机器学习基础

何世柱
(shizhu.he@nlpr.ia.ac.cn)
中国科学院自动化研究所
模式识别国家重点实验室

目录

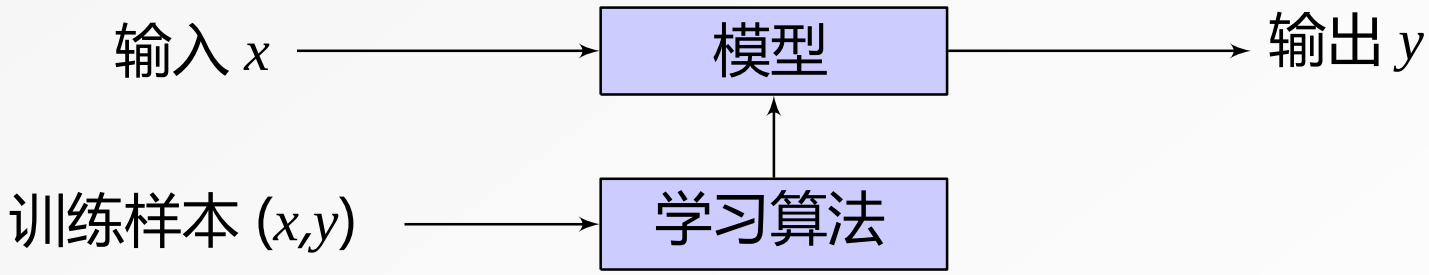
目录

机器学习

- 机器学习(Machine Learning, ML)是一门多领域交叉学科，涉及概率论、统计学、逼近论、凸分析、算法复杂度理论等多门学科。专门研究计算机怎样模拟或实现人类的学习行为，以获取新的知识或技能，重新组织已有的知识结构使之不断改善自身的性能；
- 机器学习是人工智能的一个分支,其目的在于使得机器可以根据数据进行自动学习,**通过算法使得机器能从大量历史数据中学习规律从而对新的样本做决策**；
- 它目前是人工智能的核心，是使计算机具有智能的根本途径，其应用遍及人工智能的各个领域，它主要使用归纳、综合而不是演绎。

机器学习

- 机器学习主要是研究如何使计算机从给定的数据中学习规律，即从观测数据（样本）中寻找规律，并利用学习到的规律（模型）对未知或无法观测的数据进行预测。目前，主流的机器学习算法是基于统计的方法，也叫统计机器学习



规则思维 vs. 统计思维

- 规则思维
  - 人：总结数据/应用中的规律
  - 机器：应用规律，计算机让这些规律大规模更便携地应用
- 统计思维
  - 人：提供数据，定义任务
  - 机器：自动学习数据中的规律（对应具体任务），然后应用规律

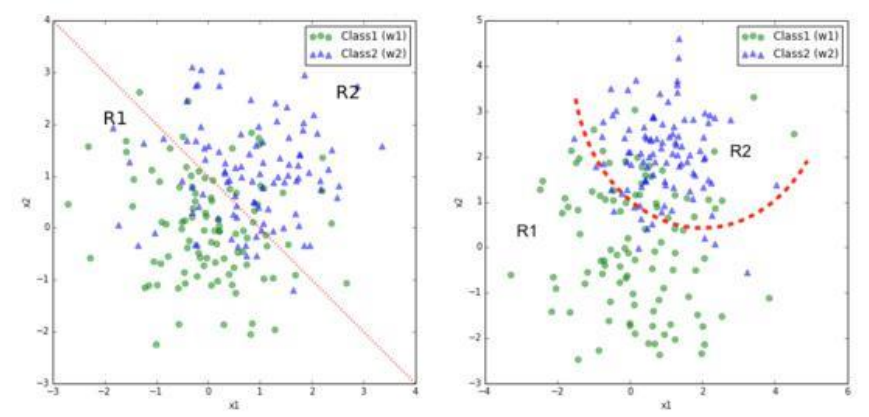
机器学习方法分类

- **监督学习**：从给定的训练数据集中学习出一个函数，当新的数据到来时，可以根据这个函数预测结果。训练集中的目标是由人标注的。常见的监督学习算法包括回归分析和统计分类。
- **无监督学习**：训练集没有人为标注的结果。常见的无监督学习算法有聚类。
- **半监督学习**：介于监督学习与无监督学习之间，部分数据有人为标注，部分数据没有。
- **增强学习**：强调如何基于环境而行动，以取得最大化的预期利益。它并不需要出现正确的输入/输出对。强化学习更加专注于规划，需要在探索（在未知的领域）和遵从（现有知识）之间找到平衡。

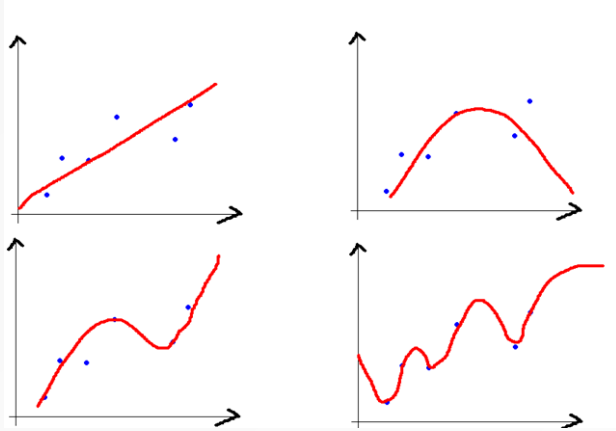
基本机器学习问题类型

- **分类（Classification）：** y 是离散的类别标记（符号），就是分类问题。损失函数有一般用 0-1 损失函数或负对数似然函数等。在分类问题中，通过学习得到的决策函数 f (x,θ) 也叫分类器。
- **回归（Regression）：** y 是连续值（实数或连续整数），f (x) 的输出也是连续值。这种类型的问题就是回归问题。对于所有已知或未知的 (x, y)，使得 f (x,θ) 和 y 尽可能地一致。损函数通常定义为平方误差。
- **聚类（ Clustering）：** 只有原始数据x，没有确定的目标$f(x)$，它基于数据的内部结构寻找观察样本的自然族群（即集群）。聚类的特点是训练数据没有标注，通常使用数据可视化等方式评价结果。

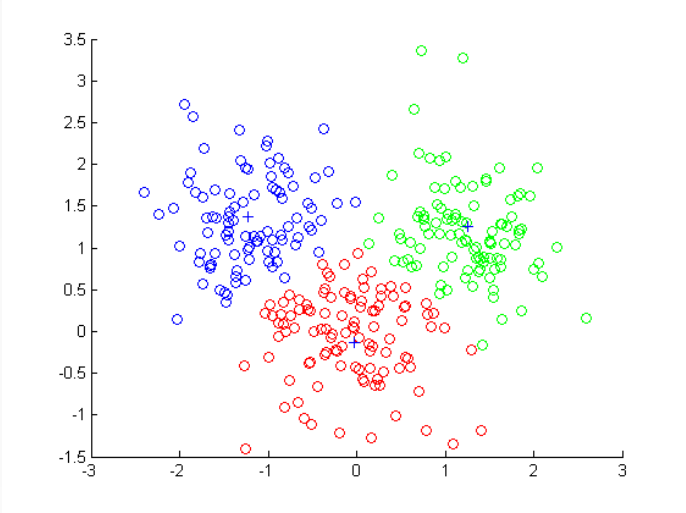
分类、回归和聚类



分类



回归



聚类

机器学习三要素

- **模型**：首先要考虑的问题是学习什么样的模型。在监督学习中，模型就是所要学习的条件概率函数或决策函数。模型的假设空间包含所有可能的条件概率函数或决策函数。决策函数表示的模型为非概率模型，条件概率的模型为概率模型。
- **策略**：有了模型的假设空间，接着需要考虑是按上面准则学习或者选择最优模型，统计学习的目标在于从假设空间中选取最优模型。
- **算法**：根据学习策略，从假设空间中选择最优的模型的计算方法。往往这个时候就将问题转化为最优化问题。通常问题的解析解不存在，需要用数值计算的方法求解，如何保证找到全局最优解就是个重要问题。

方法 = 模型 + 策略 + 算法

机器学习三要素

- 模型
  - 线性模型，非线性模型；概率模型，非概率模型；...
- 策略
  - 损失函数和风险函数：损失函数度量模型一次预测的好坏，风险函数度量平均意义下模型预测的好坏。（如：均方差、交叉熵）
  - 经验风险最小化与结构风险最小化：经验风险最小化的策略就是认为经验风险最小的模型就是最优的模型，在样本足够大时，可以保证有良好的学习效果。（如：极大似然估计，最大后验概率）
- 算法
  - 迭代学习（如EM），梯度下降法等最优化算法。

机器学习方法示例

训练数据： $(x_i, y_i), 1 \le i \le m$

模型：

线性方法：$y = f(x) = \boldsymbol{w}^T\boldsymbol{x} + b$

非线性方法：神经网络

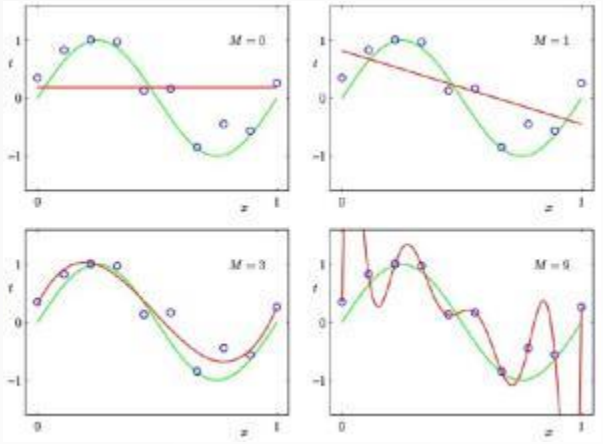
策略：

损失函数：$L(y, f(x))$

经验风险最小化：$Q(\theta) = \frac{1}{m} \cdot \sum_{i=1}^{m} L(y_i, f(x_i, \theta))$

正则化：$\|\theta\|^2$

优化目标函数：$Q(\theta) + \lambda\|\theta\|^2$



奥卡姆剃刀原则

机器学习

狭义地讲，机器学习是给定一些训练样本 $(x_i, y_i), 1 \le i \le N$ （其中 x_i 是　　输入，y_i 是需要预测的目标），让计算机自动寻找一个决策函数 $f(\cdot)$ 来建立 x 和 y 之间的关系。

$$\hat{y} = f(\Phi(x), \theta)$$

这里，$\hat{y}$是模型输出, θ为决策函数的参数，$\Phi(x)$表示样本 x 对应的特征表示。

因为 x 不一定都是数值型的输入，因此需要通过 $\Phi(x)$ 将 x 转换为数值型的输入。

损失函数

在机器学习算法中，一般定义一个**损失函数** $L(y, f(x, \theta))$，在所有的训练样本上来评价决策函数的好坏（风险）。

$$R(\theta)=\frac{1}{N}\sum_{i=1}^{N}L\left(y^{(i)}, f\left(x^{(i)},\theta\right)\right)$$

风险函数 R(θ) 是在已知的训练样本（经验数据）上计算得来的，因此被称之为经验风险。参数的求解其实就是寻求一组参数，使得经验风险函数达到最小值，就是我们常说的经验风险最小化原则（Empirical Risk Minimization）

$$\theta^{*}=\arg\min_{\theta} R(\theta)$$

损失函数

如何度量错误的程度。

0-1 损失函数

$$L\left(y, f\left(x,\theta\right)\right)=\begin{cases}0 & if \quad y=f\left(x,\theta\right)\\ 1 & if \quad y\neq f\left(x,\theta\right)\end{cases}$$
$$=l\left(y\neq f\left(x,\theta\right)\right)$$

平方损失函数

$$L\left(y,\hat{y}\right)=\left(y-f\left(x,\theta\right)\right)^2$$

损失函数

交叉熵损失函数 对于分类问题，模型输出$f(x, \theta)$为每个类y的条件概率。

假设$y \in \{1,\cdots,C\}$，模型预测样本属于第 i 个类的条件概率$P(y = i \mid x) = f_i(x,\theta)$，则$f(x,\theta)$满足

$$f_i(x,\theta) \in [0,1], \quad \sum_{i=1}^{C} f_i(x,\theta) = 1$$

$f_y(x,\theta)$可以看作对于所标注类别y的似然函数。参数可以直接用最大似然估计来优化。考虑到计算问题，我们经常使用最小化负对数似然，即**负对数似然损失函数**（Negative Log Likelihood function）。

$$L(y, f(x,\theta)) = -\log f_y(x,\theta)$$

损失函数

如果我们用 one-hot 向量y来表示目标类别 c，其中只有 $y_c = 1$，其余向量元素都为 0。
则目标函数可以写为：

$$L\left(y, f\left(x,\theta\right)\right) = -\sum_{i=1}^{C} y_i \log f_i\left(x,\theta\right)$$

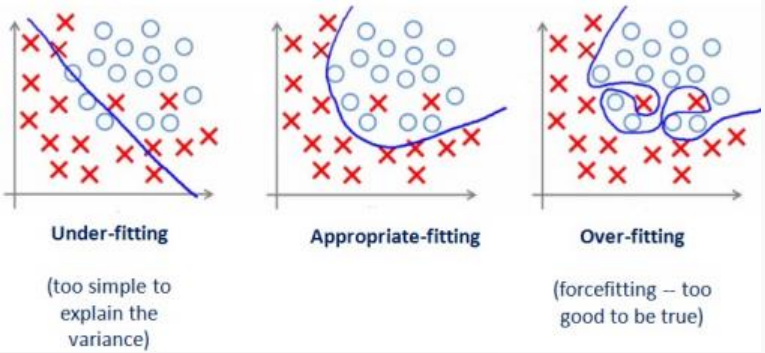
y_i是所标注真实类别的分布，上式恰好是交叉熵的形式。因此，损失函数也称之为交叉熵损失函数（Cross-Entropy Loss function）。

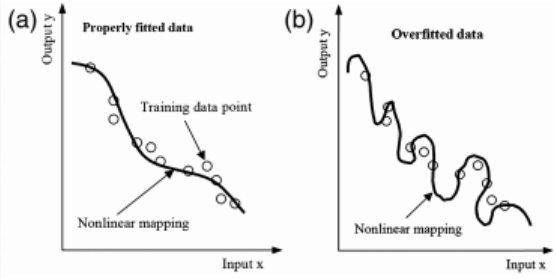
损失函数

Hinge 损失函数 对于两类分类问题，假设 y 和 $f(x,\theta)$ 的取值为 $\{-1,+1\}$。Hinge 损失函数（Hinge Loss Function）的定义如下：

$$L\left(y, f\left(x,\theta\right)\right)=\max\left(0,1-yf\left(x,\theta\right)\right)$$
$$=\left|1-yf\left(x,\theta\right)\right|_{+}$$

过拟合 overfitting





结构风险最小化原则

为了解决过拟合问题，一般在经验风险最小化的原则上加参数的**正则化（**Regularization**）**，也叫**结构风险最小化原则**（Structure Risk Minimization）。

$$\theta^* = \arg\min_{\theta} R(\theta) + \lambda \|\theta\|^2$$

$$= \arg\min_{\theta} \frac{1}{N} \sum_{i=1}^{N} L\left(y^{(i)}, f\left(x^{(i)}, \theta\right)\right) + \lambda \|\theta\|^2$$

正则化项

λ 用来控制正则化的强度，正则化项也可以使用其它函数，比如 $L1$ 范数

学习

在机器学习问题中，我们需要学习到参数 θ，使得风险函数最小化。

$$\theta^* = \arg\min_{\theta} R(\theta_t)$$
$$= \arg\min_{\theta} \frac{1}{N} \sum_{i=1}^{N} L\left(y^{(i)}, f\left(x^{(i)}, \theta\right)\right)$$

如果用梯度下降法进行参数学习，

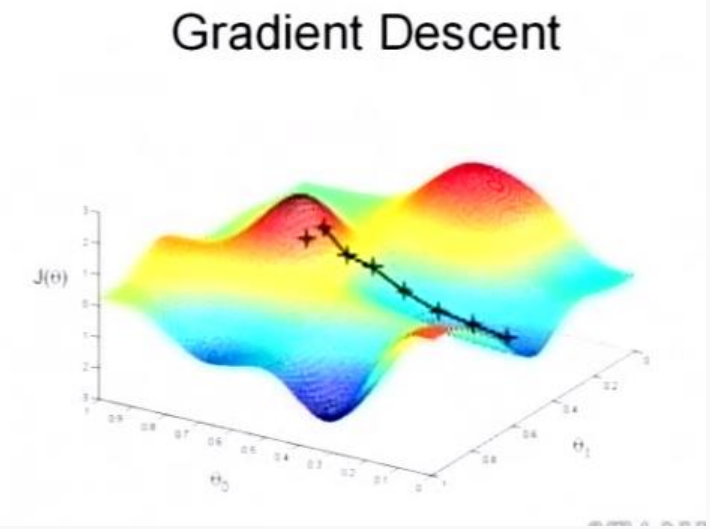
$$a_{t+1} = a_t - \lambda \frac{\partial R(\theta)}{\partial \theta_t}$$
$$= a_t - \lambda \sum_{i=1}^{N} \frac{\partial R\left(\theta_t; x^{(i)}, y^{(i)}\right)}{\partial \theta}$$

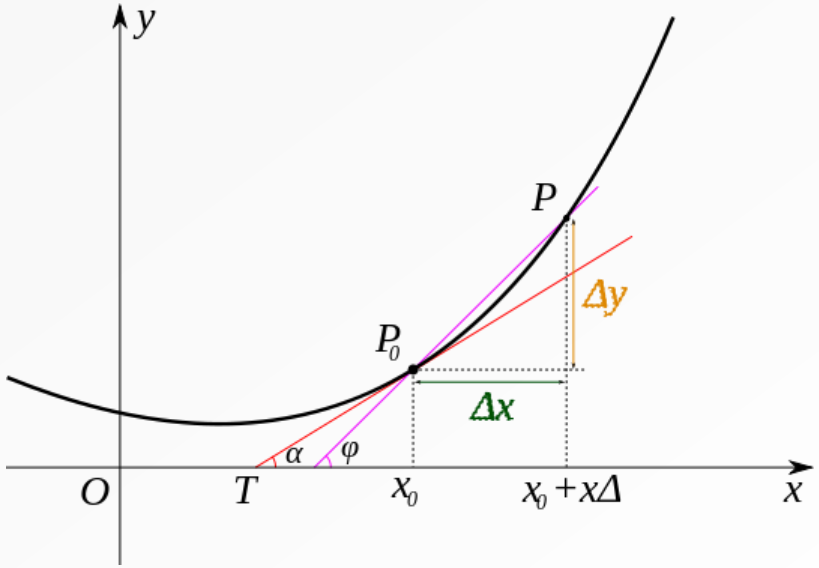
搜索步长 λ 在机器学习中也叫作学习率（Learning Rate）。

梯度下降法

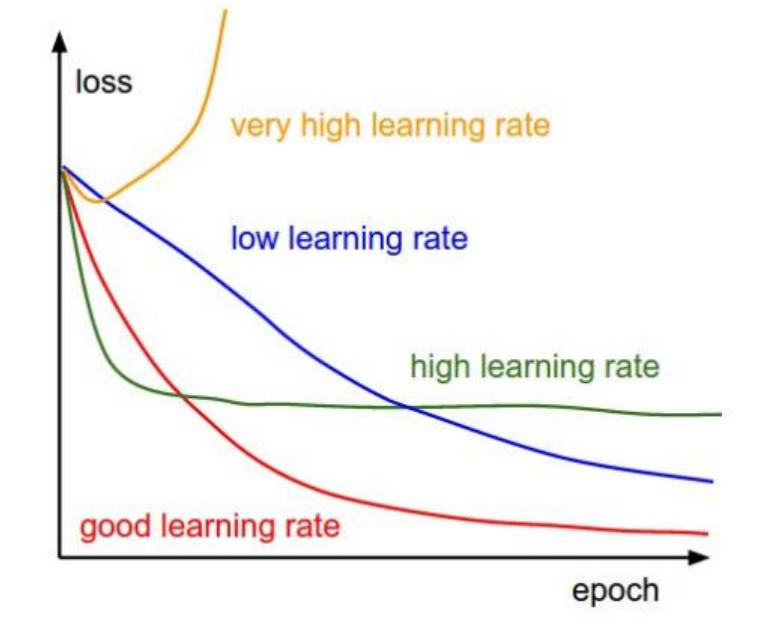
$$a_{t+1}=a_t-\lambda\frac{\partial R(\theta)}{\partial\theta_t}$$

$$=a_t-\lambda\sum_{i=1}^{N}\frac{\partial R\left(\theta_t;x^{(i)},y^{(i)}\right)}{\partial\theta}$$





学习率


loss
very high learning rate
low learning rate
high learning rate
good learning rate
epoch

学习率设置：自适应法

AdaGrad（Adaptive Gradient）算法是借鉴 L_2 正则化的思想。在第 t次迭代时，

$$\theta_t = \theta_{t-1} - \frac{\rho}{\sqrt{\sum_{\tau=1}^{t} g_\tau^2}} g_t$$

其中，ρ是初始的学习率，$g_\tau \in \mathrm{R}^{|\theta|}$ 是第 τ次迭代时的梯度。随着迭代次数的增加，梯度逐渐缩小。

SGD

梯度下降是求得所有样本上的风险函数最小值，叫做**批量梯度下降法**。若样本个数 N 很大，输入 x 的维数也很大时，那么批量梯度下降法每次迭代要处理所有的样本，效率会较低。为此，**有一种改进的方法即随机梯度下降法**。随机梯度下降法（Stochastic Gradient Descent，SGD）也叫**增量梯度下降**，每个样本都进行更新

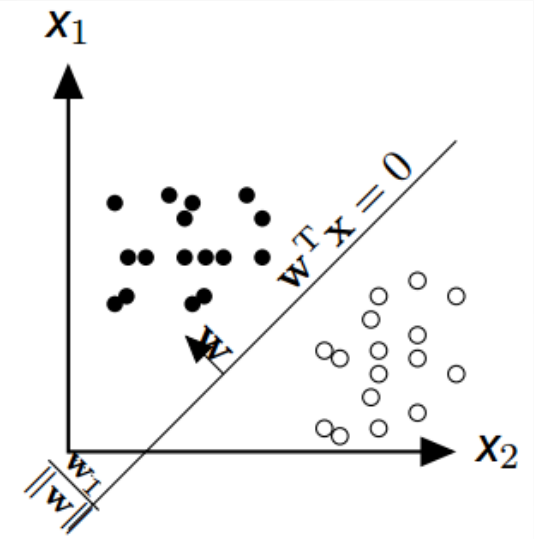
$$a_{t+1}=a_t-\lambda\frac{\partial R\left(\theta_t;x^{(t)},y^{(t)}\right)}{\partial\theta}$$

$x(t),y(t)$ 是第 t 次迭代选取的样本。

线性分类

线性分类是机器学习中最常见并且应用最广泛的一种分类器。

$$\hat{y} = \begin{cases} 1 & if \quad w^T x > 0 \\ 0 & if \quad w^T x \le 0 \end{cases} = l\left(w^T x\right)$$



图：两类分类线性判别函数

Logistic Regression

Logistic 回归

我们定义目标类别 $y = 1$ 的后验概率为：

$$P(y = 1|x) = \sigma(w^T x) = \frac{1}{1 + \exp(-w^T x)}$$

其中，$\sigma(\cdot)$ 为 logistic 函数，x 和 w 为增广的输入向量和权重向量。

$y = 0$ 的后验概率为

$$P(y = 0|x) = 1 - P(y = 1|x) = \frac{\exp(-w^T x)}{1 + \exp(-w^T x)}$$

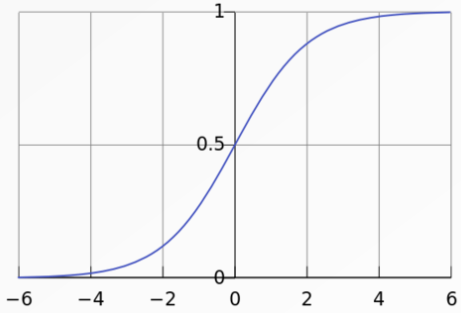
Logistic 函数

logistic 函数经常用来将一个实数空间的数映射到 (0,1) 区间，记为 $\sigma(x)$

$$\sigma(x)=\frac{1}{1+e^{-x}}$$

其导数为

$$\sigma'(x) = \sigma(x)(1 - \sigma(x))$$

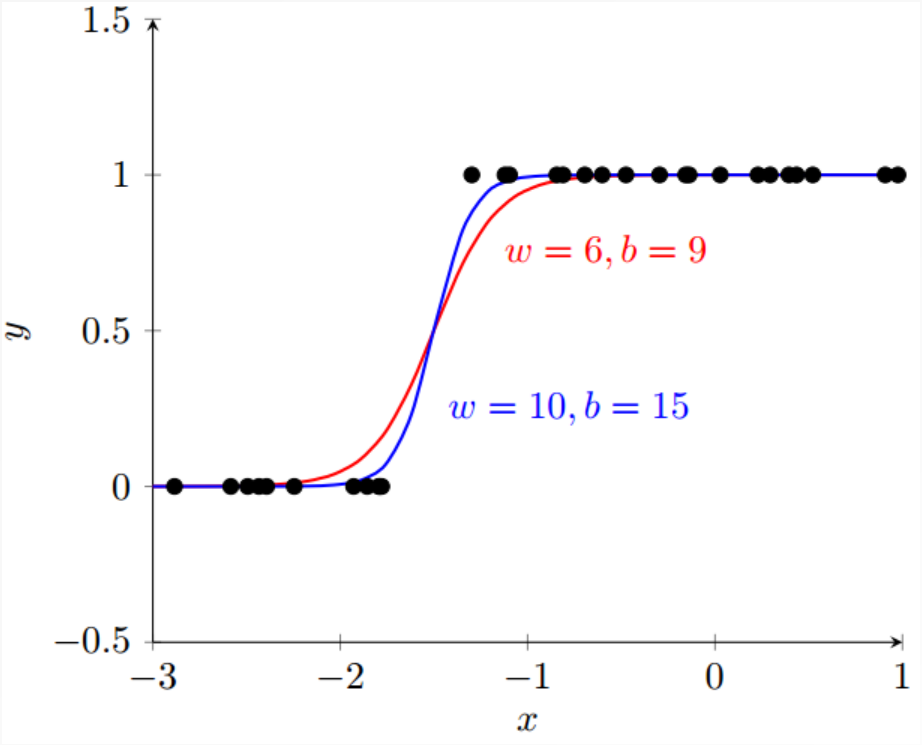


当输入为 K 维向量 $x = [x_1, \cdots, x_K]^T$ 时，其导数为

$$\sigma'(x) = \mathrm{diag}(\sigma(x) \odot (1 - \sigma(x)))$$

Logistic Regression

$y = \sigma(w\,x + b)$



Logistic Regression

给定 N 给样本 $(x^{(i)},y^{(i)}),1 \le i \le N$，我们使用交叉熵损失函数，模型在训练集的风险函数为：

$$
\begin{aligned}
J(w) &= -\sum_{i=1}^{N}\left(y^{(i)}\log\left(\sigma\left(w^T x^{(i)}\right)\right)+\left(1-y^{(i)}\right)\log\left(1-\sigma\left(w^T x^{(i)}\right)\right)\right) \\
&= -\sum_{i=1}^{N}\left(y^{(i)}\log\left(\frac{1}{1+\exp\left(-w^T x^{(i)}\right)}\right)+\left(1-y^{(i)}\right)\log\left(1-\frac{1}{1+\exp\left(-w^T x^{(i)}\right)}\right)\right) \\
&= -\sum_{i=1}^{N}\left(y^{(i)}\log\left(\frac{\exp\left(w^T x^{(i)}\right)}{1+\exp\left(w^T x^{(i)}\right)}\right)+\left(1-y^{(i)}\right)\log\left(\frac{1}{1+\exp\left(w^T x^{(i)}\right)}\right)\right) \\
&= -\sum_{i=1}^{N}\left(w^T x^{(i)} y^{(i)} - \log\left(1+\exp\left(w^T x^{(i)}\right)\right)\right)
\end{aligned}
$$

Logistic Regression

采用梯度下降法，J(w) 关于 w 的梯度为：

$$\frac{\partial J(w)}{\partial w} = -\sum_{i=1}^{N}\left(x^{(i)}y^{(i)} - x^{(i)}\exp\left(w^T x^{(i)}\right)\frac{1}{1+\exp\left(w^T x^{(i)}\right)}\right)$$

$$= -\sum_{i=1}^{N}\left(x^{(i)}y^{(i)} - x^{(i)}\frac{1}{1+\exp\left(-w^T x^{(i)}\right)}\right)$$

$$= -\sum_{i=1}^{N}\left(x^{(i)}\left(y^{(i)} - \sigma\left(w^T x^{(i)}\right)\right)\right)$$

$$= \sum_{i=1}^{N}\left(x^{(i)}\left(\sigma\left(w^T x^{(i)}\right) - y^{(i)}\right)\right)$$

我们可以初始化 $w_0 = 0$，然后用梯度下降法进行更新

$$w_{t+1} = w_t + \lambda\frac{\partial J(w)}{\partial w}$$

Example: logistic regression

- X_data = $\{x_1, x_2, \dots, x_n\}$, Y_data = $\{y_1, y_2, \dots, y_n\}$; $x_i \in R^K, y_i \in \{0,1\}$

$$p(y = 1|x, w) = f(x) = sigmod(w^T x + b) = \frac{1}{1 + e^{-w^T x + b}}$$

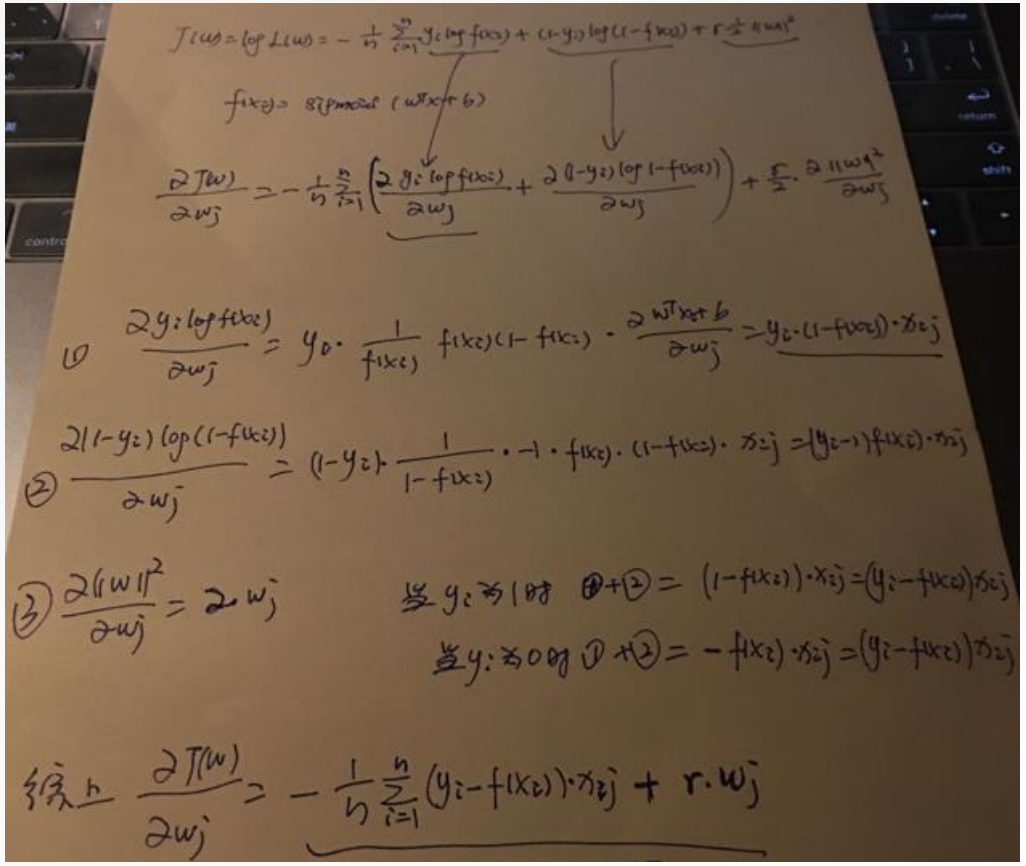
- Loss function:

$$L(w) = \prod_{i=1}^{n} p(yi|xi, w) = \prod_{i=1}^{n} f(xi)^{y_i}(1 - f(x_i))^{1-yi}$$

$$J(w) = logL(w) = -\frac{1}{n}\sum_{i=1}^{n} y_i log f(x_i) + (1 - yi) log\big(1 - f(x_i)\big) + \gamma\frac{1}{2}||w||^2$$

公式推导

- 输入
- 输出
- 模型
  - 参数
- 损失函数



- 损失函数对参数求导
- 根据导数更新参数

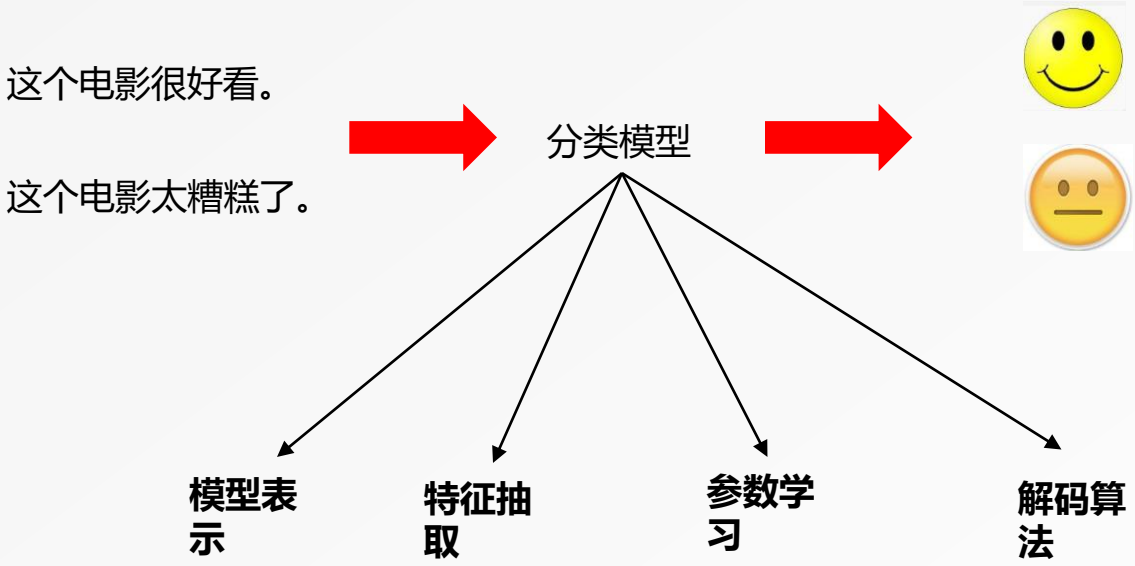
编程实践

```
# 通过求导，更新参数
def _gradientDescend(self, iters):
    """
    X: 输入数据特征
    Y: 输出目标
    theta: 模型参数
    alpha: 学习率
    lam: 正则化权重
   """

    m, n = self.X.shape
    for i in xrange(0, iters):
        theta_temp = self.theta # 参数原始值

        h_theta = self._sigmoid(numpy.dot(self.X, self.theta)) # 模型预测结果
        diff = h_theta - self.Y
        self.theta[0] = theta_temp[0] - self.alpha * \
            (1.0 / m) * sum(diff * self.X[:, 0]) # 偏置项求导及其更新

        for j in xrange(1, n):
            val = theta_temp[
                j] - self.alpha * (1.0 / m) * (sum(diff * self.X[:, j]) + self.lam * m * theta_temp[j])
            # 参数求导及其更新
            self.theta[j] = val
        cost = self._costFunc() # 计算当前参数值的情况下，损失函数是多少，损失函数逐步下降才说明程序可能正确

        if self.printIter:
            print "Iteration", i, "\tcost=", cost
```

```
# sigmoid函数定义
def _sigmoid(self, x):
    z = 1.0 / (1.0 + numpy.exp((-1) * x))
    return z

# 定义损失函数
def _costFunc(self):
    m, n = self.X.shape
    h_theta = self._sigmoid(numpy.dot(self.X, self.theta)) # 模型预测结果

    # 每个样本的模型损失函数
    cost1 = (-1) * self.Y * numpy.log(h_theta)
    cost2 = (1.0 - self.Y) * numpy.log(1.0 - h_theta)

    # 加上参数的正则项
    cost = (
        sum(cost1 - cost2) + 0.5 * self.lam * sum(self.theta[1:] ** 2)) / m
    return cost
```

```
# 给定输入数据，根据模型（模型定义及其参数值），计算预测结果
def predict(self, X):
    m, n = X.shape
    x = numpy.array(X)
    # 数据预处理
    x = (x - self.xMean) / self.xStd
    # 补常数1
    const = numpy.array([1] * m).reshape(m, 1)
    X = numpy.append(const, x, axis=1)
    # 计算预测得分
    pred = self._sigmoid(numpy.dot(X, self.theta))
    numpy.putmask(pred, pred >= 0.5, 1.0) # 得分大于等于0.5则是正例
    numpy.putmask(pred, pred < 0.5, 0.0)

    return pred
```

情感分类：分类问题

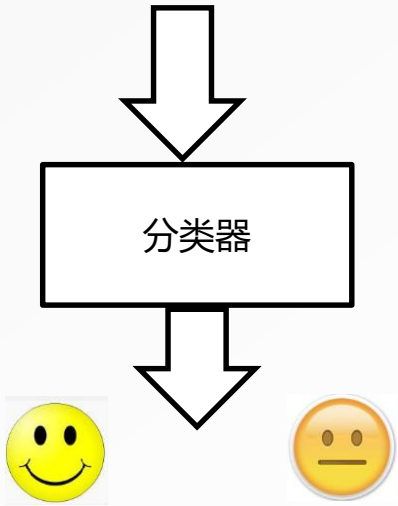


情感分类

诺基亚5800屏幕很好，操作也很方便，通话质量也不错，

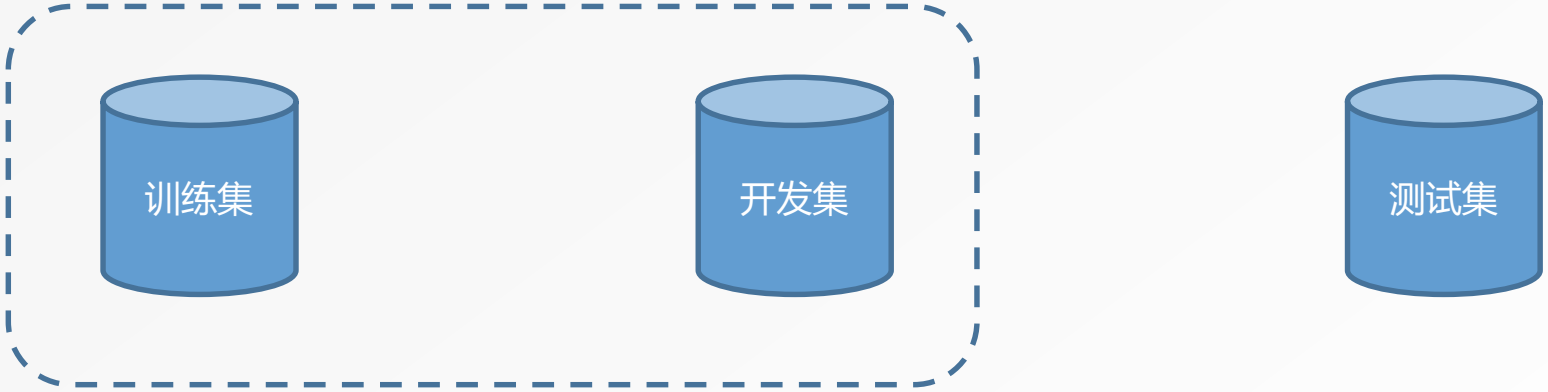
诺基亚	1
5800	1
屏幕	1
很好	1
操作	1
也	2
很	1
方便	1
通话	1
质量	1
不错	1

(1, 0, 0, 1, 0, 1, 1…. . , 0, 1)



开发集

在梯度下降训练的过程中，由于过拟合的原因，在训练样本上收敛的参数，并不一定在测试集上最优。因此，我们使用一个**验证集**（Validation Dataset）（也叫**开发集**, Development Dataset）来测试每一次迭代的参数在验证集上是否最优。如果在验证集上的错误率不再下降，就停止迭代。如果没有验证集，可以在训练集上进行**交叉验证。**



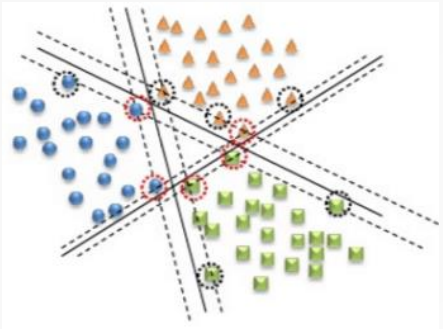
多类分类

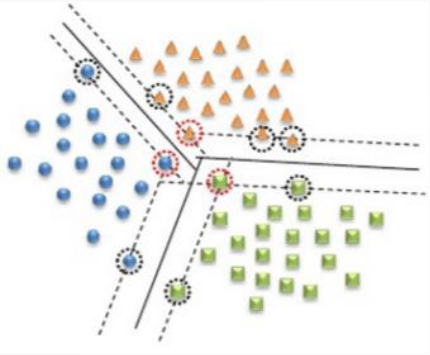
对于多类分类问题（假设类别数为 $C(C > 2)$），一般有两种多类转两类的转换方式：

- 把多类分类问题转换为 C 个两类分类问题，构建 C 个一对多的分类器。每个两类分类问题都是把某一类和其他类用一个超平面分开

- 把多类分类问题转换为 C(C – 1)/2 个两类分类问题，构建C(C – 1)/2 个两两分类器。每个两类分类问题都是把 C 类中某两类用一个超平面分开。

缺陷：一起区域中，点的类别是不能区分确定的





Softmax分类

SoftMax 回归是 Logistic 回归的多类推广。
我们定义目标类别 $y = c$ 的后验概率为：

$$P(y=c \mid x) = soft\max\left(w_c^T x\right) = \frac{\exp\left(w_c^T x\right)}{\sum_{i=1}^{C} \exp\left(w_i^T x\right)}$$

评价方法

常见的评价标准有正确率、准确率、召回率和 F 值等。

给定测试集 $T = (x_1,y_1),\cdots,(x_N,y_N)$，对于所有的 $y_i \in \{\omega_1,\cdots,\omega_C\}$。
假设分类结果为 $Y = \hat{y_1},\cdots,\hat{y_N}$。
则**正确率**（Accuracy，Correct Rate）为：

$$Acc = \frac{\sum_{i=1}^{N} \left| y_i = \hat{y}_i \right|}{N}$$

其中，$|\cdot|$ 为指示函数
和正确率相对应的就是**错误率**（Error Rate）。

$$Err = \frac{\sum_{i=1}^{N} \left| y_i \neq \hat{y}_i \right|}{N}$$

正确率是平均的整体性能。

评价方法

在很多情况下，我们需要对每个类都进行性能估计，这就需要计算准确率和召回率。正确率和召回率是广泛用于信息检索和统计学分类领域的两个度量值，在机器学习的评价中也被大量使用。

准确率（Precision，P），是识别出的个体总数中正确识别的个体总数的比例。对于类 c 来说，

$$P_c = \frac{\sum_{\substack{i=1 \\ \hat{y}_i=c}}^{N} \left| y_i = \hat{y}_i \right|}{\sum_{\substack{i=1 \\ \hat{y}_i=c}}^{N} 1}$$

召回率（Recall，R），也叫查全率，是测试集中存在的个体总数中正确识别的个体总数的比例。

$$R_c = \frac{\sum_{\substack{i=1 \\ y_i=c}}^{N} \left| y_i = \hat{y}_i \right|}{\sum_{\substack{i=1 \\ y_i=c}}^{N} 1}$$

其他机器学习方法/模型/任务

- 序列标注（Sequence Labeling）
- 主动学习（Active Learning）
- 集成学习（Ensemble Learning）
- 迁移学习（Transfer Learning）
- 多任务学习（Multi-task Learning）
- 强化学习（Reinforcement Learning）
- 终生学习（Life-long Learning）
- 课程学习（Curriculum Learning）
- 零样本学习（One/zero shot Learning）
- ……

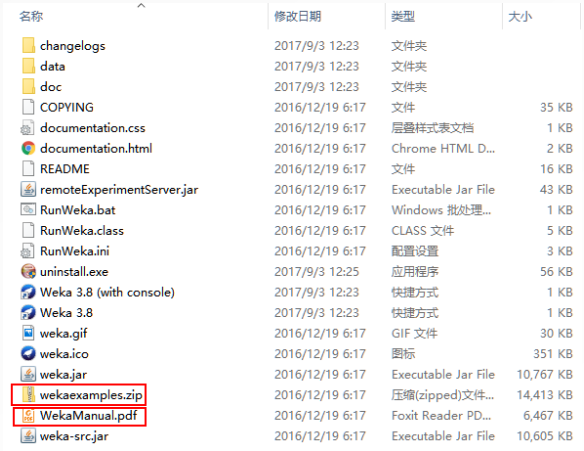
机器学习工具包：WEKA简介

- 作为一个大众化的数据挖掘工作平台， WEKA集成了大量能承担数据挖掘任务的机器学习算法，包括对数据进行预处理、分类、回归、聚类、关联分析以及在新的交互式界面上的可视化等等。通过其接口，可在其基础上实现自己的数据挖掘算法。

Data Mining: Practical Machine Learning Tools and Techniques (Fourth Edition)

《数据挖掘实用机器学习技术(原书第3版)》

http://www.cs.waikato.ac.nz/ml/weka/book.html



目录

机器学习

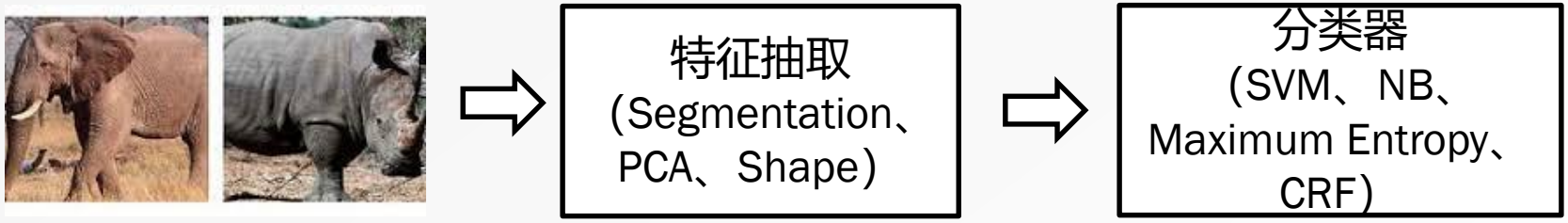
表示

目标函数

优化策略

传统机器学习

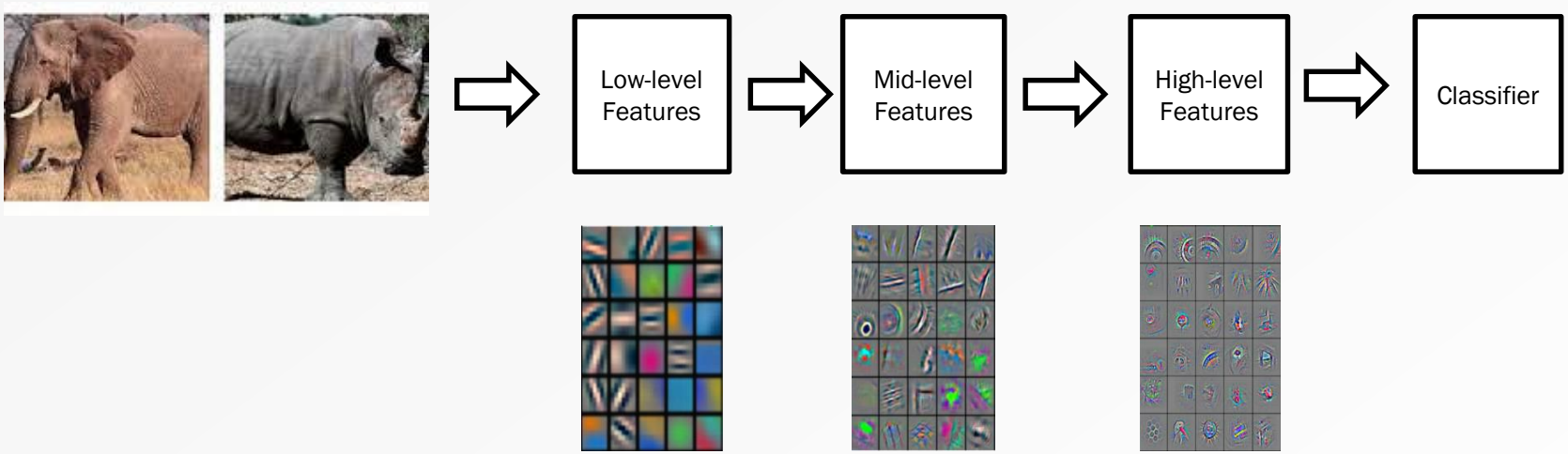
- 人工特征工程+分类器



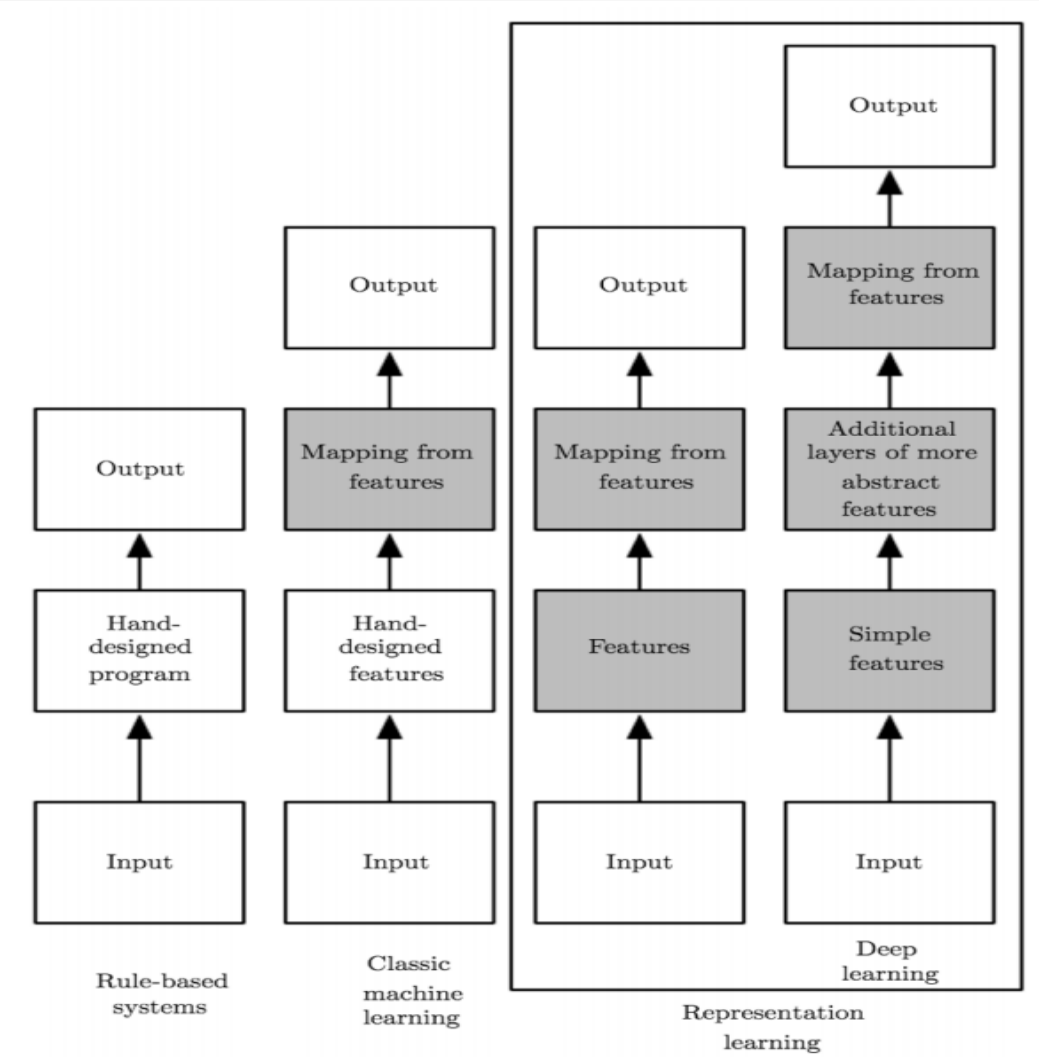
大数据下的机器学习

深度学习

- 自动学习多尺度的特征表示



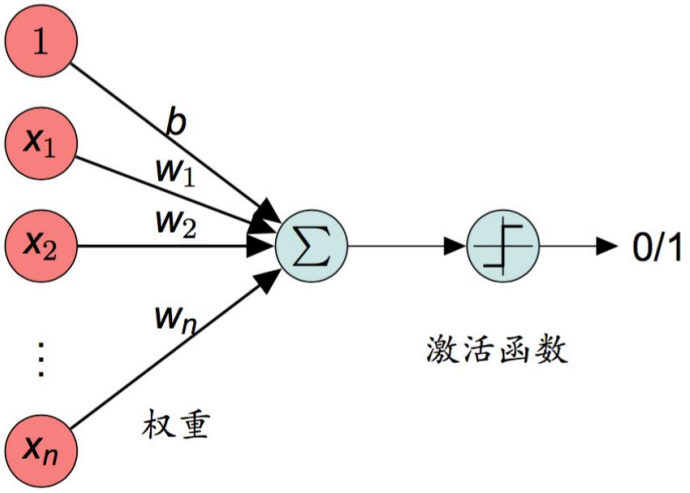
深度学习和表示学习


Output
Output
Output
Mapping from features
Output
Mapping from features
Mapping from features
Additional layers of more abstract features
Hand-designed program
Hand-designed features
Features
Simple features
Input
Input
Input
Input
Rule-based systems
Classic machine learning
Deep learning
Representation learning

感知机 Perception

- 感知器是对生物神经细胞的简单数学模拟,是最简单的人工神经网络, 只有一个神经元。 感知器也可以看出是线性分类器的一个经典学习算法。
- 细胞体(Soma)中的神经细胞膜上有各种受体和离子通道,胞膜的受体可与相应的化学物质神经递质结合,引起离子通透性及膜内外电位差发生改变,产生相应的生理活动:兴奋或抑制。 细胞突起是由细胞体延伸出来的细长部分,又可分为**树突和轴突。**
  - 树突(Dendrite)可以接受刺激并将兴奋传入细胞体。每个神经元可以有一或多个树突。
  - 轴突 (Axons) 可以把兴奋从胞体传送到另一个神经元或其他组织。 每个神经元只有一个轴突。
- 抑制与兴奋
  - 神经 细胞的状态取决于从其它的神经细胞收到的输入信号量,及突触的强度(抑制或加强)。当信号量总和超过了某个阈值时,细胞体就会兴奋,产生电脉冲。电脉冲沿着轴突并通过突触传递到其它神经元。

感知机



$$\text{output} = \begin{cases} 0 & \text{if } \sum_j w_j x_j \leq \text{threshold} \\ 1 & \text{if } \sum_j w_j x_j > \text{threshold} \end{cases}$$

感知机

- 给定输入 $x=(x_1,x_2,x_3,\cdots x_n)$

$$y=f(x)=sign(w\cdot x+b)$$

$$sign(x)=\begin{cases}+1, & x\geq 0\\ -1, & x<0\end{cases}$$

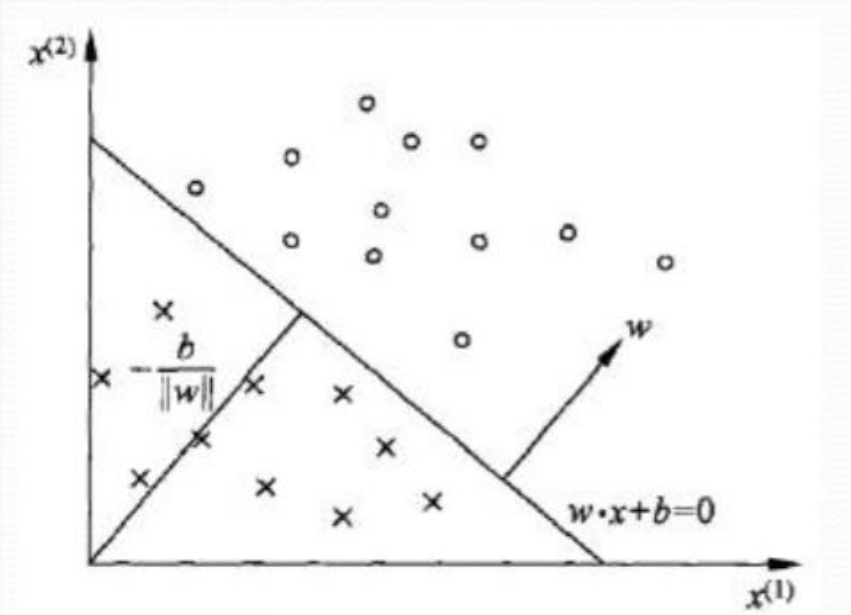
$$y=f(x)=sign(w\cdot x)$$

$$x=(b,x_1,x_2,x_3,\cdots x_n)$$

$$w=\{1,w_1,w_2,w_3,\cdots w_n\}$$

感知机几何解释

$$w \cdot x + b = 0$$



感知机参数学习

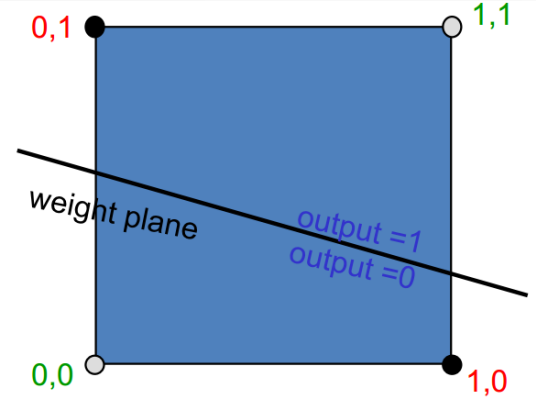
$$L(w,b)=\frac{1}{2}\sum_{x_i\in M}(y_i-(w\cdot x_i+b))^2 \qquad \widetilde{y_i}=w\cdot x_i+b$$

$$\frac{\partial L(w,b)}{\partial w}=-\sum_{x_i\in M}(y_i-\widetilde{y_i})x_i \qquad \frac{\partial L(w,b)}{\partial b}=-\sum_{x_i\in M}(y_i-\widetilde{y_i})$$

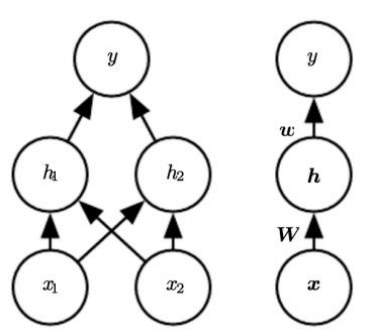
$$w\leftarrow w+\lambda\sum_{x_i\in M}(y_i-\widetilde{y_i})x_i \qquad b\leftarrow b+\lambda\sum_{x_i\in M}(y_i-\widetilde{y_i})$$

The Problems of Perception

- 异或问题 1969, Minsky and Papert



Non-linear Hidden Layer

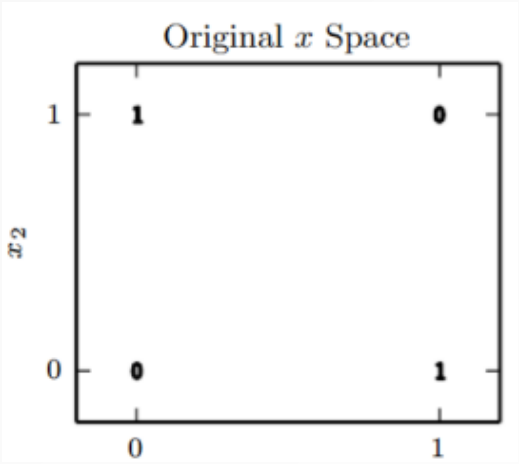


$$\boldsymbol{W} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix},$$

$$\boldsymbol{c} = \begin{bmatrix} 0 \\ -1 \end{bmatrix},$$

$$\boldsymbol{w} = \begin{bmatrix} 1 \\ -2 \end{bmatrix},$$

$$b = 0.$$



$$f(\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{c}, \boldsymbol{w}, b) = \boldsymbol{w}^\top \max\{0, \boldsymbol{W}^\top \boldsymbol{x} + \boldsymbol{c}\} + b$$

Example

- [0,0]
  - $\max(0, \left(\begin{bmatrix}1 & 1\\ 1 & 1\end{bmatrix}\begin{bmatrix}0\\0\end{bmatrix} + \begin{bmatrix}0\\-1\end{bmatrix}\right) = \begin{bmatrix}0\\0\end{bmatrix}$
  - $\begin{bmatrix}1\\-2\end{bmatrix}^{\mathrm{T}} \times \begin{bmatrix}0\\0\end{bmatrix} + 0 = 0$
- [1,0]
  - $\max(0, \left(\begin{bmatrix}1 & 1\\ 1 & 1\end{bmatrix}\begin{bmatrix}1\\0\end{bmatrix} + \begin{bmatrix}0\\-1\end{bmatrix}\right) = \begin{bmatrix}1\\0\end{bmatrix}$
  - $\begin{bmatrix}1\\-2\end{bmatrix}^{\mathrm{T}} \times \begin{bmatrix}1\\0\end{bmatrix} + 0 = 1$
- [0,1]
  - $\max(0, \left(\begin{bmatrix}1 & 1\\ 1 & 1\end{bmatrix}\begin{bmatrix}0\\1\end{bmatrix} + \begin{bmatrix}0\\-1\end{bmatrix}\right) = \begin{bmatrix}1\\0\end{bmatrix}$
  - $\begin{bmatrix}1\\-2\end{bmatrix}^{\mathrm{T}} \times \begin{bmatrix}1\\0\end{bmatrix} + 0 = 1$
- [1,1]
  - $\max(0, \left(\begin{bmatrix}1 & 1\\ 1 & 1\end{bmatrix}\begin{bmatrix}1\\1\end{bmatrix} + \begin{bmatrix}0\\-1\end{bmatrix}\right) = \begin{bmatrix}2\\1\end{bmatrix}$
  - $\begin{bmatrix}1\\-2\end{bmatrix}^{\mathrm{T}} \times \begin{bmatrix}2\\1\end{bmatrix} + 0 = 0$

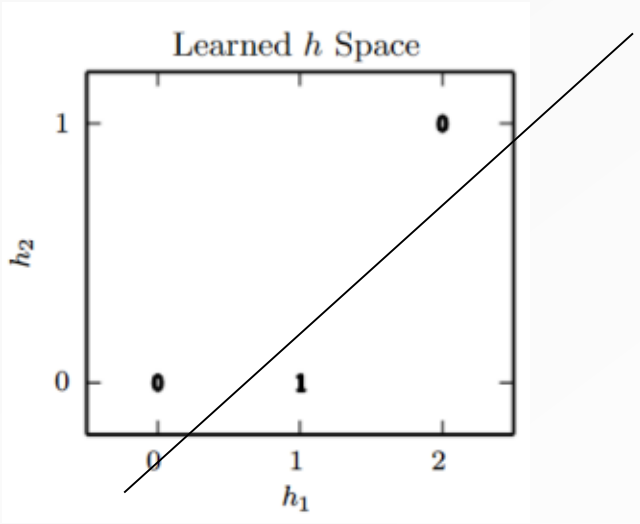
$$f(\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{c}, \boldsymbol{w}, b) = \boldsymbol{w}^{\top} \max\{0, \boldsymbol{W}^{\top}\boldsymbol{x} + \boldsymbol{c}\} + b$$

$$\boldsymbol{W} = \begin{bmatrix}1 & 1\\ 1 & 1\end{bmatrix},$$

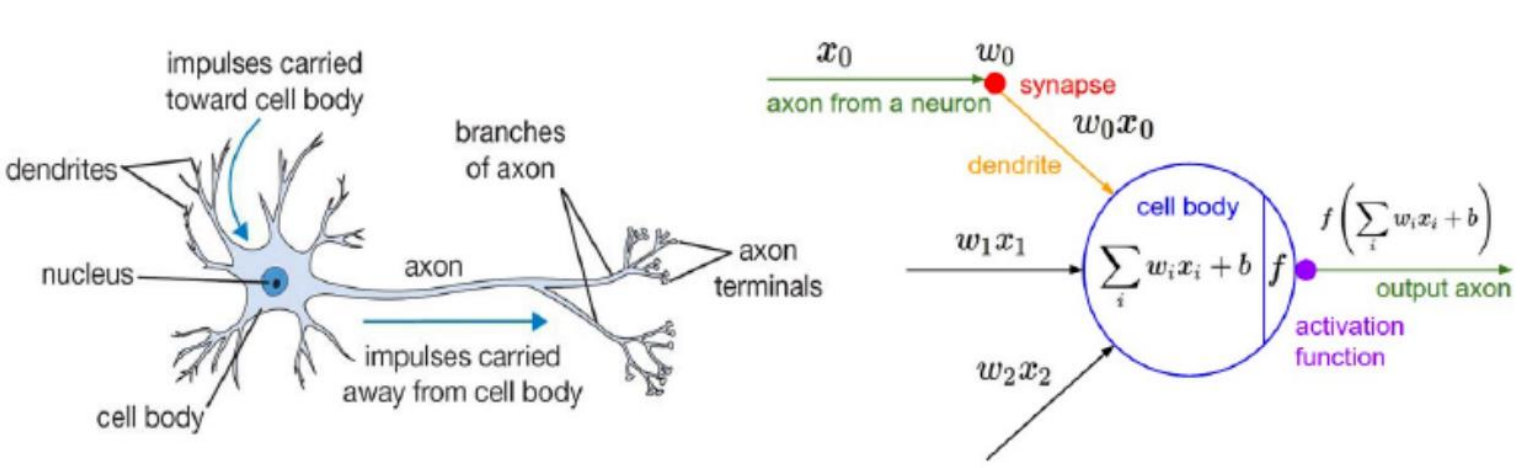
$$\boldsymbol{c} = \begin{bmatrix}0\\-1\end{bmatrix},$$

$$\boldsymbol{w} = \begin{bmatrix}1\\-2\end{bmatrix},$$

$$b = 0.$$



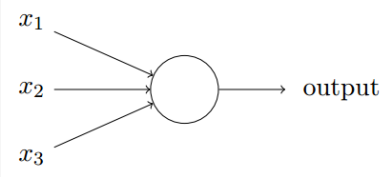
Non-linear Neurons



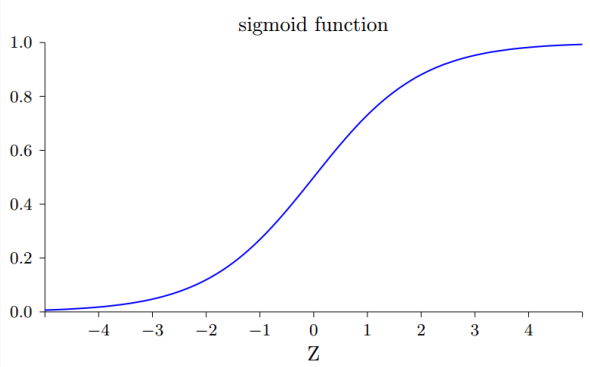
$$z = w \cdot x + b$$

$$y = f(z)$$

Sigmoid



Sigmoid Function $$y = \sigma(z) = \frac{1}{1 + e^{-z}} = \frac{1}{1 + \exp(-\sum_j w_j x_j - b)}$$

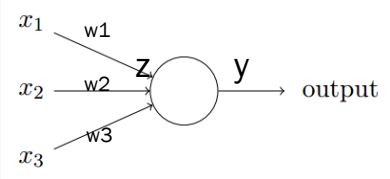


Training

$$E = \frac{1}{2} \sum_{output} (t - y)^2$$

$$y = \frac{1}{1+e^{-z}}$$

$$z = \sum_i w_i x_i$$



$$\frac{\partial E}{\partial w_i} = \frac{\partial E}{\partial y} \frac{\partial y}{\partial z} \frac{\partial z}{\partial w_i}$$

$$\frac{\partial E}{\partial y} = -(t - y)$$

$$\frac{\partial y}{\partial z} = y(1 - y)$$

$$\frac{\partial z}{\partial w_i} = x_i$$

$$\frac{\partial E}{\partial w_i} = -(t - y)\, y(1 - y) x_i$$

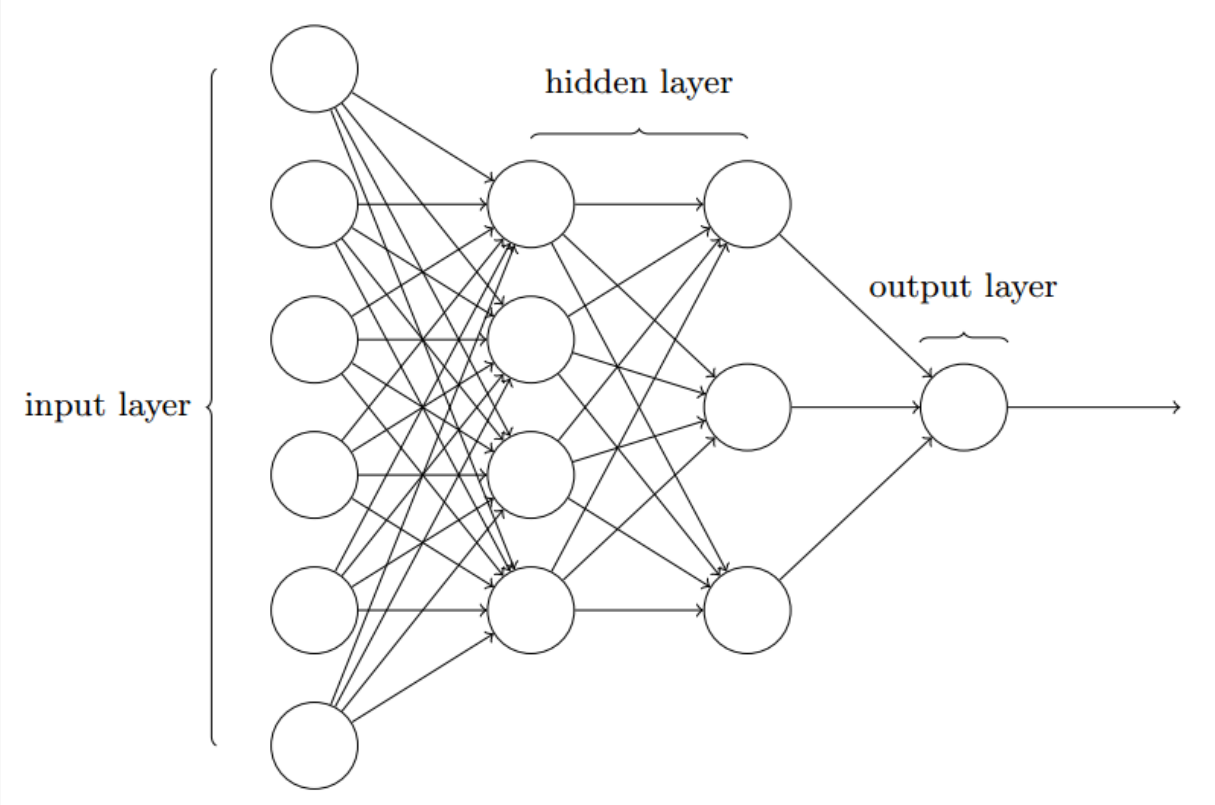
Other Activation Functions

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)[7]		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU)[8]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$

rectifier(x) = max(0, x)

rectifier 函数被认为有生物上的解释性。神经科学家发现神经元具有单侧抑制、宽兴奋边界、稀疏激活性等特性。采用 rectifier 函数的单元也叫作正线性单元(rectified linear unit, ReLU)

前馈神经网络
Feed Forward Neural Networks



前馈计算

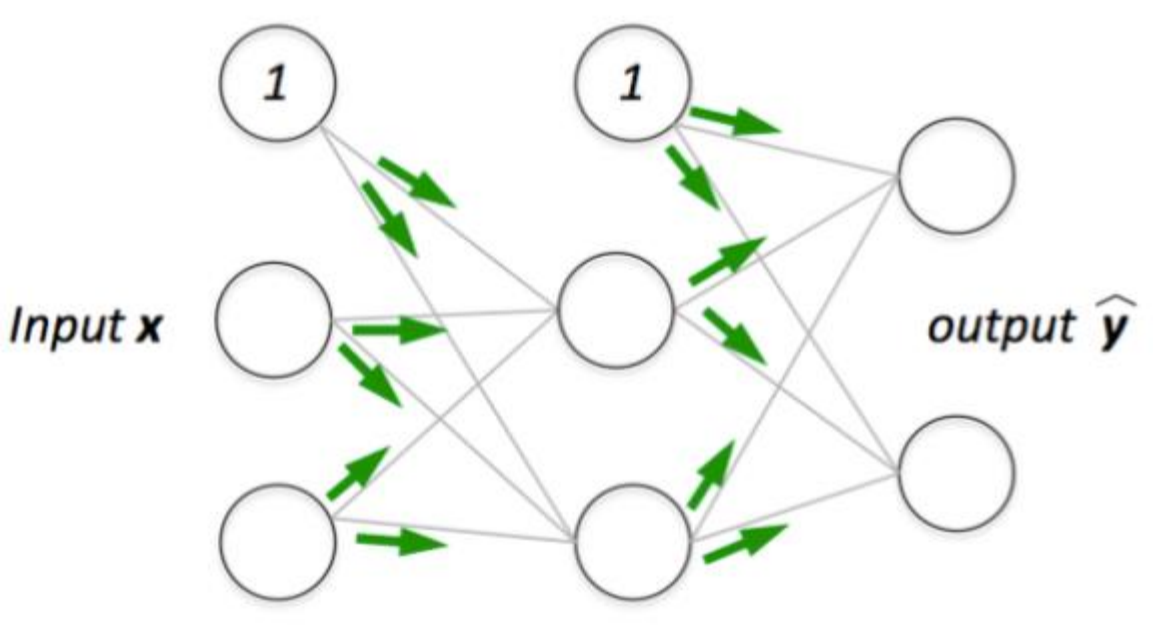
- L表示神经网络的层数
- n^l表示第l层神经元的个数
- $f_l(z)$表示第l层的激活函数
- w^l表示第l层的权重
- b^l表示第l层神经元的偏置
- z^l表示第l层神经元的状态
- y^l表示第l层神经元的输出

$$\mathrm{z}^l = \mathrm{w}^l \cdot \mathrm{y}^{l-1} + \mathrm{b}^l$$

$$\mathrm{y}^l = f_l(\mathrm{z}^l)$$

$$\mathrm{z}^l = \mathrm{w}^l \cdot f_l(\mathrm{z}^{l-1}) + \mathrm{b}^l$$

前馈计算

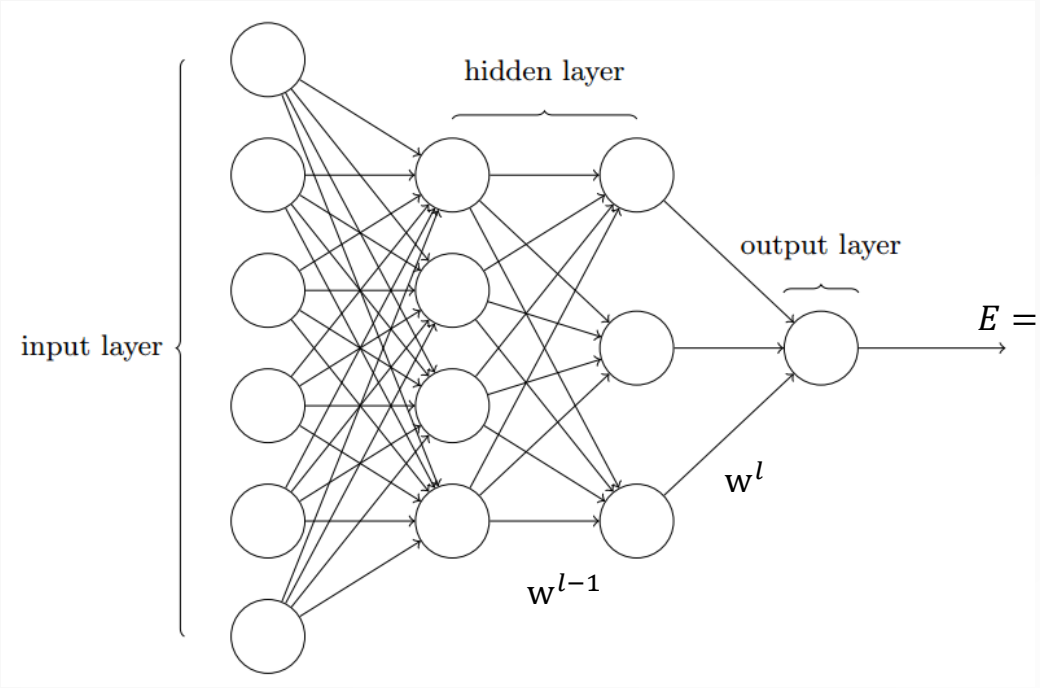


$$x \to z^1 \to f_1(z^1) \to z^2 \to f_2(z^2) \to \dots\dots \to z^l \to f_l(z^l) \to \dots\dots \to z^L \to f_L(z^L) \to y$$

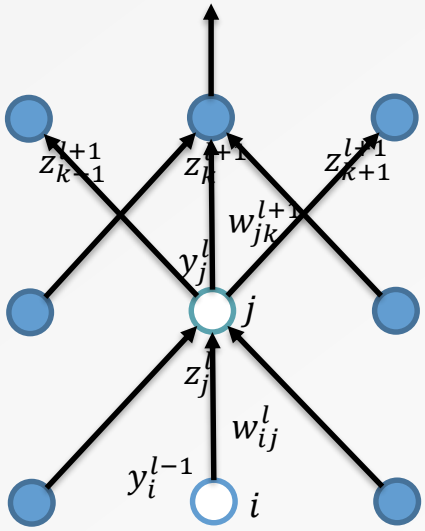
训练：梯度下降法

$$E = \frac{1}{2}\sum_{output}(t_j - y_j)^2$$

$$w_{ij} \leftarrow w_{ij} - \lambda\frac{\partial E}{\partial w_{ij}}$$



训练：Error Back Propagating



$$E = \frac{1}{2}\sum_{output}(t_j - y_j^L)^2$$

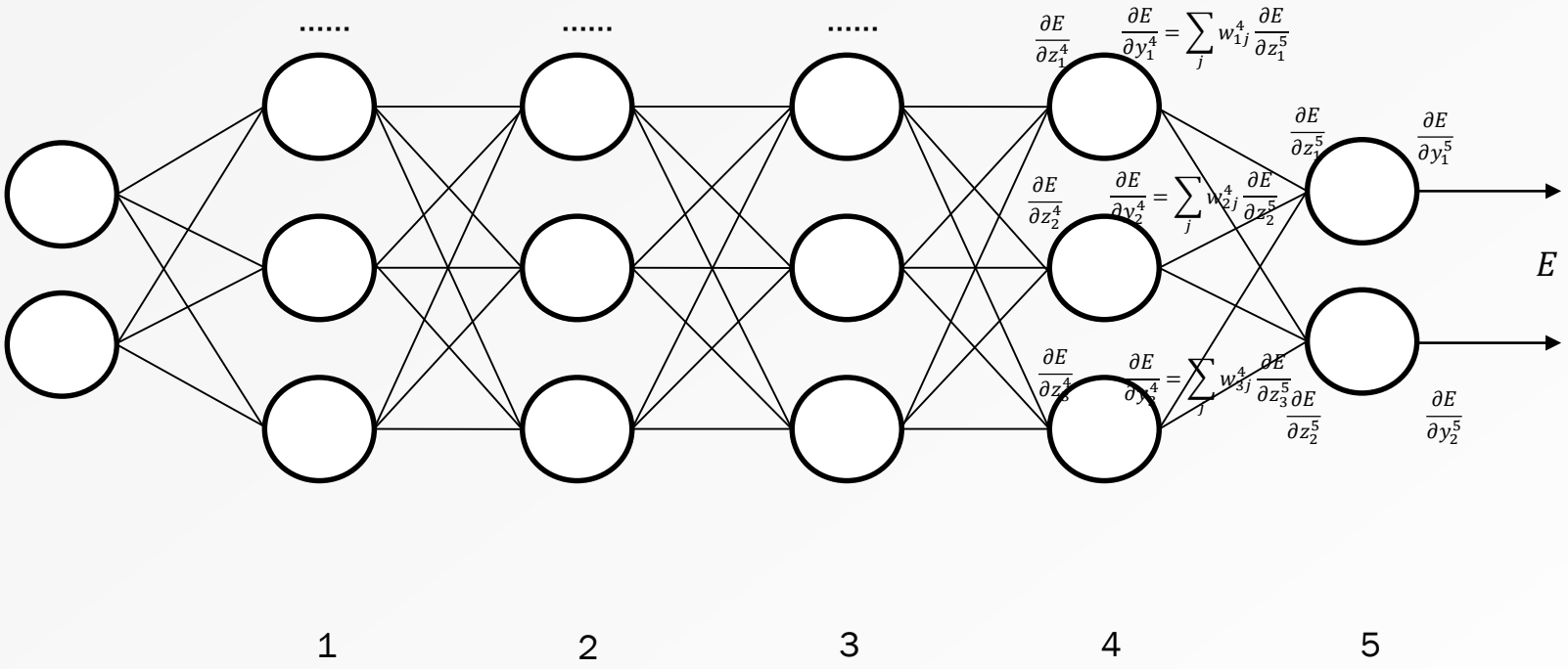
$$\frac{\partial E}{\partial w_{ij}^l} = \frac{\partial z_j^l}{\partial w_{ij}^l}\frac{\partial E}{\partial z_j^l} = \frac{\partial E}{\partial z_j^l}y_i^{l-1}$$

$$\frac{\partial E}{\partial z_j^l} = \frac{\partial y_j^l}{\partial z_j^l}\frac{\partial E}{\partial y_j^l} = \frac{\partial E}{\partial y_j^l}y_j^l(1-y_j^l)$$

$$y = \frac{1}{1+e^{-z}} \qquad z = \sum_i w_i y_i$$

$$\frac{\partial E}{\partial y_j^l} = \sum_k \frac{\partial z_k^{l+1}}{\partial y_j^l}\frac{\partial E}{\partial z_k^{l+1}} = \sum_k w_{jk}^{l+1}\frac{\partial E}{\partial z_k^{l+1}}$$

反向错误传播

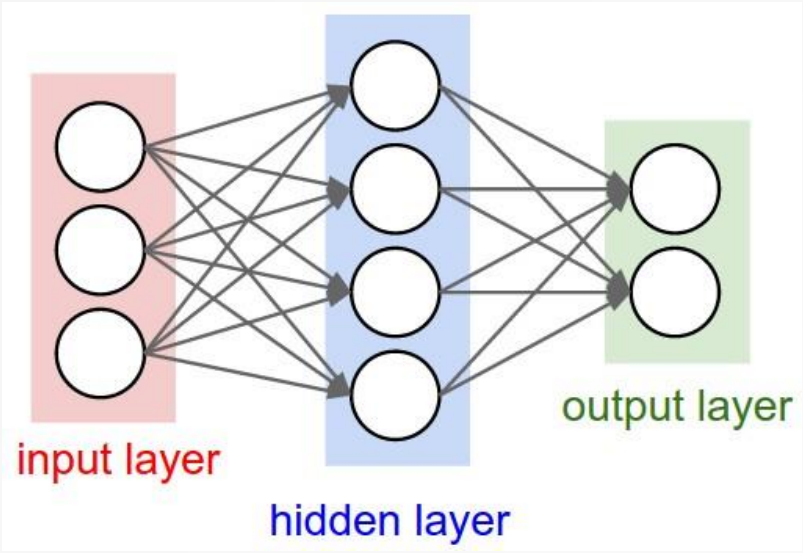

.....
.....
.....
$\frac{\partial E}{\partial z_1^4}$
$\frac{\partial E}{\partial y_1^4} = \sum_j w_{1j}^4 \frac{\partial E}{\partial z_1^5}$
$\frac{\partial E}{\partial z_2^4}$
$\frac{\partial E}{\partial y_2^4} = \sum_j w_{2j}^4 \frac{\partial E}{\partial z_2^5}$
$\frac{\partial E}{\partial z_3^4}$
$\frac{\partial E}{\partial y_3^4} = \sum_j w_{3j}^4 \frac{\partial E}{\partial z_3^5}$
$\frac{\partial E}{\partial z_1^5}$
$\frac{\partial E}{\partial y_1^5}$
$\frac{\partial E}{\partial z_2^5}$
$\frac{\partial E}{\partial y_2^5}$
E
1
2
3
4
5

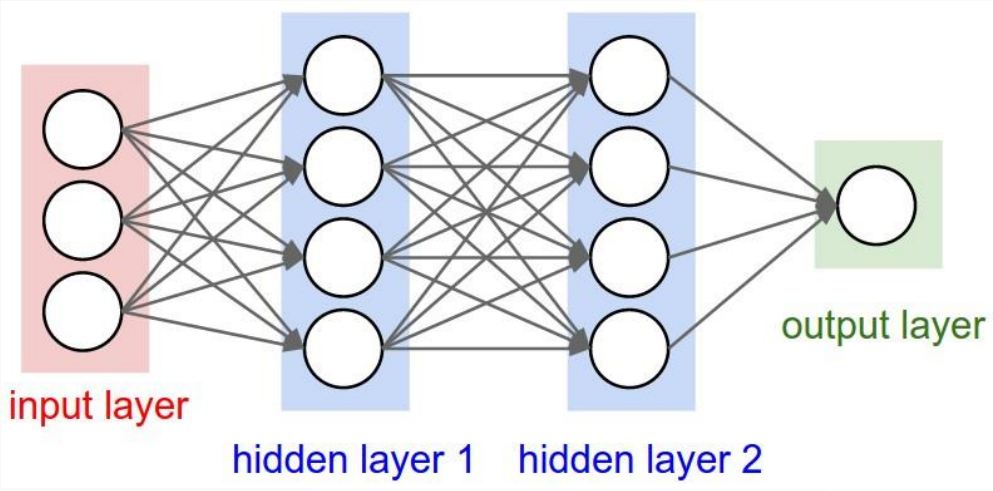
反向传播

- 在第l层神经元上的误差：$\delta^l = \frac{\partial E}{\partial z^l}$
- 各层神经元上的误差
  - $\delta^L = \frac{\partial E}{\partial y^L} \odot f'(z^L)$
  - $\delta^l = \left(w^{l+1}\delta^{l+1}\right) \odot f'\left(z^l\right)$

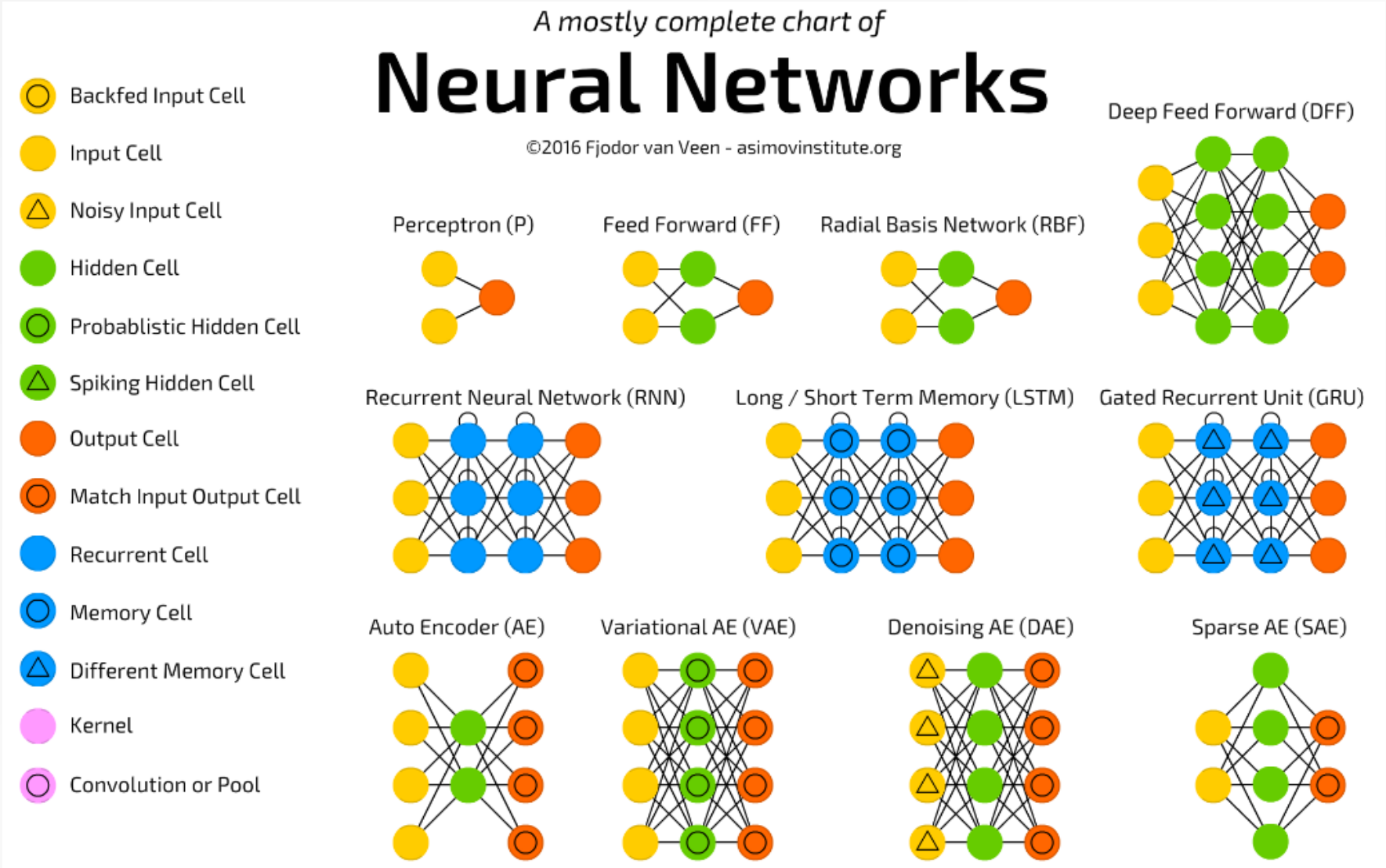
  - $\frac{\partial E}{\partial b^l} = \delta^l$
  - $\frac{\partial E}{\partial w^l} = y^{l-1}\delta^l$

深度学习：由浅入深


input layer
hidden layer
output layer


input layer
hidden layer 1
hidden layer 2
output layer

http://www.asimovinstitute.org/neural-network-zoo/



深度学习的发展历史

1958 Perceptron

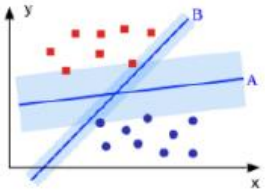
1969
Perceptron criticized

1974 Backpropagation

awkward silence (AI Winter)

Convolution Neural Networks for
Handwritten Recognition
1998

1995
SVM reigns



2006
Restricted
Boltzmann
Machine

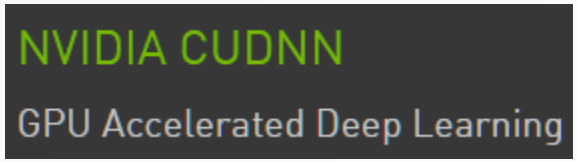
Google Brain Project on
16k Cores
2012

2012
AlexNet wins
ImageNet
IMAGENET

http://www.andreykurenkov.com/writing/a-brief-history-of-neural-nets-and-deep-learning/

常用的深度学习工具包

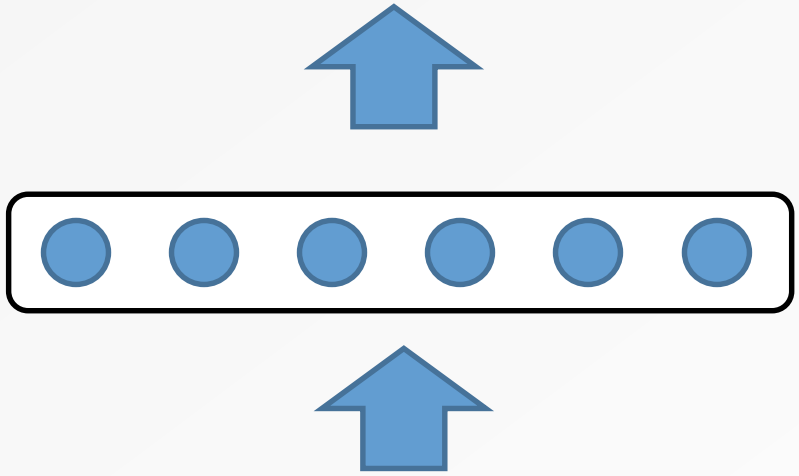
- TensorFlow
  - **谷歌开发的 Python 工具包，Linux环境，可以在任意具备 CPU 或者 GPU 的设备上运行**
- ~~Theano~~
  - ~~Theano:蒙特利尔大学的 Python工具包，用来高效地定义、优化和执行多维数组数据对应数学表达式。~~
- (Py)Torch
  - Facebook 开发 Lua 工具包，现在有Python版本。
  - 无缝 CPU 和 GPU 切换
- Caffe （ Convolutional Architecture for Fast Feature Embedding ）
  - 一个计算 CNN 相关算法的框架。Caffe 是用 C++ 和 Python 实现的。
  - 所要实现的网络结构可以在配置文件中指定,不需要编码。
  - 目前在 GPU 上实现 CNN 最快,在单个 Tesla K20 机器上,每天处 理 20million 图片。
  - 主要用于计算机视觉
- Keras
  - 一个极简和高度模块化的神经网络库
  - 无缝 CPU 和 GPU 切换
  - 支持任意链接
- DyNet
  - （处理NLP灵活），观望
- MXNet
  - （据说）效率高



目录

自然语言处理

分类, 聚类, （结构）预测, 排序...



伦齐在宪改公投惨败后，宣布有意辞职。

自然语言处理

- 分类　　内容→类标
- 生成　　结构→内容
- 匹配　　<内容,内容> → 类标(数值)
- 翻译　　内容 → 内容
- 结构预测　内容 → 结构

内容→表示→类标

结构→表示→内容

<内容,内容>→表示→类标(数值)

内容 →表示→ 内容

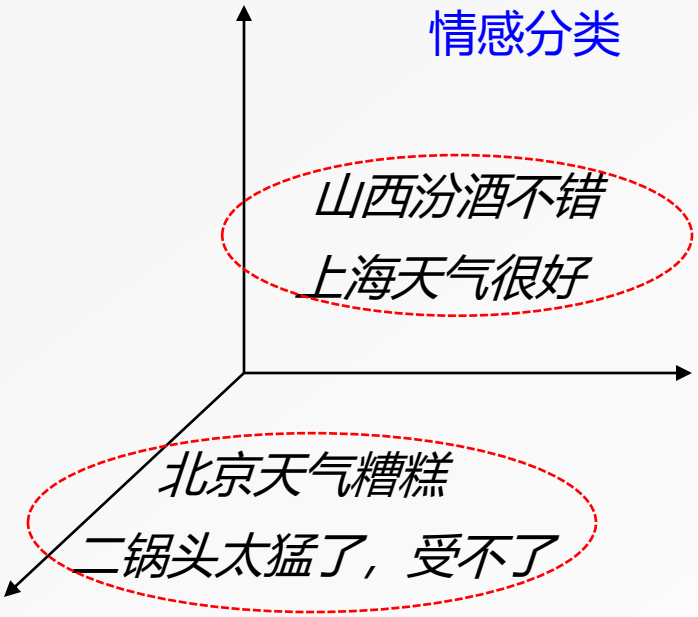
内容 →结构+表示

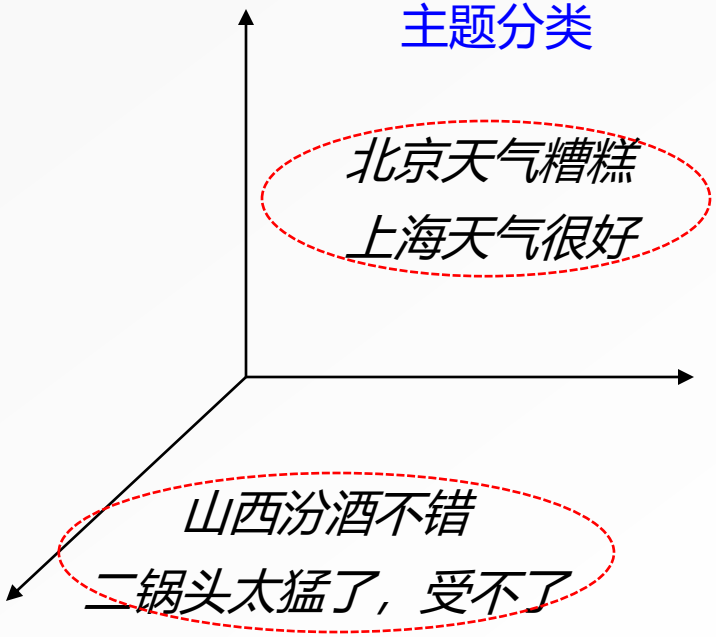
自然语言处理：

使用计算机表示和处理自然语言（中的语言单元）

组合语义原则

- 一个复杂对象的意义是由其各组成部分的意义以及它们的组合规则来决定。
- The meaning of a complex expression is determined by the meanings of its constituent expressions and the rules used to combine them.





词的分布表示

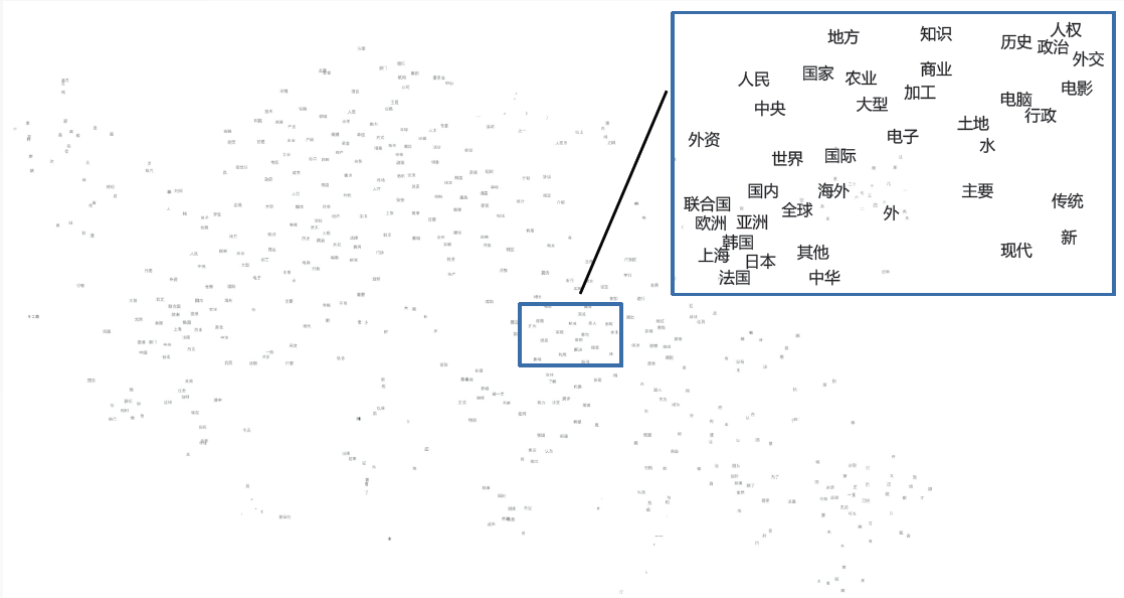
The first thing you do with a word symbol is you convert it to a word vector. And you learn to do that, you learn for each word how to turn a symbol into a vector, say, 300 components, and after you’ ve done learning, you’ ll discover the vector for Tuesday is very similar to the vector for Wednesday.


Science | DOI:10.1145/2874915 Gregory Goth

Deep or Shallow, NLP Is Breaking Out

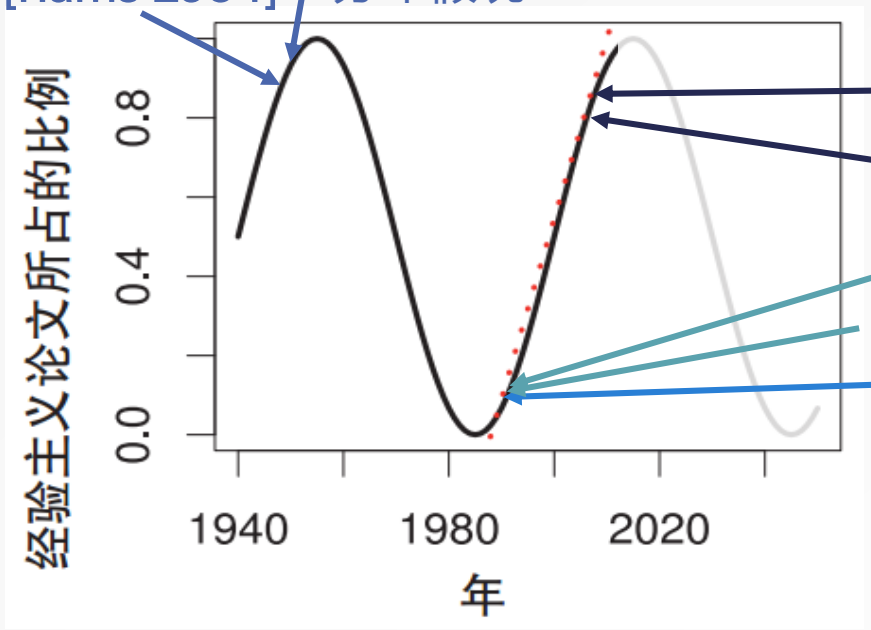
Neural net advances improve computers' language ability in many fields.

[Geoffrey E. Hinton. 2015]



词的分布表示

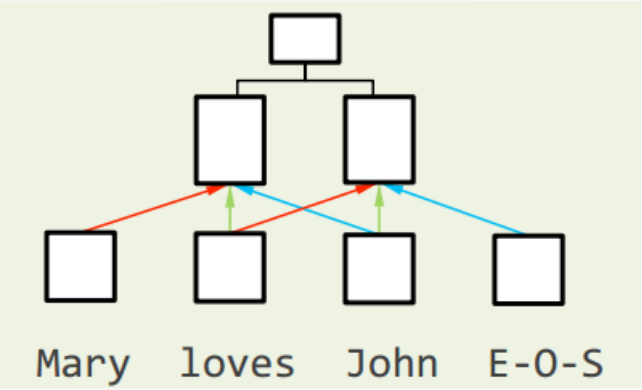
- 分布假说：上下文相似的词 → 词义相似
  - 语义可以统计获得
  - 相似度可以度量



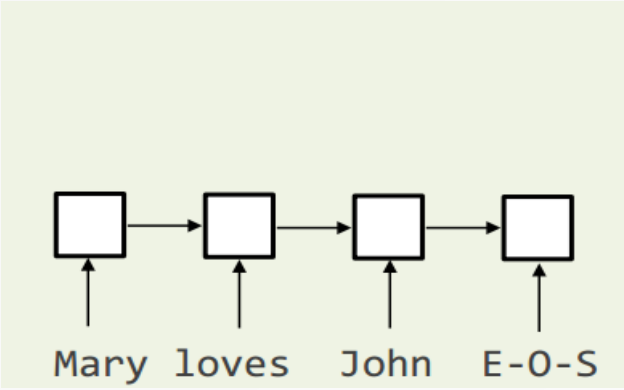
[Church 2011]
《钟摆摆得太远》

语义组合模型

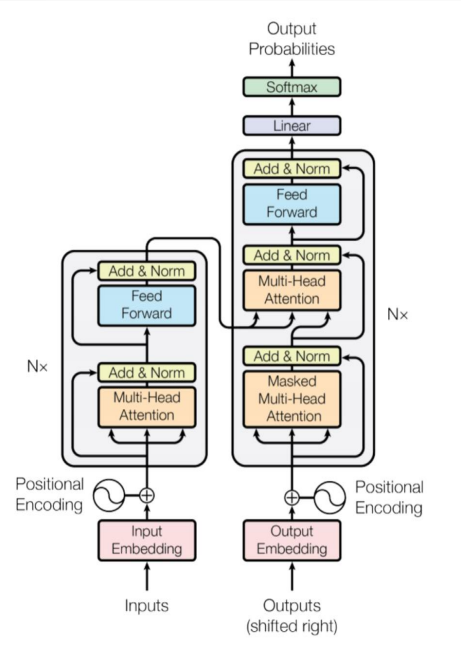
- 基于词、短语等基本语义单元表示句子等更大语义单元的过程
- 建模句子的句法/语义信息（结构）



卷积神经网络



卷积神经网络

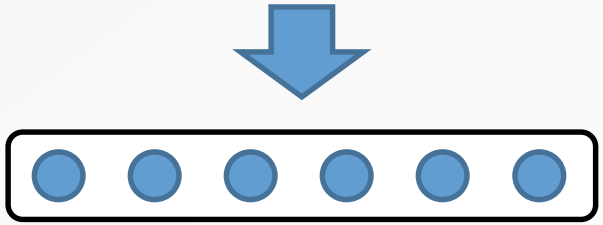


Transformer

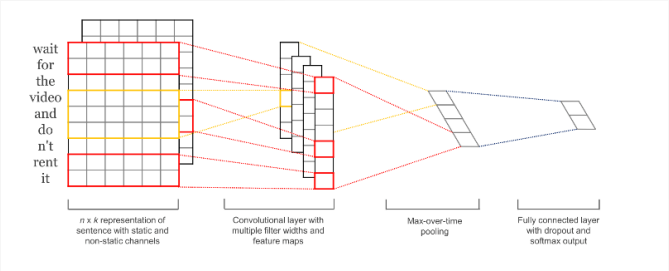
目录

“Big” units Modeling

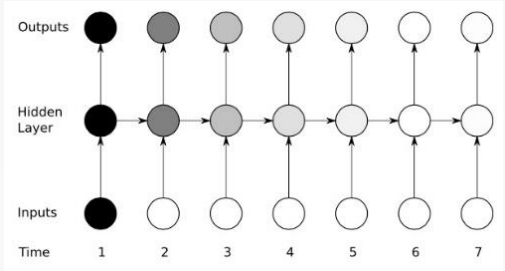
I want to play game with that little boy.



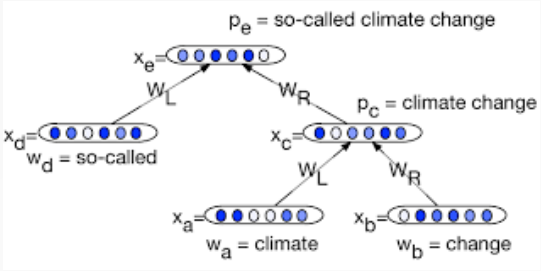
典型语义组合模型



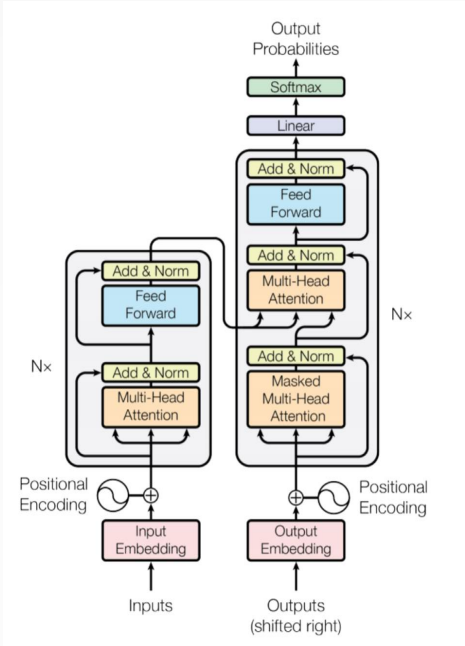
Convolutional Neural Network (CNN)



Recurrent Neural Network (RNN)
循环神经网络



Recursive Neural Network (RNN)
递归神经网络

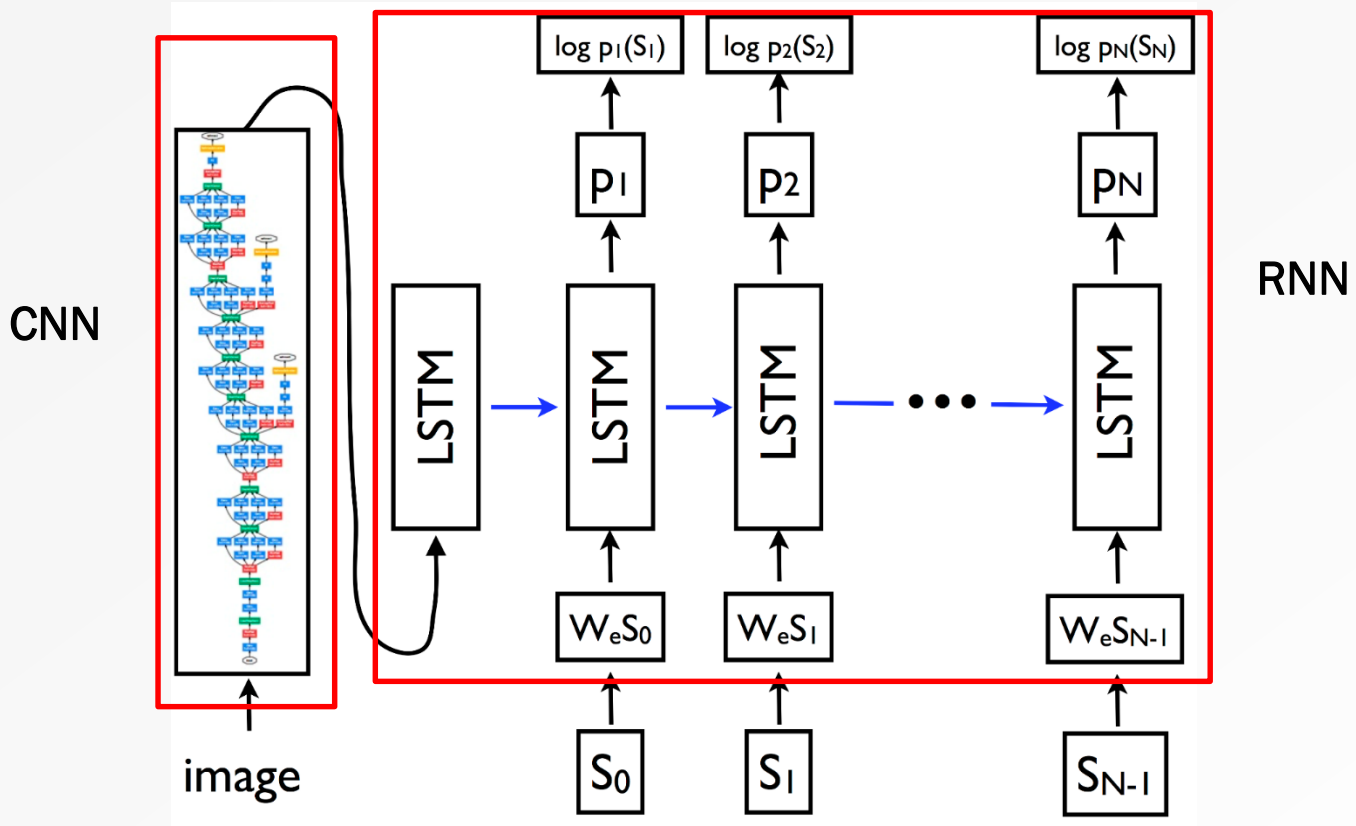


Transformer

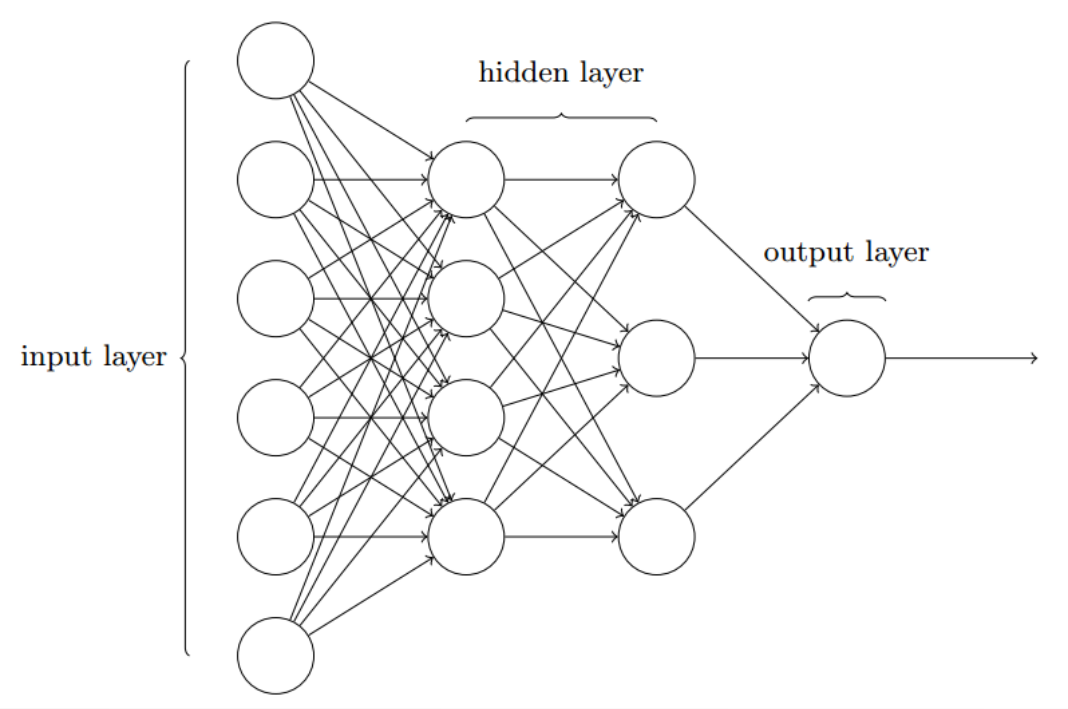
“Big” units Generating

a man riding a wave on top of a surfboard .

Neural Image Caption Generator



Feed Forward Neural Networks



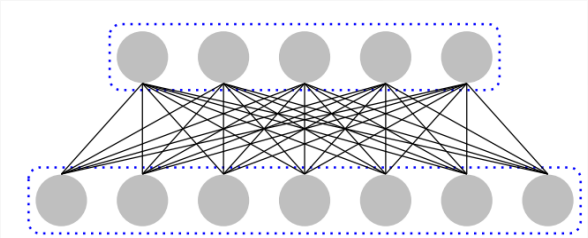
全连接

- n^l表示第l层神经元的个数
- L表示神经网络的层数
- 参数个数：$\prod_{l=1}^{L} n^l$
- 权重矩阵参数非常多，训练效率低下
- 数据不足时，欠学习

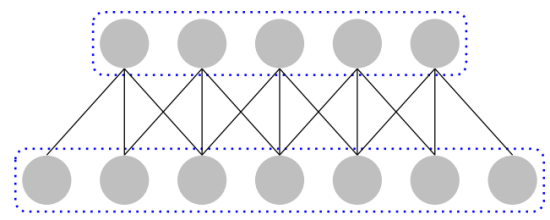
CNN

- Convolutional Neural Network是一种前馈神经网络。卷积神经网络是受生物学上感受野(Receptive Field) 的机制而提出的。一个神经元的感受野是指特定区域，只有这个区域内的刺激才能够激活该神经元。
  - 局部链接
  - 权值共享
  - 采样
- 具有平移、缩放和扭曲不变性

全连接 vs. 卷积



全连接

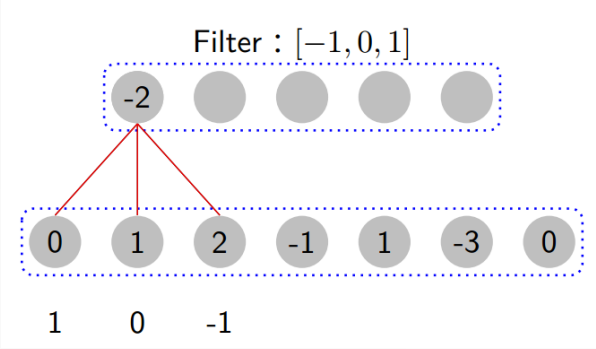


CNN

一维卷积

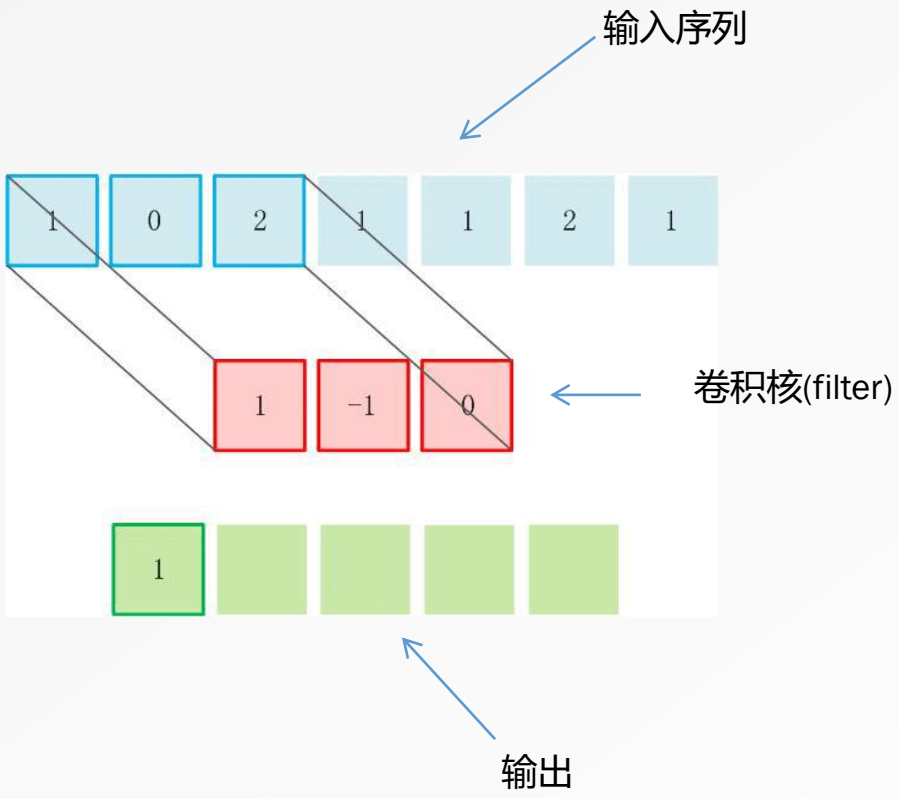
- 信号x，信号长度n
- 滤波器f，滤波器长度m

$$y_t = \sum_{k=1}^{n} f_k \cdot x_{t-k+1}$$



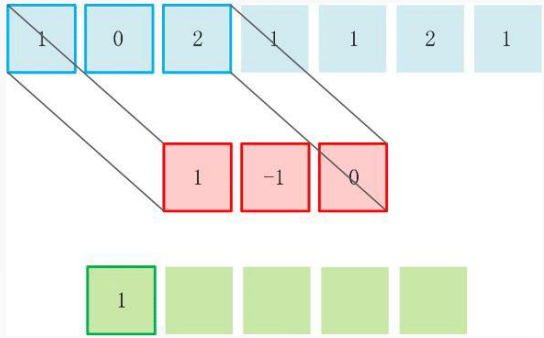
当$f_k = \frac{1}{m}$时，相当于移动平均

Example

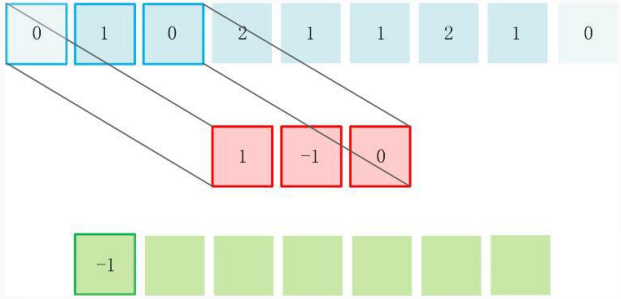


卷积类型

- 窄卷积
  - 信号两端不补0
  - 输出信号长度为n-m+1
- 宽卷积
  - 信号两端补0
  - 输出信号长度为n+m-1
- 等长卷积
  - 信号两端补0
  - 输出信号长度为n



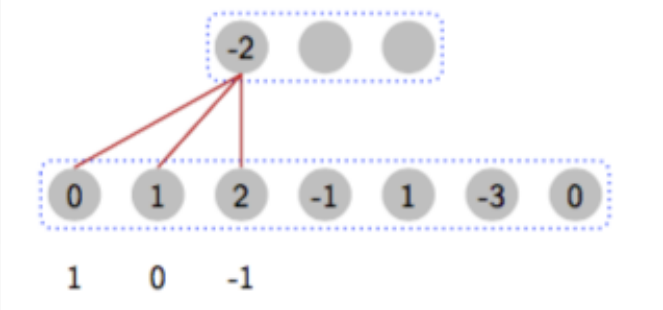
窄卷积



等长卷积

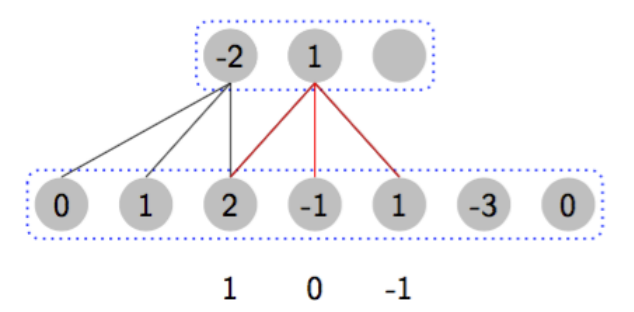
滤波器步长

- Stride=2



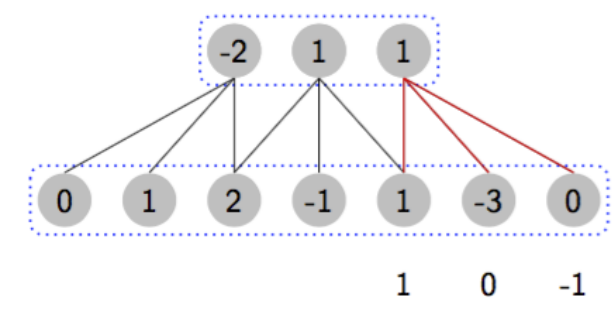
滤波器步长

- Stride=2



滤波器步长

- Stride=2



二维卷积

- 信号x，信号长度M*N
- 滤波器f，滤波器长度m*n
- 在图像中，卷积意味着区域内像素的加权平均

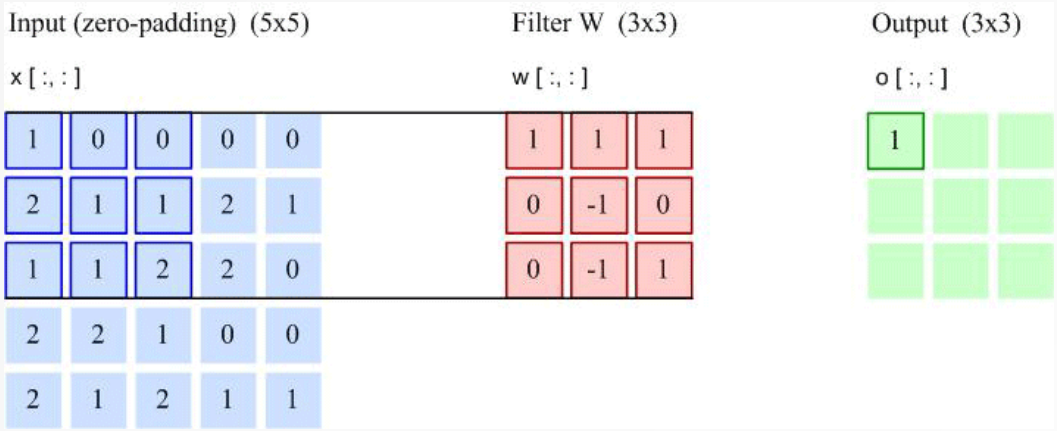
$$y_{ij} = \sum_{u=1}^{m} \sum_{v=1}^{n} f_{uv} \cdot x_{i-u+1,j-v+1}$$

当$f_{uv} = \frac{1}{mn}$时，相当于平均

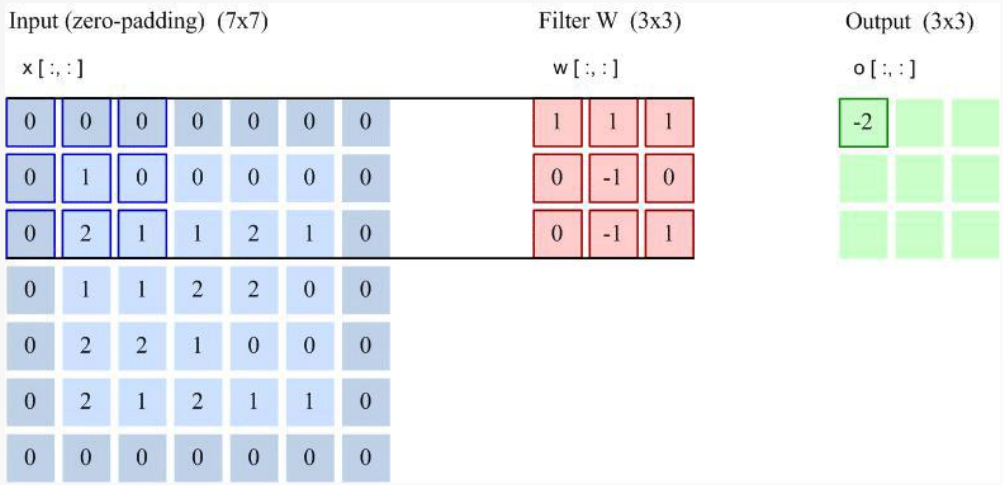
卷积类型

- 窄卷积
  - 信号四周不补0
  - 输出信号长度为M-m+1*N-n+1
- 宽卷积
  - 信号四周补0
  - 输出信号长度为M+m-1 *N+n-1
- 等长卷积
  - 信号四周补0
  - 输出信号长度为M*N

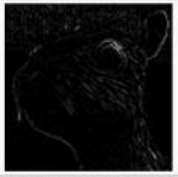
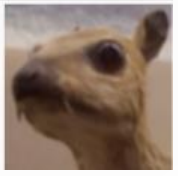
Examples


Input (zero-padding) (5x5)
x [:, :]
1 0 0 0 0
2 1 1 2 1
1 1 2 2 0
2 2 1 0 0
2 1 2 1 1
Filter W (3x3)
w [:, :]
1 1 1
0 -1 0
0 -1 1
Output (3x3)
o [:, :]
1

Examples


Input (zero-padding) (7x7)
x [:, :]
0 0 0 0 0 0 0
0 1 0 0 0 0 0
0 2 1 1 2 1 0
0 1 1 2 2 0 0
0 2 2 1 0 0 0
0 2 1 2 1 1 0
0 0 0 0 0 0 0
Filter W (3x3)
w [:, :]
1 1 1
0 -1 0
0 -1 1
Output (3x3)
o [:, :]
-2

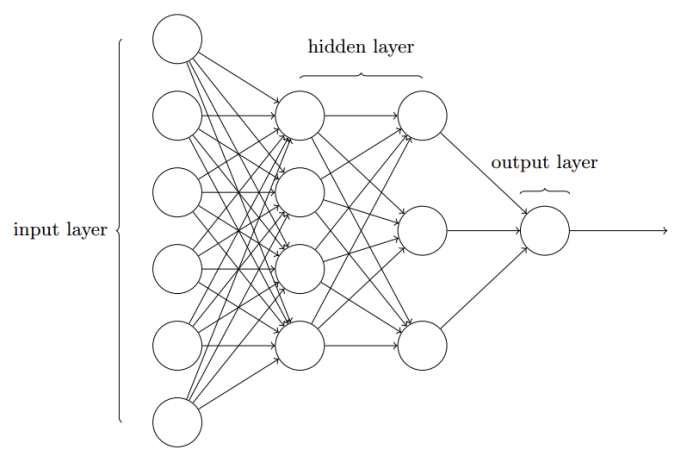
两维卷积实例

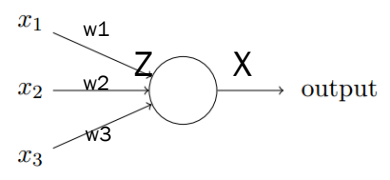
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$				
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$		Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$		Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$		Gaussian blur (approximation)	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

卷积层

全连接前馈神经网络

$$X_i^{(l)} = f\left(\sum_j^{n^{(l)}} w_j^{(l)} X_j^{(l-1)} + b_i^{(l)}\right)$$





$$z_i = \sum_i w_i x_i + b_i \qquad x_{i+1} = f(z_i)$$

卷积层

卷积神经网络

第 l 层的每一个神经元都只和第 l − 1 层的一个局部窗口内的神经元相连，构成一个局部连接网络。

$$X_i^{(l)} = f(\sum_j^m w_j^{(l)} X_{i-l+m}^{(l-1)} + b^{(l)})$$
$$= f(w^{(l)} \cdot X_{i-l+m}^{(l-1)} + b_i)$$

其中，$w^{(l)} \in \mathbb{R}^m$ 为m维的滤波器，

$$X^{(l)} = f(w^{(l)} \otimes X_{i-l+m}^{(l-1)} + b^{(l)})$$

权值共享。这样，在卷积层里，我们只需要 m + 1 个参数。另外，第 l + 1 层的神经元个数不是任意选择的，而是满足$n^{(l+1)} = n^{(l)} - m + 1$

二维卷积层

在图像处理中，图像是以二维矩阵的形式输入到神经网络中，因此我们需要二维卷积。假设 $x^l \in \mathbb{R}^{(w_l \times h_l)}$ 和 $x^{(l-1)} \in \mathbb{R}^{(w_{l-1} \times h_{l-1})}$分别是第$l$层和第 $l-1$层的神经元活性。x^l 的每一个元素为：

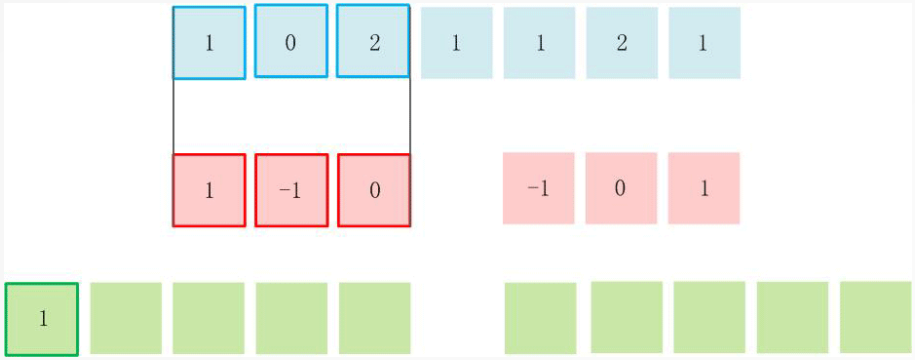
$$X_{s,t}^{(l)} = f\left(\sum_{i=1}^{u}\sum_{j=1}^{v} W_{i,j}^{(l)} \cdot X_{s-i+u,t-j+v}^{(l-1)} + b^{(l)}\right)$$

其中，$W^{(l)} \in \mathbb{R}^{u \times v}$为两维的滤波器， $b^{(l)}$为偏置矩阵。

$$X^{(l)} = f\left(W^{(l)} \otimes X^{(l-1)} + b^{(l)}\right)$$

第 $l-1$ 层的神经元个数为 $w_l \times h_l$，并且$w_l = w_{l-1} - u + 1, h_l = h_{l-1} - v + 1$

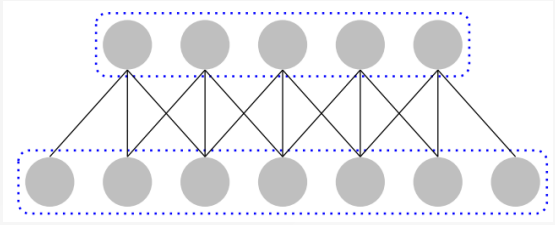
两个filters

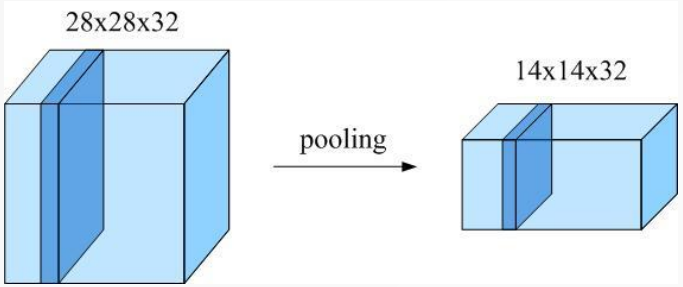


特征映射 Feature Map

为了增强卷积层的表示能力，我们可以使用K个不同的滤波器来得到K组输出。每一组输出都共享一个滤波器。如果我们把滤波器看成一个特征提取器，每一组输出都可以看成是输入图像经过一个特征抽取后得到的特征。因此，在卷积神经网络中每一组输出也叫作一组**特征映射**（Feature Map）。

子采样层





- 卷积层虽然可以显著减少连接的个数，但是每一个特征映射的神经元个数并没有显著减少
- 高维数据→过拟合
- 降低维度：Pooling、Subsampling
- 特征选择、特征抽取

子采样层

对于卷积层得到的一个特征映射 $X^{(l)}$，我们可以将 $X^{(l)}$ 划分为很多区域R_k, $k = 1,\cdots,K$，这些区域可以重叠，也可以不重叠。一个子采样函数**down**(…) 定义为：

$$\begin{aligned} X_k^{(l+1)} &= f\left(Z_k^{(l+1)}\right) \\ &= f\left(w^{(l+1)} \cdot down\left(R_k\right) + b^{(l+1)}\right) \end{aligned}$$

其中，$w^{(l+1)}$ 和 $b^{(l+1)}$ 分别是可训练的权重和偏置参数。

$$\begin{aligned} X^{(l+1)} &= f\left(Z^{(l+1)}\right) \\ &= f\left(w^{(l+1)} \cdot down\left(X^{l}\right) + b^{(l+1)}\right) \end{aligned}$$

down(X^l) 是指子采样后的特征映射

子采样层

- 最大值采样（Maximum Pooling）

$$pool_{max}\ (R_k) = \max_{i \in R_k} a_i$$

- 最小值采样（Minimum Pooling）

$$pool_{min}\ (R_k) = \min_{i \in R_k} a_i$$

- 平均值（Average Pooling）

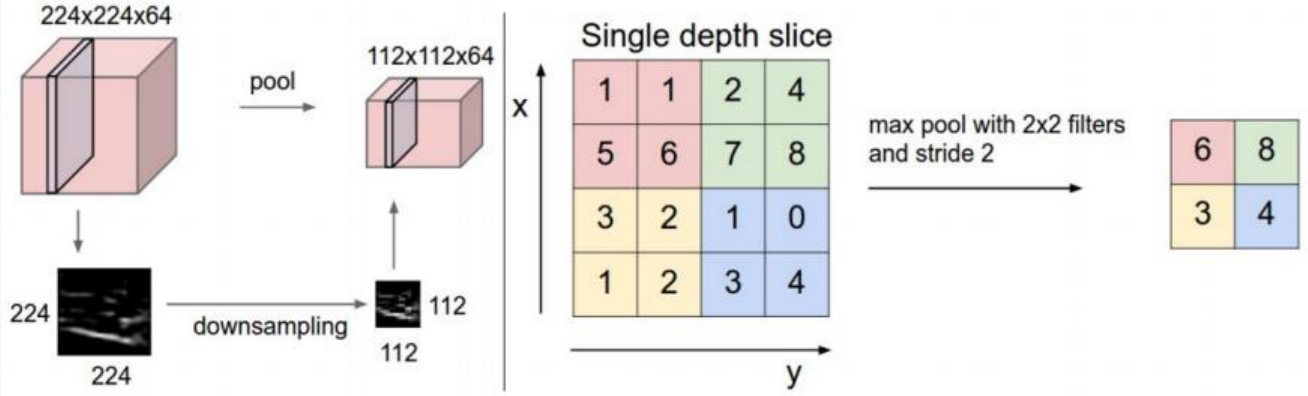
$$pool_{ave}\ (R_k) = \frac{1}{|R_k|} \sum_{i \in R_k}^{|R_k|} a_i$$

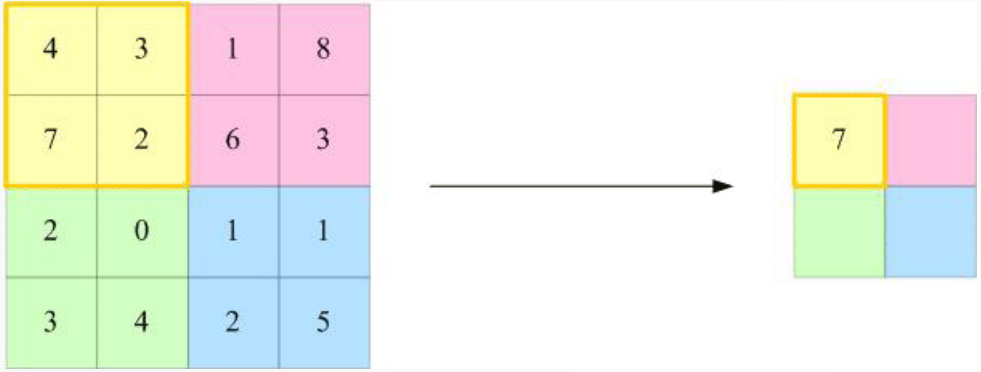
- TopK采样（Average Pooling）

$$pool_{k}\ (R_k) = \operatorname*{topk}_{i \in R_k} a_i$$

子采样使得下一层的神经元对一些小的形态改变保持不变性

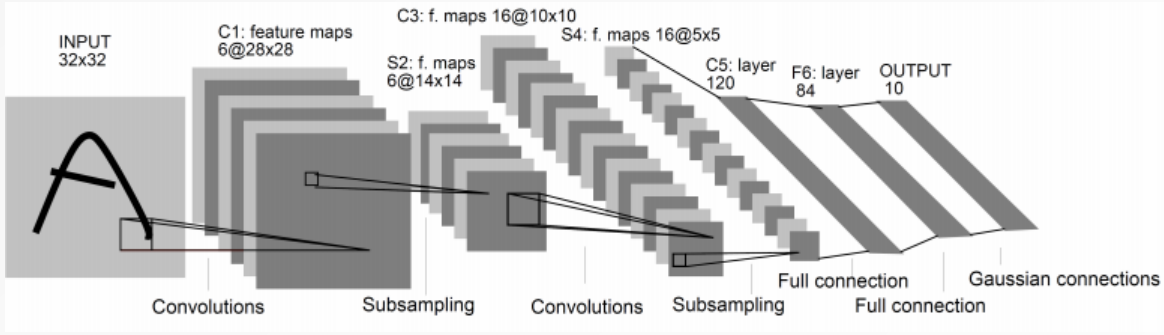
子采样层示例


224x224x64
pool
112x112x64
224
downsampling
112
112
Single depth slice
x
y
1 1 2 4
5 6 7 8
3 2 1 0
1 2 3 4
max pool with 2x2 filters
and stride 2
6 8
3 4


4 3 1 8
7 2 6 3
2 0 1 1
3 4 2 5
7

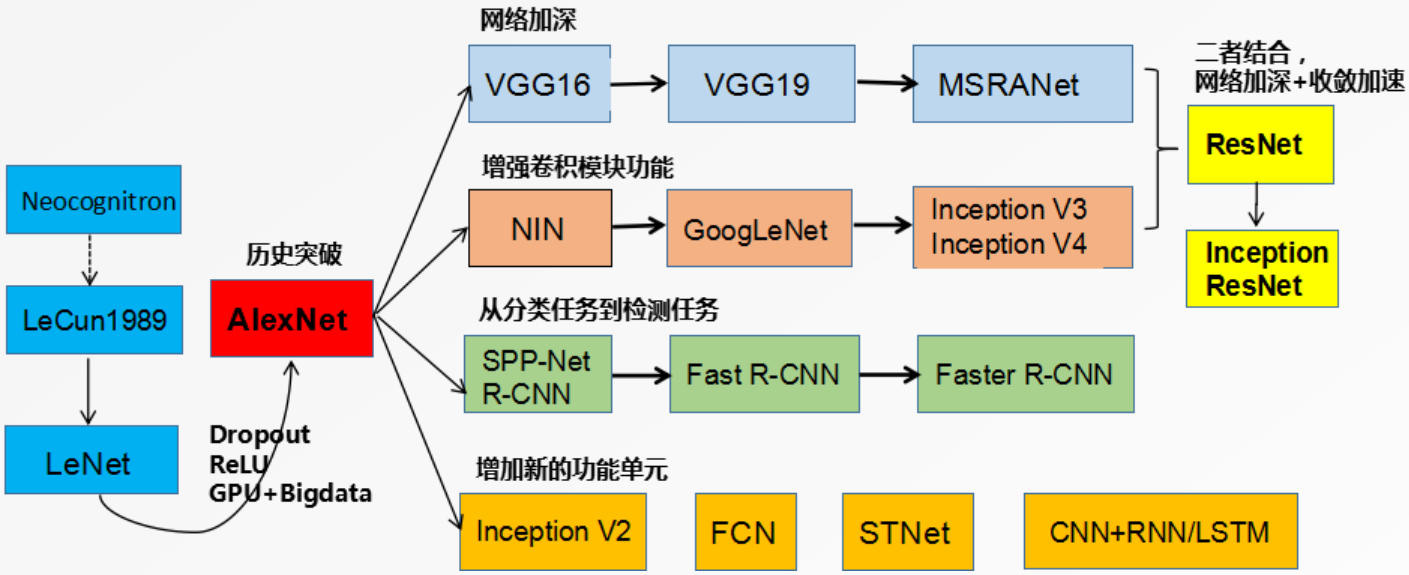
CNN在图像处理中的应用

LeNet-5 虽然提出时间比较早，但是是一个非常成功的神经网络模型。基于 LeNet-5 的手写数字识别系统在 90 年代被美国很多银行使用，用来识别支票上面的手写数字。LeNet-5共有 7 层。



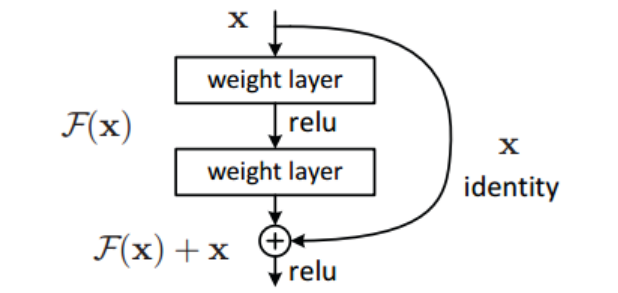
LeNet-5 网络结构

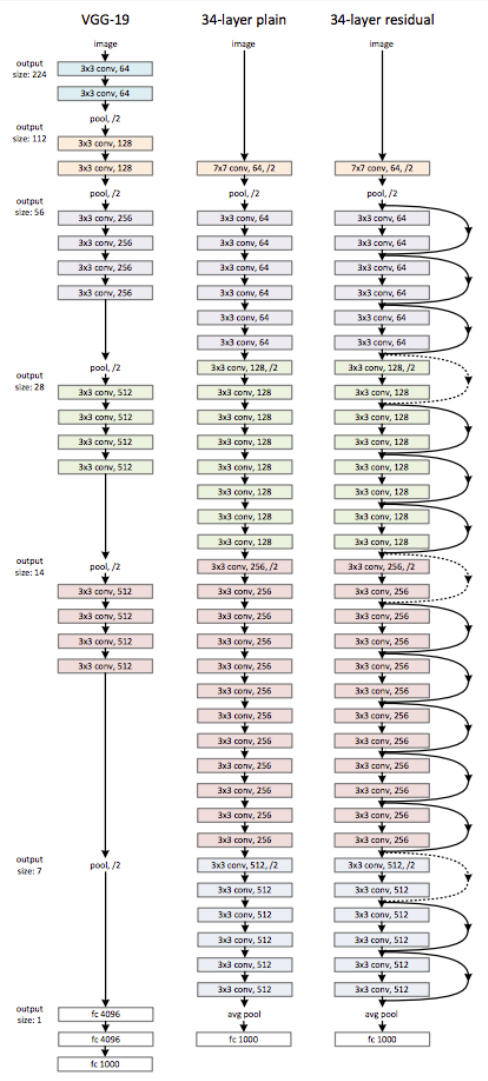
AlexNet


网络加深
VGG16
VGG19
MSRANet
二者结合，
网络加深+收敛加速
ResNet
Inception
ResNet
增强卷积模块功能
NIN
GoogLeNet
Inception V3
Inception V4
Neocognitron
LeCun1989
LeNet
历史突破
AlexNet
Dropout
ReLU
GPU+Bigdata
从分类任务到检测任务
SPP-Net
R-CNN
Fast R-CNN
Faster R-CNN
增加新的功能单元
Inception V2
FCN
STNet
CNN+RNN/LSTM

ResNet

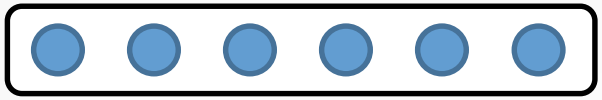
- 2015 ILSVRC winner
- 152层
- 错误率：3.57%





CNN在自然语言处理中的应用

I want to play with that boy.



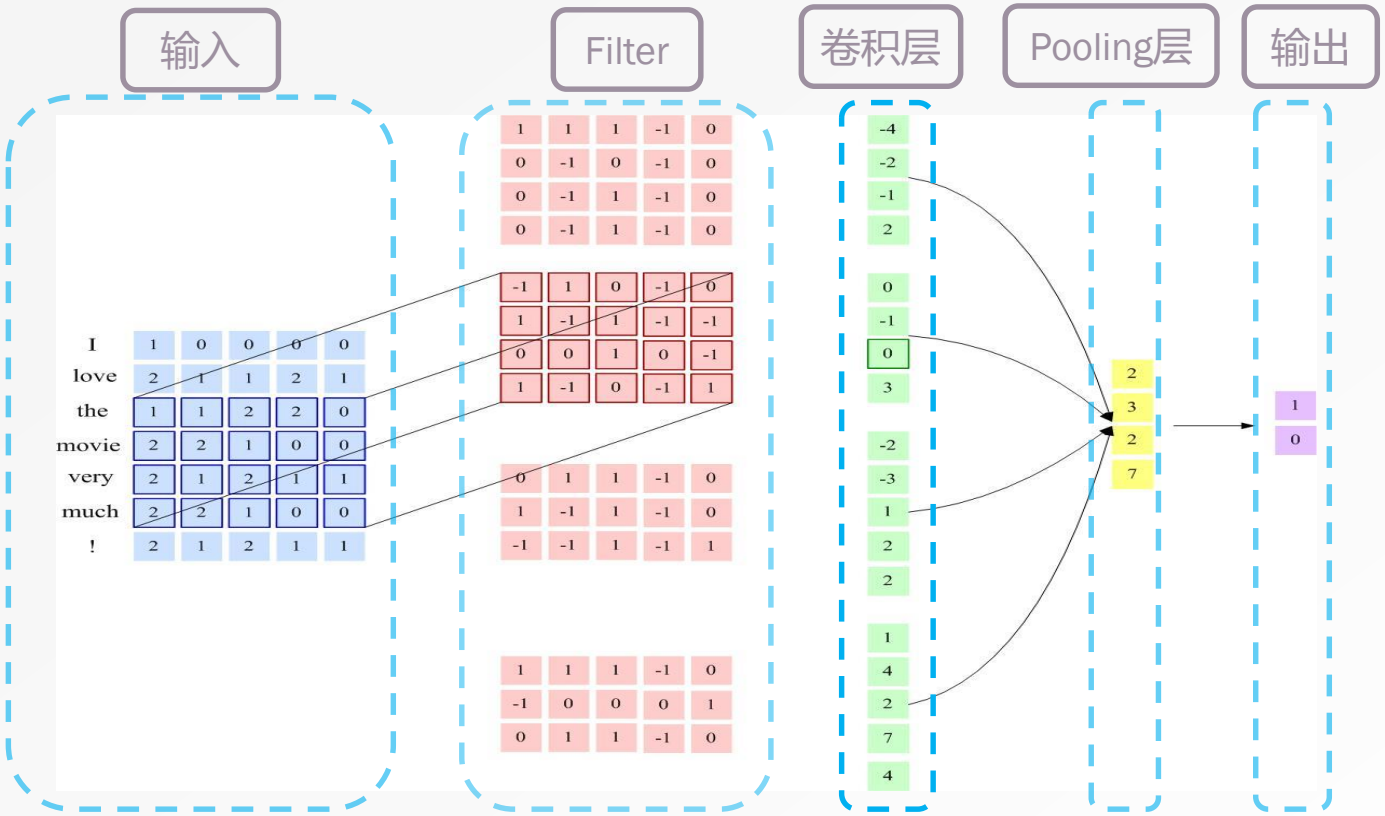
文本分类：情感分类

- 任务描述给定一个句子，判断其情感倾向的正负性（二分类）
  - 输入：一个句子
  - 输出：分类结果

正面：I love this movie.
负面：The director could do better than this.

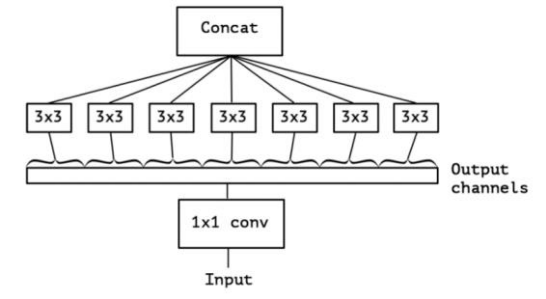
CNN-句子建模框架

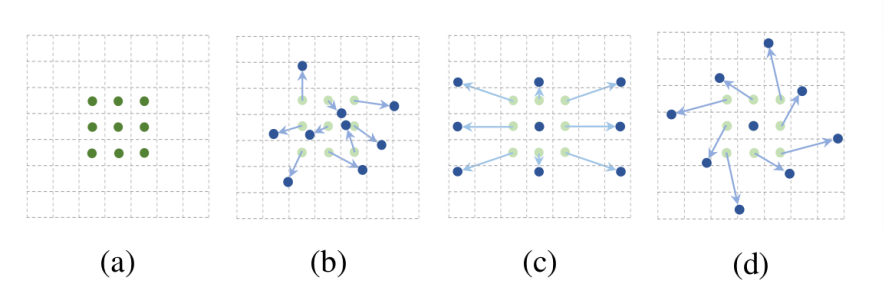
paper: http://arxiv.org/pdf/1510.03820v4.pdf
code: https://github.com/dennybritz/cnn-text-classification-tf



其他各种（*奇怪的*）卷积模型

- 多隐层非线性版本
- 空洞卷积
- 深度可分离卷积
- 可变形卷积（不规则卷积）
- 特征重标定卷积
- …



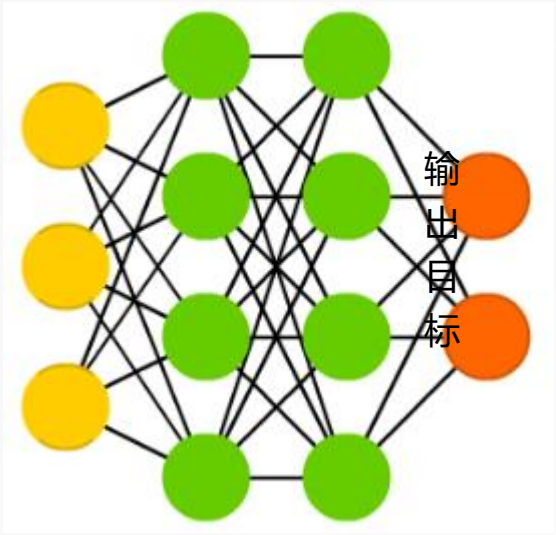


https://zhuanlan.zhihu.com/p/29367273

前馈神经网络

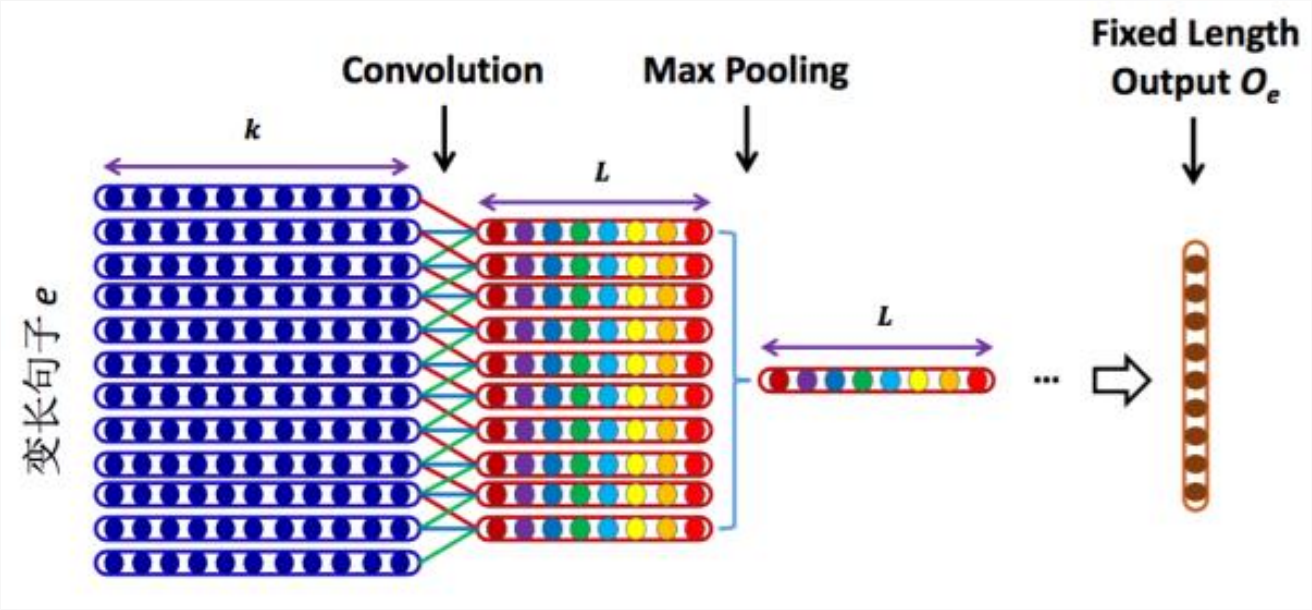
隐层表示/抽象

输入句子



输出目标

卷积神经网络



前馈神经网络的不足

- 连接存在层与层之间，每层的节点之间是无连接的。
- 输入和输出的维数都是固定的，不能任意改变。无法处理变长的序列数据。
- 假设每次输入都是独立的，也就是说每次网络的输出只依赖于当前的输入。

各种处理任务

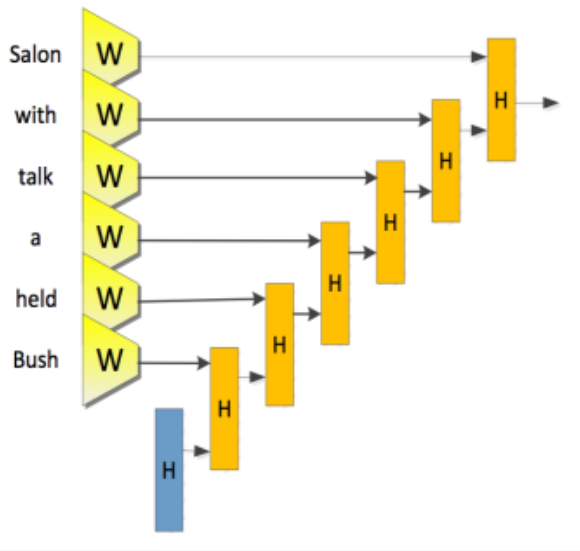
- 变长输入
  - 不同大小的图片
  - 时长不一的视频
  - 长短不同的句子
  - 序列长度不同的对话
  - …
- 复杂结构输入

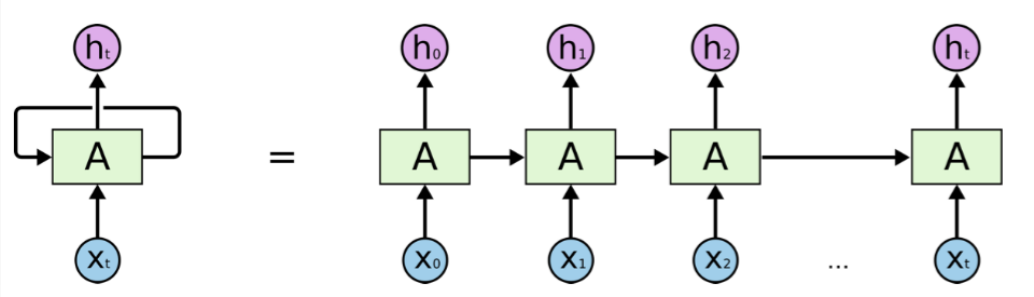
（树、表、图）

- 相互依赖
  - 前言要搭后语
  - 动画片由连续的图片组成
  - 字词搭配
  - 土豪陪美女
  - …
- 复杂结构输出

（句法树、知识图谱）

循环神经网络





循环神经网络

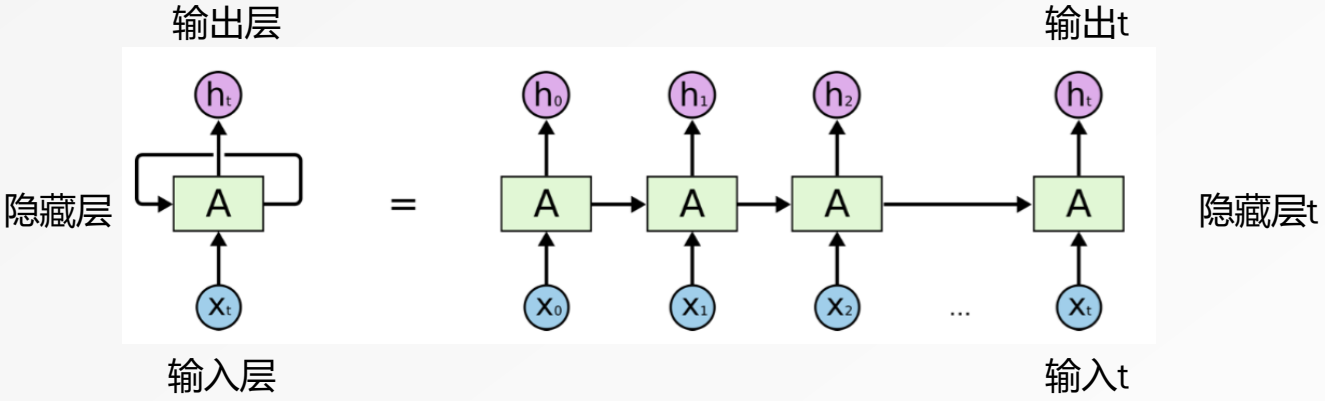
- **循环神经网络**（Recurrent Neural Network，RNN），也叫递归神经网络。这里为了区别与另外一种**递归神经网络**（Recursive Neural Network），我们称为**循环神经网络**。
  - 前馈神经网络的输入和输出的维数都是固定的，不能任意改变。无法处理变长的序列数据。
  - 假设每次输入都是独立的，也就是说每次网络的输出只依赖于当前的输入。
- 循环神经网络通过使用带自反馈的神经元，能够处理任意长度的序列。循环神经网络比前馈神经网络更加符合生物神经网络的结构。循环神经网络已经被广泛应用在语音识别、图像处理、语言模型以及自然语言生成等任务上。

循环神经网络

- 给定一个输入序列$\boldsymbol{x}^{(1:n)} = (\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(t)}, \ldots, \boldsymbol{x}^{(n)})$，循环神经网络通过下面公式更新带反馈边的隐藏层的活性值 h(t)，即抽象表示：

$$\mathbf{h}_t = \begin{cases} 0 & t = 0 \\ f(\mathbf{h}_{t-1}, \mathbf{x}_t) & \text{otherwise} \end{cases}$$

循环神经网络的示例



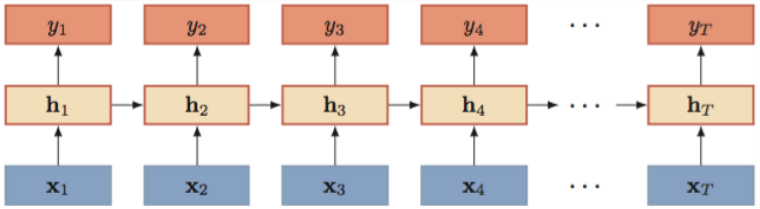
简单循环网络

- 假设时刻 t 时，输入为 x_t ，隐层状态（隐层神经元活性）为 h_t 。 h_t 不仅和当前时刻的输入相关，也和上一个时刻的隐层状态相关。

- 一般我们使用如下函数：

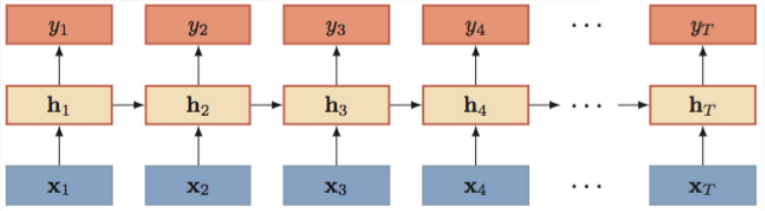
$$\boldsymbol{h}_t = f(\boldsymbol{U}\boldsymbol{h}_{\boldsymbol{t-1}} + \boldsymbol{W}\boldsymbol{x}_{\boldsymbol{t}} + b) \qquad \boldsymbol{y}_t = softmax(\boldsymbol{W}^{(S)}\boldsymbol{h}_t)$$

  - 这里，f 是非线性函数，通常为 sigmod函数或 tanh 函数。



简单循环神经网络的前向计算

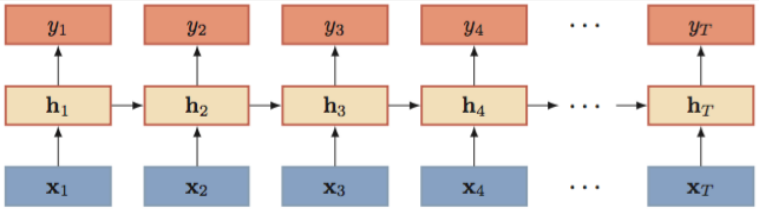
- U= [[1,1,1],[2,2,2],[2,1,2]]
- W=[[2,1],[1,2],[1,3]]
- x_1=[1,2]
- x_2=[2,3]
- x_3=[2,1]
- h_0 =[1,1,1]
- h_1 =?
- h_2 =?
- h_3 =?



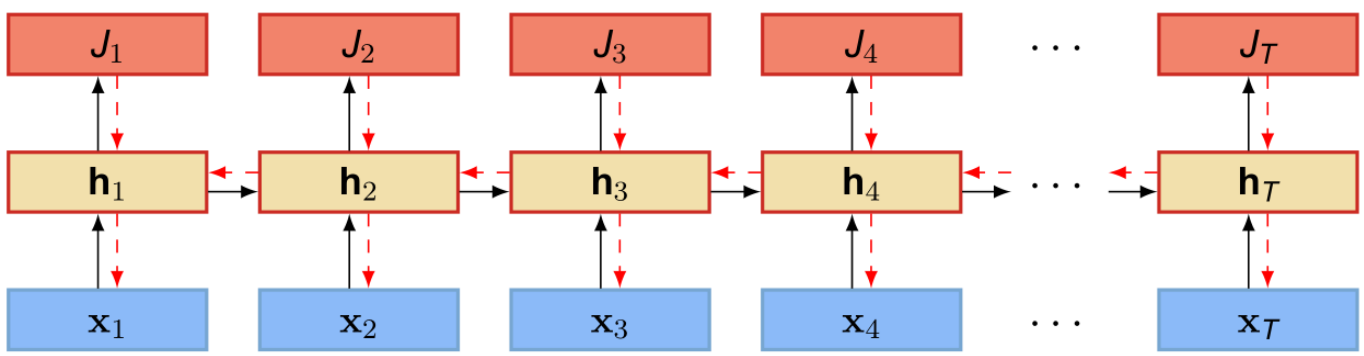
$$\boldsymbol{h_t = Uh_{t-1} + Wx_t}$$

$$\boldsymbol{h_t = f(Uh_{t-1} + Wx_t} + b)$$

梯度计算



- 循环神经网络的参数训练可以通过随时间进行反向传播（Backpropagation Through Time，BPTT）算法。



P. J. Werbos. “Backpropagation through time: what it does and how to do it”. In: Proceedings of the IEEE 10 (1990).

梯度

- 假设循环神经网络在每个时刻 t 都有一个监督信息，损失为 J_t。则整个序列的损失为 $J = \sum_{t=1}^{T} J_t$。
- 损失 J 关于 U 的梯度为：

$$\frac{\partial J}{\partial U} = \sum_{t=1}^{T} \frac{\partial J_t}{\partial U}$$
$$= \sum_{t=1}^{T} \frac{\partial \mathbf{h}_t}{\partial U} \frac{\partial J_t}{\partial \mathbf{h}_t},$$

  - 其中，$\boldsymbol{h}_t$ 是关于U和$\boldsymbol{h}_{t-1}$的函数，而$\boldsymbol{h}_{t-1}$又是关于U和$\boldsymbol{h}_{t-2}$的函数。

梯度

- 用链式法则得到

$$\boldsymbol{h}_t = f(\boldsymbol{U}\boldsymbol{h}_{\boldsymbol{t-1}} + \boldsymbol{W}\boldsymbol{x}_{\boldsymbol{t}} + b)$$

$$\frac{\partial J}{\partial U} = \sum_{t=1}^{T}\sum_{k=1}^{t} \frac{\partial \mathbf{h}_k}{\partial U}\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_k}\frac{\partial \mathbf{y}_t}{\partial \mathbf{h}_t}\frac{\partial J_t}{\partial \mathbf{y}_t},$$

- 其中，

$$\begin{aligned}\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_k} &= \prod_{i=k+1}^{t}\frac{\partial \mathbf{h}_i}{\partial \mathbf{h}_{i-1}} \\ &= \prod_{i=k+1}^{t} U^T \operatorname{diag}[f'(h_{i-1})].\end{aligned}$$

梯度

- 因此，总的梯度为

$$\frac{\partial J}{\partial U} = \sum_{t=1}^{T} \sum_{k=1}^{t} \frac{\partial \mathbf{h}_k}{\partial U} \left(\prod_{i=k+1}^{t} U^T \mathbf{diag}(f'(h_{i-1})) \right) \frac{\partial \mathbf{y}_t}{\partial \mathbf{h}_t} \frac{\partial J_t}{\partial \mathbf{y}_t}.$$

长期依赖问题/梯度消失问题

- 我们定义 $\gamma = ||\boldsymbol{U}^T \boldsymbol{diag}(f'(\boldsymbol{h}_{i-1}))||$，则在上面公式中的括号里面为 γ^{t-k} 。如果γ> 1，当 t − k → ∞ 时，$\gamma^{t-k} \to \infty$，会造成系统不稳定，也就是所谓的梯度爆炸**梯度爆炸**问题；相反，如果γ< 1，当 t − k → ∞ 时，$\gamma^{t-k} \to 0$，会出现和深度前馈神经网络类似的**梯度消失**问题。

- 因此，虽然简单循环网络从理论上可以建立长时间间隔的状态之间的依赖关系（Long-Term Dependencies），但是由于梯度爆炸或消失问题，实际上只能学习到短周期的依赖关系。这就是所谓的**长期依赖问题**。

长短时记忆神经网络：LSTM

- 长短时记忆神经网络（Long Short-Term Memory Neural Network，LSTM）是循环神经网络的一个变体，可以有效地解决简单循环神经网络的梯度爆炸或消失问题。

- LSTM 模型的关键是引入了一组记忆单元（Memory Units），允许网络可以学习何时遗忘历史信息，何时用新信息更新记忆单元。在时刻 t 时，记忆单元 $\mathbf{c}_t$ 记录了到当前时刻为止的所有历史信息，并受三个“门”控制：输入门 $\mathbf{i}_t$，遗忘门 $\mathbf{f}_t$ 和输出门 $\mathbf{o}_t$。三个门的元素的值在 [0, 1] 之间。

Long Short Term Memory (LSTM)

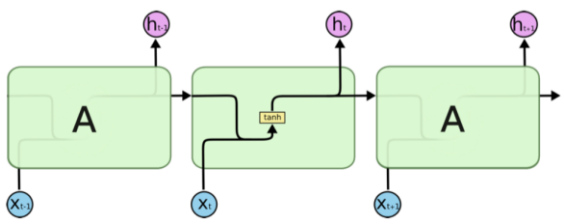
- 在时刻 t 时 LSTM 的更新方式如下：

$$\boldsymbol{h}_t = f(\boldsymbol{U}\boldsymbol{h}_{\boldsymbol{t-1}} + \boldsymbol{W}\boldsymbol{x}_{\boldsymbol{t}} + b)$$

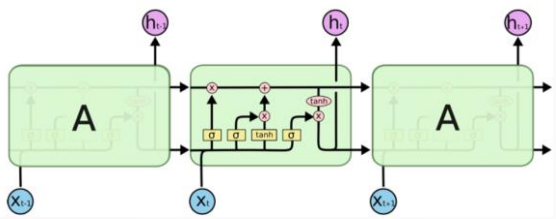
$$\begin{aligned}
\mathbf{i}_t &= \sigma(W_i\mathbf{x}_t + U_i\mathbf{h}_{t-1} + V_i\mathbf{c}_{t-1}),\\
\mathbf{f}_t &= \sigma(W_f\mathbf{x}_t + U_f\mathbf{h}_{t-1} + V_f\mathbf{c}_{t-1}),\\
\mathbf{o}_t &= \sigma(W_o\mathbf{x}_t + U_o\mathbf{h}_{t-1} + V_o\mathbf{c}_t),\\
\tilde{\mathbf{c}}_t &= \tanh(W_c\mathbf{x}_t + U_c\mathbf{h}_{t-1}),\\
\mathbf{c}_t &= \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{c}}_t,\\
\mathbf{h}_t &= \mathbf{o}_t \odot \tanh(\mathbf{c}_t),
\end{aligned}$$

这里，x_t是当前时刻的输入，σ 是 logistic 函数，V_i，V_f，V_o是对角矩阵。遗忘门 f_t控制每一个内存单元需要遗忘多少信息，输入门 i_t控制每一个内存单元加入多少新的信息，输出门o_t控制每一个内存单元输出多少信息。

Long Short Term Memory (LSTM)



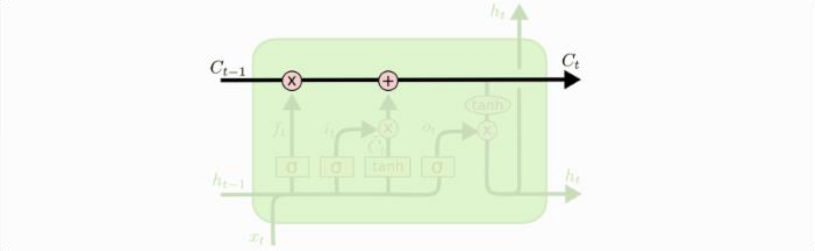
General RNN

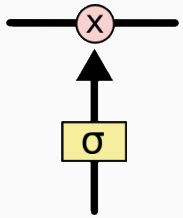


LSTM

LSTM

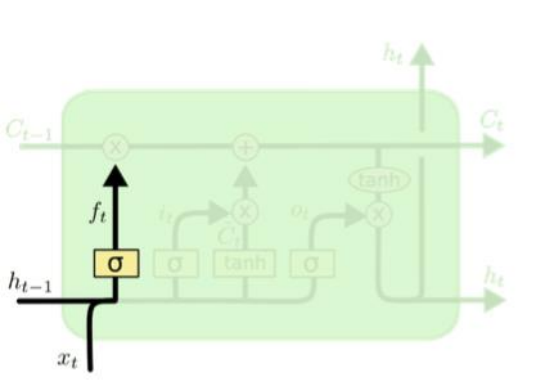
- 核心：记忆（细胞状态）





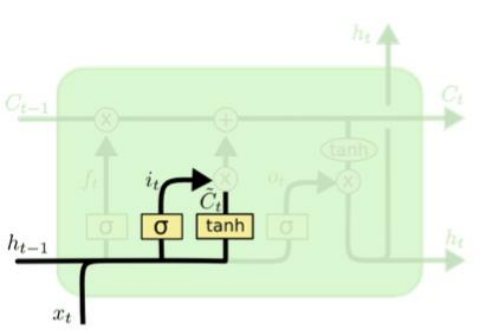
门
Sigmoid：1 or 0
Pointwise乘法

Forget Gate



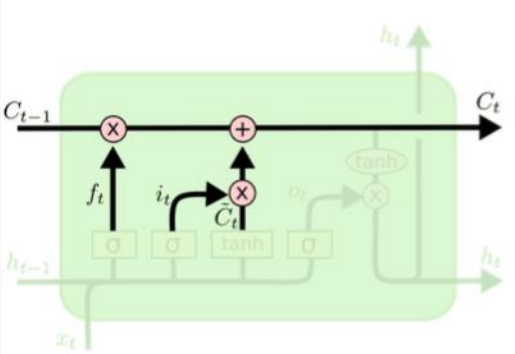
$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

Input Gate



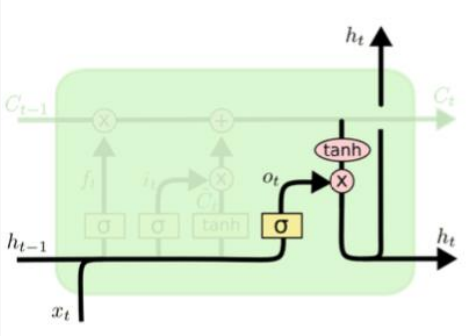
$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$
$$\tilde{C}_t = tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

Update Memory



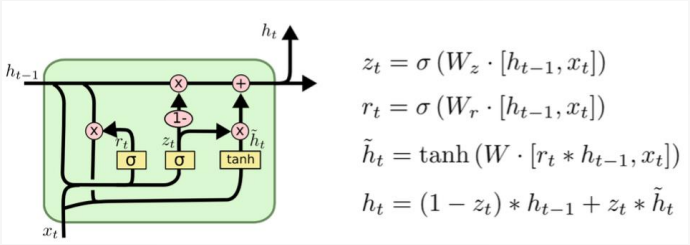
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Output Gate



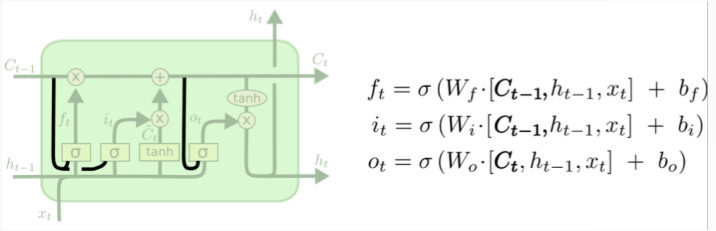
$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$
$$h_t = o_t * tanh(C_t)$$

LSTM变种



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

GRU



$$f_t = \sigma\left(W_f \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] + b_f\right)$$
$$i_t = \sigma\left(W_i \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] + b_i\right)$$
$$o_t = \sigma\left(W_o \cdot [\boldsymbol{C_t}, h_{t-1}, x_t] + b_o\right)$$

Peephole链接

门限循环单元：GRU

- 门限循环单元（Gated Recurrent Unit，GRU）是一种比 LSTM 更加简化的版本。在 LSTM 中，输入门和遗忘门是互补关系，因为同时用两个门比较冗余。GRU 将输入门与和遗忘门合并成一个门：更新门（Update Gate），同时还合并了记忆单元和神经元活性。GRU 模型中有两个门：更新门 z 和重置门 r。更新门 z 用来控制当前的状态需要遗忘多少历史信息和接受多少新信息。重置门 r用来控制候选状态中有多少信息是从历史信息中得到。

- GRU 模型的更新方式如下：

$$
\begin{aligned}
\mathbf{r}_t &= \sigma(\mathbf{W}_r\mathbf{x}_t + \mathbf{U}_r\mathbf{h}_{t-1}) \\
\mathbf{z}_t &= \sigma(\mathbf{W}_z\mathbf{x}_t + \mathbf{U}_z\mathbf{h}_{t-1}) \\
\tilde{\mathbf{h}}_t &= \tanh(\mathbf{W}_c\mathbf{x}_t + \mathbf{U}(\mathbf{r}_t \odot \mathbf{h}_{t-1})), \\
\mathbf{h}_t &= \mathbf{z}_t \odot \mathbf{h}_{t-1} + (1 - \mathbf{z}_t) \odot \tilde{\mathbf{h}}_t,
\end{aligned}
$$

  - 这里选择 tanh 函数也是因为其导数有更大的值域

BasicRNNCell

```
class BasicRNNCell(RNNCell):
  def __init__(self, num_units, input_size=None, activation=tanh):
    if input_size is not None:
      logging.warn("%s: The input_size parameter is deprecated.", self)
    self._num_units = num_units
    self._activation = activation

  @property
  def state_size(self):
    return self._num_units

  @property
  def output_size(self):
    return self._num_units

  def __call__(self, inputs, state, scope=None):
    """Most basic RNN: output = new_state = activation(W * input + U * state + B)."""
    with vs.variable_scope(scope or type(self).__name__):  # "BasicRNNCell"
      output = self._activation(_linear([inputs, state], self._num_units, True))
    return output, output
```

$$\boldsymbol{h_t} = f(\boldsymbol{U h_{t-1}} + \boldsymbol{W x_t} + b)$$

BasicLSTMCell

$$
\begin{aligned}
f_t &= \sigma_g(W_f x_t + U_f h_{t-1} + b_f) \\
i_t &= \sigma_g(W_i x_t + U_i h_{t-1} + b_i) \\
o_t &= \sigma_g(W_o x_t + U_o h_{t-1} + b_o) \\
c_t &= f_t \circ c_{t-1} + i_t \circ \sigma_c(W_c x_t + U_c h_{t-1} + b_c) \\
h_t &= o_t \circ \sigma_h(c_t)
\end{aligned}
$$

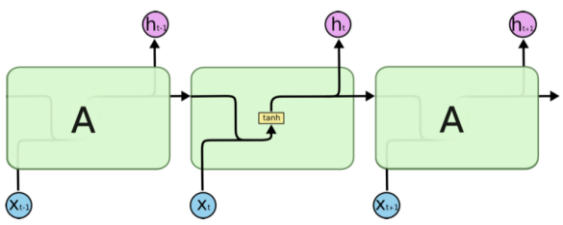
```
def call(self, inputs, state):
  """Long short-term memory cell (LSTM)."""
  sigmoid = math_ops.sigmoid
  # Parameters of gates are concatenated into one multiply for efficiency.
  if self._state_is_tuple:
    c, h = state
  else:
    c, h = array_ops.split(value=state, num_or_size_splits=2, axis=1)

  concat = _linear([inputs, h], 4 * self._num_units, True)

  # i = input_gate, j = new_input, f = forget_gate, o = output_gate
  i, j, f, o = array_ops.split(value=concat, num_or_size_splits=4, axis=1)

  new_c = (
      c * sigmoid(f + self._forget_bias) + sigmoid(i) * self._activation(j))
  new_h = self._activation(new_c) * sigmoid(o)

  if self._state_is_tuple:
    new_state = LSTMStateTuple(new_c, new_h)
  else:
    new_state = array_ops.concat([new_c, new_h], 1)
  return new_h, new_state
```

GRUCell

$$
\begin{aligned}
z_t &= \sigma_g(W_z x_t + U_z h_{t-1} + b_z) \\
r_t &= \sigma_g(W_r x_t + U_r h_{t-1} + b_r) \\
h_t &= z_t \circ h_{t-1} + (1 - z_t) \circ \sigma_h(W_h x_t + U_h(r_t \circ h_{t-1}) + b_h)
\end{aligned}
$$

```
def call(self, inputs, state):
  """Gated recurrent unit (GRU) with nunits cells."""
  with vs.variable_scope("gates"):  # Reset gate and update gate.
    # We start with bias of 1.0 to not reset and not update.
    bias_ones = self._bias_initializer
    if self._bias_initializer is None:
      dtype = [a.dtype for a in [inputs, state]][0]
      bias_ones = init_ops.constant_initializer(1.0, dtype=dtype)
    value = math_ops.sigmoid(
        _linear([inputs, state], 2 * self._num_units, True, bias_ones,
                self._kernel_initializer))
    r, u = array_ops.split(value=value, num_or_size_splits=2, axis=1)
  with vs.variable_scope("candidate"):
    c = self._activation(
        _linear([inputs, r * state], self._num_units, True,
                self._bias_initializer, self._kernel_initializer))
  new_h = u * state + (1 - u) * c
  return new_h, new_h
```

rnn



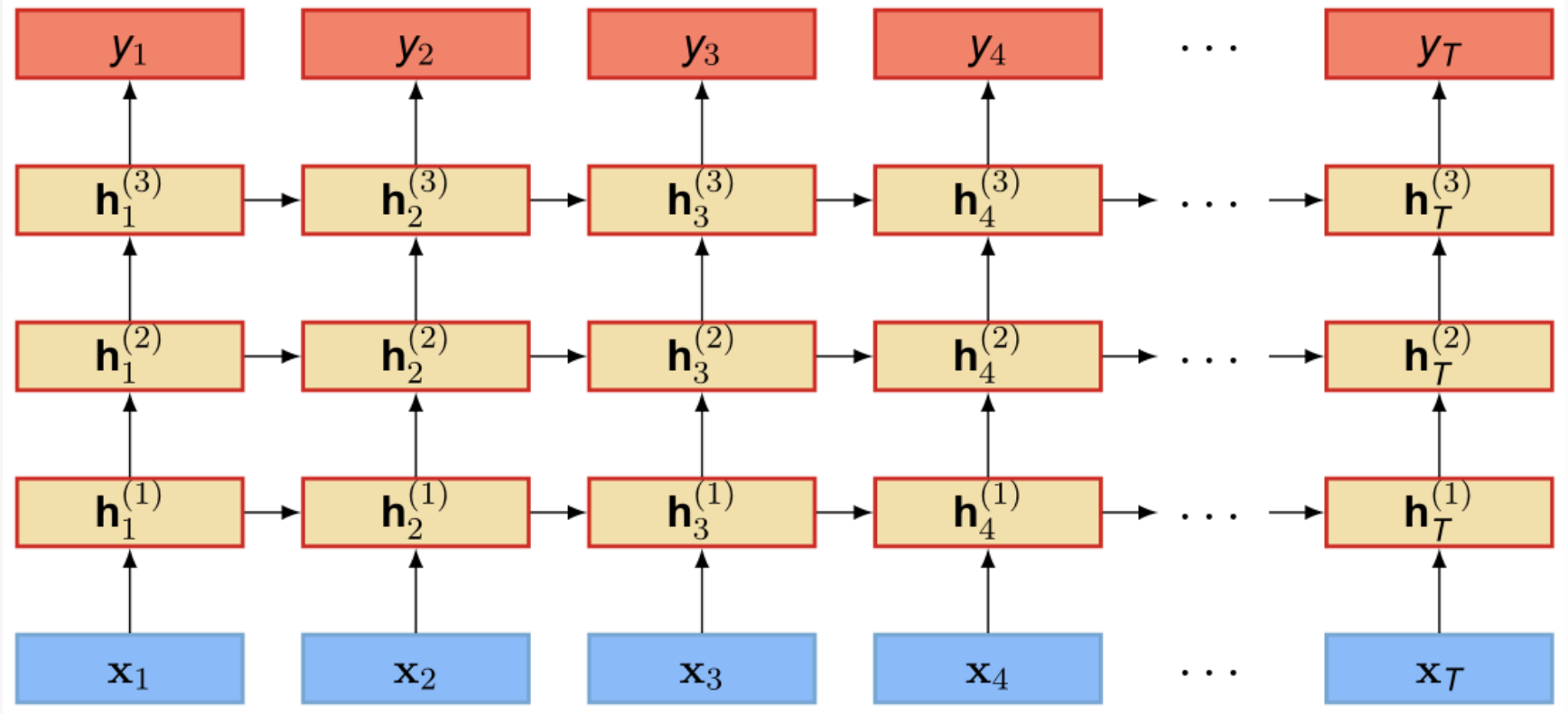
outputs, states = rnn.rnn(cell, embed_inputs, sequence_length=self.lengths, dtype=dtypes.float32)

```
for time, input_ in enumerate(inputs):
  if time > 0:
    varscope.reuse_variables()
  # pylint: disable=cell-var-from-loop
  call_cell = lambda: cell(input_, state)
  # pylint: enable=cell-var-from-loop
  if sequence_length is not None:
    (output, state) = _rnn_step(
        time=time,
        sequence_length=sequence_length,
        min_sequence_length=min_sequence_length,
        max_sequence_length=max_sequence_length,
        zero_output=zero_output,
        state=state,
        call_cell=call_cell,
        state_size=cell.state_size)
  else:
    (output, state) = call_cell()

  outputs.append(output)
```

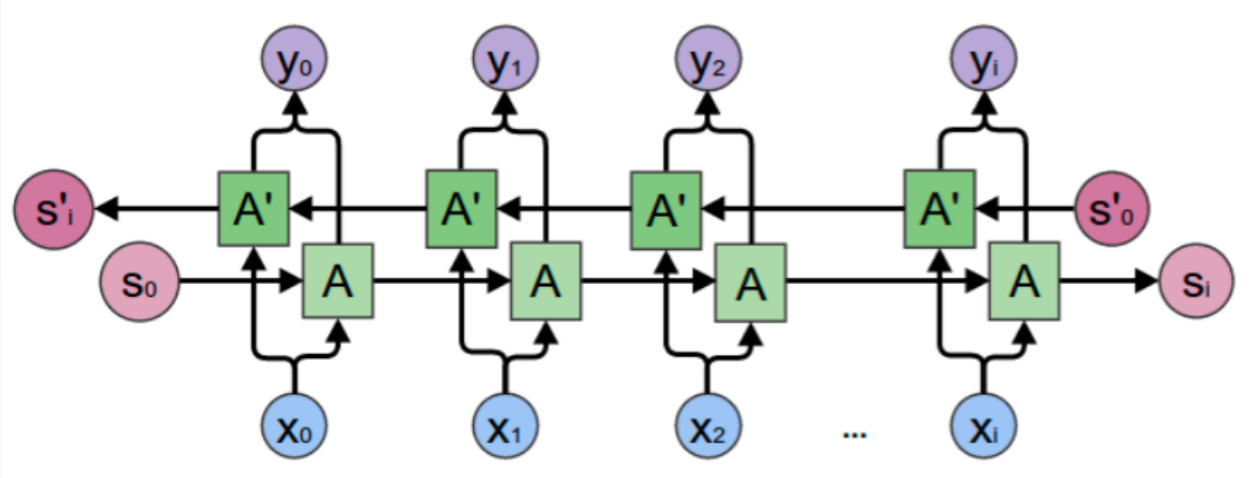
深层循环神经网络

- 循环神经网络的深度是一个有争议的话题。
- 一方面来说，如果我们把循环网络按时间展开，不同时刻的状态之间存在非线性连接，循环网络已经是一个非常深的网络了。
- 从另一方面来说，这个网络是非常浅的。任意两个相邻时刻的隐藏状态（$\boldsymbol{h_{t-1} \rightarrow h_t}$），隐藏状态到输出（$\boldsymbol{h_{t-1} \rightarrow y_t}$），以及输入到隐藏状态之间（$\boldsymbol{x_t \rightarrow h_t}$）之间的转换只有一个非线性函数。

堆叠(Stack)循环神经网络



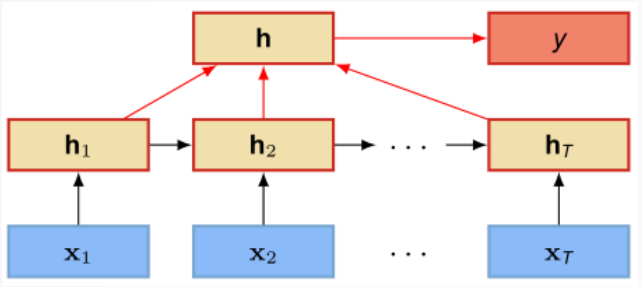
双向循环神经网络



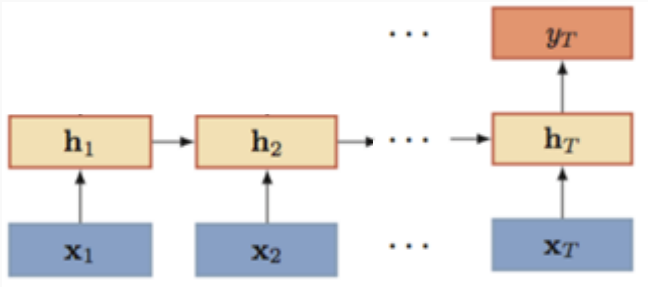
按时间展开的双向循环神经网络

应用：序列到类别

- 输入为序列，输出为类别。比如在文本分类中，输入数据为单词的序列，输出为该文本的类别。



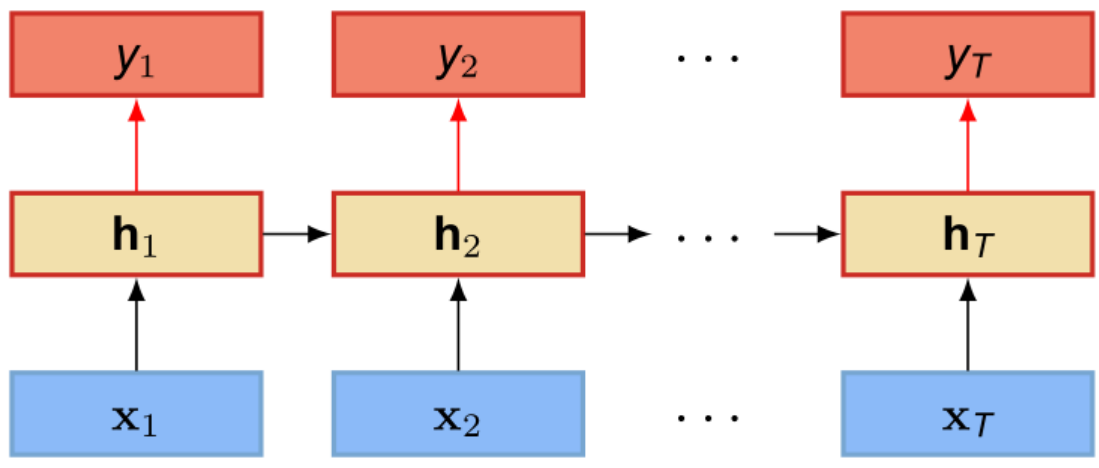
平均



不平均

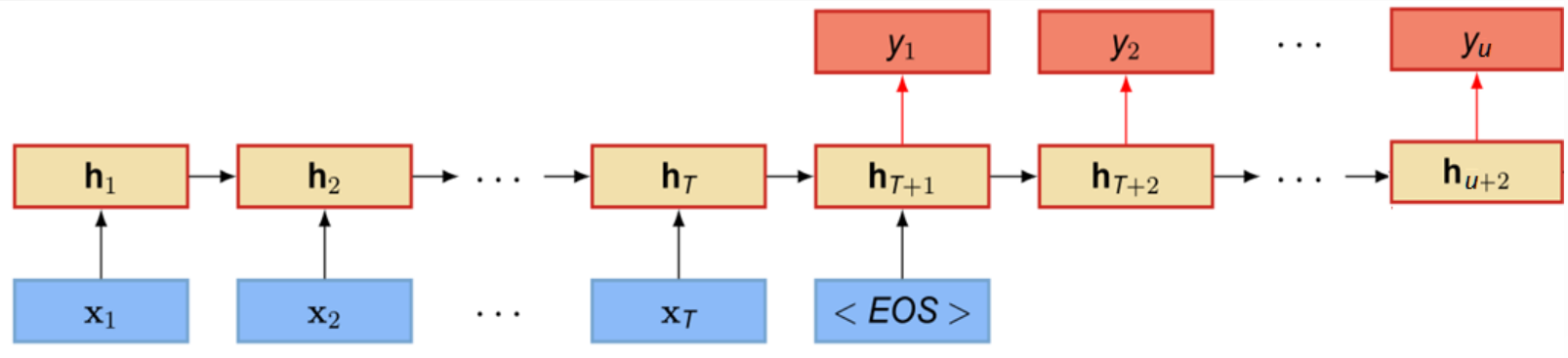
应用：同步序列到序列

- 输入和输出同步，即每一时刻都有输入和输出。比如在序列标注问题，每个时刻的输入都需要有一个输出。输入序列和输出序列的长度相同。

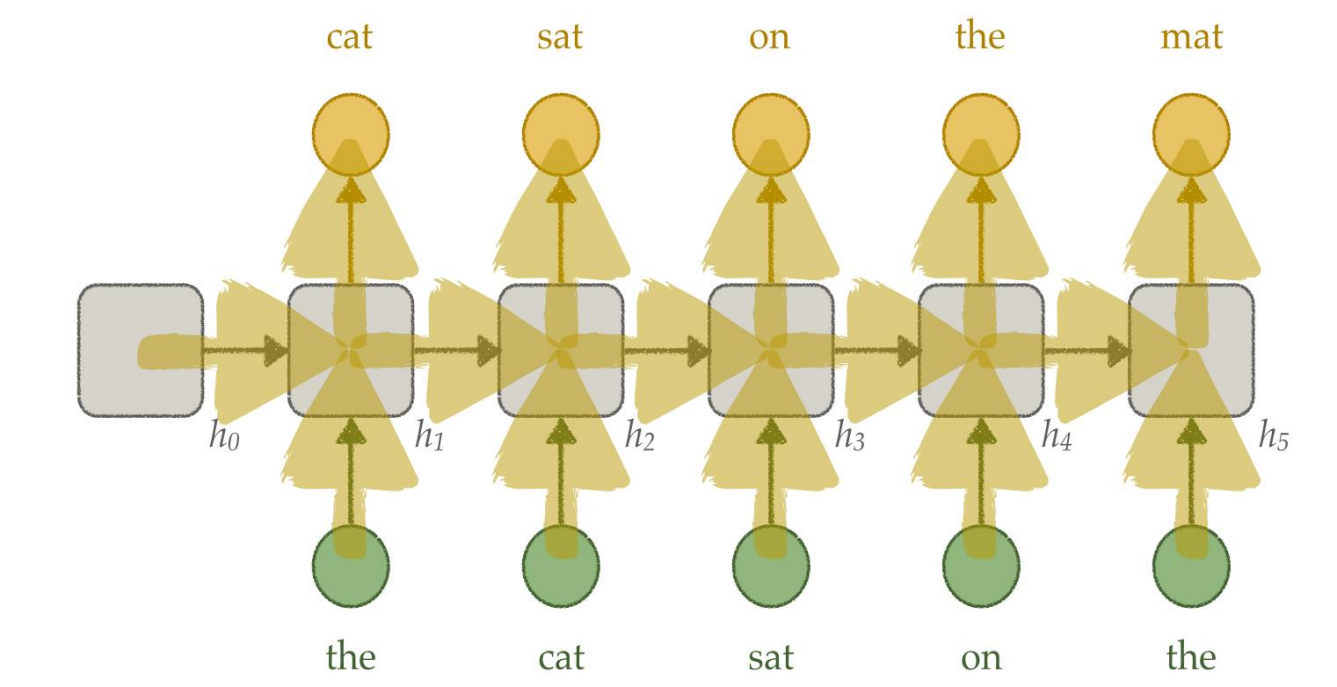


应用：异步序列到序列

- 输入和输出不需要有严格的对应关系。比如在机器翻译中，输入为源语言的单词序列，输出为目标语言的单词序列。输入和输出序列并不需要保持相同的长度。

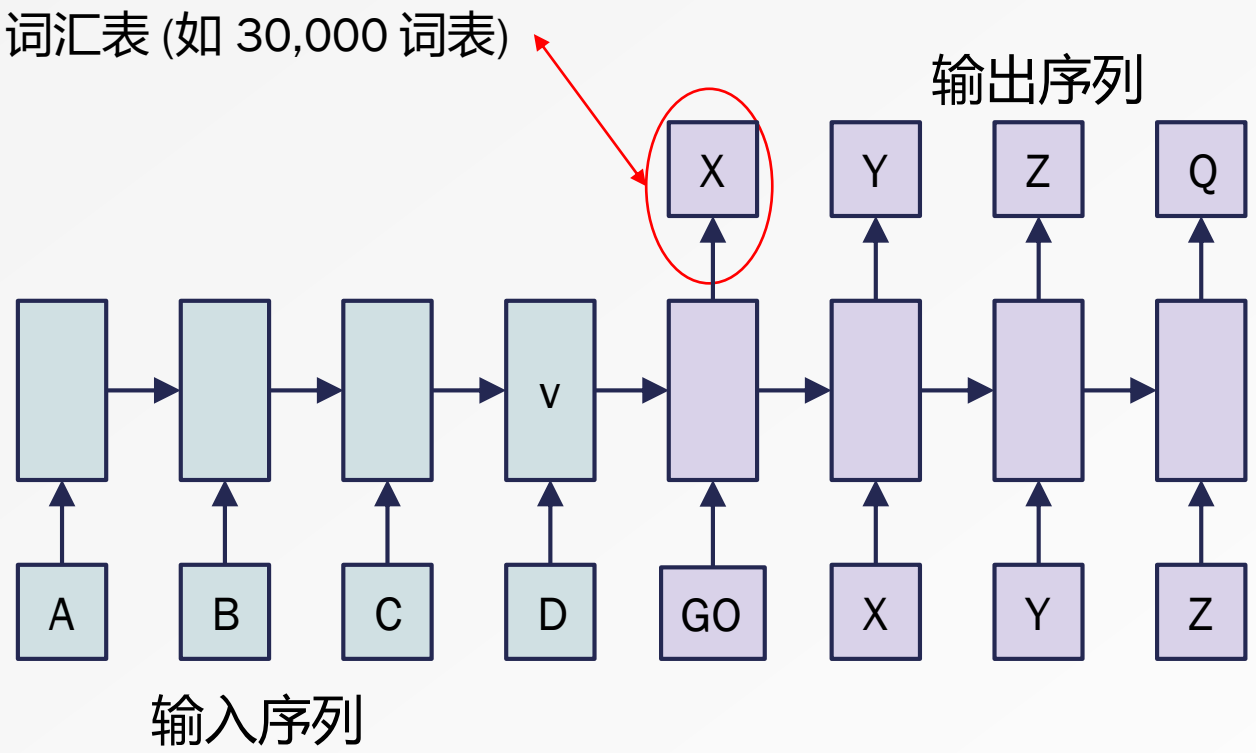


基于RNN的语言模型



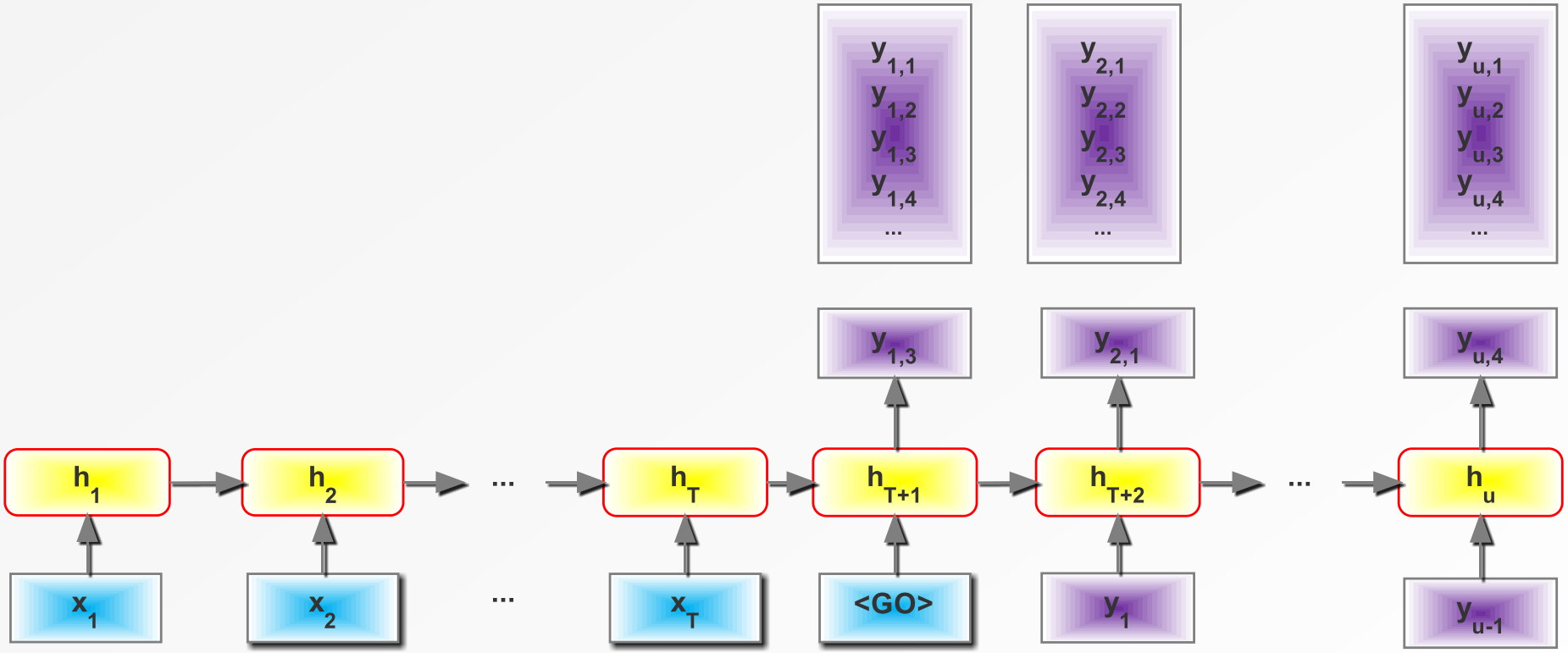
[ICML 2017, Tutorial]

Seq2Seq：Sequence to Sequence Learning

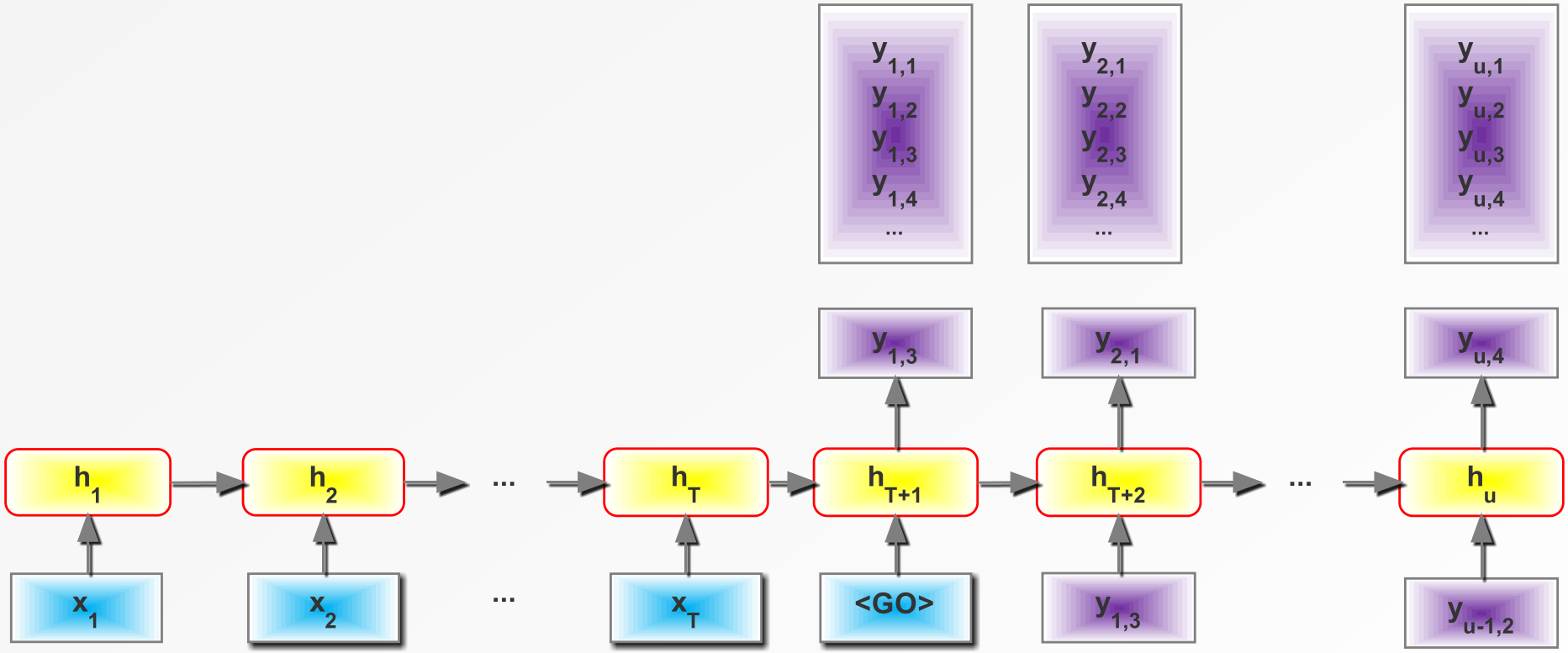


$$P(y_1, \ldots, y_{T'} | x_1, \ldots, x_T) = \prod_{t=1}^{T'} p(y_t | v, y_1, \ldots, y_{t-1})$$

Seq2Seq: train-time

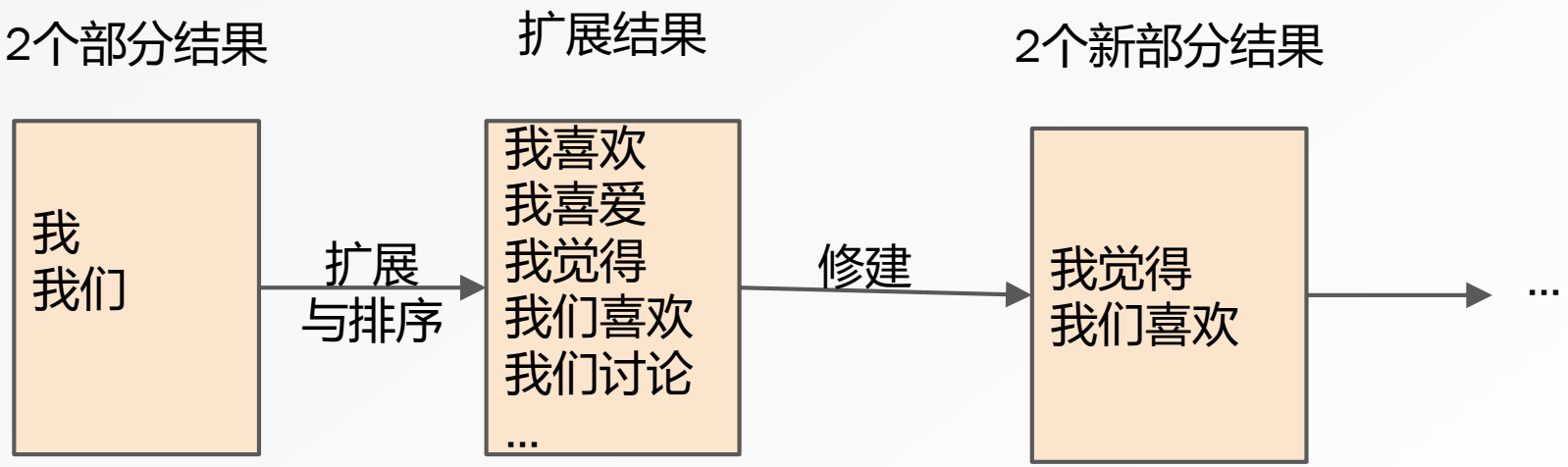


Seq2Seq: test-time



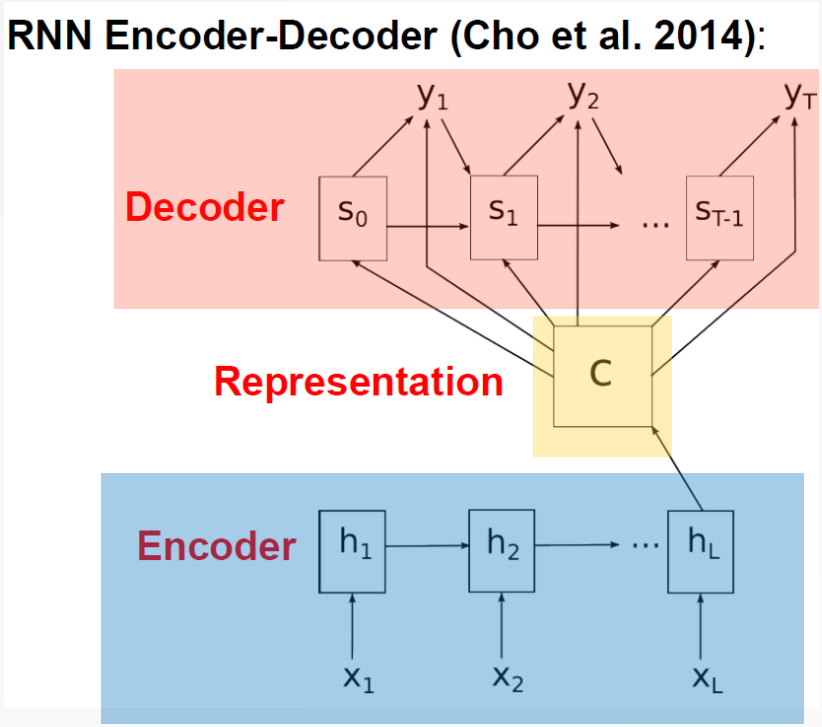
Beam Search（解码过程）

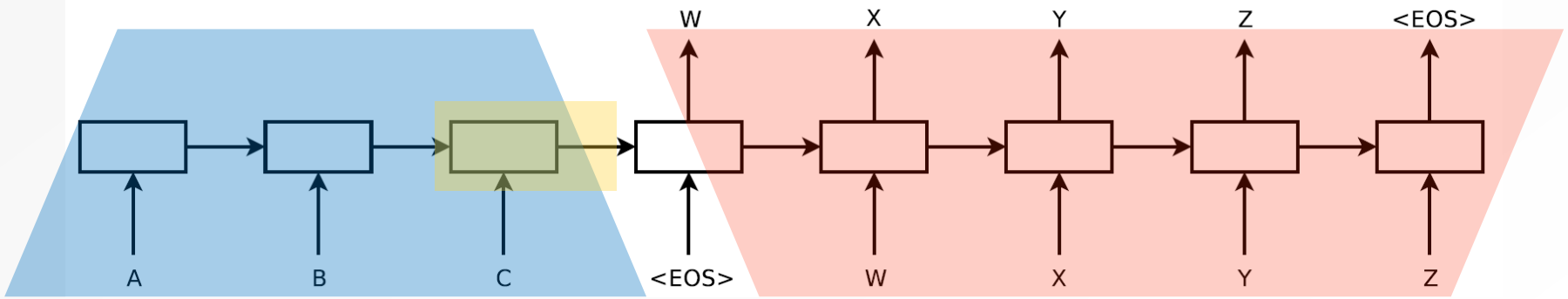
$$y^* = \arg \max_{y_1,\ldots,y_{T'}} P(y_1, \ldots, y_{T'} | x_1, \ldots, x_T)$$



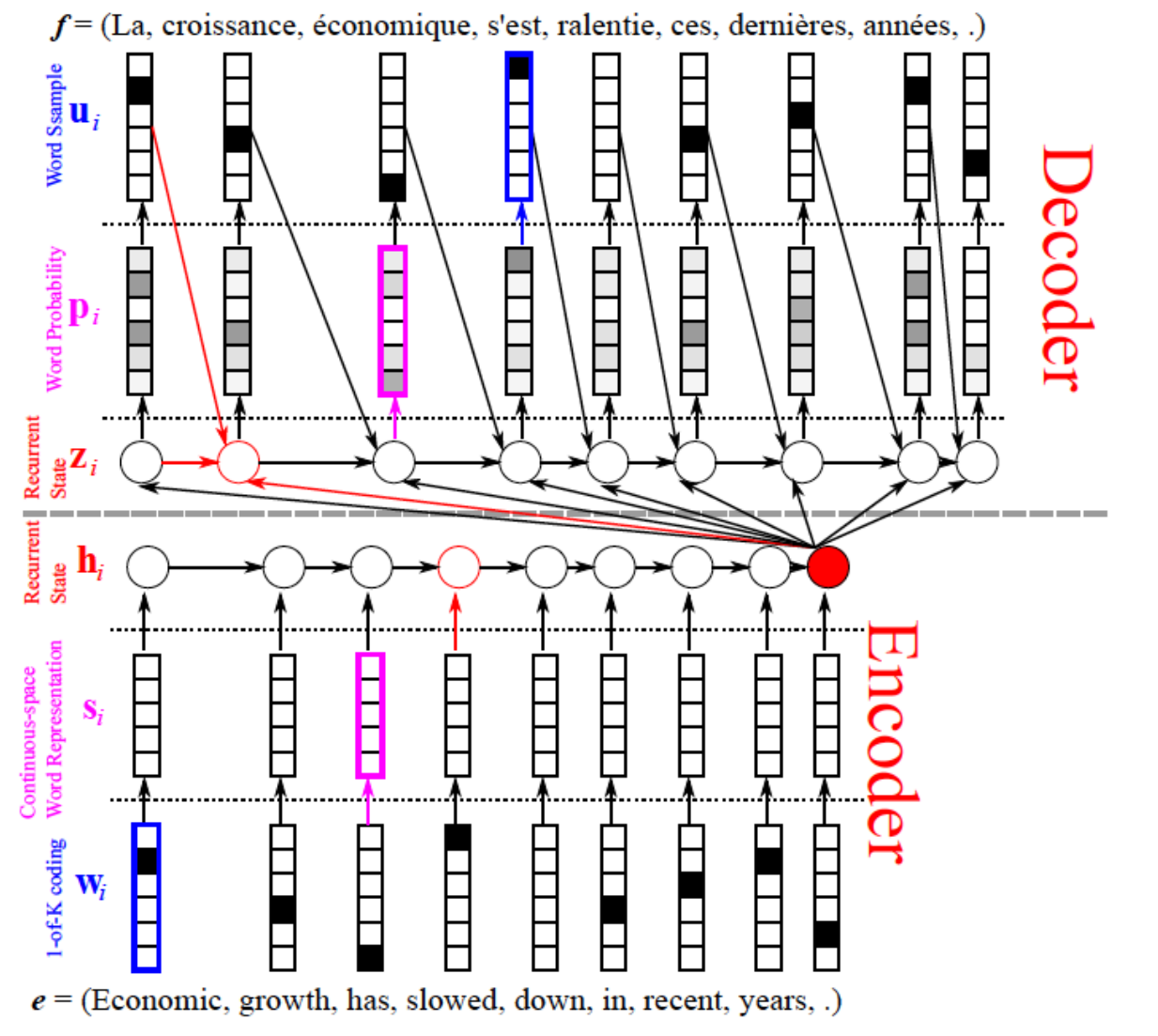
Beam为2的解码过程示例

RNN-based Seq2Seq



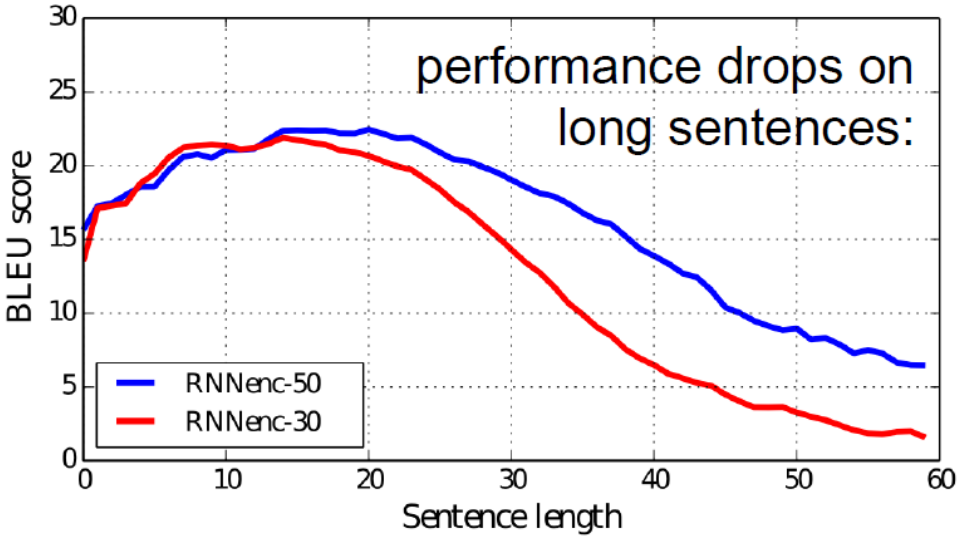


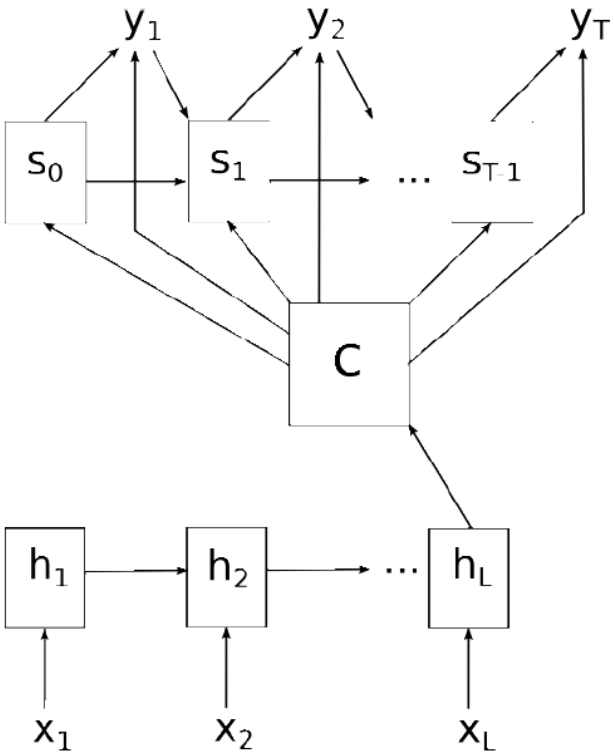
Neural Machine Translation



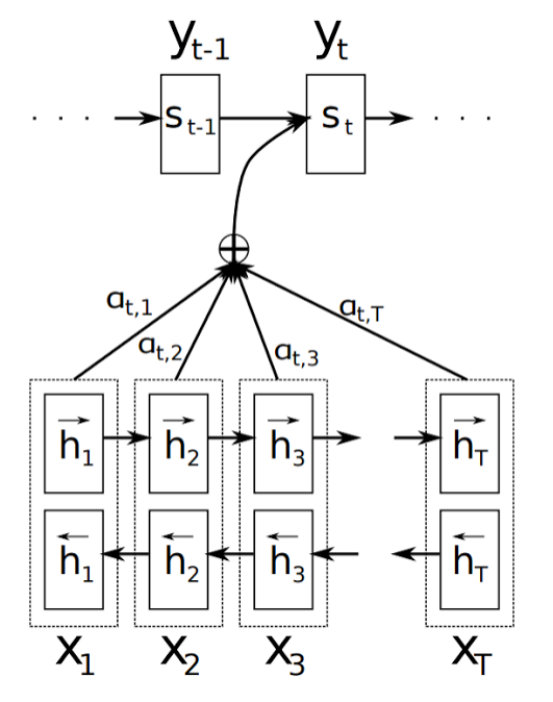
Issues

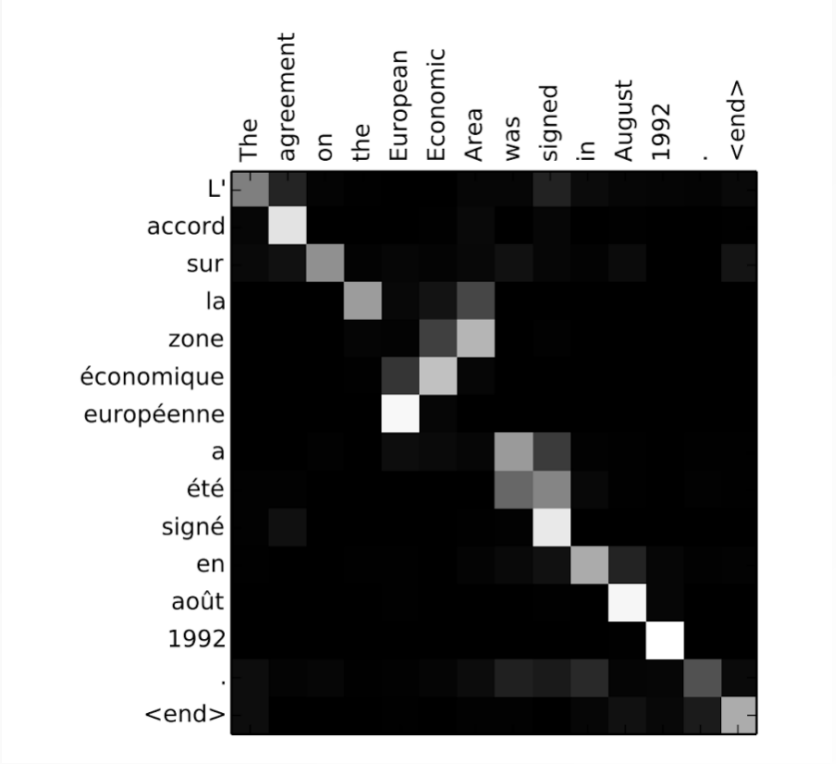
- has to remember the whole sentence
- fixed size representation can be the bottleneck
- humans do it differently





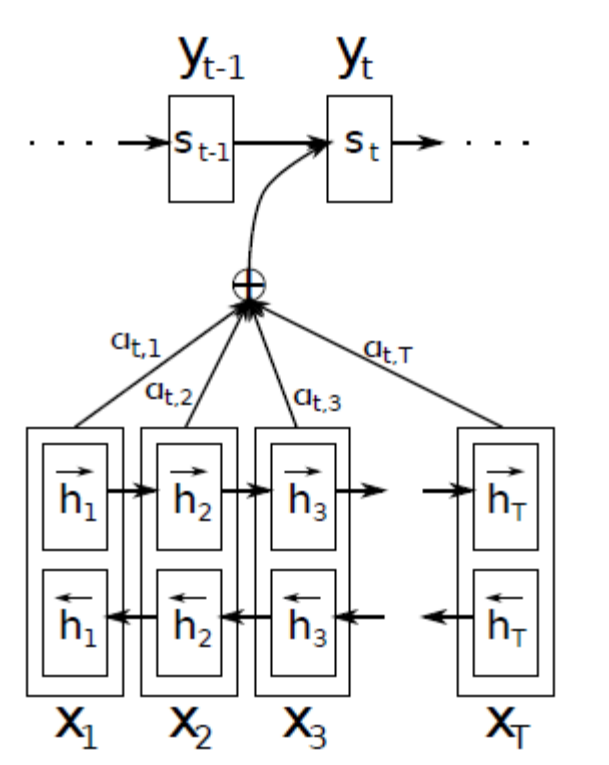
Seq2Seq with Attention





Neural Machine Translation by Jointly Learning to Align and Translate (ICLR 2015)

Seq2Seq with Attention



$$p(y_i|y_1,\ldots,y_{i-1},\mathbf{x}) = g(y_{i-1},s_i,c_i)$$

$$s_i = f(s_{i-1},y_{i-1},c_i)$$

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j$$

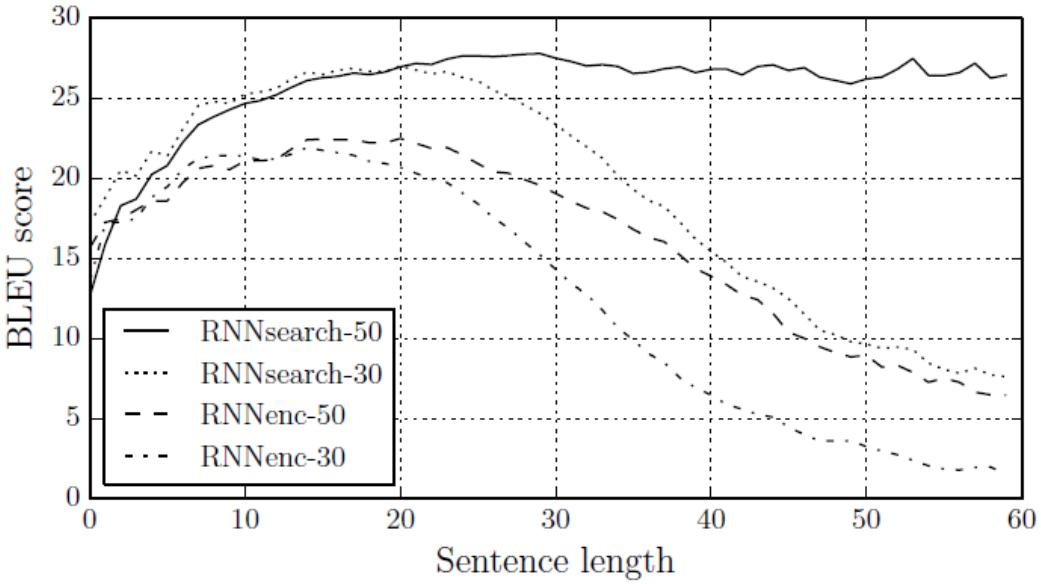
$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{ik})}$$

$$e_{ij} = a(s_{i-1},h_j)$$

Key: 当前应该利用什么信息进行翻译?

$$a(s_{i-1},h_j) = v_a^\top \tanh(W_a s_{i-1} + U_a h_j)$$

Seq2Seq with Attention



Model	All	No UNK°
RNNencdec-30	13.93	24.19
RNNsearch-30	21.50	31.44
RNNencdec-50	17.82	26.71
RNNsearch-50	26.75	34.16
RNNsearch-50⋆	28.45	36.15
Moses	33.30	35.63

阅读理解中的Attention机制

by *ent362* , *ent300* updated 6:06 pm et , thu march 26 , 2015 (*ent300*) the `` *ent321* '' series will have to handcuff a new director . *ent201* , who directed `` *ent71* , '' told *ent286* that she wo n't be back for the sequel , `` *ent100* . '' `` directing ' *ent135* ' has been an intense and incredible journey for which i am hugely grateful , '' she said in a statement to the site . `` while i will not be returning to direct the sequels , i wish nothing but success to whosoever takes on the exciting challenges of films two and three . '' ' *ent71* ' : what fans hoped for ? the first film in the best - selling book series has been hugely successful , pulling in more than $ 550 million worldwide since it premiered in mid-february , but there have been rumbles that creative clashes were in the offing for the sequel . author *ent341* has a great deal of control in how her books are presented on screen , and she made it clear that she wanted to write the screenplay for the second film , *ent184* reported last month . *ent28* wrote the screenplay for `` *ent71* . '' the story behind mr. *ent289* 's suits the film stars *ent344* as billionaire *ent275* -- a man of certain sexual proclivities -- and *ent407* as his romantic partner , *ent389* .

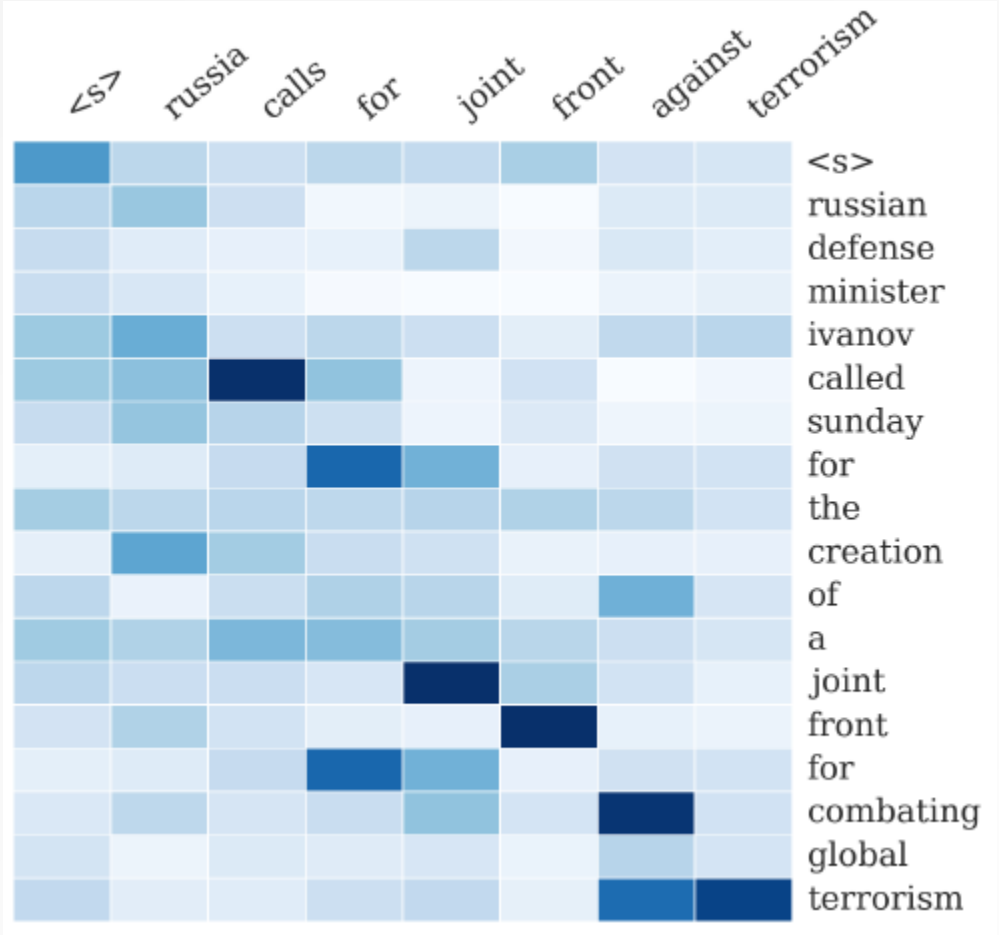
X bows out of the `` *ent321* '' sequel

by *ent339* , *ent42* updated 2:59 pm et , thu march 26 , 2015 (*ent42*) call it `` *ent351* . '' a *ent396* state trooper caught a driver using a cardboard cutout of *ent421* , the *ent364* beer pitchman known as `` *ent397* . '' the driver , who was by himself , was attempting to use the *ent214* . `` the trooper immediately recognized it was a prop and not a passenger , '' trooper *ent367* told the *ent375* . `` as the trooper approached , the driver was actually laughing . '' *ent143* sent out a tweet with a photo of the cutout -- who was clad in what looked like a knit shirt , a far cry from his usual attire -- and the unnamed laughing driver : `` i do n't always violate the *ent303* lane law ... but when i do , i get a $ 124 ticket ! we 'll give him an a for creativity ! '' the driver was caught on *ent300* near *ent327* , *ent396* , just outside *ent53* . `` he could have picked a less recognizable face to put on his prop , '' *ent143* told the *ent375* . `` we see that a lot . usually it 's a sleeping bag . this was very creative . ''

a driver was caught in the **X** with a cutout of `` *ent7* ''

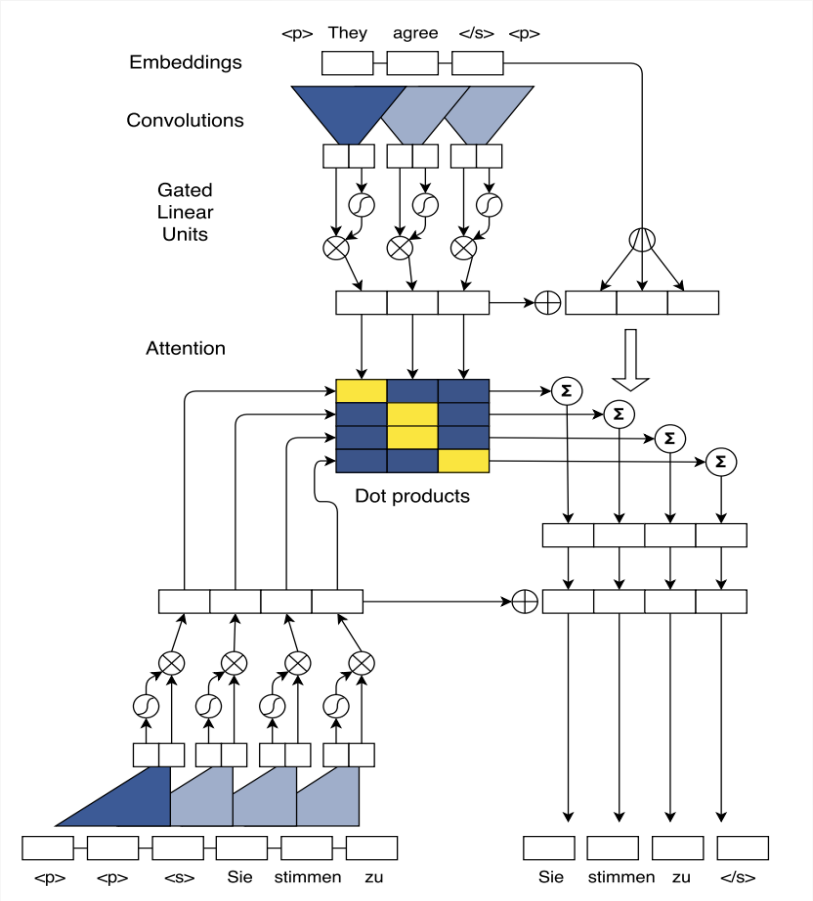
Teaching Machines to Read and Comprehend (NIPS 2015)

自动摘要中的Attention机制

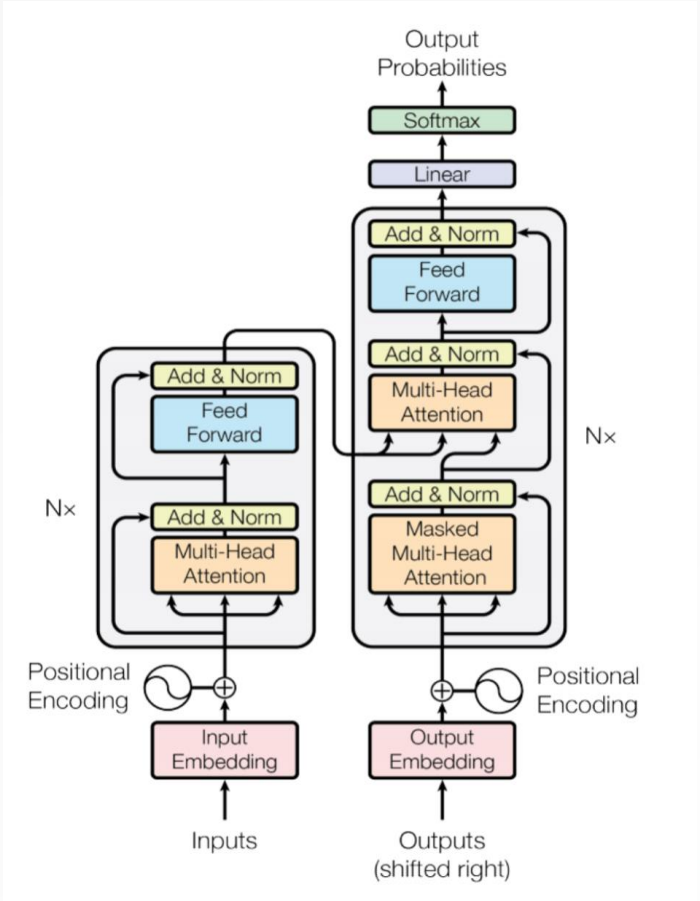


A Neural Attention Model for Abstractive Sentence Summarization (EMNLP 2015)

Beyond RNN

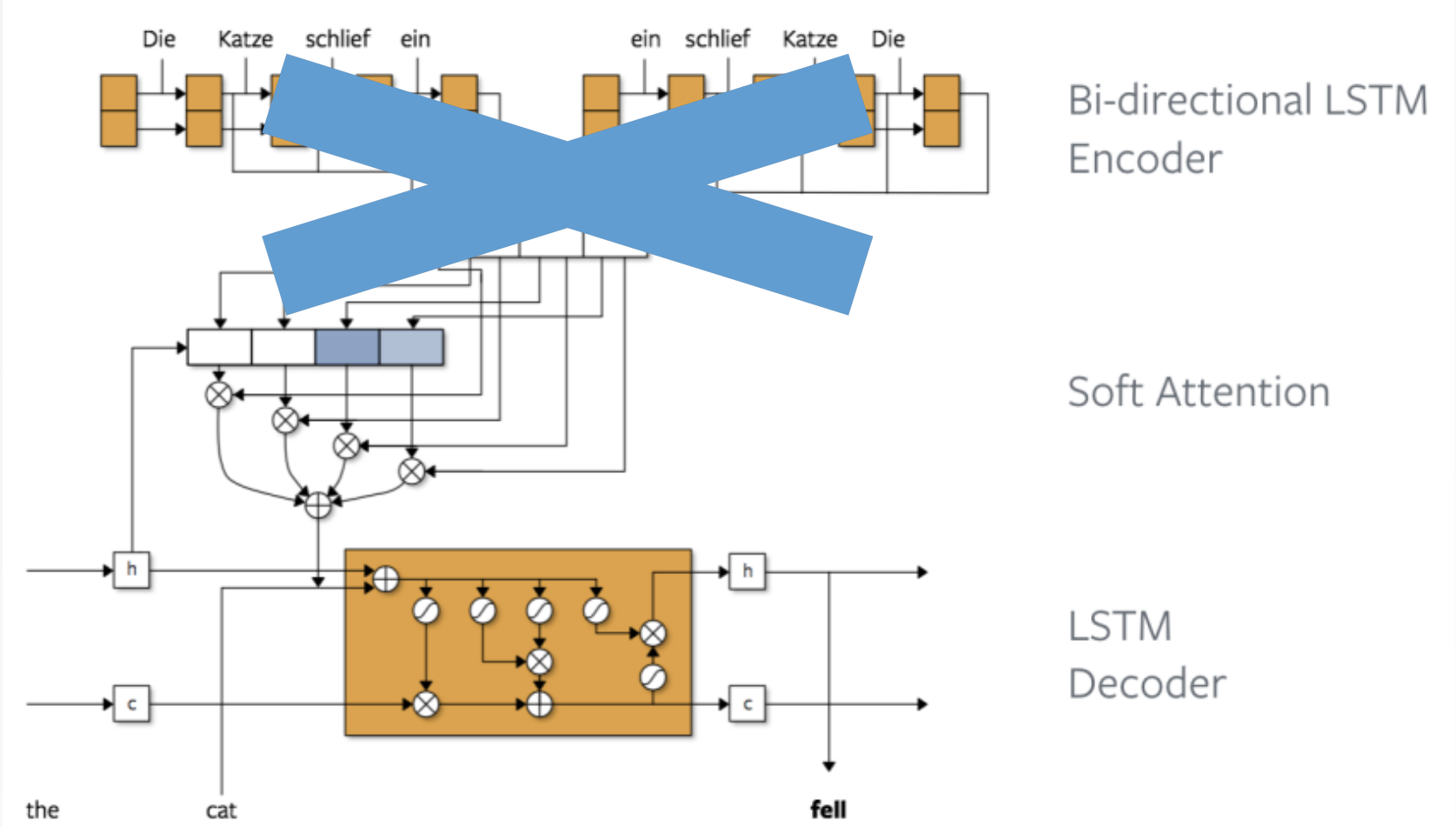


Conv seq2seq, Gehring, et al, 2017



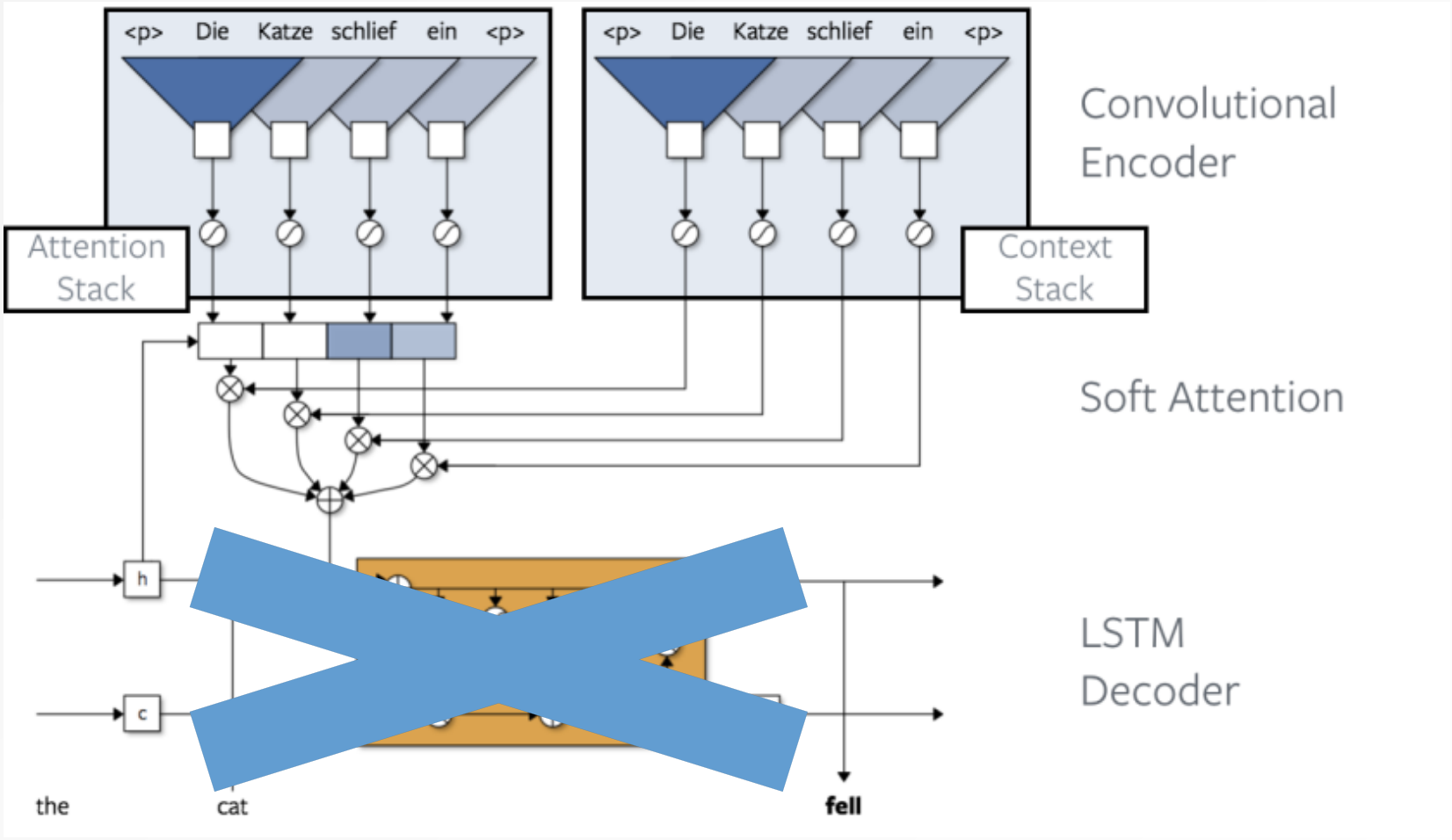
Att is all you need,
Vaswani, et al, 2017

ConvS2S



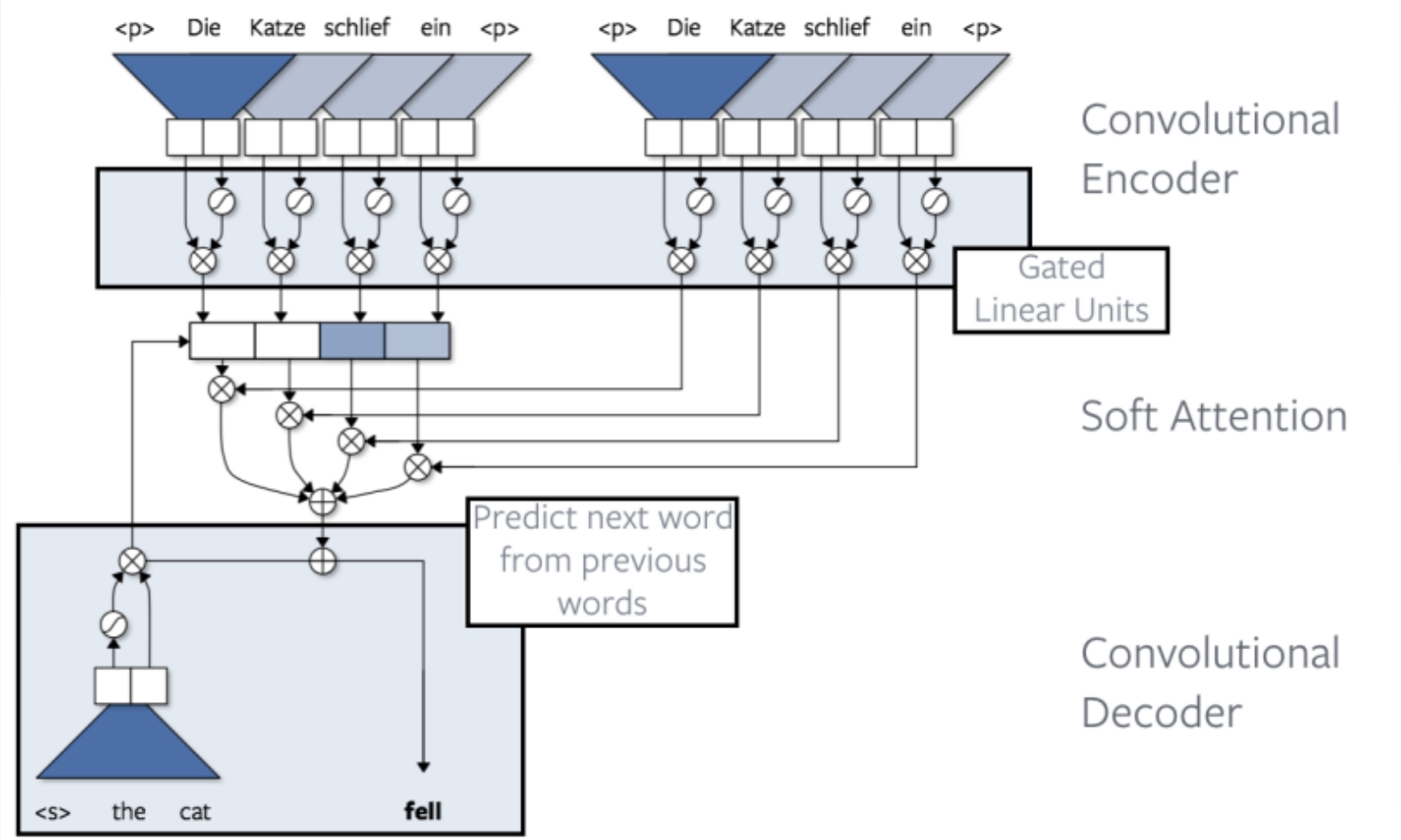
[Denis Yarats, 2017]

ConvS2S



[Denis Yarats, 2017]

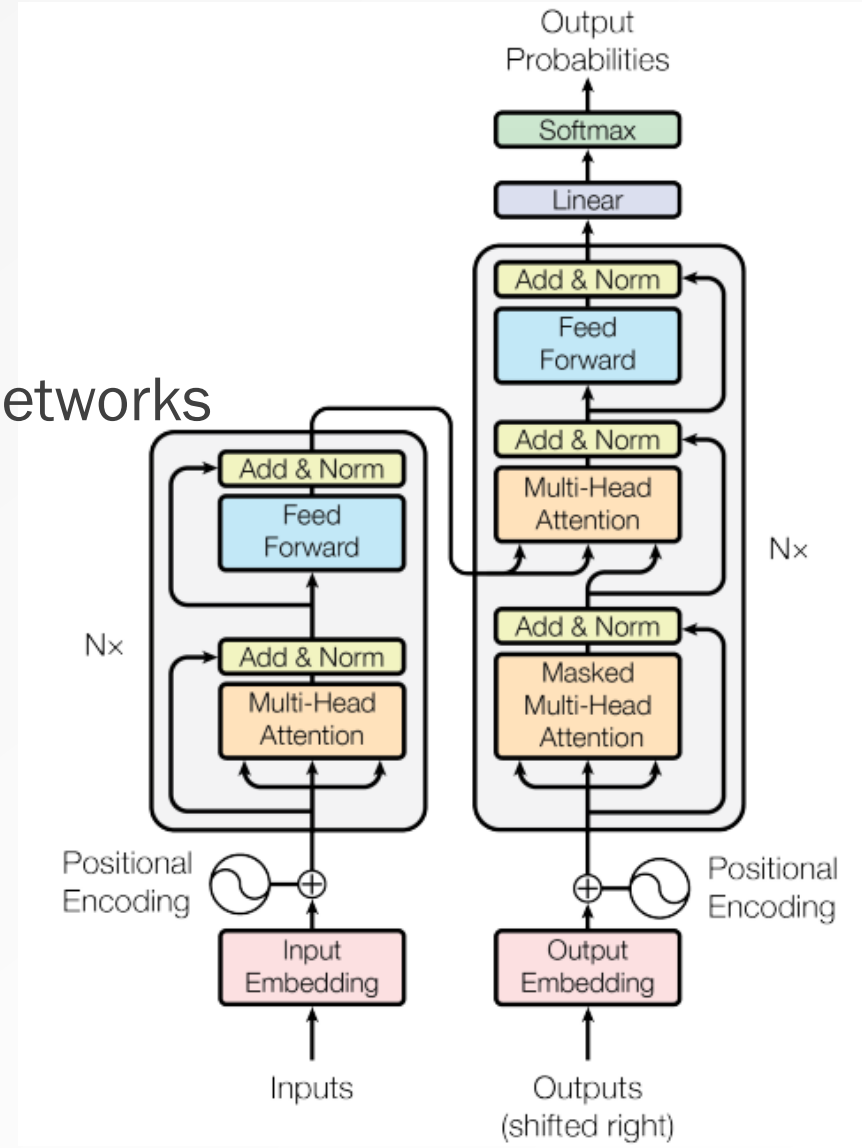
ConvS2S



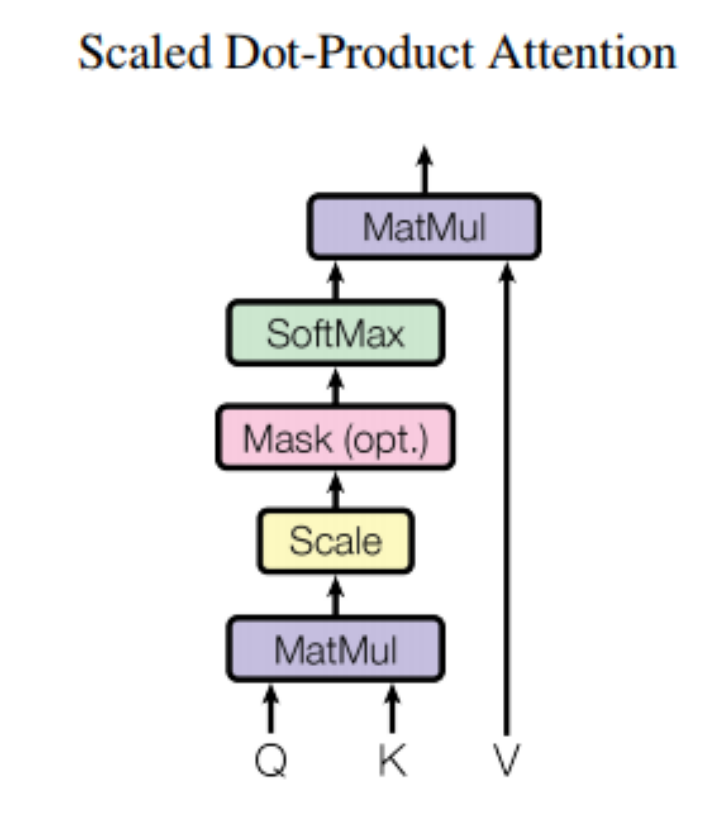
[Denis Yarats, 2017]

Transformer: attention is all your need

- Scaled Dot-Product Attention
- Multi-Head Attention
- Position-wise Feed-Forward Networks
- Positional Encoding



Transformer : Scaled Dot-Product Attention

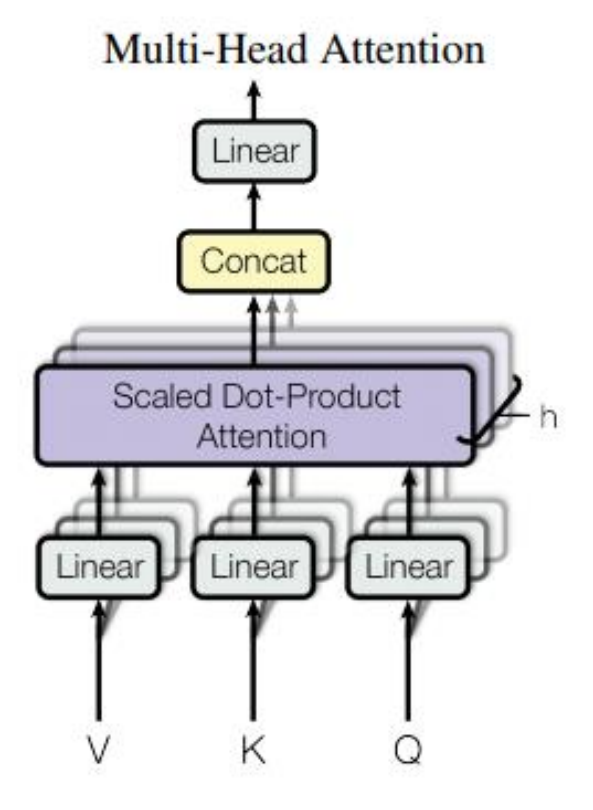


Q: queries, K: keys, V: values

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

```
def attention(query, key, value, mask=None, dropout=None):
    "Compute 'Scaled Dot Product Attention'"
    d_k = query.size(-1)
    scores = torch.matmul(query, key.transpose(-2, -1)) \
             / math.sqrt(d_k)
    if mask is not None:
        scores = scores.masked_fill(mask == 0, -1e9)
    p_attn = F.softmax(scores, dim = -1)
    if dropout is not None:
        p_attn = dropout(p_attn)
    return torch.matmul(p_attn, value), p_attn
```

Transformer : Multi-Head Attention

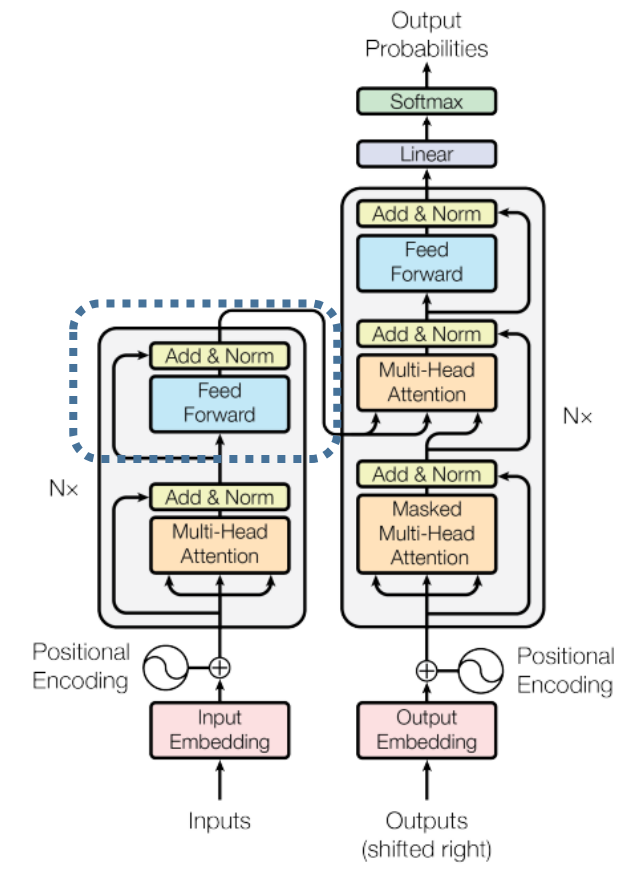


$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_\text{h})W^O$$
$$\text{where head}_\text{i} = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

Where the projections are parameter matrices $W_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$ and $W^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$.

```
class MultiHeadedAttention(nn.Module):
    def __init__(self, h, d_model, dropout=0.1):
        "Take in model size and number of heads."
        super(MultiHeadedAttention, self).__init__()
        assert d_model % h == 0
        # We assume d_v always equals d_k
        self.d_k = d_model // h
        self.h = h
        self.linears = clones(nn.Linear(d_model, d_model), 4)
        self.attn = None
        self.dropout = nn.Dropout(p=dropout)

    def forward(self, query, key, value, mask=None):
        "Implements Figure 2"
        if mask is not None:
            # Same mask applied to all h heads.
            mask = mask.unsqueeze(1)
        nbatches = query.size(0)

        # 1) Do all the linear projections in batch from d_model => h x d_k
        query, key, value = \
            [l(x).view(nbatches, -1, self.h, self.d_k).transpose(1, 2)
             for l, x in zip(self.linears, (query, key, value))]

        # 2) Apply attention on all the projected vectors in batch.
        x, self.attn = attention(query, key, value, mask=mask,
                                 dropout=self.dropout)

        # 3) "Concat" using a view and apply a final linear.
        x = x.transpose(1, 2).contiguous() \
             .view(nbatches, -1, self.h * self.d_k)
        return self.linears[-1](x)
```

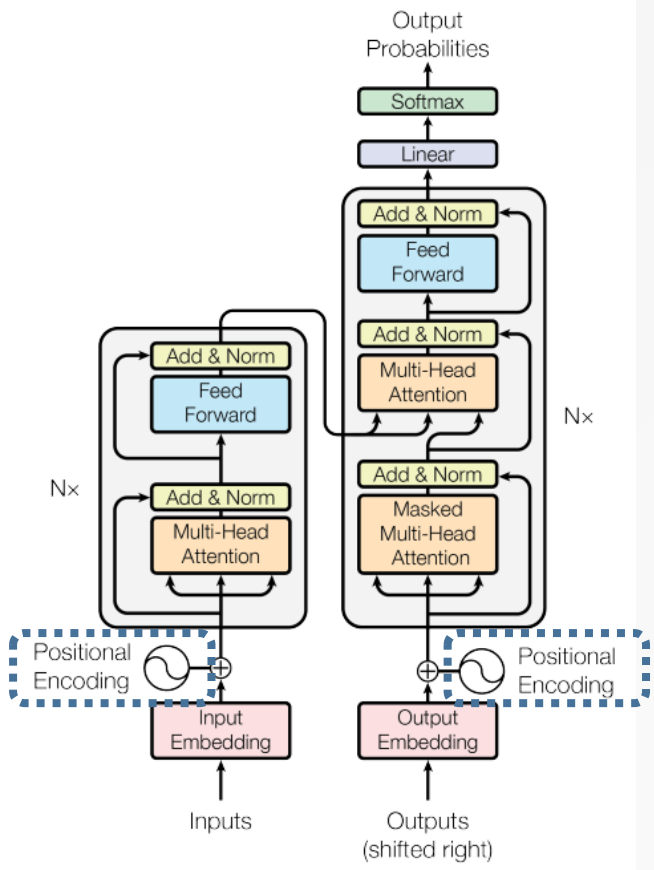
Transformer : Position-wise Feed-Forward Networks



$$\mathrm{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

```
class PositionwiseFeedForward(nn.Module):
    "Implements FFN equation."
    def __init__(self, d_model, d_ff, dropout=0.1):
        super(PositionwiseFeedForward, self).__init__()
        self.w_1 = nn.Linear(d_model, d_ff)
        self.w_2 = nn.Linear(d_ff, d_model)
        self.dropout = nn.Dropout(dropout)

    def forward(self, x):
        return self.w_2(self.dropout(F.relu(self.w_1(x))))
```

Transformer : Positional Encoding



动机: 没有RNN建模序列位置信息

两种类型:

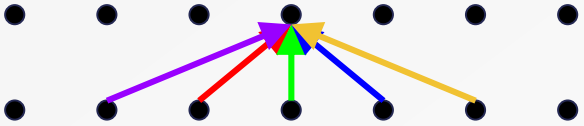
- 位置embeddings (arXiv:1705.03122v2)
- Sinusoid:

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

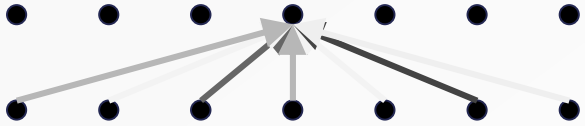
```
class PositionalEncoding(nn.Module):
    "Implement the PE function."
    def __init__(self, d_model, dropout, max_len=5000):
        super(PositionalEncoding, self).__init__()
        self.dropout = nn.Dropout(p=dropout)

        # Compute the positional encodings once in log space.
        pe = torch.zeros(max_len, d_model)
        position = torch.arange(0, max_len).unsqueeze(1)
        div_term = torch.exp(torch.arange(0, d_model, 2) *
                             -(math.log(10000.0) / d_model))
        pe[:, 0::2] = torch.sin(position * div_term)
        pe[:, 1::2] = torch.cos(position * div_term)
        pe = pe.unsqueeze(0)
        self.register_buffer('pe', pe)

    def forward(self, x):
        x = x + Variable(self.pe[:, :x.size(1)],
                         requires_grad=False)
        return self.dropout(x)
```

Self-Attention

Convolution



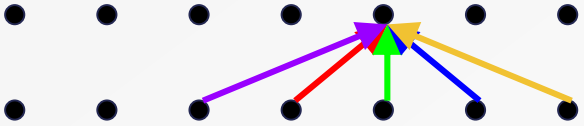
Self-Attention



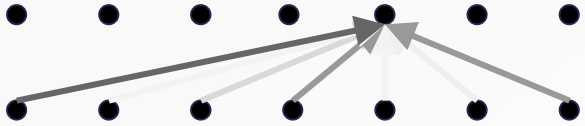
[ICML 2017, Tutorial]

Self-Attention

Convolution

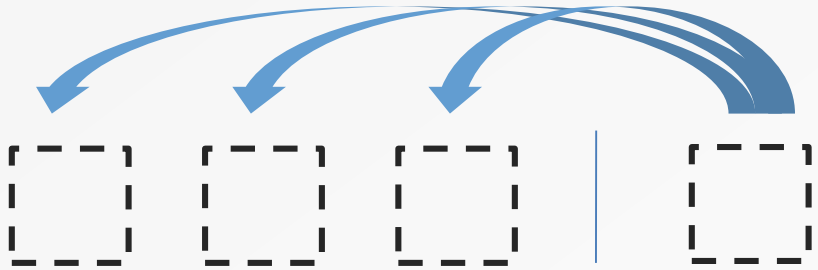


Self-Attention

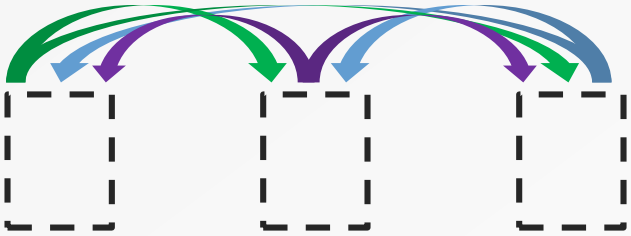


[ICML 2017, Tutorial]

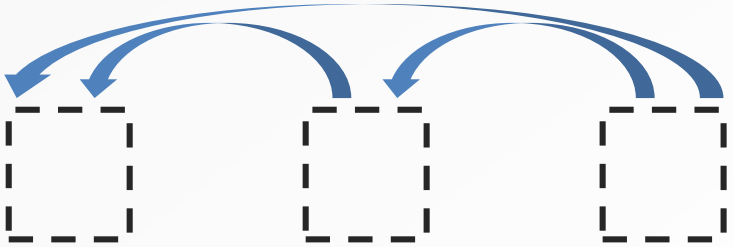
Three ways of attention



Encoder-Decoder Attention



Encoder Self-Attention



MaskedDecoder Self-Attention

[ICML 2017, Tutorial]

Machine Translation Results: WMT-14

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [17]	23.75			
Deep-Att + PosUnk [37]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [36]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [9]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [31]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [37]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [36]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [9]	26.36	**41.29**	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	**$3.3 \cdot 10^{18}$**	
Transformer (big)	**28.4**	**41.0**	$2.3 \cdot 10^{19}$	

语义组合模型

- 组合大语义单元的语义表示（组合语义原则）
  - 图像由局部子图像构成
  - 句子由词构成
- 卷积神经网络
  - 处理空间数据
  - 在局部空间上共享参数
- 循环神经网络
  - 处理序列数据
  - 在时间轴上共享参数
- Transformer
  - 处理更长距离依赖
  - 重量级模型

目录

- 机器学习基础理论与概念
- 神经网络与深度学习基础
- 自然语言处理中的深度学习
- 语义组合模型
  - 卷积神经网络
  - 循环神经网络
  - Transformer和Seq2Seq Learning
- 词表示模型
  - 神经语言模型
  - 词向量 2.0 (word2vec)
  - 词向量 3.0 (ELMo、BERT)
- 总结

语言模型

- 目标：计算一个词串的概率

$$
\begin{aligned}
P(S) &= P(w_1, w_2, w_3, \cdots, w_n) \\
&= P(w_1)P(w_2 \mid w_1)P(w_3 \mid w_1, w_2)\cdots P(w_n \mid w_1, w_2, w_3, \cdots, w_{n-1}) \\
&= \prod_i P(w_i \mid w_1, w_2, w_3, \cdots, w_{i-1})
\end{aligned}
$$

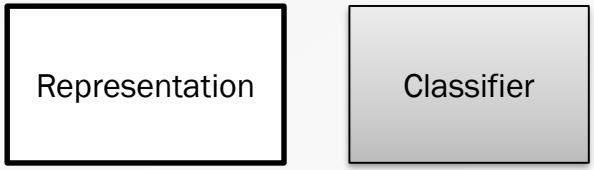
$$
P(w_i \mid w_1, w_2, w_3, \cdots, w_{i-1})
$$

$$
P(w_i \mid w_1, w_2, w_3, \cdots, w_{i-1}) = \frac{Count(w_1, w_2, w_3, \cdots, w_{i-1}, w_i)}{Count(w_1, w_2, w_3, \cdots, w_{i-1})}
$$

S : 他是研究生物的(他 是 研究 生物 的)

p(S) = p(他|<BOS>)×p(是|他)×p(研究|是)×p(生物|研究)×p(的|生物)×p(的|<EOS>)

语言模型



$$p(quick|C) \gg p(fast|C)$$

$$R(fast) \gg R(quick)$$

NNLM

- Neural Network Language Model [Y.Bengio et al. 2003]

$$P(w_1,\ldots,w_N)=\prod_t P(w_t \mid w_{t-1},\ldots,w_{t-n+1})$$

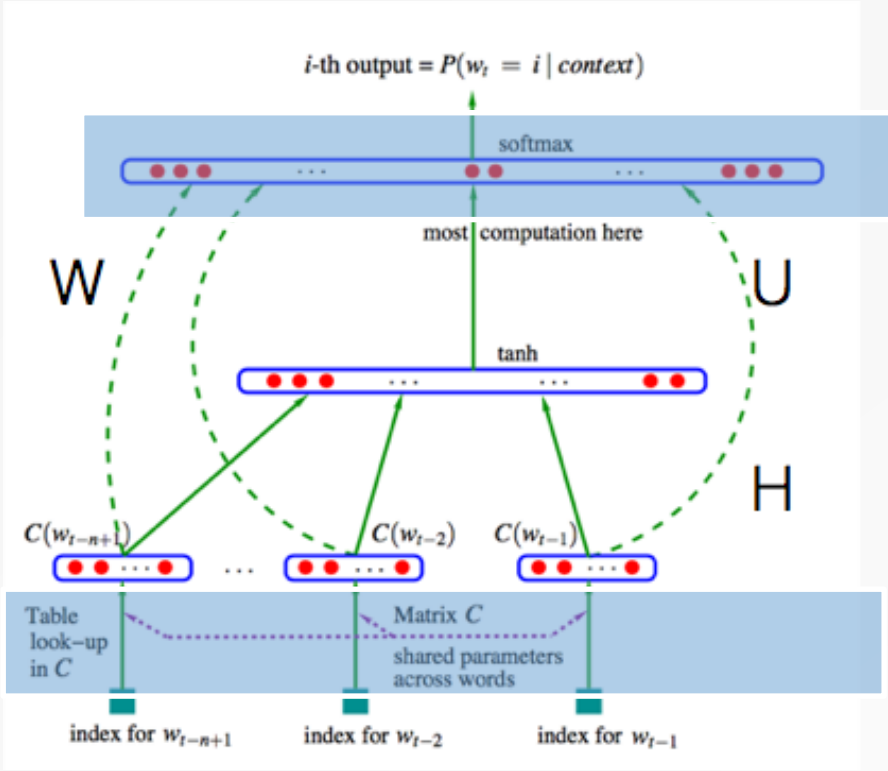
$$f(w_t,w_{t-1},\cdots,w_{t-n+1})=\hat{P}(w_t \mid w_{t-1},\cdots,w_{t-n+1})$$

$$f(w_t,w_{t-1}\ldots,w_{t-n+1})=g(w_t,C(w_{t-1})\ldots,C(w_{t-n+1}))$$

$$L=\frac{1}{T}\sum_t \log f(w_t,w_{t-1},\cdots,w_{t-n+1};\theta)+R(\theta)$$

$$\hat{P}(w_t|w_{t-1},\cdots w_{t-n+1})=\frac{e^{y_{w_t}}}{\sum_i e^{y_i}}$$

NNLM

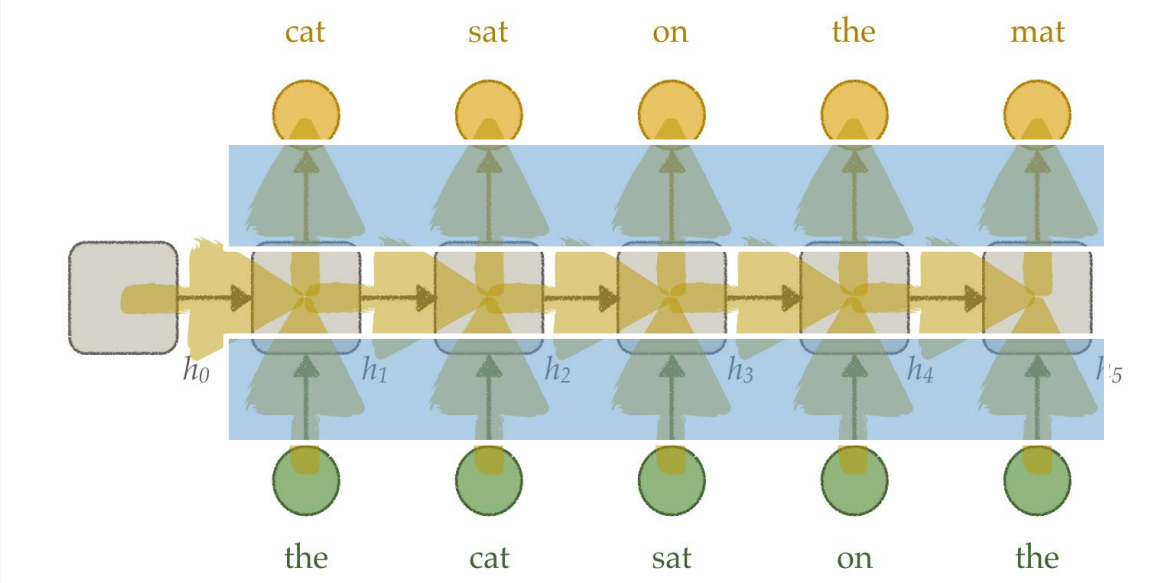


$$y = b + Wx + U\tanh(d + Hx)$$

$$x = (C(w_{t-1}), C(w_{t-2}), \cdots, C(w_{t-n+1}))$$

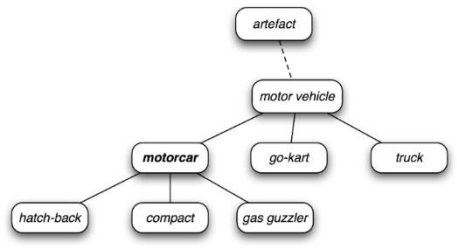
$$\theta \leftarrow \theta + \varepsilon \frac{\partial \log \hat{P}(w_t | w_{t-1}, \cdots w_{t-n+1})}{\partial \theta}$$

RNNLM

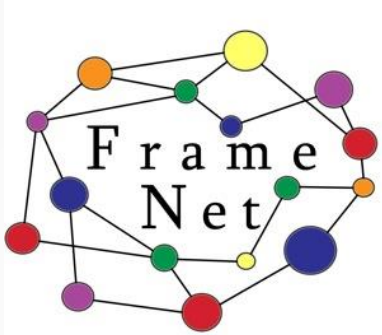


词表示

- 词是语言处理中最基本的语言单元
- 词以及词间关系的表示和建模是NLP任务中重要的基础工作



WordNet [Miller 1995]

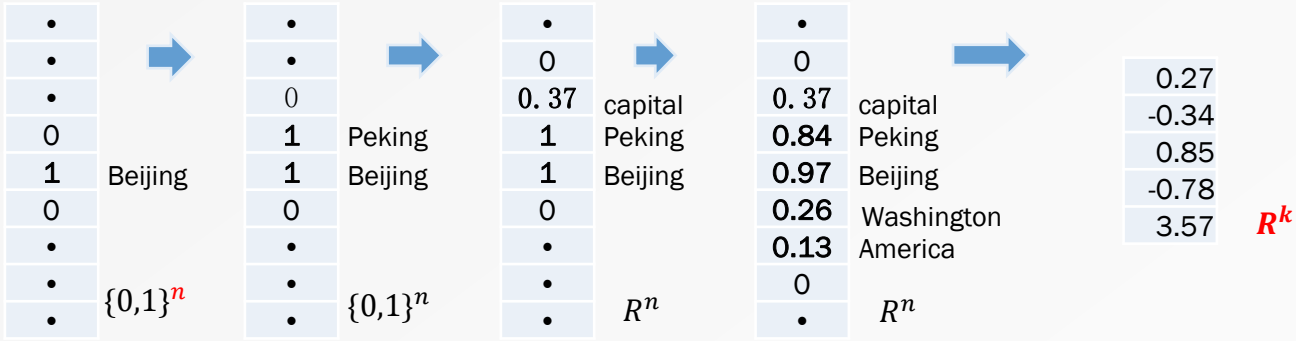


FrameNet [Baker 1998]

表示词之间多维度关系

词表示

- 词是语言处理中最方便处理的语言单元



- 目标：语义相似的词表示相近
  - 如何判定语义相似?
  - 如何数值化表示词?
  - 如何刻画数值表示的相近?

The first thing you do with a word symbol is you convert it to a word vector.

[Geoffrey E. Hinton. 2015]


Science | DOI:10.1145/2874915 Gregory Goth

Deep or Shallow, NLP Is Breaking Out

Neural net advances improve computers' language ability in many fields.

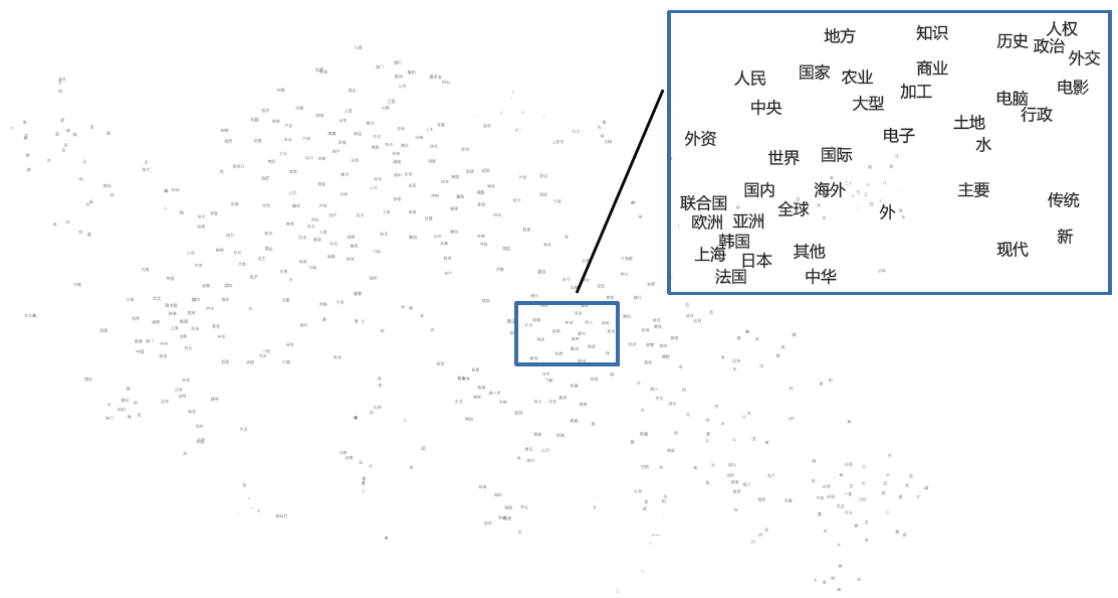
词表示学习

- 分布式假设：上下文相似的词 → 词义相似 [Harris 1954]
  - 语义可以统计获得
  - 相似度可以度量

江西省地处中国东南部
携程带你玩转江西省

四川省地处中国西南腹地
携程带你玩转四川省

向量 & 内积/余弦



词表示学习

- 上下文的表示：文档、词、n元词组

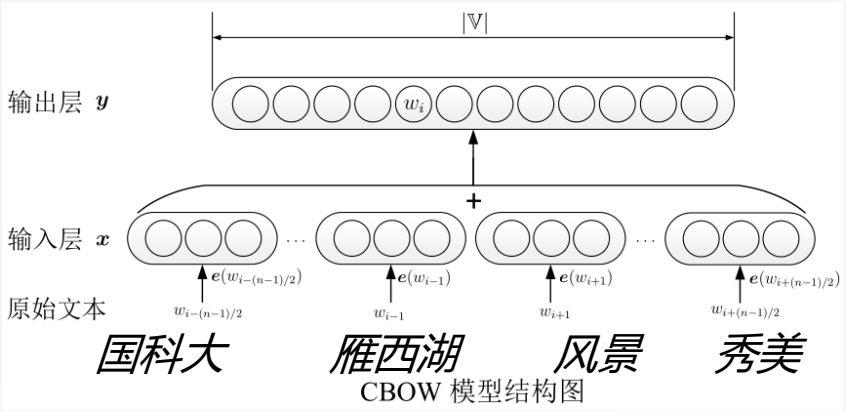
- 相似度的衡量：向量的内积（余弦）

词表示方法

- 基于预测的方式
  - 给定上下文对目标词进行预测
- 基于计数的方式：
  - 统计词-上下文共现情况，对共现矩阵进行分解

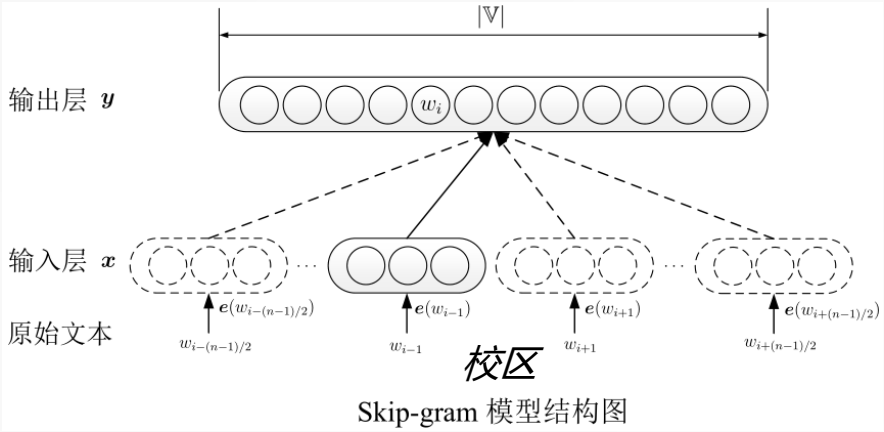
基于预测的方法：word2vec

校区



CBOW 模型结构图

国科大 雁西湖 风景 秀美



Skip-gram 模型结构图

国科大 雁西湖 校区 风景 秀美

$$\boldsymbol{x} = \frac{1}{n-1}\sum_{w_j \in c} \boldsymbol{e}(w_j)$$

$$\sum_{(w,c)\in\mathbb{D}} \sum_{w_j \in c} \log P(w_j|w)$$

$$P(w|c) = \frac{\exp\left(\boldsymbol{e}'(w)^{\mathrm{T}}\boldsymbol{x}\right)}{\sum_{w'\in\mathbb{V}} \exp\left(\boldsymbol{e}'(w')^{\mathrm{T}}\boldsymbol{x}\right)}$$

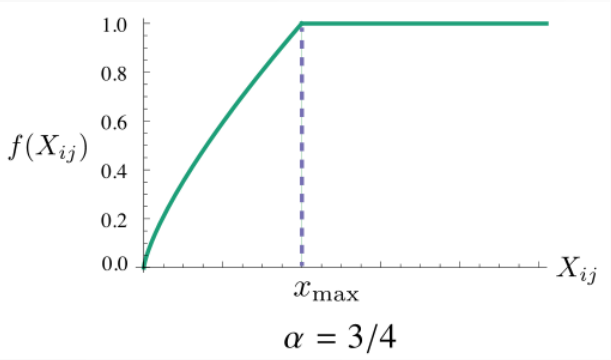
$$P(w_j|w) = \frac{\exp\left(\boldsymbol{e}'(w_j)^{\mathrm{T}}\boldsymbol{e}(w)\right)}{\sum_{w'\in\mathbb{V}} \exp\left(\boldsymbol{e}'(w')^{\mathrm{T}}\boldsymbol{e}(w)\right)}$$

基于计数的方法：GloVe

- 基于潜在因子分解模型（Latent Factor Model）
  - 矩阵ij项的值为词i和词j共现次数x_{ij}的对数

$$\sum_{i,j\in\mathbb{V},x_{ij}\neq 0} f\left(x_{ij}\right)\left(\log(x_{ij})-\boldsymbol{p}_i^{\mathrm{T}}\boldsymbol{q}_j+\boldsymbol{b}_i^{(1)}+\boldsymbol{b}_j^{(2)}\right)^2$$

$$f(x)=\begin{cases}(x/x_{max})^{\alpha} & \text{如果 } x<x_{max}\\ 1 & \text{其它情况}\end{cases}$$



$x_{\max} = 100$

模型改进

- 上下文与目标词关系建模
  - Log双线性模型

$$\boldsymbol{y}(w_i) = \boldsymbol{b}^{(2)} + \boldsymbol{e}(w_i)^{\mathrm{T}}\boldsymbol{b}^{(1)} + \boldsymbol{e}(w_i)^{\mathrm{T}} H \left[\boldsymbol{e}(w_{i-(n-1)}); \ldots; \boldsymbol{e}(w_{i-1})\right]$$

  - CBOW

$$\boldsymbol{y}(w_i) = \boldsymbol{e}'(w_i)^T \sum_{j=i-k \ldots i-1; j=i+1, i+k} \boldsymbol{e}(wj)$$

  - ...

- 处理低频词
  - Swivel

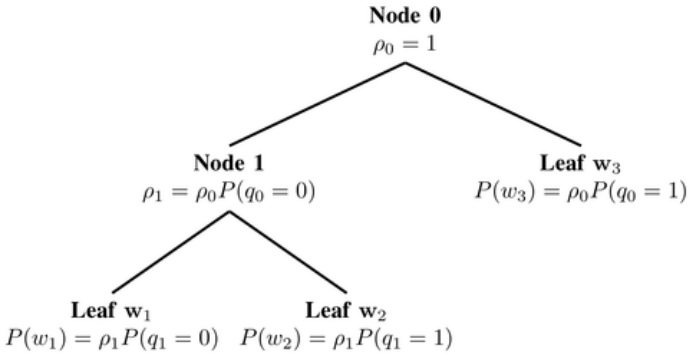
[Shazeer 2016]

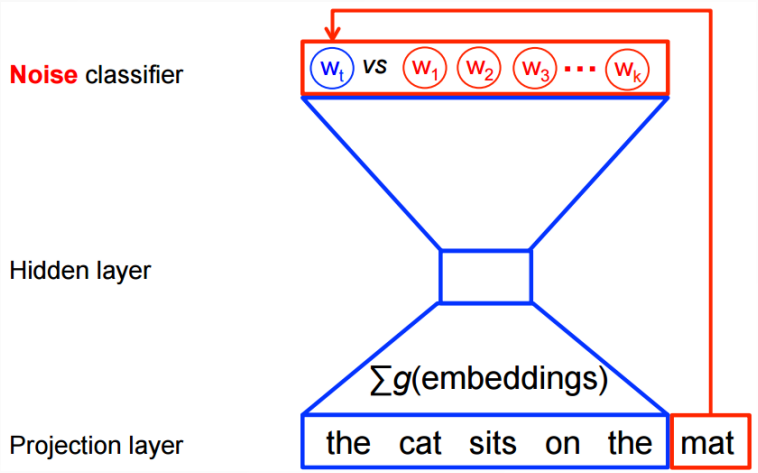
Count	Cost
$x_{ij} > 0$	$\frac{1}{2} f(x_{ij})(w_i^\top \tilde{w}_j - \mathbf{pmi}(i;j))^2$
$x_{ij} = 0$	$\log\left[1 + \exp\left(w_i^\top \tilde{w}_j - \mathbf{pmi}^*(i;j)\right)\right]$

技术改进

- Softmax层的计算加速
  - Softmax-based
    - Hierarchical
    - Differentiated
    - CNN-Softmax
    - ...
  - Sampling-based
    - Importance Sampling
    - Noise-Contrastive Estimation
    - Negative Sampling
    - ...

$$p(w|h) = \frac{\exp(hTw)}{\sum_{w_i \in V} \exp(h^T w_i)}$$





词向量2.0的若干（实验）结论

- word2vec：计算换存储；GloVe：存储换计算
- 没有最好，只有适合
  - 适合任务，用（任务的）领域内语料训练
- 确定合适领域的语料之后，语料越大越好
- 大语料，使用简单模型（CBOW）
- 小语料，使用复杂模型（Skip-gram）
- 使用任务相关的开发集，而非词向量的开发集
- 词的主向量/副向量 （主-副表示组合关系，主-主/副-副表示聚合关系）

词向量3.0的突破

机器之心 翻译
2018/10/12 16:26

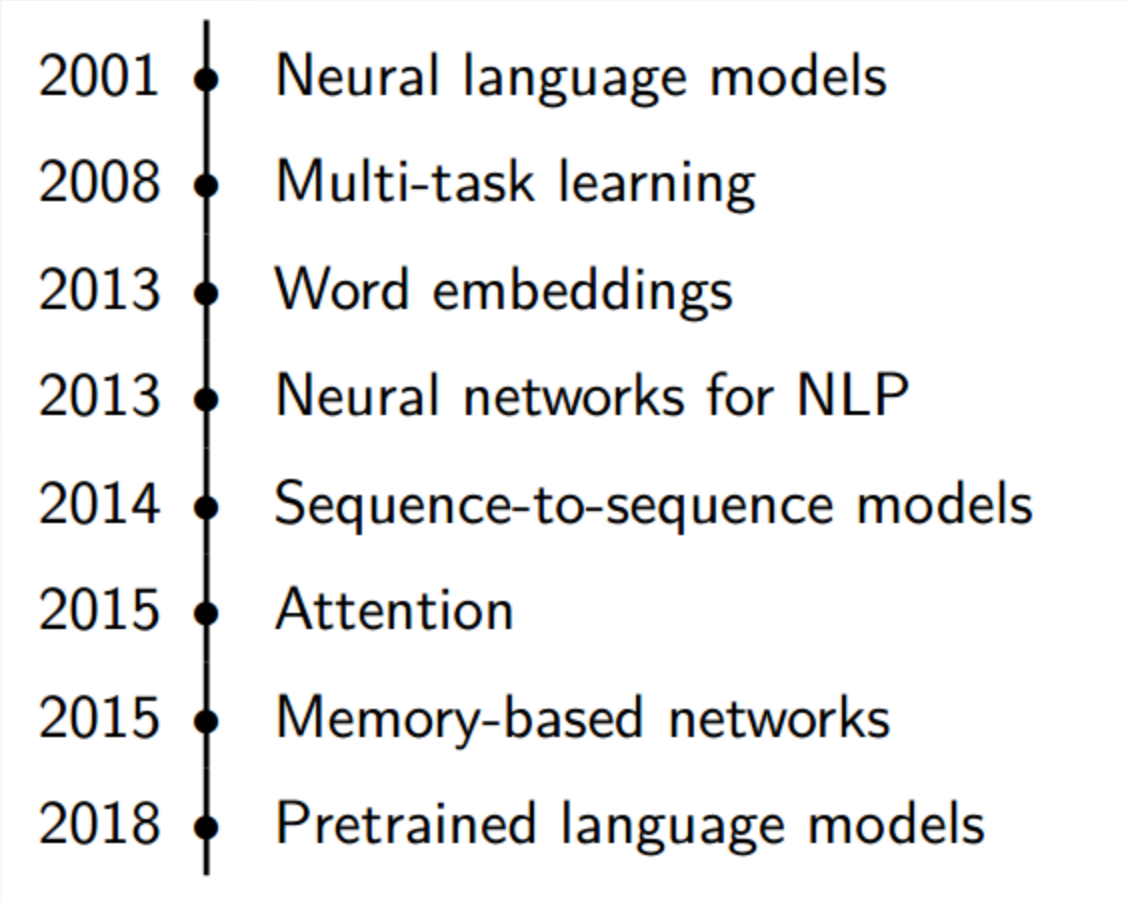
Jacob Devlin等 作者
路雪 王淑婷 张倩 编译

最强NLP预训练模型！谷歌BERT横扫11项NLP任务记录

> 本文介绍了一种新的语言表征模型 BERT——来自 Transformer 的双向编码器表征。与最近的语言表征模型不同，BERT 旨在基于所有层的左、右语境来预训练深度双向表征。BERT 是首个在大批句子层面和 token 层面任务中取得当前最优性能的基于微调的表征模型，其性能超越许多使用任务特定架构的系统，刷新了 11 项 NLP 任务的当前最优性能记录。

A Review of the Recent History of NLP

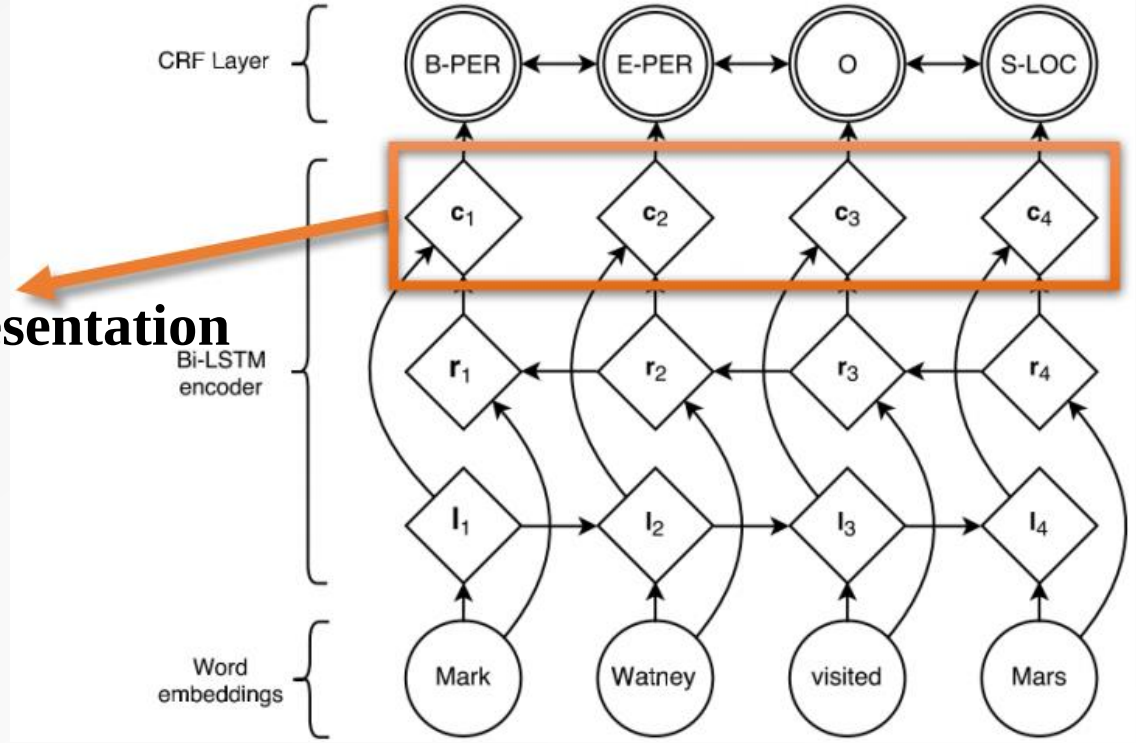
by Sebastian Ruder (Deep Learning Indaba 2018)



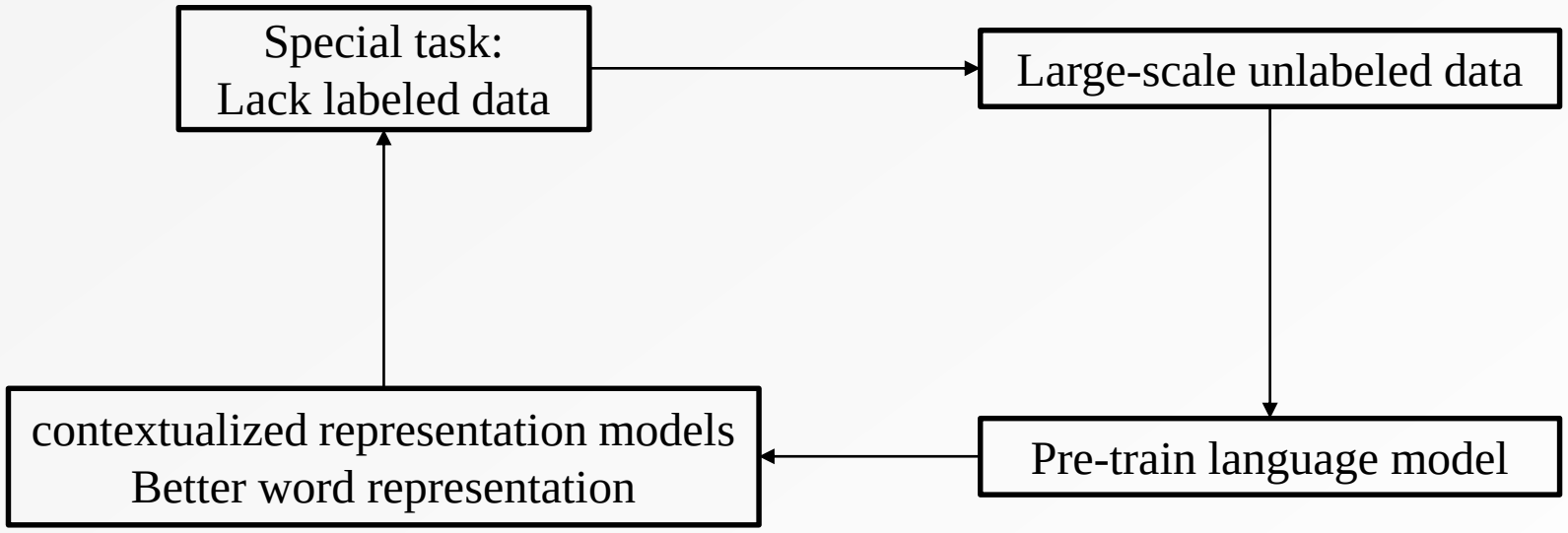
上下文相关的词表示

- 当前词向量为静态表示，与上下文无关

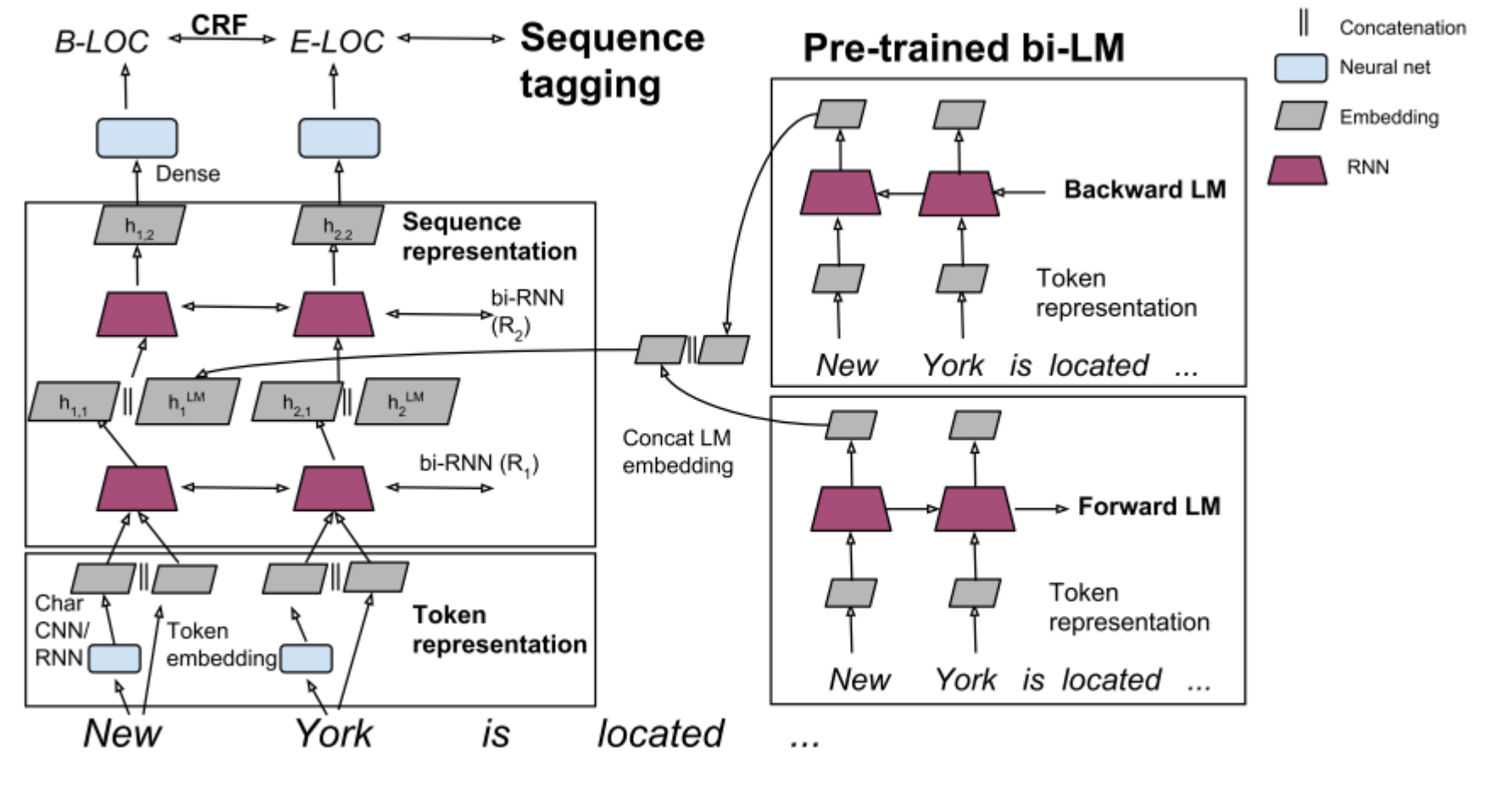
Contextualized Word Representation



预训练语言模型


Special task:
Lack labeled data
Large-scale unlabeled data
contextualized representation models
Better word representation
Pre-train language model

如何利用语言模型


B-LOC
CRF
E-LOC
Sequence tagging
Pre-trained bi-LM
Concatenation
Neural net
Embedding
RNN
Dense
Sequence representation
bi-RNN (R_2)
Backward LM
Token representation
New York is located ...
Concat LM embedding
bi-RNN (R_1)
Forward LM
Char CNN/ RNN
Token embedding
Token representation
New York is located ...

ELMo: Embeddings from Language Models

- 对于第k个字符，L层双向语言模型得到2L+1个词表示

$$\begin{aligned} R_k &= \{\mathbf{x}_k^{LM}, \overrightarrow{\mathbf{h}}_{k,j}^{LM}, \overleftarrow{\mathbf{h}}_{k,j}^{LM} \mid j = 1, \ldots, L\} \\ &= \{\mathbf{h}_{k,j}^{LM} \mid j = 0, \ldots, L\}, \end{aligned}$$

- 对于后续任务，ELMo学习以不同权重组合这些表示

$$\mathbf{ELMo}_k^{task} = E(R_k; \Theta^{task}) = \gamma^{task} \sum_{j=0}^{L} s_j^{task} \mathbf{h}_{k,j}^{LM}$$

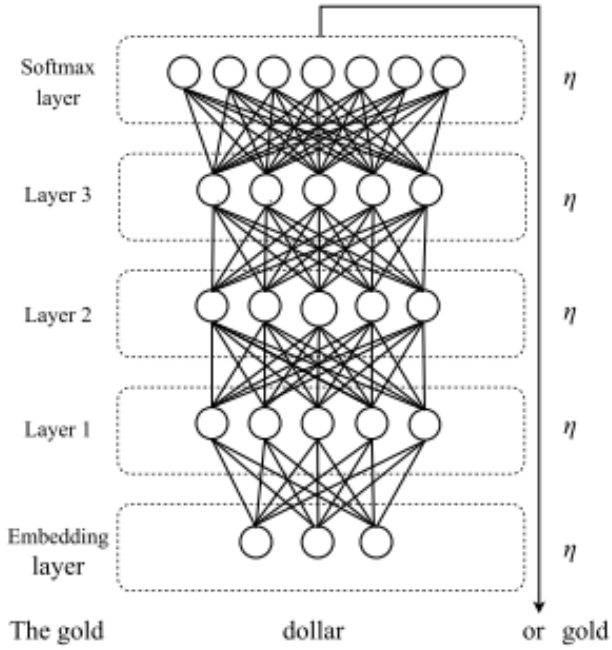
- 使用biLMs加强NLP监督任务的输入表示

$$[\mathbf{x}_k; \mathbf{ELMo}_k^{task}]$$

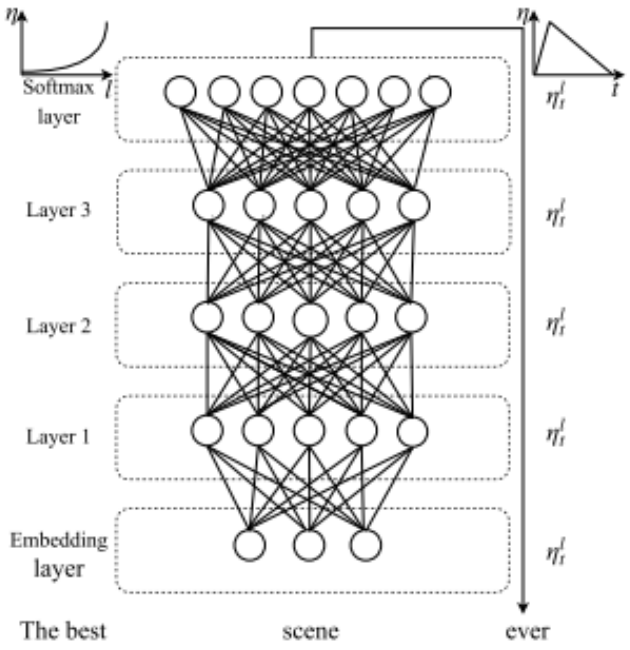
实验结果

Task	Previous SOTA		Our baseline	ELMo + baseline	Increase (absolute/ relative)
SQuAD	Liu et al. (2017)	84.4	81.1	85.8	4.7 / 24.9%
SNLI	Chen et al. (2017)	88.6	88.0	88.7 ± 0.17	0.7 / 5.8%
SRL	He et al. (2017)	81.7	81.4	84.6	3.2 / 17.2%
Coref	Lee et al. (2017)	67.2	67.2	70.4	3.2 / 9.8%
NER	Peters et al. (2017)	91.93 ± 0.19	90.15	92.22 ± 0.10	2.06 / 21%
SST-5	McCann et al. (2017)	53.7	51.4	54.7 ± 0.5	3.3 / 6.8%

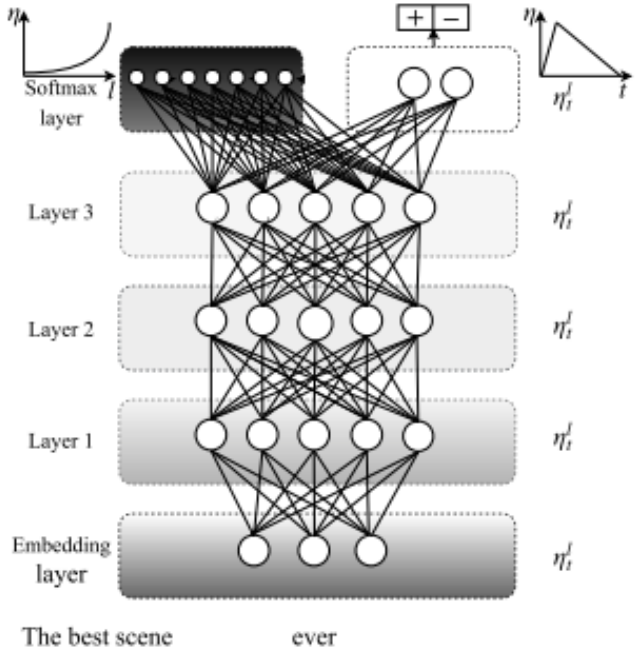
ULMFiT: Universal Language Model Fine-Tuning



(a) LM pre-training



(b) LM fine-tuning



(c) Classifier fine-tuning

GPT: Generative Pre-Training

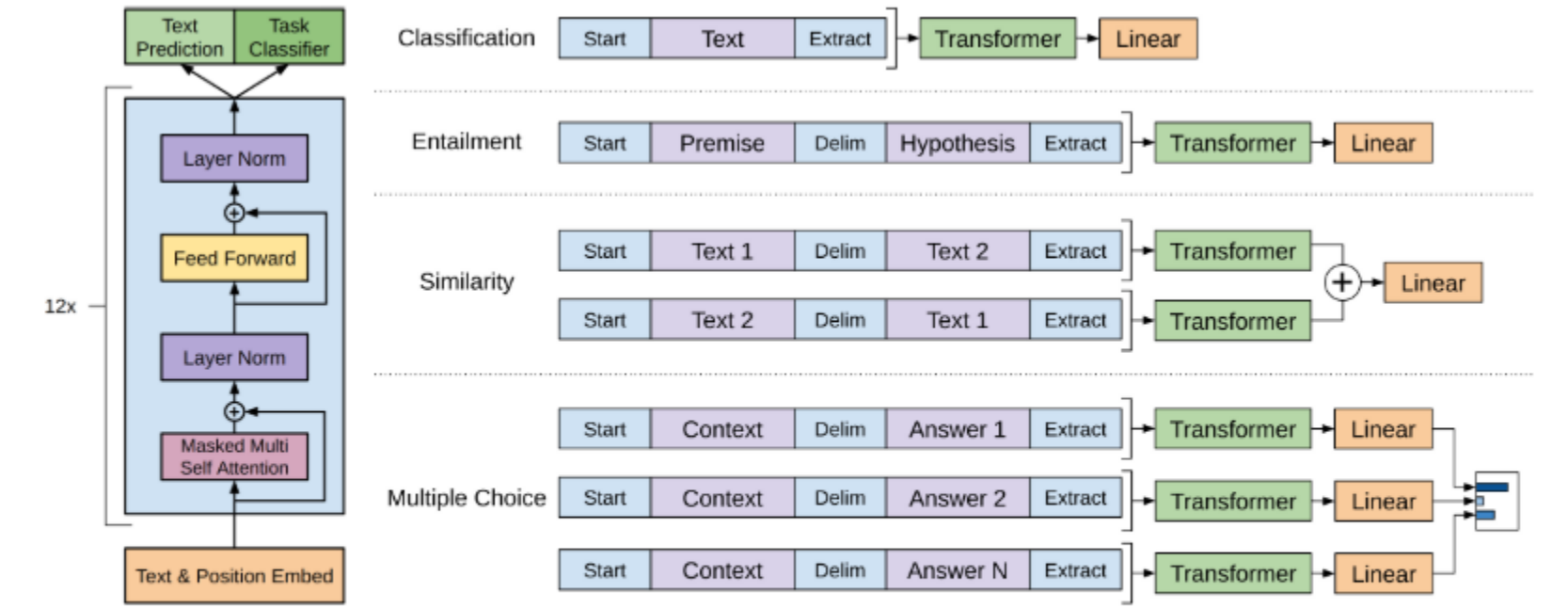
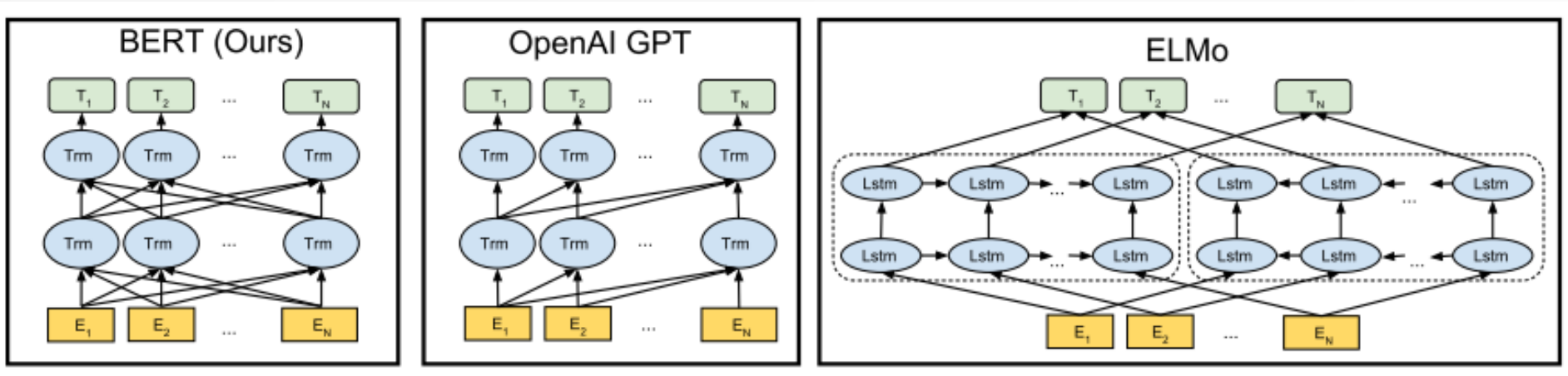


Figure 1: **(left)** Transformer architecture and training objectives used in this work. **(right)** Input transformations for fine-tuning on different tasks. We convert all structured inputs into token sequences to be processed by our pre-trained model, followed by a linear+softmax layer.

Bi-direction?

$$L_1(\mathcal{U}) = \sum_i \log P(u_i | u_{i-k}, \ldots, u_{i-1}; \Theta)$$



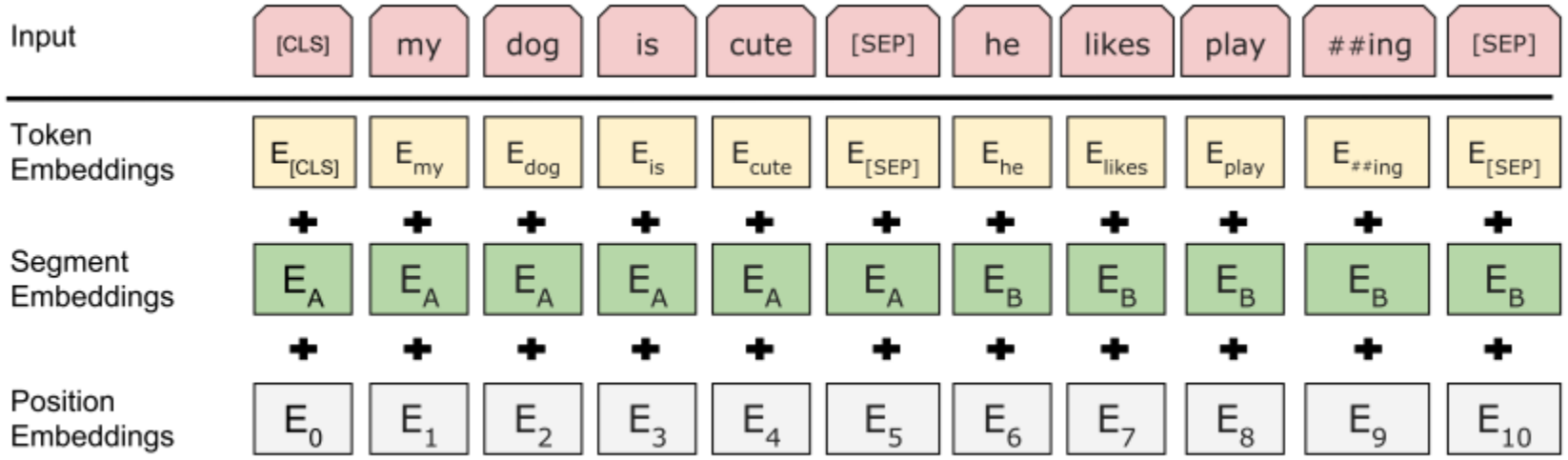
Masked Language Model(MLM)

Cloze task

$$\sum_{k=1}^{N} (\log p(t_k \mid t_1, \ldots, t_{k-1}; \Theta_x, \overrightarrow{\Theta}_{LSTM}, \Theta_s) + \log p(t_k \mid t_{k+1}, \ldots, t_N; \Theta_x, \overleftarrow{\Theta}_{LSTM}, \Theta_s))$$

```
[CLS] the man went to [MASK] store [SEP]
```

BERT的输入表示


Input
[CLS] my dog is cute [SEP] he likes play ##ing [SEP]
Token Embeddings
E[CLS] Emy Edog Eis Ecute E[SEP] Ehe Elikes Eplay E##ing E[SEP]
Segment Embeddings
EA EA EA EA EA EA EB EB EB EB EB
Position Embeddings
E0 E1 E2 E3 E4 E5 E6 E7 E8 E9 E10

预训练任务：Masked Language Model

- [MASK]: 15% in **pre-training**
- [MASK] token is never seen during **fine-tuning**
- Solution
  - 80%: [MASK], e.g. my dog is hairy → my dog is [MASK]
  - 10%: randomly replace, e.g. my dog is hairy → my dog is apple
  - 10%: unchanged, e.g. my dog is hairy → my dog is hairy

预训练任务: Next Sentence Prediction

- 问答和自然语言推理任务
  - 需要理解句子之间的关系

真实pair (50%)

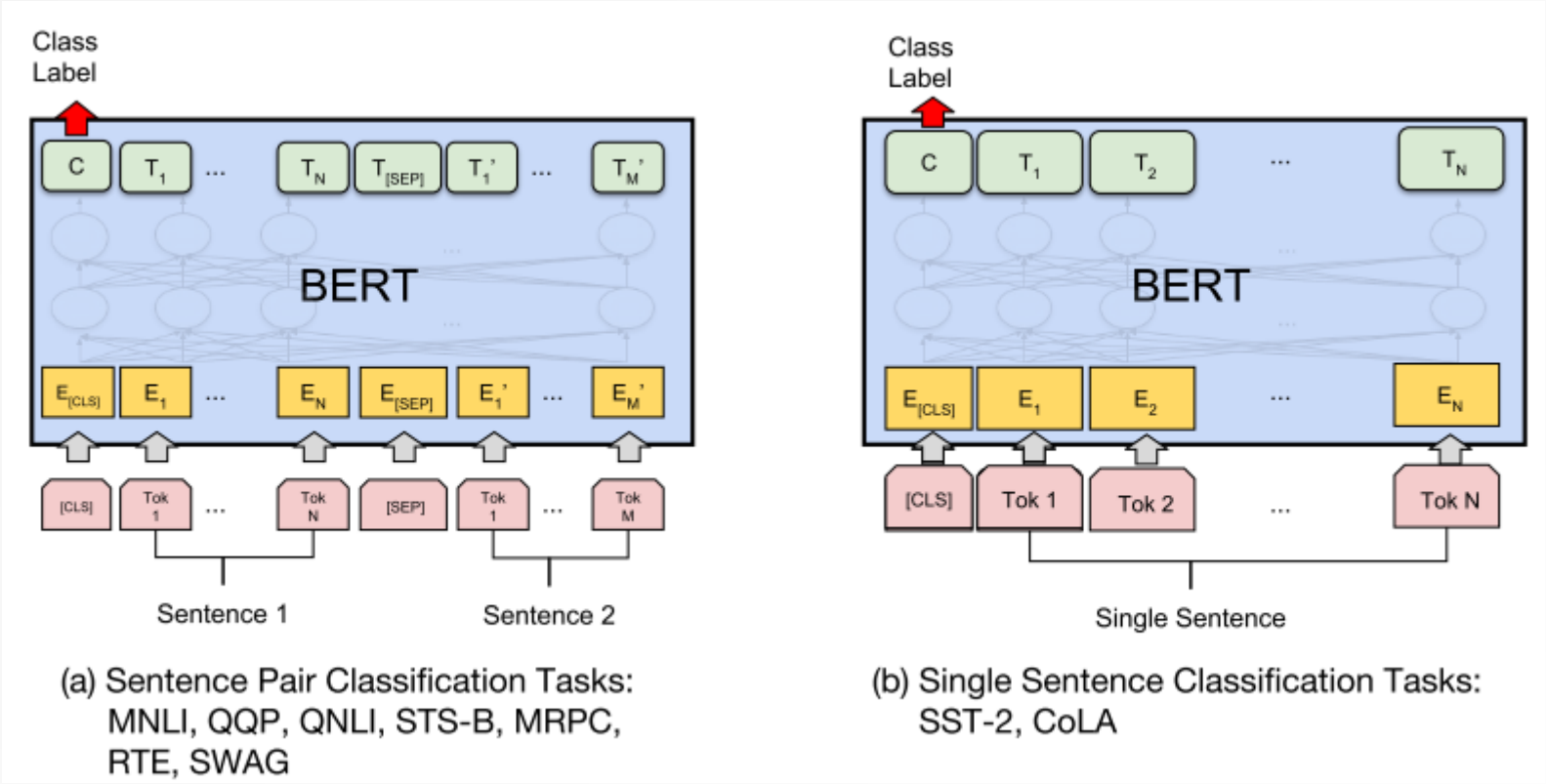
Input = [CLS] the man went to [MASK] store [SEP]
he bought a gallon [MASK] milk [SEP]
Label = IsNext

随机pair (50%)

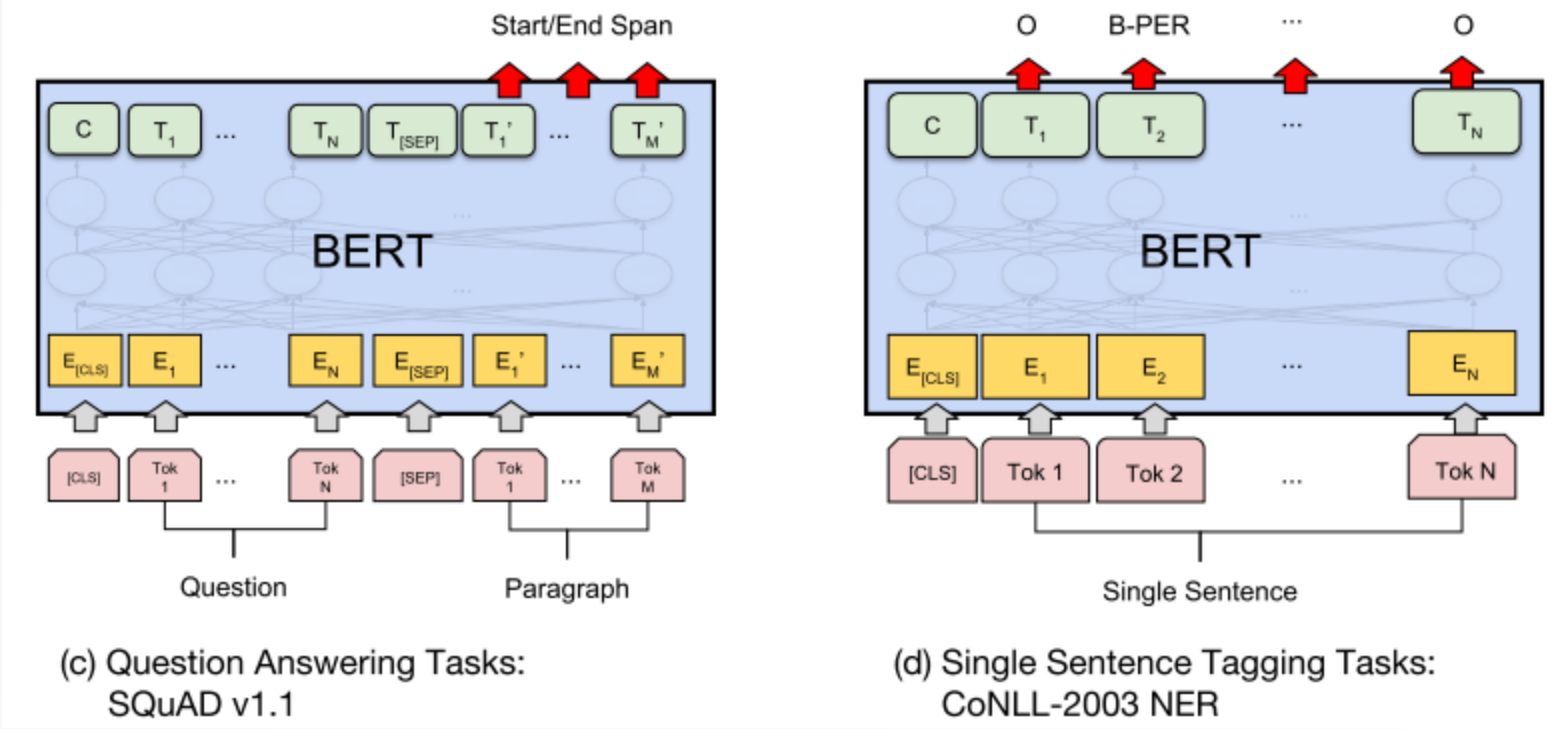
Input = [CLS] the man [MASK] to the store [SEP]
penguin [MASK] are flight ##less birds [SEP]
Label = NotNext

97%-98% accuracy

面向目标任务精调


Class Label
C
T1
...
TN
T[SEP]
T1'
...
TM'
BERT
E[CLS]
E1
...
EN
E[SEP]
E1'
...
EM'
[CLS]
Tok 1
...
Tok N
[SEP]
Tok 1
...
Tok M
Sentence 1
Sentence 2
(a) Sentence Pair Classification Tasks:
MNLI, QQP, QNLI, STS-B, MRPC,
RTE, SWAG
Class Label
C
T1
T2
...
TN
BERT
E[CLS]
E1
E2
...
EN
[CLS]
Tok 1
Tok 2
...
Tok N
Single Sentence
(b) Single Sentence Classification Tasks:
SST-2, CoLA

面向目标任务精调



(c) Question Answering Tasks: SQuAD v1.1

(d) Single Sentence Tagging Tasks: CoNLL-2003 NER

GLUE Test

System	MNLI-(m/mm) 392k	QQP 363k	QNLI 108k	SST-2 67k	CoLA 8.5k	STS-B 5.7k	MRPC 3.5k	RTE 2.5k	**Average** -
Pre-OpenAI SOTA	80.6/80.1	66.1	82.3	93.2	35.0	81.0	86.0	61.7	74.0
BiLSTM+ELMo+Attn	76.4/76.1	64.8	79.9	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	88.1	91.3	45.4	80.0	82.3	56.0	75.2
$\text{BERT}_{\text{BASE}}$	84.6/83.4	71.2	90.1	93.5	52.1	85.8	88.9	66.4	79.6
$\text{BERT}_{\text{LARGE}}$	**86.7/85.9**	**72.1**	**91.1**	**94.9**	**60.5**	**86.5**	**89.3**	**70.1**	**81.9**

Table 1: GLUE Test results, scored by the GLUE evaluation server. The number below each task denotes the number of training examples. The "Average" column is slightly different than the official GLUE score, since we exclude the problematic WNLI set. OpenAI GPT = (L=12, H=768, A=12); $\text{BERT}_{\text{BASE}}$ = (L=12, H=768, A=12); $\text{BERT}_{\text{LARGE}}$ = (L=24, H=1024, A=16). BERT and OpenAI GPT are single-model, single task. All results obtained from https://gluebenchmark.com/leaderboard and https://blog.openai.com/language-unsupervised/.

ELMo / ULMFiT / GPT / BERT

	ELMo	ULMFiT	GPT	BERT
特点	基于特征，需要特定任务模型组合	精调：语言模型与特定任务模型相似，可共享		
词向量	提供了基于当前输入、实时计算的词向量	提供与目标任务模型结构相似的语言模型，与词向量无关		
优点	可解决一词多义、词表大小不受限	迁移学习、适合分类任务	捕获长距离依赖、可并行、效果比ELMo和ULMFiT好	捕获双向长距离(网络可更深)、预训练句子表示及分类、预训练分隔符等表示、预训练语料更大
缺点	每个token需计算词向量(慢)	复杂任务难以迁移(Squad)	某些特定任务需调整输入	每次预测15%mask处token，收敛慢
适用任务	Classification/Sequence labeling/NLI	Classification Sequence labeling	Classification/Sequence labeling/NLI/QA and Commonsense reasoning	Classification/Sequence labeling/NLI/QA and Commonsense reasoning

词向量应用范式

- 模型参数初始化
- 模型额外特征
- 精调策略：在目标任务中学习并更新参数
- 联合学习：目标任务与语言模型联合训练

目录

机器学习

- 机器学习(Machine Learning, ML)是一门多领域交叉学科，涉及概率论、统计学、逼近论、凸分析、算法复杂度理论等多门学科。专门研究计算机怎样模拟或实现人类的学习行为，以获取新的知识或技能，重新组织已有的知识结构使之不断改善自身的性能。
- 机器学习是人工智能的一个分支,其目的在于使得机器可以根据数据进行自动学习,通过算法使得机器能从大量历史数据中学习规律从而对新的样本做决策。
- 它目前是人工智能的核心，是使计算机具有智能的根本途径，其应用遍及人工智能的各个领域，它主要使用归纳、综合而不是演绎。

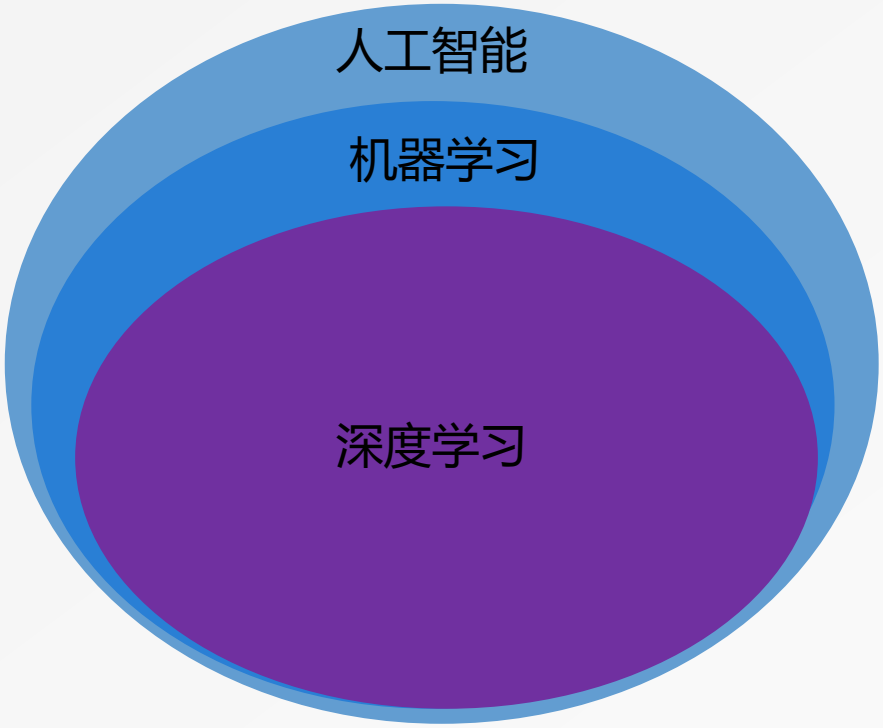
深度学习

- 深度学习(deep learning，DL)是机器学习的分支，它试图使用包含复杂结构或由多重非线性变换构成的多个处理层对数据进行高层抽象的算法。
- 深度学习是一种多层神经网络模型，它能够接受/处理原始数据，能够自动对数据间表征学习。表征学习是寻求更好的表示方法并创建更好的模型，来从大规模(未)标记数据中学习这些数据的抽样表示。
- 目前，有数种深度学习框架，如深度神经网络、卷积神经网络和深度置信网络和递归神经网络已被应用计算机视觉、语音识别、自然语言处理、音频识别与生物信息学等领域并获取了极好的效果。

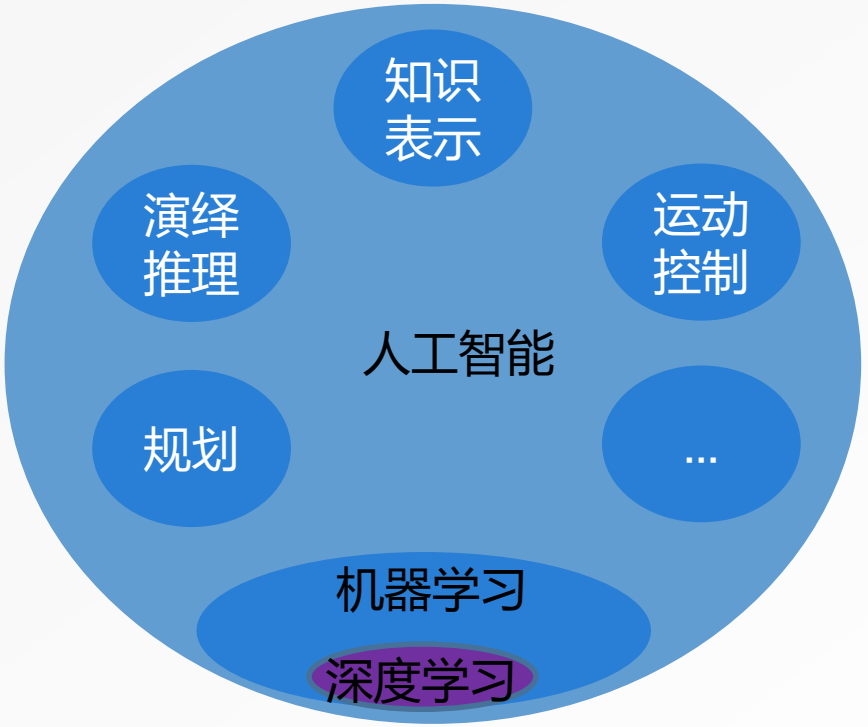
机器学习五大流派

符号主义	逻辑学	 Tom Mitchell	 Steve Muggleton	 Ross Quinlan
联结主义	人工神经网络	 Geoff Hinton	 Yann Lecun	 Yoshua Bengio
进化主义	进化计算	 John Koza	 John Holland	 Hop Lipson
贝叶斯主义	Bayes Inference	 Judea pearl	 Michael Jordan	 David Heckerman
行为类推主义	KNN、SVM	 Peter Hart	 Vladimir Vapnik	 Douglas Hofstadter

机器学习 VS. 深度学习 VS. 人工智能



我们以为的关系



它们实际的关系

信息表示发展历程

数据网络

知识图谱

语言

视觉
听觉
触觉
味觉
嗅觉

感知化

符号化

关联化

度量化

还有很长的路要走

- 计算机无法真正理解符号、数字背后的语义
- 所有的都是在“猜”
- 图像、语音：原始信号
- 语言：认知信息

中国科学院自动化研究所
INSTITUTE OF AUTOMATION
CHINESE ACADEMY OF SCIENCES

谢谢！Q&A！