Neural networks

1 Derivation of the backpropagation equations

> *Lasciate ogne speranza, voi ch'intrate* - Dante Alighieri, Inferno

1.1 Initial concepts

The activation function o_j for the j-th neuron in a neural network is given by equations

$$o_j = \sigma(n_j),$$
$$n_j = \sum_{\ell \in L} w_{\ell j} o_\ell,$$

where o_j is the activation of a neuron in an arbitrary layer of the network and L is the set of all neurons in the layer, prior to the the o_j's layer.

1.2 Loss function

A commonly used loss function for the given neural network type is the mean squared error. This is, for one sample (input-output pair) given by the equation

$$E = ||\vec{o} - \vec{t}||^2,$$

where $\vec{o}$ is the output layer and $\vec{t}$ are the target values for the output layer. For more input-output pairs, the MSE can be defined as

$$E = \frac{1}{n} \sum_x ||\vec{o}(x) - \vec{t}(x)||^2.$$

A better way to denote this could be

$$E = \frac{1}{n} \sum_x ||\vec{o}[x] - \vec{t}[x]||^2,$$

as the set of samples is a discrete set.

1.3 Motivation

The function of ANNs resides primarily within the weights of the connections in the network. These determine the output, as well as the future error of the neural network; it is crucial for these to be set to correct values. Setting all of these values manually would soon lead one to insanity, thus mathematical algorithms have been developed for such a daunting task. One of these is the gradient descent-based backpropagation algorithm.

1.4 Gradient descent

Gradient descent depends on two values, current position on the function and the gradient of the function at that position. Let $f(a)$ be a function at some point a, then, the gradient of the function at point a is characterized by

$$\nabla f(a) = \sum_i \frac{\partial f(a)}{\partial e_i} \hat{e}_i,$$

where e_i is the i-th coordinate and $\hat{e}_i$ is its corresponding unit vector. The steepest descent from point a is then characterized by the negative gradient of $f(a)$. With this, the iterative equation can be derived

$$a_{i+1} = a_i - \nabla f(a_i),$$

where a_i is the i-th iteration of the gradient descent. Furthermore, a rate η can be inserted before the gradient term, modifying the "step" size

$$a_{i+1} = a_i - \eta \nabla f(a_i).$$

1.5 Applying gradient descent to FF-MP ANNs - single sample

A neural network is best characterized by its loss function, weights and neuron activations at sample inputs. For a single-sample learning, the error function can be constructed from just a single input-output pair, with the target being characterized by the vector $\vec{t}$. The resulting loss function, can then, from its initial form, introduced in 1.2, be rewritten as

$$E = \sum_{\ell \in L} (o_\ell - t_\ell)^2,$$

where L is the output layer and t_ℓ is the target value for the neuron ℓ. The definition of the loss function is recursive in nature, as any neuron in the network, besides input neurons, can be characterized by the neurons in the preceding layer and the weights between these layers.

But where does on apply gradient descent? As any neuron, and the ANN as a whole can be characterized merely by the weights of the connections (and in some cases by biases), the loss function is really just a function of all these weights, thus, it would make sense to ask how much does a particular weight (or, rather, an infinitesimally small change in its value) affect the loss function of the network as a whole. It would make sense to take a derivative of the loss function with respect to the weight being studied. As all other weights are ignored, a partial derivative should be taken, specifically

$$\frac{\partial E}{\partial w_{ij}},$$

where w_{ij} is the weight of interest, connecting neurons i and j.

Differentiating the loss function directly would soon cause one a tension headache, thus the chain rule is applied. It would be ideal to be able to differentiate the loss function with respect to o_j, as it would be characterized by it on the output layer, thus

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial o_j}\frac{\partial o_j}{\partial w_{ij}},$$

is much easier to deal with. Furthermore, differentiating o_j with respect to w_{ij} also seems quite daunting. Thus the definition of o_j using n_j from the section 1.1 becomes quite useful, as it trivializes the problem. That allows us to use the chain rule once more, yielding

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial o_j}\frac{\partial o_j}{\partial n_j}\frac{\partial n_j}{\partial w_{ij}}.$$

Solving backwards, the partial derivative of n_j with respect to w_{ij} is by far the easiest. Plugging in the definition of n_j gives

$$\partial_{w_{ij}} n_j = \sum_{\ell \in L} \partial_{w_{ij}} (w_{\ell j} o_\ell).$$

Implicitly, the sum rule of a derivative was used. As can be shown, for all terms in the sum, only the term with a weight coefficient, matching that with respect to which its being differentiated to can result in a non-zero value, thus the partial derivative simplifies to and results in

$$\partial_{w_{ij}} n_j = \partial_{w_{ij}} (w_{ij} o_i) \implies \partial_{w_{ij}} n_j = o_i.$$

The other, fairly simple derivative is the partial derivative of o_j with respect to n_j

$$\partial_{n_j} o_j = \partial_{n_j} \sigma(n_j) \implies \partial_{n_j} o_j = \sigma'(n_j).$$

The actual derivative of the logistic function is not really needed for now, thus leaving it in such form will suffice.

At last, the most perplexing partial derivative of all of these, the derivative of the loss function with respect to o_j.

When the neuron j belongs to the output layer, such derivative is trivial,

$$\partial_{o_j} E = \partial_{o_j} (o_j - t_j)^2 \implies \partial_{o_j} E = 2(o_j - t_j).$$

When dealing with hidden (interior) layers, this becomes quite perplexing. One would in this case hope for some sort of recursive definition of such derivative, so that the errors from the past layers could be reused in the next (preceding) layers. Applying chain rule to the partial derivative of the loss function with respect to just one of its arguments, o_ℓ, where the neuron ℓ is a neuron to which the neuron j is an input to, gives

$$\frac{\partial E}{\partial o_\ell}\frac{\partial o_\ell}{\partial o_j}.$$

Furthermore, this can be expanded on by considering the intermediate result of the neuron activation o_ℓ

$$\frac{\partial E}{\partial o_\ell}\frac{\partial o_\ell}{\partial n_\ell}\frac{\partial n_\ell}{\partial o_j}.$$

This, however, is only the derivative of the loss function with respect to one specific next-layer neuron, with one specific argument it takes. For a complete definition, all arguments need to be considered. To do this, total derivative can be used, thus the approximation to the partial derivative of the loss function with respect to an arbitrary neuron activation in the ANN is defined as

$$\partial_{o_j} E = \sum_{\ell \in L} \frac{\partial E}{\partial o_\ell}\frac{\partial o_\ell}{\partial n_\ell}\frac{\partial n_\ell}{\partial o_j}.$$

This completes the recursive definition of the partial derivative of the loss function with respect to some neuron's activation. As can be seen, the

$$\cdots \frac{\partial E}{\partial o_\ell}\frac{\partial o_\ell}{\partial n_\ell} \cdots$$

"term" of the equation is identical to the

$$\cdots \frac{\partial E}{\partial o_j}\frac{\partial o_j}{\partial n_j} \cdots$$

"term" in the output layer derivative (when the neuron j is identical to the neuron ℓ), thus, a function can be designated for it. Let δ_j be that function, then

$$\delta_j = \begin{cases} \frac{\partial E}{\partial o_j}\frac{\partial o_j}{\partial n_j} & \text{if output neuron} \\ \sum\limits_{\ell \in L} \delta_\ell \frac{\partial n_\ell}{\partial o_j} & \text{otherwise} \end{cases} .$$

The partial derivative of n_ℓ with respect to o_j is given by

$$\partial_{o_j} n_\ell = \sum_{k \in K} \partial_{o_j} w_{k\ell} o_k ,$$

where the layer K is the layer preceding the layer of the neuron ℓ. The partial derivative

$$\partial_{o_j} w_{k\ell} o_k$$

can result in a non-zero function if and only if the neurons j and k are identical, thus

$$\partial_{o_j} n_\ell = \partial_{o_j} w_{j\ell} o_j \implies \partial_{o_j} n_\ell = w_{j\ell} .$$

The δ_j can then be reformulated as

$$\delta_j = \begin{cases} \frac{\partial E}{\partial o_j}\frac{\partial o_j}{\partial n_j} & \text{if output neuron} \\ \sum\limits_{\ell \in L} \delta_\ell w_{j\ell} & \text{otherwise} \end{cases} .$$

With all that, finally, the needed change in a weight w_{ij} can be calculated using the gradient descent equation, resulting in

$$\Delta w_{ij} = -\eta o_i \delta_j .$$

This works because the targeted descent is meant to affect the loss function in only two dimensions, that is, the dimension given by the weight w_{ij} and the output dimension.

1.6 Applying gradient descent to FF-MP ANNs - multiple samples

Given the loss function

$$E = \frac{1}{n} \sum_{x,i} (o_i(x) - t_i(x))^2 ,$$

to perform the gradient descent on w_{ij}, δ_j must be calculated for all samples. In this case, then, it must be defined as a function of some sample x,

$$\delta_j(x) = \begin{cases} \frac{\partial E}{\partial o_j(x)} \frac{\partial o_j(x)}{\partial n_j(x)} & \text{if output neuron} \\ \sum_{\ell \in L} \delta_\ell(x) w_{j\ell} & \text{otherwise} \end{cases}.$$

Every neuron's activation is now also considered a function of some sample x.

The partial derivative of the loss function with respect to $o_j(x)$ is calculated almost exactly the same as in the previous section (except the neuron activation $o_j(x)$ is now explicitly a function of x). The i in the second sum must match j for the term to produce a non-zero result when differentiated

$$\partial_{o_j(x)} E = \frac{1}{n} \partial_{o_j(x)} (o_j(x) - t_j(x))^2 \implies \partial_{o_j(x)} E = \frac{2}{n}(o_j(x) - t_j(x)).$$

The partial derivative of the neuron's activation with respect to its raw value is pretty much the same as in the previous section

$$\partial_{n_j(x)} o_j(x) = \sigma'(n_j(x)).$$

Since the loss function gets averaged before the calculations of the partial derivatives, the change in a weight w_{ij} can simply be written as the sum of all gradients over all inputs

$$\Delta w_{ij} = -\eta \sum_x o_i(x) \delta_j(x).$$

1.7 Reflections

The derived equations for multisample learning must necessarily produce output, identical to those for single sample output when given only one sample. It can be shown that this is true, because when $X = \{u\}$, then

$$\sum_{x \in X} a(x) \to a(u),$$

where u is the first and only element in the set X. This can be applied to the change in weight equation

$$-\eta \sum_{x \in X} o_i(x) \delta_j(x) \to -\eta o_i(u) \delta_j(u).$$

Now that the input can be ignored, the equation becomes

$$\Delta w_{ij} = -\eta o_i \delta_j .$$

Unless incorrect assumptions were made, it can be seen that equations for multisample learning result in equations for single sample learning when only single sample is provided.

2 Backpropagation in complete neural networks

The previous section deals with reduced neural networks. These lack activation biases, which are fairly common in multilayer perceptrons.

2.1 Initial concepts... revisited

The activation function of the j-th neuron from the section 1.1 can be extended by adding activation bias b_j to the raw activation of the neuron

$$o_j = \sigma(n_j),$$
$$n_j = \left(\sum_{\ell \in L} w_{\ell j} o_\ell \right) - b_j .$$

2.2 Applying chain rule... again

To be able to find the gradient descent step, a partial derivative of the loss function with respect to the variable of interest must be taken. In this case, that is the partial derivative of the loss function with respect to the activation bias, $\partial_{b_j} E$. Using chain rule, this can be expanded to

$$\partial_{b_j} E = \partial_{o_j} E \cdot \partial_{b_j} o_j ,$$

this can be done because o_j depends on the activation bias. Applying the chain rule once again gives

$$\partial_{b_j} E = \partial_{o_j} E \cdot \partial_{n_j} o_j \cdot \partial_{b_j} n_j .$$

This is quite nice, because

$$\partial_{o_j} E \cdot \partial_{n_j} o_j = \delta_j ,$$

meaning the only remaining partial derivative is $\partial_{b_j} n_j$, which is quite simple to calculate

$$\partial_{b_j} n_j = \partial_{b_j}\left(\sum_{\ell\in L} w_{\ell j} o_\ell\right) - \partial_{b_j} b_j \implies \partial_{b_j} n_j = -1.$$

Knowing all these partial derivatives, the gradient descent with respect to an activation bias of the network, then, is given by

$$\Delta b_j = \eta \sum_x \delta_j(x).$$

The change in notation only occurs because the network should perform gradient descent over b_j based on all of the samples x.

Footnotes

Logistic curve

Sigma prime, $\sigma'(x)$ refers to the partial derivative of the sigmoid/logistic function with respect to its input. The logistic function is defined as

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

with the range

$$R = (0,1).$$

This is because as x approaches infinity, the logistic function approaches one, or

$$\lim_{x\to\infty}\left(\frac{1}{1+e^{-x}}\right) = \frac{1}{1+\lim\limits_{x\to\infty} e^{-x}},$$
$$\lim_{x\to\infty} e^{-x} = 0 \implies \lim_{x\to\infty}\sigma(x) = \frac{1}{1+0} = 1.$$

The same can be (but will not be) demonstrated for $\lim\limits_{x\to-\infty}\sigma(x) = 0$.

Due to data type limitations, the range of the logistic function when implemented in code is

$$R = [0,1].$$

This technically makes the function discontinuous at those *breaking points*, but despite that, its derivative at and beyond those points is considered to be identical to the derivative of the *exact* logistic function.

Vectorized equations

Consider a matrix $\mathbf{W}$ with rows, corresponding with the neurons in the next layer and columns, corresponding with the neurons in the current layer. Let the weight matrix $\mathbf{W}_c$ be a matrix of connections between neurons in the c-th and n-th layer. The equations from the section 1.6 can, in a vectorized form, be expressed as

$$\vec{\delta}_c(x) = \begin{cases} \frac{2}{n}(\vec{o}_c(x) - \vec{t}_c(x)) \circ \vec{o}_c(x) \circ (\vec{1} - \vec{o}_c(x)) & \text{if output neuron} \\ \mathbf{W}_c^\top \vec{\delta}_n & \text{otherwise} \end{cases},$$

$$\Delta \mathbf{W}_c = -\eta \sum_x \vec{\delta}_n(x) \otimes \vec{o}_c(x).$$

The vectorized equation for biases can be expressed as

$$\Delta \vec{b}_c = \eta \sum_x \vec{\delta}_c(x).$$