# Neural networks

@rjmbcn14

July 25, 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

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

where the $f(h_j^l)$ is known as the activation function whose argument is given by the weighted input to the cell

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

here the weight $w_{jk}^l$ connects the activation $a_k^{l-1}$ 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

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

where the weighted input is

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

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 of the network is equal to 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 the cost function given a set of known inputs and known observations $\{\mathbf{x}, \mathbf{y}\}$. The typical optimization method is gradient descent.

**Network performance.** The performance of a neural network can be described by its cost function C, which quantifies the difference $\mathbf{y} - \hat{\mathbf{y}}$ between the neural net output $\hat{\mathbf{y}}$, given the fetures vector $\mathbf{x}$ and weight matrices $\mathbf{W}^l$, and the correspoding observation $\mathbf{y}$. In classification tasks the performance of the network is also described by its accuracy.

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

and computing the errror

iteritavely changing the weight values s

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:

$$\Delta \mathbf{W}^l = -\gamma \nabla_{\mathbf{W}^l} \mathrm{C}, \tag{6}$$

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 propagating the error of the cost function, for a given set of $\{\mathbf{x}, \mathbf{y}\}$, through the different hidden layers in the network so that $\Delta \mathbf{W}^l$ can be computed.

- Error in output layer:

$$\delta_j^L = \frac{\partial C}{\partial a_j^L} g^{'}\left(h_j^L\right) \rightarrow \text{ matrix form :} \tag{7}$$

$$\delta^L = \nabla_{\mathbf{a}} C \bigodot g^{'}\left(\mathbf{h}^L\right) \rightarrow\rightarrow \text{ Numpy implementation :} \tag{8}$$

$$\text{error\_output} = \text{partial\_cost\_partial\_activation} * \text{g\_prime} \tag{9}$$

- Error in hidden layer:

$$\delta_k^{l-1} = \sum_j w_{jk}^l \delta_j^l f^{'}\left(h_k^{l-1}\right) \rightarrow: \tag{10}$$

$$\delta^{l-1} = \left(\left(\mathbf{W}^l\right)^T \delta^l\right) \bigodot f^{'}\left(\mathbf{h}^{l-1}\right) \rightarrow\rightarrow: \tag{11}$$

$$\text{error\_hidden} = \left(\text{np.dot(weights\_hidden.}^{\mathrm{T}}\text{,}\delta^{\mathrm{l}}\text{)}\right) * \text{f\_prime} \tag{12}$$

- 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:

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

$$\frac{\partial C}{\partial \mathbf{W}} = \mathbf{a}^{l-1}\left(\delta^l\right)^T \rightarrow\rightarrow: \tag{16}$$

$$\text{partial\_cost\_partial\_weight} = \text{np.dot}(\mathbf{a}^{\mathrm{l}-1}, \delta^{\mathrm{l}}\text{.}^{\mathrm{T}}) \tag{17}$$

**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. Simplification of the network 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 is shown when the cost function, normalized to a relevant scale, is 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 very simple: exploit known structures in the input data to make the learning and inference process of the network more efficient. For instance consider the problem of recognizing a cat in an image where it does not matter the location of the cat within the image. If it is in the top left or the bottom right, it is still a cat to our eyes. We would like our CNNs to also possess this ability known as translation invariance.

One of them is Recurrent Neural Networks (RNN) which are appropriate to handle cases where inputs....
**Characteristics**

1. The inputs are ordered. Does the order matter?
2. One would like to predict the current state of a system given its previous states.
3. 

Basic equation for activation function in RNN:

## References

[1] Press, W. H. (2007). Numerical recipes 3rd edition: The art of scientific computing. Cambridge university press.