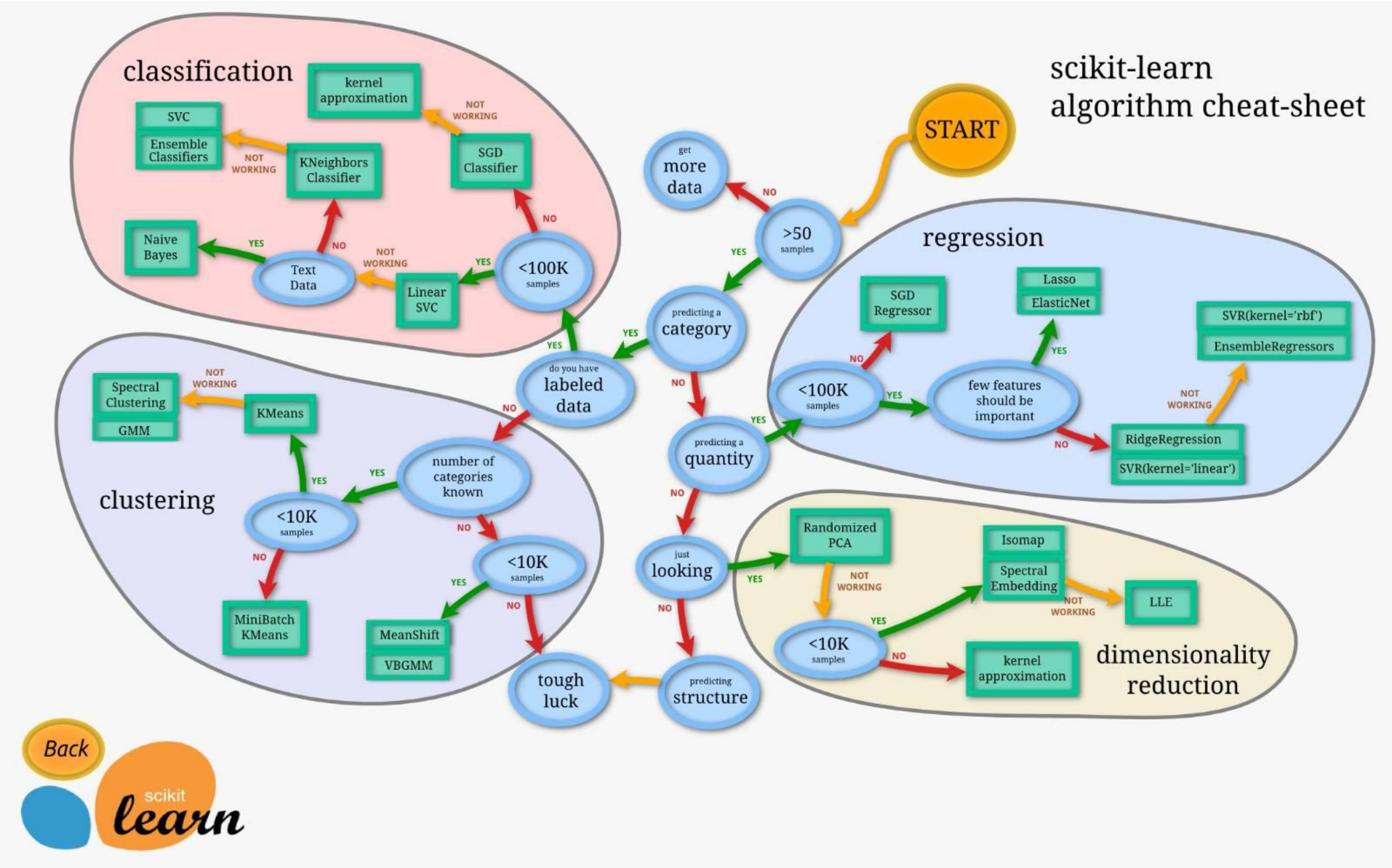
BIG DATA DIGEST 大数据文摘 稀牛学院 www.xiniuedu.com

Lecture 1
深度学习与图像分类
Deep Learning and Image Classification

课程合作请联系

info@bigdatadigest.cn

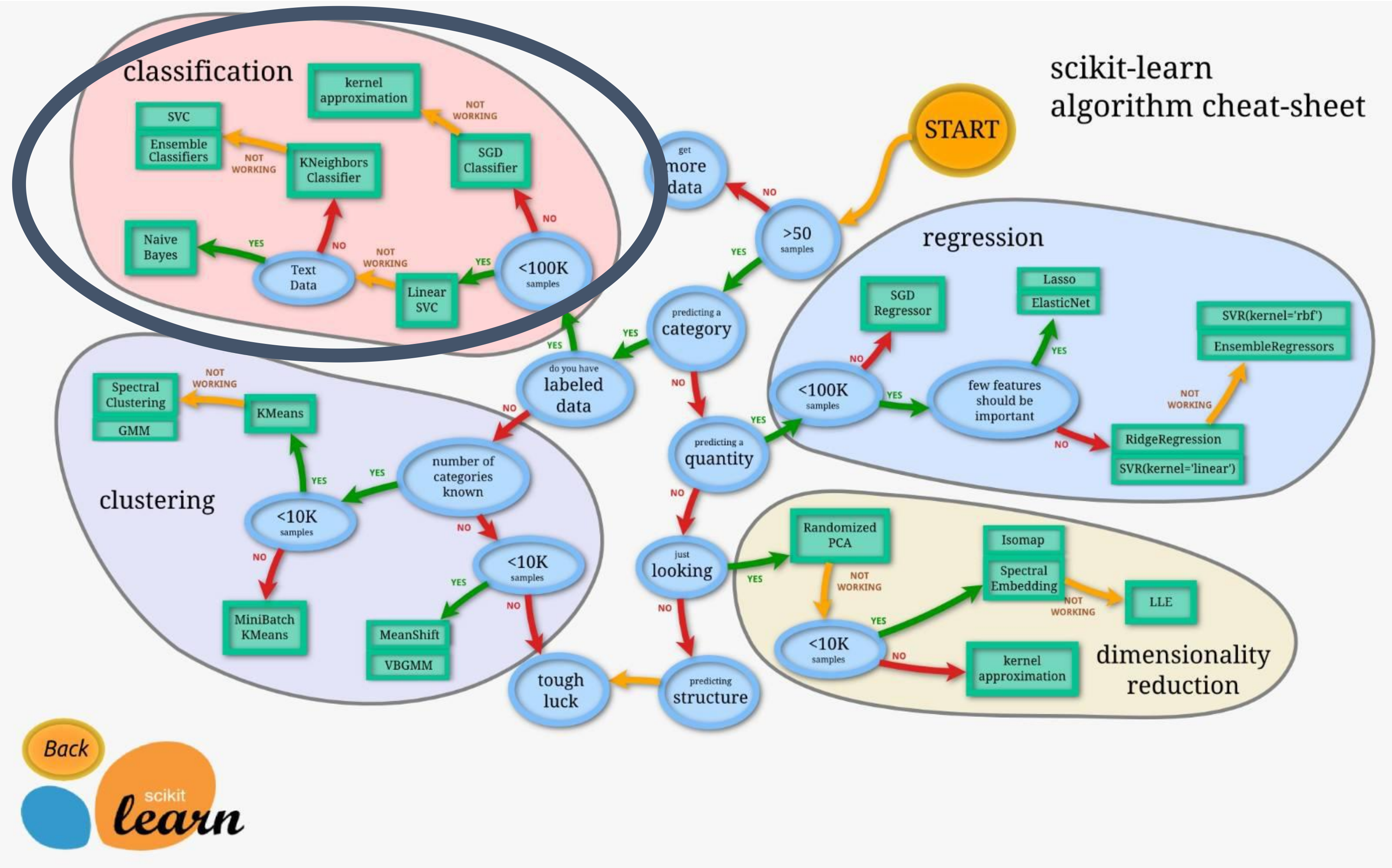
为什么是深度学习？



Scikit-learn 算法速查表

Sklearn官网: http://scikit-learn.org/stable/

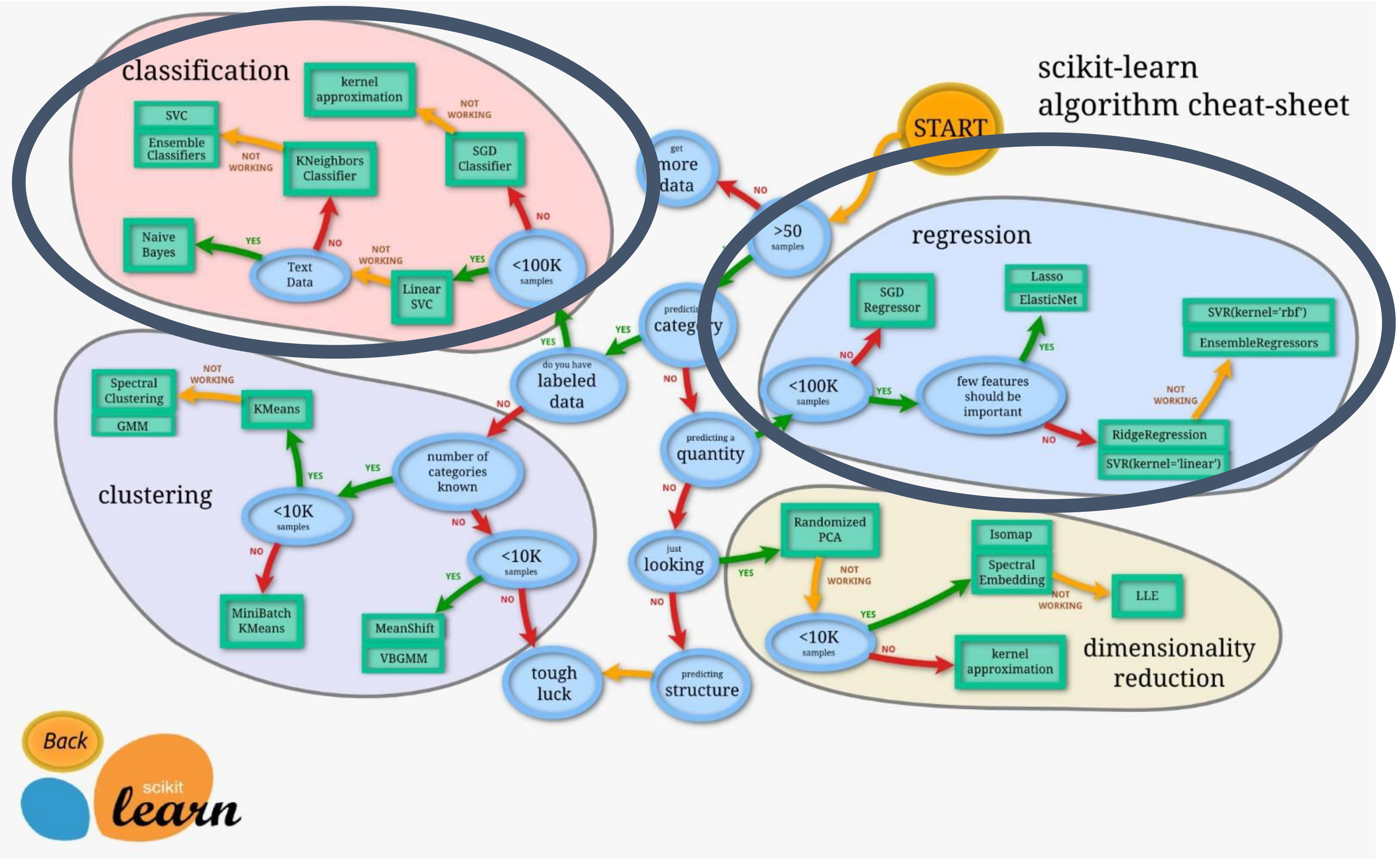
为什么是深度学习？



Scikit-learn 算法速查表

Sklearn官网: http://scikit-learn.org/stable/

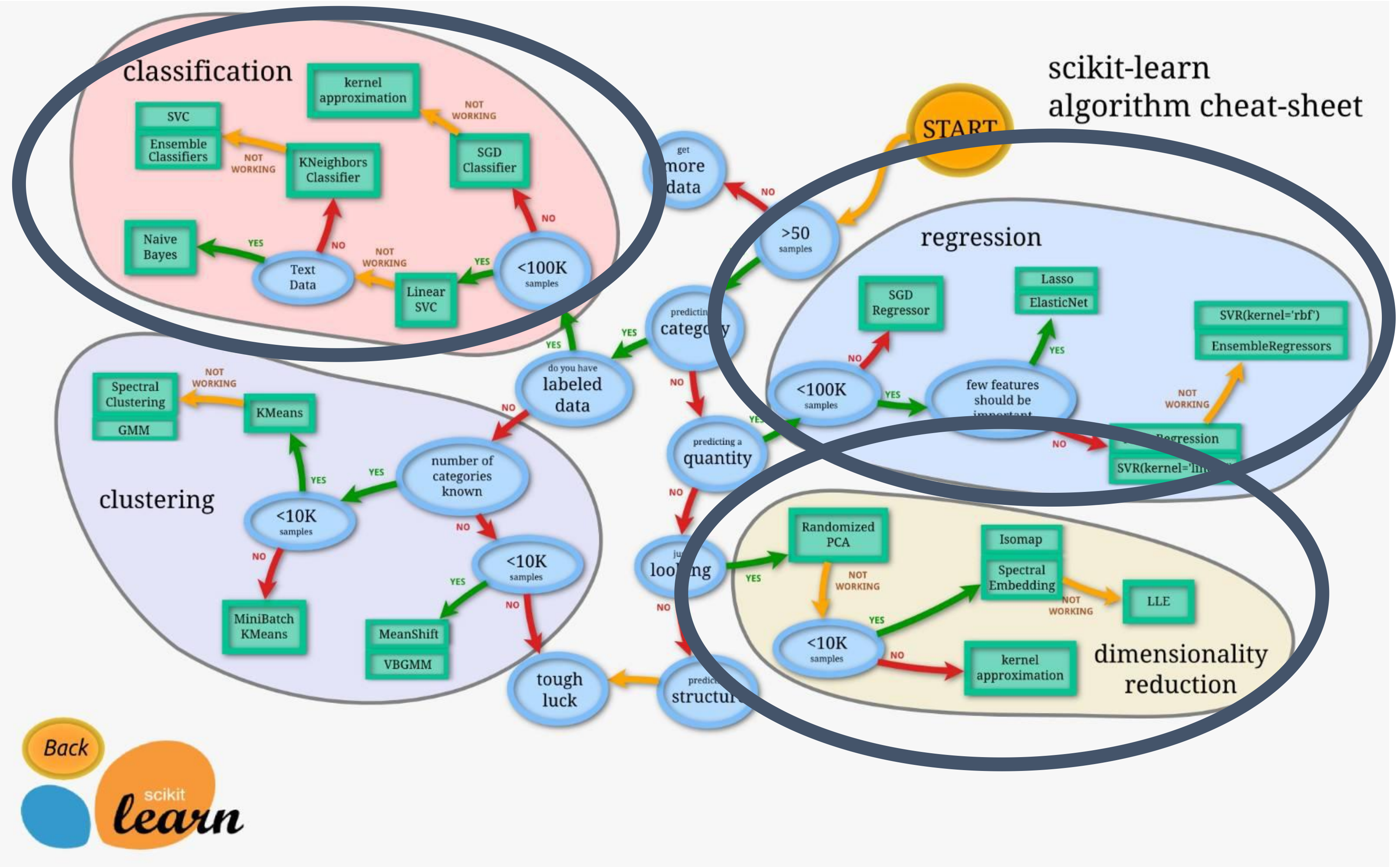
为什么是深度学习？



Scikit-learn 算法速查表

Sklearn官网: http://scikit-learn.org/stable/

为什么是深度学习？



Scikit-learn 算法速查表

Sklearn官网: http://scikit-learn.org/stable/

为什么是深度学习？

- Optical Character Recognition

  （OCR，光学字符识别，LeNet-5）

- Object Recognition

  （物体识别，Alex-Net）

- Machine Translation

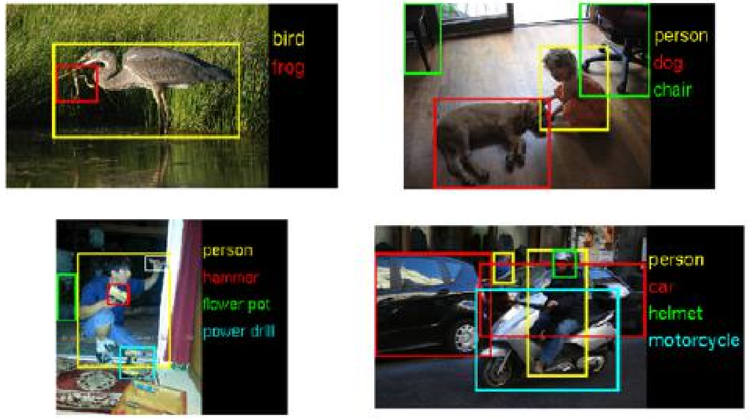
  （机器翻译，谷歌翻译）

- Dialog System

  （对话系统，Google Home）

- Speech Recognition

  （语音识别，Deep-Speech 2）



为什么是深度学习？

BIG DATA DIGEST 大数据文摘 稀牛学院 www.xiniuedu.com

MIT Technology Review

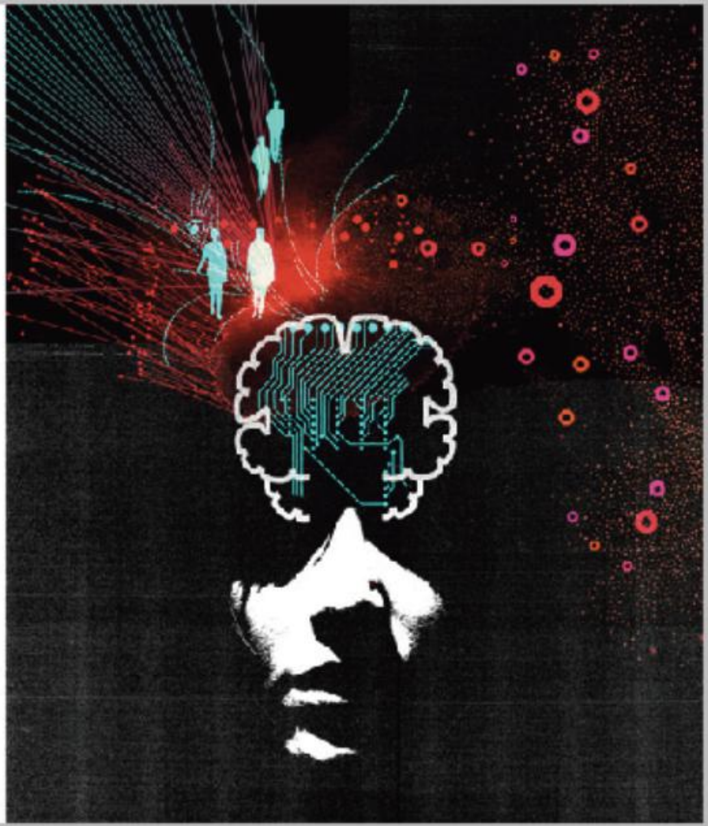
10 BREAKTHROUGH TECHNOLOGIES 2013

Introduction   The 10 Technologies   Past Years

译：2013年10项突破性科技

Deep Learning

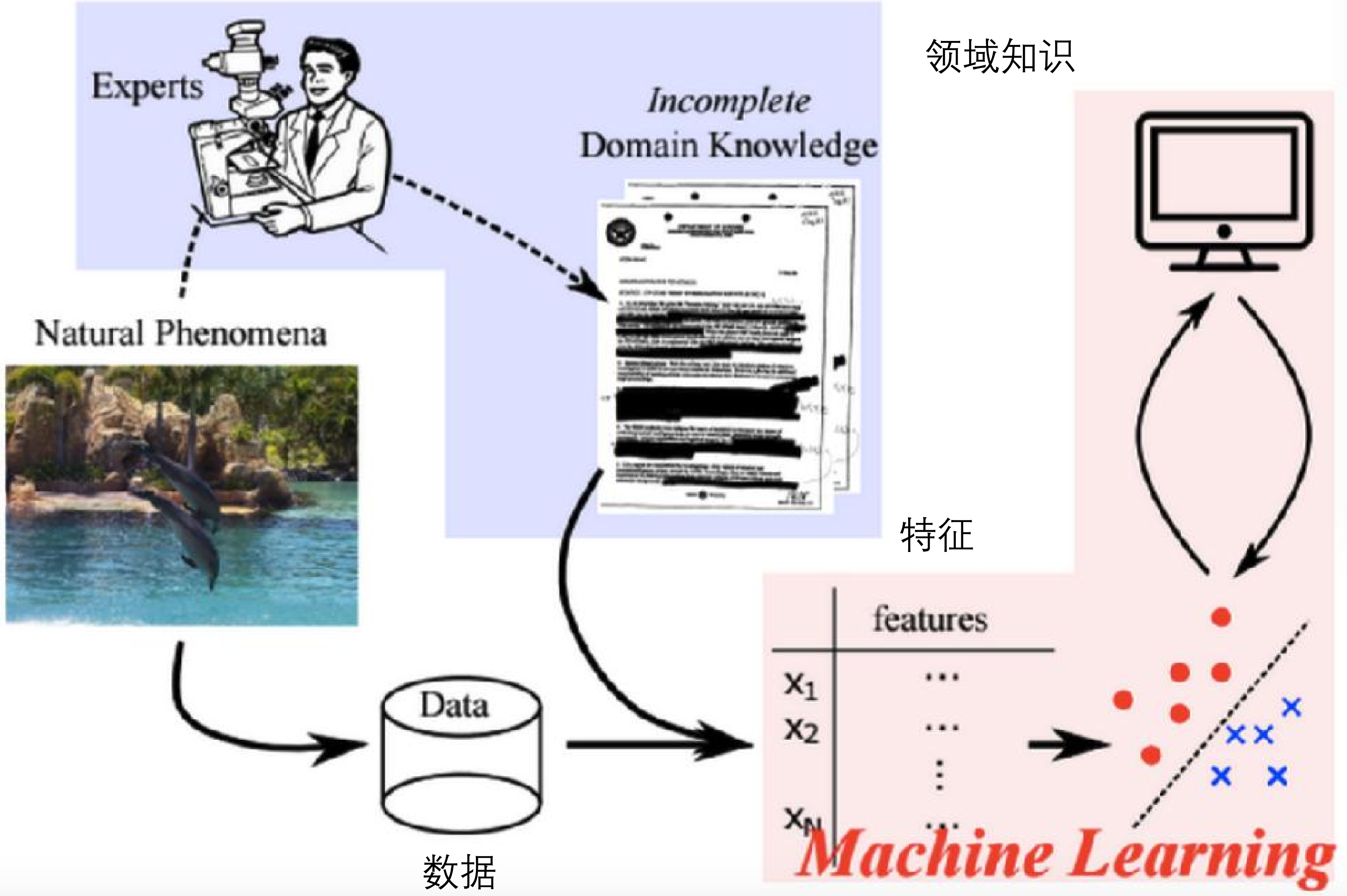
With massive amounts of computational power, machines can now recognize objects and translate speech in real time. Artificial intelligence is finally getting smart.



译：深度学习

随着大量计算资源的产生，机器可以实时的进行物体识别与语音翻译。AI终于变得聪明了。

为什么是深度学习？


专家
Experts
Incomplete
Domain Knowledge
领域知识
自然现象
Natural Phenomena
特征
features
x_1
x_2
x_N
Data
数据
Machine Learning
机器学习

为什么是深度学习？

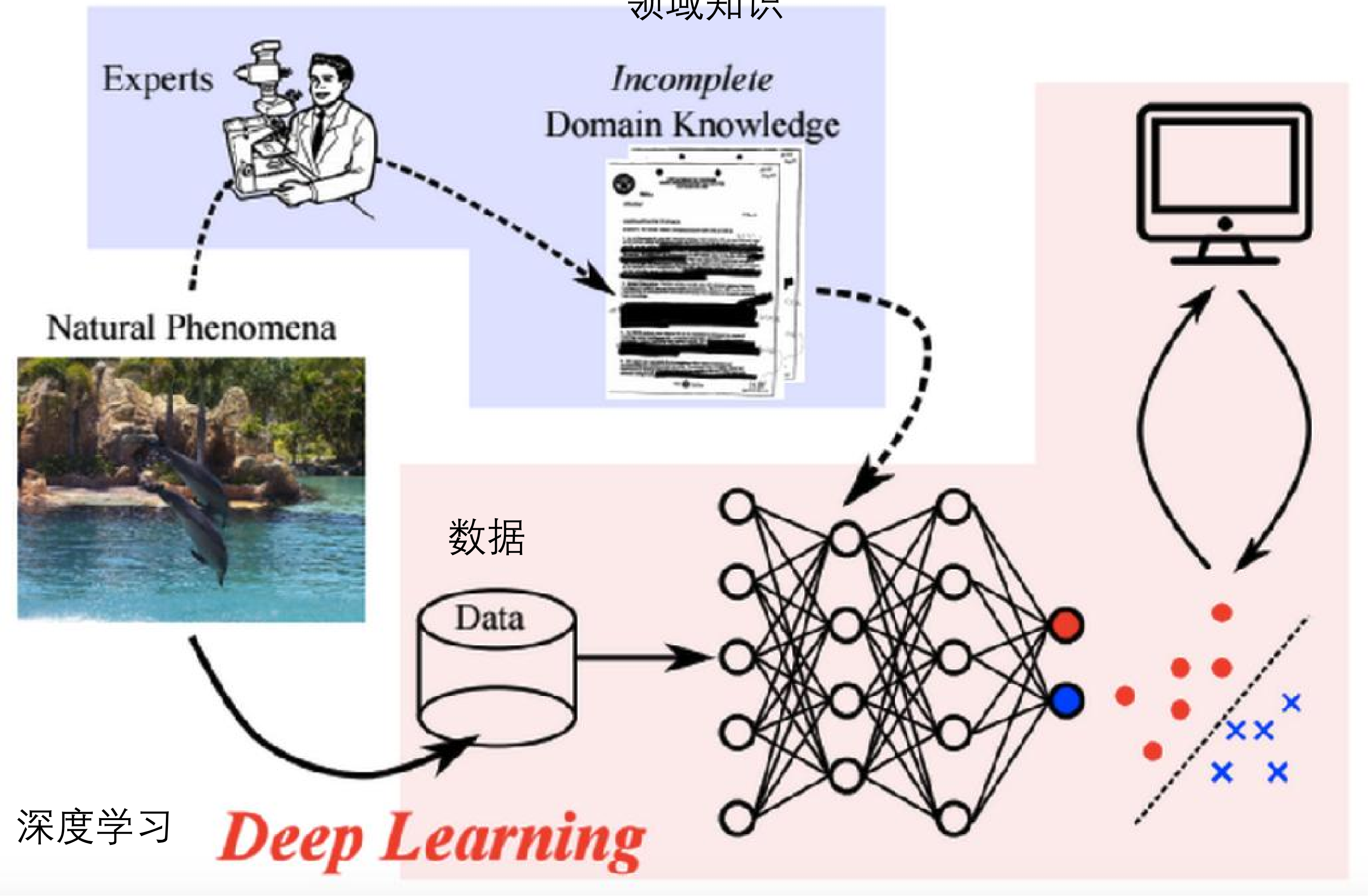
领域知识

专家

自然现象

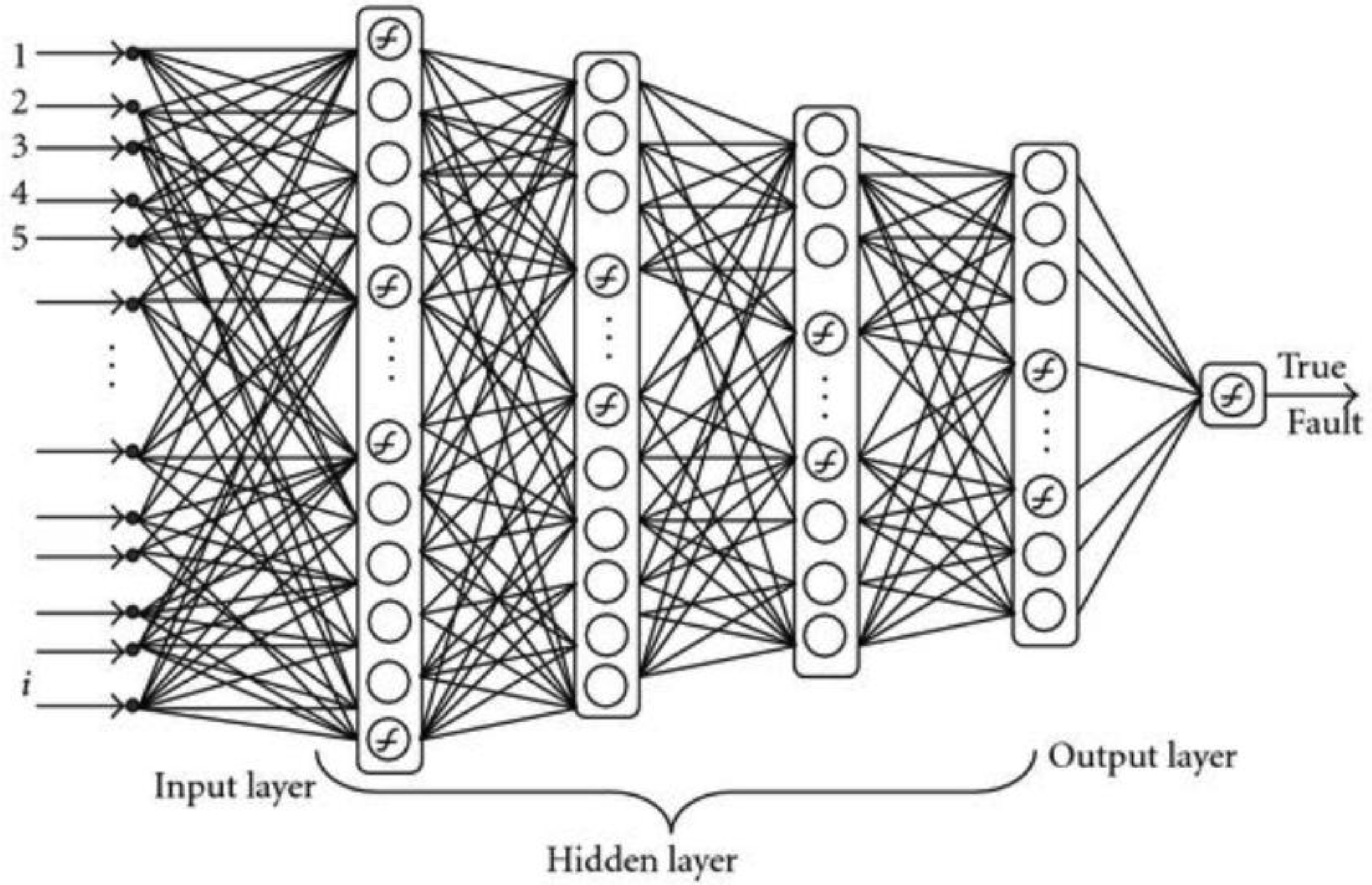
数据

深度学习



深层感知机（Multilayer Perceptron）

前向神经网络（Feed-forward Neural Networks）



神经网络（Neural Network Models）

- ✓ Feed-forward Neural Networks/ Multilayer Perceptron
- ✓ Convolutional Neural Networks(Image processing)
- ✓ Recurrent Neural Networks(Sequential processing)
- ✓ Auto-encoders(unsupervised learning)
- ✓ Generative Adversarial Networks(Generative Model, Adversarial techniques)
- ✓ Variational Auto-encoders(Generative Model, Variational techniques)

- 前向神经网络/深层感知机
- 卷积神经网络（图像处理）
- 循环神经网络（序列处理）
- 自编码器（无监督学习）
- 生成对抗模型（生成模型，对抗技术）
- 变分自编码器（生成模型，变分技术）

神经网络（Neural Network Models）

- ✓ **Feed-forward Neural Networks/ Multilayer Perceptron**
- ✓ Convolutional Neural Networks(Image processing)
- ✓ Recurrent Neural Networks(Sequential processing)
- ✓ Auto-encoders(unsupervised learning)
- ✓ Generative Adversarial Networks(Generative Model, Adversarial techniques)
- ✓ Variational Auto-encoders(Generative Model, Variational techniques)

- **前向神经网络/深层感知机**
- 卷积神经网络（图像处理）
- 循环神经网络（序列处理）
- 自编码器（无监督学习）
- 生成对抗模型（生成模型，对抗技术）
- 变分自编码器（生成模型，变分技术）

大纲（Outline）

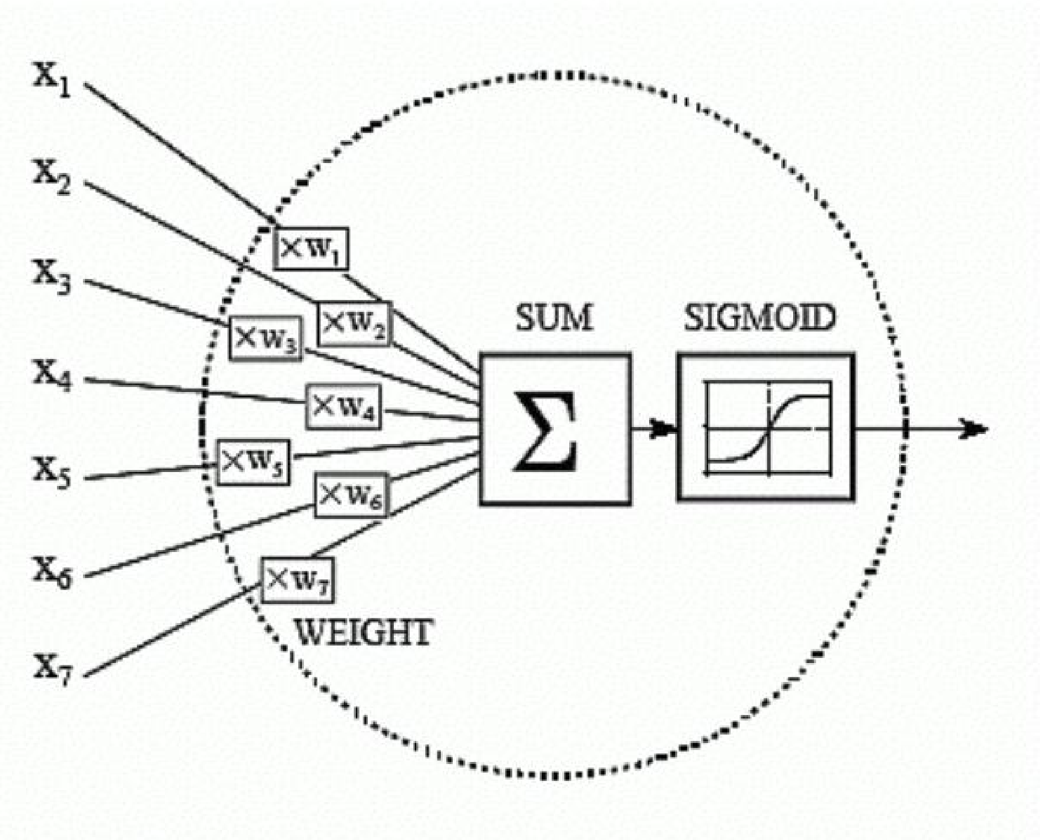
- 深度学习之图像分类
    - 前向运算
    - 神经元种类
    - 神经网络的表达能力
- 神经网络之训练
    - 损失/目标函数
    - 反向传播算法
    - 随机梯度下降
    - 更多高阶技术

大纲（Outline）

- **深度学习之图像分类**
  - **前向运算**
  - **神经元种类**
  - **神经网络的表达能力**
- 神经网络之训练
  - 损失/目标函数
  - 反向传播算法
  - 随机梯度下降
  - 更多高阶技术

深层感知机（Multilayer Perceptron）

基本运算单元-人工神经元



$h(x) = g(a(x)) = g(\sum_i w_i x_i + b)$ $\qquad$ $g(y) = \frac{1}{1+\exp(-y)}$

深层感知机（Multilayer Perceptron）

基本运算单元-人工神经元

$$h(x) = g\big(a(x)\big) = g(\sum_i w_i x_i + b)$$

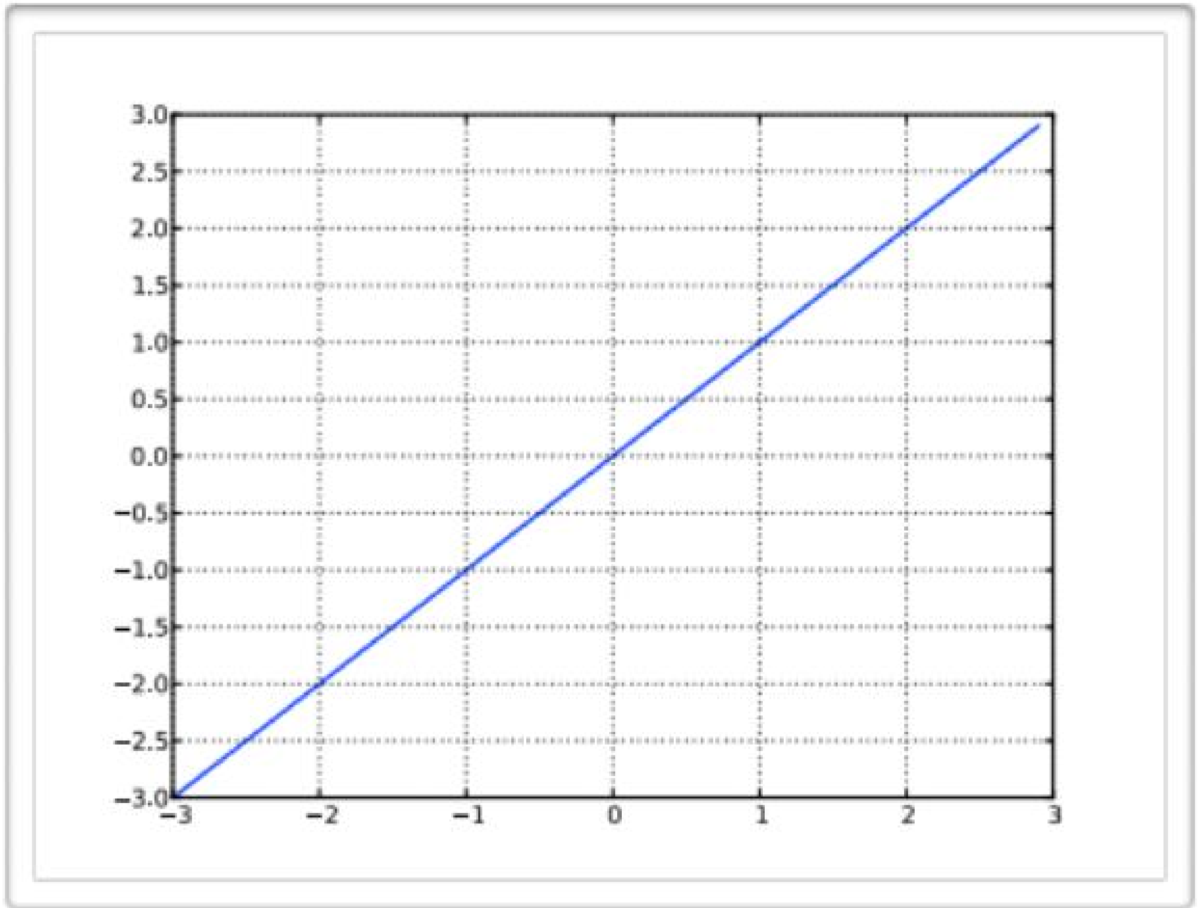
- W：神经元权重（模型参数）
- b：截距（模型参数）
- g：线性/非线性激励函数

关键点：$a(x)$定义为输入信号x的放射变换。激励变换前的$a(x)$为超平面。

深层感知机（Multilayer Perceptron）

激励函数：线性

$$g(a) = a$$



深层感知机（Multilayer Perceptron）

激励函数：非线性神经元Sigmoid函数

$$g(a) = sign(a) = \frac{1}{1+\exp(-a)}$$

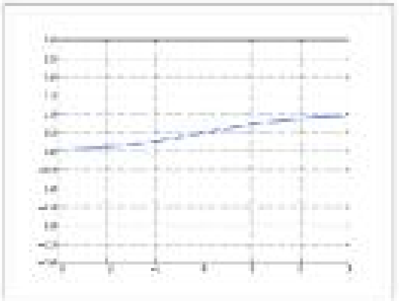
范围 （0， 1）
输出总为正值
输出总是有界的
单调递增

Multilayer Perceptron

- Activation function: sigmoid

$$g(a) = \mathrm{sigm}(a) = \frac{1}{1+\exp(-a)}$$

- Range (0, 1)
- Output is always positive
- Output is bounded
- Monotonously increasing



深层感知机（Multilayer Perceptron）

激励函数：非线性神经元tanh函数

$$g(a) = tanh(a) = \frac{\exp(a) - \exp(-a)}{\exp(a) + \exp(-a)}$$

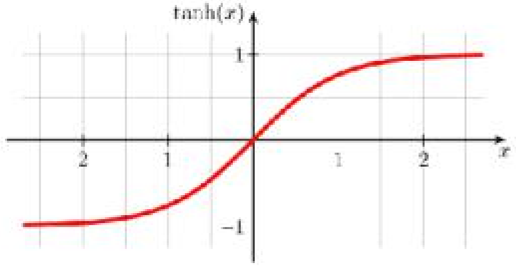
范围 （-1，1）
输出有正有负
输出总是有界的
单调递增

Multilayer Perceptron

- Activation function: tanh

$$g(a) = \tanh(a) = \frac{\exp(a) - \exp(-a)}{\exp(a) + \exp(-a)}$$

- Range (-1, 1)
- Both positive and negative output
- Output is bounded
- Monotonously increasing



深层感知机（Multilayer Perceptron）

激励函数：非线性神经元ReLU函数

$$g(a) = \max\{0, a\}$$

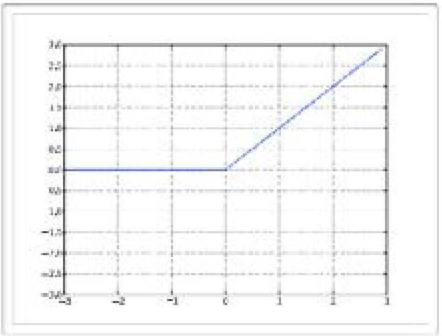
范围无界
输出非负稀疏
输出无界
单调递增

Multilayer Perceptron

- Activation function: ReLU

$$g(a) = \max\{0, a\}$$

- Unbounded range
- Output is non-negative, sparse
- Unbounded output
- Monotonously increasing



深层感知机（Multilayer Perceptron）

激励函数：非线性神经元softmax函数

$$g(\boldsymbol{a}) = softmax(\boldsymbol{a}) = [\frac{\exp(a_1)}{\sum_i \exp(a_i)}, \ldots, \frac{\exp(a_C)}{\sum_i \exp(a_i)}]^T$$

范围$(0,1)^C$, 总和为1；
针对分类问题的选择，可用数学公式表达为Pr（y=c|**x**）；
Sigmoid 激励函数是softmax在C=2时的特殊情况；
关键点：softmax激励函数经常被用于整个网络模型中最后一个激励函数，用于定义多类别分类的损失函数。

Multilayer Perceptron

- Activation function: softmax

$$g(\mathbf{a}) = \mathrm{softmax}(\mathbf{a}) = [\frac{\exp(a_1)}{\sum_i \exp(a_i)}, \ldots, \frac{\exp(a_C)}{\sum_i \exp(a_i)}]^T$$

- Range (0, 1)^C, sums to 1
- The choice for classification problem, can be interpreted as $\Pr(y = c \mid \mathbf{x})$
- sigmoid activation is a special case of softmax when C = 2

Key Point: softmax activation function is usually used as the final activation function of a complex model to define loss function for multi-class classification

深层感知机（Multilayer Perceptron）

深层感知机MLP的特殊情况

带有线性激励函数的一层感知机-- 线性回归；
带有sigmoid激励函数的一层感知机–逻辑回归；

线性回归 $\min_{w}||y - \boldsymbol{w}^{T}\boldsymbol{x}||^{2}$

逻辑回归 $\Pr(y = 1|\boldsymbol{x}) = \frac{1}{1-\exp(-\boldsymbol{w}^{T}\boldsymbol{x})}$

Multilayer Perceptron

- Special cases of MLP:
  - One layer MLP with linear activation function ~ Linear regression
  - One layer MLP with sigmoid activation function ~ Logistic regression
    - Linear regression $\min_{\mathbf{w}} \quad ||y - \mathbf{w}^{T}\mathbf{x}||^{2}$
    - Logistic regression $\Pr(y = 1 \mid \mathbf{x}) = \frac{1}{1+\exp(-\mathbf{w}^{T}\mathbf{x})}$

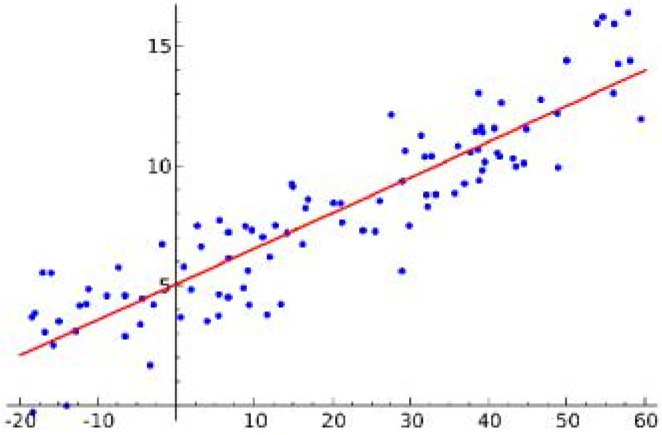
深层感知机（Multilayer Perceptron）

带有线性激励函数的深层感知机– 线性回归

Multilayer Perceptron

- MLP with linear activation ~ Linear regression

$$\hat{y} = g(\mathbf{w}^T\mathbf{x} + b) = \mathbf{w}^T\mathbf{x} + b$$



$$\min_{\mathbf{w}} \frac{1}{2}||Y - X\mathbf{w}||_F^2 \Leftrightarrow \min_{\mathbf{w}} \frac{1}{2}\sum_{i=1}^{n}(y_i - \mathbf{w}^T\mathbf{x}_i)^2 \Leftrightarrow \min_{\mathbf{w}} \frac{1}{2}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

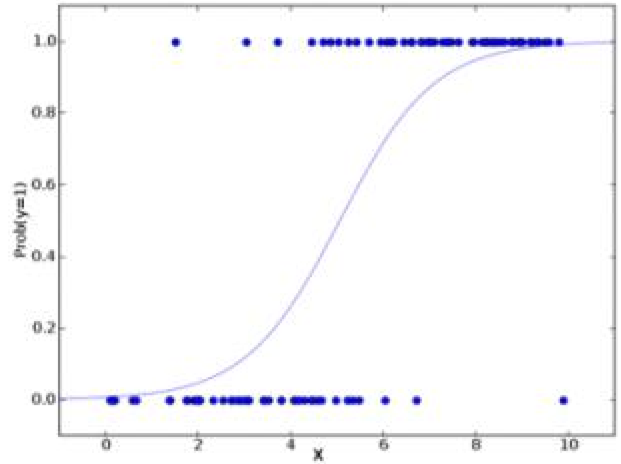
深层感知机（Multilayer Perceptron）

带有sigmoid激励函数的深层感知机– 逻辑回归

Multilayer Perceptron

- MLP with sigmoid activation ~ Logistic regression

$$\Pr(\hat{y}=1 \mid \mathbf{x}) = g(\mathbf{w}^T\mathbf{x}+b) = \frac{1}{1+\exp\{-(\mathbf{w}^T\mathbf{x}+b)\}}$$



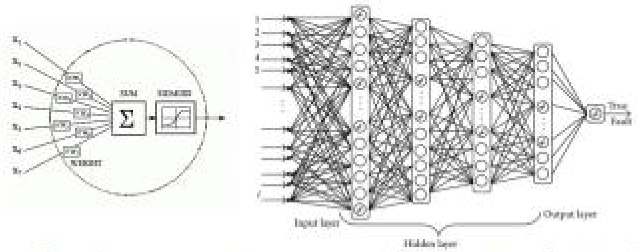
$$\max_{\mathbf{w}} \sum_{i=1}^{n} \mathbb{I}_{y_i=1} \log \Pr(\hat{y}_i = 1 \mid \mathbf{x}_i) + \mathbb{I}_{y_i=0} \log \Pr(\hat{y} = 0 \mid \mathbf{x}_i)$$

深层感知机（Multilayer Perceptron）

Multilayer Perceptron

矩阵形式

- Matrix version:



单个神经元，两步：线性变换+非线性激励

Single neuron, two steps: linear transformation + nonlinear activation

$$a(\mathbf{x}) = \sum_i w_i x_i + b = \mathbf{w}^T\mathbf{x} + b \qquad h(\mathbf{x}) = g(a(\mathbf{x}))$$

单个神经元，两步：线性变换+非线性激励

Single neuron, two steps: linear transformation + nonlinear activation

$$\mathbf{a}(\mathbf{x}) = A\mathbf{x} + b \qquad \mathbf{a}_j(\mathbf{x}) = A_{j,:}\mathbf{x} + b_j = \sum_i A_{ji}x_i + b_j$$

$$\mathbf{h}(\mathbf{x}) = g(\mathbf{a}(\mathbf{x}))$$

g (•)是按照元素相乘来应用的函数

$g(\cdot)$ is applied elementwise.

深层感知机（Multilayer Perceptron）

大纲

用于图像分类的深度神经网络
- 前向计算
- 神经元的类别
- 神经网络的表达力

神经网络的训练
- 损失/目标函数
- 反向传播
- 随机梯度下降
- 更多前沿技术

Outline

- Deep neural networks for image classification
  - Forward computation
  - Types of neurons
  - Expressive power of neural networks
- **Training of neural networks**
  - **Loss/Objective functions**
  - **Backward propagation**
  - **Stochastic gradient descent**
  - **More advanced techniques**

训练

- 经验风险最小化：

学习=正确定义目标函数的数值优化

- 对于分类问题，最小化分类误差

- 当精确地最小化分类误差在计算上很困难时，我们通常使用代理损失函数（eg.一个上限）

Training

- Empirical risk minimization:

$$\min_\theta \quad \frac{1}{n}\sum_{i=1}^{n} \underbrace{\ell(f(\mathbf{x}_i;\theta), y_i)}_{\text{Loss function}} + \underbrace{\lambda\Omega(\theta)}_{\text{Regularizer}}$$

Learning = Numeric optimization of properly defined objective function.

- For classification problems, minimization of classification errors

- When exact minimization of classification errors is computationally hard, we usually use **surrogate loss function (e.g. an upper bound)**

训练

- 随机梯度下降
- 观察每个实例后进行权重更新操作：

1. 模型参数初始化 $\theta = \{W^{(1)}, b^{(1)}, \ldots, W^{(L)}, b^{(L)}\}$
2. For j=1 to T:
    1.对于每个训练样本$(\mathbf{x}_i, y_i)$
    计算随机梯度$\Delta_i = \nabla_\theta \ell(f(\mathbf{x}_i;\theta), y_i) + \lambda \nabla_\theta \Omega(\theta)$
    更新模型参数$\theta \leftarrow \theta - \gamma \Delta_i$

直到收敛时停止

- 关键部分：
- 损失函数 $\ell(f(\mathbf{x};\theta), y)$
- 计算梯度的过程 $\Delta_i = \nabla_\theta \ell(f(\mathbf{x}_i;\theta), y_i) + \lambda \nabla_\theta \Omega(\theta)$

Training

- Stochastic gradient descent
  - Perform weight updates after observing each instance:
    1. Initialization of model parameter $\theta = \{W^{(1)}, b^{(1)}, \ldots, W^{(L)}, b^{(L)}\}$
    2. For j = 1 to T:
        1. For each training example $(\mathbf{x}_i, y_i)$

           Compute stochastic gradient $\Delta_i = \nabla_\theta \ell(f(\mathbf{x}_i;\theta), y_i) + \lambda \nabla_\theta \Omega(\theta)$

           Update model parameter $\theta \leftarrow \theta - \gamma \Delta_i$

    Stop until convergence

  - Key components:
    - Loss function $\ell(f(\mathbf{x};\theta), y)$
    - A procedure to compute gradient $\Delta_i = \nabla_\theta \ell(f(\mathbf{x}_i;\theta), y_i) + \lambda \nabla_\theta \Omega(\theta)$

训练

- 随机梯度下降
- 观察每个实例后进行权重更新操作：

1. 模型参数初始化 $\theta = \{W^{(1)}, b^{(1)}, \ldots, W^{(L)}, b^{(L)}\}$
2. For j=1 to T:

    1.对于每个训练样本 $(\mathbf{x}_i, y_i)$

    计算随机梯度 $\Delta_i = \nabla_\theta \ell(f(\mathbf{x}_i; \theta), y_i) + \lambda \nabla_\theta \Omega(\theta)$

    更新模型参数 $\theta \leftarrow \theta - \gamma \Delta_i$

直到收敛时停止

- 关键部分：
- 损失函数 $\ell(f(\mathbf{x}; \theta), y)$
- 计算梯度的过程 $\Delta_i = \nabla_\theta \ell(f(\mathbf{x}_i; \theta), y_i) + \lambda \nabla_\theta \Omega(\theta)$

Training

- Stochastic gradient descent
  - Perform weight updates after observing each instance:
    1. Initialization of model parameter $\theta = \{W^{(1)}, b^{(1)}, \ldots, W^{(L)}, b^{(L)}\}$
    2. For j = 1 to T:
        1. For each training example $(\mathbf{x}_i, y_i)$

           Compute stochastic gradient $\Delta_i = \nabla_\theta \ell(f(\mathbf{x}_i; \theta), y_i) + \lambda \nabla_\theta \Omega(\theta)$

           Update model parameter $\theta \leftarrow \theta - \gamma \Delta_i$

    Stop until convergence
  - Key components:
    - Loss function $\ell(f(\mathbf{x}; \theta), y)$
    - A procedure to compute gradient $\Delta_i = \nabla_\theta \ell(f(\mathbf{x}_i; \theta), y_i) + \lambda \nabla_\theta \Omega(\theta)$

训练

- 分类问题的损失函数

对于分类问题的神经网络的Softmax输出层

给出输入最大化正确类的对数概率：

等价地，这和最小化我们模型中数据的负对数似然度是一样的

这种损失函数通常被称为多类分类问题的交叉熵损失函数

Training

- Loss function for classification problems

Softmax output layer of neural networks for classification problems

$$f_c(\mathbf{x};\theta) = \Pr(y = c \mid \mathbf{x})$$

Maximize the log-probability of the correct class given an input: $\log \Pr(y = c \mid \mathbf{x})$

Equivalently, this is the same as minimizing the negative log-likelihood of the data under our model:

$$\ell(f(\mathbf{x};\theta), y) = \sum_c \mathbb{I}(y = c) \log \Pr(y = c \mid \mathbf{x}) = \sum_c \mathbb{I}(y = c) \log f_c(\mathbf{x};\theta)$$

This loss function is usually called the **cross-entropy loss function** for multi-class classification problem

训练

- 随机梯度下降
- 观察每个实例后进行权重更新操作：

1. 模型参数初始化 $\theta = \{W^{(1)}, b^{(1)}, \ldots, W^{(L)}, b^{(L)}\}$
2. For j=1 to T:

    1.对于每个训练样本 $(\mathbf{x}_i, y_i)$

    计算随机梯度 $\Delta_i = \nabla_\theta \ell(f(\mathbf{x}_i;\theta), y_i) + \lambda \nabla_\theta \Omega(\theta)$

    更新模型参数 $\theta \leftarrow \theta - \gamma \Delta_i$

直到收敛时停止

- 关键部分：
- 损失函数 $\ell(f(\mathbf{x};\theta), y)$
- 计算梯度的过程 $\boxed{\Delta_i = \nabla_\theta \ell(f(\mathbf{x}_i;\theta), y_i) + \lambda \nabla_\theta \Omega(\theta)}$

Training

- Stochastic gradient descent
- Perform weight updates after observing each instance:

1. Initialization of model parameter $\theta = \{W^{(1)}, b^{(1)}, \ldots, W^{(L)}, b^{(L)}\}$
2. For j = 1 to T:

    1. For each training example $(\mathbf{x}_i, y_i)$

    Compute stochastic gradient $\Delta_i = \nabla_\theta \ell(f(\mathbf{x}_i;\theta), y_i) + \lambda \nabla_\theta \Omega(\theta)$

    Update model parameter $\theta \leftarrow \theta - \gamma \Delta_i$

Stop until convergence

- Key components:
  - Loss function $\ell(f(\mathbf{x};\theta), y)$
  - A procedure to compute gradient $\boxed{\Delta_i = \nabla_\theta \ell(f(\mathbf{x}_i;\theta), y_i) + \lambda \nabla_\theta \Omega(\theta)}$

训练

- 反向传播
- 考虑一个具有L个隐藏层的网络

预激活对于k>0层：

隐藏层激活对于k>0层：

第一层 k=0:

最后一层 k=L:

$\sigma(\cdot)$ 是softmax激活函数

Training

- Backward propagation
- Consider a network with L hidden layers:
  - Pre-activation for k > 0 layer:

$$\mathbf{a}^{(k)}(\mathbf{x}) = W^{(k)}\mathbf{h}^{(k-1)}(\mathbf{x}) + \mathbf{b}^{(k)}$$

  - Hidden layer activation for k > 0 layer:

$$\mathbf{h}^{(k)}(\mathbf{x}) = g(\mathbf{a}^{(k)}(\mathbf{x}))$$

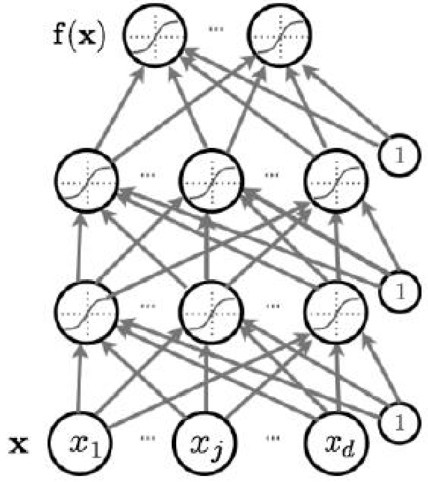
  - First layer k = 0:

$$\mathbf{h}^{(0)}(\mathbf{x}) = \mathbf{x}$$

  - Last layer k = L:

$$\mathbf{h}^{(L)}(\mathbf{x}) = f(\mathbf{x}) = \sigma(\mathbf{a}^{(L)}(\mathbf{x}))$$

$\sigma(\cdot)$ is the softmax activation function.



训练

- 反向传播
- 关键思路：链式法则

关于输出神经元的偏导数

把所有的值写成矩阵形式

Training

- Backward propagation
- **Key idea: Chain rule**

$$\ell(f(\mathbf{x}), y) = -\sum_c \mathbb{I}(y = c) \log \Pr(y = c \mid \mathbf{x}) = -\sum_c \mathbb{I}(y = c) \log f_c(\mathbf{x})$$

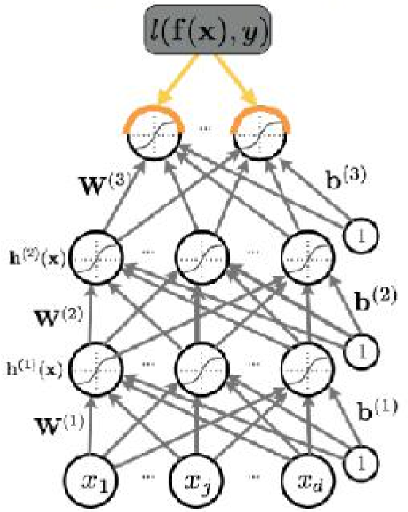
Partial derivative w.r.t. output neuron

$$\frac{\partial \ell(f(\mathbf{x}), y)}{\partial f_c(\mathbf{x})} = -\frac{\mathbb{I}(y=c)}{f_c(\mathbf{x})}$$

Put everything into matrix form:

$$\nabla_{f(\mathbf{x})} \ell = -\begin{pmatrix} \mathbb{I}(y=1)/f_1(\mathbf{x}) \\ \vdots \\ \mathbb{I}(y=C)/f_C(\mathbf{x}) \end{pmatrix}$$

$$= -\frac{1}{f_y(\mathbf{x})} \begin{pmatrix} \mathbb{I}(y=1) \\ \vdots \\ \mathbb{I}(y=C) \end{pmatrix} = -\frac{\mathbf{e}(y)}{f_y(\mathbf{x})}$$



训练

- 反向传播
- 关键思路：链式法则

关于输出神经元预激活函数的偏导数

把所有的值写成矩阵形式

思考两分钟，看看其原因？

Training

- Backward propagation
- **Key idea: Chain rule**

$$\ell(f(\mathbf{x}), y) = -\sum_c \mathbb{I}(y=c)\log \Pr(y=c \mid \mathbf{x}) = -\sum_c \mathbb{I}(y=c)\log f_c(\mathbf{x})$$

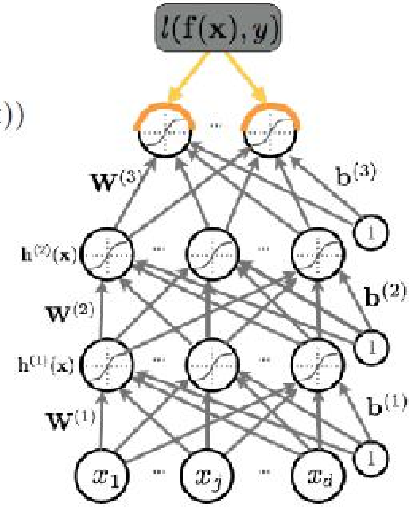
Partial derivative w.r.t. pre-activation of output neuron

$$\frac{\partial \ell(f(\mathbf{x}),y)}{\partial a_c^{(L)}(\mathbf{x})} = -\frac{\partial \log f_y(\mathbf{x})}{\partial a_c^{(L)}(\mathbf{x})} = -\frac{1}{f_y(\mathbf{x})}\frac{\partial f_y(\mathbf{x})}{\partial a_c^{(L)}(\mathbf{x})} = -(\mathbb{I}(y=c) - f_c(\mathbf{x}))$$

Put everything into matrix form:

$$\nabla_{a^{(L)}(\mathbf{x})}\ell = -(\mathbf{e}(y) - f(\mathbf{x}))$$

Think for 2 mins, why?



训练

- 证明

Training

- Proof

$$\frac{\partial \ell(f(\mathbf{x}),y)}{\partial a_c^{(L)}(\mathbf{x})} = \frac{\partial \ell(f(\mathbf{x}),y)}{\partial f_y(\mathbf{x})} \frac{\partial f_y(\mathbf{x})}{\partial a_c^{(L)}(\mathbf{x})} = -\frac{1}{f_y(x)} \frac{\partial f_y(\mathbf{x})}{\partial a_c^{(L)}(\mathbf{x})}$$

$$\begin{aligned}
\frac{\partial f_y(\mathbf{x})}{\partial a_c^{(L)}(\mathbf{x})} &= \frac{\partial}{\partial a_c^{(L)}} \left(\frac{\exp(a_y^{(L)}(\mathbf{x}))}{\sum_{c'} \exp(a_{c'}^{(L)}(\mathbf{x}))} \right) \\
&= \frac{\mathbb{I}(y=c)\exp(a_y^{(L)}(\mathbf{x})) \left(\sum_{c'} \exp(a_{c'}^{(L)}(\mathbf{x})) \right) - \exp(a_c^{(L)}(\mathbf{x}))\exp(a_y^{(L)}(\mathbf{x}))}{\left(\sum_{c'} \exp(a_{c'}^{(L)}(\mathbf{x})) \right)^2} \\
&= \mathbb{I}(y=c) f_y(\mathbf{x}) - f_c(\mathbf{x}) f_y(\mathbf{x})
\end{aligned}$$

$$\Rightarrow \quad \frac{\partial \ell(f(\mathbf{x}),y)}{\partial a_c^{(L)}(\mathbf{x})} = -(\mathbb{I}(y=c) - f_c(\mathbf{x}))$$

训练

- 反向传播
- 关键思路：链式法则

关于第k层的中间神经元的偏导数

把所有的值写成矩阵形式

Training

- Backward propagation
- **Key idea: Chain rule**

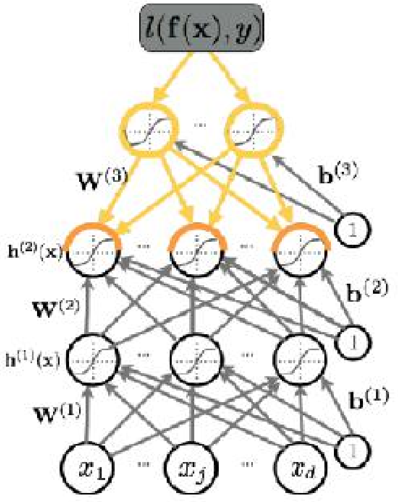
$$\mathbf{h}^{(k)}(\mathbf{x}) = g(\mathbf{a}^{(k)}(\mathbf{x})) \qquad \mathbf{a}^{(k)}(\mathbf{x}) = W^{(k)}\mathbf{h}^{(k-1)}(\mathbf{x}) + \mathbf{b}^{(k)}$$

Partial derivative w.r.t. intermediate neuron at layer k:

$$\frac{\partial \ell(f(\mathbf{x}, y))}{\partial h_i^{(k)}(\mathbf{x})} = \sum_j \frac{\partial \ell(f(\mathbf{x}), y)}{\partial a_j^{(k+1)}(\mathbf{x})} \frac{\partial a_j^{(k+1)}(\mathbf{x})}{\partial h_i^{(k)}(\mathbf{x})}$$
$$= \sum_j \nabla_{a_j^{(k+1)}} \ell \cdot W_{ji}$$
$$= (W^T \nabla_{a^{(k+1)}} \ell)_i$$

Put everything into matrix form:

$$\nabla_{h^{(k)}(\mathbf{x})} \ell = W^T \nabla_{a^{(k+1)}} \ell$$



训练

- 反向传播
- 关键思路：链式法则

关于第k层的中间神经元预激活函数的偏导数

把所有的值写成矩阵形式

Training

- Backward propagation
- **Key idea: Chain rule**

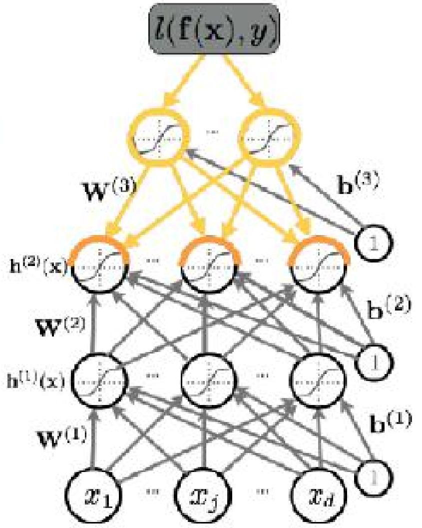
$$\mathbf{h}^{(k)}(\mathbf{x}) = g(\mathbf{a}^{(k)}(\mathbf{x})) \qquad \mathbf{a}^{(k)}(\mathbf{x}) = W^{(k)}\mathbf{h}^{(k-1)}(\mathbf{x}) + \mathbf{b}^{(k)}$$

Partial derivative w.r.t. pre-activation of intermediate neuron at layer k:

$$\frac{\partial \ell(f(\mathbf{x},y))}{\partial a_i^{(k)}(\mathbf{x})} = \frac{\partial \ell(f(\mathbf{x},y))}{\partial h_i^{(k)}(\mathbf{x})} \frac{\partial h_i^{(k)}(\mathbf{x})}{\partial a_i^{(k)}(\mathbf{x})} = \nabla_{h_i^{(k)}(\mathbf{x})} \ell \cdot g'(a_i^{(k)}(\mathbf{x}))$$

Put everything into matrix form:

$$\nabla_{a^{(k)}(\mathbf{x})} \ell = \nabla_{h^{(k)}(\mathbf{x})} \ell \odot g'(h^{(k)}(\mathbf{x}))$$



训练

- 反向传播

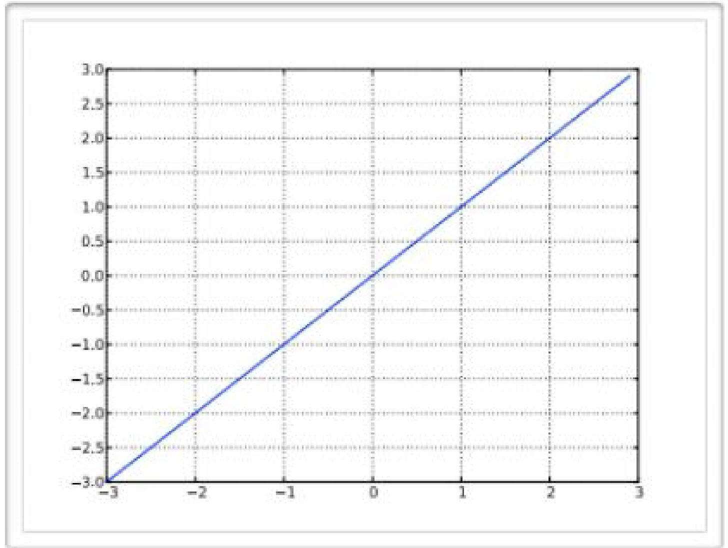
激活函数的导数：线性

Training

- Backward propagation

Derivatives of activation functions: linear

$$g(a) = a \quad \Rightarrow \quad g'(a) = 1$$



训练

- 反向传播

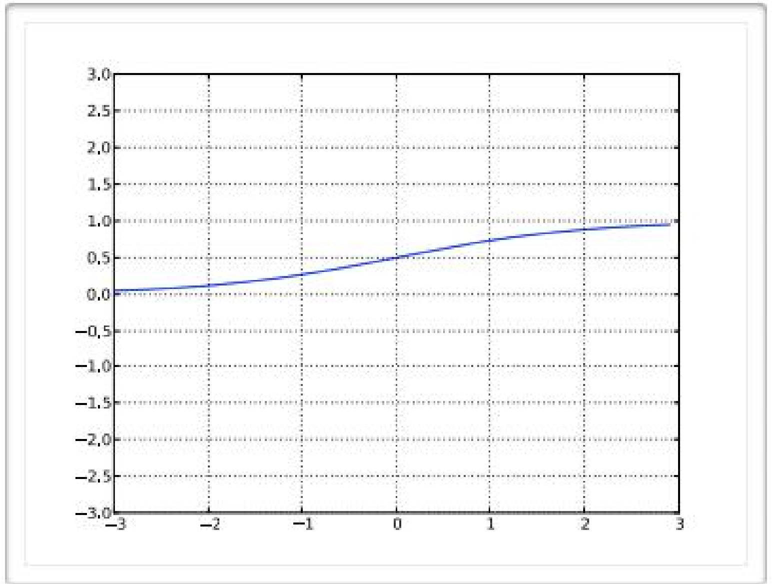
激活函数的导数：S型

Training

- Backward propagation

Derivatives of activation functions: sigmoid

$$g(a) = \mathrm{sigm}(a) = \frac{1}{1+\exp(-a)} \quad \Rightarrow \quad g'(a) = g(a)(1-g(a))$$



训练

Training

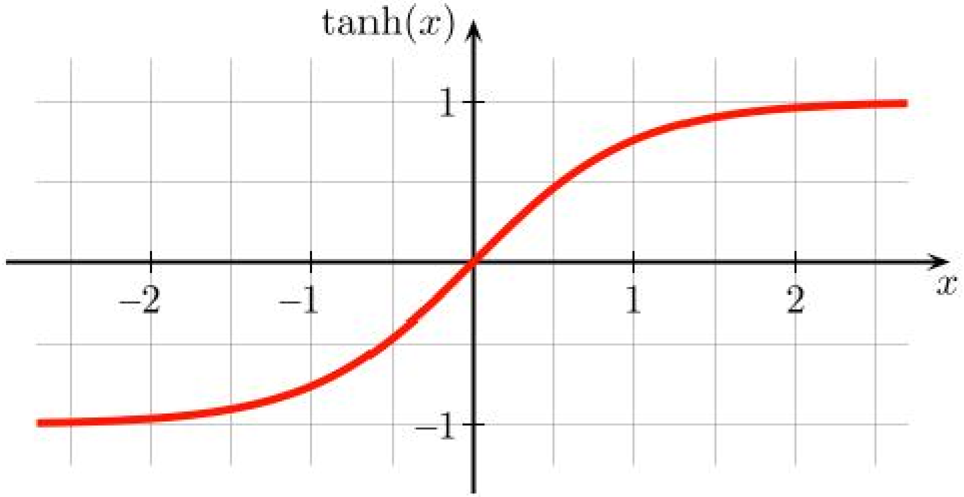
- 反向传播

  激活函数的导数：正切

- Backward propagation

Derivatives of activation functions: tanh

$$g(a) = \tanh(a) = \frac{\exp(a)-\exp(-a)}{\exp(a)+\exp(-a)} \quad \Rightarrow \quad g'(a) = 1 - g^2(a)$$



训练

Training

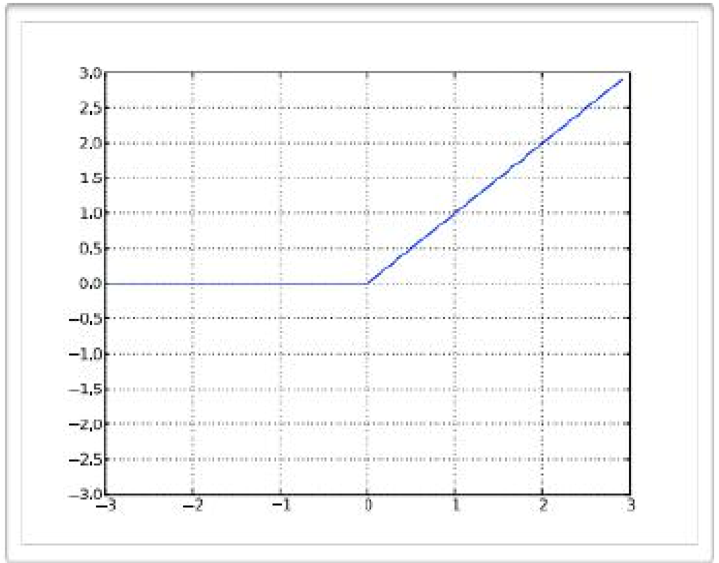
- 反向传播

激活函数的导数：修正线性单元

- Backward propagation

Derivatives of activation functions: ReLU

$$g(a) = \max\{0, a\} \quad \Rightarrow \quad g'(a) = \mathbb{I}(a \geq 0)$$



训练

- 梯度计算
- 关键思路：链式法则

关于在第k层模型权重的偏导数

把所有的值写成矩阵形式

主要是前向评估信号和后向微分信号的外积

Training

- Gradient computation
- **Key idea: Chain rule**

$$\mathbf{a}^{(k)}(\mathbf{x}) = W^{(k)}\mathbf{h}^{(k-1)}(\mathbf{x}) + \mathbf{b}^{(k)}$$

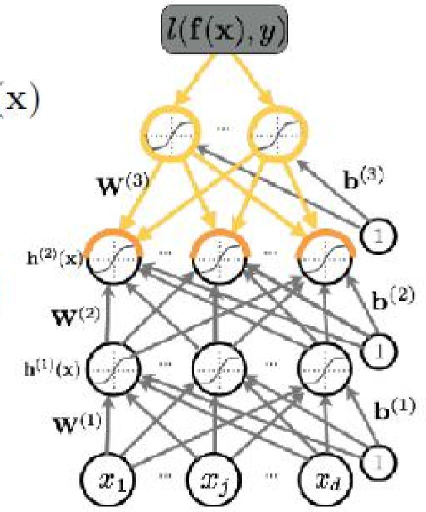
Partial derivative w.r.t. model weight at layer k:

$$\frac{\partial \ell(f(\mathbf{x}),y)}{\partial W_{ij}^{(k)}} = \frac{\partial \ell(f(\mathbf{x}),y)}{\partial a_i^{(k+1)}(\mathbf{x})} \frac{\partial a_i^{(k+1)}(\mathbf{x})}{\partial W_{ij}^{(k)}} = \nabla_{a_i^{(k+1)}(\mathbf{x})}\ell \cdot h_j^{(k)}(\mathbf{x})$$

Put everything into matrix form:

$$\frac{\partial \ell(f(\mathbf{x}),y)}{\partial W^{(k)}} = \nabla_{a^{(k+1)}(\mathbf{x})}\ell \cdot h^{(k)^T}(\mathbf{x})$$

Basically, outer product of forward evaluation signal and backward differentiation signal



BIG DATA DIGEST 大数据文摘 稀牛学院 www.xiniuedu.com

训练

- 梯度计算
- 关键思路：链式法则

关于在第k层截距权重的偏导数
家庭作业

把所有的值写成矩阵形式
家庭作业

Training

- Gradient computation
- **Key idea: Chain rule**

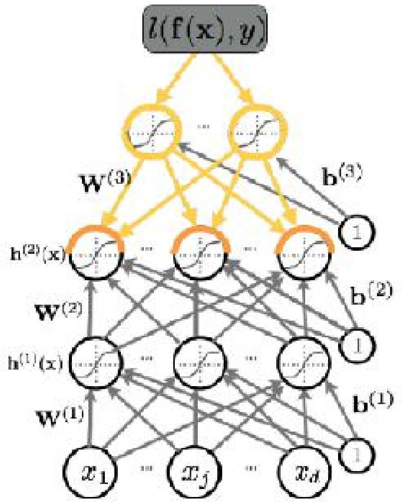
$$\mathbf{a}^{(k)}(\mathbf{x}) = W^{(k)}\mathbf{h}^{(k-1)}(\mathbf{x}) + \mathbf{b}^{(k)}$$

Partial derivative w.r.t. intercept weight at layer k:

Homework

Put everything into matrix form:

Homework



反向传播算法

- 执行前向传播
- 在每个softmax输出层计算梯度（预激活）：

$$\nabla_{a^{(L)}(\mathbf{x})}\ell = -(\mathbf{e}(y) - f(\mathbf{x}))$$

- K = L至 1：
- 计算关于模型第k层的梯度参数：

$$\nabla_{W^{(k)}}\ell = \nabla_{a^{(k+1)}(\mathbf{x})}\ell \cdot h^{(k)^T}(\mathbf{x}), \quad \nabla_{b^{(k)}} = ?$$

计算关于中间神经元第k层的梯度参数：

$$\nabla_{h^{(k)}(\mathbf{x})}\ell = W^T \nabla_{a^{(k+1)}}\ell$$

计算关于中间神经元第k层的预激活的梯度参数：

$$\nabla_{a^{(k)}(\mathbf{x})}\ell = \nabla_{h^{(k)}(\mathbf{x})}\ell \odot g'(h^{(k)}(\mathbf{x}))$$

正则化

- 权重衰减（L2正则化）：

$$\Omega(\theta) = \sum_k \sum_{ij} (W_{ij}^{(k)})^2 = \sum_k ||W^{(k)}||_F^2$$

- 梯度计算：

$$\nabla_{W^{(k)}} \Omega(\theta) = 2W^{(k)}$$

- 防止过拟合
- 等价于模型权重的高斯先验MAP推断
- 仅应用于权重（没有截距）

初始化

- 初始化全部 截距(b)=0
- 初始化模型权重并归一化到[-U,U]

$$U = \frac{\sqrt{6}}{\sqrt{\text{fan\_in}+\text{fan\_out}}}$$

- fan_in = 最后一层神经元
- fan_out = 现在层神经元
- 初始化对成功训练神经网络非常重要

总结

- 主要思想：有向无环计算图
- 前向传播：以模块化的方式模块整个计算过程
- 每个模块计算它给定的孩子值当作输入。

- 反向传播：每个模块计算梯度损失
- 反向传播工作是前向传播的逆过程。

- 这两个特性使自动微分成为可能,下节课我们将讨论更多。

作业

- 用3层的MLP进行MNIST数据集的图像分类
- 图像大小为28*28，MLP大小为784-500-10，用ReLU进行非线性激活函数
- 批大小= 200
- 学习率 = 0.1
- 迭代 = 20
- 我们提供了一个初始的Python脚本给你练习

- 你需要实现：
- 前向计算
- 梯度的反向传播
- 用小批量梯度下降进行训练
- 如果正确实现,您应该看到一个测试集分类准确性~ 0.935
- 在3.6GHz上的CPU上运行，大概要45秒

- 你需要安装numpy和tensorflow

作者与译者介绍

BIG DATA DIGEST 大数据文摘　稀牛学院 www.xiniuedu.com

作者

Ivan老师

前Google AI实验室工程师。清华大学本科，CMU PhD，在AAAI、IJCAI、AISTATS等顶级会议上发表过多篇论文。熟练机器学习、深度学习与统计、凸优化。

翻译团队成员

邹远炳（组长）

中国农业大学机器学习、计算机视觉放方向硕士，流式计算JStorm研究者，曾参加北京市互联网创业大赛获三等奖，现有一篇关于流式计算的论文已投。

王姝

四川大学电子信息学院本科，帝国理工信号处理专业硕士，伦敦大学国王学院准博士；深度学习入门级选手，但是喜欢数学和前沿科技，曾参加中兴算法大赛等代码大赛，之前曾发表一篇一作压缩传感领域SCI会议论文，现有两篇三作关于超分辨率的会议论文已投。

王鑫同

华南理工大学计算机学院本科，保送本校研究生。本科研究方向时空大数据分布式存储与计算，本科阶段曾发表EI期刊论文一篇，国际会议论文两篇。硕士研究方向大数据智能分析，发表国际会议论文一篇。

肖潇

计算机视觉方向研究生，有出国经历，多篇相关方向论文发表（资料待完善）。

翻译说明

Deep Learning领域最新突破常以英文发表和公布，所以英文课件的表述更精准。

《深度学习与计算机视觉》课程的学员以Case的形式，对照讲师发布的原英文课件，进行了中文的翻译和校对，以降低英语基础较弱学员的学习难度。

翻译团队为精准、一致的翻译进行了多种努力，但仍然难免疏漏。欢迎学员提出修改意见和建议！

最后，英语不难哦！希望大家不要有畏难情绪~

课程合作请联系

info@bigdatadigest.cn

BIG DATA DIGEST 大数据文摘 x 稀牛学院

扫码获取最新数据科学资讯