Neural Networks

Philippe Vergnet

April 13, 2016

Abstract

This paper presents Neural Networks in their matricial form. It shows how to compute the gradient of a cost function, provided this cost follows a specific functional form.

1 Pass Forward

Let X be the m x h_0 input matrix.
Let Y be the m x h_L target matrix.

We want a model that will approximate Y based on X.

In regular linear regression, the predictions that are meant to be close to Y are given by $XW_1' + B_1$ (B_1 being the biaises).
W_1 and B_1 can be optimized easily and the solution has an analytic form ($W_1 = (X'X)^{-1}X'Y$).

The drawback is that linear regressions can only emulate linear relationship ($f(x) = ax+b$).
It could not replicate more complex relationships like "*if day is Monday or Friday then 0.8 * stock else 0.5 * stock*" or even a simple x^2.

A possibility would be to compound the weights, such as to have $L > 1$ of them.
We would achieve it by defining the following:

For $l \in [1; L]$:

$$Z_l = Z_{l-1}W_l' + B_l'$$

Where:

W_l: a h_l x h_{l-1} matrix, is the weight matrix for layer l
B_l: a h_l vector, is the biais vector for layer l
Z_0 is X
Z_L is the prediction matrix.

NB. We will speak of hidden layer when $l < L$

This is already more powerful as a layer can "meta-learn" from what has been learned by the previous layer.
A bit like Einstein managed to achieve great discoveries thanks to Newton who previously discovered the law of Gravity and Descartes who detailed how to solve polynomial equations.

However, it would require a large number of hidden layers to fit the most complex functions, without guarantee of even being close enough.
After all, those layers are all learning in a similar way, linearly.
If Newton were to learn from Newton's discoveries, he would certainly achieve great things, but somehow be limited. Discoveries would be far greater if a different mind -like Einstein's- would come into play.

So, in the light of this, we want to add more value in the training of each layer, and especially break the boring linearity.
That's why we will apply to the results of each layer some non linear function s such as for $l \in [1; L]$:

$$Z_l = A_{l-1} W'_l + B'_l \tag{1}$$
$$A_l = s(Z_l) \tag{2}$$

Note that A_L is now the output of the network (the predictions).
Note also that A_0 is now X.
Applying those relationships iteratively until L is called a "pass forward".
One last note: for the hidden layers, h_l is the number of "neurons" in the layer.

2 Transfer Function

s is called the transfer function (or activation function).
It needs to be univariate (take and return one real number) and at least once differentiable.
Note that it is applied elementwise on Z_l.
It's main purpose is usually to introduce some non linearity and thus enable the network to emulate complex functions.
It could be sinusoid ($sin(x)$), gaussian ($\exp(-x^2)$) etc.
The most often used are the Sigmoid (a.k.a Logistic) function ($\frac{1}{1+\exp(-x)}$) and the Hyperbolic Tangent ($tanh(x)$).
They are great because they have a linear part (around zero) and nonlinear parts (when the input is either rather positively or negatively big).
Because they are bounded ($[0; 1$ for the Sigmoid, $[-1; 1]$ for the $tanh$), they work well with classification problems.

One can use different transfer functions for different layers and really (2) should be $A_l =$

$s_l(Z_l)$.
This enable to leverage on different ways of learning (combining different minds).
In this paper we omit the indexing of s for notation simplicity.

It has been mathematicaly shown that a neural networks can approximate as closely as needed any continuous function (provided there is enough layers and/or neurons).

3 Cost

To assess the accuracy of the network, we need a function that assess how far the predictions are from the actual target values hold in the Y matrix.

The most common distance measure is the Euclidean (or L2) norm:

$$Cost = \sqrt{\sum_{s=1}^{m} \sum_{c_L=1}^{h_L} (A_L[s, c_L] - Y[s, c_L])^2}$$

Actually, to avoid struggling with the square root in the gradient calculation, we are better off removing it in favor of a sample averaging:

$$Cost = \frac{1}{m} \sum_{s=1}^{m} \sum_{c_L=1}^{h_L} (A_L[s, c_L] - Y[s, c_L])^2$$

This has the merit of looking like the Mean Squared Errors, well known in statisctics.

NB. For neural network, we usually divide this cost by 2 as a convenient way to simplify some calculations of the gradient.

This objective function has the benefit of working with any numeric outputs.
In the case where outputs only take value in 0 and one (binary), another cost function might be preferred, the logistic cost:

$$Cost = \frac{1}{m} \sum_{s=1}^{m} \sum_{c_L=1}^{h_L} [\log(1 - A_L[s, c_L]) * (1 - Y[s, c_L]) - \log(A_L[s, c_L]) * (Y[s, c_L])]$$

If $Y[s, j]$ is 0, then the second term disappear and the cost is minimum (close to zero) if $A_L[s, j]$ is close to zero and goes to plus infinity if it instead is far off, close to one.
The same reasoning applies when $Y[s, j]$ is one.
So this function represents a good distance (always positive, close to zero when close, bigger and bigger with distance).

In any case, I believe that a cost function should exhibit the following functional structure:

$$Cost = \frac{1}{m} \sum_{s=1}^{m} \sum_{c_L=1}^{h_L} f(A_L[s, c_L]) \tag{3}$$

Where the function f is at least once differentiable.

4 Optimization

Now that we have an objective function, we know that a good network is such that the cost/error function is small.
To reduce the cost we can play on two aspects:

* The network structure: number of hidden layers and their respective number of neurons
* For a given structure, optimize the weights and biaises

Clearly, the more complex the structure the lower the cost. However, assuming your goal is to predict using data not seen in the training set, then you might want to use a lighter structure (and possibly accept a higher cost on the training set) in order to avoid overfitting. There is no rigorous science to find the best structure. Only some heuristics and a good deal of trial and errors.

So lets' focus on the second aspect.
To find the weights and biaises that minimize the cost, one need to use some numeric method as the complex and iterative structure of networks don't allow for a closed form solution.
One can use miscellaenous methods, like Gradient Descent, Conjugate Gradient methods, Quasi Newton methods (BFGS,L-BFGS...) etc.

In any case, we need the gradient of the cost function in respect of each weight Matrix and biais Vector.

5 Gradient

Let's define the following matrices:

$$D_L = f'(A_L).s'(Z_L) \tag{4}$$
$$D_{l-1} = D_l W_l.s'(Z_{l-1}) \text{ where } l \in [1; L] \tag{5}$$

The gradient of the *Cost* function with respect to a weight matrix is then given by:

$$\forall j \in [1; L] \quad \frac{\delta Cost}{\delta W_j} = \frac{1}{m} D'_j A_{j-1} \tag{6}$$

The gradient of the $Cost$ function with respect to a biais vector is then given by:

$$\forall j \in [1; L] \quad \frac{\delta Cost}{\delta W_j} = \frac{1}{m} D'_j 1_j \tag{7}$$

Where 1_j is a vector of m ones.

6 Gradient's Proof

Notations:

* Upper case letter are for matrix and vector.
* Sqare brakets [] are used for matrix indices: $A[s, j]$ is the element in A at the i^{th} row and j^{th} column.
* Curly braces {} are also used to show that the result of whatever inside is a matrix.
* Dot "." is for elementwise matrix multiplication.

To prove (6), let's first prove that we have $\forall j \in [1; L-1]$, $\forall l \in [j+1; L]$:

$$\frac{\delta Cost}{\delta W_j[z,k]} = \frac{1}{m} \sum_{s=1}^{m} \sum_{c_l=1}^{h_l} D_l[s, c_l] \sum_{c_{l-1}=1}^{h_{l-1}} W'_l[c_{l-1}, c_l] \frac{\delta(A_{l-1}[s, c_{l-1}])}{\delta W_j[z,k]} \tag{8}$$

Let's check (8) is true for $l = L$:

$$\begin{aligned}
\frac{\delta Cost}{\delta W_j[z,k]} &= \frac{\delta(\frac{1}{m} \sum_{s=1}^{m} \sum_{c_L=1}^{h_L} f(A_L[s, c_L]))}{\delta W_j[z,k]}, \text{ according to } Cost \text{ defined in (3)} \\
&= \frac{1}{m} \sum_{s=1}^{m} \sum_{c_L=1}^{h_L} \frac{\delta(f(A_L[s, c_L]))}{\delta W_j[z,k]}, \text{ by linearity of the } \delta \text{ operator} \\
&= \frac{1}{m} \sum_{s=1}^{m} \sum_{c_L=1}^{h_L} f'(A_L[s, c_L]) \frac{\delta(A_L[s, c_L])}{\delta W_j[z,k]}, \text{ by chain rule} \\
&= \frac{1}{m} \sum_{s=1}^{m} \sum_{c_L=1}^{h_L} f'(A_L[s, c_L]) \frac{\delta(s(Z_L[s, c_L]))}{\delta W_j[z,k]}, \text{ by (2)} \\
&= \frac{1}{m} \sum_{s=1}^{m} \sum_{c_L=1}^{h_L} f'(A_L[s, c_L]) s'(Z_L[s, c_L]) \frac{\delta(Z_L[s, c_L])}{\delta W_j[z,k]}, \text{ by chain rule} \\
&= \frac{1}{m} \sum_{s=1}^{m} \sum_{c_L=1}^{h_L} \{f'(A_L).s'(Z_L)\}[s, c_L] \frac{\delta(Z_L[s, c_L])}{\delta W_j[z,k]} \text{ (elementwise multiplication)}
\end{aligned}$$

$$= \frac{1}{m}\sum_{s=1}^{m}\sum_{c_L=1}^{h_L} D_L[s,c_L]\frac{\delta(Z_L[s,c_L])}{\delta W_j[z,k]}, \text{ by (4)}$$

$$= \frac{1}{m}\sum_{s=1}^{m}\sum_{c_L=1}^{h_L} D_L[s,c_L]\frac{\delta(\{A_{L-1}W'_L\}[s,c_L]+B'_L[c_L])}{\delta W_j[z,k]}, \text{ by (1)}$$

$$= \frac{1}{m}\sum_{s=1}^{m}\sum_{c_L=1}^{h_L} D_L[s,c_L]\frac{\delta(\{A_{L-1}W'_L\}[s,c_L])}{\delta W_j[z,k]} \text{ as } B_L \text{ is not sensitive to the weights}$$

$$= \frac{1}{m}\sum_{s=1}^{m}\sum_{c_L=1}^{h_L} D_L[s,c_L]\frac{\delta(\sum_{c_{L-1}=1}^{h_{L-1}} A_{L-1}[s,c_{L-1}]W'_L[c_{L-1},c_L])}{\delta W_j[z,k]} \text{ (matrix multiplication)}$$

$$= \frac{1}{m}\sum_{s=1}^{m}\sum_{c_L=1}^{h_L} D_L[s,c_L]\sum_{c_{L-1}=1}^{h_{L-1}} W'_L[c_{L-1},c_L]\frac{\delta(A_{L-1}[s,c_{L-1}])}{\delta W_j[z,k]}$$

The last expression is obtained by linearity of the δ operator and the fact that W_L doesn't depend on weights in previous layers.

Now let's assume (8) is true down to l, where $l \in [min(L, j+2); L]$.
Let's show that the equation also holds for $l-1$:

$$\frac{\delta Cost}{\delta W_j[z,k]} = \frac{1}{m}\sum_{s=1}^{m}\sum_{c_l=1}^{h_l} D_l[s,c_l]\sum_{c_{l-1}=1}^{h_{l-1}} W'_l[c_{l-1},c_l]\frac{\delta(A_{l-1}[s,c_{l-1}])}{\delta W_j[z,k]}, \text{ by (8)}$$

$$= \frac{1}{m}\sum_{s=1}^{m}\sum_{c_l=1}^{h_l} D_l[s,c_l]\sum_{c_{l-1}=1}^{h_{l-1}} W_l[c_l,c_{l-1}]\frac{\delta(A_{l-1}[s,c_{l-1}])}{\delta W_j[z,k]}, \text{ by transposition of } W_l$$

$$= \frac{1}{m}\sum_{s=1}^{m}\sum_{c_{l-1}=1}^{h_{l-1}}\sum_{c_l=1}^{h_l} D_l[s,c_l]W_l[c_l,c_{l-1}]\frac{\delta(A_{l-1}[s,c_{l-1}])}{\delta W_j[z,k]}, \text{ after rearranging the nested sums}$$

$$= \frac{1}{m}\sum_{s=1}^{m}\sum_{c_{l-1}=1}^{h_{l-1}} \{D_l W_l\}[s,c_{l-1}]\frac{\delta(A_{l-1}[s,c_{l-1}])}{\delta W_j[z,k]} \text{ (matrix multiplication)}$$

$$= \frac{1}{m}\sum_{s=1}^{m}\sum_{c_{l-1}=1}^{h_{l-1}} \{D_l W_l\}[s,c_{l-1}]\frac{\delta(s(Z_{l-1}[s,c_{l-1}])}{\delta W_j[z,k]}, \text{ by (2)}$$

$$= \frac{1}{m}\sum_{s=1}^{m}\sum_{c_{l-1}=1}^{h_{l-1}} \{D_l W_l\}[s,c_{l-1}]s'(Z_{l-1}[s,c_{l-1}])\frac{\delta(Z_{l-1}[s,c_{l-1}])}{\delta W_j[z,k]}, \text{ by chain rule}$$

$$= \frac{1}{m}\sum_{s=1}^{m}\sum_{c_{l-1}=1}^{h_{l-1}} \{\{D_l W_l\}.s'(Z_{l-1})\}[s,c_{l-1}]\frac{\delta(Z_{l-1}[s,c_{l-1}])}{\delta W_j[z,k]} \text{ (elementwise multiplication)}$$

$$= \frac{1}{m}\sum_{s=1}^{m}\sum_{c_{l-1}=1}^{h_{l-1}} D_{l-1}[s,c_{l-1}]\frac{\delta(Z_{l-1}[s,c_{l-1}])}{\delta W_j[z,k]} \text{ by (5)}$$

$$= \frac{1}{m}\sum_{s=1}^{m}\sum_{c_{l-1}=1}^{h_{l-1}} D_{l-1}[s,c_{l-1}]\frac{\delta(\{A_{l-2}W'_{l-1}\}[s,c_{l-1}]+B'_{l-1}[c_{l-1}])}{\delta W_j[z,k]}, \text{ by (1)}$$

$$= \frac{1}{m}\sum_{s=1}^{m}\sum_{c_{l-1}=1}^{h_{l-1}} D_{l-1}[s,c_{l-1}]\frac{\delta(\{A_{l-2}W'_{l-1}\}[s,c_{l-1}])}{\delta W_j[z,k]}, \text{ biaises being insensitive}$$

$$= \frac{1}{m}\sum_{s=1}^{m}\sum_{c_{l-1}=1}^{h_{l-1}} D_{l-1}[s,c_{l-1}]\frac{\delta(\sum_{c_{l-2}=1}^{h_{l-2}} A_{l-2}[s,c_{l-2}]W'_{l-1}[c_{l-2},c_{l-1}])}{\delta W_j[z,k]}$$

$$= \frac{1}{m}\sum_{s=1}^{m}\sum_{c_{l-1}=1}^{h_{l-1}} D_{l-1}[s,c_{l-1}]\sum_{c_{l-2}=1}^{h_{l-2}} W'_{l-1}[c_{l-2},c_{l-1}]\frac{\delta(A_{l-2}[s,c_{l-2}])}{\delta W_j[z,k]}$$

Which concludes the proof for (8).

Let's examine (8) for $l = j+1$:

$$\frac{\delta Cost}{\delta W_j[z,k]} = \frac{1}{m}\sum_{s=1}^{m}\sum_{c_{j+1}=1}^{h_{j+1}} D_{j+1}[s,c_{j+1}]\sum_{c_j=1}^{h_j} W'_{j+1}[c_j,c_{j+1}]\frac{\delta(A_j[s,c_j])}{\delta W_j[z,k]}, \text{ by (8)}$$

$$= \frac{1}{m}\sum_{s=1}^{m}\sum_{c_j=1}^{h_j}\sum_{c_{j+1}=1}^{h_{j+1}} D_{j+1}[s,c_{j+1}]W_{j+1}[c_{j+1},c_j]\frac{\delta(s(Z_j[s,c_j]))}{\delta W_j[z,k]}, \text{ by (2)}$$

$$= \frac{1}{m}\sum_{s=1}^{m}\sum_{c_j=1}^{h_j}\{D_{j+1}W'_{j+1}\}[s,c_j]s'(Z_j[s,c_j])\frac{\delta(Z_j[s,c_j])}{\delta W_j[z,k]}, \text{ by chain rule}$$

$$= \frac{1}{m}\sum_{s=1}^{m}\sum_{c_j=1}^{h_j} D_j[s,c_j]\frac{\delta(\{A_{j-1}W'_j\}[s,c_j]+B'_j[c_j])}{\delta W_j[z,k]}, \text{ by (5) and (1)}$$

$$= \frac{1}{m}\sum_{s=1}^{m}\sum_{c_j=1}^{h_j} D_j[s,c_j]\frac{\delta(\{A_{j-1}W'_j\}[s,c_j])}{\delta W_j[z,k]}, \text{ biaises being insensitive}$$

$$= \frac{1}{m}\sum_{s=1}^{m}\sum_{c_j=1}^{h_j} D_j[s,c_j]\frac{\delta(\sum_{c_{j-1}=1}^{h_{j-1}} A_{j-1}[s,c_{j-1}]W'_j[c_{j-1},c_j])}{\delta W_j[z,k]}$$

$$= \frac{1}{m}\sum_{s=1}^{m}\sum_{c_j=1}^{h_j} D_j[s,c_j]\sum_{c_{j-1}=1}^{h_{j-1}} A_{j-1}[s,c_{j-1}]\frac{\delta(W_j[c_j,c_{j-1}])}{\delta W_j[z,k]}, \text{ as } A_{j-1} \text{ is independent from } W_j$$

$$= \frac{1}{m}\sum_{s=1}^{m}\sum_{c_j=z}^{z} D_j[s,c_j]\sum_{c_{j-1}=k}^{k} A_{j-1}[s,c_{j-1}]\frac{\delta(W_j[c_j,c_{j-1}])}{\delta W_j[z,k]}$$

$$= \frac{1}{m}\sum_{s=1}^{m} D_j[s,z]A_{j-1}[s,k]$$

$$= \frac{1}{m} \sum_{s=1}^{m} D'_j[z,s] A_{j-1}[s,k]$$
$$= \frac{1}{m} \left\{ D'_j A_{j-1} \right\} [z,k]$$

And the last equation finish proving that $\frac{\delta Cost}{\delta W_j} = \frac{1}{m} D'_j A_{j-1}$, i.e. (6).

For the biaises, the proof is similar.