Physics Teams: Introduction to Neural Networks

Christian Reichelt

Universität Heidelberg

April 7, 2017

What is a Neural Network?

Initial motiviation: Model of biological intelligence

- Connected web of neurons
- Each neuron fires if a sum of "inputs" reaches a threshold

The application in a wide range of subjects (including physics):

- Remove any biological motivation
- Simply a clever non-linear transformation on a set of input variables $\mathbf{x}$ to a set of output variables $\mathbf{y}$.
- The non-linear transformation is adjustable, and tuned on training sets of data

Why Neural Networks?

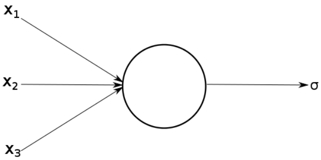
- High ability to detect and derive complicated patterns and trends in highly complex data.
- *Universal approximation theorem*: Neural networks can approximate any function (on a compact set) arbitrarily well

We will see applications later, but for now keep in mind **Multivariate Analysis** in Particle Physics:

- An event is characterised by some data $\mathbf{x}$ in a d-dimensional feature space.
- These variables can in general be correlated
- We seek a transformation $f : \mathbb{R}^d \to \mathbb{R}^N$, $N << d$, for example separating events from background

Sigmoid Neurons

A single **sigmoid neuron** takes an input $\mathbf{x}$ and gives an output $\sigma(a)$

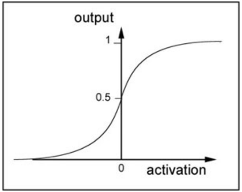


by first multiplying a weight $\mathbf{w}$ and a bias b to give the *activation*:

$$\mathbf{a} = \mathbf{w}^{\mathsf{T}}\mathbf{x} + b = \tilde{\mathbf{w}}^{\mathsf{T}}\tilde{\mathbf{x}}, \qquad \tilde{\mathbf{x}} = (1, x_1, x_2, \ldots). \tag{1}$$

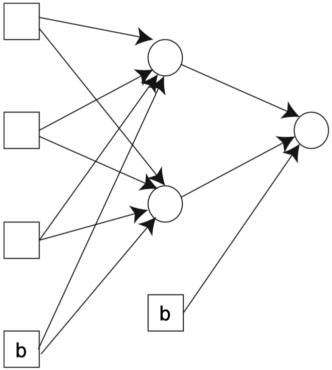
Then apply the *activation function* σ (Here a sigmoid - $\exists$ many other choices)

$$\sigma(a) = \frac{1}{1 + \exp(-a)}$$



Feed-forward Neural Network

- Simplest form of neural networks
- Several layers: **Input layer**, **output layer** and **hidden layers**
- Information travels in one direction - no loops



- Can easily be generalized to arbitrary number of layers, nodes, and different activation functions.

Feed-forward Neural Network

Use labels ijl to describe input i to node j in layer l. Then the activation in the hidden layer is

$$a_{j1} = \sum_{i=0}^{3} w_{ij1} x_i, \quad j = 1, 2 \tag{2}$$

which provides inputs z_j to the output layer, by for example an activation function of:

$$z_j = h\left(a_{j1}\right) = \tanh\left(a_{j1}\right), \quad j = 1, 2. \tag{3}$$

The activation in the output layer is then

$$a = \sum_{i=0}^{2} w_{i12} z_i \tag{4}$$

with a final output

$$\sigma\left(a\right) = \left\{1 + \exp\left[-w_{012} - \sum_{j=1}^{2} w_{j12} \tanh\left(w_{0j1} + \sum_{i=1}^{3} w_{ij1} x_i\right)\right]\right\}^{-1} \tag{5}$$

Solution of the Neural Network

- **Task:** Find the weights $\mathbf{w}$ which solve our classification problem
- **Solution:** The set of weights which minimizes a defined cost/error function
- **How:** By training on known data and adjusting the weights in the direction which minimizes the error

Gradient Descent

First we define an error function of the network, e.g. for a training set $\{(\mathbf{x}_i, \mathbf{y}_i)\}$,

$$E = \frac{1}{2n} \sum_i \|\sigma(\mathbf{x}_i) - \mathbf{y}_i\|^2 \tag{6}$$

which by our choice of activation function is a continuous and differentiable function of the weights. E is minimized through a *gradient descent* where

$$\nabla E = \left(\frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}, \ldots, \frac{\partial E}{\partial w_l} \right) \tag{7}$$

is used to update the weights in the opposite direction of the gradient:

$$\Delta \mathbf{w} = -\eta \nabla E \tag{8}$$

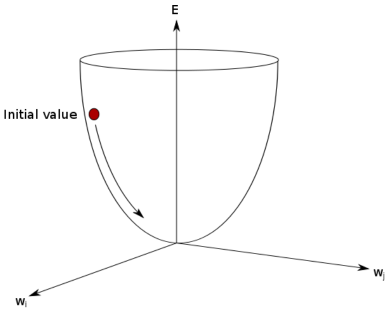
Here $\eta > 0$ is the *learning rate*.

- Small value $\eta \to$ slow learning
- Large value $\eta \to$ fast learning, but potential instability

The learning rate can be gradually decreased during the training.

Gradient Descent

Ideally, the error function would be some globally convex function in weight space:



That is of course not always the case... but global optimization is a different subject.

Task: Evaluate the gradient efficiently $\rightarrow$ **Backpropagation algorithm**

Backpropagation

The backpropagation algorithm for training and updating weights follows this pattern:

- **Forward Propagation**: Run through the network with a training sample and receive outputs
- **Output error:** Find the error of the output layer
- **Backpropagate:** Propagate the error to inner layers. The error of a layer is based on the error of neurons it provides inputs to (Basic use of the chain rule)
- **Gradient:** Finally the gradient of the cost function in weight space can be computed, and the weights are updated

Backpropagation

Consider the error contribution from a single training E_n

$$\frac{\partial E_n}{\partial w_{ijl}} = \frac{\partial E_n}{\partial a_{jl}} \frac{\partial a_{jl}}{\partial w_{ijl}} = \frac{\partial E_n}{\partial a_{jl}} z_{ijl} \equiv E_{njl} z_{ijl} \tag{9}$$

This we could easily evaluate for the output layer $l = L$. Assume only a single output node, then for the layer before $l = L - 1$

$$E_{njl} = \frac{\partial E_n}{\partial a_{1,l+1}} \frac{\partial a_{1,l+1}}{\partial a_{jl}} = E_{n1,l+1} \frac{\partial a_{1,l+1}}{\partial a_{jl}} = E_{n1,l+1} \frac{\partial a_{1,l+1}}{\partial z_{i1,l+1}} \frac{\partial z_{i1,l+1}}{\partial a_{jl}} \tag{10}$$

$$= E_{n1,l+1} \sum_i w_{i1,l+1} \frac{\partial h\left(a_{il}\right)}{\partial a_{jl}} = E_{n1,l+1} w_{j1,l+1} h'\left(a_{jl}\right) \tag{11}$$

i.e. the derivatives are computable recursively by data from later layers.

Regularization

Avoid overtraining, i.e. adjusting weights to noise/fluctuations. Signs are for example if:

- The accuracy stops increasing, even though the error function decreases
- Accuracy in training reaches 100% but tests are much lower

To avoid this, use a *validation sample* and stop when the accuracy on the validation data saturates.
Alternatively one can for example add terms to the error function (*weight decay*)

$$E \to E + \frac{\lambda}{2}\mathbf{w}^{\mathsf{T}}\mathbf{w} \tag{12}$$

making smaller weights more preferable.

In conclusion, both theory and experience goes into choosing hyper-parameters such as η, λ etc. in order to optimize the neural network performance.

Questions

Questions to the basics?