Nonlinear models

Jitkomut Songsiri

Department of Electrical Engineering
Faculty of Engineering
Chulalongkorn University

CUEE

February 14, 2023

Outline

Regression splines

- basis function
- regression splines
- smoothing splines

Basis functions

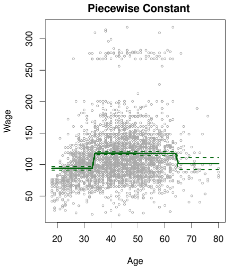
fit a response y_i with a linear combination of basis functions

$$y_i = f(x_i) = \beta_0 + \beta_1 g_1(x_i) + \beta_2 g_2(x_i) + \cdots + \beta_M g_M(x_i) + e_i, \quad i = 1, 2, \ldots, N$$

- the basis functions $g_1(\cdot), \ldots, g_M(\cdot)$ are known and fixed
- example of g_j: polynomial, piecewise constant, sine/cosine in Fourier series
  - $g_j(x) = x_j$ for $j = 1, \ldots, p$ recovers the original linear model
  - $g_j(x) = x_j^2$ or $g_j(x) = x_j x_k$ yields higher-order polynomial terms
  - $g_j(x) = \log(x_j), \sqrt{x_j}, \ldots$ permits other nonlinear transformations
  - $g_j(x) = I(l \leq x_k < u)$, an indicator function for a region of x_k
- the model is linearly parametrized in $\beta_0, \ldots, \beta_p$; they can be estimated using linear least-squares

Example: Wage dataset

piecewise-constant fit on `Wage` dataset



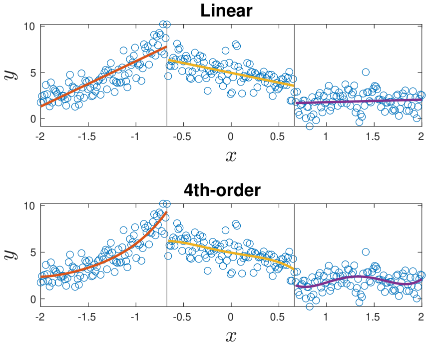
- take age as ordered categorical variable
- $g_j(x) = I(c_j \leq x \leq c_{j+1})$ (step function)
- the breakpoints in x must be chosen to capture a trend change in y

next,

- a general goal is to devise a flexible f that explains y
- polynomials are one of good choices but limited by their global nature (adjusting coefficients by little can make the function not generalize well for other x)
- we focus on regression splines that are used for local polynomial representation

Regression splines

piecewise polynomial fit with K knots



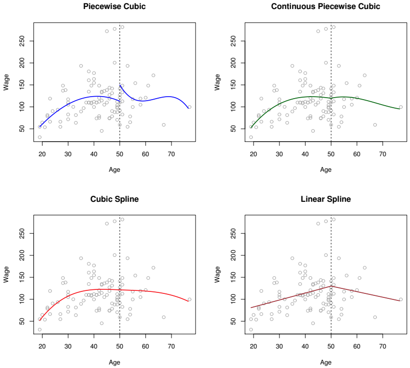
use different polynomials in each interval of x

$$y_i = \begin{cases} p_1(x_i), & \text{if } x_i \leq c_1, \\ p_2(x_i), & \text{if } c_1 < x_i \leq c_2, \\ & \vdots \\ p_K(x_i), & \text{if } c_{K-1} < x_i \leq c_K \\ p_{K+1}(x_i), & \text{if } x_i > c_K \end{cases}$$

- placing K knots into the range of X results in fitting $K+1$ polynomials
- fitting n-degree polynomial with K knots use $(n+1)(K+1)$ degree of freedoms
- immediate flaw: the fitted function is not continuous

Splines: polynomial fit with constraints

we can impose additional constraints at c (breakpoints)



- continuity: $p_j(c) = p_{j+1}(c)$
- smoothness:

$$p'_j(c) = p'_{j+1}(c), \quad p''_j(c) = p''_{j+1}(c)$$

  (derivatives are continuous)
- each constraint frees up one degree of freedom

definition: a degree-d spline is a piecewise degree-d polynomial with continuity in derivatives up to degree $d-1$ at each knot

Basis representation of splines

example: let number of knots be K (here, two knots at c_1 and c_2)

no.of parameters $=$ (no.of knots+1) $\times$ (no. parameters per region) - (no.of knots) $\times$ (no. of constraints per knot)

- linear spline: no.parameters $= (K+1)(2) - K \cdot 1 = K+2$

$$g_1(x) = 1, \quad g_2(x) = x, \quad g_3(x) = (x - c_1)_+, \quad g_4(x) = (x - c_2)_+$$

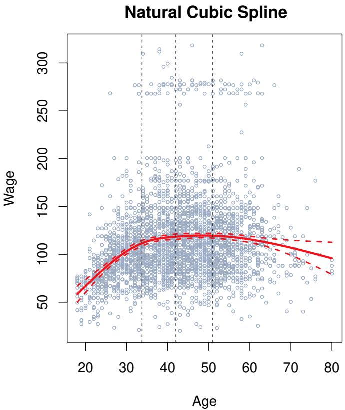
- cubic spline: no.parameters $= (K+1)(4) - K \cdot 3 = K+4$

$$g_1(x) = 1, \quad g_2(x) = x, \quad g_3(x) = x^2, \quad g_4(x) = x^3$$
$$g_5(x) = (x - c_1)_+^3, \quad g_6(x) = (x - c_2)_+^3$$

the notation $z_+ \triangleq \max(z, 0)$ denotes the positive part of z

Spline knots: number and locations

guideline: more knots in regions for which the model should be more flexible

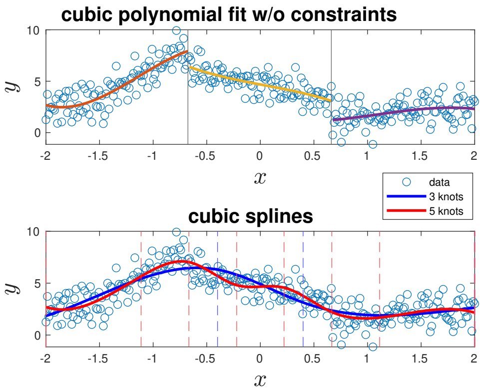


- we specified 4 degrees of freedom
- knot locations were chosen automatically as the 25th, 50th and 75th percentiles of age (in fact, there are 5 knots including the boundary for natural cubic spline)
- the number of knots can be tried out by using cross-validation

there are many rules to choose the locations of knots

Example: cubic spline

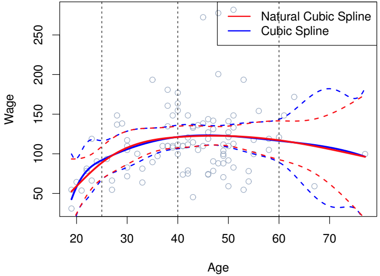
fit a cubic spline using `spap2` in MATLAB



- knots assigned by `aptknt` (MATLAB) (others include `optknt`,`augknt`,`newknt`)
- the function changes more rapidly when more number of knots is used

Natural cubic splines

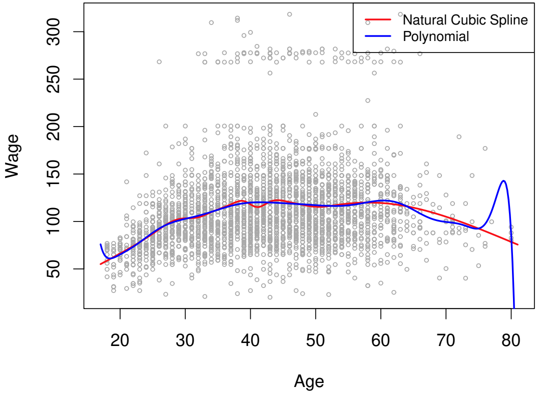
each line is the fitted regression spline with 3 knots to different subset of `Wage` data



- splines can have high variance at the outer range of X
- a natural spline is a regression spline with additional boundary constraints – that is required to be linear at the boundary (two constraints for each endpoint)
- natural cubic spline have $K + 4 - 4 = K$ basis functions

Comparison to polynomial regression

a natural cubic spline with 15 df versus polynomial of degree 15



the polynomial wiggles abruptly at the boundary, while natural spline still provides a reasonable fit

Smoothing splines

among all functions $f(x)$ with two continuous derivatives, find one that minimizes

$$\text{RSS}(\lambda) = \sum_{i=1}^{N} \left[(y_i - f(x_i))^2\right] + \lambda \int |f''(t)|^2 dt$$

- $f''(t)$ measures how the slope of f is changing (the larger, the more wiggly f is)
- $\int |f''(t)|^2 dt$ is the total change in f', indicating smoothness of f
- λ is a fixed smoothing parameter
- the penalized RSS is a trade-off between the goodness of fit and the curvature of f
- when $\lambda = 0$, f can be any function that interpolates the data
- when $\lambda \to \infty$, f is close to the simple least-squares line fit

Solution of penalized RSS

it can be shown that the solution to minimization on page 13 has properties:

- a piecewise cubic polynomial with knots at every unique values of $x_1, \ldots, x_N$
- it has continuous first and second derivatives at each knot
- it is linear in the region outside of the extreme knots

(exercise 5.7 in ESL book)

conclusion:

- f that minimizes the penalized RSS is, in fact, a natural cubic spline with knots at $x_1, \ldots, x_N$
- however, it is a shrunken version of a natural cubic spline (not the same one we would obtain from the basis function approach)

Example

let $h_j(x)$ for $j = 1, \ldots, N$ be basis functions of a natural cubic spline

$$y = f(x) = \sum_{j=1}^{N} \beta_j h_j(x)$$

$$H \in \mathbf{R}^{N\times N},\ H_{ij} = h_j(x_i), \quad G \in \mathbf{R}^{N\times N},\ G_{jk} = \int h_j''(t) h_k''(t) dt$$

the penalized RSS can be represented as

$$\mathrm{RSS}(\lambda) = (y - H\beta)^T (y - H\beta) + \lambda \beta^T G \beta$$

the minimizer β can be seen as a generalized ridge solution

$$\hat{\beta} = (H^T H + \lambda G)^{-1} H^T y$$

however, the computation part of smoothing spline is done more efficiently via B-spline basis representation – further read in ESL book

Smoother matrix

we can represent $\hat{f}$ as a smoothing operation on y

$$\hat{y} = \hat{f}(x) = H(H^T H + \lambda G)^{-1} H^T y \quad \triangleq \quad S_\lambda y$$

we call S_λ as smoother matrix (which depends only x and λ) with properties:

- symmetric and positive semidefinite with $\mathbf{rank}(S_\lambda) = N$
- $S_\lambda S_\lambda \preceq S_\lambda$ (a meaning of shrinking nature)

we define the effective degrees of freedom of a smoothing spline to be

$$\mathrm{df}(\lambda) = \mathbf{tr}(S_\lambda) = \sum_{i=1}^{N} (S_\lambda)_{ii}$$

generally speaking, it gives a sum of (diagonal) weights from each y_i to $\hat{y}$

larger λ gives smaller effective df – the resulting model is simpler

Choosing a smoothing parameter

one approach is to find λ that makes cross-validated RSS small

LOOCV cross-validation error

$$\text{RSS}_{\text{loocv}}(\lambda) = \sum_{i=1}^{N} (y_i - \hat{f}_\lambda^{(-i)}(x_i))^2$$

$\hat{f}_\lambda^{(-i)}$ is the fitted λ-smoothing spline trained on all observations except ith sample

it can be shown that the CV error can be computed *efficiently* by the formula

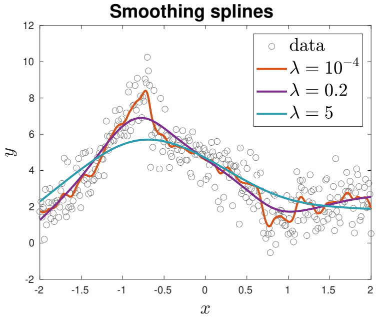
$$\text{RSS}_{\text{loocv}}(\lambda) = \sum_{i=1}^{N} \left[\frac{y_i - \hat{f}_\lambda(x_i)}{1 - (S_\lambda)_{ii}} \right]^2$$

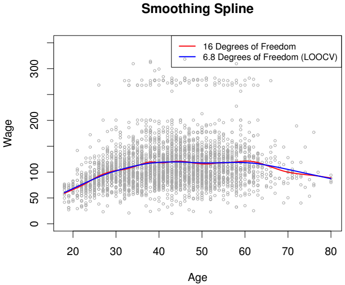
$\hat{f}_\lambda$ is the fitted λ-smoothing spline trained on all observations

benefit: can compute LOOCV error using only the original fit to all data

Example: smoothing spline fit

smoothing spline fit to (left) simulated (right) `Wage` data





- (left) MATLAB: check `fit` with `smoothingspline` option
- (right) λ was choosen by LOOCV, which resulted in 6.8 effective df
- little difference between two splines – a simpler model is preferred

Reinsch form of smoother matrix

it can be shown that the smoother matrix can be presented as the Reinsch form

$$S_\lambda = (I + \lambda K)^{-1}$$

where K does not depend on λ and known as penalty matrix

(use SVD of $H = \tilde{U}\Sigma V^T$ to show that, in fact, $K = \tilde{U}^T \Sigma^{-1} V^T G V \Sigma^{-1} \tilde{U}$)

fact: K is symmetric and admits $K = UDU^T$ with $d_1 = d_2 = 0$

this gives the eigenvalue decomposition of S_λ as

$$S_\lambda = \sum_{k=1}^{N} \rho_k(\lambda) u_k u_k^T, \quad \rho_k(\lambda) = \frac{1}{1+\lambda d_k} \quad \Rightarrow \quad S_\lambda y = \sum_{k=1}^{N} \left(\frac{u_k^T y}{1 + \lambda d_k} \right) \cdot u_k$$

smoothing splines operate by projecting y onto the basis u_k and shrink the kth contribution with weight $1/(1+\lambda d_k)$

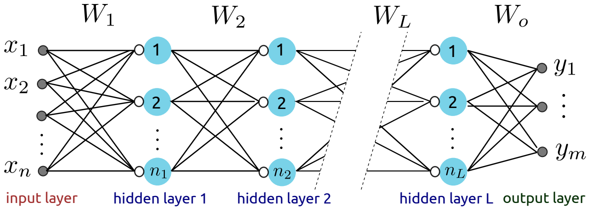
Generalized additive models

Feedforward neural network

- structure and parameters
- mathematical relations
- loss functions for regression and classification

Feedforward NN structure

fully connected L-hidden layers; each of which has n_i units and the weight matrix W_i



- $x = (x_1, x_2, \ldots, x_p)$ is the input (assume the first element is constant)
- $y = (y_1, y_2, \ldots, y_m)$ is the output (or target)
- hidden-layer weight matrices: $W_1 \in \mathbf{R}^{n_1 \times p}$ and $W_j \in \mathbf{R}^{n_j \times n_{j-1}}$, $j = 2, \ldots, L$
- output-layer weight: $W_0 \in \mathbf{R}^{m \times (n_L + 1)}$
- $h : \mathbf{R}^d \to \mathbf{R}^d$ is an activation function for units in hidden layer
- $g : \mathbf{R}^m \to \mathbf{R}^m$ is a transformation for output layer

Compact mathematical representations

linear transform of input and pass through a nonlinear activation function

- $(W_k)_{ij}$ is the weight of the kth layer that maps input i to output j (assume $x_1 = 1$, so $(W_k)_{i1}$ is a bias term)
- the functions h and g are element-wise operations
- activation function examples: step (heaviside), sigmoid, ReLU, tanh, RBF
- example: single hidden-layer of n units; tanh activation:

$$h(W_1 x) = \begin{bmatrix} \tanh[(W_1)_{11} \cdot 1 + (W_1)_{12} x_2 + \cdots (W_1)_{1p} x_p] \\ \tanh[(W_1)_{21} \cdot 1 + (W_1)_{22} x_2 + \cdots (W_1)_{2p} x_p] \\ \vdots \\ \tanh[(W_1)_{n1} \cdot 1 + (W_1)_{n2} x_2 + \cdots (W_1)_{np} x_p] \end{bmatrix} \triangleq \begin{bmatrix} h_1 \\ h_2 \\ \vdots \\ h_n \end{bmatrix}$$

$$z = (W_o)_0 \cdot 1 + (W_o)_1 h_1 + (W_o)_2 h_2 + \cdots + (W_o)_n h_n \in \mathbf{R}^m$$

Task of NN

the transformation of output unit depends on the task of NN

- **regression:** g is linear; $y = z = W_o h(W_1 x)$
- **multi-class classification:** g is softmax function: $g_k(z) = \frac{e^{z_k}}{\sum_{i=1}^m e^{z_k}}$, $k = 1, \ldots, m$

$$y = g(z) = g(W_o(h(W_1 x)))$$

  (y_k is the probability of classifying the input to class k)
- **binary classification:** y has a single node; g reduces to the sigmoid function

Feedforward NN as composites of nonlinear functions

example of L hidden-layer: $y = g(W_o h(W_L h(W_{L-1} h(\cdots h(W_1 x)))))$

to differentiate the notation of NN output from the true description y, we often use

$$\hat{y} = f(x; \Theta)$$

as the output of NN

- conceptually, a nonlinear function of x, parametrized by $\Theta = (W_1, \ldots, W_L, W_o)$
- nonlinearity of a model is introduced via a choice of activation function
- the overall number of parameters is specified by the **depth** (number of hidden layers) and number of **units**

Regression task of NN

let $\hat{y} = f(x; \Theta)$ be the output of neural network using input data x

$\{x_i, y_i\}_{i=1}^N$ are N-sample of input/output data; $\hat{y}_i$ is a model output from sample i

regression: loss functions that are tied with the regression task

- MSE: $(1/N)\sum_{i=1}^N \|y_i - \hat{y}_i\|_2^2$
- MAE: $(1/N)\sum_{i=1}^N \|y_i - \hat{y}_i\|_1$
- huber: $(1/N)\sum_{i=1}^N \text{huber}(r_i)$ where $r_i = y - \hat{y}_i$;

$$\text{huber}(x) = \begin{cases} (1/2)x^2, & |x| \leq M \\ M(|x| - M/2), & x > M \end{cases}$$

Interpretation of MSE

in regression task, the output are linear units

- we can model the output to be an estimate of the mean of a conditional **Gaussian** distribution

$$f(y|x) = \mathcal{N}(y; \hat{y}(x; \Theta), I)$$

- the log likelihood of Gaussian distribution is a negative quadratic function (in y)
- using the maximum likelihood estimation (MLE), it is known that the problem is equivalent to minimizing the MSE

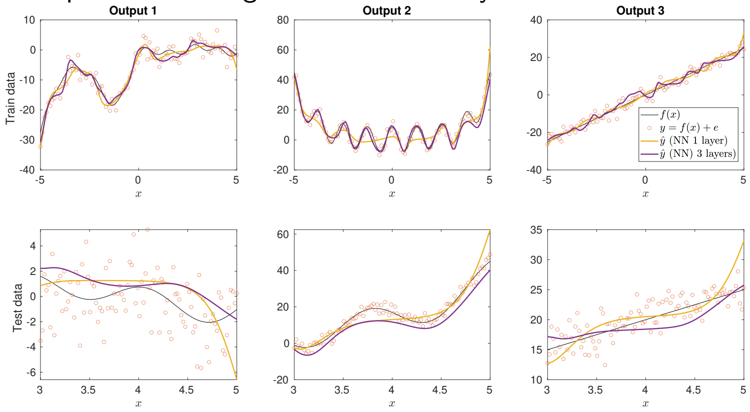
$$\sum_{i=1}^{N} \|y_i - \hat{f}(x_i; \Theta)\|_2^2$$

Example: function approximation

data are generated from nonlinear functions: $y = f(x) + e$, 100 samples, $\sigma^2 = 2$

$$f_1(x) = \sin(5x) - e^{-3\tanh(x)+\cos(x)} + \cos(2x),\ f_2(x) = 8\cos(5x) + 0.5\cosh(x),\ f_3(x) = 5x$$

comparison of using 1 and 3 hidden layers with 10 neurons



- NN can adapt to high fluctuation in y due to nonlinearity of f
- test result shows the models cannot generalize well for y_1 and y_3

Example: solar power forecasting

data: solar irradiance (I), solar power (P), ambient temperature (T) every 15-min

time and station: Jan-Aug, 2022, collected at RAMA IV, max power = 250 kW

target: P and **input:** I, T

consider four experiments with different data arrangement patterns

1. date-time vectors of target and input are delay shifted by 1 hour
2. date-time vectors of target and input are corresponding
3. date-time vectors of target and input are delay shifted by 30 minutes
4. date-time vectors of target and input are delay shifted by 30 minutes and one additional input

$$I_{\mathrm{ema}}(t+1) = \beta I(t) + (1-\beta) I_{\mathrm{ema}}(t), \quad \beta \in [0.8, 1)$$

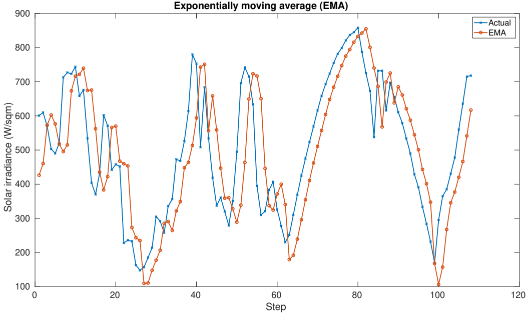
this is an exponentially moving average of $I(t)$

Example: data arrangement

left: target datetime, right: input datetime

```
CASE 1:
    2022-01-01 08:00:00    2022-01-01 07:00:00
    2022-01-01 08:15:00    2022-01-01 07:15:00
    2022-01-01 08:30:00    2022-01-01 07:30:00
    2022-01-01 08:45:00    2022-01-01 07:45:00
    2022-01-01 09:00:00    2022-01-01 08:00:00
CASE 2:
    2022-01-01 08:00:00    2022-01-01 08:00:00
    2022-01-01 08:15:00    2022-01-01 08:15:00
    2022-01-01 08:30:00    2022-01-01 08:30:00
    2022-01-01 08:45:00    2022-01-01 08:45:00
    2022-01-01 09:00:00    2022-01-01 09:00:00
CASE 3:
    2022-01-01 08:00:00    2022-01-01 07:30:00
    2022-01-01 08:15:00    2022-01-01 07:45:00
    2022-01-01 08:30:00    2022-01-01 08:00:00
    2022-01-01 08:45:00    2022-01-01 08:15:00
    2022-01-01 09:00:00    2022-01-01 08:30:00
```

CASE 4: date-time as CASE 3 but with additional input

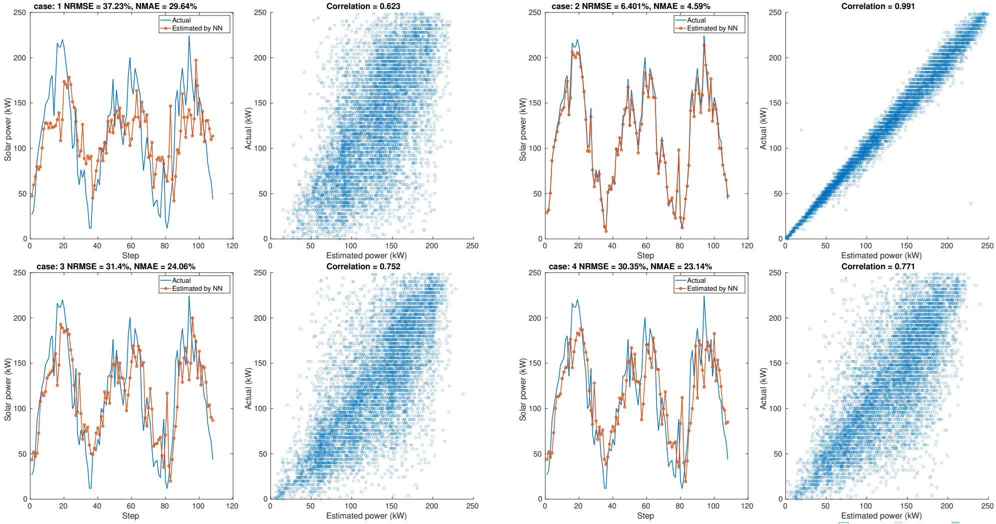


- can you interpret this input-output mapping into a mathematical form ? write $y(t)$ as some function of $x(t)$ or $x(t-1)$? and specify time step of t
- which cases correspond to a practical setting ?

Solar forecast results by NN

test with 3 hidden-layer with 20 neurons

which case would you use ?



Brief summary on data mapping in NN

- a mapping between input and target corresponds to a mathematical representation of model – even we use the same architecture of NN
- when presenting a feedforward NN with lagged inputs $y(t) = f(x(t-1))$, it can represent a form of dynamical model
- arranging data to train a NN should be verified if it is also meaningful when implementing the model in practice
- when using with time series, the concept of 'causal system' should be realized – no output can occur before an input starts

Binary classification task

the output unit predicts the probability of one class

- class labels have two choices: $y \in \{1, -1\}$ or $y \in \{1, 0\}$
- y is modeled to have a Bernoulli distribution: $p(y|x) = \pi^y (1-\pi)^{1-y}$
- the negative loglikelihood is aka cross-entropy:
  -$\log p(y|x) = -[y \log \pi + (1-y)(1-\pi)]$
- modeling: predict $\pi = P(y=1|x)$ using NN (or other models); replace π by $\hat{\pi} = \hat{y}(x;\Theta)$

loss functions used to train NN for binary classification

- cross-entropy: labels are $0, 1$; $\hat{y}_i \triangleq \hat{y}_i(x_i;\Theta) = P(y_i = 1|x_i)$ (classify to class 1)

$$\text{loss} = -\sum_{i=1}^{N} y_i \log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)$$

Binary classification task

- hinge loss (or ReLU, perceptron cost): labels are $1, -1$; normalize $\hat{y}_i$ to $(-1, 1)$

$$\text{loss} = \sum_{i=1}^{N} \max(0, 1 - y_i \cdot \hat{y}_i), \quad \text{(when } \hat{y}_i \neq y_i \text{ the loss is 2)}$$

- scores motivated from F1 or **dice similarity coefficient**

$$\text{F1} = \frac{2\text{TP}}{2\text{TP} + \text{FP} + \text{FN}}, \quad \text{(no TN, predicting majority samples correctly)}$$

meaning: $\text{TP} = \sum_i y_i \hat{y}_i$, $\text{FP} = \sum_i (1 - y_i)\hat{y}_i$, and $\text{FN} = \sum_i y_i (1 - \hat{y}_i)$

- minimizing these losses is similar to maximizing F1 score

$$\text{soft-dice loss} = 1 - \frac{2\sum_{i=1}^{N} y_i \hat{y}_i}{\sum_{i=1}^{N}(y_i + \hat{y}_i)}, \quad \text{squared-dice loss} = 1 - \frac{2\sum_{i=1}^{N} y_i \hat{y}_i}{\sum_{i=1}^{N}(y_i^2 + \hat{y}_i^2)}$$

K-class classification using NN

label y is a standard unit vector in $\mathbf{R}^K$

$$y = (y_1, y_2, \ldots, y_K)$$

(only one of $y_1, y_2, \ldots, y_K$ has value of 1; the rest is all zero)

- denote π_k the probability that $y = (0, 0, \underbrace{1}_{k\text{th}}, 0, \ldots, 0)$ where $\sum_{i=1}^K \pi_i = 1$
- generalize Bernoulli distribution to an K-dimensional binary variable y

$$p(y|x) = \pi_1^{y_1} \pi_2^{y_2} \cdots \pi_K^{y_K}$$

- the (conditional) loglikelihood is called **(multi-class) cross entropy**

$$\log p(y|x) = y_1 \log \pi_1 + y_2 \log \pi_2 + \cdots + y_K \log \pi_K$$

- modeling: NN has K-dimensional output units that predictds π_k's

$$\hat{y}_k = \hat{\pi}_k \approx \pi_k, \quad k = 1, \ldots, K$$

K-class classification using NN

let i be a sample index, $i = 1, \ldots, N$

cross-entropy loss: $\hat{y}_i$ is the output of the **softmax function**

$$\text{loss} = -\sum_{i=1}^{N} y_{i1}\log(\hat{y}_{i1}) + y_{i2}\log(\hat{y}_{i2}) + \cdots + y_{iK}\log(\hat{y}_{iK})$$

$$= -\sum_{i=1}^{N} \log(\hat{y}_{i,\text{correct class}}) = -\sum_{i=1}^{N} \log\left(\frac{e^{z_{i,\text{correct class}}}}{\sum_{k=1}^{K} e^{z_{ik}}}\right)$$

$z_i \in \mathbf{R}^K$ is predicted output from a model; before being mapped to probabilities

(alo referred to multi-class softmax cost, softplus cost, multi-class cross entropy loss)

Considerations in learning NN

- hidden units: properties and recent choices of activation functions (leaky/parametric ReLU, softplus, etc.)
- architecture design: determine overall structure of the network (theoretical result: **universal approximation theorem**
- recent advances in proposing new choices of **loss functions**
- model training
  - gradient-based learning requires computing derivatives of the composition: concept of **backprogation** based on chain rule in calculus
  - how a learning algorithm in optimization process affects a model capacity (which are the **effective capacity**, and representational capacity; the latter defined by the family of model)
  - computation: automatic differentiation, justification of non-differentiability of some activation functions by numerical point of view, batch/**mini-batch optim**
- regularization: ℓ_1 and ℓ_2, dropout

References

most figures in spline examples are taken from

1. Chapter 7 in G.James, D. Witten, T. Hastie, and R. Tibshirani, *An Introduction to Statistical Learning: with Applications in R*, Springer, 2015
2. Chapter 5 in T. Hastie, R. Tibshirani and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference and Prediction*, 2nd edition, Springer, 2009

further reading on neural networks:

3. Ian Goodfellow, Yoshua Bengio, and Aaron Courville, *Deep Learning*, The MIT Press, 2016