13. Neural networks

- introduction
- training a neural network
- the backpropagation algorithm

Neural network success

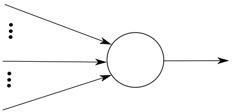
neural networks achieved tremendous success in many real life applications such as

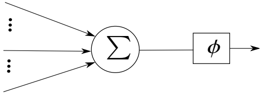
- speech recognition
- natural language processing
- image classifications
- recommendations systems
- cancer cell detection
- ...etc

Neuron

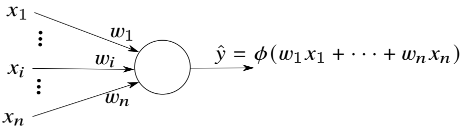
an *artificial neural network* (NN) is composed of simple subsystems called *neurons*

Neuron symbol





Single-neuron output



- output $\hat{y}$ is a function of a linear combination of inputs
- $\phi : \mathbb{R} \to \mathbb{R}$ is the *activation function*
- x_i is the ith input; $x = (x_1, \ldots, x_n) \in \mathbb{R}^n$ the vector of inputs
- w_i is the *weight* multiplied by x_i; $w = (w_1, \ldots, w_n) \in \mathbb{R}^n$ is the weight vector

Activation functions

- *linear* (no activation): $\phi(v) = v$
- *softplus*:

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

- *sigmoid or logistic, soft step*:

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

- *binary step*:

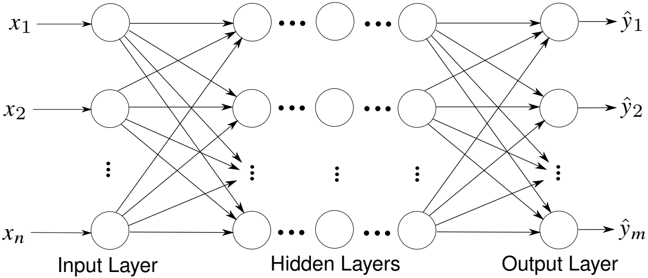
$$\phi(v) = \begin{cases} 0 & v \leq 0 \\ 1 & v > 0 \end{cases}$$

- *rectified linear unit* (ReLU):

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

Feedforward neural network

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



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

Neural network predictor

Data fitting

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

$$(x^{(1)}, y^{(1)}), \dots, (x^{(N)}, y^{(N)}) \in \mathbb{R}^n \times \mathbb{R}^m$$

- each $y^{(i)}$ corresponds to the output of the map F for the input $x^{(i)}$, *i.e.*,

$$y^{(i)} = F(x^{(i)}), \quad i = 1, \dots, N$$

NN predictor

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

$$\hat{y} = g(x; w) \approx F(x) \quad \text{over our data}$$

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

Outline

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

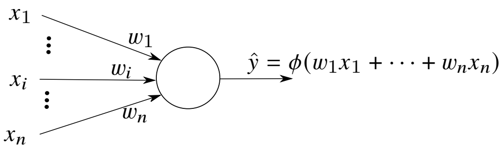
NN training

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

$$\text{minimize} \quad \sum_{i=1}^{N} \|y^{(i)} - g(w; x^{(i)})\|^2$$

- 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 (depending on activation functions)
  - gradient descent (called backpropagation)
  - Levenberg-Marquardt method

Single neuron training



$$\text{minimize} \quad (1/2)\sum_{i=1}^{N}(y^{(i)} - g(x^{(i)T}w))^2$$

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

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

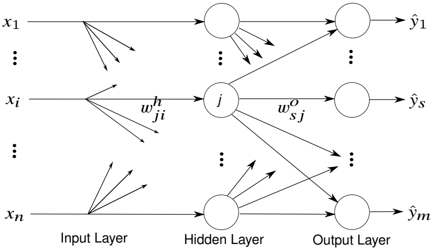
$$\text{minimize} \quad (1/2)\|y - X^T w\|^2$$

where $X = [x^{(1)} \ \cdots \ x^{(N)}] \in \mathbb{R}^{n \times N}$ and $y = (y^{(1)}, \ldots, y^{(N)}) \in \mathbb{R}^N$

Outline

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

Three-layered neural network



- n inputs x_i, $i = 1, \ldots, n$, and m outputs $\hat{y}_s$, $s = 1, \ldots, m$
- l neurons in hidden layer
  - w^h_{ji}: weight from the ith input neuron to the jth hidden neuron
  - w^o_{sj}: weight from the jth hidden neuron to the sth output neuron
- $\phi^h_j : \mathbb{R} \to \mathbb{R}$ are activation functions of the neurons in hidden layer $j = 1, \ldots, l$,
- ϕ^o_s activation functions of the neurons in the output layer by, where $s = 1, \ldots, m$

Input-output representation

- denote the input to the jth neuron activation function in hidden layer by v_j
- the output of the jth 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 sth 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, $i = 1, \ldots, n$ and the sth output y_s is related by

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

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

$$\begin{bmatrix} \hat{y}_1 \\ \vdots \\ \hat{y}_m \end{bmatrix} = \begin{bmatrix} g_1(x_1, \ldots, x_n) \\ \vdots \\ g_m(x_1, \ldots, x_n) \end{bmatrix}$$

The training problem

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

$$\text{minimize} \quad (1/2) \sum_{s=1}^{m} (y_s - \hat{y}_s)^2$$

- $\hat{y}_s$, $s = 1, \ldots, m$, are outputs of the NN from the inputs $x_1, \ldots, x_n$
- this minimization is taken over

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

- the neural network requires minimizing the objective function

$$\begin{aligned} f(w) &= (1/2) \sum_{s=1}^{m} (y_s - \hat{y}_s)^2 \\ &= (1/2) \sum_{s=1}^{m} \left(y_s - g_s^o \left(\sum_{j=1}^{l} w_{sj}^o g_j^h \left(\sum_{i=1}^{n} w_{ji}^h x_i \right) \right) \right)^2 \end{aligned}$$

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

Partial derivatives

- compute the partial derivative of f with respect to w^o_{sj}:

$$f(w) = (1/2) \sum_{q=1}^{m} \left(y_q - g^o_q \left(\textstyle\sum_{r=1}^{l} w^o_{qr} z_r \right) \right)^2$$

where $z_r = g^h_r \left(\sum_{i=1}^{n} w^h_{ri} x_i \right)$

- applying the chain rule, we derive:

$$\frac{\partial f}{\partial w^o_{sj}}(w) = -(y_s - \hat{y}_s) g^{o\prime}_s \left(\textstyle\sum_{r=1}^{l} w^o_{sr} z_r \right) z_j = -\delta_s z_j$$

where $\delta_s = (y_s - \hat{y}_s) g^{o\prime}_s \left(\sum_{r=1}^{l} w^o_{sr} z_r \right)$

- the partial derivative of f with respect to w^h_{ji}:

$$\frac{\partial f}{\partial w^h_{ji}}(w) = -x_i \delta_j, \quad \delta_j = g^{h\prime}_j \left(\textstyle\sum_{i=1}^{n} w^h_{ji} x_i \right) \sum_{s=1}^{m} \delta_s w^o_{sj}$$

The backpropagation algorithm

$$w_{sj}^{o(k+1)} = w_{sj}^{o(k)} + \alpha\delta_s^{(k)} z_j^{(k)}$$
$$w_{ji}^{h(k+1)} = w_{ji}^{h(k)} + \alpha \left(\sum_{q=1}^m \delta_q^{(k)} w_{qj}^{o(k)}\right) g_j^{h'}(v_j^{(k)})x_i$$

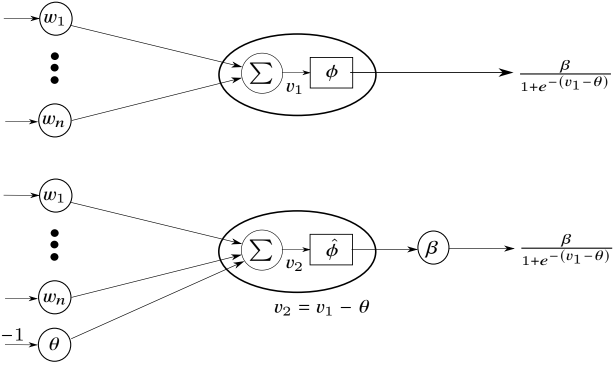
where α is the (fixed) step size and

$$v_j^{(k)} = \sum_{i=1}^n w_{ji}^{h(k)} x_i, \quad z_j^{(k)} = g_j^h(v_j^{(k)})$$
$$\hat{y}_s^{(k)} = g_s^o\left(\sum_{r=1}^l w_{sr}^{o(k)} z_r^{(k)}\right), \quad \delta_s^{(k)} = (y_s - \hat{y}_s^{(k)}) g_s^{o'}\left(\sum_{r=1}^l w_{sr}^{o(k)} z_r^{(k)}\right)$$

- $\delta_1^{(k)}, \ldots, \delta_m^{(k)}$ are propagated back from the output layer to the hidden layer
- **forward pass of the algorithm:** using the inputs x_i and the current set of weights, we first compute the quantities $v_j^{(k)}$, $z_j^{(k)}$, $\hat{y}_s^{(k)}$, 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

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



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)