

CPU9444_Final1_yann

Autor: Yann

latest: June 2024

Reference:

Artificial Intelligence - A Morden Approach

University of Melbourne - COMP30024 Lecture slides

USYD - COMP5329 Lecture slides

UNSW - COMP9414 Lecture slides

UNSW - COMP9814 Lecture slides

Linear Perceptron Model

Linear Regression

Simple Linear Regression

Multivariant Linear Regression

Training Linear Model

Linear Regression Limitation

Perceptron

Step Function

Train Perceptron

Perceptron Limitation

Multilayer Perceptron

What is MLP?

Structure

Activation Function

*Softmax & Cross Entropy

Training NN

Feedforward (前馈)

Back-propagation (反向传播)

Gradient Descent (梯度下降)

Types of GDs

Chain Rule

MLP Inference

Linear & Non-Linear Stacking

Universal Approximation Theorem

Why Deep, not Fat or Wide Learning?

Vanishing Gradient Problem

Optimisation

Problems in Optimisation?

Saddle Points (鞍点)

Learning Rate Size (学习率大小)

Feature Sparsity (特征稀疏性)

Optimiser

Momentum (动量法)

Adagrad

Adadelta

RMSprop

Adam

Exercise

Ex 1

Ex 2 (Different with 9414)

Ex 3

Ex 4

Ex 5

Ex 6

Ex 7

Ex 8

Ex 9

Ex 10

Ex 11

Ex 12

Linear Perceptron Model

Linear Regression

Simple Linear Regression

单元线性回归是一种基本的回归模型，用于描述两个变量之间的线性关系。它可以用以下公式表示：

$$y = wx + b$$

假设我们要进行房价预测，可以使用单元线性模型，通过引入线性假设，我们假设房价与面积之间存在线性关系，即面积越大，房价越高，并且这种关系可以通过一条直线来近似表示。在这个模型中：

- y 是因变量（响应变量），它是我们想要预测的目标，在我们的例子中是房价。
- x 是自变量（解释变量），它是我们用来预测房价的特征，在我们的例子中是房屋的面积。
- w 是斜率（slope），表示自变量变化一个单位时，因变量的变化量。它也被称为权重（weight）。斜率 w 表示每单位面积的增加对房价的影响。如果 w 很大，说明面积对房价的影响很大；如果 w 很小，说明面积对房价的影响很小。例如，如果 $w = 2000$ ，那么面积每增加一平方单位，房价增加2000元。
- b 是截距（intercept），表示当自变量为零时，因变量的值。它也被称为偏置项（bias）。截距 b 表示当面积为零时的房价。在实际中，面积不可能为零，但截距 b 提供了一个基础价格，即使房子很小也会有一个起始价格。例如，如果 $b = 50000$ ，说明即使面积为零，房价也有50000元。这可以解释为基本建设成本或其他基础费用。

在现实中，房价不仅仅受面积影响，还可能受很多其他因素影响，比如卧室数量、距离市中心的距离等。为了建模这些复杂关系，我们需要从单元线性模型扩展到多元线性模型。

Multivariant Linear Regression

多元线性回归预测值是输入特征的线性组合。这意味着输出（或响应变量）是输入（或解释变量）的加权和。线性模型的基本形式可以表示为：

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_n x_n + \epsilon$$

其中：

- y 是响应变量或输出。
- x_1, x_2, \dots, x_n 是解释变量或输入特征。
- $\beta_0, \beta_1, \dots, \beta_n$ 是模型的系数或权重。
- ϵ 是随机误差项（在严格的线性统计模型中，误差项的分布通常假设为正态分布）。

为了简化表示，我们可以使用矩阵和向量形式，包含所有输入特征以及截距项，这样可以利用计算机进行更快速的运算：

$$Y = X\beta + \epsilon$$

其中：

- Y 是 $n \times 1$ 的响应变量向量。

$$\mathbf{Y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

- X 是 $n \times (p + 1)$ 的设计矩阵，其中 p 是解释变量的数量。矩阵 X 的第一列通常为全1，用于表示截距项（bias），其余列为输入特征。

$$\mathbf{X} = \begin{bmatrix} 1 & x_{11} & x_{12} & \dots & x_{1p} \\ 1 & x_{21} & x_{22} & \dots & x_{2p} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & x_{n2} & \dots & x_{np} \end{bmatrix}$$

- β 是 $(p + 1) \times 1$ 的系数向量。

$$\beta = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \vdots \\ \beta_p \end{bmatrix}$$

- ϵ 是 $n \times 1$ 的误差向量。

$$\epsilon = \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_n \end{bmatrix}$$

Training Linear Model

训练多元线性模型的目标是找到最佳的系数向量 β ，使得预测值 $\hat{\mathbf{Y}} = \mathbf{X}\beta$ 尽可能接近真实值 \mathbf{Y} 。最常用的方法是最小化均方误差（Mean Squared Error, MSE），即：

$$\min_{\beta} \|\mathbf{Y} - \mathbf{X}\beta\|^2$$

其中， $\|\mathbf{Y} - \mathbf{X}\beta\|^2$ 表示预测值和真实值之间的平方误差和。通过求解正规方程，可以得到系数向量的最优解：

$$\beta = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{Y}$$

其中， \mathbf{X}^T 是设计矩阵 \mathbf{X} 的转置， $(\mathbf{X}^T \mathbf{X})^{-1}$ 是 $\mathbf{X}^T \mathbf{X}$ 的逆矩阵。

Linear Regression Limitation

但在许多现实世界的情况下，因变量和自变量之间的关系可能是非线性的。例如，考虑房价问题，不仅仅是面积影响房价，房价还可能受到诸如房龄、房屋位置等多种因素的复杂非线性影响。线性模型无法捕捉这些复杂的非线性关系，导致模型的表达能力不足，不能准确地进行预测。

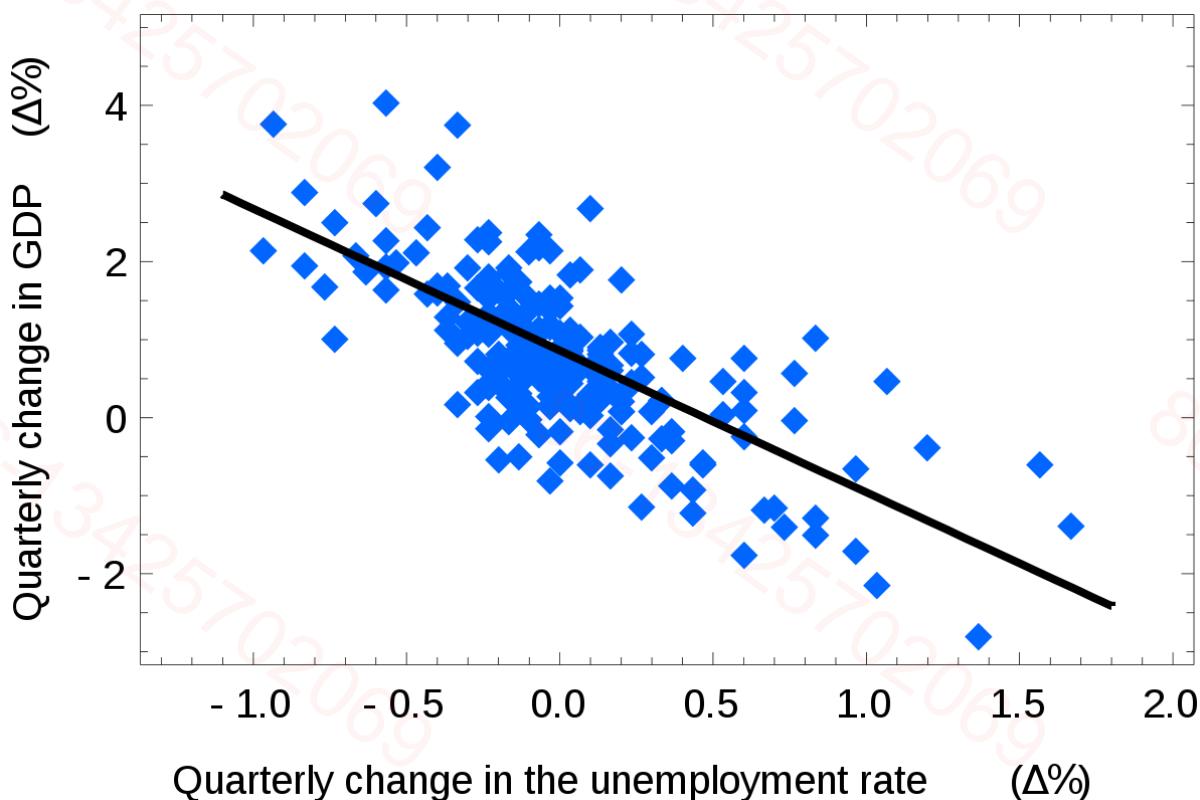
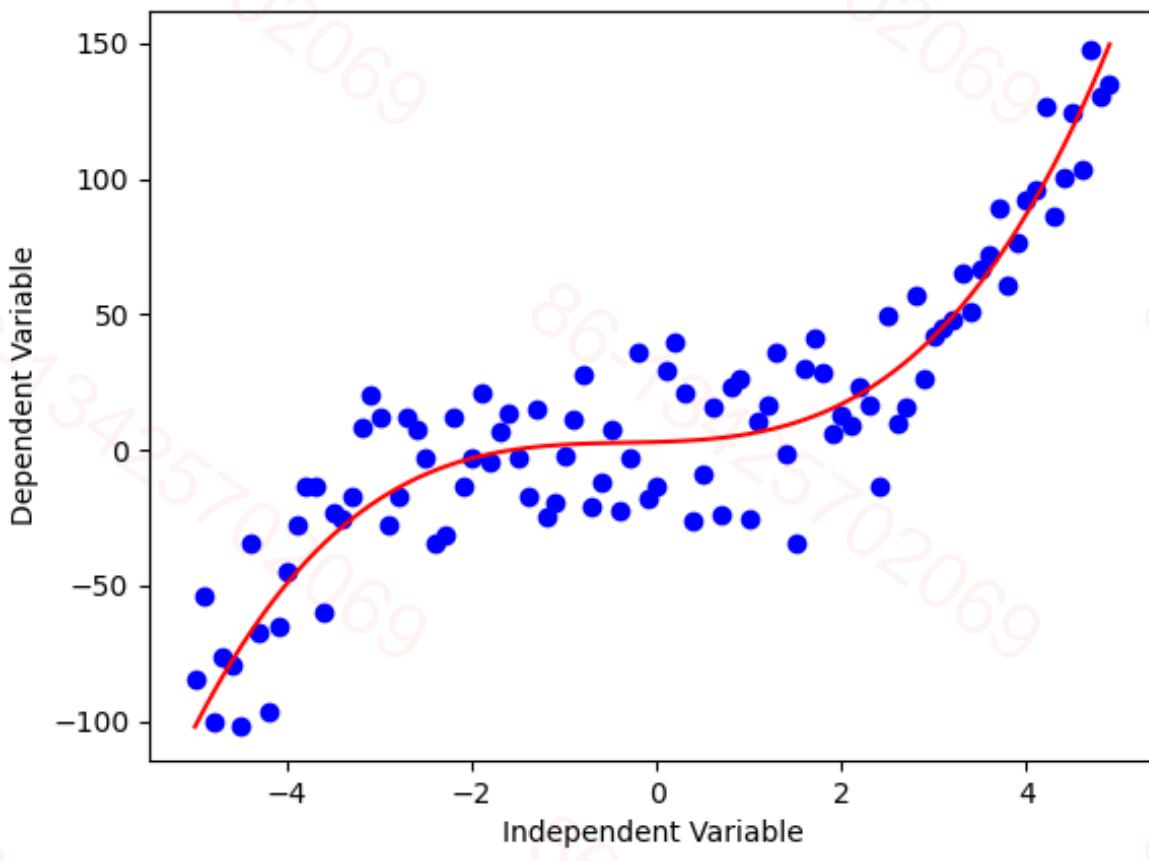


Image credit to Wikipedia

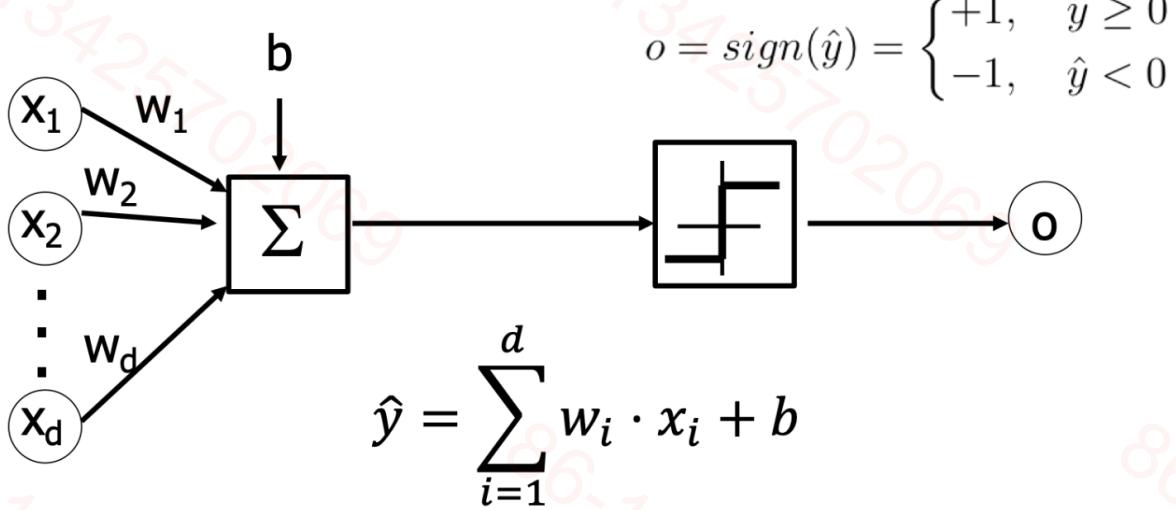
当模型的复杂度不足以捕捉数据中的模式时，就会产生高偏差问题（high bias）。线性模型由于其简单的形式，容易对非线性数据产生高偏差，表现为模型无法很好地拟合训练数据，预测结果误差较大。

在实际问题中，各个特征变量之间可能存在复杂的交互作用。例如，房屋的面积和房屋的年限可能共同影响房价，而不仅仅是各自独立地影响。线性模型在处理这些交互作用时，可能无法捕捉到这种复杂性。此外，当特征变量之间存在多重共线性时，线性模型的稳定性和预测精度也会受到影响。



Perceptron

[Rosenblatt, et al. 1958]



$$o = \text{sign}(\hat{y}) = \begin{cases} +1, & \hat{y} \geq 0 \\ -1, & \hat{y} < 0 \end{cases}$$

感知器是最简单的神经网络模型之一，用于二分类任务。它本质上是一个线性模型，通过一个阶跃函数（step function）将连续的输入映射到离散的输出。感知器可以被认为是一个单层神经网络的基本单位，它只能解决线性可分的问题。一个单层感知器的模型可以表示为：

$$y = f(W \cdot X + b)$$

其中：

- X 是输入特征向量， $X = [x_1, x_2, \dots, x_n]$ 。
- W 是权重向量， $W = [w_1, w_2, \dots, w_n]$ 。
- b 是偏置项 (bias)。
- \cdot 表示向量点积。
- f 是激活函数，这里使用阶跃函数 (step function)。

Step Function

阶跃函数 f 将输入的线性组合映射到二分类的输出。定义为：

$$f(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{if } z < 0 \end{cases}$$

其中， $z = W \cdot X + b$ 。

Train Perceptron

If an example can be correctly classified, we have

$$y(w^T x + b) \geq 0$$

otherwise

$$y(w^T x + b) < 0$$

The objective function of Perceptron can be written as

$$\min_{w,b} L(w, b) = - \sum_{x_i \in \mathcal{M}} y_i (w^T x_i + b)$$

where \mathcal{M} stands for the set of the misclassified examples.

感知器的训练过程使用感知器学习算法，目标是减少分类错误的数量，这是一种迭代的过程，通过调整权重和偏置项，使得模型能够正确分类训练数据。训练步骤如下：

1. **初始化权重和偏置**：将权重向量 W 和偏置 b 初始化为零或小的随机值。
2. **前向传播**：计算预测值 $\hat{y} = f(W \cdot X + b)$ 。
3. **计算误差**：对于每个训练样本，计算预测值 \hat{y} 与真实标签 y 之间的误差。
4. **更新权重和偏置**：根据误差调整权重和偏置，使模型的预测更加准确。更新规则为：

$$w_j := w_j + \eta(y - \hat{y})x_j$$

$$b := b + \eta(y - \hat{y})$$

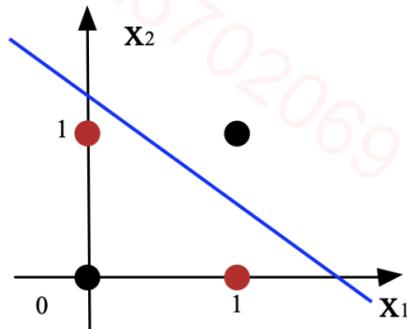
其中， η 是学习率，控制每次更新的步长。

1. **重复迭代**：重复步骤2到4，直到误差收敛或达到预设的迭代次数。

Perceptron Limitation

- Solving XOR (exclusive-or) with Perceptron?

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0



- Linear classifier cannot identify non-linear patterns.

$$w_1x_1 + w_2x_2 + b$$

This failure caused the majority of researchers to walk away.

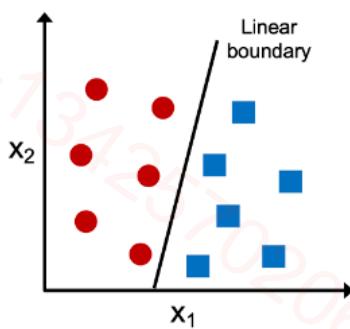
The University of Sydney

Page 9

USYD COMP5329 Lecture slides - Week1

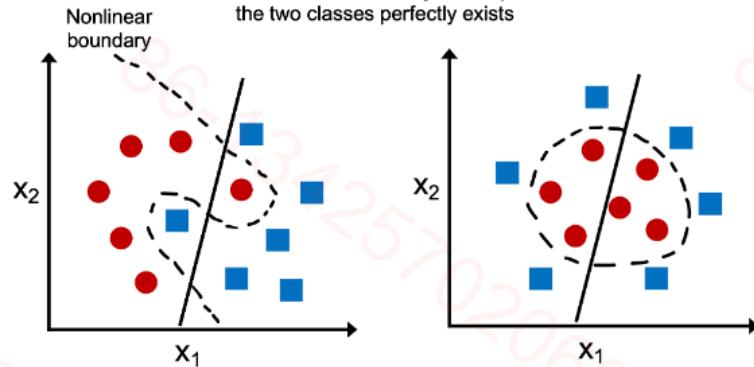
Linearly separable

A linear decision boundary that separates the two classes exists



Not linearly separable

No linear decision boundary that separates the two classes perfectly exists



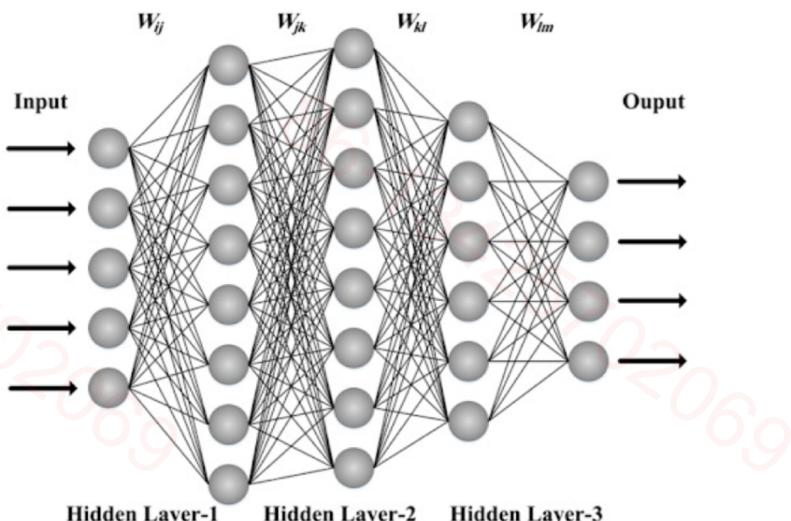
vitalflux

感知器和linear regression一样都有线性假设，所以它只能解决线性可分的分类问题。也就是说，当数据集中的正负样本可以用一条直线（在二维情况下）或一个超平面（在高维情况下）分开时，感知器才能正确分类。如果数据集是线性不可分的，感知器将无法找到合适的分类边界。

Multilayer Perceptron

What is MLP?

多层感知机（MLP）是神经网络中的一种基本结构，常被称为前馈神经网络（Feedforward Neural Network）或全连接网络（Fully Connected Network）。为了处理线性不可分的问题，可以引入多层神经网络结构，通过非线性激活函数，可以捕捉更复杂的数据模式和关系。MLP 是多个线性模型和非线性激活函数的叠加，通过层层传递信息来实现复杂的非线性映射。



- Function composition can be described by a directed acyclic graph (hence feedforward networks)
- Depth is the maximum i in the function composition chain
- Final layer is called the output layer

Structure

MLP 由一个输入层、一个或多个隐藏层和一个输出层组成。每一层中的每个神经元都与上一层的所有神经元全连接。以下是 MLP 的结构和数学定义：

- **输入层**：接受输入特征向量为 $\mathbf{X} = [x_1, x_2, \dots, x_n]$ 。对于第一个隐藏层中的第 j 个神经元，其输出 $h_j^{(1)}$ 可以表示为：

$$h_j^{(1)} = f \left(\sum_{i=1}^n w_{ij}^{(1)} x_i + b_j^{(1)} \right)$$

其中， $w_{ij}^{(1)}$ 是输入层到第一个隐藏层的权重。 $b_j^{(1)}$ 是第一个隐藏层的偏置。 f 是激活函数（如 Sigmoid、ReLU 等）。

- **隐藏层**：由一个或多个隐藏层组成，每个隐藏层中的神经元应用线性变换和非线性激活函数。对于第 l 个隐藏层中的第 j 个神经元，其输出 $h_j^{(l)}$ 可以表示为：

$$h_j^{(l)} = f \left(\sum_{i=1}^{n_{l-1}} w_{ij}^{(l)} h_i^{(l-1)} + b_j^{(l)} \right)$$

其中， $w_{ij}^{(l)}$ 是第 $(l - 1)$ 层到第 l 层的权重。 $b_j^{(l)}$ 是第 l 层的偏置。 n_{l-1} 是第 $(l - 1)$ 层的神经元数量。

- **输出层**：输出最终的预测结果。对于输出层中的第 k 个神经元，其输出 y_k 可以表示为：

$$y_k = f \left(\sum_{j=1}^{n_L} w_{jk}^{(L)} h_j^{(L-1)} + b_k^{(L)} \right)$$

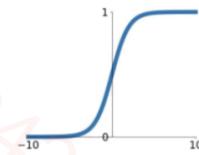
其中， $w_{jk}^{(L)}$ 是最后一个隐藏层到输出层的权重。 $b_k^{(L)}$ 是输出层的偏置。 n_L 是最后一个隐藏层的神经元数量。

Activation Function

- Popular choice of activation functions (single input)

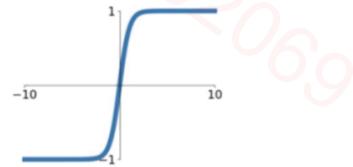
- Sigmoid function

$$f(s) = \frac{1}{1 + e^{-s}}$$



- Tanh function: shift the center of Sigmoid to the origin

$$f(s) = \frac{e^s - e^{-s}}{e^s + e^{-s}}$$

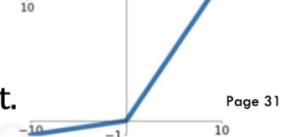


- Rectified linear unit (ReLU)

$$f(s) = \max(0, s)$$

- Leaky ReLU

The University of Sydney $f(s) = \begin{cases} s, & \text{if } s \geq 0 \\ \alpha s, & \text{if } s < 0, \alpha \text{ is a small constant.} \end{cases}$



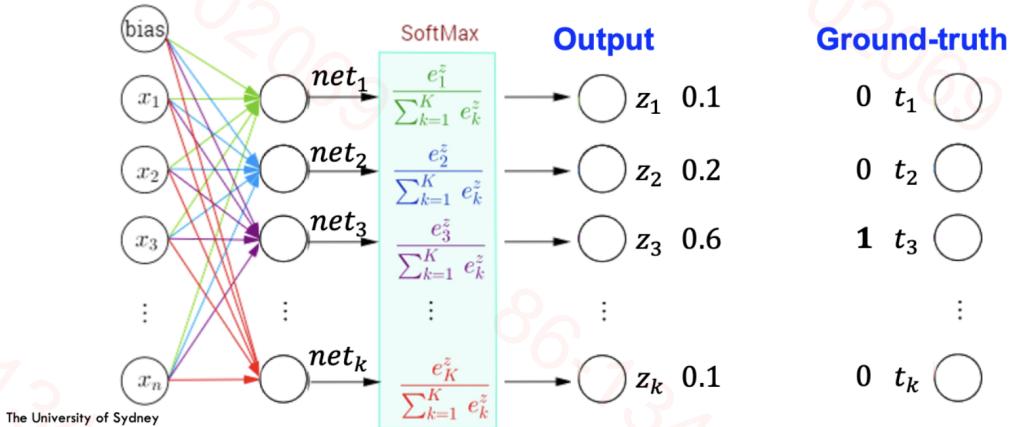
USYD COMP5329 Lecture slides - Week2

激活函数 f 引入了非线性，使 MLP 能够处理复杂的非线性问题。激活函数 (Activation Function) 在神经网络中起到了关键的作用，它定义了一个给定输入或一组输入的输出。基本上，激活函数决定了一个神经元是否应该被激活。

*Softmax & Cross Entropy

- In multi-class classification, **softmax function** before the output layer is to assign conditional probabilities (given x) to each one of the K classes.

$$\hat{P}(\text{class}_k|x) = z_k = \frac{e^{net_k}}{\sum_{i=1}^K e^{net_i}}$$



Page 38

USYD COMP5329 Lecture slides - Week2

Softmax 函数是一种特殊的激活函数，通常用于神经网络的输出层，特别是在多分类任务中。其目的是将输入向量转换为概率分布。Softmax 函数将输入的未归一化的得分（称为 logits）转换为归一化的概率值，使得输出向量的所有元素之和为 1。

Softmax 函数通常与交叉熵损失一起使用，以训练多分类模型。**概率分布**：Softmax 函数的输出是一个概率分布。输出向量中的每个元素都是非负的，并且所有元素之和为 1。**归一化**：Softmax 函数将输入的 logits 归一化，使得它们可以解释为类别的概率。**放大对比度**：Softmax 函数通过指数运算放大了输入向量的差异，使得较大的 logit 对应的概率值更大，较小的 logit 对应的概率值更小，从而提高了模型的判别力。

在多分类任务中，Softmax 函数通常用作神经网络的输出层。假设我们有一个分类问题，需要将输入样本分为 K 类。在网络的最后一层，我们得到一个大小为 K 的向量 z ，这个向量包含了每个类别的 logit 分数。通过应用 Softmax 函数，我们可以将这些 logits 转换为每个类别的概率分布。Softmax 函数通常与交叉熵损失一起使用，以优化模型的参数。交叉熵损失度量了预测的概率分布与真实标签的概率分布之间的差异。

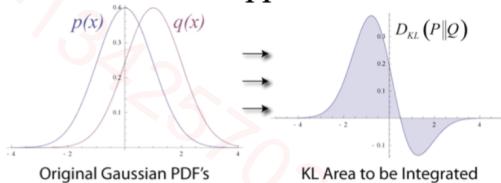
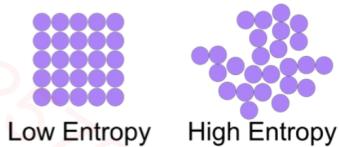
给定一个真实标签向量 $y = [y_1, y_2, \dots, y_K]$ 和一个预测概率向量 $\hat{y} = [\hat{y}_1, \hat{y}_2, \dots, \hat{y}_K]$ ，交叉熵损失定义为：

$$L = - \sum_{i=1}^K y_i \log(\hat{y}_i)$$

- Given two probability distributions t and z ,

$$\begin{aligned} \text{CrossEntropy}(t, z) &= - \sum_i t_i \log z_i \\ &= - \sum_i t_i \log t_i + \sum_i t_i \log t_i - \sum_i t_i \log z_i \\ &= - \sum_i t_i \log t_i + \sum_i t_i \log \frac{t_i}{z_i} \\ &= \text{Entropy}(t) + D_{KL}(t|z) \end{aligned}$$

Entropy is a measure of degree of disorder or randomness.



USYD COMP5329 Lecture slides - Week2

交叉熵损失 (Cross-Entropy Loss) 是度量两个概率分布之间差异的常用方法。它通常用于分类任务中，用于衡量模型预测的概率分布与真实概率分布之间的差异。交叉熵损失越小，表示模型预测的概率分布与真实概率分布越接近。交叉熵可以分解为熵 $H(P)$ 和 KL 散度 $D_{KL}(P|Q)$ 的和。

其中：

- $H(P)$ 是真实分布 P 的熵，表示分布 P 的固有不确定性。
- $D_{KL}(P|Q)$ 用于度量两个概率分布之间的差异，是一种非对称的度量。是分布 P 和分布 Q 之间的 KL 散度，表示分布 Q 对分布 P 的近似程度。

交叉熵用于度量模型预测的概率分布与真实概率分布之间的差异，常用于分类任务中。可以分解为真实分布的熵和两个分布之间的 KL 散度之和。通过最小化交叉熵损失，可以使模型预测的概率分布更接近真实分布。

Training NN

前馈算损失，后馈算梯度。训练神经网络的过程主要分为两个阶段：前馈 (Feedforward) 和反向传播 (Backpropagation)。前馈用于计算损失，反向传播用于计算梯度并更新网络参数。

Feedforward (前馈)

前馈是神经网络从输入层到输出层的正向计算过程。在前馈过程中，输入数据经过每一层的加权求和和激活函数处理，最终生成网络的输出或预测。

Back-propagation (反向传播)

反向传播是神经网络训练中的关键步骤，用于计算损失函数相对于网络参数（权重和偏置）的梯度，并使用这些梯度来更新参数。神经网络的目标是最小化损失函数。为了做到这一点，我们需要知道每个参数是如何影响损失的，这样我们就可以相应地调整它。反向传播提供了一种有效的方法来计算损失函数相对于网络参数的梯度。通过这些梯度，我们可以知道每个参数如何影响损失，并相应地调整参数，以最小化损失函数。这是神经网络训练的核心，使得模型能够不断优化其预测能力。

Gradient Descent (梯度下降)

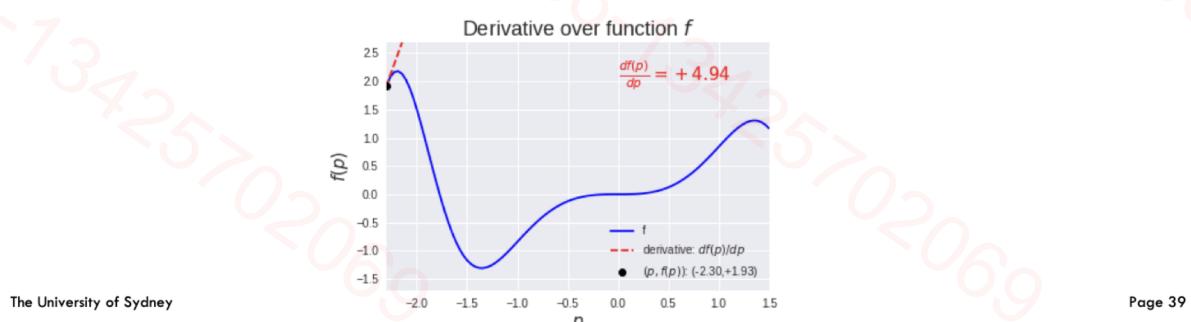
- Weights are initialized with random values, and then are updated in a direction reducing the error.

$$\Delta \mathbf{w} = -\eta \frac{\partial J}{\partial \mathbf{w}}$$

where η is the learning rate.

Iteratively update

$$\mathbf{w}(t+1) = \mathbf{w}(t) + \Delta \mathbf{w}(t)$$



USYD COMP5329 Lecture slides - Week2

梯度下降是一种常用的优化算法，用于调整模型的参数（例如神经网络中的权重和偏置），以最小化目标函数（通常是损失函数）。其核心思想是通过计算目标函数的梯度，沿着梯度的反方向更新参数，逐步逼近目标函数的最小值。

可以将梯度下降类比为一个人在山上行走，目标是找到最快的方式走到山谷的最低点。这个人没有地图，所以他决定在每一步都选择使自己下降最快的方向。这个方向就是当前位置的“梯度”。通过多次迭代这个过程，他最终会到达山谷的底部，这就是损失函数的最小值。

假设我们有一个目标函数 $L(\theta)$ ，其中 θ 表示模型的参数。梯度下降的基本步骤如下：

1. **计算梯度**：计算目标函数 $L(\theta)$ 相对于参数 θ 的梯度，即 $\nabla_{\theta} L(\theta)$ 。
2. **更新参数**：沿着梯度的反方向更新参数。更新公式为：

$$\theta := \theta - \eta \nabla_{\theta} L(\theta)$$

其中， η 是学习率 (learning rate)，控制每次更新的步长。

Types of GDs

- **Batch gradient descent:**

Compute the gradient for the **entire** training dataset.

$$\theta = \theta - \eta \cdot \nabla_{\theta} J(\theta, \mathcal{X}^{(1:end)})$$

- **Stochastic gradient descent:**

Compute the gradient for **each** training example.

$$\theta = \theta - \eta \cdot \nabla_{\theta} J(\theta, \mathcal{X}^{(i)})$$

- **Mini-batch gradient descent:**

Compute the gradient for every **mini-batch** training examples.

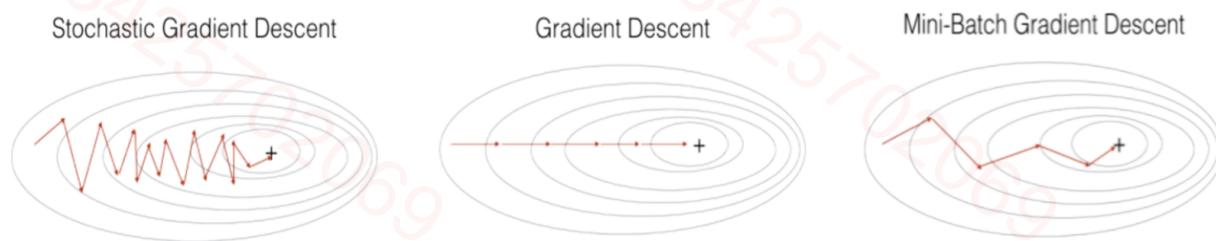
$$\theta = \theta - \eta \cdot \nabla_{\theta} J(\theta, \mathcal{X}^{(i:i+n)})$$

1. **批量梯度下降 (Batch Gradient Descent)**：每次使用整个训练集计算梯度并更新参数。优点是更新方向准确，但计算开销大，尤其是当训练集很大时。

2. 随机梯度下降 (**Stochastic Gradient Descent, SGD**) : 每次使用一个训练样本计算梯度并更新参数。优点是计算速度快，但更新方向噪声较大，可能导致震荡。

3. 小批量梯度下降 (**Mini-Batch Gradient Descent**) : 每次使用一个小批量的训练样本计算梯度并更新参数，结合了批量梯度下降和随机梯度下降的优点。

- Divide the training set into mini-batches.
- In each epoch, randomly permute mini-batches and take a mini-batch sequentially to approximate the gradient
 - One epoch is usually defined to be one complete run through all of the training data.
 - The estimated gradient at each iteration is more reliable.



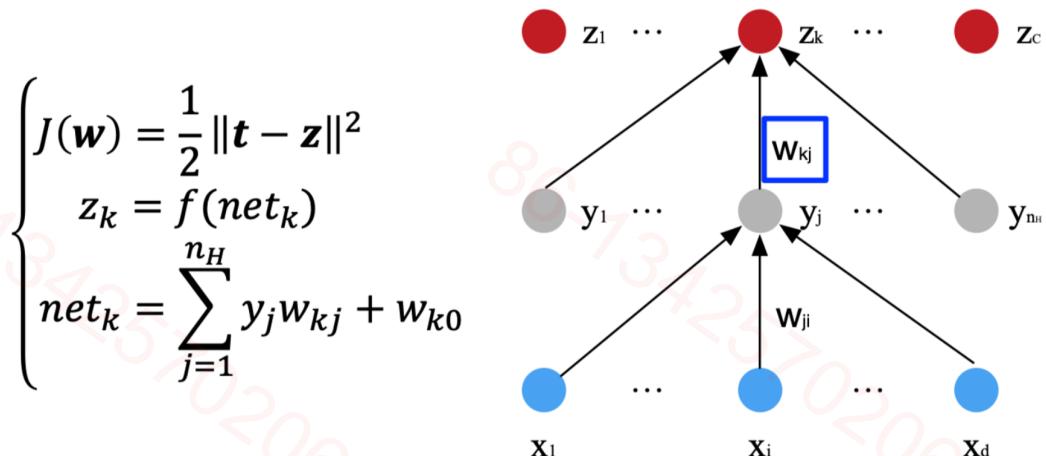
- One-example based SGD
 - Estimation of the gradient is noisy, and the weights may not move precisely down the gradient at each iteration
 - Faster than batch learning, especially when training data has redundancy
 - Noise often results in better solutions
 - The weights fluctuate and it may not fully converge to a local minimum
- Min-batch based SGD
 - Conditions of convergence are well understood
 - Some acceleration techniques only operate in batch learning
 - Theoretical analysis of the weight dynamics and convergence rates are simpler

- One-example based SGD
 - Estimation of the gradient is noisy, and the weights may not move precisely down the gradient at each iteration
 - Faster than batch learning, especially when training data has redundancy
 - Noise often results in better solutions
 - The weights fluctuate and it may not fully converge to a local minimum
- Min-batch based SGD
 - Conditions of convergence are well understood
 - Some acceleration techniques only operate in batch learning
 - Theoretical analysis of the weight dynamics and convergence rates are simpler

USYD COMP5329 Lecture slides - Week2

Chain Rule

$$\frac{\partial J}{\partial w_{kj}} = \frac{\partial J}{\partial \text{net}_k} \frac{\partial \text{net}_k}{\partial w_{kj}} = \frac{\partial J}{\partial \text{net}_k} y_j$$



The University of Sydney

Page 40

USYD COMP5329 Lecture slides - Week2

链式法则是微积分中的一个重要工具，用于求复合函数的导数。在梯度下降和反向传播算法中，链式法则被广泛应用于计算损失函数相对于神经网络各层参数的梯度。在神经网络中，前向传播计算每一层的输出，反向传播计算损失函数相对于每一层参数的梯度。反向传播过程中，链式法则用于逐层计算梯度。

MLP Inference

Linear & Non-Linear Stacking

在神经网络中，线性函数的堆叠并不能增加模型的复杂性和表达能力，因此必须引入非线性函数来实现复杂的非线性映射。线性函数的堆叠不会增加模型的复杂性，因为堆叠的结果仍然是一个线性变换。这可以通过矩阵代数来证明。然而，通过在线性变换后引入非线性函数（如 ReLU、Sigmoid 等），我们可以打破这种线性限制，使模型能够表示复杂的非线性关系。这也是多层感知机（MLP）能够处理复杂问题的核心原理。让我们从数学上来解释这一点。

设 $g(x)$ 是一个线性函数，其形式为：

$$g(x) = Ax + b$$

其中， A 是矩阵， x 是输入向量， b 是偏置向量。如果我们堆叠两个线性函数 g ，通过展开和合并项，我们可以得到：

$$g(g(x)) = A_2(A_1x + b_1) + b_2 = A_2A_1x + A_2b_1 + b_2$$

这仍然是一个线性变换。从数学上看，两个线性变换的堆叠本质上等价于一个新的线性变换，因此线性函数的堆叠并不会增加模型的复杂性。

设 $f(x)$ 是一个非线性函数，其形式可以是 Sigmoid 函数、ReLU 函数等。例如，ReLU 函数定义为：

$$f(x) = \max(0, x)$$

如果我们将非线性函数 f 应用于线性变换 $g(x)$ ，则有：

$$f(g(x)) = f(Ax + b)$$

由于 f 是非线性的，它能够引入非线性的复杂性，使得模型能够表示更复杂的函数关系。这也是神经网络能够处理复杂问题的关键。

Universal Approximation Theorem

Let $f(\cdot)$ be a **nonconstant, bounded, and monotonically-increasing continuous** function. Let I_m denote the m -dimensional unit hypercube $[0,1]^m$. The space of continuous functions on I_m is denoted by $C(I_m)$. Then, given any $\varepsilon > 0$ and any function $f \in C(I_m)$, **there exists an integer N** , real constants $v_i, b_i \in \mathbb{R}^m$ and real vectors $w_i \in \mathbb{R}^m$, where $i = 1, \dots, N$, such that we may define:

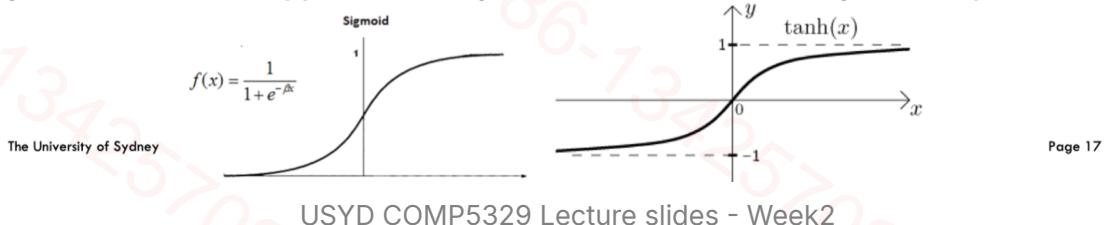
$$\hat{F}(x) = \sum_i^N v_i f(w_i^T x + b_i)$$

as an approximate realization of the function F where F is independent of f ; that is,

$$|\hat{F}(x) - F(x)| < \varepsilon$$

for all $x \in I_m$. In other words, functions of the form $\hat{F}(x)$ are dense in $C(I_m)$.

Universality theorem (Hecht-Nielsen 1989): “Neural networks with a single hidden layer can be used to approximate any continuous function to any desired precision.”



Universal Approximation Theorem 是神经网络理论中的一个重要定理。它表明，一个前馈神经网络 (feedforward neural network)，只要具有足够的隐藏单元并使用非线性激活函数，就可以以任意的精度逼近任何定义在有限闭区间上的连续函数。

网络结构：该定理通常适用于具有单隐藏层的神经网络（也称为单层前馈神经网络，Single-Layer Feedforward Network）。虽然定理主要关注单隐藏层，但实际上，多层神经网络 (deep neural networks) 也具有类似的逼近能力。**非线性激活函数：**网络必须使用非线性激活函数，如 Sigmoid、Tanh 或 ReLU。如果使用线性激活函数，无论网络有多少神经元，它都只能表示线性函数。非线性激活函数引入了非线性变换，使得神经网络可以表示和学习复杂的非线性关系。**任意精度：**定理表明，只要隐藏层中的神经元数量足够多，网络可以以任意精度逼近任何连续函数。这意味着在理论上，网络可以无限接近目标函数，但在实践中，为了达到特定的精度，可能需要大量的神经元。**连续函数：**定理适用于目标函数是连续的情况。对于非连续函数，该定理不直接适用，但在实际应用中，神经网络通常也能很好地逼近这些函数。

神经网络的非线性特性主要来源于激活函数。因此，激活函数也被称为非线性单元。只有当我们在计算中引入非线性函数时，输出结果才会展现出更强的非线性特点。与此相反，线性函数的计算过程称为线性变换 (linear transformation)。这意味着，当我们把多个线性操作串联起来时，无论串联多少次，最终的输出仍然是线性的。因此，仅仅通过增加网络的深度而不引入非线性，是不能带来任何新的功能或提高模型

的表达能力的。这也是为什么在设计神经网络时，强调非线性激活函数的重要性，它确保了网络可以捕捉到复杂的、非线性的数据模式。

Why Deep, not Fat or Wide Learning?

"深度学习"这个术语中的"深度"主要是指神经网络模型的层数。一个深度学习模型通常由多层的神经网络组成，每一层包含许多神经元。这些层的深度结构使得深度学习模型能够表示和学习复杂的非线性关系。深度学习模型通常由多层神经网络组成。例如，一个深度学习模型可能有数十、数百甚至更多的层。这些层可以分为输入层、隐藏层和输出层。隐藏层的增加使模型能够捕捉到更多层次的特征。

每一层都可以捕捉到数据不同层次的特征。例如，在图像识别中：

- **初级层**：可能捕捉到简单的边缘和纹理。
- **中间层**：可能捕捉到部分和简单的形状。
- **高级层**：可能捕捉到更复杂的结构和对象。

宽度 vs 深度：增加神经网络的宽度（即每层的神经元数量）确实可以增加模型的容量，但研究表明，增加模型的深度更能有效地增加其表示能力。与其使用一个非常宽的浅层网络，不如使用一个深层网络来实现更复杂的特征提取和表示。深层网络通常比浅层但更宽的网络更为**参数高效**。这是因为深层结构可以通过组合低级特征来学习高级特征，这种分层的特征学习在许多任务中都是非常有效的。

Vanishing Gradient Problem

深层网络在理论上有更高的表示能力，但它们也更难训练。幸运的是，随着优化技术、正则化方法和硬件的进步（如 GPU、TPU 的发展），我们现在可以有效地训练非常深的网络。在深度神经网络中，随着网络层数的增加，**反向传播过程中梯度可能会逐层消失**。这意味着在更新前面层的参数时，梯度可能非常小，导致这些层的参数几乎不更新，使得训练深度神经网络变得困难。为了解决梯度消失问题，何凯明提出了残差网络（ResNet）的概念，使用了残差连接（skip connection）。残差连接允许梯度在反向传播过程中直接跨越若干层，从而缓解梯度消失问题。

1. **Vanishing Gradient Problem**：在使用 Sigmoid 和 Tanh 激活函数时，梯度在反向传播过程中可能会变得非常小，导致梯度消失问题，使得训练变得困难。
2. **ReLU 函数**：ReLU 函数通过在正区间保持梯度为 1，避免了梯度消失问题，但可能会导致 Dead ReLU 问题。

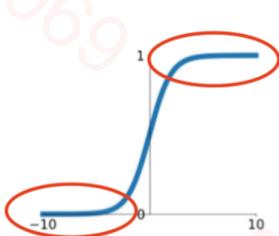
3. **Dead ReLU Problem**：指的是在训练过程中，部分神经元的输出恒为零，导致这些神经元无法更新。

4. **Leaky ReLU**：通过引入一个小小的斜率，Leaky ReLU 确保所有神经元都能得到更新，避免了 Dead ReLU 问题。

- **Saturation.**

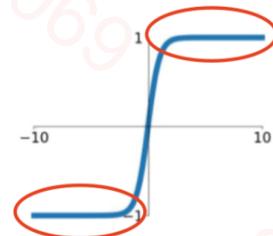
Sigmoid function

$$f(s) = \frac{1}{1 + e^{-s}}$$



Tanh function

$$f(s) = \frac{e^s - e^{-s}}{e^s + e^{-s}}$$



- ❖ At their extremes, their derivatives are close to 0, which kills gradient and learning process

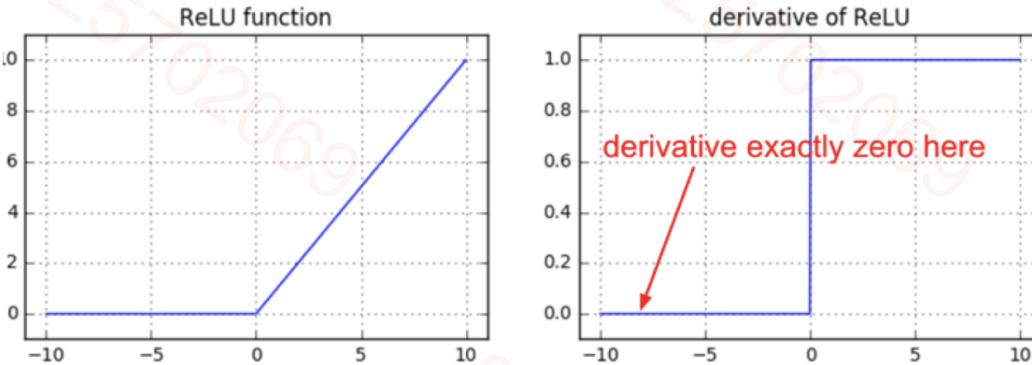
USYD COMP5329 Lecture slides - Week2

在深度神经网络中，梯度消失问题（Vanishing Gradient Problem）是训练过程中常见的难题，尤其在使用 Sigmoid 和 Tanh 激活函数时。这个问题会导致模型训练速度变慢甚至无法收敛。在 x 远离 0 时，Sigmoid 函数的梯度趋近于 0，这导致在反向传播时，梯度不断减小，甚至消失，使得前面的层无法有效更新参数。类似的问题出现在 Tanh 函数在 x 远离 0 时，其梯度也会变得很小，导致梯度消失问题。

Dead ReLUs

ReLU function

$$f(s) = \max(0, s)$$



- ❖ For negative numbers, ReLU gives 0, which means some part of neurons will not be activated and be dead.
- ❖ Avoid large learning rate and wrong weight initialization, and try Leaky ReLU, etc.

The University of Sydney

Page 33

USYD COMP5329 Lecture slides - Week2

ReLU (Rectified Linear Unit) 优势在于其导数为 1 (当 $x > 0$ 时) , 这意味着它不会在正区间内出现梯度消失的问题，使得训练过程更加高效。然而，ReLU 也存在一些问题，如 Dead ReLU 问题。Dead ReLU 问题指的是在训练过程中，ReLU 函数的输入在某些神经元上可能总是小于等于零，导致这些神经元的输出恒为零，从而使得这些神经元不再更新参数。这些神经元被称为“死神经元”。

Leaky ReLU 是对 ReLU 函数的一种改进，它通过引入一个小的斜率来避免 Dead ReLU 问题。Leaky ReLU 通过引入一个小的斜率，Leaky ReLU 确保了所有神经元都能够得到更新，从而避免了 Dead ReLU 问题。

Optimisation

Problems in Optimisation?

在神经网络优化过程中，可能会遇到多个问题，这些问题会影响训练的效率和效果。以下是三种常见的优化问题及其详细解释：鞍点 (Saddle Points) 、学习率大小 (Learning Rate Size) 和特征稀疏性 (Feature Sparsity) 。

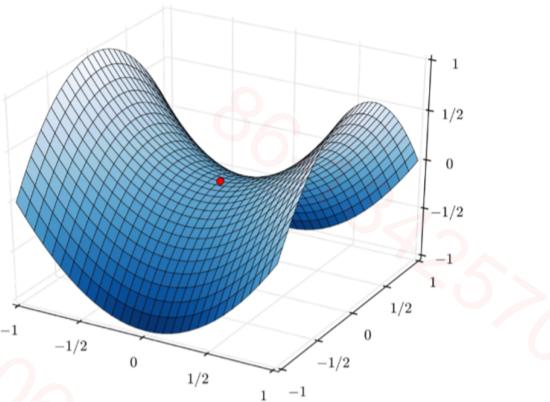
Saddle Points (鞍点)

定义：鞍点是指在损失函数的某一点上，梯度（导数）为零，但该点既不是局部最小值也不是局部最大值。

- Saddle points are points where one dimension slopes up and another slopes down. These saddle points are usually surrounded by a plateau of the same error, which makes it notoriously hard for SGD to escape, as the gradient is close to zero in all dimensions.

The University of Sydney

Page 8



USYD COMP5329 Lecture slides - Week3

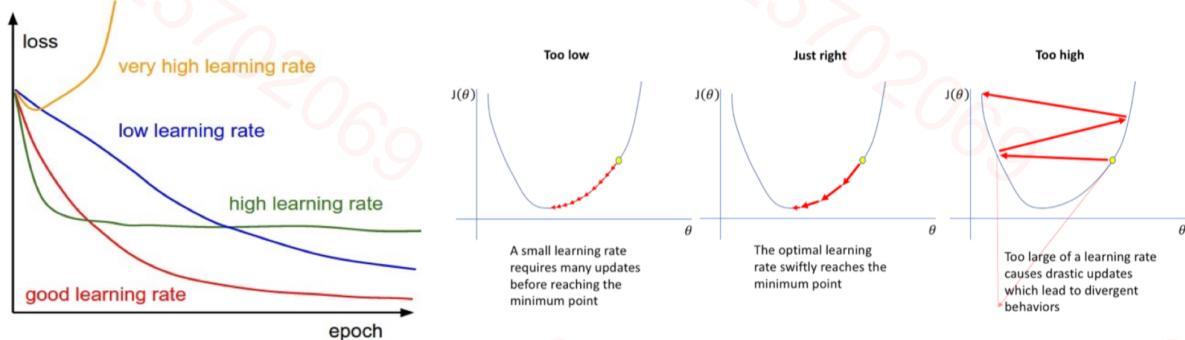
造成的影响可以总结成：

- **平坦区域：**鞍点通常被同一错误水平的平坦区域所包围。这意味着在这些区域内，梯度接近于零，导致 SGD（随机梯度下降）在这些区域内的参数更新量非常小，几乎无法从鞍点处逃脱。
- **逃逸困难：**由于梯度接近零，SGD 很难确定哪个方向是离开鞍点的最佳路径。这可能导致优化过程在鞍点附近停滞不前，从而使训练进程变得异常缓慢。

Learning Rate Size (学习率大小)

定义：选择合适的学习率是神经网络训练过程中的一个重要而又具挑战性的任务。学习率决定了在优化过程中参数更新的幅度。

- A learning rate that is too small leads to painfully slow convergence.
- A learning rate that is too large can hinder convergence and cause the loss function to fluctuate around the minimum or even to diverge.



<http://srdas.github.io/DLBook/GradientDescentTechniques.html>

The University of Sydney

Page 6

USYD COMP5329 Lecture slides - Week3

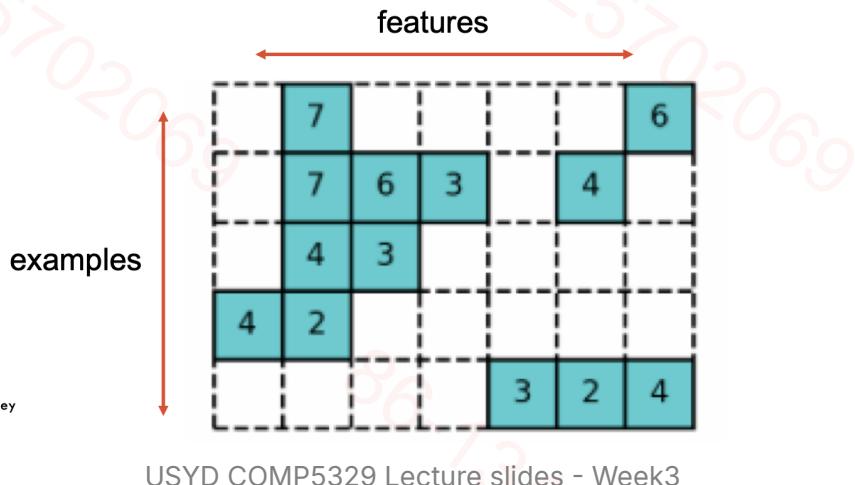
造成的影响可以总结成：

- **学习率太小：收敛速度慢**，当学习率设定得太低时，参数更新的步长非常小。这意味着网络需要更多的迭代次数来达到最优解或损失函数的最小值，导致训练过程极其缓慢，消耗大量的时间和计算资源。
- **学习率太大：阻碍收敛**，如果学习率设定得太高，每次参数更新的步长就会过大。这可能会导致优化过程在最小值附近振荡，甚至从最小值处越过去，使得损失函数无法稳定下来，影响模型的学习效果。**发散**，在极端情况下，如果学习率过大，参数更新可能会使得损失函数值不是向着减小的方向变化，而是变得更大，导致训练过程完全偏离正确的轨道，即所谓的“发散”。

Feature Sparsity (特征稀疏性)

定义：当处理稀疏特征时，模型可能会忽略罕见特征的重要性，因为这些特征更新频率较低，从而影响模型的性能。当我们见到某些特别稀少特征的时候，其实我们认为他们是比较宝贵的，应该需要更大的权重，所以更新幅度希望大一些。想象一下如果更新频率都一样，那么模型是会倾向于忽略掉这些罕见特征的。因为那些常见的特征会让参数频繁更新，从而降低罕见特征对参数的影响。

- If our data is sparse and our features have very different frequencies, we might not want to update all of them to the same extent, but perform a larger update for rarely occurring features.



所以造成的影响可以总结成：

- **忽略罕见特征**：如果更新频率都一样，模型会倾向于忽略罕见特征，因为常见特征会让参数频繁更新，从而降低罕见特征对参数的影响。
- **信息丢失**：罕见特征可能包含重要的信息，但由于其更新频率低，模型未能有效学习这些信息。

Optimiser

优化器是神经网络训练中用于更新网络参数以最小化损失函数的算法。以下是几种常见的优化器：Momentum、Adagrad、Adadelta、RMSprop 和 Adam。总结为：

- **Momentum (动量法)**：加速收敛，尤其在鞍点附近。
- **Adagrad**：自适应学习率，但可能导致学习率过小。
- **Adadelta**：改进 Adagrad，限制累积的梯度平方和。
- **RMSprop**：类似 Adadelta，限制累积的梯度平方和。
- **Adam**：结合动量法和 RMSprop 的优点，自适应调整学习率。

一些符号说明：

- θ ：模型参数

- η : 学习率
- g_t : 在时间步 t 的梯度, 即 $g_t = \nabla_{\theta} L(\theta_t)$
- m_t : 动量项
- v_t : 梯度平方的指数加权平均
- ϵ : 防止除零的小常数, 通常取 $\epsilon = 10^{-8}$
- β_1, β_2 : 指数加权平均的衰减率

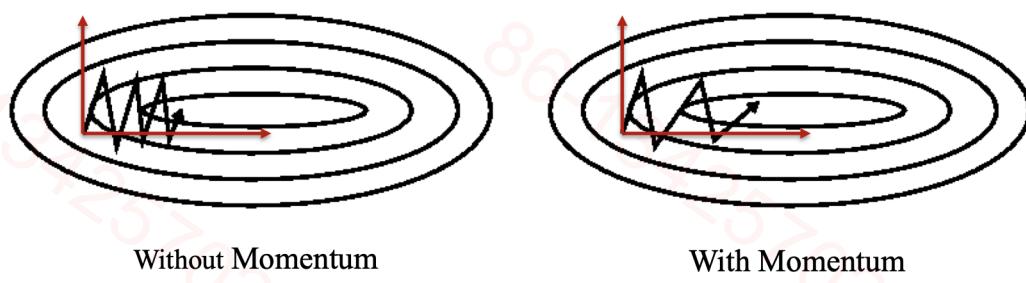
Momentum (动量法)

动量法通过累积梯度的历史信息来加速梯度下降算法, 尤其是在鞍点附近。更新公式:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

$$\theta_{t+1} = \theta_t - \eta m_t$$

SGD has trouble navigating ravines, i.e. areas where the surface curves much more steeply in one dimension than in another, which are common around local optima.



<https://www.willamette.edu/~gorr/classes/cs449/momrate.html>

Momentum tries to accelerate SGD in the relevant direction and dampens oscillations.

动量 (Momentum) 是一种用于加速神经网络训练的优化技术, 特别是在面对鞍点和减少梯度下降过程中的振荡方面非常有效。动量项通常用符号 β 表示, 其值一般设置在 0.9 左右 (可以是 0.8 到 0.99 之间, 根据具体问题进行调整)。这意味着在更新参数

时，大约90%的更新方向来自于过去累积的梯度，而10%来自于当前梯度。它的基本思想是引入了一个“动量项”，类似于物理中的动量，使参数更新不仅依赖于当前步骤的梯度，还依赖于之前步骤的梯度。这样做的结果是在相同方向连续多次移动时加速学习，而在方向频繁改变时减缓学习速度。在标准的梯度下降中，每次参数更新都是基于当前的梯度。当引入动量后，每次更新不仅考虑当前的梯度，还考虑之前的更新方向，能够带来下面好处：

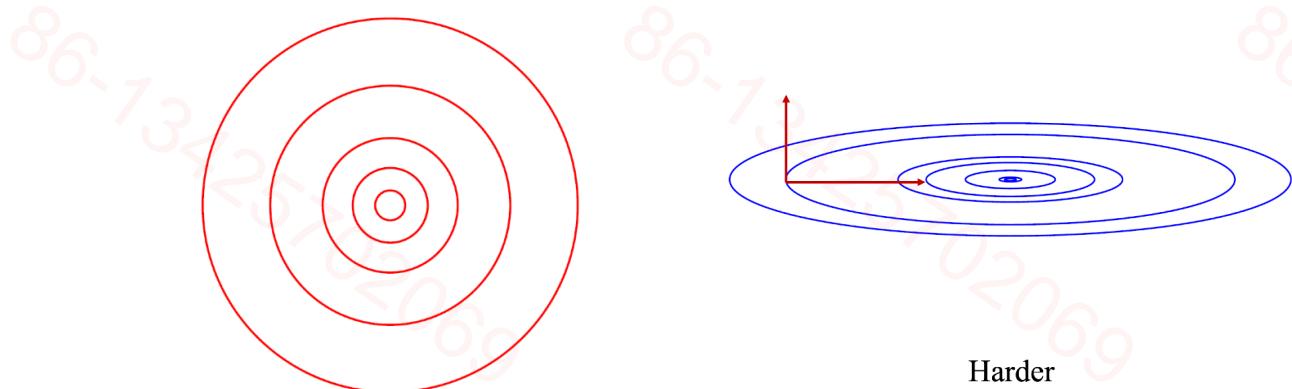
- **快速收敛**：如果连续多个梯度指向相同的方向，动量项会累积这些梯度，从而加大在该方向上的更新步长。这有助于加速收敛，特别是在损失函数的某些方向上梯度较为一致时。
- **减少震荡**：当梯度方向发生改变时，动量会因为累积了之前与当前梯度方向不同的梯度而减小更新量，减缓在方向上频繁变化的更新。这有助于减少在梯度方向变化大的区域（如峡谷边缘）的振荡，使得收敛路径更加平滑。

Adagrad

Adagrad 自适应地调整每个参数的学习率，基于每个参数的历史梯度平方和。更新公式：

$$G_t = G_{t-1} + g_t^2$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} g_t$$



Nice (all features are equally important)

USYD COMP5329 Lecture slides - Week3

Adagrad（自适应梯度算法）是一种特别适合处理稀疏数据的优化算法，它通过自适应地调整每个参数的学习率来改进模型的训练过程。Adagrad的核心思想是对频繁更

新的参数进行较小的更新，而对不频繁更新的参数进行较大的更新，从而使得所有的参数更均匀地被训练。Adagrad算法中，每个参数的学习率会根据过去梯度的累积平方进行调整。具体来说：

- **梯度累积**：对于每个参数，Adagrad维护一个累积梯度的平方和，记作 G 。这是一个对角矩阵，其中每个对角元素是该参数过去所有梯度平方和。
- **自适应学习率**：Adagrad通过将全局学习率除以 \sqrt{G} （加上一个平滑项，通常是一个很小的常数，避免除以0）来调整每个参数的学习率。这意味着对于那些过去梯度累积较大的参数，其学习率将被降低；对于那些梯度累积较小的参数，其学习率相对较高。

但是会面临一个**学习率衰减**的问题，随着时间推移，累积梯度的平方和会不断增加，导致学习率逐渐减小，最终变得非常小。这意味着在训练后期，参数更新的幅度会变得极小，可能导致学习过程提前停止。尽管Adagrad可以自适应调整每个参数的学习率，但仍需要手动设置一个全局学习率。这个全局学习率的选择可能会影响到算法的效果。

Adadelta

Adadelta 是 Adagrad 的改进版，它通过限制累积的梯度平方和，克服了学习率单调递减的问题。更新公式：

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

$$\Delta\theta_t = -\frac{\sqrt{\Delta\theta_{t-1}^2 + \epsilon}}{\sqrt{v_t + \epsilon}} g_t$$

$$\theta_{t+1} = \theta_t + \Delta\theta_t$$

Adadelta是一种自适应学习率的优化算法，旨在克服Adagrad算法中学习率不断减小直至接近于零的问题。Adadelta通过仅累积最近的梯度信息来改进Adagrad算法，具体实现方式是使用了一个固定大小的窗口来存储过去平方梯度的指数衰减平均，这样可以避免学习率的无限减小。但Adadelta算法实现不是存储所有过去梯度的平方和，而是仅仅关注最近的梯度。这通过计算过去平方梯度的指数移动平均(EMA)来实现，因此不需要手动设置一个全局学习率。这种方法使得学习率的更新基于最近的梯度变化，而不是累积所有历史信息，从而提供了一种更稳定的自适应学习率，但其实实现方式是动态衰减。

- Storing fixed previous squared gradients cannot accumulate to infinity and instead becomes a local estimate using recent gradients.
- Adadelta implements it as an exponentially decaying average of all past squared gradients.

$$E[g^2]_t = \gamma E[g^2]_{t-1} + (1 - \gamma) g_t^2$$

$$RMS[g]_t = \sqrt{E[g^2]_t + \epsilon}$$

$$\Delta\theta_t = -\frac{\eta}{RMS[g]_t} g_t$$

另外，Adadelta解决的一个问题是梯度的单位不一致。在标准的SGD和动量方法中，更新的单位是基于梯度的，而不是参数的单位。为了解决这个问题，Adadelta使用了二阶方法（如牛顿法）中的单位修正概念，通过近似海森矩阵（Hessian matrix）来调整梯度单位：

- The units in SGD and Momentum relate to the gradient, not the parameter.

$$\theta = \theta - \eta \cdot g$$

$$\text{units of } \Delta\theta \propto \text{units of } g \propto \frac{\partial J}{\partial \theta} \propto \frac{1}{\text{units of } \theta}$$

- Second order methods such as Newton's method do have the correct units.

$$\theta = \theta - H^{-1}g$$

$$\text{units of } \Delta\theta \propto H^{-1}g \propto \frac{\frac{\partial J}{\partial \theta}}{\frac{\partial^2 J}{\partial \theta^2}} \propto \text{units of } \theta$$

The University of Sydney

* Assume cost function J is unitless. Page 23

USYD COMP5329 Lecture slides - Week3

$$H^{-1}g = \Delta\theta$$

$$\Delta\theta = \frac{\frac{\partial J}{\partial \theta}}{\frac{\partial^2 J}{\partial \theta^2}} \Rightarrow H^{-1} \propto \frac{1}{\frac{\partial^2 J}{\partial \theta^2}} \propto \frac{\Delta\theta}{\frac{\partial J}{\partial \theta}} \propto \frac{\Delta\theta}{g}$$

We assume the curvature is locally smooth and approximate $\Delta\theta_t$ by computing the exponentially decaying RMS of $\Delta\theta$.

$$\Delta\theta_t = -\frac{\eta}{RMS[g]_t} g_t \longrightarrow \Delta\theta_t = -\frac{RMS[\Delta\theta]_{t-1}}{RMS[g]_t} g_t$$

$$\theta_{t+1} = \theta_t + \Delta\theta_t$$

The University of Sydney

Page 24

USYD COMP5329 Lecture slides - Week3

$$H^{-1}g = \Delta\theta$$

$$\Delta\theta = \frac{\frac{\partial J}{\partial\theta}}{\frac{\partial^2 J}{\partial\theta^2}} \Rightarrow H^{-1} \propto \frac{1}{\frac{\partial^2 J}{\partial\theta^2}} \propto \frac{\Delta\theta}{\frac{\partial J}{\partial\theta}} \propto \frac{\Delta\theta}{g}$$

We assume the curvature is locally smooth and approximate $\Delta\theta_t$ by computing the exponentially decaying RMS of $\Delta\theta$.

$$\Delta\theta_t = -\frac{\eta}{RMS[g]_t} g_t \xrightarrow{\text{red arrow}} \Delta\theta_t = -\frac{RMS[\Delta\theta]_{t-1}}{RMS[g]_t} g_t$$

$$\theta_{t+1} = \theta_t + \Delta\theta_t$$

所以总结Adagrad优点是：

- 无需手动设置学习率**：与Adagrad和其他需要手动设置学习率的算法不同，Adadelta算法的设计使其无需外部学习率设置。
- 减少学习率衰减问题**：通过限制累积的梯度信息的数量，Adadelta避免了学习率持续衰减到极小值的问题。
- 自动调整更新步长**：Adadelta通过考虑参数更新的单位来自动调整每个参数的更新步长，从而提高了优化过程的稳定性和效率。

RMSprop

RMSprop 也是对 Adagrad 的改进，通过限制累积的梯度平方和，但方式不同于 Adadelta。更新公式：

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{v_t + \epsilon}} g_t$$

Adam

Adam结合了Adagrad和RMSprop算法的优点，并在此基础上引入了偏差校正机制，以提高算法的稳定性和效率。Adam旨在通过自适应学习率为不同参数进行高效的深度学习模型训练，同时计算梯度的一阶动量和二阶动量的指数加权平均。更新公式：

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1)g_t$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2)g_t^2$$

偏差修正：

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

参数更新：

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t$$

Adam引入了偏差校正机制，以解决早期估计偏向零的问题。这主要是因为 m （梯度的均值）和 v （梯度的未平方均值）最初都被初始化为0向量。由于这种初始化，算法在初始阶段会低估这两个估计量，特别是当衰减率接近1时。

Exercise

Ex 1

- a. Construct by hand a Perceptron which correctly classifies the following data; use your knowledge of plane geometry to choose appropriate values for the weights w_0 , w_1 and w_2 .

Training Example	x_1	x_2	Class
a.	0	1	-1
b.	2	0	-1
c.	1	1	+1

Ex 2 (Different with 9414)

- b. Demonstrate the Perceptron Learning Algorithm on the above data, using a learning rate of 1.0 and initial weight values of

$$w_0 = -1.5$$

$$w_1 = 0$$

$$w_2 = 2$$

Iteration	w ₀	w ₁	w ₂	Training Example	x ₁	x ₂	Class	s=w ₀ +w ₁ x ₁ +w ₂ x ₂	Action
1	-1.5	0	2	a.	0	1	-	+0.5	Subtract
2	-2.5	0	1	b.	2	0	-	-2.5	None
3	-2.5	0	1	c.	1	1	+	-1.5	Add
4	-1.5	1	2	a.	0	1	-	+0.5	Subtract
5	-2.5	1	1	b.	2	0	-	-0.5	None
6	-2.5	1	1	c.	1	1	+	-0.5	Add
7	-1.5	2	2	a.	0	1	-	+0.5	Subtract
8	-2.5	2	1	b.	2	0	-	+1.5	Subtract
9	-3.5	0	1	c.	1	1	+	-2.5	Add
10	-2.5	1	2	a.	0	1	-	-0.5	None
11	-2.5	1	2	b.	2	0	-	-0.5	None
12	-2.5	1	2	c.	1	1	+	+0.5	None

Ex 3

(2 marks) Briefly explain the difference between Perceptron learning and backpropagation.

Ex 4

(2 marks) Briefly explain the difference between batch learning and online learning.

Ex 5

(3 marks) What would happen if the transfer functions at the hidden layers in a multi-layer perceptron would be omitted; i.e. if the activation would simply be the weighted sum of the activations at the previous layer? Explain why this (simpler) activation scheme is not normally used in MLPs although it would simplify and accelerate the calculations for the backpropagation algorithm?

Ex 6

· (3 marks) Sketch the following activation functions, and write their formulas:

- (i) sigmoid
- (ii) tanh
- (iii) ReLU

Ex 7

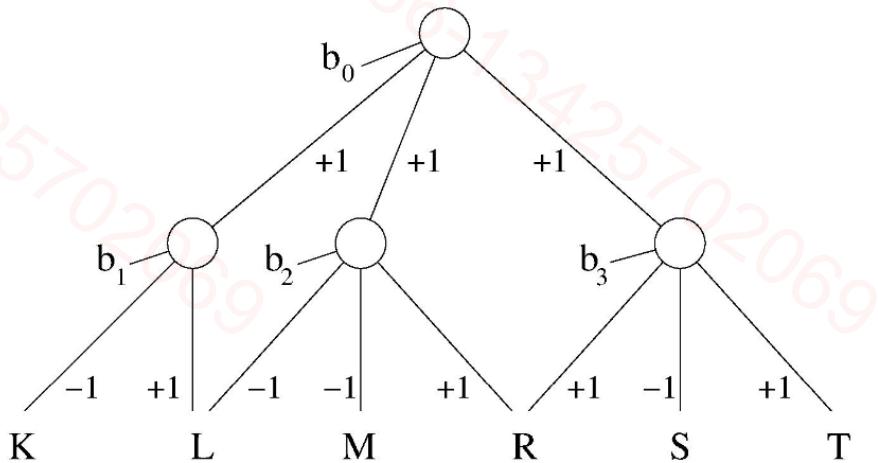
Construct by hand a Neural Network (or Multi-Layer Perceptron) that computes the XOR function of two inputs. Make sure the connections, weights and biases of your network are clearly visible.

Ex 8

Two-layer Neural Network to compute the function
 $(A \vee B) \wedge (\neg B \vee C \vee \neg D) \wedge (D \vee \neg E)$:

Ex 9

Consider the following multi-layer perceptron, using the threshold activation function, and assume that TRUE is represented by 1; FALSE by 0.



For which values of the biases b_0, b_1, b_2, b_3 would this network compute this logical function ?

$$(\neg K \vee L) \wedge (\neg L \vee \neg M \vee R) \wedge (R \vee \neg S \vee T)$$

i. value of b_0 :

ii. value of b_1 :

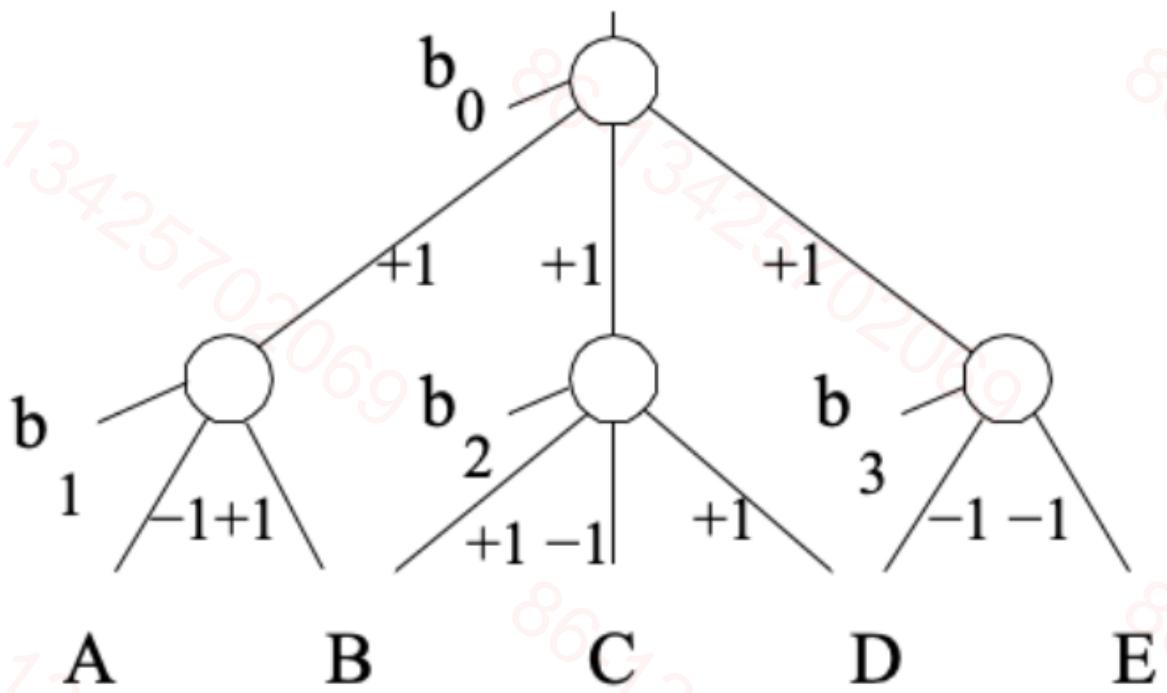
iii. value of b_2 :

iv. value of b_3 :

Ex 10

Consider the following multi-layer perceptron, with threshold activation function, and assume that TRUE is represented by 1; FALSE by 0. For which values of the biases b_0, b_1, b_2 and b_3 would this network compute the logical function?

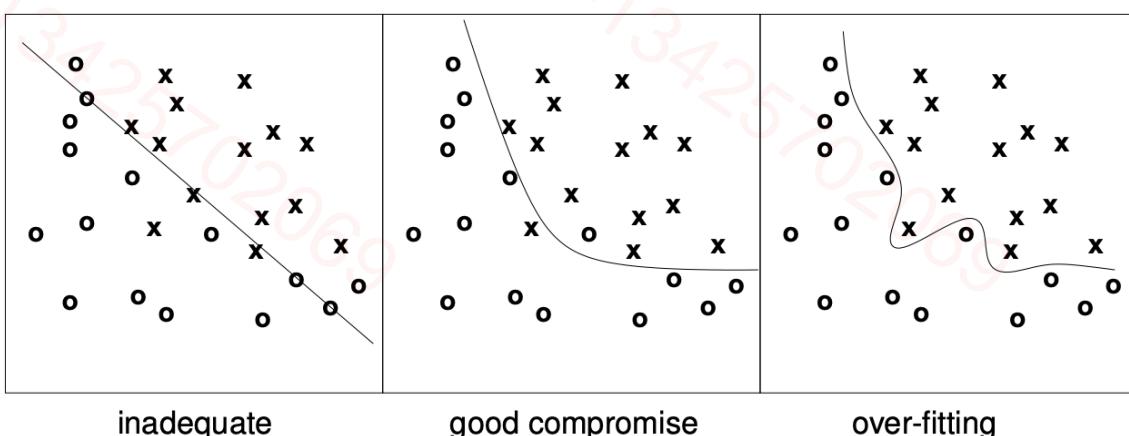
$$(\neg A \vee B) \wedge (B \vee \neg C \vee D) \wedge (\neg D \vee \neg E)$$



- $b_0 = -2.5, b_1 = +0.5, b_2 = +0.5, b_3 = +1.5$
- $b_0 = -2.5, b_1 = -0.5, b_2 = -1.5, b_3 = +0.5$
- $b_0 = -2.5, b_1 = -0.5, b_2 = +0.5, b_3 = +1.5$
- $b_0 = -0.5, b_1 = -0.5, b_2 = -1.5, b_3 = +0.5$

Ex 11

"The most likely hypothesis is the **simplest** one consistent with the data."



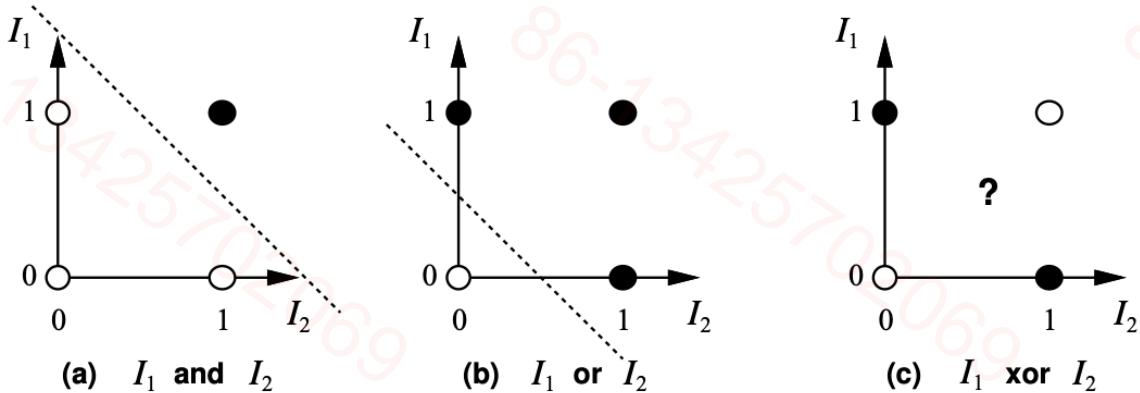
Since there can be **noise** in the measurements, in practice need to make a tradeoff between simplicity of the hypothesis and how well it fits the data.

The principle "The most likely hypothesis is the simplest one consistent with the data" is called:

- Overfitting
- Occam's Razor
- Resolution by refutation
- Entropy minimisation

Ex 12

Problem: many useful functions are not linearly separable (e.g. XOR)

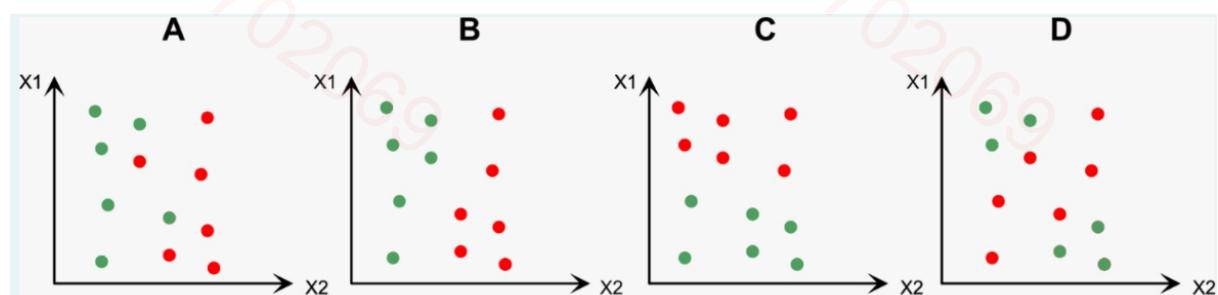


Possible solution:

$x_1 \text{ XOR } x_2$ can be written as: $(x_1 \text{ AND } x_2) \text{ NOR } (x_1 \text{ NOR } x_2)$

Recall that AND, OR and NOR can be implemented by perceptrons.

Consider the data distribution diagrams below, what case is it possible to construct a perceptron that correctly classifies all data samples that belong to either the green or red class?



- A, C
- B, C

- A
- A, B