Nonlinear models

Introduction

Recall that a function f is linear if $f(\alpha a + \beta b) = \alpha f(a) + \beta f(b)$. When is a model linear? More to the point, when can it be trained with linear methods?

Nonlinear transformations

Recall the objective of the optimisation process in the algorithms so far: find a parameter vector w that minimises $E_{in}(X, w, Y)$ (where X and Y are the training dataset inputs and outputs, respectively). So now the function in question (the one we want to minimise) is (on a given dataset) a function of w. Hence, if it is a linear function in w, linear/logistic regression, PLA can be used.

Take, for example, the following polynomial model: $f(x, w) = \sum_{i=0}^{k} w_i x^k$ ($x \in \mathbb{R}$, $w \in \mathbb{R}^{k+1}$). It is linear in the parameters, so we can use linear regression on it. If we transform the sample input x into a vector $\phi = [1, x, x^2, \dots, x^k]^T$, the model becomes the familiar $f(x, w) = w^T \phi$. This method is called *polynomial regression*.

In general, the transformed input, ϕ, is called the *feature vector*.

Neural networks

We don't actually need to use linear methods for training: gradient descent can optimise nonlinear functions as well[1].

The only hurdle is to calculate the gradient of the model (with respect to the parameters). Luckily, this can be automated using the chain rule of derivation and some tricks: search for the *backpropagation* algorithm. This is usually built into any respectable neural network library, so in Task 2, you do not actually have to code it from scratch (for a change).

[1] OK, it may not converge to a global mininum because of nonconvexity, and it may have problems with saddle points or large sets with zero gradients, but it generally works all right.

Building a feed-forward neural network is most simply done by composing layers: linear layers (matrix product with bias), different activation layers (sigmoid, softmax, ReLU, etc.), and others. Look at the Keras documentation to see more. If you have, eg. a network with a linear layer L_1, an activation layer L_2, another linear layer L_3 and a softmax layer L_4, then the output is calculated as $\hat{y} = L_4(L_3(L_2(L_1(x))))$. The width of a layer is simply the dimension of its output.

There is a special type of linear layer, called a *convolutional layer*, which is usually used for images. It exploits the space-invariance of features in images: if you have an eye in the image, it doesn't really matter where it is, it will look more or less the same. It works by applying discrete convolution to its input with a small weight matrix (in neural networks, parameters are sometimes called weights). This is still a linear operation, since convolution is linear.