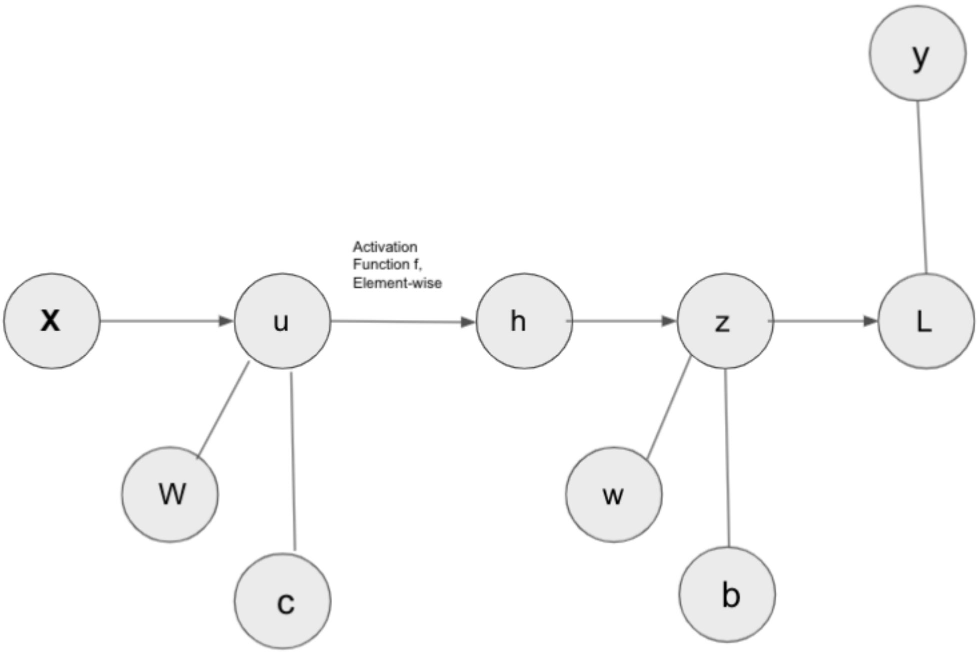
Forward Pass: Multi-layer Perceptron



Hidden Layer

$$\mathbf{u} = \mathbf{x}W + c$$

$$\mathbf{h} = \max(\mathbf{0}, \mathbf{u})$$

Output Layer

$$z = \mathbf{h}w + b$$

Softmax Cross-Entropy Loss Layer

$$L_i = -\log\left(\frac{e^{z_{y_i}}}{\sum_j e^{z_j}}\right)$$

Note that y_i is the true class label

$$L = \underbrace{\frac{1}{N}\sum_i L_i}_{\text{data loss}} + \underbrace{\frac{1}{2}\lambda\sum_k\sum_l W_{k,l}^2}_{\text{regularization loss}}$$

Backpropogation: Multi-layer Perceptron

Loss Layer

Denote the softmax probability for element z_k as p_k.

$$p_k = \frac{e^{z_k}}{\sum_j e^{z_j}}$$

Recall the Softmax Cross-Entropy Loss function.

$$CE_{loss} = -\sum_j^C y_j \log{(p_j)}$$

Simplifying, we get:

$$L_i = -\log{(p_{y_i})}$$

and

$$\frac{\partial L_i}{\partial z_k} = p_k - \not\Vdash(y_i = k)$$

where $\not\Vdash$ is the Indicator function

Output Layer

First, note that:

$$\frac{\partial z}{\partial h} = w^T$$

Thus,

$$\frac{\partial L}{\partial h} = \frac{\partial L}{\partial z} w^T$$

Hence, we can perform gradient descent using the following gradients: (Note: The λw is obtained by taking the gradient of the regularization term within our Loss function, $\frac{1}{2}\lambda w^2$)

$$\frac{\partial L}{\partial w} = h^T\frac{\partial L}{\partial z} + \lambda w$$

$$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial z}$$

Hidden Layer

Weight updates:

$$\frac{\partial L}{\partial W} = \mathbf{X}^T \frac{\partial L}{\partial u} + \lambda W$$
$$\frac{\partial L}{\partial c} = \frac{\partial L}{\partial u}$$

But how do we obtain $\frac{\partial \mathbf{h}}{\partial \mathbf{u}}$ and by extension $\frac{\partial \mathbf{L}}{\partial \mathbf{u}}$???

Derivative of a vector with respect to another vector: Using the Jacobian Matrix to compute $\frac{\partial \mathbf{h}}{\partial \mathbf{u}}$

But, what is a Jacobian?

Let $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^m$ be a function that takes $\mathbf{x} \in \mathbb{R}^n$ as input and produces the vector $\mathbf{f}(\mathbf{x}) \in \mathbb{R}^m$ as output. The Jacobian matrix of $\mathbf{f}$ is then defined to be an $m \times n$ matrix, denoted by $\mathbf{J}$, whose (i, j) the entry is $\mathbf{J}_{ij} = \frac{\partial f_i}{\partial x_j}$.

$$\mathbf{J} = \begin{bmatrix} \frac{\partial \mathbf{f}}{\partial x_1} & \cdots & \frac{\partial \mathbf{f}}{\partial x_n} \end{bmatrix} = \begin{bmatrix} \nabla^{\mathrm{T}} f_1 \\ \vdots \\ \nabla^{\mathrm{T}} f_m \end{bmatrix} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}$$

where $\nabla^{\mathrm{T}} f_i$ (now a row vector) is the transpose of the gradient of the i component .

In our case, we have the RELU activation function that serves as function f.

$$\begin{array}{l} \mathbb{R}^2 \leftarrow \mathbb{R}^2 \\ \mathbf{h} = \max(\mathbf{0}, \mathbf{u}) \end{array} \qquad \frac{\partial \mathbf{h}}{\partial \mathbf{u}} = \begin{bmatrix} \frac{\partial h_1}{\partial u_1} & \frac{\partial h_1}{\partial u_2} \\ \frac{\partial h_2}{\partial u_1} & \frac{\partial h_2}{\partial u_2} \end{bmatrix} = \begin{bmatrix} \frac{\partial h_1}{\partial u_1} & 0 \\ 0 & \frac{\partial h_2}{\partial u_2} \end{bmatrix}$$

We can write,

$$\frac{\partial L}{\partial \mathbf{u}} = \left(\frac{\partial \mathbf{h}}{\partial \mathbf{u}}\right)^T \frac{\partial L}{\partial h}$$

However, since our activation function is only a function of each individual element, the partials with respect to the other dimensions is 0. Thus, the Jacobian is a diagonal matrix and hence, we can simplify this expression (making it easier to implement in our code) into an element-wise product as follows:

Let $g(z) = \max(0, z)$
Let $f =$ " $\frac{dg}{dz}$ ". $f =$

$$\begin{cases} 0 & x \leq 0 \\ 1 & x > 0 \end{cases}$$

We can now write $\frac{\partial L}{\partial \mathbf{u}}$ as:

$$\frac{\partial L}{\partial \mathbf{u}} = \mathbf{f}\left(\mathbf{u}\right) \odot \frac{\partial L}{\partial h}$$

where $\odot$ is the element-wise multiplication operator, also known as the Hadamard operator.

Note: We derive the Bias gradient also leveraging the Jacobian

Similar to above, for bias vector c, we need the Jacobian matrix $\nabla_{\mathbf{c}} L$. However, this is also a diagonal matrix and we can use the same 'rewriting as element-wise product' trick.

$$\frac{\partial L}{\partial c} = \left(\frac{\partial \mathbf{u}}{\partial \mathbf{c}}\right)^T \frac{\partial L}{\partial \mathbf{u}}$$

$$\frac{\partial L}{\partial c} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}^T \frac{\partial L}{\partial \mathbf{u}} = \frac{\partial L}{\partial \mathbf{u}}$$

References

[1] The Matrix Calculus You Need For Deep Learning, Terence Parr and Jeremy Howard

[2] Stanford CS231n Course Notes

[3] Deep Learning for Computer Vision Slides, Prof. Belhumeur (Columbia University)