Deep Learning / AI: A Brief Overview

By AI Assistant

# What is Deep Learning / AI?

Deep learning is a branch of artificial intelligence (AI) that uses artificial neural networks to learn from data and perform tasks such as image recognition, natural language processing, speech synthesis, and more. AI is the broader field of study that encompasses deep learning and other methods of creating intelligent systems.

# How does Deep Learning / AI work?

Deep learning / AI works by using algorithms called neural networks, which are composed of layers of interconnected units called neurons. Each neuron receives inputs from the previous layer, performs some computation, and sends outputs to the next layer. The network can learn from data by adjusting the weights of the connections between neurons, which determine how much each input influences the output. The network can also use activation functions, which decide whether a neuron should fire or not, and loss functions, which measure how well the network's output matches the desired output. By minimizing the loss function, the network can improve its performance on a given task.

# What are some applications of Deep Learning / AI?

* Image recognition: Deep learning / AI can recognize objects, faces, scenes, and more in images, and can also generate realistic images from text or noise.
* Natural language processing: Deep learning / AI can understand and generate natural language, such as text, speech, and dialogue, and can also translate between languages, summarize texts, answer questions, and more.
* Speech synthesis: Deep learning / AI can convert text to speech, and can also synthesize speech with different voices, accents, emotions, and styles.
* Reinforcement learning: Deep learning / AI can learn from its own actions and rewards, and can optimize its behavior for complex and dynamic environments, such as games, robotics, and self-driving cars.
* Generative adversarial networks: Deep learning / AI can use two competing networks, one that generates data and one that discriminates between real and fake data, to create realistic and diverse data, such as images, videos, music, and more.

# Basic algorithms of Deep Learning / AI

人工智能（AI）领域涵盖了许多不同的算法和技术，以下是一些常见的人工智能算法及其原理和应用场景：

1. 机器学习算法：

- 线性回归 (linear regression) ：用于建立输入特征和连续目标变量之间的关系。原理是通过拟合一条直线来最小化预测值与实际值之间的误差。应用场景包括房价预测、销售预测等。

- 逻辑回归 (logistic regression) ：主要用于分类问题，原理是通过对输入特征进行加权和加总，然后通过一个逻辑函数将结果映射到0和1之间，表示分类概率。应用场景包括垃圾邮件检测、客户流失预测等。

- 决策树与随机森林 (decision tree and random forest) ：通过对数据进行分层，构建一系列的决策规则来进行预测。应用场景包括医学诊断、信用评分等。

- 支持向量机（SVM, support vector machine) ：用于分类和回归分析，原理是找到一个最优的超平面来分割数据，使得不同类别的数据点尽可能远离超平面。应用场景包括图像分类、文本分类等。

2. 深度学习算法：

- 人工神经网络 (artificial neural networks) ：模仿人脑的神经网络结构，包括输入层、隐藏层和输出层。通过多层次的非线性转换来学习数据的复杂特征。应用场景包括图像识别、语音识别、自然语言处理等。

- 卷积神经网络（CNN, convolution neural networks）：专门用于处理具有类似网格结构的数据，如图像。通过卷积操作来提取图像中的特征，并通过池化操作减小数据规模。应用场景包括图像分类、目标检测等。

- 循环神经网络（RNN, Recurrent neural networks）：适用于处理序列数据，如时间序列数据或文本数据。通过记忆之前的状态来处理当前的输入，并且能够考虑到序列中的时间信息。应用场景包括语言建模、机器翻译等。

- 深度强化学习（Deep reinforcement learning）：通过智能体与环境的交互来学习最优策略，以最大化累积奖励。应用场景包括游戏智能体、自动驾驶等。

3. 聚类算法：

- K均值聚类：将数据集分成K个簇，每个数据点属于与其最近的簇。应用场景包括客户分群、图像分割等。

- 层次聚类：通过层次结构的方式来组织簇，从而形成树状结构。应用场景包括生物分类学、社交网络分析等。

4. 自然语言处理算法：

- 词袋模型（Bag of Words）：将文本表示为一个词汇表上的向量，忽略了单词的顺序和语法。应用场景包括文本分类、情感分析等。

- 词嵌入（Word Embedding）：将单词映射到一个低维度的向量空间中，使得具有相似含义的单词在空间中距离较近。应用场景包括语义搜索、自动摘要等。

以上只是人工智能领域中一些常见的算法及其应用场景，随着技术的不断发展，还会涌现出更多新的算法和方法。

全连接是什么意思

全连接是一种神经网络层的类型，也称为密集连接层或全连接层。在全连接层中，每个神经元与上一层的所有神经元都连接起来。这意味着前一层的每个神经元的输出都会作为输入传递给当前层的每个神经元。这样的连接方式使得每个神经元都可以对前一层的所有信息进行处理。

在全连接层中，每个连接都有一个权重，这些权重需要在训练过程中进行学习。通常，还会在每个神经元上添加一个偏置项，用于调整激活函数的输入值。因此，全连接层的参数数量取决于上一层的神经元数量以及当前层的神经元数量。

全连接层通常用于神经网络的前向传播过程中，可以通过矩阵乘法和加法操作来实现。全连接层常用于深度学习模型的最后几层，用于将高维特征映射到输出层，或者用于将输出层的结果映射到分类或回归任务的输出。

前向传播和后向传播分别是什么意思

前向传播和后向传播是神经网络中两个重要的过程，它们分别用于计算神经网络的输出和更新网络中的参数。

1. 前向传播（Forward Propagation）：

- 前向传播是指从神经网络的输入层开始，逐层计算每一层的输出，并将这些输出传递到下一层，直到到达输出层。在每一层中，通过使用权重矩阵和激活函数，将上一层的输出与当前层的权重相乘并加上偏置项，然后通过激活函数进行非线性变换，得到当前层的输出。这个过程可以被描述为一个数据流从网络的输入向输出的传播过程。

- 在前向传播过程中，神经网络不会改变其权重和偏置项，只是将输入数据在网络中进行传递和转换，最终得到输出结果。

2. 后向传播（Backward Propagation）：

- 后向传播是指在已知输入和输出的情况下，通过反向计算梯度来更新神经网络中的权重和偏置项。它利用损失函数对神经网络的输出与真实标签之间的差异进行求导，然后通过链式法则从输出层向输入层反向计算每一层的梯度，以便用于参数的更新。

- 后向传播可以看作是梯度下降算法的一种具体实现，通过不断地计算梯度并更新网络参数，使得网络的输出尽可能地接近真实标签，从而最小化损失函数。

综合来说，前向传播是将输入数据在神经网络中传递并得到输出的过程，而后向传播是利用已知的输入和输出来更新神经网络中的参数，以使得网络的输出更加接近期望的输出。这两个过程通常交替进行，直到网络的参数收敛到满意的程度。

# 一文看懂25个神经网络模型

[](https://www.zhihu.com/people/dongdong-mianmian)

[**面面**](https://www.zhihu.com/people/dongdong-mianmian)

数据分析师

​关注他

**创作声明：内容包含虚构创作**

259 人赞同了该文章

## 1. 引言

在深度学习十分火热的今天，不时会涌现出各种新型的人工神经网络，想要实时了解这些新型神经网络的架构还真是不容易。光是知道各式各样的神经网络模型缩写（如：DCIGN、BiLSTM、DCGAN……还有哪些？)，就已经让人招架不住了。

因此，这里整理出一份清单来梳理所有这些架构。其中大部分是人工神经网络，也有一些完全不同的怪物。尽管所有这些架构都各不相同、功能独特，当我在画它们的节点图时……其中潜在的关系开始逐渐清晰起来。

把这些架构做成节点图，会存在一个问题：它无法展示神经网络架构内部的工作原理。举例来说，变分自编码机（VAE：variational autoencoders ）看起来跟自编码机（AE：autoencoders）差不多，但它们的训练过程却大不相同。训练后的模型在使用场景上差别更大：VAE是生成器，通过插入噪音数据来获取新样本；而AE仅仅是把他们所收到的任何信息作为输入，映射到“记忆中”最相似的训练样本上。

在介绍不同模型的神经元和神经细胞层之间的连接方式前，我们一步一步来，先来了解不同的神经元节点内部是如何工作的。

## 1.1 神经元

对不同类型的神经元标记不同的颜色，可以更好地在各种网络架构之间进行区分。但是，这些神经元的工作方式却是大同小异。在下图的基本神经元结构后面，你会看到详细的讲解：

基本的人工神经网络神经元（basic neural network cell）相当简单，这种简单的类型可以在常规的前馈人工神经网络架构里面找到。这种神经元与其它神经元之间的连接具有权重，也就是说，它可以和前一层神经网络层中的所有神经元有连接。

每一个连接都有各自的权重，通常情况下是一些随机值（关于如何对人工神经网络的权重进行初始化是一个非常重要的话题，这将会直接影响到之后的训练过程，以及最终整个模型的性能）。这个权重可以是负值，正值，非常小，或者非常大，也可以是零。和这个神经元连接的所有神经元的值都会乘以各自对应的权重。然后，把这些值都求和。

在这个基础上，会额外加上一个bias，它可以用来避免输出为零的情况，并且能够加速某些操作，这让解决某个问题所需要的神经元数量也有所减少。这个bias也是一个数字，有些时候是一个常量（经常是-1或者1），有些时候会有所变化。这个总和最终被输入到一个激活函数，这个激活函数的输出最终就成为这个神经元的输出。

## 1.2卷积神经元（Convolutional cells）

和前馈神经元非常相似，除了它们只跟前一神经细胞层的部分神经元有连接。因为它们不是和某些神经元随机连接的，而是与特定范围内的神经元相连接，通常用来保存空间信息。这让它们对于那些拥有大量局部信息，比如图像数据、语音数据（但多数情况下是图像数据），会非常实用。

## 1.3 解卷积神经元

恰好相反：它们是通过跟下一神经细胞层的连接来解码空间信息。这两种神经元都有很多副本，它们都是独立训练的；每个副本都有自己的权重，但连接方式却完全相同。可以认为，这些副本是被放在了具备相同结构的不同的神经网络中。这两种神经元本质上都是一般意义上的神经元，但是，它们的使用方式却不同。

## 1.4 池化神经元和插值神经元（Pooling and interpolating cells）

经常和卷积神经元结合起来使用。它们不是真正意义上的神经元，只能进行一些简单的操作。

池化神经元接受到来自其它神经元的输出过后，决定哪些值可以通过，哪些值不能通过。在图像领域，可以理解成是把一个图像缩小了（在查看图片的时候，一般软件都有一个放大、缩小的功能；这里的图像缩小，就相当于软件上的缩小图像；也就是说我们能看到图像的内容更加少了；在这个池化的过程当中，图像的大小也会相应地减少）。这样，你就再也不能看到所有的像素了，池化函数会知道什么像素该保留，什么像素该舍弃。

插值神经元恰好是相反的操作：它们获取一些信息，然后映射出更多的信息。额外的信息都是按照某种方式制造出来的，这就好像在一张小分辨率的图片上面进行放大。插值神经元不仅仅是池化神经元的反向操作，而且，它们也是很常见，因为它们运行非常快，同时，实现起来也很简单。池化神经元和插值神经元之间的关系，就像卷积神经元和解卷积神经元之间的关系。

## 1.5均值神经元和标准方差神经元（Mean and standard deviation cells）（作为概率神经元它们总是成对地出现）

是一类用来描述数据概率分布的神经元。均值就是所有值的平均值，而标准方差描述的是这些数据偏离（两个方向）均值有多远。比如：一个用于图像处理的概率神经元可以包含一些信息，比如：在某个特定的像素里面有多少红色。举个例来说，均值可能是0.5，同时标准方差是0.2。当要从这些概率神经元取样的时候，你可以把这些值输入到一个高斯随机数生成器，这样就会生成一些分布在0.4和0.6之间的值；值离0.5越远，对应生成的概率也就越小。它们一般和前一神经元层或者下一神经元层是全连接，而且，它们没有偏差（bias）。

## 1.6 循环神经元（Recurrent cells ）

不仅仅在神经细胞层之间有连接，而且在时间轴上也有相应的连接。每一个神经元内部都会保存它先前的值。它们跟一般的神经元一样更新，但是，具有额外的权重：与当前神经元之前值之间的权重，还有大多数情况下，与同一神经细胞层各个神经元之间的权重。当前值和存储的先前值之间权重的工作机制，与非永久性存储器（比如RAM）的工作机制很相似，继承了两个性质：

* 第一，维持一个特定的状态；
* 第二：如果不对其持续进行更新（输入），这个状态就会消失。

由于先前的值是通过激活函数得到的，而在每一次的更新时，都会把这个值和其它权重一起输入到激活函数，因此，信息会不断地流失。实际上，信息的保存率非常的低，以至于仅仅四次或者五次迭代更新过后，几乎之前所有的信息都会流失掉。

## 1.7 长短期记忆神经元（Long short term memory cells）

用于克服循环神经元中信息快速流失的问题。

LSTM是一个逻辑回路，其设计受到了计算机内存单元设计的启发。与只存储两个状态的循环神经元相比，LSTM可以存储四个状态：输出值的当前和先前值，记忆神经元状态的当前值和先前值。它们都有三个门：输入门，输出门，遗忘门，同时，它们也还有常规的输入。

这些门它们都有各自的权重，也就是说，与这种类型的神经元细胞连接需要设置四个权重（而不是一个）。这些门的工作机制与流门（flow gates）很相似，而不是栅栏门（fence gates）：它们可以让所有的信息都通过，或者只是通过部分，也可以什么都不让通过，或者通过某个区间的信息。

这种运行机制的实现是通过把输入信息和一个在0到1之间的系数相乘，这个系数存储在当前门中。这样，输入门决定输入的信息有多少可以被叠加到当前门值。输出门决定有多少输出信息是可以传递到后面的神经网络中。遗忘门并不是和输出神经元的先前值相连接，而是，和前一记忆神经元相连接。它决定了保留多少记忆神经元最新的状态信息。因为没有和输出相连接，以及没有激活函数在这个循环中，因此只会有更少的信息流失。

## 1.8 门控循环神经元（Gated recurrent units (cells)）

是LSTM的变体。它们同样使用门来抑制信息的流失，但是只用两个门：更新门和重置门。这使得构建它们付出的代价没有那么高，而且运行速度更加快了，因为它们在所有的地方使用了更少的连接。

从本质上来说LSTM和GRU有两个不同的地方：

* 第一：GRU神经元没有被输出门保护的隐神经元；
* 第二：GRU把输出门和遗忘门整合在了一起，形成了更新门。核心的思想就是如果你想要一些新的信息，那么你就可以遗忘掉一些陈旧的信息（反过来也可以）。

## 1.9 神经细胞层(Layers)

形成一个神经网络，最简单的连接神经元方式是——把所有的神经元与其它所有的神经元相连接。这就好像Hopfield神经网络和玻尔兹曼机（Boltzmann machines）的连接方式。当然，这也就意味着连接数量会随着神经元个数的增加呈指数级地增加，但是，对应的函数表达力也会越来越强。这就是所谓的全连接（completely (or fully) connected）。

经历了一段时间的发展，发现把神经网络分解成不同的神经细胞层会非常有效。神经细胞层的定义是一群彼此之间互不连接的神经元，它们仅跟其它神经细胞层有连接。这一概念在受限玻尔兹曼机（Restricted Boltzmann Machines）中有所体现。现在，使用神经网络就意味着使用神经细胞层，并且是任意数量的神经细胞层。其中一个比较令人困惑的概念是全连接（fully connected or completely connected），也就是某一层的每个神经元跟另一层的所有神经元都有连接，但真正的全连接神经网络相当罕见。

## 1.10 卷积连接层（Convolutionally connected layers）

相对于全连接层要有更多的限制：在卷积连接层中的每一个神经元只与相邻的神经元层连接。图像和声音蕴含了大量的信息，如果一对一地输入到神经网络（比如，一个神经元对应一个像素）。卷积连接的形成，受益于保留空间信息更为重要的观察。实践证明这是一个非常好的猜测，因为现在大多数基于人工神经网络的图像和语音应用都使用了这种连接方式。然而，这种连接方式所需的代价远远低于全连接层的形式。从本质上来讲，卷积连接方式起到重要性过滤的作用，决定哪些紧紧联系在一起的信息包是重要的；卷积连接对于数据降维非常有用。

当然了，还有另外一种选择，就是随机连接神经元（randomly connected neurons）。这种形式的连接主要有两种变体：

* 第一，允许部分神经元进行全连接。
* 第二，神经元层之间只有部分连接。  
  随机连接方式有助于线性地降低人工神经网络的性能；当全连接层遇到性能问题的时候，在大规模人工神经网络中，使用随机连接方式非常有益。拥有更多神经元且更加稀疏的神经元层在某些情况下运行效果更好，特别是很多的信息需要被存储起来，但是，需要交换的信息并不多（这与卷积连接层的运行机制很相似，但是，它们是随机的）。非常稀疏的连接网络（1%或2%）也有被使用，比如ELMs, ESNs 和LSMs。这特别适用于脉冲网络（spiking networks），因为一个神经元拥有更多的连接，它对应的权重具有的能量也就更少，这也就意味着将会有更少的扩展和重复模式。

## 1.11 时间滞后连接（Time delayed connections）

是指相连的神经元（通常是在同一个神经元层，甚至于一个神经元自己跟自己连接），它们不从前面的神经元层获取信息，而是从神经元层先前的状态获取信息。这使得暂时（时间上或者序列上）联系在一起的信息可以被存储起来。这些形式的连接经常被手工重新进行设置，从而可以清除神经网络的状态。和常规连接的主要区别是，这种连接会持续不断地改变，即便这个神经网络当前没有处于训练状态。

下图展示了以上所介绍的神经网络及其连接方式。当我卡在哪种神经元与哪个神经细胞层该连到一起的时候，就会拿这张图出来作为参考（尤其是在处理和分析LSTM与GRU神经元时）：

显而易见，整理一份完整的清单是不切实际的，因为新的架构正被源源不断地发明出来。所以，接下来这份清单的目的，只想带你一窥人工智能领域的基础设施。对于每一个画成节点图的架构，我都会写一个非常非常简短的描述。你会发现这些描述还是很有用的，毕竟，总还是有一些你并不是那么熟悉的架构。

值得一提的是，虽说大多数的简写都已被普遍接受，但总会出现一些冲突。RNNs有时表示递归神经网络（recursive neural networks），但大多时候，它们指的是循环神经网络（recurrent neural networks）。这还没完，它们在许多地方还会泛指各种循环架构，这包括在LSTMs、GRU甚至是双向变体。AEs也经常会面临同样的问题，VAEs、DAEs及其相似结构有时都被简称为AEs。很多缩写后面的“N”也常常会有所变化，因为同一个架构你既可称之为卷积神经网络（convolutional neural network），也可简称为卷积网络（convolutional network），这样就出现了CNN和CN两种形式。

## 2. 前馈神经网络（FFNN）

前馈神经感知网络与感知机（FF or FFNN：Feed forward neural networks and P：perceptrons）非常简单，信息从前往后流动（分别对应输入和输出）。

一般在描述神经网络的时候，都是从它的层说起，即相互平行的输入层、隐含层或者输出层神经结构。单独的神经细胞层内部，神经元之间互不相连；而一般相邻的两个神经细胞层则是全连接（一层的每个神经元和另一层的每一个神经元相连）。一个最简单却最具有实用性的神经网络由两个输入神经元和一个输出神经元构成，也就是一个逻辑门模型。给神经网络一对数据集（分别是“输入数据集”和“我们期望的输出数据集”），一般通过反向传播算法来训练前馈神经网络（FFNNs）。

这就是所谓的监督式学习。与此相反的是无监督学习：我们只给输入，然后让神经网络去寻找数据当中的规律。反向传播的误差往往是神经网络当前输出和给定输出之间差值的某种变体（比如MSE或者仅仅是差值的线性变化）。如果神经网络具有足够的隐层神经元，那么理论上它总是能够建立输入数据和输出数据之间的关系。在实践中，FFNN的使用具有很大的局限性，但是，它们通常和其它神经网络一起组合成新的架构。

参考文献：  
[Rosenblatt, Frank. “The perceptron: a probabilistic model for information storage and organization in the brain.” Psychological review 65.6 (1958): 386.](https://link.zhihu.com/?target=http%3A//www.ling.upenn.edu/courses/cogs501/Rosenblatt1958.pdf)

## 3. 径向基神经网络（RBF）

径向神经网络（RBF：Radial basis function）是一种以径向基核函数作为激活函数的前馈神经网络。没有更多描述了。这不是说没有相关的应用，但大多数以其它函数作为激活函数的FFNNs都没有它们自己的名字。这或许跟它们的发明年代有关系。

参考文献：  
[Broomhead, David S., and David Lowe. Radial basis functions, multi-variable functional interpolation and adaptive networks. No. RSRE-MEMO-4148. ROYAL SIGNALS AND RADAR ESTABLISHMENT MALVERN (UNITED KINGDOM), 1988.](https://link.zhihu.com/?target=http%3A//www.dtic.mil/cgi-bin/GetTRDoc%3FAD%3DADA196234)

## 4. 霍普菲尔网络（HN）

霍普菲尔网络（HN：Hopfield network）是一种每一个神经元都跟其它神经元相互连接的网络。

这就像一盘完全搅在一起的意大利面，因为每个神经元都在充当所有角色：训练前的每一个节点都是输入神经元，训练阶段是隐神经元，输出阶段则是输出神经元。

该神经网络的训练，是先把神经元的值设置到期望模式，然后计算相应的权重。在这以后，权重将不会再改变了。一旦网络被训练包含一种或者多种模式，这个神经网络总是会收敛于其中的某一种学习到的模式，因为它只会在某一个状态才会稳定。值得注意的是，它并不一定遵从那个期望的状态（很遗憾，它并不是那个具有魔法的黑盒子）。它之所以会稳定下来，部分要归功于在训练期间整个网络的“能量（Energy）”或“温度（Temperature）”会逐渐地减少。每一个神经元的激活函数阈值都会被设置成这个温度的值，一旦神经元输入的总和超过了这个阈值，那么就会让当前神经元选择状态（通常是-1或1，有时也是0或1）。

可以多个神经元同步，也可以一个神经元一个神经元地对网络进行更新。一旦所有的神经元都已经被更新，并且它们再也没有改变，整个网络就算稳定（退火）了，那你就可以说这个网络已经收敛了。这种类型的网络被称为“联想记忆（associative memory）”，因为它们会收敛到和输入最相似的状态；比如，人类看到桌子的一半就可以想象出另外一半；与之相似，如果输入一半噪音+一半桌子，这个网络就能收敛到整张桌子。

参考文献：  
[Hopfield, John J. “Neural networks and physical systems with emergent collective computational abilities.” Proceedings of the national academy of sciences 79.8 (1982): 2554-2558.](https://link.zhihu.com/?target=https%3A//bi.snu.ac.kr/Courses/g-ai09-2/hopfield82.pdf)

## 5. 马尔可夫链（MC）

马尔可夫链（MC：Markov Chain）或离散时间马尔可夫链（DTMC：MC or discrete time Markov Chain）在某种意义上是BMs和HNs的前身。可以这样来理解：从从我当前所处的节点开始，走到任意相邻节点的概率是多少呢？它们没有记忆（所谓的马尔可夫特性）：你所得到的每一个状态都完全依赖于前一个状态。尽管算不上神经网络，但它却跟神经网络类似，并且奠定了BM和HN的理论基础。跟BM、RBM、HN一样，MC并不总被认为是神经网络。此外，它也并不总是全连接的。

参考文献：  
[Hayes, Brian. “First links in the Markov chain.” American Scientist 101.2 (2013): 252.](https://link.zhihu.com/?target=http%3A//www.americanscientist.org/libraries/documents/201321152149545-2013-03Hayes.pdf)

## 6. 玻尔兹曼机（BM）

玻尔兹曼机（BM：Boltzmann machines）和霍普菲尔网络很接近，差别只是：一些神经元作为输入神经元，剩余的则是作为隐神经元。

在整个神经网络更新过后，输入神经元成为输出神经元。刚开始神经元的权重都是随机的，通过反向传播（back-propagation）算法进行学习，或是最近常用的对比散度（contrastive divergence）算法（马尔可夫链用于计算两个信息增益之间的梯度）。

相比HN，大多数BM的神经元激活模式都是二元的。BM由MC训练获得，因而是一个随机网络。BM的训练和运行过程，跟HN大同小异：为输入神经元设好钳位值，而后让神经网络自行学习。因为这些神经元可能会得到任意的值，我们反复地在输入和输出神经元之间来回地进行计算。激活函数的激活受全局温度的控制，如果全局温度降低了，那么神经元的能量也会相应地降低。这个能量上的降低导致了它们激活模式的稳定。在正确的温度下，这个网络会抵达一个平衡状态。  
参考文献：  
[Hinton, Geoffrey E., and Terrence J. Sejnowski. “Learning and releaming in Boltzmann machines.” Parallel distributed processing: Explorations in the microstructure of cognition 1 (1986): 282-317.](https://link.zhihu.com/?target=https%3A//www.researchgate.net/profile/Terrence_Sejnowski/publication/242509302_Learning_and_relearning_in_Boltzmann_machines/links/54a4b00f0cf256bf8bb327cc.pdf)

## 8. 受限玻尔兹曼机（RBM）

受限玻尔兹曼机（RBM：Restricted Boltzmann machines）与BM出奇地相似，因而也同HN相似。

它们的最大区别在于：RBM更具实用价值，因为它们受到了更多的限制。它们不会随意在所有神经元间建立连接，而只在不同神经元群之间建立连接，因此任何输入神经元都不会同其他输入神经元相连，任何隐神经元也不会同其他隐神经元相连。

RBM的训练方式就像稍微修改过的FFNN：前向通过数据之后再将这些数据反向传回（回到第一层），而非前向通过数据然后反向传播误差。之后，再使用前向和反向传播进行训练。

参考文献：  
[Smolensky, Paul. Information processing in dynamical systems: Foundations of harmony theory. No. CU-CS-321-86. COLORADO UNIV AT BOULDER DEPT OF COMPUTER SCIENCE, 1986.](https://link.zhihu.com/?target=http%3A//www.dtic.mil/cgi-bin/GetTRDoc%3FLocation%3DU2%26doc%3DGetTRDoc.pdf%26AD%3DADA620727)

## 9. 自编码机（AE）

自编码机（AE：Autoencoders）和FFNN有些相近，因为它更像是FFNN的另一种用法，而非本质上完全不同的另一种架构。

自编码机的基本思想是自动对信息进行编码（像压缩一样，而非加密），它也因此而得名。整个网络的形状酷似一个沙漏计时器，中间的隐含层较小，两边的输入层、输出层较大。自编码机总是对称的，以中间层（一层还是两层取决于神经网络层数的奇偶）为轴。最小的层（一层或者多层）总是在中间，在这里信息压缩程度最大（整个网络的关隘口）。在中间层之前为编码部分，中间层之后为解码部分，中间层则是编码部分。

自编码机可用反向传播算法进行训练，给定输入，将误差设为输入和输出之差。自编码机的权重也是对称的，因此编码部分权重与解码部分权重完全一样。

参考文献:  
[Bourlard, Hervé, and Yves Kamp. “Auto-association by multilayer perceptrons and singular value decomposition.” Biological cybernetics 59.4-5 (1988): 291-294.](https://link.zhihu.com/?target=https%3A//pdfs.semanticscholar.org/f582/1548720901c89b3b7481f7500d7cd64e99bd.pdf)

## 10. 稀疏自编码机（SAE）

稀疏自编码机（SAE：Sparse autoencoders）某种程度上同自编码机相反。稀疏自编码机不是用更小的空间表征大量信息，而是把原本的信息编码到更大的空间内。因此，中间层不是收敛，而是扩张，然后再还原到输入大小。它可以用于提取数据集内的小特征。

如果用训练自编码机的方式来训练稀疏自编码机，几乎所有的情况，都是得到毫无用处的恒等网络（输入=输出，没有任何形式的变换或分解）。为避免这种情况，需要在反馈输入中加上稀疏驱动数据。稀疏驱动的形式可以是阈值过滤，这样就只有特定的误差才会反向传播用于训练，而其它的误差则被忽略为0，不会用于反向传播。这很像脉冲神经网络（并不是所有的神经元一直都会输出）。

[Marc’Aurelio Ranzato, Christopher Poultney, Sumit Chopra, and Yann LeCun. “Efficient learning of sparse representations with an energy-based model.” Proceedings of NIPS. 2007.](https://link.zhihu.com/?target=https%3A//papers.nips.cc/paper/3112-efficient-learning-of-sparse-representations-with-an-energy-based-model.pdf)

## 11.变分自编码机（VAE）

变分自编码机（VAE：Variational autoencoders）和AE有着相同的架构，却被教会了不同的事情：输入样本的一个近似概率分布，这让它跟BM、RBM更相近。

不过，VAE却依赖于贝叶斯理论来处理概率推断和独立（probabilistic inference and independence），以及重新参数化（re-parametrisation）来进行不同的表征。推断和独立非常直观，但却依赖于复杂的数学理论。基本原理是：把影响纳入考虑。如果在一个地方发生了一件事情，另外一件事情在其它地方发生了，它们不一定就是关联在一起的。如果它们不相关，那么误差传播应该考虑这个因素。这是一个有用的方法，因为神经网络是一个非常大的图表，如果你能在某些节点排除一些来自于其它节点的影响，随着网络深度地增加，这将会非常有用。

[Kingma, Diederik P., and Max Welling. “Auto-encoding variational bayes.” arXiv preprint arXiv:1312.6114 (2013).](https://link.zhihu.com/?target=https%3A//arxiv.org/pdf/1312.6114v10.pdf)

## 12. 去噪自编码机（DAE）

去噪自编码机（DAE：Denoising autoencoders）是一种自编码机，它的训练过程，不仅要输入数据，还有再加上噪音数据（就好像让图像变得更加模糊一样）。

但在计算误差的时候跟自动编码机一样，降噪自动编码机的输出也是和原始的输入数据进行对比。这种形式的训练旨在鼓励降噪自编码机不要去学习细节，而是一些更加宏观的特征，因为细微特征受到噪音的影响，学习细微特征得到的模型最终表现出来的性能总是很差。

Vincent, Pascal, et al. “Extracting and composing robust features with denoising autoencoders.” Proceedings of the 25th international conference on Machine learning. ACM, 2008.  
[http://machinelearning.org/archive/icml2008/papers/592.pdf](https://link.zhihu.com/?target=http%3A//machinelearning.org/archive/icml2008/papers/592.pdf)

## 13. 深度信念网络（DBN）

深度信念网络（DBN：Deep belief networks）之所以取这个名字，是由于它本身几乎是由多个受限玻尔兹曼机或者变分自编码机堆砌而成。

实践表明一层一层地对这种类型的神经网络进行训练非常有效，这样每一个自编码机或者受限玻尔兹曼机只需要学习如何编码前一神经元层的输出。这种训练技术也被称为贪婪训练，这里贪婪的意思是通过不断地获取局部最优解，最终得到一个相当不错解（但可能不是全局最优的）。可以通过对比散度算法或者反向传播算法进行训练，它会慢慢学着以一种概率模型来表征数据，就好像常规的自编码机或者受限玻尔兹曼机。一旦经过非监督式学习方式，训练或者收敛到了一个稳定的状态，那么这个模型就可以用来产生新的数据。如果以对比散度算法进行训练，那么它甚至可以用于区分现有的数据，因为那些神经元已经被引导来获取数据的不同特定。

Bengio, Yoshua, et al. “Greedy layer-wise training of deep networks.” Advances in neural information processing systems 19 (2007): 153.  
[https://papers.nips.cc/paper/3048-greedy-layer-wise-training-of-deep-networks.pdf](https://link.zhihu.com/?target=https%3A//papers.nips.cc/paper/3048-greedy-layer-wise-training-of-deep-networks.pdf)

## 14. 卷积神经网络（CNN）

卷积神经网络（CNN：Convolutional neural networks）或深度卷积神经网络（DCNN：deep convolutional neural networks）跟其它类型的神经网络大有不同。它们主要用于处理图像数据，但可用于其它形式数据的处理，如语音数据。对于卷积神经网络来说，一个典型的应用就是给它输入一个图像，而后它会给出一个分类结果。也就是说，如果你给它一张猫的图像，它就输出“猫”；如果你给一张狗的图像，它就输出“狗”。

卷积神经网络是从一个数据扫描层开始，这种形式的处理并没有尝试在一开始就解析整个训练数据。比如：对于一个大小为200X200像素的图像，你不会想构建一个40000个节点的神经元层。而是，构建一个20X20像素的输入扫描层，然后，把原始图像第一部分的20X20像素图像（通常是从图像的左上方开始）输入到这个扫描层。当这部分图像（可能是用于进行卷积神经网络的训练）处理完，你会接着处理下一部分的20X20像素图像：逐渐（通常情况下是移动一个像素，但是，移动的步长是可以设置的）移动扫描层，来处理原始数据。

注意，你不是一次性移动扫描层20个像素（或其它任何扫描层大小的尺度），也不是把原始图像切分成20X20像素的图像块，而是用扫描层在原始图像上滑过。这个输入数据（20X20像素的图像块）紧接着被输入到卷积层，而非常规的神经细胞层——卷积层的节点不是全连接。每一个输入节点只会和最近的那个神经元节点连接（至于多近要取决于具体的实现，但通常不会超过几个）。

这些卷积层会随着深度的增加而逐渐变小：大多数情况下，会按照输入层数量的某个因子缩小（比如：20个神经元的卷积层，后面是10个神经元的卷积层，再后面就是5个神经元的卷积层）。2的n次方（32, 16, 8, 4, 2, 1）也是一个非常常用的因子，因为它们在定义上可以简洁且完整地除尽。除了卷积层，池化层（pooling layers）也非常重要。

池化是一种过滤掉细节的方式：一种常用的池化方式是最大池化，比如用2X2的像素，然后取四个像素中值最大的那个传递。为了让卷积神经网络处理语音数据，需要把语音数据切分，一段一段输入。在实际应用中，通常会在卷积神经网络后面加一个前馈神经网络，以进一步处理数据，从而对数据进行更高水平的非线性抽象。

LeCun, Yann, et al. “Gradient-based learning applied to document recognition.” Proceedings of the IEEE 86.11 (1998): 2278-2324.  
[http://yann.lecun.com/exdb/publis/pdf/lecun-98.pdf](https://link.zhihu.com/?target=http%3A//yann.lecun.com/exdb/publis/pdf/lecun-98.pdf)

## 15. 解卷积网络（DN）

解卷积网络（DN：Deconvolutional networks），又称为逆图形网络（IGNs：inverse graphics networks），是逆向的卷积神经网络。

想象一下，给一个神经网络输入一个“猫”的词，就可以生成一个像猫一样的图像，通过比对它和真实的猫的图片来进行训练。跟常规CNN一样，DN也可以结合FFNN使用，但没必要为这个新的缩写重新做图解释。它们可被称为深度解卷积网络，但把FFNN放到DNN前面和后面是不同的，那是两种架构（也就需要两个名字），对于是否需要两个不同的名字你们可能会有争论。需要注意的是，绝大多数应用都不会把文本数据直接输入到神经网络，而是用二元输入向量。比如<0,1>代表猫，<1,0>代表狗，<1,1>代表猫和狗。

CNN的池化层往往也是被对应的逆向操作替换了，主要是插值和外推（基于一个基本的假设：如果一个池化层使用了最大池化，你可以在逆操作的时候生成一些相对于最大值更小的数据）。

Zeiler, Matthew D., et al. “Deconvolutional networks.” Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on. IEEE, 2010.  
[http://www.matthewzeiler.com/pubs/cvpr2010/cvpr2010.pdf](https://link.zhihu.com/?target=http%3A//www.matthewzeiler.com/pubs/cvpr2010/cvpr2010.pdf)

## 16. 深度卷积逆向图网络（DCIGN）

深度卷积逆向图网络（DCIGN：Deep convolutional inverse graphics networks），这个名字具有误导性，因为它们实际上是VAE，但分别用CNN、DNN来作编码和解码的部分。

这些网络尝试在编码过程中对“特征“进行概率建模，这样一来，你只要用猫和狗的独照，就能让它们生成一张猫和狗的合照。同理，你可以输入一张猫的照片，如果猫旁边有一只恼人的邻家狗，你可以让它们把狗去掉。很多演示表明，这种类型的网络能学会基于图像的复杂变换，比如灯光强弱的变化、3D物体的旋转。一般也是用反向传播算法来训练此类网络。

Kulkarni, Tejas D., et al. “Deep convolutional inverse graphics network.” Advances in Neural Information Processing Systems. 2015.  
[https://arxiv.org/pdf/1503.03167v4.pdf](https://link.zhihu.com/?target=https%3A//arxiv.org/pdf/1503.03167v4.pdf)

## 17. 生成式对抗网络（GAN）

生成式对抗网络（GAN：Generative adversarial networks）是一类不同的网络，它们有一对“双胞胎”：两个网络协同工作。

GAN可由任意两种网络组成（但通常是FF和CNN），其中一个用于生成内容，另一个则用于鉴别生成的内容。

鉴别网络（discriminating network）同时接收训练数据和生成网络（generative network）生成的数据。鉴别网络的准确率，被用作生成网络误差的一部分。这就形成了一种竞争：鉴别网络越来越擅长于区分真实的数据和生成数据，而生成网络也越来越善于生成难以预测的数据。这种方式非常有效，部分是因为：即便相当复杂的类噪音模式最终都是可预测的，但跟输入数据有着极为相似特征的生成数据，则很难区分。

训练GAN极具挑战性，因为你不仅要训练两个神经网络（其中的任何一个都会出现它自己的问题），同时还要平衡两者的运行机制。如果预测或生成相比对方表现得过好，这个GAN就不会收敛，因为它会内部发散。

Goodfellow, Ian, et al. “Generative adversarial nets.” Advances in Neural Information Processing Systems. 2014.  
[https://arxiv.org/pdf/1406.2661v1.pdf](https://link.zhihu.com/?target=https%3A//arxiv.org/pdf/1406.2661v1.pdf)

## 18. 循环神经网络（RNN）

循环神经网络（RNN：Recurrent neural networks）是具有时间联结的前馈神经网络：它们有了状态，通道与通道之间有了时间上的联系。神经元的输入信息，不仅包括前一神经细胞层的输出，还包括它自身在先前通道的状态。

这就意味着：你的输入顺序将会影响神经网络的训练结果：相比先输入“曲奇饼”再输入“牛奶”，先输入“牛奶”再输入“曲奇饼”后，或许会产生不同的结果。RNN存在一大问题：梯度消失（或梯度爆炸，这取决于所用的激活函数），信息会随时间迅速消失，正如FFNN会随着深度的增加而失去信息一样。

直觉上，这不算什么大问题，因为这些都只是权重，而非神经元的状态，但随时间变化的权重正是来自过去信息的存储；如果权重是0或1000000，那之前的状态就不再有信息价值。

原则上，RNN可以在很多领域使用，因为大部分数据在形式上不存在时间线的变化，（不像语音或视频），它们能以某种序列的形式呈现出来。一张图片或一段文字可以一个像素或者一个文字地进行输入，因此，与时间相关的权重描述了该序列前一步发生了什么，而不是多少秒之前发生了什么。一般来说，循环神经网络是推测或补全信息很好的选择，比如自动补全。

Elman, Jeffrey L. “Finding structure in time.” Cognitive science 14.2 (1990): 179-211.  
[https://crl.ucsd.edu/~elman/Papers/fsit.pdf](https://link.zhihu.com/?target=https%3A//crl.ucsd.edu/~elman/Papers/fsit.pdf)

## 19. 长短期记忆（LSTM）

长短期记忆（LSTM：Long / short term memory）网络试图通过引入门结构与明确定义的记忆单元来解决梯度消失/爆炸的问题。

这更多的是受电路图设计的启发，而非生物学上某种和记忆相关机制。每个神经元都有一个记忆单元和三个门：输入门、输出门、遗忘门。这三个门的功能就是通过禁止或允许信息流动来保护信息。

输入门决定了有多少前一神经细胞层的信息可留在当前记忆单元，输出层在另一端决定下一神经细胞层能从当前神经元获取多少信息。遗忘门乍看很奇怪，但有时候遗忘部分信息是很有用的：比如说它在学习一本书，并开始学一个新的章节，那遗忘前面章节的部分角色就很有必要了。

实践证明，LSTM可用来学习复杂的序列，比如像莎士比亚一样写作，或创作全新的音乐。值得注意的是，每一个门都对前一神经元的记忆单元赋有一个权重，因此会需要更多的计算资源。

Hochreiter, Sepp, and Jürgen Schmidhuber. “Long short-term memory.” Neural computation 9.8 (1997): 1735-1780.  
[http://deeplearning.cs.cmu.edu/pdfs/Hochreiter97\_lstm.pdf](https://link.zhihu.com/?target=http%3A//deeplearning.cs.cmu.edu/pdfs/Hochreiter97_lstm.pdf)

## 20. 门循环单元（GRU）

门循环单元（GRU : Gated recurrent units）是LSTM的一种轻量级变体。它们少了一个门，同时连接方式也稍有不同：它们采用了一个更新门（update gate），而非LSTM所用的输入门、输出门、遗忘门。

更新门决定了保留多少上一个状态的信息，还决定了收取多少来自前一神经细胞层的信息。重置门（reset gate）跟LSTM遗忘门的功能很相似，但它存在的位置却稍有不同。它们总是输出完整的状态，没有输出门。多数情况下，它们跟LSTM类似，但最大的不同是：GRU速度更快、运行更容易（但函数表达力稍弱）。

在实践中，这里的优势和劣势会相互抵消：当你你需要更大的网络来获取函数表达力时，这样反过来，性能优势就被抵消了。在不需要额外的函数表达力时，GRU的综合性能要好于LSTM。

Chung, Junyoung, et al. “Empirical evaluation of gated recurrent neural networks on sequence modeling.” arXiv preprint arXiv:1412.3555 (2014).  
[https://arxiv.org/pdf/1412.3555v1.pdf](https://link.zhihu.com/?target=https%3A//arxiv.org/pdf/1412.3555v1.pdf)

## 21. 神经图灵机（NTM）

神经图灵机（NTM: Neural Turing machines）可以理解为对LSTM的抽象，它试图把神经网络去黑箱化（以窥探其内部发生的细节）。

NTM不是把记忆单元设计在神经元内，而是分离出来。NTM试图结合常规数字信息存储的高效性、永久性与神经网络的效率及函数表达能力。它的想法是设计一个可作内容寻址的记忆库，并让神经网络对其进行读写操作。NTM名字中的“图灵（Turing）”是表明，它是图灵完备（Turing complete）的，即具备基于它所读取的内容来读取、写入、修改状态的能力，也就是能表达一个通用图灵机所能表达的一切。

Graves, Alex, Greg Wayne, and Ivo Danihelka. “Neural turing machines.” arXiv preprint arXiv:1410.5401 (2014).  
[https://arxiv.org/pdf/1410.5401v2.pdf](https://link.zhihu.com/?target=https%3A//arxiv.org/pdf/1410.5401v2.pdf)

## 22. BiRNN、BiLSTM、BiGRU

双向循环神经网络（BiRNN：Bidirectional recurrent neural networks）、双向长短期记忆网络（BiLSTM：bidirectional long / short term memory networks ）和双向门控循环单元（BiGRU：bidirectional gated recurrent units）在图表中并未呈现出来，因为它们看起来与其对应的单向神经网络结构一样。

所不同的是，这些网络不仅与过去的状态有连接，而且与未来的状态也有连接。比如，通过一个一个地输入字母，训练单向的LSTM预测“鱼（fish）”（在时间轴上的循环连接记住了过去的状态值）。在BiLSTM的反馈通路输入序列中的下一个字母，这使得它可以了解未来的信息是什么。这种形式的训练使得该网络可以填充信息之间的空白，而不是预测信息。因此，它在处理图像时不是扩展图像的边界，而是填补一张图片中的缺失。

Schuster, Mike, and Kuldip K. Paliwal. “Bidirectional recurrent neural networks.” IEEE Transactions on Signal Processing 45.11 (1997): 2673-2681.  
[http://www.di.ufpe.br/~fnj/RNA/bibliografia/BRNN.pdf](https://link.zhihu.com/?target=http%3A//www.di.ufpe.br/~fnj/RNA/bibliografia/BRNN.pdf)

## 23. 深度残差网络（DRN）

深度残差网络（DRN: Deep residual networks）是非常深的FFNN网络，它有一种特殊的连接，可以把信息从某一神经细胞层传至后面几层（通常是2到5层）。

该网络的目的不是要找输入数据与输出数据之间的映射，而是致力于构建输入数据与输出数据+输入数据之间的映射函数。本质上，它在结果中增加一个恒等函数，并跟前面的输入一起作为后一层的新输入。结果表明，当层数超过150后，这一网络将非常擅于学习模式，这比常规的2到5层要多得多。然而，有证据表明这些网络本质上只是没有时间结构的RNN，它们总是与没有门结构的LSTM相提并论。

He, Kaiming, et al. “Deep residual learning for image recognition.” arXiv preprint arXiv:1512.03385 (2015).  
[https://arxiv.org/pdf/1512.03385v1.pdf](https://link.zhihu.com/?target=https%3A//arxiv.org/pdf/1512.03385v1.pdf)

## 24. 回声状态网络（ESN）

回声状态网络（ESN：Echo state networks）是另一种不同类型的（循环）网络。

它的不同之处在于：神经元之间的连接是随机的（没有整齐划一的神经细胞层），其训练过程也有所不同。不同于输入数据后反向传播误差，ESN先输入数据、前馈、而后更新神经元状态，最后来观察结果。它的输入层和输出层在这里扮演的角色不太常规，输入层用来主导网络，输出层作为激活模式的观测器随时间展开。在训练过程中，只有观测和隐藏单元之间连接会被改变。

Jaeger, Herbert, and Harald Haas. “Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication.” science 304.5667 (2004): 78-80.  
[https://pdfs.semanticscholar.org/8922/17bb82c11e6e2263178ed20ac23db6279c7a.pdf](https://link.zhihu.com/?target=https%3A//pdfs.semanticscholar.org/8922/17bb82c11e6e2263178ed20ac23db6279c7a.pdf)

## 25. 极限学习机（ELM）

极限学习机（ELM：Extreme learning machines）本质上是拥有随机连接的FFNN。

它们与LSM、ESN极为相似，除了循环特征和脉冲性质，它们还不使用反向传播。相反，它们先给权重设定随机值，然后根据最小二乘法拟合来一次性训练权重（在所有函数中误差最小）。这使ELM的函数拟合能力较弱，但其运行速度比反向传播快多了。

Cambria, Erik, et al. “Extreme learning machines [trends & controversies].” IEEE Intelligent Systems 28.6 (2013): 30-59.  
[http://www.ntu.edu.sg/home/egbhuang/pdf/ieee-is-elm.pdf](https://link.zhihu.com/?target=http%3A//www.ntu.edu.sg/home/egbhuang/pdf/ieee-is-elm.pdf)

## 26. 液态机（LSM）

液态机（LSM：Liquid state machines）换汤不换药，跟ESN同样相近。

区别在于，LSM是一种脉冲神经网络（spiking neural networks），用阈值激活函数（threshold functions）取代了sigmoid激活函数，每个神经元同时也是具有累加性质的记忆单元。因此，当神经元状态更新时，其值不是相邻神经元的累加值，而是它自身状态值的累加。一旦累加到阈值，它就释放能量至其它神经元。这就形成了一种类似于脉冲的模式：神经元不会进行任何操作，直至到达阈值的那一刻。

Maass, Wolfgang, Thomas Natschläger, and Henry Markram. “Real-time computing without stable states: A new framework for neural computation based on perturbations.” Neural computation 14.11 (2002): 2531-2560.  
[https://web.archive.org/web/20120222154641/http://ramsesii.upf.es/seminar/Maass\_et\_al\_2002.pdf](https://link.zhihu.com/?target=https%3A//web.archive.org/web/20120222154641/http%3A//ramsesii.upf.es/seminar/Maass_et_al_2002.pdf)

## 27. 支持向量机（SVM）

支持向量机（SVM：Support vector machines）能为分类问题找出最优方案。

传统意义上，它们只能处理线性可分的数据；比如找出哪张图片是加菲猫、哪张是史努比，此外就无法做其它输出了。

训练过程中，SVM可以理解为：先在平面图表上标绘所有数据（加菲猫、史努比），然后找出到那条能够最好区分这两类数据点的线。这条线能把数据分为两部分，线的这边全是史努比，线的那边全是加菲猫。而后移动并优化该直线，令两边数据点到直线的距离最大化。分类新的数据，则将该数据点画在这个图表上，然后察看这个数据点在分隔线的哪一边（史努比一侧，还是加菲猫一侧）。

通过使用核方法，SVM便可用来分类n维空间的数据。这就引出了在3维空间中标绘数据点，从而让SVM可以区分史努比、加菲猫与西蒙，甚至在更高的维度对更多卡通人物进行分类。SVM并不总被视为神经网络。

Cortes, Corinna, and Vladimir Vapnik. “Support-vector networks.” Machine learning 20.3 (1995): 273-297.  
[http://image.diku.dk/imagecanon/material/cortes\_vapnik95.pdf](https://link.zhihu.com/?target=http%3A//image.diku.dk/imagecanon/material/cortes_vapnik95.pdf)

## 28. Kohonen 网络

最后，我们来介绍一下Kohonen网络（KN，也称之为自组织（特征）映射（SOM/SOFM：self organising (feature) map））。

KN利用竞争学习来对数据进行分类，不需要监督。先给神经网络一个输入，而后它会评估哪个神经元最匹配该输入。然后这个神经元会继续调整以更好地匹配输入数据，同时带动相邻的神经元。相邻神经元移动的距离，取决于它们与最佳匹配单元之间的距离。KN有时也不被认为是神经网络。

参考文献：  
Kohonen, Teuvo. “Self-organized formation of topologically correct feature maps.” Biological cybernetics 43.1 (1982): 59-69.  
[http://cioslab.vcu.edu/alg/Visualize/kohonen-82.pdf](https://link.zhihu.com/?target=http%3A//cioslab.vcu.edu/alg/Visualize/kohonen-82.pdf)  
原文链接：

* 中文链接：[http://www.toutiao.com/i6432188985530909186/](https://link.zhihu.com/?target=http%3A//www.toutiao.com/i6432188985530909186/)
* 英文链接1：[http://www.asimovinstitute.org/neural-network-zoo/](https://link.zhihu.com/?target=http%3A//www.asimovinstitute.org/neural-network-zoo/" \t "_blank)
* 英文链接2：[http://www.asimovinstitute.org/neural-network-zoo-prequel-cells-layers/](https://link.zhihu.com/?target=http%3A//www.asimovinstitute.org/neural-network-zoo-prequel-cells-layers/" \t "_blank)

发布于 2020-07-01 12:28

## FullLstmNode

我们这样作：一轮训练中，并行的准备3个LSTM单元,分别命名为node1,node2,node3。用同一个初始C（t-1）和H(t-1)和X(t),得到2种共6个输出：Hn1(t),Hn2(t),Hn3(t)和Cn1(t),Cn2(t),Cn3(t)。然后每一种输出的全部3个元素执行点乘，得到Hn(t)与Cn(t)。将Hn(t)与Cn(t)和X(t)，作为输入各自输入给另外3个并行的LSTM单元[node1b, node2b,node3b]，从而得到新的2种共6个输出：Hn1b(t),Hn2b(t),Hn3b(t)和Cn1b(t),Cn2b(t),Cn3b(t)，和上一次一样执行点乘得到H(t)和C(t)。前面的运算整体命名为一个FullLstmNode，然后如同普通LSTM运算一样执行多层计算。

GPT:

你的想法很有趣，是一种创新性的 **并行计算和信息融合机制** 的扩展。你通过引入多个并行 LSTM 单元，尝试在每一层的计算中融合更多的信息流，以下是我对你描述的细化和解读。

**1. 设计概述**

**单个 FullLstmNode 的过程：**

1. **输入准备：**
   * 初始输入：当前时间步的输入 X(t)X(t)，以及上一时间步的隐藏状态 H(t−1)H(t-1) 和单元状态 C(t−1)C(t-1)。
2. **第 1 组并行计算：**
   * 使用 3 个 LSTM 单元 node1,node2,node3\text{node1}, \text{node2}, \text{node3}，每个单元独立处理相同的输入 [X(t),H(t−1),C(t−1)][X(t), H(t-1), C(t-1)]，得到 6 个输出：
     + 隐藏状态：Hn1(t),Hn2(t),Hn3(t)Hn1(t), Hn2(t), Hn3(t)
     + 单元状态：Cn1(t),Cn2(t),Cn3(t)Cn1(t), Cn2(t), Cn3(t)
3. **信息融合（点乘）：**
   * 对第 1 组 LSTM 的输出进行逐元素点乘： Hn(t)=Hn1(t)⊙Hn2(t)⊙Hn3(t)Hn(t) = Hn1(t) \odot Hn2(t) \odot Hn3(t) Cn(t)=Cn1(t)⊙Cn2(t)⊙Cn3(t)Cn(t) = Cn1(t) \odot Cn2(t) \odot Cn3(t)
4. **第 2 组并行计算：**
   * 将融合后的 Hn(t)Hn(t)、Cn(t)Cn(t) 和输入 X(t)X(t) 作为新的输入，送入第 2 组并行 LSTM 单元 node1b,node2b,node3b\text{node1b}, \text{node2b}, \text{node3b}，得到新的 6 个输出：
     + 隐藏状态：Hn1b(t),Hn2b(t),Hn3b(t)Hn1b(t), Hn2b(t), Hn3b(t)
     + 单元状态：Cn1b(t),Cn2b(t),Cn3b(t)Cn1b(t), Cn2b(t), Cn3b(t)
5. **再次融合（点乘）：**
   * 对第 2 组输出再次点乘，得到最终的隐藏状态和单元状态： H(t)=Hn1b(t)⊙Hn2b(t)⊙Hn3b(t)H(t) = Hn1b(t) \odot Hn2b(t) \odot Hn3b(t) C(t)=Cn1b(t)⊙Cn2b(t)⊙Cn3b(t)C(t) = Cn1b(t) \odot Cn2b(t) \odot Cn3b(t)
6. **输出结果：**
   * H(t)H(t) 和 C(t)C(t) 将作为该 FullLstmNode 的输出，供下一层或者下一时间步使用。

**2. 多层架构扩展**

* **层次结构：** 你可以将多个这样的 FullLstmNode 堆叠在一起，类似于标准 LSTM 的多层结构，每一层的输出 [H(t),C(t)][H(t), C(t)] 会成为下一层的输入。
* **时间序列：** 在时间序列的处理上，与普通 LSTM 相同，H(t)H(t) 和 C(t)C(t) 也会传递到下一时间步 t+1t+1。

**3. 优势分析**

**并行计算提升模型容量**

* 通过多个 LSTM 单元并行处理相同的输入，引入了更多参数和计算通路，使得模型可以学习更复杂的特征和非线性关系。

**点乘实现信息融合**

* 点乘操作引入了一种简单但有效的机制，将多个并行单元的输出融合在一起，类似于一种“软投票”机制，从而增强信息表达能力。

**适应多模态输入**

* 如果你的数据来自不同模态（例如文本、图像等），这种架构可能更适合，因为不同的 LSTM 单元可以学习不同模态的信息。

**4. 潜在问题**

1. **计算成本：**
   * 并行 LSTM 单元增加了计算复杂度，可能会对训练时间和资源造成较大开销。
2. **梯度消失问题：**
   * 点乘操作可能使梯度在反向传播时变得更小，需要仔细设计初始化和正则化策略。
3. **需要额外调优：**
   * 在这种架构中，多个 LSTM 单元可能会引入更多的超参数（例如不同单元的维度、门控函数等），需要额外的调优。

**5. 进一步优化建议**

* **加权融合：** 考虑对 Hn1(t),Hn2(t),Hn3(t)Hn1(t), Hn2(t), Hn3(t) 的融合引入可学习的权重：

Hn(t)=α1Hn1(t)+α2Hn2(t)+α3Hn3(t),Hn(t) = \alpha\_1 Hn1(t) + \alpha\_2 Hn2(t) + \alpha\_3 Hn3(t),

其中 α1,α2,α3\alpha\_1, \alpha\_2, \alpha\_3 是通过网络学习的。

* **注意力机制：** 引入注意力机制，在融合时对不同 LSTM 单元的输出进行动态加权。
* **正则化：** 使用 Dropout 或其他正则化方法，防止过拟合。

**6. 总结**

你的想法非常具有创造性，结合并行计算和点乘融合，将单个 LSTM 单元扩展为一个更强大的信息处理模块（FullLstmNode）。通过适当优化和调整，这种架构有潜力在复杂时间序列任务中表现出色。下一步可以尝试设计代码并测试性能，看看是否能带来实际的性能提升！

### 一个问题：曾经有人提出过这样的或者类似的架构吗？有的话请介绍一下。

GPT:

类似的架构思路曾经在某些领域或场景中被提出或部分实现，虽然未必与您的设计完全一致，但有一定的相关性。以下是与您的架构设计理念接近的几类研究和实现：

**1. 深层递归神经网络（Deep RNNs）**

**与您的架构的相似点**：

* 深层递归神经网络在每个时间步会堆叠多个层，类似于您的 FullLstmNode 中多单元并行计算的层次结构。
* 不同层的输出可以融合，来提高模型对复杂序列的建模能力。

**不同点**：

* 标准深层 RNN 的层次结构是串联的，而您的设计通过并行单元和点乘融合的机制来实现信息整合。

**相关论文**：

* Pascanu et al., 2014. *How to Construct Deep Recurrent Neural Networks*. 提出了如何通过深度结构扩展 RNN 的表达能力。

**2. 门控循环单元的变种**

**Mixture of Experts (MoE) 或 Mixture of RNNs**

**概念**：

* Mixture of Experts 是一种通过多个专家网络并行处理同一输入，然后通过一个门控网络（Gate Network）对这些专家输出进行加权融合的方法。
* 在一些实现中，这些“专家”可以是并行的 LSTM 单元，类似于您的 node1、node2、node3。

**与您的架构的相似点**：

* 多个 LSTM 单元（专家）对输入进行并行处理。
* 专家输出被进一步融合，形成统一的表示。

**不同点**：

* MoE 通常使用加权和来融合专家输出，而您的设计通过逐元素点乘融合信息。

**相关论文**：

* Shazeer et al., 2017. *Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer*. Google 提出的 MoE 思想。

**3. 神经网络中的 Capsule Networks**

**概念：**

* Capsule 网络（CapsNet）是一种改进 CNN 的结构，用于捕获复杂模式之间的关系。
* 在 Capsule 网络中，多个子网络会分别生成“胶囊”，然后通过动态路由融合这些子网络的输出。

**与您的架构的相似点**：

* 多个子网络的输出会融合生成最终的特征表示。
* 使用并行结构捕获更复杂的信息关系。

**不同点**：

* Capsule 网络的动态路由方法与您的点乘方式不同。

**相关论文**：

* Sabour et al., 2017. *Dynamic Routing Between Capsules*. 提出了 Capsule 网络。

**4. Ensemble RNNs**

**概念**：

* 在一些序列建模任务中，会使用多个并行的 RNN/LSTM 单元作为子模型，然后对这些子模型的输出进行融合。
* 融合方式包括简单的平均、加权平均或者基于门控的动态融合。

**与您的架构的相似点**：

* 多个 RNN/LSTM 单元并行处理输入。
* 子模型的输出被融合为最终的状态。

**不同点**：

* Ensemble RNN 通常是针对整个模型训练多个子模型，而您的设计是每一层都使用多个并行 LSTM 单元。

**相关论文**：

* Yang et al., 2018. *Ensemble Recurrent Neural Networks for Multimodal Sequence Classification*. 提出了如何融合多个 RNN 模型来处理多模态数据。

**5. RNN with Attention Mechanisms**

**概念**：

* 在注意力机制中，隐藏状态会被不同的权重重新加权，来决定哪部分信息更重要。
* 某些实现中，多个并行的 LSTM 单元可以通过注意力机制动态融合输出。

**与您的架构的相似点**：

* 并行单元的输出被整合，生成最终的隐藏状态。

**不同点**：

* 注意力机制是通过学习的加权平均，而您的点乘操作可能更倾向于信息的选择性保留。

**相关论文**：

* Bahdanau et al., 2015. *Neural Machine Translation by Jointly Learning to Align and Translate*. 提出了序列到序列模型中的注意力机制。

**6. 多模态 LSTM**

在处理多模态数据时（如同时输入文本、图像、音频等），会使用多个并行的 LSTM 单元分别处理不同模态的信息，然后融合这些单元的输出。

**与您的架构的相似点**：

* 并行 LSTM 单元分别处理不同的信息来源。
* 单元的输出被融合成一个统一表示。

**相关研究**：

* Zadeh et al., 2018. *Memory Fusion Network for Multi-view Sequential Learning*. 提出了多模态数据的融合方法。

**结论**

虽然没有发现完全与您提出的架构设计完全一致的研究，但上述方法在某些方面与您的思路具有一定相似性。您的架构设计通过 **并行 LSTM 单元、点乘融合、多层计算**，是一种独特的扩展方式，可以被视为对现有门控循环网络的创新。

您可以尝试实现并进行实验，看看这种架构在特定任务上的效果，同时也可以通过对比上述方法来进一步优化！

## 数据处理

### 在RNN中，对O(t)以Tanh()处理后得到H(t)。在这里为什么要用Tanh()?它起到什么作用？

在 RNN（或其变种）中，对 O(t) 应用 tanh 函数后得到 H(t)，主要是出于以下几个原因：

1. **. 限制输出范围，稳定模型训练**

* **限制范围**：tanh⁡(x)\tanh(x) 的输出值被限制在 (-1, 1) 区间内。这种范围限制对模型的稳定性非常重要，可以防止输出值过大或过小导致数值不稳定。

A mathematical equation with a couple of emoticons

AI-generated content may be incorrect.

* **减少梯度爆炸风险**：在反向传播中，如果没有这样的激活函数，输出可能会随层数增多而指数级增长，导致梯度爆炸。tanh 的范围限制有助于缓解这种问题。

1. **提供非线性变换**

* 如果不使用 tanh，模型输出可能只是输入的线性组合，从而降低了模型的表达能力。tanh 引入了非线性变换，使 RNN 能够学习复杂的时间序列关系。

1. **增强隐藏状态的动态表示**

* H(t) 是当前时间步的隐藏状态，它会被用作下一时间步的输入。因此，隐藏状态需要适当的压缩和非线性处理，以更紧凑地表示历史信息。
* tanh 的“平滑”性质有助于隐藏状态的更新更具连续性和动态性，从而更有效地捕捉时间序列中的模式。

1. **对称性有助于梯度流动**

* tanh(x) 是关于原点对称的激活函数：

tanh(-x) = -tanh(x)

这种对称性使得正负方向的梯度能够均匀传播，避免梯度偏向某一方向，提升模型的学习效率。

1. **对比 ReLU 或 Sigmoid**

* **与 ReLU**： tanh 在处理隐藏状态时更适合，因为隐藏状态可能包含正负值，而 ReLU 仅保留正值，可能导致信息丢失。
* **与 Sigmoid**： tanh 的输出范围更广（ (-1, 1)），比 Sigmoid（ (0, 1)）更适合在隐藏状态中表示正负变化。

1. **总结：为什么要用 tanh？**

* 限制范围，防止数值不稳定。
* 引入非线性，提高模型表达能力。
* 适应隐藏状态的动态更新需求。
* 提供对称性，促进梯度高效传播。

在实际中，虽然 tanh⁡\tanh 一直是经典 RNN 的默认选择，但在更复杂的网络（如 LSTM 和 GRU）中，部分场景可能会结合 ReLU 或其他激活函数以优化特定任务的表现。

### 如果有一些运算注定会得到很小的H值，可能在运算中被忽略。那么，如何解决这样的问题，把它放大到一定的程度？

如果某些运算导致 H(t)H(t) 值很小，从而可能被忽略或对后续计算影响较小，可以通过以下几种方式来放大这些值，从而解决这一问题：

1. **使用标准化和偏移（Normalization and Shift）**

通过对隐藏状态进行归一化后加偏移，可以将较小的值放大到一个有效的范围。

**方法：**

* **批量归一化（Batch Normalization）**： 对 H(t) 进行归一化，使其分布具有均值 0 和标准差 1：



然后通过可学习参数放缩和偏移：



其中 γ 和 β 是放大和偏移的可学习参数。

* **层归一化（Layer Normalization）**： 对每个时间步的隐藏状态进行归一化，类似于批量归一化，但计算的是当前层的归一化。

**优点：**

* 提高小值在梯度中的相对重要性。
* 防止数值范围的极端差异。

1. **修改激活函数的特性**

选择能够更敏感地处理小值的激活函数。

**方法：**

* **Leaky ReLU**： 替代 tanh 或 ReLU，使用 Leaky ReLU，它可以通过一个小的斜率（如 0.01）为负值区域提供梯度：

A group of black text

AI-generated content may be incorrect.

其中 α>0（通常取 0.01）。

* **Scaled Tanh**： 增大 tanh⁡\tanh 的范围，比如用一个可控参数 α\alpha 进行放缩：



例如 α=2 可以使输出范围变为 (-2, 2)。

**优点：**

* 保留小值的重要性，避免其被忽略。
* 提供更大的梯度，促进小值区域的学习。

1. **加权放大（Attention or Scaling）**

通过引入可学习的放缩因子，直接对 H(t)H(t) 进行加权。

**方法：**

* **静态加权**： 为 H(t) 乘以一个固定的放大系数：



* **动态加权（Attention Mechanism）**： 根据上下文动态生成一个加权因子 α(t)，然后对 H(t) 进行加权：



这种方式通过注意力机制自动放大关键的隐藏状态。

**优点：**

* 适配性强，放缩程度可控。
* 提高模型对小值的敏感性。

1. **加入残差连接（Residual Connection）**

为隐藏状态引入残差连接，可以防止小值完全被忽略。

**方法：**

* 修改隐藏状态更新公式：

H'(t) = H(t) + X(t)

或者直接将小值加权后加入：

H'(t) = H(t) + α⋅H(t), α> 0

**优点：**

* 增强信号保留，避免小值被消除。
* 稳定梯度流动，特别是深层网络中。

1. **自定义损失函数**

修改损失函数，使其对小值更加敏感，间接推动网络更关注这些值。

**方法：**

* 为小值区域加入加权惩罚：

A black text on a white background

AI-generated content may be incorrect.

其中 small(H) 是所有小于某阈值的 H 元素。

**优点：**

* 通过直接约束优化过程，强制模型学习小值区域。

1. **总结**

根据具体情况，以下策略适合解决 H(t) 值过小的问题：

1. 对于数值范围问题：使用 **归一化和偏移**。
2. 对于激活函数不足：尝试 **Leaky ReLU 或 Scaled Tanh**。
3. 对于动态变化需求：引入 **加权放大或注意力机制**。
4. 对于小值保留问题：加入 **残差连接**。
5. 对于学习不足：修改 **损失函数**。

这些方法可以单独使用，也可以组合使用，具体选择取决于模型需求和数据特性。

### 如果这个值是N个正常范围内的因子的点积，那么是否也可以将它开N次方得到合理的结果？

是的，**如果这个值是 NN 个正常范围内的因子的点积**，将其开 NN 次方可以得到一个合理的结果，但前提是我们需要明确开 NN 次方在数学上和模型中的意义。

1. **理论依据：开 NN 次方的作用**

* 如果 x 是 N 个因子的点积：



那么开 N 次方的计算公式为：



这相当于将结果均衡化，返回到与原因子的尺度更接近的范围。其数学意义是找到所有因子的**几何平均值**。

* 如果每个因子 ai 都在合理的范围（例如 [0, 1] 或其他紧凑区间），那么点积 x 可能会因多个小因子导致结果过小。开 N 次方可以有效放大这些小值，防止信息丢失。

1. **几何意义：几何平均值的合理性**

* **几何平均值的特点**：
  1. 几何平均值的值介于所有因子中的最小值和最大值之间。
  2. 减少了点积直接计算时小值对整体结果的过大影响。
* **适用场景**：
  1. 如果因子是独立贡献且相乘可能导致小值时，几何平均值提供了一种折中方案，保持了结果的合理范围。

1. **数值稳定性和范围调整**

* **数值稳定性**：
  + 点积容易因因子较小导致值趋近于 0，开 N 次方可以将其放大到原始因子范围的均衡值。
  + 对于因子分布较广的情况（例如因子在 [0.01,10] 范围），需要确保计算 x^{1/N} 时不会引入额外的不稳定性。
* **范围调整**： 如果点积结果 x 本身非常接近 0（例如 10^{-6}），直接开 N 次方后可能仍然较小。此时，可以进一步放大结果：



通过增加放缩因子 α\alpha，提升数值范围。

1. **注意事项**

* **输入因子的合理性**：
  + 所有输入因子 ai 应尽可能处于合理的范围（例如非负且非零），否则开 N 次方可能会导致数值异常（如开负数或除零错误）。
* **对于小值的适用性**：
  + 如果某些因子本身非常小甚至趋于零，几何平均值可能仍不足以提升其影响力，可能需要额外放缩处理。
* **适用场景**：
  + 当点积值过小而实际意义较大时。
  + 当模型需要对小值进行合理放大但不想完全舍弃小值信息时。

1. **示例**

假设有 4 个因子 a1,a2,a3,a4，它们的值为：

a1=0.8, a2=0.6, a3=0.9, a4=0.7

它们的点积为：

x=0.8⋅0.6⋅0.9⋅0.7=0.3024

直接使用点积结果可能显得过小。通过开 N = 4 次方：



结果回到了更接近因子的范围 [0.6, 0.9]。

1. **结论**

将点积结果开 N 次方是一种合理的放大方法，特别适用于因子在正常范围内且点积结果过小的情况。这种方法能够：

1. 提升数值稳定性。
2. 确保输出结果更接近原因子的分布范围。
3. 保留小值的重要信息，减少忽略风险。

此外，可以结合放缩因子或其他归一化方法进一步调整结果的数值范围，满足具体任务的需求。

### 对于大量计算来说，开方的代价太大。那么有没有近似效果的快速算法？

对于大量计算来说，开方运算的代价确实较高，尤其是高次根的计算。以下是几种可以替代开 N 次方的**近似算法**或**优化策略**，它们能够在保证结果合理性的同时显著提高计算效率。

1. **使用对数和指数变换**

开 N 次方的计算：



可以通过对数和指数转换来快速近似：

**公式：**



**优化点：**

* **对数和指数函数的效率高于直接开方**：现代硬件（如 GPU）对对数和指数函数的优化比开方函数更好。
* **对批量计算更友好**：对数和指数运算可以矢量化，减少单次运算的开销。

**缺点：**

如果 x 的值非常小或接近于 0，可能会导致ln(x) 的计算不稳定（负无穷）。对此，可以先对 x 添加一个小偏移量 ϵ：



1. **使用泰勒展开或快速近似公式**

**方法：**

通过对 x^{1/N} 使用泰勒展开或快速近似公式，可以避免直接调用开方函数。

**公式：**

对于 x^{1/N}，可以用以下线性近似（适用于 x≈1 的情况）：



**优点：**

* 计算非常快，仅需要加法和乘法。
* 对 xx 接近 1 的场景非常准确。

**扩展：**

为了增加准确性，可以使用二次或三次多项式近似：

A number and a number

AI-generated content may be incorrect.

1. **直接利用指数衰减代替开方**

在许多深度学习或物理模型中，对开方操作的需求往往来自需要降低结果范围的情况。此时可以使用指数衰减来替代开方：

**替代公式：**

将 x^{1/N} 替换为：



其中 α∈[0,1) 是一个调节因子，用于控制范围压缩程度。

**优点：**

* 完全避免复杂运算，仅需要加法和乘法。
* α 可根据具体任务进行调整，适配灵活。

1. **预计算查表法**

如果 xx 的取值范围是有限的，可以预先计算出对应的 x1/Nx^{1/N} 值，并存储在查找表中。

**方法：**

1. 确定 x 的范围（例如 [xmin, xmax]）。
2. 均匀采样计算出对应的开方结果，并存储为查找表。
3. 查询时，根据输入的 xx 快速插值获取近似值。

**优点：**

* 对于大规模重复计算非常高效。
* 插值方法（如线性插值）也可以进一步提高精度。

**缺点：**

* 存储需求较高。
* 适用于输入范围较窄的场景。

1. **动态调整点积权重**

如果原始点积结果过小且需要放大，直接调整点积的权重可以避免额外的开方操作。

**方法：**

将点积公式修改为加权和形式：

A number and a number of symbols

AI-generated content may be incorrect.

即用因子的**算术平均值**替代点积的结果。这是一种更快的近似方法，因为加法和除法比开方计算更高效。

**优点：**

* 避免非线性开方操作。
* 能在一定程度上平衡小值的影响。

1. **基于近似的硬件优化**

如果你在硬件上实现大规模计算，可以利用以下优化技术：

* **FP16（半精度浮点数）优化**：对于精度要求不高的场景，可以使用半精度浮点数表示 xx，从而显著降低开方的计算复杂度。
* **硬件支持的近似函数**：一些深度学习框架（如 TensorRT）提供了硬件优化的近似函数，可以用更低的计算代价实现非线性操作。

1. **对比总结**

|  |  |  |  |
| --- | --- | --- | --- |
| **方法** | **计算代价** | **精度** | **适用场景** |
| 对数和指数变换 | 中等 | 高 | 大规模计算，支持硬件优化的场景 |
| 泰勒展开 | 低 | 中 | x≈1 或范围较窄 |
| 指数衰减替代 | 低 | 中 | 对值范围无严格限制的任务 |
| 查表法 | 非常低 | 高 | 输入值离散且范围固定的场景 |
| 算术平均替代点积 | 非常低 | 中 | 快速替代，输入值范围不敏感场景 |
| 硬件优化 | 低 | 高 | 硬件加速支持的大规模计算 |

1. **推荐策略**

* 如果可以接受一定的误差，使用 **泰勒展开** 或 **指数衰减** 是最快的方法。
* 对于精度要求高的场景，使用 **对数和指数变换** 或 **查表法**。
* 对超大规模计算任务，建议结合 **硬件优化** 或直接采用更简单的替代方法（如算术平均）。