Getting the Jacobian of myfnn()

We can think of myfnn() as a succession of simple functions applied to the input, and apply the chain rule to each one. As a reminder, the Chain Rule in general states that, for a function

$$\boldsymbol{h}(\boldsymbol{x}) = \boldsymbol{g}\big(\boldsymbol{f}(\boldsymbol{x})\big)$$

the Jacobian of $\boldsymbol{h}$ with respect to $\boldsymbol{x}$ is given by (in Einstein tensor notation)

$$\left.\frac{\partial h_i}{\partial x_j}\right|_{\boldsymbol{x}} = \left.\frac{\partial g_i}{\partial f_k}\right|_{\boldsymbol{f}(\boldsymbol{x})} \left.\frac{\partial f_k}{\partial x_j}\right|_{\boldsymbol{x}}$$

, where I use the notation $\boldsymbol{s}|_{\boldsymbol{u}}$ to mean $\boldsymbol{s}$ evaluated at $\boldsymbol{u}$.

We can apply this inductively to each layer in myfnn(). As per the notation in our Neural Networks paper

$$y_j^{(k)} = f_{act}\Big(b_j^{(k)} + a_{jl}^{(k)} y_l^{(k-1)}\Big)$$

, where $y_j^{(k)}$ are the values of the k −th layer, $y_l^{(k-1)}$ are the values of the $k-1$ layer, $a_{jl}^{(k)}$ and $b_j^{(k)}$ are respectively the weights and biases of the k −th layer. I'm putting the superscripts in brackets as a standard way to indicate that (k) and $(k-1)$ should not be treated as tensor indices, unlike j and l

We can think of the 0 −th layer, $\boldsymbol{y}^{(0)}$ as the input, and the last layer, $\boldsymbol{y}^{(6)}$ as the output (where f_{act} is the trivial identity function for the last layer, and mish for all others).

So we want to calculate $\left.\frac{\partial y_i^{(6)}}{\partial y_j^{(0)}}\right|_{\boldsymbol{y}^{(0)}}$. We can do this inductively: we start with $\left.\frac{\partial y_i^{(6)}}{\partial y_j^{(0)}}\right|_{\boldsymbol{y}^{(0)}} = \delta_{ij}$.

If we have $\left.\frac{\partial y_i^{(k-1)}}{\partial y_j^{(0)}}\right|_{\boldsymbol{y}^{(0)}}$ from the Chain Rule applied first to the affine linear transformation $b_j^{(k)} + a_{jl}^{(k)} y_l^{(k-1)}$ and then to the activation function, we get that

$$\left.\frac{\partial y_i^{(k)}}{\partial y_j^{(0)}}\right|_{\boldsymbol{y}^{(0)}} = f'_{act}\Big(b_i^{(k)} + a_{im}^{(k)} y_m^{(k-1)}\Big)\, a_{im}^{(k)} \left.\frac{\partial y_m^{(k-1)}}{\partial y_j^{(0)}}\right|_{\boldsymbol{y}^{(0)}}$$

, where f'_{act} is the 1D derivative of the activation function relative to its argument.

You can write a new function similar to myfnn() which does this, layer by layer. The sensitivity is a matrix, you start with the identity and for each layer you multiply it on the left by $a_{im}^{(k)}$, calculate the value of the new layer, $b_i^{(k)} + a_{im}^{(k)} y_m^{(k-1)}$, and then multiply the $i-th$ row of the matrix by the activation derivative $f'_{act}\left(b_i^{(k)} + a_{im}^{(k)} y_m^{(k-1)}\right)$

Better yet, since this procedure requires the evaluation of all the neural net layers anyway, you can extend myfnn() so that it does the additional sensitivity calculations and outputs a sensitivity in addition to the neural network function. This way we don't do redundant NN calculations for need[1] = 1.