Derivation of Backpropagation Algorithm for Feedforward Neural Networks

The elements of computation intelligence

Paweł Liskowski

1 Logistic regression as a single-layer neural network

In the following, we briefly introduce binary logistic regression model. The goal of logistic regression is to correctly estimate the probability $P(y = 1 \mid \mathbf{x})$. Parameters of the model are $\mathbf{x} \in \mathbb{R}^n$ and $b \in \mathbb{R}$. Training examples are represented as n-dimensional vectors $\mathbf{x} \in \mathbb{R}^n$. We use the following notation

$$a = \sigma(\mathbf{w}^T\mathbf{x} + b)$$
$$\sigma(z) = \frac{1}{1 + \exp(-z)}$$

The loss function for binary logistic regression is

$$L(y, a) = -y \log(a) - (1 - y) \log(1 - a)$$

Given a dataset of training examples, we may learn the parameters $\mathbf{w}$ and b using gradient descent

$$w_i \leftarrow w_i + \Delta w_i$$
$$b \leftarrow b + \Delta b$$

$$\Delta w_i = -\alpha \frac{\partial L}{\partial w_i}$$
$$\Delta b = -\alpha \frac{\partial L}{\partial b}$$

Derived update rules are as follows (see also Algorithm 1)

$$\frac{\partial L}{\partial w_i} = (a - y)x_i \tag{1}$$
$$\frac{\partial L}{\partial b} = (a - y) \tag{2}$$

We now prove the (1) and (2):

$$\frac{\partial L}{\partial w_i} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial w_i} = \frac{\partial L}{\partial a}\frac{\partial a}{\partial z}\frac{\partial z}{\partial w_i}$$

Let us first derive $\frac{\partial L}{\partial a}$

$$\begin{aligned}\frac{\partial L}{\partial a} &= \frac{\partial}{\partial a}[-y \log(a) - (1 - y) \log(1 - a)] \\ &= -\frac{y}{a} + \frac{(1 - y)}{1 - a} = -\frac{y(1 - a) + a(1 - y)}{a(1 - a)} \\ &= \frac{-y + ay + a - ay}{a(1 - a)} = \frac{a - y}{a(1 - a)}\end{aligned}$$

Algorithm 1 Gradient descent for logistic regression

Require: a set of training examples D, learning rate α.

1. **Repeat** until the termination condition is met:

   Initialize Δw_i and Δb with zeros

   For each training example $(\mathbf{x}, y) \in D$ **do:**

   Propagate the input $\mathbf{x}$ forward through the model, i.e.,
   input $\mathbf{x}$ to the model and compute its output as $a = \sigma(\mathbf{w}^T\mathbf{x} + b)$.

   Accumulate updates for each weights w_i and b:

$$\Delta w_i \leftarrow \Delta w_i - \alpha(a - y)x_i$$
$$\Delta b \leftarrow \Delta b - \alpha(a - y)$$

   Update parameters of the model

$$w_i \leftarrow w_i + \Delta w_i$$
$$b \leftarrow b + \Delta b$$

Partial derivative $\frac{\partial a}{\partial z}$ is just $\sigma'(z)$

$$\begin{aligned}
\frac{\partial a}{\partial z} &= \frac{\partial}{\partial z}[1 + \exp(-z)^{-1}] \\
&= -[1 + \exp(-z)]^{-2}\exp(-z))(-1) \\
&= \frac{1}{1 + \exp(-z)}\frac{\exp(-z)}{1 + \exp(-z)} \\
&= \sigma(z)\frac{1 + \exp(-z) - 1}{1 + \exp(-z)} \\
&= \sigma(z)(\frac{1 + \exp(-z)}{1 + \exp(-z)} - \frac{1}{1 + \exp(-z)}) \\
&= \sigma(z)(1 - \sigma(z)) = a(1 - a)
\end{aligned}$$

Finding partial derivative $\frac{\partial z}{\partial w_i}$ is straightforward

$$\frac{\partial z}{\partial w_i} = \frac{\partial}{\partial w_i}(w_1x_1 + \cdots + w_ix_i + \cdots + w_nx_n) = x_i$$

Finally

$$\frac{\partial L}{\partial w_i} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial w_i} = \frac{\partial L}{\partial a}\frac{\partial a}{\partial z}\frac{\partial z}{\partial w_i} = \frac{a - y}{a(1 - a)}a(1 - a)x_i = (a - y)x_i$$
$$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial b} = \frac{\partial L}{\partial a}\frac{\partial a}{\partial z}\frac{\partial z}{\partial b} = \frac{a - y}{a(1 - a)}a(1 - a) = (a - y)$$

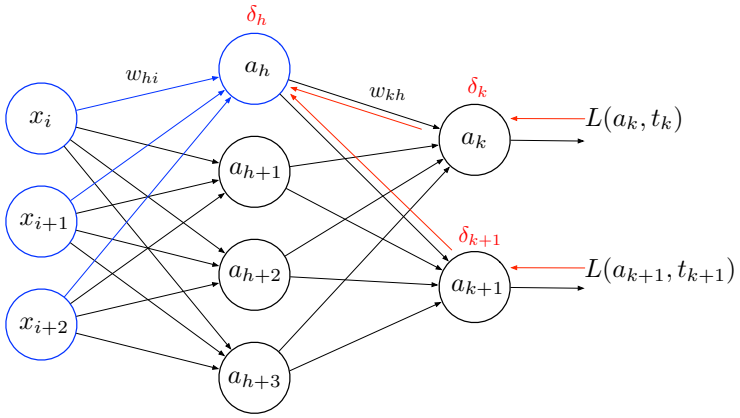

Figure 1: A simple two-layer feedforward neural network.

2 Feedforward neural networks

2.1 The model

In the following, we describe the stochastic gradient descent version of backpropagation algorithm for feedforward networks containing two layers of sigmoid units (cf. Algorithm 2). The backpropagation algorithm learns the weights of a given network. It employs gradient descent to minimize the loss function between the network outputs and the target values for these outputs. In the following, we briefly present the algorithm and derive the gradient descent weight update rules used by the algorithm.

We typically consider networks with multiple output units rather than just a single unit, therefore the cost function sums the errors over all of the network output units, i.e.:

$$J(\mathbf{w}) = \frac{1}{2} \sum_{d \in D} \sum_{k \in O} (t_{kd} - o_{kd})^2, \tag{5}$$

where O is the set out output units in the network, and t_{kd} and o_{kd} are the target and output values associated with kth output and training example d.

Notice how a feedforward neural network consists of several interconnected units (neurons), each of which can be considered as implementing logistic regression (see unit a_h and its corresponding inputs and weights marked in blue in Fig. 1).

Given a fixed structure of a neural network, the Algorithm 2 repeatedly iterates over the training examples. For each example d, it applies the network to the examples, calculates the error of the network on this example, computes the gradient with respect to the error on this example, and finally updates all weights in the network. The gradient descent step is iterated thousands of times using the same set of examples D multiple times until the network performs acceptably well.

2.2 Intuitions about backpropagation

The gradient descent rule in backprop is actually quite similar to the conventional *delta rule* for a linear unit (i.e., $\Delta w_i = \alpha(t - a)x_i$). It updates the weights in *proportion* to the learning rate α, the input x_{ji} from the node i to the node j (in other words this is the activation of ith unit a_i) to which the weight is applied, and the *error* in the output of the unit. The major difference is that the simple error term $(t - a)$ is replaced by a more complex error term δ_j. To understand it intuitively, consider how δ_k is computed for the kth output unit. The error δ_k is simply $(t_k - a_k)$ multiplied by the derivative of activation function (sigmoid in this case), i.e. $a_k(1 - a_k)$. The value δ_h for a hidden unit h is conceptually quite similar. However, since the training examples in D provide targets t_k only for the units in the output layer, there are no target values directly available to indicate the error committed by hidden units. Instead, the error term for hidden unit h is calculated by summing the error terms δ_k for each output unit *influenced* by h, and weighting each of δ_k's

Algorithm 2 Backpropagation algorithm for feedforward networks.

Require: a set of training examples D, learning rate α.

1. Create a feed-forward network with n_{in} inputs, n_h hidden units, and n_o output units.
2. Initialize all network weights to *small* random numbers.
3. **Repeat** until the termination condition is met:

   For each training example $(\mathbf{x}, \mathbf{t}) \in D$ **do:**

   Propagate the input $\mathbf{x}$ forward through the network, i.e.:
   1. Input $\mathbf{x}$ to the network and compute the output a_k of units in the output layer.

   Backpropagate the errors through the network:
   1. **For** each network output unit k, calculate its error term δ_k:

$$\delta_k \leftarrow -a_k(1 - a_k)(t_k - a_k) \tag{3}$$

   2. **For** each hidden unit h, calculate its error term δ_h:

$$\delta_h \leftarrow a_h(1 - a_h) \sum_k \delta_k w_{kh} \tag{4}$$

   3. Update each network weight w_{ji}

$$w_{ji} \leftarrow w_{ji} + \Delta w_{ji}$$

   where

$$\Delta w_{ji} = -\alpha \delta_j x_{ji}$$

by w_{kh}, the weight form hidden unit h to output unit k. In other words, the weight w_{kh} characterizes the degree to which hidden unit h is *responsible for* the error in output unit k.

2.3 Derivation of the backpropagation rule

In this section we derive the backprogation training rule. Recall that the stochastic gradient descent rule involves iterating through the examples in D, for each training example descending the gradient of the error function with respect to this example. More specifically, for each example d every weight w_{ji} is updated by adding to it Δw_{ji}

$$\Delta w_{ji} = -\alpha \frac{\partial L}{\partial w_{ji}} \tag{6}$$

where L is the error on training example d, summed over all output units in the output layer of a network

$$L = \frac{1}{2} \sum_{k \in O} (t_k - a_k)^2$$

We use the following notation:

- x_{ji} – the ith input to unit j
- w_{ji} – the weight associated with ith input to unit j
- z_j – the weighted sum of input for unit j, i.e. $z_j = \sum_i w_{ji} x_{ji}$
- a_j – the output computed by unit j, i.e. $a_j = g(z_j)$ where g is an activation function (sigmoid here)

Let us now derive $\frac{\partial L}{\partial w_{ji}}$ to implement the gradient descent rule in (6). Notice first that weight w_{ji} can influence the network's output only through z_j. Using the chain rule we can write

$$\begin{aligned}\frac{\partial L}{\partial w_{ji}} &= \frac{\partial L}{\partial z_j}\frac{\partial z_j}{\partial w_{ji}} \\ &= \frac{\partial L}{\partial z_j}x_{ji}\end{aligned}$$

Our objective is now to derive $\frac{\partial L}{\partial z_j}$. We consider two cases: the case where unit j is an output unit for the network, and the case j is an internal unit.

Case 1: unit j is an output unit

Using the chain rule, we obtain

$$\begin{aligned}\frac{\partial L}{\partial z_j} &= \frac{\partial L}{\partial a_j}\frac{\partial a_j}{\partial z_j} \\ &= \frac{\partial L}{\partial a_j}a_j(1-a_j)\end{aligned} \tag{7}$$

Notice that $\frac{\partial a_j}{\partial z_j}$ is just the derivative of our activation function (sigmoid). We now proceed with finding the derivative $\frac{\partial L}{\partial a_j}$

$$\begin{aligned}\frac{\partial L}{\partial a_j} &= \frac{\partial}{\partial a_j}\frac{1}{2}\sum_{k\in O}(t_k-a_k)^2 \\ &= \frac{\partial}{\partial a_j}\frac{1}{2}(t_j-a_j)^2 \\ &= \frac{1}{2}2(t_j-a_j)\frac{\partial}{\partial a_j}(t_j-a_j) \\ &= -(t_j-a_j)\end{aligned} \tag{8}$$

The summation term over output units is dropped because the derivatives $\frac{\partial}{\partial a_j}(t_k-a_k)^2$ will be zero for all output units k except for the case when $k=j$. By substituting (8) into (7) we obtain (3)

$$\frac{\partial L}{\partial z_j} = -(t_j-a_j)a_j(1-a_j) = \delta_j$$

As a concrete example, consider unit a_k from Fig. 1. Using the above rules, we can easily derive weight update for w_{kh}

$$\begin{aligned}\frac{\partial L}{\partial w_{kh}} &= \frac{\partial L}{\partial z_k}\frac{\partial z_k}{\partial w_{kh}} \\ &= \frac{\partial L}{\partial z_k}a_h \\ &= -(t_k-a_k)a_k(1-a_k)a_h \\ &= \delta_k a_h\end{aligned}$$

Finally

$$\Delta w_{kh} = -\alpha\frac{\partial L}{\partial w_{kh}} = -\alpha\delta_k a_h \tag{9}$$

Case 2: unit j is an internal (hidden) unit

When unit j is an internal unit we must also consider every unit immediately downstream of unit j (i.e., all units whose direct input include the output of unit j). This is because a change in w_{ji} (and there in z_j) influences the network outputs only through these units. Let $ds(j)$ denote units downstream of unit j. Then

$$\begin{aligned}\frac{\partial L}{\partial z_j} &= \sum_{k \in ds(j)} \frac{\partial L}{\partial z_k} \frac{\partial z_k}{\partial z_j} \\ &= \sum_{k \in ds(j)} \delta_k \frac{\partial z_k}{\partial a_j} \frac{\partial a_j}{\partial z_j} \qquad (10) \\ &= \sum_{k \in ds(j)} \delta_k w_{kj} a_j (1 - a_j) \qquad (11)\end{aligned}$$

Let us now use δ_j to denote $\frac{\partial L}{\partial z_j}$ which gives us (4)

$$\delta_j = a_j(1 - a_j) \sum_{k \in ds(j)} \delta_k w_{kj}$$

As an example, consider unit a_h from Fig. 1. Notice that the change in w_{hi} influences unit a_h directly and units a_k and a_{k+1} indirectly. To compute the error δ_h in unit a_h, we sum the errors δ_k and δ_{k+1} weighted by w_{kh} and $w_{k+1,h}$, respectively (see red arrows in Fig. 1). Derivation of $\frac{\partial L}{\partial w_{hi}}$ is now straightforward

$$\begin{aligned}\frac{\partial L}{\partial w_{hi}} &= \frac{\partial L}{\partial z_h} \frac{\partial z_h}{\partial w_{hi}} \\ &= \sum_{k \in ds(h)} \frac{\partial L}{\partial z_k} \frac{\partial z_k}{\partial z_j} x_i \\ &= a_h(1 - a_h) \sum_{k \in ds(h)} \delta_k w_{kh} x_i \\ &= \delta_h x_i\end{aligned}$$

Finally

$$\Delta w_{hi} = -\alpha \frac{\partial L}{\partial w_{hi}} = -\alpha \delta_h x_i \qquad (12)$$