Auto-differentiation

peter.vincent.14

September 2020

1 Autograd

Autodifferentiation is a technique used to calculate gradients in deep neural networks. Symbolically evaluating these gradients can become extremely slow, while numerical differentiation can be quite inaccurate (particularly when hardware is of a given precision).

Autodif takes advantage of the fact that all computer programs are composed of a small set of fundamental operations (+ , − , x) for which the derivatives are well known (for the sake of mathematical functions, we can expand this elementary list to expressions like $\exp[x]$, $\log(x)$, x^{-1}). We can therefore use the chain rule to break down any arbitrary function down to a series of smaller expressions. This is the called an evaluation graph.

1.0.1 Chain rule

$$y = f(g(h(x)))$$
$$\frac{dy}{dx} = \frac{dy(x)}{df(x)} \cdot \frac{df(x)}{dg(x)} \cdot \frac{dg(x)}{dh(x)} \cdot \frac{dh(x)}{dx}$$

1.1 Forward and backward accumulation

1.1.1 Forward accumulation

$$\frac{dw_i}{dx} = \frac{dw_i}{dw_{i-1}} \frac{dw_{i-1}}{dx}$$

with $w_{\max(i)} = y$

1.1.2 Backward accumulation

$$\frac{y}{dw_i} = \frac{dy}{dw_{i+1}} \frac{dw_{i+1}}{dw_i}$$

with $w_{\min(i)} = x$

Backwards propagation is an example of Backward accumulation

1.2 BackProp on a linear layer

Lets say we have a network composed of 2 linear layers, and each layer has 2 neurons. At each pass for give 2-dimensional stimuli, and then calculate the gradients (note this is very unrealistic, but works for demo purposes)

$$X = \begin{pmatrix} x_{1_1} & x_{1,2} \\ x_{2,1} & x_{2,2} \end{pmatrix} \ , \ W = \begin{pmatrix} w_{1,1} & w_{1,2} \\ w_{2,1} & w_{2,2} \end{pmatrix} \ , \ \mathbf{Y} = XW$$

Following a forward pass, calculated above, we find some **loss** $L = f(Y)$, from which we can calculate $\frac{\partial L}{\partial Y}$. We want to know $\frac{\partial L}{\partial W}$. This can be calculated by $\frac{\partial L}{\partial W} = \frac{\partial L}{\partial Y} \frac{\partial Y}{\partial W}$. Unfortunately, computing and storing $\frac{\partial Y}{\partial W}$ is extremely expensive for large networks, since it will have $N \times M \times N \times D$ elements. Storing these at 32 bit precision can easily take up hundreds of gigabytes.

1.2.1 Efficient calculation of gradients

Fortunately, we can do the sums analytically on small networks and use this to derive general forms. We will do this below for linear layers in the network we described up above.

$$\frac{\partial L}{\partial W} = \sum_i \sum_j \frac{\partial L}{\partial y_{i,j}} \cdot \frac{\partial y_{i,j}}{\partial w_{i,j}}$$

$$\frac{\partial L}{\partial Y} = \begin{pmatrix} \frac{\partial L}{\partial y_{1,1}} & \frac{\partial L}{\partial y_{1,2}} \\ \frac{\partial L}{\partial y_{2,1}} & \frac{\partial L}{\partial y_{2,2}} \end{pmatrix}$$

$$Y = \begin{pmatrix} x_{1,1}w_{1,1} + x_{1,2}w_{2,1} & x_{1,1}w_{1,2} + x_{1,2}w_{2,2} \\ x_{2,1}w_{1,1} + x_{2,2}w_{2,1} & x_{2,1}w_{1,2} + x_{2,2}w_{2,2} \end{pmatrix}$$

So, if we want $\frac{\partial L}{\partial w_{1,1}}$

$$\frac{\partial Y}{\partial w_{1,1}} = \begin{pmatrix} x_{1,1} & 0 \\ x_{2,1} & 0 \end{pmatrix}$$

and then

$$\frac{\partial L}{\partial w_{1,1}} = \begin{pmatrix} \frac{\partial L}{\partial y_{1,1}} & \frac{\partial L}{\partial y_{1,2}} \\ \frac{\partial L}{\partial y_{2,1}} & \frac{\partial L}{\partial y_{2,2}} \end{pmatrix} \cdot \begin{pmatrix} x_{1,1} & 0 \\ x_{2,1} & 0 \end{pmatrix} = \frac{\partial L}{\partial y_{1,1}} x_{1,1} + \frac{\partial L}{\partial y_{2,1}} x_{2,1}$$

Similarly

$$\frac{\partial L}{\partial w_{2,2}} = \begin{pmatrix} \frac{\partial L}{\partial y_{1,1}} & \frac{\partial L}{\partial y_{1,2}} \\ \frac{\partial L}{\partial y_{2,1}} & \frac{\partial L}{\partial y_{2,2}} \end{pmatrix} \cdot \begin{pmatrix} 0 & x_{1,2} \\ 0 & x_{2,2} \end{pmatrix} = \frac{\partial L}{\partial y_{1,2}} x_{1,2} + \frac{\partial L}{\partial y_{2,2}} x_{2,2}$$

And so on....

Eventually we can see the general form will be

$$\frac{\partial L}{\partial W} = X^T \frac{\partial L}{\partial Y}$$

This same procedure can be followed if we add a bias term $Y = XW + B$, eventually yielding

$$\frac{\partial L}{\partial B} = \mathbf{I} \frac{\partial L}{\partial Y}$$