Lecture 7 – Neural Nets

Martin Spindler

2016-04-08

Recap: What have we already learnt?

1. Introduction (Definitions, Basic Concepts, Challenges in High-Dimensions)
2. Linear Regression and Extensions (Linear Regression, Regression and Smoothing Splines)
3. Ridge Regression
4. Lasso Regression (Basic Principle, Some Results, Extensions)
5. Of Trees and Forests (Regressions Trees, Bagging, Random Forests)

Today: Neural Nets / Deep Learning

Neural Networks | Introduction

- Inspired by the mode of operation of the brain, imitation of the human brain
- Idea: Extract linear combinations of the inputs as derived features, and model the target (Y) as a nonlinear function of these features
- Fields: Statistics, Artificial Intelligence

Project Pursuit Regression

- Input vector X with p components; target Y
- $\omega_m, m = 1, \ldots, M$ unit $p-$vectors of unknown parameters
- Project Pursuit Regression (PPR) model:

$$f(x) = \sum_{m=1}^{M} g_m(\omega_m' x)$$

  $V_m = \omega_m' x$ derived feature; projection on ω_m
- g_m estimated along with ω_m by flexible smoothing methods
- $g_m(\omega_m' x)$ "ridge function" in $\mathbb{R}^p$
- Useful for prediction; difficult to interpret

Neural Networks

- Large class of models / learning methods
- Here: single hidden layer back-propagation network / single layer perceptron
- Two-stage regression or classification model represented by network diagram
- Can be seen as nonlinear statistical models
- Diagram cf blackboard

Neural Networks

- $Z_m = \sigma(\alpha_{0m} + \alpha_m' x)), m = 1, \ldots, M$
- $T_k = \beta_{0k} + \beta_k' z, k = 1, \ldots, K$
- $f_k(x) = g_k(T), k = 1, \ldots, K$

Neural Networks

- Activation function: $\sigma(v) = \frac{1}{1+e^{-v}}$ (sigmoid) (cf blackboard)
- Regression case: $g_k(T) = T_k$;
- Classification case: $g_k(T) = \frac{e^{T_k}}{\sum_{l=1}^{K} e^{T_l}}$ (softmax fct.)
- Related to PPR
- Measure of fit $R(\theta)$: sum-of-squared errors (regression), or squared error / cross entropy
- Estimation: $R(\theta)$ by gradient descent ("back propagation"); regularization might be needed

Fitting Neural Networks

- unknown parameters, called weights, θ:
- $\{\alpha_{0m}, \alpha_m; m = 1, 2, \ldots, M\}$ $M(p+1)$ weights
- $\{\beta_{0m}, \beta_m; k = 1, 2, \ldots, K\}$ $K(p+1)$ weights
- Criterion function:
  $R(\theta) = \sum_{k=1}^{K} \sum_{i=1}^{N} (y_{ik} - f_k(x_i))^2 = \sum_{i=1}^{N} R_i$
- Derivatives: $\frac{\partial R_i}{\partial \beta_{km}} = -2(y_i - f_k(x_i)) g_k'(\beta_k^T z_i) z_{mi}$
- Analog Derivatives $\frac{\partial R_i}{\partial \alpha_{ml}}$

Fitting Neural Networks

A gradient descent updata at the (r+1)st iteration is given by

* $\beta_{km}^{(r+1)} = \beta_{km}^{(r)} - \gamma_r \sum_{i=1}^{N} \frac{\partial R_i}{\partial \beta_{km}^{(r)}}$
* $\alpha_{ml}^{(r+1)} = \alpha_{ml}^{(r)} - \gamma_r \sum_{i=1}^{N} \frac{\partial R_i}{\partial \alpha_{ml}^{(r)}}$
* γ_r learning rate

Fitting Neural Networks

- Rewrite Derivatives as
- $\frac{\partial R_i}{\partial \beta_{km}} = \delta_{ki} z_{mi}$
- $\frac{\partial R_i}{\partial \alpha_{ml}} = s_{mi} x_{il}$
- $s_{mi} = \sigma'(\alpha_m^T x_i) \sum_{k=1}^{K} \beta_{km} \delta_{ki}$ (*)
- δ_{ki} and s_{mi} "errors"

Fitting Neural Networks

Estimation via back-propagation equations:

- Updates in updating step with two-pass algorithm
- Forward pass: current weights are fixed, calculate $\hat{f}_k(x_i)$
- Backward pass calculate δ_{ki}, back-propogate via (*), calculate gradients and update.

Neural Networks

- Starting values: random values near zero. Intuition: model starts out nearly linear and becomes nonlinear as the weights increase.
- Overfitting: to prevent overfitting early stopping and penalization (weight decay; $R(\theta) + \lambda J(\theta)$)
- Scaling of the inputs: large effects on the quality of the final solution. Default: standarization and normlization of of inputs (mean zero and unit variance)
- Number of hidden units and layers: better to have too many hidden units than too few. (Flexibility + Regularization!)
- Multiple Minima: nonconvex crtierio function with many local minima (different starting values, average of predictions of collection of neural nets)