Mathematical foundation of supervised learning

Oxford ML summer school 2022

Hao Ni
UCL, ATI and DataSig

DataSig

A rough path between
mathematics and data science

University of Oxford
The Alan Turing Institute
Imperial College London
UCL
UKRI Engineering and Physical Sciences Research Council

Objectives of the tutorial

- To introduce a high-level picture of Machine Learning (ML);
- To provide you with a systematic framework of supervised learning;
- To uncover the black-box of deep learning and help you develop the mathematical understanding of neural networks;
- To illustrate how mathematical understanding of supervised learning can help you conduct numerical experiments to solve empirical data challenges.

This tutorial will cover

- An overview of Machine Learning
- Supervised Learning, including the linear regression and neural networks
- The prediction of future price movement on high frequency limit order book data using neural networks.

Outline

Big Data Era

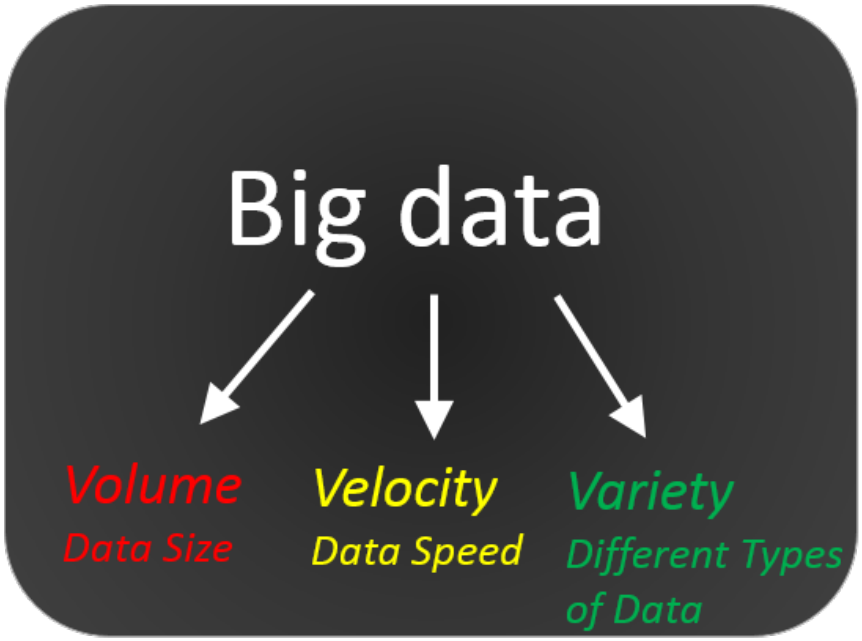


Figure: The Main Characteristics of Big Data and examples of big data.

Example (CASIA-OLHWDB1 Dataset)

1. 4,037 categories (3,866 Chinese characters and 171 symbols)
2. 420 writers and 1,694,741 samples.

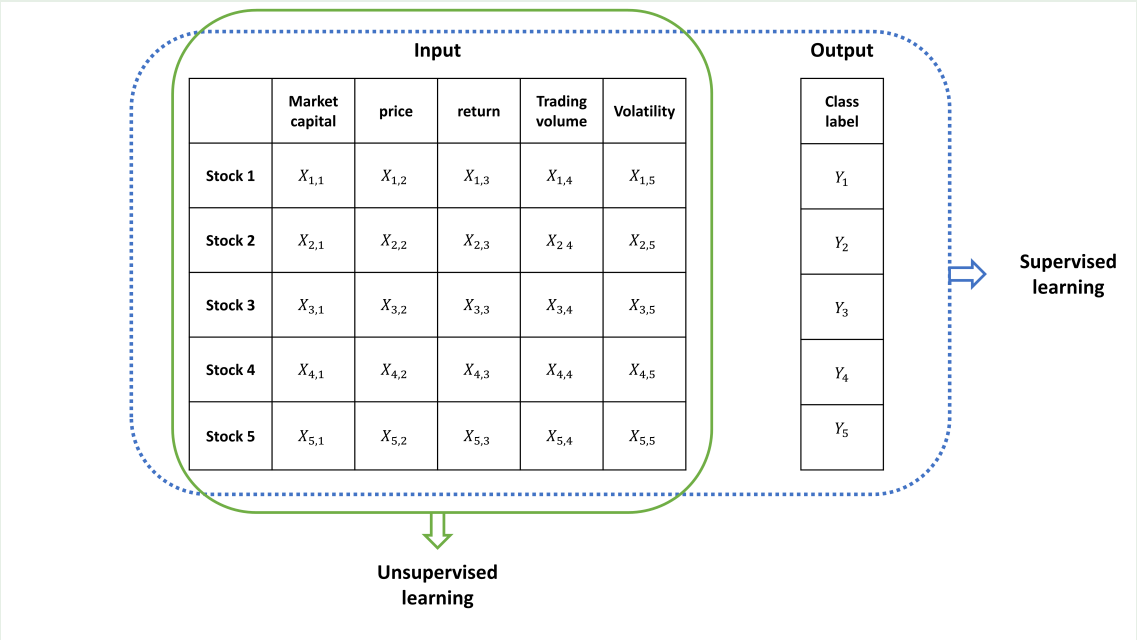
Machine Learning is a field of study to give computer systems the ability to ”learn” with data, without being explicitly programmed.

Machine learning *is ideal for exploiting the opportunities hidden in* ***big data***.

Categories of Machine Learning Tasks

- Supervised Learning
  - Regression (e.g. forecast returns);
  - Classification (e.g. predict the direction of returns).
- Unsupervised Learning
  - Clustering (e.g. identify the most common signs of market stress).
- Reinforcement Learning (e.g. learning trading strategy)

Example (Supervised v.s. Unsupervised Learning)



Opportunities and Challenges of ML

The Recent Achievements of Machine Learning Applications

- Supervised Learning: Image Recognition, Speech Recognition [3], drug discovery (AlphaFold [1]).
- Reinforcement Learning: Atari videogame [4], AlphaGO[7].

Challenges of Deep Learning

- Data greedy and heavy computation cost;
- Difficult to interpret;
- Lack of theoretical understanding.

Outline

Supervised Learning

Supervised learning is the machine learning task of learning a function (f) that maps an input (X) to an output (Y) based on example input-output pairs.

Example (Housing Data Example)

Housing data downloaded via the link have 20640 samples.

Input X:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	8.3252
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	8.3014
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	7.2574
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	5.6431
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0	3.8462

Output Y:

median_house_value
452600.0
358500.0
352100.0
341300.0
342200.0

Problem setup

Regression analysis

Dataset: $\mathscr{D} := \mathscr{D}_{\text{training}} \cup \mathscr{D}_{\text{test}} = \{(x_i, y_i)\}_{i=1}^{N}$ satisfies that $y_i = f(x_i) + \varepsilon_i$, where $x_i := (x_i^{(1)}, \cdots, x_i^{(d)}) \in \mathbb{R}^d$, ε_i is iid with $\mathbb{E}[\varepsilon_i | x_i] = 0$.

Question:

1. For any new input data x^*, what is $\mathbb{E}[\underbrace{y}_{f(x)+\varepsilon} | x = x^*]$, i.e. $f(x^*)$? $\Leftrightarrow$ How to estimate f from $\mathscr{D}_{\text{training}}$?
2. Given the estimator $\hat{f}$, how to quantify the goodness of fitting of $\hat{f}$?

General Framework

Regression Framework

Dataset:	$\mathscr{D} = \{(x_i, y_i)\}_i = \mathscr{D}_{\text{training}} \cup \mathscr{D}_{\text{test}}$
Model:	$y = f_\theta(x) + \varepsilon, \quad \forall x \in \mathbb{R}^d. \ (f_\theta \sim f)$
Empirical Loss:	$L(\theta \vert \mathscr{D}_{\text{training}}) \rightarrow$Minimize
Optimization:	$\hat{\theta} = \arg\min_\theta (L(\theta \vert \mathscr{D}_{\text{training}}))$
Prediction:	$\hat{y}_* = f_{\hat{\theta}}(x_*).$
Validation:	Compute the test metrics on $\mathscr{D}_{\text{test}}$.

For ease of notation, we adopt the matrix form for $\mathscr{D}_{\text{training}} = (X, Y)$, where

$$Y = \begin{pmatrix} y_1 \\ \vdots \\ y_{N_1} \end{pmatrix} \text{ and } X = \begin{pmatrix} x_1^{(1)}, & x_1^{(2)}, & \cdots, & x_1^{(d)} \\ \vdots & \vdots & & \vdots \\ x_{N_1}^{(1)}, & x_{N_1}^{(2)}, & \cdots, & x_{N_1}^{(d)} \end{pmatrix}.$$

Linear Regression

Ordinary Least Squares (OLS)

Model:	$f_\theta(x) = x\theta.$
Loss Function:	$L(\theta \mid X, Y) = (Y - X\theta)^T (Y - X\theta) \to \min;$
Optimization:	$\hat{\theta} = (X^T X)^{-1} X^T y.$
Prediction:	$\hat{y}_* = x_* \hat{\theta}.$
Validation:	Compute RMSE, R^2, the adjusted R^2, p-value.

Derivation of OLS estimator $\hat{\theta}$

Assume that $X^T X$ is invertible. Recall that the loss function of OLS yields that

$$\begin{aligned} L(\theta \mid X, Y) &= (Y - X\theta)^T (Y - X\theta) \\ &= Y^T Y - 2\theta^T X^T Y + \theta^T X^T X \theta \end{aligned} \tag{1}$$

as $Y^T X\theta = (X\theta)^T Y$. Due to the quadratic loss, the optimal parameter $\hat{\theta}$ should satisfy that the derivative of $L(\theta \mid X, Y)$ evaluated at $\theta = \hat{\theta}$ is equal to zero, i.e.

Continued

$$\frac{\partial L(\theta|X,Y)}{\partial \theta}|_{\theta=\hat{\theta}} = 0.$$

By Equation (1), it follows that $\frac{\partial L(\theta|X,Y)}{\partial \theta} = -2X^TY + 2X^TX\theta$. By setting $\frac{\partial L(\theta|X,Y)}{\partial \theta}$ to zero, we have an linear equation system for $\hat{\theta}$, i.e.

$$-2X^TY + 2X^TX\hat{\theta} = 0. \tag{2}$$

Hence it follows that $\hat{\theta} = (X^TX)^{-1}X^TY$.

The Limitations of OLS

- Even if linear model is a reasonable model, X^TX might not be invertible:
  - There are co-linear dependence between different coordinates of the input variables (Dimension Reduction);
  - The sample size N is smaller than the dimension of the input space. (Overfitting issue)
- Linear model might not be adequate (Underfitting issue);

Overfitting and Regularization

Overfitting

Overfitting is the phenomena of using over-complicated models to fit the data, which results in poor performance on the test data set.

Remark

For OLS, when the sample size is smaller than the input dimension d, there are infinite many θ such that

$$L(\theta|X,Y) = (Y - X\theta)^T(Y - X\theta) = 0,$$

and it leads to the overfitting issue.

Regularization

In order to resolve the overfitting issue here, we consider the constraint optimization problem

$$\min_{\beta}(Y - X\beta)^T(Y - X\beta), \text{ s.t. } ||\beta|| \leq t.$$

Extension of OLS

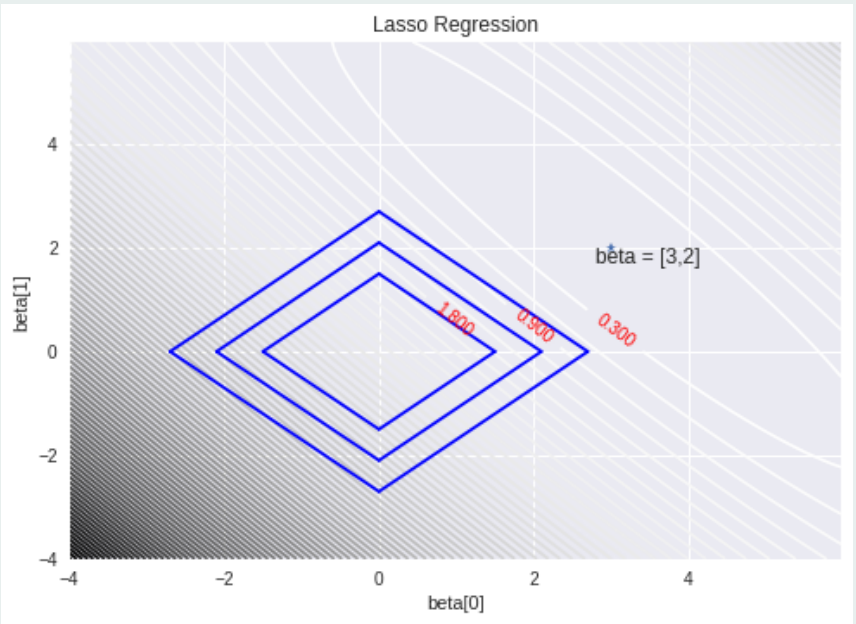
- Regularization (avoid overfitting issue):
  - Lasso Regression: $L(\beta|X,Y) = (Y - X\beta)^T(Y - X\beta) + \lambda||\beta||_1$;
  - Ridge Regression: $L(\beta|X,Y) = (Y - X\beta)^T(Y - X\beta) + \lambda||\beta||_2^2$;
  - Elastic Net:

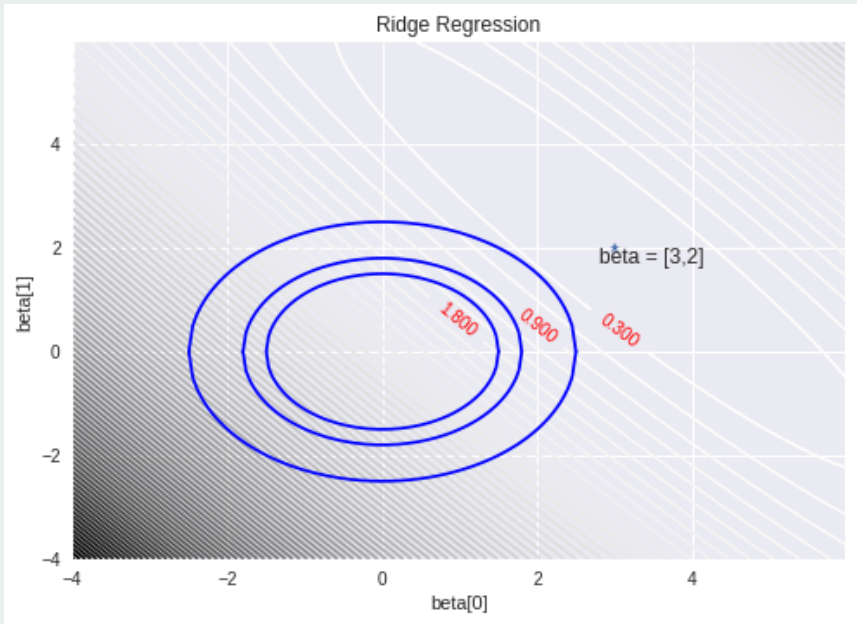
$$L(\beta|X,Y) = (Y - X\beta)^T(Y - X\beta) + \lambda\left(\frac{1-\alpha}{2}||\beta||_2^2 + \alpha||\beta||_1\right);$$

  where $||\beta||_p = \left(\sum_{i=1}^{d}(\beta^{(i)})^p\right)^{1/p}$.
- Robust to outliers:
  - Robust Regression: Modify the Loss function to the weighted sum of the residuals and the weighting is done iteratively.

Comparison of Different Regularization Methods

- Lasso requires numerical techniques to solve, but it can be used for feature selection;
- Ridge regression has the closed form solution for the optimal parameters.
- Elastic nets is a combination of Lasso and Ridge regression.





Example (Sparse Linear Model)

We simulate 1000 samples of the input-output pairs (x_i, y_i) based on the following model:

$$y = x\beta + \varepsilon$$

where $x \in \mathbb{R}^{800}$, $y \in \mathbb{R}$ and β is a sparse vector, i.e. there are only three non-zero coordinates of β.

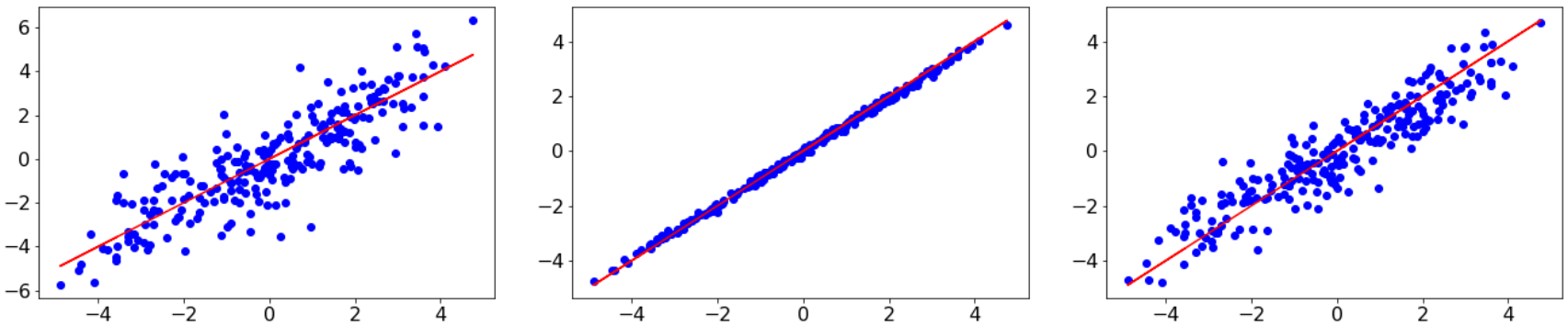


Figure: Plot of the model estimated output by (L) Linear Regression; (M) Lasso Regression; and (R) Ridge Regression v.s.the actual output Ridge.

Sparsity of Lasso coefficient estimator

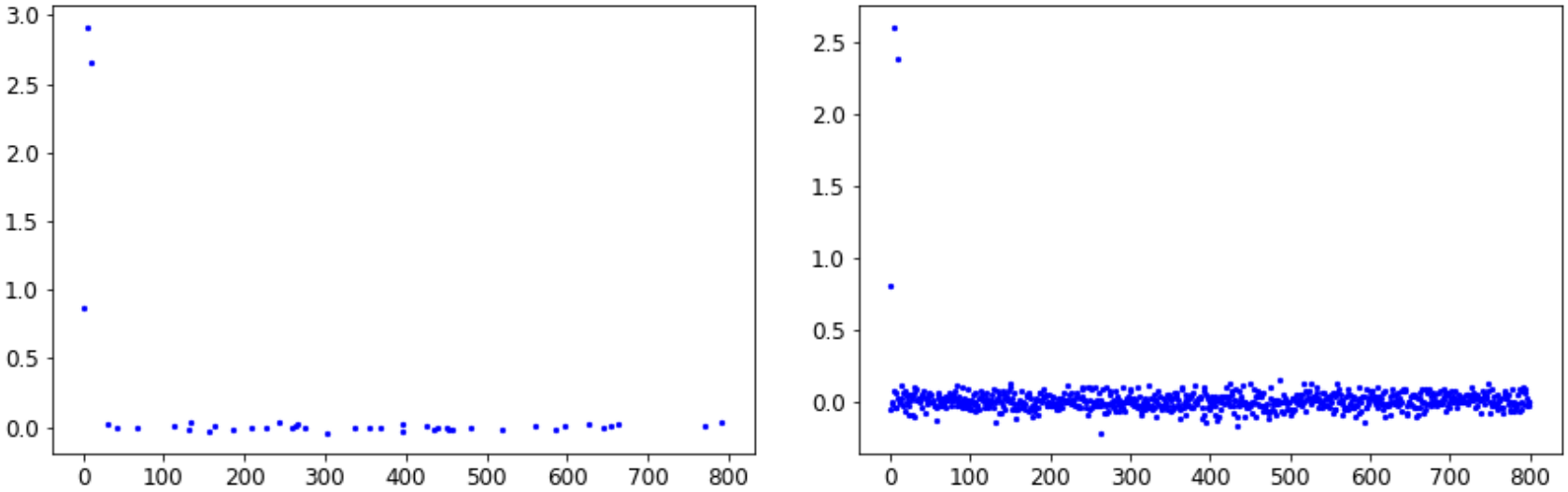


Figure: Visualisation of non-zeros linear coefficients of (Left) Lasso Regression and (Right) Ridge Regression.

Non-Linear Regression

Underfitting issue

An underfitted model is a model where some parameters or terms that would appear in a correctly specified model are missing. Underfitting would occur, for example, when fitting a linear model to non-linear data.

Basis Expansion

$$y = f(x) + \varepsilon;$$

$$f(x) \approx \mathscr{L}(\phi_1(x), \dots, \phi_n(x)) = \sum_{i=1}^{n} \theta_i \phi_i(x), x \in \mathbb{R}^d.$$

Fixed basis functions (Features), e.g.

- Polynomial basis: $x^0, x^1, x^2, \dots, x^n$;
- Spline basis...

Alternative approach

Propose non-linear parameterized functions, e.g. Neural Network.

Optimization

Gradient Descent

Goal: Find the optimal θ such as to minimize the function $L(\theta)$, where L is continuously differentiable.

Algorithm: Initialize θ_0. For any integer $n \geq 1$,

$$\boxed{\theta_{n+1} = \theta_n - \eta \nabla L(\theta_n)}$$

where η is called the learning rate, $\eta > 0$ and $\theta = (\theta_1, \cdots, \theta_p) \in \mathbb{R}^p$ and $\nabla L(\theta) = (\partial_{\theta_1} L(\theta), \cdots, \partial_{\theta_p} L(\theta))$.

Idea: We aim to construct a sequence of $\{\theta_n\}_{n \geq 0}$, which converges to the local minimum θ_*, which implies $\nabla L(\theta_*) = 0$.

- For some N, $(L(\theta_n))_{n \geq N}$ is a decreasing sequence, i.e. $L(\theta_N) \geq L(\theta_{N+1}) \geq \cdots$.
- $\lim_{n \to \infty} \nabla L(\theta_n) = 0$.

Explanation

- When η is small enough,

$$L(\theta_{n+1}) - L(\theta_n) \approx \nabla L(\theta_n)^T \underbrace{(\theta_{n+1} - \theta_n)}_{-\eta \nabla L(\theta_n)} = -\eta \left|\nabla L(\theta_n)\right|^2 \leq 0.$$

  It follows that for some N, $(L(\theta_n))_{n \geq N}$ is a decreasing sequence.

- Suppose that $\{\theta_n\}$ is a convergent series, then

$$\lim_{n\to\infty} \theta_{n+1} = \lim_{n\to\infty} \theta_n - \eta \lim_{n\to\infty} \nabla L(\theta_n) = \theta_* = \theta_* - \eta \lim_{n\to\infty} \nabla L(\theta_n).$$

  It follows $\lim_{n\to\infty} \nabla L(\theta_n) = 0$.

Gradient Descent

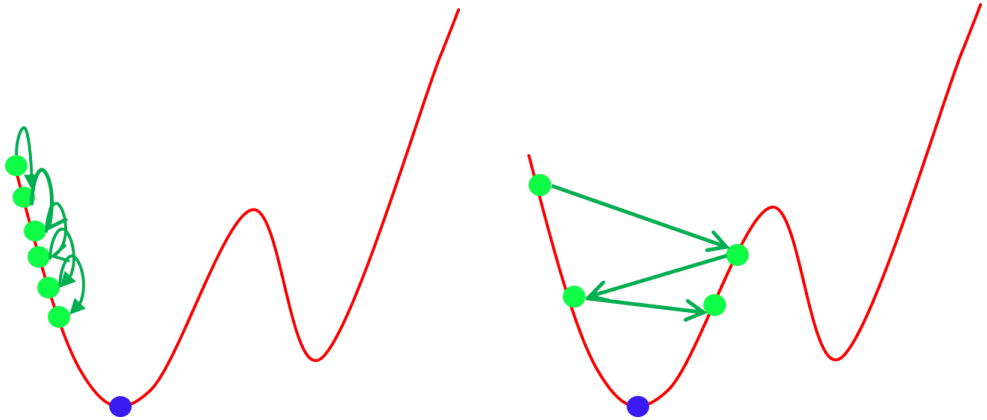


Figure: The Effect of Learning Rate

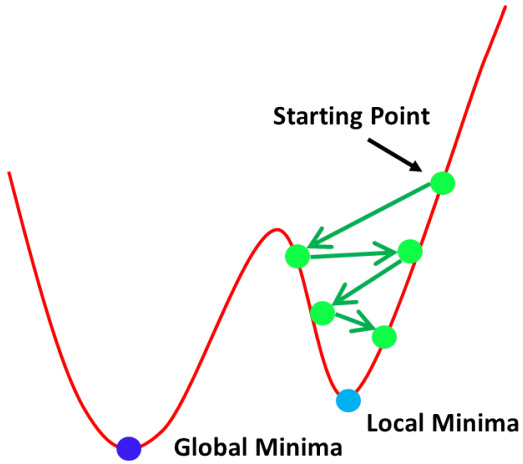


Figure: Global Minimum and Local Minimum

.

Batch Gradient Descent

Goal: Find optimal θ such as to minimize $L(\theta|\mathscr{D})$ in the following form:

$$L(\theta|\mathscr{D}) = \frac{1}{N}\sum_{i=1}^{N} Q_\theta(x_i, y_i),$$

where e.g. $Q_\theta(x, y) = (y - f_\theta(x))^2$ in regression tasks.

Algorithm:

$$\theta_{n+1} = \theta_n - \eta \underbrace{\nabla L(\theta_n|\mathscr{D})}_{\text{Gradient Term}} = \theta_n - \eta \underbrace{\frac{1}{N}\sum_{i=1}^{N} \nabla_\theta Q_{\theta_n}(x_i, y_i)}_{\text{Gradient Term}}.$$

Idea: Direct application of GD to empirical loss function.

Pros

- Stable gradient update and convergence on some problem.
- The computation of gradient term can be implemented in a parallel manner.

Cons

- Stuck at the local minima.
- Large memory required and model updates may become very slow for large datasets.

Variants of Gradient Descent Methods (I)

Stochastic Gradient Descent (SGD)

Algorithm: At each step n, randomly choose the index i_n from $\{1, \cdots, N\}$, and update the weights θ_n as follows:

$$\theta_{n+1} = \theta_n - \eta_n \underbrace{\nabla_\theta Q_{\theta_n}(x_{i_n}, y_{i_n})}_{\text{Stochastic Gradient Term}} .$$

for a suitably chosen decreasing step-size sequence $\{\eta_n\}_n$.
Idea:

$$\mathbb{E}_{x,y}[\nabla_\theta Q_\theta(x_{i_n}, y_{i_n})] = \nabla L(\theta|\mathscr{D})$$

where (x_{i_n}, y_{i_n}) is sampled from the empirical distribution of $(x_i, y_i)_{i=1}^N$.

Pros

- The increased model update frequency may lead to faster learning on some problems;
- Noisy gradient updates can avoid the model to stuck local minima.

Cons

- Too frequent model updates, which may result in longer time to train models on large datasets;
- The unstable and noisy estimate of the gradient → difficulty in convergence.

Variants of Gradient Descent Methods (II)

Mini-Batch Gradient Descent

The mini-batch Gradient Descent computes the gradients on small random sets of instances called mini-batches. Mini-batch GD can be viewed as a combination of BGD and SGD.

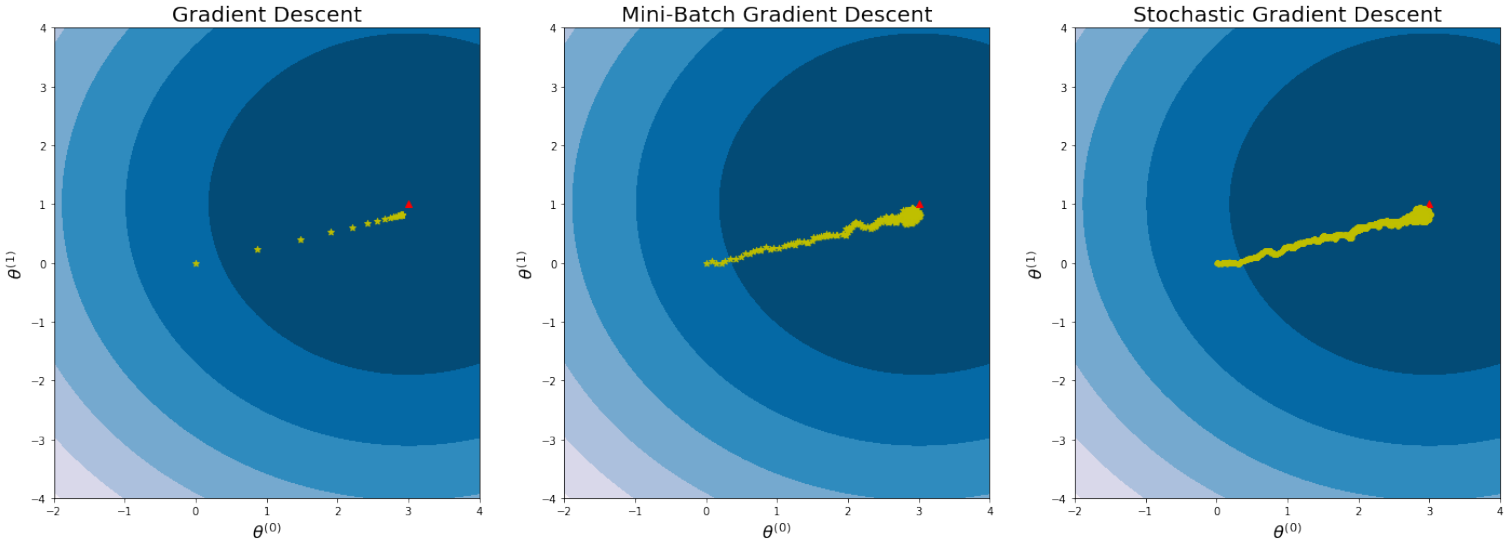
Procedure

In Mini-batch GD, the typical way of creating mini-batches includes the two steps:

- Shuffle the dataset to avoid existing order of samples.
- Split the entire training data set into several non-overlapping mini-batches of the batch size b. we apply the batch gradient descent for each mini-batch until going throughout all the samples (this is called one epoch).

We repeat this procedure until the number of epochs reaches the maximum epoch number N_e.

Comparison of BGD, SGD and Mini-batch GD



- SGD: It is very quick to evaluate each iteration. Randomness helps to escape local minimum, but makes the convergence of the minimum difficult.
- Mini-batch GD: A combination of batch GD and SGD.

3 methods comparison

	BGD	SGD	Mini-Batch GD
Update Frequency	Low	High	Medium
Update Complexity	High	Low	Medium
Fidelity of loss gradient	High	Low	Medium
Stuck the local minimum	Easy	Difficult	Difficult
Easy to Converge	Yes	No	In-between

Table: Comparison of various GD based methods.

From Regression to Classification

Classification

Classification is one type of supervised learning where the output is categorical.

Categorical Variable

A categorical variable means that the variable only has finite many possible values of y_i, denoted by $\mathcal{Y}$. W.l.o.g $\mathcal{Y} = \{1, \cdots, n_o\}$, where n_o denotes the number of possible categories.

Two representations of categorical variables

- Integer encoding (the i^{th} class is represented using an integer i);
- One-hot vector encoding (the i^{th} class is represented using a binary vector of length n_o, which has the unique non-zero element at the i^{th} position).

Example

Blood type	A	B	AB	O
Integer encoding	1	2	3	4
One-hot vector encoding	0001	0010	0100	1000

Table: Different encoding methods for the blood type example.

Remark

The expectation of a random categorical variable does not make sense!! Hence the main objective of the classification is to estimate *the probability of the output being y on conditional on an input x.*

Models

In classification framework, a model $f_\theta : \mathcal{X} \to \mathbb{R}^{n_o}$ is to approximate the conditional probability of the output being y given input x, in formula,

$$\langle f_\theta(x), \bar{y} \rangle \approx \mathbb{P}[y|x],$$

where $\bar{y}$ is one-hot encoding of the class y and $\langle ., . \rangle$ is the inner product of two vectors of length n

Loss Function

Cross Entropy

For discrete probability distributions p and q with the same support $\mathcal{Y}$, the cross entropy is defined to be

$$H(p,q) := -\sum_{j\in\mathcal{Y}} p(j)\log(q(j)).$$

Definition (Cross Entropy Loss Function)

The cross entropy loss function $Q_\theta : \mathcal{X}\times\mathcal{Y}\to\mathbb{R}$ is defined to be

$$Q_\theta(x,y) = -\langle \bar{y}, \log f_\theta(x)\rangle,$$

where $x\in E$, $\bar{y}$ is one-hot encoding in $\mathcal{Y}$, θ are model parameters of f_θ, and $\langle .,.\rangle$ is the inner product.

Empirical Cross Entropy Loss

The empirical cross entropy loss function is given as the average of the above cross entropy loss function evaluated at all samples:

$$L(\theta|\mathscr{D}) = -\frac{1}{N}\sum_{i=1}^{N}\langle \log f_\theta(x_i), \bar{y}_i\rangle.$$

MLE Interpretation of Cross Entropy

$$\prod_{i=1}^{N}\langle f_\theta(x_i), \bar{y}_i\rangle \to \max,$$

which is equivalent to minimize the negative log-likelihood ratio, i.e.

$$-\sum_{i=1}^{N}(\langle \log f_\theta(x_i), \bar{y}_i\rangle) \to \min.$$

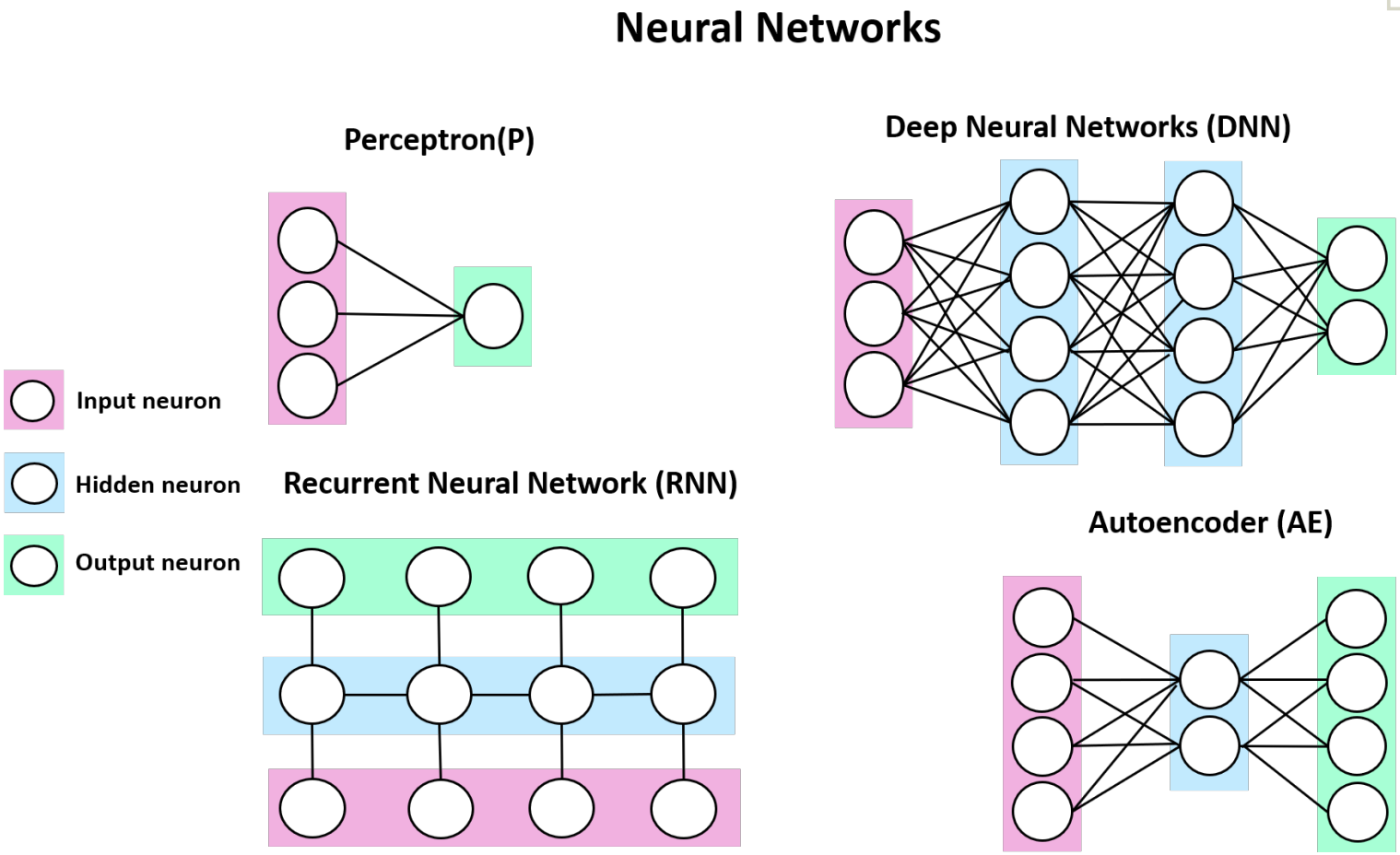
Comparison between Regression and Classification

	Regression	Classification
Dataset:	$\mathscr{D} = \{(x_i, y_i)\}_{i=1}^N, (x_i, y_i) \in \mathscr{X} \times \mathscr{Y}$	
	$\mathscr{Y} = \mathbb{R}^d$	$\mathscr{Y}$ is a finite set.
Model:	$y = f_\theta(x) + \varepsilon$	
	$f_\theta(x)(\approx \mathbb{E}[y\|x])$	$f_\theta(x)(\approx \mathbb{P}[y\|x])$
Empirical Loss:	$L(\theta\|\mathscr{D}) \rightarrow$Minimize	
	(e.g. MSE)	(e.g. Cross entropy)
Optimization:	$\theta^* = \arg\min_\theta(L(\theta\|\mathscr{D}))$	
Prediction:	$\hat{y}_* = f_{\theta^*}(x_*)$.	$\hat{y}_* = \arg\max_{i \in \mathscr{Y}} f^i_{\theta^*}(x_*)$.
Validation:	Compute the test metrics	
	e.g.MSE.	e.g. Accuracy.

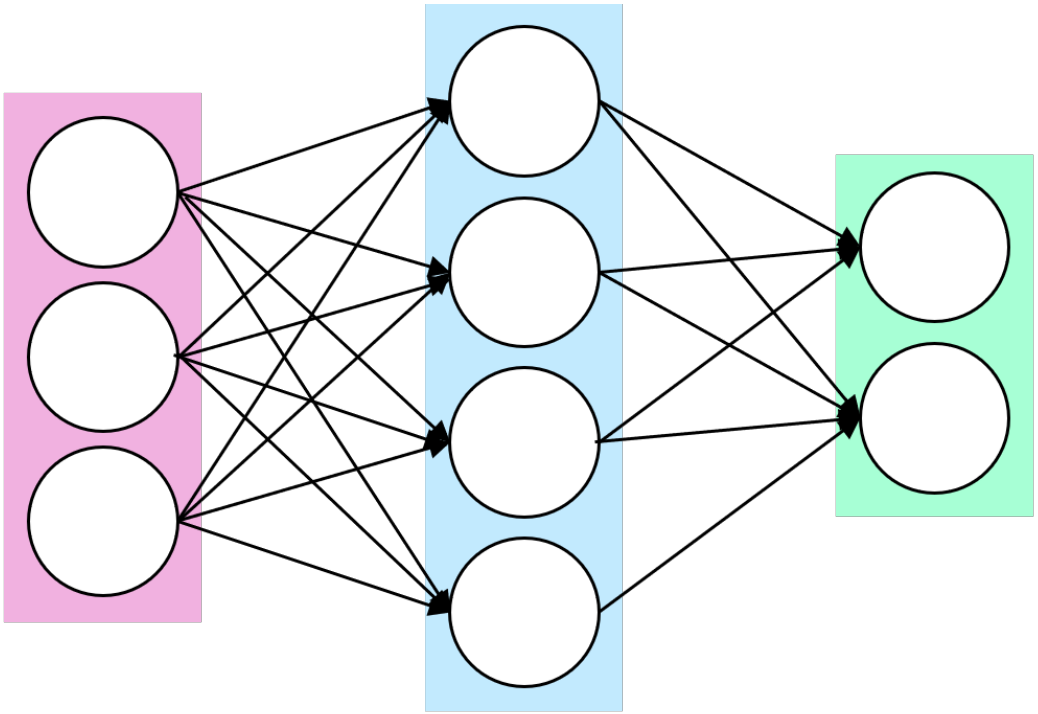
Table: Summary of the framework of regression and classification. The difference between them is highlighted in blue.

Outline

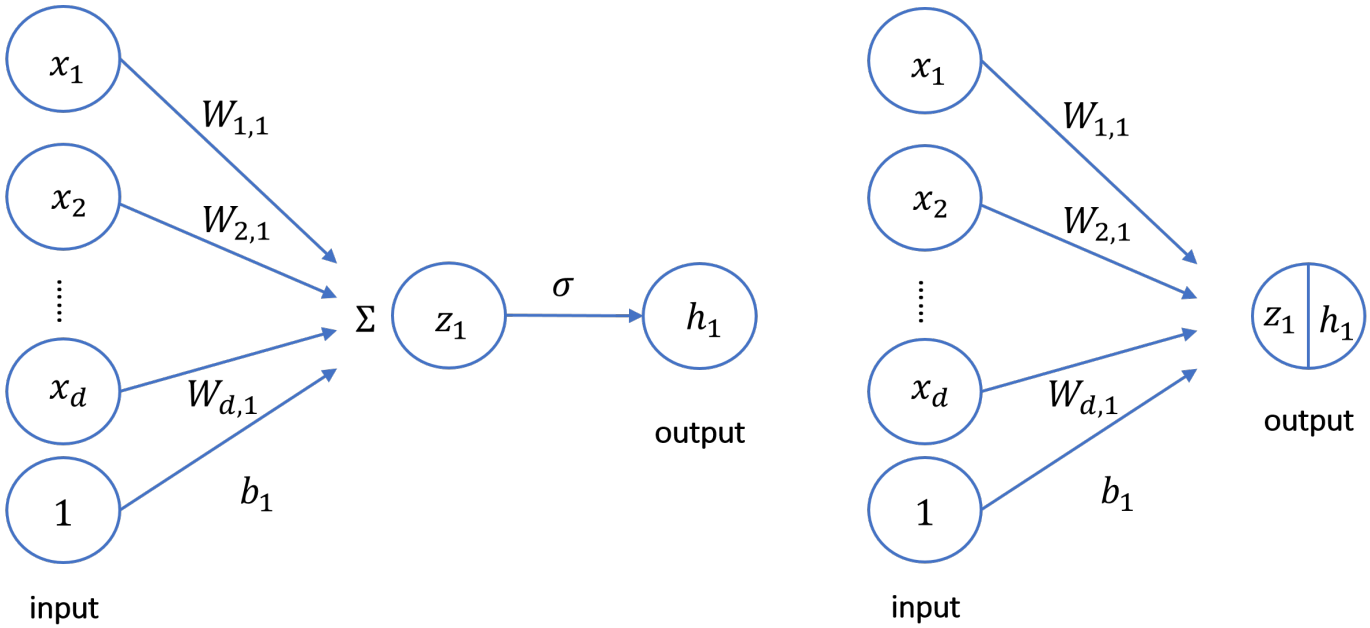
Neural Networks



Shallow Neural Network



Dense layer



$$z_1 = \sum_{i=1}^{n_1} W_{i,j} x^{(i)} + b_1, h_1 = \sigma(z_1)$$

Figure: Dense layer.

Deep Neural Network(DNN)

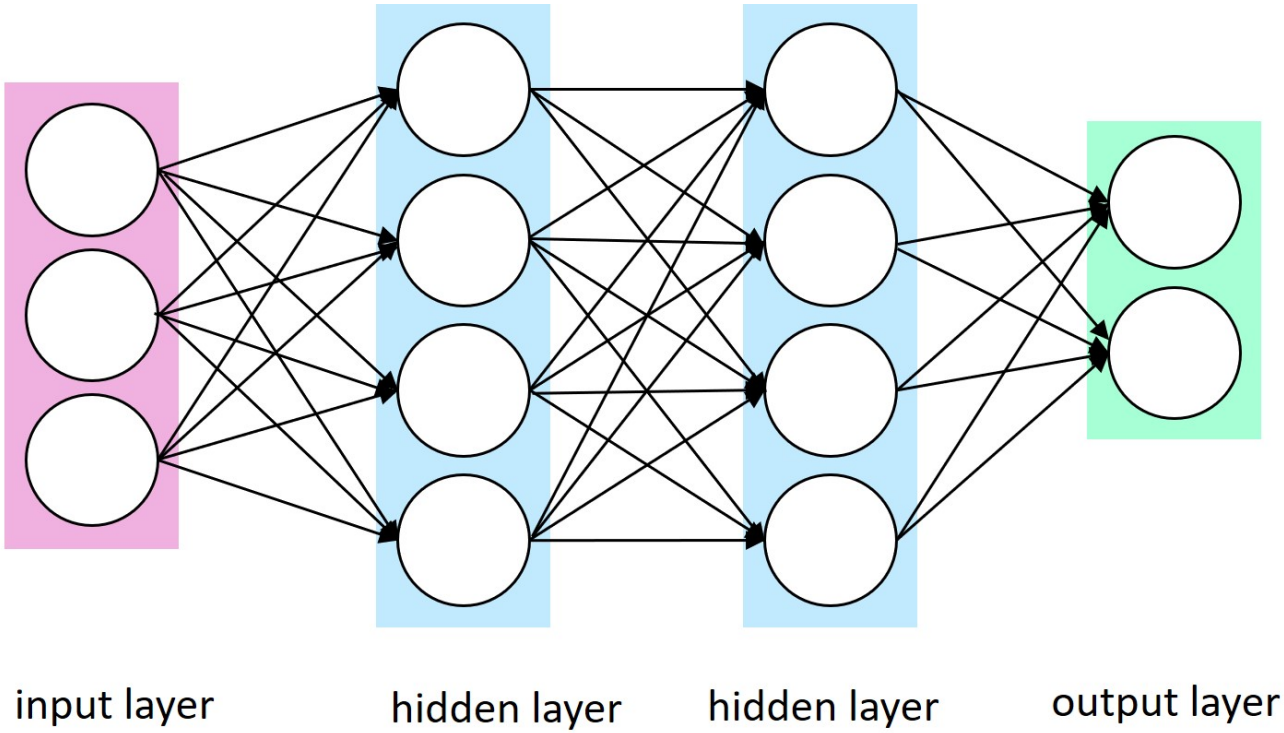


Figure: An example of Deep Neural Network.

Model

Let n_l denote the dimension of neurons at the l^{th} layer.

- Input layer ($h^{(0)} : \mathbb{R}^d \to \mathbb{R}^{n_0}$) with $n_0 = d$:

$$h^{(0)}(x) = x.$$

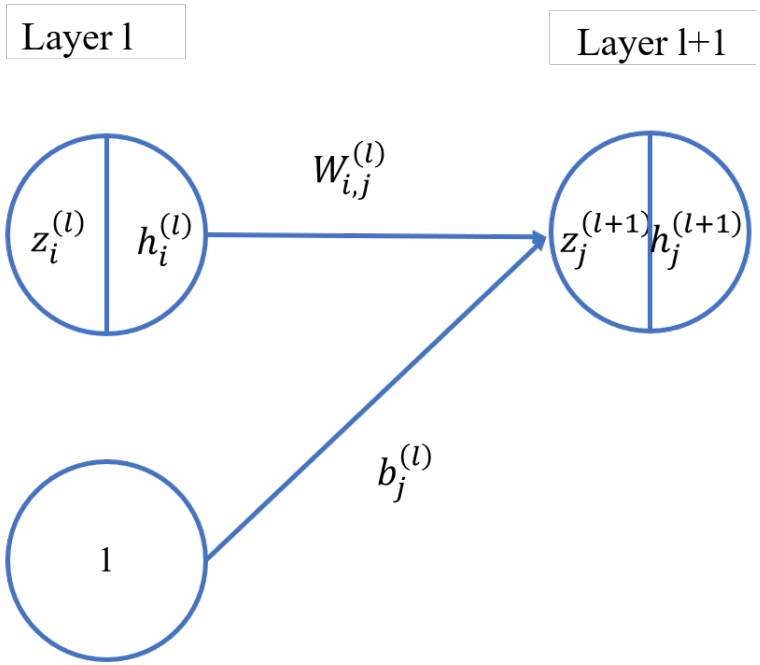
- Hidden layer ($h^{(l)} : \mathbb{R}^d \to \mathbb{R}^{n_l}$): $\forall l \in \{1, 2, \cdots, L-1\}$.
- Output layer ($h^{(L)} : \mathbb{R}^d \to \mathbb{R}^e$).

$(h^{(l)})_{l=1}^L$ is defined in the following recursive way that for any $x \in \mathbb{R}^d$,

$$\begin{aligned} z^{(l+1)}(x) &= h^{(l)}(x)W^{(l)} + b^{(l)} \\ h^{(l+1)}(x) &= \sigma_{l+1}(z^{(l+1)}(x)), \end{aligned}$$

where $W^{(l)}$ is a $n_l \times n_{l+1}$ matrix, $b^{(l)}$ is a n_{l+1} dimensional vector, σ_l is an activation function, e.g. the sigmoid function $\sigma(x) = \frac{1}{1+e^{-x}}$. The parameter set is $\theta = (W^{(l)}, b^{(l)})_{l=1}^L$.

Neural Network Building Block



Formulas

For any $l \in \{2, \cdots, (L-1)\}$,

$$h_j^{(l)} = \sigma_l(z_j^{(l)})$$

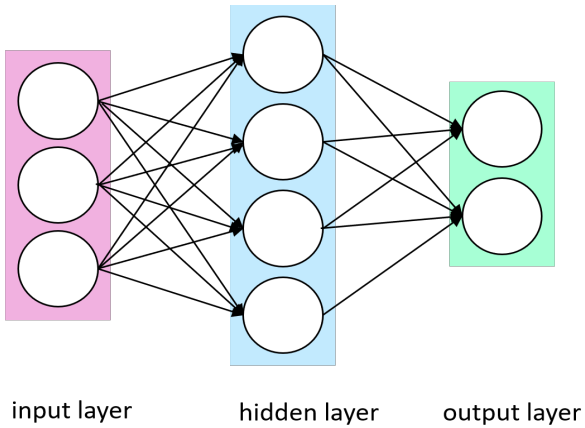
$$z_j^{(l+1)} = \sum_{i=1}^{n_l} W_{i,j}^{(l)} h_i^{(l)} + b_j^{(l)},$$

where $j \in \{1, \cdots, n_{l+1}\}$, $W_{i,j}^{(l)}$ is the weight from incoming node i to output node j on the layer l.

Universality

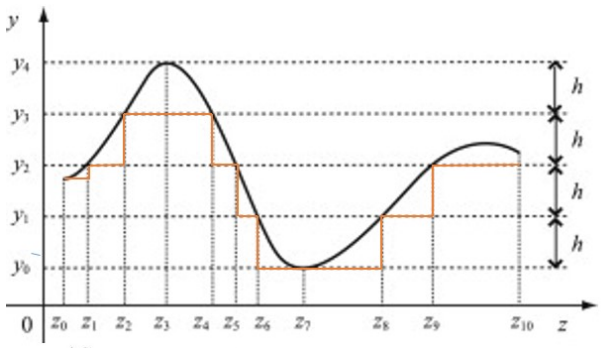
Neural network is NOT a Magic! It is about *function approximation theory*.

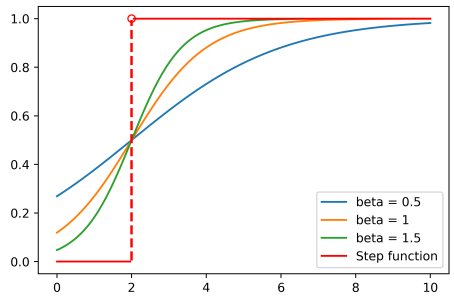
$$f \approx f_\Theta$$



$$f_\Theta(x) = \sum_{j=1}^{M} \alpha_j \sigma\left(\beta_j x + \theta_j\right),$$

where $\Theta := \left\{\alpha_j, \theta_j, \beta_j\right\}_{j=1}^{M}$.





Neural Networks

Theorem (Universal Approximation Theorem[2])

Let σ be any continuous discriminant function (e.g. sigmoid function). The finite sum of the form

$$\sum_{j=1}^{M} \alpha_j \sigma\left(\beta_j^T x + \theta_j\right)$$

are dense in $C(I_n) := \left\{ f : I_n \to \mathbb{R} | \quad f \text{ is continuous } \right\}$, where $I_n = [0,1]^n$. In other words, given any $f \in C(I_n)$ and $\varepsilon > 0$, $\exists f_\Theta(x)$ of the above form, for which

$$|f_\Theta(x) - f(x)| < \varepsilon, \forall x \in I_n$$

where $\Theta := \left\{\alpha_j, \theta_j, \beta_j\right\}_{j=1}^{M}$.

Caution

Universality does not mean usefulness. Over-complicated model may lead to the overfitting issue.

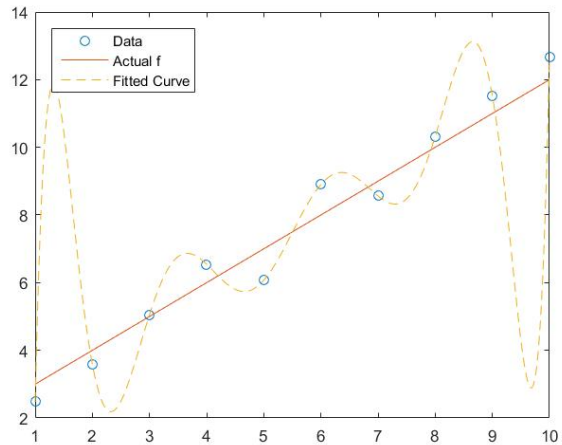
Fitting a Machine Learning Model is a challenge!

Underfitting issue

Underfit can be avoided by sophisticated machine learning model thanks to the universal approximation theory.

Overfitting Issue

Overfitting is the production of an analysis that corresponds too closely or exactly to a particular set of data, and may therefore fail to fit additional data or predict future observations reliably.



Parameter Optimization

Recall that the optimal model parameter θ^* is the one to minimise the loss function $L(\theta|\mathscr{D})$, which is often in the additive form as follows:

$$L(\theta|\mathscr{D}) = \frac{1}{N}\sum_{i=1}^{N} Q_\theta(x_i, y_i).$$

E.g., $Q_\theta(x, y) = Q(h^{(L)}(x) - y)$.

Goal: To compute $\nabla Q_\theta(x, y)$, i.e.

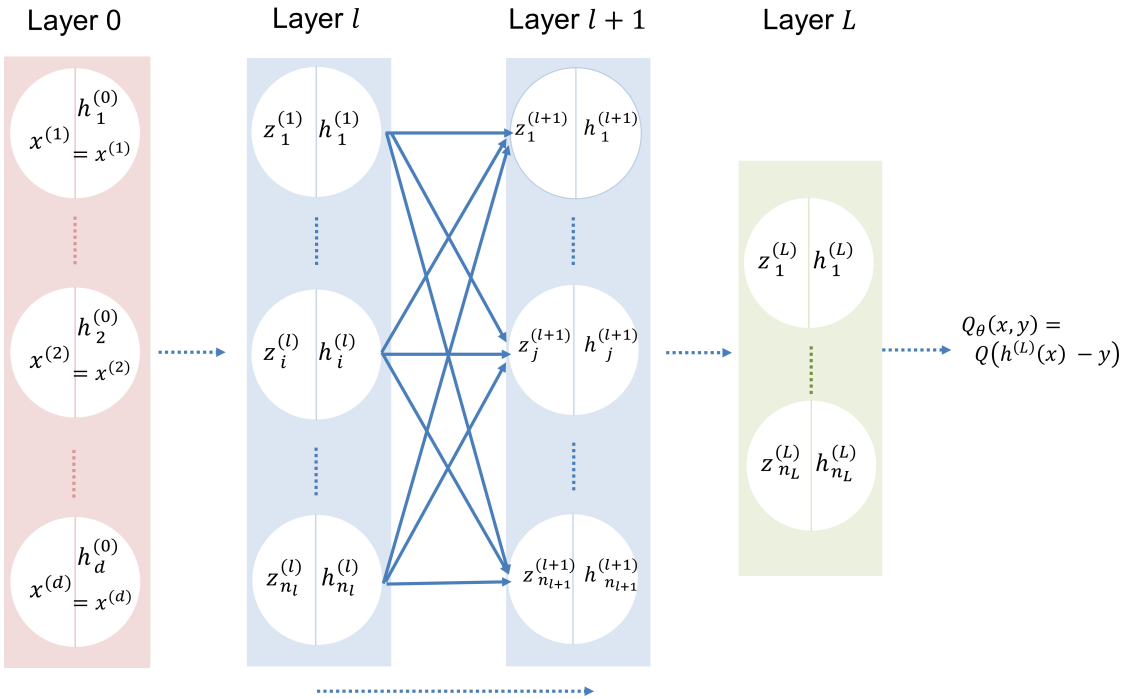
$$\partial_{W_{i,j}^{(l)}} Q_\theta(x, y) \text{ and } \partial_{b_i^{(l)}} Q_\theta(x, y).$$

Solution: Recursion and Chain Rule.

Forward Phase

For each $l \in \{1, \cdots, L-1\}$, compute

$$z^{(l+1)}(x) = W^{(l)} h^{(l)}(x) + b^{(l)},$$
$$h^{(l+1)}(x) = \sigma_{l+1}(z^{(l+1)}(x)).$$

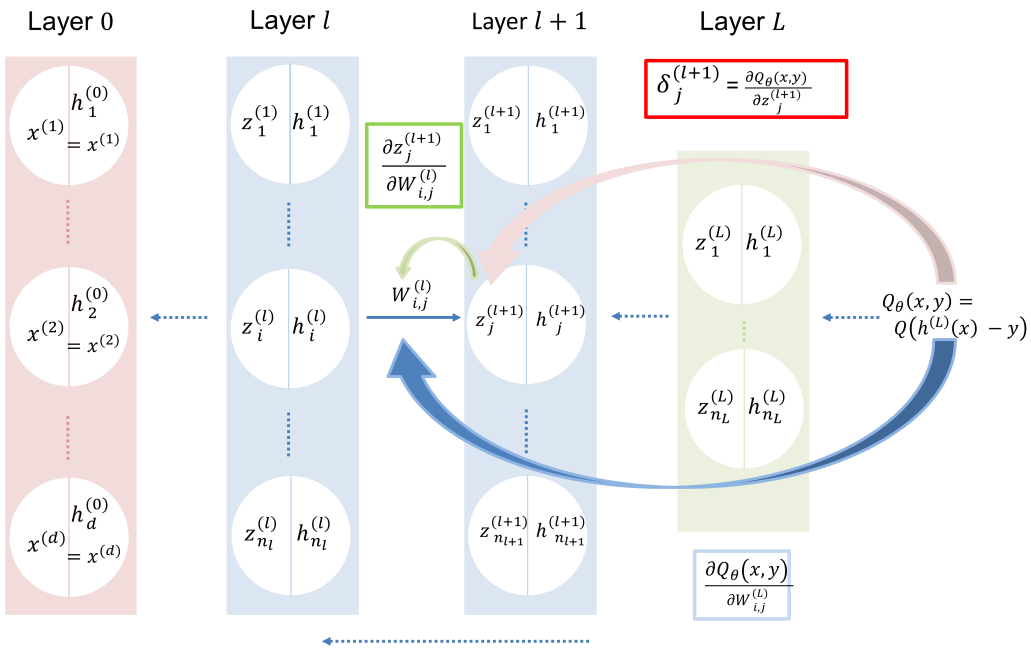


Backward Pass

How to compute gradient $\nabla Q_\theta(x, y)$ efficiently?

Recursion: Only $z^{(l+1)}(x)$ depends on $W^{(l)}$ and $b^{(l)}$.

$$Q_\theta(x, y) = q(z^{(l+1)}(x), W^{(l+1)}, b^{(l+1)}, \cdots, W^{(L-1)}, b^{(L-1)}, y)$$



The important intermediate variable is

$$\delta_i^{(l)} := \partial_{z_i^{(l)}} Q_\theta(x, y),$$

as $\partial_{W^{(l)}} Q_\theta(x)$ and $\partial_{b^{(l)}} Q_\theta(x)$ follows

$$\begin{aligned}
\partial_{W_{i,j}^{(l)}} Q_\theta(x) &= \partial_{W_{i,j}^{(l)}} z_j^{(l+1)} \cdot \partial_{z_j^{(l+1)}} Q_\theta(x) = \left(\partial_{W_{i,j}^{(l)}} z_j^{(l+1)} \right) \delta_j^{(l+1)}. \\
\partial_{b_i^{(l)}} Q_\theta(x) &= \partial_{b_i^{(l)}} z_j^{(l+1)} \cdot \partial_{z_j^{(l+1)}} Q_\theta(x) = \left(\partial_{b_i^{(l)}} z_j^{(l+1)} \right) \delta_j^{(l+1)}.
\end{aligned}$$

The problem is reduced to compute $\partial_{W_{i,j}^{(l)}} z_j^{(l+1)}$, $\partial_{b_i^{(l)}} z_j^{(l+1)}$ and $\delta_i^{(l)}$.

$$\delta_i^{(l)} = \sum_{j=1}^{n_{l+1}} \delta_j^{(l+1)} \partial_{z_i^{(l)}} z_j^{(l+1)}.$$

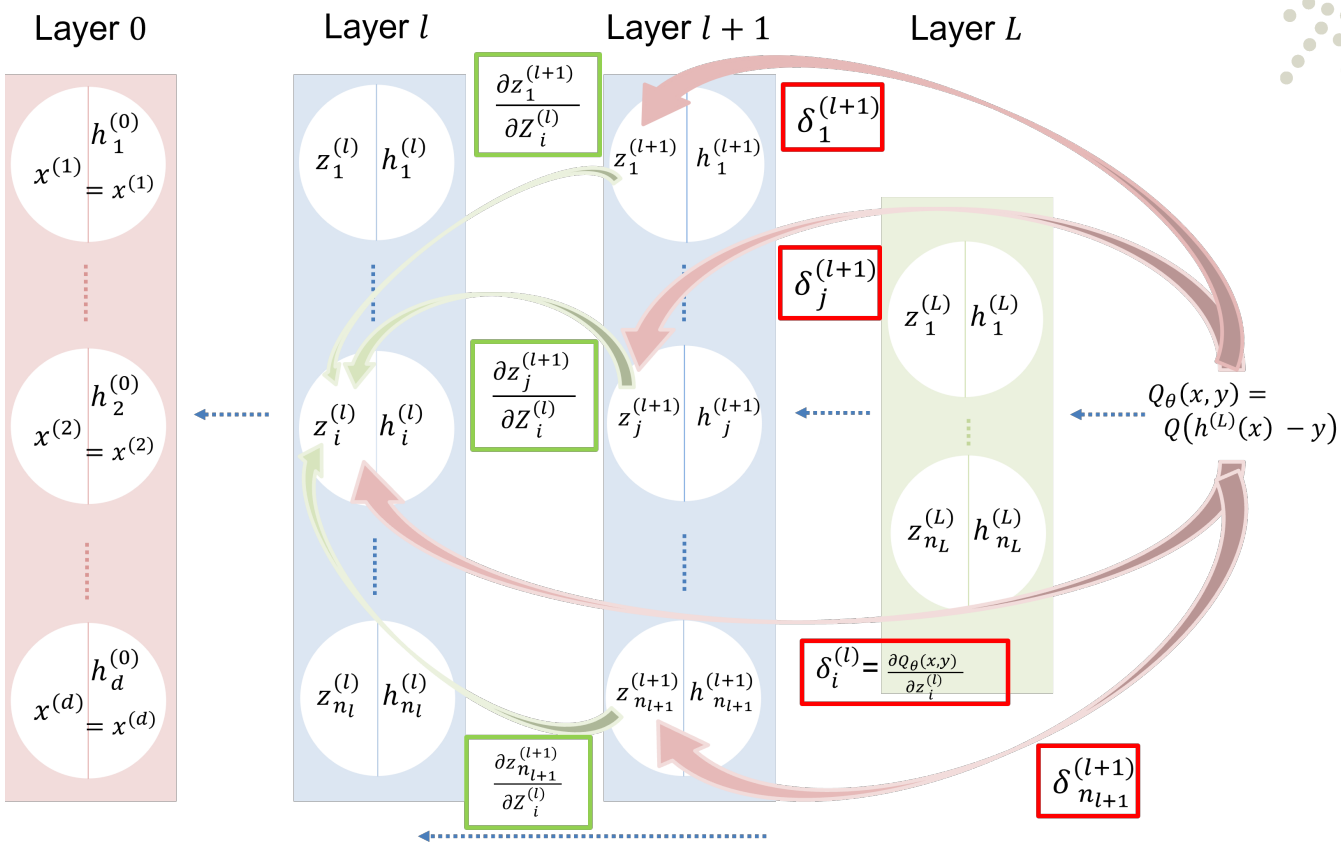


Figure: Illustration of recursive computation for $\delta_i^{(l)}$

Derivatives between Layers

$$
\begin{aligned}
\partial_{W_{i,j}^{(l)}} z_j^{(l+1)} &= \partial_{W_{ij}^{(l)}} \Big(\sum_k W_{k,j}^{(l)} h_k^{(l)} + b_j^{(l)} \Big) = h_i^{(l)} \\
\partial_{b_i^{(l)}} z_j^{(l+1)} &= \partial_{b_i^{(l)}} \Big(\sum_k W_{k,j}^{(l)} h_k^{(l)} + b_j^{(l)} \Big) = 1 \\
\partial_{z_i^{(l)}} z_j^{(l+1)} &= \partial_{z_i^{(l)}} \Big(\sum_k W_{k,j}^{(l)} h_k^{(l)} + b_j^{(l)} \Big) = \partial_{z_i^{(l)}} (W_{i,j}^{(l)} h_i^{(l)}) = W_{i,j}^{(l)} \partial_{z_i^{(l)}} (\sigma(z_i^{(l)})) \\
&= W_{i,j}^{(l)} \sigma_l'(z_i^{(l)}).
\end{aligned}
$$

Recursively Compute $\delta_i^{(l)} = \partial_{z_i^{(l)}} Q_\theta(x, y)$.

$$
\delta_i^{(l)} = \sum_j \partial_{z_i^{(l)}} z_j^{(l+1)} \delta_j^{(l+1)} = \sum_j \sigma_l'(z_i^{(l)}) W_{i,j}^{(l)} \delta_j^{(l+1)} = \sigma_l'(z_i^{(l)}) (W^{(l)} \delta^{(l+1)})_i.
$$

Backward Propagation Algorithm

Input: θ and (x, y); *Output*: $\nabla_\theta Q_\theta(x, y)$.

1. **Forward Phase**: for each $l \in \{1, \cdots, L-1\}$, compute

$$z^{(l+1)}(x) = W^{(l)} h^{(l)}(x) + b^{(l)},$$
$$h^{(l+1)}(x) = \sigma(z^{(l+1)}(x)).$$

2. **Backward Phase**:
For $l = L$,
$\delta^L = \partial_{z^{(L)}} Q_\theta(x, y) = \partial_{z^{(L)}} Q(h^{(L)}(x) - y) = Q'(h^{(L)}(x) - y)\sigma_L'(z^{(L)}(x))$;
For $l = L-1 : -1 : 1$,

$$\delta^{(l)} = \sigma_l'(z^{(l)}(x)) \odot (W^{(l)} \delta^{(l+1)});$$
$$\partial_{W_{i,j}^{(l)}} Q_\theta(x, y) = h_i^{(l)} \delta_j^{(l+1)}; \qquad \partial_{b_j^{(l)}} Q_\theta(x, y) = \delta_j^{(l+1)}.$$

Recurrent Neural Network (RNN)

Let x_t, s_t and o_t denote the input neuron, hidden neuron and output neuron at time t respectively. Let $\bar{x} = (x_t)_{t=1}^T \in \mathbb{R}^{T\times d}$, $\bar{s} = (x_t)_{t=1}^T \in \mathbb{R}^{T\times n_1}$ and $\bar{o} = (o_t)_{t=1}^T \in \mathbb{R}^{T\times e}$.

- Input Layer $(l^0) : \bar{x} \mapsto \bar{x}$.
- Hidden Layer $(l^1) : \bar{x} \mapsto \bar{s}$,

$$s_t = h(\boldsymbol{U} x_t + \boldsymbol{W} s_{t-1}).$$

- Output Layer $(l^2) : \bar{x} \mapsto \bar{o} = (o_t)_{t=1}^T$ or o_T,

$$o_t = g(\boldsymbol{V} s_t).$$

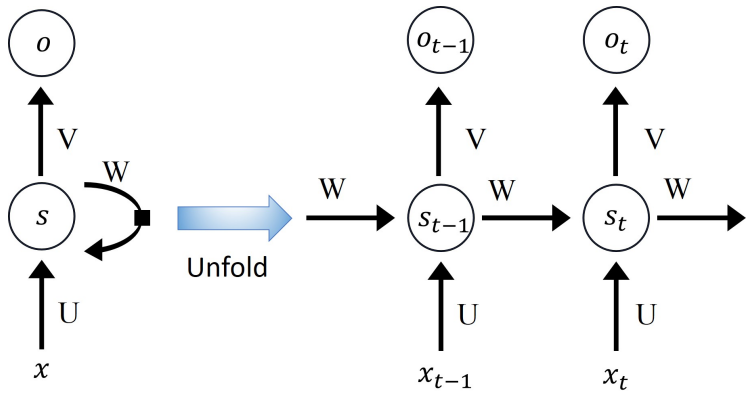


Table: Model Summary of RNN.

Here $\boldsymbol{U}$ is a $n_1 \times d$ matrix of weights, $\boldsymbol{W}$ is a $n_1 \times n_1$ matrix of weights and $\boldsymbol{V}$ is a $e \times n_1$ matrix of weights. g and h are two activation functions. $(\boldsymbol{U}, \boldsymbol{W}, \boldsymbol{V})$ are trainable model parameters.

Loss function

For a task of predicting the sequential output, the loss function is often in the additive form of the following equation:

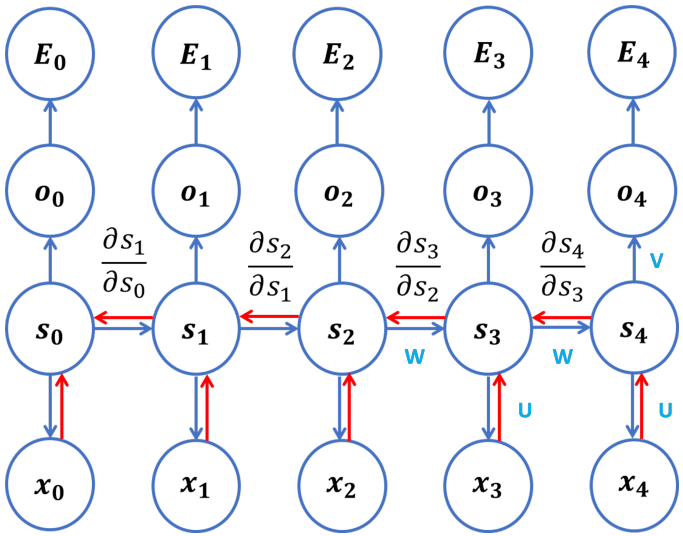
$$\text{Loss Function:} L(\bar{o}, \bar{y}) = \sum_{t=1}^{T} E_t(o_t, y_t),$$

where y_t and o_t are the actual/estimated output at time t respectively, $\bar{o} = (o_t)_{t=1}^{T}$ and $\bar{y} = (y_t)_{t=1}^{T}$. The simplest choice for $E_t(o, y)$ is the squared error of o and y for the regression problem. However, E_t can be in a more complicated form, e.g. a time-dependent function.

Optimization

Optimization of RNN

- Stochastic/Mini-batch Gradient Decent;
- Gradient calculation: Backpropagation through Time.



Goal: To compute $\frac{dE_t}{dV}, \frac{dE_t}{dU}, \frac{dE_t}{dW}$.

$$\frac{dE_t}{dV} = \frac{\partial E_t}{\partial o_t} \cdot \frac{\partial o_t}{\partial V}$$

$$\frac{dE_t}{dU} = \frac{\partial E_t}{\partial o_t} \cdot \frac{\partial o_t}{\partial U} = \frac{\partial E_t}{\partial o_t} \frac{\partial o_t}{\partial s_t} \cdot \frac{ds_t}{dU}$$

$$\frac{dE_t}{dW} = \frac{\partial E_t}{\partial o_t} \frac{\partial o_t}{\partial s_t} \cdot \frac{ds_t}{dW}$$

Derivative Computation

The computation of total derivatives boils down to

- $\frac{\partial E_t}{\partial o_t}$, $\frac{\partial o_t}{\partial s_t}$, $\frac{\partial o_t}{\partial V}$;
- $\frac{ds_t}{dW}$, $\frac{ds_t}{dU}$.

First let us explain the derivation of the partial derivatives $\frac{\partial E_t}{\partial o_t}$ and $\frac{\partial o_t}{\partial s_t}$. The computation of $\frac{\partial o_t}{\partial V}$ is similar and hence left for the homework.

$$\frac{\partial E_t}{\partial o_t} = \frac{\partial (o_t - y_t)^2}{\partial o_t} = 2(o_t - y_t).$$

Recall that $o_t = g(V s_t)$ where $s_t \in \mathbb{R}^{n_1}$, $o_t \in \mathbb{R}^e$ and V is a matrix of size (e, n_1). $\frac{\partial o_t}{\partial s_t}$ is a matrix of size (e, n_1), i.e.

$$\frac{\partial o_t}{\partial s_t} = \left(\frac{\partial o_t^i}{\partial s_t^j} \right)_{i \in [e], j \in [n_1]}. \tag{3}$$

Lemma

If $g : \mathbb{R} \to \mathbb{R}$ *is differentiable and* $o_t = g(V s_t)$*, then it holds that* $\forall i \in [e]$ *and* $j \in [n_1]$*,*

$$\frac{\partial o_t^i}{\partial s_t^j} = V_{ij} g' \left(\sum_{k=1}^{n_1} V_{ik} s_t^k \right). \tag{4}$$

Proof

For any $\forall i \in [e]$,

$$o_t^i = g((V s_t)^i = g \left(\sum_{k=1}^{n_1} V_{ik} s_t^k \right).$$

Then by Chain rule it follows that for any $j \in [n_1]$,

$$\frac{\partial o_t^i}{\partial s_t^j} = V_{ij} g' \left(\sum_{k=1}^{n_1} V_{ik} s_t^k \right).$$

□

In the following, we focus on the discussion on the derivation of $\frac{ds_t}{dU}$ and $\frac{ds_t}{dW}$.

By the definition of s_t, the recurrence of $\frac{ds_t}{dU}$ and $\frac{ds_t}{dW}$ holds as follows:

$$s_t = h(U x_t + W s_{t-1}) \implies \frac{ds_t}{dU} = \frac{\partial s_t}{\partial U} + \frac{\partial s_t}{\partial s_{t-1}} \frac{ds_{t-1}}{dU},$$
$$\frac{ds_t}{dW} = \frac{\partial s_t}{\partial W} + \frac{\partial s_t}{\partial s_{t-1}} \frac{ds_{t-1}}{dW}.$$

Lemma (Recurrence Structure of $\frac{ds_t}{dU}$ and $\frac{ds_t}{dW}$)

For any $t \in \{1, 2, \cdots, T\}$,

$$\frac{ds_t}{dU} = \frac{\partial s_t}{\partial U} + \sum_{k=0}^{t-1} \left(\prod_{j=k+1}^{t} \frac{\partial s_j}{\partial s_{j-1}} \right) \frac{\partial s_k}{\partial U}, \tag{5}$$

$$\frac{ds_t}{dW} = \frac{\partial s_t}{\partial W} + \sum_{k=0}^{t-1} \left(\prod_{j=k+1}^{t} \frac{\partial s_j}{\partial s_{j-1}} \right) \frac{\partial s_k}{\partial W}. \tag{6}$$

Proof

Since $s_t = h(\boldsymbol{U} x_t + \boldsymbol{W} s_{t-1})$ and s_{t-1} also depends on $\boldsymbol{U}$, it follows:

$$\frac{ds_t}{dU} = \frac{\partial s_t}{\partial U} + \frac{\partial s_t}{\partial s_{t-1}} \frac{ds_{t-1}}{dU}.$$

Applying the above equation for the term $\frac{ds_{t-1}}{dU}$, it follows that

$$\begin{aligned} \frac{ds_t}{dU} &= \frac{\partial s_t}{\partial U} + \frac{\partial s_t}{\partial s_{t-1}} \frac{ds_{t-1}}{dU} \\ &= \frac{\partial s_t}{\partial U} + \frac{\partial s_t}{\partial s_{t-1}} \left(\frac{\partial s_{t-1}}{\partial U} + \frac{\partial s_{t-1}}{\partial s_{t-2}} \frac{ds_{t-2}}{dU} \right) \end{aligned}$$

Repeating this procedure until reaching $t = 0$, we have the formula Equation (5). The rigorous proof can be done as follows by induction. □

Gradient issues of RNNs

Cons of Backpropagation through time

- Heavy computational cost
- vanishing/exploding gradient problems

Long short-term memory (LSTM)

LSTM is a variant of RNN models to use the gate mechanism to better capture long term temporal dependency.

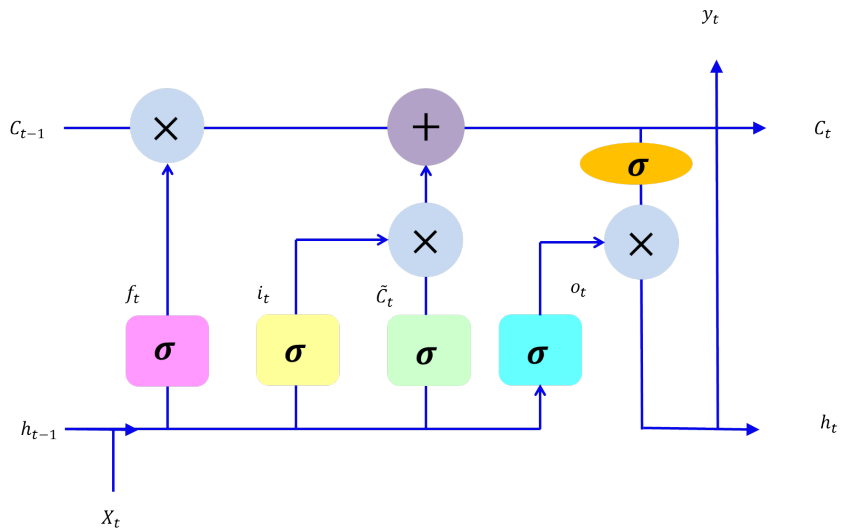
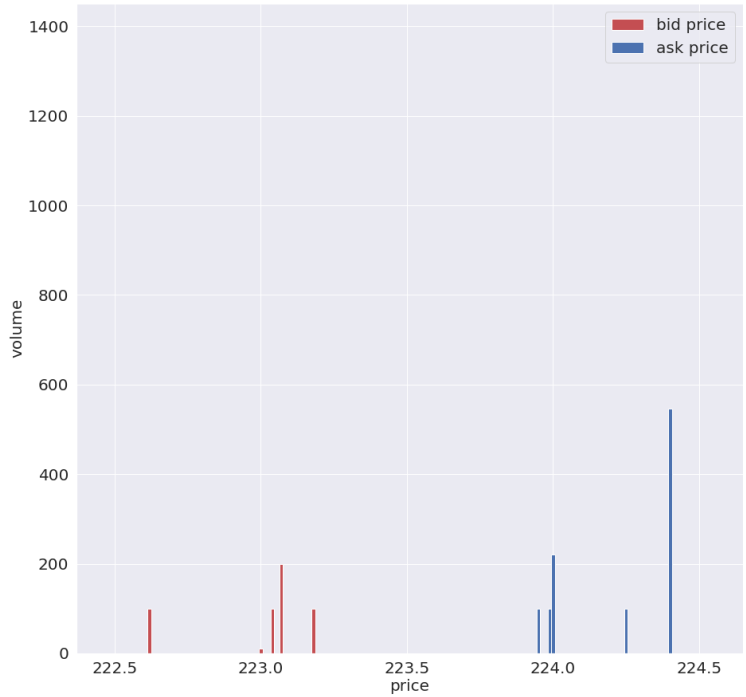


Figure: Long Short Term Memory.

A standard LSTM unit is composed of a cell, an input gate, an output gate and a forget gate. The cell remembers values over arbitrary time intervals, and three gates regulate the flow of information into and out of the cell (See Figure 9). The mathematical formulation of LSTM can refer to Chapter 5.4 in [6] and [5].

Outline

Application: Predicting future price movement



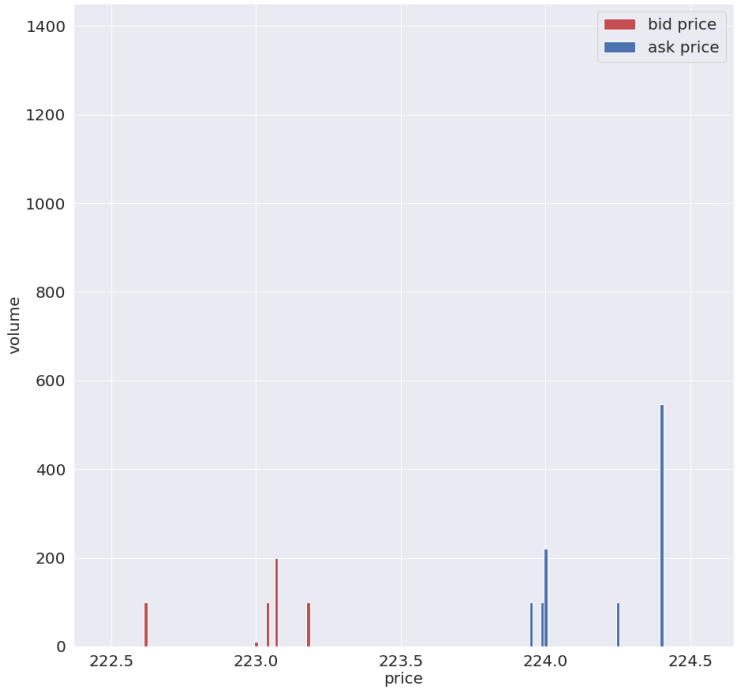


Figure: Left: the video of the Limit order book movements; Right: the snapshot of limit order book, which is composed with the bid/ask prices/volume up to certain depth.

Pipelines of ML application

Question

Given the history of the limit order book of one asset, how to build an accurate model to predict whether the next mid-price is going up or down?

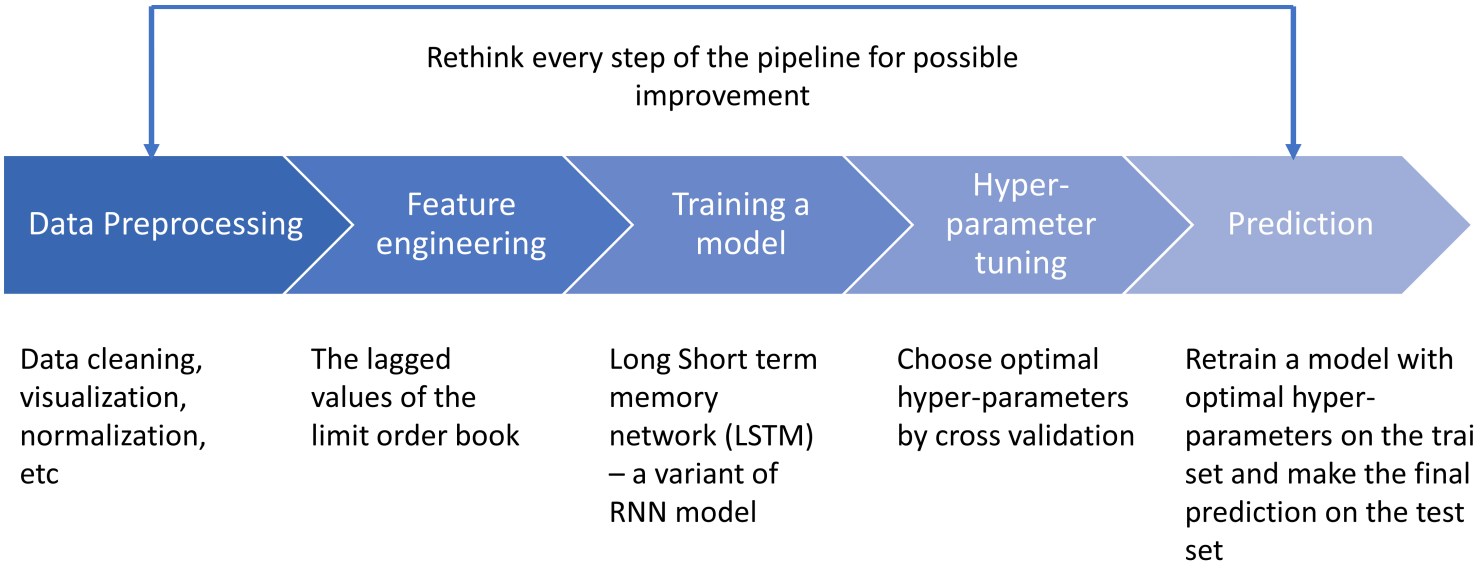


Figure: To tackle this problem, we choose the features and model for this task following [8].

Let LOB_t denote the limit order book information including the bid/ask prices/volumes of limit order up to depth 10 at the t^{th} time when the mid-price changes.The mid-price is the average of the best bid and ask price.

3-layer LSTM for binary classification

Data: Input-Output pairs $(X_t, Y_t)_{t=}^{N}$, where
$X_t := (\text{LOB}_i)_{i=t-49}^{t} \in \mathbb{R}^{40\times 50}$ and
$Y_t := \mathbf{1}(\text{MidPrice}_{t+1} > \text{MidPrice}_t) \in \{0, 1\}$

Model: $f_\theta(X) \sim \mathbb{P}(Y|X)$.

Loss function: cross entropy

Optimization: mini-batch Gradient Descent.

Prediction: $\hat{Y} = \arg\max_{i\in\{0,1\}} \underbrace{\hat{P}(Y = i|X)}_{f_{\hat{\theta}}^{(i)}(X)}$.

Validation: Accuracy, Precision, Recall, f1 score.

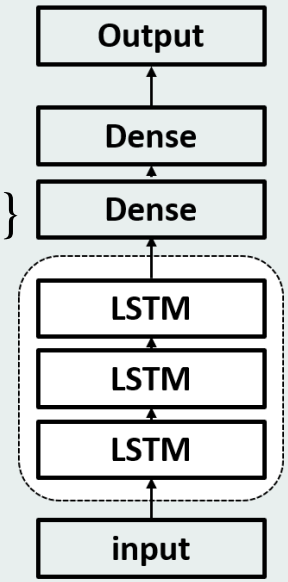


Figure: Model architecture f_θ

Implementation of Recurrent Neural Network

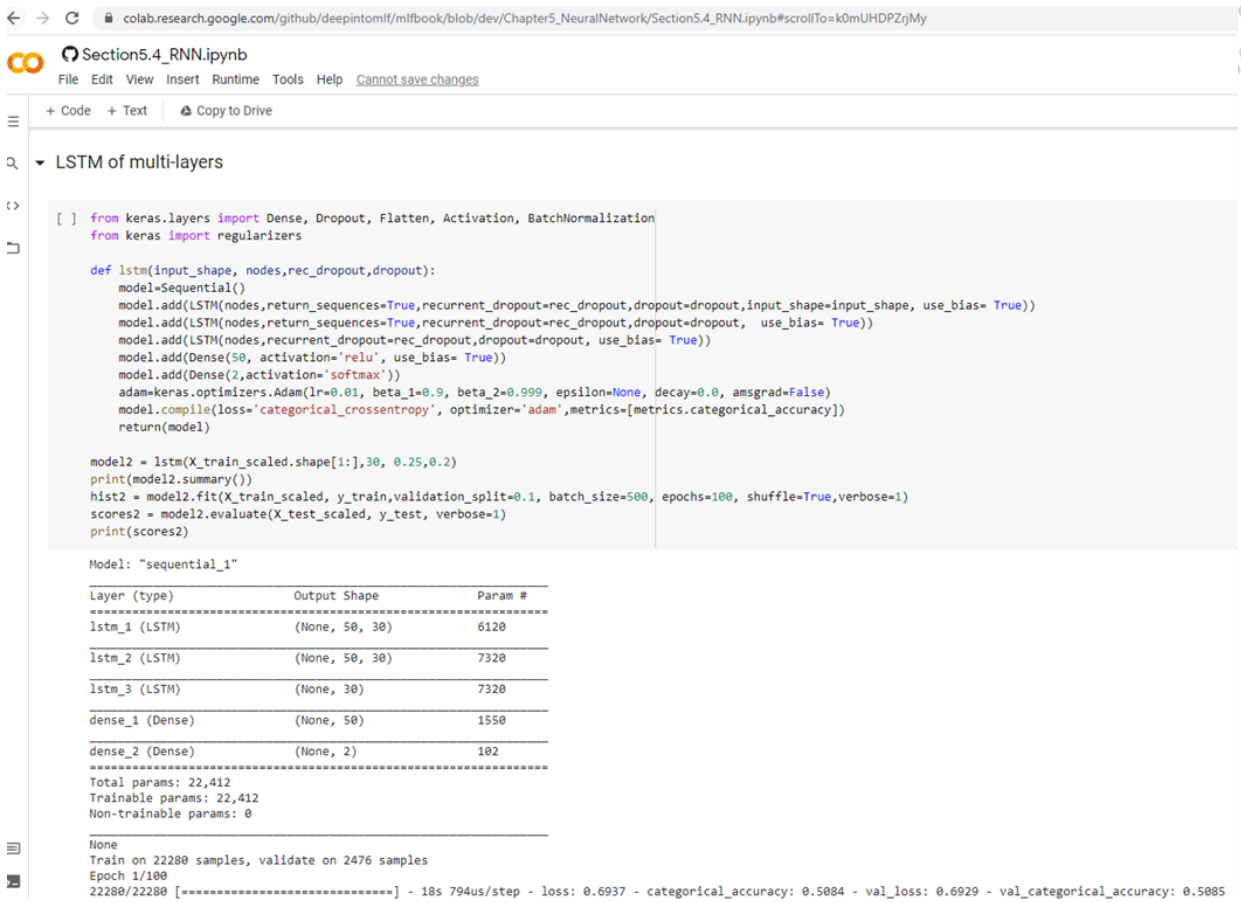


Figure: The snapshot of the Python notebook to showcase how to apply the Recurrent Neural Network to this prediction task step by step. Interested readers can download this notebook as Section5.4_RNN.ipynb under the folder Chapter5_NeuralNetwork/ at the GitHub repository `https://github.com/deepintomlf/mlfbook.git`.

Numerical Results

	accuracy	precision	recall	f1 score
One-Layer LSTM	56.32%	54.12%	76.46%	62.11%
Three-Layer LSTM	57.57%	55.24%	75.27%	62.64%

Table: The performance of the single-layer LSTM model and the three -layer LSTM model for predicting the next price direction in the test data.

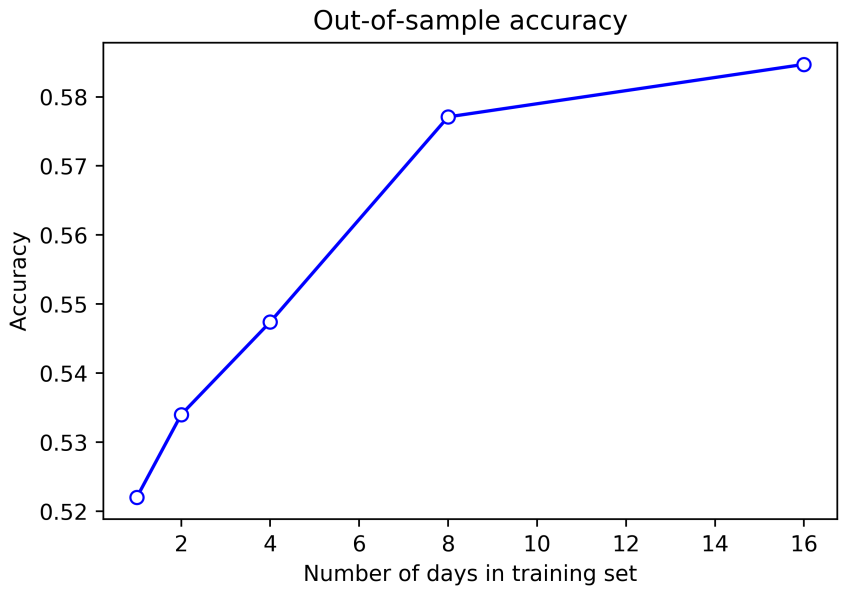


Figure: The plot of the out-of-sample accuracy v.s. the number of days in the training set.

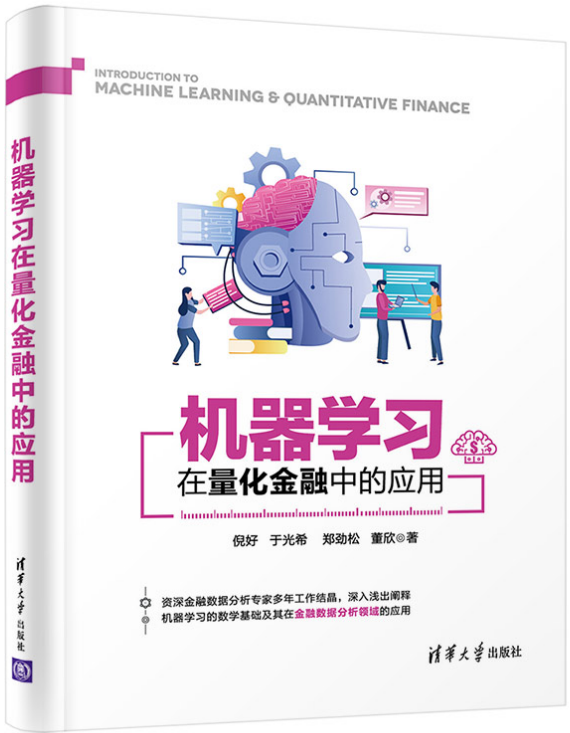
Conclusion

Takeaway Messages

- The general framework of Supervised Learning include data, model, loss function, optimization, prediction, and validation.
- Regularization methods, e.g. Lasso or Ridge regression, can help with the overfitting issues.
- Linear models and Neural networks including shallow/deep neural network and recurrent neural network are discussed.
- A financial application of neural network showcase the pipeline of ML projects.

This tutorial is based on my joint book entitled *"An introduction to machine learning in quantitative finance"* jointed with X. Dong, J. Zheng and G. Yu. It provides a systematic and rigorous introduction to machine learning and includes ample examples of financial applications. The GitHub repository for the supplementary python codes of the book is public available at https://github.com/deepintomlf/mlfbook.git.

We hope that this book can help the beginners in ML and quantitative finance to get a quick start and enjoy their journey of applying ML to financial problems!



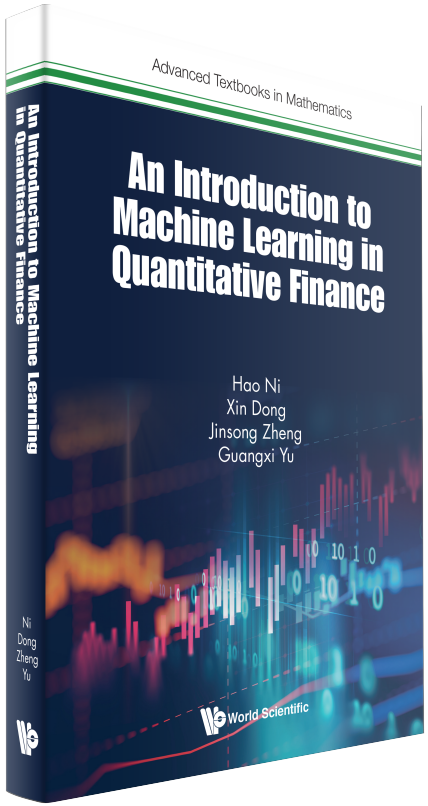


Figure: (Left) Chinese version: Tsinghua University Press [5]; (Right) English version: World Scientific Press [6].

Thanks for your attention!

Now it is time for the Q&A session.

References I


Mohammed AlQuraishi.
Alphafold at casp13.
Bioinformatics, 35(22):4862–4865, 2019.


G Gybenko.
Approximation by superposition of sigmoidal functions.
Mathematics of Control, Signals and Systems, 2(4):303–314, 1989.


Yann LeCun, Yoshua Bengio, and Geoffrey Hinton.
Deep learning.
nature, 521(7553):436, 2015.


Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller.
Playing atari with deep reinforcement learning.
arXiv preprint arXiv:1312.5602, 2013.

References II

Hao Ni, Xin Dong, Jinsong Zheng, and Guangxi Yu.
An Introduction to Machine Learning in Quantitative Finance (Chinese version).
Tsinghua University Press, 2021.

Hao Ni, Xin Dong, Jinsong Zheng, and Guangxi Yu.
An Introduction to Machine Learning in Quantitative Finance (English version).
World Scientific, 2021.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al.
Mastering the game of go with deep neural networks and tree search.
nature, 529(7587):484–489, 2016.

References III


Justin Sirignano and Rama Cont.
Universal features of price formation in financial markets: perspectives from deep learning.
Quantitative Finance, 19(9):1449–1459, 2019.