Neural networks

Chmiela Bartosz

University of Wrocław

March 6, 2022

Contents

Introduction

- Deep learning,
- Supervised learning method,
- Inspired by biological neural networks.

Basic idea

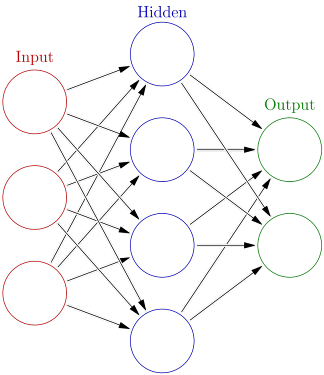


Figure: Layers of an artificial neural network, by Glosser.ca - Own work, source: wiki.

History

- Research started in 1940-50s,
- Perceptron is created in 1958,
- Self-driving car in 1995,
- NNs achieve human level pattern recognition in 2010s.

Projection pursuit regression

Projection pursuit regression

Let $X \in \mathbb{R}^p$ be the input vector, Y the target and $\omega_m \in \mathbb{R}^p$, $m = 1, \ldots, M$ unit vector of unknown parameters. The projection pursuit regression (PPR) has the form:

$$f(X) = \sum_{m=1}^{M} g_m \left(w_m^T X \right).$$

The function $g_m : \mathbb{R} \to \mathbb{R}^p$ is unknown and is called a ridge function.

Projection pursuit regression

Universal approximator

For Arbitrarily large M and appropriate choice of g_m the PPR model can approximate any continuous function in $\mathbb{R}^p$ arbitrarily well. Such class of models is called a *universal approximator*.

Single index model

When $M = 1$ the model is known in econometric as the *single index model*.

Fitting PPR

Given the training data $(x_i, y_i),\ i = 1, \ldots, N$ we seek to minimize:

$$\sum_{i=1}^{N} \left[y_i - \sum_{m=1}^{M} g_m(w_m^T x_i) \right]^2 .$$

For $M = 1$ we have one-dimensional smoothing problem and we can apply a smoothing spline to obtain estimate of g.

Gaussian-Newton search

This is a quasi-Newton method where the part of the Hessian involving the second derivative of g is discarded. Let ω_{old} be the current estimate for ω.

$$g(\omega^T x_i) \approx g(\omega_{old}^T x_i) + g'(\omega_{old}^T x_i)(\omega - \omega_{old})^T x_i.$$

and then:

$$\sum_{i=1}^{N} \left[y_i - g(w^T x_i)\right]^2 \approx$$
$$\sum_{i=1}^{N} g'(\omega_{old}^T x_i) \left[\left(\omega_{old}^T x_i + \frac{y_i - g(\omega_{old}^T x_i)}{g'(\omega_{old}^T x_i)}\right) - \omega^T x_i\right]^2.$$

Connection to neural networks

Central idea

The central idea is to extract linear combinations of the inputs and then model the target as a nonlinear function of these features.

The PPR evolved in the domain of semiparametric statistics and smoothing. The next step are neural networks.

Single layer neural network

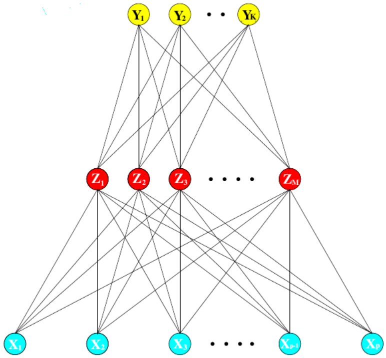


Figure: Schematic of a single hidden layer, feed-forward neural network, source: *The elements of statistical learning, T Hastie, R Tibshirani, JH Friedman*.

Single layer neural network

$$
\begin{aligned}
Z_m &= \sigma\left(\alpha_{0m} + \alpha_m^T X\right), \quad m = 1, \dots M, \\
T_k &= \beta_{0k} + \beta^T Z, \quad k = 1, \dots, K, \\
f_k(X) &= g_k(T), \quad k = 1, \dots, K,
\end{aligned}
$$

Where $Z = (Z_1, Z_2, \dots, Z_M)$, $T = (T_1, T_2, \dots, T_K)$ and $\sigma(v) = (1 + e^{-v})^{-1}$

Output function

Regression

$$g_k(T) = T_k \text{ (identity).}$$

K-class classification

$$g_k(T) = \frac{e^{T_k}}{\sum_{l=1}^{K} e^{T_l}} \text{ (softmax).}$$

Activation function

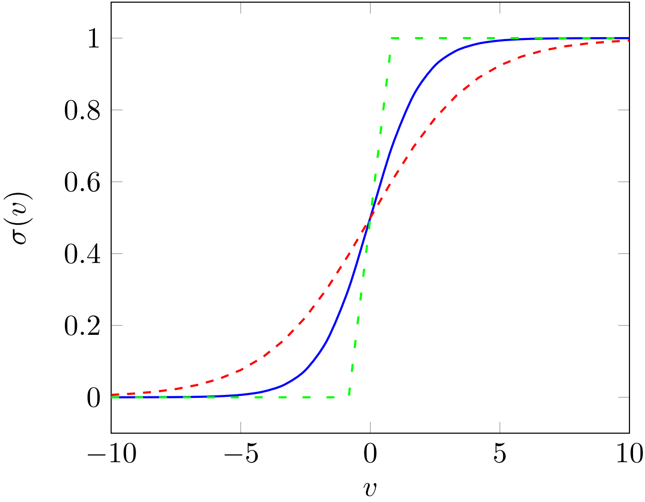


Figure: Sigmoid functions of form $\sigma(sv)$ with: $s = 1$ (blue), $s = \frac{1}{2}$ (red), $s = 10$ (green).

Connection to PPR

$$\begin{aligned} g_m(\omega_m^T X) &= \beta_m \sigma(\alpha_{0m} + \alpha_m^T X) \\ &= \beta_m \sigma(\alpha_{0m} + \|\alpha_m\|(\omega_m^T X)), \end{aligned}$$

where $\omega_m = \frac{\alpha_m}{\|\alpha_m\|}$ is the m-th unit vector.

Parameters

Set of weights θ:

$$
\begin{aligned}
\{\alpha_{0m}, \alpha_m;\ m = 1, \ldots, M\}\ & M(p+1)\text{weights}, \\
\{\beta_{0k}, \beta_k;\ k = 1, \ldots, K\}\ & K(M+1)\text{weights}.
\end{aligned}
$$

Measure of fit

Regression (sum-of-squared errors)

$$R(\theta) = \sum_{k=1}^{K} \sum_{i=1}^{N} (y_{ik} - f_k(x_i))^2.$$

Classification (cross-entropy)

$$R(\theta) = -\sum_{k=1}^{K} \sum_{i=1}^{N} y_{ik} \log f_k(x_i),$$

and classifier:

$$G(x) = \underset{k}{\operatorname{argmax}} f_k(x).$$

Back-propagation

Let $z_{mi} = \sigma(\alpha_{0m} + \alpha_m^T x_i)$ and $z_i = (z_{1i}, z_{2i}, \ldots, z_{Mi})$, and:

$$R(\theta) \equiv \sum_{i=1}^{N} R_i = \sum_{k=1}^{K} \sum_{i=1}^{N} (y_{ik} - f_k(x_i))^2.$$

with derivatives

$$\frac{\partial R_i}{\partial \beta_{km}} = -2(y_{ik} - f_k(x_i)) g_k'(\beta_k^T z_i) z_{mi},$$

$$\frac{\partial R_i}{\partial \alpha_{ml}} = -\sum_{k=1}^{K} 2(y_{ik} - f_k(x_i)) g_k'(\beta_k^T z_i) \beta_{km} \sigma'(\alpha_m^T x_i) x_{il}.$$

Gradient descent

Gradient descent update at $r+1$ iteration:

$$\beta_{km}^{(r+1)} = \beta_{km}^{(r)} - \gamma_r \sum_{i=1}^{N} \frac{\partial R_i}{\partial \beta_{km}^{(r)}},$$

$$\alpha_{ml}^{(r+1)} = \alpha_{ml}^{(r)} - \gamma_r \sum_{i=1}^{N} \frac{\partial R_i}{\partial \alpha_{ml}^{(r)}},$$

where γ_r is the learning rate.

Back-propagation equations

Let's write as

$$\frac{\partial R_i}{\partial \beta_{km}} = \delta_{ki} z_{mi}, \quad \frac{\partial R_i}{\partial \alpha_{ml}} = s_{mi} x_{il}.$$

From their definitions, these satisfy

$$s_{mi} = \sigma'(\alpha_m^T x_i) \sum_{k=1}^{K} \beta_{km} \delta_{ki},$$

known as the *back-propagation equations*.

Additional info

- back-propagation is simple and can be efficient,
- batch learning,
- training epoch,
- learning rate.

Issues in training neural networks

- Starting values,
- overfitting,
- scaling of the inputs,
- number of hidden units and layers,
- multiple minima

Starting values

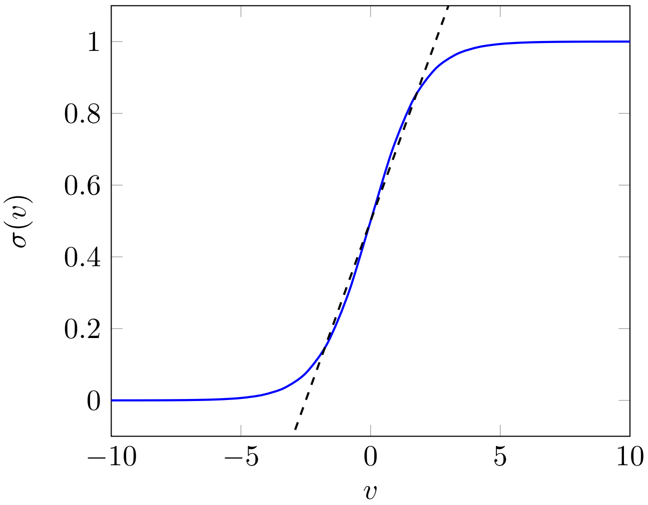


Figure: Sigmoid function $\sigma(v)$ and approximation by a line.

Overfitting

Often neural networks have too many weights and will overfit the data at the global minimum of R.

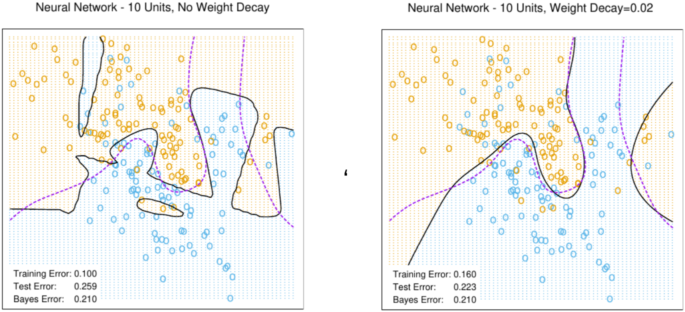
Weight decay

$$R(\theta) + \lambda J(\theta),\ \lambda \geq 0,$$

where

$$J(\theta) = \sum_{km} \beta_{km}^2 + \sum_{ml} \alpha_{ml}^2$$

Weight decay example



(a) NN with no weight decay.

(b) NN with weight decay.

Figure: Example of neural network, source: *The elements of statistical learning, T Hastie, R Tibshirani, JH Friedman*.

Scaling of the input

- Scaling of input has influence on the weights,
- standardize inputs to have mean 0 and standard deviation 1,
- random uniform weights over the range $[-0.7, 0.7]$.

Number of hidden units and layers

- generally better to have too many hidden units,
- typically the number of hidden units is in the range of 5 to 100,
- large number of hidden units are trained with regularization,
- choice of the number of hidden layers is guided by background knowledge.

Multiple minima

- $R(\theta)$ is nonconvex, posseses many local minima,
- solution depends on the starting weights,
- try a number of random starting configurations,
- it's better to average over the collection of networks.

Thank you for your attention.