Fundamentals of Machine Learning (Part II)

Mohammad Emtiyaz Khan
AIP (RIKEN), Tokyo

http://emtiyaz.github.io

emtiyaz.khan@riken.jp

April 12, 2018

Goals

Learn a few methods for classification and understand why they might work and sometimes fail. [1].

1. Classification.
2. K-Nearest Neighbors (k-NN).
3. Logistic Regression.
4. Deep Neural Networks.

[1]Some figures are taken from Hastie, Tibshirani, and Friedman's book on statistical learning. Some from Chris Bishop's Machine learning book and one from Kevin Murphy's book. The CNN picture taken from http://www.mdpi.com/1099-4300/19/6/242

1 Classification

Similar to regression, classification relates input variables $\mathbf{x}$ to the output variable y, but now y can take only discrete values, i.e. y is a categorical (or nominal) variable.

Binary classification

When y can only take two discrete values, it is called binary classification. We will denote these values as $y \in \{\mathcal{C}_1, \mathcal{C}_2\}$. These values are also called class labels or simply classes. Other common notations are $y \in \{-1, +1\}$ or $y \in \{0, 1\}$, although there may not necessarily be any ordering between the two classes.

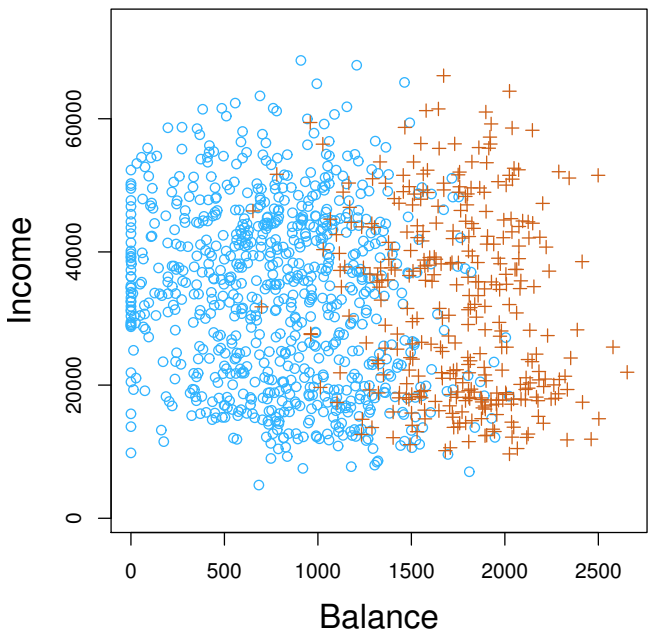
Multi-class classification

In a multi-class classification, y can take multiple discrete values i.e. $y \in \{\mathcal{C}_0, \mathcal{C}_1, \ldots, \mathcal{C}_{K-1}\}$ for a K-class problem. Again, there is no notion of ordering among these classes, but we may ignore this fact and may sometimes use the following notation for convenience: $y \in \{0, 1, 2, \ldots, K-1\}$.

Examples of classification problems

An online banking service must be able to determine whether or not a transaction being performed on the site is fraudulent, on the basis of the users IP address, past transaction history, and so forth.

An example is shown below on the *default* dataset. The annual incomes and monthly credit card balances of a number of individuals. The individuals who defaulted on their credit card payments are shown in orange, and those who did not are shown in blue.



A person arrives at the emergency room with a set of symptoms that could possibly be attributed to one of the two medical conditions. Which of the two conditions does the individual have?

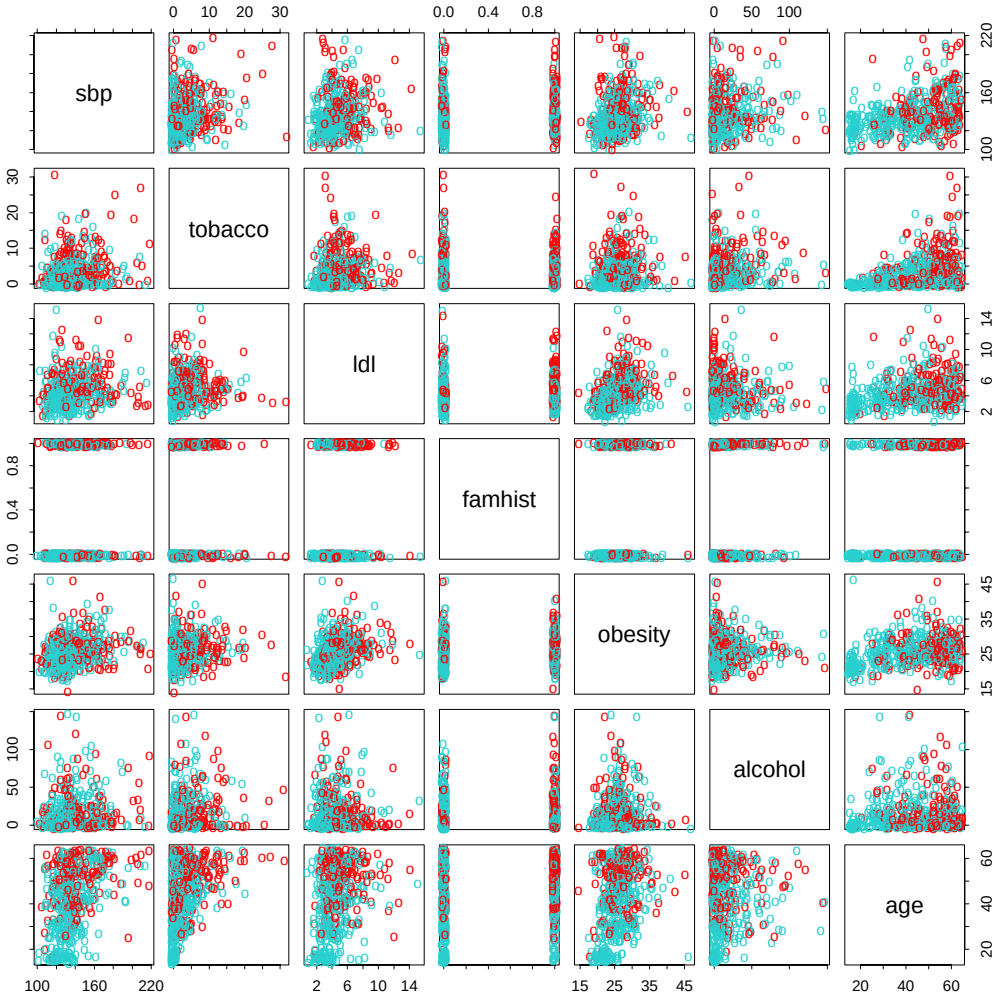


FIGURE 4.12. *A scatterplot matrix of the South African heart disease data. Each plot shows a pair of risk factors, and the cases and controls are color coded (red is a case). The variable* family history of heart disease (`famhist`) *is binary (yes or no).*

Classifier

A classifier will divide the input space into a collection of regions belonging to each class. The boundaries of these regions are calld decision boundaries. A classifier can be linear or nonlinear.

Elements of Statistical Learning (2nd Ed.) ©Hastie, Tibshirani & Friedman 2009 Chap 4

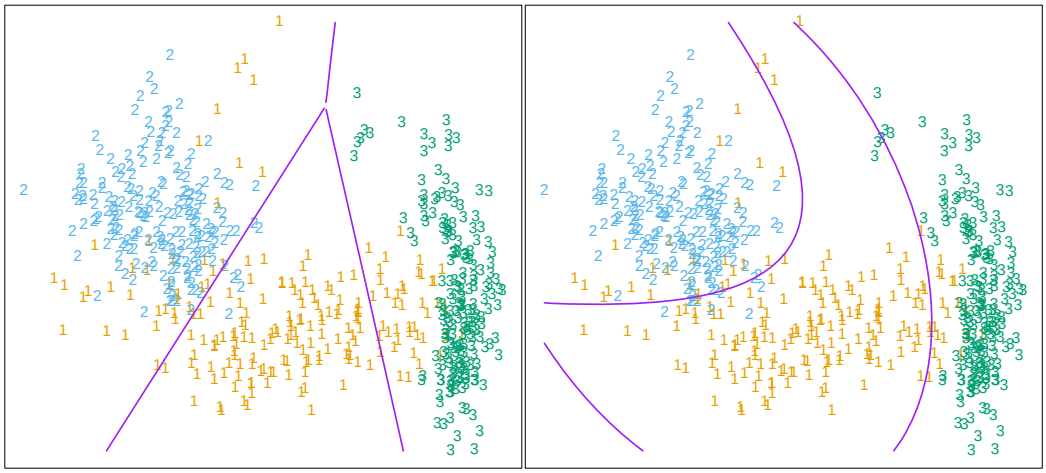
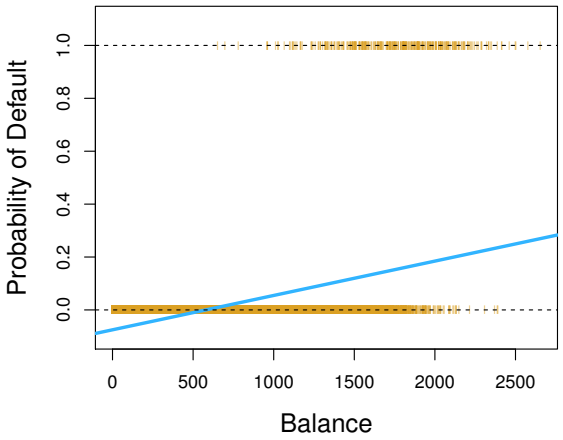


FIGURE 4.1. *The left plot shows some data from three classes, with linear decision boundaries found by linear discriminant analysis. The right plot shows quadratic decision boundaries. These were obtained by finding linear boundaries in the five-dimensional space* $X_1, X_2, X_1X_2, X_1^2, X_2^2$. *Linear inequalities in this space are quadratic inequalities in the original space.*

2 K-Nearest Neighbors (k-NN)

Classification with linear regression

We can use $y = 0$ for $\mathcal{C}_1$ and $y = 1$ for $\mathcal{C}_2$ (or vice-versa), and simply use least-squares to predict $\hat{y}_*$ given $\mathbf{x}_*$. We can predict $\mathcal{C}_1$ when $\hat{y}_* < 0.5$ and $\mathcal{C}_2$ when $\hat{y}_* > 0.5$.



Any problems with this approach?

FIGURE 2.1. *A classification example in two dimensions. The classes are coded as a binary variable (BLUE = 0, ORANGE = 1), and then fit by linear regression. The line is the decision boundary defined by $x^T\hat{\beta} = 0.5$. The orange shaded region denotes that part of input space classified as ORANGE, while the blue region is classified as BLUE.*

k-Nearest Neighbor (k-NN)

The k-NN prediction for an $\mathbf{x}_*$ is,

$$f_k(\mathbf{x}_*) = \frac{1}{k} \sum_{\mathbf{x}_n \in nbh_k(\mathbf{x}_*)} y_n \ ,$$

where $nbh_k(\mathbf{x})$ is the neighborhood of $\mathbf{x}$ defined by the k closest points $\mathbf{x}_n$ in the training data.

We show results for $k = 1$ and $k = 15$ respectively.

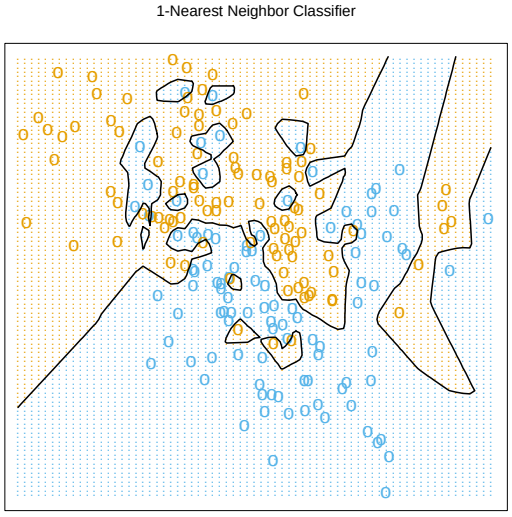
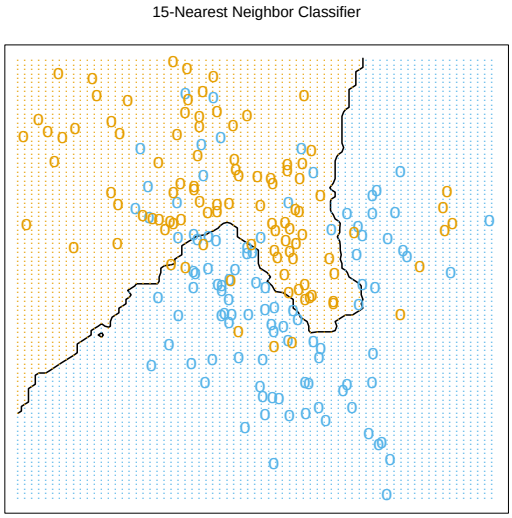


FIGURE 2.3. *The same classification example in two dimensions as in Figure 2.1. The classes are coded as a binary variable* (BLUE $= 0$, ORANGE $= 1$)*, and then predicted by* 1*-nearest-neighbor classification.*



Bias-variance revisited

How should train and test error vary with k?

Curse of dimensionality

According to Pedro Domingos: "Intuitions fail in high dimensions". This is also known as the curse of dimensionality (Bellman, 1961).

Claim 1: "Generalizing correctly becomes exponentially harder as the dimensionality grows because fixed-size training sets cover a dwindling fraction of the input space."

The expected edge length is $e_D(r) = r^{1/D}$, e.g.

$$e_{10}(0.01) = 0.63\,,\; e_{10}(0.1) = 0.80$$

i.e. to capture 1% or 10% of the data, we must cover 63% or 80% of the range of each input variable.

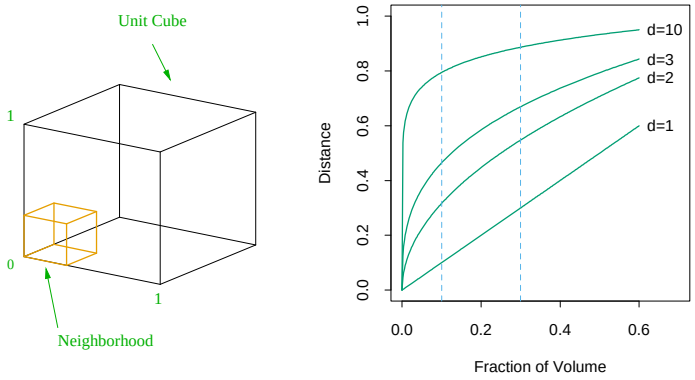


FIGURE 2.6. *The curse of dimensionality is well illustrated by a subcubical neighborhood for uniform data in a unit cube. The figure on the right shows the side-length of the subcube needed to capture a fraction r of the volume of the data, for different dimensions p. In ten dimensions we need to cover* 80% *of the range of each coordinate to capture* 10% *of the data.*

As a result, the sampling density is proportional to $N^{1/D}$, i.e. if $N_1 = 100$ is the sample size for a 1-input problem, then $N_{10} = 100^{10}$ is required for the same sampling density with 10 inputs.

Claim 2: In high-dimension, data-points are far from each other. Consequently, "as the dimensionality increases, the choice of nearest neighbor becomes effectively random."

Consider N data points uniformly distributed in a D-dimensional unit ball centered at the origin. We consider a nearest-neighbor estimate at the origin. The median distance from the origin to the closest data point is,

$$\left(1 - \tfrac{1}{2}^{1/N}\right)^{1/D}$$

For $N = 500, D = 10$, this number is 0.52, more than halfway to the boundary.

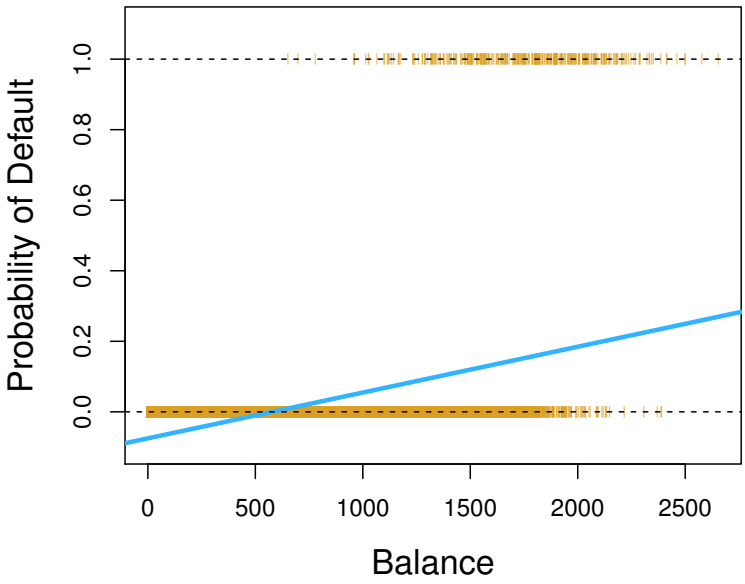
Discussion

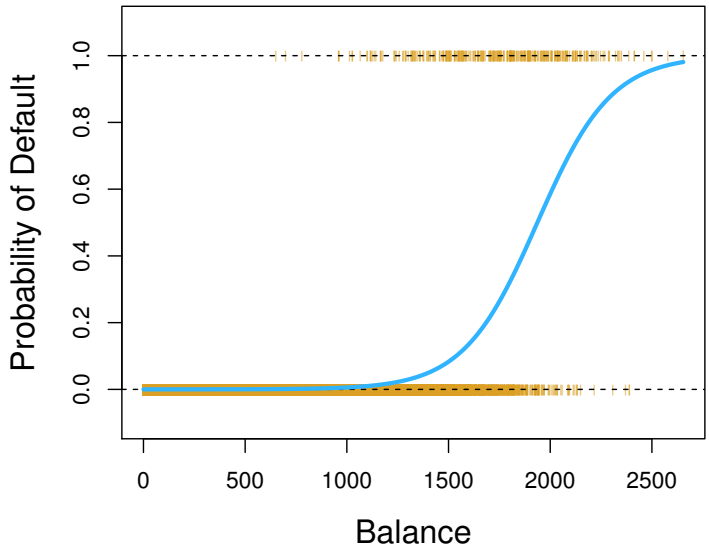
(Taken from HTF). We will see (in the next few lectures) that there is a whole spectrum of models between the rigid linear models and the extremely flexible 1-NN model. Each model comes with their own assumptions and biases.

(Based on Domingos). You might think that gathering more input variables never hurts, since at the worst they provide no new information about the output. But in fact their benefits may be outweighed by the curse of dimensionality.

3 Logistic Regression

Revisting Classification with linear Reg





Logistic regression

We need to model $p(y = \mathcal{C}_1|\mathbf{x})$ and $p(y = \mathcal{C}_2|\mathbf{x})$ such that they both are > 0 and also sum to 1. For a new input $\mathbf{x}_*$, we can classify to $\mathcal{C}_1$ when $p(\hat{y}_*|\mathbf{x}_*) < 0.5$.

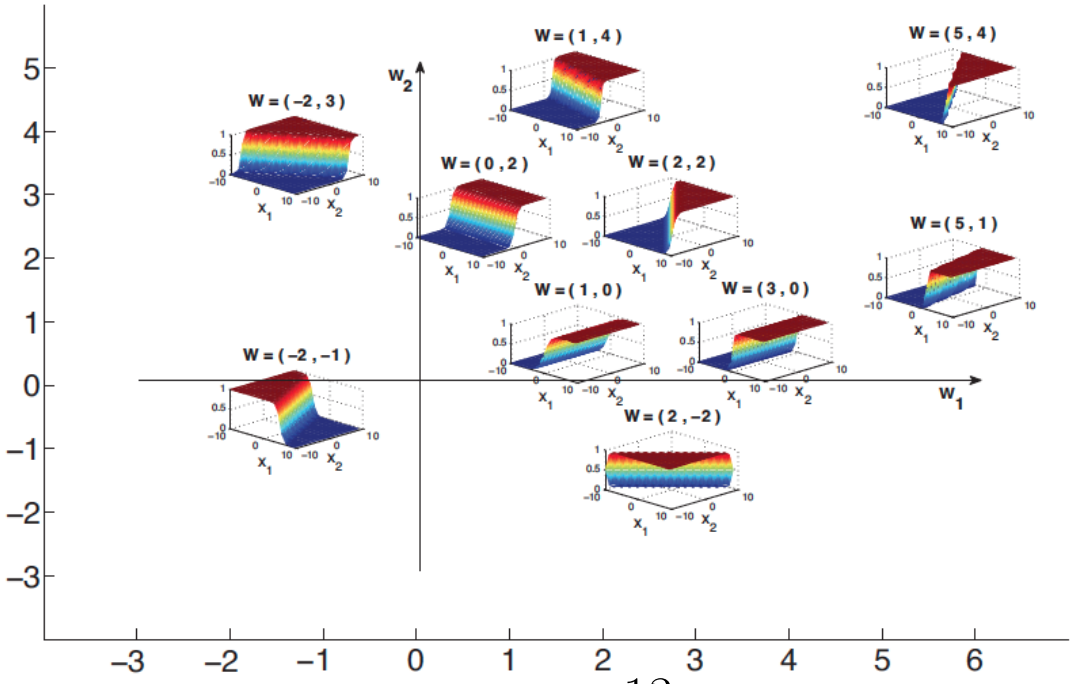
We will use the logistic function.

$$\sigma(x) = \frac{\exp(x)}{1 + \exp(x)}, \qquad 1 - \sigma(x) = \frac{1}{1 + \exp(x)}$$

We pass the linear-regression model $\eta_n = \widetilde{\mathbf{x}}^T \boldsymbol{\beta}$ through the logistic function to get the probabilities.

$$p(y_n = \mathcal{C}_1|\mathbf{x}_n) = \sigma(\eta_n), \qquad p(y_n = \mathcal{C}_2|\mathbf{x}_n) = 1 - \sigma(\eta_n)$$

This figure visualizes the probabilities obtained for a 2-D problem (taken from KPM Chapter 7).



The probabilistic model

Assuming that each y_n is independent of others, we can define the probability of $\mathbf{y}$ given $\mathbf{X}$ and $\boldsymbol{\beta}$:

$$p(\mathbf{y}|\mathbf{X}, \boldsymbol{\beta}) = \prod_{n=1}^{N} p(y_n|\mathbf{x}_n)$$
$$= \prod_{n:y_n=\mathcal{C}_1} p(y_n = \mathcal{C}_1|\mathbf{x}_n) \prod_{n:y_n=\mathcal{C}_2} p(y_n = \mathcal{C}_2|\mathbf{x}_n)$$

A better way to write this is to use the coding $y_n \in \{0, 1\}$.

$$p(\mathbf{y}|\mathbf{X}, \boldsymbol{\beta}) = \prod_{n=1}^{N} \sigma(\eta_n)^{y_n}[1 - \sigma(\eta_n)]^{1-y_n}$$

The log-likelihood is given as follows:

$$\begin{aligned}\mathcal{L}_{mle}(\boldsymbol{\beta}) &= \log p(\mathbf{y}|\mathbf{X}, \boldsymbol{\beta}) \\ &= \sum_{n=1}^{N} y_n \log \sigma(\widetilde{\mathbf{x}}_n^T \boldsymbol{\beta}) + (1 - y_n) \log[1 - \sigma(\widetilde{\mathbf{x}}_n^T \boldsymbol{\beta})] \\ &= \sum_{n=1}^{N} y_n \widetilde{\mathbf{x}}_n^T \boldsymbol{\beta} - \log[1 + \exp(\widetilde{\mathbf{x}}_n^T \boldsymbol{\beta})]\end{aligned}$$

Optimization using SGD

We will use the following fact to derive the gradient.

$$\frac{\partial}{\partial x}\log[1+\exp(x)] = \sigma(x)$$

Taking the gradient of the log-likelihood, we get the following stochastic gradient estimate:

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\beta}} \approx N\left[\sigma(\widetilde{\mathbf{x}}_i^T\boldsymbol{\beta}) - y_i\right]\widetilde{\mathbf{x}}_i$$

This is very similar to SGD for linear regression, i.e., the gradient is in the direction of $\widetilde{\mathbf{x}}_i$ with magnitude equal to prediction error for the i'th example.

Penalized Logistic Regression

The cost-function can be unbounded when the data is linearly separable.

To get a well-defined problem, we will regularize.

$$\min_{\beta} -\sum_{n=1}^{N}\log p(y_n|\mathbf{x}_n^T,\boldsymbol{\beta}) + \lambda\sum_{d=1}^{D}\beta_d^2$$

4 Deep Neural Networks

Multi-Layer Perceptron (MLP)

Also known as feed-forward neural network,

$$\mathbf{x}_n \to \mathbf{a}_n^{(1)} \to \mathbf{z}_n^{(1)} \to \mathbf{a}_n^{(2)} \to \mathbf{z}_n^{(2)} \to \ldots \to \mathbf{z}_n^{(K-1)} \to \mathbf{y}_n$$

where $\{y_n, \mathbf{x}_n\}$ is the n'th input-output pair, $\mathbf{z}_n^{(k)}$ is the k'th hidden vector, $\mathbf{a}_n^{(k)}$ is the corresponding activation. There are a total of K layers.

For the k'th layer, we obtain the m'th activation $a_{mn}^{(k)}$ and the corresponding hidden variable $z_{mn}^{(k)}$, as shown below:

$$a_{mn}^{(k)} = \left(\boldsymbol{\beta}_m^{(k)}\right)^T \mathbf{z}_n^{(k-1)}, \quad z_{mn}^{(k)} = h\left(a_{mn}^{(k)}\right)$$

where $\mathbf{z}_n^{(k-1)}$ is the hidden vector for the previous layer. For the first layer, we set $\mathbf{z}_n^{(0)} = \mathbf{x}_n$. For the last layer, we use a link function to map $\mathbf{z}_n^{(K-1)}$ to the output $\mathbf{y}_n$.

Note that a 1-Layer MLP is simply a generalization of linear/logistic regression.

Defining $\mathbf{B}^{(k)}$ as a matrix with rows $(\boldsymbol{\beta}_m^{(k)})^T$, we can express the computation of activation and hidden vectors as follows:

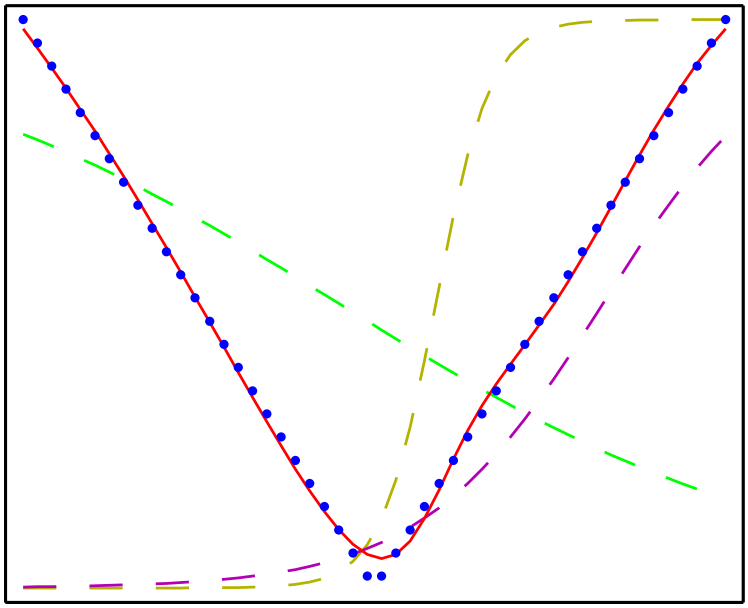
$$\mathbf{a}_n^{(k)} = \mathbf{B}^{(k)}\mathbf{z}_n^{(k-1)}, \quad \mathbf{z}_n^{(k)} = h\left(\mathbf{a}_n^{(k)}\right)$$

In a more compact notation, we can express the input-output relationship as follows:

$$\hat{y}_n = g((\boldsymbol{\beta}^{(K-1)})^T * h(\mathbf{B}^{(K-2)} * h(* \ldots * h(\mathbf{B}^{(1)} * \mathbf{x}_n))),$$

where g is an appropriate link function to match the output.

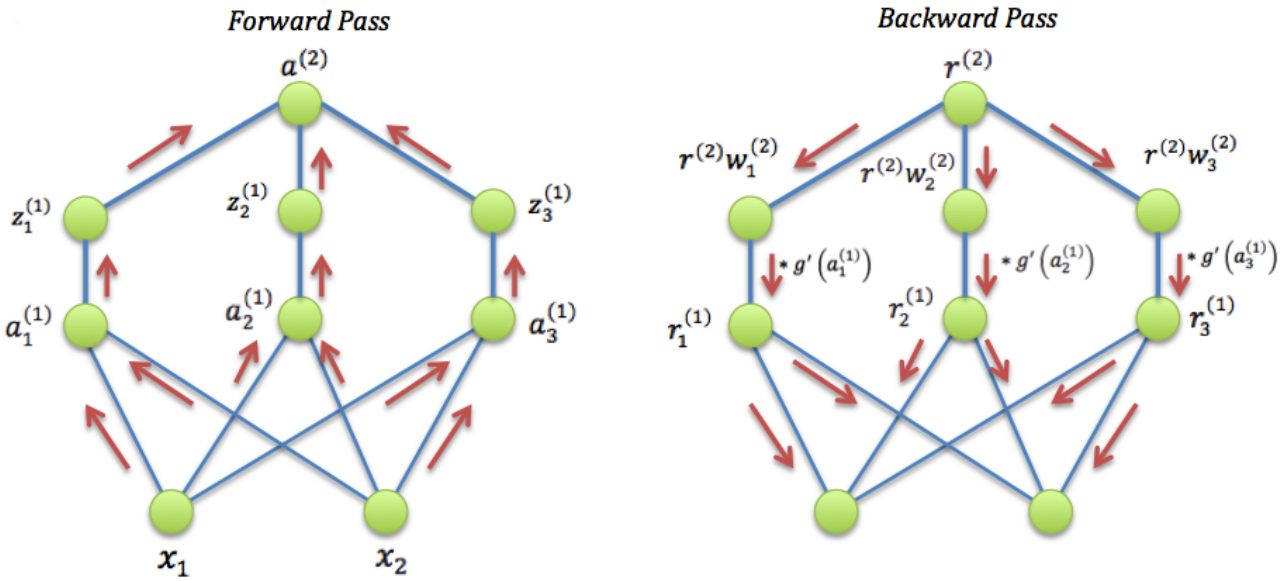
An illustration below shows reconstruction of the function $|x|$ at $N = 50$ data points sampled at the blue dots. The trained network has 2 layers and 3-hidden variables with $tanh()$ activation function.



Optimization and Back-propagation

We can learn parameters $\mathbf{B}$ using stochastic gradient-descent.

Gradient computation can be complicated due to the *deep* structure of the network. We can use back-propagation to simplify the computation. The key-idea is to express the derivatives in terms of activations $\mathbf{a}_n^{(k)}$ and hidden variables $\mathbf{z}_n^{(k)}$ using the chain rule. Below is the outline of the algorithm:



Step 1: Compute $\mathbf{a}_n^{(k)}$ and $\mathbf{z}_n^{(k)}$ using forward propagation.

Step 2: Compute $\boldsymbol{\delta}_n^{(k)} := \partial \mathcal{L} / \partial \mathbf{a}_n^{(k)}$ using backward propagation:

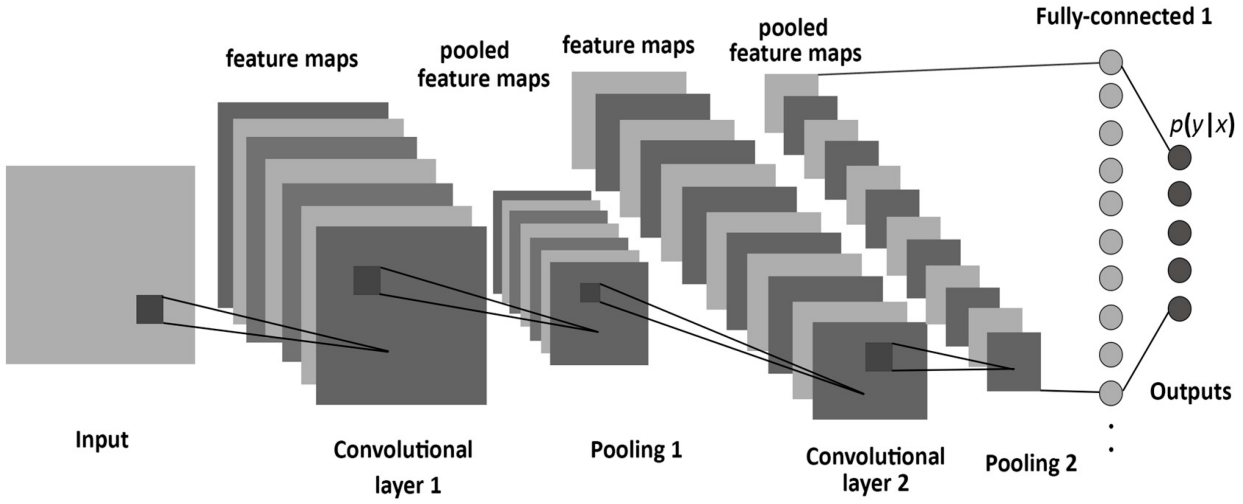
$$\boldsymbol{\delta}_n^{(k-1)} = \text{diag}\left[\mathbf{h}'(\mathbf{a}_n^{(k)}\right] \left(\mathbf{B}^{(k)}\right)^T \boldsymbol{\delta}_n^{(k)}$$

Step 3: Compute $\partial \mathcal{L} / \partial \mathbf{B}^{(k)}$ using the above derivatives.

$$\frac{\partial \mathcal{L}}{\partial \mathbf{B}^{(k)}} = \sum_n \boldsymbol{\delta}_n^{(k)} \left(\mathbf{z}_n^{(k)}\right)^T$$

Convolutional Neural Network (CNN)

CNN produce the state-of-the-art results for image classification.



Tricks for Deep Learning

Obtaining a good generalization error with neural networks and avoiding overfitting requires a lot of hacks and tricks. A good summary of these are given in Bottou's paper "Stochastic gradient tricks". In addition, initialization seems to play a huge role in improving the performance. See the following paper "On the importance of initialization and momentum in deep learning" by Ilya Sutskever et. al.

Modern stochastic optimization methods, such as RMSprop and Adam, are currently heavily used by practitioners to optimizer neural networks.