A.I. notes

Cooper

1 Linear Regression

- **Hypothesis** function

$$y = h(x) = \theta_0 + \theta_1 x$$
$$y = h(x) = \boldsymbol{\theta} \cdot \boldsymbol{x}$$

- **Loss function**

$$J(\theta_0.\theta_1) = \frac{1}{2}\frac{1}{m}\sum_{i=1}^{m}\Big(h(x^{(i)}) - y_i\Big)^2 = \frac{1}{2}\frac{1}{m}\sum_{i=1}^{m}\Big(\theta_0 + \theta_1 x^{(i)} - y_i\Big)^2$$
$$J(\theta_0, \theta_1, \ldots, \theta_N) = \frac{1}{2}\frac{1}{m}\sum_{i=1}^{m}\Big(\theta_0 + \sum_{k=1}^{N}\theta_k x_k^{(i)} - y_i\Big)^2$$

- **Calculate the Gradient** gradient is the direction along which the loss function increases the most

$$\nabla_\theta J_{(0)} = \begin{pmatrix} \frac{\partial J}{\partial \theta_0} \\ \\ \frac{\partial J}{\partial \theta_1} \end{pmatrix}_{(0)} \qquad \nabla_\theta J_{(0)} = \begin{pmatrix} \frac{\partial J}{\partial \theta_0} \\ \frac{\partial J}{\partial \theta_1} \\ \\ \vdots \\ \frac{\partial J}{\partial \theta_N} \end{pmatrix}_{(0)}$$

- **Update**

$$\boldsymbol{\theta}_{(1)} = \boldsymbol{\theta}_{(0)} - \alpha \nabla_\theta J_{(0)}$$
$$\boldsymbol{\theta}_{(n+1)} = \boldsymbol{\theta}_n - \alpha \frac{1}{m}\mathbf{X}^T(\mathbf{X}\boldsymbol{\theta}_n - \boldsymbol{y})$$

- **Repeat**.

$$\boldsymbol{\theta}_{(n+1)} = \boldsymbol{\theta}_{(n)} - \alpha \nabla_\theta J_n \qquad n = 0, 1, 2, \ldots$$
$$\text{Stop if} \quad \big|J_{(n+1)} - J_n\big| \leq \text{tol} \quad \text{or} \quad n + 1 \geq \text{ max}$$

1.1 Regularisation

A good machine learning model should have a good *bias-variance trade-off*. This can be achieved by minimising the generalisation error, which is the sum of the squared bias and the variance.

$$J(\theta_0.\theta_1) = \frac{1}{2}\frac{1}{m}\sum_{i=1}^{m}\Big(h(x^{(i)}) - y_i\Big)^2 + \lambda\sum_{j=1}^{N}\theta_j^2$$

If λ is small, large values of $\sum_{j=1}^{N}\theta_j^2$ are allowed, meaning that the model contains too many parameters, thus it can **overfit**. If λ is large, only small values of $\sum_{j=1}^{N}\theta_j^2$ are allowed. This means that the training will reduce the number of parameters, this it might underfit.

2 Logistic Regression

- **Hypothesis function**, models the probability that a given set of features belongs to class

$$h(\mathbf{x}) = \sigma(\theta \cdot \mathbf{x}) = \frac{1}{1 + exp(-\theta \cdot x)}$$

- **Loss function** (cross-entropy), entropy measures uncertainty on the possible values of a prediction

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\Big\{y_i ln\Big[h(x^{(i)})\Big] + (1-y_i) ln\Big[1-h(x^{(i)})\Big]\Big\}$$

- **Gradient descent** for logistic Regression:

  Choose a starting point ⇒ Calculate hte gradient ⇒ Update and repeat.

- If encounter **multi-class**, perform the same as many times as the number of categories of logistic regressions, each of which splits the data into two classes. End up with the same number of Hypothesis functions, select the highest probability.

- Metrics for classification

		Ground Truth		
		1	**0**	
Model Prediction	**1**	True Positive	False Positive	**Precision**
	0	False Negative	True Negative	
		Recall		

2.1 Supporting Vector Machines (SVMs)

In SVMs, we want to compute the optimal linear decision boundary, which is the **hyperplane** with the largest margin between the two classes.

- Support hyperplanes (optimal hyperplane between two points)

$$\mathbf{w}\cdot\mathbf{x}+b=1$$
$$\mathbf{w}\cdot\mathbf{x}+b=-1$$

  How is the hyperplane with largest margin oriented can be found by solving an optimisation problem

$$\text{Find } \mathbf{w}, b \text{ such that } \frac{2}{\|w\|} \text{ is maximum}$$
$$\text{or } \min_{(w,b)} \frac{1}{2}\|w\|^2 \quad \text{subject to} \quad y_i(\mathbf{w}\cdot\mathbf{x}^{(i)}+b)\geq 1 \quad i=1,2,\ldots,m$$

2.1.1 Rgularisation

if the data is not linearly separable, we need to introduce regularisation to allow some misclassified data points by introducing a slack variable $\varepsilon_i > 0$ such that: (i) $\varepsilon_i = 0$ for correctly classified (ii) $\varepsilon_i = \big|y_i - f(\mathbf{x}^{(i)})\big|$ otherwise.

$$\min_{(w,b,\varepsilon)} \frac{1}{2}\|w\|^2 + C\sum_{i}^{m}\varepsilon_i \quad \text{subject to} \quad y_i(\mathbf{w}\cdot\mathbf{x}^{(i)}+b)\geq 1-\varepsilon_i \quad \varepsilon_i \geq 0$$

small C - soft (wide margin) $\quad C\to\infty$ - hard margin

2.1.2 Kernel Trick

For nonlinearly separable data, one option is to enrich the data with nonlinearities. This can be achieved by mapping the data into a nonlinear higher-dimensional space, $\mathbf{x}\to\phi(\mathbf{x})$, where ϕ is a nonlinear map, for example, a polynomial $(x_1^{(i)}, x_2^{(i)}) \to (x_1^{(i)}, x_2^{(i)}, x_1^{(i)2} + x_2^{(i)2})$. Intuitively, by adding dimensions in the feature space, it is more likely that the data becomes linearly separated.

Kernel Trick decomposes the weight vector in the transformed feature space

$$\mathbf{w} = \sum_{j=1}^{m}\beta_j \boldsymbol{\phi}(\boldsymbol{x}^{(j)})$$

where β_j become the new weights to find. In other words, we express the weight vector with coordinates defined in the transformed feature space.

$$f(\boldsymbol{x}^{(i)}) = \sum_{j=1}^{m} \beta_j \boldsymbol{\phi}(\boldsymbol{x}^{(j)}) \cdot \boldsymbol{\phi}(\boldsymbol{x}^{(i)}) + b$$

we resort to a theorem of functional analysis, which ensures that there exists a function, K, such that

$$K(\boldsymbol{x}^{(j)} \cdot \boldsymbol{x}^{(i)}) = \boldsymbol{\phi}(\boldsymbol{x}^{(j)}) \cdot \boldsymbol{\phi}(\boldsymbol{x}^(i))$$

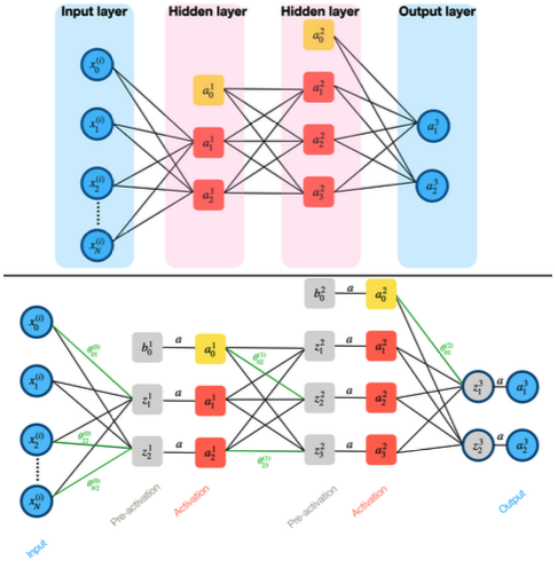
where K is the *kernel*. K suggested that we DO NOT need to increase the dimension of the feature space to include nonlinearities, we only need to apply the kernel to inner products in the original feature space. essentially finding optimal separating hyperplane without calculating anything about $\boldsymbol{\phi(x^{(i)})}$.

3 Feedforward Neural Network

Neural Network generalise the principle of linearly combining simple nonlinear functions

$$y \approx h(x : \theta)$$

function that maps input data "x" to output "y", with the function parameters represented by "θ".



3.1 Mathematical expression

- Input layer → First hidden layer

$$x = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{pmatrix} \Rightarrow \quad \theta^{(0)} = \begin{pmatrix} \theta^{(0)}_{0,1} & \theta^{(0)}_{0,2} \\ \theta^{(0)}_{1,1} & \theta^{(0)}_{1,2} \\ \vdots & \\ \theta^{(0)}_{N,1} & \theta^{(0)}_{N,2} \end{pmatrix}, \quad z^1 = \begin{pmatrix} b^1_0 \\ \theta^{(0)T} x \end{pmatrix}, \quad a^1 = a\{z^1\}$$

  where z is the *pre-activation*

- Second hidden layer

$$\theta^{(1)} = \begin{pmatrix} \theta^{(1)}_{0,1} & \theta^{(1)}_{0,2} & \theta^{(1)}_{0,3} \\ \theta^{(1)}_{1,1} & \theta^{(1)}_{1,2} & \theta^{(1)}_{1,3} \\ \theta^{(1)}_{2,1} & \theta^{(1)}_{2,2} & \theta^{(1)}_{2,3} \end{pmatrix}, \quad z^2 = \begin{pmatrix} b^2_0 \\ \theta^{(2)T} a^1 \end{pmatrix}, \quad a^2 = a\{z^2\}$$

- Output layer

$$\theta^{(1)} = \begin{pmatrix} \theta^{(2)}_{0,1} & \theta^{(2)}_{0,2} \\ \theta^{(2)}_{1,1} & \theta^{(2)}_{1,2} \\ \theta^{(2)}_{2,1} & \theta^{(2)}_{2,2} \\ \theta^{(2)}_{3,1} & \theta^{(2)}_{3,2} \end{pmatrix}, \quad z^3 = \theta^{(2)T} a^2, \quad h = a^3 = a\{z^3\}$$

  Or in generalised from

$$a^{L+1} = a\{\theta^{(L)T} a^L\}$$

3.2 Number of parameters

$$\begin{aligned} \text{n. of parameters} &= \underbrace{(s_0+1)\times s_1}_{(N+1)\times 2} + \underbrace{(s_1+1)\times s_2}_{3\times 3} + \underbrace{(s_2+1)\times s_3}_{4\times 2} = (N+1)\times 2 + 17 \\ &= \sum_{i=0}^{L} (s_i+1)\times s_{i+1} \end{aligned}$$

3.3 Training

Training dataset (60-8-%), Validation (10-20%), and test (10-20%)

Initialise ⇒ Forward propagation ⇒ Backpropagation ⇒ Update ⇒ Repeat until convergence

4 Convolution Neural Network

Drawbacks from feed-forward neural networks

- textitfully connected layers not suitable for images: contains too many parameters
- need to *flatten* 2D data into 1D array — losing spatial information.

Generic process

- input → filter (convolutional layer)→ pooling layer → (repeat) → flatten → fully-connected layer
- Numbers of neurons in a convolutional layer

$$N_d = \frac{W_d - F_d + 2P_d}{S_d} + 1$$

  where N_d is the number of neurons, W is the width of the input channel, F is the width of the filter, P is the width of the pooling, and S is the Stride number.

- Number of parameter in convolution layer l

$$l = (F_x \times F_y \times \text{n. channels} + 1 \text{ bias}) \times K$$

  where K is the number of filters (also =number of feature maps)

5 Sequence Modelling

When both the data and its order contain information to perform the task, the points in the dataset are dependent on the position of other points in the dataset, e.g., speech recognition, DNA, time series. We lose key information if we shuffle the data.

5.1 Sequence modelling with feedforward neural network

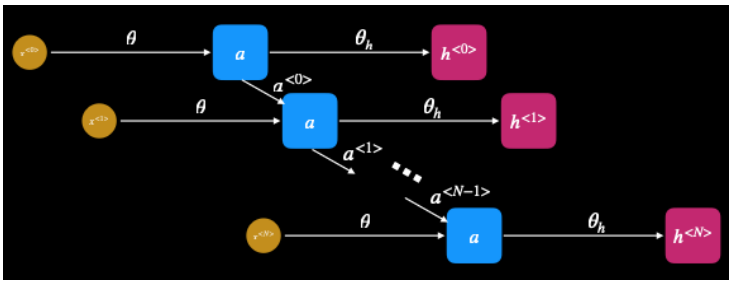
Consider time as **feature (inputs)**, and the physical variable as **label**, may be OK for short time series, but generally not a good approach due to:

- do not learn correlation as opposed to dynamical system having temporal correlations, and patterns.
- large number of parameters
- needs the inputs of a fixed size.

5.2 Recurrent cell

Store the previous time-step and feed it to the next time step as the hidden state.

$$a^{<N>} = a(\theta_a^{<N-1>} + \theta x^{<N>} + b)$$
$$h^{<N>} = \theta_h a^{<N>} + b_h$$



- The weights and biases are shared across the time-step
- **time is not a feature**
- the loss function is a sum of the loss functions of every time step by applying chain rule across different time steps.

$$J = \sum_{i=0}^{N} J^{<N>}$$
$$\frac{\partial J}{\partial a^{<i>}} = \frac{\partial J}{\partial a^{<N>}} \frac{\partial a^{<N>}}{\partial a^{<N-1>}} \cdots \frac{\partial a^{<i+1>}}{\partial a^{<i>}} = \frac{\partial J}{\partial a^{<N>}} \underbrace{\theta_a \theta_a \dots \theta_a}_{N-i \text{ times}} = \frac{\partial J}{\partial a^{<N>}} \theta_a^{N-1}$$

- Unstable gradient occurs from above equation, when θ_a is small, the gradient will become smaller and leads to **vanishing gradient** problem, make it difficult for the network to remember information from previous time steps. This can be solved by using ReLU activation functions due to its unbounded positive range.
- If θ_a is large, this leads to exploding gradient, which can be solved by clip (rescale) the gradient
- Recurrent neural network are not ideal for learning temporal correlations in long time series due to short-term memory.

5.3 Long-short time memory units LSTMs

Introducing a *cell state* C to convey (store) information from the far past – long term memory, with the combination of the same short-memory past.

- Consists of three gates to control the status of the states (forget, input, output). Each gate is controlled by a **sigmoid** activation function, due to the extreme nature of being either 1 or 0.
- **Forget gate.** This determines what percentage of the Long-term memory is remembered.
- **Candidate state** combines the shor-term memory and the input to create a **potential long-term memory** g. By using **tanh** to restrain the data in the range of -1 to 1.
- **Input gate** determines what percentage of the potential long-term memory is added to the long term memory C.

- **Output gate** calculates the potential long term memory from the cell state using a tanh function. Then, a sigmoid function is also used to determine what percentage of this potential long term memory is pass on as an output of the LSTM unit.

5.4 Gated recurrent units GRUs

- got rid of the cell state, and stored both long and short term memory in the **hidden state** and consists of 2 gates only.
- **Update gate** controls both the *input* and *forget* gate in determining what to forget and what to add on.
- **Reset gate** decides how much past information to forget.
- **Candidate state** contains past information.
- GRUs are simpler and with less parameters, making them easier to train than LSTMs

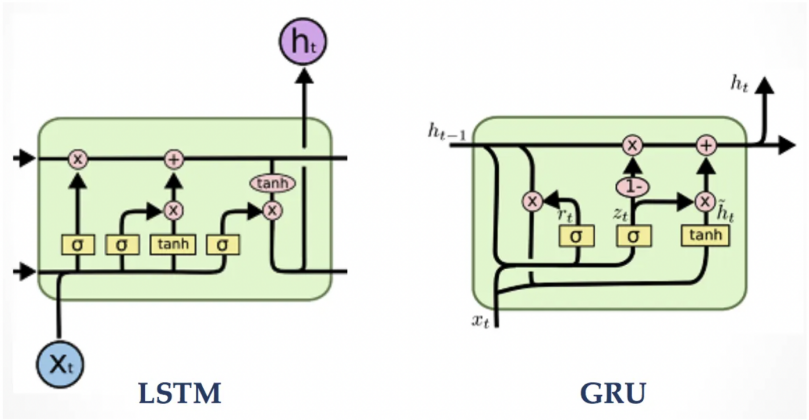


image by www.cc.gatech.edu

6 Unsupervised Learning