

13. Neural networks

- introduction
- training a neural network
- the backpropagation algorithm

Neural network success

neural networks achieved tremendous success in many real life applications such as

- speech recognition
- natural language processing
- image classifications
- recommendations systems
- cancer cell detection
- ...etc

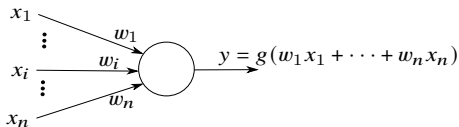
Neuron

an *artificial neural network* (NN) is composed of simple subsystems called *neurons*

Neuron symbol



Single-neuron output



- the output of a single-neuron is a function of a linear combination of inputs
- $f : \mathbb{R} \rightarrow \mathbb{R}$ is the *activation function*
- x_i is the i th input; $x = (x_1, \dots, x_n) \in \mathbb{R}^n$ the vector of inputs
- w_i is the *weight* multiplied by x_i ; $w = (w_1, \dots, w_n) \in \mathbb{R}^n$ is the weight vector

Activation functions

- *linear* (no activation): $g(v) = v$

- *softplus*:

$$g(v) = \log(1 + e^v)$$

- *sigmoid or logistic, soft step*:

$$g(v) = \frac{1}{1 + e^{-v}}$$

- *binary step*:

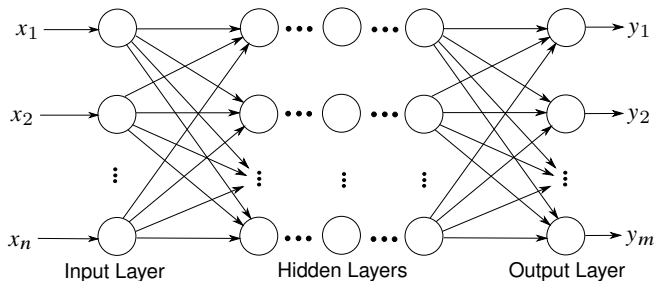
$$g(v) = \begin{cases} 0 & v \leq 0 \\ 1 & v > 0 \end{cases}$$

- *rectified linear unit (ReLU)*:

$$g(v) = \max(v, 0) = \begin{cases} 0 & v \leq 0 \\ v & v > 0 \end{cases}$$

Feedforward neural network

in *feedforward* NN, neurons are interconnected in layers; data flow in one direction



- the first layer is the *input layer*
- the last layer is the *output layer*
- middle layers are *hidden layers*
- NN is a mapping $g : \mathbb{R}^n \rightarrow \mathbb{R}^m$ that is a composition of functions
 - for three layer network $y = g(x) = g^3(g^2(g^1(x)))$ where each g^i is called a layer

Neural network predictor

Data fitting

- we have a mapping $F : \mathbb{R}^n \rightarrow \mathbb{R}^m$ we aim to approximate using a neural network
- we do not know F but we have observation data

$$(x_{d,1}, y_{d,1}), \dots, (x_{d,p}, y_{d,p}) \in \mathbb{R}^n \times \mathbb{R}^m$$

- each $y_{d,i}$ corresponds to the output of the map F for the input $x_{d,i}$, i.e.,

$$y_{d,i} = F(x_{d,i}), \quad i = 1, \dots, p$$

NN predictor

- consider a neural network as a specific mapping $g(x; w) : \mathbb{R}^n \rightarrow \mathbb{R}^m$
- w represent the weights of the neural network interconnections
- our objective becomes fine-tuning the network's interconnection weights such that

$$y = g(x; w) \approx F(x)$$

over our data

- $g(x; w)$ is called a neural network *predictor*

Outline

- introduction
- **training a neural network**
- the backpropagation algorithm

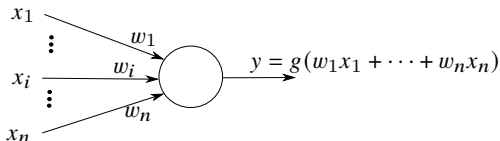
NN training

finding the weights of the NN can be cast as the following optimization problem

$$\text{minimize} \quad \sum_{i=1}^P \|y_{d,i} - g(w; x_{d,i})\|^2$$

- variable w is typically very large in practice
- this process is called the *training* or *learning* phase of the neural network
- after the training phase, NN is used to predict output for unseen input
- can be solved using any algorithm such gradient descent or Levenberg-Marquardt depending on activation functions

Single neuron training



$$\text{minimize} \quad (1/2) \sum_{i=1}^P (y_{d,i} - g(x_{d,i}^T w))^2$$

- variable $w = (w_1, \dots, w_n) \in \mathbb{R}^n$
- the choice of the method typically depends on the activation function f

Example: when f is the identity function, problem reduces to least squares problem:

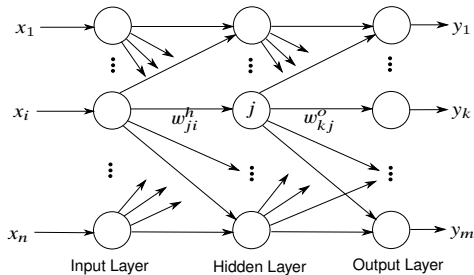
$$\text{minimize} \quad (1/2) \|y_d - X_d^T w\|^2$$

where $x_d = [x_{d,1} \cdots x_{d,p}] \in \mathbb{R}^{n \times P}$ and $y_d = (y_{d,1}, \dots, y_{d,p}) \in \mathbb{R}^P$

Outline

- introduction
- training a neural network
- **the backpropagation algorithm**

Three-layered neural network



- n inputs $x_i, i = 1, \dots, n$, and m outputs $y_s, s = 1, \dots, m$
- l neurons in hidden layer; outputs of neurons in hidden layer are $z_j, j = 1, \dots, l$
- inputs x_1, \dots, x_n are distributed to the neurons in the hidden layer
- $g_j^h : \mathbb{R} \rightarrow \mathbb{R}$ are activation functions of the neurons in hidden layer $j = 1, \dots, l$,
- g_s^o activation functions of the neurons in the output layer by, where $s = 1, \dots, m$

Input-output representation

- denote the input to the j th neuron in the hidden layer by v_j
- the output of the j th neuron in the hidden layer by z_j
- then, we have

$$v_j = \sum_{i=1}^n w_{ji}^h x_i$$
$$z_j = g_j^h \left(\sum_{i=1}^n w_{ji}^h x_i \right)$$

- the output from the s th neuron of the output layer is

$$y_s = g_s^o \left(\sum_{j=1}^l w_{sj}^o z_j \right)$$

Input-output representation

inputs $x_i, i = 1, \dots, n$ and the s th output y_s is related by

$$\begin{aligned}y_s &= g_s^o \left(\sum_{j=1}^l w_{sj}^o g_j^h(v_j) \right) \\&= g_s^o \left(\sum_{j=1}^l w_{sj}^o g_j^h \left(\sum_{i=1}^n w_{ji}^h x_i \right) \right) \\&= g_s(x_1, \dots, x_n)\end{aligned}$$

the overall mapping that the neural network implements is therefore given by

$$\begin{bmatrix} y_1 \\ \vdots \\ y_m \end{bmatrix} = \begin{bmatrix} g_1(x_1, \dots, x_n) \\ \vdots \\ g_m(x_1, \dots, x_n) \end{bmatrix}$$

The training problem

given single training set (x_d, y_d) , $x_d \in \mathbb{R}^n$ and $y_d \in \mathbb{R}^m$, problem reduces to

$$\text{minimize } (1/2) \sum_{s=1}^m (y_{ds} - y_s)^2$$

- y_s , $s = 1, \dots, m$, are outputs of the NN from the inputs x_{d1}, \dots, x_{dn}
- this minimization is taken over

$$w = \{w_{ji}^h, w_{sj}^o : i = 1, \dots, n, j = 1, \dots, l, s = 1, \dots, m\}$$

- the neural network requires minimizing the objective function

$$\begin{aligned} E(w) &= (1/2) \sum_{s=1}^m (y_{ds} - y_s)^2 \\ &= (1/2) \sum_{s=1}^m \left(y_{ds} - g_s^o \left(\sum_{j=1}^l w_{sj}^o g_j^h \left(\sum_{i=1}^n w_{ji}^h x_{di} \right) \right) \right)^2 \end{aligned}$$

- we can solve using the gradient method with stepsize α
- doing so leads to the backpropagation algorithm

Partial derivatives

- compute the partial derivative of E with respect to w_{sj}^o :

$$E(w) = (1/2) \sum_{p=1}^m \left(y_{dp} - g_p^o \left(\sum_{q=1}^l w_{pq}^o z_q \right) \right)^2$$

$$\text{where } z_q = g_q^h \left(\sum_{i=1}^n w_{qi}^h x_{di} \right), q = 1, \dots, l$$

- applying the chain rule, we derive:

$$\frac{\partial E}{\partial w_{sj}^o}(w) = -(y_{ds} - y_s) g_s^{o'} \left(\sum_{q=1}^l w_{sq}^o z_q \right) z_j = -\delta_s z_j$$

$$\text{where } \delta_s = (y_{ds} - y_s) g_s^{o'} \left(\sum_{q=1}^l w_{sq}^o z_q \right)$$

- the partial derivative of E concerning w_{ji}^h is relation:

$$\frac{\partial E}{\partial w_{ji}^h}(w) = -x_{di} \delta_j, \quad \delta_j = g_j^{h'} \left(\sum_{i=1}^n w_{ji}^h x_{di} \right) \sum_{s=1}^m \delta_s w_{sj}^o$$

The back-propagation algorithm

$$w_{sj}^{o(k+1)} = w_{sj}^{o(k)} + \alpha \delta_s^{(k)} z_j^{(k)}$$
$$w_{ji}^{h(k+1)} = w_{ji}^{h(k)} + \alpha \left(\sum_{p=1}^m \delta_p^{(k)} w_{pj}^{o(k)} \right) g_j^{h'}(v_j^{(k)}) x_{di}$$

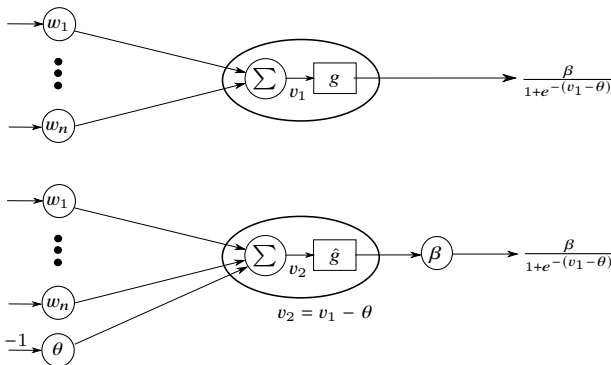
where α is the (fixed) step size and

$$v_j^{(k)} = \sum_{i=1}^n w_{ji}^{h(k)} x_{di}, \quad z_j^{(k)} = g_j^h(v_j^{(k)})$$
$$y_s^{(k)} = g_s^o \left(\sum_{q=1}^l w_{sq}^{o(k)} z_q^{(k)} \right), \quad \delta_s^{(k)} = (y_{ds} - y_s^{(k)}) g_s^{o'} \left(\sum_{q=1}^l w_{sq}^{o(k)} z_q^{(k)} \right)$$

- $\delta_1^{(k)}, \dots, \delta_m^{(k)}$ are propagated back from the output layer to the hidden layer
- **forward pass of the algorithm:** using the inputs x_{di} and the current set of weights, we first compute the quantities $v_j^{(k)}$, $z_j^{(k)}$, $y_s^{(k)}$, and $\delta_s^{(k)}$, in turn
- **reverse pass of the algorithm:** compute the updated weights using the quantities computed in the forward pass

Generalized sigmoid function

$$g(v) = \frac{\beta}{1 + e^{-(v-\theta)}}$$



References and further readings

- E. K.P. Chong, Wu-S. Lu, and S. H. Zak. *An Introduction to Optimization: With Applications to Machine Learning*. John Wiley & Sons, 2023. (ch 13)