深度学习在 EEG 数据的应用探索及实验

摘要

本文档主要介绍时间序列基础、循环神经网络模型以及其在时间序列数据上的应用，主要分为时间序列预测、时间序列分类两个方面来阐述，最后给出了神经网络方法在 EEG 数据上的应用以及相关实验分析。

刘 彦超

autuanliu@163.com

第一章 时间序列基础

1.1 时序数据基础

1.1.1 时间序列定义

在统计研究中[9]，常用按时间顺序排列的一组随机变量：

$$X_1, X_2, \cdots X_t, \cdots \tag{1.1.1}$$

来表示一个随机事件的时间序列，简记为 $\{X_t, t \in T\}$ 或者 $\{X_t\}$。用 $x_1, x_2, \cdots, x_n$ 或者 $\{x_t, t = 1, 2, \cdots, n\}$ 表示该随机序列的 n 个有序观察值，称为序列长度为 n 的观察值序列。

1.1.2 时间序列分析常见模型

时间序列一般有平稳时间序列和非平稳时间序列两种。根据限制条件的严格程度可以将平稳时间序列分为严平稳和宽平稳时间序列。所谓严平稳就是一种条件比较苛刻的平稳性定义，它认为只有当序列所有的统计特性都不会随着时间的推移而发生变化时，该序列才能被认为平稳。由于严平稳时间序列要求严格，所以其通常只存在于理论上，在实际应用中，我们使用的都是宽平稳时间序列。宽平稳只要求序列二阶平稳，对于高于二阶的矩没有任何要求。

1.1.2.1 AR 模型

时间序列的 $\mathrm{AR}(p)$ 模型可以表述为：

$$x_t = \phi_0 + \phi_1 x_{t-1} + \phi_2 x_{t-2} + \cdots + \phi_p x_{t-p} + \varepsilon_t \tag{1.1.2}$$

上述模型的阶数为 p，所以 $\phi_p \neq 0$。当 $\phi_0 = 0$ 时，自回归模型(1.1.2)被称为中心化 $\mathrm{AR}(p)$ 模型。非中心化模型可以通过变换转化为中心化模型。

1.1.2.2 MA 模型

具有如下结构的模型称为 q 阶移动平均（moving average）模型，记为 $\mathrm{MA}(q)$：

$$x_t = \mu + \varepsilon_t - \theta_1 \varepsilon_{t-1} - \theta_2 \varepsilon_{t-2} - \cdots - \theta_q \varepsilon_{t-q} \tag{1.1.3}$$

当 $\mu = 0$ 时，模型(1.1.3)称为中心化 $\mathrm{MA}(q)$ 模型。对于非中心化模型，只需要做一个位移 $y_t = x_t - \mu$ 就可以转化为中心化模型。中心化运算不会影响序列值之间的关系。

1.1.2.3 ARMA 模型

具有如下结构的模型称为自回归移动平均模型，记为 $\mathrm{ARMA}(p, q)$：

$$x_t = \phi_0 + \phi_1 x_{t-1} + \phi_2 x_{t-2} + \cdots + \phi_p x_{t-p} + \varepsilon_t - \theta_1 \varepsilon_{t-1} - \theta_2 \varepsilon_{t-2} - \cdots - \theta_q \varepsilon_{t-q} \tag{1.1.4}$$

当 $\phi_0 = 0$ 时，自回归移动平均模型(1.1.4)被称为中心化 $\mathrm{ARMA}(p, q)$ 模型。非中心化模型可以通过变换转化为中心化模型。显然，当 $q = 0$，$\mathrm{ARMA}(p, q)$ 模型就退化为 $\mathrm{AR}(p)$ 模型；当 $p = 0$，$\mathrm{ARMA}(p, q)$ 模型就退化为 $\mathrm{MA}(q)$ 模型。

1.1.2.4 MVAR 模型

MVAR 模型是使用自身过去的数据的线性组合加上不相关白噪声的一种参数模型，主要用来描述多元时间序列之间的关系。给定集合 $S = \{x_n(k), 1 \le m \le M\}$ 为 M 个同步观测的稳态时间序列，那么 p 阶 MVAR 模型可以被定义为：

$$\begin{pmatrix} x_1(k) \\ \vdots \\ x_M(k) \end{pmatrix} = \sum_{r=1}^{p} A_r \begin{pmatrix} x_1(k-r) \\ \vdots \\ x_M(k-r) \end{pmatrix} + \begin{pmatrix} w_1(k) \\ \vdots \\ w_M(k) \end{pmatrix} \tag{1.1.5}$$

其中，$A_1, A_2, \cdots, A_p$ 代表系数矩阵（维度为 $M \times M$），其元素 $a_{ij}(r)$ 描述的是 $x_j(k-r)$ 对 $x_i(k)$ 的线性影响程度。$w_i(k)$ 表示随机高斯白噪声。模型的阶数主要通过多通道版本的 AIC（Akaike criterion）来确定。

1.1.2.5 MARX 模型

MVAR 模型是一个线性模型，无法揭示信号之间的非线性关系，并且，现实生活中的大多数模

型都是非线性模型（EEG 等），所以，我们使用 MNARX 模型（式(1.1.6)）来描述非线性关系。

$$y(t)=\sum_{n=1}^{M}\sum_{p=0}^{n}\sum_{k_1,k_{p+q}=1}^{K}c_{p,q}\left(k_1,\cdots,k_{p+q}\right)\times\prod_{i=1}^{p}y\left(t-k_i\right)\prod_{i=p+1}^{p+q}x\left(t-k_i\right)+e_y\left(t\right) \tag{1.1.6}$$

n 是最大阶为 M 的非线性系统的当前项的非线性次数（ $p+q=n$， $k_i=1,\cdots,K$，$\sum_{k_1,k_{p+q}=1}^{K}\equiv\sum_{k_1=1}^{K}\cdots\sum_{k_{p+q}=1}^{K}$ ），q、p 分别是模型的输入输出项的阶，K 是最大的时延。

1.2 时序数据的处理

1.2.1 时序数据 to 监督学习数据

时间序列[3]是一组按照时间发生的先后顺序进行排列的数据点序列，通常一组时间序列的时间间隔为一个恒定值。而我们所熟悉的预测任务、分类任务等都需要数据是拥有(input、target)这样的数据对的形式，所以，想要使用机器学习、深度学习的方法来处理时间序列就必须首先将时序数据转换为(input、target)这样的形式。（如表 1.2.1 所示）

表 1.2.1 时序数据与监督学习数据对比

时间序列数据			**监督学习数据**	
t	y		x	y
0	2		1	2
1	3		2	3
2	4		3	4
3	5		4	5
4	6		5	6

通常我们所使用的监督学习方法都可以被理解为：我们利用已知的数据尽可能准确地构建出输入 input 到目标 target 的一种函数映射 $Y=f(X)$，从而可以用来对未曾见过的 $x\in X$ 数据进行预测或分类。这里的 X 理解为全部的数据集合。例如，我们想要预测瓜田里的西瓜是否是成熟（Y 的情况），我们不可能把所有的西瓜 X 都打开检查一遍，因此，我们会从中选取一部分西瓜抽取特征并构建模型 f。那么被抽取的那部分西瓜可以被理解为已知数据，而我们想要构建的模型是针对全体西瓜集合 X 的，我们想要预测的是瓜田里其它西瓜是否成熟。

从概率学上来分析[5]，如果我们想要直接学习决策函数 $f(X)$ 或者条件概率分布 $P(Y\mid X)$，那么这一类模型被称为**判别模型**，而另有一类模型就是直接从数据中学习联合分布 $P(X,Y)$，这一类模型就被称之为**生成模型**。判别模型直接面对预测，往往学习的准确率更高，一定程度上简化了所要学习的问题，生成方法的学习收敛速度更快。这两种模型都可以同样被应用到时间序列数据上。

通常使用**滑动窗**[4]（Sliding Window）的方法将时序数据转化为监督学习数据，将先前时间点 $(t-1,t-2,t-3,\cdots)$ 的数据作为输入变量，将当前时间或者以后的时间点 $(t+1,t+2,t+3,\cdots)$ 的数据作为输出变量。

表 1.2.2 时序数据转化前后对比

转化前时序数据			**转化后时序数据**	
t	y		y(t-1)->x	y(t)
0	100		?	100
1	110		100	110
2	108		110	108
3	115		108	115
4	120		115	120
			120	?

时间序列数据中的时间信息是非常重要的，在转化的过程中应当继续保留时间信息，也即变量值的观察顺序不应该被改变，另外，从表 1.2.2 中我们也发现了我们没有用来预测 $t=0$ 的数据，所以，第一行数据应当被删除。序列的最后一个值可以在建立模型后被预测出来，所以在训练模型的

时候，表 1.2.2 最后一行的数据也是不被使用的。这种使用先前数据来预测以后时间点的数据的方法就被称为滑动窗方法，上述示例中的窗宽为 1，在时间序列模型（AR 或者 VAR 等）中，我们称之为时间延长（lag）或者时间序列模型的阶（order），模型的阶数可以通过 AIC 或者 BIC 等方法估计。滑动窗的窗宽能够决定当前时间点的观察值会受到多久之前的观察值的影响，我们可以理解为当前时间点的观察值能够记忆之前数据信息的时间跨度。滑动窗的窗宽越宽，所构造的模型就具有越强的记忆能力。

在一般的机器学习方法中，我们称一对 (x, y) 为全体数据的一个样本，x 的维度称为特征数量或者特征空间的维度。在时间序列数据中，一个滑动窗中的数据可以被称为整个序列的一个样本，一般将滑动窗中最后一个时间点的数据当做 y 而将其余数据当做 x，x 子序列的长度被当做特征数量或者特征空间的维度。

除了上述示例所描述的**一元时间序列**以外，还存在有**多元时间序列**。大多数时间序列分析方法，都集中在单变量数据上。这是因为它是最容易理解和使用的。多变量的数据通常比较难处理。它很难建模，而且许多经典的方法都表现不佳。

表 1.2.3 多元时序数据转化前后对比（窗宽=1）

转化前时序数据				**转化后时序数据**			
t	y1(t)	y2(t)		y1(t-1)	y1(t)	y2(t-1)	y2(t)
0	0.2	88		?	0.2	?	88
1	0.5	89		0.2	0.5	88	89
2	0.7	87		0.5	0.7	89	87
3	0.4	88		0.7	0.4	87	88
4	1.0	90		0.4	1.0	88	90
				1.0	?	90	?

表 1.2.3 中的窗宽设为 1，预测变量设为 y2。所以转化以后，我们可以使用 $y1(t-1)$、$y2(t-1)$、$y1(t)$ 预测 $y2(t)$（单输出问题）或者使用 $y1(t-1)$、$y2(t-1)$、$y2(t)$ 预测 $y1(t)$。同样，在训练模型的时候我们需要删除表 1.2.3 中的第一行与最后一行的数据。当然，我们也可以同时预测 $y1(t)$、$y2(t)$（多输出问题），只需要将 $y1(t-1)$、$y2(t-1)$ 作为输入即可。

1.2.2 数据标准化/归一化

数据归一化操作对不同特征维度进行伸缩变换使得各个特征维度对目标函数的影响权重是一致的，这种方法改变了原始数据的分布，但是，数据归一化提高了模型迭代求解的速度和精度。数据标准化也会对不同特征维度进行伸缩变换，但其目的是使得不同度量之间的特征具有可比性。同时并不改变原始数据的分布。归一化会改变原始数据的距离、分布等而标准化只是转换了原始数据的特征空间并没有改变数据的距离、分布等信息。事实上，以上是对归一化和标准化的严格定义而在实际操作中归一化与标准化并没有太大的区别，这里主要介绍 z-score 和 min-max 归一化方法，这也是机器学习任务中经常使用的归一化方法。

1.2.2.1 z-score 归一化

$$x' = \frac{x - \bar{x}}{\sigma} \tag{1.2.1}$$

1.2.2.2 min-max 归一化

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)} \tag{1.2.2}$$

1.2.2.3 mean 归一化

$$x' = \frac{x - \bar{x}}{\max(x) - \min(x)} \tag{1.2.3}$$

时间序列数据的标准化我们通常根据当前的任务类型选择合适的标准化方式，一般在训练集上是在单个窗口上进行标准化而在训练集上是对整个集合的数据进行标准化。

1.2.3 训练集测试集划分

使用滑动窗方法可以对时间序列采样，一段时间序列被划分为多少个时间窗就会有多少个样本（这里还涉及到滑动窗的滑动步长 shift，通常情况下，认为步长=1）。在评估时间序列预测模型时，

我们感兴趣的是模型对不用于训练它的数据的性能。在机器学习中，我们称之为看不见的或者是样本外的数据。我们可以通过分割我们已有的数据来做到这一点。我们用其中的一部分数据训练模型，而且需要保留一些数据，让训练出来的模型对那些数据做出预测。如果对保留数据的预测效果很好，那么在这个训练出来的模型就是值得推广的或者具有很强的泛化能力。因此，如何对这些样本进行划分是我们面临的一个新问题。

这里使用太阳黑子的观测数据集作为示例[8]，这个数据集记录了 1749 年到 1983 年之间每月的太阳黑子情况。如果，只需要划分一个训练集、一个测试集，那么只需要在整个序列上任意地选取一个分割点或者通过实现确定的分割比例选取分割点。通常将该分割点以前的数据（历史数据）作为训练集而将该分割点之后的数据当做测试集。划分情况如图 1.2.1 所示。如果在训练模型的过程中采用边训练边评估的训练方式，那么可以继续在训练集中选取一个分割点将分割点之前的数据当做训练集而将分割点之后的数据当做验证集。

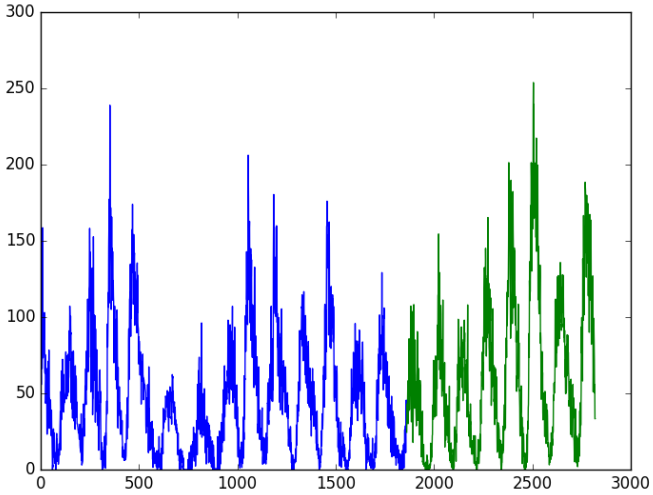


图 1.2.1 单个训练集、测试集划分

当然，我们也可以在整个时序数据上执行多次训练集、测试集划分以期望可以获得多组实验数据，多个模型。我们可以通过对所构建的多个模型进行综合评估从而获得更健壮的模型。假设有 100 个样本点，我们想要得到两个训练集、测试集，那么具体的划分就是子集 1：{1-33 个样本当做训练集，接下来的 33 个样本点当做测试集}、子集 2{1-66 个样本当做训练集，67-100 个样本点当做测试集}。训练集、测试集的划分过程中测试集的大小应该保持一致。这意味着每个训练模型的预测计算的性能统计数据将是一致的（模型都是在同样多的数据上进行评估保证了评估结果是可比较的），并且可以组合和比较。图 1.2.2 表示的是太阳黑子数据 3 个训练集测试集的划分情况（n=3）。

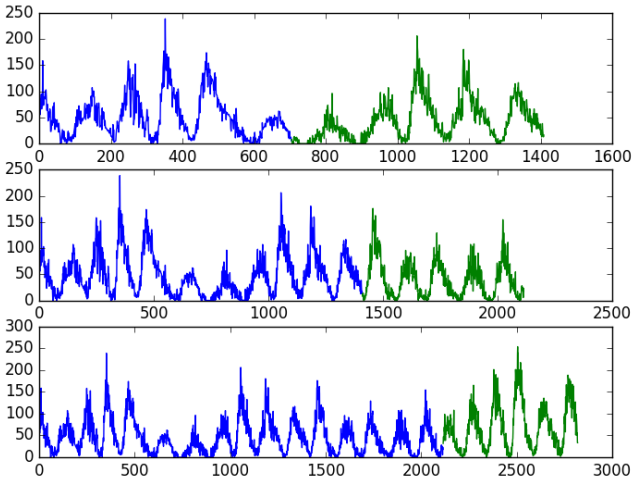


图 1.2.2 多个训练集、测试集划分

1.2.4 交叉验证

在一般的机器学习中，模型评估通常使用 K 折交叉验证法，而时间序列数据不能够直接使用这

一方法，这是因为一般机器学习中所使用的数据都被假设为相互独立的个体，样本与样本之间的顺序不会影响到模型的构建，更不会影响到模型的结果。但是，时间序列中的时间信息是至关重要的，我们在使用时序数据的时候必须注意不能更改数据或样本的顺序。

在处理时间序列数据时使用 **Walk Forward Validation**[6]方法评估模型性能。实际中，当出现新的可用数据时，很有可能需要重新训练模型，为了保证得到最稳健的模型，在训练过程中采用交叉验证的方法来评估模型性能是最行之有效的方法。在时间序列数据的处理过程中，我们在进行交叉验证之前首先需要确定以下信息：

1. **观测数量的最小值**。也即能够成功的训练出一个模型的最少样本数，这通常理解为滑动窗的窗宽。
2. **滑动窗的设置**。滑动窗的步长，也即是否需要扩充窗口的大小。同时，需要确定预测类型。模型是在当前的窗口进行预测还是在整个数据集上进行训练。这会有很多模型被训练出来。

Walk Forward Validation 方法是应用在时间序列数据上的 K 折交叉验证，这在评估模型性能的过程中非常有用，**Walk Forward Validation** 通过创建多个模型来选择出最稳健的模型参数，有利于提升所选择模型的泛化能力，但是，这种方法会增加计算成本。图 1.2.3 给出了传统机器学习数据的 K 折交叉验证的数据集划分情况，图 1.2.4 给出了时间序列数据上的交叉验证数据集划分情况。

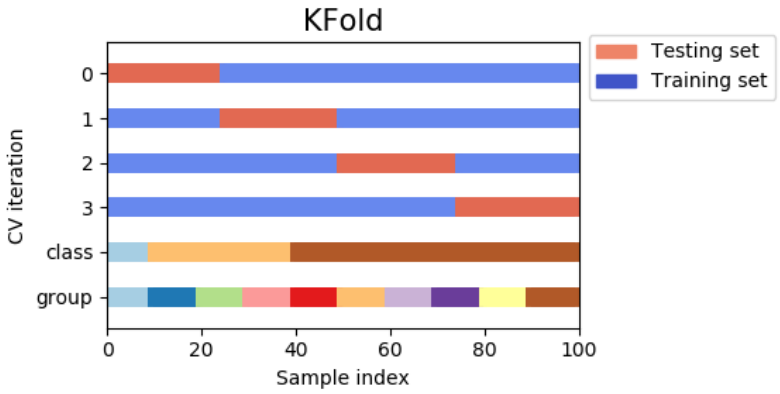


图 1.2.3 K-折交叉验证的数据集分割

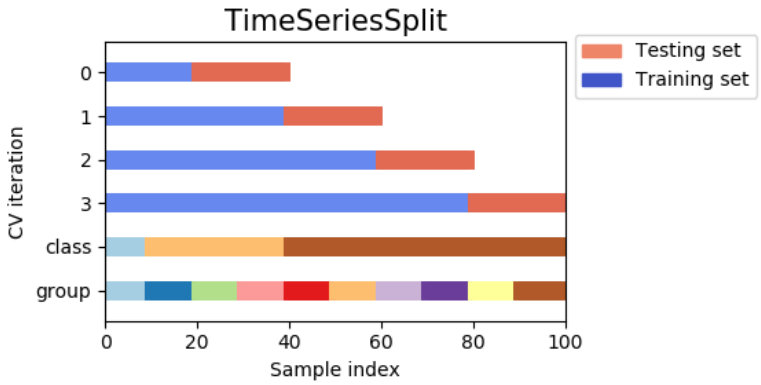


图 1.2.4 时间序列数据交叉验证的数据集分割

第二章 深度网络模型

2.1 循环神经网络模型

2.1.1 RNN 模型

传统的神经网络没有记忆功能，也就是说传统的神经网络不能根据之前的信息来推断之后会发生的事情，但是循环神经网络可以做到。在 RNNs 的网络中，有一个循环的操作，使得它们能够保留之前学习到的内容[15]。

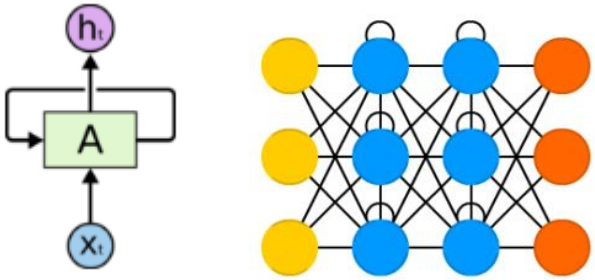


图 2.1.1 RNN 的结构

RNN 网络的结构如图 2.1.1 所示（X_t 理解为一个长度为 t 的序列， A 为神经网络操作，自身循环操作理解为将前一时刻的输出作为当前网络的输入），所有循环神经网络结构都是由完全相同结构的（神经网络）模块进行复制而成的。其运算可以表述为[16]：

$$h_t = \tanh\left(w_{ih}x_t + b_{ih} + w_{hh}h_{(t-1)} + b_{hh}\right) \tag{2.1.1}$$

式(2.1.1)中 h_t 表示时刻 t 的隐藏状态， x_t 表示时刻 t 的输入， $h_{(t-1)}$ 表示之前的 RNN 单元在时刻 t-1的隐藏状态或者时刻 0 的初始隐藏状态，b_{ih} 和 b_{hh} 都表示的是偏置。RNN 模型可以简化为图 2.1.2 所示（这里的 X_t 表示为 t 时间点的输入）：

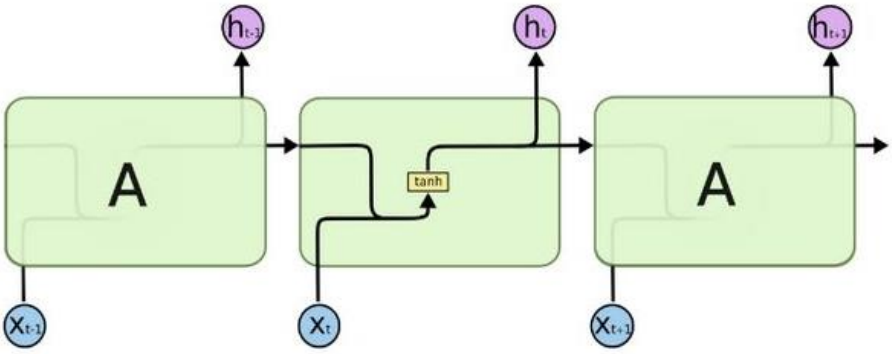


图 2.1.2 RNN 的简化模型

PyTorch 的具体实现：

```
1.  def RNN(x, h, W_h, U_h, b_h):
2.    y = []
3.    for t in range(x.size(0)):
4.      h = torch.tanh(x[t] @ W_h + h @ U_h + b_h)
5.      y += [h]
6.    return torch.stack(y), h
```

其实一个普通的 RNN 和一般的神经网络区别不大，依然是从输入到输出，信号的值一层一层地往后面推送。从另外一个角度看，一个 RNN 就是由不同神经点的普通网络组成，并且前一个和后一个时间点的网络是密切相关的。一个 RNN 在展开表示之后（图 2.1.3），就是在时间上复用同一个网络，这样的结果就像我们的序列数据一样在每个时间点都会有所不同，但是有都被之前的状态影响着，很多 RNNs 都建立在这种网络结构之上。在预测单个数值或者分类任务中总是将最后一个序列的输出结果作为预测输出结果，也可以把序列输出结果作为序列，即后面要说明的 SeqSeq 模型。另外，在处理一些不太长的相关信息和位置间隔时，RNN 会丧失学习到连接较远的信息的能力。

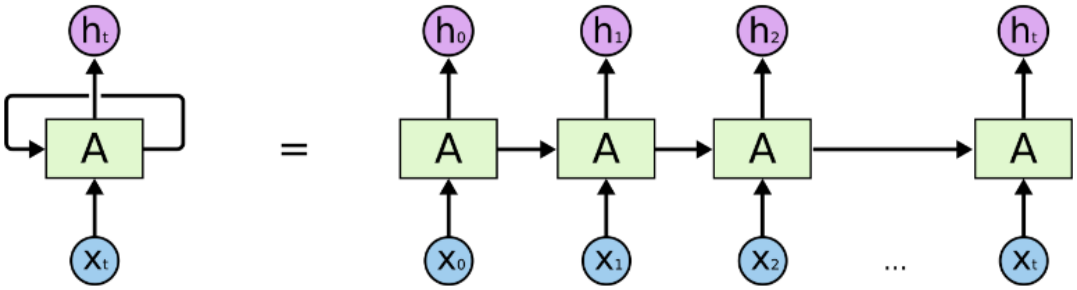


图 2.1.3 RNN 模型的展开表示

2.1.1.1 RNN 的梯度消失问题

RNN 训练困难的主要原因在于隐藏层参数 w 的传播：由于误差传播在展开后的 RNN 上，无论在前向传播过程还是在反向传播过程中 w 都会乘上多次，这就导致：

1) **梯度消失**：如果梯度很小的话（<1），乘上多次指数级下降，对输出几乎就没有影响了
2) **梯度爆炸**：反过来，如果梯度很大的话，乘上多次指数级增加，又导致了梯度爆炸

当然了，这个问题其实存在于任何深度神经网络中，只是由于 RNN 的递归结构导致其尤其明显。

对于梯度爆炸问题，可以通过截断的方式来有效避免，而对梯度消失问题，则有很多不同的方案：

- 有效初始化+ReLU 激活函数能够得到较好效果。
- 算法上的优化，例如截断的 BPTT 算法。
- 模型上的改进，例如 LSTM、GRU 单元都可以有效解决长期依赖问题。
- 在 BPTT 算法中加入 skip connection，此时误差可以间歇的向前传播。
- 加入一些 Leaky Units，思路类似于 skip connection。

2.1.1.2 双向 RNN

单向 RNN 的问题在于 t 时刻进行分类的时候只能利用 t 时刻之前的信息， 但是在 t 时刻进行分类或者预测的时候可能也需要利用未来时刻的信息。双向 RNN（bi-directional RNN）模型（如图 2.1.4）正是为了解决这个问题， 双向 RNN 在任意时刻 t 都保持两个隐藏层，一个隐藏层用于从左往右的信息传播， 另一个隐藏层用于从右往左的信息传播。这种网络的主要思路是在输入和目标之间添加延迟，进而可以给网络一些时步来加入未来的上下文信息，也就是加入 M 时间帧的未来信息来一起预测输出。理论上，M 可以非常大来捕获所有未来的可用信息，但事实上发现如果 M 过大，预测结果将会变差。这是因为网络把精力都集中记忆大量的输入信息，而导致将不同输入向量的预测知识联合的建模能力下降。因此，M 的大小需要手动来调节。

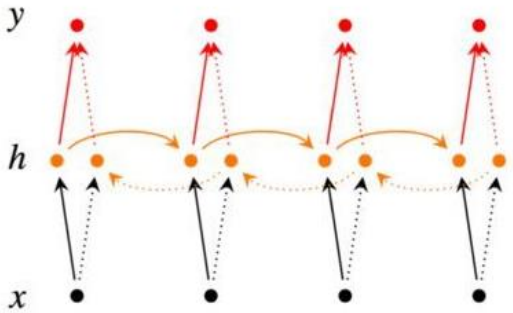


图 2.1.4 双向 RNN 模型

双向 RNN 的过程可以描述为：

从前往后传播过程：

$$h_i^1 = f\left(U^1 * X_i + W^1 * h_{i-1}\right) \tag{2.1.2}$$

从后往前传播过程：

$$h_i^2 = f\left(U^2 * X_i + W^2 * h_{i-1}\right) \tag{2.1.3}$$

网络模型的输出为：

$$y_i = \text{softmax}\left(V * [h_i^1; h_i^2]\right) \tag{2.1.4}$$

式(2.1.4)中的$[h_i^1; h_i^2]$表示将前向传播和后向传播的信息拼接起来。另外，双向 RNN 需要保存两个方向的权重矩阵，所以需要的内存约为 RNN 的两倍。

Deep(Bidirectional)RNNs 与 Bidirectional RNNs 相似，只是对于每一步的输入有多层网络。这样，该网络便有更强大的表达与学习能力，但是复杂性也提高了，同时需要更多的训练数据。

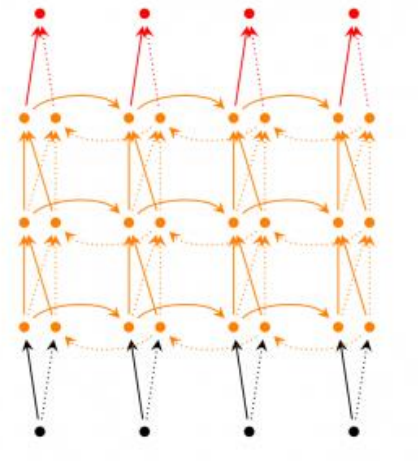


图 2.1.5 Deep 双向 RNN 模型

2.1.2 LSTM 模型

RNNs 的出现，主要是因为它们能够把以前的信息联系到现在，从而解决现在的问题。有时候，我们在处理当前任务的时候，只需要看一下比较近的一些信息。在这样的情况下，我们所要预测的内容和相关信息之间的间隔很小，这种情况下 RNNs 就能够利用过去的信息， 很容易的实现。但是，有些情况是需要更多的上下文信息。随着预测信息和相关信息间的间隔增大， RNNs 很难去把它们关联起来了。长短期记忆网络（Long Short Term Memory networks），通常叫做“LSTMs”。主要是为了避免长时期依赖（long-term dependency ）问题。它们的本质就是能够记住很长时期内的信息，而且非常轻松就能做到。区别于普通的 RNN 结构在中间只有单一的 tanh 层，LSTMs 在中间部分（A 操作）有四个相互作用的层。单一的 LSTMs 的结构如图 2.1.6 所示：

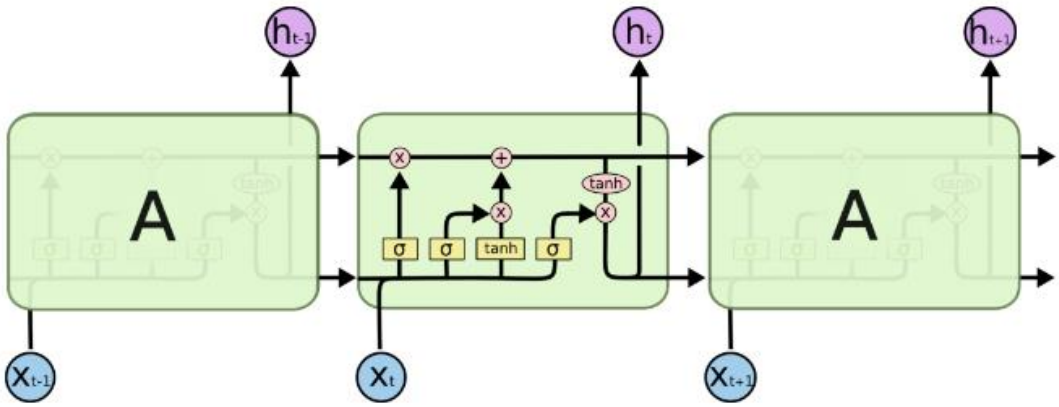


图 2.1.6 LSTMs 的结构

上述一个绿色的结构称为一个 cell，其对应一个输出状态h_t，这个状态在不同的 cell 之间进行传输，在传输的过程中只是做了少量的线性操作，这种结构能够很轻松地实现信息从整个 cell 中穿过而不做改变，从而可以实现了长时期的记忆保留。LSTMs 通过一种叫做门（gate）的结构来实现增加或者删除信息。门可以实现选择性地让信息通过，主要是通过一个 sigmoid 的神经层 和一个逐点相乘的操作来实现的。sigmoid 层输出（是一个向量）的每个元素都是一个在 0 和 1 之间的实数，表示让对应信息通过的权重（或者占比）。比如，0 表示“不让任何信息通过”，1 表示“让所有信息通过”。每个 LSTM 有三个这样的门结构，来实现保护和控制信息。分别是遗忘门（forget gate layer）、输入门（input gate layer）、输出门（output gate layer）。注意，这里的激活函数只能是

sigmoid 函数而不可以是 ReLU 等函数，因为每一个闸门是用 0 代表关闭，1 代表打开，而其他激活函数不具备这种特征。

- **遗忘门**

遗忘门（如图 2.1.7）决定让多少上一次的输出影响这一次的输出。遗忘门的作用就是减少上一个 cell 状态对下一个 cell 状态的影响。

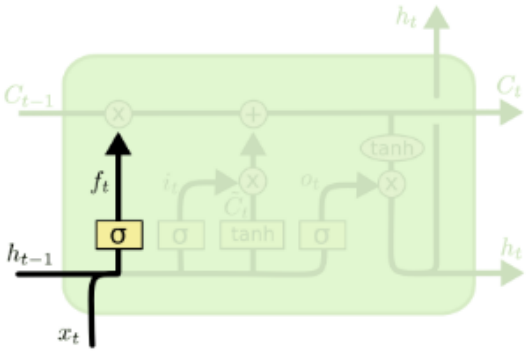


图 2.1.7 遗忘门

遗忘门的公式表示为：

$$f_t = \sigma\left(W_f \bullet \left[h_{t-1}, x_t\right] + b_f\right) \tag{2.1.5}$$

- **输入门**

输入门（图 2.1.8）决定让多少新的信息加入到 cell 状态中来。实现这个需要包括两个步骤：首先使用 Tanh 提取有效的信息，然后使用 Sigmoid 闸门来对信息进行筛选。这一步决定了 cell 状态被更新的程度，C_t 是新进入到网络中的信息，i_t 就作为C_t 影响到更新程度的标量。

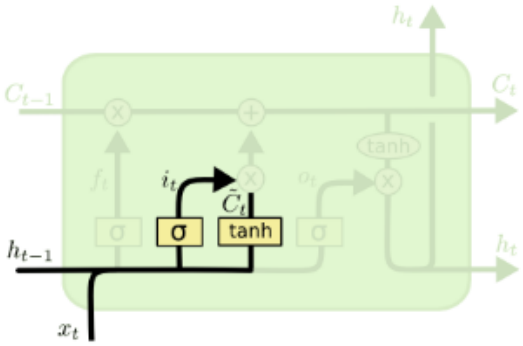


图 2.1.8 输入门

输入门的公式表示为：

$$\begin{cases} i_t = \sigma\left(W_i \bullet [h_{t-1}, x_t] + b_i\right) \\ C_t = \tanh\left(W_C \bullet [h_{t-1}, x_t] + b_C\right) \end{cases} \tag{2.1.6}$$

在经过遗忘门和输入门后，我们就需要更新 cell 状态了。如图 2.1.9 所示，这时只需要把要加入的值和已经被遗忘门清理过的向量相加，即可得到新的 cell 状态向量了。

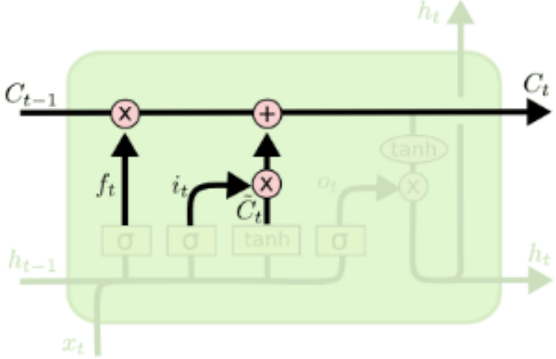


图 2.1.9 更新 cell 状态

具体的公式表述为：

$$C_t = f_t * C_{t-1} + i_t * C_t \tag{2.1.7}$$

- **输出门**

输出门（图 2.1.10）需要输出新的 cell 状态到下一步，并计算新的隐状态输出。Cell 状态和隐状态都会输入到下一个 LSTM 中，而隐状态会作为 LSTM 的输出，和普通 RNN 一样输出到其他网络中。输出门使用 sigmoid 层决定 C_t 中被保留的信息的比例，使用 tanh 层将数值映射到-1 和 1 之间。最后将 tanh 层的输出和 sigmoid 层计算出来的权重相乘，这样就得到了最后输出的结果。

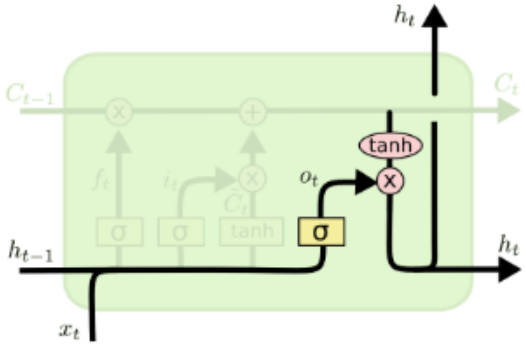


图 2.1.10 输出门

输出门的公式表示为：

$$\begin{cases} o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right) \\ h_t = o_t * \tanh\left(C_t\right) \end{cases} \tag{2.1.8}$$

上述给出的是单个 LSTM 单元的运算方式，大型的 LSTM 网络是由这样的结构连接起来的（图 2.1.12）。同样，为了在预测当前时间点的数值时使用将来的信息，童谣已经有学者提出双向 LSTM 模型（如图 2.1.11 所示）。

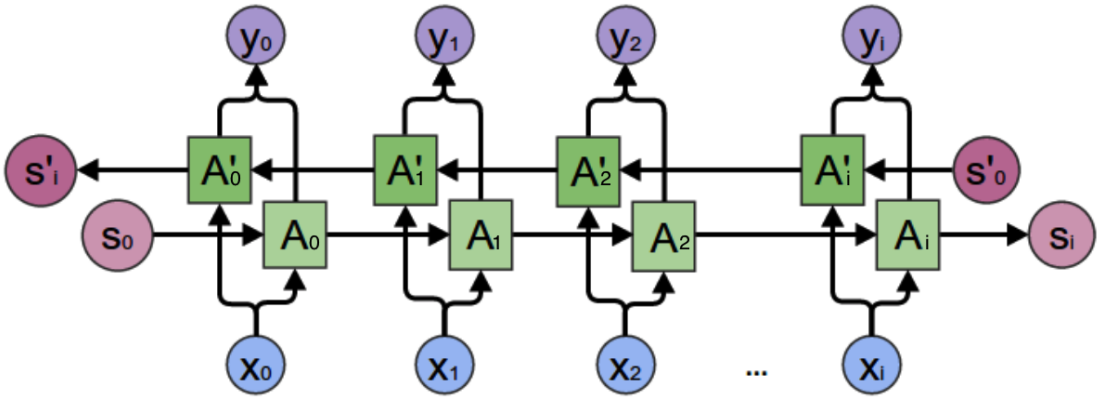


图 2.1.11 双向 LSTM

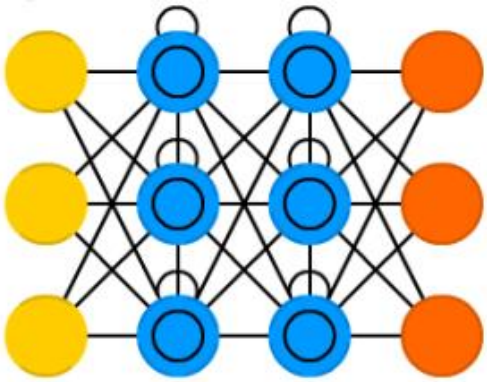


图 2.1.12 LSTM 网络

2.1.3 GRU 模型

GRU 模型是 LSTM 模型的一种变种，在 GRU[19]中只有两个门：重置门（reset gate）和更新门（update gate），更新门由遗忘门和输入门合并而来。同时在这个结构中，把细胞状态和隐藏状态进行了合并。最后模型比标准的 LSTM 结构要简单，而且，GRU 模型的计算复杂度较低。GRU 网络 cell 表述为图 2.1.13 所示，完整的 GRU 网络表述为图 2.1.14 所示。

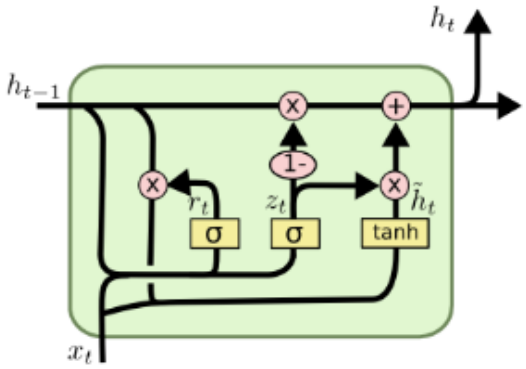


图 2.1.13 GRU 的 cell 结构

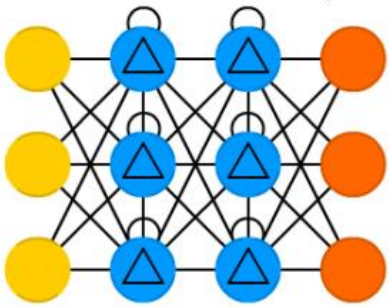


图 2.1.14 完整的 GRU 模型

- **重置门**

  重置门的公式表述为：

$$r_t = \sigma\left(W_r \bullet \left[h_{t-1}, x_t\right]\right) \tag{2.1.9}$$

计算出重置门 r_t 以后，需要计算候选隐藏层 h_t，这个候选隐藏层和 LSTM 中的 C_t 是类似，可以看成是当前时刻的新信息，其中 r_t 用来控制需要保留多少之前的记忆。h_t 的计算方式为：

$$h_t = \tanh\left(W \bullet \left[r_t * h_{t-1}, x_t\right]\right) \tag{2.1.10}$$

- **更新门**

  更新门的公式表述为：

$$z_t = \sigma\left(W_z \bullet \left[h_{t-1}, x_t\right]\right) \tag{2.1.11}$$

最后 z_t 控制需要从前一时刻的隐藏层 h_{t-1} 中遗忘多少信息，需要加入多少当前时刻的隐藏层信息 h_t，最后得到 h_t，直接得到最后输出的隐藏层信息，需要注意这里与 LSTM 的区别是 GRU 中没有输出门（output gate）。h_t 的更新公式为：

$$h_t = \left(1 - z_t\right) * h_{t-1} + z_t * h_t \tag{2.1.12}$$

2.1.4 RNN 的类型

RNN 的类型主要分为五种，分别是 1 对 1、1 对多、多对一、等长多对多、不等长多对多（如表 2.1.1 所示），每一个图形都是一个向量（长方形），每一个箭头都是一个方程（神经网络），也代表输入流的方向。

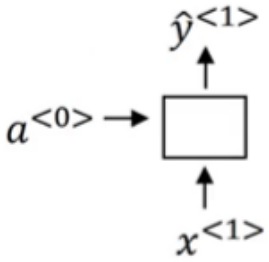
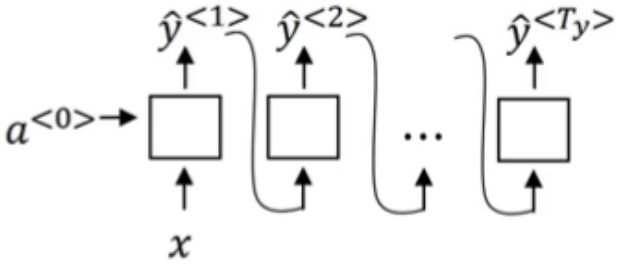
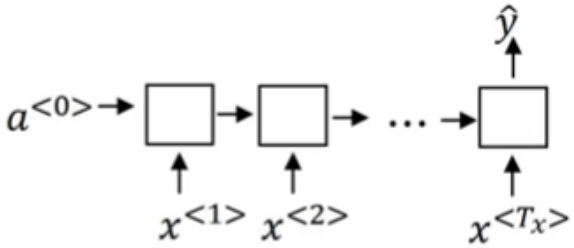
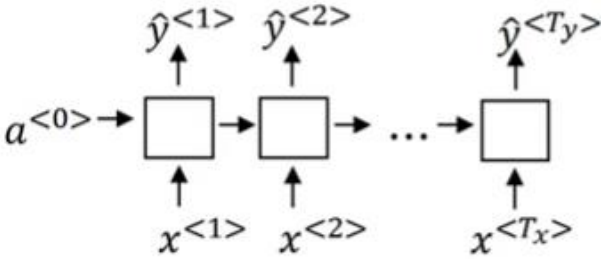
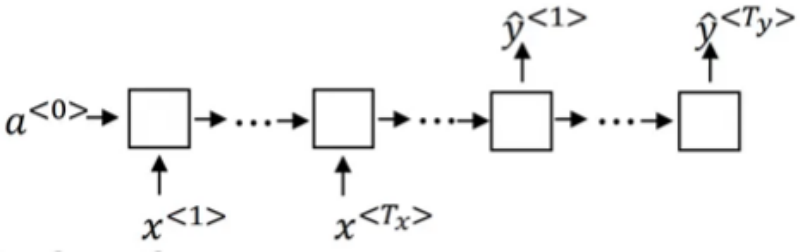
网络类型	网络结构
一对一 RNN 模型	
一对多 RNN 模型	
多对一 RNN 模型	
等长多对多 RNN 模型	
不等长多对多 RNN 模型	

表 2.1.1 不同类型的 RNN 模型

1) 1 对 1 模型就是普通的前馈神经网络，没有 RNNs 的参与。
2) 一个输入对很多输出：比如输入一张图片，然后输出这张图片的描述。
3) 多对一模型：比如情感分析，对于一段话，想要判断这段话是令人开心还是忧虑的，只有在输入完整的整个句子后才能输出答案。
4) 等长多对多模型：这种模型是输入输出共同进退的模型，比如，在医疗运用里对一个癫痫并发症的预测，就要在每时每刻的闹心好输入之后，同时对并发症的来临的判断。
5) 不等长多对多模型：这种多对多模型也成为 Seq2Seq 模型，在翻译里面会经常使用，如输入完整的一句英文后输出一句完整的中文。

2.2 Seq2Seq 模型

在很多应用中，输入和输出都可以是不定长序列。当输入输出都是不定长序列时，我们可以使用编码器—解码器（encoder-decoder）或者 seq2seq 模型。这两个模型本质上都用到了两个循环神经网络，分别叫做编码器和解码器。编码器对应输入序列，解码器对应输出序列。

2.2.1 编码器

编码器的作用是把一个不定长的输入序列变换成一个定长的背景变量 $\mathbf{c}$，并在该背景变量中编码输入序列信息。常用的编码器是循环神经网络。

考虑一个输入序列 $x_1,\cdots x_T$，在时间步 t，循环神经网络将输入 x_t 和上个时间步的隐藏状态 h_{t-1} 变换为当前时间步的隐藏状态 h_t。可以用函数 f 表达循环神经网络隐藏层的变换：

$$h_t = f\left(x_t, h_{t-1}\right) \tag{2.2.1}$$

假设输入序列的总时间步数为 T，接下来编码器通过自定义函数 q 将各个时间步的隐藏状态变换为背景变量：

$$\mathbf{c} = q\left(h_1,\cdots h_T\right) \tag{2.2.2}$$

例如，当选择 $q\left(h_1,\cdots h_T\right)=h_T$ 是，背景变量是输入序列最终时间步的隐藏状态 h_T。这种编码器是一个单项的循环神经网络，每个时间步的隐藏状态只取决于该时间步及之前的输入子序列。当然，也可以使用双向循环神经网络构造编码器。这种情况下，编码器每个时间步的隐藏状态同时取决于该时间步之前和之后的子序列（包括当前时间步的输入），并编码了整个序列的信息。

2.2.2 解码器

编码器输出的背景变量 $\mathbf{c}$ 编码了真个输入时间序列 $x_1\cdots x_T$ 的信息。给定训练样本中的输出序列 $y_1,y_2,\cdots,y_{T'}$，对每个时间步 t'，解码器输出 $y_{t'}$ 的条件概率将基于之前的输出序列 $y_1,\cdots,y_{t'-1}$ 和背景变量 $\mathbf{c}$，即 $P\left(y_{t'}\mid y_1,\cdots,y_{t'-1},\mathbf{c}\right)$。

为此，可以使用另一个循环神经网络作为解码器。在输出序列的时间步 t'，解码器将上一时间步的输出 $y_{t'-1}$ 以及背景变量 $\mathbf{c}$ 作为输入，并将它们与上一时间步的隐藏状态 $s_{t'-1}$ 变换为当前时间步的隐藏状态 $s_{t'}$。因此，可以使用函数 g 表达解码器隐藏层的变换：

$$s_{t'} = g\left(y_{t'-1},\mathbf{c},s_{t'-1}\right) \tag{2.2.1}$$

有了解码器的隐藏状态后，可以使用自定义的输出层和 softmax 运算来计算 $P\left(y_{t'}\mid y_1,\cdots,y_{t'-1},\mathbf{c}\right)$，例如基于当前时间步的解码器隐藏状态为 $s_{t'}$、上一时间步的输出 $y_{t'-1}$ 以及背景变量 $\mathbf{c}$ 来计算当前时间步输出 $y_{t'}$ 的概率分布。

训练时，根据最大似然估计，可以最大化输出序列基于输入序列的条件概率：

$$\begin{aligned} P\left(y_1,\cdots,y_{T'}\mid x_1,\cdots,x_T\right) &= \prod_{t'=1}^{T'} P\left(y_{t'}\mid y_1,\cdots,y_{t'-1},x_1,\cdots,x_T\right) \\ &= \prod_{t'=1}^{T'} P\left(y_{t'}\mid y_1,\cdots,y_{t'-1},\mathbf{c}\right) \end{aligned} \tag{2.2.2}$$

并得到该输出序列的损失：

$$-\log P\left(y_1,\cdots,y_{T'}\mid x_1,\cdots,x_T\right) = -\sum_{t'=1}^{T'}\log P\left(y_{t'}\mid y_1,\cdots,y_{t'-1},\mathbf{c}\right) \tag{2.2.3}$$

在模型训练中，通过最小化这个损失函数来得到模型参数。

2.3 Attention 机制

注意力机制的思想是，在每一步中，都让 RNN 从一个更大的信息集合中去选择信息。同时可以在解码器的每个时间步使用不同的背景变量，并对输入序列中不同时间步编码的信息分配不同的注

意力。

在时间步 t'，设解码器的背景变量为 $\mathbf{c}_{t'}$，输出 $y_{t'}$ 的特征向量为 $\mathbf{y}_{t'}$。和输入的特征向量一样，这里每个输出的特征向量也是模型参数。解码器在时间步 t' 的隐藏状态：

$$s_{t'} = g\left(\mathbf{y}_{t'-1}, \mathbf{c}_{t'}, s_{t'-1}\right) \tag{2.3.1}$$

令编码器在时间步 t 的隐藏状态为 h_t，且总时间步数为 T。解码器在时间步 t' 的背景变量为：

$$\mathbf{c}_{t'} = \sum_{t=1}^{T} \alpha_{t't} \mathbf{h}_t \tag{2.3.2}$$

其中 $\alpha_{t't}$ 是权值。也就是说，给定解码器的当前时间步 t'，我们需要对编码器中不同时间步 t 的隐藏状态求加权平均。这里的权值也称注意力权重。它的计算公式是

$$\alpha_{t't} = \frac{\exp\left(e_{t't}\right)}{k = \sum_{k=1}^{T} \exp\left(e_{t'k}\right)} \tag{2.3.3}$$

其中 $e_{t't} \in \mathbb{R}$ 的计算为：

$$e_{t't} = a\left(s_{t'-1}, \mathbf{h}_t\right) \tag{2.3.4}$$

上式中的函数 a 有多种设计方法。 Bahanau 等使用了多层感知机：

$$e_{t't} = v^T \tanh\left(\mathbf{W}_s s_{t'-1} + \mathbf{W}_h \mathbf{h}_t\right) \tag{2.3.5}$$

其中 v、$\mathbf{W}_s$、$\mathbf{W}_h$ 以及编码器与解码器中的各个权重和偏差都是模型参数。

Bahanau 等在编码器和解码器中分别使用了门控循环单元。在解码器中，我们需要对门控循环单元的设计稍作修改。解码器在 t' 时间步的隐藏状态为：

$$s_{t'} = z_{t'} \odot s_{t'-1} + \left(1 - z_{t'}\right) \odot \tilde{s}_{t'} \tag{2.3.6}$$

其中的重置门、更新门和候选隐含状态分别为：

$$r_{t'} = \sigma\left(\mathbf{W}_{yr} \mathbf{y}_{t'-1} + \mathbf{W}_{sr} s_{t'-1} + \mathbf{W}_{cr} c_{t'} + b_r\right) \tag{2.3.7}$$

$$z_{t'} = \sigma\left(\mathbf{W}_{yz} \mathbf{y}_{t'-1} + \mathbf{W}_{sz} s_{t'-1} + \mathbf{W}_{cz} c_{t'} + b_z\right) \tag{2.3.8}$$

$$\tilde{s}_{t'} = \tanh\left(\mathbf{W}_{ys} \mathbf{y}_{t'-1} + \mathbf{W}_{ss}\left(s_{t'-1} \odot r_{t'}\right) + \mathbf{W}_{cs} c_{t'} + b_s\right) \tag{2.3.9}$$

第三章 时间序列预测任务

3.1 预测的类型

针对时间序列数据的特殊性，时间序列数据的预测任务主要可以分为三种类型 point-by-point prediction、multi-sequence prediction、full sequence prediction[1][2]。这三种预测都有一定的特点和具体的使用场景，三种预测的具体实现可以参看代码[2]。

假设我们有一列数据（一元数据），它的窗口我们设为 50，也即 50 个点当做一个采样点（前 49 个点当做输入 x，最后一个点当做输出 y），假设每次窗口滑动一个时间点。那么训练模型的过程可以表示为：

Step1：

使用$\{x_1, x_2, \cdots, x_{49}\}$，预测$t=50$时的数值$x_{50}$，计算 loss。

Step2：

使用$\{x_2, x_3, \cdots, x_{50}\}$，预测$t=51$时的数值$x_{51}$，计算 loss。

Step3：

使用$\{x_3, x_4, \cdots, x_{51}\}$，预测$t=52$时的数值$x_{52}$，计算 loss。

……

Step{j}：

使用$\{x_j, x_{j+1}, \cdots, x_{j+48}\}$，预测$t=j+49$时的数值$x_{j+49}$，计算 loss。

上述过程称为一个完整的 epoch，这里的 batchsize=1,，当然也可以设置 batchsize 为其他数值。例如，当我们设 batchsize=32 时，计算时就会有并行的效果，每次会有 32 个样本被同时送入模型进行训练，当然也会同时得到 32 个预测值。关于损失 loss 可以计算 32 个预测值的损失和也可以计算其平均值（每个预测点的损失 loss），batchsize 的大小由计算机的内存决定（CPU 或者 GPU），因为 batchsize 决定了每次送去内存运算的数据量。模型训练的优化目标就是使得在整个数据集上 loss 和或者平均 loss 最小。经过多个 epoch 的训练，最终得到表现优异的模型，记为 $f(t)$。

3.1.1 point-by-point prediction

point-by-point prediction 是一种单点预测方式，每次只预测当前时间点的数值。假设测试集有 1000 个数据点（$t=5000 \sim t=6000$），那么，point-by-point prediction 的预测为 $\{f(5050), f(5051), \cdots, f(6000)\}$。

Step1：

使用$\{x_{5001}, x_{5002}, \cdots, x_{5049}\}$，预测$t=5050$时的数值$x_{5050}$。

Step2：

使用$\{x_{5002}, x_{5003}, \cdots, x_{5050}\}$，预测$t=5051$时的数值$x_{5051}$。

……

Step{j}：

使用$\{x_{j+5000}, x_{j+5001}, \cdots, x_{j+5048}\}$，预测$t=j+5049$时的数值$x_{j+5049}$。

也就是说这种预测方式，窗口滑动一次就会做一次预测，最终会得到$t=5000$之后的时间点的所有预测序列，每次预测时使用的都是真实的观测数值。这里的窗宽可以理解为时间序列模型的阶。通常情况下，窗口的滑动步长都设为一个时间单位。

point-by-point prediction 方式的预测效果大致如图 3.1.1 所示[2]：

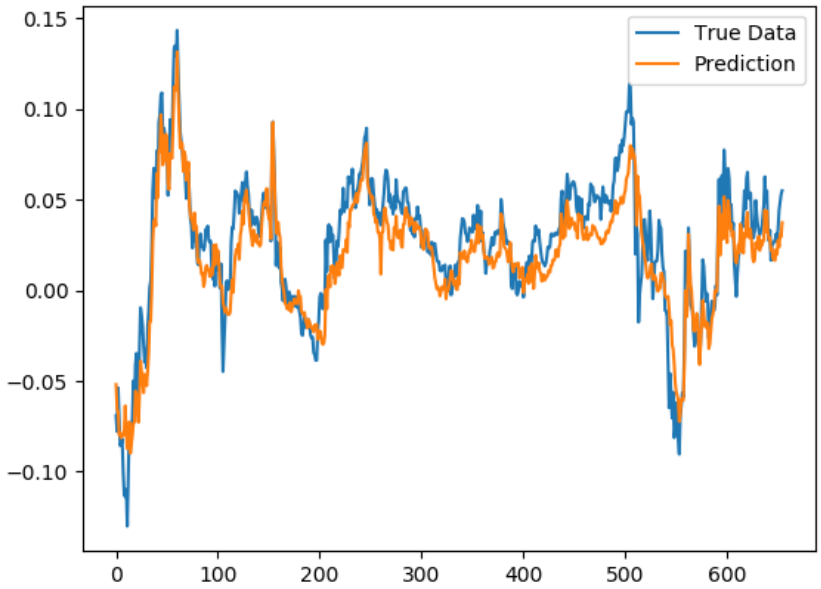


图 3.1.1 point-by-point prediction

3.1.2 multi-sequence prediction

multi-sequence prediction 的预测方式每次要先确定预测的长度（其他配置同上），这里假设为 50，那么每隔 50 个时间点（或者说窗口移动 50 次）进行一次初始化操作。具体过程可以被简单地表述为：

Step1：

使用 $\{x_{5001}, x_{5002}, \cdots, x_{5049}\}$，预测 $t=5050$ 时的数值 x_{5050}。

Step2：

使用 $\{x_{5002}, x_{5003}, \cdots, x_{5050}\}$，预测 $t=5051$ 时的数值 x_{5051}。

……

Step51：

使用 $\{x_{5051}, x_{5052}, \cdots, x_{5099}\}$，预测 $t=5100$ 时的数值 x_{5100}。

Step52：

使用 $\{x_{5100}, x_{5101}, \cdots, x_{5100}\}$，预测 $t=5101$ 时的数值 x_{5101}。

……

multi-sequence prediction 预测方式效果大致如图 3.1.2 所示[2]：

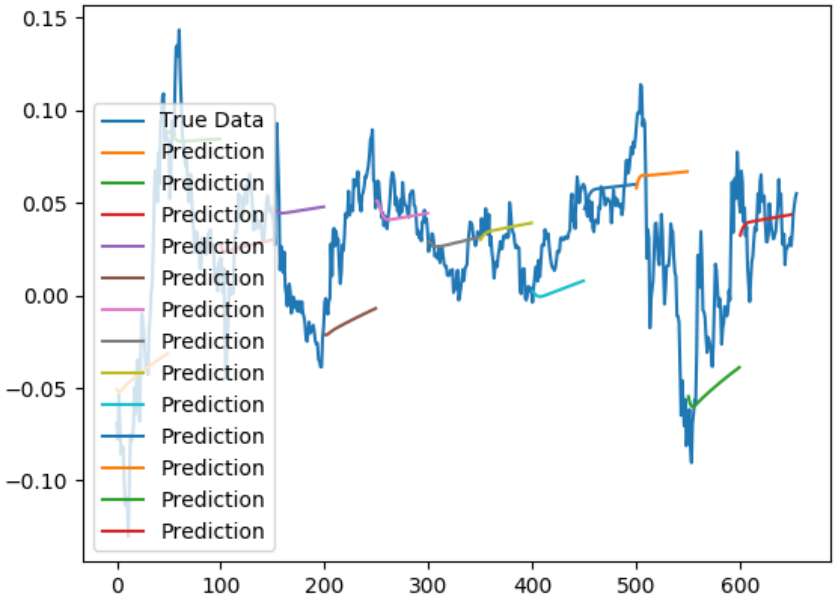


图 3.1.2 multi-sequence prediction

3.1.3 full sequence prediction

full sequence prediction 的预测方式使用第一个窗口的数据进行初始化操作，之后我们使用之前预测的数据来预测当前点的数值。例如，第二次预测时，最后一个数据点就是使用的第一次预测出来的数值而不是真实的数值。可以被简单表述为如下的过程（各种配置同上）：

Step1：

使用 $\{x_{5001}, x_{5002}, \cdots, x_{5049}\}$，预测 $t=5050$ 时的数值 x_{5050}。

Step2：

使用 $\{x_{5002}, x_{5003}, \cdots, x_{5050}\}$，预测 $t=5051$ 时的数值 x_{5051}。

Step3：

使用 $\{x_{5003}, x_{5004}, \cdots, x_{5050}, x_{5051}\}$，预测 $t=5052$ 时的数值 x_{5052}。

……

Step{j}：

使用 $\{x_{j+5000}, x_{j+5001}, \cdots, x_{j+5048}\}$，预测 $t=j+5049$ 时的数值 x_{j+5049}。

full sequence prediction 预测方式效果大致如图 3.1.3 所示[2]：

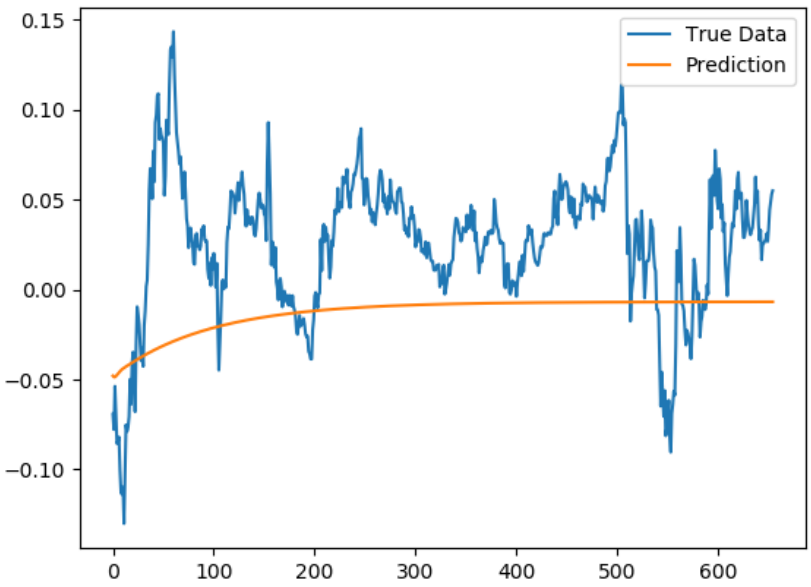


图 3.1.3 full sequence prediction

3.1 节所提到的三种预测时间序列的方法都有一定的使用环境，point-by-point prediction 主要用来预测每个时间点的数值变化，也是最常用的预测方法，multi-sequence prediction 主要用来预测时间序列的局部趋势，full sequence prediction 主要用来预测时间序列的整体趋势。

第四章 时间序列分类任务

4.1 分类

分类任务和预测（回归任务）有异曲同工之妙，不同点在于在得到模型输出之后，需要单独增加一个激活函数，二元分类使用 sigmoid 激活函数，多元分类使用 softmax 激活函数。即(4.1.1)

$$y_c = \sigma\left(W_{hy} * y + b_{hy}\right) \tag{4.1.1}$$

式(4.1.1)中的 σ 表示激活函数。目标值也应该转换为相应的类别，一般使用 one-hot 编码表示。Sigmoid 或者 softmax 激活函数会将模型的输出转换为相应类别的概率值。

丞待完善。相关资料参考文献[21]、[22]、[23]、[24]、[25]及其引申文献。

第五章 DNN 在仿真数据上的实验

5.1 Granger Causality

在时间序列情形下，两个时间序列 x_i 、 x_j 之间的格兰杰因果关系（Granger Causality）定义为：若在包含了序列 x_i 、 x_j 的过去信息的条件下，对序列 x_j 的预测效果要优于只单独由 x_j 的过去信息对 x_j 进行的预测效果，即序列 x_i 有助于解释序列 x_j 的将来变化，则认为序列 x_i 是引致序列 x_j 的格兰杰原因。

假设 $\mathbf{x}$ 代表一个多元稳态时间序列，其阶为 P ，我们可以使用 MVAR 模型(5.1.1)来表述时间序列之间的关系：

$$\mathbf{x}(t) = \sum_{p=1}^{P} A_p \mathbf{x}(t-p) + bias + \varepsilon(t) \tag{5.1.1}$$

式(5.1.1)中的 A 是系数矩阵，$bias$ 是一个向量，其代表模型的偏移项，$\varepsilon(t)$ 是高斯噪声。如果时间序列 x_i 对时间序列 x_j 的预测有着显著的贡献，那么时间序列 x_i 就是时间序列 x_j 的格兰杰因（Granger cause）。为了从数量上来衡量这种贡献，我们提出如下形式的 MVAR 模型（式(5.1.2)）。

$$\mathbf{x}_{\setminus x_i}(t)=\sum_{p^*=1}^{P^*}A_{p^*}^{*}\mathbf{x}_{\setminus x_i}\left(t-p^*\right)+bias^*+\varepsilon_{\setminus x_i}^{*}(t) \tag{5.1.2}$$

式(5.1.2)中 A^* 是模型系数，$\mathbf{x}_{\setminus x_i}(t)$ 表示不包含 x_i 的 $\mathbf{x}$ 向量。所以，可以将从 x_i 到 x_j 的格兰杰因果（Granger Causality）定义[12]为：

$$G_{x_i\to x_j}=\ln\frac{\mathrm{var}\left(\varepsilon_{x_j\setminus x_i}^{*}(t)\right)}{\mathrm{var}\left(\varepsilon_{x_j}(t)\right)} \tag{5.1.3}$$

式(5.1.3)中 $\varepsilon_{x_j\setminus x_i}^{*}(t)$ 和 $\varepsilon_{x_j}(t)$ 分别是式(5.1.2)和式(5.1.1)中的预测误差，$\mathrm{var}(\cdot)$ 代表模型的统计误差。

5.2 Granger Causality with neural network

文献[20]提出了基于前馈神经网络的格兰杰因果，这里，重点讨论基于循环神经网络的格兰杰因果。**不同于前馈神经网络[12]，循环神经网络可以充分利用时间序列的时间信息，因此，不像自回归模型等在拟合数据时假设模型有一个固定的时延 lag（通常这个时延或者模型的阶是难以准确估计的），循环神经网络可以通过自身的循环学到各种会对当前时间序列产生影响的时延 lag，由于神经网络可以共享参数的特性在存在更大的 lag 时，模型的系数并不会增加太多，计算的时间复杂度也会相较于传统方法小。**参看 5.1 节，我们可以将格兰杰因果关系在神经网络上进行扩展，我们使用循环神经网络来对时间序列进行拟合或者预测。

$$\mathbf{x}(t)=\mathrm{RNN}\left(\mathbf{x}(t-p)\right)+\varepsilon(t) \tag{5.2.1}$$

$$\mathbf{x}_{\setminus x_i}(t)=\mathrm{RNN}\left(\mathbf{x}_{\setminus x_i}\left(t-p^*\right)\right)+\varepsilon_{\setminus x_i}^{*}(t) \tag{5.2.2}$$

$$G_{x_i\to x_j}=\ln\frac{\mathrm{var}\left(\varepsilon_{x_j\setminus x_i}^{*}(t)\right)}{\mathrm{var}\left(\varepsilon_{x_j}(t)\right)} \tag{5.2.3}$$

式(5.2.1)与式(5.2.2)中的 RNN 代表循环神经网络（LSTM 等）且带有偏移量 $bias$，p 和 p^* 可以理解为取值为 $[1,P]$ 的数值，P 代表窗宽，一般在神经网络的训练时，设置一个最大的阶，神经网络可以自动学习到达到最佳预测效果的最优模型阶。

5.3 仿真模型

1. 线性模型

$$\begin{cases}x_1(t)=0.95\sqrt{2}x_1(t-1)-0.9025x_1(t-2)+\varepsilon_1(t)\\ x_2(t)=0.5x_1(t-2)+\varepsilon_2(t)\\ x_3(t)=-0.4x_1(t-3)+\varepsilon_3(t)\\ x_4(t)=-0.5x_1(t-2)+0.25\sqrt{2}x_4(t-1)+0.25\sqrt{2}x_5(t-1)+\varepsilon_4(t)\\ x_5(t)=-0.25\sqrt{2}x_4(t-1)+0.25\sqrt{2}x_5(t-1)+\varepsilon_5(t)\end{cases} \tag{5.3.1}$$

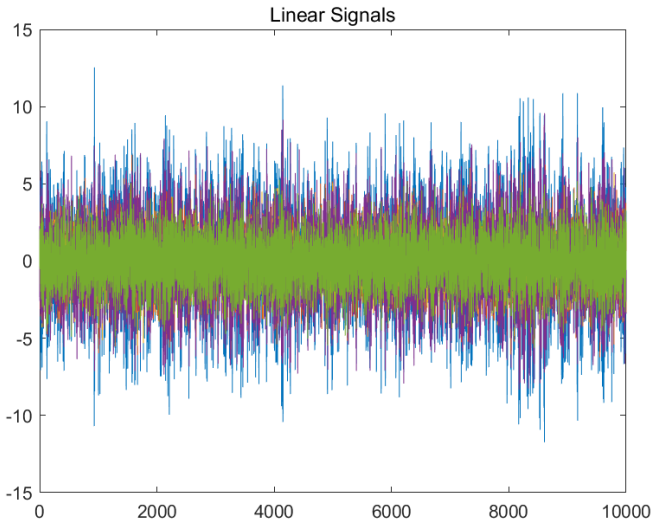


图 5.1 线性信号的基本信息

2. 非线性模型

$$\begin{cases} x_1(t) = 0.95\sqrt{2}x_1(t-1) - 0.9025x_1(t-2) + \varepsilon_1(t) \\ x_2(t) = 0.5x_1^2(t-2) + \varepsilon_2(t) \\ x_3(t) = -0.4x_1(t-3) + \varepsilon_3(t) \\ x_4(t) = -0.5x_1^2(t-2) + 0.25\sqrt{2}x_4(t-1) + 0.25\sqrt{2}x_5(t-1) + \varepsilon_4(t) \\ x_5(t) = -0.25\sqrt{2}x_4(t-1) + 0.25\sqrt{2}x_5(t-1) + \varepsilon_5(t) \end{cases} \tag{5.3.2}$$

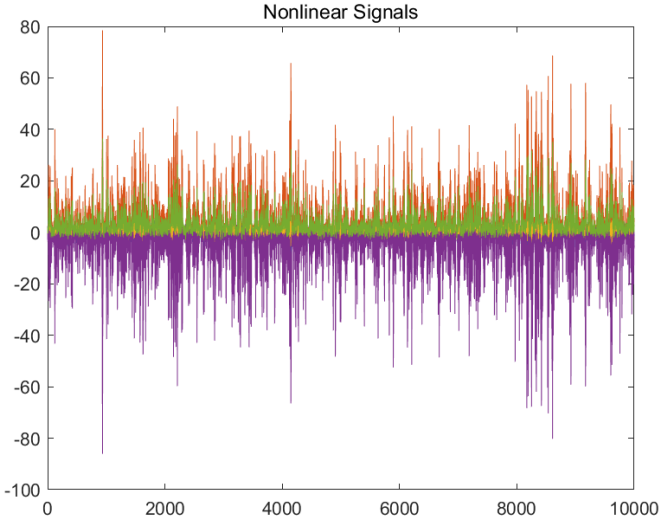


图 5.2 非线性信号的基本信息

3. 长时延非线性模型

$$\begin{cases} x_1(t) = 0.95\sqrt{2}x_1(t-1) - 0.9025x_1(t-2) + \varepsilon_1(t) \\ x_2(t) = 0.5x_1^2(t-10) + \varepsilon_2(t) \\ x_3(t) = -0.4x_1(t-3) + \varepsilon_3(t) \\ x_4(t) = -0.5x_1^2(t-2) + 0.25\sqrt{2}x_4(t-1) + 0.25\sqrt{2}x_5(t-1) + \varepsilon_4(t) \\ x_5(t) = -0.25\sqrt{2}x_4(t-1) + 0.25\sqrt{2}x_5(t-1) + \varepsilon_5(t) \end{cases} \tag{5.3.3}$$

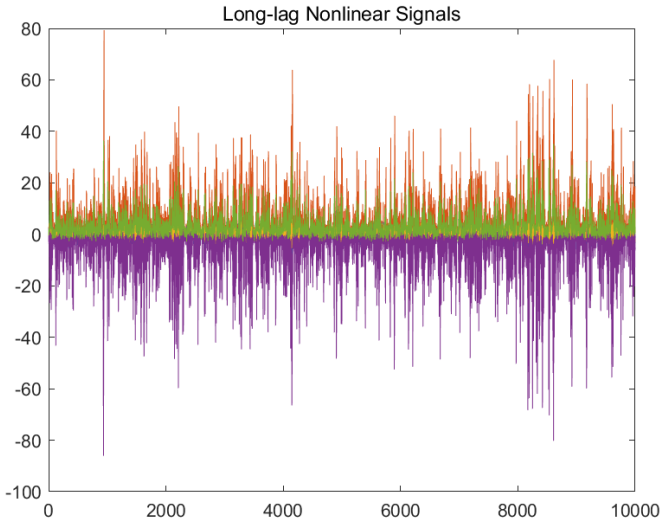


图 5.3 长时延非线性信号的基本信息

5.4 实验结果

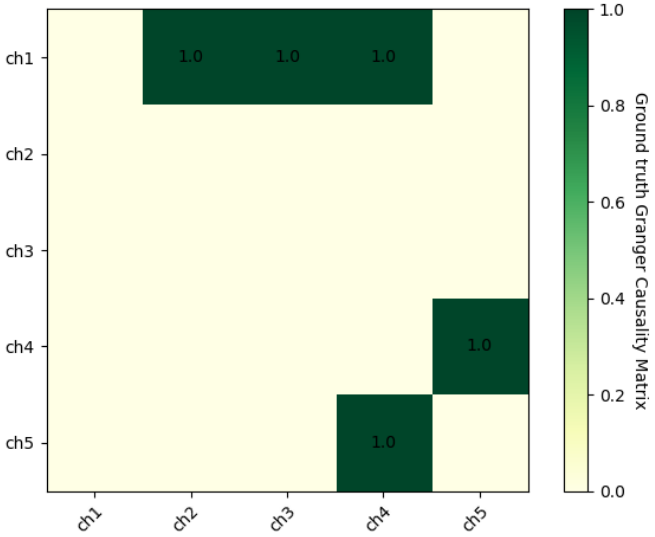


图 5.4 Ground Truth

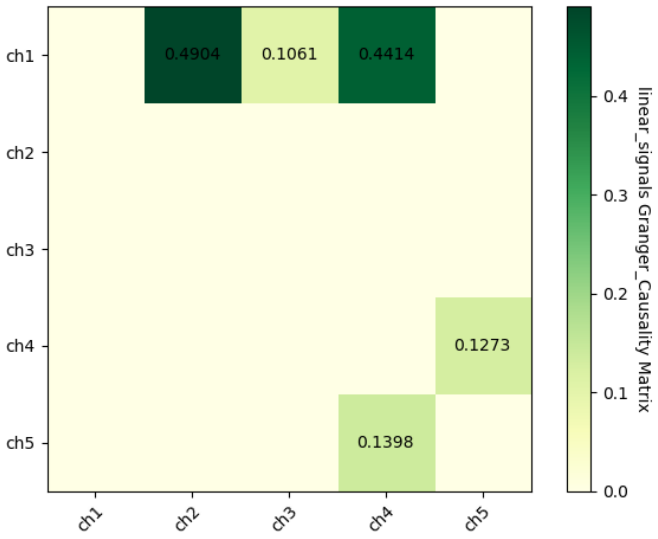


图 5.5 线性信号的实验结果

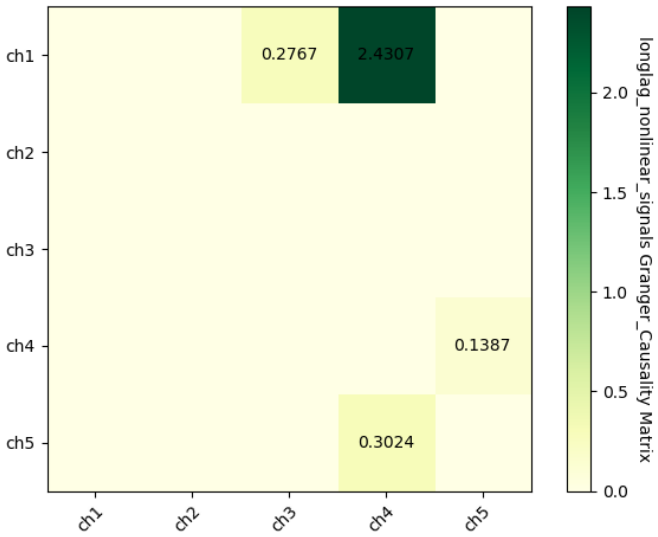


图 5.6 非线性信号的实验结果

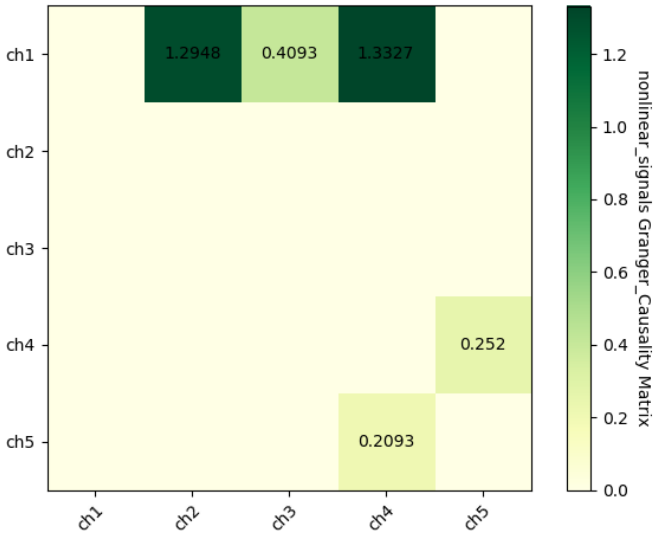


图 5.7 长时延非线性信号的实验结果

第六章 DNN 在 EEG 上的应用

6.1 应用

完善中，最新结果参考文献[26]

第七章 参考文献

[1] https://www.altumintelligence.com/articles/a/Time-Series-Prediction-Using-LSTM-Deep-Neural-Networks
[2] https://github.com/AutuanLiu/LSTM-Neural-Network-for-Time-Series-Prediction
[3] https://zh.wikipedia.org/wiki/%E6%99%82%E9%96%93%E5%BA%8F%E5%88%97
[4] https://machinelearningmastery.com/time-series-forecasting-supervised-learning/
[5] 李航. 统计学习方法[M]. 清华大学出版社, 2012.
[6] https://machinelearningmastery.com/backtest-machine-learning-models-time-series-forecasting/
[7] http://scikit-learn.org/stable/auto_examples/model_selection/plot_cv_indices.html#sphx-glr-auto-examples-model-selection-plot-cv-indices-py
[8]https://datamarket.com/data/set/22ti/zuerich-monthly-sunspot-numbers-1749-1983#!ds=22ti&display=line
[9] 王燕，应用时间序列分析（第四版），中国人民大学出版社，2015.
[10] Varotto G, Tassi L, Franceschetti S, et al. Epileptogenic networks of type II focal cortical dysplasia: a stereo-EEG study.[J]. Neuroimage, 2012, 61(3):591-598.
[11] https://github.com/AutuanLiu/Deep-Learning-for-Time-Series
[12] Wang, Y., Lin, K., Qi, Y., Lian, Q., Feng, S., Wu, Z., & Pan, G. (2018). Estimating Brain Connectivity With Varying-Length Time Lags Using a Recurrent Neural Network. IEEE Transactions on Biomedical Engi-neering, 65, 1953-1963.
[13] Jonathan Schiefer, Alexander Niederbühl, Volker Pernice, Carolin Lennartz, Pierre LeVan, Jürgen Henning, Stefan Rotter. From Correlation to Causation: Estimation of Effective Connectivity from Continuous Brain Signals based on Zero-Lag Covariance. 2018.
[14] Subhrajit Roy, Isabell Kiral-Kornek, Stefan Harrer. ChronoNet: A Deep Recurrent Neural Network for Abnormal EEG Identification. 2018
[15] http://colah.github.io/posts/2015-08-Understanding-LSTMs/
[16] https://pytorch.org/docs/stable/index.html
[17] Nick McClure 著，曾益强译，TensorFlow 机器学习实战指南，机械工业出版社
[18] 董豪等编著，深度学习：一起玩转 TensorLayer，电子工业出版社
[19] Cho, et al. (2014), Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation
[20] Montalto, A., Stramaglia, S., Faes, L., Tessitore, G., Prevete, R., & Marinazzo, D. (2015). Neural networks with non-uniform embedding and explicit validation phase to assess granger causality. *Neural Networks,71*(C), 159-171.
[21] Liu, C. L. , Hsaio, W. H. , & Tu, Y. C. . (0). Time series classification with multivariate convolutional neural network. IEEE Transactions on Industrial Electronics, PP(99), 1-1.
[22] Deep learning for time series classification: a review
[23] Data augmentation using synthetic data for time series classification with deep residual networks
[24] https://github.com/hfawaz/aaltd18; https://github.com/hfawaz/bigdata18; https://github.com/hfawaz/dl-4-tsc
[25] Transfer learning for time series classification
[26] https://github.com/AutuanLiu/Deep-Learning-for-Time-Series