Machine Learning for Autonomous Robots

Tutorial ANN I / Backpropagation

Melvin Laux

21.01.2021

1 Backpropagation: Mathematical derivation

Given the error for the n^{th} training example

$$E^{(n)} = \frac{1}{2}\sum_{k=1}^{K}(y_k^{(n)} - t_k^{(n)})^2, \tag{1}$$

where $y^{(n)}$ is the output of the neural net for the n^{th} training example, $t^{(n)}$ is the expected output and both are vectors with K real valued components. To minimise the error, we perform (Stochastic) Gradient Descent. We obtain the gradient by deriving the error $E^{(n)}$ with respect to each weight w_{ji}:

$$\nabla_w E^{(n)} = \begin{pmatrix} \vdots \\ \frac{\partial E^{(n)}}{\partial w_{ji}} \\ \vdots \end{pmatrix}. \tag{2}$$

For Batch Gradient Descent, the average gradient over all N training examples is used to update the weights:

$$\nabla_w E = \frac{1}{N}\sum_{n=1}^{N}\nabla_w E^{(n)}. \tag{3}$$

For Stochastic (or Minibatch) Gradient Descent, the order of the training set is randomised and in each training step the gradient for only a subset of size M is computed (Stochastic Gradient Descent often refers to the choice $M = 1$).

The weights are updated after each step by applying the negative gradient scaled by a learning rate $\eta > 0$:

$$w_{ji} \leftarrow w_{ji} - \eta\frac{\partial E}{\partial w_{ji}}. \tag{4}$$

The partial derivative of the error function with respect to a weight is obtained by applying the chain rule:

$$\frac{\partial E^{(n)}}{\partial w_{ji}} = \frac{\partial E^{(n)}}{\partial a_j} \frac{\partial a_j}{\partial w_{ji}} \tag{5}$$

At this point, it is useful to take a closer look at what is actually happening inside the neural network. Consider a single unit x_j which receives its input from several units x_i as shown below:

$$\begin{tikzcd}
x_1 \arrow[dr, "w_{j1}"] & \\
\vdots \arrow[r, "w_{ji}"] & x_j \\
x_n \arrow[ur, "w_{jn}"'] &
\end{tikzcd}$$

The value of x_j is the weighted sum of all its inputs a_j with an activation function g applied:

$$\begin{aligned} x_j &= g(a_j) \\ a_j &= \sum_i w_{ji} x_i. \end{aligned} \tag{6}$$

The error function E depends on each weight w_{ji} (this is why we can derive it with respect to each weight) but it also does it depend on the activations a_j. As a_j also depends on w_{ji}, we can extend the partial derivative by applying the chain rule as outlined in Equation 5.

Given the formula in Equation 5, the partial derivative of the error with respect to each weight consists of three components:

1. Compute $\frac{\partial a_j}{\partial w_{ji}}$
2. Compute $\delta_i = \frac{\partial E^{(n)}}{\partial a_i}$ for hidden units x_i and
3. Compute $\delta_k = \frac{\partial E^{(n)}}{\partial a_k}$ for output units x_k

Step 1: $\frac{\partial a_j}{\partial w_{ji}}$: The partial derivative of the activation a_j with respect to the weight w_{ji} is straightforward:

$$\begin{aligned} \frac{\partial a_j}{\partial w_{ji}} &= \frac{\partial \left(\sum_{i'} w_{ji'} x_{i'} \right)}{\partial w_{ji}} \\ &= \frac{\partial \left(w_{j1} x_1 + w_{j2} x_2 + \cdots + \boldsymbol{w_{ji} x_i} + \cdots \right)}{\partial w_{ji}} \\ &= x_i \end{aligned} \tag{7}$$

Step 2: $\delta_i = \frac{\partial E^{(n)}}{\partial a_i}$ for a hidden unit x_i: To obtain the derivative of the error function with respect to the activation a_i of a hidden unit the chain rule and the sum rule for derivatives has to be applied:

$$\begin{aligned}\delta_i = \frac{\partial E^{(n)}}{\partial a_i} &= \sum_j \frac{\partial E^{(n)}}{\partial a_j} \frac{\partial a_j}{\partial a_i} \\ &= \sum_j \delta_j \frac{\partial a_j}{\partial a_i}\end{aligned} \tag{8}$$

As obviously $a_j = \sum_{i'} w_{ji'} x_{i'} = \sum_{i'} w_{ji'} g(a_{i'})$ it follows

$$\begin{aligned}\frac{\partial a_j}{\partial a_i} &= \frac{\partial \left(\sum_{i'} w_{ji'} g(a_{i'})\right)}{\partial a_i} \\ &= \frac{\partial \left(w_{j1} g(a_1) + w_{j2} g(a_2) + \cdots + \boldsymbol{w_{ji} g(a_i)} + \cdots\right)}{\partial a_i} \\ &= g'(a_i) w_{ji}\end{aligned} \tag{9}$$

and therefore

$$\delta_i = \sum_j \delta_j g'(a_i) w_{ji} \tag{10}$$

$$ = g'(a_i) \sum_j \delta_j w_{ji} \tag{11}$$

Step 3: $\delta_k = \frac{\partial E^{(n)}}{\partial a_k}$ for an output unit y_k: By again applying the chain rule to the partial derivative, we formulate:

$$\begin{aligned}\delta_k = \frac{\partial E^{(n)}}{\partial a_k} &= \frac{\partial E^{(n)}}{\partial y_k} \frac{\partial y_k}{\partial a_k} \\ &= \frac{\partial E^{(n)}}{\partial y_k} \frac{\partial g(a_k)}{\partial a_k} \\ &= \frac{\partial E^{(n)}}{\partial y_k} g'(a_k)\end{aligned} \tag{12}$$

It is common to use the identity function $g(a_k) = a_k$ as activation function in the output layer. The derivative of the identity function is $g'(a_k) = 1$ and therefore (omitting the index for the training sample n)

$$
\begin{aligned}
\delta_k &= \frac{\partial E^{(n)}}{\partial y_k} \\
&= \frac{\partial \left(\frac{1}{2}\sum_{k'}(y_{k'} - t_{k'})^2\right)}{\partial y_k} \\
&= \frac{\partial \left(\frac{1}{2}((y_1 - t_1)^2 + (y_2 - t_2)^2 + \cdots + (y_k - t_k)^2 + \cdots)\right)}{\partial y_k} \\
&= \frac{\partial \left(\frac{1}{2}(y_1^2 - 2y_1t_1 + t_1^2 + y_2^2 - 2y_2t_2 + t_2^2 + \cdots + y_k^2 - 2y_kt_k + t_k^2 + \cdots)\right)}{\partial y_k} \\
&= \frac{\partial \left(\frac{1}{2}y_1^2 - y_1t_1 + \frac{1}{2}t_1^2 + \frac{1}{2}y_2^2 - y_2t_2 + \frac{1}{2}t_2^2 + \cdots + \boldsymbol{\frac{1}{2}y_k^2 - y_kt_k + \frac{1}{2}t_k^2} + \cdots\right)}{\partial y_k} \\
&= y_k - t_k
\end{aligned}
\tag{13}
$$

which is the difference between the networks output and the expected value for the k^{th} output unit.

2 Algorithm

The training of an artificial neuronal network consists of repeating the following two steps in a loop until convergence:

1. Forward propagation (forward pass)
2. Backpropagation (backward pass)

For the forward pass, starting from the input layer, the activation a_j for each unit in the following layer has to be computed as weighted sum of all the current layers units x_i. The value of the unit x_j in the following layer is then $g(a_j)$ where $g : \mathbb{R} \to \mathbb{R}$ is some activation function.

After the output layer is reached, compute

$$\delta_k = y_k - t_k \qquad \text{For each output unit } y_k \tag{14}$$

$$\delta_i = g'(a_i)\sum_j \delta_j w_{ji} \qquad \text{For each hidden unit } x_i \tag{15}$$

and for each weight w_{ji} in the network

$$\frac{\partial E^{(n)}}{\partial w_{ji}} = \delta_j x_i \tag{16}$$

and update the weights depending on the chosen algorithm (Gradient Descent or Stochastic Gradient Descent) based on equation 4.

Note that each δ_i depends on the δ_j values from the following layer (which themselves depend on the δ values of the next layer after that and so on), therefore the computation has to be performed starting from the output layer.

3 Vanishing gradients

Mind the recursive definition of 15. It is basically a product of derivatives of the activation function in each layer with the corresponding weights. As the majority of common activation functions have a derivative with values between 0 and 1, this product can get very close to 0 for lower layers in a network with many layers. This means that the weights in the lower layers are not affected any more by the error. This problem is called **vanishing gradient** and will be discussed in the deep learning lecture next Thursday.

4 Implementation

For an actual implementation of the forward and backward pass within a neural network, we would like to use a method that utilizes modern computer hardware effeciently. As such modern computers are especially good at linear algebra operations like matrix multiplication and addition, due to hardware support (vector operations in CPUs and in general highly parallel computation on GPUs) as well as effecient software libraries for such operations, it is recommended to align our data structures accordingly. In the following section an overview is provided on how to do that.

4.1 The forward pass

For the forward pass, we can compute all activations $\boldsymbol{A}$ for several inputs at once, by multiplying a matrix $\boldsymbol{X}$ containing an input sample per row with a matrix $\boldsymbol{W}$ holding all the weights of that layer. Accordingly we receive the outputs of that layer $\boldsymbol{Y}$ (for all input samples) by applying the layer's activation function to $\boldsymbol{A}$ elementwise.

$$\boldsymbol{A} = \boldsymbol{X}\boldsymbol{W}^T \tag{17}$$

$$\boldsymbol{Y} = g(\boldsymbol{A}) \tag{18}$$

where

- $\boldsymbol{Y} \in \mathbb{R}^{N \times J}$ contains an output vector in each row
- $\boldsymbol{X} \in \mathbb{R}^{N \times I}$ contains an input vector in each row
- $\boldsymbol{W} \in \mathbb{R}^{J \times I}$ is the weight matrix of the layer, $\boldsymbol{W}_{ji}$ is the weight between input i and output j.

We make those computations starting from the input layer and ending at the output layer. The output of the previous layer becomes the input of the following layer.

The error function then becomes

$$E = \frac{1}{2}||\boldsymbol{Y} - \boldsymbol{T}||_2^2 \tag{19}$$

4.2 Backpropagation

For the backwards pass, start on the output layer and compute

$$\Delta = g'(\boldsymbol{A}) * \frac{\partial E}{\partial \boldsymbol{Y}} \tag{20}$$

where $\boldsymbol{Y} - \boldsymbol{T} = \frac{\partial E}{\partial \boldsymbol{Y}}$. Note that $*$ is the elementwise multiplication of two matrices. From Δ we get the gradient for the weights of the current layer directly by

$$\frac{\partial E}{\partial \boldsymbol{W}} = \Delta^T \cdot \boldsymbol{X} \tag{21}$$

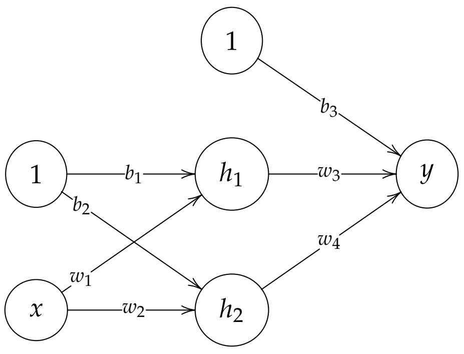
proceeding backwards following the topology of the neural network we need the partial derivatives of the error function with respect to output of the previous layer, which are equivalent to the gradients of the error function with respect to the inputs of the current layer (as the current inputs are the previous outputs). Those can again be calculated in one step by

$$\frac{\partial E}{\partial \boldsymbol{X}} = \Delta \cdot \boldsymbol{W} \tag{22}$$

and then directly be used in equation 20 for the previos layer.

5 Exercise

Consider the following simple network with one hidden layer:



With the following initial weights set: The activation function for the hidden layer is the ReLU function:

Weight	Value
w_1	1
w_2	2
b_1	-1
b_2	1
w_3	-3
w_4	4
b_3	1

$$g_h(a) = \max\{0, a\}$$

The activation function for the output layer is the identity function:

$$g_o(a) = a$$

Note that the input and the hidden layer have a bias node always taking the value 1.

Exercise:

1. For a simple training set having only one example $T = \{(1,1)\}$ perform the forward pass and calculate the value of each unit.
2. Compute the error E.
3. Perform the backpropagation algorithm for this one training example. Set the learning rate to $\eta = 0.1$.

Note: You do not need do any programming for this exercise, all values can be computed manually.

Solution:

1. **Note:** x and t are from the training data.

$$\begin{aligned} x &= 1 \\ t &= 1 \\ h_1 &= g_h(b_1 + w_1 x) = g_h(-1+1) = g_h(0) = 0 \\ h_2 &= g_h(b_2 + w_2 x) = g_h(1+2) = g_h(3) = 3 \\ y &= g_o(b_3 + w_3 h_1 + w_4 h_2) = g_o(1+0+12) = 13 \end{aligned}$$

2.

$$E = \frac{1}{2}(y-t)^2 = \frac{1}{2}(13-1)^2 = 72$$

3. First the derivatives of the activation function for the hidden layer:

$$g_h(a)' = \begin{cases} 1 & \text{if } a > 0 \\ 0 & \text{else} \end{cases}$$

We then calculate the gradient starting from the output layer backwards. First δ_y:

$$\delta_y = y - t = 13 - 1 = 12$$

with δ_y we can calculate the gradient for the weights between hidden and output layer.

$$\frac{\partial E}{\partial b_3} = \delta_y * 1 = 12$$
$$\frac{\partial E}{\partial w_3} = \delta_y * h_1 = 12 * 0 = 0$$
$$\frac{\partial E}{\partial w_4} = \delta_y * h_2 = 12 * 3 = 36$$

As next stept we calculate the deltas for the weights between input and hidden layer:

$$\delta_{h_1} = g_h'(a_{h_1})\delta_y w_3 = 0 * 12 * (-3) = 0$$
$$\delta_{h_2} = g_h'(a_{h_2})\delta_y w_4 = 1 * 12 * 4 = 48$$

And finally the gradients for the remaining weights:

$$\frac{\partial E}{\partial b_1} = \delta_{h_1} * 1 = 0$$
$$\frac{\partial E}{\partial b_2} = \delta_{h_2} * 1 = 48$$
$$\frac{\partial E}{\partial w_1} = \delta_{h_1} * x = 0$$
$$\frac{\partial E}{\partial w_2} = \delta_{h_2} * x = 48$$

The remaining step, substracting the calculated gradient weighted by η from the weights, is left to the interested reader.