Deep Learning - Foundations and Concepts

Chapter 8. Backpropagation

nonlineark@github

February 28, 2025

Outline

Single-layer networks

Consider a simple linear model:

$$y_k = \sum_i w_{ki} x_i$$

together with a sum-of-squares error function:

$$E_n = \frac{1}{2} \sum_k (y_k^n - t_k^n)^2$$

The gradient of this error function with respect to a weight w_{ji} is given by:

$$\frac{\partial E_n}{\partial w_{ji}} = \frac{1}{2} \sum_k 2(y_k^n - t_k^n) \frac{\partial y_k^n}{\partial w_{ji}} = (y_j^n - t_j^n) x_i^n$$

This is a local computation involving:

- An error signal $y_j^n - t_j^n$ associated with the output end.
- The variable x_i^n associated with the input end.

General feed-forward networks

Consider a unit in a feed-forward network:

$$a_j = \sum_i w_{ji} z_i \qquad z_j = h(a_j)$$

Now consider the evaluation of the derivative of E_n with respect to a weight w_{ji}:

$$\frac{\partial E_n}{\partial w_{ji}} = \sum_k \frac{\partial E_n}{\partial a_k} \frac{\partial a_k}{\partial w_{ji}} = \frac{\partial E_n}{\partial a_j} \frac{\partial a_j}{\partial w_{ji}}$$

If we introduce $\delta_j = \frac{\partial E_n}{\partial a_j}$, and use the fact that $\frac{\partial a_j}{\partial w_{ji}} = z_i$, we have:

$$\frac{\partial E_n}{\partial w_{ji}} = \delta_j z_i$$

This takes the same form as that found for the simple linear model.

General feed-forward networks

To evaluate the derivatives, we need calculate only the value of δ_j for each hidden and output unit. For output units:

$$\delta_j = \frac{\partial E_n}{\partial a_j} = \sum_k \frac{\partial E_n}{\partial y_k} \frac{\partial y_k}{\partial a_j} = y_j - t_j$$

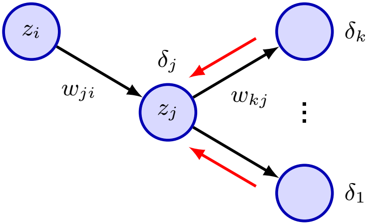
provided we are using the canonical link as the output-unit activation function. For hidden units:

$$\delta_j = \frac{\partial E_n}{\partial a_j} = \sum_k \frac{\partial E_n}{\partial a_k} \frac{\partial a_k}{\partial a_j} = \sum_k \frac{\partial E_n}{\partial a_k} \frac{\partial a_k}{\partial z_j} \frac{\mathrm{d} z_j}{\mathrm{d} a_j} = h'(a_j) \sum_k w_{kj} \delta_k$$

which tells us that the value of δ for a particular hidden unit can be obtained by propagating the δ's backwards from units higher up in the network.

General feed-forward networks

Figure: Illustration of the calculation of δ_j for hidden unit j



General feed-forward networks

Algorithm 1: Backpropagation

for $j \in$ *all hidden and output units* **do**

  $a_j \leftarrow \sum_i w_{ji} z_i$;

  $z_j \leftarrow h(a_j)$;

end

for $k \in$ *all output units* **do**

  $\delta_k \leftarrow \frac{\partial E_n}{\partial a_k}$;

end

for $j \in$ *all hidden units* **do**

  $\delta_j \leftarrow h'(a_j) \sum_k w_{kj} \delta_k$;

  $\frac{\partial E_n}{\partial w_{ji}} \leftarrow \delta_j z_i$;

end

return ∇E_n;

Numerical differentiation

One of the most important aspects of backpropagation is its $\mathcal{O}(W)$ computational efficiency. Consider an alternative approach to use finite differences with:

$$\frac{\partial E_n}{\partial w_{ji}} \approx \frac{E_n(w_{ji} + \epsilon) - E_n(w_{ji})}{\epsilon}$$

or use symmetrical central differences for better accuracy of the approximation:

$$\frac{\partial E_n}{\partial w_{ji}} \approx \frac{E_n(w_{ji} + \epsilon) - E_n(w_{ji} - \epsilon)}{2\epsilon}$$

There are W weights in the network each of which must be perturbated individually, so that the overall computational cost is $\mathcal{O}(W^2)$.

The Jacobian matrix

Here we consider the network outputs as a function of the network inputs, and calculate the Jacobian of this function:

$$J_{ki} = \frac{\partial y_k}{\partial x_i} = \sum_j \frac{\partial y_k}{\partial a_j}\frac{\partial a_j}{\partial x_i} = \sum_j w_{ji}\frac{\partial y_k}{\partial a_j}$$

$$\frac{\partial y_k}{\partial a_j} = \sum_l \frac{\partial y_k}{\partial a_l}\frac{\partial a_l}{\partial a_j} = \sum_l \frac{\partial y_k}{\partial a_l}\frac{\partial a_l}{\partial z_j}\frac{\mathrm{d}z_j}{\mathrm{d}a_j} = h'(a_j)\sum_l w_{lj}\frac{\partial y_k}{\partial a_l}$$

where in the first equation, the sum runs over all units j to which the input unit i sends connections, while for the second equation, the sum runs over all units l to which unit j sends connections. This backpropagation starts at the output units, for which the required derivatives can be found directly from the functional form of the output-unit activation function.

The Hessian matrix

Here we consider all the weight and bias parameters as elements w_i of a single vector w, and calculate the second derivatives of the error function E with respect to w:

$$H_{ij} = \frac{\partial^2 E}{\partial w_i \partial w_j}$$

Extension of the backpropagation procedure allows the Hessian matrix to be evaluated efficiently in $\mathcal{O}(W^2)$ steps. But because the huge number of parameters, evaluating the Hessian matrix for many models is infeasible. Thus there is interest in finding effective approximations to the full Hessian.

The Hessian matrix

The outer product approximation. Consider a regression application using a sum-of-squares error function of the form:

$$E(w) = \frac{1}{2}\sum_{n=1}^{N}(y_n - t_n)^2$$

$$D^2E(w) = \sum_{n=1}^{N}(Dy_n(w))^2 + \sum_{n=1}^{N}(y_n - t_n)D^2y_n(w)$$

We can ignore the second term because the quantity $y_n - t_n$ is a random variable with zero mean, which leads us to:

$$H \approx \sum_{n=1}^{N}\nabla y_n \nabla y_n^T = \sum_{n=1}^{N}\nabla a_n \nabla a_n^T$$

Gradient evaluation

There are four ways in which the gradient of a neural network error function can be evaluated:

- Derive the backpropagation equations by hand and then implement them explicitly in software: time-consuming, error prone, code redundancy.
- Finite differences: limited computational accuracy, scales poorly.
- Symbolic differentiation: expression swell, long evaluation times, requires closed form.
- Automatic differentiation: accurate, efficient, can deal with control flow elements.

Forward-mode automatic differentiation

Augment each intermediate variable v_i, known as a primal variable, with an additional variable representing the value of some derivative of that variable, which we can denote $\dot{v}_i$, known as a tangent variable.

Instead of simply doing forward propagation to compute $\{v_i\}$, the code now propagates tuples $(v_i, \dot{v}_i)$ so that variables and derivatives are evaluated in parallel.

Forward-mode automatic differentiation

Consider the following function:

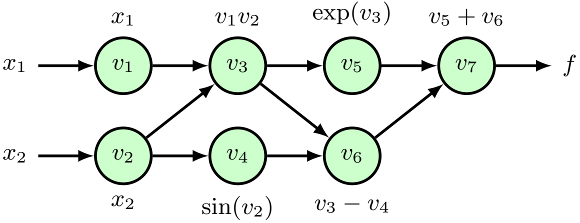
$$f(x_1, x_2) = x_1 x_2 + \exp(x_1 x_2) - \sin x_2$$

Suppose we want to evaluate the derivative $\frac{\partial f}{\partial x_1}$. Let's define the primal variables:

$$v_1 = x_1 \qquad v_2 = x_2$$
$$v_3 = v_1 v_2 \qquad v_4 = \sin v_2$$
$$v_5 = \exp(v_3) \qquad v_6 = v_3 - v_4$$
$$v_7 = v_5 + v_6$$

Forward-mode automatic differentiation

Figure: Evaluation trace diagram



Forward-mode automatic differentiation

Let's define the tangent variables by $\dot{v}_i = \frac{\partial v_i}{\partial x_1}$. Expressions for evaluating these can be constructed automatically using the chain rule:

$$\dot{v}_i = \frac{\partial v_i}{\partial x_1} = \sum_{j \in \mathrm{pa}(i)} \frac{\partial v_i}{\partial v_j} \frac{\partial v_j}{\partial x_1} = \sum_{j \in \mathrm{pa}(i)} \dot{v}_j \frac{\partial v_i}{\partial v_j}$$

where $\mathrm{pa}(i)$ denotes the set of parents of the node i in the evaluation trace diagram. Thus for the tangent variables, we have:

$$\dot{v}_1 = 1 \qquad \dot{v}_2 = 0$$

$$\dot{v}_3 = \frac{\partial v_3}{\partial v_1}\dot{v}_1 + \frac{\partial v_3}{\partial v_2}\dot{v}_2 = v_2\dot{v}_1 + v_1\dot{v}_2 \qquad \dot{v}_4 = \frac{\partial v_4}{\partial v_2}\dot{v}_2 = \dot{v}_2 \cos v_2$$

$$\dot{v}_5 = \frac{\partial v_5}{\partial v_3}\dot{v}_3 = \dot{v}_3 \exp(v_3) \qquad \dot{v}_6 = \frac{\partial v_6}{\partial v_3}\dot{v}_3 + \frac{\partial v_6}{\partial v_4}\dot{v}_4 = \dot{v}_3 - \dot{v}_4$$

$$\dot{v}_7 = \frac{\partial v_7}{\partial v_5}\dot{v}_5 + \frac{\partial v_7}{\partial v_6}\dot{v}_6 = \dot{v}_5 + \dot{v}_6$$

Forward-mode automatic differentiation

Now consider an example with two outputs:

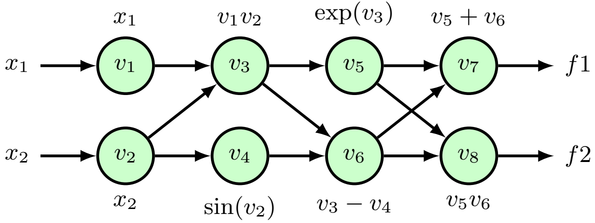
$$f_1(x_1, x_2) = x_1 x_2 + \exp(x_1 x_2) - \sin x_2$$
$$f_2(x_1, x_2) = (x_1 x_2 - \sin x_2) \exp(x_1 x_2)$$

We see that both $\frac{\partial f_1}{\partial x_1}$ and $\frac{\partial f_2}{\partial x_1}$ can be evaluated together in a single forward pass. But if we want to evaluate derivatives with respect to x_2 then we have to run a separate forward pass.

In general, if we have a function with D inputs and K outputs then a single pass of forward-mode automatic differentiation produces a single column in the Jacobian matrix. To evaluate the full Jacobian matrix, we need D forward passes. This is very efficient for networks such that $K \gg D$.

Forward-mode automatic differentiation

Figure: Evaluation trace diagram for two outputs



Reverse-mode automatic differentiation

As with forward mode, we augment each intermediate variable v_i with additional variables, in this case called adjoint variables, denoted $\bar{v}_i$. For a single output function f, we define $\bar{v}_i$ as:

$$\bar{v}_i = \frac{\partial f}{\partial v_i}$$

Using the chain rule we see that:

$$\bar{v}_i = \frac{\partial f}{\partial v_i} = \sum_{j \in \mathrm{ch}(i)} \frac{\partial f}{\partial v_j} \frac{\partial v_j}{\partial v_i} = \sum_{j \in \mathrm{ch}(i)} \bar{v}_j \frac{\partial v_j}{\partial v_i}$$

where $\mathrm{ch}(i)$ denotes the children of node i in the evaluation trace diagram.

Reverse-mode automatic differentiation

Working on the previous example again, we see that:

$$\bar{v}_7 = 1 \qquad \bar{v}_6 = \frac{\partial v_7}{\partial v_6}\bar{v}_7 = \bar{v}_7$$

$$\bar{v}_5 = \frac{\partial v_7}{\partial v_5}\bar{v}_7 = \bar{v}_7 \qquad \bar{v}_4 = \frac{\partial v_6}{\partial v_4}\bar{v}_6 = -\bar{v}_6$$

$$\bar{v}_3 = \frac{\partial v_5}{\partial v_3}\bar{v}_5 + \frac{\partial v_6}{\partial v_3}\bar{v}_6 = \bar{v}_5 \exp(v_3) + \bar{v}_6$$

$$\bar{v}_2 = \frac{\partial v_3}{\partial v_2}\bar{v}_3 + \frac{\partial v_4}{\partial v_2}\bar{v}_4 = v_1\bar{v}_3 + \bar{v}_4 \cos v_2 \qquad \bar{v}_1 = \frac{\partial v_3}{\partial v_1}\bar{v}_3 = v_2\bar{v}_3$$

In general, if we have a function with D inputs and K outputs then a single pass of reverse-mode automatic differentiation produces a single row in the Jacobian matrix. To evaluate the full Jacobian matrix, we need K reverse passes.