Neural Network

- Neural networks are a highly flexible black box prediction algorithm.
- Initially neural networks were designed to mimic information processing of the brain's neurons and axons. They, however, have progressed substantially beyond this simplistic brain model.
- Similar models were around in the statistical sciences for several decades.
- Several packages are available:
  - **`nnet`** is part of the standard distribution and predominately is used for multinomial logistic classification models.
  - **`RSNNS`** provide a complete of neural network functionality
  - **`keras`** with **`TENSORFLOW`** is a professional software implementation for deep learning and it is optimized to use the parallel computation capabilities of **NIVIDA** graphics cards.
- Strength and weaknesses of artificial neural networks (ANN)

Strength	Weaknesses
Can be used in a classification context and for numerical predictions	Extremely computationally intensive with a small likelihood of the algorithms to crash during the gradient search
Can handle extremely complex pattern	Prone to overfitting training data
No assumptions about the data's relationships	Results virtually cannot be interpreted except for very simple cases.
Extensions to dynamic models (adaptive NN) and image processing (convolution NN) is possible	Risk of the algorithm be become stuck in a local minimum of the loss-function.
Self-learning with generative adversarial networks is possible (generative and autoencoder NN)	

Deep Learning

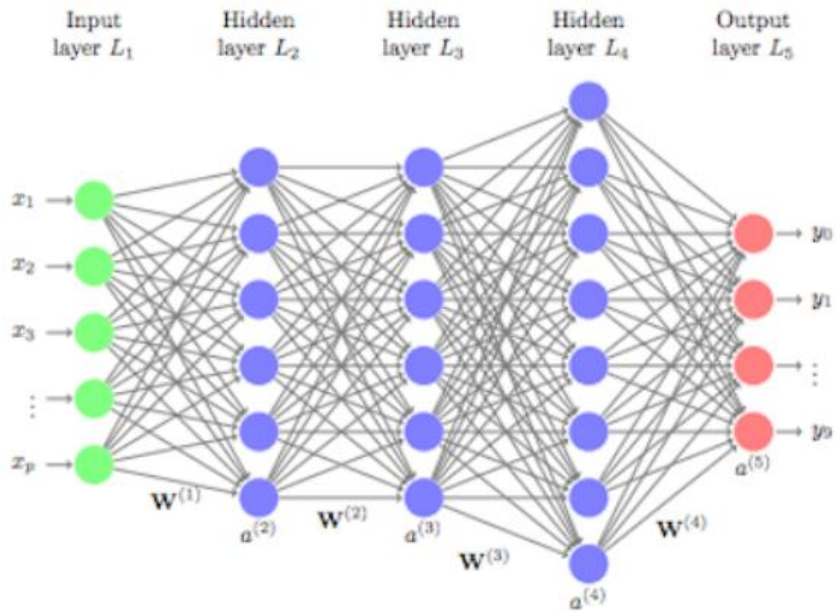


Figure 13.3: Representation of a deep feedforward neural network.

- It is a specialized subfield of artificial neural networks with a large number of intermediate layers.
- With the increase in affordable computing power and an increasing availability of data deep learning network became prominent in the early 2010.
- It is based on successive layers of rules with increasingly more meaningful representations of the target y.
- One can think of neural network as a series of filters (the layers) that distill the input information x into a purified version that is closely related to the output y.
- Building blocks of understanding neural networks are logistic regression and gradient descent optimizers.
- A sequence of layers weight feature input vectors $\mathbf{x}_i$ that will generate an output label y_i:

$$y_i = f_{NN}(\mathbf{x}_i) = f_3\big(f_2(f_1(\mathbf{x}_i))\big)$$

- Deep learning involves more than two non-output layers. In $y_i = f_{NN}(\mathbf{x}_i) = f_3\big(f_2(f_1(\mathbf{x}_i))\big)$ the function f_3 is the output layer.

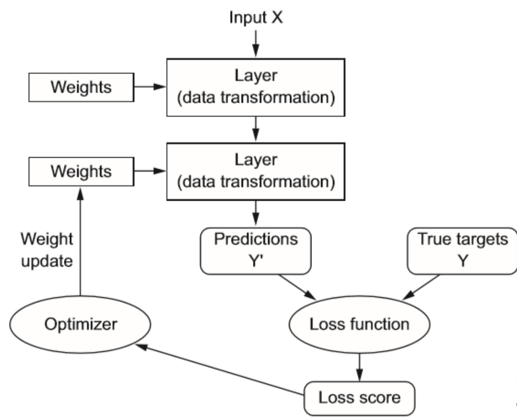


Figure 1.9 The loss score is used as a feedback signal to adjust the weights.

- The objective of specifying the layer functions $f_l(\quad)$ is to minimize the prediction error, which is measured by a loss score functions.

Neural Network topology

- Each neuron comprises for the i^{th} observation of a set of input features $\{x_{i1}, \dots, x_{im}\}$, that are weighted by some initially unknown coefficients $\{w_1, \dots, w_m\}$. In addition, there may be a bias feature $x_{i0} = 1$ with its bias weight w_0.

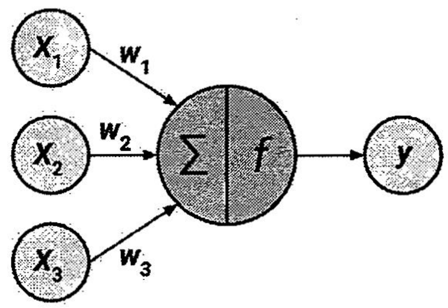


Figure 7.2: An artificial neuron is designed to mimic the structure and function of a biological neuron

- These are combined together a linear function

$$\sum_{j=0}^{m} w_j \cdot x_{ij}$$

- A non-linear activation function is applied on this linear combination to produce the output of a neuron

$$y_i = f\left(\sum_{j=0}^{m} w_j \cdot x_{ij}\right)$$

- There are three common choices for activation functions

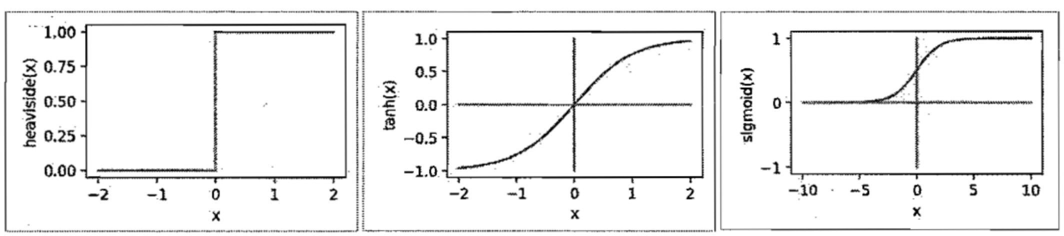


Figure 9.7 Three typical activation functions of *Heaviside, tanh* and *sigmoid.*

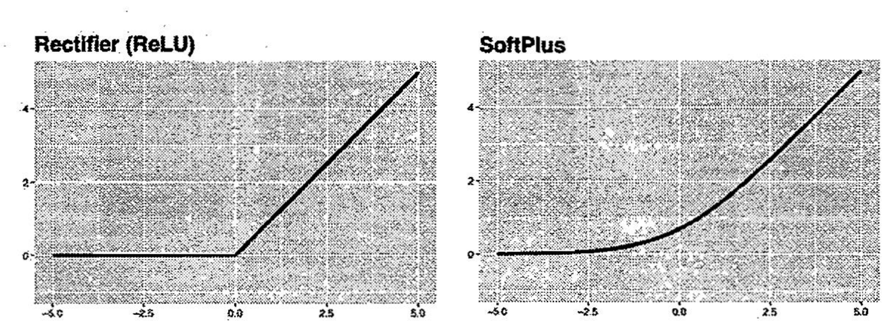


Figure 7.13: The softplus activation function provides a smooth, differentiable approximation of ReLU

- The rectifier activation function is nowadays replaced by the differentiable *softplus* function $f(x) = \log(1 + \exp x)$.
- A simple feed-forward neural network with just one hidden layer $\mathbf{z}_n$ consisting of N neurons is displayed below. The output of this layer is again weighted by w_{jk} and an activation function is applied to each weighted sum to produce the predicted value y_{ik}.

last calculation do not need activation function

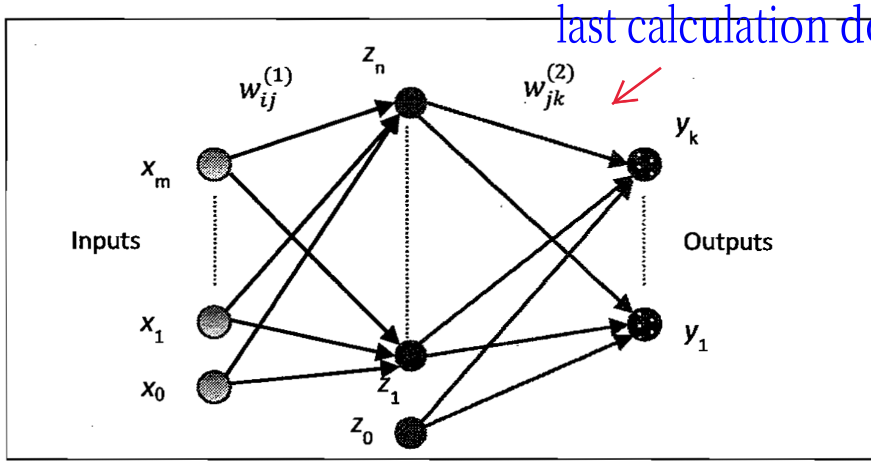


Figure 9.8 Feed-forward neural network architecture.

- For classification purposed there are as many y_{ik}'s as there are K classes. Their activation function is the *softmax* function

$$g(x_k) = \frac{\exp x_k}{\sum_{l=1}^{K} \exp x_l}$$

- This expression is equivalent to the predicted class probability of multivariate logistic regression.
- For metric target variables there is only one y_i and the activation function is usually a simple identity function.
- Combining both steps a neutral network can be writing in as

activation function for Nth neuron

activation function for prediction y

$$y_k = g\left(\sum_{n=0}^{N} w_{kn} \cdot f_n\left(\sum_{j=0}^{m} w_{nj} \cdot x_j\right)\right)$$

- In the unknown set of weights w_{kj} and w_j which must be estimated iteratively by a backpropagation gradient descent algorithm.

Demonstration: Solving a bivariate regression problem with gradient search

- The loss function in regression analysis is defined by

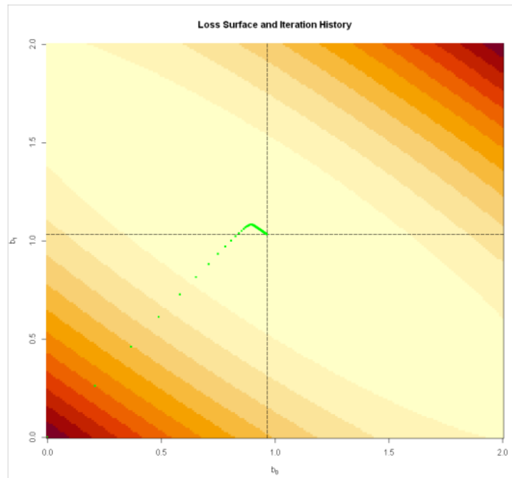
$$l(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{n} \cdot \sum_{i=1}^{n} \left(y_i - (b_0 + b_1 \cdot x_i)\right)^2$$

- The first derivatives with respect to the regression coefficients b_0 and b_1 are:

$$\frac{\partial l}{\partial b_0} = \frac{1}{n} \cdot \sum_{i=1}^{n} -2 \cdot \left(y_i - (b_0 + b_1 \cdot x_i)\right)$$

$$\frac{\partial l}{\partial b_1} = \frac{1}{n} \cdot \sum_{i=1}^{n} -2 \cdot x_i \cdot \left(y_i - (b_0 + b_1 \cdot x_i)\right)$$

- These derivatives measure the slope of the loss-function at any given point (b_0, b_1).
- At that point, where $\frac{\partial l}{\partial b_0} = 0$ and $\frac{\partial l}{\partial b_1} = 0$ the optimal point (b_0, b_1), where the quadratic loss function is at its minimum, has been found.



- In order to find this minimum location iteratively the tentative point is updated by following the slope at its current location.
- The current location $\left(b_{0,i}, b_{1,i}\right)$ is becomes:

$$b_{0,i+1} = b_{0,i} - \alpha \cdot \frac{\partial l}{\partial b_0}$$

$$b_{1,i+1} = b_{1,i} - \alpha \cdot \frac{\partial l}{\partial b_1}$$

- The derivatives, which are expressed as growth rate of a function $f(x)$ as x is increasing, are subtracted from the current parameter values because the aim is to minimize the loss function.

- The step-length (learning rate) α determines how fast the minimum is found and whether for large α's the minimum is skipped over.

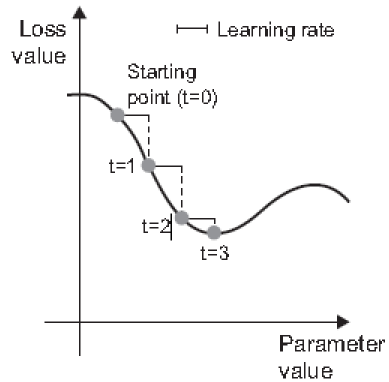


Figure 2.11 SGD down a 1D loss curve (one learnable parameter)

- On the other hand, if the step length is too short the solution may be stuck in a local minimum.

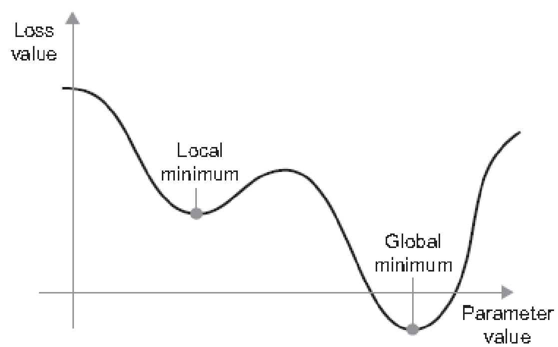


Figure 2.13 A local minimum and a global minimum