TTIC 31230, Fundamentals of Deep Learning

David McAllester, Winter 2022

Backpropagation with Arrays and Tensors

Program Values as Objects

Consider a scalar product (x, y and z are each just real numbers).

$$z = xy$$

In a framework the values of the variables x, y z are objects in the sence of object oriented programming or Python.

In computing z.value we assume that x.value and y.value are known. In the base case these are are just inputs or parameters.

Program Values as Objects

$$z = xy$$

The forward pass calls the procedure z.forward on each computed value z.

The object z holds its own inputs in its attributes.

Since z computed from a product the procedure z.forward assigns

$$z.\mathrm{value} = x.\mathrm{value} * y.\mathrm{value}$$

Backprop with Objects

$$z = xy$$

Each object x has an attribure x.grad which holds the gradient of the loss with respect to x.

We want

$$z.\mathrm{grad} = \frac{\partial \mathcal{L}}{\partial z}$$

Backpropagation calls z.backward on each computed value z in the reverse order.

For $z = xy$ we have that z.backward does

$$x.\mathrm{grad} \;\texttt{+=}\; y.\mathrm{value} * z.\mathrm{grad}$$
$$y.\mathrm{grad} \;\texttt{+=}\; x.\mathrm{value} * z.\mathrm{grad}$$

Handling Arrays

Consider an inner product between vectors

$$z = x^\top y$$

In this case case z.forward does

$$z.\mathrm{value} = 0$$

$$\text{for } i \quad z.\mathrm{value} \mathrel{\texttt{+=}} x.\mathrm{value}[i] * y.\mathrm{value}[i]$$

The backward procedure z.backward treats each += instruction seperately and does.

$$\text{for } i \quad x.\mathrm{grad}[i] \mathrel{\texttt{+=}} y.\mathrm{value}[i] * z.\mathrm{grad}$$

$$\text{for } i \quad y.\mathrm{grad}[i] \mathrel{\texttt{+=}} x.\mathrm{value}[i] * z.\mathrm{grad}$$

Handling Arrays

Now consider multiplying a vector x by a matrix W.

$$y = Wx$$

In this case case y.forward does

$$\text{for } j \;\; y.\text{value}[j] = 0$$

$$\text{for } i, j \;\; y.\text{value}[j] \mathrel{\texttt{+=}} W.\text{value}[j, i] * x.\text{value}[i]$$

The backward procedure y.backward treats each individual += as a scalar product and does

$$\text{for } i, j \;\; x.\text{grad}[i] \mathrel{\texttt{+=}} W.\text{value}[j, i] * y.\text{grad}[j]$$

$$\text{for } i \;\; W.\text{grad}[j, i] \mathrel{\texttt{+=}} x.\text{value}[i] * z.\text{grad}[j]$$

A Linear Threshold Layer

$$s = \sigma\left(W^1 h - B^1\right)$$

$$
\begin{array}{lrcl}
\text{for } j & \tilde{s}[j] & = & 0 \\
\text{for } j, i & \tilde{s}[j] & \mathtt{+=} & W^1[j,i]h[i] \\
\text{for } j & s[j] & = & \sigma(\tilde{s}[j] - B^1[j])
\end{array}
$$

backpropagation is also done with loops treating each individual assigningments and **+=** instructions.

General Tensor Operations

In practice all deep learning source code can be written unsing scalar assignments and loops over scalar assignments. For example:

$$\begin{aligned} \text{for } h, i, j, k \;\; \tilde{Y}[h, i, j] &\;\; \texttt{+=} \;\; A[h, i, k]\; B[h, j, k] \\ \text{for } h, i, j \;\; Y[h, i, j] &\;\; = \;\; \sigma(\tilde{Y}[h, i, j]) \end{aligned}$$

has backpropagation loops

$$\begin{aligned} \text{for } h, i, j \;\; \tilde{Y}.\text{grad}[h, i, j] &\;\; \texttt{+=} \;\; Y.\text{grad}[h, i, j]\;\; \sigma'(\tilde{Y}.\text{grad}[h, i, j]) \\ \text{for } h, i, j, k \;\; A.\text{grad}[h, i, k] &\;\; \texttt{+=} \;\; \tilde{Y}.\text{grad}[h, i, j]\; B[h, j, k] \\ \text{for } h, i, j, k \;\; B.\text{grad}[h, j, k] &\;\; \texttt{+=} \;\; \tilde{Y}.\text{grad}[h, i, j]\; A[h, i, k] \end{aligned}$$

END