# Neural networks

@rjmbcn14

July 26, 2021

**Neural networks, setup, architecture definitions.** Neural networks consists of $L$ layers labeled $l = 1, 2...L$. In a given $l$ layer there are $n_l$ cells, whose activation outputs are denoted $a_j^l$ and given by

$$a_j^l = f(z_j^l), \tag{1}$$

where $f(z_j^l)$ is known as the activation function whose argument is given by

$$z_j^l = \sum_{k=1}^{n_{l-1}} w_{jk} a_k^{l-1}, \tag{2}$$

here the weight $w_{jk}^l$ connects the output $a_k^{l-1}$ of the $k$-th cell in the previous layer to the $j$ cell in the $l$-th layer.

Matrix and vector notation is used to efficiently represent a neural network. Considering layer $l$ its activation outputs are contained in a column vector $\mathbf{a}^l = [a_1^l \; a_2^l \; ... \; a_{n_l}^l]^T$, given by

$$\mathbf{a}^l = f\left(\mathbf{z}^l\right) \tag{3}$$

where the input is

$$\mathbf{z}^l = \mathbf{W}^l \mathbf{a}^{l-1}, \tag{4}$$

with the weight matrix

$$\mathbf{W}^l = \begin{pmatrix} w_{11}^l & w_{12}^l & \dots & w_{1n_{l-1}}^l \\ w_{21}^l & w_{22}^l & \dots & w_{2n_{l-1}}^l \\ \vdots & \vdots & & \vdots \\ w_{n_l 1}^l & w_{n_l 2}^l & \dots & w_{n_l n_{l-1}}^l \end{pmatrix}. \tag{5}$$

*Input and output layers.* The activation vector of the input layer consists of the input vector $\mathbf{x}$. For each input there is an observation vector $\mathbf{y}$. The activation vector of the output layer is denoted by $\hat{\mathbf{y}}$

*Hidden layers.* Hidden layers are those layers connecting the input layer to the output layer.

*Training a neural network.* A network is trained by adjusting the elements in its weight matrices $\mathbf{W}^l$ to yield a minimum value in its cost function C. Given a set of inputs and observations $\{\mathbf{x}, \mathbf{y}\}$ the cost function can defined as

$$\mathrm{C} = \frac{1}{2} \sum_j (y_j - \hat{y}_j)^2 \tag{6}$$

other cost functions could be used. The typical training/optimization technique is gradient descent.

**Network optmization.** Given a set of known inputs and known observations $\{\mathbf{x}, \mathbf{y}\}$ a network is optimized by minimizing its cost function C. This is typically achieved by first initializing the weights $\mathbf{W}^l$. Then given a subset (also known as a *batch*) of the inputs $\mathbf{x}$ is used to compute the network outputs $\hat{\mathbf{y}}$. This step is known as forward propagation. From the computed network outputs and the known outputs the cost function is computed, for instance using the squared error

$$\mathrm{C} = \frac{1}{2}\sum_j (y_j - \hat{y}_j)^2 \tag{7}$$

The typical optimization method, i.e. finding minima in the Cost function, is gradient descent.

**Gradient descent.** (From wikipedia) Gradient descent is a first-order iterative optimization algorithm. To find a local minimum of the cost function using gradient descent, one takes steps proportional to the negative of the gradient of the function with respect weights at the current point:

$$\Delta\mathbf{W}^l = -\gamma \nabla_{\mathbf{W}^l} \mathrm{C}, \tag{8}$$

where proportionality factor $\gamma$ is a positive constant known as the learning rate of the network.

**Backpropagation.** Backpropagation is the name that gradient descent receives when used to train a neural network. It consists in computing the error, for a given set of $\{\mathbf{x}, \mathbf{y}\}$, and propagating it through the different hidden layers in the network so that $\Delta\mathbf{W}^l$ can be computed.

The following equations describe a summary of backpropagation, including matrix notation where $\odot$ and $\otimes$ denote element-wise matrix multiplication and outer-product respectively.

- Error in output layer:

$$\delta_j^L = \frac{\partial C}{\partial a_j^L} f'\left(z_j^L\right) \rightarrow \text{ matrix form :} \tag{9}$$

$$\delta^L = \nabla_{\mathbf{a}} C \odot g'\left(\mathbf{h}^L\right) \tag{10}$$

- Error in hidden layer:

$$\delta_k^{l-1} = \sum_j w_{jk}^l \delta_j^l f'\left(h_k^{l-1}\right) \rightarrow: \tag{11}$$

$$\delta^{l-1} = \left(\left(\mathbf{W}^l\right)^T \delta^l\right) \odot f'\left(\mathbf{h}^{l-1}\right) \tag{12}$$

- Rate of change of cost function with respect bias in the network:

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l \rightarrow: \tag{13}$$

$$\frac{\partial C}{\partial \mathbf{b}^l} = \delta^l \tag{14}$$

- Rate of change of cost function with respect weights in the network:

$$\frac{\partial C}{\partial w_{jk}^l} = \delta_j^l a_k^{l-1} \rightarrow: \tag{15}$$

$$\frac{\partial C}{\partial \mathbf{W}} = \delta^l \otimes \mathbf{a}^{l-1} \tag{16}$$

**Neural network performance.** A neural network is trained on a set of data $\{\mathbf{x}, \mathbf{y}\}$ with the purpose to make predictions on the outcome of a process given a new set of input values. Two common pitfalls

can limit the quality of predictions: oversimplication of network model and complexity of the model. Oversimplification of the model can produce *underfitting* of the training data, that is a model that misses important aspects of the process in their predictions. On the other had a high level of complexity in the model can produce *overfitting*, that is a model that is too specific for the given training data set. A healthy balance is achieved in the middle, that is a network model that takes into account the most relevant aspects of the process but that is general enough to make accurate predictions based on a new set of inputs.

To achieve such a model the data available to train a network is divided into three mutually exclusive sets: *training data*, *validation data*, and *testing data*. The approach consists on tuning the weights in the network using the *training data*, then making predictions using the *validation data*. *Underfitting* of the model is shown when the cost function, normalized to a relevant scale, evaluated on the validation set is too high. *Overfitting* is present when the cost function for the training data is low but the cost computed based on the validation data differs significantly from the training data cost. When both costs agree with each other the network is said to be well trained, its perfromance can then be estimated by calculating the cost of the nework using the testing data set.

**Advanced neural networks** Some problems are better tackled using specialized versions of neural networks, such as in convolutional neural networks (CNN) and recurrent-neural networks (RNN). These two types of neural networks are implemented taking advantage of known features in the input data such as in images or in time series. The idea behind is simple: exploit known structures in the input data to make the learning and inference process of the network more efficient.

**Derivation of backpropagation equations** It is all about the chain rule. The goal is to find the weights that minimize the cost function of the model, which can be implemented numerically using gradient descent

$$\mathbf{W}^l = \mathbf{W}^l - \gamma \nabla_{\mathbf{W}^l} \mathrm{C}, \tag{17}$$

For instance to get the rate of change of the cost function with respect coefficients in the second to last layer

$$\frac{\partial C}{\partial w_{jk}^{L-1}} = \frac{\partial C}{\partial a_j^L} \frac{\partial a_j^L}{\partial z_j^L} \frac{\partial z_j^L}{\partial w_{jk}} \tag{18}$$

To get the rate of change of teh cost function with respect to coefficients in inner layers consider

$$\frac{\partial C}{\partial w_{kl}^{L-2}} = \sum_j \frac{\partial C}{\partial a_j^L} \frac{\partial a_j^L}{\partial z_j^L} \frac{\partial z_j^L}{\partial w_{kl}} \tag{19}$$

where the sum running over $j$ is due to the fact that the $w_{kl}^{L-2}$ weight in the $L-2$ layer couples to all cells in the $L$ layer by the output $a_k^{L-1}$ of the $k$ cell in the $L-1$ layer. That is

$$z_j^L = \sum_k w_{jk} a_k^{L-1} \tag{20}$$

$$z_j^L = \sum_k w_{jk} f(z_k^{L-1}) \tag{21}$$

where $z_k^{L-1} = \sum_l w_{kl} a_l^{L-2}$