TTIC 31230, Fundamentals of Deep Learning

David McAllester, Autumn 2024

Backpropagation with Arrays and Tensors

Program Values as Objects

In a framework the program (or deep model) variables are objects in the sense of object oriented programming or Python.

Each object x stores its input objects in its instance variables and has an instance variable x.value storing its value.

The instance variable x.value is filled by sending x a forward message after its inputs have computed their values.

Each object x has an instance variable x.grad storing $\partial\mathcal{L}/\partial x$.

x.grad is filled by the backward methods of objects y that use x as an input. The backward method for y is called after y.grad has been filled and adds into x.grad for each input x.

Scalar Products

Consider a scalar product $z = xy$.

The forward method for z computes.

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

The backward method for z computes

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

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

Handling Arrays

Consider an inner product between vectors

$$z = x^\top y$$

In this case z.forward does

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

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

The backward method for z treats each += instruction separately and does.

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

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

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

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

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

A Linear Threshold Layer

$$s = \sigma\left(Wh - B\right)$$

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

Backpropagation is also done with loops treating each individual assignment and **+=** instruction.

General Tensor Operations

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

$$\text{for } h, i, j, k \;\; Y[h, i, j] \;\; \texttt{+=} \;\; A[h, i, k]\, B[h, j, k]$$

has backpropagation loops

$$\text{for } h, i, j, k \;\; A.\text{grad}[h, i, k] \;\texttt{+=}\; Y.\text{grad}[h, i, j]\; B.\text{value}[h, j, k]$$
$$\text{for } h, i, j, k \;\; B.\text{grad}[h, j, k] \;\texttt{+=}\; A.\text{value}[h, i, k]\; Y.\text{grad}[h, i, j]$$

I call this the "swap rule". To get the gradient of A we swap A and Y. To get the gradient of B we swap B and Y.

END