《机器学习：从入门到入魔》

Machine Learning: From Zero to Hero

第三讲：神经网络模型

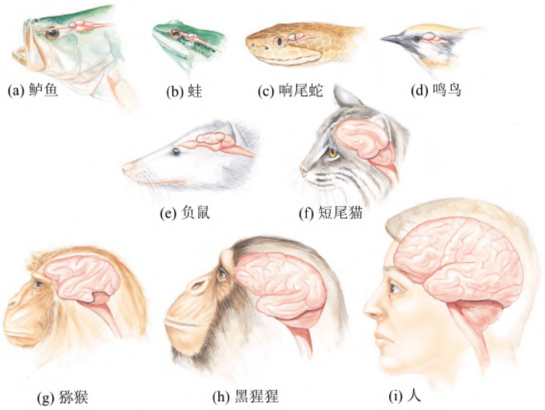
Lecture 3: Neural Networks as Learning Machines

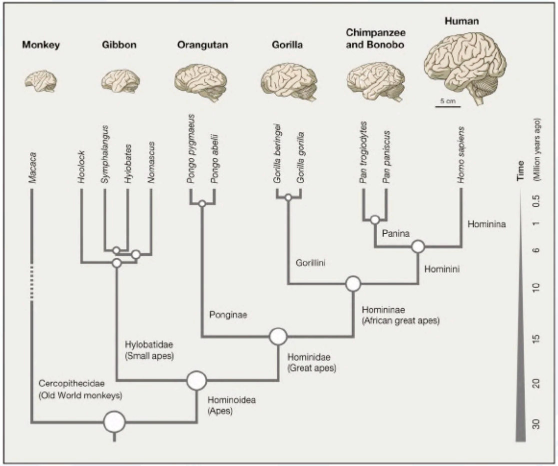
薛延波 *CSL BOSS 直聘*

2019-06-27

目录

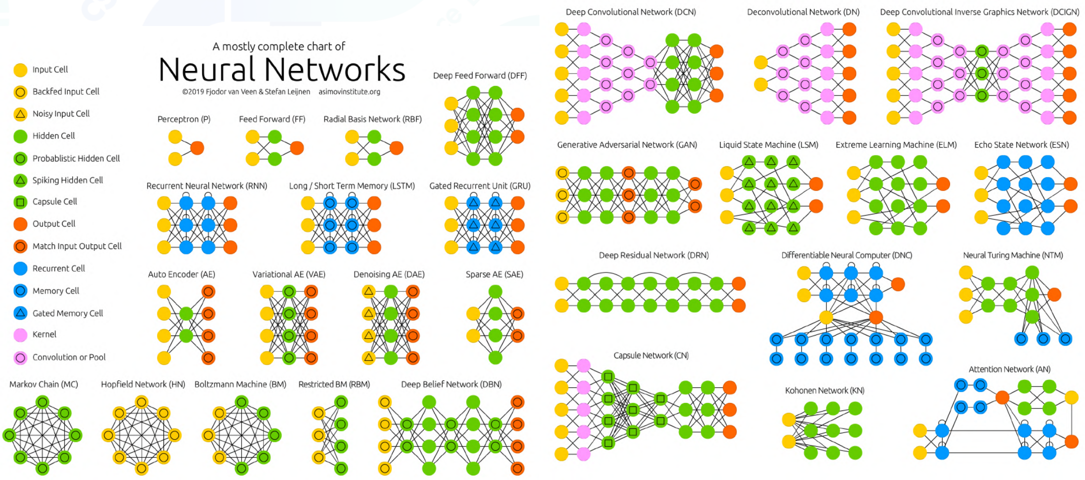
1 神经网络大观园





补充材料：

- https://www.3blue1brown.com/neural-networks
- https://www.coursera.org/
- http://www.asimovinstitute.org/neural-network-zoo/
- http://playground.tensorflow.org
- S. Haykin, “Neural networks and learning machines”, New York: Prentice Hall, 2009. [Hay09]
- 邱锡鹏，神经网络与深度学习，https://github.com/nndl/nndl.github.io

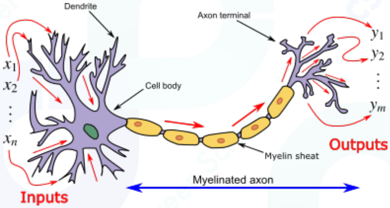


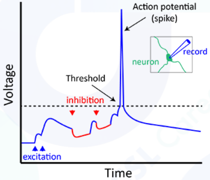
补充材料：

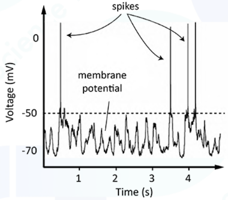
- https://www.3blue1brown.com/neural-networks
- https://www.coursera.org/
- http://www.asimovinstitute.org/neural-network-zoo/
- http://playground.tensorflow.org
- S. Haykin, “Neural networks and learning machines”, New York: Prentice Hall, 2009. [Hay09]
- 邱锡鹏，神经网络与深度学习，https://github.com/nndl/nndl.github.io

2 神经元的种类

- 新元时代的肛肠动物，10 亿-4.5 亿年前
- 分为细胞体 (cell body)、树突 (dendrites)和轴突 (axon)：树突到胞体为接受区，胞体末端为触发区，轴突末端的突触为输出区
- 人脑有 860 亿个神经元
- 神经元接受并集成其他神经元发送的脉冲信息，并在达到某个门限值之后激活

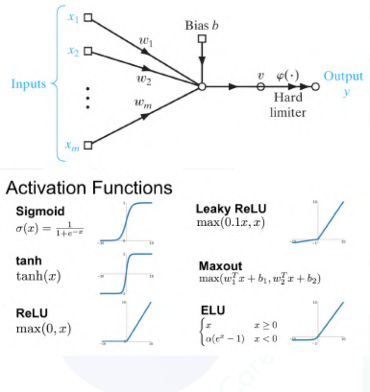


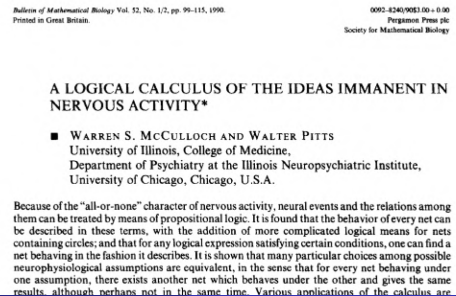




人工神经元

- 1943 年由 McCulloch 和 Pitts 提出 [MP43]
- 多个输入信号与人工神经元连接，通过门限或者其他非线性函数计算输出信号
- 最大的人工神经网络具有 1600 万神经元（相当于青蛙的大脑），需要超级计算机





Bulletin of Mathematical Biology Vol. 52, No. 1/2, pp. 99–115, 1990.
Printed in Great Britain.

0092-8240/90$3.00 + 0.00
Pergamon Press plc
Society for Mathematical Biology

A LOGICAL CALCULUS OF THE IDEAS IMMANENT IN NERVOUS ACTIVITY*

■ WARREN S. MCCULLOCH AND WALTER PITTS
University of Illinois, College of Medicine,
Department of Psychiatry at the Illinois Neuropsychiatric Institute,
University of Chicago, Chicago, U.S.A.

Because of the "all-or-none" character of nervous activity, neural events and the relations among them can be treated by means of propositional logic. It is found that the behavior of every net can be described in these terms, with the addition of more complicated logical means for nets containing circles; and that for any logical expression satisfying certain conditions, one can find a net behaving in the fashion it describes. It is shown that many particular choices among possible neurophysiological assumptions are equivalent, in the sense that for every net behaving under one assumption, there exists another net which behaves under the other and gives the same results, although perhaps not in the same time. Various applications of the calculus are

神经元的种类

神经元的分类：

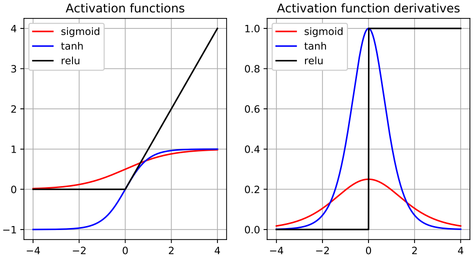
- 按照**功能**分：输入、输出、隐含
- 按照**行为**分：确定性、概率型
- 按照**复杂度**分：记忆、核、胶囊、卷积（滤波）
- 按照**激活**分：线性、门限、Sigmoid、tanh、ReLU
- 按照**信号传播方式**分：数值、脉冲（幅度和频率）

神经元连接的分类：

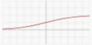
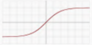
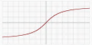
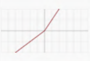
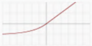
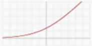
- 按照**方向**分：有向、无向
- 按照**起始/终点**分：前馈、反馈（循环）

激活函数与梯度消失

- **门限函数**：$f(x) = 1$ if $x \geq \sigma$，$f(x) = 0$ if $x < \sigma$
- **线性激活**：$f(x) = ax, a > 0$
- **线性整流** (ReLU)
  - ReLU: $f(x) = \max(x, 0)$, $f'(x) = 1$
  - leaky ReLU: $f(x) = \begin{cases} x & \text{if } x > 0 \\ 0.01x & \text{otherwise} \end{cases}$
  - noisy ReLU: $f(x) = \max(0, x + Y)$ with $Y \sim \mathcal{N}(0, \sigma(x))$
- **非线性**：逻辑函数/Sigmoid $f(x) = 1/(1 + e^{-x})$ 等



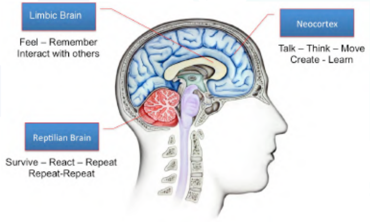
激活函数 cheat sheet

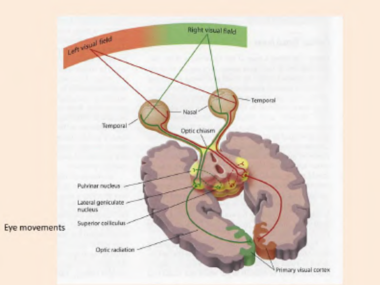
Name	Plot	Equation	Derivative
Identity		$f(x)=x$	$f'(x)=1$
Binary step		$f(x)=\begin{cases}0 & \text{for } x<0\\ 1 & \text{for } x\geq 0\end{cases}$	$f'(x)=\begin{cases}0 & \text{for } x\neq 0\\ ? & \text{for } x=0\end{cases}$
Logistic (a.k.a Soft step)		$f(x)=\frac{1}{1+e^{-x}}$	$f'(x)=f(x)(1-f(x))$
TanH		$f(x)=\tanh(x)=\frac{2}{1+e^{-2x}}-1$	$f'(x)=1-f(x)^2$
ArcTan		$f(x)=\tan^{-1}(x)$	$f'(x)=\frac{1}{x^2+1}$
Rectified Linear Unit (ReLU)		$f(x)=\begin{cases}0 & \text{for } x<0\\ x & \text{for } x\geq 0\end{cases}$	$f'(x)=\begin{cases}0 & \text{for } x<0\\ 1 & \text{for } x\geq 0\end{cases}$
Parameteric Rectified Linear Unit (PReLU) [2]		$f(x)=\begin{cases}\alpha x & \text{for } x<0\\ x & \text{for } x\geq 0\end{cases}$	$f'(x)=\begin{cases}\alpha & \text{for } x<0\\ 1 & \text{for } x\geq 0\end{cases}$
Exponential Linear Unit (ELU) [3]		$f(x)=\begin{cases}\alpha(e^x-1) & \text{for } x<0\\ x & \text{for } x\geq 0\end{cases}$	$f'(x)=\begin{cases}f(x)+\alpha & \text{for } x<0\\ 1 & \text{for } x\geq 0\end{cases}$
SoftPlus		$f(x)=\log_e(1+e^x)$	$f'(x)=\frac{1}{1+e^{-x}}$

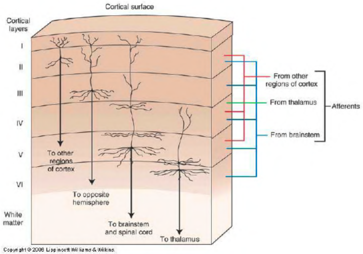
3 神经网络的结构

大脑皮层的分层结构

- 大脑皮层(Cerebral Cortex) 绝大部分由新皮质(Neocortex) 构成：哺乳动物，2-4mm
- 新皮质呈现分层结构（六层 V1-V6）







神经网络的结构

- **感知机**（1G）：模拟计算机
- **多层感知机**（2G）：CPU/GPU/TPU
  - 循环多层感知机
  - 自编码器
- **概率型神经网络**（2G）：QPU
  - 玻尔兹曼机 BM
  - 温度为 0，BM→Hopfield 网络
- **脉冲神经网络** SNN（3G）[Maa97]：IBM TrueNorth

```python
import numpy as np

class Neural_Network(object):
    def __init__(self):
        #Define Hyperparameters
        self.inputLayerSize = 2
        self.outputLayerSize = 1
        self.hiddenLayerSize = 3

        #Weights (parameters)
        self.W1 = np.random.randn(self.inputLayerSize, self.hiddenLayerSize)
        self.W2 = np.random.randn(self.hiddenLayerSize, self.outputLayerSize)

    def forward(self, X):
        #Propagate inputs though network
        self.z2 = np.dot(X, self.W1)
        self.a2 = self.sigmoid(self.z2)
        self.z3 = np.dot(self.a2, self.W2)
        yHat = self.sigmoid(self.z3)
        return yHat

    def sigmoid(self, z):
        #Apply sigmoid activation function to scalar, vector, or matrix
        return 1/(1+np.exp(-z))
```

前馈神经网络 FFNN[Ros58]

- 全连接神经网络、多层感知机
- 各神经元分别属于不同的层，层内无连接
- 相邻两层之间的神经元全部两两连接
- 整个网络中无反馈，信号从输入层向输出层单向传播，可用一个有向无环图DAG 表示

通用近似定理 Universal Approximation Theorem [Hay09]

也叫**万能逼近定理**，如果一个 FFNN 具有线性输出层和至少一层隐藏层，只要给予网络足够数量的神经元，便可以实现以足够高精度来逼近任意一个在 $\mathbb{R}^n$ 的紧子集 (Compact subset) 上的连续函数。

$$F(\boldsymbol{x}) = \sum_{i=1}^{N} \alpha_i \phi(\boldsymbol{w}_i^T \boldsymbol{x} + b_i)$$

使得 $|F(\boldsymbol{x}) - f(x)| < \epsilon$，$N$：神经网络数目，$\alpha_i$：输出权重，$\phi$：激活函数

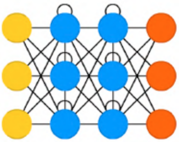
基于核的神经网络 RBF [BL88]

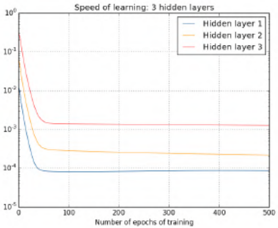
- 径向基函数 RBF：类似于多层感知机，区别：
  - 只有一个隐含层
  - 隐含层包含 Gaussian 函数为核函数的神经元：

$$\phi_i(\boldsymbol{x}) = \phi(\boldsymbol{x} - \boldsymbol{x}_i) = \exp(-||\boldsymbol{x} - \boldsymbol{x}_i||^2/(2\sigma_i^2))$$

$\boldsymbol{x}_i$：核函数的中心，σ_i：核函数的宽度

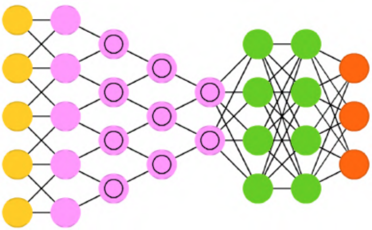
- 通用函数近似模型（universal approximator）：
  - MLP：通过嵌套的加权求和来近似（把复杂度暴露出来）
  - RBF：通过单一的加权求和（把复杂度藏入核中）





- 为什么需要循环?
  - 反馈是智能系统的核心
  - 循环可以为系统增加记忆
- RNN 家族的成员：RMLP、ESN、LSTM
- 按照时间展开 → 无限深度的前馈网络
- 梯度消失/爆炸问题：对于深度网络，越靠近输入层的参数学习速度越慢

卷积神经网络 CNN [LBB+98]



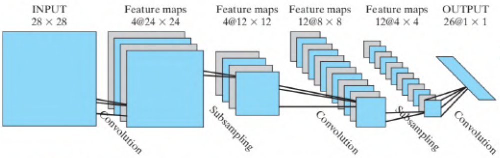
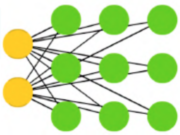


FIGURE 4.23 Convolutional network for image processing such as handwriting recognition. (Reproduced with permission of MIT Press.)

- **结构**：卷积 → 整流 ReLU → 池化 → 损失函数
- **优点**：局部连接、权重共享、空间/时间次采样
- 卷积的**数学定义**：$[f*g](t)=\int_0^t f(\tau)g(t-\tau)d\tau$
  在频域：$[f*g](t)=F^{-1}(F(f)\cdot F(g))$，$F(\cdot)$ 为傅立叶变换，g 为滤波器，$f*g$ 输出为特征图
- **严格意义上**：$f\cdot g$ 为互相关
  CNN→ 互相关神经网络/（转置）卷积神经网络

自组织映射网络 SOM [Koh82]



- 又称为 Kohonen Networks
- 非监督学习

步骤

1. **初始化**：所有的连接权 w_{ij} 随机初始化
2. **竞争**：对每一个输入 → 用判别函数（discriminant function）计算所有神经元的输出 → 输出值最大的获胜
   判别函数：$\mathcal{L}_j(\boldsymbol{x}) = ||\boldsymbol{x} - \boldsymbol{w}_j||^2$
3. **合作**：获胜神经元决定邻域内其他激活神经元的拓扑结构
   邻域：$h_{i(\boldsymbol{x}),j} = \exp(-d^2_{i(\boldsymbol{x}),j}/(2\sigma^2))$
   $d_{i(\boldsymbol{x}),j}$ 为神经元 i 和 j 的距离
4. **自适应**：按照 Hebb 更新法则，获胜神经元和其拓扑邻域内的其他神经元更新权重
   $\Delta w_{ij} = \lambda(t) \cdot h_{i(\boldsymbol{x}),j}(t) \cdot (x_i - w_{ij})$

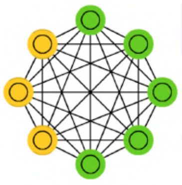


图: 概率型网络

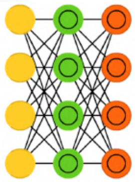


图: 变分自编码器

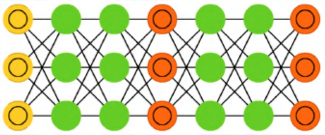


图: 生成对抗网络

填空题: ____ is all you need

- Attention [VSP$^+$17]
- Memory [GKMZ18]
- CNN [CW17]
- Good Init [XXP17]

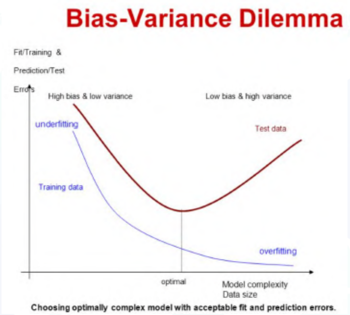
记忆力网络 → 注意力网络 [VSP$^+$17] → 空间变换网络 [JSZ$^+$15]

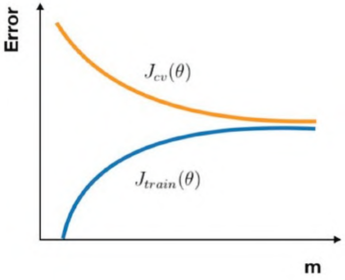
4　神经网络的训练

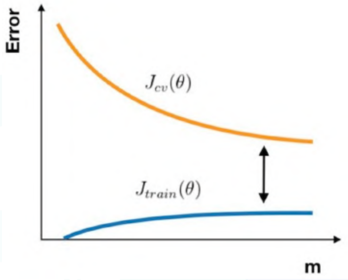
How to train your neural network (dragon)?

- know his temper
- be his friend
- use the cloud
- understand his valley

神经网络的训练







偏差：模型输出与样本真实结果的差距

- 引入更多的相关特征
- 采用多项式特征
- 减小正则化参数

方差：模型对于给定值的输出稳定性

- 采集更多的样本数据
- 减少特征数量，去除非主要的特征
- 增加正则化参数

感知机

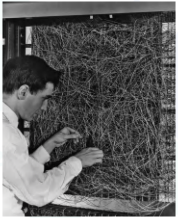


图: Rosenblatt 在连接感知机，与 20x20 的相机相连，神经元由可变电阻连接

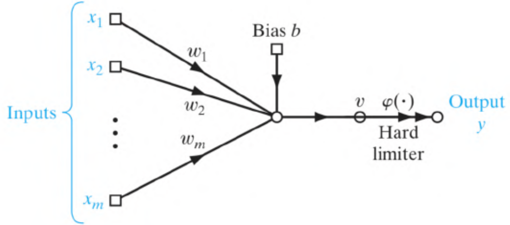
```python
import numpy as np

class Perceptron(object):

    def __init__(self, no_of_inputs, threshold=100, learning_rate=0.01):
        self.threshold = threshold
        self.learning_rate = learning_rate
        self.weights = np.zeros(no_of_inputs + 1)

    def predict(self, inputs):
        summation = np.dot(inputs, self.weights[1:]) + self.weights[0]
        if summation > 0:
          activation = 1
        else:
          activation = 0
        return activation

    def train(self, training_inputs, labels):
        for _ in range(self.threshold):
            for inputs, label in zip(training_inputs, labels):
                prediction = self.predict(inputs)
                self.weights[1:] += self.learning_rate * (label - prediction) * inputs
                self.weights[0] += self.learning_rate * (label - prediction)
```



1958 年由 Rosenblatt 提出 [Ros58]

损失函数：经验风险

训练方法：$\boldsymbol{w}_{n+1} = \boldsymbol{w}_n + \lambda(\boldsymbol{d} - \tilde{\boldsymbol{y}})\boldsymbol{x}$

多层感知机

```python
# Backpropagation
def backprop(self, X, y, o):
    self.o_error = y - o # error in output
    self.o_delta = self.o_error*self.sigmoidPrime(o) # applying derivative of sigmoid to error

    self.z2_error = self.o_delta.dot(self.W2.T) # z2 error: how much our hidden layer weights contributed to output
    self.z2_delta = self.z2_error*self.sigmoidPrime(self.z2) # applying derivative of sigmoid to z2 error

    self.W1 += X.T.dot(self.z2_delta) # adjusting first set (input --> hidden) weights
    self.W2 += self.z2.T.dot(self.o_delta) # adjusting second set (hidden --> output) weights
```

$$\frac{\partial \mathcal{L}}{\boldsymbol{\theta}_{K-1}} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{f}_K} \frac{\partial \boldsymbol{f}_K}{\partial \boldsymbol{\theta}_{K-1}} \tag{1}$$

$$\frac{\partial \mathcal{L}}{\boldsymbol{\theta}_{K-2}} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{f}_K} \frac{\partial \boldsymbol{f}_K}{\partial \boldsymbol{f}_{K-1}} \frac{\boldsymbol{f}_{K-1}}{\partial \boldsymbol{\theta}_{K-2}} \tag{2}$$

$$\frac{\partial \mathcal{L}}{\boldsymbol{\theta}_{i}} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{f}_K} \frac{\partial \boldsymbol{f}_K}{\partial \boldsymbol{f}_{K-1}} \cdots \frac{\partial \boldsymbol{f}_{i+2}}{\partial \boldsymbol{f}_{i+1}} \frac{\boldsymbol{f}_{i+1}}{\partial \boldsymbol{\theta}_{i}} \tag{3}$$

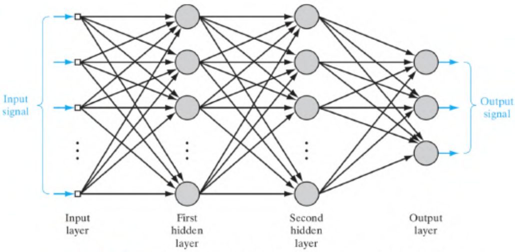


FIGURE 4.1 Architectural graph of a multilayer perceptron with two hidden layers.

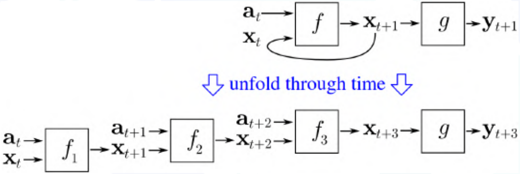
损失函数：经验风险 (瞬间误差能量)
训练方法：反向传播算法 BP

$$
\begin{aligned}
\boldsymbol{x}_{I,n+1} &= \boldsymbol{f}(\boldsymbol{x}_{I,n}, \boldsymbol{u}_n) && (4)\\
\boldsymbol{x}_{II,n+1} &= \boldsymbol{f}(\boldsymbol{x}_{II,n}, \boldsymbol{x}_{I,n+1}) && (5)\\
&\vdots \\
\boldsymbol{x}_{o,n+1} &= \boldsymbol{f}(\boldsymbol{x}_{o,n}, \boldsymbol{x}_{K,n+1}) && (6)
\end{aligned}
$$

训练方法：基于时间的反向传播算法 BPTT

```python
def bptt(self, x, y):
    assert len(x) == len(y)
    output = Softmax()
    layers = self.forward_propagation(x)
    dU = np.zeros(self.U.shape)
    dV = np.zeros(self.V.shape)
    dW = np.zeros(self.W.shape)

    T = len(layers)
    prev_s_t = np.zeros(self.hidden_dim)
    diff_s = np.zeros(self.hidden_dim)
    for t in range(0, T):
        dmulv = output.diff(layers[t].mulv, y[t])
        input = np.zeros(self.word_dim)
        input[x[t]] = 1
        dprev_s, dU_t, dW_t, dV_t = layers[t].backward(input, prev_s_t, self.U, self.W, self.V, diff_s, dmulv)
        prev_s_t = layers[t].s
        dmulv = np.zeros(self.word_dim)
        for i in range(t-1, max(-1, t-self.bptt_truncate-1), -1):
            input = np.zeros(self.word_dim)
            input[x[i]] = 1
            prev_s_i = np.zeros(self.hidden_dim) if i == 0 else layers[i-1].s
            dprev_s, dU_i, dW_i, dV_i = layers[i].backward(input, prev_s_i, self.U, self.W, self.V, dprev_s, dmulv)
            dU_t += dU_i
            dW_t += dW_i
        dV += dV_t
        dU += dU_t
        dW += dW_t
    return (dU, dW, dV)
```



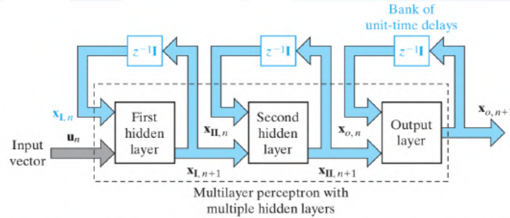


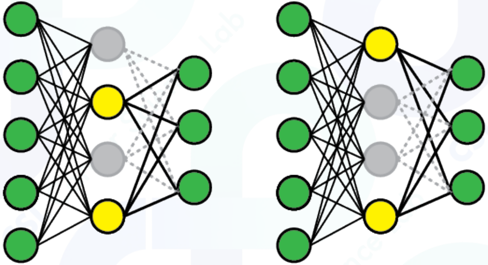
FIGURE 15.4 Recurrent multilayer perceptron; feedback paths in the network are printed in red.

正则化理论 ≈ 受限优化

$$\begin{pmatrix}\text{Regularized}\\\text{cost}\\\text{function}\end{pmatrix} = \begin{pmatrix}\text{Empirical}\\\text{cost}\\\text{function}\end{pmatrix} + \begin{pmatrix}\text{Regularization}\\\text{parameter}\end{pmatrix} \times (\text{Regularizer})$$

- **经典正则化理论**：Tikhonov，1963[Tik63]
  - 主要思想：通过非负函数将对解的先验信息嵌入代价函数，如输入-输出映射为平滑函数
  - 统计学中称为脊回归（ridge regression）
  - 机器学习中称为权重衰减（weight decay）
  - 主要考虑标签数据生成空间的特性，标签数据 (x, d) 是根据联合分布 $p_{x,d}(x, d)$ 产生
- **广义正则化理论**：Belkin 等，2006[BNS06]
  - 主要思想：对经典正则化理论进行扩展，以便考虑不含标签的输入空间的特性
  - 输入数据 x 是根据边缘概率 $p_x(x)$ 产生
  - 为半监督学习提供数学基础
- **自适应正则化理论**：不同于传统正则化对所有的参数/模型采用均匀的约束 → 对每个参数自适应约束 [LSAH98]，DropOut 可以看作一种特例 [WWL13]
- **非原则性正则化** Non-principled：Early-stopping、数据增强

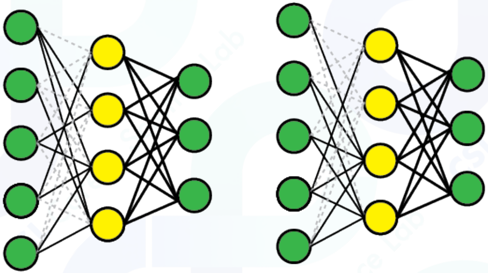
Dropout：通过对神经元进行抑制而实现正则化



DropOut [SHK+14]

- $r = m \circ a(\mathbf{W}\mathbf{v})$
  m: 二进制掩码
  $\circ$: element-wise(Hadamard) product
- 随机 (通常 $\rho = 0.5$) 扔掉一些神经元 $\rightarrow$ 对原始的大网络进行采样，需要在测试的时候乘上 ρ
- 一种特殊的集成学习 *ensemble learning* $\rightarrow$ 训练一组弱分类器并联合使用
- **推理**：使用原始大网络，但需要对神经元的输出乘上 ρ

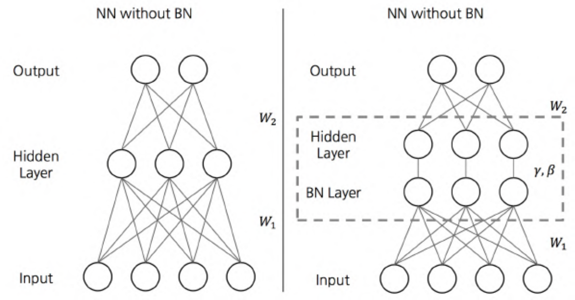
DropConnect：通过对神经元连接进行抑制而实现正则化



DropConnect [WZZ+13]

- $r = a((M \circ \mathbf{W})\mathbf{v})$
  M: 二进制矩阵
- 随机扔掉连接权：对 DropOut 的泛化
- **推理**:
  1. 在激活之前计算均值 μ 和方差 σ^2
  2. 从高斯分布采样 $\mathcal{N}(\mu, \sigma^2)$
  3. 通过激活函数并平均
  4. 输入下一层

BatchNorm：通过对每一层进行归一而实现正则化



Batch normalization [IS15]

- 问题：协方差漂移 → 模型艰难应对输入分布的变化
- 解决方法：去除平均值和方差，标准化；将输入保持在激活函数的线性区间内

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

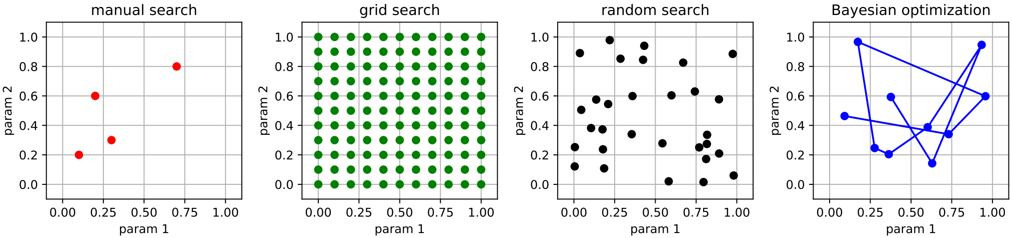
$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

超参调优

超参：learning rate, momentum, mini-batch size，建议 $3 \rightarrow 1 \rightarrow 2 \rightarrow 4$

1. **手动搜索**：经验法 *empirical approach*
2. **网络搜索**：系统法 *systematic approach*, choose bounds and step-size for each hyperparameter
3. **随机搜索**：有效法 *effective approach*, navigate grid randomly
4. **贝叶斯优化** [SLA12]：智能法 *smart approach*，代价函数为随机函数，使用高斯过程先验 → 评估代价函数 → 更新先验 → 选择下一个参数以最大化期望提高



5　总结和课程大纲

总结

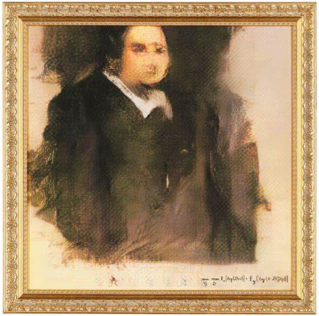


图: 第一幅 GAN 生成的油画拍卖了 300 万

- 感知机几乎为所有神经网络的祖先
- 神经网络的种类:
  - **前馈神经网络**
  - **反馈/循环神经网络**
  - **卷积神经网络**
  - **自组织映射网络**
- 当代神经网络主要的发展方向:
  - **老树发新枝**: 旧的神经网络模型重新被唤起
  - **玩具变实用**: 实验室 → 工业界
  - **老狗学新技**: 各种技巧和方法来加快训练
  - **鸟枪换大炮**: CPU→GPU→TPU/FPGA
  - **一石杀二鸟**: 一个模型完成多项任务

课程大纲

- ~~机器学习简介~~
- ~~机器学习的数学基础~~
- ~~线性模型（线性回归、感知机、支持向量机）~~
- **神经网络模型（本节）**
- 概率类模型
- 高级模型
- 工业实践

References I

- David S Broomhead and David Lowe, *Radial basis functions, multi-variable functional interpolation and adaptive networks*, Tech. report, Royal Signals and Radar Establishment Malvern (United Kingdom), 1988.
- Mikhail Belkin, Partha Niyogi, and Vikas Sindhwani, *Manifold regularization: A geometric framework for learning from labeled and unlabeled examples*, Journal of machine learning research **7** (2006), no. Nov, 2399–2434.
- Qiming Chen and Ren Wu, *Cnn is all you need*, arXiv preprint arXiv:1712.09662 (2017).
- Jeffrey L Elman, *Finding structure in time*, Cognitive science **14** (1990), no. 2, 179–211.
- Sylvain Gelly, Karol Kurach, Marcin Michalski, and Xiaohua Zhai, *Memgen: Memory is all you need*, CoRR **abs/1803.11203** (2018).
- S. Haykin, *Neural networks and learning machines*, Prentice Hall, 2009.

References II

- Sergey Ioffe and Christian Szegedy, *Batch normalization: Accelerating deep network training by reducing internal covariate shift*, International Conference on Machine Learning (2015).
- Max Jaderberg, Karen Simonyan, Andrew Zisserman, et al., *Spatial transformer networks*, Advances in neural information processing systems, 2015, pp. 2017–2025.
- Teuvo Kohonen, *Self-organized formation of topologically correct feature maps*, Biological cybernetics **43** (1982), no. 1, 59–69.
- Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al., *Gradient-based learning applied to document recognition*, Proceedings of the IEEE **86** (1998), no. 11, 2278–2324.
- Jan Larsen, Claus Svarer, Lars Nonboe Andersen, and Lars Kai Hansen, *Adaptive regularization in neural network modeling*, Neural Networks: Tricks of the Trade, Springer, 1998, pp. 113–132.

References III

- Wolfgang Maass, *Networks of spiking neurons: the third generation of neural network models*, Neural networks **10** (1997), no. 9, 1659–1671.
- Warren S McCulloch and Walter Pitts, *A logical calculus of the ideas immanent in nervous activity*, The bulletin of mathematical biophysics **5** (1943), no. 4, 115–133.
- Frank Rosenblatt, *The perceptron: a probabilistic model for information storage and organization in the brain.*, Psychological review **65** (1958), no. 6, 386.
- N. Srivastava, Geoffery Hinton, Alex Krizhevsky, I. Sutskever, and Salakhutdinov, *Dropout: a simple way to prevent neural networks from overfitting*, Journal of machine learning research **15** (2014), no. 1, 1929–1958.
- Jasper Snoek, Hugo Larochelle, and Ryan P. Adams, *Practical bayesian optimization of machine learning algorithms*, Advances in neural information processing systems] (2012), 2951–2959.

References IV

- Andrei Nikolaevich Tikhonov, *On the solution of ill-posed problems and the method of regularization*, Doklady Akademii Nauk, vol. 151, Russian Academy of Sciences, 1963, pp. 501–504.
- Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin, *Attention is all you need*, Advances in neural information processing systems, 2017, pp. 5998–6008.
- Stefan Wager, Sida Wang, and Percy S Liang, *Dropout training as adaptive regularization*, Advances in neural information processing systems, 2013, pp. 351–359.
- Li Wan, Matthew Zeiler, Sixin Zhang, Yann LeCun, and Rob Fergus, *Regularization of neural network using dropconnect*, International Conference on Machine Learning (ICML) (2013).

Di Xie, Jiang Xiong, and Shiliang Pu, *All you need is beyond a good init: Exploring better solution for training extremely deep convolutional neural networks with orthonormality and modulation*, Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2017, pp. 6176–6185.