

# CPU9444\_Overview\_yann

**Autor:** Yann

**Reference:**

**UNSW COMP9444 Lecture slides - Prof. Alan Blair**

**USYD COMP5329/5318 Lecture slides - Prof. Chang Xu**

Artificial Neural Networks

1. 权重 (Weights):

2. 偏置 (Bias):

3. 激活函数 (Activation Function):

多层感知机的工作原理 :

更新公式

Linear Separability

1. 二维空间中的点

2. 更高维度的空间

Network Structure

Objective Function

为什么需要目标函数 ?

常见的目标函数 :

Optimisation

1. 局部极小值 (Local Minima):

2. 全局极小值 (Global Minima):

为什么这些概念在神经网络优化中很重要 ?

Activation Function

1. 为什么需要激活函数 ?

2. 常用的激活函数:

3. 选择激活函数的注意事项:

拓展 : 从非线性角度理解激活函数

Gradient Descent

Chain Rule

Feedforward & Back-propagation

1. 前馈 (Feedforward):

2. 反向传播 (Back-propagation):

Bayes' Formula

例子 : 医学检测

Types of Learning

1. 监督学习 (Supervised Learning)

2. 无监督学习 (Unsupervised Learning)

3. 强化学习 (Reinforcement Learning)

Overfitting VS. Underfitting

从数学角度思考 ? Bias & Variance Trade-off

那我们怎么知道这个时候的bias和variance呢 ?

## Generalisation

Error Plots

Early Stopping

工作原理

Dropout

1. 工作原理:

2. 为什么有效:

Ensembling

多样性 (Diversity)

实例：多模型集成

Dropout 与集成

优点与缺点

## Softmax

1. 定义:

2. 性质:

3. 应用:

4. 直观理解:

## Cross-entropy

1. 定义:

2. 直观理解:

3. 应用于机器学习:

4. 性质:

5. 与 KL 散度的关系:

## 深度神经网络启蒙发展时期

LeNet — 1998

主要技术和特点 :

网络结构 :

AlexNet — 2012

AlexNet 网络结构

主要技术和创新点 :

VGG — 2014

主要技术和创新点 :

GoogLeNet — 2014

主要技术和创新点 :

## Images

2D Image

像素 (Pixel)

精度与范围

RGB Image

为什么是“3D”?

彩色信息

在 CNN 中的应用

## 卷积神经网络

Convolutional Layer

卷积操作

卷积与傅立叶变换

Stride & Padding

步长 (Stride)

填充 (Padding)

Receptive Filed

基础概念：

捕获信息：

Pooling

常见类型：

影响和作用：

Multi-channel CNN

总结

Exercise

Entropy

高斯分布的熵

KL-Divergence

相关性

Wasserstein Distance

为什么在这种情况下WD更好？

Weight Initialisation

随机初始化

零初始化

Batch Normalisation

CNN模块结构

卷积运算

池化层

Stride & Pooling

Deeper Network — ResNet

残差块

优点

Smaller Kernel

架构特点

优点和局限性

Inception Module

Inception 模块

架构特点

优点和局限性

SENet

Squeeze-and-Excitation 模块

Data Augmentation

Regularisation

Weight Decay

数学描述

工作原理

优点和局限性

Dropout

工作原理

优点和局限性

Language Statistics Model

Counting Frequencies

Applications

Next-word Prediction

Model

Analysis

Example

Summary

N-Gram Model

数学模型

Analysis

Example

Summary

Co-occurrence Matrix

Model

Analysis

Example

Summary

Extension

Language Semantic Model

Word Embeddings

Major Algorithms

Analysis

Examples

Summary

Word2Vec

Skip-gram 和 CBOW

Analysis

Example

Summary

Extension Details

Cost Function

CBOW

Model

数学模型

Analysis

Example

Summary

Skip-Gram Model

Model

Model

Analysis

Summary

Training Method

Hierarchical Softmax

Negative Sampling

NLP Model Summary

## Applications

文本分类

机器翻译

Q & A

Text Summary

## Preprocessing

One-hot Encoding

Count Vectorizer

TF-idf Vectorizer

## Sequence Model

数据结构

数据表示

网络架构

时间复杂性

空间复杂性

## MLP ?

MLP (多层感知器)

RNN (循环神经网络)

比较

## Recurrent Neural Network

1. 循环连接 (Sequential Connectivity)

2. 记忆模块 (Memory)

RNN 与 CNN 的不同 :

## Back Propagation

"展开" 架构

通过时间反向传播 (BPTT)

反向传播的时间跨度

例子

## Model of Computation

Regular (正则)

Context-Free (上下文无关)

Context-Sensitive (上下文相关)

Recursively Enumerable (递归可枚举)

Finite State Automaton (有限状态自动机)

Push Down Automaton (下推自动机)

Linear Bounded Automaton (线性有界自动机)

Turing Machine (图灵机)

## RNN & Finite State Machine

## Formal Language Prediction

## RNN Structure

1. 输入向量  $x_t$

2. 隐藏层激活  $h_t$

3. 输出向量  $y_t$

## Gradient Vanishing in RNN

长短时记忆网络 (Long Short-Term Memory, LSTM)

基本结构

LSTM与基础RNN的优势

梯度问题与基础RNN

LSTM的优势

GRU

结构复杂性

参数数量

记忆能力

信息流

计算复杂性

应用场景

Attention Mechanism

基本思想

数学表示

优点

Supervised & Reinforcement Learning

Supervised Learning Recap

Reward as supervisor

RL Framework

Example

Policy

Example

Value Function Learning

Action Selection

TD-Learning

Q-Learning

Limitation

Deep Q-Network

AutoEncoder

Encoder

Decoder

编码器 (Encoder)

反卷积 (Deconvolution)

应用

AutoEncoder

Pre-train by AE

1. 自编码器训练

2. 移除解码器部分

3. 添加新的层

4. 进一步训练

Variational Autoencoder

关键特点

工作原理

Structure

KL-Divergence Recap

Loss function

第一部分：重构损失 (Reconstruction Loss)

## 第二部分：KL散度 (Kullback-Leibler Divergence)

总体目标

### Generative Adversarial Networks

组件

竞争游戏

应用

Structure

Loss

GAN损失函数

组件

损失函数的两部分

博弈过程

结果

1. 生成器和判别器的能力

2. 博弈达到纳什均衡

3. 判别器的不确定性

### Converge Failure

# Artificial Neural Networks

**多层感知机 (MLP, Multi-Layer Perceptron)** 是一种前馈人工神经网络，它由三种类型的层组成：输入层、一个或多个隐藏层和一个输出层。每一层都由一个或多个节点（也称为“神经元”或“单元”）组成。

## 1. 权重 (Weights):

每个连接（从一个节点到另一个节点）都有一个权重，这个权重决定了信号的强度。权重可以被认为是连接的“强度”或“重要性”。在训练过程中，这些权重会被调整，以最小化模型的预测错误。

## 2. 偏置 (Bias):

除了权重之外，每个神经元还有一个偏置单元，它是神经元的一个额外输入，值始终为1，但它有自己的权重（也称为偏置权重）。偏置允许神经元进行更灵活的调整，因为它确保了即使所有输入都是0，神经元也能有非零输出。

## 3. 激活函数 (Activation Function):

激活函数是应用于神经元的加权输入和偏置之和的函数，它决定了神经元的输出。激活函数的目的是引入非线性性，使得MLP能够模拟非线性函数。常用的激活函数有：

- **Sigmoid:** 它返回介于0和1之间的值。
- **Tanh (Hyperbolic Tangent):** 它返回介于-1和1之间的值。
- **ReLU (Rectified Linear Unit):** 它返回大于0的值，否则返回0。
- **Leaky ReLU:** 它是ReLU的一个变种，允许小的负值当输入值是负的。

# Artificial Neural Networks

(Artificial) Neural Networks are made up of nodes which have

- inputs edges, each with some *weight*
- outputs edges (with *weights*)
- an *activation level* (a function of the inputs)

Weights can be positive or negative and may change over time (learning).

The *input function* is the weighted sum of the activation levels of inputs.

The activation level is a non-linear *transfer function*  $g$  of this input:

$$\text{activation}_i = g(s_i) = g\left(\sum_j w_{ij}x_j\right)$$

Some nodes are inputs (sensing), some are outputs (action)

5



## 多层感知机的工作原理：

1. **前向传播:** 输入数据从输入层开始，通过所有隐藏层，最后到达输出层。在每个神经元，数据被加权、加偏置并通过激活函数。
2. **反向传播:** 一旦得到了输出，就可以与真实标签进行比较，计算误差。然后，这个误差会被反向传播到网络中，以调整权重和偏置。
3. 这个过程会在多次迭代（或称为“纪元”）中重复，直到模型的预测误差达到一个可接受的水平。

## 更新公式

多层感知机 (MLP) 的权重和偏置的更新通常基于反向传播算法和梯度下降方法。下面是基本的更新公式，简化为一个神经元的情况。但在实际应用中，这些公式会在整个网络的所有神经元和层上进行迭代。

假设我们有以下符号：

- $X$ : 输入数据
- $W$ : 权重
- $B$ : 偏置
- $A$ : 神经元的加权输入和偏置之和，即  $A = W \cdot X + B$
- $Y$ : 神经元的输出，即  $Y = f(A)$ ，其中  $f$  是激活函数
- $\eta$ : 学习率
- $L$ : 损失函数
- $\frac{\partial L}{\partial Y}$ : 输出  $Y$  相对于损失  $L$  的偏导数

- $f'(A)$ : 激活函数的导数

权重和偏置的更新公式:

1. 计算输出误差的梯度:

$$\delta = \frac{\partial L}{\partial Y} \times f'(A)$$

1. 更新权重:

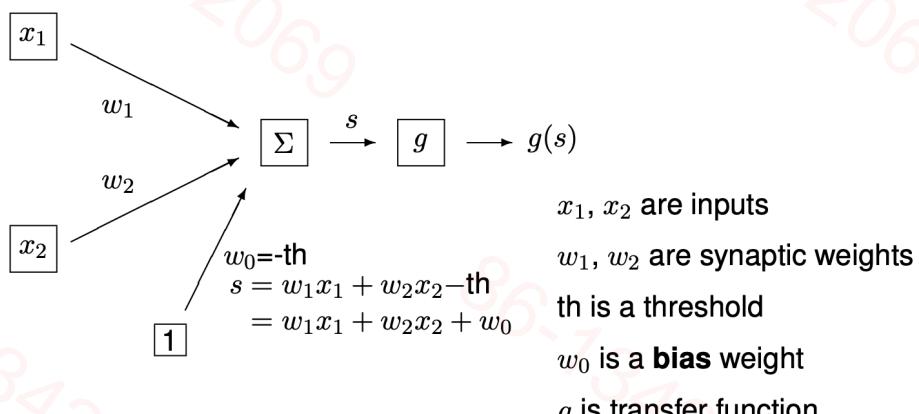
$$W = W - \eta \times \delta \times X$$

1. 更新偏置:

$$B = B - \eta \times \delta$$

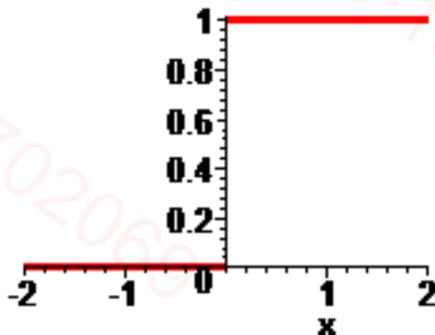
这些公式是基于链式法则进行梯度计算的结果。在实际的MLP中，这个过程从输出层开始，然后逐层反向进行，直到输入层。每一层的权重和偏置都会根据相应的梯度进行更新。

## McCulloch & Pitts Model of a Single Neuron



## Transfer function

Originally, a (discontinuous) step function was used for the transfer function:



$$g(s) = \begin{cases} 1, & \text{if } s \geq 0 \\ 0, & \text{if } s < 0 \end{cases}$$

(Later, other transfer functions were introduced, which are continuous and smooth)

7



## Linear Separability

线性可分性 (Linear Separability) 是机器学习和统计学中的一个概念，尤其是在分类问题中。一个数据集被认为是线性可分的，当且仅当存在一个线性超平面能够完美地将不同的类别分开，而不会出现任何错误。

让我用几个简单的例子来解释这个概念：

### 1. 二维空间中的点

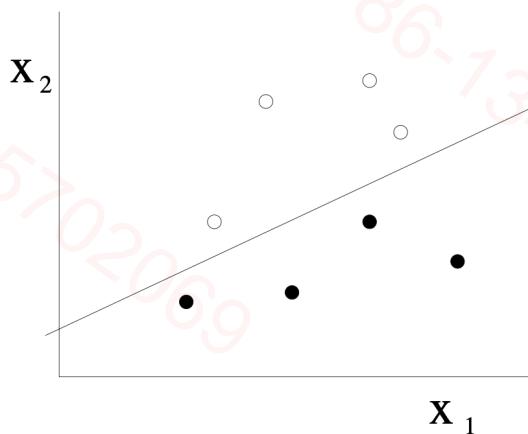
想象你在二维空间（一个普通的XY坐标系）中有两组点。一组点是红色的，另一组点是蓝色的。

- 如果你可以用一条直线（在这种情况下，这条直线就是一个超平面）来完美地分开这两组点，而不让任何红点在蓝点的一侧，反之亦然，那么这两组点就是线性可分的。
- 如果不论你怎么画这条直线，都有红点和蓝点混在同一侧，那么这两组点就是线性不可分的。

### 2. 更高维度的空间

在更高的维度中，这个直线的概念变成了一个超平面。例如，在三维空间中，这个超平面是一个平面；在四维空间中，这是一个三维的对象，以此类推。

Question: what kind of functions can a perceptron compute?



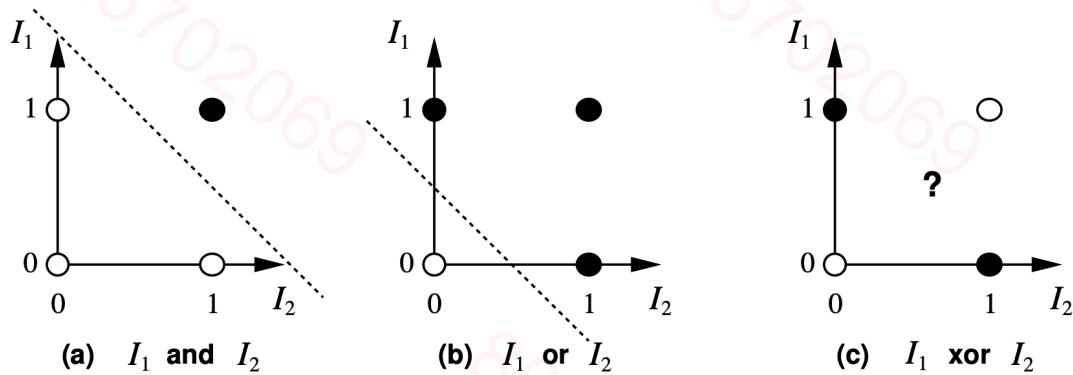
Answer: linearly separable functions

8



## Limitations of Perceptrons

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



Possible solution:

$x_1$  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.

18



## Network Structure

一个基本的前馈神经网络通常包括输入层、隐藏层和输出层。下面是这三层的详细说明：

### 1. 输入层 (Input Layer):

- 它是神经网络的第一层，负责接收外部数据。
- 输入层的节点数通常与数据的特征数相同。例如，对于一个包含10个特征的数据点，输入层通常有10个节点。
- 输入层的神经元通常不执行任何计算，它们只是传递数据到下一层。

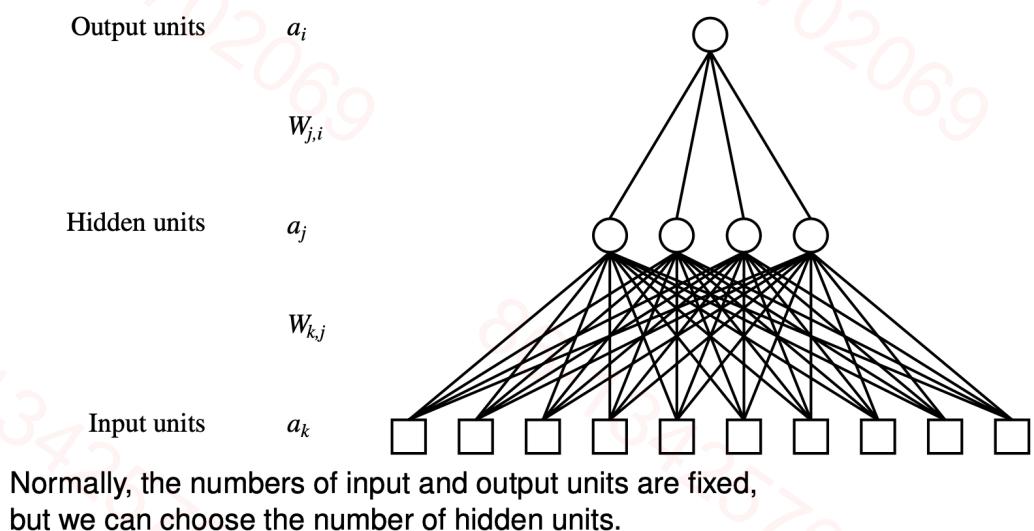
### 2. 隐藏层 (Hidden Layers):

- 这些层位于输入层和输出层之间。
- 一个神经网络可以有一个或多个隐藏层。当有多个隐藏层时，这种网络通常被称为“深度神经网络”。
- 隐藏层的神经元执行计算，并对从前一层接收到的数据进行某种变换。
- 每个隐藏层的神经元都有权重、偏置和激活函数。这些神经元的输出是基于其加权输入、偏置和激活函数的计算结果。
- 确定隐藏层的数量和每层的神经元数量是神经网络设计中的关键决策。

### 3. 输出层 (Output Layer):

- 这是神经网络的最后一层，负责产生网络的输出或预测。
- 输出层的节点数取决于任务的类型。例如，对于二分类问题，输出层可能只有一个神经元；而对于10类分类问题，输出层可能有10个神经元。
- 输出层的激活函数也取决于任务的类型。例如，对于二分类问题，通常使用sigmoid激活函数；而对于多类分类问题，通常使用softmax激活函数。

## Two-Layer Neural Network



5



## Objective Function

在神经网络中，**目标函数 (Objective Function)** 或称为 **损失函数 (Loss Function)** 或 **成本函数 (Cost Function)** 是一个核心组件。它为网络提供了一个衡量其预测与真实值之间差异的方式。简单地说，目标函数描述了网络的表现有多差。

### 为什么需要目标函数？

1. **优化**: 通过最小化目标函数，我们可以优化网络的权重和偏置，使网络的预测更加准确。
2. **反馈**: 它为网络提供了关于其当前表现的反馈，这有助于调整和更新参数。

3. 训练终止: 当目标函数的值达到一个可接受的范围或不再显著减少时, 可以停止训练。

### 常见的目标函数 :

1. **均方误差 (Mean Squared Error, MSE)**: 主要用于回归问题。计算预测值和真实值之间的平方差的平均值。

$$MSE = \frac{1}{N} \sum_{i=1}^N (y_i - \hat{y}_i)^2$$

其中,  $y_i$  是真实值,  $\hat{y}_i$  是预测值,  $N$  是数据点的数量。

1. **交叉熵损失 (Cross-Entropy Loss)**: 主要用于分类问题。衡量真实概率分布和预测概率分布之间的差异。

$$CE = - \sum_i y_i \log(\hat{y}_i)$$

其中,  $y_i$  是真实的概率分布 (通常是0或1),  $\hat{y}_i$  是预测的概率分布。

## NN Training as Cost Minimization

We define an **error** function or **loss** function  $E$  to be (half) the sum over all input patterns of the square of the difference between actual output and **target** output

$$E = \frac{1}{2} \sum_i (z_i - t_i)^2$$

If we think of  $E$  as height, it defines an error **landscape** on the weight space.  
The aim is to find a set of weights for which  $E$  is very low.

## Optimisation

在优化中, **局部极小值 (Local Minima)** 和 **全局极小值 (Global Minima)** 是两个重要的概念。这些术语用于描述函数的特定点, 其中函数值低于其邻近的点。

### 1. 局部极小值 (Local Minima):

- 局部极小值是函数上的一个点, 其值比其周围的邻近点都要小, 但可能比其他地方的点要大。

- 在这个点，函数的导数（或梯度，在多维情况下）为0。
- 函数可能有多个局部极小值，或者可能根本没有。

## 2. 全局极小值 (Global Minima):

- 全局极小值是函数上的一个点，其值是整个函数上的最小值。
- 每个函数只有一个全局极小值，但它可能出现在多个位置（例如，一个平坦的U形函数）。
- 全局极小值也是局部极小值，但反之则不成立。

### 为什么这些概念在神经网络优化中很重要？

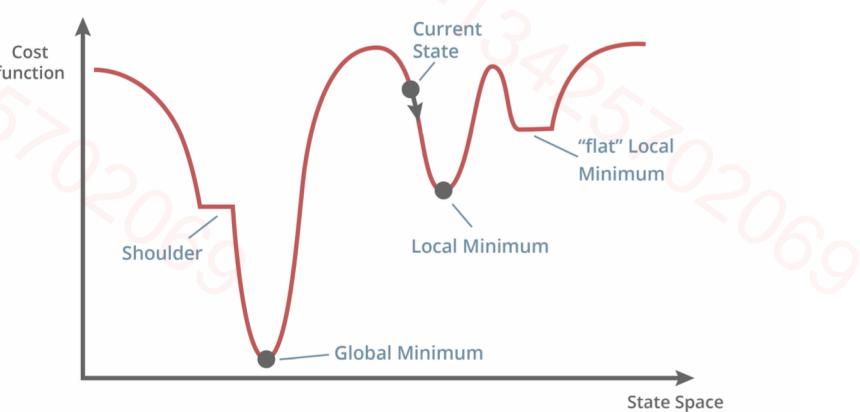
当我们使用梯度下降或其他优化方法训练神经网络时，我们的目标是找到损失函数的最小值。但这个过程可能会受到一些问题的影响：

1. **陷入局部极小值**: 由于神经网络的损失函数通常是非凸的，这意味着它可能有许多局部极小值。优化算法可能会陷入其中一个局部极小值，而不是找到全局极小值。
2. **高原和鞍点**: 除了局部极小值，损失函数可能还有高原（梯度接近于0的区域）和鞍点（在某些方向上是局部极小值，在其他方向上是局部极大值）。这些区域可能会减慢训练过程或使算法停滞。

为了避免这些问题，研究人员已经开发了多种优化技术，如动量、RMSProp、Adam等，它们试图更有效地导航损失函数的复杂景观，以找到一个好的最小值。

**形象例子**：想象一个山谷和多个小坑的地形。每个小坑都是一个局部极小值，而山谷的最低点是全局极小值。如果你是一个盲目的探险者，只能通过你脚下的坡度来感知地形，你的目标是找到山谷的最低点。但在你的旅途中，你可能会陷入一个小坑并认为你已经到达了最低点，尽管真正的山谷底部还在其他地方。这就是局部极小值和全局极小值之间的差异。

### Local Search in Weight Space



Problem: because of the step function, the landscape will not be smooth, but will instead consist almost entirely of flat local regions and "shoulders", with occasional discontinuous jumps.

# Activation Function

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。

下面是一些关于激活函数的关键点和常用的激活函数：

## 1. 为什么需要激活函数？

- **非线性:** 如果没有激活函数，无论神经网络有多少层，它总是会表现出线性的行为。激活函数为神经网络引入了非线性性，使其能够学习和执行更复杂的任务。

## 2. 常用的激活函数：

- **Sigmoid:**

- 输出范围是  $(0, 1)$ 。
- 早期的神经网络经常使用。
- 但现在较少使用，因为它在输入值很大或很小时会导致梯度消失的问题。

- **Tanh (Hyperbolic Tangent):**

- 输出范围是  $(-1, 1)$ 。
- 它是Sigmoid函数的变种，但比Sigmoid表现得更好，因为它的输出是零中心的。

- **ReLU (Rectified Linear Unit):**

- 在输入小于0时输出0，输入大于0时输出输入值。
- 是目前最受欢迎的激活函数之一，因为它在许多任务中都表现得很好，并且计算效率高。
- 但它有一个问题，那就是“死亡ReLU”问题，即某些神经元可能在整个训练过程中永远不会被激活。

- **Leaky ReLU:**

- 是ReLU的变种，它允许小的负梯度当输入值是负的。
- 这解决了“死亡ReLU”问题。

- **Softmax:**

- 通常用于神经网络的输出层，特别是在分类任务中。
- 它将输入转化为概率分布。

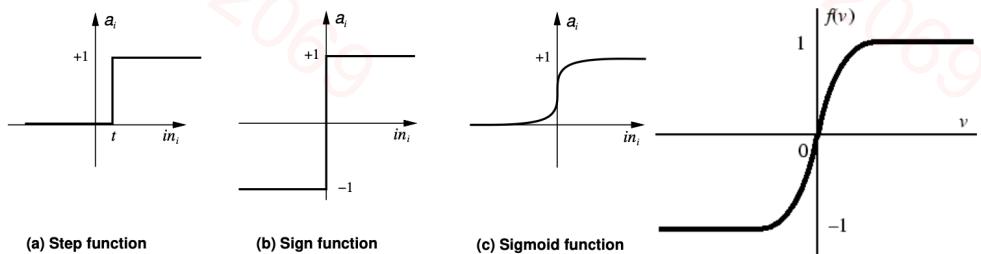
## 3. 选择激活函数的注意事项：

- 任务的性质：例如，对于二分类问题，Sigmoid可能是输出层的合适选择；对于多类分类，Softmax可能更合适。
- 梯度消失/爆炸问题：ReLU和它的变种如Leaky ReLU可以帮助缓解这个问题。

- 计算效率: ReLU和它的变种通常比Sigmoid和Tanh更快，因为它们的计算更简单。

**形象的比喻**: 你可以将激活函数看作是神经网络中的“开关”。基于其输入，激活函数决定输出应该是什么，或者说“开关”应该是开还是关。这些“开关”在整个网络中决定了信息的流动，从而使网络能够学习复杂的模式和功能。

## Continuous Activation Functions (3.10)



Key Idea: Replace the (discontinuous) step function with a differentiable function, such as the sigmoid:

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

or hyperbolic tangent

$$g(s) = \tanh(s) = \frac{e^s - e^{-s}}{e^s + e^{-s}} = 2\left(\frac{1}{1 + e^{-2s}}\right) - 1$$

## 拓展：从非线性角度理解激活函数

从非线性的角度来看，激活函数的核心作用是为神经网络引入非线性。这种非线性是神经网络能够学习复杂函数和模式的关键因素。

### 为什么需要非线性？

- 如果我们只使用线性变换（无激活函数或线性激活函数），无论我们有多少层，多层网络的输出始终可以简化为单层网络的输出。这是因为线性函数的组合始终是线性的。
- 引入非线性意味着我们可以构建更复杂、更有表现力的模型。这些模型可以捕捉数据中的高度复杂的关系。
- 根据泛化近似定理，一个具有足够数量的隐藏单元并使用非线性激活函数的神经网络可以逼近任何连续函数。这意味着，理论上，这样的神经网络可以学习表示任何复杂的函数或模式。

### 非线性的形象理解，一点非线性的叠加可以造成无限非线性：

想象你正在试图通过一堆乐高积木构建一个复杂的结构，比如一个城堡。如果你只使用一种形状的积木（比如正方形），你的构建可能会受到限制。但是，如果你有各种形状和大小的积木，你可以创建更复杂、更有趣的结构。

同样，线性变换就像是一种形状的乐高积木，而非线性激活函数为你提供了其他形状的积木。只有通过组合各种形状，你才能构建真正复杂的结构。

在神经网络中，每一层的线性变换（权重和偏置的计算）为你提供了基本的“积木”形状，而非线性激活函数则允许你以新的、更复杂的方式组合这些“积木”。通过叠加多个这样的层，你可以建立一个能够表示任何复杂函数的模型。

## Gradient Descent

提督下降，我认为你可能是指“梯度下降”(Gradient Descent)。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）。

**梯度下降的基本思想：**

1. 梯度是损失函数相对于参数的偏导数。它指示了函数值上升最快的方向。因此，函数下降最快的方向就是梯度的负方向。
2. 在每次迭代中，我们都会按照梯度的负方向更新参数，这样可以逐渐减小损失函数的值。

**梯度下降的步骤：**

1. 选择一个初始值或一组初始值（例如权重和偏置）。
2. 计算这个点的梯度。
3. 更新参数值，将其移向梯度的负方向。
4. 重复上述步骤，直到满足某个停止准则（例如梯度接近0、损失值不再显著减小或达到预定的迭代次数）。

**学习率：**

在梯度下降中，学习率是一个关键参数。它决定了在每次迭代中参数更新的步长。太大的学习率可能会导致算法在最小值附近震荡，甚至偏离最小值；而太小的学习率可能会导致收敛速度很慢。

**梯度下降的变种：**

1. **批量梯度下降 (Batch Gradient Descent)**: 在每次迭代中使用整个数据集来计算梯度。
2. **随机梯度下降 (Stochastic Gradient Descent, SGD)**: 在每次迭代中随机选择一个样本来计算梯度。
3. **小批量梯度下降 (Mini-batch Gradient Descent)**: 在每次迭代中使用一小批样本来计算梯度。

**形象理解：**

想象你在一个山上，目标是找到最快的方式走到山谷的最低点。你没有地图，所以你决定在每一步都选择使你下降最快的方向。这个方向就是你当前位置的“梯度”。通过多次迭代这个过程，你最终会到达山谷的底部，这就是损失函数的最小值。梯度下降的过程就像是你不断地选择下坡的方向，直到你找到一个位置，你无论往哪个方向走，都不能再下降了。

## Gradient Descent (4.3)

Recall that the **loss** function  $E$  is (half) the sum over all input patterns of the square of the difference between actual output and target output

$$E = \frac{1}{2} \sum_i (z_i - t_i)^2$$

The aim is to find a set of weights for which  $E$  is very low.

If the functions involved are smooth, we can use multi-variable calculus to adjust the weights in such a way as to take us in the steepest downhill direction.

$$w \leftarrow w - \eta \frac{\partial E}{\partial w}$$

Parameter  $\eta$  is called the *learning rate*.

11



## Chain Rule

梯度下降法优化神经网络时所使用的“链式法则”(Chain Rule)。链式法则是微积分中的一个基本概念，用于计算复合函数的导数。

在神经网络的背景下，链式法则尤其重要，因为我们经常需要计算损失函数相对于网络中某个权重的导数。由于损失是网络输出的函数，而网络输出又是权重和其他输入的函数，因此我们需要链式法则来计算这种“函数的函数”的导数。

### 链式法则的基本思想：

假设我们有两个函数 $y = g(u)$ 和 $u = f(x)$ ，那么复合函数 $y$ 关于 $x$ 的导数可以表示为：

$$\frac{dy}{dx} = \frac{dy}{du} \times \frac{du}{dx}$$

这就是链式法则的基本形式。

### 在神经网络中的应用：

考虑一个简单的神经网络，其中有一个隐藏层。我们想要计算损失函数 $L$ 相对于该隐藏层的权重 $W$ 的导数。损失是输出层输出的函数，而输出层的输出又是隐藏层输出的函数，隐藏层的输出又是权重和输入的函数。因此，我们需要链式法则来计算这种导数。

具体来说，我们会这样计算：

1. 损失相对于输出层输出的导数。
2. 输出层输出相对于隐藏层输出的导数。
3. 隐藏层输出相对于权重的导数。
4. 将这三个导数相乘，得到损失相对于权重的导数。

这个过程是神经网络反向传播算法的核心部分，它允许我们有效地计算损失函数相对于所有权重和偏置的导数，从而使用梯度下降法更新这些参数。

---

### Chain Rule (6.5.2)

If, say

$$y = y(u)$$

$$u = u(x)$$

Then

$$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial u} \frac{\partial u}{\partial x}$$

This principle can be used to compute the partial derivatives in an efficient and localized manner. Note that the transfer function must be differentiable (usually sigmoid, or tanh).

Note: if  $z(s) = \frac{1}{1 + e^{-s}}$ ,  $z'(s) = z(1 - z)$ .

if  $z(s) = \tanh(s)$ ,  $z'(s) = 1 - z^2$ .

12



## Feedforward & Back-propagation

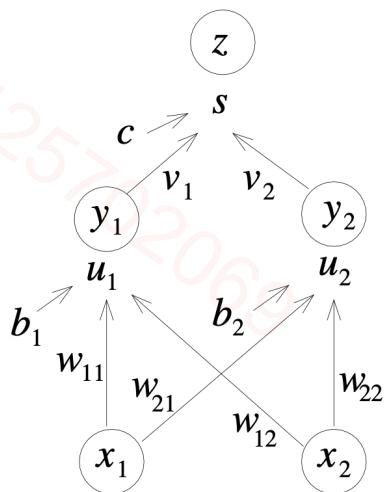
前馈算损失，后馈算梯度。

### 1. 前馈 (Feedforward):

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，输入数据在每一层都经过加权求和和激活函数处理。
- 这个过程从输入层开始，经过所有隐藏层，最终到达输出层，生成网络的输出或预测。
- 前馈的主要目标是生成网络的输出，这可以用于预测或与真实标签比较来计算损失。

## Forward Pass



$$\begin{aligned}u_1 &= b_1 + w_{11}x_1 + w_{12}x_2 \\y_1 &= g(u_1) \\s &= c + v_1y_1 + v_2y_2 \\z &= g(s) \\E &= \frac{1}{2} \sum (z - t)^2\end{aligned}$$

## 2. 反向传播 (Back-propagation):

反向传播是神经网络训练中的关键步骤，它是一种有效计算损失函数相对于网络参数（权重和偏置）的梯度的方法。

- 开始于计算损失函数的梯度。
- 这个梯度然后被传播回网络，从输出层到输入层。
- 在每一层，使用链式法则来计算损失函数相对于该层参数的梯度。
- 一旦计算出所有的梯度，就可以使用优化算法（如梯度下降）来更新网络的权重和偏置。

### 为什么需要反向传播？

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计算这些影响的方法，即损失函数相对于参数的梯度。

## Backpropagation

Partial Derivatives

$$\frac{\partial E}{\partial z} = z - t$$

$$\frac{dz}{ds} = g'(s) = z(1 - z)$$

$$\frac{\partial s}{\partial y_1} = v_1$$

$$\frac{dy_1}{du_1} = y_1(1 - y_1)$$

Useful notation

$$\delta_{\text{out}} = \frac{\partial E}{\partial s} \quad \delta_1 = \frac{\partial E}{\partial u_1} \quad \delta_2 = \frac{\partial E}{\partial u_2}$$

Then

$$\delta_{\text{out}} = (z - t) z (1 - z)$$

$$\frac{\partial E}{\partial v_1} = \delta_{\text{out}} y_1$$

$$\delta_1 = \delta_{\text{out}} v_1 y_1 (1 - y_1)$$

$$\frac{\partial E}{\partial w_{11}} = \delta_1 x_1$$

Partial derivatives can be calculated efficiently by backpropagating deltas through the network.

14



## Bayes' Formula

### Bayes' Rule (3.11)

The formula for conditional probability can be manipulated to find a relationship when the two variables are swapped:

$$P(A \wedge B) = P(A | B)P(B) = P(B | A)P(A)$$

$$\rightarrow \text{Bayes' rule} \quad P(A | B) = \frac{P(B | A)P(A)}{P(B)}$$

This is often useful for assessing the probability of an underlying *Cause* after an *Effect* has been observed:

$$P(\text{Cause} | \text{Effect}) = \frac{P(\text{Effect} | \text{Cause})P(\text{Cause})}{P(\text{Effect})}$$

11



贝叶斯定理是概率论中的一个重要定理，用于描述在已知某些其他条件或信息的情况下，某事件发生的条件概率。贝叶斯定理通常用于从先验概率（Prior Probability）和似然度（Likelihood）来更新或计算后验概率（Posterior Probability）。

贝叶斯定理的数学表达式为：

$$P(A | B) = \frac{P(B | A) \times P(A)}{P(B)}$$

其中：

- $P(A)$  和  $P(B)$  是事件  $A$  和  $B$  的先验（或边际）概率。
- $P(A|B)$  是给定  $B$  发生时  $A$  的后验概率。
- $P(B|A)$  是给定  $A$  发生时  $B$  的条件概率，也称为似然度。

## 例子：医学检测

假设有一个用于检测某种疾病的医学测试，该测试的准确率（True Positive Rate）为 99%，误报率（False Positive Rate）为 1%。在一群人中，有 1% 的人患有这种疾病。

- $A$ ：一个人患有这种疾病。
- $B$ ：测试结果为阳性。

我们知道：

- $P(A) = 0.01$  (先验概率)
- $P(B|A) = 0.99$  (True Positive Rate)
- $P(B) = P(B|A) \times P(A) + P(B|A^c) \times P(A^c) = 0.99 \times 0.01 + 0.01 \times 0.99 = 0.0099 + 0.0099 = 0.0198$  (总阳性率)

使用贝叶斯定理，我们可以计算出给定测试结果为阳性的情况下，一个人实际患病的概率（后验概率）：

$$P(A|B) = \frac{P(B|A) \times P(A)}{P(B)} = \frac{0.99 \times 0.01}{0.0198} \approx 0.5$$

意外地，即使测试的准确率非常高（99%），由于疾病本身的低发病率（1%），测试结果为阳性的人实际患病的概率仍然只有约 50%。

贝叶斯定理在多个领域都有广泛应用，包括医学诊断、自然语言处理、机器学习和人工智能等。它提供了一种基于现有证据更新我们对不确定事件概率估计的方法。

## Types of Learning

## Types of Learning (5.1)

- Supervised Learning
  - agent is presented with examples of inputs and their target outputs
- Reinforcement Learning
  - agent is not presented with target outputs, but is given a reward signal, which it aims to maximize
- Unsupervised Learning
  - agent is only presented with the inputs themselves, and aims to find structure in these inputs

3



### 1. 监督学习 (Supervised Learning)

监督学习就像一名考生在准备考试时只看真题学习。这里有一个具体的输入和对应的输出，学生会根据历年真题（训练数据）进行学习，并努力在未来的考试（测试数据）中获得好成绩。

- **训练阶段**：老师（算法）根据真题（训练数据）和参考答案（标签）教学生（模型）如何解答。
- **测试阶段**：学生在未见过的题目上尝试解答，老师评估其表现（精度、召回率等）。

### 2. 无监督学习 (Unsupervised Learning)

无监督学习就像一个人尝试了解一个全新的城市但没有地图。他没有具体的指导或标签，需要自己探索和观察，以找到城市中的模式和结构。

- **例如**：一个人到达一个陌生的城市，开始观察街道、建筑物、商店等，自己归纳出哪些地区是商业区，哪些地区是居民区等。这就像聚类分析，试图在没有标签的数据中找到隐藏的结构或模式。

### 3. 强化学习 (Reinforcement Learning)

强化学习就像训练宠物做特定动作。与宠物互动时，你会根据宠物的行为给予奖励或惩罚，引导它逐渐学习想要的行为。

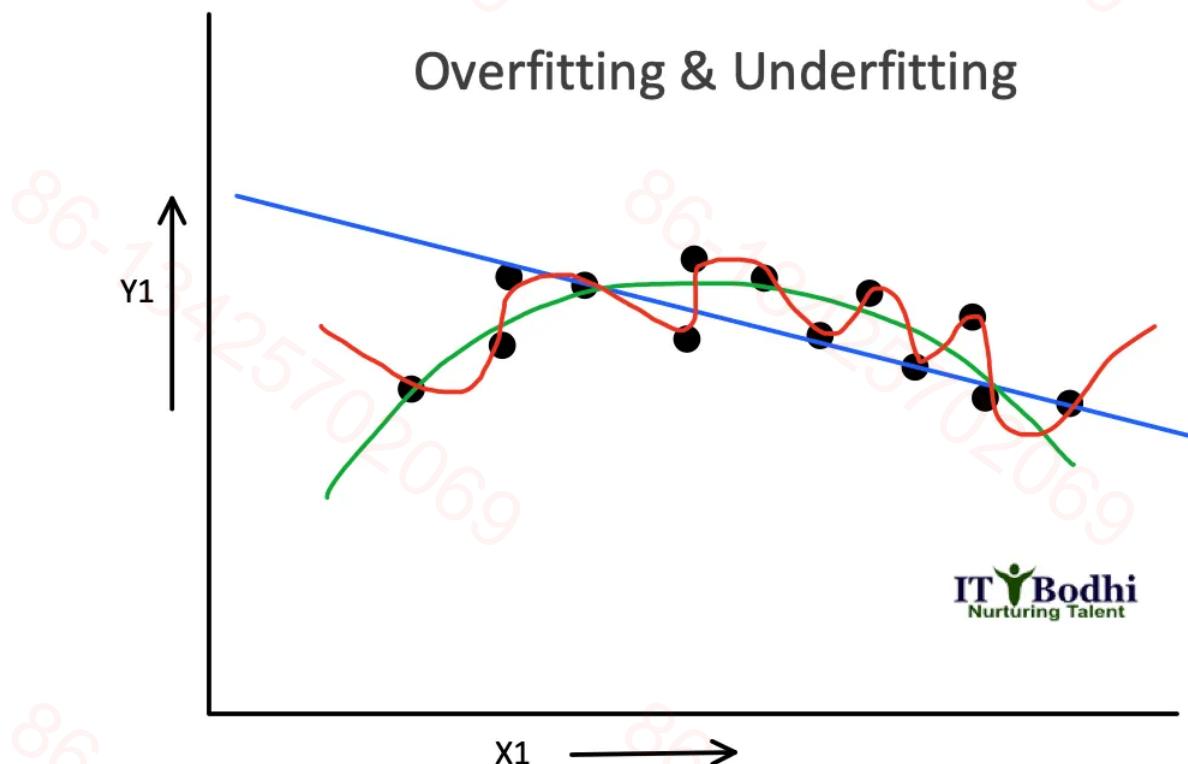
- **例如**：你正在教你的狗坐下。一开始，狗可能对你的命令反应迟钝。当它坐下时，你给它一块饼干作为奖励。随着时间的推移，你的狗学会了每次听到“坐下”这个命令时就应该坐下。在这个过程中，奖励机制（正反馈）和惩罚机制（负反馈）使得学习目标明确，激励狗学习特定的行为。

## Overfitting VS. Underfitting

过拟合 (Overfitting) 和欠拟合 (Underfitting) 是机器学习中常见的两种现象，也同样存在于神经网络的训练中。它们主要描述的是模型对训练数据和新数据的泛化能力。

**过拟合 (Overfitting)** : 是指模型在训练数据上的表现非常好，但在新的，未见过的数据上表现较差。这是因为模型过度学习了训练数据中的特性（包括噪声），以至于没有很好地抓住数据的真正、基本的趋势。过拟合往往在模型过于复杂，训练数据过少，或者训练时间过长时出现。

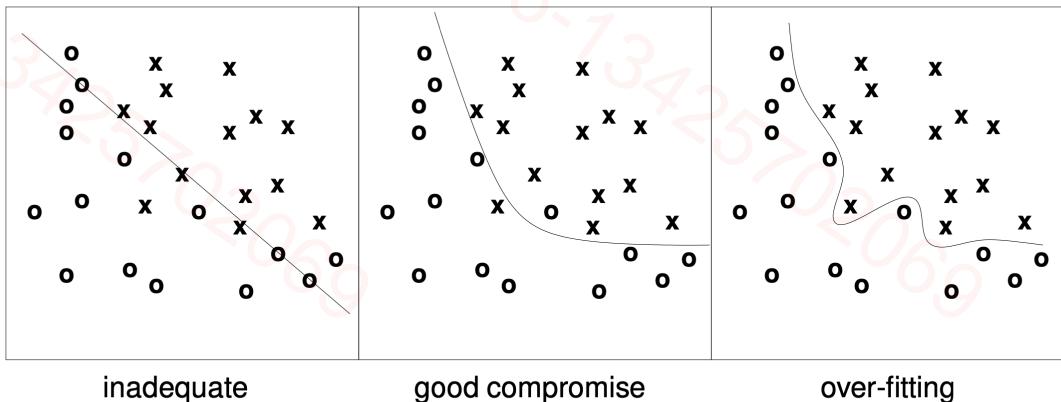
**欠拟合 (Underfitting)** : 则是模型在训练数据上的表现就不好，当然在新的，未见过的数据上也表现不好。这是因为模型没有足够地学习到数据中的特性。欠拟合通常在模型过于简单，或者训练时间不足时出现。



(Image credit to: <https://medium.com/@itbodhi/overfitting-and-underfitting-in-machine-learning-models-76cb60dbdaf6>)

## Ockham's Razor (5.2)

"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.

11



## 从数学角度思考？Bias & Variance Trade-off

在机器学习和统计学中，偏差-方差权衡（Bias-Variance Tradeoff）是一个重要的概念，它用于描述模型的一种常见困境，即如何在防止过拟合（Overfitting，高方差）和防止欠拟合（Underfitting，高偏差）之间找到平衡。

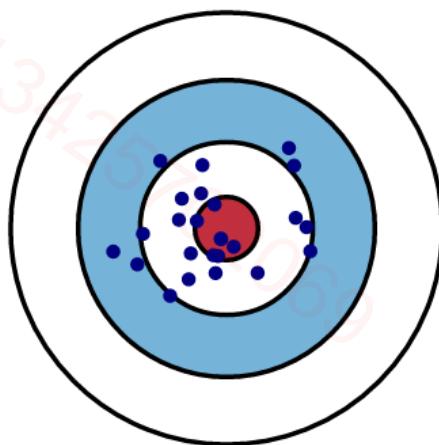
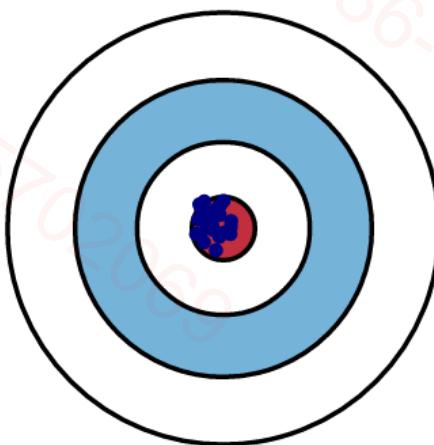
**偏差 (Bias)**：是指模型的预测值与实际值之间的差异，或者说模型的预测准确性。高偏差模型通常表现为欠拟合，即模型没有足够地学习到数据中的模式和关系，导致在训练数据和测试数据上都表现不佳。

**方差 (Variance)**：是指模型对于训练数据的微小变化的敏感度。高方差模型通常表现为过拟合，即模型过度学习了训练数据中的特性（包括噪声），使得在训练数据上表现很好，但是对于新的、未见过的数据却表现差。

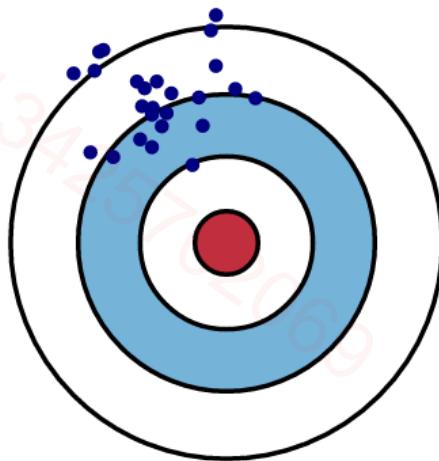
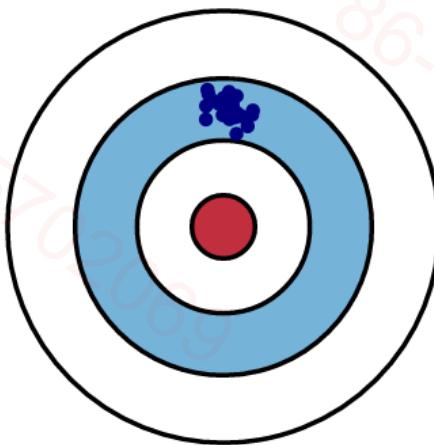
如果要降低Bias，通常做法是增加模型复杂度，然而会造成过拟合，也就是高方差，所以偏差-方差权衡揭示了一个关键的挑战：难以同时最小化偏差和方差。通常，为了减少偏差而增加模型的复杂度（如增加神经网络的层数或节点数），会使模型变得更复杂，导致方差增大，从而增加了过拟合的风险。相反，为了减少方差而简化模型，可能会导致模型不能完全学习数据的全部特性，增加偏差，从而增加了欠拟合的风险。

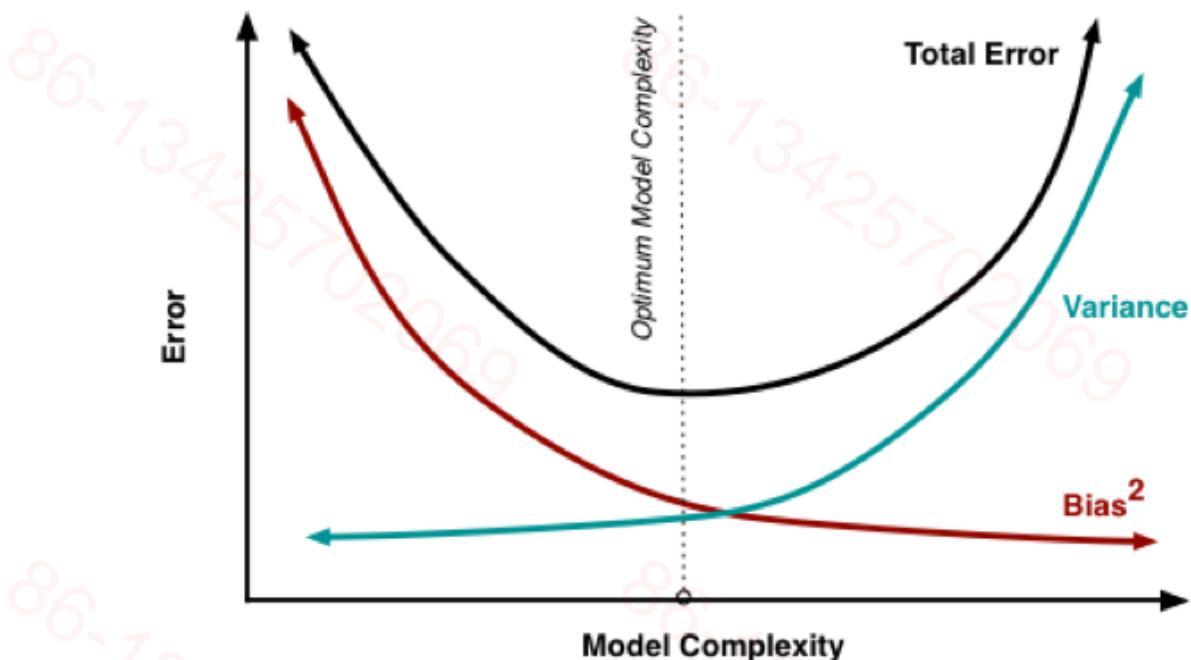
Low Variance      High Variance

Low Bias



High Bias





(Image credit to: <https://www.cs.cornell.edu/courses/cs4780/2018fa/lectures/lecturenote12.html>)

## 那我们怎么知道这个时候的bias和variance呢？

通常做法是将数据集分成training data和validation data。在training集上训练，在validation集上做测试。如果在training上很高，但是在validation上很低，就说明过拟合了。我们希望两边的结果差不多。

## Generalisation

### Ways to Avoid Overfitting in Neural Networks

- Limit the number of hidden nodes or connections
- Limit the number of training epochs (weight updates)
- Dropout
- Weight Decay (Week 3)
- Data Augmentation (Week 4)
- Regularization

## 参数初始化 Initialization.

① 初始化的高：快点收敛  
非对称性，所有向量是无 Neurons 输出相同。  
⇒ 相同的梯度，相同的参数。  
  
不能太大：梯度爆炸。  
不能太小：梯度消失。  
  
Mean 应该为 0. 每层的 variance 应该一样。

### 2) Xavier Initialization.

① Assumption: 假设函数为 tanh，  
并且在训练初期是 linear regime.

$$\text{假设 } \text{tanh}(z) \approx z - 0 \\ z^l = W^l z^{l-1} + b^l \rightarrow z^l = \tanh(z^l) \\ \text{通过线性化 } z^l = \tanh(z^l) \approx z^l \\ \text{所以, } \text{Var}(z^l) = \text{Var}(z^0)$$

$$= \text{Var}(z_1^0 w_1^1 z_2^1 + \dots) \\ = z_1^0 \text{Var}(w_1^1 z_2^1) \dots$$

$$\text{将 Variance 公式代入 } \text{② 化简.} \rightarrow \text{Var}(y^l) \rightarrow \text{Var}(y^l) \text{ had form}$$

因为 assume 假设 mean 是 0，输出是正态分布。

$$\Rightarrow \text{E}(y^l) = 0, \text{Var}(X) = 1$$

$$\Rightarrow \text{Var}(W^l y^l) = \text{Var}(W^l) \text{Var}(y^l) \dots$$

$$\Rightarrow \text{Var}(a^l) = z_1^0 \text{Var}(w_1^1 z_2^1) \dots \cdot z_n^0 \text{Var}(w_n^l z^l) \dots$$

$$\Rightarrow \text{Var}(W^l) = \frac{1}{n} \text{Var}(z^0) \text{backward.} \text{Var}(W^l) = \frac{1}{n}$$

② 初始值公式：let  $n = n^{l-1} n^l$

Xavier normal:  $N(0, 1/n)$

Xavier uniform:  $U(-\sqrt{1/n}, \sqrt{1/n})$ .

### 3) He initialization 选择的激活函数 ReLU

$$\text{① 推导. } \text{Var}(y) = \text{Var}(Z^T w + b) \\ = n \text{Var}(w) \text{Var}(Z^T) \text{ by } \text{Var}(b) = \text{E}(b^2) - \text{E}(b)^2 \\ \text{因为 ReLU, } x^k = \max(x^k, 0) \\ \Rightarrow \text{E}(x^k) = \frac{1}{2} \text{Var}(x^k) \\ \Rightarrow \text{Var}(x^k) = n \text{Var}(w) \cdot \frac{1}{2} \text{Var}(x^k) \\ \Rightarrow \text{Var}(w) = 2/n.$$

### ② 公式：

He normal:  $N(0, 2/n)$

He uniform:  $U(-\sqrt{2/n}, \sqrt{2/n})$

### 2) Weight Decay (各种 Lp 正则化解释)

① Motivation: 前向梯度下降更好的参数。

$$\text{假设 } l(\theta) = \hat{l}(\theta) + \frac{\lambda}{2} \|W\|^2 \text{ Engineers view:} \\ \hat{l}(\theta) = l(\theta) + \frac{\lambda}{2} \|W\|^2 \text{ 表示梯度 } \frac{\partial}{\partial \theta} \text{ of parameter}$$

### ② 数据集加入 Noise 相当于加入正则化。

$$\begin{aligned} g &= f(x) - W^T x - E \sim N(0, \lambda I) \\ l(y) &= E_{\theta} [f(x) - y]^2 = E_{\theta} [f(x) + W^T g]^2 \\ &= E_{\theta} [f(x) - y]^2 + 2E_{\theta} [W^T g]^2 + E_{\theta} [g]^2 \\ &= E_{\theta} [f(x) - y]^2 + 2\lambda \|W\|^2 \end{aligned}$$

### 3) Dropout → 参数的稀疏性。

① Motivation: 只用少部分参数进行更新。

② 实现: 训练阶段 P% 的神经元被遮住。

测试阶段 使用所有神经元。Weight scale down P%.

g. Inverted Dropout { At training: weight = 1/p weight.

At testing: use the whole network.

相当于训练了 p 个网络，测试时取所有结果。

③ 优势: ① scale free 对大规模到 feature 不会有影响。

② Invariant to parameter scaling

④ Dropoutnet: 相当于随机丢弃神经元，随机丢弃连接。

→ 更丰富的网络组合。

### 4) Batch Normalization 减少 feature unit 带来的影响。→ 更大的学习率。

① Motivation: 容易 internal covariance shift, 加速收敛, 缓解 gradient vanishing.

② Group Normalizations: BN 在 batch size 上表现不好。

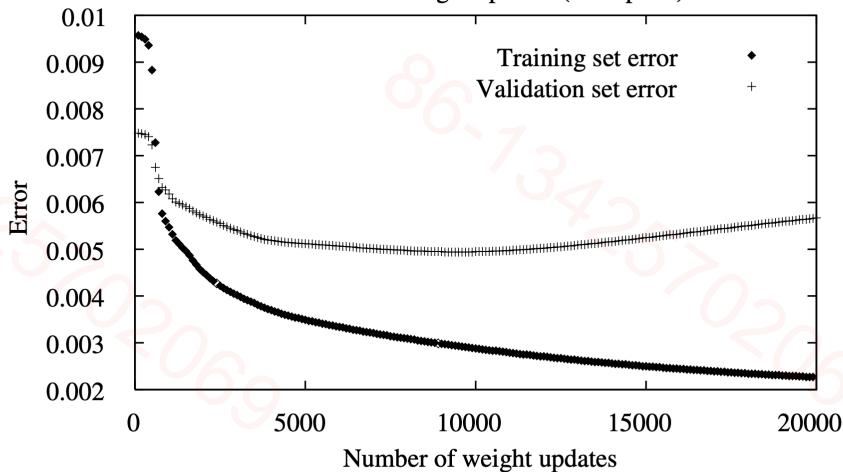
将 channel 分成 group 后归一化，对 batch size 不敏感。

Reference: COMP5329 Deep Learning. Dr Chang Xu  
University of Sydney. 卷尾文字。

## Error Plots

### Training, Validation and Test Error

Error versus weight updates (example 1)



x-axis could be number of weight updates, hidden nodes, dropout, etc.

Training error decreases but validation and test error flatter or increase.

为什么我们在乎分隔数据集来进行模型评估？直接把所有的数据拟合一下，然后抽一点出来测试不就好了吗？

如果把所有的数据拿来拟合，也就是说模型已经将这些数据都见过一遍了，就好比模型已经提前见过了所有的答案。只要参数越大，模型的效果就会越好。我们从无从得知模型是否过拟合了。为了避免这个问题，我们需要保留一份数据专门用来做测试。所以我们就有了两个概念，一个叫做**training data**，另外一个叫做**testing data**。

但是在更多的情况下，比如kaggle竞赛的时候，**testing data**是我们无法得到的，所以我们还要从**training data**自己分一部分数据出来，作为测试。这部分数据叫做**validation set**。

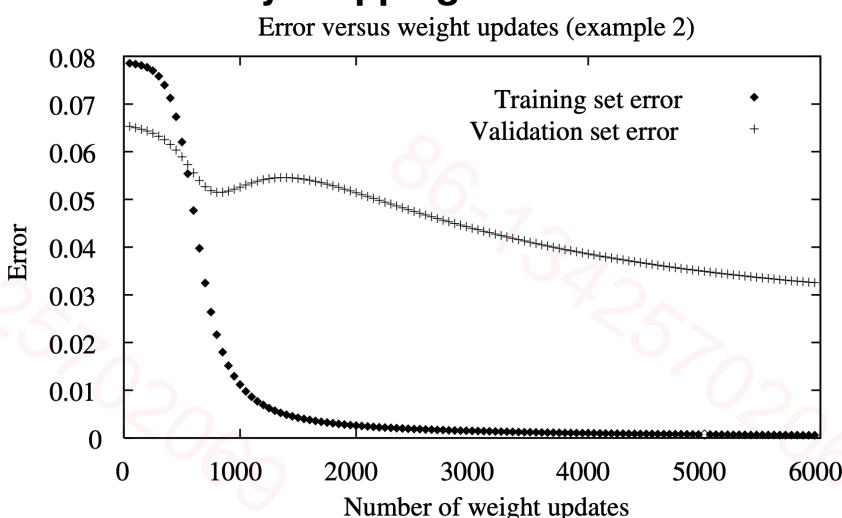
- **训练数据（Training Data）**：用于训练模型的参数。
- **验证数据（Validation Data）**：用于调整模型的超参数和检查模型是否过拟合。
- **测试数据（Testing Data）**：用于最终评估模型的性能。

主要原因：

1. 防止过拟合（Overfitting）
2. 超参数调优（Hyperparameter Tuning）
3. 泛化性能评估（Generalization Performance）
4. 避免信息泄露（Data Leakage）

## Early Stopping

### Be Careful about Early Stopping



We may need to continue training in order to be sure what is going on.

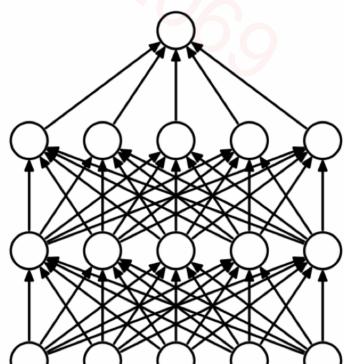
提前停止是一种用于防止过拟合（Overfitting）的正则化技术，特别常用于训练深度学习模型和其他迭代优化算法。过拟合是机器学习中一个常见的问题，出现在模型对训练数据“学得太好”，以至于它失去了对未见过的数据（即测试数据或新数据）的泛化能力，并且加速训练时间。

## 工作原理

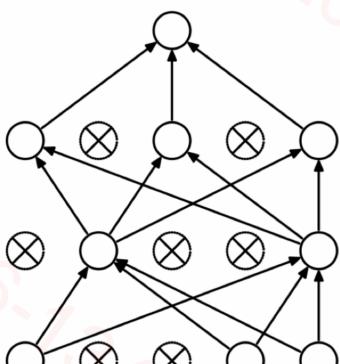
提前停止的基本思想是在模型训练过程中监控某个性能指标（通常是验证集上的损失函数或准确度）。当该指标不再改善或开始恶化时，就停止训练过程。这样，模型就不会有机会过拟合训练数据。

## Dropout

### Dropout (7.12)



(a) Standard Neural Net



(b) After applying dropout.

For each minibatch, randomly choose a subset of nodes to not be used.  
Each node is chosen with some fixed probability (usually, one half).

17



### Dropout (7.12)

- When training is finished and the network is deployed, all nodes are used, but the activation of each node is multiplied by the probability of “keeping” the node during training.
- Thus, the activation received by each node is the *average* value of the activation it would have received during training.
- Dropout forces the network to achieve *redundancy* because it must deal with situations where some features are missing.
- Another way to view dropout is that it implicitly (and efficiently) simulates an *ensemble* of different architectures.

18



简而言之就是，模型太复杂了。通过随机关闭一些神经元，让模型的复杂度下降，push模型中的每个神经元学到最关键的信息，并且保持独立性。

Dropout 是深度学习中常用的一种正则化技术，特别是在训练大型神经网络时。它的核心思想是在每次训练迭代中随机地“关闭”（即设置为0）网络中的一部分神经元，以防止过拟合。

以下是关于 Dropout 的一些关键点和细节：

## 1. 工作原理:

- 在每次训练迭代中，每个神经元都有概率  $p$  被丢弃，这意味着在前向传播和反向传播过程中，这个神经元不会有任何贡献。
- 在测试或验证过程中，我们不使用 Dropout，即所有神经元都是活跃的。但为了平衡训练时的丢弃，神经元的输出会乘以  $p$ 。

## 2. 为什么有效:

- **防止过拟合**: 通过随机关闭神经元，Dropout 强迫网络在每次迭代中使用不同的神经元子集进行学习。这样，网络不太可能依赖于任何单个神经元或特定的神经元组合，从而增强了其泛化能力。
- **模拟集成学习**: 每次迭代都使用不同的网络结构，这可以视为训练多个神经网络的集成。在测试时，我们使用整个网络，这可以视为这些模型的平均，从而增加了鲁棒性。

## Ensembling

简单来说就是将不同角度学习的特征综合在一起进行最终的决定，以构建一个全面的预测。

集成方法是一种机器学习策略，其中多个分类器（或回归器）被训练用于解决同一任务，最终的输出是通过这些模型的“投票”来决定的。

## 多样性 (Diversity)

要从集成方法中受益，不同的分类器之间需要有多样性。这种多样性可以通过以下几种方式来实现：

- **不同的算法**：使用不同类型的机器学习算法，如决策树、支持向量机、神经网络等。
- **不同的超参数**：对于同一算法，使用不同的超参数。
- **不同的特征子集**：每个模型使用输入特征的不同子集。
- **不同的训练数据子集**：使用自举采样（Bootstrap Sampling）或其他方法从原始训练数据中生成不同的训练数据子集。

## 实例：多模型集成

例如，在一个 Kaggle 竞赛中，我们可能会训练三个不同结构的神经网络，三个使用不同维度和核函数的支持向量机，以及两个其他类型的分类器。然后，所有这些模型会被集成在一起，以产生一个最终的结果。

## Dropout 与集成

Dropout 是一种特殊的正则化技术，用于防止神经网络过拟合。在每次训练迭代中，Dropout 通过随机地关闭一部分节点（神经元）来生成网络的不同“子结构”。

- **隐式集成**：通过将每个节点的输出乘以 Dropout 的概率，隐式地对所有这些不同模型的输出进行了平均。这可以视为一种简单的集成方法。

## 优点与缺点

- **优点：**
  - 提高模型的泛化能力。
  - 可以捕捉数据中更复杂的模式。
- **缺点：**
  - 计算成本高，因为需要训练和预测多个模型。
  - 可能会导致模型变得复杂，难以解释。

## Softmax



### Softmax: Converting output values to probabilities

- The NN outputs can be post-processed to turn them into probabilities
- Motivation: interpret the outputs as probabilities that sum up to 1
- Let the NN outputs are  $(o_1, \dots, o_n)$ ;  $n$  – number of output neurons
- Softmax transformation:  $p_i = \frac{e^{o_i}}{\sum_j e^{o_j}}$
- Example: 3 output neurons with values:  $o_1=0.3$ .  $o_2=0.8$  and  $o_3=0.2$
- Applying softmax:  $p_1=0.28$ ,  $p_2=0.46$ ,  $p_3=0.26$
- Checking:  $0.28+0.46+0.26=1$

Softmax 是机器学习和深度学习中常用的函数，特别是在分类问题中。它可以将一个向量的值转换为概率分布，这些概率分布常用于多类分类任务中表示模型对每个类别的预测概率。

以下是关于 Softmax 的详细介绍：

### 1. 定义:

给定一个向量  $z = [z_1, z_2, \dots, z_n]$ ，其中  $n$  是类别的数量，Softmax 函数的输出是另一个向量  $\sigma(z) = [\sigma(z_1), \sigma(z_2), \dots, \sigma(z_n)]$ ，其中每个元素  $\sigma(z_i)$  的计算方式为：

$$\sigma(z_i) = \frac{e^{z_i}}{\sum_{j=1}^n e^{z_j}}$$

## 2. 性质:

- **输出范围**：Softmax 函数的输出值位于 0 和 1 之间，这使其成为表示概率的理想选择。
- **归一化**：Softmax 函数的输出值之和总是等于 1，这确保了它们形成了一个合适的概率分布。
- **非线性**：尽管 Softmax 对每个输入元素进行指数运算，但它是一个非线性函数。

## 3. 应用:

- 在神经网络的分类任务中，Softmax 常用于输出层，将网络的原始输出转换为概率分布。
- 与 Softmax 结合使用的损失函数是交叉熵损失，它度量模型的预测概率分布与真实概率分布之间的差异。

## 4. 直观理解:

想象有一个多类分类任务，网络的原始输出是一组分数或对数，其中每个数值代表一个类别的得分。这些分数可能会很大或很小，而且没有明确的界限。Softmax 函数的作用是将这些分数转换为一个清晰、归一化的概率分布，这样我们就可以清楚地看到模型对每个类别的预测信心有多强。

总的来说，Softmax 是一个非常有用的函数，它可以将网络的原始输出转换为明确、解释性强的概率分布，从而使我们更容易解释和理解模型的预测结果。

## Cross-entropy

- Using cross entropy loss instead of MSE
- Classification task, one-hot encoding to represent class labels
  - C – number of classes, N - number of examples
  - $y_i$  – a one-hot encoded class of example  $i$  (C-dimensional)
  - $\hat{y}_i$  - predicted class (C-dimensional vector)
- Categorical cross entropy loss for the classification of example  $i$ :

$$CCE_i = - \sum_{j=1}^C y_{ij} \cdot \log \hat{y}_{ij}$$

- Cross entropy loss function - sum of losses of individual examples:

$$CCE = \sum_{i=1}^N CCE_i$$

---

56

交叉熵 (Cross-Entropy) 是评估概率分布之间相似性的一个重要指标，经常被用作深度学习分类问题中的损失函数。交叉熵损失函数度量真实概率分布与模型预测概率分布之间的差异。

以下是关于交叉熵的详细介绍：

## 1. 定义:

对于两个概率分布  $p$  和  $q$ , 交叉熵定义为 :

$$H(p, q) = - \sum_i p(i) \log q(i)$$

其中,  $p(i)$  是真实概率分布的第  $i$  个值, 而  $q(i)$  是预测概率分布的第  $i$  个值。

## 2. 直观理解:

交叉熵度量我们使用预测分布  $q$  来表示真实分布  $p$  时所需要的"惊讶度"或"不确定性"。如果  $q$  完全匹配  $p$ , 那么交叉熵的值就是  $p$  的熵; 否则, 其值会增大, 表示两个分布之间的差异。

## 3. 应用于机器学习:

在分类任务中,  $p$  通常是一个独热编码的向量, 表示真实的类别, 而  $q$  是模型的预测概率分布, 通常来自于 Softmax 函数的输出。交叉熵损失函数度量模型的预测与真实标签之间的差异。

## 4. 性质:

- **非负性**：交叉熵的值总是非负的，且当  $p$  和  $q$  完全相同时，其值最小，为0。
- **用于优化**：交叉熵损失函数与 Softmax 激活函数结合使用时，其梯度具有良好的性质，这使得模型能够有效地学习。

## 5. 与 KL 散度的关系：

交叉熵与 Kullback-Leibler 散度（KL 散度）密切相关。KL 散度度量两个概率分布之间的差异，可以视为交叉熵与真实分布的熵之差。

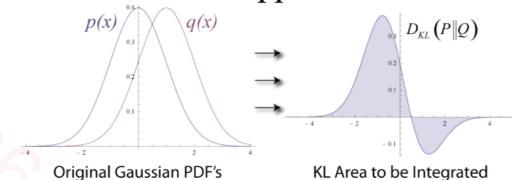
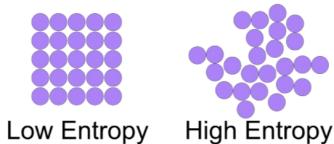
## Cross Entropy Loss

- 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.

**KL (Kullback-Leibler)**  
**divergence** is a measure  
of the information lost  
when Z is used to  
approximate T



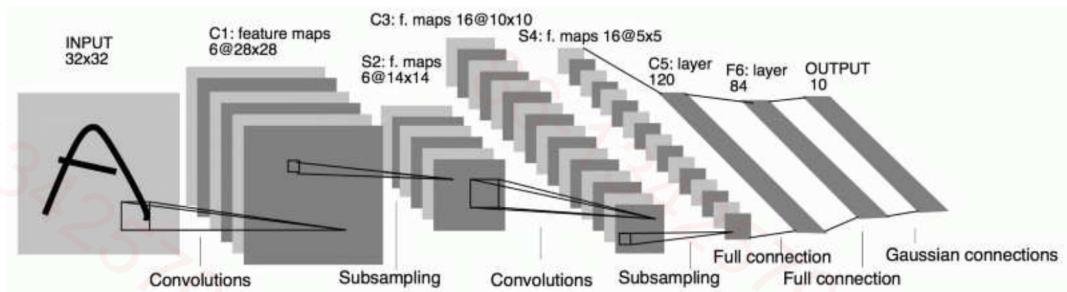
The University of Sydney

Page 37

## 深度神经网络启蒙发展时期

### LeNet — 1998

<https://hal.science/hal-03926082/document>



- the  $5 \times 5$  window of the first convolution layer extracts from the original  $32 \times 32$  image a  $28 \times 28$  array of features; subsampling then halves this size to  $14 \times 14$ .
- the second Convolution layer uses another  $5 \times 5$  window to extract a  $10 \times 10$  array of features, which the second subsampling layer reduces to  $5 \times 5$ .
- these activations then pass through two fully connected layers into the 10 output units corresponding to the digits '0' to '9'.

15



LeNet 是由 Yann LeCun 在 1990 年代提出的，这是一种早期用于手写数字识别（特别是用于 USPS 数据集和 MNIST 数据集）的卷积神经网络。这个网络模型为后来的卷积神经网络（CNN）发展奠定了基础，并且 Yann LeCun 也因这项工作而获得了图灵奖。

## 主要技术和特点：

1. **卷积层（Convolutional Layers）**：LeNet 是最早采用卷积层的网络之一，用于从输入图像中自动学习空间层次特征。
2. **子采样（Subsampling）/平均池化（Average Pooling）**：在卷积层之后，LeNet 使用子采样或平均池化层来减少每个特征图（Feature Map）的维度。
3. **全连接层（Fully-Connected Layers）**：接在卷积层和池化层之后的是一到两个全连接层，用于执行高级推理和分类。
4. **S型激活函数（Sigmoid Activation Functions）**：在当时，LeNet 使用了 S型激活函数，这是那个时代的标准选择。
5. **局部连接和权重共享**：与传统的全连接神经网络不同，LeNet 的卷积层采用局部连接和权重共享，从而大大减少了模型参数数量。
6. **灰度图像输入**：由于当时的计算和数据存储限制，LeNet 通常用于灰度图像，而不是彩色图像。

## 网络结构：

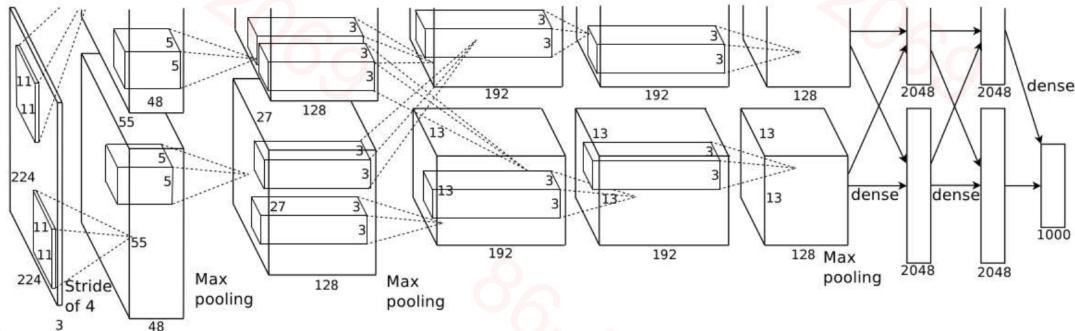
- 2 个卷积层（Convolutional Layers）
- 2 个平均池化层（Average Pooling Layers）
- 3 个全连接层（Fully-Connected Layers）

点评：尽管 LeNet 在当时的硬件环境下被限制于相对简单的任务（如手写数字识别），但它的核心思想和基础结构对后续的卷积神经网络发展产生了深远影响。LeNet 和其对应的 MNIST 数据

集已经成为深度学习和计算机视觉领域的“Hello, World!”，经常被用作教学和入门级研究。

## AlexNet — 2012

<https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf>



- 5 convolutional layers + 3 fully connected layers
- max pooling with overlapping stride
- softmax with 1000 classes
- 2 parallel GPUs which interact only at certain layers

## AlexNet 网络结构

AlexNet 是由 Alex Krizhevsky、Ilya Sutskever 和 Geoffrey Hinton 在 2012 年提出的，并在当年的 ImageNet 大规模视觉识别挑战赛 (ILSVRC) 中取得了突破性的成绩。这一模型可以说是现代深度学习历史上的一个里程碑，因为它首次证明了深度神经网络能在大规模数据集上实现高效、准确的视觉识别。

### 主要技术和创新点：

1. **更深的网络结构**：与 LeNet 相比，AlexNet 的网络更深，具有 12 层（包括卷积层、全连接层和激活函数层等）。这大大增加了网络的表达能力。
2. **ReLU 激活函数**：AlexNet 是首个大规模应用 Rectified Linear Unit (ReLU) 激活函数的深度神经网络。ReLU 函数非常简单但有效，能加速神经网络的训练。
3. **Dropout 正则化**：为了减少过拟合，AlexNet 在全连接层中使用了 Dropout 正则化方法。
4. **数据增强**：AlexNet 使用了数据增强技术，如图像翻转、裁剪和颜色变化，以增加模型的泛化能力。
5. **局部响应归一化 (Local Response Normalization)**：这是一种正则化方法，用于模仿生物神经系统，减少训练过程中的过拟合。
6. **重叠最大池化 (Overlapping Max Pooling)**：与传统的池化层相比，重叠最大池化可以减少过拟合。

7. **GPU 加速**：由于其深度和复杂性，AlexNet 的训练非常耗时。因此，AlexNet 使用了两个并行的 GPU 进行计算，这在当时是一个创新之举。
8. **分布式存储和计算**：由于网络非常大，一台机器可能无法存储所有的参数和中间状态。AlexNet 的设计允许分布式存储和计算。

点评：AlexNet 使用了多种技术的组合，这些技术目前还被广泛地用在了视觉模型上，AlexNet 可以说一个非常经典的Benchmark，可以说是神经网络在CV上的开山鼻祖。

## VGG — 2014

<https://arxiv.org/pdf/1409.1556.pdf>

VGG 是由牛津大学的 Visual Geometry Group 团队在 2014 年提出的，特点是其简单的结构和深度。该模型在 ImageNet 竞赛中表现出色，证明了深度网络的有效性。

### 主要技术和创新点：

1. **统一的卷积核大小**：VGG 网络最显著的特点是它使用一种统一的小尺寸（3x3）卷积核，而不是使用多种不同尺寸的卷积核。这大大简化了网络结构。
2. **更深的网络结构**：VGG 的另一个重要特点是它非常深，有 16 层（VGG-16）或 19 层（VGG-19）版本。这样的深度帮助网络捕获更复杂、更抽象的特征。
3. **池化层**：VGG 使用了多个最大池化层（Max Pooling）来逐渐减小特征图的尺寸，从而减少计算量。

点评：使用小尺寸的 3x3 卷积核是 VGG 的一项主要创新。通过叠加多个小卷积核，网络能以更少的参数和更少的计算量达到与大卷积核相同的感受野。尽管 VGG 网络参数多、计算量大，但其结构的简单性和易于理解的特点使其成为深度学习研究和应用中的一个重要基准模型。

## GoogLeNet — 2014

<https://arxiv.org/pdf/1409.4842.pdf>

GoogLeNet 是由 Google 的研究人员在 2014 年提出的，并在当年的 ImageNet 竞赛中获得了第一名。这一模型采用了一种名为 “Inception” 的新型架构，使其在不增加计算复杂性的情况下具有更大的深度和宽度。

### 主要技术和创新点：

1. **Inception 模块**：GoogLeNet 的核心是 Inception 模块，该模块并行地使用多种尺寸的卷积核和最大池化，然后将所有结果拼接在一起。这样做的好处是网络能自动地从多尺度特征中学习。
2. **深度和宽度**：通过使用多个 Inception 模块，GoogLeNet 达到了非常高的深度（总共 22 层），同时也具有较大的宽度。
3. **全局平均池化**：与其他流行模型（如 AlexNet 和 VGG）不同，GoogLeNet 使用全局平均池化层代替全连接层，以减少模型参数并缓解过拟合。

4. **辅助分类器**：为了解决梯度消失问题，GoogLeNet 在网络的中间层添加了辅助分类器，用于生成额外的损失。

5. **高效的计算**：虽然模型非常深，但由于 Inception 模块和全局平均池化的使用，GoogLeNet 的计算效率非常高。

点评：GoogLeNet 是第一个成功地展示了深度和宽度都可以通过合适的架构来扩展的模型，而不仅仅是通过堆叠更多的层。这一模型开启了一系列后续研究和改进，包括多个 Inception v2、v3、v4 和 Inception-ResNet 等变体。

## Images

### 2D Image

Images are represented as matrices of pixel values

- Black and white images: a value from 0 (black) to 255 (white) – 1 matrix
- Color images: the same but for 3 channels: red, green and blue, so 3 matrices

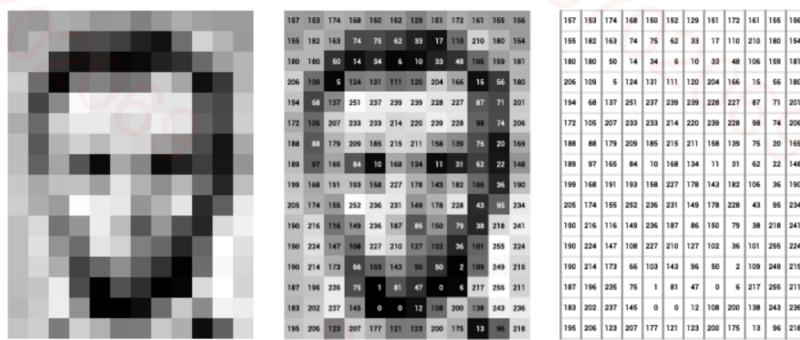


Image ref: <https://ai.stanford.edu/~syyeung/cvweb/tutorial1.html>

图片（2D）本质上就是一个由光亮度表示的一张矩阵。从物理角度来看，图片是对现实世界中光的一种抽象表示。具体来说，当你看到一张图片，你实际上看到的是一个二维矩阵，其中每个元素（或像素）代表了一个特定位置的光亮度。

### 像素 (Pixel)

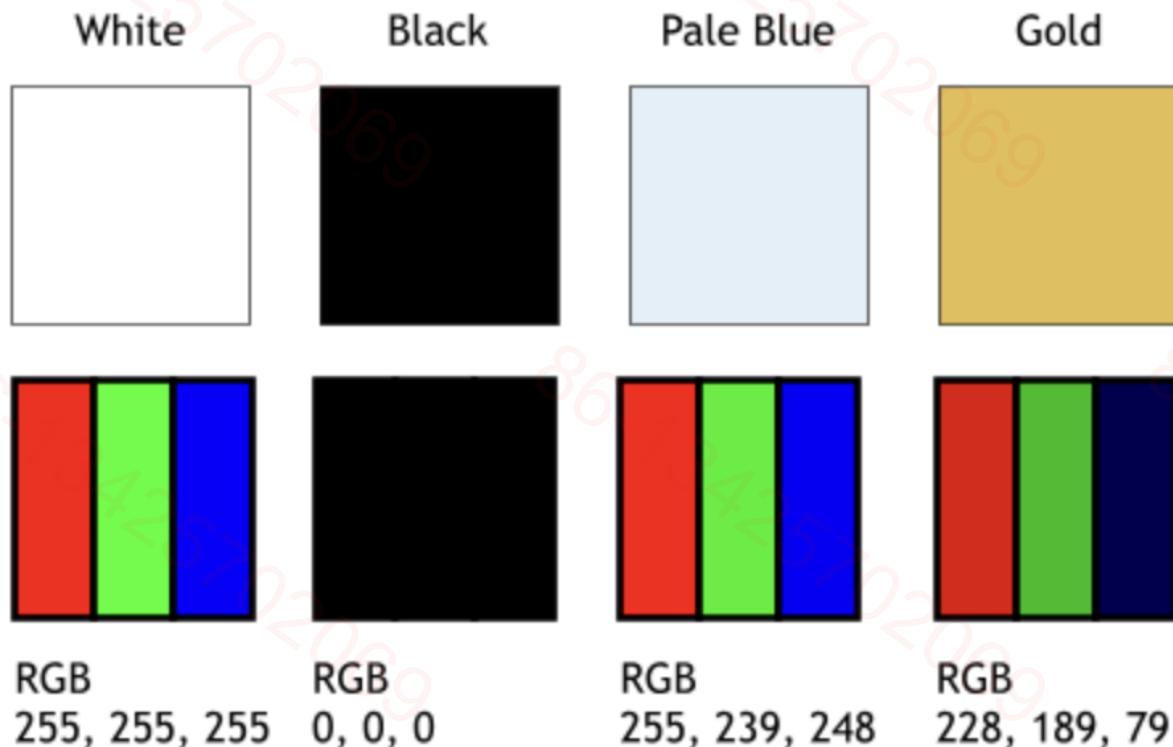
- **像素是图片的基本单元**：在这个二维矩阵中，每一个小方格就是一个像素。
- **灰度与颜色**：在灰度图像中，每个像素通常用一个介于 0 到 255 之间的整数来表示，其中 0 表示黑色，255 表示白色，其他值表示不同的灰度。在彩色图像中，每个像素通常由三个数值组成，分别代表红、绿和蓝（RGB）三种颜色的亮度。

### 精度与范围

- **数值范围**：尽管 0-255 是最常见的像素值范围，但这并不是唯一的选择。根据需要，这个范围可以被扩展。例如，在医学成像或卫星成像中，可能需要更高的精度。

- **更多信息**：像素值范围越大，每个像素能表示的信息就越多。这就意味着图像可以捕获更多关于光（例如，亮度、颜色等）的细节。

## RGB Image



<https://www.datarobot.com/blog/introduction-to-computer-vision-what-it-is-and-how-it-works/>

RGB 图片可以从一个“3D”的角度来理解，因为它实际上是由三个单色（即单通道）图像组合而成的。这三个单色图像分别表示红色（Red）、绿色（Green）和蓝色（Blue）三个颜色通道。这三个通道的图像具有相同的尺寸，并且它们是按照深度方向（第三个维度）堆叠在一起的。

## 为什么是“3D”？

- **三个通道**：红、绿和蓝这三个通道分别是一个 2D 图像，它们堆叠在一起就形成了一个 3D 数据体。
- **像素深度**：在每个像素位置，你都会有三个数值，分别对应红、绿和蓝通道。你可以想象在每个像素位置都有一个小小的三维向量。

## 彩色信息

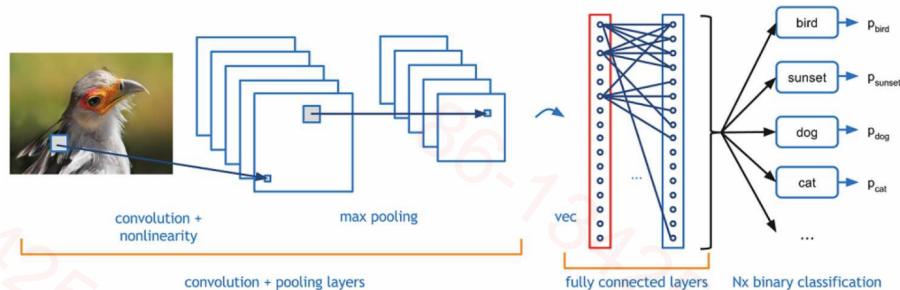
- **颜色组合**：通过改变这三个通道中每个像素的亮度，你可以形成各种各样的颜色。例如，红色和绿色的组合会产生黄色。
- **富有表现力**：这种方式使得 RGB 图像能以非常高的精度和丰富度来表示现实世界中的颜色。

## 在 CNN 中的应用

- **多通道输入**：当使用卷积神经网络（CNN）处理 RGB 图片时，通常将整个 3D 数据体作为网络的输入。这就是为什么在设置 CNN 时，输入层的通道数通常设置为 3。
- **特征学习**：由于有三个通道，CNN 能够学习从各个通道中捕获的复杂特征和它们之间的相互作用。

## 卷积神经网络

### Convolutional Networks (7.9)

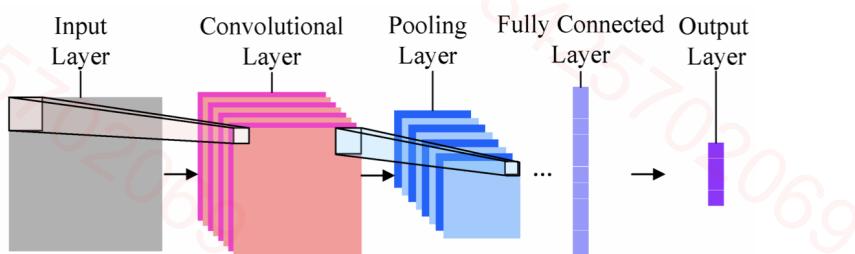


- Suppose we want to classify an image as a bird, sunset, dog, cat, etc.
- If we can identify features such as feather, eye, or beak which provide useful information in one part of the image, then those features are likely to also be relevant in another part of the image.
- We can exploit this regularity by using a *convolution layer* which applies the same weights to different parts of the image.

3



### Convolutional Network Components

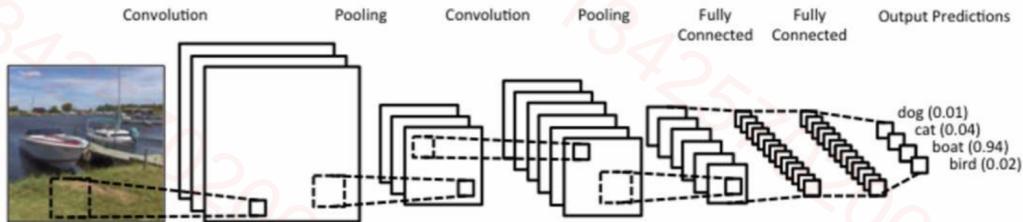


- **convolution layers**: extract shift-invariant features from the previous layer
- **subsampling or pooling layers**: combine the activations of multiple units from the previous layer into one unit
- **fully connected layers**: collect spatially diffuse information
- **output layer**: choose between classes

5



# Convolutional Network Architecture



- There can be multiple steps of convolution followed by pooling, before reaching the fully connected layers.
- Note how pooling reduces the size of the feature map (usually, by half in each direction).

8



## Convolutional Layer

### Convolution Operator (9.1-9.2)

Continuous convolution

$$s(t) = (x * w)(t) = \int x(a)w(t-a)da$$

Discrete convolution

$$s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a)$$

Two-dimensional convolution

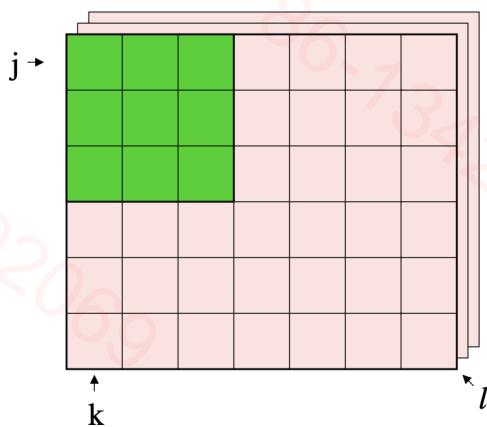
$$S(j, k) = (K * I)(j, k) = \sum_m \sum_n K(m, n)I(j + m, k + n)$$

Note: Theoreticians sometimes write  $I(j - m, k - n)$  so that the operator is commutative. But, computationally, it is easier to write it with a plus sign.

9



## Convolutional Neural Networks



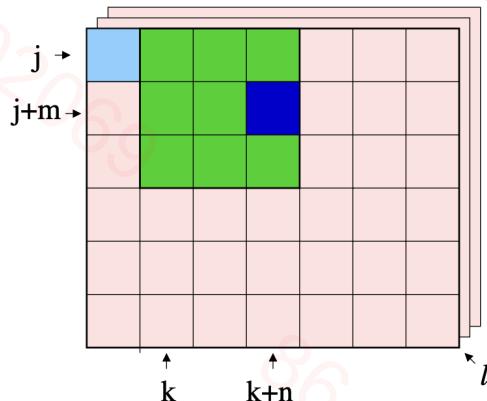
Assume the original image is  $J \times K$ , with  $L$  channels.

We apply an  $M \times N$  “filter” to these inputs to compute one hidden unit in the convolution layer. In this example  $J = 6, K = 7, L = 3, M = 3, N = 3$ .

10



## Convolutional Neural Networks



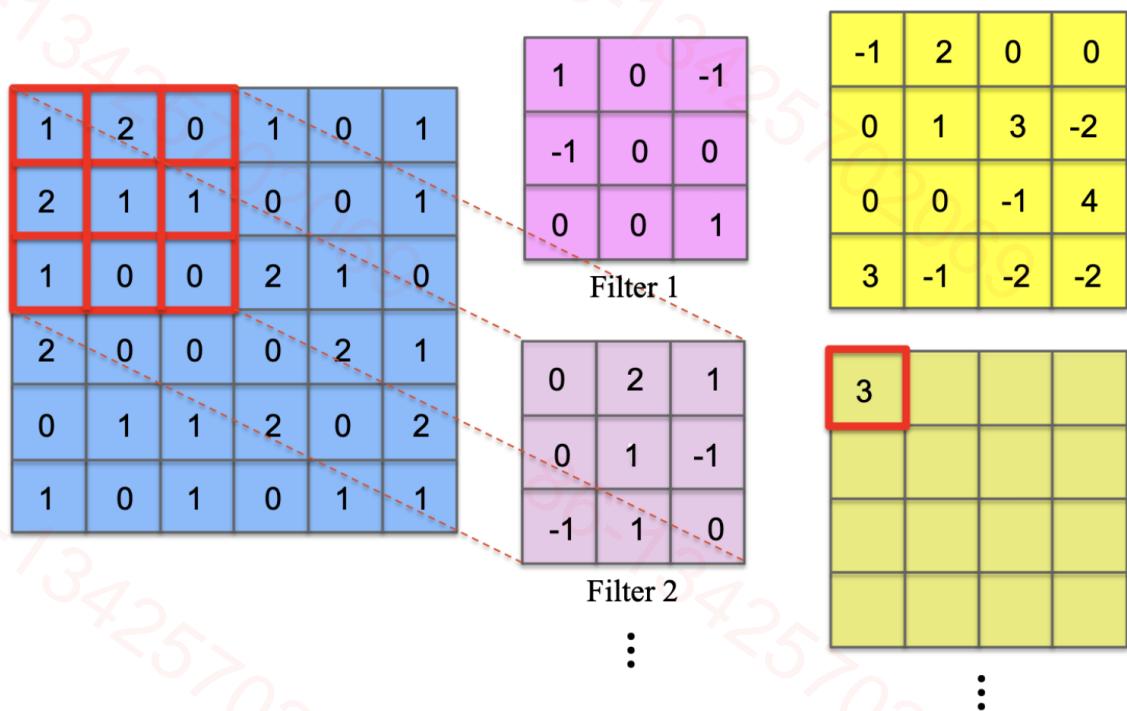
$$Z_{j,k}^i = g(b^i + \sum_l \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} K_{l,m,n}^i V_{j+m, k+n}^l)$$

The same weights are applied to the next  $M \times N$  block of inputs, to compute the next hidden unit in the convolution layer (“weight sharing”).

11



## Learn multiple filters

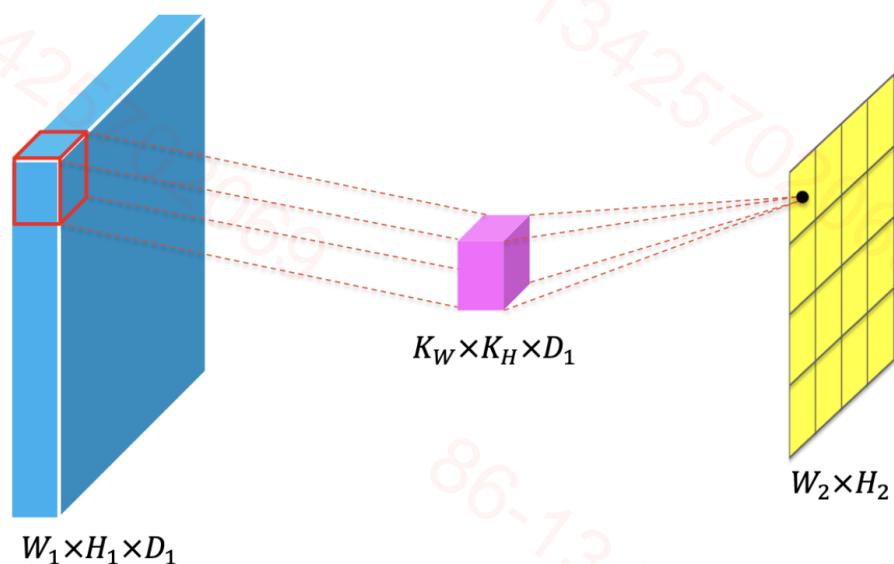


The University of Sydney

Page 23

## Convolutional Layer

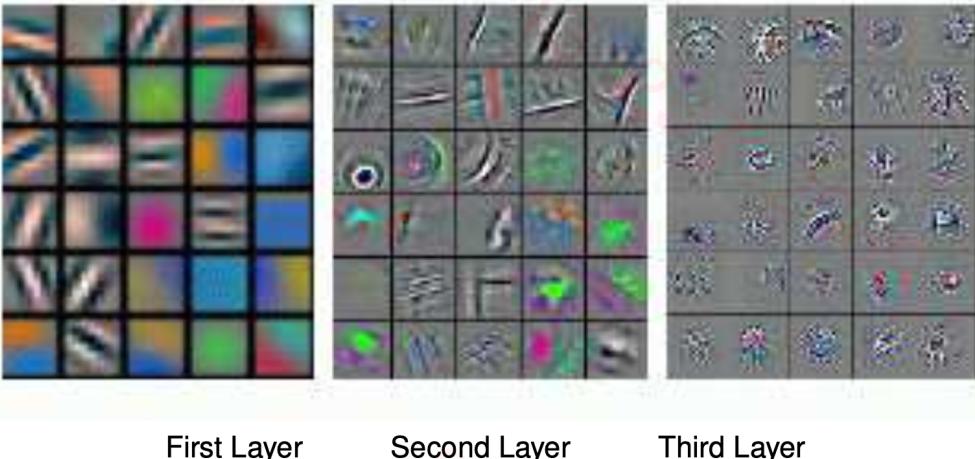
- Above, we have only considered a 2-D image as input
- When the input has depth (e.g. RGB images), the convolution ops should be...



The University of Sydney

Page 25

## Convolutional Filters



26

UNSW

在卷积神经网络（CNN）中，卷积层是最核心的组成部分。卷积的本质就是通过不同的滤镜，多角度捕捉图片的信息从而学习特征，正所谓横看成林侧成峰，卷积层负责从输入图像中提取局部特征，这些特征随着网络层级的增加而逐渐变得更抽象和复杂。

### 卷积操作

- **局部感受野**：卷积操作是通过一个小的滤波器（或卷积核）在输入图像上滑动来实现的。这个滤波器只覆盖图像的一个小区域（局部感受野）。
- **特征映射**：每次滤波器滑过一个局部区域时，都会进行一次点乘和求和操作，生成一个新的像素值。这样，滤波器便从输入图像中提取出一个特征映射（或特征图）。

### 卷积与傅立叶变换

- **频域分析**：傅立叶变换是一种在频域中分析信号的工具。它能将一个复杂的信号分解成多个简单的正弦和余弦波。
- **局部与全局**：与傅立叶变换不同，卷积操作是局部的，它只关注输入图像的小区域。然而，在频域里，卷积操作相当于两个信号的傅立叶变换的点积，再进行反傅立叶变换。
- **滤波操作**：在频域中，卷积操作可以看作是一种滤波操作。通过傅立叶变换，你可以更容易地理解这种滤波是如何在频域中去除噪声或强调某些特征的。

### Stride & Padding

- Bigger strides produce smaller feature maps

**Stride = 1 (filter = 3 x 3)**

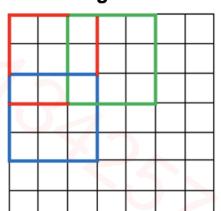
**7 x 7 image**



**5 x 5 feature map**

**Stride = 2 (filter = 3 x 3)**

**7 x 7 image**



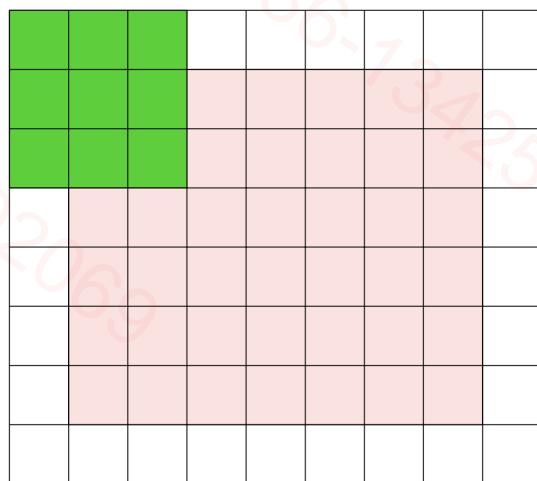
**3 x 3 feature map**

19

## 步长 (Stride)

- 定义**：步长是卷积核在输入图像上移动的距离。例如，步长为 1 意味着卷积核每次移动一个像素。
- 影响**：
  - 输出大小**：较大的步长会导致输出特征图的尺寸减小。
  - 信息捕获**：较小的步长能更细致地捕获信息，但会增加计算复杂性。
- 应用场景**：在对象检测和图像分割等需要精确定位的任务中，通常使用较小的步长。

## Convolution with Zero Padding



Sometimes, we treat the off-edge inputs as zero (or some other value).

16



## 填充 (Padding)

- 定义：填充是在输入图像的边缘添加额外像素的过程，通常用 0 像素进行填充。
- 影响：
  1. 输出大小：填充可以用来控制输出特征图的尺寸。
  2. 边缘信息：通过使用填充，卷积核能够更好地捕获边缘区域的信息。

## Receptive Filed

## Receptive Field

- From the left column, we are hard to tell the receptive field size, especially for deep CNNs.
- The right column shows the fixed-sized CNN visualization, which solves the problem by keeping the size of all feature maps constant and equal to the input map. Each feature is then marked at the center of its receptive field location.

The University of Sydney

Page 38

```
CLASS torch.nn.Conv2d(in_channels, out_channels, kernel_size, stride=1, padding=0, dilation=1, [SOURCE]
groups=1, bias=True, padding_mode='zeros')
```

Applies a 2D convolution over an input signal composed of several input planes.

In the simplest case, the output value of the layer with input size  $(N, C_{\text{in}}, H, W)$  and output  $(N, C_{\text{out}}, H_{\text{out}}, W_{\text{out}})$  can be precisely described as:

$$\text{out}(N_i, C_{\text{out}_j}) = \text{bias}(C_{\text{out}_j}) + \sum_{k=0}^{C_{\text{in}}-1} \text{weight}(C_{\text{out}_j}, k) * \text{input}(N_i, k)$$

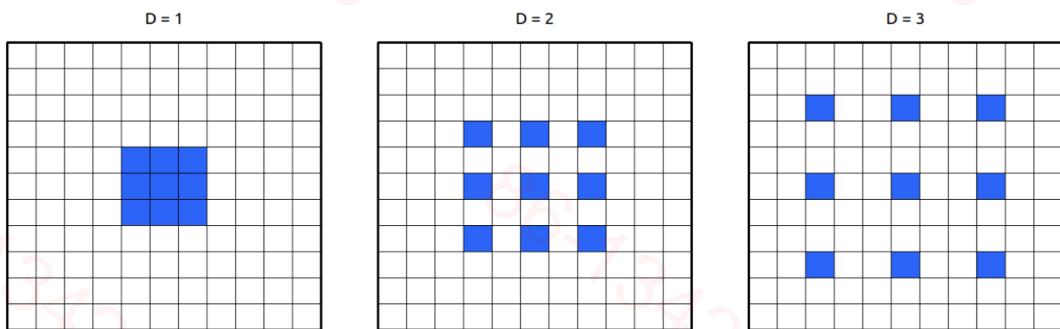
where  $*$  is the valid 2D cross-correlation operator,  $N$  is a batch size,  $C$  denotes a number of channels,  $H$  is a height of input planes in pixels, and  $W$  is width in pixels.

The University of Sydney

Page 39

## Dilated Convolution

- In simple terms, dilated convolution is just a convolution applied to input with defined gaps.
- Dilation: Spacing between kernel elements. Default: 1.
- $D=2$  means skipping one pixel per input
- The receptive field grows exponentially while the number of parameters grows linearly.



(Yu et al, 2015)

在卷积神经网络（CNN）中，感受野是一个非常重要的概念。它指的是输出特征图（或特征映射）上的某个单位对输入图像中的哪个区域具有响应，或者说，输出特征图上的一个单位是由输入图像中的哪个区域的信息生成的。

### 基础概念：

- **局部感受野**：在单个卷积层中，感受野通常与卷积核的大小相同。例如，如果你有一个  $3 \times 3$  的卷积核，那么每个输出像素的感受野在输入图像中就是一个  $3 \times 3$  的区域。
- **累积感受野**：在多层卷积网络中，一个输出像素的感受野是由多个卷积层共同决定的。随着网络层次的增加，感受野会逐渐变大，这意味着网络能够捕获更大范围内的上下文信息。

### 捕获信息：

- **局部到全局**：在网络的前几层，感受野较小，主要捕获局部特征（如边缘和纹理）。在更深的层次，感受野变大，能够捕获更全局、更抽象的特征（如物体的整体形状）。
- **层级特征**：通过不同大小的感受野，网络能够逐层提取从简单到复杂的特征，最终形成一个分层的特征表示。

## Pooling

- The exact location of a feature is not so important; we just want to know where approximately it is and how it relates to other features
- We can use a **pooling** layer (also called sub-sampling layer)
  - “Summarizes” the input from the previous layer
  - Reduces the number of parameters (by 75% in the example below) and helps prevent overfitting
  - Improves robustness to shifts in the receptive field

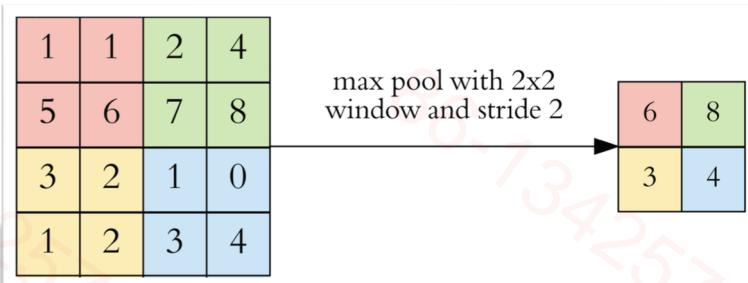


Image ref.: M. Kubat, Introduction to ML, Springer

23

## Pooling

### Average pooling

- Filter size: (2,2)
- Stride: (2,2)
- Pooling ops:  $\text{mean}(\cdot)$



A 4x4 matrix representing a feature map:

-1	4	1	2
0	1	3	-2
1	5	-2	6
3	-1	-2	-2

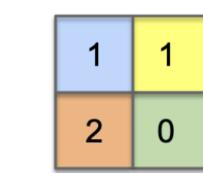
Feature map



A 2x2 matrix resulting from Max pooling:

4	3
5	6

Max pooling



A 2x2 matrix resulting from Average pooling:

1	1
2	0

Average pooling

在卷积神经网络（CNN）中，池化层通常用于减小特征图的尺寸，以减少模型的计算复杂性。同时，池化操作也能增加模型对于小的平移和变换的鲁棒性。

### 常见类型：

1. **最大池化（Max Pooling）**：选择一个局部区域内的最大值作为输出。
2. **平均池化（Average Pooling）**：计算一个局部区域内的平均值作为输出。

### 影响和作用：

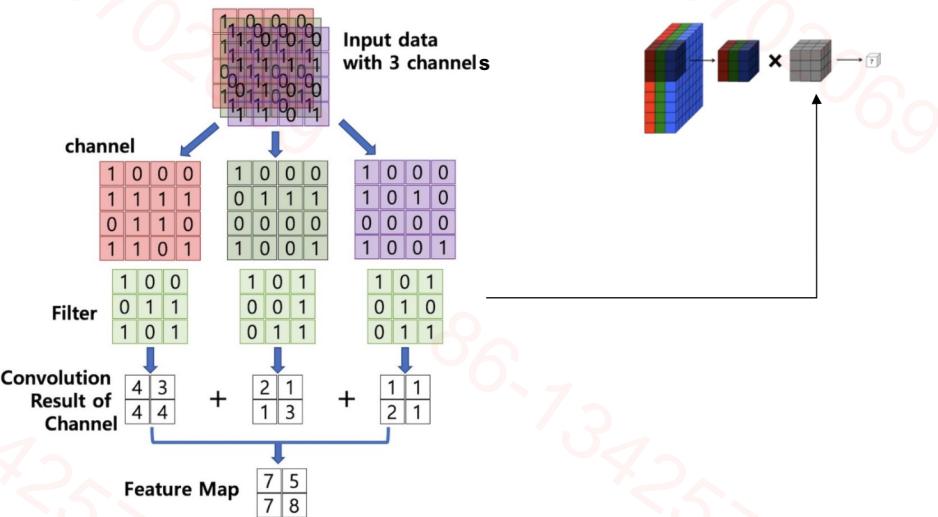
1. **减小尺寸**：池化操作通常会减小特征图的尺寸，从而减少后续层需要处理的数据量。
2. **增加鲁棒性**：池化层能提高模型对小的平移、旋转和尺度变化的鲁棒性。
3. **特征压缩**：通过池化，一些非必要的信息（通常是低层次的、高频的信息）会被丢弃，而更重要的全局信息会被保留。
4. **计算效率**：减少了特征图的尺寸，有助于减少模型的参数和计算量，使模型更易于训练。
5. **信息丢失**：虽然池化有很多优点，但也有一个主要的缺点，即可能导致一些重要信息的丢失。

Max Pooling能够保留高频信息，Average Pooling可以减少对noise的影响。

## Multi-channel CNN



- We can modify the CNN architecture to work with multiple channels

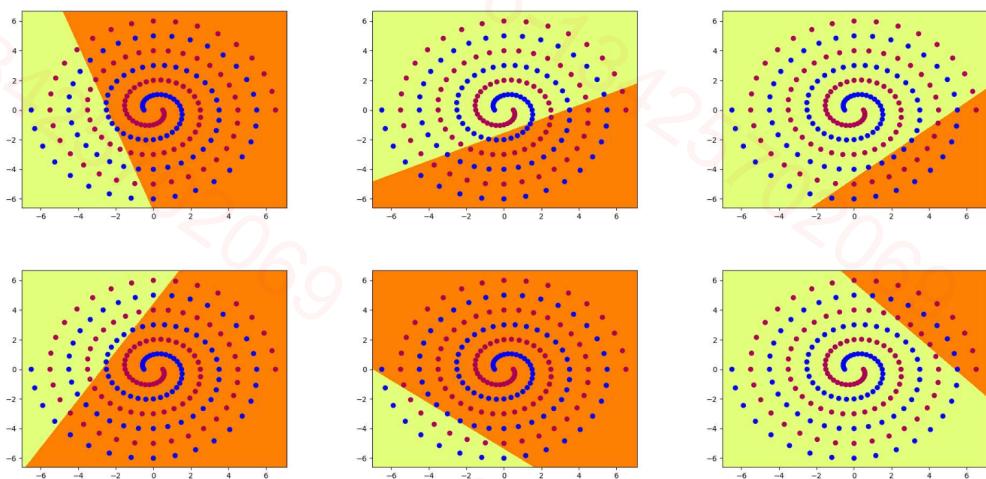


30

## 总结

CNN卷积层的作用是抓取空间信息，将其转化成一个一维的feature vector，然后使用MLP进行分类（当然也可以接各种头，比如分割头，目标检测头等等）。和MLP相似的，CNN随着深度的加深，抓取的特征将会拥有更多的非线性。

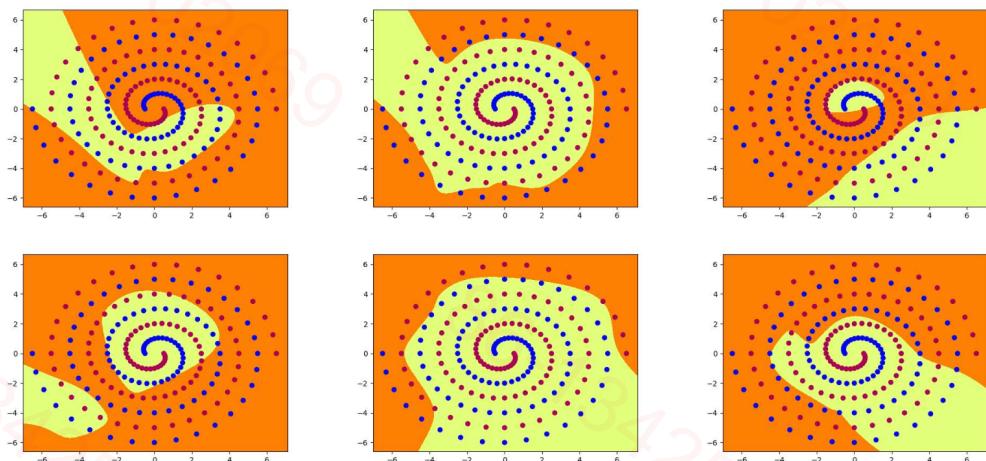
## First Hidden Layer



12



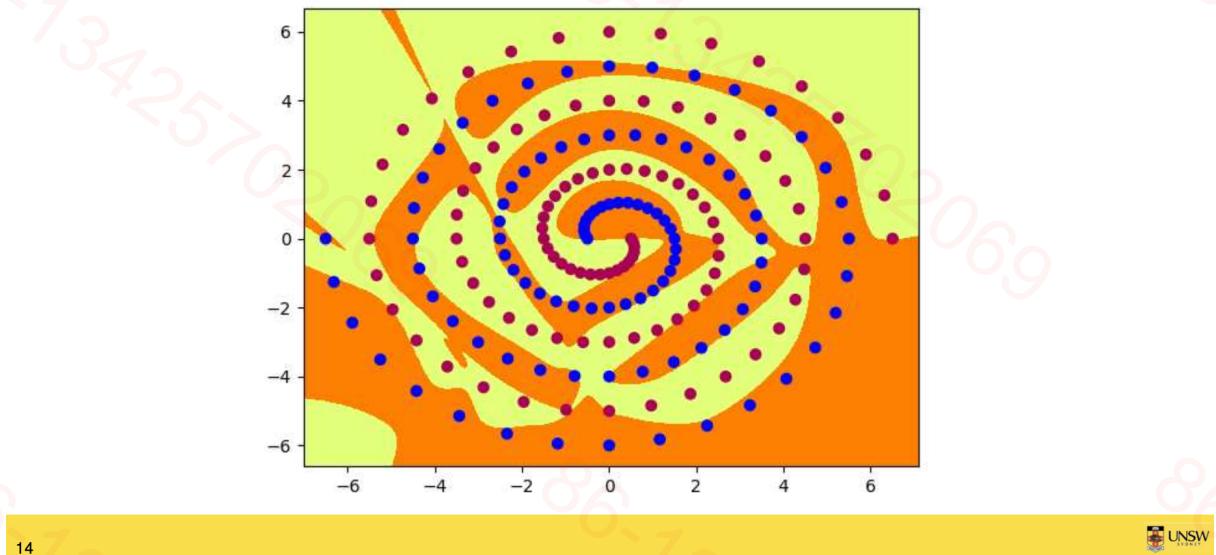
## Second Hidden Layer



13

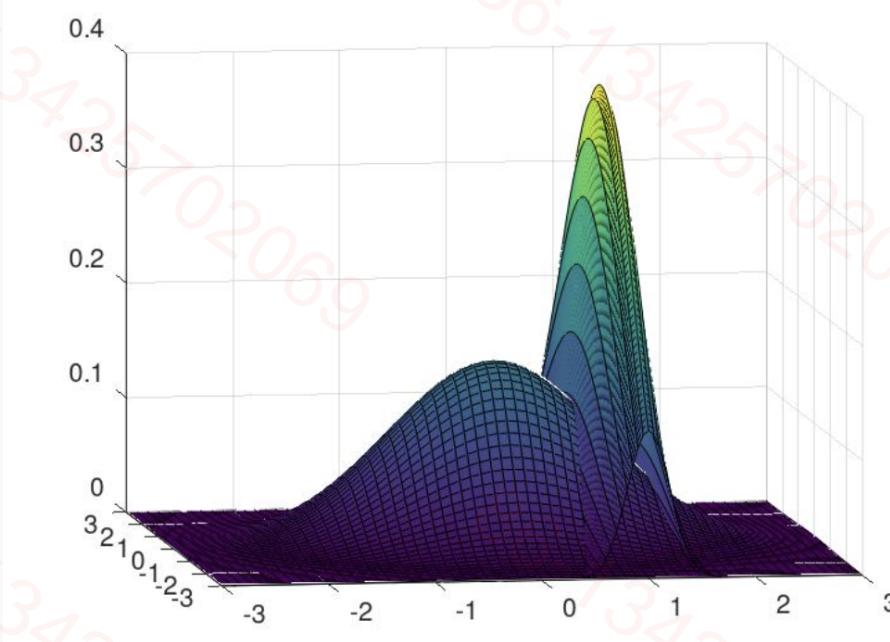


## Network Output



## Exercise

### 3. Entropy, KL-Divergence and $W_2$ Distance for Bivariate Gaussians



Consider two bivariate Gaussian distributions  $p$  and  $q$ .

$$q \text{ has mean } \mu_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \text{ and variance } \Sigma_1 = \begin{bmatrix} 0.04 & 0 \\ 0 & 4 \end{bmatrix}$$

$$p \text{ has mean } \mu_2 = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \text{ and variance } \Sigma_2 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

a. Compute the Entropy  $H(p)$  and  $H(q)$ . Which one is larger? Why?

b. Compute the KL-Divergence in each direction  $D_{KL}(q \parallel p)$  and  $D_{KL}(p \parallel q)$ .  
Which one is larger? Why?

c. Compute the Wasserstein Distance  $W_2(q, p)$

## Entropy

熵是一个衡量集合中的不确定性或混乱程度的度量。在决策树中，给定一组样本和它们的类别，熵就用来衡量这组样本相对于其类别的同质性（纯度）。

### (Shannon) entropy

- Shannon entropy of a random variable  $X$ :

$$H(X) = \sum_{x \in A_x} -p(x) \log_2 p(x)$$

熵的计算公式是：

$$\text{Entropy}(S) = - \sum_{i=1}^n p_i \log_2(p_i)$$

其中

$p_i$  是第  $i$  个类别在样本集  $S$  中出现的概率。

## 高斯分布的熵

### Entropy for Gaussian Distributions

Entropy of Gaussian with mean  $\mu$  and standard deviation  $\sigma$  :

$$\frac{1}{2}(1 + \log(2\pi)) + \log(\sigma)$$

Entropy of a  $d$ -dimensional Gaussian  $p()$  with mean  $\mu$  and variance  $\Sigma$  :

$$H(p) = \frac{d}{2}(1 + \log(2\pi)) + \frac{1}{2} \log |\Sigma|$$

If  $\Sigma = \text{diag}(\sigma_1^2, \dots, \sigma_d^2)$  is diagonal, the entropy is:

$$H(p) = \frac{d}{2}(1 + \log(2\pi)) + \sum_{i=1}^d \log(\sigma_i)$$

21



对于连续型随机变量，特别是服从高斯分布（Gaussian distribution）的随机变量，熵的计算方式不同。对于多元高斯分布（如本例中的二元高斯分布），熵的公式是：

$$H(p) = \frac{1}{2} \ln ((2\pi e)^n |\Sigma|)$$

其中， $n$  是随机变量的维度（在二元高斯分布中， $n = 2$ ）， $\Sigma$  是协方差矩阵， $|\Sigma|$  是其行列式。

For  $q$  with  $\Sigma_1 = \begin{pmatrix} 0.04 & 0 \\ 0 & 4 \end{pmatrix}$ :

$$H(q) = \frac{1}{2} \ln ((2\pi e)^2 |\Sigma_1|)$$

$$H(q) = \frac{1}{2} \ln ((2\pi e)^2 \times 0.04 \times 4)$$

For  $p$  with  $\Sigma_2 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$ :

$$H(p) = \frac{1}{2} \ln ((2\pi e)^2 |\Sigma_2|)$$

$$H(p) = \frac{1}{2} \ln ((2\pi e)^2 \times 1 \times 1)$$

```
import numpy as np

# Constants
two_pi_e_squared = (2 * np.pi * np.e)**2

# Covariance matrices
sigma1 = np.array([[0.04, 0], [0, 4]])
sigma2 = np.array([[1, 0], [0, 1]])

# Determinants
det_sigma1 = np.linalg.det(sigma1)
det_sigma2 = np.linalg.det(sigma2)

# Entropy calculations
H_q = 0.5 * np.log(two_pi_e_squared * det_sigma1)
H_p = 0.5 * np.log(two_pi_e_squared * det_sigma2)

H_q, H_p
```

```
>>> (1.9215863345351902, 2.8378770664093453)
```

## KL-Divergence

## Continuous Entropy and KL-Divergence

→ the *entropy* of a continuous distribution  $p()$  is

$$H(p) = \int_{\theta} p(\theta)(-\log p(\theta)) d\theta$$

→ *KL-Divergence* between two continuous distributions  $p()$  and  $q()$  is

$$D_{KL}(p \parallel q) = \int_{\theta} p(\theta)(\log p(\theta) - \log q(\theta)) d\theta$$

KL散度 (Kullback-Leibler Divergence) 和熵 (Entropy) 都是用于量化信息或不确定性的指标，但它们的应用和解释有所不同。

### 相关性

1. **从熵到KL散度:** 从数学上看，KL散度实际上可以解释为两个分布  $p$  和  $q$  的熵  $H$  的“差异”。具体而言， $D_{KL}(p \parallel q)$  可以看作是  $p$  相对于  $p$  的额外不确定性。这是因为  $D_{KL}(p \parallel q) = H(p, q) - H(p)$ ，其中  $H(p, q)$  是交叉熵 (Cross Entropy)。
2. **相对熵 (Relative Entropy) :** KL散度有时也被称为  $p$  相对于  $q$  的相对熵。这是因为它度量了当用  $q$  来近似  $p$  时，需要多少额外的信息。
3. **熵作为特例:** 当  $p = q$  时， $D_{KL}(p \parallel q) = 0$ ，这意味着当两个分布完全相同时，没有额外的信息或不确定性，所以其KL散度为零。
4. **对称性:** 熵是一个对称的概念，即  $H(p) = H(p)$ ，而KL散度是非对称的，即  $D_{KL}(p \parallel q) \neq D_{KL}(q \parallel p)$ 。

## KL-Divergence for Gaussians

KL-Divergence between Gaussians  $q()$ ,  $p()$  with mean  $\mu_1, \mu_2$  and variance  $\Sigma_1, \Sigma_2$ :

$$D_{KL}(q||p) = \frac{1}{2} \left[ (\mu_2 - \mu_1)^T \Sigma_2^{-1} (\mu_2 - \mu_1) + \text{Trace}(\Sigma_2^{-1} \Sigma_1) + \log \frac{|\Sigma_2|}{|\Sigma_1|} - d \right]$$

In the case where  $\mu_2 = 0, \Sigma_2 = I$ , the KL-Divergence simplifies to:

$$D_{KL}(q||p) = \frac{1}{2} \left[ \|\mu_1\|^2 + \text{Trace}(\Sigma_1) - \log |\Sigma_1| - d \right]$$

If  $\Sigma_1 = \text{diag}(\sigma_1^2, \dots, \sigma_d^2)$  is diagonal, this reduces to:

$$D_{KL}(q||p) = \frac{1}{2} \left[ \|\mu_1\|^2 + \sum_{i=1}^d (\sigma_i^2 - 2 \log(\sigma_i) - 1) \right]$$

22



```
# Means
mu1 = np.array([1, 0])
mu2 = np.array([0, 0])

# Dimension
n = 2

# Compute the inverse of the covariance matrices
inv_sigma1 = np.linalg.inv(sigma1)
inv_sigma2 = np.linalg.inv(sigma2)

# Compute KL divergence DKL(q || p)
DKL_qp = 0.5 * (np.trace(np.dot(inv_sigma2, sigma1)) +
                  np.dot(np.dot((mu1 - mu2).T, inv_sigma2), (mu1 - mu2)) / n +
                  np.log(det_sigma2 / det_sigma1))

# Compute KL divergence DKL(p || q)
DKL_pq = 0.5 * (np.trace(np.dot(inv_sigma1, sigma2)) +
                  np.dot(np.dot((mu2 - mu1).T, inv_sigma1), (mu2 - mu1)) / n +
                  np.log(det_sigma1 / det_sigma2))

DKL_qp, DKL_pq
```

```
>>> (2.436290731874155, 23.208709268125844)
```

计算出的KL散度如下：

- $D_{KL}(q \parallel p) \approx 2.436$
- $D_{KL}(p \parallel q) \approx 23.209$

$D_{KL}(p \parallel q)$ 显然比  $D_{KL}(q \parallel p)$  大得多。

这个差异主要源于两个分布的协方差矩阵和均值向量。具体来说：

$\Sigma_1$  和  $\Sigma_2$  的行列式差异大，特别是因为  $\Sigma_1$  的一个维度的方差非常小（0.04），这使得从  $p$  到  $q$  的“距离”增大。

因此，从  $p$  到  $q$  的KL散度相对更高，这表明  $p$  分布相对于  $q$  分布有更大的不确定性或“扩散”。

## Wasserstein Distance

### Wasserstein Distance

Another commonly used measure is the *Wasserstein Distance* which, for multivariate Gaussians, is given by

$$W_2(q, p)^2 = \|\mu_1 - \mu_2\|^2 + \text{Trace}(\Sigma_1 + \Sigma_2 - 2(\Sigma_1 \Sigma_2)^{\frac{1}{2}})$$

In the case where  $\mu_2 = 0$ ,  $\Sigma_2 = I$ , the KL-Divergence simplifies to:

$$W_2(q, p)^2 = \|\mu_1\|^2 + d + \text{Trace}(\Sigma_1 - 2(\Sigma_1)^{\frac{1}{2}})$$

If  $\Sigma = \text{diag}(\sigma_1^2, \dots, \sigma_d^2)$  is diagonal, this reduces to:

$$W_2(q, p)^2 = \|\mu_1\|^2 + \sum_{i=1}^d (\sigma_i - 1)^2$$

当两个分布  $p$  和  $q$  主要通过平移 (translation) 或者说位置偏移来区分时，Wasserstein距离 (WD) 通常更适用。这是因为Wasserstein距离直观地度量了从一个分布转移到另一个分布所需的“成本”，并且它能够捕捉这种平移或位置偏移的影响。

### 为什么在这种情况下WD更好？

1. **几何直观**: Wasserstein距离具有几何解释，能够直观地反映通过平移能够达到的两个分布之间的相似性。
2. **平滑性**: 相对于均值的小幅度变化，Wasserstein距离通常会有更平滑的变化，这在某些应用中可能是有益的。
3. **支持不重叠问题**: 即使两个分布在空间中有一定距离（即它们的支持不重叠或仅部分重叠），Wasserstein距离仍然能够给出有意义的度量。而KL散度在这种情况下可能会变得无穷大。

相比之下，KL散度更多地关注了两个分布的形状差异而非位置差异。因此，如果两个分布仅通过平移就能够得到，使用Wasserstein距离通常更为合适。

```
from scipy.linalg import sqrtm

# Compute squared Euclidean distance between means
squared_distance_means = np.sum((mu1 - mu2)**2)

# Compute the second term in the formula
sigma_sqrt_q = sqrtm(sigma1)
term2 = sqrtm(np.dot(np.dot(sigma_sqrt_q, sigma2), sigma_sqrt_q))
trace_term = np.trace(sigma1 + sigma2 - 2 * term2)

# Compute squared Wasserstein distance W2^2
W2_squared = squared_distance_means + trace_term

# Compute Wasserstein distance W2
W2 = np.sqrt(W2_squared)

W2
```

```
>>> 1.624807680927192
```

## Weight Initialisation

权重初始化在神经网络的训练过程中起着重要的作用。如果权重初始化不当，网络可能会受到梯度消失或梯度爆炸的影响，从而导致训练过程变得低效或不稳定。下面是几种常用的权重初始化方法及其特点：

### 随机初始化

- **均匀分布 (Uniform Distribution)**：权重从一个均匀分布中随机采样。
- **正态分布 (Normal Distribution)**：权重从一个正态分布中随机采样。

### 零初始化

- 所有权重都初始化为零。这种方法通常不推荐，因为它会导致对称性问题，即所有神经元都会学习到相同的特征。

## 参数初始化 Initialization.

① 初始化的宗旨：快点收敛  
非对称性，所有训练层元 Neuron 的输出相同。  
⇒ 相同的梯度，相同的参数。  
不能太大：梯度爆炸。  
不能太小：梯度消失。  
Mean 应该为 0，每层的 variance 应该一样。

### 2) Xavier Initialization.

① Assumption: 激活函数为 tanh。  
并且在训练前固定 linear stage。  
 $\text{所以 } \text{tanh}(z^e) \approx z^e \dots \text{①}$   
 $z^e = w^e a^e + b^e \rightarrow a^e = \text{tanh}(z^e)$   
固定  $a^e$  和  $b^e$ :  $a^e = \text{tanh}(z^e) \approx z^e$   
所以  $\text{Var}(a^e) = \text{Var}(z^e)$   
 $= \text{Var}(\sum_{j=1}^n w_j^e a_j^e + b^e) \dots \text{②}$   
 $\text{将 variance 式子代入化简: } w_j^e \rightarrow g_j^e \rightarrow \text{short term}$   
因为 assume  $w_j^e$  是 0，所以是常数项。  
 $\Rightarrow \text{E}(x^2) = 0, \text{Var}(x) = 1$   
 $\Rightarrow \text{Var}(w_j^e a_j^e) = \text{Var}(w_j^e) \text{Var}(a_j^e) \dots \text{③}$   
 $\Rightarrow \text{Var}(a^e) = \sum_{j=1}^n \text{Var}(w_j^e a_j^e), n^e \cdot 0 = n^e \text{Var}(w_j^e) \text{Var}(a_j^e)$   
 $\Rightarrow \text{Var}(w^e) = \frac{1}{n^e} \text{backward } \text{Var}(w^e) = \frac{1}{n^e}$

② 初始值公式:  
Let  $n = n^e + n^b$   
Xavier normal:  $N(0, 2/n)$   
Xavier uniform:  $U(-\sqrt{6}/n, \sqrt{6}/n)$

### 3) He initialization.

① 推导:  $\text{Var}(y) = \text{Var}(\text{ReLU}(z^e w^e))$   
 $= n \text{Var}(w^e) \text{E}[z^e]^2$  by  $\text{Var}(X) = \text{E}[X^2] - \text{E}[X]^2$   
因为 ReLU:  $x^e = \max(0, z^e)$   
 $\Rightarrow \text{E}(x^e) = \frac{1}{2} \text{Var}(z^e)$   
 $\Rightarrow \text{Var}(y) = n \text{Var}(w^e) \frac{1}{2} \text{Var}(z^e)$   
 $\Rightarrow \text{Var}(w^e) = 2/n$ .

### ② 公式:

He normal:  $N(0, 1/n)$   
He uniform:  $U(-\sqrt{n}/n, \sqrt{n}/n)$

## 正则化 Regularization.

① 正则化，为了避免过拟合。  
② 影响因素: ① 参数过多 ② 权重过大。  
Dropout: Lp 正则化。

### ③ 罚约束 vs 软约束。

a) Hard L2 constraint:  
 $\min_{\theta} \sum_{i=1}^n \| \theta_i \|^2 \text{ s.t. } \|\theta\|_2^2 \leq r$ .  
b) Soft L2 constraint: trading parameters.  
 $\min_{\theta} \sum_{i=1}^n \| \theta_i \|^2 + \lambda \|\theta\|^2$

### ④ Bayesian Regularization.

Bayesian rule:  $P(\theta|x) = \frac{P(x|\theta)P(\theta)}{P(x)}$   
Maximum A Posteriori (MAP): 带着 prior  
 $\max_{\theta} \log(p(x|\theta)) + \log(p(\theta))$   
MLE loss regularization.

### ⑤ Weight Decay.

Notation: 模型里所有参数的集合。  
 $\min_{\theta} \sum_{i=1}^n \|\theta_i\|^2 + \frac{\lambda}{2} \|\theta\|^2$  Engineers view:  
 $\nabla_{\theta}^2(\theta) + \frac{\lambda}{2} \|\theta\|^2$  表示梯度  $\theta$  的  
rate of parameter

### ⑥ 数据集加入 Noise 相当于加入正则化。

$y_{\text{true}} = \text{E}[y] + \text{E}[y - \text{E}[y]]$   
 $L(y) = \text{E}[y - (\text{E}[y] + \text{E}[y - \text{E}[y]])]^2 = \text{E}[y - (\text{E}[y] + \text{E}[y - \text{E}[y]])]^2 + 2\text{E}[y - \text{E}[y]]^2 \text{E}[y - \text{E}[y]]^2$   
 $= \text{E}[y - (\text{E}[y] - \text{E}[y])]^2 + 2\|\text{E}[y]\|^2$

### ⑦ Dropout.

Motivation: 希望只用少部分的连接来更新。

关键: 训练阶段 P% 的神经元被遮住。  
测试阶段 使用所有的神经元。Weight scale down P%.

a) Inserted Dropout { At training: weight = 1/p weight.  
At testing: use the whole network.  
相当于训练了 p 个网络，测试时平均了所有的结果。

⑧ 优势: ① scale-free 对大规模对 feature 不会有影响。  
② Invariant to parameter scaling

⑨ DropConnect. 相当于随机丢弃神经元，随机丢弃连接。  
→ 更丰富的网络组合。

### ⑩ Batch Normalization.

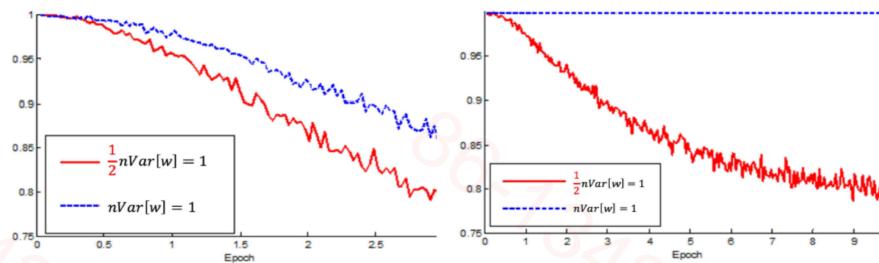
减少 feature unit 间带来的影响。更大的学习率。

① Motivation: 减少 internal variance shift，加速收敛。缓解 gradient vanishing.

② Group Normalizations. BN 在小 batch size 上表现不好。  
将 channel 分成 group 之后批归一化，对 batch size 不敏感。

Reference: COMP329 Deep Learning. Dr. Chang Xu  
University of Sydney. 悉尼大学.

## Weight Initialization



- 22-layer ReLU network (left),  
 $\text{Var}[w] = \frac{2}{n}$  converges faster than  $\text{Var}[w] = \frac{1}{n}$
- 30-layer ReLU network (right),  
 $\text{Var}[w] = \frac{2}{n}$  is successful while  $\text{Var}[w] = \frac{1}{n}$  fails to learn at all

## Weight Initialization

In order to have healthy forward and backward propagation, each term in the product must be approximately equal to 1. Any deviation from this could cause the activations to either vanish or saturate, and the differentials to either decay or explode exponentially.

$$\text{Var}[z] \simeq \left( \prod_{i=1}^D G_0 n_i^{\text{in}} \text{Var}[w^{(i)}] \right) \text{Var}[x]$$
$$\text{Var}\left[\frac{\partial}{\partial x}\right] \simeq \left( \prod_{i=1}^D G_1 n_i^{\text{out}} \text{Var}[w^{(i)}] \right) \text{Var}\left[\frac{\partial}{\partial z}\right]$$

We therefore choose the initial weights  $\{w_{jk}^{(i)}\}$  in each layer ( $i$ ) such that

$$G_1 n_i^{\text{out}} \text{Var}[w^{(i)}] = 1$$



Slide Credit: Alan Blair

14

## Initialization (Xavier) -- [Xavier Glorot, Yoshua Bengio 2010]

$$\text{Var}(a^{[L]}) = \left[ \prod_{l=1}^L n^{[l-1]} \text{Var}(W^{[l]}) \right] \text{Var}(x)$$

$$n^{[l-1]} \text{Var}(W^{[l]}) \begin{cases} < 1 & \rightarrow \text{Vanishing Signal} \\ = 1 & \rightarrow \text{Var}(a^{[L]}) = \text{Var}(x) \\ > 1 & \rightarrow \text{Exploding Signal} \end{cases}$$

$$\text{Var}(W^{[l]}) = \frac{1}{n^{[l-1]}}$$

[https://pytorch.org/docs/stable/nn.init.html#torch.nn.init.xavier\\_normal\\_](https://pytorch.org/docs/stable/nn.init.html#torch.nn.init.xavier_normal_)

The University of Sydney

Page 43

## Initialization (Xavier) -- [Xavier Glorot, Yoshua.Bengio 2010]

We worked on activations computed during the forward propagation, and get

$$Var(W^{[l]}) = \frac{1}{n^{[l-1]}}$$

The same result can be derived for the backpropagated gradients:

$$Var(W^{[l]}) = \frac{1}{n^{[l]}}$$

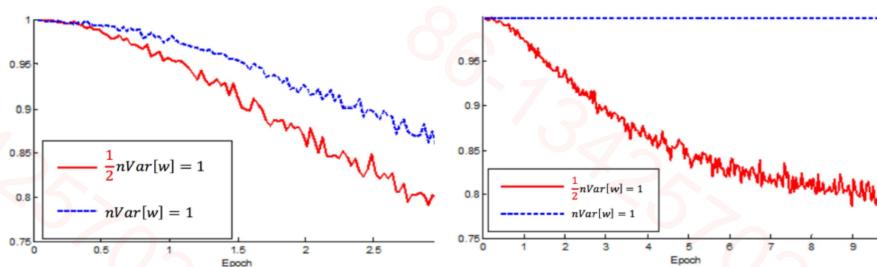
Xavier initialization would either initialize the weights as

xavier\_normal  $\mathcal{N}(0, \frac{2}{n^{[l-1]} + n^{[l]}})$

xavier\_uniform  $\mathcal{U}(-\frac{\sqrt{6}}{\sqrt{n^{[l-1]}+n^{[l]}}}, \frac{\sqrt{6}}{\sqrt{n^{[l-1]}+n^{[l]}}})$

Page 44

### Weight Initialization



- 22-layer ReLU network (left),  
 $Var[w] = \frac{2}{n}$  converges faster than  $Var[w] = \frac{1}{n}$
- 30-layer ReLU network (right),  
 $Var[w] = \frac{2}{n}$  is successful while  $Var[w] = \frac{1}{n}$  fails to learn at all

## Batch Normalisation

## Batch Normalization

We can normalize the activations  $x_k^{(i)}$  of node  $k$  in layer  $(i)$  relative to the mean and variance of those activations, calculated over a mini-batch of training items:

$$\hat{x}_k^{(i)} = \frac{x_k^{(i)} - \text{Mean}[x_k^{(i)}]}{\sqrt{\text{Var}[x_k^{(i)}]}}$$

These activations can then be shifted and re-scaled to

$$y_k^{(i)} = \beta_k^{(i)} + \gamma_k^{(i)} \hat{x}_k^{(i)}$$

$\beta_k^{(i)}, \gamma_k^{(i)}$  are additional parameters, for each node, which are trained by backpropagation along with the other parameters (weights) in the network.

After training is complete,  $\text{Mean}[x_k^{(i)}]$  and  $\text{Var}[x_k^{(i)}]$  are either pre-computed on the entire training set, or updated using running averages.



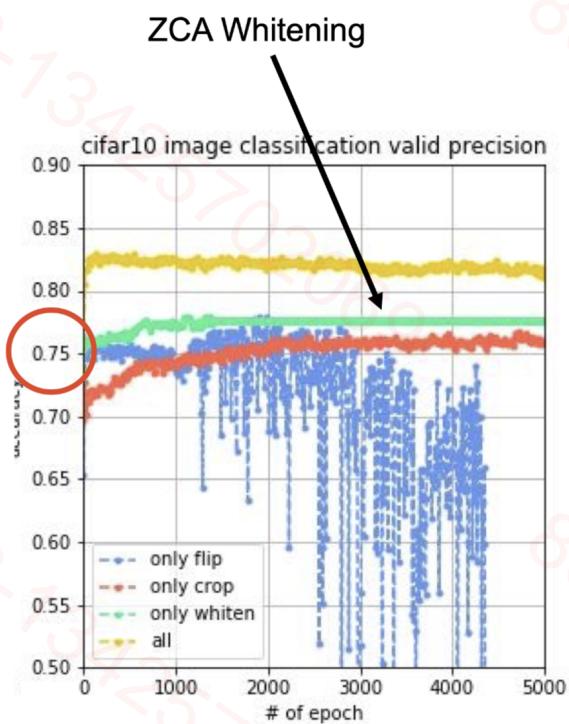
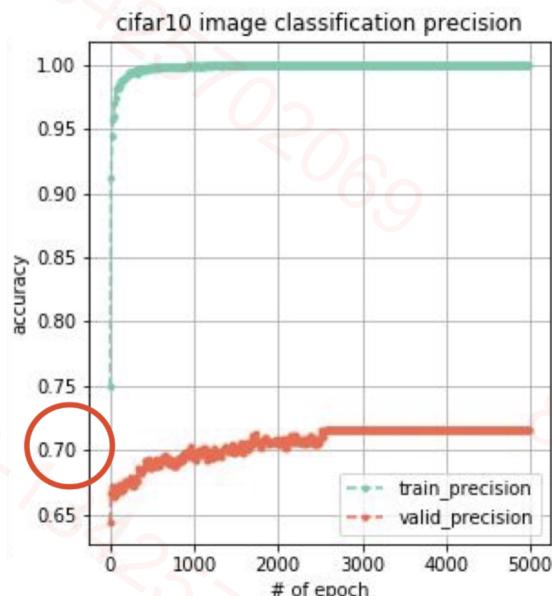
Slide Credit: Alan Blair

16

## Feature scaling

- In machine learning algorithms, the functions involved in the optimization process are sensitive to normalization
  - For example: Distance between two points by the Euclidean distance. If one of the features has a broad range of values, the distance will be governed by this particular feature.
  - After normalization, each feature contributes approximately proportionately to the final distance.
- In general, gradient descent converges much faster with feature scaling than without it.
- Good practice for numerical stability for numerical calculations, and to avoid ill-conditioning when solving systems of equations.

## Validation Performance

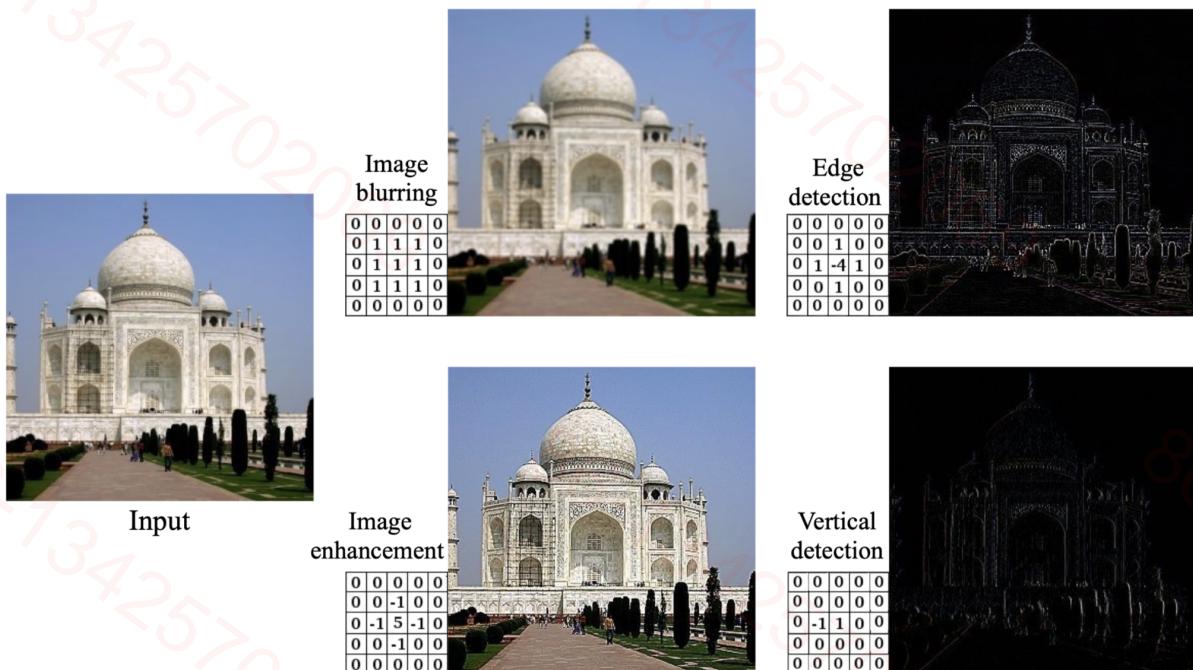


## CNN模块结构

我们复习一下一个CNN的组成部分

## Motivation: convolution

- Multiple filters: intuitive examples



The University of Sydney

Page 54

## 卷积运算

### Convolution Operator (9.1-9.2)

Continuous convolution

$$s(t) = (x * w)(t) = \int x(a)w(t-a)da$$

Discrete convolution

$$s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a)$$

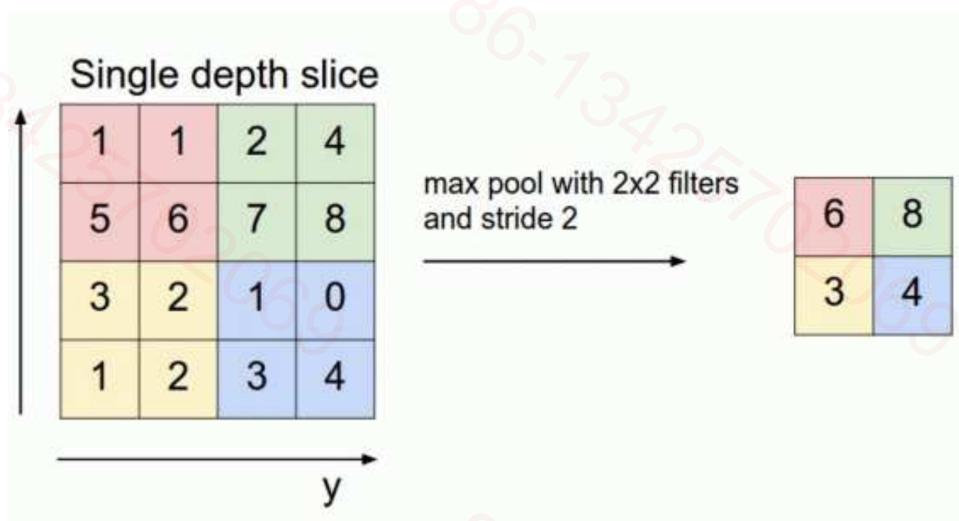
Two-dimensional convolution

$$S(j, k) = (K * I)(j, k) = \sum_m \sum_n K(m, n)I(j + m, k + n)$$

Note: Theoreticians sometimes write  $I(j - m, k - n)$  so that the operator is commutative. But, computationally, it is easier to write it with a plus sign.

## 池化层

## Max Pooling (9.3-9.4)

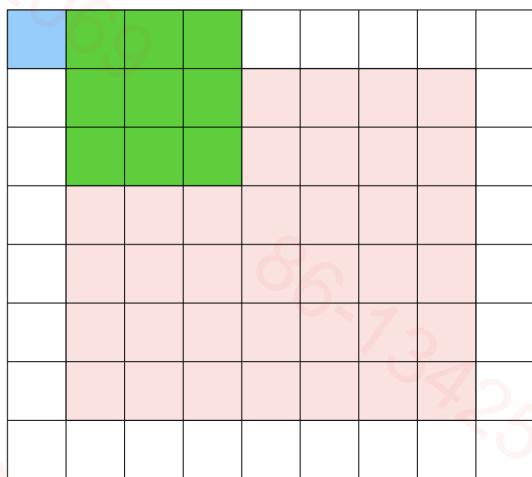


14



## Stride & Pooling

### Convolution with Zero Padding



This is known as "Zero-Padding".

17



## 卷积神经网络 CNNs.

① Convolutional Neural Network. by Yann.

② 组成部分 卷积层，池化层，全连接层。  
操作特征 局部感受野 分类层。

③ 卷积层，Convolution layers

$$g: f \begin{bmatrix} 0 & 2 & 0 \\ 2 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \xrightarrow{\text{filter}} \begin{bmatrix} 0 & 0 & -1 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \xrightarrow{\text{输出}} -1$$

卷积计算:  $\left. \begin{array}{l} (1+2+0+0-1) = 1 \\ +(2-1+0+1) = -2 \\ +(1-0+0+0) = 0 \end{array} \right\} 1-2+0=-1$

rules:  $f_{ij} = \sum_k f_{ik} g_{kj}$

Stride 步长，卷积核扫描步长。

Padding 填充，将进像张高宽0补齐。

Output size. 图片尺寸是卷积核的大小。

$$\text{size} \leftarrow \frac{n+2p-k}{s} + 1$$

④ Receptive Field. 感受野，一个单位(神经元)对空间的感知范围。  
随着网络的增加，一个像素包含了前面层更多的信息。

$g: \text{Input} \rightarrow \text{Output}$ . 后一个像素影响前面的7个像素。

⑤ Dilated Convolution 扩大步长，在卷积核上引入扩张率。

$$g: \begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix} \xrightarrow{D=2} \begin{smallmatrix} 0 & 2 & 0 \\ 2 & 1 & 0 \\ 1 & 0 & 0 \end{smallmatrix} \rightarrow \text{在 kernel 中插入空洞。}$$

因此能更快地增加感受野，但灵敏度会降低。

⑥ Pooling 池化层，压缩特征，干移不变性。  
种类，平均池化，最大池化，归一化。

## 网络结构 NN Architectures.

① 卷积神经网络 总结

阶段	进化发展	论文	更多 filters	层级
LeNet	简单卷积层	LeNet		2 layer
MINST	更复杂的卷积层	ZFNet		3 layer
2012	更深的卷积层	GoogleNet	101层	22 layer
2013	更深的卷积层	ResNet		152层

④ More Architectures.

① DenseNet. 深度监督 Deep supervision.  
每层和之前所有的层直接连接，特征重用，增加效率。  
优点：缓解梯度消失，更好的捕捉特征，特征重用，减少参数。

② SqueezeNet. 把 AlexNet  $\rightarrow$  只保留数值是效果一样。  
模块：Squeeze + Expand. 压缩层和双3x3 kernel.

③ MobileNet. 特征分离也很 Depthwise Separable Convolution.  
标准复杂度:  $D_x \cdot D_y \cdot M \cdot N \cdot D_f \cdot D_g$   
DSC 复杂度:  $D_x \cdot D_y \cdot M \cdot D_f \cdot D_g + M \cdot N \cdot D_f \cdot D_g$

④ ShuffleNet. 分组卷积 Partition Group convolution.

在输入的通道进行分组，组上进行卷积。

$g: 4 \times 4 \times 64 \rightarrow \text{Standard } 4 \times 4 \times 64, \text{ PGC } 2 \times (2+2) \times 2 \times 4 \times 8$   
 $\hookrightarrow$  造成问题，分组信息不独立，利用 shuffle，随机排列，交换位置。

## 循环神经网络 RNNs.

① Recurrent Neural Network. (文本情感分析)

① Input: 独热编码 One-hot Encoding. 二进制编码  
输出和时间距离一样，但是维度太高。替代: Word embedding  
Output: 概率分布 Probability distribution  
 $\hookrightarrow$  通过 BERT 生成 2

② 关键：通过 Memory 读取标注序列。

③ 问题：对于需要储存过去的时间序列的信息。  
Input 的长度将会很大，导致梯度消失/爆炸。  
 $g: \text{如果序列过长, } 0.9^{100} \approx 2000$   
 $\text{gradient} = \text{softmax}^T \cdot \text{softmax} \rightarrow \text{ReLU}(0,00)$

Reference: COMP5329 Deep Learning. Dr Chang Xu  
University of Sydney. 香港大学.

## Deeper Network — ResNet

ResNet (残差网络) 是由 Kaiming He 等人在 2015 年提出的一种深度神经网络架构。其主要贡献是引入了“残差块” (Residual Blocks)，有效地解决了深度网络中的梯度消失和梯度爆炸问题。

### 残差块

残差块的主要思想是在网络中添加“短路连接” (Shortcut Connections)，也就是跳过一到多个层。数学上，这可以表示为：

$$\text{Output} = F(\text{Input}) + \text{Input}$$

其中  $F(\text{Input})$  是一系列卷积层、激活函数等组成的子网络的输出。

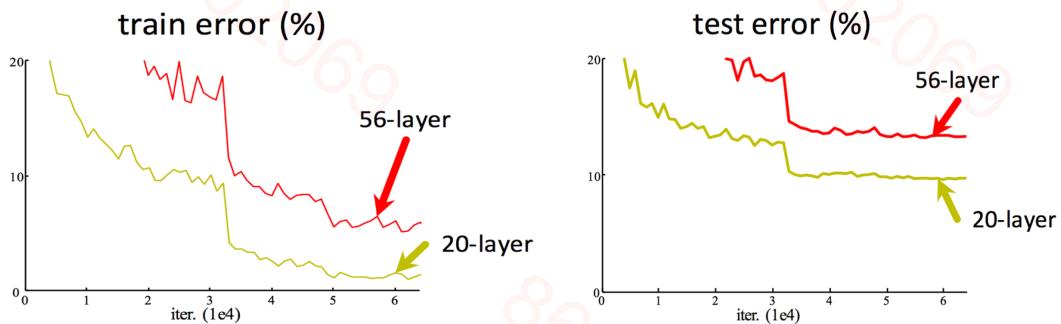
这种设计让网络有能力学习恒等映射 (Identity Mapping)，这在训练非常深的网络时是非常有用的。具体来说，如果一个或多个层没有提供任何有用的转换，那么网络可以学习关闭这些层，只使用短路连接。

### 优点

1. 允许更深的网络：通过使用残差块，ResNet 可以训练几百甚至几千层的网络，而不会遇到梯度消失或梯度爆炸问题。

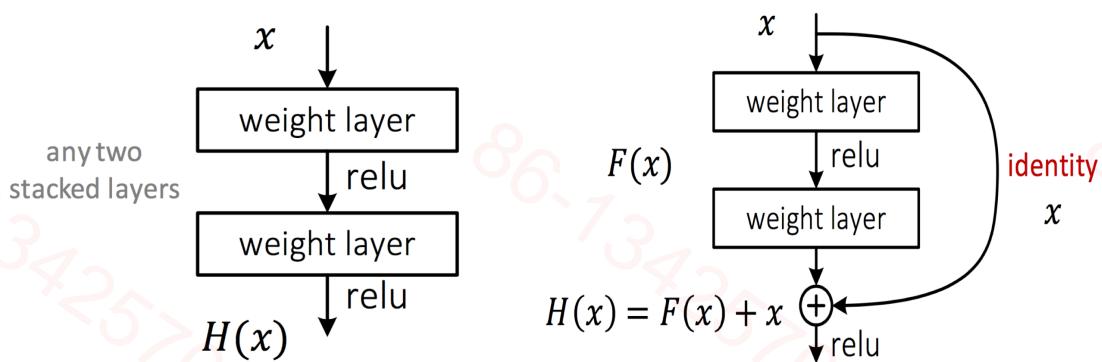
- 参数效率高：尽管网络很深，但由于残差结构的存在，ResNet 通常比其他深度网络更加参数高效。
- 泛用性强：ResNet 在图像分类、目标检测和许多其他任务上都表现得非常好。

## Going Deeper



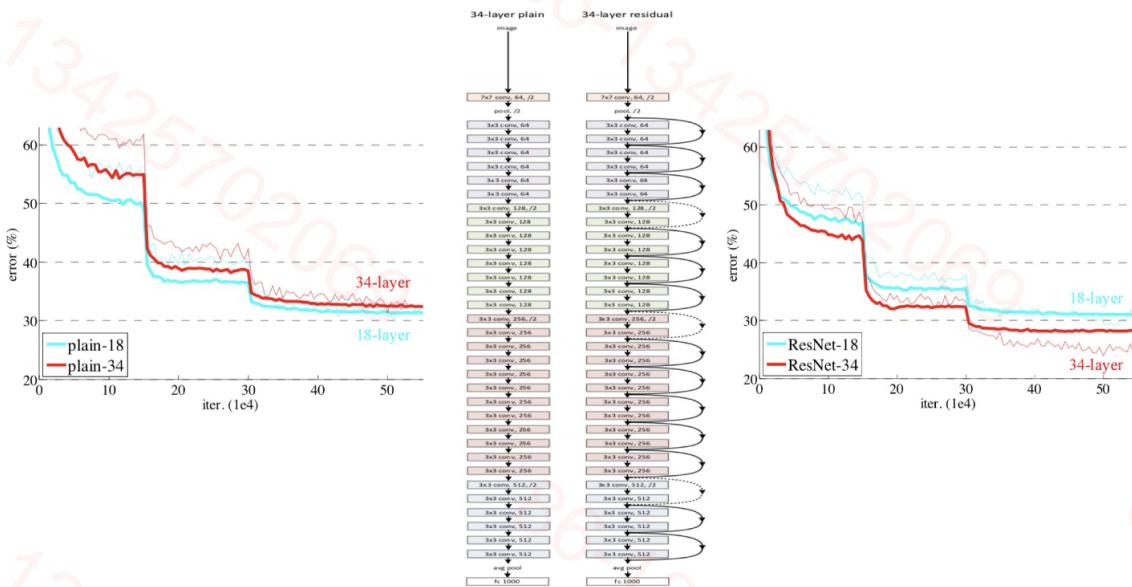
➤ If we simply stack additional layers, it can lead to higher training error as well as higher test error

## Residual Networks



➤ Idea: Take any two consecutive stacked layers in a deep network and add a “skip” connection which bypasses these layers and is added to their output.

## □ ResNet [He et al., 2015]



## Smaller Kernel

VGG Net (也称为 VGG 网络) 是由牛津大学的 Visual Geometry Group 提出的，是一种用于图像识别任务的深度卷积神经网络。该网络在 2014 年 ImageNet 竞赛中获得了很高的排名，并因其简单、易于理解的架构而广受好评。

### 架构特点

- 统一的卷积核大小**：VGG 使用了尺寸为  $(3 \times 3)$  的卷积核，并且在所有卷积层中都使用这种大小的卷积核。
- 多个卷积层堆叠**：VGG 的一个显著特点是多个卷积层的连续堆叠，而不是使用较大的卷积核或更复杂的架构。
- 增加网络深度**：VGG 有多个版本，包括 VGG16 (16 层) 和 VGG19 (19 层)。这些网络通过增加卷积层的数量来增加网络的深度。
- 全连接层**：VGG 网络通常包含几个全连接层，用于最后的分类任务。
- ReLU 激活函数**：在所有的卷积层和全连接层后都使用了 ReLU (Rectified Linear Unit) 激活函数。

### 优点和局限性

- 优点**：结构简单、易于理解，很容易进行修改和扩展。
- 局限性**：由于网络很深并且包含许多全连接层，因此需要大量的计算资源和内存。这使得 VGG 在移动设备或资源有限的环境中不太适用。

VGG

- Developed at Visual Geometry Group (Oxford) by Simonyan and Zisserman
  - 1<sup>st</sup> runner up (Classification) and Winner (localization) of ILSVRC 2014 competition
  - VGG-16 comprises of 138 million parameters
  - VGG-19 comprises of 144 million parameters



**Image Credit:** Medium. <https://medium.com/coinmonks/paper-review-of-vggnet-1st-runner-up-of-ilsvrc-2014-image-classification-d02355543a11>

21

# Inception Module

GoogleNet 是由 Google 的研究团队在 2014 年提出的一种深度卷积神经网络架构。该网络在 ImageNet Large Scale Visual Recognition Challenge (ILSVRC) 2014 中取得了显著成绩。GoogleNet 最引人注目的特点是其 Inception 模块，该模块有效地增加了网络的宽度和深度，同时还维持了计算效率。

## Inception 模块

Inception 模块的核心思想是同时进行多种大小和类型的卷积操作，然后将结果拼接在一起。一个典型的 Inception 模块可能包括：

- $1 \times 1$  的卷积
  - $3 \times 3$  的卷积
  - $5 \times 5$  的卷积
  - $3 \times 3$  的最大池化

这些操作通常还会与 $1 \times 1$  的卷积（用于减少维度）和 ReLU 激活函数一起使用。

## 架构特点

- 高效的计算**：通过精心设计的 Inception 模块，GoogleNet 能够在减少参数数量的同时，保持或提高模型性能。
  - 去除全连接层**：GoogleNet 基本上没有使用全连接层，这也有助于减少模型的参数数量。
  - 辅助分类器**：为了解决梯度消失问题，GoogleNet 在网络的中间层添加了辅助的分类器进行中间预测。

4. 深度和宽度：GoogleNet 架构具有很高的深度（通常有 22 层或更多），并且通过 Inception 模块增加了网络的宽度。

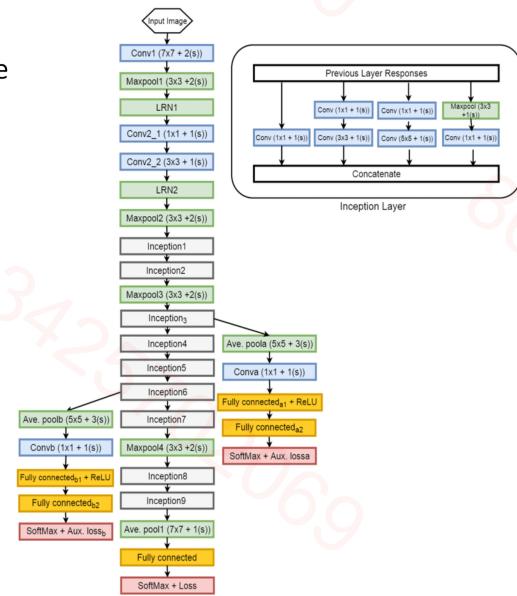
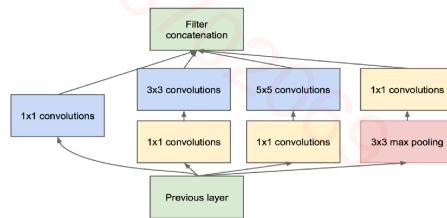
## 优点和局限性

1. 优点：高性能、计算效率、参数效率。

2. 局限性：虽然 Inception 模块可以自动学习多尺度特征，但模型架构相对复杂，需要更多的调优。

## GoogLeNet

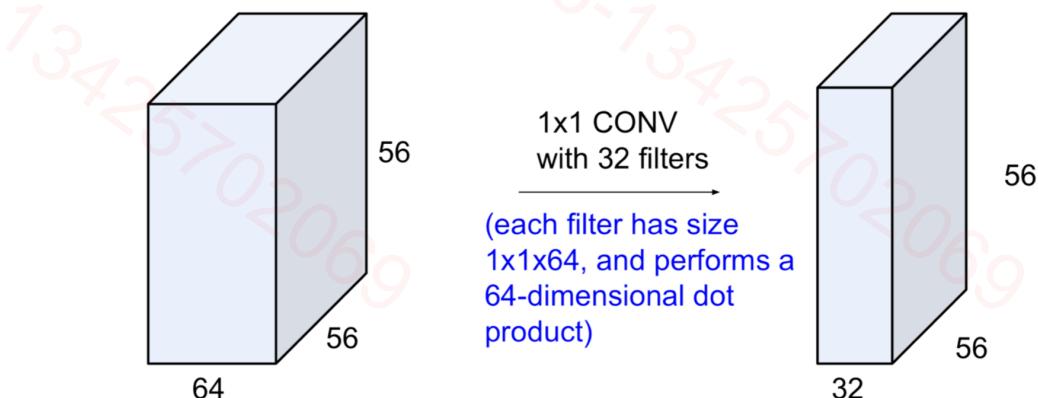
- A 22-layer CNN developed by researchers at Google
- Deeper networks prone to overfitting and suffer from exploding or vanishing gradient problem
- Core idea “Inception module”
- Adding Auxiliary loss as an extra supervision
- Winner of 2014 ILSVRC Challenge



Source: Convolutional Neural Networks. <https://medium.com/@rajar.k.91/convolutional-neural-networks-why-what-and-how-f8f6dbbb2f9>

22

## □ GoogLeNet [Szegedy et al., 2014]

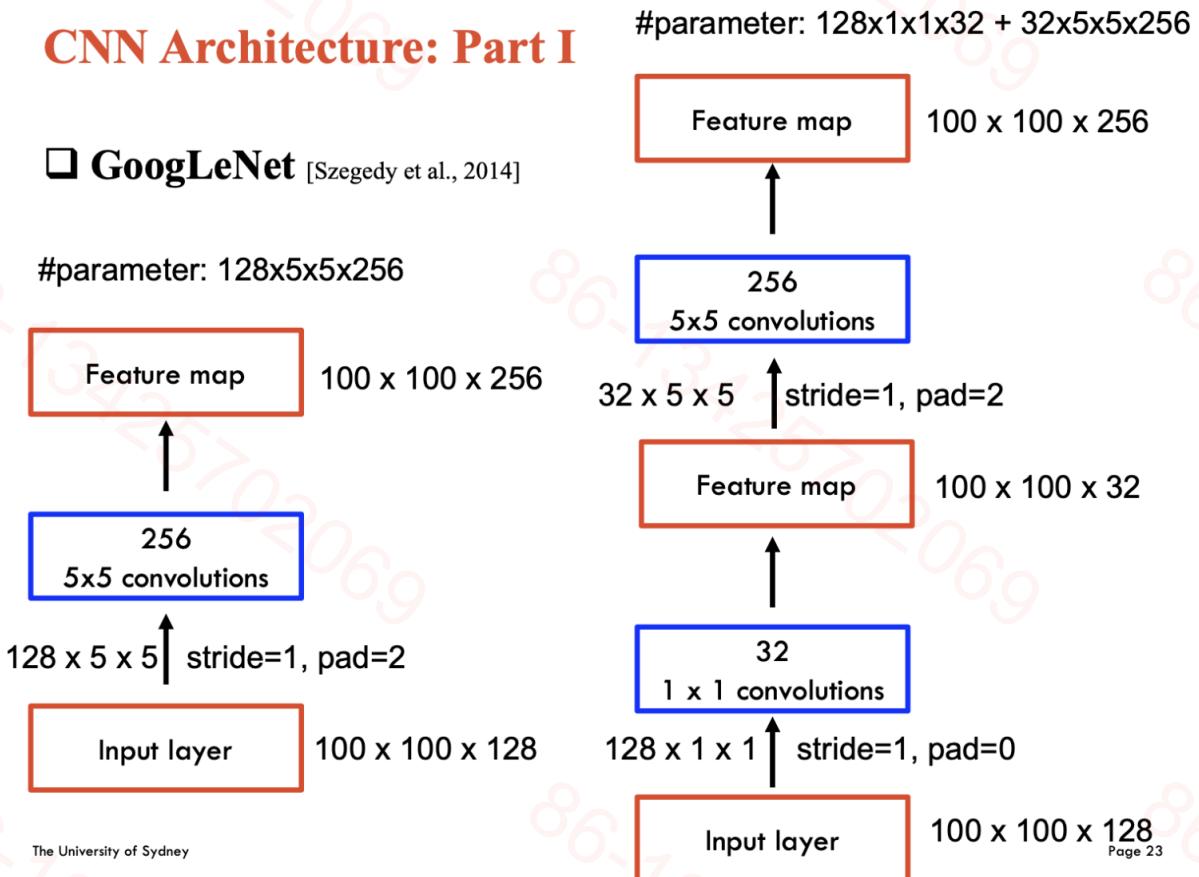


1x1 convolutional layer

- preserve spatial dimensions (56x56)
- reduces depth (64 -> 32)

## CNN Architecture: Part I

### □ GoogLeNet [Szegedy et al., 2014]



要达到最后一样的 $100 \times 100 \times 256$ 的feature map，如果使用 $5 \times 5$ 的卷积核需要的参数要比两个 $1 \times 1$ 和 $3 \times 3$ 的参数量多。后者还能提供更深的网络结构和更加精准的细节。

## SENet

SENet (Squeeze-and-Excitation Network) 是一种在 2017 年 ImageNet 竞赛中获得冠军的卷积神经网络架构。该网络由 Jie Hu、Li Shen 和 Samuel Albanie 等人提出。SENet 的主要创新在于引入了一个名为 "Squeeze-and-Excitation" 的模块，该模块可以显著提高网络对通道间依赖关系的建模能力。

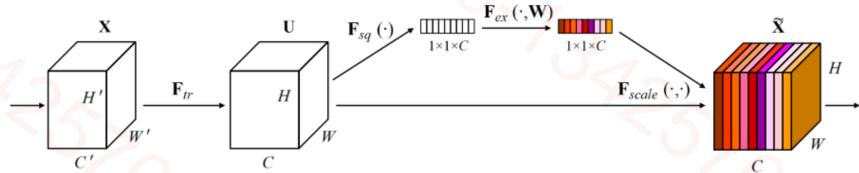
### Squeeze-and-Excitation 模块

这个模块分为两个主要步骤：

1. **Squeeze**：全局平均池化（Global Average Pooling）用于压缩每个通道，从而产生一个通道描述符。简单来说，这一步从空间维度中提取出每个通道的全局信息。
2. **Excitation**：通过一个全连接（全连接）层，对这些通道描述符进行重新加权。然后，这些加权因子用于重新调整原始特征图。

### SENet (Squeeze-and-Excitation Network)

- CNNs fuse the spatial and channel information to extract features to solve the task
- Before this, networks weights each of its channels equally when creating the output feature maps
- SENets added a content aware mechanism to weight each channel adaptively
- SE block helps to improve representation power of the network, able to better map the channel dependency along with access to global information



Source: Convolutional Neural Networks. <https://medium.com/@raijat.k.91/convolutional-neural-networks-why-what-and-how-f8f6dbbb2f9>

25

## Data Augmentation

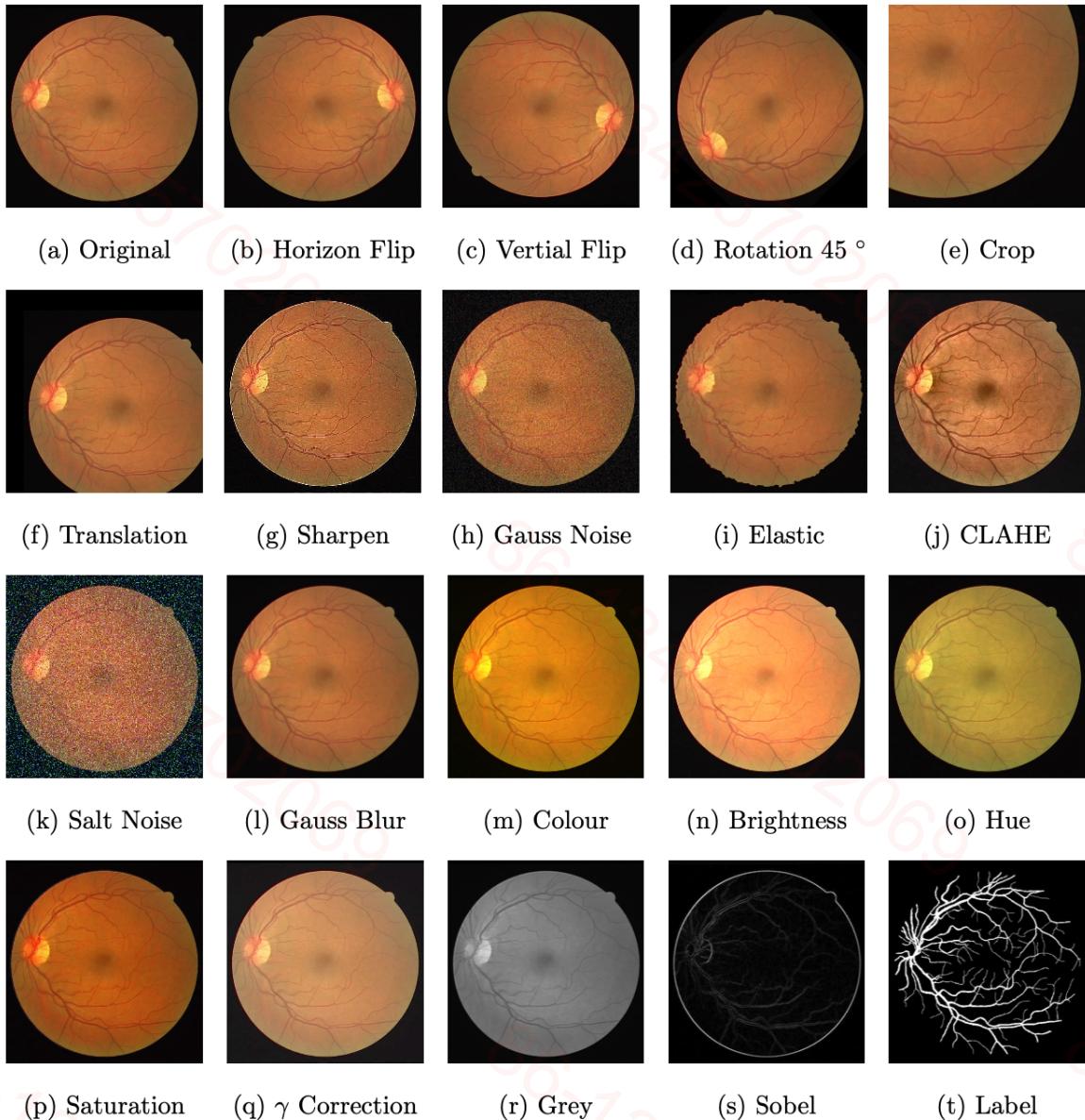


Figure 3.7: Visual examples of data augmentation methods for the retina image from the DRIVE dataset. Where horizon and vertical file, rotation, crop and translation belong to spatial transformation, the last image is the ground truth label for better visual comparison among the methods. And remaining augmentation methods are pixel-wise transformations. This figure serves as a particular visual illustration of the transformed retina images.

## Regularisation

### Weight Decay

## Regularization: Weight Decay

- It adds a penalty term to the loss function on the training set to reduce the complexity of the learned model
- Popular choice for weight decay:
  - L1: The L1 penalty aims to minimize the absolute value of the weights

$$L(x, y) \equiv \sum_{i=1}^n (y_i - h_\theta(x_i))^2 + \lambda \sum_{i=1}^n |\theta_i|$$

- L2: The L2 penalty aims to minimize the squared magnitude of the weights

$$L(x, y) \equiv \sum_{i=1}^n (y_i - h_\theta(x_i))^2 + \lambda \sum_{i=1}^n \theta_i^2$$



Credit: 5 Techniques to Prevent Overfitting in Neural Networks. <https://www.kdnuggets.com/2019/12/5-techniques-prevent-overfitting-neural-networks.html>

47

权重衰减（Weight Decay）是一种常用的正则化技术，用于防止神经网络模型过拟合。这种方法通过在模型的损失函数中添加一个与权重相关的额外项来实现。

### 数学描述

在没有权重衰减的情况下，一个常见的损失函数（如均方误差）可以表示为：

$$\text{Loss} = \text{MSE}(y, \hat{y})$$

引入权重衰减后，损失函数变为：

$$\text{Loss} = \text{MSE}(y, \hat{y}) + \lambda \sum_i w_i^2$$

其中， $\lambda$  是权重衰减系数，一个超参数； $w_i$  是模型权重； $\sum_i w_i^2$  是所有权重的平方和。

### 工作原理

1. **惩罚大权重**：通过添加权重平方和作为正则项，模型被鼓励使用较小的权重。这有助于模型泛化，因为大权重可能会导致模型在训练数据上过拟合。
2. **超参数调整**：权重衰减系数 $\lambda$  需要通过交叉验证等方法来选择。

### 优点和局限性

1. **优点**：简单、有效，易于实现，几乎不增加计算成本。
2. **局限性**：可能不适用于所有类型的问题或数据分布。

## Regularization: Weight Decay

- It adds a penalty term to the loss function on the training set to reduce the complexity of the learned model
- Popular choice for weight decay:
  - L1: The L1 penalty aims to minimize the absolute value of the weights

$$L(x, y) \equiv \sum_{i=1}^n (y_i - h_\theta(x_i))^2 + \lambda \sum_{i=1}^n |\theta_i|$$

- L2: The L2 penalty aims to minimize the squared magnitude of the weights

$$L(x, y) \equiv \sum_{i=1}^n (y_i - h_\theta(x_i))^2 + \lambda \sum_{i=1}^n \theta_i^2$$



Credit: 5 Techniques to Prevent Overfitting in Neural Networks. <https://www.kdnuggets.com/2019/12/5-techniques-prevent-overfitting-neural-networks.html>

47

## Dropout

Dropout 是一种用于防止神经网络过拟合的正则化技术。这种方法由 Geoffrey Hinton 和他的学生在 2014 年提出。Dropout 在训练过程中通过随机“关闭”一部分神经元来实现正则化。

### 工作原理

在每次训练迭代中，Dropout 会以预定的概率  $p$  随机地将一层神经元的输出设置为零。这样做的效果相当于从原始网络中随机删除了一些连接。

### 优点和局限性

#### 1. 优点：

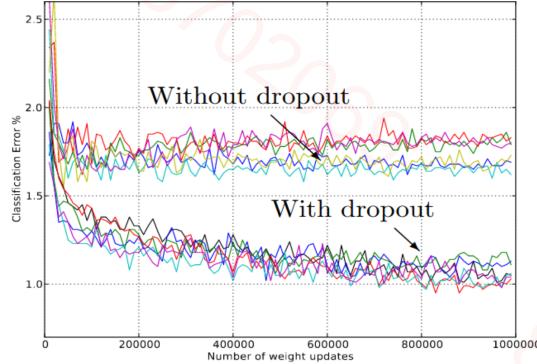
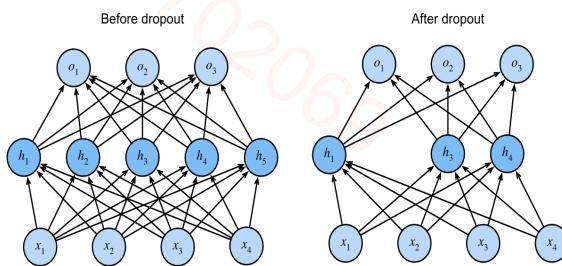
- 简单但有效：容易实现，而且通常能显著提高模型的泛化能力。
- 训练更快：由于每次迭代只使用网络的一部分，因此训练通常会更快。

#### 2. 局限性：

- 在推理（inference）阶段不应使用：Dropout 仅用于训练，不用于模型最终的预测。
- 随机性可能导致不稳定：由于 Dropout 引入了随机性，可能需要更多的训练迭代来收敛。

## Regularization: Dropout

- L1 and L2 reduce overfitting by modifying the cost function
- Dropout modify the network by randomly dropping neurons from the neural network during training
- Dropout is an efficient way to average many large neural networks



Credit: Srivastava et al., Dropout: A Simple Way to Prevent Neural Networks from Overfitting. JMLR, 2014  
[https://colab.research.google.com/github/d2l-ai/d2l-en-colab/blob/master/chapter\\_multilayer-perceptrons/dropout.ipynb](https://colab.research.google.com/github/d2l-ai/d2l-en-colab/blob/master/chapter_multilayer-perceptrons/dropout.ipynb)

48

## Language Statistics Model

我们在训练一个NLP模型的时候，并不仅是希望训练一个语言的映射关系，比如apple的相似词语，pear, banna等等。

### Statistical Language Processing

Synonyms for “elegant”

*stylish, graceful, tasteful, discerning, refined, sophisticated, dignified, cultivated, distinguished, classic, smart, fashionable, modish, decorous, beautiful, artistic, aesthetic, lovely; charming, polished, suave, urbane, cultured, dashing, debonair; luxurious, sumptuous, opulent, grand, plush, high-class, exquisite*

Synonyms, antonyms and taxonomy require human effort, may be incomplete and require discrete choices. Nuances are lost. Words like “king”, “queen” can be similar in some attributes but opposite in others.

Could we instead extract some statistical properties automatically, without human involvement?

4



如果只是希望理解同义词，这样的任务是相当简单的。并不需要一个语言模型来进行训练，甚至可以用一个model-free的方法来实现。

## 思考：如何实现？

方法其实非常简单，只需要用一些统计值或者概率值就可以，比如相似性（similarity），相关度（correlation），共聚性（co-occurrence）等等，都能实现简单的单词理解和替换。然而在不同的上下文中进行理解是一个困难的任务。

## Counting Frequencies

### Counting Frequencies

word	frequency
a	7
all	1
and	2
bought	1
cat	1
caught	1
crooked	7
found	1
he	1
house	1
in	1
little	1
lived	1
man	1
mile	1
mouse	1
sixpence	1
stile	1
there	1
they	1
together	1
upon	1
walked	1
was	1
who	2

- some words occur frequently in all (or most) documents
- some words occur frequently in a particular document, but not generally
- this information can be useful for document classification

我们可以通过数不同word在一个document中的信息，然后取出现最多的一些文字我们就可以初步来判断则整篇文章是讲述关于什么内容的。

## Document Classification

word	doc 1	doc 2	doc X
a	.	.	7
all	.	.	1
and	.	.	2
bought	.	.	1
cat	.	.	1
caught	.	.	1
crooked	.	.	7
found	.	.	1
he	.	.	1
house	.	.	1
in	.	.	1
little	.	.	1
lived	.	.	1
man	.	.	1
mile	.	.	1
mouse	.	.	1
sixpence	.	.	1
stile	.	.	1
there	.	.	1
they	.	.	1
together	.	.	1
upon	.	.	1
walked	.	.	1
was	.	.	1
who	.	.	2

7



- each column of the matrix becomes a vector representing the corresponding document
- words like “cat”, “mouse”, “house” tend to occur in children’s books or rhymes
- other groups of words may be characteristic of legal documents, political news, sporting results, etc.
- words occurring many times in one document may skew the vector – might be better to just have a “1” or “0” indicating whether the word occurs at all

## Applications

有个研究领域叫做Topic Modelling。主题模型是自然语言处理（NLP）和机器学习中的一种技术，用于从大量文档中自动识别主题或主题分布。这种方法能够帮助我们在无需人工标注的情况下，快速理解文本集合的主要内容和结构。其中一些有用的应用场景是Information retrieval，比如Turnitin查重等等。

## Next-word Prediction

通过统计模型，我们不光可以快速理解整篇文章的内容，我们还可以进行下一个单词的预测。

### 思考：如何实现？

原理也非常简单，只需要统计每个单词下一个出现单词的频率就可以。

## Counting Consecutive Word Pairs

word	a	all	and	bought	cat	caught	crooked	found	he	house	in	little	lived	man	mile	mouse	sixpence	stile	there	they	together	upon	walked	was	who
a																									
all																									
and																									
bought																									
cat																									
caught																									
crooked																									
found																									
he																									
house																									
in																									
little																									
lived																									
man																									
mile																									
mouse																									
sixpence																									
stile																									
there																									
they																									
together																									
upon																									
walked																									
was																									
who																									

9



所以，这样我们可以将一个统计值拓展成一个统计模型，被称作Predictive 1-Gram Word Model。

## Predictive 1-Gram Word Model

word	a	all	and	bought	cat	caught	crooked	found	he	house	in	little	lived	man	mile	mouse	sixpence	stile	there	they	together	upon	walked	was	who
a																									
all																									
and																									
bought																									
cat																									
caught																									
crooked																									
found																									
he																									
house																									
in																									
little																									
lived																									
man																									
mile																									
mouse																									
sixpence																									
stile																									
there																									
they																									
together																									
upon																									
walked																									
was																									
who																									

10



Predictive 1-Gram Word Model 是自然语言处理（NLP）中最简单的预测模型之一，它用于预测下一个词的出现。在这个模型中，每个词的出现只取决于它本身的概率，而与前文或后文没有关系。

## Model

这种模型基于词  $w$  在训练语料库中出现的频率来计算其概率  $P(w)$ 。

$$P(w) = \frac{\text{Occurrence of } w}{\text{Total Number of Words}}$$

## Analysis

优点：

1. **简单快速**：由于模型只考虑单个词的频率，因此计算速度非常快。
2. **易于实现**：只需统计词频即可。

缺点：

1. **信息损失**：忽略了词与词之间的关系，因此模型非常简单，无法捕获复杂的语义或语法信息。
2. **预测能力有限**：由于模型简单，其预测能力通常远低于更复杂的模型。

## Example

假设我们有一个句子“我喜欢吃”，在 Predictive 1-Gram Word Model 中，下一个词可能是“苹果”、“香蕉”、“鱼”等，模型会根据这些词在训练集中的频率来进行预测。但因为它不考虑前文，所以下一个词同样可能是“跑步”或“游泳”，即使这些词在语境中并不合适。

## Summary

由于其简单性和局限性，这种模型通常仅用作基线模型或在计算资源非常有限的情况下使用。总体而言，Predictive 1-Gram Word Model 是一个非常基础的模型，主要用于简单的词预测任务，而在更复杂的应用场景下，通常会使用更高级的模型，如 n-gram、RNN 或 Transformer 等。

# N-Gram Model

## N-Gram Model

- by normalizing each row (to sum to 1) we can estimate the probability  $\text{prob}(w_j|w_i)$  of word  $w_j$  occurring after  $w_i$
- need to aggregate over a large corpus, so that unusual words like “crooked” will not dominate
- the model captures some common combinations like “there was”, “man who”, “and found”, “he bought”, “who caught”, “and they”, “they all”, “lived together”, etc.
- this **unigram** model can be generalized to a bi-gram, tri-gram, . . . ,  $n$ -gram model by considering the  $n$  preceding words
- if the vocabulary is large, we need some tricks to avoid exponential use of memory

N-Gram 模型是一种用于自然语言处理和文本挖掘的概率模型。在这个模型中，一个 "n-gram" 是由  $n$  个连续的词或字符组成的序列。N-Gram 模型的基本思想是，给定一个词序列，下一个词的出现概率只依赖于这个序列中的最后  $n - 1$  个词。

## 数学模型

对于一个词  $w_i$  和其前面的  $n - 1$  个词（称为上下文） $w_{i-n+1}^{i-1}$ ，N-Gram 模型用于计算这个词的条件概率：

$$P(w_i | w_{i-n+1}^{i-1}) = \frac{\text{Count}(w_{i-n+1}^i)}{\text{Count}(w_{i-n+1}^{i-1})}$$

这里， $\text{Count}(w_{i-n+1}^i)$  是  $n$  个词  $w_{i-n+1}^i$  在语料库中出现的次数， $\text{Count}(w_{i-n+1}^{i-1})$  是  $n - 1$  个词  $w_{i-n+1}^{i-1}$  在语料库中出现的次数。

## Analysis

优点：

1. 简单易于实现：N-Gram 模型基于词频统计，实现起来相对简单。
2. 效率高：预测速度快，尤其适用于实时应用。

缺点：

1. 数据稀疏问题：当  $n$  很大时，可能遇到训练数据中没有出现过的  $n$ -gram，导致概率为零。
2. 不能捕捉长距离依赖：只能捕捉到最近  $n - 1$  个词的局部信息。

## Example

假设你正在编写一个文本生成器，你希望生成与“我喜欢吃”相关的句子。在一个 3-gram ( $n = 3$ ) 模型中，你可能会找到如“我喜欢吃\_苹果\_”、“我喜欢吃\_香蕉\_”等句子，因为这些句子中的“苹果”和“香蕉”是在上下文“我喜欢吃”之后出现频率较高的词。

## Summary

N-Gram 模型广泛应用于各种 NLP 任务，如文本生成、分词、词性标注、机器翻译等。综合来看，N-Gram 是一种相对简单但功能强大的模型，适用于各种文本分析和生成任务。然而，由于其无法捕捉长距离依赖和可能存在的数据稀疏问题，因此在复杂应用场景下，通常会与其他更高级的模型（如 RNN、LSTM、Transformer 等）结合使用。

## Co-occurrence Matrix

## Co-occurrence Matrix

- sometimes, we don't necessarily predict the next , but simply a "nearby word" (e.g. a word occurring within an  $n$ -word window centered on that word)
- we can build a matrix in which each row represents a word, and each column a nearby word
- each row of this matrix could be considered as a vector representation for the corresponding word, but the number of dimensions is equal to the size of the vocabulary, which could be very large ( $\sim 10^5$ )
  - is there a way to reduce the dimensionality while still preserving the relationships between words?

13



共现矩阵是一种用于捕获词与词之间关系的数据结构。在这个矩阵中，行和列分别代表语料库中的唯一词，而每个单元格  $(i, j)$  包含词  $i$  和词  $j$  在某个特定上下文（通常是一个窗口大小内）中同时出现的次数。

### Model

假设有一个词汇表  $V$ ，其大小为  $|V|$ ，共现矩阵  $C$  是一个  $|V| \times |V|$  的矩阵，其中  $C[i][j]$  表示词  $i$  和词  $j$  在给定窗口大小内共同出现的次数。

### Co-occurrence Matrix (2-word window)

word	a	all	and	bought	cat	caught.	crooked	found	he	house	in	little	lived	man	mile	mouse	sixpence	stile	there	they	together	upon	walked	was	who
a				1	1	6	1				1	1	1	1	1	1	1		1	1	1	1	1		
all																									
and																									
bought																									
cat																									
caught																									
crooked																									
found																									
he																									
house																									
in																									
little																									
lived																									
man																									
mile																									
mouse																									
sixpence																									
stile																									
there																									
they																									
together																									
upon																									
walked																									
was																									
who																									

14



## Co-occurrence Matrix (10-word window)

word	a	all	and	bought	cat	caught	crooked	found	he	house	in	little	lived	man	mile	mouse	sixpence	stile	there	they	together	upon	walked	was	who
a	10	2	3	2	2	13	3	2	1	1	1	1	1	2	2	1	2	2	1	2	1	1	2	1	4
all	2	1																							
and	3	1																							
bought	2																								
cat	2																								
caught	2																								
crooked	13	1	3	2	2	2	10	2	2	1	1	1	2	2	2	1	2	3	1	1	1	2	2	1	4
found	3	1																							
he	2																								
house	1																								
in	1	1	1																						
little	1	1																							
lived	1	1	1																						
man	2																								
mile	2	1																							
mouse	1	1	1																						
sixpence	2	1																							
stile	2		1																						
there	1																								
they	2	1	1																						
together	1	1	1																						
upon	2	1	1																						
walked	2	1																							
was	1																								
who	4	2	1	1	1	4	1																		

15



## Analysis

优点：

- 简单有效：共现矩阵易于构建，且能有效捕获词义和词关系。
- 不需要复杂的训练：与基于神经网络的模型相比，构建共现矩阵不需要迭代训练。

缺点：

- 高维性：词汇表越大，矩阵维度越高，这可能导致存储和计算问题。
- 稀疏性：在大多数情况下，大量的矩阵元素会是零，这导致了数据稀疏问题。

## Example

假设有一句话：“我喜欢吃苹果和香蕉。”如果我们取窗口大小为1（即考虑前后各一个词），那么“喜欢”和“吃”、“吃”和“苹果”、“苹果”和“和”等词对会在共现矩阵中的对应单元格里加一。

- by aggregating over many documents, pairs (or groups) of words emerge which tend to occur near each other (but not necessarily consecutively)
  - “cat”, “caught”, “mouse”
  - “walked”, “mile”
  - “little”, “house”
- common words tend to dominate the matrix
  - could we sample common words less often, in order to reveal the relationships of less common words?

## Summary

- 词义相似度和关联性分析：通过共现矩阵，可以计算词之间的相似度或关联性。

2. 文本聚类和分类：共现矩阵可以作为特征用于文本聚类和分类。

3. 信息检索：在搜索引擎中，共现矩阵可以帮助改进查询的相关性。

总体来说，共现矩阵是一种简单但强大的工具，用于捕捉词与词之间的关系和相似度。然而，由于其高维和稀疏的特点，通常需要额外的降维技术（如PCA）或者与其他模型（如TF-IDF、Word2Vec等）结合使用。

## Extension

### Singular Value Decomposition

Co-occurrence matrix  $X_{(L \times M)}$  can be decomposed as  $X = U S V^T$  where  $U_{(L \times L)}$ ,  $V_{(M \times M)}$  are unitary (all columns have unit length) and  $S_{(L \times M)}$  is diagonal, with diagonal entries  $s_1 \geq s_2 \geq \dots \geq s_M \geq 0$

$$\begin{matrix} & M \\ L & \boxed{\phantom{X}} \end{matrix} \approx \begin{matrix} & N \\ L & \boxed{\begin{matrix} \vdash u_1 \vdash \\ \vdash u_2 \vdash \\ \vdash u_k \vdash \end{matrix}} \end{matrix} \begin{matrix} & N \\ & \boxed{\begin{matrix} s_1 & s_2 \\ & \vdash s_N \vdash \end{matrix}} \end{matrix} \begin{matrix} & M \\ & \boxed{\begin{matrix} \mid & \mid \\ \mid & V_1 V_2 \\ \mid & \mid \end{matrix}} \\ N \end{matrix} \tilde{V}^T$$

We can obtain an approximation for  $X$  of rank  $N < M$  by truncating  $U$  to  $\tilde{U}_{(L \times N)}$ ,  $S$  to  $\tilde{S}_{(N \times N)}$  and  $V$  to  $\tilde{V}_{(N \times M)}$ . The  $k$ th row of  $\tilde{U}$  then provides an  $N$ -dimensional vector representing the  $k^{\text{th}}$  word in the vocabulary.

### Eigenvalue vs. Singular Value Decomposition

Eigenvalue Decomposition:

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} = \Omega D \Omega^{-1}, \quad \text{where } \Omega = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}, \quad D = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$
$$\begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} = \Omega D \Omega^{-1}, \quad \text{where } \Omega = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ -i & i \end{bmatrix}, \quad D = \begin{bmatrix} i & 0 \\ 0 & -i \end{bmatrix}$$

Singular Value Decomposition:

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} = USV^T, \quad U = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \quad S = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad V = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$
$$\begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} = USV^T, \quad U = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \quad S = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad V = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

## Eigenvalue vs. Singular Value Decomposition

- if  $X$  is symmetric and positive semi-definite, eigenvalue and singular value decompositions are the same.
- in general, eigenvalues can be negative or even complex, but singular values are always real and non-negative.
- even if  $X$  is a square matrix, singular value decomposition treats the source and target as two entirely different spaces.
- the word co-occurrence matrix is symmetric but not positive semi-definite; for example, if the text consisted entirely of two alternating letters ..ABABABABABAB.. then A would be the context for B, and vice-versa.

23



## Language Semantic Model

然而语言是具包含语义信息的，语义信息代表关于词、短语或句子内在意义的信息。在自然语言处理中，语义分析试图理解文本的意义，这包括但不限于词义消歧（确定多义词的正确意义）、关系抽取（识别实体之间的关系）以及情感分析（判断文本的情感倾向）等。

比如在不同的语境下，一个单词的含义是不一样的。例如，"Apple"这个词在"我吃了一个Apple"和"Apple公司发布了新产品"这两个句子中有完全不同的意义。第一个"Apple"是一种水果，而第二个"Apple"是一个公司。语义分析的目标之一就是能够准确地识别这种情况。

总体来说，语义信息是理解和生成自然语言的关键组成部分，我们希望通过深度学习技术，比如BERT、GPT这样的预训练模型，不光光能够捕捉语言表层的信息，比如近义词替换，更多的是捕捉语言背后的语义信息和复杂情感。比如不同上下文中，一句话的含义。或者一句话在不同语境下表现的情感是否是积极或者消极。所以我们相比之前的统计学模型，我们希望能够有一个模型，能够捕捉latent semantic feature of the language。那么什么是feature呢？其实feature就是一个张量，或者说，是经过维度压缩之后包含最多的信息的一串embedding。

## Word Embeddings

# Word Embeddings

*"Words that are used and occur in the same contexts tend to purport similar meanings."*

Z. Harris (1954)

*"You shall know a word by the company it keeps."*

J.R. Firth (1957)

Aim of Word Embeddings:

*Find a vector representation of each word, such that words with nearby representations are likely to occur in similar contexts.*

17



词嵌入是自然语言处理（NLP）中用于表示词汇的一种技术。它将每个词映射到一个固定大小的向量，这些向量能够捕捉词的语义信息和语境关系。与传统的词袋模型或共现矩阵相比，词嵌入更能有效地捕获词与词之间的相似性和关联性。

## Major Algorithms

1. **Word2Vec**: 由 Google 发表，主要包括 Skip-gram 和 CBOW (Continuous Bag of Words) 两种模型。
2. **GloVe (Global Vectors for Word Representation)** : 由斯坦福大学提出，基于全局词频统计。
3. **FastText**: 是 Facebook 发表的，能够更好地处理罕见词和词根信息。
4. **BERT、GPT**: 基于 Transformer 架构的预训练模型，不仅提供词嵌入还能捕捉更丰富的上下文信息。

## Analysis

优点：

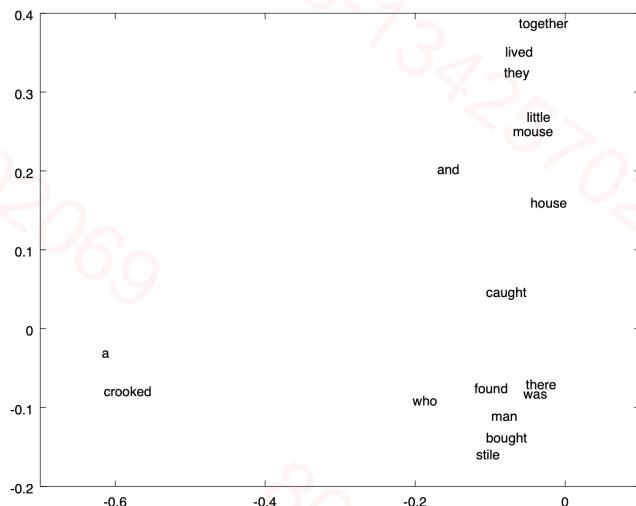
1. **捕捉丰富的语义信息**: 词嵌入能够捕捉同义词、反义词、上下文关系等。
2. **低维性**: 与共现矩阵等高维表示相比，词嵌入通常具有更低的维度（如 50, 100, 300 等）。

缺点：

1. **需要大量数据**: 获得高质量的词嵌入通常需要大量的标注数据。
2. **解释性差**: 词嵌入是通过神经网络学习得到的，通常难以解释。

## Examples

## Word Embeddings



19



假设你有一个词嵌入模型，你会发现与“国王”（king）向量最接近的可能是“王后”（queen）、“王室”（royalty）等，与“苹果”（apple）向量最接近的可能是“香蕉”（banana）、“橙子”（orange）等。

## Summary

1. **文本分类:** 词嵌入可以用作文本分类算法的输入特征。
2. **机器翻译:** 在翻译模型中，源语言和目标语言通常都会使用词嵌入。
3. **问答系统、聊天机器人:** 用于理解用户输入和生成合适的回应。

总体来说，词嵌入是现代NLP中非常核心的一个概念，它为各种复杂的语言模型和应用提供了基础。然而，也正因为其强大的表达能力，如何正确地使用和解释词嵌入仍然是一个挑战。

## History of Word Embeddings

- Structuralist Linguistics (Firth, 1957)
- Recurrent Networks (Rumelhart, Hinton & Williams, 1986)
- Latent Semantic Analysis (Deerwester et al., 1990)
- Hyperspace Analogue to Language (Lund, Burgess & Atchley, 1995)
- Neural Probabilistic Language Models (Bengio, 2000)
- NLP (almost) from Scratch (Collobert et al., 2008)
- word2vec (Mikolov et al., 2013)
- GloVe (Pennington, Socher & Manning, 2014)

18



## Word2Vec

### word2vec (Mikolov et al., 2013)

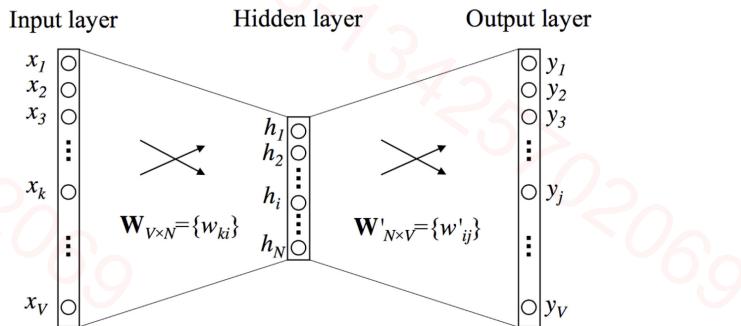
- **Idea:** predict rather than count
- Instead of counting how often each word  $w$  occurs near "university" train a classifier on a **binary prediction task**:
  - Is  $w$  likely to show up near "university"?
- We don't actually care about this task
  - But we'll take the learned classifier weights as the word embeddings
- Use running text as implicitly supervised training data
- No need for hand-labeled supervision

24



Word2Vec 是由 Tomas Mikolov 等人于 2013 年提出的一种用于计算词向量的浅层神经网络模型。该模型主要有两种架构：Skip-gram 和 Continuous Bag of Words (CBOW)。

## Word2Vec 1-word Context Model



The  $k^{\text{th}}$  row  $\mathbf{v}_k$  of  $\mathbf{W}$  is a representation of word  $k$ .

The  $j^{\text{th}}$  column  $\mathbf{v}'_j$  of  $\mathbf{W}'$  is an (alternative) representation of word  $j$ .

If the (1-hot) input is  $k$ , the linear sum at each output will be  $u_j = \mathbf{v}'^T \mathbf{v}_k$

25



## Skip-gram 和 CBOW

- Skip-gram:** 给定一个中心词，预测其周围的上下文词。这种方法通常更擅长学习罕见词的向量。
- CBOW:** 给定一个词的上下文，预测这个词本身。CBOW 通常训练得更快，并且对高频词表现得更好。

## Analysis

### Word2Vec issues

- Word2Vec is a linear model in the sense that there is no activation function at the hidden nodes
- this 1-word prediction model can be extended to multi-word prediction in two different ways:
  - Continuous Bag of Words
  - Skip-Gram
- need a computationally efficient alternative to Softmax (Why?)
  - Hierarchical Softmax
  - Negative Sampling
- need to sample frequent words less often

27



优点：

1. 高效的词向量表示：Word2Vec 能有效地捕捉词义和词的多样性。

2. 可扩展性：模型能在大规模数据集上进行训练。

缺点：

1. 不能处理词义消歧：一个词在不同的上下文中可能有不同的意义，但 Word2Vec 会把它分配一个固定的向量。
2. 需要大量数据：为了获得有用的词向量，需要大量的文本数据。

## Example

如果你训练了一个 Word2Vec 模型，你可以进行一些有趣的词向量运算，比如：

$\text{vector}(\text{'king'}) - \text{vector}(\text{'man'}) + \text{vector}(\text{'woman'})$ ，结果可能非常接近  $\text{vector}(\text{'queen'})$ 。

## Summary

Word2Vec 在各种 NLP 应用中都有广泛的应用，包括但不限于文本分类、推荐系统、命名实体识别等。总体来说，Word2Vec 是一种非常流行和有效的词嵌入方法，其开创性的贡献在于使用浅层神经网络以高效、可扩展的方式捕捉词义和语法信息。然而，由于其不能处理词义消歧等问题，后来有更多高级的模型（如 BERT、GPT 等）进行了改进。

## Extension Details

### Cost Function

#### Cost Function

Softmax can be used to turn these linear sums  $u_j$  into a probability distribution estimating the probability of word  $j$  occurring in the context of word  $k$

$$\text{prob}(j|k) = \frac{\exp(u_j)}{\sum_{j'=1}^V \exp(u_{j'})} = \frac{\exp(\mathbf{v}_j^T \mathbf{v}_k)}{\sum_{j'=1}^V \exp(\mathbf{v}_{j'}^T \mathbf{v}_k)}$$

We can treat the text as a sequence of numbers  $w_1, w_2, \dots, w_T$  where  $w_i = j$  means that the  $i^{\text{th}}$  word in the text is the  $j^{\text{th}}$  word in the vocabulary.

We then seek to maximize the log probability

$$\frac{1}{T} \sum_{t=1}^T \sum_{-c \leq r \leq c, r \neq 0} \log \text{prob}(w_{t+r}|w_t)$$

where  $c$  is the size of training context (which may depend on  $w_t$ )

在 Skip-gram 中，目标是最大化以下条件概率：

$$\prod_{t=1}^T \prod_{-c \leq j \leq c, j \neq 0} P(w_{t+j} | w_t)$$

其中  $w_t$  是句子中的第  $t$  个词， $c$  是上下文窗口大小。

在 CBOW 中，目标是最大化：

$$\prod_{t=1}^T P(w_t | w_{t-c}, \dots, w_{t+c})$$

## CBOW vs Skip-Gram

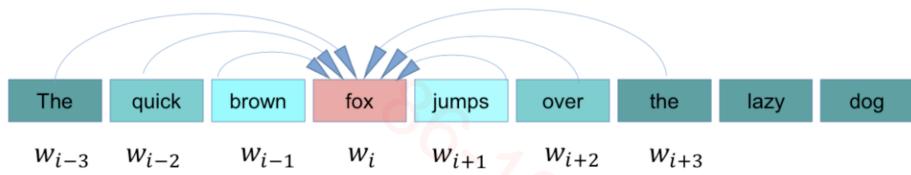


Figure: Continuous Bag of Words (CBOW)

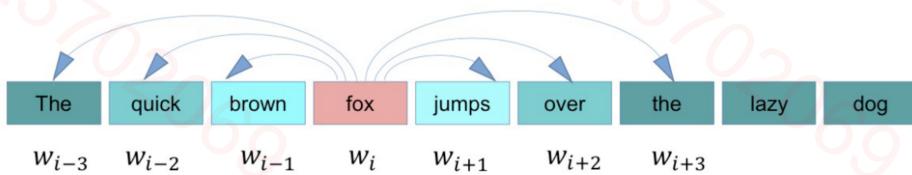
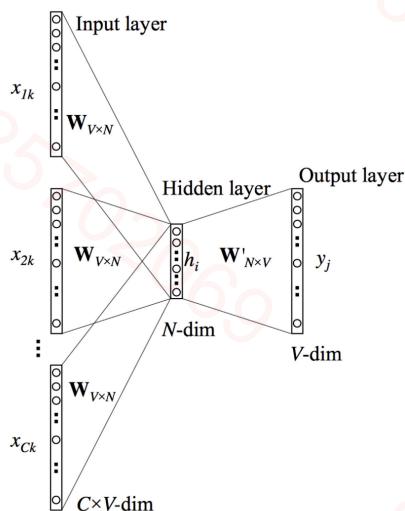


Figure: Skip-Gram model

## CBOW

## Continuous Bag of Words (CBOW)



- If several context words are each used independently to predict the center word, the hidden activation becomes a sum (or average) over all the context words
- Note the difference between this and NetTalk – in word2vec (CBOW) all context words share the same input-to-hidden weights

31



Continuous Bag of Words (CBOW) 是 Word2Vec 模型中的一种架构，它的核心思想是给定一个词的上下文（周围的词），预测这个词本身。与 Skip-gram 模型相反，Skip-gram 是给定一个词，预测其上下文。

### Model

CBOW 模型通常由一个输入层、一个隐藏层和一个输出层组成。输入是上下文词的 one-hot 编码，输出则是目标词的概率分布。

### 数学模型

对于一个给定的上下文  $C = \{w_{i-1}, w_{i+1}, \dots, w_{i+n-1}\}$ ，CBOW 模型试图最大化以下条件概率：

$$P(w_i|C) = \frac{\exp(v' w_i^T v_C)}{\sum j=1^{|V|} \exp(v'_j^T v_C)}$$

其中， $v_{w_i}$  和  $v'_j$  是词  $w_i$  的输入和输出向量， $v_C$  是上下文向量，通常是上下文词向量的平均值， $|V|$  是词汇表的大小。

### Analysis

优点：

1. 快速训练：由于同时考虑了所有上下文词，CBOW 通常比 Skip-gram 训练得更快。
2. 对高频词效果好：CBOW 更擅长准确地表示高频词。

缺点：

1. 忽视词序和语法结构：CBOW 只是简单地平均了上下文词向量，没有考虑词序。

2. 对罕见词效果差：相比 Skip-gram，CBOW 在处理罕见词或者短语上通常表现得没有那么好。

## Example

假设你有一个句子“狗喜欢跑”。如果你使用一个窗口大小为 1 的 CBOW 模型，当上下文是“狗”和“跑”时，模型会试图预测中间的词“喜欢”。

## Summary

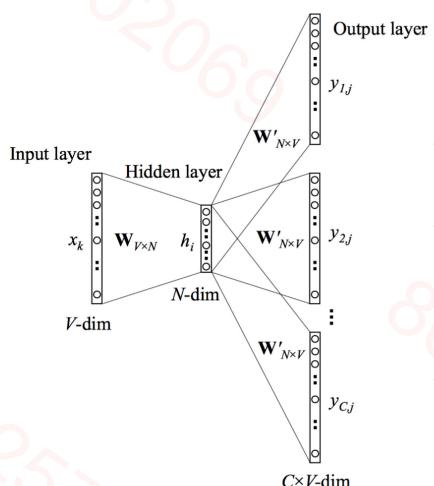
CBOW 在多种 NLP 任务中有应用，包括但不限于：

1. 文本分类和情感分析：使用词向量作为特征输入。
2. 命名实体识别和关系抽取：用于生成词或实体的密集向量表示。

总体来说，CBOW 是一个高效而简单的词嵌入模型，尤其适用于拥有大量高频词的数据集。然而，对于捕捉复杂的词序和语法结构，或者对罕见词的处理，它可能不如其他更复杂的模型。

## Skip-Gram Model

### Word2Vec Skip-Gram Model



- try to predict the context words, given the center word
- this skip-gram model is similar to CBOW, except that in this case a single input word is used to predict multiple context words
- all context words share the same hidden-to-output weights

Skip-Gram 是 Word2Vec 中的一种模型架构，其主要目的是通过一个中心词来预测或分类周围的上下文词。与 CBOW (Continuous Bag of Words) 相比，Skip-Gram 是反向操作：它使用一个单一的输入词来预测它周围的上下文，而不是使用上下文来预测一个单一的词。

## Model

Skip-Gram 模型结构相对简单，通常包括一个输入层、一个隐藏层和一个输出层。输入层通常使用 one-hot 编码来表示中心词，输出层则是一个 softmax 函数，用于预测上下文词的概率分布。

## Model

Skip-Gram 模型的目标是最大化以下条件概率：

$$\prod_{t=1}^T \prod_{-c \leq j \leq c, j \neq 0} P(w_{t+j} | w_t)$$

其中， $w_t$  是句子中的第 $t$ 个词， $c$  是上下文窗口大小。通常，这个最大似然函数会通过诸如随机梯度下降的优化算法来进行优化。

## Analysis

优点：

1. 对罕见词表现良好：Skip-Gram 模型通常能更好地学习罕见词或短语的高质量向量。
2. 捕获更复杂的词关系：相较于 CBOW，Skip-Gram 更能捕获两个词之间的复杂关系。

缺点：

1. 训练速度较慢：相比于 CBOW，Skip-Gram 通常需要更多的时间来训练，尤其是在大型数据集上。

## Summary

Skip-Gram 模型被广泛应用于多种 NLP 任务，包括：

1. 文本分类和情感分析：词向量可以用作其他机器学习模型的输入。
2. 命名实体识别（NER）和关系抽取：用于生成更具代表性的词或短语向量。
3. 词义消歧：由于 Skip-Gram 能够捕获词与多个上下文之间的关系，因此它在词义消歧任务中也表现得相对较好。

总体而言，Skip-Gram 是一种强大而灵活的模型，尤其适用于需要捕获复杂词关系和处理罕见词或短语的应用场景。然而，需要注意的是，由于其训练速度相对较慢，因此通常需要更多的计算资源。

## Training Method

## Word2Vec Weight Updates

If we assume the full softmax, and the correct output is  $j^*$ , then the cost function is

$$E = -u_{j^*} + \log \sum_{j'=1}^V \exp(u_{j'})$$

the output differentials are

$$\mathbf{e}_j = \frac{\partial E}{\partial u_j} = -\delta_{jj^*} + \frac{\partial}{\partial u_j} \log \sum_{j'=1}^V \exp(u_{j'})$$

where

$$\delta_{jj^*} = \begin{cases} 1, & \text{if } j = j^*, \\ 0, & \text{otherwise.} \end{cases}$$

28



## Word2Vec Weight Updates

hidden-to-output differentials

$$\frac{\partial E}{\partial w'_{ij}} = \frac{\partial E}{\partial u_j} \frac{\partial u_j}{\partial w'_{ij}} = \mathbf{e}_j h_i$$

hidden unit differentials

$$\frac{\partial E}{\partial h_i} = \sum_{j=1}^V \frac{\partial E}{\partial u_j} \frac{\partial u_j}{\partial h_i} = \sum_{j=1}^V \mathbf{e}_j w'_{ij}$$

input-to-hidden differentials

$$\frac{\partial E}{\partial w_{ki}} = \frac{\partial E}{\partial h_i} \frac{\partial h_i}{\partial w_{ki}} = \sum_{j=1}^V \mathbf{e}_j w'_{ij} x_k$$

29



Word2Vec 使用负采样 (Negative Sampling) 或分层 Softmax (Hierarchical Softmax) 等技术来优化训练速度和效果。

## Hierarchical Softmax

Hierarchical Softmax 是一种用于大规模输出空间 (如大词汇表) 的优化技术。在自然语言处理任务，特别是词嵌入和语言模型中，常常需要计算一个词汇表中所有词的概率分布。在标准的 Softmax 中，这会涉及大量的计算，尤其是当词汇表非常大时。Hierarchical Softmax 通过构建一个二叉树来代表词汇表，并用这个树来有效地计算词的概率，从而显著减少了计算复杂性。

1. **构建二叉树**：词汇表中的每个词都被表示为二叉树中的一个叶节点，而内部节点则用于存储额外的信息。
2. **路径和编码**：从根节点到每一个叶节点（词）的路径被编码为一系列的 0 和 1，这些编码用于计算从根节点到特定词的条件概率。
3. **概率计算**：给定一个上下文（例如，在 Word2Vec 中的中心词），通过二叉树从根节点到目标词节点的路径，计算该词的条件概率。

在自然语言处理中，尤其是在有大量输出类别（例如大词汇表）的情况下，使用传统的 Softmax 通常会非常低效。具体来说，传统的 Softmax 需要对整个词汇表中的所有词进行计算和归一化，这通常是一个时间复杂度为  $O(V)$  的操作，其中  $V$  是词汇表的大小。

Hierarchical Softmax 的出现就是为了解决这个问题。它通过使用一个二叉树来表示词汇表，将词的概率从一个  $O(V)$  的问题简化为一个  $O(\log V)$  的问题。在这个二叉树中，每个叶子节点都代表词汇表中的一个词，而内部节点则用于进行概率的近似计算。

使用层次结构的主要好处如下：

1. **计算效率**：在二叉树中，从根到任何一个叶子（词）的路径长度为  $O(\log V)$ ，这大大减少了计算概率所需的时间。
2. **内存效率**：由于不需要存储和计算一个大小为  $V$  的概率分布，因此也节省了大量的内存。
3. **可扩展性**：对于非常大的词汇表，使用传统的 Softmax 可能是不可行的，而 Hierarchical Softmax 则可以很容易地扩展到这些情况。

因此，在面对大词汇表或需要高效计算的场景时，Hierarchical Softmax 是一个非常有用的优化技术。

## Negative Sampling

负采样是一种用于优化词嵌入和其他自然语言处理模型的训练算法。它是一种近似优化方法，旨在解决当词汇表非常大时，标准 Softmax 计算成本过高的问题。

在标准的 Softmax 训练中，每次更新都需要考虑词汇表中的所有词，这是一个复杂度为  $O(V)$  的操作。负采样通过将这个问题转化为一个二分类问题（目标词为正类，随机选取的其他词为负类）来简化这一过程。

具体来说，对于每一对正样本（中心词和上下文词），负采样会从词汇表中随机选择  $k$  个负样本（即与中心词无关的词）。然后，模型的任务就是正确地分类这些正样本和负样本。

**优点：**

1. **计算效率**：与标准 Softmax 相比，负采样大大降低了计算复杂性。
2. **适用于大规模词汇表**：对于具有大量词汇的数据集，负采样是一种有效的优化手段。

**缺点：**

1. **近似优化**：由于是一种近似方法，可能会有一定的精度损失。
2. **超参数选择**：需要手动选择负样本的数量  $k$ ，这可能会影响模型的性能。

总体而言，负采样是一种非常有效的优化技术，尤其在需要快速训练或处理大规模词汇表的情况下。然而，由于它是一种近似方法，所以在某些对精度要求非常高的应用中可能不是最佳选择。

## NLP Model Summary

### Applications

#### 文本分类

##### | NLP Applications

- Text Classification

"I love this movie.  
I've seen it many times  
and it's still awesome."



"This movie is bad.  
I don't like it at all.  
It's terrible."



Image Credit: Sentiment Analysis. [https://colab.research.google.com/github/shangeth/Google-ML-Academy/blob/master/2-Deep-Neural-Networks/2\\_9\\_ANN\\_Natural\\_Language\\_Processing.ipynb](https://colab.research.google.com/github/shangeth/Google-ML-Academy/blob/master/2-Deep-Neural-Networks/2_9_ANN_Natural_Language_Processing.ipynb)

5

### 机器翻译

## NLP Applications

- Machine Translation

The screenshot shows the Google Translate interface. At the top, it says "Google Translate" and has a "Sign in" button. Below that is a toolbar with "Text" and "Documents" tabs, and language selection buttons for "DETECT LANGUAGE", "ENGLISH", "SPANISH", "FRENCH", "GERMAN", "ENGLISH", and "SPANISH". The main area shows a text input "I love teaching humans and machines" and its German translation "Ich liebe es, Menschen und Maschinen beizubringen". There are also audio playback icons, a progress bar (35 / 5000), and a "Send feedback" link.



6

## Q & A

### NLP Applications

- Question Answering/Comprehension

#### Passage Sentence

In meteorology, precipitation is any product of the condensation of atmospheric water vapor that falls under gravity.

#### Question

What causes precipitation to fall?

#### Answer Candidate

gravity

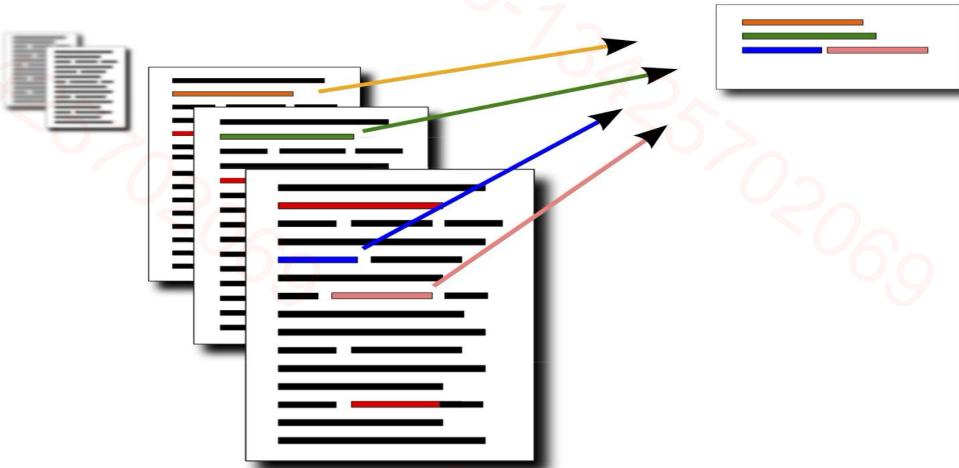
Source: The Stanford Question Answering Dataset (SQuAD). <https://rajpurkar.github.io/MLX/qa-and-squad/>

7

## Text Summary

## NLP Applications

### ➤ Text Summarization



Source: Automatic Text Summarization, Cognitive Science and Knowledge Management Series

8

## Preprocessing

对于机器来说，只能进行数字运算，并不能直接理解dog, doggy, dogg, 的含义。所以需要将其转化成数字的表达。

### One-hot Encoding

#### Representing text: One-hot Encoding

- One-hot vector of an ID is a vector filled with 0s, except for a 1 at the position associated with the ID.
  - For e.g., for vocabulary size D=10, the one-hot vector of word (w) having ID=4 is  
 $e(w) = [0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0]$
- One-hot encoding makes no assumption about word similarity
  - All words are equally similar/different from each other
- This is a natural representation to start with, though a poor one

Deep	Learning	is	hard	fun
1	0	0	0	0
0	1	0	0	0
0	0	1	0	0
0	0	0	1	0
0	0	0	0	1



Slide Credit: Hugo Larochelle. Neural Networks

10

独热编码是一种用于表示类别变量的数字编码方法。在自然语言处理（NLP）和其他机器学习应用中，独热编码常用于将文本数据转换为能够输入到模型中的数值形式。

假设我们有一个词汇表  $V$ , 其中包含  $n$  个唯一词。每个词都可以用一个长度为  $n$  的向量来表示。  
在这个向量中, 该词对应的索引位置为 1, 其他位置都是 0。

例如, 如果我们的词汇表是 {"苹果", "香蕉", "橙子"}, 那么:

- “苹果” 可以表示为 [1, 0, 0]
- “香蕉” 可以表示为 [0, 1, 0]
- “橙子” 可以表示为 [0, 0, 1]

总的来说, 独热编码是一种非常基础但有局限性的文本编码方法。它适用于词汇表较小或不需要捕捉词义复杂性的场景。然而, 在需要捕捉词之间的相似性或者处理大规模词汇表的应用中, 通常会选择更复杂的编码方式, 如词嵌入。

## Count Vectorizer

### Representing text: Count Vectorizer

- Counts the number of occurrences of each word appearing in a document
- Converts a collection of text documents to a vector of term/token counts

	the	red	dog	cat	eats	food
1. the red dog ➔	1	1	1	0	0	0
2. cat eats dog ➔	0	0	1	1	1	0
3. dog eats food ➔	0	0	1	0	1	1
4. red cat eats ➔	0	1	0	1	1	0



Image Credit: Ishan Kotian. Fake news classification – NLP @Kaggle. <https://www.kaggle.com/lykin22/fake-news-classification-nlp>

11

Count Vectorizer 是一种用于将文本数据转换为数值形式的技术, 在自然语言处理 (NLP) 中非常常用。与独热编码不同, Count Vectorizer 不仅仅标记词是否出现, 还计算每个词在文档中出现的次数 (频率)。

例如, 假设我们有两个文档:

- 文档 1: "apple apple orange"
- 文档 2: "apple banana"

如果词汇表是 {"apple", "banana", "orange"}, 那么:

- 文档 1 的词频向量可能是 [2, 0, 1]

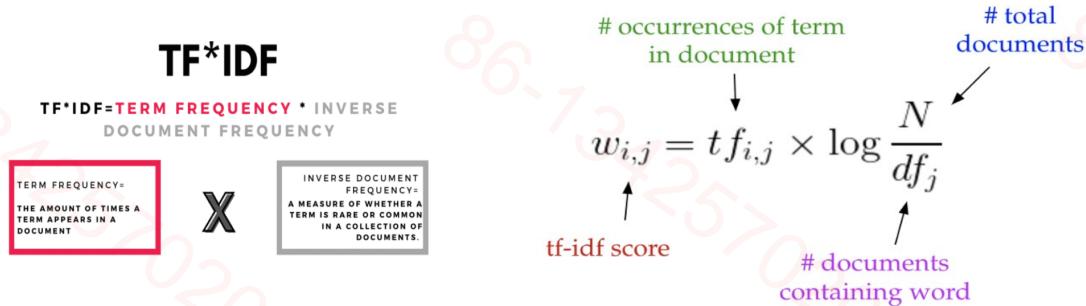
- 文档 2 的词频向量可能是 [1, 1, 0]

总体来说，Count Vectorizer 是一种基础但有用的文本向量化方法，尤其适用于词频信息对模型性能有影响的场景。然而，由于其无法捕捉词义和语序信息，以及可能存在的高维和稀疏问题，在更复杂的 NLP 任务中，通常会用更高级的方法（如 TF-IDF、词嵌入等）来替代它。

## TF-idf Vectorizer

### Representing text: tf-idf vectorizer

- Stands for Term Frequency – Inverse Document Frequency (tf-idf)
- Common algorithm to transform text into a meaningful representation of numbers



- This cheesecake is really **yummy**. I'm going for another slice.
- The restaurant service was very slow.



Slide Credit: Hugo Larochelle. Neural Networks

12

TF-IDF (Term Frequency-Inverse Document Frequency) 是一种常用于自然语言处理 (NLP) 和信息检索 (IR) 的文本向量化方法。与简单的词频 (Term Frequency, TF) 不同，TF-IDF 还考虑了一个词在所有文档中的出现频率，即逆文档频率 (Inverse Document Frequency, IDF)。

TF-IDF 分为两个部分：

1. **词频 (TF)**：与 Count Vectorizer 类似，TF 衡量的是一个词在单个文档中出现的频率。
2. **逆文档频率 (IDF)**：这是一个衡量词的“重要性”的指标。如果一个词在很多文档中都出现，那么它通常被认为是“不重要”的。IDF 通过下面的公式来计算：

$$\text{IDF}(t) = \log \left( \frac{N}{1 + \text{DF}(t)} \right)$$

其中， $N$  是文档总数， $\text{DF}(t)$  是包含词  $t$  的文档数。

最终，词  $t$  在文档  $d$  中的 TF-IDF 值由以下公式计算：

$$\text{TF-IDF}(t, d) = \text{TF}(t, d) \times \text{IDF}(t)$$

总体来说，TF-IDF 是一种比 Count Vectorizer 更高级的文本向量化方法，它不仅考虑了词频，还考虑了词的“稀有性”或“重要性”。然而，由于它同样无法捕捉词义和语序信息，因此在一些更复杂的 NLP 任务中，人们通常会选择更为复杂的向量化方法，如词嵌入。

## Sequence Model

### Processing Temporal Sequences

There are many tasks which require a sequence of inputs to be processed rather than a single input.

- speech recognition
- time series prediction
- machine translation
- handwriting recognition

How can neural network models be adapted for these tasks?

在深度学习中，处理文本数据和图像数据有几个本质区别：

### 数据结构

1. **文本数据**：通常是一维的序列，例如单词或字符组成的数组。
2. **图像数据**：通常是多维的矩阵（通常是三维，考虑到颜色通道）。

### 数据表示

1. **文本数据**：通常需要进行词嵌入（如 Word2Vec、GloVe）或使用 one-hot 编码。
2. **图像数据**：像素值通常直接用于模型训练，或者通过标准化进行简单处理。

### 网络架构

1. **文本数据**：经常使用 RNN（循环神经网络）、LSTM（长短时记忆）或 Transformer 等模型来捕获序列信息。
2. **图像数据**：CNN（卷积神经网络）是处理图像最常用的网络架构。

### 时间复杂性

1. **文本数据**：因为序列的长度可能不同，处理时间可能不确定。

2. **图像数据**：通常有固定的尺寸，因此处理时间相对更可预测。

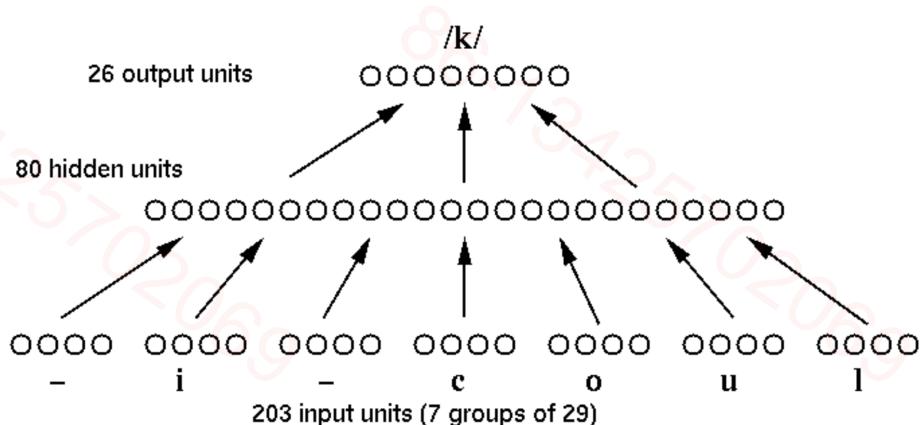
## 空间复杂性

1. **文本数据**：通常更加稀疏，需要更复杂的处理步骤来捕获语义信息。

2. **图像数据**：通常更加密集，局部特征（如边缘、纹理等）更容易捕获。

## MLP ?

### NetTalk Architecture



6



使用多层感知器（MLP，Multi-Layer Perceptron）和循环神经网络（RNN，Recurrent Neural Network）处理自然语言处理（NLP）任务有一些显著的差异和潜在问题。

### MLP（多层感知器）

1. **缺乏序列信息捕获**：MLP 通常不能有效地处理序列数据，因为它没有内置的机制来捕获序列中的时序信息。
2. **固定输入大小**：MLP 需要固定大小的输入，这对于自然语言处理中常见的可变长度文本是一个问题。
3. **参数数量**：如果输入文本很长，为了能够处理它，可能需要一个非常大的输入层，这会导致参数数量剧增。
4. **高计算复杂性**：由于参数数量可能会很大，计算复杂性也相应增加。
5. **局限的上下文理解**：由于没有时序建模，MLP 很难理解诸如词序、依赖关系等复杂的语言特点。

## RNN (循环神经网络)

1. **序列信息捕获**: RNN 能够捕获序列中的时间依赖性，这在自然语言处理中是非常有用的。
2. **可变输入大小**: RNN 可以处理不同长度的序列。
3. **更少的参数**: 与 MLP 相比，RNN 通常有更少的参数，因为它们在序列的各个时间步上共享参数。
4. **梯度消失和爆炸**: RNNs 容易遭受梯度消失和梯度爆炸问题，尤其是在处理长序列时。
5. **计算效率**: 尽管参数更少，但 RNN 的递归性质使得并行化更为困难，这可能会影响训练和推理速度。

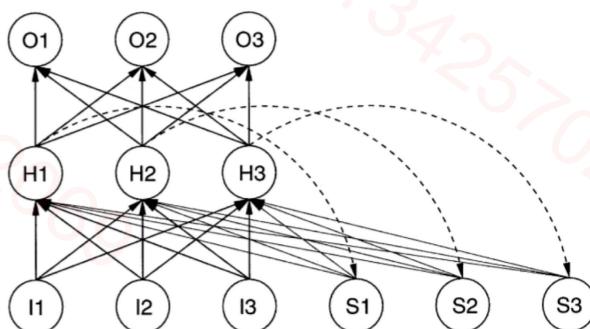
## 比较

1. **对于序列数据**: RNN 明显优于 MLP，因为它能够捕获时间序列信息。
2. **模型复杂性和参数数量**: MLP 可能需要更多的参数和更复杂的结构来捕获文本中的复杂关系，而 RNN 通过参数共享和序列建模减少了这种需要。
3. **计算效率**: MLP 通常更容易并行化，但可能会有更多的参数。RNN 虽然参数更少，但并行化更困难。
4. **灵活性和应用范围**: RNN 通常更适合处理有序列依赖性的任务，而 MLP 更为通用，但在处理序列数据时可能不够有效。

综上所述，MLP 在处理 NLP 任务时的主要问题是其无法捕获序列信息和处理可变长度的输入。相比之下，RNN 能更好地处理这些问题，但可能面临梯度消失和梯度爆炸等挑战。

# Recurrent Neural Network

## Simple Recurrent Network (Elman, 1990)



- at each time step, hidden layer activations are copied to “context” layer
- hidden layer receives connections from input and context layers
- the inputs are fed one at a time to the network, it uses the context layer to “remember” whatever information is required for it to produce the correct output

RNN是处理文字的网络，和CNN不同的是，RNN需要看完整段文章才能进行预测，然而CNN更像是一个特征提取器。所以RNN是需要连接所有的文字，同时需要保留有用的信息。这里有两个关键的要点，一个是需要连接所有的问题。这种对连接性的需求就将RNN设置成循环连接的结构，也是RNN名字的来源。第二个要点是，RNN需要保留有用的信息，所以RNN需要一个记忆模块。

循环神经网络（Recurrent Neural Networks, RNN）是一种专门用于处理序列数据（如文本、时间序列等）的神经网络。与卷积神经网络（CNN）主要用于图像处理不同，RNN有两个关键特点：

## 1. 循环连接（Sequential Connectivity）

- **定义**：在 RNN 中，每个单元（或称为神经元）不仅与下一层的单元连接，还与同层下一个时间步的单元进行“循环连接”。
- **目的**：这种结构使得网络能够捕获序列数据中的时序依赖关系，即前文的信息能影响后文的处理。
- **应用**：这种连接性使得 RNN 非常适用于诸如自然语言处理、语音识别和时间序列分析等任务。

## 2. 记忆模块（Memory）

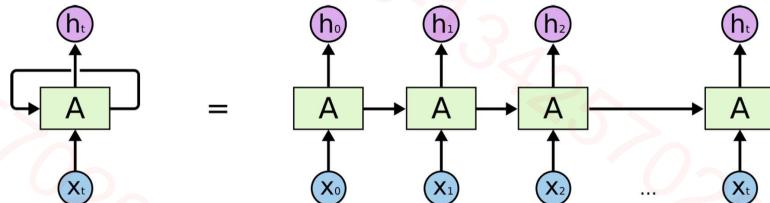
- **定义**：RNN 的循环结构允许网络维持一种“记忆”，这种记忆用于存储过去时间步中的信息。
- **目的**：通过这种记忆，RNN 能够保存有用的信息并在需要时使用，从而更好地处理长序列和复杂依赖。
- **局限性**：基础的 RNN 结构很难处理长期依赖（Long-term Dependencies），因此有了更先进的变体，如长短时记忆网络（LSTM）和门控循环单元（GRU）。

## RNN 与 CNN 的不同：

- **全局 vs 局部**：RNN 需要看完整段文章（或整个序列）才能进行有效的预测，而 CNN 更像是一个局部特征提取器。
- **序列 vs 空间**：RNN 适用于有时间/顺序依赖的数据，而 CNN 主要用于空间数据（如图像）。
- **复杂依赖 vs 局部特征**：RNN 能够捕捉更复杂的依赖关系，而 CNN 通常更擅长捕捉局部、简单的特征。

## Back Propagation

## Back Propagation Through Time



- we can “unroll” a recurrent architecture into an equivalent feedforward architecture, with shared weights
- applying backpropagation to the unrolled architecture is referred to as “backpropagation through time”
- we can backpropagate just one timestep, or a fixed number of timesteps, or all the way back to beginning of the sequence

10



RNN (Recurrent Neural Network, 循环神经网络) 的反向传播算法通常被称为“通过时间反向传播” (Backpropagation Through Time, BPTT)。这里是一些关键点：

### “展开”架构

在 RNN 中，我们可以将循环架构“展开”成一个等效的前馈架构。在这个展开的网络中，每个时间步都有一个网络层，所有这些网络层共享权重。这样做的目的是为了能够清晰地看到序列中每个时间步的输入是如何影响最终输出的。

### 通过时间反向传播 (BPTT)

一旦网络被展开，就可以像在普通的前馈神经网络中一样使用标准的反向传播算法。不过，因为权重是在所有时间步中共享的，所以在更新权重时需要考虑所有展开步骤中的梯度。具体来说，对于每个权重，我们不仅要计算当前时间步的梯度，还要计算之前所有时间步的梯度，然后将它们加在一起。

### 反向传播的时间跨度

1. **一个时间步**：你可以只反向传播一个时间步的梯度。这基本上就是标准的反向传播算法，但是这样做会失去 RNN 捕获时间依赖性的能力。
2. **固定数量的时间步**：为了减少计算负担，你可以选择反向传播固定数量的时间步。这被称为“截断的 BPTT” (Truncated BPTT)。这样做会减少一些计算量，但可能会损失一些长期依赖信息。
3. **整个序列**：最精确的方法是从序列的末尾一直反向传播到开始。这样可以捕获整个序列中的依赖性，但计算成本也最高。

### 例子

假设我们有一个用于文本生成的 RNN，输入序列是 "hello"，我们希望下一个字符是 " "（空格）。在这个情况下，展开的网络会有 5 个层，每个层对应输入序列中的一个字符 ('h', 'e', 'l', 'l', 'o')。

1. 在前向传播阶段，每个字符都通过 RNN 层传播，生成一个输出和一个新的隐藏状态。
2. 在反向传播阶段，从输出开始，计算损失函数相对于每个参数的梯度。
3. 梯度是在所有时间步累加的，然后用于更新权重。

这样，RNN 可以学习 "hello" 序列后面应该跟什么字符。通过时间反向传播算法让 RNN 能够在序列中捕获这种依赖性。

## Model of Computation

在计算理论中，不同类型的计算模型用于描述和分析算法和语言的计算能力。以下是一些主要的计算模型：

### Regular (正则)

1. **描述**：正则模型主要用于描述正则语言，通常使用正则表达式或有限状态自动机进行表示。
2. **应用**：文本搜索、词法分析等。

### Context-Free (上下文无关)

1. **描述**：上下文无关模型用于描述上下文无关语言，通常使用上下文无关文法 (CFG) 进行表示。
2. **应用**：编程语言的语法分析、自然语言处理等。

### Context-Sensitive (上下文相关)

1. **描述**：上下文相关模型用于描述上下文相关语言，通常使用上下文相关文法进行表示。
2. **应用**：某些自然语言构造、编程语言的语义分析等。

### Recursively Enumerable (递归可枚举)

1. **描述**：递归可枚举模型描述了一类最为一般的语言，它们可以由图灵机识别但不一定可判定。
2. **应用**：描述所有可计算问题。

### Finite State Automaton (有限状态自动机)

1. **描述**：这是一种最简单的计算模型，只有有限多个状态和转换规则。
2. **应用**：文本搜索、网络协议等。

### Push Down Automaton (下推自动机)

1. **描述**：这种模型比有限状态自动机更强大，因为它有一个栈来存储信息。

2. 应用：编程语言的语法分析、某些自然语言处理任务等。

### Linear Bounded Automaton (线性有界自动机)

1. 描述：这是介于下推自动机和图灵机之间的一种模型，它在输入带上有限的空间进行计算。

2. 应用：某些复杂但受限的计算任务。

### Turing Machine (图灵机)

1. 描述：图灵机是一种最为强大的计算模型，能模拟任何其他计算模型。

2. 应用：用于定义和理解“可计算性”和“计算复杂性”。

## Chomsky Hierarchy

Language	Machine	Example
Regular	Finite State Automaton	$a^n$ ( $n$ odd)
Context Free	Push Down Automaton	$a^n b^n$
Context Sensitive	Linear Bounded Automaton	$a^n b^n c^n$
Recursively Enumerable	Turing Machine	true QBF

## RNN & Finite State Machine

## Task: Formal Language Recognition

Accept	Reject
1	0
1 1	1 0
1 1 1	0 1
1 1 1 1	0 0
1 1 1 1 1	0 1 1
1 1 1 1 1 1	1 1 0
1 1 1 1 1 1 1	1 1 1 1 1 1 0
1 1 1 1 1 1 1 1	1 0 1 1 1 1 1 1

Scan a sequence of characters one at a time,  
then classify the sequence as Accept or Reject.

13



RNN（循环神经网络）在其训练和操作过程中本质上模拟了有限状态机（Finite State Machine）。这意味着它们在每个时间步都有一个内部状态，该状态不仅取决于当前输入，还取决于之前的状态。这种状态维持和转换的特性为 RNN 提供了处理序列数据和执行各种任务的能力，从某种程度上模仿了有限状态机的工作原理。

更进一步地，有没有可能某些类型的神经网络（包括 RNN）可能具有图灵完全性，也就是说，它们有潜力模拟通用图灵机。即更复杂、更强大的语言模型（如 LLM）可能被训练成类似于操作系统的结构。如果这些模型确实达到了图灵完全性，那么它们不仅能执行复杂的计算任务，还能理论上模拟任何可计算的过程或系统。

如果这些网络是图灵可计算的，那么真的可能出现一个由数据驱动的全新计算框架，新的操作系统。一种全新的计算范式，其中操作系统和计算框架可能是由数据驱动和自适应的，而不仅仅是预定义的算法和规则构建的。

## Task: Formal Language Recognition

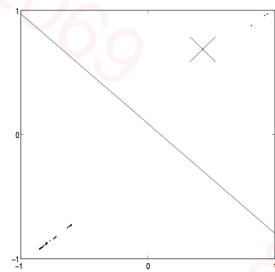
Accept	Reject
1	0
1 1	1 0
1 1 1	0 1
1 1 1 1	0 0
1 1 1 1 1	0 1 1
1 1 1 1 1 1	1 1 0
1 1 1 1 1 1 1	1 1 1 1 1 1 0
1 1 1 1 1 1 1 1	1 0 1 1 1 1 1 1

Scan a sequence of characters one at a time,  
then classify the sequence as Accept or Reject.

13



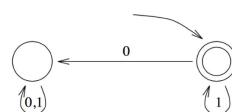
## Dynamic Recognizers



$$W_0 = \begin{bmatrix} -0.89 & -0.09 & -0.14 \\ -1.13 & -0.09 & -0.14 \end{bmatrix}$$

$$W_1 = \begin{bmatrix} 0.20 & 0.68 & 0.96 \\ 0.20 & 0.81 & 1.19 \end{bmatrix}$$

$$P = \begin{bmatrix} -0.07 & 0.66 & 0.75 \end{bmatrix}$$



- gated network trained by BPTT
- emulates exactly the behaviour of Finite State Automaton

14



## Task: Formal Language Recognition

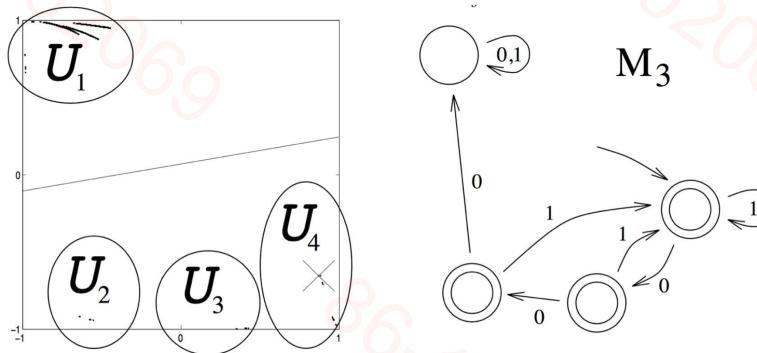
Accept	Reject
1	000
0	11000
10	0001
01	000000000
00	111110000011
100100	1101010000010111
00111110100	1010010001
0100100100	0000
11100	00000
0010	

Scan a sequence of characters one at a time,  
then classify the sequence as Accept or Reject.

15



## Dynamical Recognizers



- trained network emulates the behaviour of Finite State Automaton
- training set must include short, medium and long examples

16



## Formal Language Prediction

## Task: Formal Language Prediction

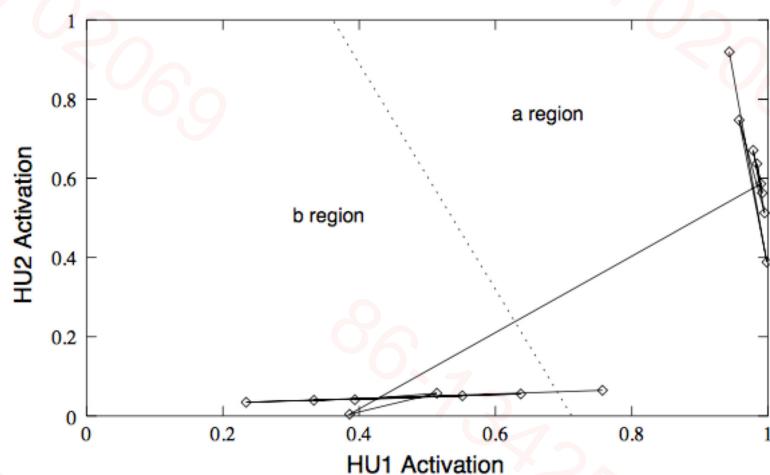
*abaabbabaaabbbaaaabbbbabaabbaaaaaabbbbb ...*

- Scan a sequence of characters one at a time, and try at each step to predict the next character in the sequence.
- In some cases, the prediction is probabilistic.
- For the  $a^n b^n$  task, the first  $b$  is not predictable, but subsequent  $b$ 's and the initial  $a$  in the next subsequence are predictable.

19



## Oscillating Solution for $a^n b^n$

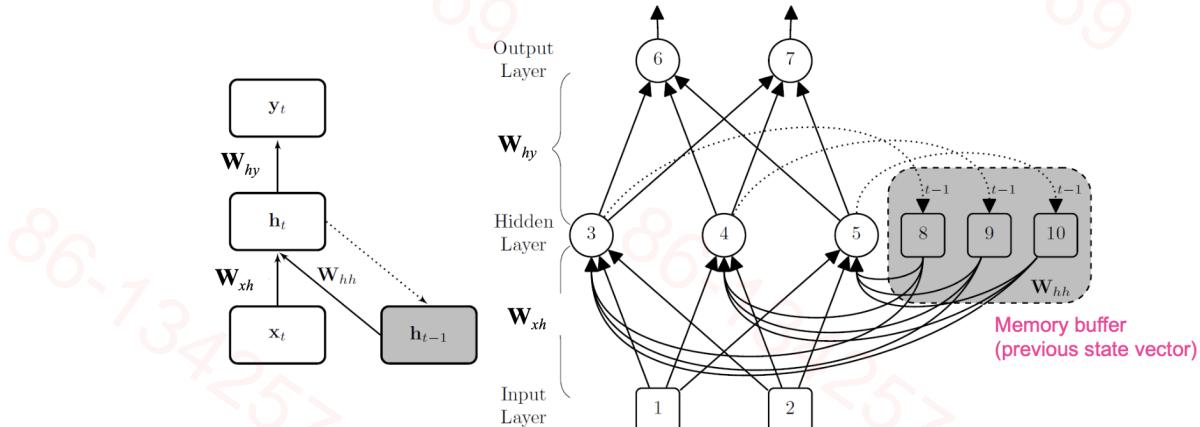


21



## RNN Structure

- $x_t$  – input vector at time  $t$
- $h_t$  - hidden layer activations at time  $t$
- $y_t$  – output vector at time  $t$
- E.g. given word at  $t$  ( $x_t$ ), predict the next word – the word at  $t+1$  ( $y_t$ )



47

在一个基础的循环神经网络（RNN）中，主要有以下几个组件：

### 1. 输入向量 $x_t$

- 定义：在时间步  $t$ ，网络接收一个输入向量  $x_t$ 。
- 示例：在自然语言处理任务中， $x_t$  可能是一个词的嵌入向量。

### 2. 隐藏层激活 $h_t$

- 定义：隐藏层激活  $h_t$  是基于当前时间步  $t$  的输入  $x_t$  和前一时间步  $t-1$  的隐藏层激活  $h_{t-1}$  计算得出的。
- 计算：通常使用如下的形式：

$$h_t = \text{Activation}(W_{hh}h_{t-1} + W_{xh}x_t + b_h)$$

其中， $W_{hh}$  和  $W_{xh}$  是权重矩阵， $b_h$  是偏置项，Activation 是激活函数（如 Sigmoid、Tanh 等）。

### 3. 输出向量 $y_t$

- 定义：在时间步  $t$ ，基于隐藏层激活  $h_t$ ，网络生成一个输出向量  $y_t$ 。
- 计算：通常使用如下的形式：

$$y_t = W_{hy} h_t + b_y$$

其中， $W_{hy}$ 是权重矩阵， $b_y$ 是偏置项。

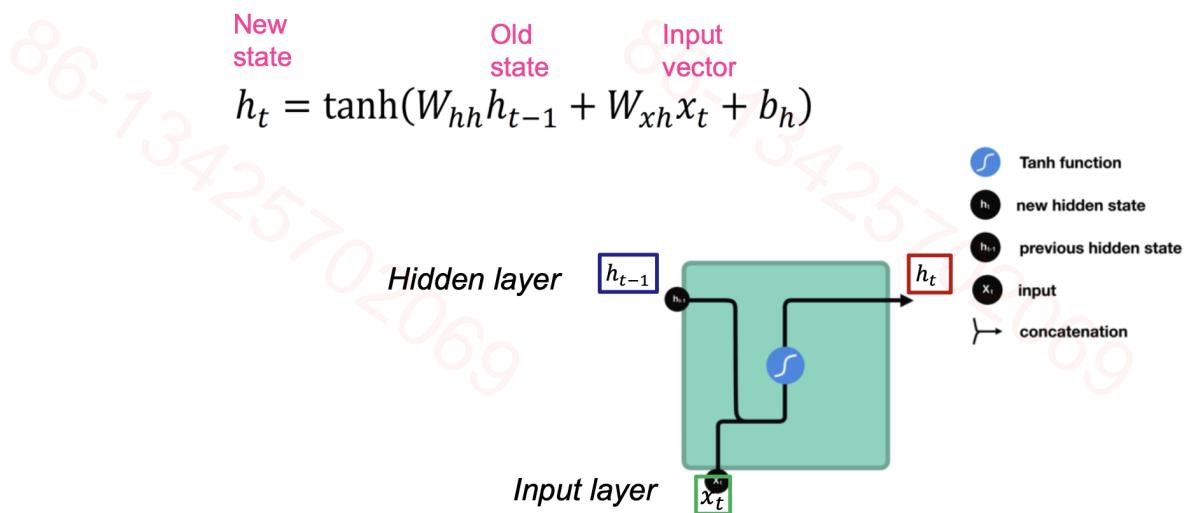
## Gradient Vanishing in RNN

由于在RNN中的更新方式是通过乘积的形式，所以梯度非常容易变成0.



### Hidden state update

- Animation :-)



<https://towardsdatascience.com/illustrated-guide-to-recurrent-neural-networks-79e5eb8049c9>

## Recurrent Neural Network

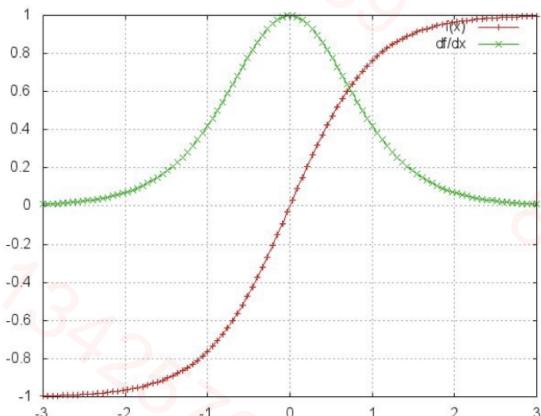
### □ Vanishing gradient problem

$$\square h_t = \tanh(w_{hh}h_{t-1} + w_{hx}x_t + b_h)$$

$$\square y_t = w_{yh}h_t$$

□ For every step, its loss is  $E_n$

$$\frac{\partial E_3}{\partial w_{hh}} = \frac{\partial E_3}{\partial y_3} \frac{\partial y_3}{\partial h_3} \frac{\partial h_3}{\partial w_{hh}} \quad \frac{(\tanh)'w_{hh}(\tanh)'w_{hh}(\tanh)'w_{hh} \dots}{}$$



$$(\tanh)' \leq 1$$

If  $0 < w_{hh} < 1$ , **Vanishing Gradients**

$$(\tanh)'w_{hh}(\tanh)'w_{hh}(\tanh)'w_{hh} \dots \rightarrow 0$$

If  $w_{hh}$  is larger, **Exploding Gradients**

$$(\tanh)'w_{hh}(\tanh)'w_{hh}(\tanh)'w_{hh} \dots \rightarrow \infty$$

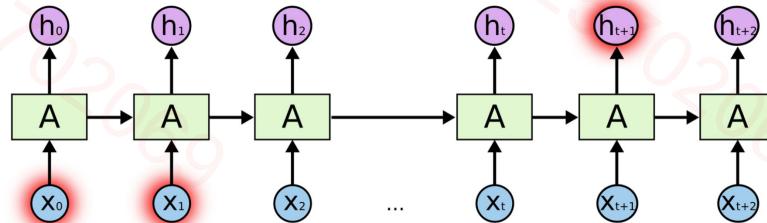


## Shortcomings of RNN

- RNN are susceptible to the vanishing gradient problem
- Reason: during the backward pass, the error gradients will be multiplied by the  $W_{hh}$  matrix multiple times - once for each time-step
- This repeated multiplication may cause the gradient to vanish if the weights are too small
- => Although in principle RNNs have the ability to learn long-term dependencies, in practice this is limited
- Long-term dependencies – dependencies over long distances in a sequence
- Ex.: predicting the next word based on the previous; predicting the last word:
  - The clouds are in the sky
  - Given the previous words, it is obvious that the next word will be "sky" – the gap between the relevant information and the point where it is needed is small (short distance)
  - I grew up in Italy...I speak fluent Italian
  - Recent information suggests we need the name of a language but which one? We need the context of Italy, from further back (long distance)
  - Another example: Translating a long sentence; by the time you have finished it, you have forgotten how it started ☺

# 长短时记忆网络 (Long Short-Term Memory, LSTM)

## Long Range Dependencies

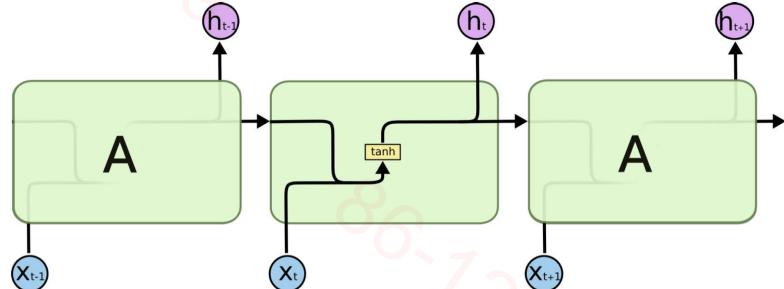


- Simple Recurrent Networks (SRNs) can learn medium-range dependencies but have difficulty learning long range dependencies
- Long Short Term Memory (LSTM) and Gated Recurrent Units (GRU) can learn long range dependencies better than SRN

28



## Simple Recurrent Network

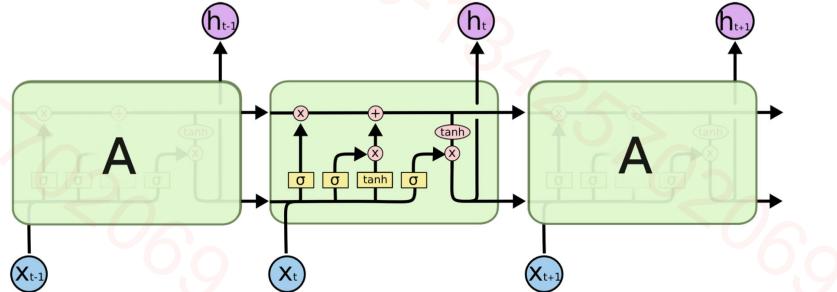


- SRN – context layer is combined directly with the input to produce the next hidden layer.
- SRN can learn Reber Grammar, but not Embedded Reber Grammar.

32



## Long Short Term Memory



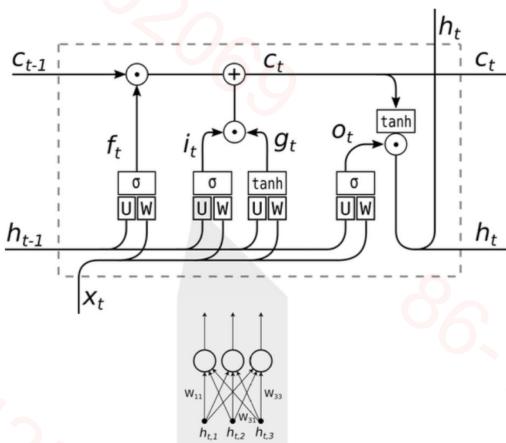
- LSTM – context layer is modulated by three gating mechanisms: forget gate, input gate and output gate.

<http://colah.github.io/posts/2015-08-Understanding-LSTMs/>

33



## Long Short Term Memory



Gates:

$$\begin{aligned}\mathbf{f}_t &= \sigma(W_f \mathbf{x}_t + U_f \mathbf{h}_{t-1} + \mathbf{b}_f) \\ \mathbf{i}_t &= \sigma(W_i \mathbf{x}_t + U_i \mathbf{h}_{t-1} + \mathbf{b}_i) \\ \mathbf{g}_t &= \tanh(W_g \mathbf{x}_t + U_g \mathbf{h}_{t-1} + \mathbf{b}_g) \\ \mathbf{o}_t &= \sigma(W_o \mathbf{x}_t + U_o \mathbf{h}_{t-1} + \mathbf{b}_o)\end{aligned}$$

State:

$$\mathbf{c}_t = \mathbf{c}_{t-1} \odot \mathbf{f}_t + \mathbf{i}_t \odot \mathbf{g}_t$$

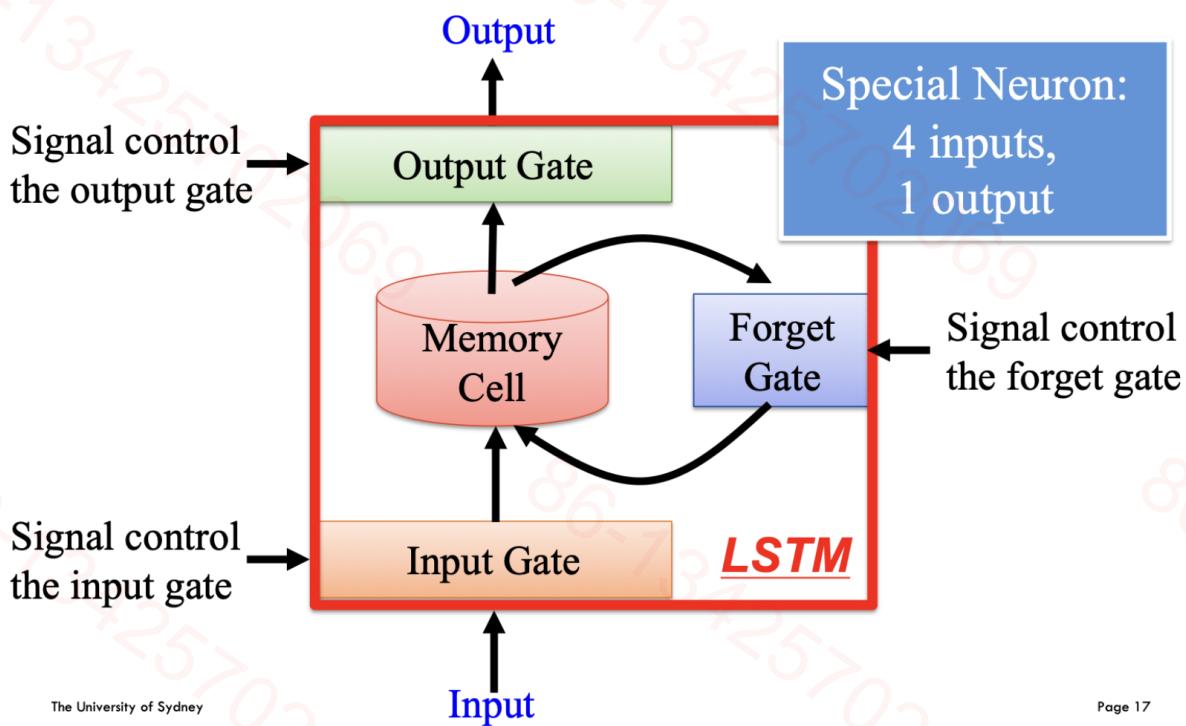
Output:

$$\mathbf{h}_t = \tanh \mathbf{c}_t \odot \mathbf{o}_t$$

34



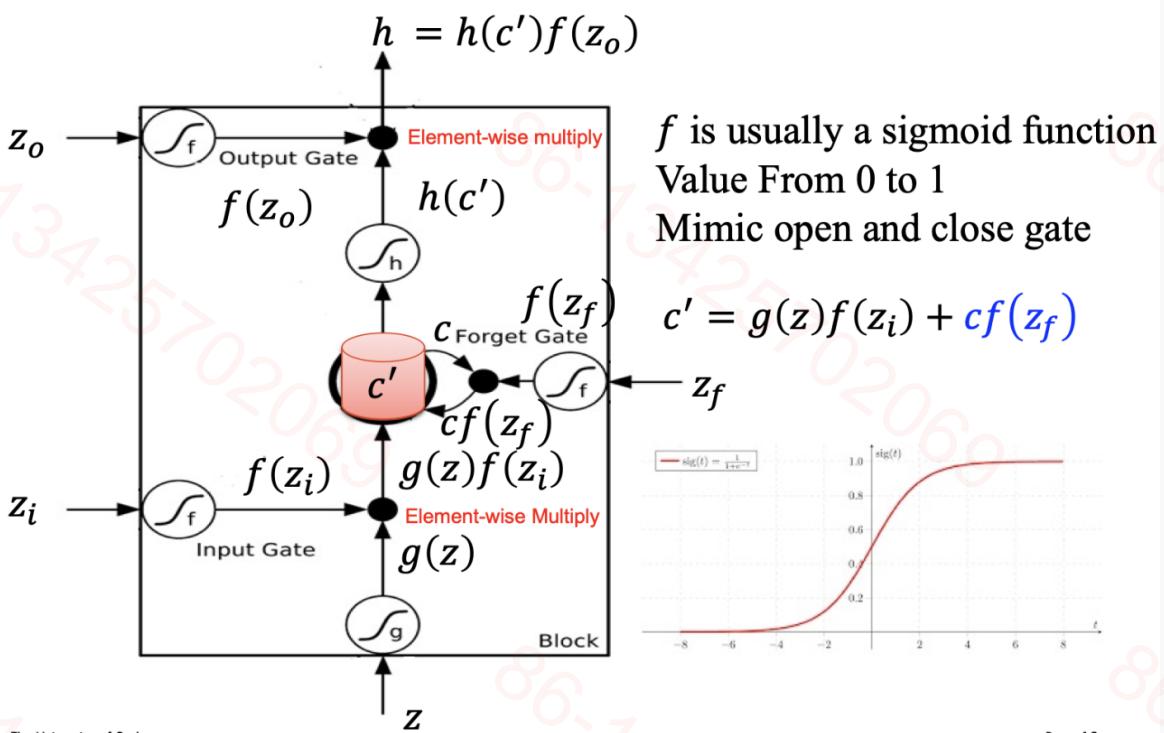
## Long Short-Term Memory



The University of Sydney

Page 17

## Long Short-Term Memory



The University of Sydney

Page 18

原本的RNN因为当hidden layer的数量上升，内存的数量也上升。然而因为梯度是乘积的形式，所以会造成梯度成指数级的乘法，最终不是梯度爆炸就是梯度消失。所以LSTM，使用了遗忘机制，通过门控来管理memory，从而捕捉更长距离的信息。

## 基本结构

LSTM不同于基础的RNN单元，它具有更复杂的内部结构，主要包括三个门（Gate）和一个单元状态（Cell State）。

1. **输入门（Input Gate）**：决定哪些信息需要更新或存储到单元状态。
2. **遗忘门（Forget Gate）**：决定哪些信息需要从单元状态中删除或遗忘。
3. **输出门（Output Gate）**：基于当前的单元状态和输入，决定最终的输出。

除此之外，还有一个**单元状态（Cell State）**，它的作用是作为一种“记忆”的载体，用于存储跨时间步的信息。

## LSTM与基础RNN的优势

LSTM（Long Short-Term Memory）网络是为了解决传统RNN（Recurrent Neural Network）在处理长序列时面临的梯度消失或梯度爆炸问题而设计的。

## 梯度问题与基础RNN

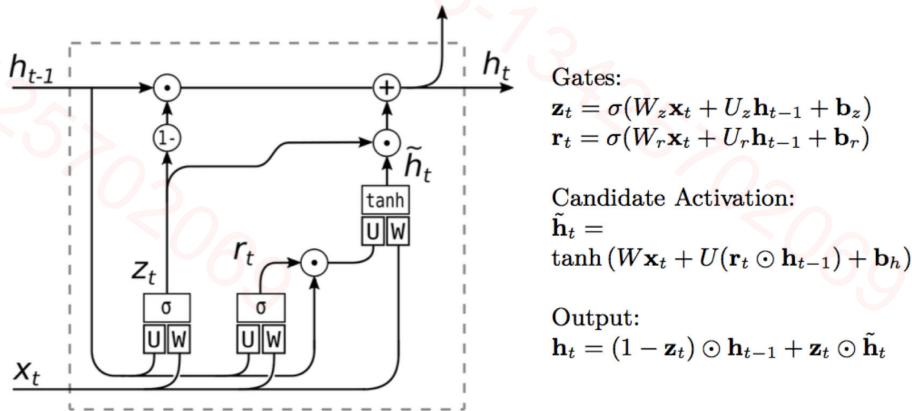
1. **梯度消失与爆炸**：在基础的RNN中，由于梯度是通过时间步的乘积形式进行反向传播的，这就导致梯度可能会快速地消失（变得非常小）或爆炸（变得非常大）。
2. **记忆限制**：这种梯度问题导致基础RNN在实践中很难捕捉到输入序列中的长距离依赖。

## LSTM的优势

1. **门控机制**：LSTM通过引入输入门、输出门和遗忘门来控制信息流，这些门控机制使得LSTM能够决定何时忘记不再重要的信息，何时更新内部状态，以及何时输出状态。
2. **单元状态（Cell State）**：LSTM有一个额外的单元状态，与隐藏状态并行。这个单元状态负责长期的信息存储，而门控机制则决定如何更新这个状态。
3. **长距离依赖**：由于这种设计，LSTM能够有效地捕捉长距离的依赖关系，而不容易受到梯度消失或梯度爆炸的影响。
4. **内存管理**：通过这些门和单元状态，LSTM可以更有效地管理其“记忆”，使其能够在各种需要长期依赖的任务中表现得更好。

## GRU

## Gated Recurrent Unit



GRU is similar to LSTM but has only two gates instead of three.

35



GRU（门控循环单元）和LSTM（长短时记忆）都是循环神经网络（RNN）的变体，设计目的是为了更有效地处理序列数据和解决梯度消失问题。尽管它们有相似的目标，但在结构和运算上有一些不同。下面是一些主要的对比点：

### 结构复杂性

1. **GRU**：GRU模型相对简单，只有两个门（更新门和重置门）。
2. **LSTM**：LSTM更复杂，有三个门（输入门、输出门和遗忘门）。

### 参数数量

1. **GRU**：由于其简单的结构，GRU通常有更少的参数。这有助于更快的训练和更好的泛化，特别是当数据集较小时。
2. **LSTM**：LSTM由于有更多的门，因此有更多的参数，这可能导致更长的训练时间和更高的过拟合风险。

### 记忆能力

1. **GRU**：GRU使用单一的隐藏状态来存储信息，这使得模型更容易训练。
2. **LSTM**：LSTM使用一个隐藏状态和一个单独的“单元状态”（cell state），这有助于模型存储和访问长期依赖。

### 信息流

1. **GRU**：GRU通过更新门和重置门进行信息流的控制。
2. **LSTM**：LSTM通过输入门、输出门和遗忘门进行更精细的信息流控制。

### 计算复杂性

1. **GRU**：由于有更少的门和更简单的结构，GRU 通常计算上更为高效。

2. **LSTM**：更多的门和更复杂的结构导致 LSTM 在计算上通常更昂贵。

## 应用场景

1. **GRU**：通常在需要更快训练和更少参数的应用场景中表现得更好。

2. **LSTM**：在需要捕捉更复杂的长期依赖和有大量数据可用的应用中，LSTM 通常更为有效。

尽管 GRU 和 LSTM 在很多方面有所不同，但它们都是为了解决传统 RNN 在处理长序列时面临的问题。选择使用哪一种通常取决于具体的应用需求和可用资源。

## Attention Mechanism

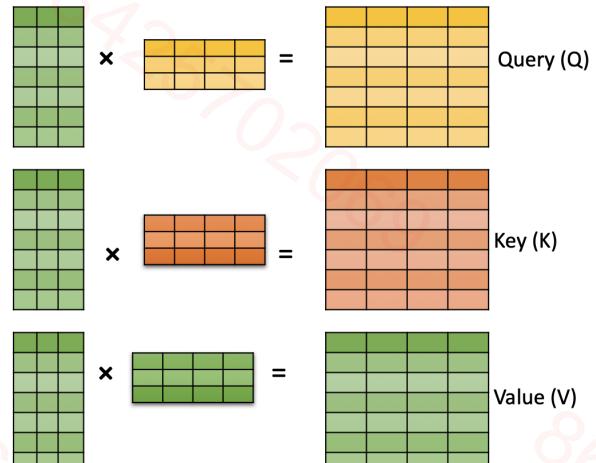
### Self-attention in transformers

Self-attention:

Identify and attend to most important features in input

Step 1: Encode position information

Step 2: Extract query (Q), key (K), and value (V)



Credit: Slide adapted from MIT 6.S191 Recurrent Neural Networks and Transformers

35

## Self-attention in transformers

Self-attention:

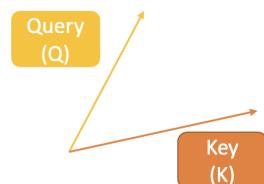
Identify and attend to most important features in input

Step 1: Encode position information

Step 2: Extract query (Q), key (K), and value (V)

Step 3: Compute attention weighting

Step 4: Extract features with high attention



$$\text{Softmax} \left\{ \frac{\mathbf{Q} \cdot \mathbf{K}^T}{\text{Scaling}} \right\} \times \mathbf{V} = \mathbf{A} (\mathbf{Q}, \mathbf{K}, \mathbf{V})$$



Credit: Slide adapted from MIT 6.S191 Recurrent Neural Networks and Transformers

38

自注意力机制是一种在序列数据处理中广泛使用的技术，特别是在自然语言处理（NLP）任务中，如机器翻译、文本生成等。它允许模型在处理一个元素（如一个词）时，查看输入序列中的其他元素，并据此对当前元素进行加权。

## 基本思想

自注意力机制的核心思想是计算序列中每一个元素与其他所有元素之间的相互关系（或“注意力”）。这种关系通常是通过权重来表示的，权重是通过一系列可学习的参数来计算得出的。

## Self-attention in transformers

➤ Understanding self-attention with Search

- Query (Q)
- Key (K)
- Value (V)

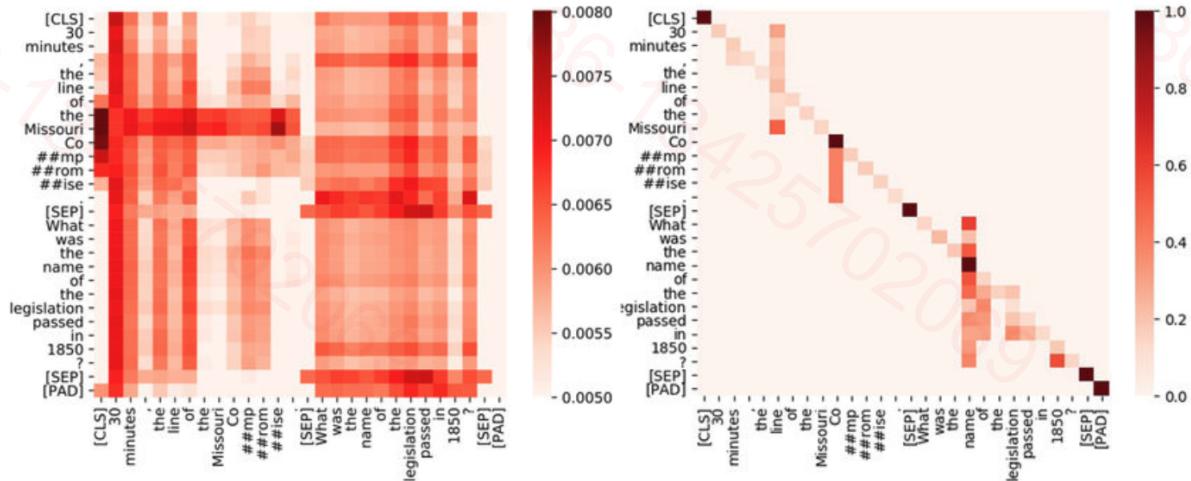
The screenshot shows a YouTube search results page for "natural language processing with deep learning". Three video thumbnails are highlighted to illustrate the components of self-attention:

- Query (Q):** A video titled "4 Hours Chopin for Studying, Concentration & Relaxation" by HALIDONMUSIC.
- Key 1 (K1):** A video titled "Stanford CS224N: Natural Language Processing with Deep Learning | Winter 2021" by Stanford Online.
- Key 2 (K2):** A video titled "Jupyter Notebook Tutorial for Beginners with Python" by Dave Gray.
- Value (V):** A video titled "Stanford CS224N NLP with Deep Learning | Winter 2021 | Lecture 1 - Intro & Word Vectors" by Stanford CS224N NLP with Deep Learning.
- Key 3 (K3):** A video titled "Stanford CS224N NLP with Deep Learning | Winter 2021 | Lecture 2 - Neural Classifiers" by Stanford CS224N NLP with Deep Learning.



Credit: MIT 6.S191 Recurrent Neural Networks and Transformers

33



Visualization of the vanilla BERT attention (left) and syntax-guided self-attention (right). Weights of attention are selected from first head of the last attention layer. For the syntax-guided self-attention, the columns with weights represent the SDOI for each word in the row. For example, the SDOI of passed contains {name, of, legislation, passed}. Weights are normalized by SoftMax for each row.

[https://www.researchgate.net/figure/Visualization-of-the-vanilla-BERT-attention-left-and-syntax-guided-self-attention\\_fig3\\_339301634](https://www.researchgate.net/figure/Visualization-of-the-vanilla-BERT-attention-left-and-syntax-guided-self-attention_fig3_339301634)

## 数学表示

给定一个输入序列  $X = [x_1, x_2, \dots, x_n]$ , 自注意力机制通常包括以下几个步骤：

**计算 Query (Q) 、 Key (K) 和 Value (V) :**

$$Q = XW_Q, \quad K = XW_K, \quad V = XW_V$$

其中  $W_Q, W_K, W_V$  是可学习的权重矩阵。

**计算注意力分数 :**

$$\text{Score} = \frac{QK^T}{\sqrt{d_k}}$$

其中  $d_k$  是 Key 的维度。

**应用 Softmax 函数 :**

$$\text{Attention Weights} = \text{Softmax}(\text{Score})$$

**计算输出序列 :**

$$\text{Output} = \text{Attention Weights} \times V$$

## 优点

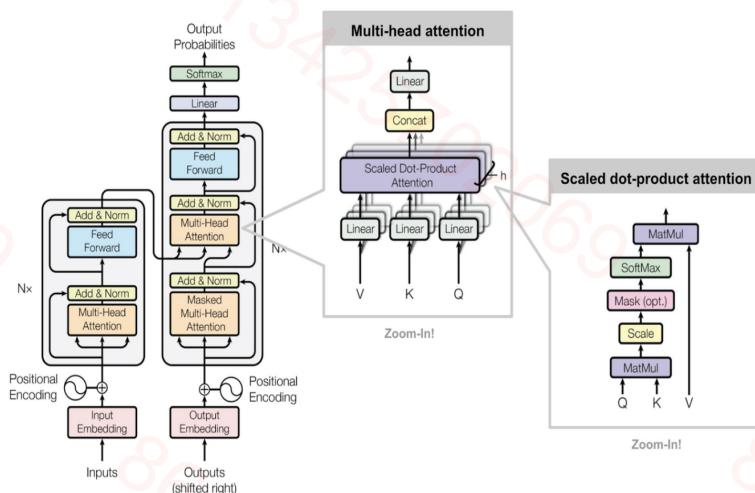
1. **捕捉长距离依赖**: 自注意力机制允许模型在处理一个元素时考虑整个序列，因此能够有效地捕捉长距离的依赖关系。
2. **并行计算**: 自注意力机制可以同时处理整个序列，而不是像 RNN 那样需要逐一处理，这大大加速了计算。
3. **可解释性**: 通过观察注意力权重，我们可以直观地了解模型在做决策时重视了哪些部分。
4. **灵活性**: 自注意力机制可以很容易地应用于各种序列到序列的任务。

由于这些优点，自注意力机制已经成为了很多现代 NLP 模型，如 Transformer 和 BERT，的核心组成部分。

## Self-attention in transformers

### Self-attention:

- Step 1: Encode position information
- Step 2: Extract query (Q), key (K), and value (V)
- Step 3: Compute attention weighting
- Step 4: Extract features with high attention



Credit: Slide adapted from MIT 6.S191 Recurrent Neural Networks and Transformers

39

## Supervised & Reinforcement Learning

强化学习（Reinforcement Learning, RL）和监督学习（Supervised Learning, SL）是机器学习中的两种主要类型，它们有不同的学习方式和应用场景。

## Supervised Learning Recap

## Supervised Learning

Recall: Supervised Learning

- We have a *training* set and a *test* set, each consisting of a set of examples.  
For each example, a number of input attributes and a target attribute are specified.
- The aim is to predict the target attribute, based on the input attributes.
- Various learning paradigms are available:
  - Decision Trees
  - Neural Networks
  - SVM
  - .. others ..

3



监督学习是这样一种机器学习方法：算法从标记的训练数据中学习，每个训练样本都有一个输入和一个期望的输出标签。算法做的是学习映射函数，将输入映射到输出，目的是在给定新的未见过的数据时能够预测出正确的输出。典型的监督学习任务包括分类和回归。

例如，假设我们有一组电子邮件数据，我们需要训练一个模型来识别哪些邮件是垃圾邮件。在监督学习中，我们会有一个由人类标记的数据集，其中包括邮件的内容（输入特征）和标签（垃圾邮件或非垃圾邮件）。模型的任务是学会根据输入特征预测邮件类别。

## Reward as supervisor

### Learning of Actions

Supervised Learning can also be used to learn Actions, if we construct a training set of situation-action pairs (called Behavioral Cloning).

However, there are many applications for which it is difficult, inappropriate, or even impossible to provide a “training set”

- optimal control
  - mobile robots, pole balancing, flying a helicopter
- resource allocation
  - job shop scheduling, mobile phone channel allocation
- mix of allocation and control
  - elevator control, backgammon

4



强化学习则不同，它关注的是如何基于环境提供的奖励来采取行动。在强化学习中，没有指导的标签，也没有确切的指示应该采取什么行动。相反，一个称为智能体的实体必须通过尝试和错误来学习策略，以在长期内最大化其累积奖励。RL涉及到智能体、环境、状态、动作和奖励这些概念。

以玩游戏为例，强化学习的智能体可以是游戏中的一个角色。智能体不会被告知具体的移动策略，而是通过游戏过程中获得的分数（奖励）来学习如何移动。如果一个特定的行动（如跳过障碍）导致奖励增加，智能体会学习在类似情况下重复这个行动。

总结一下：

- **监督学习**有明确的输入和输出标签，模型通过最小化预测和实际标签之间的差异来训练。
- **强化学习**通过智能体与环境的交互来学习，目的是最大化累积奖励，而不是最小化错误。

两者之间的主要区别在于它们的学习信号和数据格式，以及它们的目标：监督学习旨在准确地预测标签，而强化学习旨在发现能够获得最大奖励的策略。

## RL Framework

### Reinforcement Learning Framework

- An agent interacts with its environment.
- There is a set  $\mathcal{S}$  of *states* and a set  $\mathcal{A}$  of *actions*.
- At each time step  $t$ , the agent is in some state  $s_t$ .  
It must choose an action  $a_t$ , whereupon it goes into state  
 $s_{t+1} = \delta(s_t, a_t)$  and receives reward  $r_t = \mathcal{R}(s_t, a_t)$
- Agent has a *policy*  $\pi : \mathcal{S} \rightarrow \mathcal{A}$ . We aim to find an *optimal* policy  $\pi^*$  which maximizes the cumulative reward.
- In general,  $\delta$ ,  $\mathcal{R}$  and  $\pi$  can be multi-valued, with a random element, in which case we write them as probability distributions

$$\delta(s_{t+1} = s | s_t, a_t) \quad \mathcal{R}(r_t = r | s_t, a_t) \quad \pi(a_t = a | s_t)$$

在强化学习（Reinforcement Learning, RL）框架中，以下是一些核心概念：

1. **智能体（Agent）**：在RL中，智能体是执行动作的实体，它的目的是通过与环境的交互来学习最佳的行为策略。
2. **环境（Environment）**：环境是智能体所处并与之交互的外部世界。它定义了智能体可能遇到的状态和动作的结果。
3. **状态（State）**：状态是环境的一个描述，它可以是完整的或部分的，反映了环境在某一时刻的情况。状态集 $S$ 包括了所有可能的状态。

4. 动作 (Action) : 在任何给定状态下, 智能体可以选择执行的动作。动作集  $A$  包括了所有可能的动作。
5. 策略 (Policy) : 策略是从状态到动作的映射, 指导智能体在特定状态下应该采取什么动作。策略  $\pi$  可以是确定性的, 也可以是随机性的。
6. 回报 (Return) : 回报通常指的是一个时间序列内智能体获得的累积奖励。智能体的目标是最大化其在长期内的回报。

在一个典型的强化学习问题中, 智能体不断地通过与环境的交互来探索和利用知识, 以此来改进其策略。强化学习算法的目标是找到最优策略, 即能够在长期内获得最大累积奖励的策略。

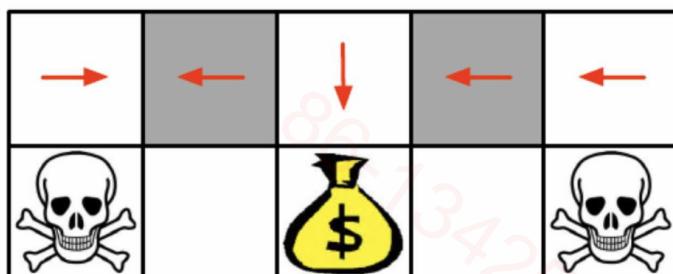
## Example

举个例子, 假设智能体是一个机器人, 环境是一个迷宫。状态可以表示为机器人在迷宫中的位置, 动作可能是“向北移动”或“向南移动”。奖励函数可以根据机器人是否接近出口来设定, 例如, 每当机器人向出口移动时获得正奖励, 而远离出口时获得负奖励。策略则指导机器人在迷宫的任何位置选择动作。智能体的目标是找到一条路径, 使得从入口到出口的累积奖励最大化。在这个过程中, 智能体可能需要尝试不同的路径(探索), 并记住哪些路径能获得更高的奖励(利用)。

## Policy

如果给定了一些state, agent应该往哪里走?

### Probabilistic Policies



There are some environments in which any deterministic agent will perform very poorly, and the optimal (reactive) policy must be stochastic (i.e. randomized).

In 2-player games like Rock-Paper-Scissors, a random strategy is also required in order to make agent choices unpredictable to the opponent.

我们的目标就是想要找到这么好的policy, 什么是好的policy呢? 这个policy应该有最大的奖励。

## Models of optimality

Is a fast nickel worth a slow dime?

Finite horizon reward	$\sum_{i=0}^{h-1} r_{t+i}$
Infinite discounted reward	$\sum_{i=0}^{\infty} \gamma^i r_{t+i}, \quad 0 \leq \gamma < 1$
Average reward	$\lim_{h \rightarrow \infty} \frac{1}{h} \sum_{i=0}^{h-1} r_{t+i}$

- Finite horizon reward is simple computationally
- Infinite discounted reward is easier for proving theorems
- Average reward is hard to deal with, because can't sensibly choose between small reward soon and large reward very far in the future.

7



在强化学习中，根据任务和应用的不同，我们可以根据预期的回报（reward）来定义不同类型的目标函数。以下是关于有限视界回报（Finite Horizon Reward），无限折扣回报（Infinite Discounted Reward），和平均回报（Average Reward）的解释：

### 1. 有限视界回报（Finite Horizon Reward）：

- 在有限视界回报设置中，智能体在一个固定长度的时间段内最大化其回报。例如，如果视界长度为  $T$ ，那么智能体只关心在时间  $t = 0$  到  $t = T$  内获得的回报。
- 计算上简单，因为它只涉及对一个有限序列的奖励进行累加，这意味着没有长期的依赖或复杂的计算。
- 这种方法的问题在于，它可能不会考虑任务完成后的长期效果，因此可能不适用于需要长期规划的情况。

### 2. 无限折扣回报（Infinite Discounted Reward）：

- 在无限折扣回报中，智能体尝试最大化整个未来的累积折扣回报。这是通过在每个即时回报上应用一个折扣因子  $\gamma$  来实现的，其中  $\gamma$  的取值范围通常在 0 到 1 之间。
- 折扣因子的作用是减少未来奖励的重要性。这意味着随着时间的推移，回报对当前决策的影响越来越小。
- 这种方法适合证明定理，因为折扣因子确保了未来回报的总和是收敛的，这简化了数学分析。

### 3. 平均回报（Average Reward）：

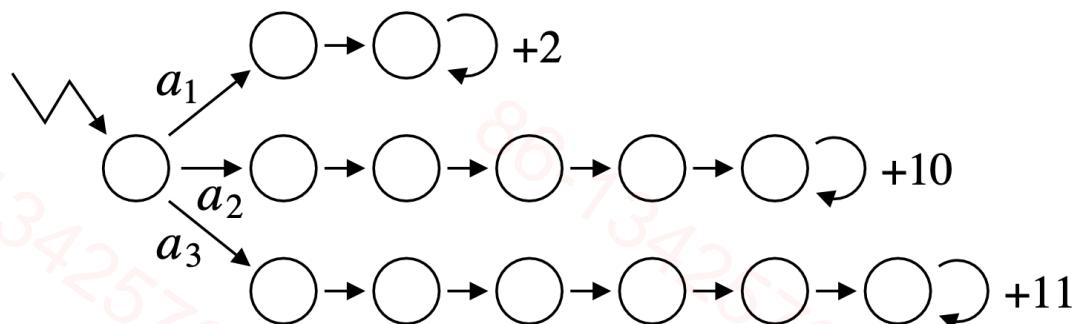
- 平均回报设置下，智能体的目标是最大化其长期平均回报，而不是总和。
- 这种方法可以处理长期稳定的环境，其中智能体的行为不会导致明显的环境变化。

- 在计算和理论分析上更复杂，因为它需要在不同时间点上获得的奖励之间做出权衡。它尤其在处理时间差异极大的奖励时显得更加复杂，因为它难以在短期奖励和可能在很远的未来获得的大奖励之间做出合理的选择。

综上所述，有限视界回报在计算上是简单的，因为我们只考虑有限的时间范围内的回报。无限折扣回报在理论上是易于处理的，因为折扣因子确保了累积回报的收敛性，这使得相关的数学分析变得可行。而平均回报则在计算和理论分析上都比较复杂，特别是在需要考虑长期效果的时候。

## Example

### Comparing Models of Optimality



- Finite horizon,  $k = 4 \rightarrow a_1$  is preferred
- Infinite horizon,  $\gamma = 0.9 \rightarrow a_2$  is preferred
- Average reward  $\rightarrow a_3$  is preferred

8



根据定义的回报不同，最后模型选择的最佳policy也不同。简单来说，如果视野很近，会收敛到局部最优解的policy，但是计算快。如果视野很远，计算效率低，但是能收敛到全局最优解的policy。

## Value Function Learning

## Value Function Learning

Every policy  $\pi$  determines a *Value Function*  $V^\pi : \mathcal{S} \rightarrow \mathbb{R}$  where  $V^\pi(s)$  is the average discounted reward received by an agent who begins in state  $s$  and chooses its actions according to policy  $\pi$ .

If  $\pi = \pi^*$  is optimal, then  $V^*(s) = V^{\pi^*}(s)$  is the maximum (expected) discounted reward obtainable from state  $s$ . Learning this optimal value function can help to determine the optimal strategy.

The agent retains its own *estimate*  $V()$  of the “true” value function  $V^*()$ .

The aim of Value Function Learning is generally to start with a random  $V$  and then iteratively improve it so that it more closely approximates  $V^*$ .

This process is sometimes called “Bootstrapping”.

10



价值函数学习 (Value Function Learning) 是强化学习中的一个核心概念。它的目标是评估一个策略的好坏，即计算在该策略下，从每个状态开始，智能体能获得的期望回报的价值。

每个策略  $\pi$  都对应一个价值函数  $V^\pi$ 。这个函数将每个状态  $s$  映射到一个实数，该实数代表了如果智能体从状态  $s$  开始，按照策略  $\pi$  选择动作，它可以期望获得的折扣回报。

价值函数  $V^\pi(s)$  的定义为：

$$V^\pi(s) = \mathbb{E} \left[ \sum_{t=0}^{\infty} \gamma^t R_t | S_0 = s, \pi \right]$$

这里  $R_t$  是在时间  $t$  获得的回报， $\gamma$  是折扣因子，它确定了未来回报相对于即时回报的重要性。期望值是根据策略  $\pi$  和初始状态  $s$  计算的。

如果  $\pi = \pi^*$  是最优的，那么对应的价值函数  $V^*(s)$  是从状态  $s$  出发可以获得的最大期望折扣回报。

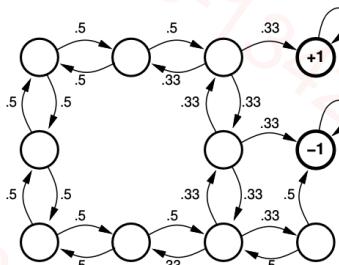
在实际应用中，智能体并不知道真正的价值函数  $V^*$ ，因此它需要通过与环境的交互来估计这个函数。智能体保留了其对“真实”价值函数  $V^*$  的估计  $V$ 。价值函数学习的目标通常是从一个随机初始化的  $V$  开始，然后通过迭代改进，使其更接近  $V^*$ 。

这个迭代过程有时被称为“自举” (Bootstrapping)。在自举中，智能体使用其当前估计的价值函数来更新其对未来状态价值的预测。这可以通过多种算法实现，如动态规划、时序差分学习 (TD Learning)，或者蒙特卡洛方法。

## Value Function

			$+1$
			$-1$
START			

(a)



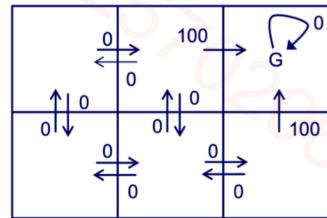
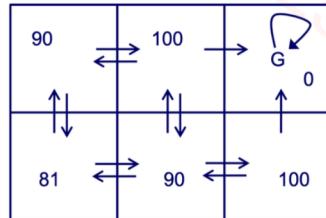
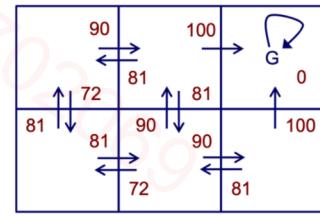
(b)

			$+1$
			$-1$
1	-0.2911	-0.0380	-0.5443

(c)

This is the Value Function  $V^\pi$  where  $\pi$  is the policy of choosing between available actions uniformly randomly.

11

 $r(s, a)$  (immediate reward) values $V^*(s)$  values $Q(s, a)$  values

## Action Selection

## Exploration / Exploitation Tradeoff

Most of the time we should choose what we think is the best action.

However, in order to ensure convergence to the optimal strategy, we must occasionally choose something different from our preferred action, e.g.

- choose a random action 5% of the time, or
- use Softmax (Boltzmann distribution) to choose the next action:

$$P(a) = \frac{e^{\mathcal{R}(a)/T}}{\sum_{b \in \mathcal{A}} e^{\mathcal{R}(b)/T}}$$

13



探索/利用权衡 (Exploration/Exploitation Tradeoff) 是强化学习中的一个核心问题，涉及如何在尝试新的、可能更好的行为 (探索) 与基于现有知识采取已知的最佳行为 (利用) 之间做出平衡。

**利用 (Exploitation) :**

- 当智能体选择利用时，它会选择当前估计的最佳行动，即根据已有的知识和经验，它认为能够带来最大回报的行动。
- 这种方式可以确保智能体获得立即的高回报，但存在一个风险：如果智能体的知识有限或者不完全准确，它可能会错过更好的行动方案。

**探索 (Exploration) :**

- 探索意味着尝试不确定的行动，即使这些行动在过去看起来不是最佳的。
- 这可以帮助智能体收集更多信息，纠正其对环境的了解，发现更好的行动策略。
- 然而，过多的探索可能会导致短期内回报的损失，因为智能体可能会选择显然不是最佳的行动。

为了解决探索与利用之间的权衡，有几种策略：

**1.  $\epsilon$ -贪婪策略 ( $\epsilon$ -greedy strategy) :**

- 这种方法中，大部分时间（例如95%的情况）智能体会选择当前看起来最佳的行动（利用），但有一小部分时间（例如5%的情况）会随机选择一个行动（探索）。
- 参数  $\epsilon$  是一个小的正数，表示选择随机行动的概率。

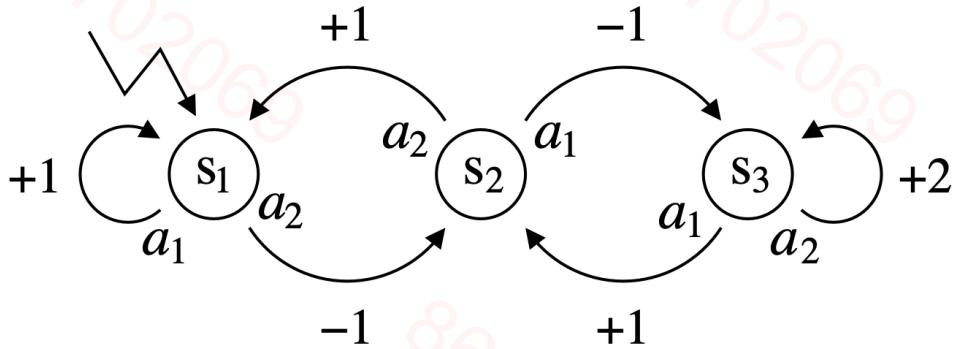
**2. Softmax选择：**

- 在这种方法中，智能体选择每个可能行动的概率与该行动的价值函数的指数成比例，通常使用Boltzmann分布（或称为Softmax分布）。

# TD-Learning

TD Learning解决的一个核心问题是delayed reinforcement。

## Delayed Reinforcement



We may need to take several actions before we can get the good reward.

15



时序差分学习（Temporal Difference Learning，简称TD Learning）是一种强化学习方法，它结合了蒙特卡洛方法和动态规划的思想。TD学习解决的核心问题是延迟强化（Delayed Reinforcement）。

在很多情况下，智能体的行动可能不会立即产生最终的回报。例如，在棋类游戏中，一个看似平常的走步可能会在多步之后导致胜利或失败。这种情况下，智能体需要能够关联当前的行为和未来可能获得的回报，这正是延迟强化问题。

TD学习如何处理这个问题：

- **学习估计值**：在没有最终结果的情况下，TD学习可以在每一步更新状态的价值估计。这允许智能体在完成整个序列之前学习，从而实现从经验中学习。
- **自举（Bootstrapping）**：TD学习使用当前的估计来更新价值函数，这是一种自举方法。即使在最终结果未知的情况下，智能体也能根据当前的价值估计和接收到的即时奖励来更新其价值估计。
- **TD误差**：TD学习通过TD误差来进行更新，TD误差是当前观察到的奖励加上下一个状态的折扣价值估计与当前状态价值估计之间的差异。这个差异反映了预测与实际结果之间的不一致，智能体通过减少这种不一致来学习。

通过不断地进行这种基于TD误差的更新，智能体能够在没有知道最终结果的情况下逐渐逼近最优的价值函数。这样，即使在面对延迟强化时，智能体也能够有效地学习，并逐步改进其策略以获得更好的长期回报。

## Temporal Difference Learning

Let's first assume that  $\mathcal{R}$  and  $\delta$  are deterministic. Then the (true) value  $V^*(s)$  of the current state  $s$  should be equal to the immediate reward plus the discounted value of the next state

$$V^*(s) = \mathcal{R}(s, a) + \gamma V^*(\delta(s, a))$$

We can turn this into an update rule for the estimated value, i.e.

$$V(s_t) \leftarrow r_t + \gamma V(s_{t+1})$$

If  $\mathcal{R}$  and  $\delta$  are stochastic (multi-valued), it is not safe to simply replace  $V(s)$  with the expression on the right hand side. Instead, we move its value fractionally in this direction, proportional to a learning rate  $\eta$

$$V(s_t) \leftarrow V(s_t) + \eta [r_t + \gamma V(s_{t+1}) - V(s_t)]$$

16



时序差分学习 (TD Learning) 是一种典型的自举方法，它在不需要完整的环境模型的情况下，允许智能体从部分经验中学习。一个基本的时序差分学习算法，如TD(0)，会在每一步更新价值函数的估计，使用公式：

$$V(s_t) \leftarrow V(s_t) + \alpha [R_{t+1} + \gamma V(s_{t+1}) - V(s_t)]$$

这里 $\alpha$ 是学习率， $R_{t+1}$ 是从状态 $s_t$ 转移到 $s_{t+1}$ 获得的即时奖励， $\gamma V(s_{t+1})$ 是下一个状态的折扣价值， $V(s_t)$ 是当前状态的价值估计。这个更新规则反映了当前估计和实际观察之间的“差距”或“误差”。通过这样的迭代过程，智能体的价值函数估计会逐渐收敛到真实的价值函数 $V^*$ ，从而使智能体能够确定最优策略。

## Q-Learning

## Q-Learning

Q-Learning is similar to TD-Learning except that we use a function  $Q^\pi : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$  which depends on a state, action pair instead of just a state.

For any policy  $\pi$  the Q-Function  $Q^\pi(s, a)$  is the average discounted reward received by an agent who begins in state  $s$ , first performs action  $a$  and then follows policy  $\pi$  for all subsequent timesteps.

If  $\pi = \pi^*$  is optimal, then  $Q^*(s, a) = Q^{\pi^*}(s, a)$  is the maximum (expected) discounted reward obtainable from  $s$ , if the agent is forced to take action  $a$  in the first timestep but can act optimally thereafter.

The agent retains its own (initially, random) estimate  $Q()$  and iteratively improves this estimate to more closely approximate the “true” function  $Q^*()$ .

17



## Q-Learning

For a deterministic environment,  $\pi^*$ ,  $Q^*$  and  $V^*$  are related by

$$\begin{aligned}\pi^*(s) &= \operatorname{argmax}_a Q^*(s, a) \\ Q^*(s, a) &= \mathcal{R}(s, a) + \gamma V^*(\delta(s, a)) \\ V^*(s) &= \max_b Q^*(s, b)\end{aligned}$$

So

$$Q^*(s, a) = \mathcal{R}(s, a) + \gamma \max_b Q^*(\delta(s, a), b)$$

This allows us to iteratively approximate  $Q$  by

$$Q(s_t, a_t) \leftarrow r_t + \gamma \max_b Q(s_{t+1}, b)$$

If the environment is stochastic, we instead write

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \eta [r_t + \gamma \max_b Q(s_{t+1}, b) - Q(s_t, a_t)]$$

18



Q学习 (Q-learning) 是一种无模型的强化学习算法，它可用于求解马尔可夫决策过程 (Markov Decision Processes, MDPs)。Q学习的目标是学习一个策略，告诉智能体在给定的状态下应该采取什么动作以最大化未来的奖励。它是一种基于价值的方法，即它估计每一对状态-动作 (state-action pair) 的价值。

Q学习的核心是Q函数，即动作价值函数  $Q : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ ，其中  $\mathcal{S}$  是状态空间， $\mathcal{A}$  是动作空间。对于每一对状态  $s$  和动作  $a$ ， $Q(s, a)$  表示在状态  $s$  下采取动作  $a$  并且之后遵循最佳策略所期望获得的累积折扣回报。

Q学习算法的更新规则是基于贝尔曼方程，具体为：

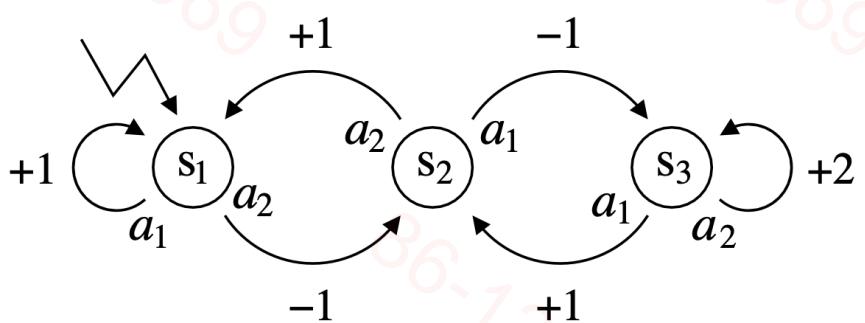
$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t)]$$

其中：

- $Q(s_t, a_t)$  是当前状态-动作对的估计价值。
- $\alpha$  是学习率 ( $0 < \alpha \leq 1$ )，它决定了新获得的信息会在多大程度上影响当前的Q值。
- $r_{t+1}$  是在采取动作  $a_t$  后从状态  $s_t$  转移到状态  $s_{t+1}$  获得的即时奖励。
- $\gamma$  是折扣因子 ( $0 \leq \gamma < 1$ )，它决定了未来奖励的当前价值。
- $\max_{a'} Q(s_{t+1}, a')$  是下一个状态  $s_{t+1}$  中所有可能动作的最大Q值，代表了最优未来行为的估计价值。

Q学习的特点是它可以在策略外进行学习 (off-policy)，这意味着智能体可以从由另一策略生成的数据中学习，或者在探索的同时学习最优策略。它不需要模型来告诉它状态转移或奖励的情况，智能体只需通过尝试不同的动作并观察结果来学习。通过不断更新Q值，智能体逐渐构建起一个Q表，该Q表为每一对状态-动作提供了价值估计。最终，智能体可以仅根据Q表来决定在每个状态下采取的动作，选择具有最高Q值的动作。随着时间的推移，如果智能体有足够的探索和学习率适当选择，Q学习保证了学习的收敛到最优的Q值，从而导致最优的行为策略。

## Q-Learning Example



Exercise:

1. compute  $V^\pi(s_3)$  if  $\pi(s_3) = a_2$  and  $\gamma = 0.9$
2. compute  $\pi^*$ ,  $V^*$  and  $Q^*$  for this environment (if  $\gamma = 0.9$ )

## Limitation

## Limitations of Theoretical Results

- Delayed reinforcement
  - reward resulting from an action may not be received until several time steps later, which also slows down the learning
- Search space must be finite
  - convergence is slow if the search space is large
  - relies on visiting every state infinitely often
- For “real world” problems, we can’t rely on a lookup table
  - need to have some kind of *generalisation* (e.g. TD-Gammon)

21



强化学习的理论成果为我们提供了许多强有力的工具和方法，但是这些理论结果在实际应用中也存在一些限制。下面是一些主要的限制：

### 1. 延迟强化 (Delayed Reinforcement) :

- 在现实世界问题中，一个动作的回报可能不会立即显现，有时需要多个时间步之后才能收到。这会使学习过程变得复杂和缓慢，因为智能体必须学会将延迟的回报与先前的动作相关联。
- 为了有效地学习这种长期依赖，算法可能需要大量的样本和时间来正确评估一个动作的长期效益。

### 2. 搜索空间必须是有限的：

- 多数理论结果是建立在状态空间和动作空间都是有限的假设上的。如果搜索空间很大，那么算法的收敛速度会非常慢，因为理论上需要在每一个状态上进行无限次的探索才能保证找到最优策略。
- 当面对高维度或连续的状态空间时，传统的表格型方法（如Q学习）就变得不切实际，因为它们需要对每个可能的状态-动作对存储和更新一个值。

### 3. 现实世界问题的复杂性：

- 现实世界的问题往往是高度复杂和动态变化的，不能仅靠一个简单的查找表来解决所有问题。
- 例如，在处理视觉输入或进行自然语言处理时，状态的可能性几乎是无限的，这就需要智能体能够从有限的经验中泛化到新的、未见过的状态。

### 4. 泛化的需求：

- 为了克服查找表在处理复杂或连续空间时的局限性，需要使用泛化技术。这通常涉及到使用函数逼近器（如神经网络）来预测价值函数或策略。

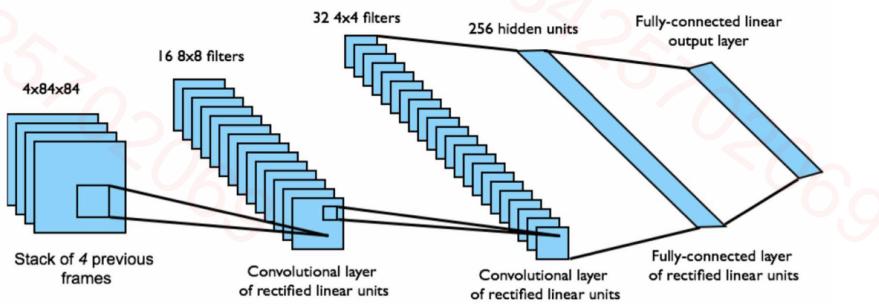
- TD-Gammon是一个著名的例子，它使用了神经网络来泛化在黑白棋游戏中的价值函数。它不依赖于具体的状态查找表，而是通过训练神经网络从输入的棋盘配置中提取特征并预测最佳动作。

## Deep Q-Network

深度Q学习（Deep Q-Learning, DQN）是一种结合了深度学习和Q学习的强化学习方法。它由DeepMind公司提出，并在多款Atari 2600游戏上取得了突破性的性能。DQN能够直接从原始像素输入学习动作的价值函数  $Q(s, a)$ 。

### Deep Q-Learning for Atari Games

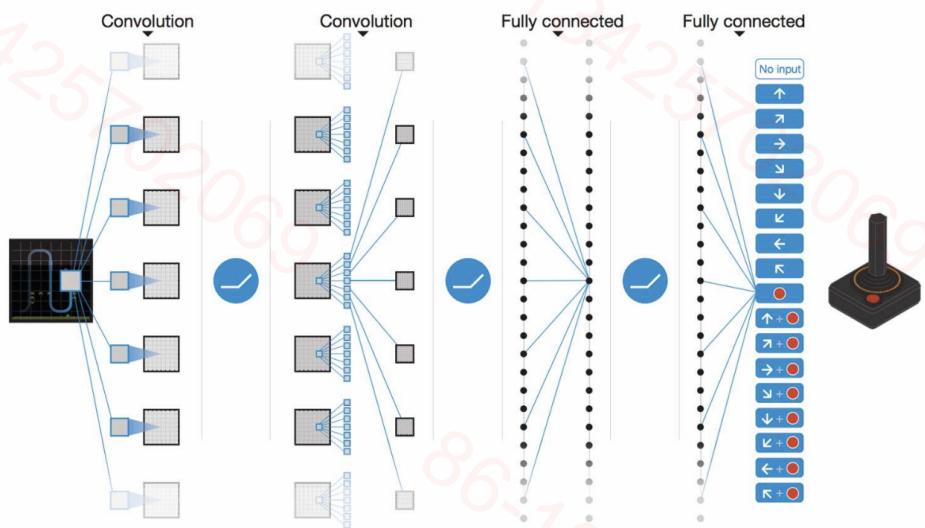
- end-to-end learning of values  $Q(s, a)$  from pixels  $s$
- input state  $s$  is stack of raw pixels from last 4 frames
  - 8-bit RGB images,  $210 \times 160$  pixels
- output is  $Q(s, a)$  for 18 joystick/button positions
- reward is change in score for that timestep



10



## Deep Q-Network



11



在Atari游戏中的应用特点如下：

1. 端到端学习 (End-to-End Learning) :

- DQN直接从观察到的像素学习，无需任何手工特征工程。这意味着从最初的游戏画面像素到最终的动作选择，所有的步骤都是通过网络自动学习的。

2. 输入状态 (Input State) :

- 输入状态 $s$ 是由最后四帧的像素堆叠起来的。这样做是为了给网络提供足够的信息来捕捉动态场景中的运动，因为一帧的静态像素图像不包含物体的速度和方向信息。
- 这些图像通常是8位RGB图像，分辨率为 $210 \times 160$ 像素。在实际应用中，为了减少计算需求，这些图像会被预处理，包括灰度化、裁剪和缩放。

3. 输出 (Output) :

- 网络的输出是针对18种不同的游戏操纵杆/按钮位置的 $Q(s, a)$ 值。这些动作代表了玩家可以在游戏中执行的所有可能动作。

4. 奖励 (Reward) :

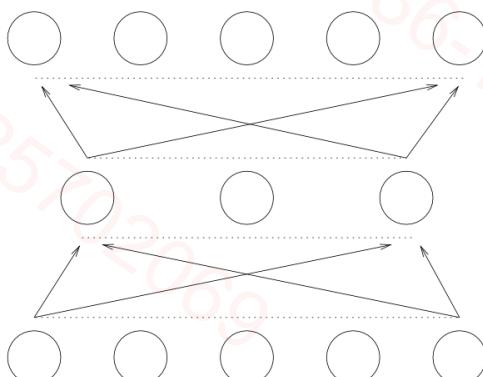
- 在每个时间步，奖励是基于该步骤的分数变化。如果玩家得分增加，奖励为正；得分减少，奖励为负。这样的设置直接将游戏得分与智能体要最大化的目标联系起来。

## AutoEncoder

### Encoder

---

#### Recall: Encoder Networks



Inputs	Outputs
10000	10000
01000	01000
00100	00100
00010	00010
00001	00001

- identity mapping through a bottleneck
- also called N-M-N task
- used to investigate hidden unit representations

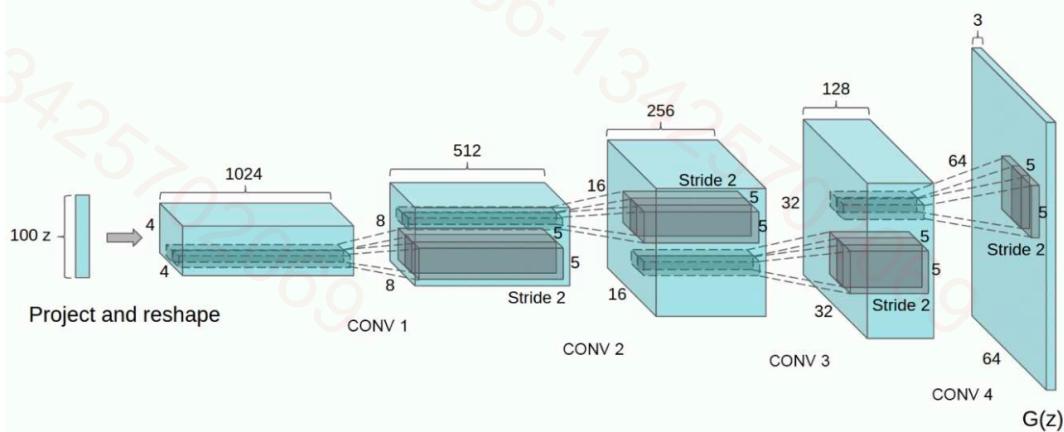
编码器网络（Encoder Network）是一种特殊类型的神经网络，它通过一个瓶颈层（bottleneck layer）来实现身份映射（identity mapping）。这个瓶颈层充当数据压缩的关键部分，其目的是捕捉输入数据的最重要特征。

在“N-M-N”任务中，编码器网络首先将高维输入数据（表示为N维）压缩到一个较低维度的表示（表示为M维），这就是瓶颈层。接着，另一个网络部分（称为解码器）尝试从这个压缩的表示中重建原始数据，恢复到N维。这个过程有助于提取输入数据中最重要的特征和结构。

编码器网络的一个关键应用是在隐藏单元表示的研究中。通过分析在瓶颈层中的数据表示，研究者可以了解网络如何理解和处理复杂数据。例如，在图像处理中，编码器可能会学习识别重要的视觉特征，而在文本处理中，则可能学习捕捉关键的语义模式。

## Decoder

### De-Convolutional Encoder for Images



Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks (Radford et al., 2016)

6



“De-Convolutional Encoder for Images”通常指的是一种在图像处理领域中用于特征提取和重建的神经网络结构，它结合了编码器（Encoder）和反卷积（Deconvolution）层。

### 编码器 (Encoder)

- 作用**：编码器部分负责从输入图像中提取特征。它通常由多个卷积层组成，每个卷积层都会对输入图像进行更加抽象的表示。
- 卷积过程**：在卷积过程中，输入图像通过一系列滤波器（或卷积核）处理，这些滤波器可以捕捉图像的不同特征（如边缘、纹理等）。
- 降维**：随着数据通过编码器的每个层次，其空间维度（即图像的宽度和高度）通常会减小，而深度（即特征数量）则增加。

### 反卷积 (Deconvolution)

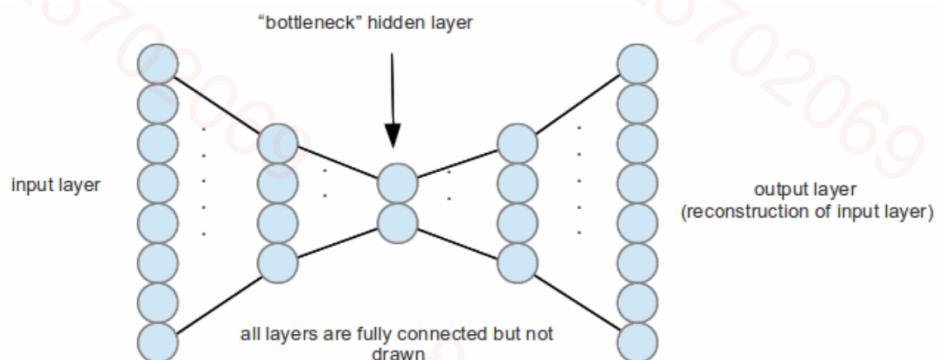
- **作用**：反卷积层，也称为转置卷积层，用于将编码器的输出扩展回原始输入图像的尺寸。
- **过程**：在这个过程中，网络学习如何重建或近似原始输入图像，这通常包括恢复图像的详细特征和纹理。
- **反卷积运算**：这种运算相当于卷积的逆过程，它将较低维度的特征映射扩展回较高维度的空间。

## 应用

- **图像重建**：这种结构在图像重建（如降噪、超分辨率等）中非常有用。
- **特征学习**：它也用于学习图像数据的有效特征表示，尤其在没有标签的数据集上进行无监督学习时。
- **生成模型**：在生成对抗网络（GANs）和变分自编码器（VAEs）中，反卷积编码器被用于生成逼真的图像。

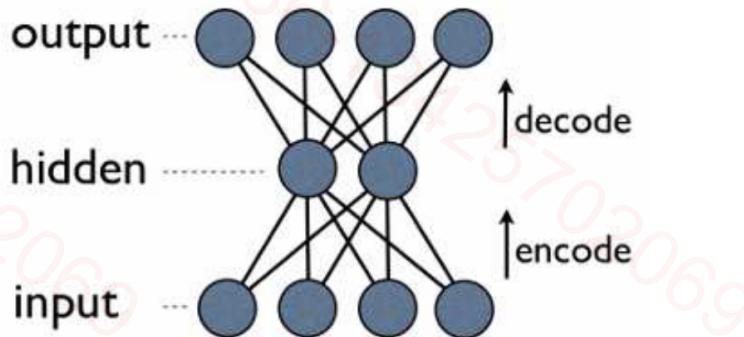
## AutoEncoder

### Autoencoder Networks



- output is trained to reproduce the input as closely as possible
- activations normally pass through a bottleneck, so the network is forced to compress the data in some way
- Autoencoders can be used to generate “fake” items, or to automatically extract abstract features from the input

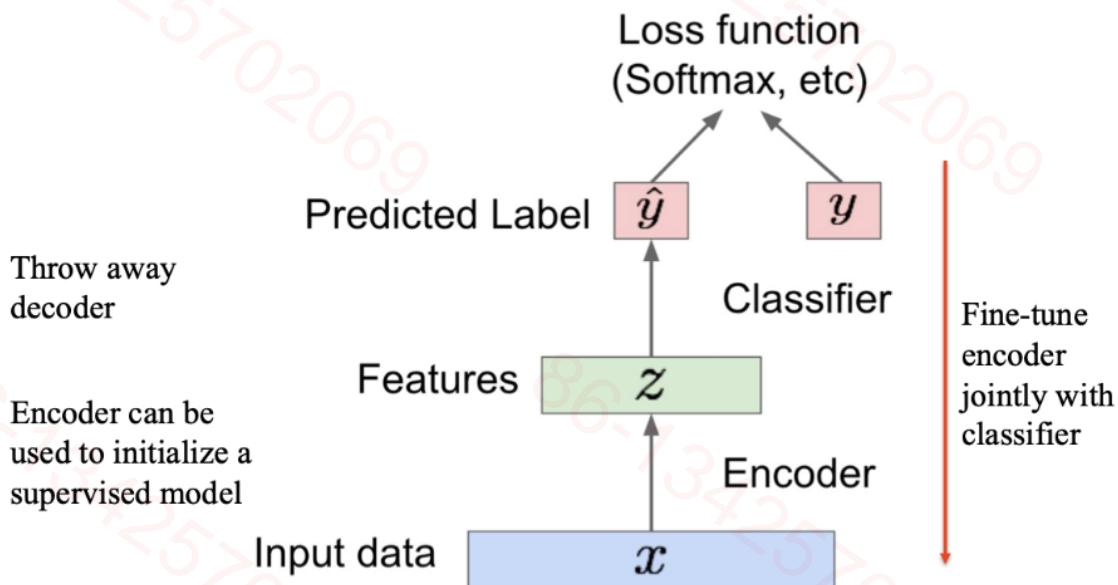
## Autoencoder Networks



If the encoder computes  $z = f(x)$  and the decoder computes  $g(f(x))$  then we aim to minimize some distance function between  $x$  and  $g(f(x))$

$$E = L(x, g(f(x)))$$

5



自编码器（Autoencoder）是一种特殊类型的神经网络，用于无监督学习。它的目标是让输出尽可能地重现输入。自编码器的结构通常包括两部分：编码器（Encoder）和解码器（Decoder）。编码器负责将输入数据转换成一个较低维度的内部表示（称为编码），而解码器则尝试从这个编码中重建原始数据。

关键特征：

- 1. 瓶颈结构：**自编码器的一个关键特点是它的激活通常通过一个瓶颈层（Bottleneck Layer），也就是编码的层。这个瓶颈迫使网络以某种方式压缩数据，因为编码层的维度通常远小于输入层的维度。这种结构迫使网络学习输入数据中最重要和最具信息性的特征。

2. **数据重建**：自编码器的训练目标是最小化重建误差，即原始输入和网络输出之间的差异。这迫使网络捕获输入数据中的关键特征，并学习一种有效的数据表示。
3. **生成“假”数据**：训练好的自编码器可以用于生成数据。例如，通过向编码层注入随机噪声，可以生成与训练数据类似但又略有不同的新数据。
4. **特征提取**：自编码器也常用于自动提取输入数据的抽象特征。这在图像处理、语音识别或任何需要特征降维的领域中都非常有用。

## Pre-train by AE

### Autoencoder as Pretraining

- after an autoencoder is trained, the decoder part can be removed and replaced with, for example, a classification layer
- this new network can then be trained by backpropagation
- the features learned by the autoencoder then serve as initial weights for the supervised learning task

7



自编码器作为预训练（Autoencoder as Pretraining）的方法是一种常用的深度学习技术，特别适用于那些标签数据有限的情况。这种方法的核心思想是利用自编码器在无监督学习环境中学习到的特征，来初始化另一个监督学习任务的神经网络。下面是这个过程的详细解释：

### 1. 自编码器训练

- **初步学习**：首先，一个自编码器被训练来重构输入数据。这个过程不需要标签数据，因为训练目标是让输出尽可能接近输入。
- **特征提取**：在这个阶段，自编码器学习捕捉输入数据的关键特征，这些特征被编码在编码器的输出（即编码层）中。

### 2. 移除解码器部分

- **修改结构**：训练完成后，自编码器的解码器部分被移除。此时，保留下来的编码器部分包含了从原始数据中学到的特征表示。

### 3. 添加新的层

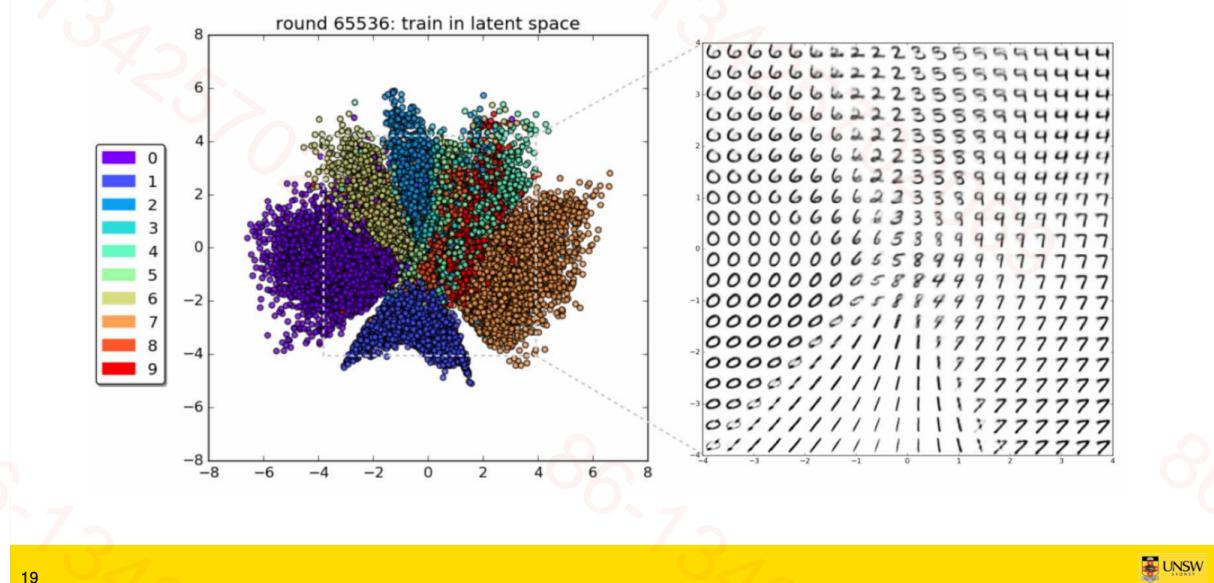
- **任务特定结构**：接着，在编码器的输出端添加新的层，例如分类层，以适应特定的监督学习任务（如图像分类、语音识别等）。

## 4. 进一步训练

- **监督学习**：这个新组合的网络然后使用标签数据进行进一步训练。在这个阶段，通过反向传播（backpropagation）调整网络权重。
- **利用预训练特征**：此时，自编码器预训练的特征作为初始化权重，这可以加速学习过程，并有助于在标签数据有限的情况下提高性能。

# Variational Autoencoder

## Variational Autoencoder Digits



## Variational Autoencoder Digits



1st Epoch



9th Epoch



Original

20



## Variational Autoencoder Faces



21



- Variational Autoencoder produces reasonable results
- tends to produce blurry images
- often end up using only a small number of the dimensions available to  $z$

变分自编码器（Variational Autoencoder，简称VAE）是一种先进的自编码器，它结合了深度学习和概率图模型的理念。VAE 不仅可以像传统自编码器一样学习数据的有效表示，还能生成新的、与训练数据类似的数据点。下面是 VAE 的关键特点和工作原理的简要介绍：

## 关键特点

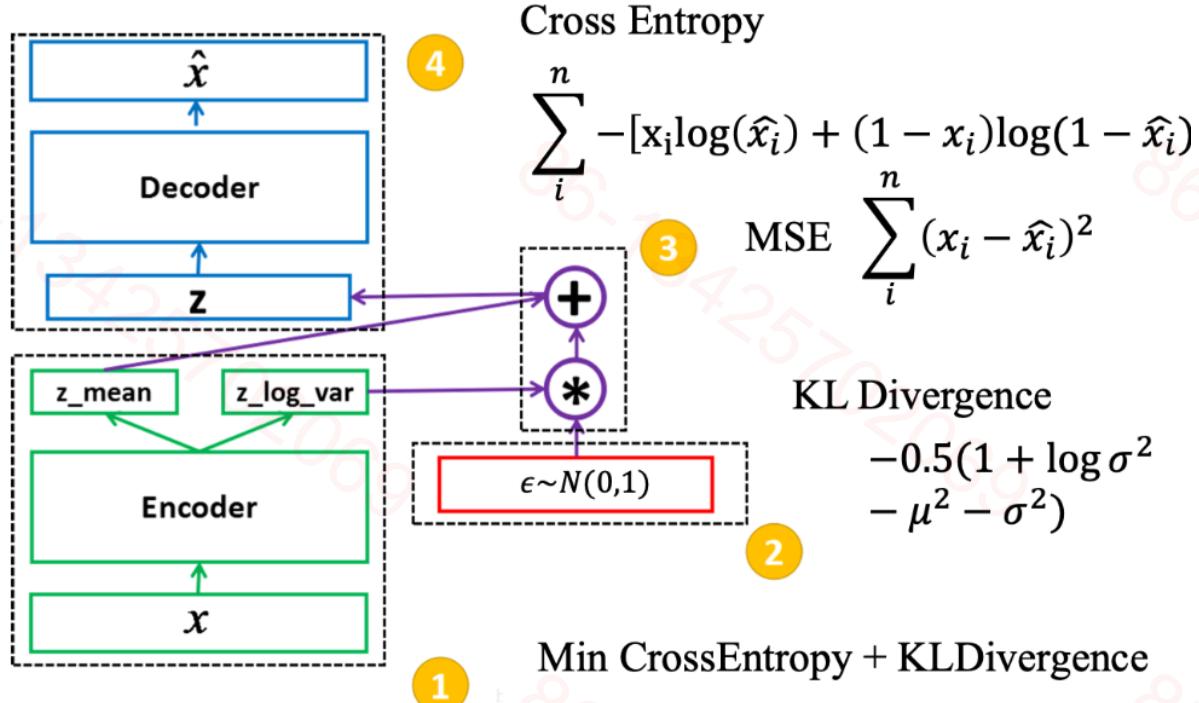
- 概率编码器**：在 VAE 中，编码器不是直接产生一个隐藏表示，而是产生一组参数，描述了输入数据的潜在特征的概率分布（通常是高斯分布）。这意味着对于每个输入数据点，编码器输出一对参数（均值和方差），它们定义了一个概率分布。
- 重参数化技巧**：为了能够通过反向传播进行训练，VAE 使用了所谓的“重参数化技巧”。这个技巧涉及从编码器产生的分布中抽样潜在特征，同时保证这个过程可导，这是通过将随机性外推到网络之外来实现的。
- 变分下界**：VAE 的训练目标是最大化所谓的“变分下界”（ELBO），它是数据对数似然的下界。这个目标包括两部分：一部分是重构损失（即原始数据和重构数据之间的差异），另一部分是正则化项，它鼓励潜在空间的平滑和连续性。

## 工作原理

- 编码阶段**：输入数据被编码成一个概率分布的参数（例如，一个高斯分布的均值和方差）。
- 采样**：从这个分布中采样得到潜在特征。
- 解码阶段**：潜在特征被解码器使用，重构出与原始数据相似的输出。

## Structure

### Variational Autoencoders



## Variational Autoencoder (20.10.3)

Instead of producing a single  $z$  for each  $x^{(i)}$ , the encoder (with parameters  $\phi$ ) can be made to produce a mean  $\mu_{z|x^{(i)}}$  and standard deviation  $\sigma_{z|x^{(i)}}$

This defines a conditional (Gaussian) probability distribution  $q_\phi(z|x^{(i)})$

We then train the system to maximize

$$\mathbf{E}_{z \sim q_\phi(z|x^{(i)})} [\log p_\theta(x^{(i)}|z)] - D_{\text{KL}}(q_\phi(z|x^{(i)}) \| p(z))$$

- the first term enforces that any sample  $z$  drawn from the conditional distribution  $q_\phi(z|x^{(i)})$  should, when fed to the decoder, produce something approximating  $x^{(i)}$
- the second term encourages  $q_\phi(z|x^{(i)})$  to approximate  $p(z)$
- in practice, the distributions  $q_\phi(z|x^{(i)})$  for various  $x^{(i)}$  will occupy complementary regions within the overall distribution  $p(z)$

18



## KL-Divergence Recap

### Entropy and KL-Divergence

- The *entropy* of a distribution  $q()$  is  $H(q) = \int_\theta q(\theta)(-\log q(\theta))d\theta$
- In Information Theory,  $H(q)$  is the amount of information (bits) required to transmit a random sample from distribution  $q()$
- For a Gaussian distribution,  $H(q) = \sum_i \log \sigma_i$
- KL-Divergence  $D_{\text{KL}}(q \| p) = \int_\theta q(\theta)(\log q(\theta) - \log p(\theta))d\theta$ 
  - $D_{\text{KL}}(q \| p)$  is the number of extra bits we need to transmit if we designed a code for  $p()$  but the samples are drawn from  $q()$  instead.
  - If  $p(z)$  is Standard Normal distribution, minimizing  $D_{\text{KL}}(q_\phi(z) \| p(z))$  encourages  $q_\phi()$  to center on zero and spread out to approximate  $p()$ .

16



KL散度 (Kullback-Leibler Divergence)，也称为相对熵，是衡量两个概率分布差异的度量。它是信息论中的一个基本概念，用于描述一个概率分布相对于另一个概率分布的不同程度。在机器学习和统计中，KL散度经常用来衡量模型或数据之间的差异。

## Loss function

你提到的变分自编码器 (VAE) 的损失函数公式是这个模型的核心，它描述了如何训练VAE以学习有效的数据表示和生成新数据。这个公式可以分为两部分进行解释：

## 第一部分：重构损失 (Reconstruction Loss)

- 公式： $\mathbb{E}_{z \sim q_\phi(z|x^{(i)})} [\log p_\theta(x^{(i)}|z)]$
- 解释：
  - 这部分是期望重构损失，它衡量的是当从条件概率分布  $q_\phi(z|x^{(i)})$  中抽样得到潜在变量  $z$  后，解码器重构的数据  $\hat{x}^{(i)}$  与原始数据  $x^{(i)}$  之间的差异。
  - $q_\phi(z|x^{(i)})$  是编码器参数  $\phi$  定义的潜在空间的条件概率分布。这通常是一个高斯分布，由均值  $\mu_{z|x^{(i)}}$  和标准差  $\sigma_{z|x^{(i)}}$  参数化。
  - 这个期望值表明，对于所有可能的潜在变量  $z$  的值，解码器应当能够准确地重构出  $x^{(i)}$

## 第二部分：KL散度 (Kullback-Leibler Divergence)

- 公式： $D_{KL}(q_\phi(z|x^{(i)})||p(z))$
- 解释：
  - 这部分是KL散度，用于衡量编码器学习的潜在分布  $q_\phi(z|x^{(i)})$  与先验分布  $p(z)$  之间的差异。先验分布  $p(z)$  通常是标准正态分布。
  - KL散度部分的目的是确保潜在空间的分布不会偏离过于特定于某些特定的输入数据，从而允许模型更好地泛化和生成新的数据点。
  - 通过最小化这一项，模型被鼓励使得对于各种不同的输入  $x^{(i)}$ ，其相应的潜在表示  $z$  分布  $q_\phi(z|x^{(i)})$  都接近一个标准的高斯分布  $p(z)$ 。

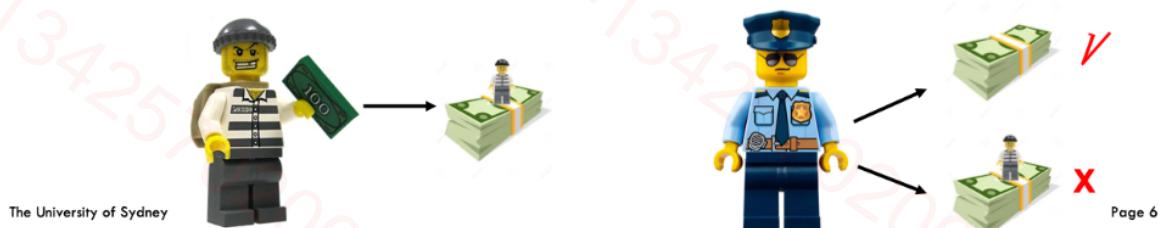
## 总体目标

- VAE的训练目标是最大化重构损失的期望值并最小化KL散度，这可以通过调整编码器和解码器的参数来实现。
- 重构部分确保了数据的有效表示和重建，而KL散度部分保证了潜在空间的良好结构，使模型能够生成新的、有意义的数据样本。

# Generative Adversarial Networks

# Generative Adversarial Networks

- A **counterfeiter-police game** between two components: a generator **G** and a discriminator **D**
- **G**: counterfeiter, trying to fool police with fake currency
- **D**: police, trying to detect the counterfeit currency
- Competition drives both to improve, until counterfeits are *indistinguishable* from genuine currency



- A **two-player game** between two components: a generator **G** and a discriminator **D**
- **G**: try to fool the discriminator by generating real-looking images
- **D**: try to distinguish between real and fake images



生成对抗网络（GAN）是一种深度学习模型，它在两个网络组件之间进行一种类似于“造假者-警察”游戏的竞争。这两个组件分别是生成器（G）和判别器（D）。

## 组件

### 1. 生成器 (G, 相当于造假者) :

- **目标**：生成器的目的是创造尽可能逼真的假数据（比如假图像）。
- **操作**：它学习如何模仿真实数据的分布，从而生成与真实数据几乎无法区分的假数据。
- **挑战**：生成器需要不断提高其生成技巧，以便欺骗判别器。

### 2. 判别器 (D, 相当于警察) :

- **目标**：判别器的任务是区分输入的数据是来自真实数据集还是生成器制造的假数据。
- **操作**：它相当于一个分类器，学习如何识别真假数据。
- **挑战**：判别器需要不断提高其识别能力，以便更准确地识别假数据。

## 竞争游戏

- **持续的改进**：在GAN的训练过程中，生成器和判别器相互竞争。生成器试图制造越来越逼真的假数据，而判别器则努力提高其识别真假数据的能力。
- **博弈平衡**：这个过程可以被视为一个动态的博弈平衡。当其中一个组件改进时，另一个组件也需要相应地改进，以保持竞争力。
- **最终目标**：理想状态是达到一种平衡，其中生成器制造的假数据如此逼真，以至于判别器无法区分真假，即假数据与真实数据无法区别。

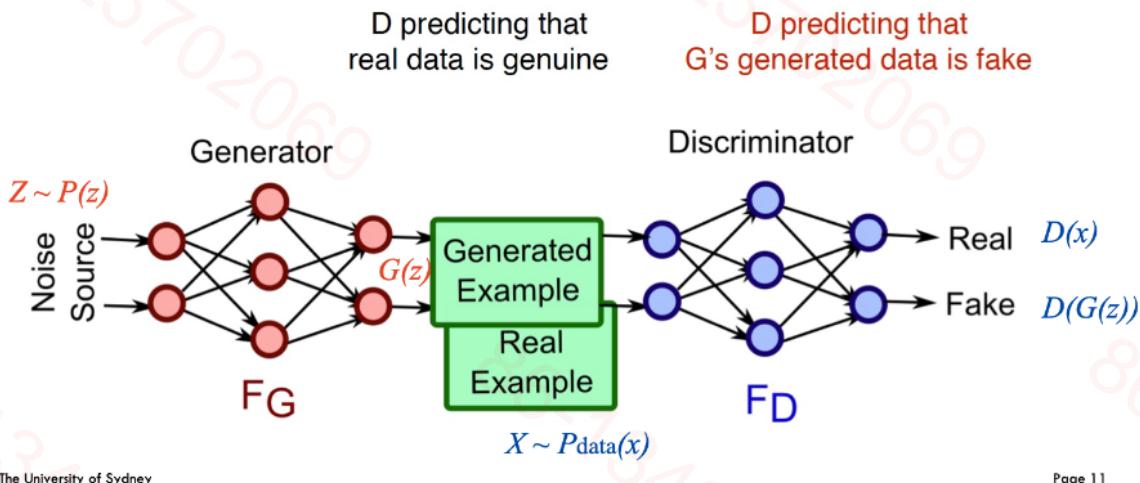
## 应用

- **图像生成**：GANs 在生成逼真的图像方面特别有效，可以用于艺术创作、游戏设计、影视特效等。
- **数据增强**：在数据有限的情况下，GANs 可以用来生成额外的训练数据。
- **风格转换**：GANs 还被用于图像风格转换，如将日常照片转换为具有特定艺术风格的图像。

## Structure

- A **two-player game** between two components: a generator **G** and a discriminator **D**

$$\min_G \max_D V(D, G) = \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x})} [\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z} \sim p_z(\mathbf{z})} [\log(1 - D(G(\mathbf{z})))]$$



## LOSS

### Generative Adversarial Networks

Generator (Artist)  $G_\theta$  and Discriminator (Critic)  $D_\psi$  are both Deep Convolutional Neural Networks.

Generator  $G_\theta : z \mapsto x$ , with parameters  $\theta$ , generates an image  $x$  from latent variables  $z$  (sampled from a standard Normal distribution).

Discriminator  $D_\psi : x \mapsto D_\psi(x) \in (0, 1)$ , with parameters  $\psi$ , takes an image  $x$  and estimates the probability of the image being real.

Generator and Discriminator play a 2-player zero-sum game to compute:

$$\min_{\theta} \max_{\psi} \left( \mathbb{E}_{x \sim p_{\text{data}}} [\log D_\psi(x)] + \mathbb{E}_{z \sim p_{\text{model}}} [\log(1 - D_\psi(G_\theta(z)))] \right)$$

Discriminator tries to maximize the bracketed expression,  
Generator tries to minimize it.

# Generative Adversarial Networks

Alternate between:

Gradient ascent on Discriminator:

$$\max_{\psi} \left( \mathbf{E}_{x \sim p_{\text{data}}} [\log D_{\psi}(x)] + \mathbf{E}_{z \sim p_{\text{model}}} [\log (1 - D_{\psi}(G_{\theta}(z)))] \right)$$

Gradient descent on Generator, using:

$$\min_{\theta} \mathbf{E}_{z \sim p_{\text{model}}} [\log (1 - D_{\psi}(G_{\theta}(z)))]$$

14



生成对抗网络 (GAN) 的损失函数体现了其核心的“最小化-最大化” (min-max) 博弈。在这个博弈中，生成器 (Generator, G) 和判别器 (Discriminator, D) 有着相反的目标。以下是 GAN 损失函数的详细解释：

## GAN损失函数

GAN的损失函数可以表示为：

$$\min_{\theta} \max_{\psi} \mathbb{E}_{x \sim p_{\text{data}}} [\log D_{\psi}(x)] + \mathbb{E}_{z \sim p_{\text{model}}} [\log (1 - D_{\psi}(G_{\theta}(z)))]$$

这里， $\theta$ 是生成器的参数， $\psi$  是判别器的参数。 $x$ 是真实数据样本， $z$  是从潜在空间（通常是标准正态分布）采样得到的噪声。

## 组件

- **生成器  $G_{\theta}$** ：它的目标是生成逼真的假图像。函数  $G_{\theta}(z)$  表示从潜在变量  $z$  生成图像  $x$ 。
- **判别器  $D_{\psi}$** ：它试图区分输入图像是真实的还是由生成器制造的。 $D_{\psi}(x)$  表示判别器对图像  $x$  是真实图像的估计概率。

## 损失函数的两部分

1. **第一部分**： $\mathbb{E}_{x \sim p_{\text{data}}} [\log D_{\psi}(x)]$  是判别器试图最大化的部分，其目的是正确识别真实图像。
2. **第二部分**： $\mathbb{E}_{z \sim p_{\text{model}}} [\log (1 - D_{\psi}(G_{\theta}(z)))]$  是判别器试图最小化的部分，其目的是正确认识到生成器产生的假图像。

## 博弈过程

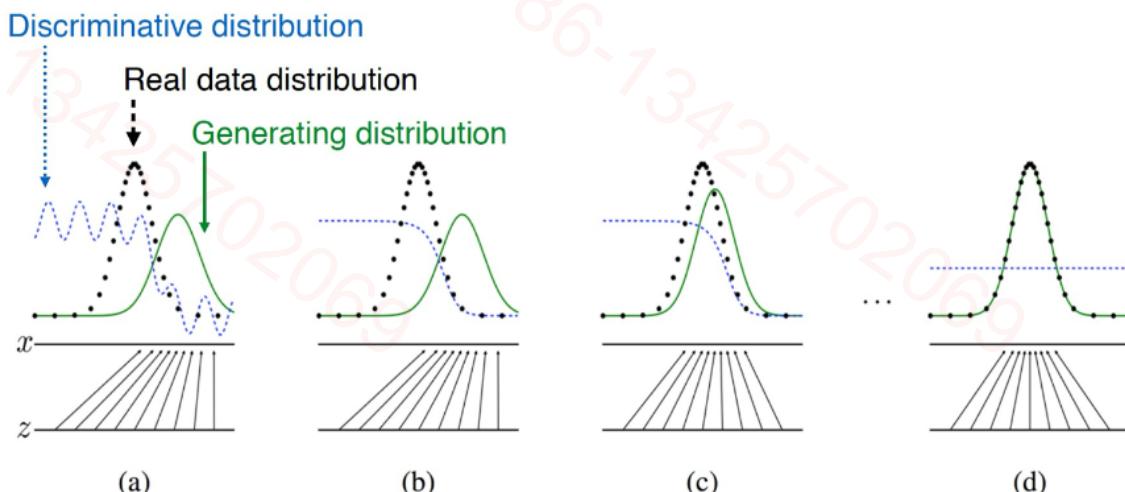
- 判别器的目标：最大化损失函数，以便更准确地区分真假图像。
- 生成器的目标：最小化损失函数，这意味着它试图欺骗判别器，使其将假图像误判为真实图像。

这个最小化-最大化博弈过程推动了生成器和判别器的持续改进。最终目标是达到一个平衡点，即判别器无法区分真实图像和生成器产生的假图像。

## 结果

- 当GAN训练平衡时，生成器能够产生非常逼真的图像，判别器的判断变得困难。
- 在理想情况下，判别器对真实图像和生成图像的判别概率都应接近50%，即无法区分真假。

总的来说，GAN的损失函数通过这种竞争博弈推动了生成器和判别器的持续进步，这使得生成器能够创造出越来越逼真的图像，而判别器则不断提高其识别能力。



Generative adversarial nets are trained by simultaneously updating the discriminative distribution (blue line) so that it discriminates between samples from the data generating distribution (black, dotted line)  $p_x$  from those of the generative distribution  $p_g$  (green line). The lower horizontal line is the domain from which  $z$  is sampled, in this case uniformly. The horizontal line above is part of the domain of  $x$ . The upward arrows show how the mapping  $x = G(z)$  imposes the non-uniform distribution  $p_g$  on transformed samples.  $G$  contracts in regions of high density and expands in regions of low density of  $p_g$ .

The University of Sydney

Image credit to [Ian Goodfellow et al., Generative Adversarial Nets, NIPS 2016]

Page 14

**After several steps of training, if  $G$  and  $D$  have enough capacity, they will reach a point at which both cannot improve because  $p_g = p_{data}$ . The discriminator is unable to differentiate between the two distributions, i.e.  $D(x) = 1/2$ .**

在GAN（生成对抗网络）的训练过程中，理想的结束状态是生成器（G）和判别器（D）达到一种动态平衡，其中生成器产生的数据分布 $p_g$ 与真实数据分布 $p_{data}$ 相等。这种情况下，判别器无法区分真实数据和生成器产生的假数据，即对于所有的 $x$ ，判别器 $D(x)$ 的输出接近于1/2。这种情况发生的原因有几个方面：

## 1. 生成器和判别器的能力

- **足够的容量**：假设生成器和判别器都具有足够的容量（即网络结构足够复杂，能够捕捉到数据的复杂性质），它们可以理想地学习和适应对方的策略。
- **持续改进**：在训练初期，生成器可能产生的假数据质量不高，容易被判别器识别。但随着训练的进行，生成器逐渐学会模仿真实数据的分布，产生更逼真的假数据。

## 2. 博弈达到纳什均衡

- **不断的竞争**：在GAN训练的过程中，生成器和判别器相互竞争，互相驱动对方改进。
- **纳什均衡**：理想情况下，当生成器完美地模仿了真实数据的分布，训练达到一种纳什均衡。在这个点上，生成器不能进一步改进，因为它已经完美复制了真实数据的分布，而判别器也不能进一步改进，因为对于它来说，区分真假数据变得不可能。

## 3. 判别器的不确定性

- **概率为 1/2**：当  $p_g = p_{\text{data}}$  时，判别器对于每个输入  $x$ ，都无法确定它是来自真实数据还是生成器。因此，它给出的最佳猜测是一个随机猜测，即对于任何  $x$ ， $D(x)$  为 1/2。
- **无法区分**：此时，判别器输出的概率值 1/2 意味着它对于判断输入数据是真是假毫无信心，因为两种数据在统计上无法区分。

# Converge Failure

## Oscillation and Mode Collapse

- Due to the coevolutionary dynamics, GANs can sometimes oscillate or get stuck in a mediocre stable state.
  - *oscillation*: GAN trains for a long time, generating a variety of images, but quality fails to improve.
  - *mode collapse*: Generator produces only a small subset of the desired range of images, or converges to a single image (with minor variations).
- Methods for avoiding mode collapse:
  - Conditioning Augmentation
  - Minibatch Features (Fitness Sharing)
  - Unrolled GANs