a_2^{(2)} + w_{13}^{(2)}a_3^{(2)} + b_1^{(2)}$$
$$h_{\vec{w},b}(\vec{X}) = a_1^{(3)} = f(z_1^{(3)}) = f(w_{11}^{(2)}a_1^{(2)} + w_{12}^{(2)}a_2^{(2)} + w_{13}^{(2)}a_3^{(2)} + b_1^{(2)})$$

Multi-Layer Networks

This feed forward network is easily generalized to multiple layers and multiple classes. Thus a three-layer network for recognizing two classes would have the form:



The activations of each layer are described by $a_j^{(l)} = f(\sum_{i=1}^{D^{(l-1)}} w_{ji}^{(l-1)} a_i^{(l-1)} + b_j^{(l-1)})$

and the output hypotheses can be seen as a vector of activations: $\vec{h}_{\vec{w},b}(\vec{X}) = \vec{a}^{(L+1)}$.

We have identified the linear term $z_j^{(l)} = \sum_{i=1}^{D^{(l-1)}} w_{ji}^{(l-1)} a_i^{(l-1)} + b_j^{(l-1)}$ because we will need its derivative below.

Note that the sigmoid function ranges from 0 to 1. The target variables for each training sample, $\{\vec{X}_m\}$ where y_m=0 for negative samples and y_m=1 for positive samples. For a network to recognize K classes, we can replace the scalar value y_m with a k dimensional vector:

$$\vec{y}_m = \begin{pmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{pmatrix}.$$

The k[th] term is set to 1 for the target class and 0 for the other classes.

So how to do we learn the weights $w_{ji}^{(l)}$ and biases $b_i^{(l)}$? In our last lecture, we saw that 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.

Backpropagation

Back-propagation adjusts the network the weights $w_{kj}^{(l)}$ and biases $b_k^{(l)}$ so as to minimize an error function between the network output $\vec{h}(\vec{X}_m; w_{kj}^{(l)}, b_k^{(l)}) = \vec{a}^{(L+1)}$ 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?

1) How do we initialize the weights?

A natural answer for the first question is to initialize the weights to 0.

By experience this causes problems. If the parameters all start with identical values, then the algorithm can end up learning the same value for all parameters. To avoid this, we initialize the parameters with a small random variable that is near 0, for example computed with a normal density with variance ε (typically 0.01).

$\forall_{j,k,l} w_{kj}^{(l)} = \mathcal{N}(0;\varepsilon)$ and $\forall_{k,l} b_k^{(l)} = \mathcal{N}(0;\varepsilon)$ where $\mathcal{N}$ is a sample from a normal density.

An even better solution is provided by Xavier GLORIOT's technique (see course web site on Xavier normalisation). However that solution is too complex for today's lecture.

2) How do we compute the error term?

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 a hypothesis $\vec{h}(\vec{X}_m; w_{kj}^{(l)}, b_k^{(l)}) = \vec{a}_m^{(L+1)}$.

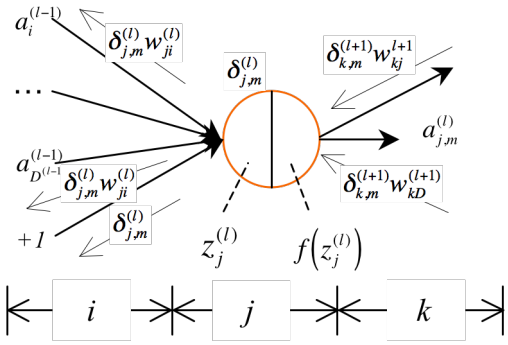
We then compute an error term. In the case, of a multi-class network, this is a vector, with components for each hypothesis.

$$\vec{\delta}_m^{(L+1)} = \vec{a}_m^{(L+1)} - \vec{y}_m$$

To keep things simple, let us consider the individual output units, so that $\delta_m^{(L+1)}$, $h(\vec{X}_m)$, $a_m^{(L+1)}$, and y_m are scalars. The results are easily generalized to vectors for multi-class networks. For each output class k:

$$\delta_m^{(L+1)} = a_m^{(L+1)} - y_m$$

This error term tells how much the unit was responsible for differences between the activation of the network $h(\vec{X}_m; w_{kj}^{(l)}, b_k^{(l)})$ and the target value y_m.



For the hidden units in layers $l < L+1$ the error $\delta_j^{(l)}$ is based on a weighted average of the error terms for $a_k^{(l+1)}$. We compute error terms, $\delta_j^{(l)}$ for each unit j in layer $l = L$ back to $l = 1$ by projecting the error back to earlier units using the current weights.

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

For the sigmoid activation function. $f(z) = \dfrac{1}{1+e^{-z}}$ the derivative is:

$$\frac{df(z)}{dz} = f(z)(1 - f(z))$$

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

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

$$\Delta w_{ji,m}^{(l-1)} = a_i^{(l-1)} \delta_{j,m}^{(l)} \quad \text{or equivalently} \quad \Delta w_{kj,m}^{(l)} = a_j^{(l)} \delta_{k,m}^{(l+1)}$$
$$\Delta b_{j,m}^{(l-1)} = \delta_{j,m}^{(l)} \quad \text{or equivalently} \quad \Delta b_{k,m}^{(l)} = \delta_{k,m}^{(l+1)}$$

Note that this correction is NOT applied until after the error has propagated all the way back to layer $l=2$. For Batch learning, the corrections terms are averaged over the training data and then only an average correction is applied.

$$w_{ji}^{(l-1)} \leftarrow w_{ji}^{(l-1)} - \eta \cdot \Delta w_{ji}^{(l-1)}$$
$$b_j^{(l-1)} \leftarrow b_j^{(l-1)} - \eta \cdot \Delta b_j^{(l-1)}$$

where η is the learning rate.

Note that back-propagation is equivalent to computing the gradient of the loss function for each layer of the network.

A 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"

$$w_{ij}^{(l-1)} \leftarrow w_{ij}^{(l-1)} - \eta \cdot \Delta w_{ij}^{(l-1)} + \mu \cdot w_{ij}^{(l-1)}$$
$$b_j^{(l-1)} \leftarrow b_j^{(l-1)} - \eta \cdot \Delta b_j^{(l-1)} + \mu \cdot b_j^{(l-1)}$$

where the terms $\mu \cdot w_{ji}^{(l-1)}$ and $\mu \cdot b_j^{(l-1)}$ 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.

Summary of Backpropagation

The Back-propagation algorithm can be summarized as:

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

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

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 hypothesis $h(\vec{X}_m; w_{ji}^{(l)}, b_j^{(l)})$. Compute the error and propagate this back through the network:

a) Compute the error term: $\delta_m^{(L+1)} = h(\vec{X}_m; w_{ji}^{(l)}, b_j^{(l)}) - y_m = a_m^{(L+1)} - y_m$

b) Propagate the error back from $l=L$ to $l=2$:

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

c) Use the error to set a vector of correction weights:

$$\begin{aligned}
\Delta w_{ji,m}^{(l-1)} &= a_i^{(l-1)} \delta_{j,m}^{(l)} \\
\Delta b_{j,m}^{(l-1)} &= \delta_{j,m}^{(l)}
\end{aligned}$$

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

$$\begin{aligned}
w_{ji}^{(l-1)} &\leftarrow w_{ji}^{(l-1)} - \eta \cdot \Delta w_{ji,m}^{(l-1)} + \mu \cdot w_{ji}^{(l-1)} \\
b_j^{(l-1)} &\leftarrow b_j^{(l-1)} - \eta \cdot \Delta b_{j,m}^{(l-1)} + \mu \cdot b_j^{(l-1)}
\end{aligned}$$

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.

Back-propagation Algorithm as Gradient Descent

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

$$W^{(l)} = \begin{pmatrix} w_{11}^{(l)} & \cdots & w_{1j}^{(l)} & \cdots & w_{1D^{(l-1)}}^{(l)} \\ \vdots & \ddots & \vdots & \iddots & \vdots \\ w_{k1}^{(l)} & \cdots & w_{kj}^{(l)} & \cdots & w_{kD^{(l-1)}}^{(l)} \\ \vdots & \iddots & \vdots & \ddots & \vdots \\ w_{kD^{(l)}}^{(l)} & \cdots & w_{kj}^{(l)} & \cdots & w_{D^{(l)}D^{(l-1)}}^{(l)} \end{pmatrix} \quad \vec{b}^{(l)} = \begin{pmatrix} b_1^l \\ \vdots \\ b_k^l \\ \vdots \\ b_{D^{(l)}}^l \end{pmatrix}$$

We can see that the weights are a 3rd order Tensor (vector of matrices) and that the biases are a matrix (vector of vectors). We can write these as the tensor W and the matrix B

Each of the M training samples $\{\vec{X}_m\}$, $\{y_m\}$ contributes an average loss to a cost function that can be defined as the square of the error between the activation and the target.

$$L(W, B; \vec{X}_m, y_m) = \frac{1}{2}(a_m^{(L+1)} - y_m)^2$$

The ½ term will simplify the algebra when we compute a derivative.
For each training sample $\vec{X}_m$, y_m, and for each unit j in layer l we use this loss function to update the weights and bias using partial derivatives.

$$w_{ji}^{(l)} \leftarrow w_{ji}^{(l)} - \eta \frac{\partial}{\partial w_{ji}^{(l)}} L(W, B; \vec{X}_m, y_m)$$

and

$$b_j^{(l)} \leftarrow b_j^{(l)} - \eta \frac{\partial}{\partial b_j^{(l)}} L(W, B; \vec{X}_m, y_m)$$

Where η is a learning rate. This is fine for the final level L. However, to propagate to earlier levels we need to apportion the error to different units. We need to determine what part of this loss is due to each coefficient $w_{jk}^{(l)}$.

To keep the notation simple, let us define the Loss at from sample m as

$$L_m = L(W, B; \vec{X}_m, y_m)$$

The contribution of each coefficient $w_{ji}^{(l-1)}$ to the loss is $\frac{\partial L_m}{\partial w_{ji}^{(l-1)}}$

Using the chain rule, we can rewrite this as

$$\frac{\partial L_m}{\partial w_{ji}^{(l-1)}} = \frac{\partial L_m}{\partial z_j^{(l)}} \frac{\partial z_j^{(l)}}{\partial w_{ji}^{(l-1)}}$$

The error term for unit *j* of level *l* is: $\delta_{j,m}^{(l)} = \frac{\partial L_m}{\partial z_j^{(l)}}$

We saw above that

$$z_j^{(l)} = \sum_{i=1}^{D^{(l)}} w_{ji}^{(l-1)} a_i^{(l-1)} + b_j^{(l-1)}$$

Thus $\frac{\partial z_j^{(l)}}{\partial w_{ji}^{(l-1)}} = a_i^{(l-1)}$ and so $\frac{\delta L_m}{\delta w_{ji}^{(l-1)}} = \delta_j^{(l)} a_i^{(l-1)}$

To propagate from layer k down to layer j we note that the same formula applies at level l+1.

$$\frac{\delta L_m}{\delta w_{kj}^{(l)}} = \delta_k^{(l+1)} a_j^{(l)}$$

By the chain rule.

$$\delta_{j,m}^{(l)} = \frac{\partial L_m}{\partial z_j^{(l)}} = \frac{\partial L_m}{\partial f(z_j^{(l)})} \frac{\partial f(z_j^{(l)})}{\partial z_j^{(l)}}$$

and

$$\frac{\partial L_m}{\partial f(z_j^{(l)})} = \sum_{k=1}^{D^{(l+1)}} w_{kj}^{(l)} \delta_{k,m}^{(l+1)}$$

Giving:

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