Lecture 7: Multilayer Perceptron (MLP) Backpropagation

Multilayer Perceptron

- Modelling non-linearity via **function composition (构成)**
- Linear models cannot solve non-linearly separable problems
    - Possible solution: composition (combine neurons)
- Perceptron: a building block for ANN
- Not restricted to binary classification, there're various **activation functions**:
    - Step function
    - Sign function
    - Logistic function
    - Tanh function
    - Rectifier
    - ...

ANN

- Can be naturally adapted to various supervised learning setups (e.g. univariate/multivariate regression, binary/multivariate classification)
- Capable of approximating plethora non-linear functions
- **Universal approximation theorem:** (就是说hidden layer可以approximate各种continuous function)
    - **an ANN with a hidden layer** with a finite number of units, and mild assumptions on the activation function, can approximate continuous functions on compact subsets of R^n arbitrarily well
- To train your network:
    - Define the loss function and find params that minimise the loss on training data (e.g. use SDG)
- Loss function for ANN:
    - As regression, can use **squared error**

Loss function for ANN training

$$L = \frac{1}{2}(\hat{f}(x, \theta) - y)^2 = \frac{1}{2}(z - y)^2$$

- Training: minimise L w.r.t. θ
    - Fortunately $L(\theta)$ is differentiable
    - Unfortunately no analytic solution in general

SGD for ANN

- Choose initial guess $\theta^{(0)}, k = 0$
    - Here θ is all set of weights from all layers
- For i from 1 to T (epochs)
    - For j from 1 to N (training examples)
        - Consider examples $\{\mathbf{x}_j, y_j\}$

- Update: $\theta^{(i+1)} = \theta^{(i)} - \eta \nabla L(\theta^{(i)})$

Backpropagation (for updating weights)

- Calculate **gradient of loss** of a composition
- Recall that the output z of an ANN is a function composition, and hence $L(z)$ (loss) is also a composition

$$L = 0.5(z-y)^2 = 0.5(h(s)-y)^2 = 0.5(s-y)^2$$

- Backpropagation makes use of this fact by applying the **chain rule** for derivatives (从后往前一层一层differentiate回去)

$$\frac{\partial L}{\partial w_j} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial s}\frac{\partial s}{\partial w_j}$$

$$\frac{\partial L}{\partial v_{ij}} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial s}\frac{\partial s}{\partial u_j}\frac{\partial u_j}{\partial r_j}\frac{\partial r_j}{\partial v_{ij}}$$

- When applying chain rules, we can define intermediate variables (nice and simple, can used as common results)

$$\delta = \frac{\partial L}{\partial s} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial s}$$

$$\epsilon_j = \frac{\partial L}{\partial r_j} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial s}\frac{\partial s}{\partial u_j}\frac{\partial u_j}{\partial r_j}$$

...

We have

1. $\frac{\partial L}{\partial w_j} = \delta u_j = (z-y)u_j$
2. $\frac{\partial L}{\partial v_{ij}} = \epsilon_j x_i = \delta w_j g'(r_j) x_i$

Forward propagation (just compute outputs for each layer, make predictions)

- Use current estimates of v_{ij} and $w_j \rightarrow$ Calculate r_j, u_j, s and z
- 一次 forward 一次 backward

Further notes on ANN

- ANN's are flexible, but flipside is over-parameterisation, hence tendency to **overfitting**
- Starting weights usually random distributed about zero
- Implicit regularisation: **early stopping**
- Explicit regularisation
  - Much like ridge regression
  - With some activation functions this also shrinks the ANN towards a linear model