AI Essentials: Backpropagation

- Author: Christian M.M. Frey (web) - (email)

- Deep Learning Basics - Backpropagation

Content

Notation

- X: training set consisting of m samples $((x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)}))$
- $x^{(i)}$: denotes the i-th sample of the training set
- $y^{(i)}$: true value of the i-th sample
- $\hat{y}^{(i)}$: predicted value of the i-th sample
- w_{ij}^{k}: weight for link connecting the i-th neuron on layer $k-1$ with the j-th neuron on layer k
- h_i^k: result of affine transformation of neuron i in the k-th layer
- a_i^k: output of node i in the k-th layer (i.e., result of activation function on h_i^k)
- g_h : activation function for a hidden layer
- g_o : activation function for the output layer
- $\mathcal{E}_d$: error calculated by using the d-th sample
- $\mathcal{E}$: total error
- δ_j^k: error term for the j-th neuron on the k-th layer
- N_h: number of neurons on the h-th layer

What is backpropagation?

Backpropagation is a training method using gradients of a neural network to adjust the weights of a network. This adjustment is being made in order to minimize the global error of a neural network as it is trained. This is achieved by descending down the gradients to lower values. For the training procedure itself there are two ways to distinguish: *Online* and *batch training*.

- **Online Training.** The weights are adjusted after every sample in the training set. Hence, the gradients being calculated for the first element in the training set used and the weights are modified. Then, the training progresses to the next sample in the training set and updates the neural network according to that. Following this procedure, the training continues until every sample in the training set has been processed. At this point, we call an epoch has being completed.
- **Batch Training.** The gradients for each sample in the training set are aggregated. Then, the weights of the neural network are updated. At this point, we call an epoch has being completed.

Although the online training method was the original way to go for backpropagation, it is nowadays considered as being inefficient because of its numerous updates. Therefore, it is generally preferable

to use (mini-)batch training. For a mini-batch the training set is subdivided into chunks that are used for training, e.g., considering a set of $60'000$ samples in your training set and one chooses to update the weights every $1'000$ samples (=mini-batch), this would lead to 60 updates for one epoch being completed.
The heuristic of backpropagation follows the idea of proceeding from the last layer to the first layer, where we calculate an error term δ_j^k for each of the j-th node in the k-th layer which is dependent on the errors on the $k+1$-th layer. The error terms on the output layer are defined by a predefined cost function $\mathcal{E}$ expressing the divergence from the predicted value $\hat{y}$ from the true value y. The backpropagation got its name from the procedure of the errors flowing backwards, i.e., in the inverse direction - compared to the forward pass - starting from the output layer to the first layer. The error terms for neurons in the h-th hidden layer are computed by a linear combination of the weights w_{jk}^{h+1} with the error terms δ_k^{h+1}. The result is then scaled by the derivative of the activation function being used on the h-th layer, i.e., $g_h'(\cdot)$. This procedure repeats until for each link we have computed an indicator on how to update its value, or in other words, till we reach the input layer. Because the computations for backpropagation are dependent on the activations and outputs of the neurons, the forward pass precedes the backward pass and stores those values. When the backward pass is completed and all partial derivatives with respect to the weights are known, the weights can be updates by gradient descent.
The alternating interplay of forward passes and backward passes in every iteration of gradient descent forces the neural net to reduce the error term till it has found a local minimum. Then, the neural net has learned a high dimensional non-linear function being an approximation for the unknown data-generating process having generated the dataset X.

What are the central formulas i need to know?

The following summarizes the formulas being used in the procedure of backpropagation using the error function having been used in classic backpropagation, the mean squared error:

$$\mathcal{E} = \frac{1}{2m} \sum_{d=1}^{m} (y^{(d)} - y^{(d)})^2$$

(i) Partial derivatives w.r.t to weight w_{ij}^k:

$$\frac{\partial \mathcal{E}_d}{\partial w_{ij}^k} = \delta_j^k a_i^{k-1} \tag{1}$$

(ii) Error term of the last layer

$$\delta_i^m = g_o'(h_i^m)(\hat{y}^{(d)} - y^{(d)}) \tag{2}$$

(iii) Summing up the partial derivatives of each sample

$$\frac{\partial \mathcal{E}}{\partial w_{ij}^k} = \frac{1}{m} \sum_{d=1}^{m} \frac{\partial \mathcal{E}_d}{\partial w_{ij}^k} = \frac{1}{m} \sum_{d=1}^{m} \frac{\partial}{\partial w_{ij}^k} \left[\frac{1}{2} (\hat{y}^{(d)} - y^{(d)})^2 \right] \tag{3}$$

(iv) Error terms of hidden layers:

$$\delta_j^h = g_h'(h_j^h) \sum_{k=1}^{N_{h+1}} \delta_k^{h+1} w_{jk}^{h+1} \tag{4}$$

(v) Delta rule - updating the weights:

$$w_{ij}^k \leftarrow w_{ij}^k - \eta \frac{\partial \mathcal{E}}{\partial w_{ij}^k} \tag{5}$$

What is general algorithm for learning a neural net?

The steps for training a neural net with (mini-)batches can be summarized as follows:

(i) **Forward pass.** For each input $(x^{(d)}, y^{(d)})$ we calculate the values h_j^k and a_j^k for each node j in the k-th layer and finally the output $\hat{y}^{(d)}$ on the output layer.

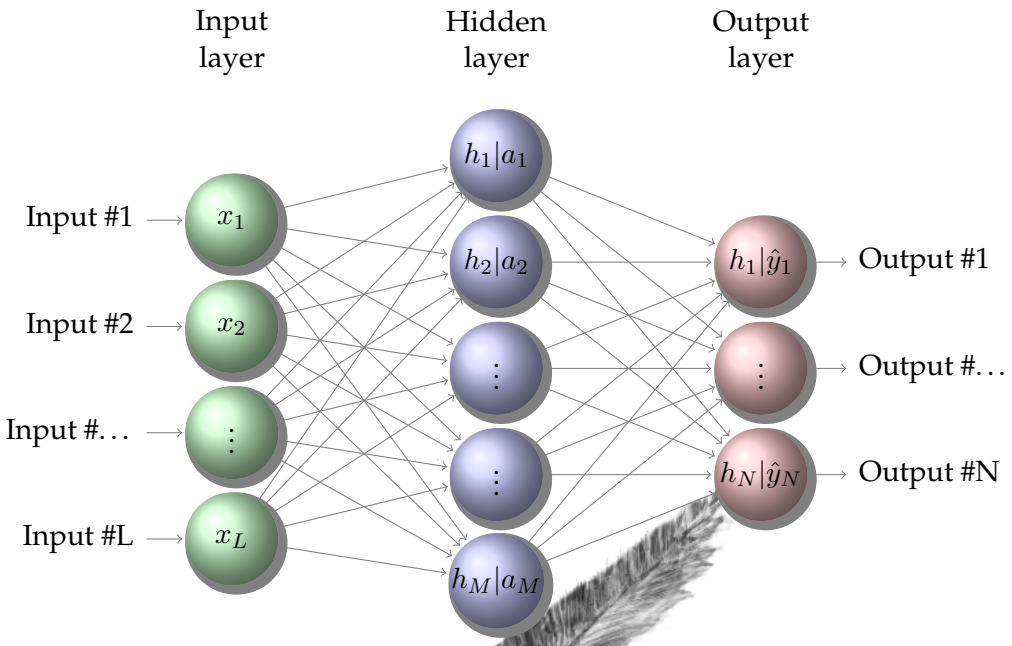
(ii) **Backward pass.** For each input $(x^{(d)}, y^{(d)})$ we calculate the gradients $\frac{\partial \mathcal{E}_d}{\partial w_{ij}^k}$ for each link proceeding from the output layer to the input layer.

(iii) **Aggregation of individual gradients.** Calculation of the total gradient $\frac{\partial \mathcal{E}}{\partial w_{ij}^k}$ for all samples in the training set X.

(iv) **Adaptation of weights.** Update of the weights according to learning rate η and the gradient $\frac{\partial \mathcal{E}}{\partial w_{ij}^k}$.

Example: Backpropagation for a simple 2-layer feedforward neural net

For the sake of simplicity, we will have a look at the process of backpropagation within a feed forward neural network consisting of an input layer, a single hidden layer and an output layer. The input layer is of size L, the hidden Layer of size M and the output layer of size N. When we also include the bias, then there are $(L+1) \times M$ weights for the connections between the input layer and the output layer being denoted as $\boldsymbol{v}$, where v_{ij} indicates the weight for the connections between neuron i of the input layer and the neuron j of the hidden layer. Analogously, the network has $(M+1) \times N$ weights for the links between the hidden layer and the output layer being denoted as $\boldsymbol{w}$ and w_{jk} indicates the link of neuron j of the hidden layer to neuron k in the output layer. The following illustration summarizes the architecture of the network (without biases):



Given the architecture above, we can summarize the steps for the prediction for an single sample x and its label y, i.e., following the idea of online training, as follows:

1. $h_j^h = \sum_{i=0}^{L} x_i v_{ij}$
2. $a_j^h = g_h(h_j^h)$
3. $h_k^o = \sum_{j=0}^{M} a_j^h w_{jk}$
4. $\hat{y}_k = g_o(h_k^o)$

where the h_j^h indicates the affine transformation of weights $(v_{ij})_{i=0,\dots,L}$ (bias included) and the input x for neuron j in the hidden layer, whereas h_k^o indicates the affine transformation for neuron k in the output layer being calculated by the weights $(w_{jk})_{j=0,\dots,M}$ (bias included) and the outputs from neurons of the hidden layer. The function $g_h(\cdot)$ and $g_o(\cdot)$ indicates the activation functions of the hidden layer, respectively, of the output layer.
In this example, we will use the sum-of-squares as error function calculating the difference between y_k and its prediction $\hat{y}_k$ for each neuron k in the output layer:

$$\mathcal{E} = \frac{1}{2}\sum_{k=1}^{N}(\hat{y}_k - y_k)^2$$

Next, we will define learning rules for the weights $\boldsymbol{v}$ and $\boldsymbol{w}$. These are given by:

(i) $v_{ij} \leftarrow v_{ij} - \eta \frac{\partial \mathcal{E}}{\partial v_{ij}}$

(ii) $w_{jk} \leftarrow w_{jk} - \eta \frac{\partial \mathcal{E}}{\partial w_{jk}}$

BACKPROPAGATION - LEARNING RULES FOR $\boldsymbol{w}$((II)).
We will derive the update rule with respect to weight w_{jk}. Using the chain rule (q.v. *AI Essentials: Derivatives*) we get:

$$\frac{\partial \mathcal{E}}{\partial w_{jk}} = \frac{\partial \mathcal{E}}{\partial \hat{y}_k}\frac{\partial \hat{y}_k}{\partial h_k^o}\frac{\partial h_k^o}{\partial w_{jk}} \tag{6}$$

We will start with the innermost term when applying the chain rule, i.e., start with the last term of equation 6:

$$\frac{\partial h_k^o}{\partial w_{jk}} = \frac{\partial \sum_{j=0}^{M} a_j^h w_{jk}}{\partial w_{jk}} \overset{\text{Derivative of a sum is the sum of its derivatives}}{=} \sum_{j=0}^{M} \frac{\partial a_j^h w_{jk}}{\partial w_{jk}} = a_j^h \tag{7}$$

Next, we will have a look at the second term:

$$\frac{\partial \hat{y}_k}{\partial h_k^o} = \frac{\partial}{\partial h_k^o} g_o(h_k^o) = g_o'(h_k^o) \tag{8}$$

The partial derivative is dependent on the activation function we have chosen. As we can see here, the choice of activation function is a crucial point in learning a neural network. For example, suppose $g_o(\cdot)$ is the sigmoid function $g_o(z) = (1 + e^{-z})^{-1}$, then we would get (q.v. *AI Essentials: Activation Functions*):

$$\frac{\partial}{\partial h_k^o}\left[\frac{1}{1+e^{-h_k^o}}\right] = \frac{1 \cdot -e^{-h_k^o}(-1)}{(1+e^{-h_k^o})^2} = \frac{1}{(1+e^{-h_k^o})} \cdot \frac{1+e^{-h_k^o}-1}{(1+e^{-h_k^o})} = \phi(h_k^o)(1-\phi(h_k^o)) \tag{9}$$

Last, the first term results in the following:

$$\frac{\partial \mathcal{E}}{\partial \hat{y}_k} = \frac{\partial}{\partial \hat{y}_k}\left[\frac{1}{2}\sum_{k=1}^{N}(\hat{y}_k - y_k)^2)\right] = \hat{y}_k - y_k \tag{10}$$

We can now summarize the terms in order to get the update rule for w_{jk}:

$$w_{jk} \leftarrow w_{jk} - \eta \frac{\partial \mathcal{E}}{\partial w_{jk}} = w_{jk} - \eta(\hat{y}_k - y_k) g_o'(h_k^o) a_j^h = w_{jk} - \eta \delta_k^o a_j^h \tag{11}$$

where $(\hat{y}_k - y_k) g_o'(h_k^o)$ is substituted by δ_k^o.

BACKPROPAGATION - LEARNING RULES FOR v ((1)).
Next, we will derive the update rule with respect to weight v_{ij}. Note that for the links v, we have to pay attention on all the neurons in the output layer, in other words, each output neuron contributes to the weighted product sum of the weights connecting the hidden layer with the output layer and the error terms being calculated in the output layer. Therefore, we have to sum over all N neurons. Using the chain rule (q.v. *AI Essentials: Derivatives*), we get:

$$\frac{\partial \mathcal{E}}{\partial v_{ij}} = \left[\sum_{k=1}^{N} \frac{\partial \mathcal{E}}{\partial \hat{y}_k} \frac{\partial \hat{y}_k}{\partial h_k^o} \frac{\partial h_k^o}{\partial a_j^h} \frac{\partial a_j^h}{\partial h_j^h} \frac{\partial h_j^h}{\partial v_{ij}}\right] \tag{12}$$

We will start again with the innermost term of equation 12:

$$\frac{\partial h_j^h}{\partial v_{ij}} = \frac{\partial}{\partial v_{ij}} \sum_{i=0}^{L} x_i v_{ij} = x_i \tag{13}$$

The next term is again dependent on an activation function $g_h(\cdot)$ (q.v. *AI Essentials: Derivatives* for an overview of derivatives for various activation functions):

$$\frac{\partial a_j^h}{\partial h_j^h} = \frac{\partial}{\partial h_j^h} g_h(h_j^h) = g_h'(h_j^h) \tag{14}$$

For example, considering $g_h(z) = (1 + e^{-z})^{-1}$ (sigmoid function), we would get:

$$\frac{\partial}{\partial h_j^h}\left[\frac{1}{1+e^{-h_j^h}}\right] = \sigma(h_j^h)(1-\sigma(h_j^h)) \tag{15}$$

The term in the middle evaluates to:

$$\frac{\partial h_k^o}{\partial a_j^h} = \frac{\partial}{\partial a_j^h}\left[\sum_j^M a_j^j w_{jk}\right] = w_{jk} \tag{16}$$

Putting all together, we have:

$$\frac{\partial \mathcal{E}}{\partial v_{ij}} = \sum_{k=1}^{N} \frac{\partial \mathcal{E}}{\partial \hat{y}_k}\frac{\partial \hat{y}_k}{\partial h_k^o} w_{jk} g_h'(h_j^h) x_i = \sum_{k=1}^{N} \delta_k^o w_{jk} g_h'(h_j^h) x_i = g_h'(h_j^h) x_i \sum_{k=1}^{N} \delta_k^o w_{jk} \tag{17}$$

Therefore, the update rule for v_{ij} is given by:

$$v_{ij} \leftarrow v_{ij} - \eta \frac{\partial \mathcal{E}}{\partial v_{ij}} = v_{ij} - \eta g_h'(h_j^h) x_i \sum_{k=1}^{N} \delta_k^o w_{jk} = v_{ij} - \eta \delta_j^h x_i \tag{18}$$

where $g_h'(h_j^h)\sum_{k=1}^{N} \delta_k^o w_{jk}$ is substituted by δ_j^h.

BACKPROPAGATION - SUMMARY.
The steps for the backpropagation can be summarized as follows:

(i) $\delta_k^o = (\hat{y}_k - y_k) g_o'(h_k^o)$

(ii) $\delta_j^h = g_h'(h_j^h \sum_{k=1}^{N} \delta_k^o w_{jk}$

(iii) $w_{jk} \leftarrow w_{jk} - \eta \delta_k^o a_j^h$

(iv) $v_{ij} \leftarrow v_{ij} - \eta \delta_j^h x_i$