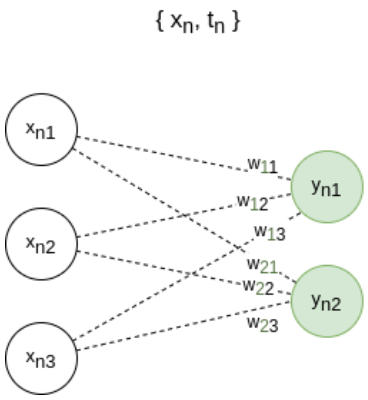
Ch.8 - Backpropagation

Created	@January 18, 2024 3:36 PM
Book	Deep Learning: Foundations and Concepts

Gradients

Single Layer



A simple linear model where the outputs $y_k = \sum_i w_{ki} x_i$ together with a sum-of-squares error function:

$$E_n = \frac{1}{2} \sum_k (y_{nk} - t_{nk})^2$$

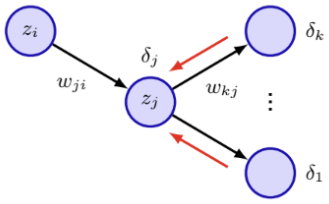
The gradient with respect to `i` th parameter associated with `j` th output unit is as follows.

$$\frac{\partial E(w)}{\partial w_{ji}} = (y_{nj} - t_{nj}) x_{ni} \tag{1}$$

This simple result can be extended to more complex neural networks.

General Network

For a generalized neural network where multiple layers are connected, we can derive a recurrent relationship in respect to pre-activation a_j and calculate $\frac{\partial E(w)}{\partial w_{ji}}$ for gradient descent.



(1) a_j , z_j, δ_j

$$a_j = \sum w_{ji} z_i,\ z_j = h(a_j),\ \delta_j = \frac{\partial E(w)}{\partial a_j}$$

where i is the previous layer, j the current, and k the next.

(2) $\partial \delta_j$

From the relationship $a_k = \sum_j w_{kj} z_j$ and $z_j = h(a_j)$, and $\frac{\partial E(w)}{\partial a_k} = \delta_k$, we can derive the recurrent equation on the right. Note that δ_j gets the error signal from all the units in the next layer, as the error function.

$$\begin{aligned}\delta_j = \frac{\partial E(w)}{\partial a_j} &= \sum_k \frac{\partial E(w)}{\partial a_k}\frac{\partial a_k}{\partial a_j} \\ &= h'(a_j)\sum_k w_{kj}\delta_k \ (\because \frac{\partial a_k}{\partial a_j} = h'(a_j) w_{kj})\end{aligned}$$

(3) ∂w_{ji}

Once we got the recurrent relationship of δ_j, it's easy to run gradient descent for parameters.

$$\frac{\partial E(w)}{\partial w_{ji}} = \frac{\partial E(w)}{\partial a_j}\frac{\partial a_j}{\partial w_{ji}} = \delta_j z_i$$

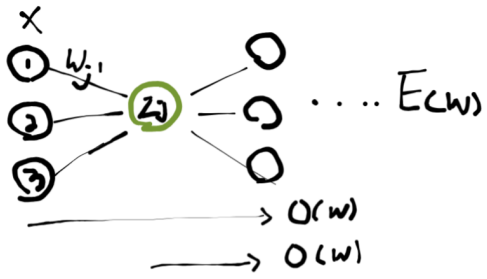
This process of sending gradient backward to adjust parameters is call ***back-propagation***.

Numerical Differentiation

An alternative way to calculate a gradient of $\frac{\partial E(w)}{\partial w_{ji}}$ is to use the definition of derivation.

$$\frac{\partial E(w)}{\partial w_{ji}} = \frac{E_n(w_{ji} + \epsilon) - E_n(w_{ji} - \epsilon)}{2\epsilon}$$

This gives a critical issue where it needs to compute $O(w)$ for the new error function for each weight, resulting in a feed-forward complexity of $O(w^2)$ instead of $O(w)$.



However, this can play a useful role to **check on the correctness** of the deep neural network.

Automatic Differentiation

There essentially are four ways to evaluate the gradient of a neural network.

Back-propagation by hand	Accurate to numerical precision
Manual derivation of gradient equations	Hard to adapt to software change, prone to error
Numerical differentiation	No need to implement back-propagation equations
Using the definition of derivation	Poor time complexity of $O(W^2)$
Symbolic differentiation	Completely mechanistic process
Manipulates the original expression	▼ Redundant computation → can become exponentially longer to compute $f(x) = u(x)v(x)$ $f'(x) = u'(x)v(x) + u(x)v'(x)$
Automatic differentiation	Able to exploit intermediate variables → efficient
Manipulates blocks of computer program	

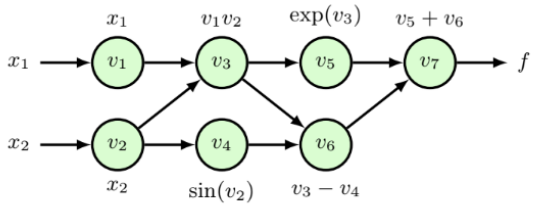
Forward Mode

In forward mode automatic differentiation, we follow these steps to get the derivative $\frac{\partial f}{\partial x_i}$:

1. Visualize the evaluation with a graph
2. Define each node as `primal` variable v_i
3. Define `tangent` variables $\dot{v}_i = \frac{\partial v_i}{\partial x_i}$

$$\dot{v}_i = \frac{\partial v_i}{\partial x_i} = \sum_{j \in parent(i)} \frac{\partial v_j}{\partial x_i} \frac{\partial v_i}{\partial v_j}$$

4. Finally compute $\frac{\partial f}{\partial x_i}$



< Evaluation Trace of x1x2 + exp(x1x2) - sin(x2) >

▼ Example of

$$f(x_1, x_2) = x_1 x_2 + e^{x_1 x_2} - sin(x_2)$$

< Primal Variables >

$$\begin{aligned}
v_1 &= x_1 \\
v_2 &= x_2 \\
v_3 &= v_1 v_2 \\
v_4 &= \sin(v_2) \\
v_5 &= \exp(v_3) \\
v_6 &= v_3 - v_4 \\
v_7 &= v_5 + v_6.
\end{aligned}$$

< Tangent Variables >

$$
\begin{aligned}
\dot{v}_1 &= 1 \\
\dot{v}_2 &= 0 \\
\dot{v}_3 &= v_1\dot{v}_2 + \dot{v}_1 v_2 \\
\dot{v}_4 &= \dot{v}_2 \cos(v_2) \\
\dot{v}_5 &= \dot{v}_3 \exp(v_3) \\
\dot{v}_6 &= \dot{v}_3 - \dot{v}_4 \\
\dot{v}_7 &= \dot{v}_5 + \dot{v}_6.
\end{aligned}
$$

The steps continue by evaluating a tuple of $(v_i, \dot{v}_i)$ until it gets to the last derivative.

Reverse Mode

In reverse mode automatic differentiation, the process is much the same as the error back-propagation applied to differentiation. In contrast to forward mode, it proceeds from parents to children so that we denote `adjoint` $\bar{v}_i = \frac{\partial f}{\partial v_i}$ instead of `tangent`.

1. Visualize the evaluation with a graph
2. Define each node as primal variable v_i
3. Define `adjoint` variables $\bar{v}_i$
4. Finally compute $\bar{x}_i = \frac{\partial f}{\partial x_i}$

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

▼ Example of

$$
f(x_1, x_2) = x_1 x_2 + e^{x_1 x_2} - sin(x_2)
$$

$$
\begin{aligned}
\overline{v}_7 &= 1 \\
\overline{v}_6 &= \overline{v}_7 \\
\overline{v}_5 &= \overline{v}_7 \\
\overline{v}_4 &= -\overline{v}_6 \\
\overline{v}_3 &= \overline{v}_5 v_5 + \overline{v}_6 \\
\overline{v}_2 &= \overline{v}_2 v_1 + \overline{v}_4 \cos(v_2) \\
\overline{v}_1 &= \overline{v}_3 v_2.
\end{aligned}
$$

Forward vs Reverse

(1) Jacobian

If we expand to a model with D inputs $\{x_1, x_2, \cdots, x_D\}$ and K outputs $\{f_1, f_2, \cdots, f_K\}$, a single parallel computation of AD produces:

- **Forward mode**: a single column of J
- **Reverse mode**: a single row of J

$$
\mathbf{J} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_D} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_K}{\partial x_1} & \cdots & \frac{\partial f_K}{\partial x_D} \end{bmatrix}.
$$

(2) Gradient of Unit

If forward/reverse mode is applied to a neural network, each unit u_i would hold the gradient of

- **Forward Mode**: Derivative of a unit with respect to input → $\frac{\partial u_i}{\partial x_i}$
- **Reverse Mode**: Derivative of an output with respect to unit → $\frac{\partial f}{\partial u_i}$