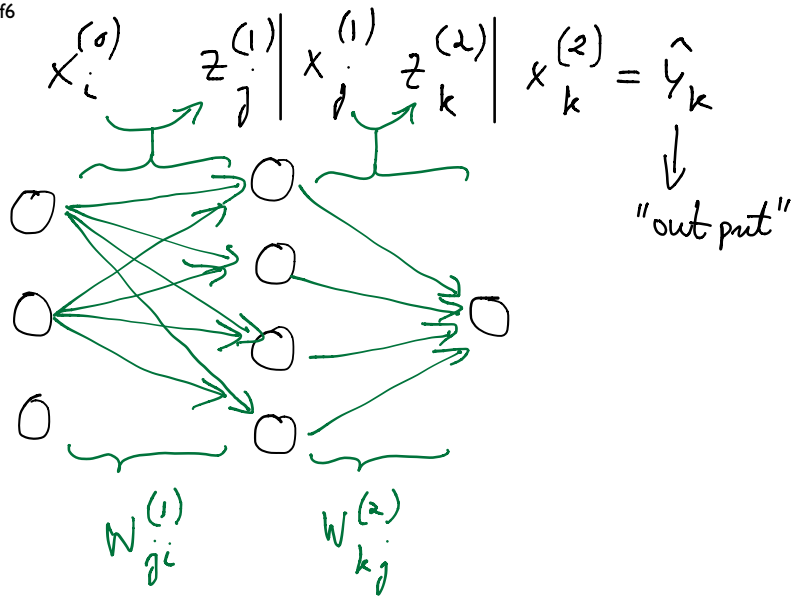
2-layer neural net with backpropagation

https://towardsdatascience.com/how-to-build-your-own-neural-network-from-scratch-in-python-68998a08e4f6



X1	X2	X3	Y
0	0	1	0
0	1	1	1
1	0	1	1
1	1	1	0

m ↓

$(x_{m,i} \;,\; y_{m,k})$

for each example m:
$x_i^{(0)} = x_{m,i}$ "input"

goal: $\hat{y}_{m,k} = y_{m,k}$ for all k and all examples m

$$x_i^{(0)} \rightarrow z_j^{(1)} = W_{ji}^{(1)} x_i^{(0)} + b_j^{(1)}$$

$$\rightarrow x_j^{(1)} = \sigma\left(z_j^{(1)}\right)$$

$$\rightarrow z_k^{(2)} = W_{kj}^{(2)} x_j^{(1)} + b_k^{(2)}$$

$$\rightarrow \hat{y}_k = x_k^{(2)} = \sigma\left(z_k^{(2)}\right)$$

define loss function $\Delta\mathcal{L}$:
"mean squared error"

$$\Delta\mathcal{L} = \sum_i \left(y_i - \hat{y}_i\right)^2$$

$$\hat{y} = x^{(2)} = \sigma\left(W^{(2)} \cdot \sigma\left(W^{(1)} \cdot x^{(0)} + b^{(1)}\right) + b^{(2)}\right)$$

back propagate through last layer
– compute gradient of $\Delta\mathcal{L}$ w.r.t. $W^{(2)}$:

$$\Rightarrow \frac{\partial \Delta\mathcal{L}}{\partial W^{(2)}_{mn}} = \frac{\partial \Delta\mathcal{L}}{\partial \hat{y}_k} \cdot \frac{\partial \hat{y}_k}{\partial z^{(2)}_k} \cdot \frac{\partial z^{(2)}_k}{\partial W^{(2)}_{mn}}$$

$$= 2\left(\hat{y}_k - y_k\right) \cdot \sigma'\big|_{z^{(2)}_k} \cdot \delta_{mk}\delta_{nj} \cdot x^{(1)}_j$$

$$= 2\left(\hat{y}_m - y_m\right) \cdot \sigma'\big|_{z^{(2)}_m} \cdot x^{(1)}_n$$

$$= 2\left(\hat{y}_m - y_m\right) \cdot \sigma\left(z^{(2)}_m\right)\cdot\left(1-\sigma\left(z^{(2)}_m\right)\right) \cdot x^{(1)}_n$$

now propagate further back one layer:

$$\frac{\partial \Delta\mathcal{L}}{\partial W^{(1)}_{mn}} = \frac{\partial \Delta\mathcal{L}}{\partial \hat{y}_k} \cdot \frac{\partial \hat{y}_k}{\partial z^{(2)}_k} \cdot \frac{\partial z^{(2)}_k}{\partial x^{(1)}_j} \cdot \frac{\partial x^{(1)}_j}{\partial z^{(1)}_j} \cdot \frac{\partial z^{(1)}_j}{\partial W^{(1)}_{mn}}$$

$$= 2\left(\hat{y}_k - y_k\right) \cdot \sigma'\big|_{z^{(2)}_k} \cdot W^{(2)}_{kj} \cdot \sigma'\big|_{z^{(1)}_j} \cdot \delta_{mj}\delta_{ni} \cdot x^{(0)}_i$$

$$= 2 \cdot W^{(2)}_{km} \cdot \sigma'\big|_{z^{(2)}_k} \cdot \left(\hat{y}_k - y_k\right) \cdot \sigma'\big|_{z^{(1)}_m} \cdot x^{(0)}_n$$

finally, for the biases we get:

$$\frac{\partial \mathcal{L}}{\partial b_m^{(2)}} = 2 \cdot (\hat{y}_m - y_m) \cdot \sigma'\big|_{z_m^{(2)}}$$

$$\frac{\partial \mathcal{L}}{\partial b_n^{(1)}} = 2 W_{km}^{(2)} \cdot \sigma'\big|_{z_k^{(2)}} \cdot (\hat{y}_k - y_k) \cdot \sigma'\big|_{z_m^{(1)}}$$

gradient descent update rule:

$$W_{mn}^{(i)} \to W_{mn}^{(i)} - \eta \cdot \frac{\partial \mathcal{L}}{\partial W_{mn}^{(i)}}$$

$$b_m^{(i)} \to b_m^{(i)} - \eta \cdot \frac{\partial \mathcal{L}}{\partial b_m^{(i)}}$$

$\eta > 0$: learning rate
(a hyperparameter of the network)

We need to choose an activation function σ. Clever choice allows to compute derivative σ' exactly solely in terms of already computed function values σ. Here, we choose as an example the sigmoid:

$$\begin{cases} \sigma'\big|_{z_k^{(2)}} = \sigma\left(z_k^{(2)}\right) \cdot \left(1 - \sigma\left(z_k^{(2)}\right)\right) \\ \qquad\qquad = \hat{y}_k \cdot \left(1 - \hat{y}_k\right) \\ \sigma'\big|_{z_m^{(1)}} = x_m^{(1)} \cdot \left(1 - x_m^{(1)}\right) \end{cases}$$

---

sigmoid function $\sigma(x)$ as activation function & its derivative:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

$$\Rightarrow \sigma'(x) = -\frac{1}{(1+e^{-x})^2} \cdot (-e^{-x})$$

$$= \frac{e^{-x}}{(1+e^{-x})^2} = \frac{1+e^{-x}-1}{(1+e^{-x})^2}$$

$$= \frac{1}{1+e^{-x}} - \frac{1}{(1+e^{-x})^2}$$

$$= \sigma(x) - \sigma^2(x)$$

$$= \sigma(x)\left(1-\sigma(x)\right)$$

$$\Rightarrow \left.\frac{d\sigma}{dx}\right|_x = \sigma|_x \cdot \left(1-\sigma|_x\right)$$

Final comment:

By including biases into the simple Boolean-function learning NN and generalizing backpropagation-based gradient descent to stochastic gradient descent using randomized mini batching, this NN can learn handwritten digits 0-9 from the MNIST image dataset at $\simeq 95\%$ recognition accuracy, if the NN has 1 hidden layer with 30 neurons densely connected.