AI CV NLP基础知识学习笔记

Yanjie Huang

Beijing Institute of Technology

School of Integrated Circuits and Electronics

Class of 2018 (Undergrad) & 2022(Master)

(Most of contents are done during my internship at MIPL, Wangxuan Institute of Computer Technology, Peking University)

E-mail：3120221317@bit.edu.cn

Github: https://github.com/YanjieHuang-ECE/

Welcome to follow!

Attention & 视觉Transformer

reference

- Transformer集合https://blog.csdn.net/zandaoguang/article/details/112598022
- 视觉Transformerhttps://blog.csdn.net/moxibingdao/article/details/109127507
- 直观解释Attention模型 https://zhuanlan.zhihu.com/p/62397974
- Attention is all you need解读 https://zhuanlan.zhihu.com/p/44731789

Attention is all you need

Q=query

V=value

K=key

（下图中提及的“如图3左所示”请忽略）

从不同输入得到的向量然后被打包成三个不同的矩阵，之后，不同输入向量之间的注意力函数通过以下步骤计算(如图3左所示):

- **Step 1:** Compute scores between different input vectors with $S = Q \cdot K^\top$;
- **Step 2:** Normalize the scores for the stability of gradient with $S_n = S/\sqrt{d_k}$;
- **Step 3:** Translate the scores into probabilities with softmax function $P = softmax(S_n)$;
- **Step 4:** Get the weighted value matrix with $Z = V \cdot P$.

这个过程可以统一为一个函数:

$$Attention(Q, K, V) = softmax(\frac{Q \cdot K^\top}{\sqrt{d_k}}) \cdot V. \quad (1)$$

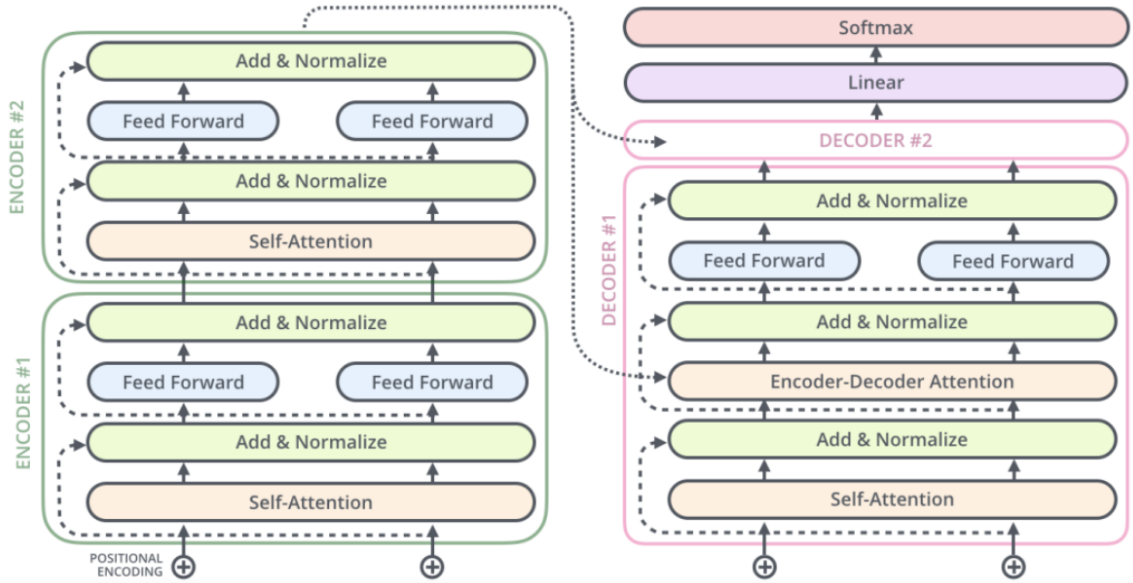
第一步计算两个不同向量之间的得分，得分是为了确定我们在当前位置对单词进行编码时对其他单词的关注程度。步骤2将分数标准化，使其具有更稳定的梯度，以便更好地训练，步骤3将分数转换为概率。最后，每个值向量乘以加总概率，具有较大概率的向量将被随后的层更多地关注。

解码器模块中的编解码注意力层与编码模块中的自注意力层几乎相同，只是键矩阵 K 和值矩阵 V 是从编码器模块中导出的，查询矩阵 Q 是从前一层中导出的。

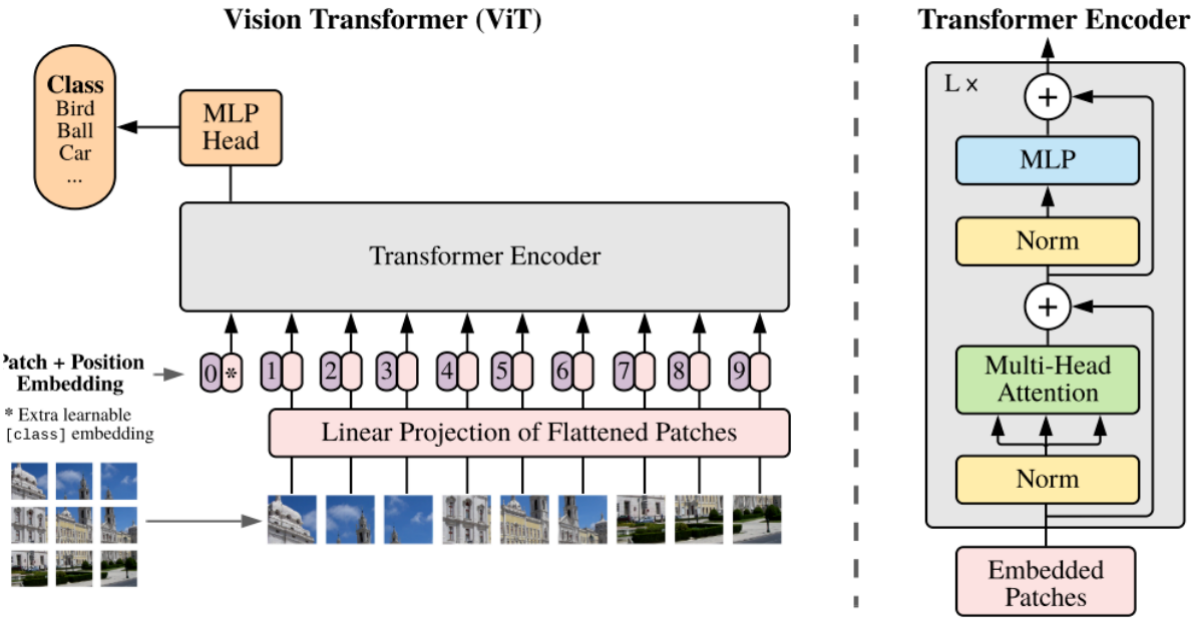
向量的点积可以认为是计算两个向量的相似性

1. Encoder-Decoder

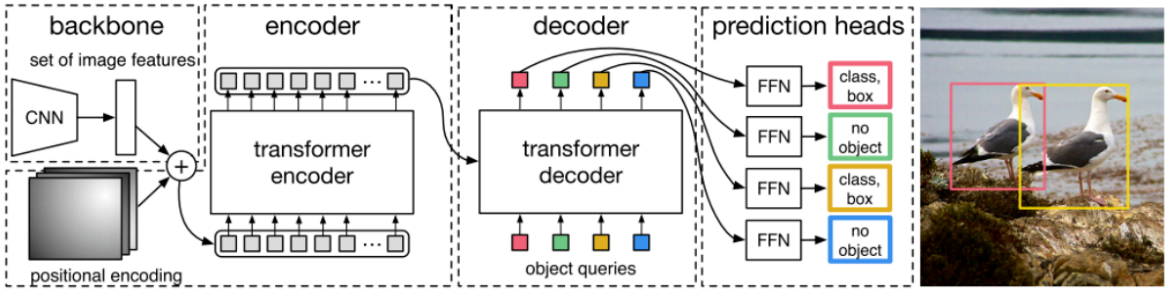
Transformer结构可以表示为Encoder和Decoder两个部分



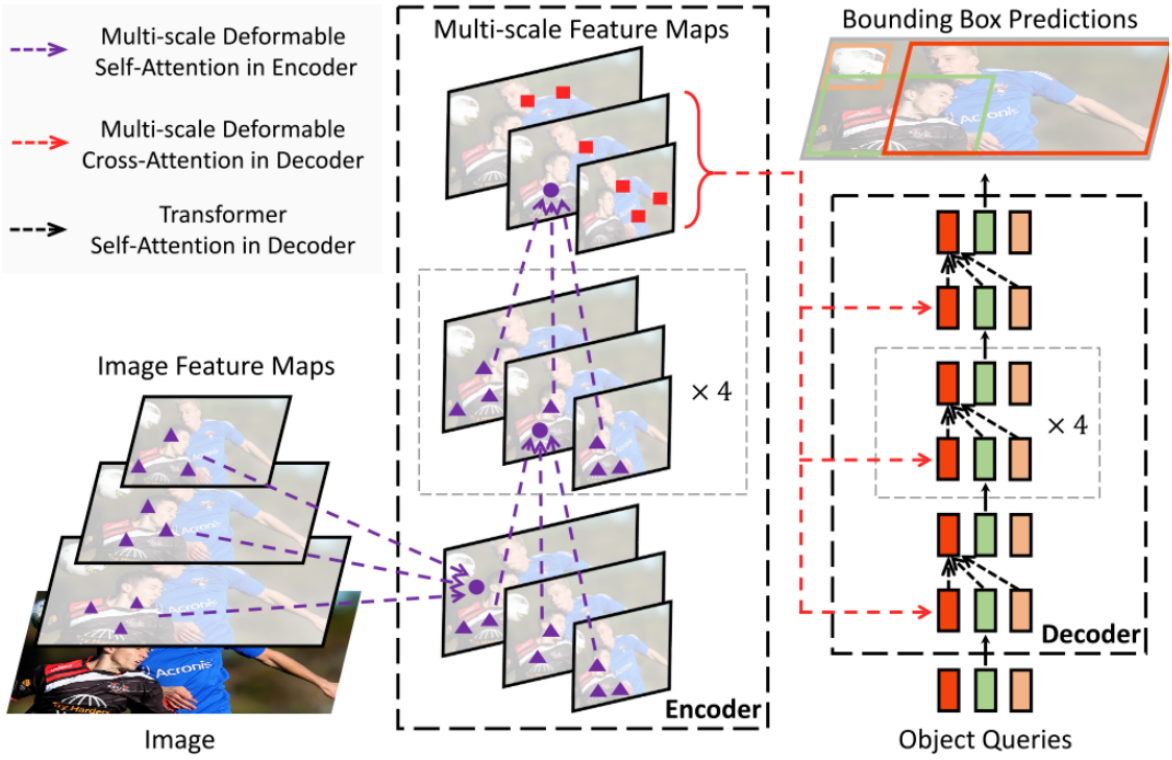
ViT



DETR



Deformable DETR



BERT

- 综合理解https://blog.csdn.net/jiaowoshouzi/article/details/89073944

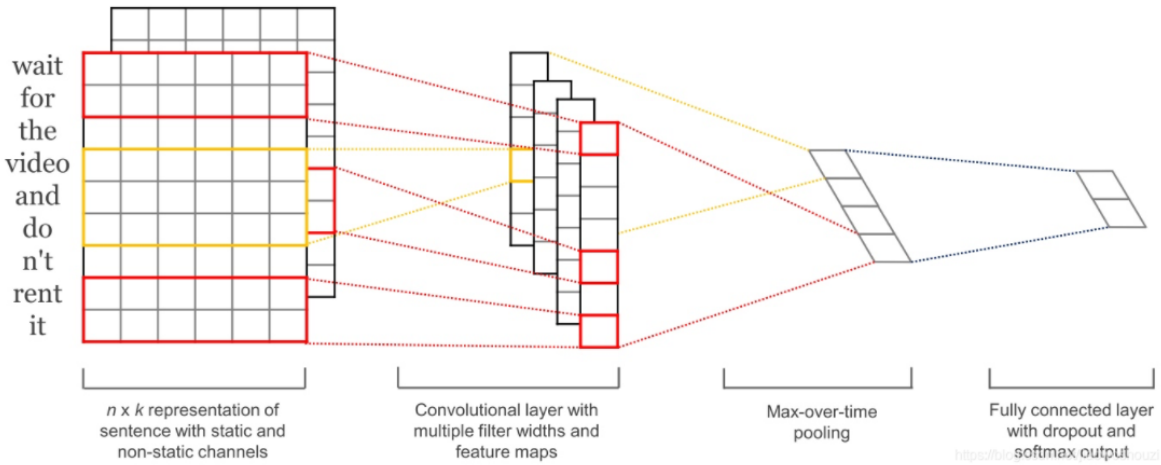
词嵌入：word2vec 的目标是简化语言建模。

sequence-to-sequence 模型：这种模型通过一次预测一个单词生成一个输出序列。

预训练语言模型：这些方法使用来自语言模型的表述进行迁移学习。

双向 LSTM通常用于处理左右两边的上下文

卷积神经网络（CNNs） 卷积神经网络本来是广泛应用于计算机视觉领域的技术，现在也开始应用于语言（Kalchbrenner等,2014；Kim等,2014）。文本的卷积神经网络只在两个维度上工作，其中滤波器（卷积核）只需要沿着时间维度移动。下面的图显示了NLP中使用的典型 CNN。



卷积神经网络的一个优点是它们比 RNN 更可并行化，因为其在每个时间步长的状态只依赖于本地上下文（通过卷积运算），而不是像 RNN 那样依赖过去所有的状态。使用膨胀卷积，可以扩大 CNN 的感受野，使网络有能力捕获更长的上下文（Kalchbrenner等,2016）。

Bert的pre training由Masked Language Model (MLM)和Next Sentence Prediction(NSP)组成。

MLM可以理解为完形填空，作者会随机mask每一个句子中15%的词，用其上下文来做预测，例如：my dog is hairy → my dog is [MASK]

此处将hairy进行了mask处理，然后采用非监督学习的方法预测mask位置的词是什么，但是该方法有一个问题，因为是mask 15%的词，其数量已经很高了，这样就会导致某些词在fine-tuning阶段从未见过，为了解决这个问题，作者做了如下的处理：

1.80%的时间是采用[mask]，my dog is hairy → my dog is [MASK]

2.10%的时间是随机取一个词来代替mask的词，my dog is hairy -> my dog is apple

3.10%的时间保持不变，my dog is hairy -> my dog is hairy

那么为啥要以一定的概率使用随机词呢？**这是因为transformer要保持对每个输入token分布式的表征，否则Transformer很可能会记住这个[MASK]就是"hairy"。**至于使用随机词带来的负面影响，文章中解释说,所有其他的token(即非"hairy"的token)共享15%*10% = 1.5%的概率，其影响是可以忽略不计的。 **Transformer全局的可视，又增加了信息的获取，但是不让模型获取全量信息。**

Bert先是用Mask来提高视野范围的信息获取量，增加duplicate再随机Mask，这样跟RNN类方法依次训练预测没什么区别了除了mask不同位置外；

全局视野极大地降低了学习的难度，然后再用A+B/C来作为样本，这样每条样本都有50%的概率看到一半左右的噪声；

但直接学习Mask A+B/C是没法学习的，因为不知道哪些是噪声，所以又加上next_sentence预测任务，与MLM同时进行训练，这样用next来辅助模型对噪声/非噪声的辨识，用MLM来完成语义的大部分的学习。

RNN 循环神经网络

- 基础科普https://zhuanlan.zhihu.com/p/30844905
- 详细推导https://blog.csdn.net/qq_32241189/article/details/80461635

RNN公式

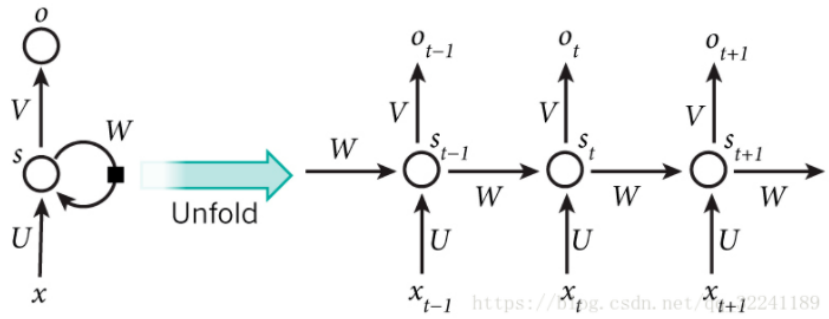


图2 Hidden Layer的层级展开图

$O_t=g(V\cdot S_t)$

$S_t=f(U\cdot X_t+W\cdot S_{t-1})$

- S_t的值不仅取决于X_t，还取决于S_{t-1}
- f激活函数，可以为$tanh,relu,sigmoid$

- g激活函数,通常取$softmax$
- W,U,V在每个时刻相等(parameter sharing)
- 隐藏状态：$S=f(现有输入+过去记忆总结)$

Back Propagation

$$\frac{\partial E_3}{\partial W}=\sum_{k=0}^{3}\frac{\partial E_3}{\partial o_3}\frac{\partial o_3}{\partial s_3}\frac{\partial s_3}{\partial s_k}\frac{\partial^+ s_k}{\partial W}$$

这里要说明的是: $\frac{\partial S_k^+}{\partial w}$ 表示的是S3对W直接求导, 不考虑S2的影响.(也就是例如y = f(x)*g(x)对x求导一样)

其次是对**U的更新方法**. 由于参数U求解和W求解类似,这里就不在赘述了,最终得到的具体的公式如下:

$$\frac{\partial E_3}{\partial U}=\sum_{k=0}^{3}\frac{\partial E_3}{\partial o_3}\frac{\partial o_3}{\partial s_3}\frac{\partial\left(W^{3-k}a_k\right)}{\partial U}\frac{\partial S_3}{\partial f}$$

最后,给出**V的更新公式**(V只和输出O有关):

$$\frac{\partial E_3}{\partial V}=\frac{\partial E_3}{\partial O_3}*\frac{\partial O_3}{\partial V}$$

LSTM&GRU

- Reference

  LSTM基本概念https://zhuanlan.zhihu.com/p/32085405

  LSTM公式总结https://zhuanlan.zhihu.com/p/76174753

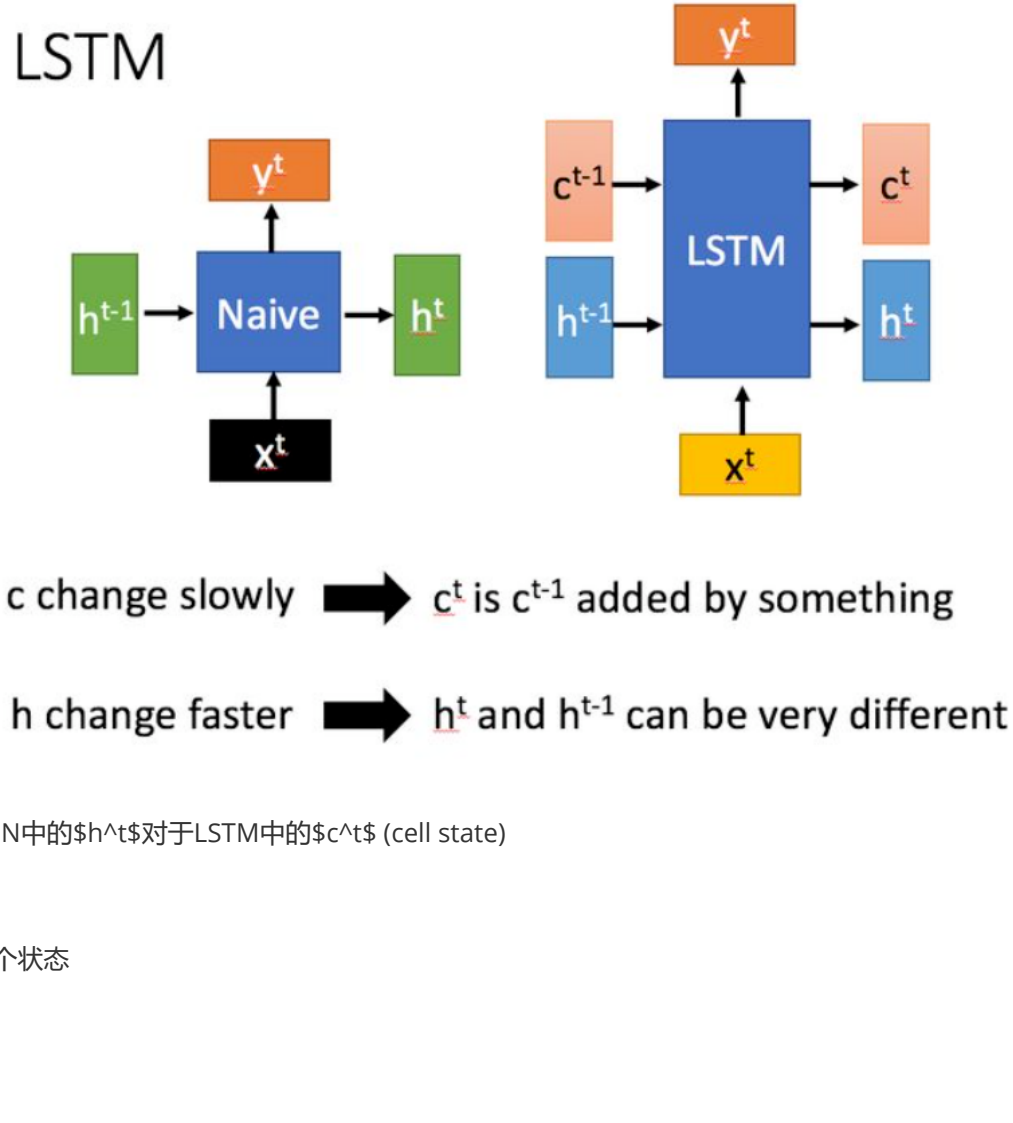
  GRU基本概念https://zhuanlan.zhihu.com/p/32481747

  GRU公式总结https://blog.csdn.net/silent_crown/article/details/84729258

- LSTM Long Short Term Memory

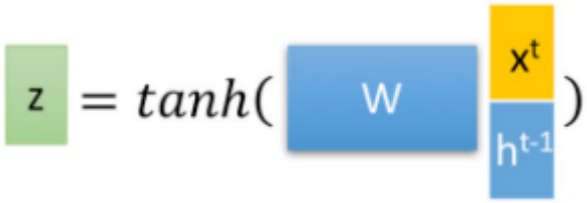
相比普通的RNN，LSTM能够在更长的序列中有更好的表现。

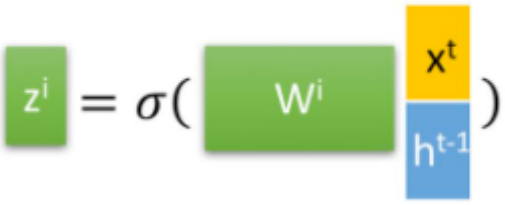
普通RNN（左）VS LSTM（右）

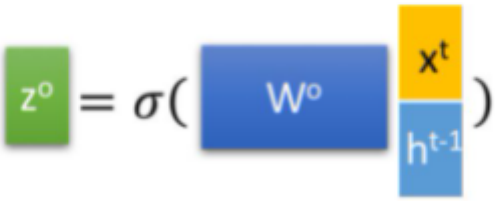


RNN中的h^t对于LSTM中的c^t (cell state)

四个状态





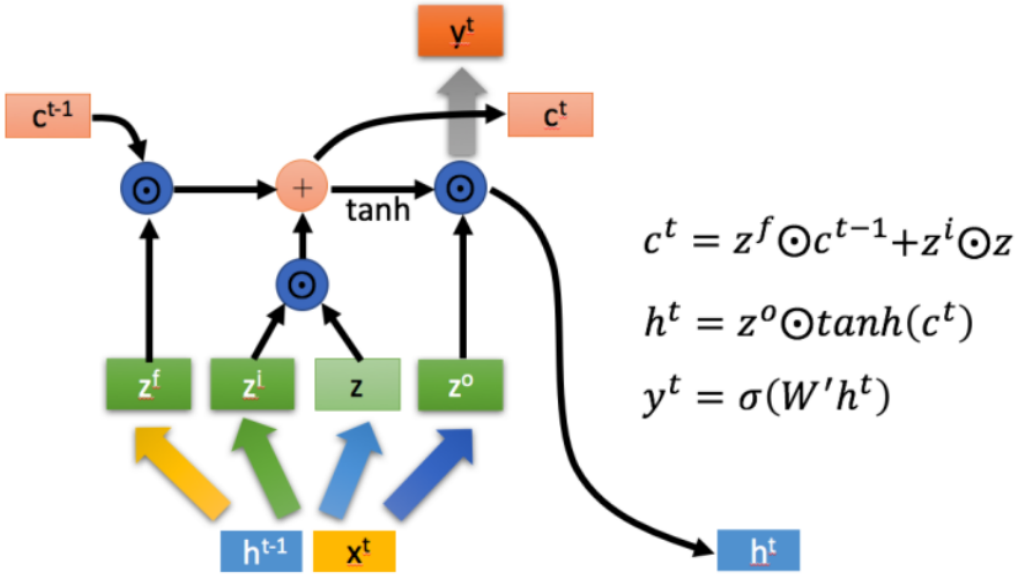


z^f 遗忘门 选择上一个状态的c^{t-1}需要遗忘的内容

z^i 信息增强门 对于输入x^t进行选择性记忆

z^o 输出门 决定当前状态的输出

内部结构



⊙ 是Hadamard Product，也就是操作矩阵中对应的元素相乘，因此要求两个相乘矩阵是同型的。 ⊕ 则代表进行矩阵加法。

- GRU Gate Recurrent Unit

GRU输入输出的结构与普通的RNN相似，其中的内部思想与LSTM相似。

与LSTM相比，GRU内部少了一个"门控"，参数比LSTM少，但是却也能够达到与LSTM相当的功能。

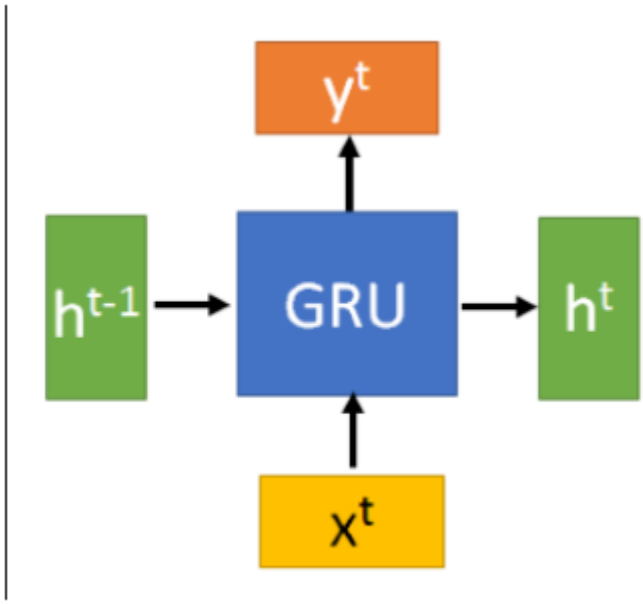


图2-1 GRU的输入输出结构

r reset gate

z update gate

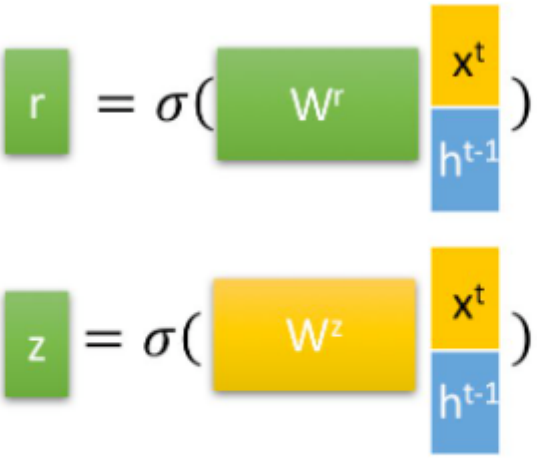
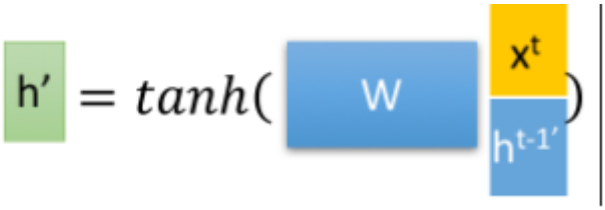


图2-2 r，z门控



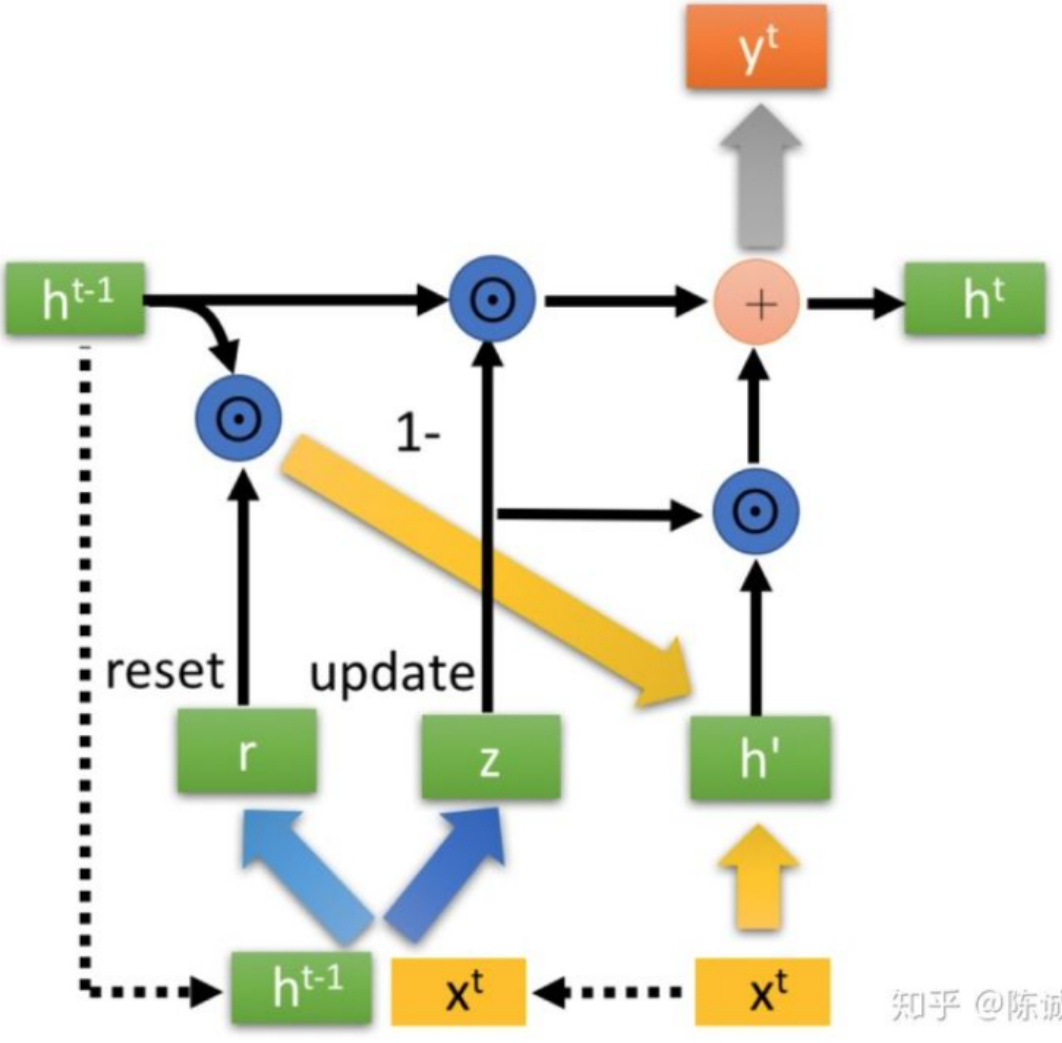


图2-4 GRU的内部结构

更新记忆：同时进行遗忘+记忆

更新表达式：$h^t=(1-z)\odot h^{t-1}+z\odot h^\prime $

前一项相当于LSTM的遗忘门 后一项相当于信息增强门

对于传递进来的维度信息，我们会进行选择性遗忘。遗忘了多少权重(z)，我们就会使用包含当前输入的$h^\prime$ 对应的权重1-z进行弥补

- LSTM与GRU的关系

r (reset gate)实际上与它的名字有点不符。我们仅仅使用它来获得了 $h^\prime$。

那么这里的 $h^\prime$ 实际上可以看成对应于LSTM中的hidden state；上一个节点传下来的h^{t-1}则对应于LSTM中的cell state。$1-z$对应的则是LSTM中的 z^fforget gate，那么z我们似乎就可以看成是信息增强门z^i 了。

交叉熵Cross Entropy Loss

- Cross Entropy Loss和MSE、softmax等的比较https://blog.csdn.net/xg123321123/article/details/80781611
- PyTorch中的Cross Entropy Losshttps://zhuanlan.zhihu.com/p/98785902
- PyTorch 中的Cross Entropy Loss vs NLL Loss vs KLDiv Losshttps://zhuanlan.zhihu.com/p/83283586

交叉熵主要是用来判定实际的输出与期望的输出的接近程度

公式：

$H_{y^\prime}(y)=-\sum_iy_i^\prime log(y_i)$

其中，y_i是预测结果，$y_i^\prime$是ground truth

另外需要注意的是，**在PyTorch中，CrossEntropyLoss其实是LogSoftMax和NLLLoss的合体。**

$$\text{loss}(x, class) = -\log\left(\frac{\exp(x[class])}{\sum_j \exp(x[j])}\right) = -x[class] + \log\left(\sum_j \exp(x[j])\right)$$

匈牙利算法&分水岭算法

- 匈牙利算法

https://blog.csdn.net/u013384984/article/details/90718287

图G的一个匹配是由一组没有公共端点的不是圈的边构成的集合。

匹配的两个重点：1. 匹配是边的集合；2. 在该集合中，任意两条边不能有共同的顶点。

总结：

1. 匈牙利算法寻找最大匹配，就是通过不断寻找原有匹配M的增广路径，因为找到一条M匹配的增广路径，就意味着一个更大的匹配$M^{\prime}$, 其恰好比M多一条边。
2. 对于图来说，最大匹配不是唯一的，但是最大匹配的大小是唯一的。

- 分水岭算法

原理简述https://blog.csdn.net/niepengpeng333/article/details/8218160

OpenCV实现https://blog.csdn.net/dcrmg/article/details/52498440

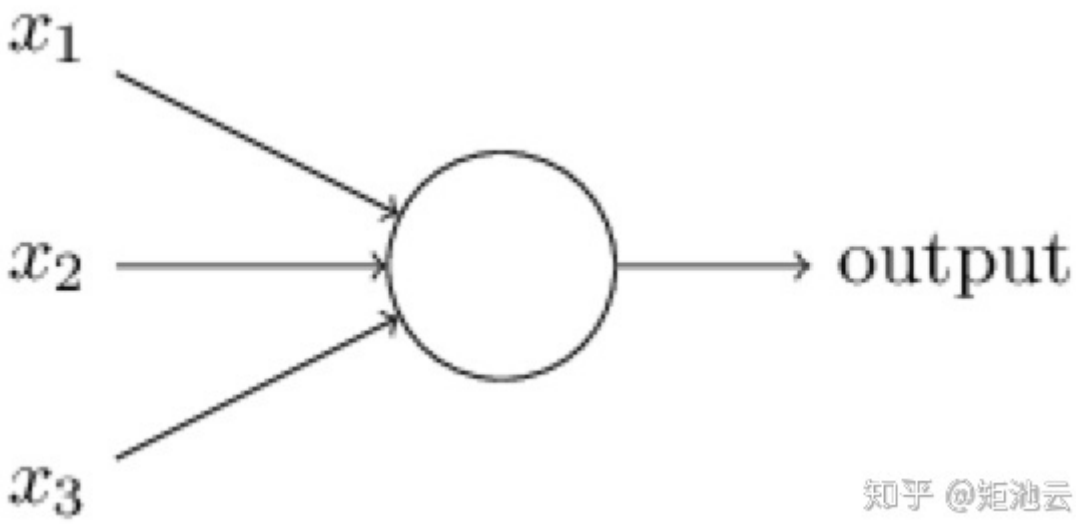
分水岭算法主要的分割目的在于找到图像的连通区域。利用梯度信息作为输入图像，会有一个矛盾点，如果对原始图像进行梯度计算时不作滤波平滑处理，很容易将 物体分割成多个物体，那是因为噪声的影响；而如果进行滤波处理，又容易造成将某些原本几个的物体合成一个物体。当然这里的物体主要还是指图像变化不大或者 说是灰度值相近的目标区域。

前向传播网络FFN

- 基本概念 https://zhuanlan.zhihu.com/p/96243893

FFN是最简单的神经网络，没有反馈

- 感知机



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

output是激活函数的最原始形态

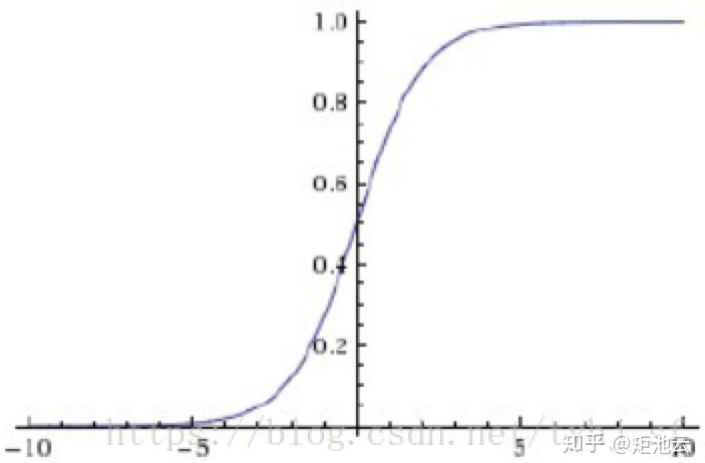
- 激活函数

使用激活函数的原因：

若不使用，则输出为输入参数线性叠加的结果。为了适应非线性问题，在输出层加入非线性函数（激活函数），将网络的线性结果映射到非线性空间。

常用的激活函数：

Sigmoid $f(z)=\frac {1}{1+e^{-z}} $

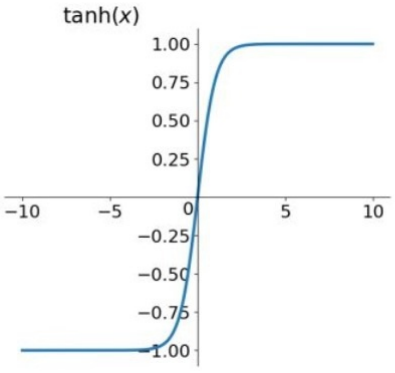


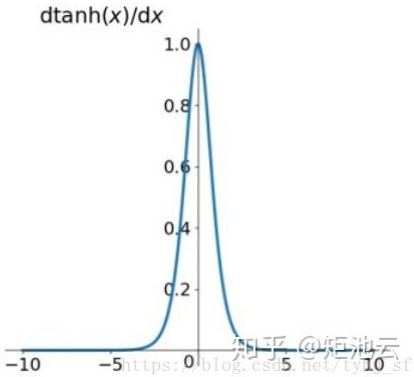
它能够把输入的连续实值变换为0和1之间的输出，特别的，如果是非常大的负数，那么输出就是0；如果是非常大的正数，输出就是1

但是这个函数也有如下的缺点：

1. 梯度消失
2. 不是以0值为中心
3. 运算比较耗时，因为需要用到幂运算

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

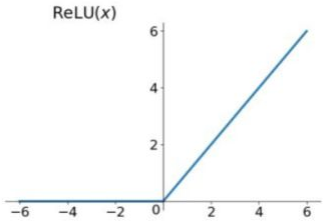


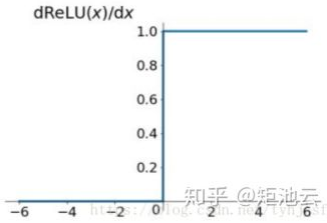


与Sigmoid比较:

解决了输出不是0中心的问题，但是梯度消失跟幂运算还是存在

Relu $Relu=max(0,x)$





Relu函数有如下几个优点：

1. 解决了gradient vanishing问题 (在正区间)
2. 计算速度非常快，只需要判断输入是否大于0
3. 收敛速度远快于sigmoid和tanh

同时它的缺点是:

1. ReLU的输出不是0中心
2. Dead ReLU Problem，指的是某些神经元可能永远不会被激活，导致相应的参数永远不能被更新。

深度残差网络ResNet

残差 数据点与它在回归直线上相应位置的差异

- ResNet综述https://zhuanlan.zhihu.com/p/31852747/
- 残差学习单元

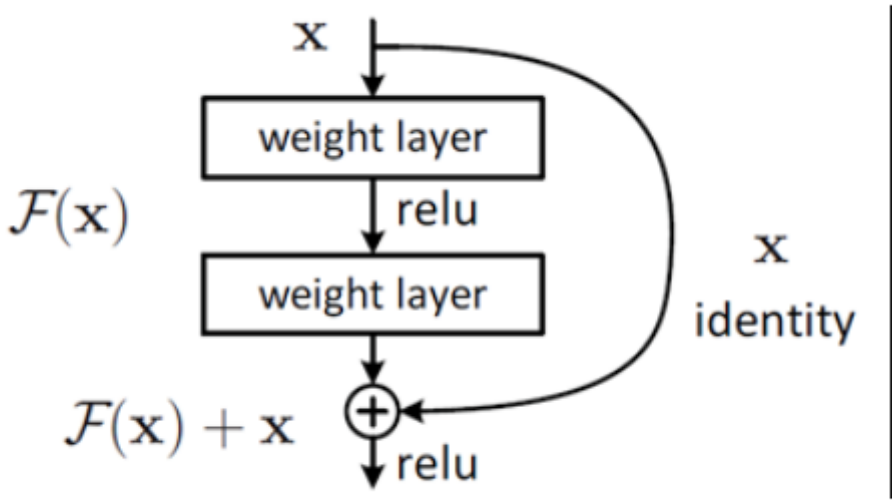


图4 残差学习单元

对于一个堆积层结构（几层堆积而成）当输入为 x 时其学习到的特征记为$H(x)$ ，现在我们希望其可以学习到残差 $F(x)=H(x)-x$ ，这样其实原始的学习特征是$F(x)+x$ 。之所以这样是因为残差学习相比原始特征直接学习更容易。当残差为0时，此时堆积层仅仅做了恒等映射，至少网络性能不会下降，实际上残差不会为0，这也会使得堆积层在输入特征基础上学习到新的特征，从而拥有更好的性能。残差学习的结构如上图所示。这有点类似与电路中的“短路”，所以是一种短路连接。

什么残差学习相对更容易，从直观上看残差学习需要学习的内容少，因为残差一般会比较小，学习难度小点。不过我们可以从数学的角度来分析这个问题，首先残差单元可以表示为：

$y_l=h(x_l)+F(x_l,W_l)$

$x_{l+1}=f(y_l)$

其中 x_l 和x_{l+1} 分别表示的是第个l残差单元的输入和输出，注意每个残差单元一般包含多层结构。F是残差函数，表示学习到的残差，而 $h(x_l)=x_l$表示恒等映射，f是ReLU激活函数。基于上式，我们求得从浅层l到深层 L 的学习特征为：

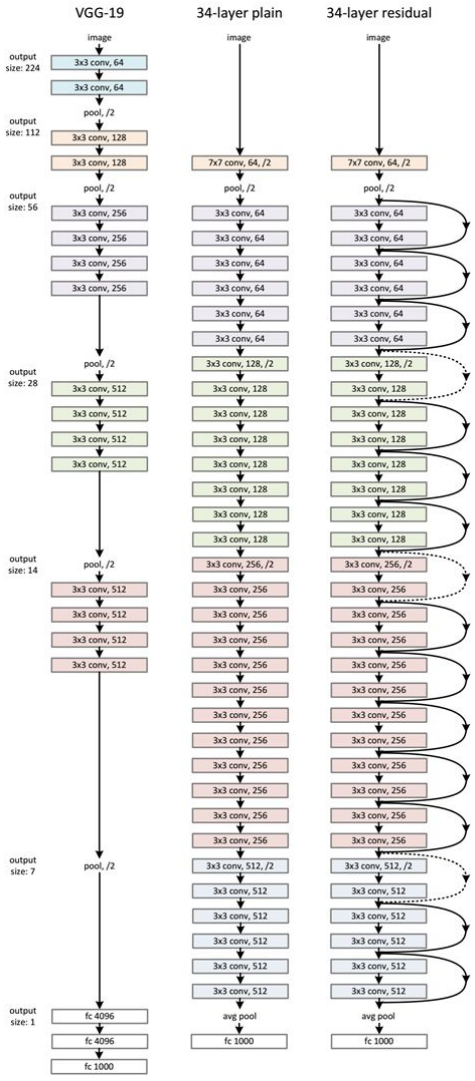
$x_L=x_l+\sum^{L-1}_{i=l}F(x_i,W_i)$

利用链式规则，可以求得反向过程的梯度：

$\frac {\partial loss}{\partial x_l}=\frac {\partial loss}{\partial x_L}\cdot\frac {\partial x_L}{\partial x_l}=\frac {\partial loss}{\partial x_L}\cdot(1+\frac{\partial}{\partial x_L}\sum^{L-1}_{i=L}F(x_i,W_i)) $

式子的第一个因子 $\frac{\partial loss}{\partial x_L}$表示的损失函数到达 L 的梯度，小括号中的1表明短路机制可以无损地传播梯度，而另外一项残差梯度则需要经过带有weights的层，梯度不是直接传递过来的。残差梯度不会那么巧全为-1，而且就算其比较小，有1的存在也不会导致梯度消失。

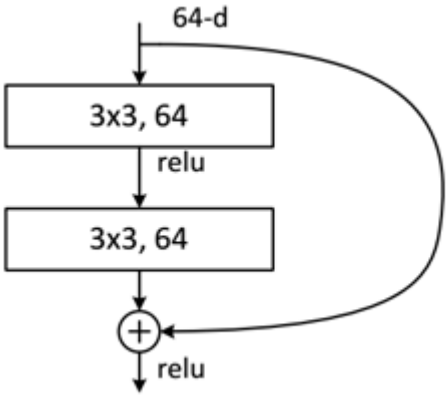
- ResNet的网络结构

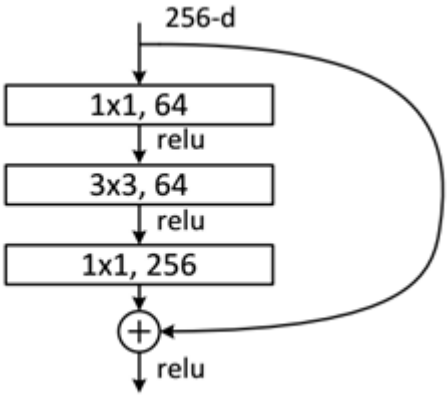


ResNet使用两种残差单元，如下图所示。左图对应的是浅层网络，而右图对应的是深层网络。对于短路连接，当输入和输出维度一致时，可以直接将输入加到输出上。但是当维度不一致时（对应的是维度增加一倍），这就不能直接相加。有两种策略：

1.采用zero-padding增加维度，此时一般要先做一个downsamp，可以采用strde=2的pooling，这样不会增加参数；

2.采用新的映射（projection shortcut），一般采用1x1的卷积，这样会增加参数，也会增加计算量。短路连接除了直接使用恒等映射，当然都可以采用projection shortcut。





ResNet很好地解决了普通网络的退化问题（网络深度增加时，网络准确度出现饱和，甚至出现下降。）

随机梯度下降SGD&Dropout

- SGD stochastic gradient descent

  简要版+Python代码实现https://blog.csdn.net/qq_38150441/article/details/80533891

  公式理解https://www.zhihu.com/question/264189719

  每个数据都计算一下损失函数，然后求梯度更新参数。每次更新时使用一个函数

  优点：计算速度快

  缺点：收敛性能不好

梯度下降更新公式：

假设Loss Function为 $J(\theta)=\frac {1}{2}\sum^{m}*{i=1}(h{\theta}(x)-y)^2$

其中$h_{\theta}(x)=\theta_0+\theta_1x_1+\theta_2x_2+...+\theta_nx_n$

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

$$\begin{aligned}
\frac{\partial}{\partial \theta_j} J(\theta) &= \frac{\partial}{\partial \theta_j} \frac{1}{2} \left(h_\theta(x) - y\right)^2 \\
&= 2 \cdot \frac{1}{2} \left(h_\theta(x) - y\right) \cdot \frac{\partial}{\partial \theta_j} (h_\theta(x) - y) \\
&= \left(h_\theta(x) - y\right) \cdot \frac{\partial}{\partial \theta_j} \left(\sum_{i=0}^{n} \theta_i x_i - y \right) \\
&= \left(h_\theta(x) - y\right) x_j
\end{aligned}$$

SGD更新公式：

Loop {

for i=1 to m, {

$$\theta_j := \theta_j + \alpha \left(y^{(i)} - h_\theta(x^{(i)}) \right) x_j^{(i)} \qquad \text{(for every } j\text{)}.$$

}

}

能收敛到的地方：最小值，极小值，鞍点。这些都是能收敛到的地方，也就是梯度为0的点。

- Dropout

原理与代码实现https://blog.csdn.net/program_developer/article/details/80737724

过拟合具体表现：模型在训练数据上损失函数较小，预测准确率较高；但是在测试数据上损失函数比较大，预测准确率较低。

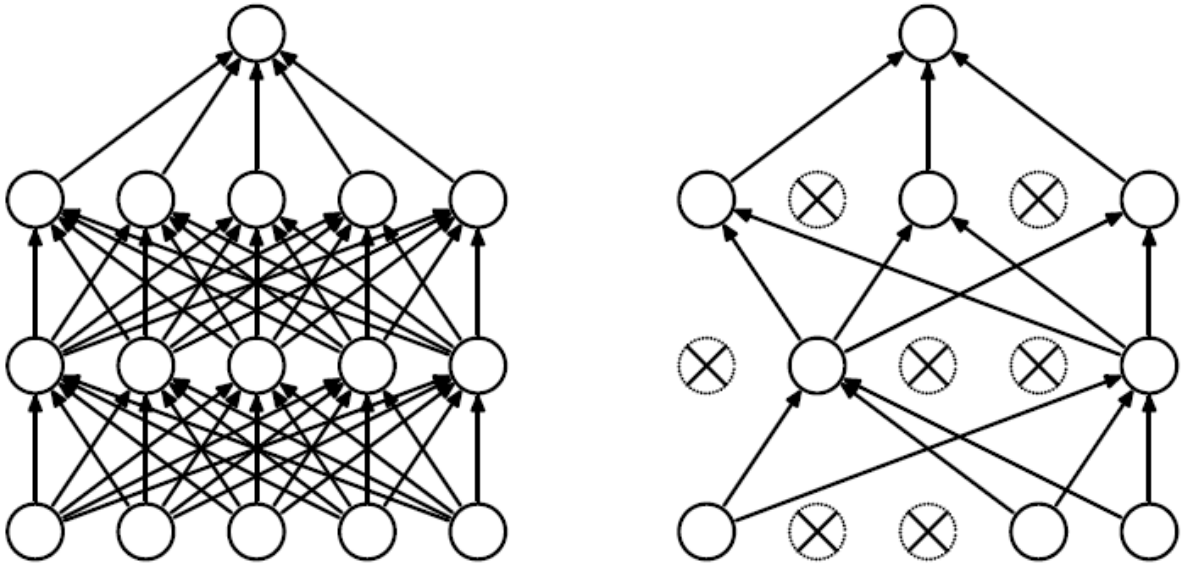
训练深度神经网络的时候，总是会遇到两大缺点：

1.容易过拟合

2.费时

Dropout可以比较有效的缓解过拟合的发生，在一定程度上达到正则化的效果。

Dropout说的简单一点就是：我们在前向传播的时候，让某个神经元的激活值以一定的概率p停止工作，这样可以使模型泛化性更强，因为它不会太依赖某些局部的特征，如下右图所示。



(a) Standard Neural Net (b) After applying dropout.

运行流程：

（1）首先随机（临时）删掉网络中一半的隐藏神经元，输入输出神经元保持不变（图3中虚线为部分临时被删除的神经元）

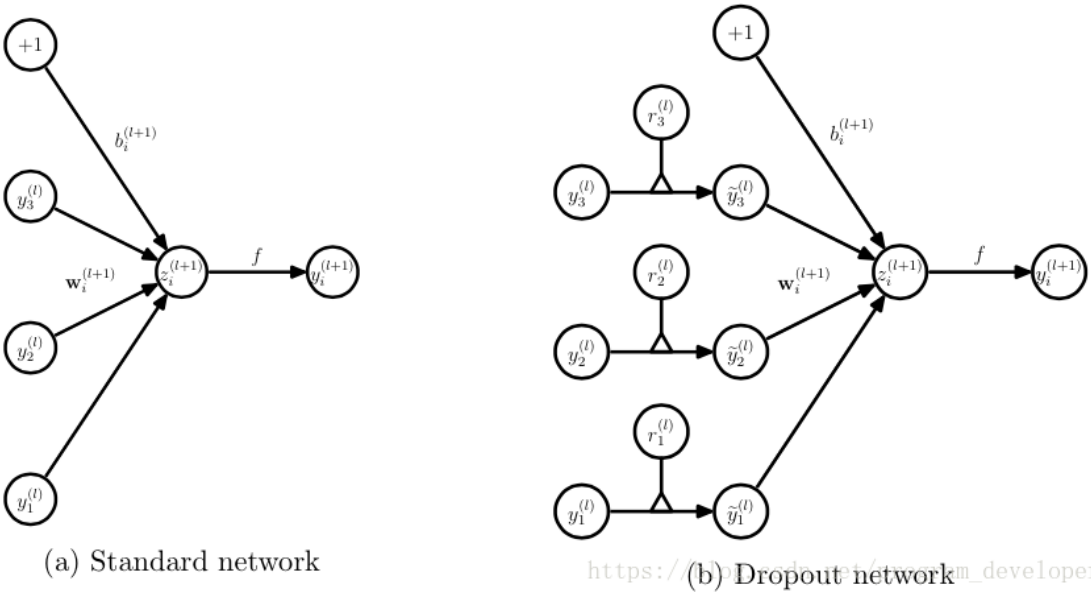
（2） 然后把输入x通过修改后的网络前向传播，然后把得到的损失结果通过修改的网络反向传播。一小批训练样本执行完这个过程后，在没有被删除的神经元上按照随机梯度下降法更新对应的参数（w，b）。

（3）然后继续重复这一过程：

. 恢复被删掉的神经元（此时被删除的神经元保持原样，而没有被删除的神经元已经有所更新）
. 从隐藏层神经元中随机选择一个一半大小的子集临时删除掉（备份被删除神经元的参数）。
. 对一小批训练样本，先前向传播然后反向传播，损失并根据随机梯度下降法更新参数（w，b）（没有被删除的那一部分参数得到更新，删除的神经元参数保持被删除前的结果）。
不断重复这一过程。

公式更新：

训练阶段



(a) Standard network　　(b) Dropout network

无Dropout：

$z^{(l+1)}*i=w^{(l+1)}*iy^l+b^{(l+1)}_i$

$y^{(l+1)}*i=f(z^{(l+1)}*i)$

使用Dropout：

$r^{(l)}_j\sim Bernoulli(p)$

$\widetilde y^{(l)}=r^{(l)}*y^{(l)}$

$z^{(l+1)}*i=w^{(l+1)}*i\widetilde y^l+b^{(l+1)}_i$

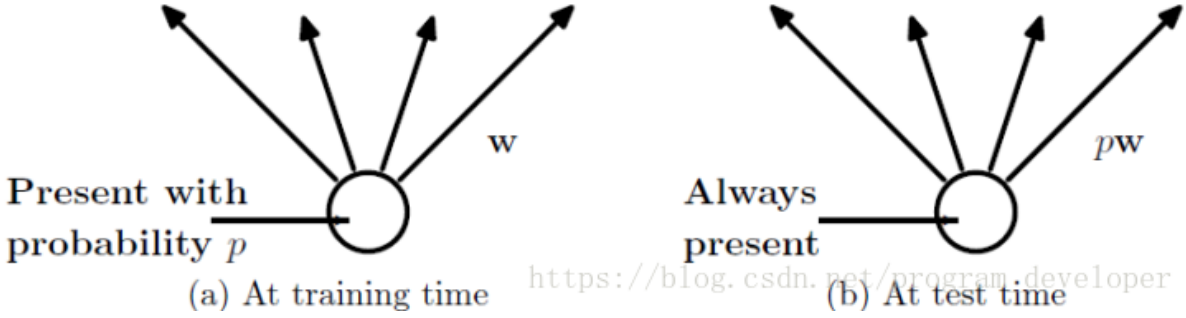
$y^{(l+1)}*i=f(z^{(l+1)}*i)$

上文中Bernoulli函数是为了生成概率r向量，也就是随机生成一个0、1的向量。

注意： 经过上面屏蔽掉某些神经元，使其激活值为0以后，我们还需要对向量y1……y1000进行缩放，也就是乘以1/(1-p)。如果你在训练的时候，经过置0后，没有对y1……y1000进行缩放（rescale），那么在测试的时候，就需要对权重进行缩放，操作如下：

测试阶段：

预测模型的时候，每一个神经单元的权重参数要乘以概率p。

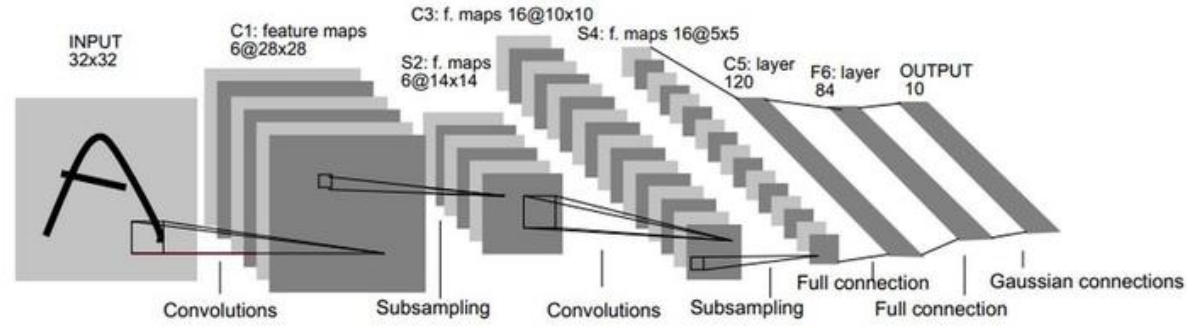


测试阶段Dorpout公式：

$w^{(l)}_{test}=pW^{(l)}$

LeNet

- LeNet详解https://blog.csdn.net/qq_42570457/article/details/81460807



LeNet-5共有7层，不包含输入，每层都包含可训练参数；每个层有多个Feature Map，每个FeatureMap通过一种卷积滤波器提取输入的一种特征，然后每个FeatureMap有多个神经元。

嵌入Embedding&L1/L2 Norm

- Embedding

概念理解https://www.zhihu.com/question/38002635

广义应用https://zhuanlan.zhihu.com/p/46016518

Embedding就是从原始数据提取出来的Feature，也就是那个通过神经网络映射之后的低维向量。

比如Word Embedding，就是把单词组成的句子映射到一个表征向量

- L1/L2 Norm

概念理解https://blog.csdn.net/qq_37466121/article/details/87855185

范数Norm=强化版距离=满足数乘运算法则的距离

L0 Norm 度量向量中非零元素的个数 L0的最优问题会被放宽到L1或L2下的最优化

L1 Norm/曼哈顿距离/最小绝对误差/稀疏规则算子（去掉一些没有信息的特征） 向量x中非零元素的绝对值之和先验服从拉普拉斯分布

可以度量两个向量之间的差异，如绝对误差和(SAD)

$SAD(x_1,x_2)=\sum^n_i|x{1i}-x{2i}|$

优化问题：

$min||x||_1$

$s.t. Ax=b$

L2 Norm 向量元素的平方再开方 可以度量两个向量之间的差异，如平方和SSD 先验服从高斯分布

$SSD(x_1,x_2)=\sum^n{i=1}(x{1i}-x_{2i})^2$

优化问题：

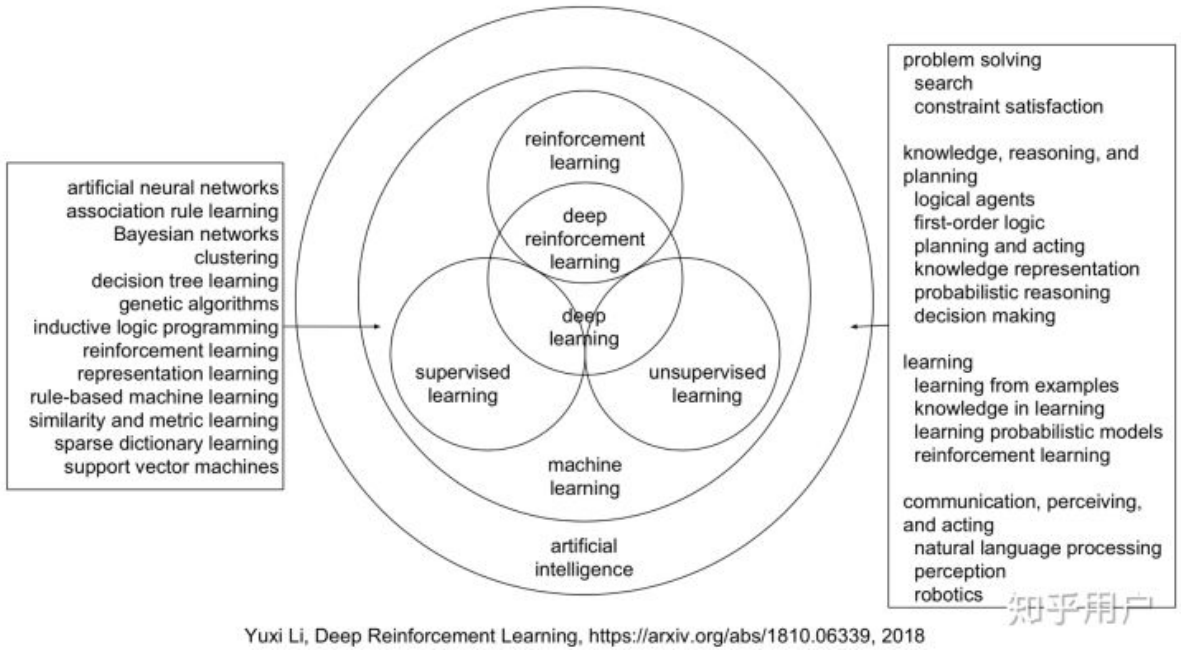
$min||x||_2$

$s.t. Ax=b$

L2范数通常会被用来做优化目标函数的正则化项，防止模型为了迎合训练集而过于复杂造成过拟合的情况，从而提高模型的泛化能力。

机器学习 深度学习 强化学习

- 三者关系https://www.zhihu.com/question/279973545
- 人工智能 机器学习 深度学习 强化学习 迁移学习 历史与基本概念https://blog.csdn.net/princexiexiaofeng/article/details/89063568



Yuxi Li, Deep Reinforcement Learning, https://arxiv.org/abs/1810.06339, 2018

机器学习：一切通过优化方法挖掘数据中规律的学科。

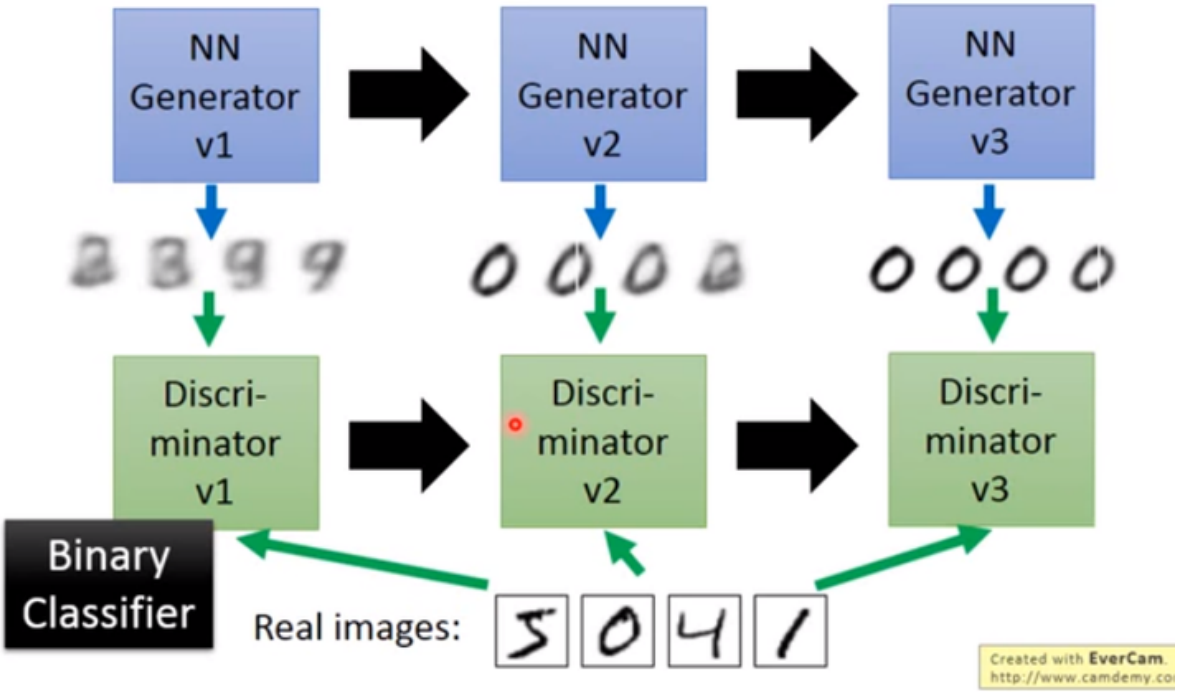
深度学习：一切运用了神经网络作为参数结构进行优化的机器学习算法。有标签。

强化学习：不仅能利用现有数据，还可以通过对环境的探索获得新数据，并利用新数据循环往复地更新迭代现有模型的机器学习算法。学习是为了更好地对环境进行探索，而探索是为了获取数据进行更好的学习。无标签，通过与环境的奖惩进行学习。

深度强化学习：一切运用了神经网络作为参数结构进行优化的强化学习算法。

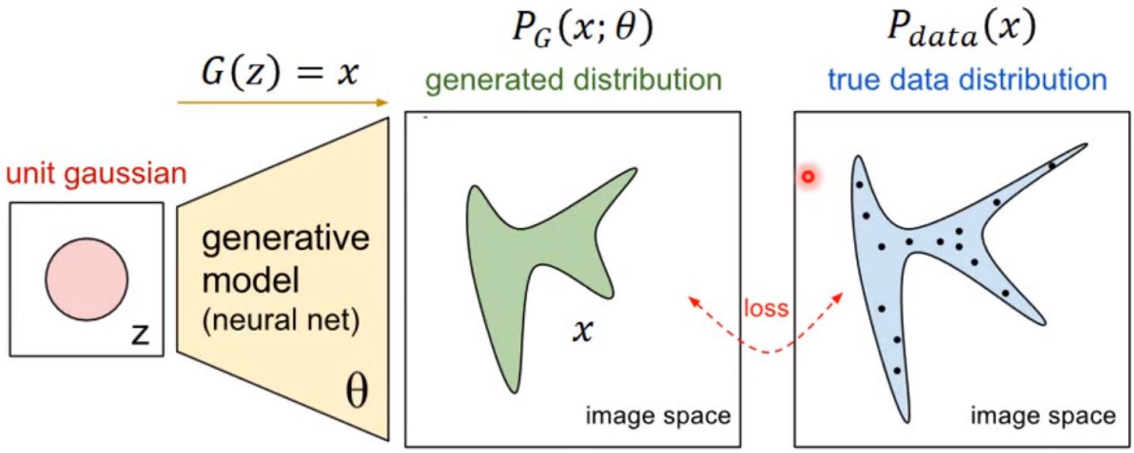
生成对抗网络GAN Generative Adversarial Network

- Reference:GAN笔记https://zhuanlan.zhihu.com/p/27295635
- generation：模型通过学习一些数据，然后生成类似的数据
- 主要流程



- 原理

似然=likelihood=可能性



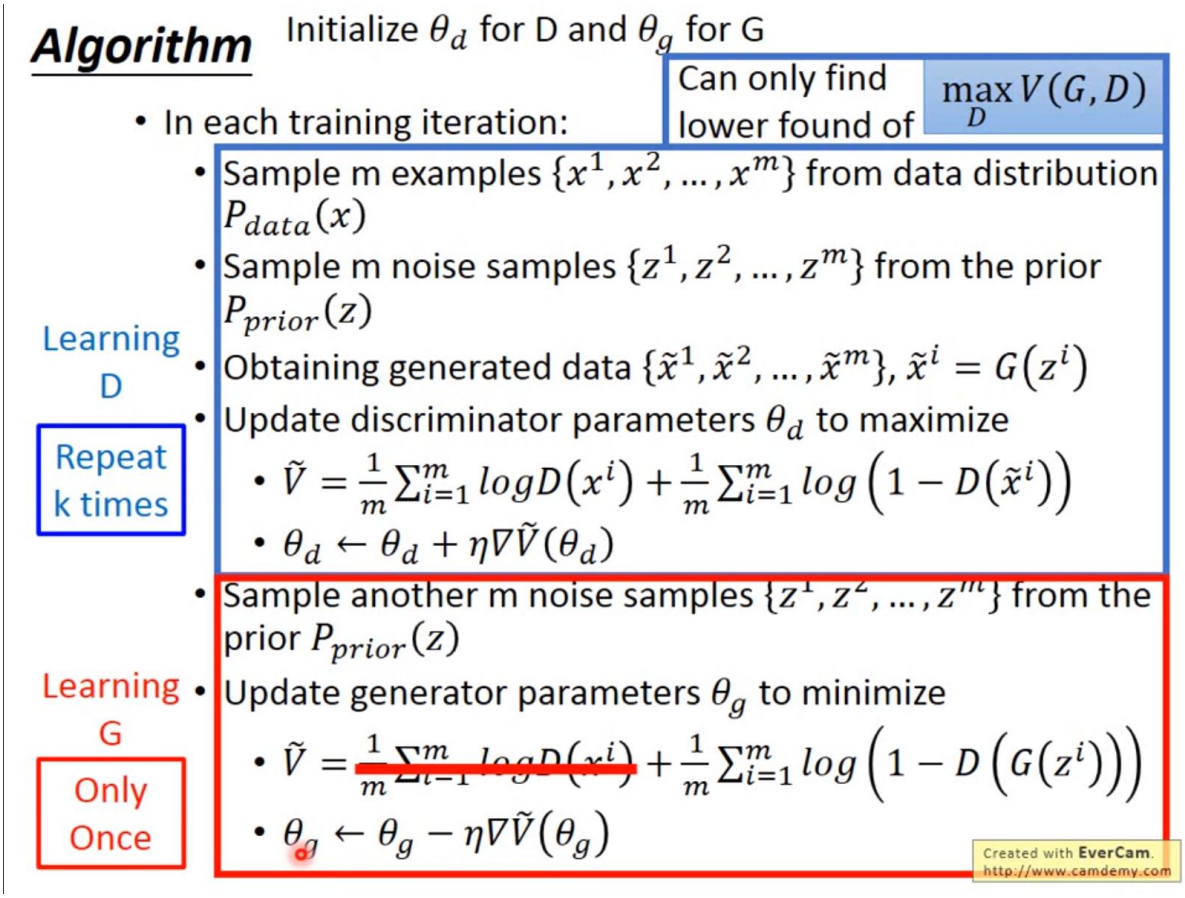
GAN公式$V(G,D)=E{x\sim P{data}}[logD(x)]+E{x\sim P{G}}[log(1-D(x))]$

这个式子的好处在于，固定G，$max V(G,D)$就表示P_G和P_{data}之间的差异，然后要找一个最好的G，让这个最大值最小，也就是两个分布之间的差异最小。

$G^*=arg\space min_G\space max_D\space V(G,D) $

表面上看这个的意思是，D要让这个式子尽可能的大，也就是对于x是真实分布中，D(x)要接近与1，对于x来自于生成的分布，D(x)要接近于0，然后G要让式子尽可能的小，让来自于生成分布中的x，D(x)尽可能的接近1。当$P_G(x)=P_{data}(x)$时，G是最优的。

- 训练



把G的loss function修改为$min V=-\frac {1}{m}\sum^m_{i=1}log(D(x^i))$，这样可以提高训练的速度。

N-Gram模型

- Reference

  基本内容https://zhuanlan.zhihu.com/p/32829048

语言模型：判断一句话是否合理

- 定义：这是一种基于统计语言模型的算法。它的基本思想是将文本里面的内容按照字节进行大小为N的滑动窗口操作，形成了长度是N的字节片段序列。每一个字节片段称为gram，对所有gram的出现频度进行统计，并且按照事先设定好的阈值进行过滤，形成关键gram列表，也就是这个文本的向量特征空间，列表中的每一种gram就是一个特征向量维度。
- 假设：第N个词的出现只与前面N-1个词相关，而与其它任何词都不相关，整句的概率就是各个词出现概率的乘积。
- N的大小对性能的影响

n更大的时候
对下一个词出现的约束性信息更多，更大的辨别力，但是更稀疏，并且n-gram的总数也更多，为V^n个（V为词汇表的大小）

n更小的时候

在训练语料库中出现的次数更多，更可靠的统计结果，更高的可靠性，但是约束信息更少

GloVe

- Reference

方法简介http://menc.farbox.com/machine-learning/2017-04-11

论文介绍https://blog.csdn.net/qq_38151401/article/details/97761667

- One-hot的优缺点：

  优点：

  1.至少让word 有了一个向量表示，让语言可以计算

  2.one hot encoding 很稀疏，在一些对稀疏数据友好的问题（如部分分类问题）有比较不错的效果

  缺点：

  1.假设太强，许多NLP task不适用

  2.对大词库的language建模，向量维度太大，难以储存

  3.无法加入新词

在One hot encoding的缺点中，无法加入新词的原因是，一个维度只表达一个word的信息，一个word的信息也只由一个维度表达。

- GloVe结合了以下两个主要模型族的优点:全局矩阵分解和局部上下文窗口方法。我们的模型只训练单词单词共现矩阵中的非零元素，而不是整个稀疏矩阵或单个上下文窗口的大型语料库，从而有效地利用统计信息。

  GloVe是一种word embedding方法（word embedding即词嵌入，将word的语义信息嵌入fix-length的向量中）

  单词矢量学习的合适起点应该是共现概率的比值，而不是概率本身

- 最一般的模型形式

  $F(w_i,w_j,\tilde{w}*k)=*frac{P*{ik}}{P_{jk}}$

  其中，$w\in R^d$是单词向量，$\tilde{w}$是独立的上下文单词向量，F可能依赖于一些尚未制指定的参数。

  $F=w_i^T\tilde{w}*k=log(P*{ik})=log(X_{ik})-log(X_i)$

  其中，$X*{ij}$ *单词j在单词i上下文中出现的次数列表,$X_i=\sum_kX*{ik}$任何单词在单词i上下文中出现的次数,$P*{ij}=P(j|i)=\frac {X*{ij}}{X_i}$单词j出现在单词i上下文中的概率。

  weighted square loss：

  $J=\sum^V*{i,j=1}f(X*{ij})(w_i^T\tilde{w}*j+b_i+*tilde{b}*j-logX_{ij})^2$

  其中,b为各个单词对应的bias

f的要求：

1. 如果f是连续的函数，它应该在x→0时快速消失，足够使$lim_{x\to 0}f(x)log^2x$是有限的。
2. f (x)应该是不递减的，这样就不会出现罕见的共现。
3. 对于较大的x值，f (x)应该相对较小，这样就不会对频繁出现的共存项进行加权。

常用$f(x)$:

$$f(x) = \begin{cases} (x/x_{\max})^{\alpha} & \text{if } x < x_{\max} \\ 1 & \text{otherwise} \end{cases} \quad (9)$$

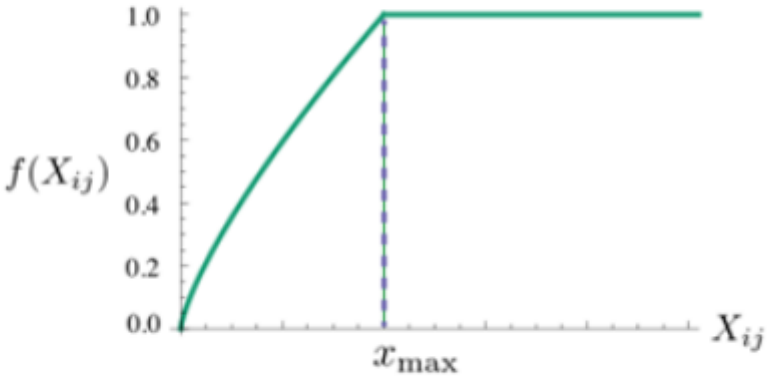


Figure 1: Weighting function f with $\alpha = 3/4$.

NLP 基础

Reference

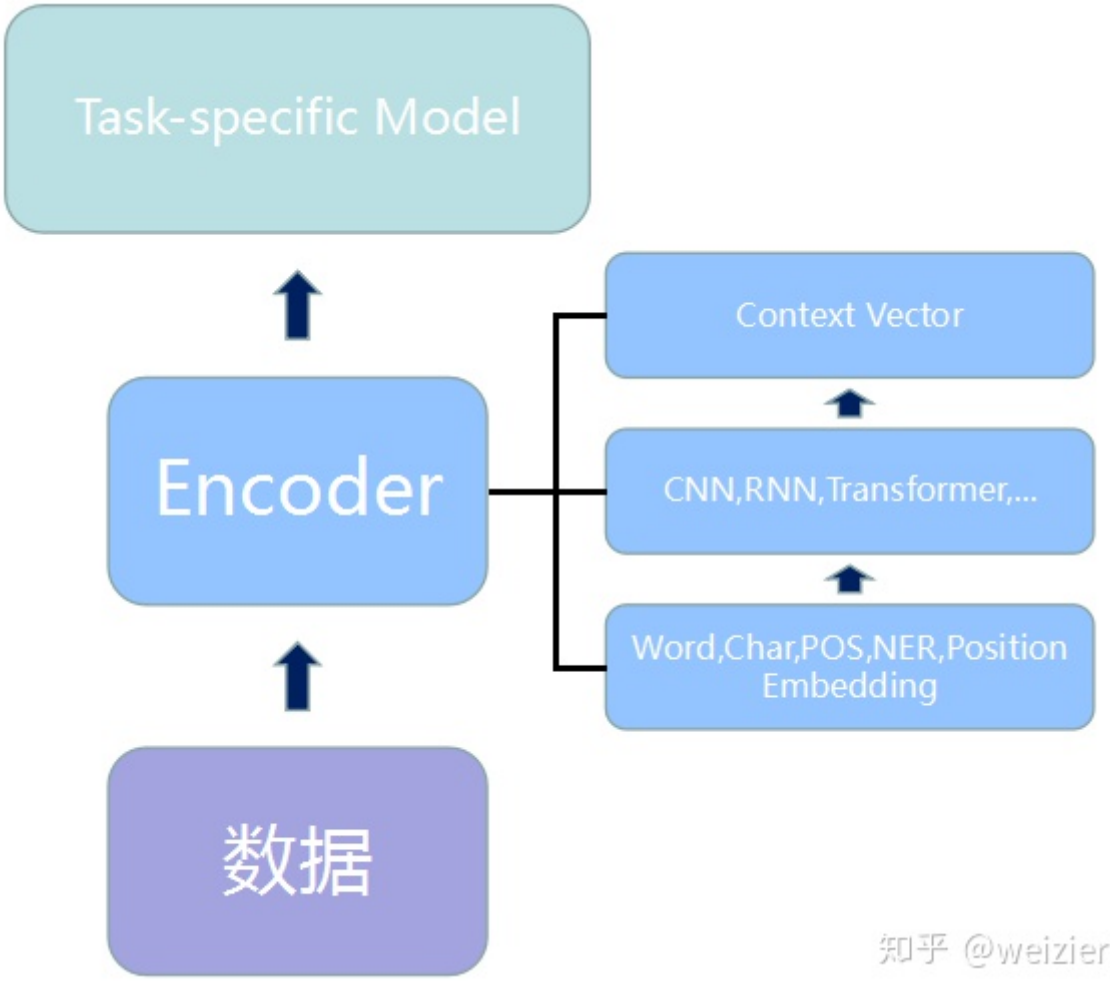
- NLP概述https://zhuanlan.zhihu.com/p/50443871

通常来说，NLP中监督任务的基本套路都可以用三个积木来进行归纳：

1.文本数据搜集和预处理

2.将文本进行编码和表征

3.设计模型解决具体任务



语言模型的本质是对一段自然语言的文本进行预测概率的大小，即如果文本用S_i来表示，那么语言模型就是要求 $P(S_i)$ 的大小。

word2vec算法本身其实并不是一个深度模型，它只有两层全连接

一般来说，残差结构能让训练过程更稳定

把这种先经过CNN得到词向量，然后再计算Softmax的方法叫做CNN Softmax，而利用CNN解决有三点优势值得注意：

第一是，CNN能够减少普通做Softmax时全连接层中的必须要有的|V|*h的参数规模，只需要保持CNN内部的参数大小即可，而一般来说，CNN中的参数规模都要比|V|*h的参数规模小得多；

另一方面，CNN可以解决OOV （Out-of-Vocabulary）问题，这个在翻译问题中尤其头疼；

最后一方面，在预测阶段，CNN对于每一个词向量的计算可以预先做好，更能够减轻inference阶段的计算压力。

预训练语言模型的优势在于

1.近乎无限量的优质数据

2.无需人工标注

3.一次学习多次复用

4.学习到的表征可在多个任务中进行快速迁移

NetVLAD

- Reference

  VLAD vs NetVLAD vs NeXtVLAD https://zhuanlan.zhihu.com/p/96718053

  VLAD与其PyTorch实现https://zhuanlan.zhihu.com/p/148401141

  VLAD相关应用论文https://blog.csdn.net/qq_24954345/article/details/86176862

- VLAD Vector of Local Aggregated Descriptors

  图像特征提取方法的一种

  图像检索经典思路：

  1. 存在一个图像库I，对每张图片I_i通过特征函数提取特征 $f(I_i)$
  2. 提供一张query图片q ，通过特征函数提取特征$f(q)$
  3. 将query特征$f(q)$ 与图库特征$f(I)$做相似度计算，一般为欧式距离:$d(q,I)=||f(q)-f(I)||$ ，距离越小，越相似

  f_{vlad}是将一个若干局部特征压缩为特定大小全局特征的方法。

Vlad计算流程：

1. 对全部的N×D特征图进行K-means聚类，获得K个聚类中心，记为 C_k
2. 通过以下公式，将N×D的局部特征图转为一个全局特征图V，全局特征图shape为K乘D：

   $V(j,k)=\sum^N_{i=1}a_k(x_i)(x_i(j)-c_k(j)),k\in K,j\in D$

   公式中 x_i 表示第i个局部特征，c_k表示第k个聚类中心，x_i和 c_k 都是D维向量。
   $a_k(x_i)$ 是一个符号函数:

   $a_k(x_i)=\begin{cases} 0& ifx_i\notin c_k\1& if x_i\in c_k \end{cases}$

- NetVLAD

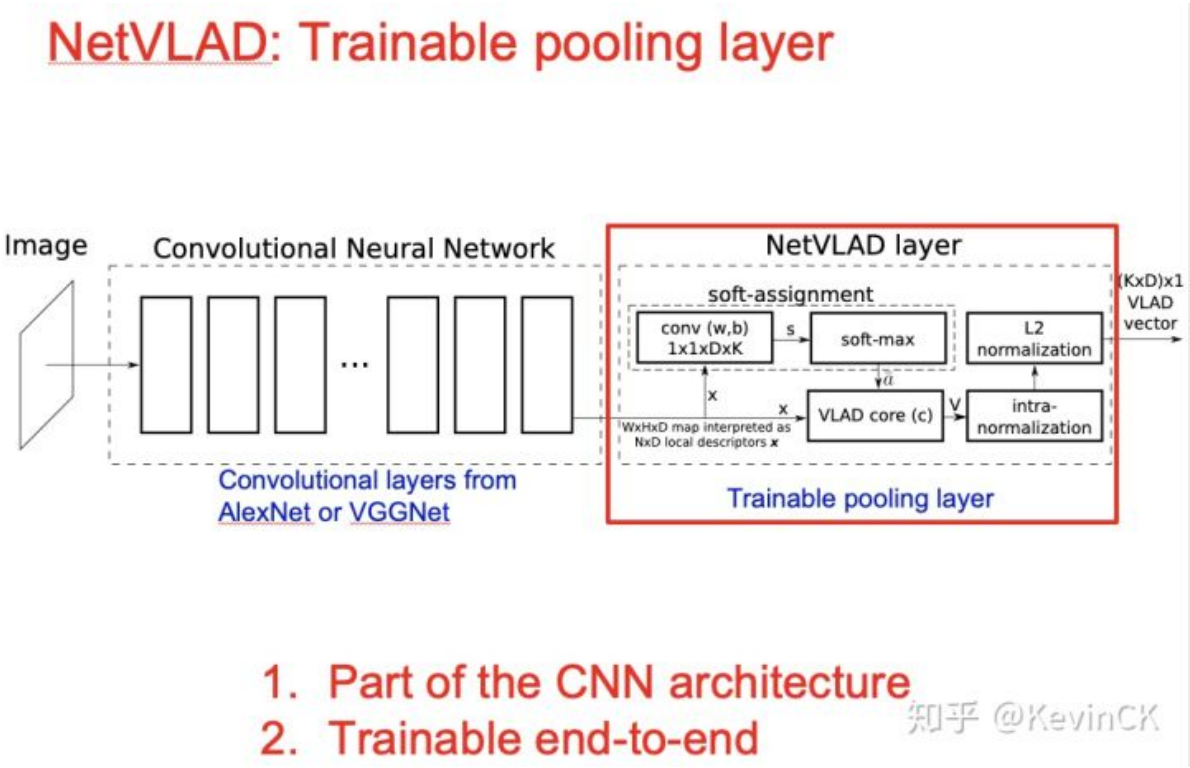
  为了让VLAD函数可导，设计：

  $\overline a(x_i)=\frac {e^{-\alpha ||x_i-c_k||^2}} {\sum*{k\prime}e^{-\alpha ||x_i-c_k||^2}} = \frac {e^{w_k^Tx_i+b_k}}{\sum*{k\prime}e^{w_k^Tx_i+b_k}}(softmax函数形式)\in (0,1)$

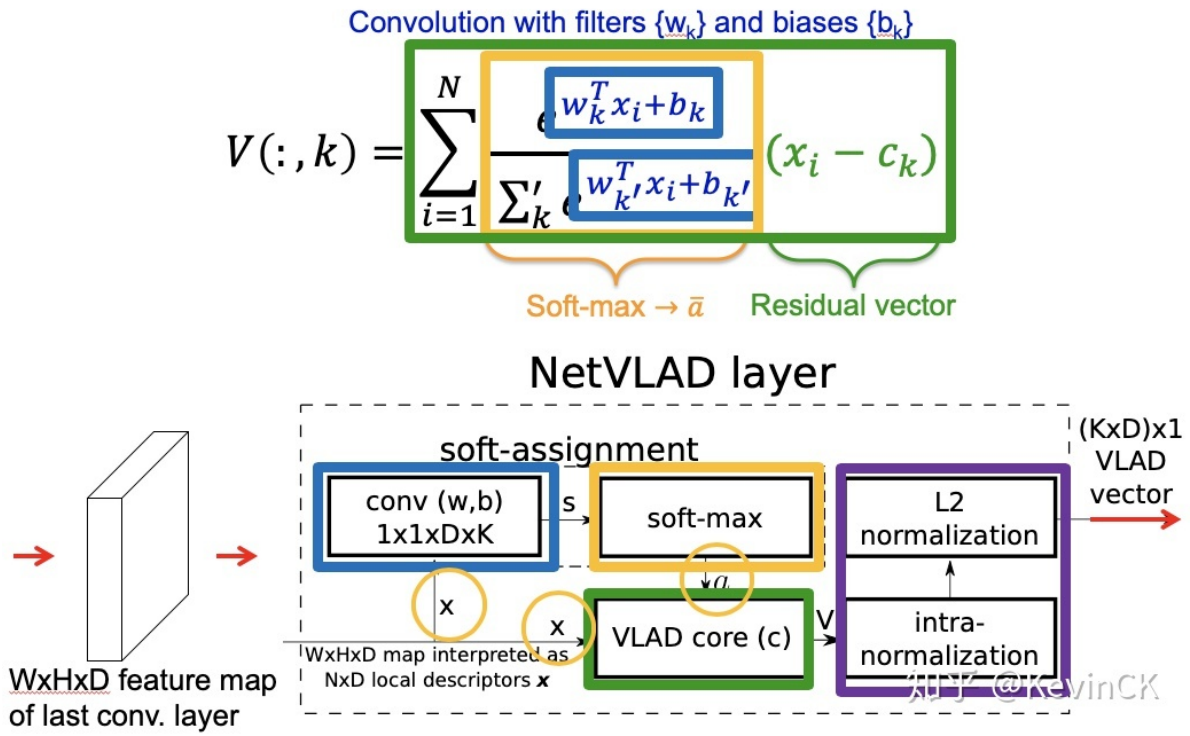
  进而，VLAD函数$V(j,k)=\sum^N*{i=1}\frac {e^{w_k^Tx_i+b_k}}{\sum*{k\prime}e^{w_k^Tx_i+b_k}} ((x_i(j)-c_k(j))$

  其中，w_k,b_k,c_k是NetVLAD需要学习的参数

从VLAD到NetVLAD的最大变化是之前需要通过聚类获得参数 c_k 变成了需要通过训练得到。这样就可以把VLAD变成了一个分类问题，即设定有K个分类，计算局部特征在这K个分类的差值分布来得到全局特征V(j,k)。



1. Part of the CNN architecture
2. Trainable end-to-end



从NxD到KxD的转化公式$w_k^T*x_i+b_k$是通过1x1卷积实现（蓝色部分）；
黄色部分是softmax公式，通过softmax函数实现；
绿色部分是局部特征与聚类中心的残差分布，通过VLAD core来实现。
紫色部分是两步归一化操作：
intra-normalization：是将每个中心点$k\in K$的特征分别做归一化，通过此操作抹去了聚类中残差的绝对大小，只保留了残差的分布。
L2 normalization：将得到的KxD数据再整体做一次归一化处理。

- NeXtVLAD

NeXtVLAD对NetVLAD的改进总结起来就是增加了VLAD层的非线性参数，但降低了VLAD输出层参数，从而整体参数也降低了。虽然整体参数量下降，但性能并没有下降。其思想类似于ResNet和ResNeXt。

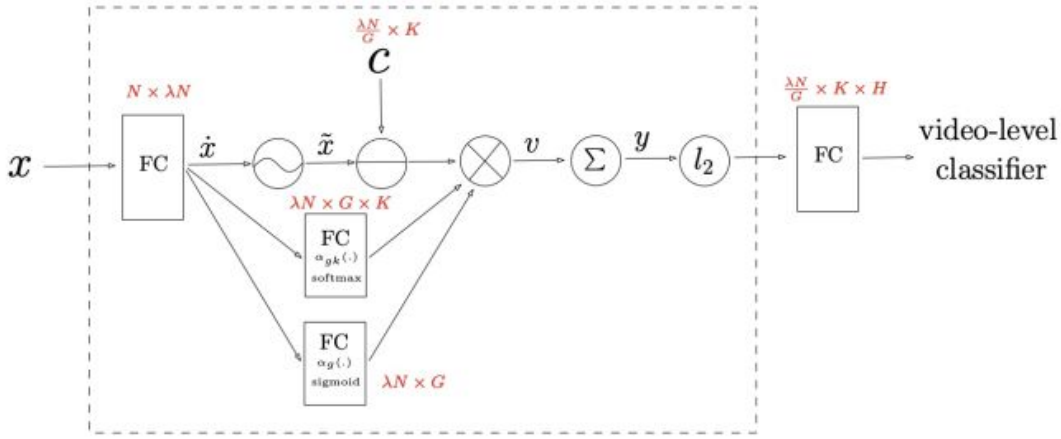


Fig. 2. Schema of our NeXtVLAD network for video classification. Formulas in red denote the number of parameters (ignoring biases or batch normalization). FC represents a fully-connected layer. The wave operation means a reshape transformation.

光流：根据相邻帧之间的变化人为提取的特征信息，并且以2D图像的方式表示出来

FPN

- FPN原理https://zhuanlan.zhihu.com/p/92005927

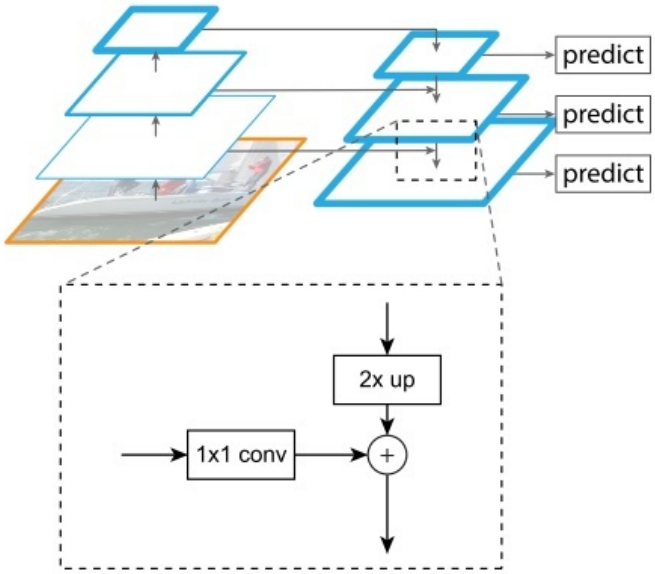


Figure 3. A building block illustrating the lateral connection and the top-down pathway, merged by addition.

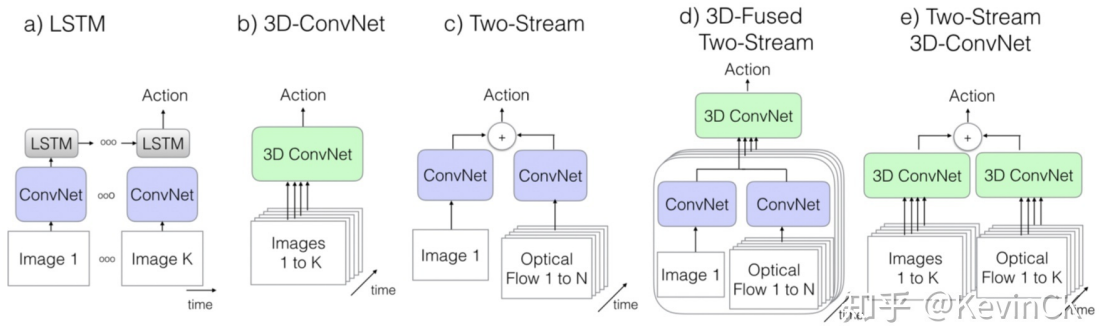
I3D

- reference:

TSN vs C3D vs I3D https://zhuanlan.zhihu.com/p/58355093

原理简介https://blog.csdn.net/u012925946/article/details/99743978

-



(e为提出的i3d模型)

相较于之前模型的提升：

1. 拓展2D卷积网络到3D
2. 将2D滤波器变为3D
3. 时间、空间和网络深度接收增长
4. 两个3D流

图卷积与图卷积神经网络

- Reference

图卷积https://zhuanlan.zhihu.com/p/89503068

图卷积神经网络https://zhuanlan.zhihu.com/p/91573076

- 图卷积

图嵌入：

1. 将图中的节点表示成低维、实值、稠密的向量形式
2. 将整个图表示成低维、实值、稠密的向量形式

图嵌入的主要方式：

1. 矩阵分解：节点间关系矩阵表达
2. Deepwalk 基于word2vec
3. Graph Neural Network (GNN) 图神经网络 可以应用于图嵌入来得到图或图节点的向量表示

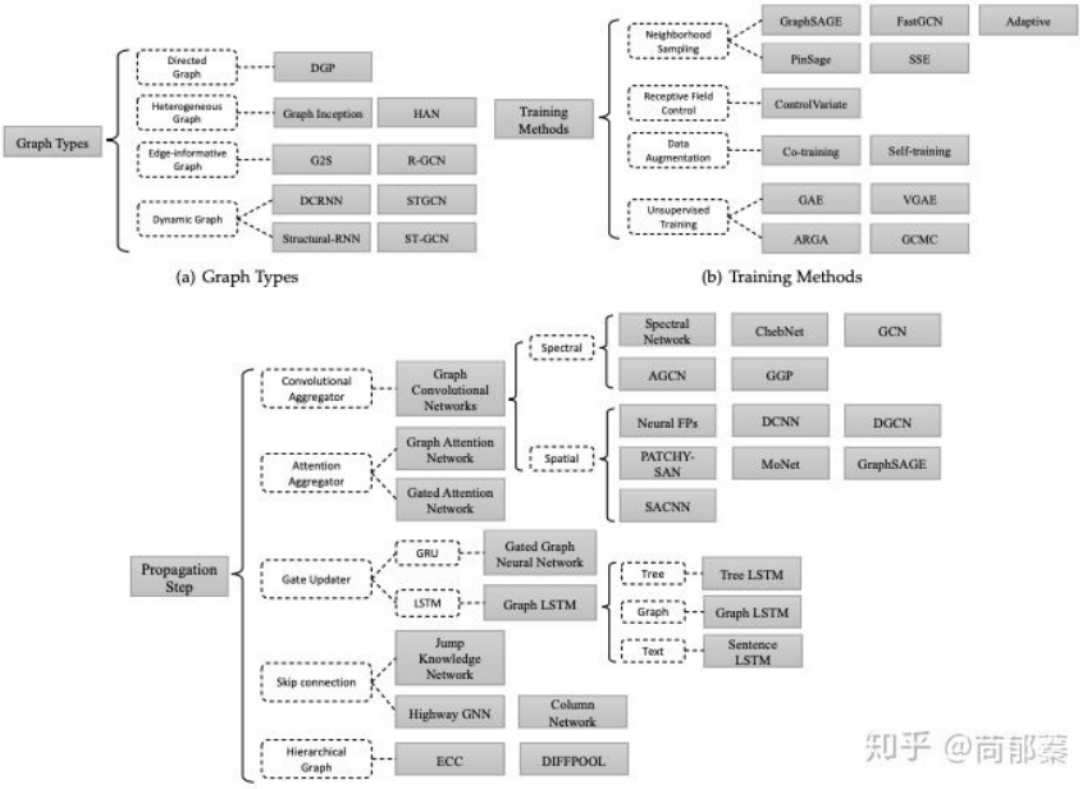


图1 图神经网络GNN的分类：分别从图的类型，训练的方式，传播的方式三个方面来对现有的图模型工作进行划分

用随机的共享的卷积核得到像素点的加权和从而提取到某种特定的特征，然后用反向传播来优化卷积核参数就可以自动的提取特征，是CNN特征提取的基石。

图卷积的通式：

任何一个图卷积层都可以写成这样一个非线性函数：

$H^{l+1}=f(H^l,A)$

$H^0=X$ 为第一层的输入，$X\in R^{N*D} $ ， N 为图的节点个数， D 为每个节点特征向量的维度， A 为邻接矩阵，不同模型的差异点在于函数 f 的实现不同。下面每一种实现的参数都统称拉普拉斯矩阵

实现一 $H^{l+1}=\sigma (AH^lW^l)$

其中，W^l为第l层的权重参数矩阵，$\sigma(\cdot)$为非线性激活函数

这样存在两个问题：

1.没有考虑节点自身对自己的影响；

2.邻接矩阵 A 没有被规范化，这在提取图特征时可能存在问题，比如邻居节点多的节点倾向于有更大的影响力。

实现二 $H^{l+1}=\sigma(LH^lW^l)$ 拉普拉斯矩阵L=D-A

解决了没有考虑自身节点信息自传递的问题

实现三 $H^{l+1}=\sigma(D^{-\frac{1}{2}}\hat{A}D^{-\frac{1}{2}}H^lW^l)$

拉普拉斯矩阵 $L^{sym}=I_n-D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$

解决自传递问题；

对邻接矩阵的归一化操作，通过对邻接矩阵两边乘以节点的度开方然后取逆得到

- 具体到每一个节点对 i， j，矩阵中的元素由下面的式子给出（对于无向无权图）：

$$L_{i,j}^{\text{sym}} := \begin{cases} 1 & \text{if } i = j \text{ and } \deg(v_i) \neq 0 \\ -\frac{1}{\sqrt{\deg(v_i)\deg(v_j)}} & \text{if } i \neq j \text{ and } v_i \text{ is adjacent to } v_j \\ 0 & \text{otherwise.} \end{cases}$$

其中 $deg(v_i), deg(v_j)$ 分别为节点 i, j 的度，也就是度矩阵在节点 i, j 处的值。

从单个节点角度看，对于第 $l+1$ 层的节点的特征 h_i^{l+1}，对于它的邻接节点 $j \in N$，N 是节点 i 的所有邻居节点的集合，可以通过以下公式计算得到：

$$h_{v_i}^{l+1} = \sigma(\sum_j \frac{1}{c_{ij}} h_{v_j}^l W^l)$$

其中，$c_{i,j} = \sqrt{d_i d_j}$，$j \in Ni$，Ni 为 i 的邻居节点，d_i, d_j 为 i, j 的度。

- 图卷积神经网络

图卷积神经网络GCN属于图神经网络GNN的一类，是采用卷积操作的图神经网络，可以应用于图嵌入GE。

CNN具有以下几个特点：

1.权重共享：同一个卷积核可以作用于不同的位置。

2.局部性：欧式空间可以简洁的支持卷积核（直接根据卷积公式计算卷积结果即可），卷积核的大小一般远小于输入信号的大小。

3.多尺度：CNN往往包含下采样，可以减少参数，并获得更大的感受野（receptive field）。

提出了两种方式构建图卷积神经网络：空域构建（Spatial Construction）和频域构建（Spectral Construction）。

1.空域构建

主要考虑CNN多尺度、层次、局部感受野的特点

定义一个有权重的无向图 $G = (\Omega, W)$，这里 Ω 表示结点，大小为 m，W 表示边，大小为 $m \times m$，是一个对称的非负矩阵。因此结点 j 的相邻结点可以表示为：

$$N_\delta = \{i \in \Omega : W_{ij} > \delta\}$$

对图中结点v做卷积，其实就是对结点v的相邻结点做加权求和

卷积的输入的结点特征可能是一个向量，维度记为 f_{k-1}。同时，一次卷积操作可能包含多个卷积核，即卷积的通道数 f_k。类似与图像中的卷积，对输入特征的每一维做卷积，然后累加求和，就可以得到某个通道 j 的卷积结果，公式如下所示：（x表示每个结点的特征，w表欧式卷积的权重，对于某个结点v做卷积）

$$o_{v,j} = h(\sum_{i=1}^{f_{k-1}} \sum_{u \in N_\delta[v]} w_{i,j,u,v} x_{u,i}) \quad (j = 1...f_k)$$

转化成矩阵形式，可以写成：

$$o_j = h(\sum_{i=1}^{f_{k-1}} F_{i,j} x_i)$$

$$F_{i,j}(u,v) = \begin{cases} w_{i,j,u,v}, & v \in N_\delta[u] \\ 0, & else \end{cases}$$

多层空域卷积

$$N_k = \{N_{k,i}; i = 1...d_{k-1}\}$$

$$o_{k,j} = h(\sum_{i=1}^{f_{k-1}} F_{k,i,j} x_{k,i})$$

输出结果 x_{k+1} 可以定义为：

$$x_{k+1,j} = L_k h(\sum_{i=1}^{f_{k-1}} F_{k,i,j} x_{k,i})$$

Ω_k和 N_k 可以通过以下方式构建：

$$W_0 = W$$
$$A_k(i,j) = \sum_{s \in \Omega_k(i)} \sum_{t \in \Omega_k(j)} W_{k-1}(s,t)$$
$$W_k = rownormalize(A_k), (k \le K)$$
$$N_k = supp(W_k), (k \le K)$$

2.频域构建

实现原理 $f*_G g=F^{-1}(F(f)\cdot F(g))$

F 傅里叶变换

F^{-1} 傅里叶反变换

度数矩阵 D (Degree matrix):

$$D(i,j) = \begin{cases} d_i & if \quad i == j \\ 0 & otherwise \end{cases}$$

邻接矩阵 A (Adjacency matrix):

$$A(i,j)=\begin{cases}1 & if\ i\in N_\delta[j]\\ 0 & otherwise\end{cases}$$

那么拉普拉斯矩阵L可以写成是:

$$L=D-A$$

这里我们使用的是无向图，所以L是对称矩阵。因此 L 可以按照如下公式进行分解：

$$L=U\Lambda U^T$$

$$U=[u_1,u_2,\dots,u_n]$$

$$\Lambda=\begin{vmatrix}\lambda_1 & \dots & 0\\ \dots & \dots & \dots\\ 0 & \dots & \lambda_n\end{vmatrix}$$

这里特征向量 U 就是我们要找的一组傅里叶变换的基，特征值 Λ 是对应的特征向量所占的比重。通过这组傅里叶变换的基，我们可以将输入 x_i 从空域转到频域， $F(x_i)=U^Tx_i$ 。

因此，图的卷积公式，从空域转换到频域，可以表示为：

$$x_i*_G F_{i,j}=U(U^Tx_i\cdot U^TF_{i,j})=U(U^TF_{i,j}\cdot U^Tx_i)$$

这里可以将 $U^TF_{i,j}$ 整体看成一个可以学习的卷积核，记为 $F_{i,j}^\theta$ ，因此最终的卷积公式可以表示成：

$$x_i*_G F_{i,j}=UF_{i,j}^\theta U^Tx_i$$

加入通道和激励函数后，第 k 层卷积可以写成：

$$o_{k,j}=h(\sum_{i=1}^{f_{k-1}}UF_{k,i,j}^\theta U^Tx_{k,i})=h(U\sum_{i=1}^{f_{k-1}}F_{k,i,j}^\theta U^Tx_{k,i})$$

这里 $F_{k,i,j}^\theta$ 是一个对角矩阵：

$$F_{k,i,j}^\theta=\begin{bmatrix}\theta_1 & \dots & 0\\ \dots & \dots & \dots\\ 0 & \dots & \theta_N\end{bmatrix}$$

通常，我们根据特征值的大小选择前 d 个特征向量作为一组傅里叶变换的基。

AlexNet

- AlexNet简介 https://blog.csdn.net/luoluonuoyasuolong/article/details/81750190
- 论文名称《ImageNet Classification with Deep Convolutional Neural Networks》
- 网络结构

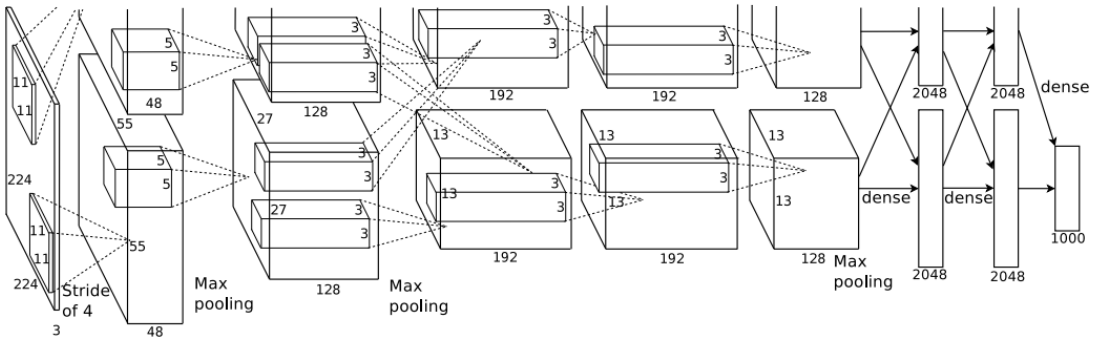


Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network's input is 150,528-dimensional, and the number of neurons in the network's remaining layers is given by 253,440–186,624–64,896–64,896–43,264–4096–4096–1000.

- 标准的L-P神经元的输出一般使用tanh 或 sigmoid作为激活函数, $tanh(x)=\frac {sinhx} {conshx}=\frac {e^x-e^{-x}} {e^x+e^{-x}}$, sigmoid$f(x)=\frac {1}{1+e^{-x}}$.但是这些饱和的非线性函数在计算梯度的时候都要比非饱和的现行函数$f(x)=max(0,x)$慢很多，在这里称为Rectified Linear Units(ReLUs)。在深度学习中使用ReLUs要比等价的tanh快很多。
- Local Response Normalization (LRN，局部响应归一化)

  效果：对ReLU得到的结果进行归一化

公式：

$$b^i_{(x,y)} = \frac{a^i_{(x,y)}}{(k + \alpha \sum\limits_{j=max(0,i-n/2)}^{min(N-1,i+n/2)} (a^j_{(x,y)})^2)^\beta}$$

这个公式什么意思呢？$a^i_{(x,y)}$代表的是ReLU在第i个kernel的(x, y)位置的输出，n表示的是$a^i_{(x,y)}$d的邻居个数，N表示该kernel的总数量。$b^i_{(x,y)}$表示的是LRN的结果。ReLU输出的结果和它周围一定范围的邻居做一个局部的归一化，怎么理解呢？我觉得这里有点类似域我们的最大最小归一化，假设有一个向量$X = [x_1, x_2, ..., x_n]$
那么将所有的数归一化到0-1之间的归一化规则是：$x_i = \frac{x_i - x_{min}}{x_{max} - x_{min}}$。

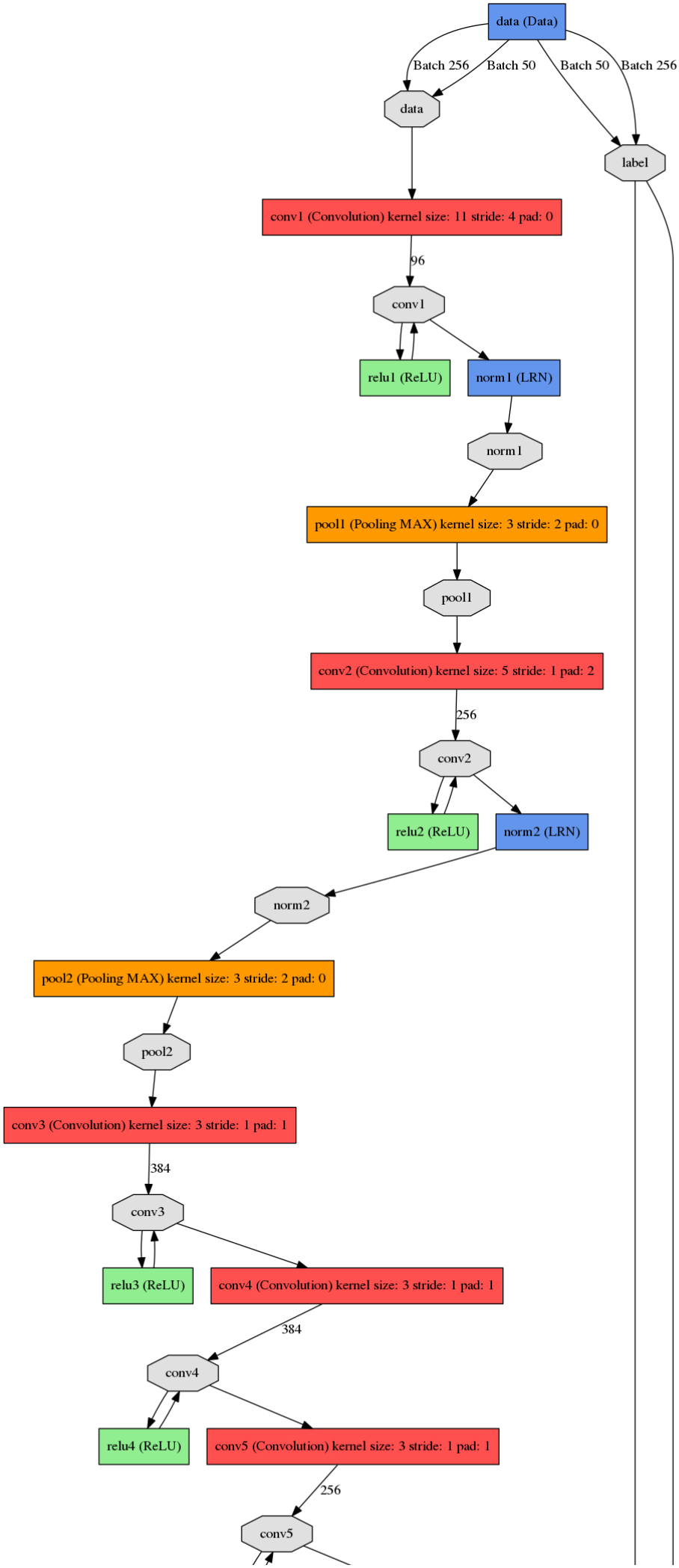
论文中，$k=2,n=5,\alpha=10^{-4},\beta=0.75$

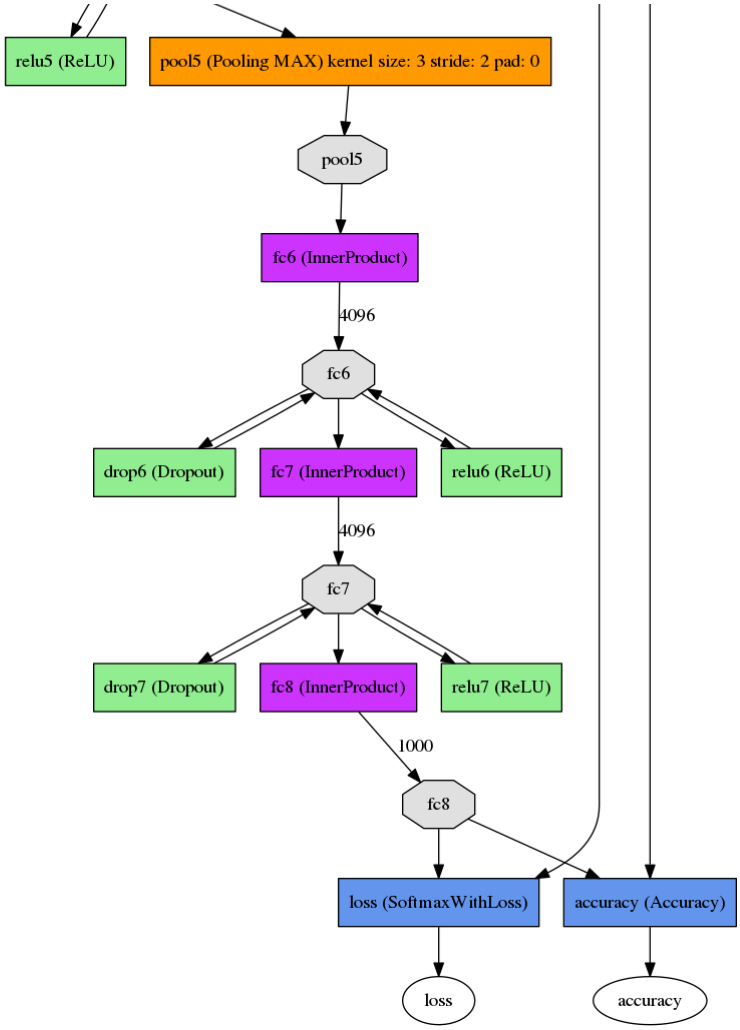
使用了数据扩充(data augmentation 对原始数据做一些变换)和dropout方法，防止过拟合

- 覆盖的池化 overlapping pooling

  stride<pool_size 类似于convolutional化 提高准确度，减少error rate

- 整体结构


data (Data)
Batch 256
Batch 50
Batch 50
Batch 256
data
label
conv1 (Convolution) kernel size: 11 stride: 4 pad: 0
96
conv1
relu1 (ReLU)
norm1 (LRN)
norm1
pool1 (Pooling MAX) kernel size: 3 stride: 2 pad: 0
pool1
conv2 (Convolution) kernel size: 5 stride: 1 pad: 2
256
conv2
relu2 (ReLU)
norm2 (LRN)
norm2
pool2 (Pooling MAX) kernel size: 3 stride: 2 pad: 0
pool2
conv3 (Convolution) kernel size: 3 stride: 1 pad: 1
384
conv3
relu3 (ReLU)
conv4 (Convolution) kernel size: 3 stride: 1 pad: 1
384
conv4
relu4 (ReLU)
conv5 (Convolution) kernel size: 3 stride: 1 pad: 1
256
conv5



Stanford CS221

Stanford CS221 2019秋 https://www.bilibili.com/video/BV16E411J7AQ?p=1

官方笔记https://stanford.edu/~shervine/teaching/cs-221/cheatsheet-reflex-models

- L1 Overview

  梯度下降GD代码

```
points用于放入需要计算的点points =[(2,4),(4,2)]  #以这两点之间计算最小二乘进行regression为例。

def F(w):
    return sum((w*x-y)**2 for x,y in points) #该函数为最小二乘实现
def dF(w):
    return sum(2*(w*x-y)*x for x,y in points) #计算导数

# Gradient descent
w=0
eta=0.01 #步长
for t in range(100):
    value=F(w)
    gradient=dF(w)
    w=w-eta*gradient
```

```
    print('iteration{}:w = {}, F(w) = {}'.format(t,w,value))
```

- L2 Machine Learning 1 - Linear Classifiers & SGD

  Feature vector

  $\phi(x)=[\phi_1(x),...,\phi_d(x)]$ for an input x

  Think of $\phi(x)\in R^d$: as a point in a high-dimensional space

  Weight vector

  $w\in R^d$

  (binary) linear classifier

  $f_w(x)=sign(\boldsymbol w\cdot \phi(x))=\begin{cases} +1& if\boldsymbol w\cdot \phi(x)>0\\-1& if \boldsymbol w\cdot \phi(x)<0 \\ ?& if\boldsymbol w\cdot \phi(x)=0\end{cases}$

  Loss function for linear classifier (output is y) $Loss(x,y,\boldsymbol w)$ -> the distance between our prediction and y -> loss ↓ result ↑

  (Confidence) score for linear classifier $ \boldsymbol w\cdot \phi(x)$ ->how confident we are in predicting y

  margin for linear classifier$(\boldsymbol w \cdot \phi(x))y$ -> how correct we are

  margin<0(if y<0)->error

  zero-one loss

  $Loss_{0-1}(x,y,\boldsymbol w)=\bold 1 [f_w(x)\ne y]=\bold 1[(\boldsymbol w\cdot \phi(x))y\leq0]$

  Note:$\bold 1(x)$ means indicator function: if True print 1;else print 0

  linear regression

  residual $\boldsymbol w \cdot \phi(x)-y$

  square loss $Loss_{squard}(x,y,\boldsymbol w)=(f_w(x)-y)^2$

  absolute deviation loss $Loss_{absdev}(x,y,\boldsymbol w)=|\boldsymbol w\cdot \phi(x)-y|$

  train loss: the loss on the entire dataset

the gradient $\triangledown_wTrainLoss(\boldsymbol w)$ is the direction that increases the loss the most

$TrainLoss(w)=\frac{1}{|D_{train}|}\sum_{(x,y)\in D_{train}}Loss(x,y,\boldsymbol w)$

Algorithm:gradient desents

Initialize $\boldsymbol w=[0,0,,,0]$

For $t=1,...,T:$

$\boldsymbol w \gets w-\eta \triangledown_wTrainLoss(\boldsymbol w)$

Stochastic gradient descent(SGD):

For each$(x,y)\in D_{train}$:(stochastic updates)

$w \gets w-\eta \triangledown_wLoss(x,y,\boldsymbol w)$

(step size η)

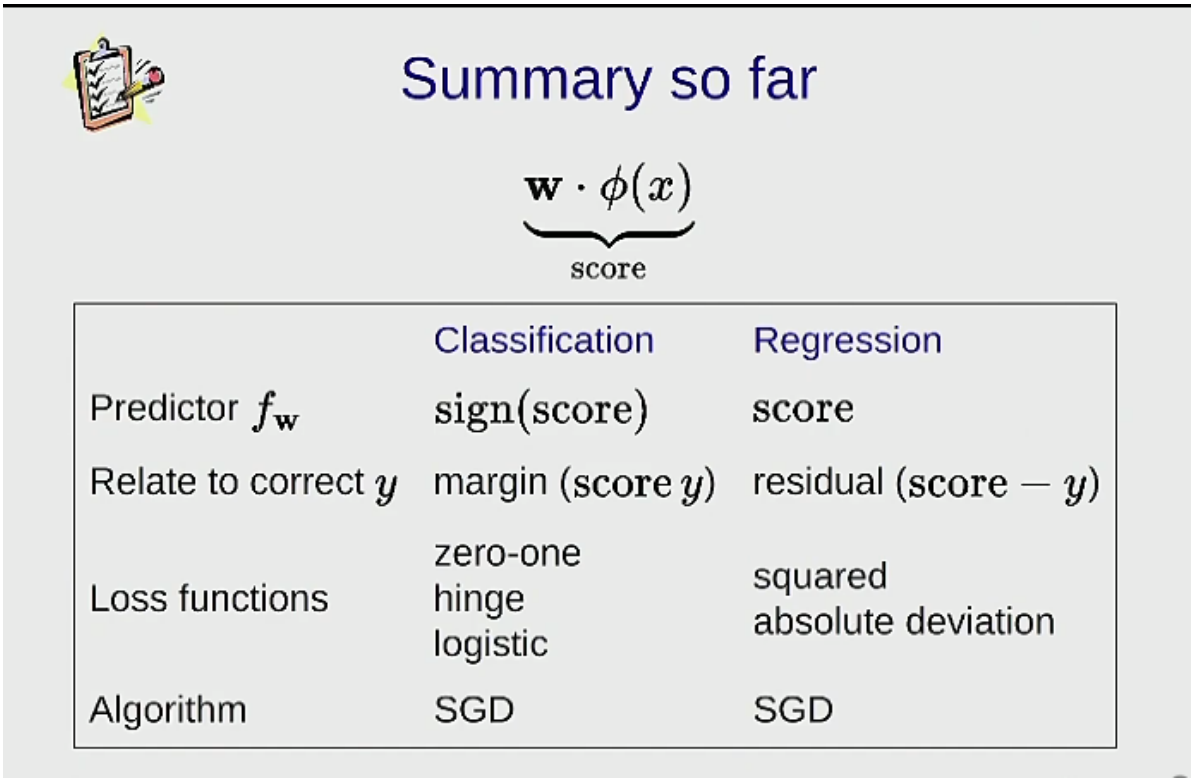
Code related with SGD:

```
import numpy as np

#Modeling: what we want to compute
#points=[(np.array([2]),4),(np.array([4]),2)]
#d=1  #output

#Generate artificial dataset ->for testing
true_w=np.array([1,2,3,4,5])  #gt weight
d=len(ture_w)
points=[]
for i in range(10000):
    x=np.random.randn(d)
    y=ture_w.dot(x)+np.randn()  #add noise to make it more robust
   # print(x,y)
    points.append((x,y))
def F(w):
        return sum((w.dot(x)-y)**2 for x,y in points)/len(points) #该函数为最小二乘实现
def dF(w):
        return sum(2*(w.dot(x)-y)*x for x,y in points)/len(points) #计算导数

def sF(w,i):
        x,y=points[i]
        return sum(w.dot(x)-y)**2
def sdF(w,i):
        return sum2*(w.dot(x)-y)*x
 # Gradient descent
 #Algorithms: how we compute it
def gradientDescent(F,dF,d):
      w=np.zeros(d)
      eta=0.01 #步长
      for t in range(1000):
             value=F(w)
          gradient=dF(w)
          w=w-eta*gradient
          print('iteration{}:w = {}, F(w) = {}'.format(t,w,value))
    #SGD
def stochasticgradientDescent(sF,sdF,d,n):
        w=np.zeros(d)
```

```
            numUpdates=0
        for t in range(1000):
              for i in range(n):
                    value=sF(w,i)
                gradient=sdF(w,i)
                  numUpdates+=1
                 eta=1.0/numUpdates  #update step size
                w=w-eta*gradient
                print('iteration{}:w = {}, F(w) = {}'.format(t,w,value))

gradientDescent(F,dF,d)
stochasticgradientDescent(sF,sdF,d,len(points))
```

Hinge Loss $Loss_{hinge}(x,y,\boldsymbol w)=max\left{1-(\boldsymbol w\cdot \phi(x))y,0\right}$

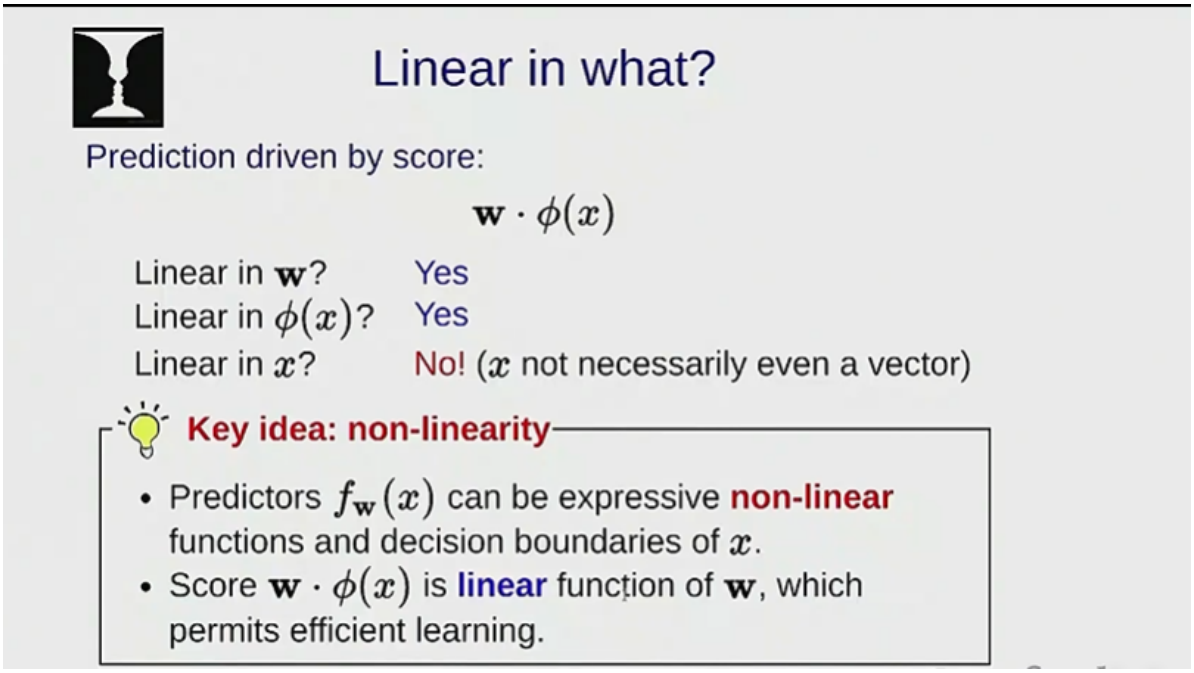
Loss for Logistic regression:$Loss_{logistic}(x,y,\boldsymbol w)=log(1+e^{-(\boldsymbol w\cdot \phi(x))y})$



- L3 Machine Learning 2 - Features & Neural Networks

  feature template: a group of features all computed in a similar way

  hypothesis class: the set of possible predictors with a fixed $\phi(x)$ and a varying w: $F={f_w:w\in R^d}$



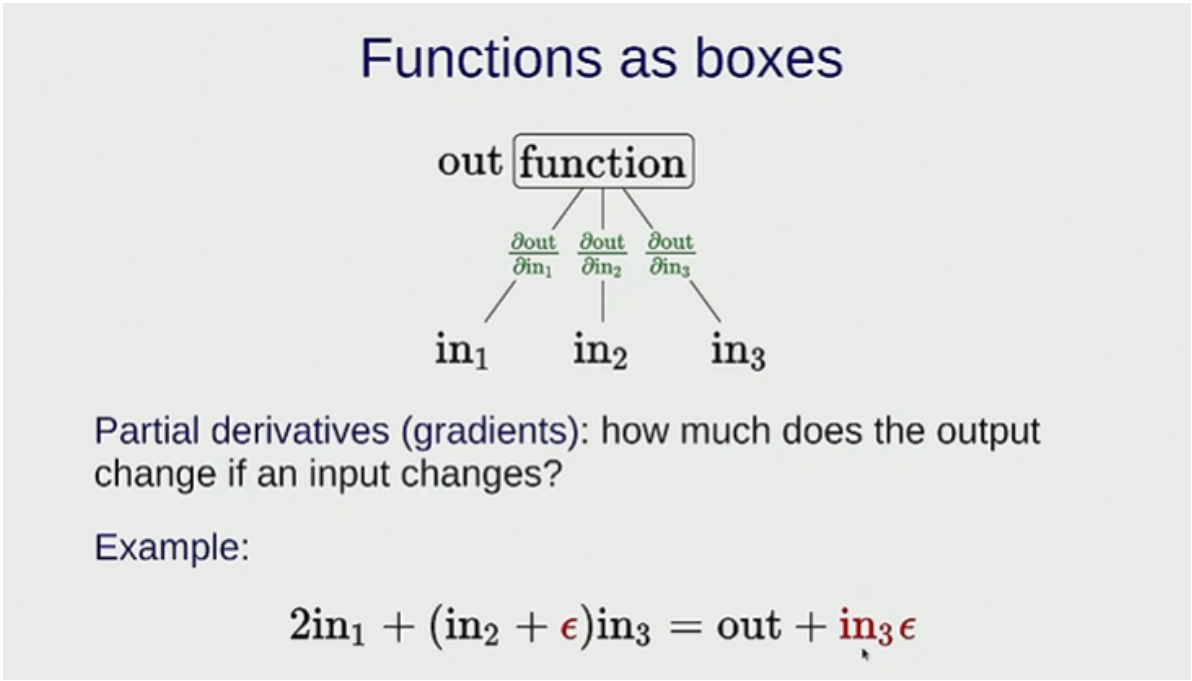
(Prerequisite: Linear Predictor)

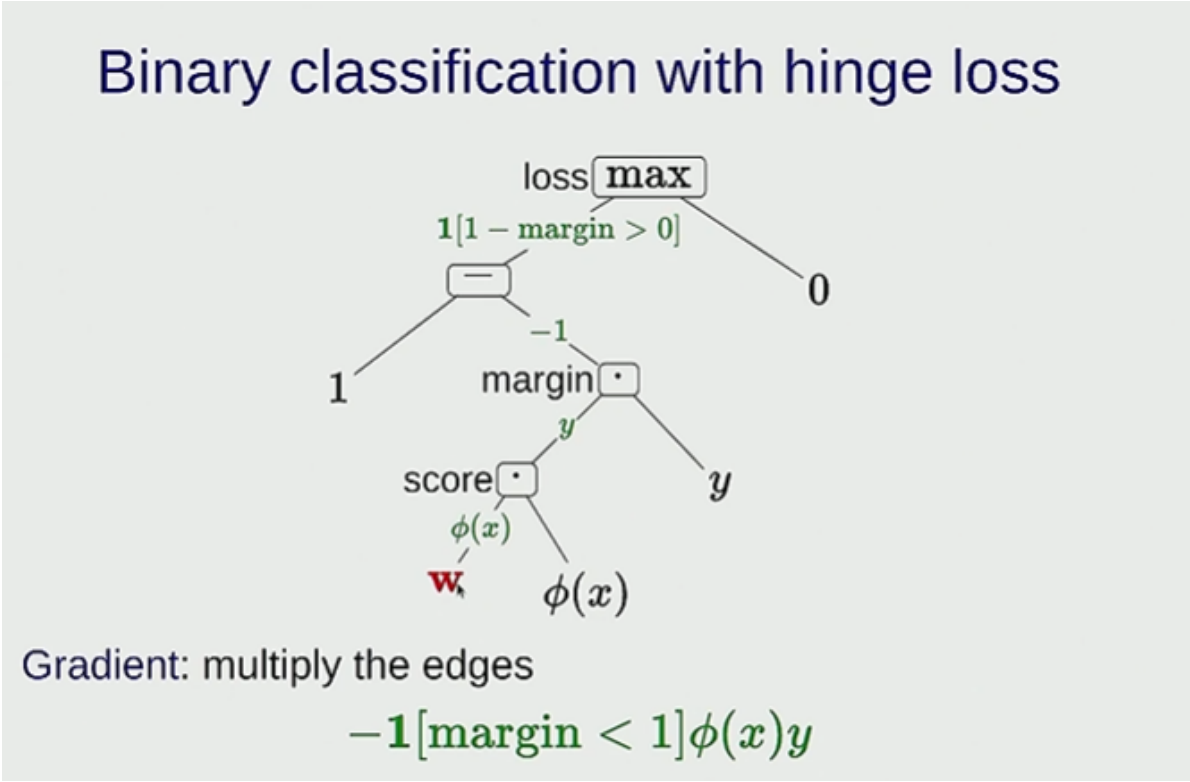
Logistic function: maps$(-\infin,\infin)$to$(0,1)$:

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

Derivative:$\sigma^{\prime}(z)=\sigma(z)(1-(\sigma(x))$

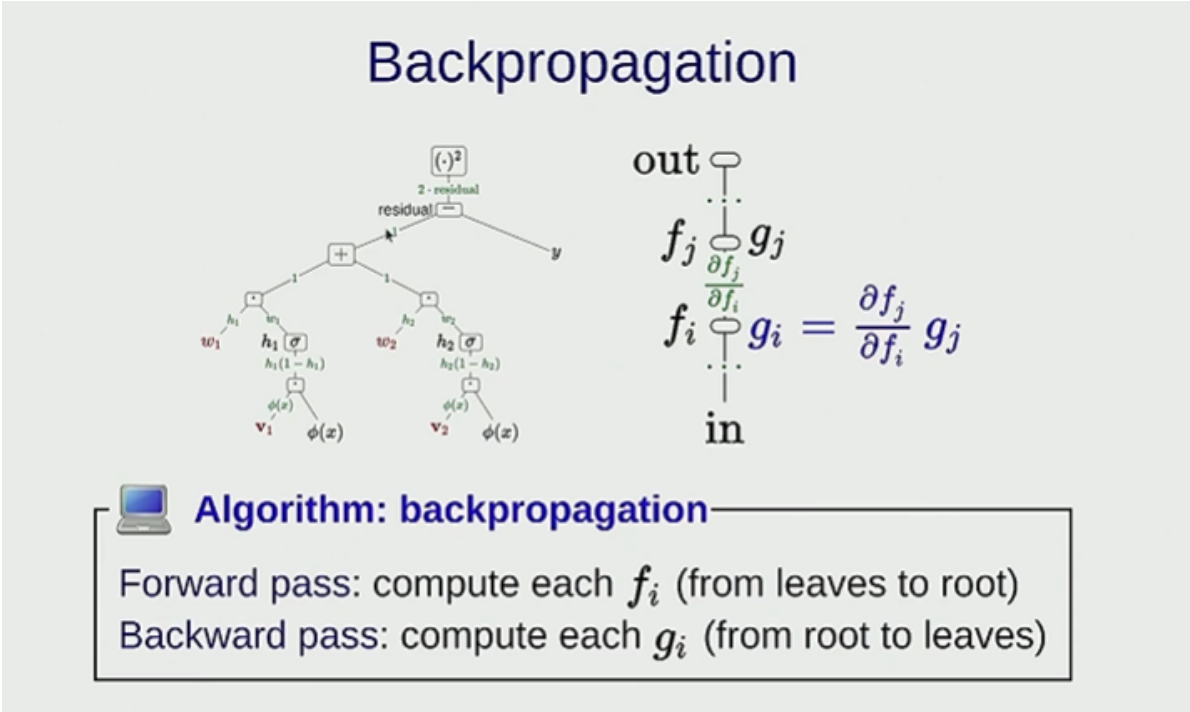
Computational graph:





Forward value: f_i is value for subexpression rooted at i

Backward value: $g_i=\frac {\partial out}{\partial f_i}$ is how f_i influences output



Nearest neighbors:

Training:just store D_{train}

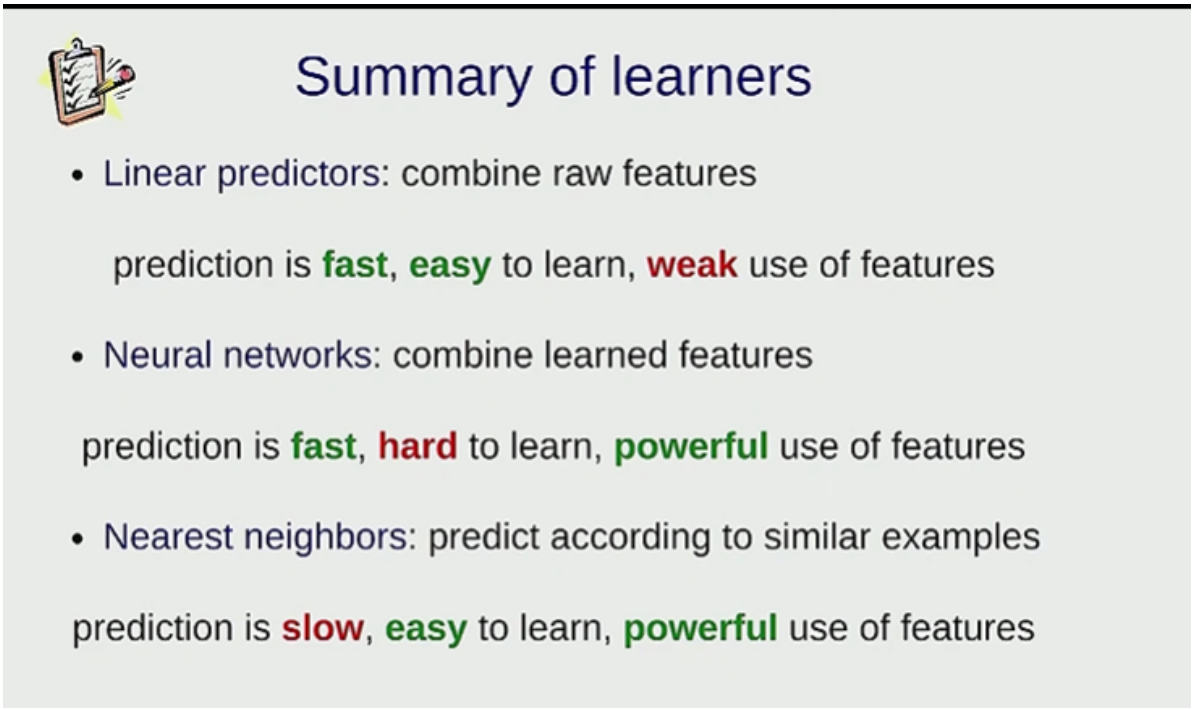
Predictor$f(x^{\prime})$:

1. find $(x,y)\in D_{train}$ where $||\phi(x)-\phi(x^{\prime})||$ is smallest

2. return y

Key idea:similarity : Similar examples tend to have similar outputs

Non-parametric: the hypothesis class adaptes to number of examples



- L4 Generalization K-means

rote learning:

Training:just store D_{train}

Predictor$f(x)$:

If $(x,y)\in D_{train}$: return y

Else:segfault

the real learning objective: to minimize error on unseen future examples

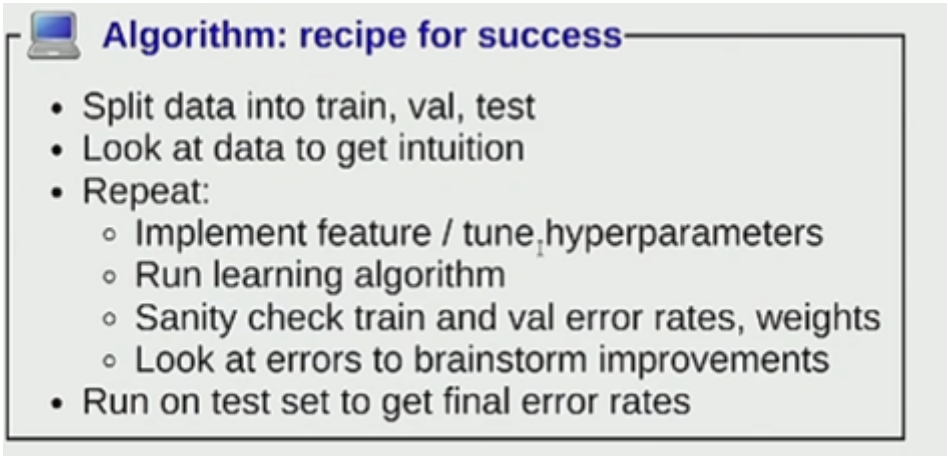
test setD_{test} contains examples not used for training

hyperparameters:properties of the learning algorithm (features, regularization paarameterλ,etc.)

Choose hyperparameters to minimize D_{train} error x->solution would be to include all features

Choose hyperparameters to minimize $D*{test}$ error x->choosing based on $D*{test}$ makes it an unreliable estimate of error

validation set: is taken of the training data which acts as a surrogate for the test set.



Key idea of unsupervised learning: Data has lots of rich latent structures; want methods to discover this structure automatically.

clusting:

input: training set of input points $D_{train}=\lbrace x_1,...,x_n\rbrace$

output:assignment of each point to a cluster$[z_1,...,z_n]$where $z_i\in \lbrace1,...K\rbrace$

K-means cluster:

Setup:

1. each cluster$k=1,...,K$is represented by a centroid $\mu_k\in R^d$
2. intuition:want each point $\phi(x_i)$close to its assigned centroid μ_{z_i}

Objective function/training loss:

$Loss*{kmeans}(z,\mu)=\sum^n*{i=1}||\phi(x_i)-\mu_{z_i}||^2$

Need to choose centroids μ and assignments z jointly

K-means algorithm (Step 1)

Goal: given centroids $\mu_1, \ldots, \mu_K$, assign each point to the best centroid.

Algorithm: Step 1 of K-means

For each point $i = 1, \ldots, n$:

Assign i to cluster with closest centroid:

$$z_i \leftarrow \arg\min_{k=1,\ldots,K} \|\phi(x_i) - \mu_k\|^2.$$

K-means algorithm (Step 2)

Goal: given cluster assignments $z_1, \ldots, z_n$, find the best centroids $\mu_1, \ldots, \mu_K$.

Algorithm: Step 2 of K-means

For each cluster $k = 1, \ldots, K$:

Set μ_k to average of points assigned to cluster k:

$$\mu_k \leftarrow \frac{1}{|\{i : z_i = k\}|} \sum_{i:z_i=k} \phi(x_i)$$

Conclusion:

Objective $min_zmin*mu Loss*{kmeans}(z,\mu)$

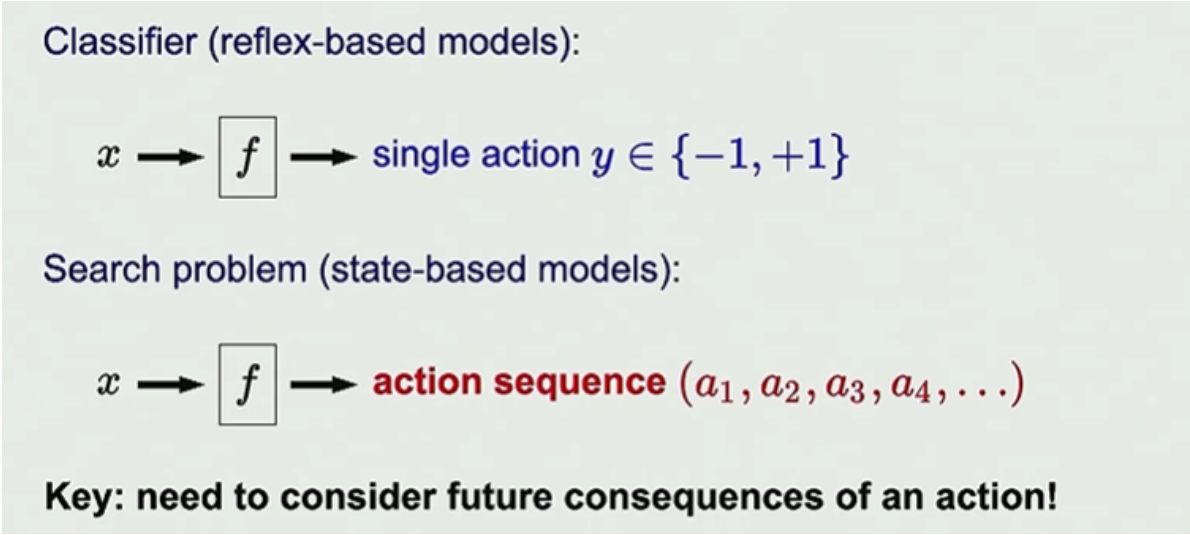
Initialize$\mu_1,...,\mu_K$ randomly.

For $t=1,...,T$:

Step1:set assignments z given μ

Step2:set centroidsμ given z

- L5 Dynamic Programming & Uniform Cost Search 1



search problem:

s_{start} starting state

$Actions(s)$ possible actions

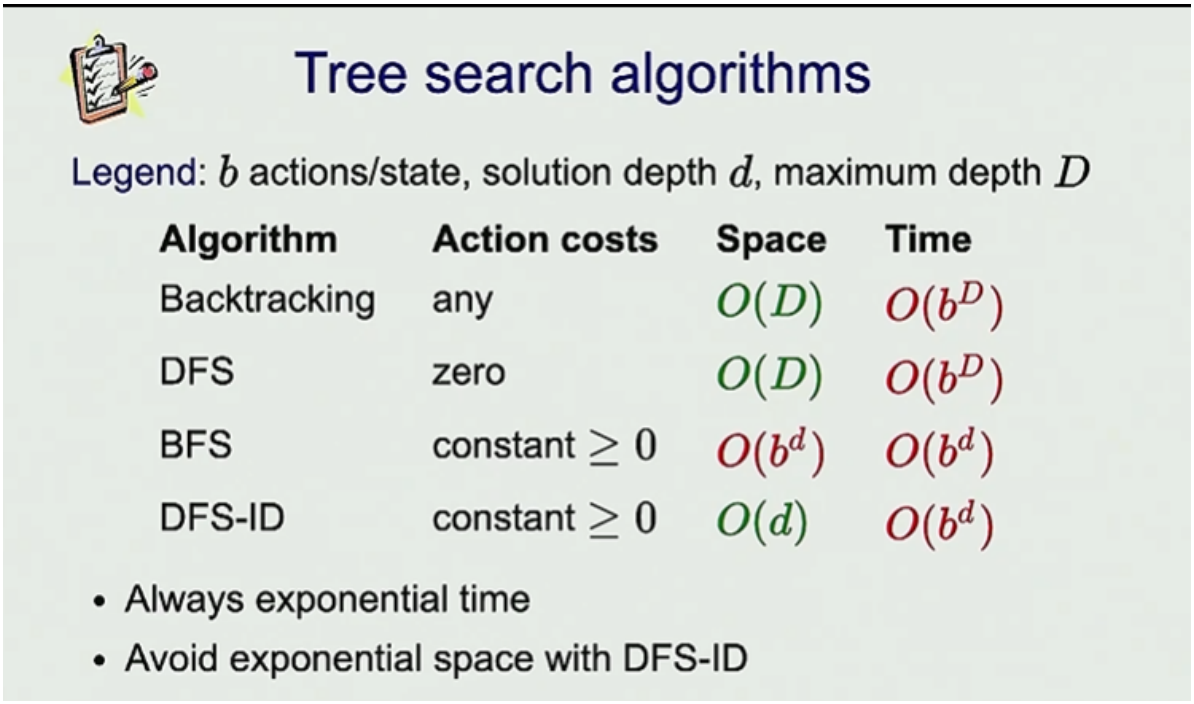
$Cost(s,a)$ action cost

$Succ(s,a)$ successor

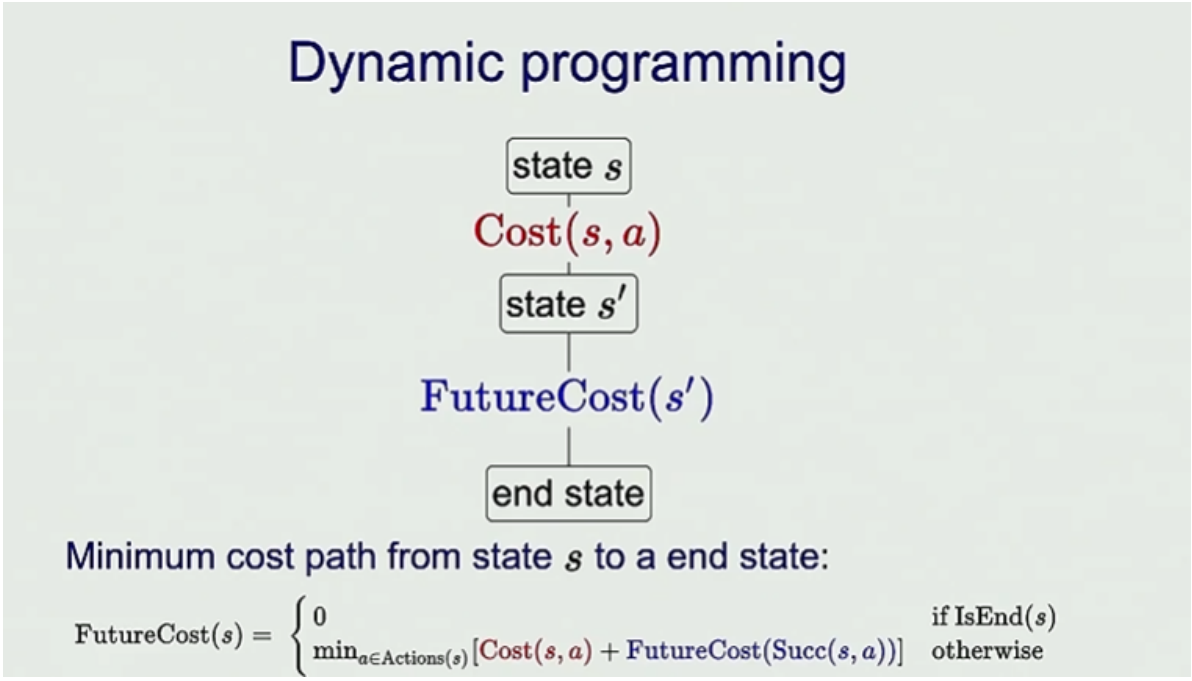
$InEnd(s)$ reach end state?

backtracking search:

```
def backtrackingSearch(s,path):
    if IsEnd(s):update minimum cost path
    For each action $a \in Action(s)$:
        Extend path with Succ(s,a) and Cost(s,a)
        Call backtracking Search(Succ(s,a),path)
    Return minimum cost path
```



Tips: in BFS, the action costs of each action should be the same



state: a summary of all the past actions sufficient to choose futures actions optimally

def DynamicProgramming(s):

If already computed for s,, return cached answer.

If IsEnd(s)L return solution

For each action $a\in Actions(s)$:...

Extend path with Succ(s,a) and Cost(s,a)

Call backtracking Search(Succ(s,a),path)

Return minimum cost path

assumption: acyclicity（非周期性） The state graph defined by Actions(s) and Succ(s,a) is acyclic

Uniform cost search(UCS)

key idea: UCS enumerates states in order of increasing past cost

Assumption: non-negativity all action costs are non-negative: $Cost(s,a)\geq 0$

- L6 Uniform Cost Search 2

uniform cost search function

```
def uniformcostsearch(problem):
    frontier=util.PriorityQueue()
    frontier.update(problem.startState(),0)
    while True:
        #Move from frontier to explored
        state,pastCost=frontier.removeMin()
        if problem.isEnd(state):
            return (pastCost,[])
        #Push out on the frontier
        for action,newState,cost in problem.succAndCost(state):
            frontier.update(newState,pastCost+cost)
```

Add s_{start} to frontier(priority queue)

Repeat until frontier is empty:

Remove s with smallest priority p from frontier

If IsEnd(s):return solution

Add s to explored

For each action $a \in Actions(s)$:

Get successor$s\prime \gets Succ(s,a)$

If $s\prime$ already in explored:continue

Update frontier with $s\prime$ and priority p+Cost(s,a)

Theorem:When a state s is popped from the frontier and moved to explored, its priority is PastCost(s), the minimum cost to s.

DP versus UCS

N total states, n of which are closer than end state

Algorithm	Cycles?	Action costs	Time/space
DP	no	any	$O(N)$
UCS	yes	≥ 0	$O(n \log n)$

Note: UCS potentially explores fewer states, but requires more overhead to maintain the priority queue

Note: assume number of actions per state is constant (independent of n and N)

Structed Perceptron(simplified)

For each action:$w[a]\gets 0$

For each iteration t=1,..,T:

For each training example$(x,y)\in D_{train}$

Compute the minimum cost path$y\prime$ given w

For each action $a\in y:w[a]\gets w[a]-1$

For each action $a\in y\prime :w[a]\gets w[a]+1$

Tips:

Try to decrease cost of true y (from training data)

Try to increase cost of preficted $y\prime$ (from search)

Heuristic（启发式的）function: a heuriatic function h(s) is any estimate of FutureCost(s)

A* search:

Run uniform cost search (UCS) with modified edgee costs:

$Cost(s,a)\prime = Cost(s,a)+h(Succ(s,a))-h(s)$

Intuition:add a penalty for how much action a takes us away from the end state

Doesn`t work when negative modified edge costs appears

Consistency: A heuristic h is consistent if

1.$Cost(s,a)\prime = Cost(s,a)+h(Succ(s,a))-h(s)\geq 0$

2.$h(s_{end})=0$

If h is consistent, A* returns the minimum cost path

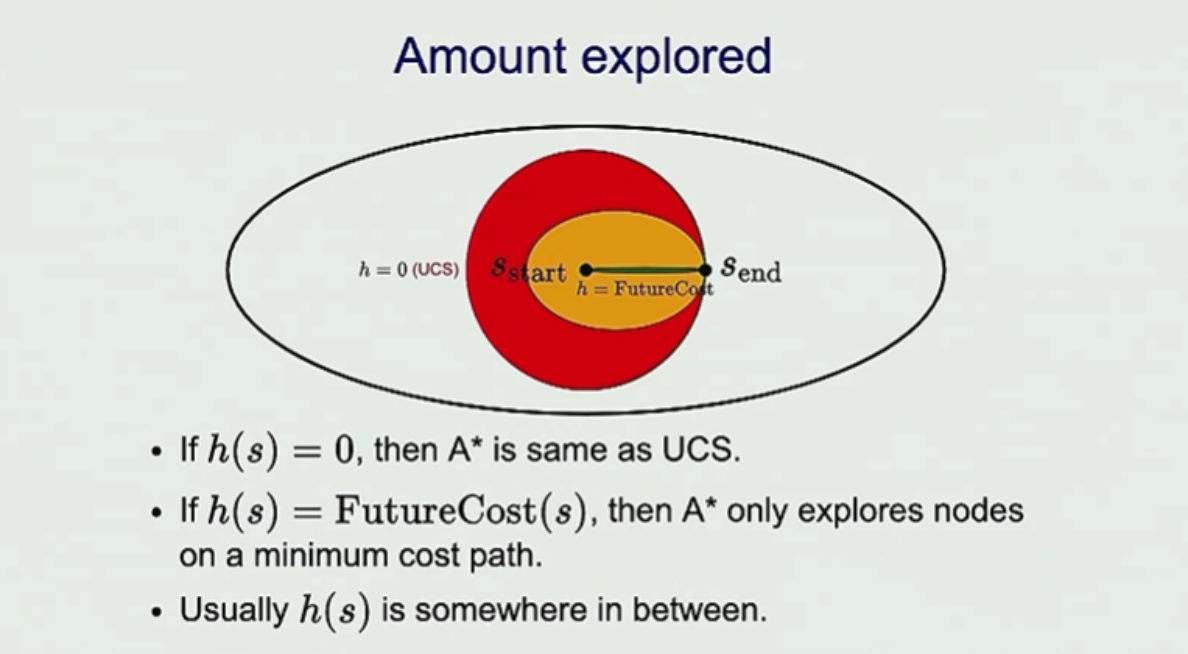
Efficiency of UCS: UCS explores all states s satisfying

$PastCost(s)\leq PastCost(s_{end})$

Efficieny of A*: A* explores all states s satisfying*

$PastCost(s)\leq PastCost(s_{end})-h(s)$

Interpretation: h(s)↑ better



Admissibility: a heuristic h(s) is admissible if

$h(s)\leq FutureCost(s)$

If a heuristic h(s) is consistent, then h(s) is admissible.

Relaxed Search Problem: A relaxation P_{rel} of a search problem P has costs that satisfy:

$Cost_{rel}(s,a)\leq Cost(s,a)$

Relaxed heuristic: Given a relaxed search problem $P*{rel}$,define the relaxed heuristic $h(s)=FutureCost*{rel}(s)$, the minimum cost from s to an end state using $Cost_{rel}(s,a)$

Here, $h(s)$ is consistent

- L7 Markov Decision Process - Value Iteration

MDP Markov Decision Process 马尔可夫决策过程

States: the set of states

$s_{start}\in States$ starting state

Actions(s): possible actions from state s

$T(s,a,s\prime)$: probability of $s\prime$ if take action a in state s

$Reward(s,a,s\prime)$:reward for the transition$(s,a,s\prime)$

IsEnd(s): whether at end of game

$0\leq \gamma \leq 1$: discount factor(default:1)

For each state s and action a:

$\sum_{s\prime \in States}T(s,a,s\prime)=1$

Successors:$s\prime$ such that $T(s,a,s\prime)> 0$

the solution of MDP: policy

A policy π is a mapping from each state $s\in States$ to an actioon $a\in Actions(s)$

Utility:

Following a policy yields a random path

The utility of a policy is the (discounted) sum of the rewards on the path (this is a random quantity)

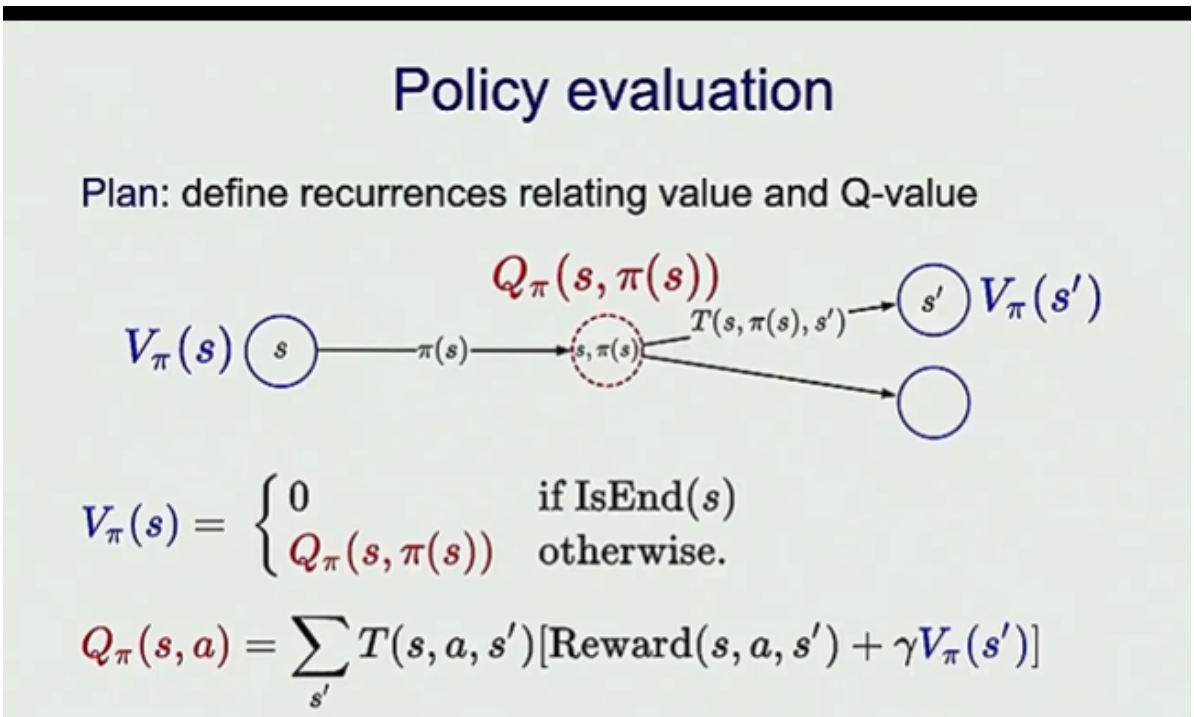
Path:$s_0,a_1r_1s_1,a_2r_2s_2,...(action,reward,new\space state)$. The utility with discount γ is

$u_1=r_1+\gamma r_2+\gamma ^2r_3+\gamma ^3r_4+...$

Value: the value of a policy is the expected utility

Let $V_{\pi}(s)$be the expected utility received by following policy π from state s

Q-value of a policy: Let $Q_{\pi}(s,a)$ be the expected utility of taking action a from state s , and then following policy π



Policy Evaluation (using iterative algorithm):

Initialize $V_{\pi}^{(0)}(s)\gets 0$ for all states s.

For iteration $t=1,...,t_{PE}$:

For each state s:

$V{\pi}^{(t)}(s)\gets \sum{s^{\prime}} T(s,\pi (s),s\prime)[Reward(s,\pi (s),s\prime)+\gamma V_{\pi}^{(t-1)}(s\prime)]=Q^{(t-1)}(s,\pi(s))$

Note:

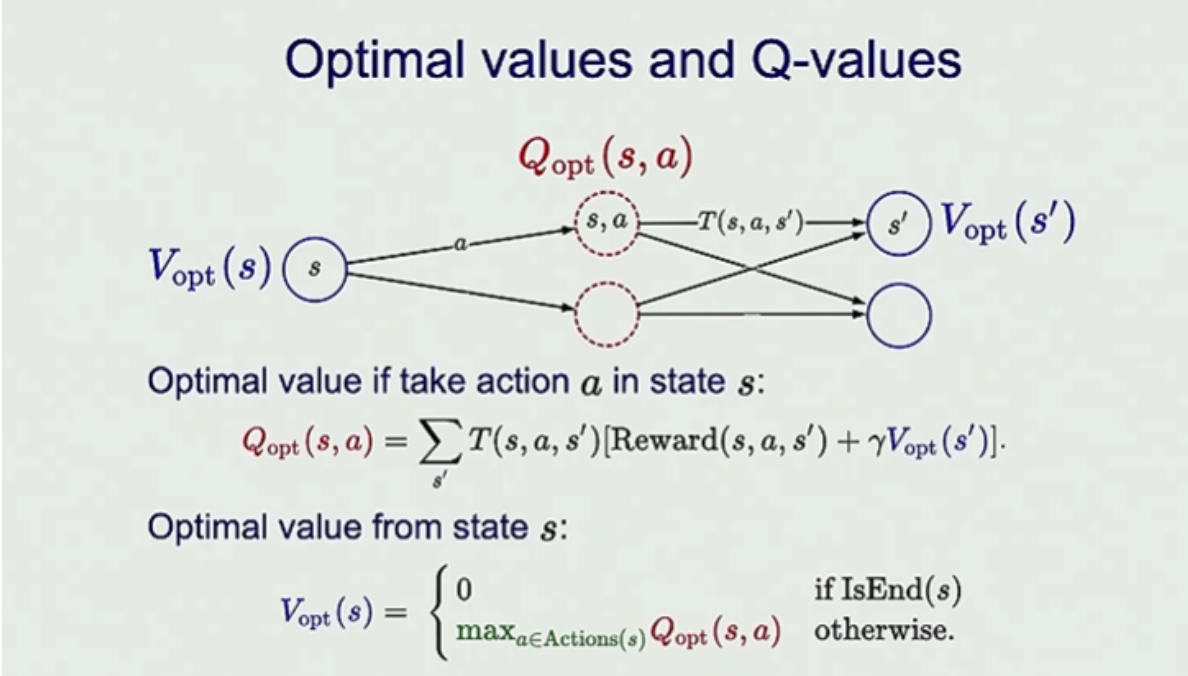
1.How many iterations(t_{PE})? Repeat until values don`t change much:

$max{s\in States}|V{\pi}^{(t)}-V_{\pi}^{(t-1)}|\leq \epsilon$

2. Don`t store $V{\pi}^{(t)} $for each iteration t, need only last two:$V{\pi}^{(t)}$and $V_{\pi}^{(t-1)} $

Time: $O(t_{PE}SS^{\prime})$

Optimal value $V_{opt}(s)$: the maximum value attained by any policy



Given Q_{opt}, read off the optimal policy"

$\pi{opt}(s)=arg max{a\in Actions(s)}Q_{opt}(s,a)$

Value Evaluation (using iterative algorithm):

Initialize $V_{opt}^{(0)}(s)\gets 0$ for all states s.

For iteration $t=1,...,t_{VI}$:

For each state s:

$V{opt}^{(t)}(s)\gets \sum{s^{\prime}} T(s,a,s\prime)[Reward(s,a,s\prime)+\gamma V{opt}^{(t-1)}(s\prime)]=Q^{(t-1)}{opt}(s,a)$

Time $O(t_{VI}SAS^{\prime})$

Code related with value interation:

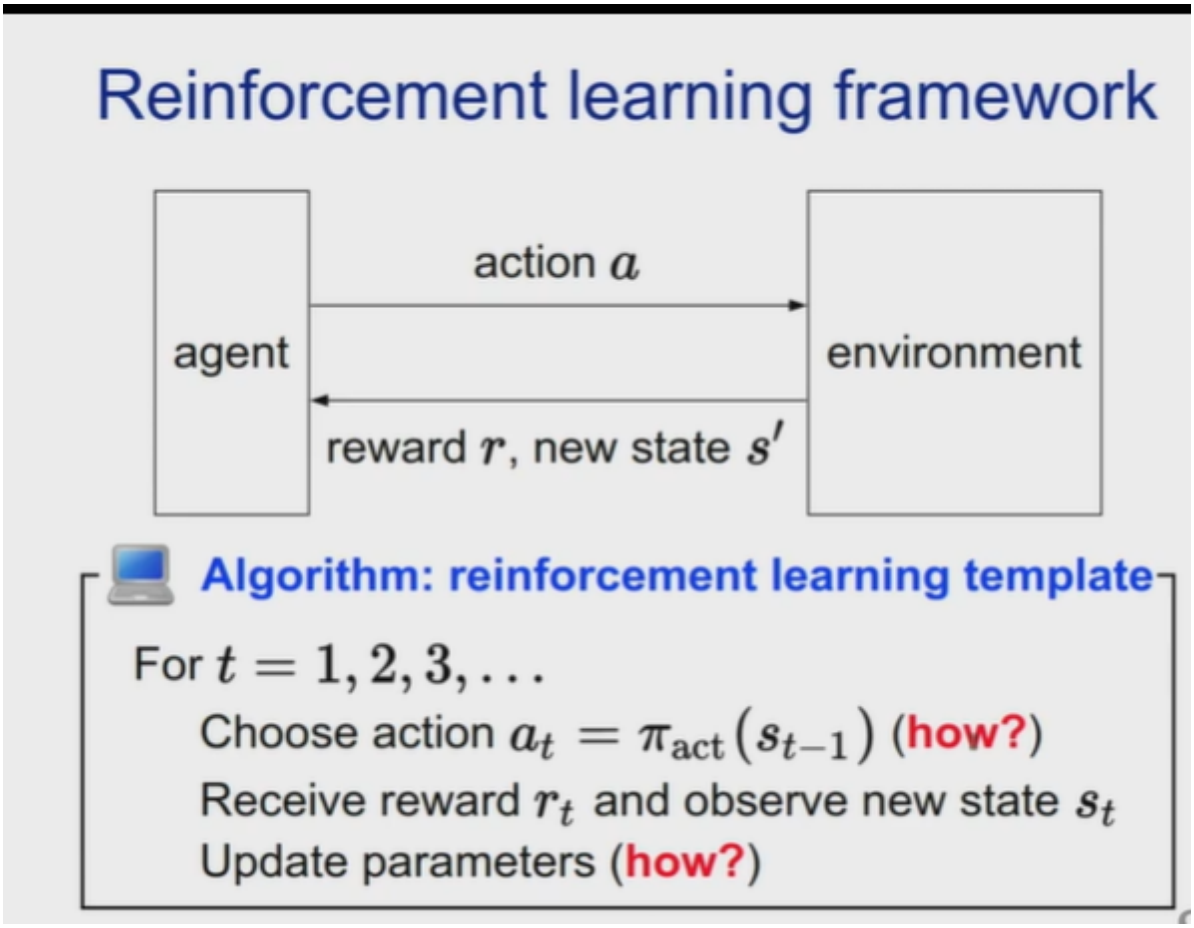
```
def valueIteration(mdp):  #display of value iteration
    #Initialize
    v={} #state->Vopt[state]
    for state in mdp.states():
        v[state]=0
```

```
    def Q(state,action):
        return sum(prob*(reward +mdp.discount()+V[newState])\
                   for newState,prob,reward in mdp.succProbReward(state,action))
    #prob means probilities
    while True:
        #compute the new values (newV) given the old values(V)
        newV={}
        for state in mdp.states():
            if mdp.isEnd(state):
                newV[state]=0
            else:
                newV[state]=max(Q(state,action) for action in
mdp.actions(state))
           #check for convergence
        if max(abs(V[state]-newV[state])for state in mdp.states())<1e-10:
            break
        V=newV

        #read out policy
        pi={}
        for state in mdp.states():
            if mdp.isEnd[state]:
                pi[state]='none'
            else:
                pi[state]=max((Q(state,action),action)\
                              for action in mdp.acxtions(state))[1]
         #print stuff out
        os.system('clear')
        print('{:20}{:20}{:20}'.format('s','V(s)','pi(s)') )
        for state in mdp.states():
            print('{:15}{:15}{:15}'.format(state,V[state],pi[state]) )
        input()
```

Converge:

discount$\gamma <1$ or MDP graph is acyclic

- L8 Markov Decision Process - Reinforcement Learning



model-based Monte Carlo

Key idea: model-based learning

Estimate the MDP: $T(s,a,s^{\prime})$ and $Reward(s,a,s^{\prime})$

Transitions:

$\hat T(s,a,s^{\prime})=\frac {# times(s,a,s^{\prime}) occurs} {# times(s,a) occurs}$

Rewards:

$\hat {Reward}(s,a,s^{\prime})=r\space in(s,a,r,s^{\prime})$

model-free Monte Carlo

Try to estimate $Q_{opt}(s,a)$directly

$\hat Q{\pi}(s,a)=average\space of\space u_t \space where\space s$t-1}=s,a_t=a$ and s,a doesn`t occur in $s_t,...$

$u_t=r_t+\gamma\cdot r{t+1}+\gamma^2\cdot r$t+2}+...$

Equivalent formulation (convex combination):

On each (s,a,u):

$\eta =\frac{1}{1+(# updates\space to\space (s,a))}$

$\hat Q{\pi}(s,a)\gets(1-\eta)\hat Q$\pi}(s,a)+\eta u$

Equivalent formulation (stochastic gradient):

On each (s,a,u):

$\hat Q{\pi}(s,a)\gets\hat Q{\pi}(s,a)-\eta [\hat Q(s,a)-u]$

prediction:$\hat Q_{\pi}(s,a)$

target: u

Implied objective: least squares regression $(\hat Q_{\pi}(s,a)-u)^2$

Note: we are estimating $Q{\pi}$ now, not $Q{opt}$

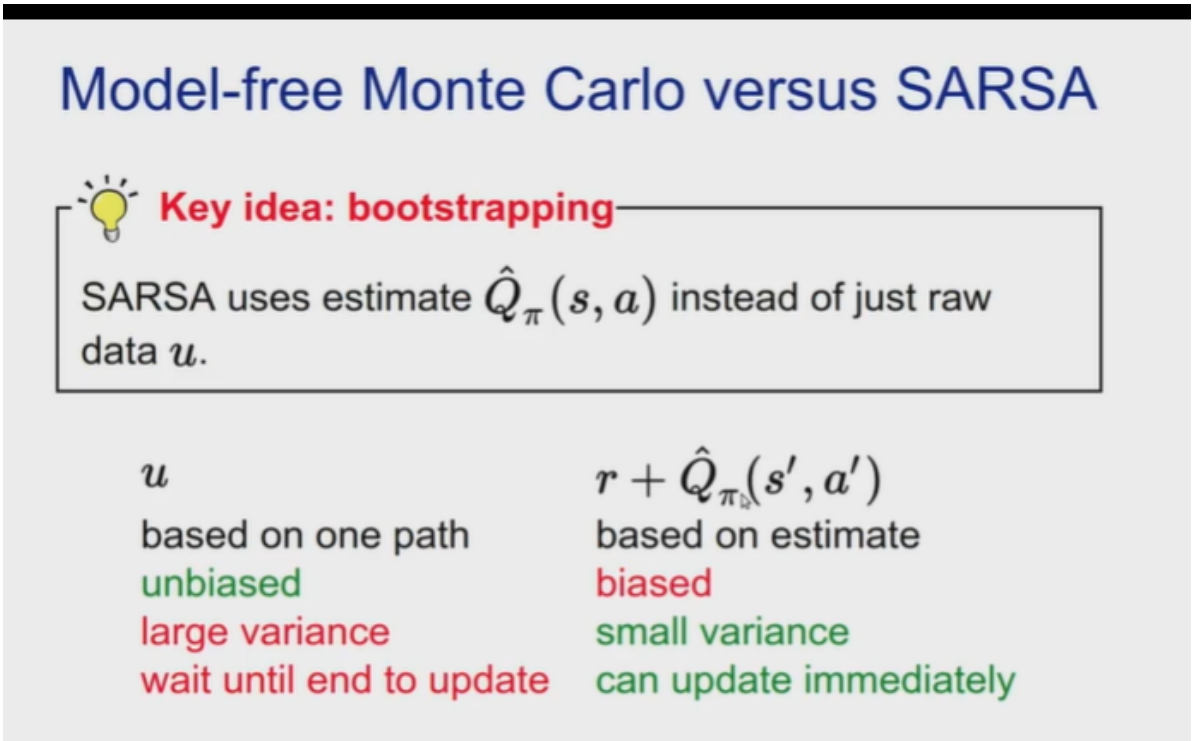
On-policy:estimate the value of data-generating policy

Off-policy: estimate the value of another policy

SARSA

On each $(s,a,r,s^{\prime},a^{\prime})$:

$\hat Q{\pi}(s,a)\gets (1-\eta)\hat Q{\pi}(s,a)+\eta[r+\gamma \hat Q_{\pi}(s^{\prime},a^{\prime})]$



Note: SARSA doesn`t allow us to estimate Q_{opt}

SARSA is on-policy

Q-learning

On each $(s,a,r,s^{\prime})$:

$\hat Q{opt}(s,a)\gets (1-\eta)\hat Q{opt}(s,a)+\eta[r+\gamma \hat V_{opt}(s^{\prime})]$

Recall:$\hat V{opt}(s^{\prime})=max{a^{\prime}\in Actions(s^{\prime})}\hat Q_{opt}(s^{\prime},a^{\prime})$

Q-learning is off-policy

epsilon-greedy policy: a way to balance exploitation（对于已知） & exporation （对于未知）

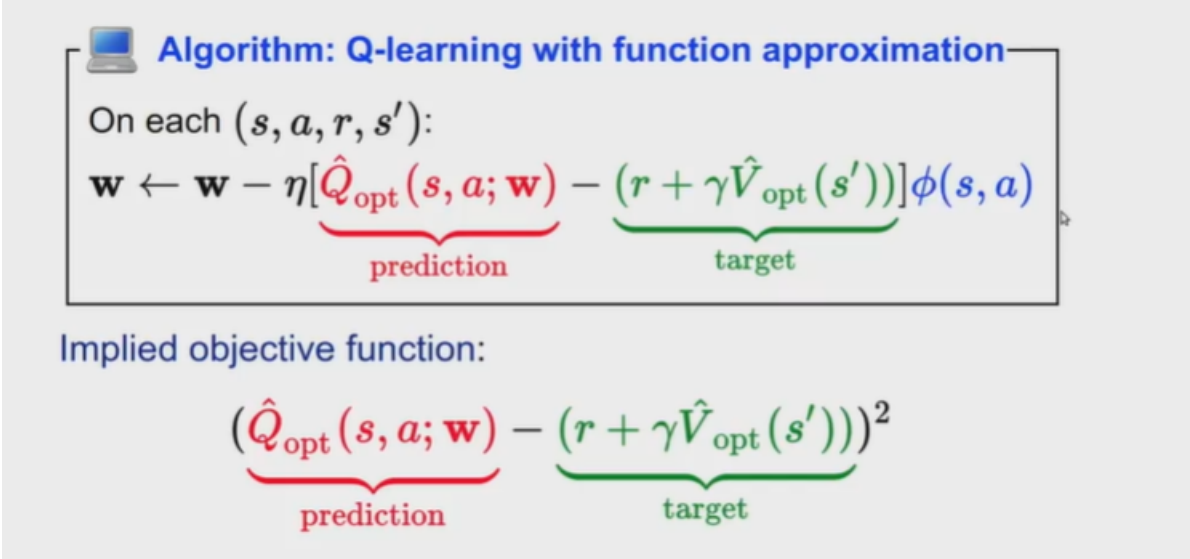
$\pi{act}(s)=\begin{cases} argmax{a\in Actions}\hat Q_{opt}(s,a)& probability\space 1-\epsilon\ random\space from\space Actions(s)& probability\space \epsilon \end{cases}$

Function approximation:

we can use linear regression model:

define features $\phi(s,a)$ amd weights W:

$\hat Q_{opt}(s,a;w)=w\cdot \phi(s,a)$



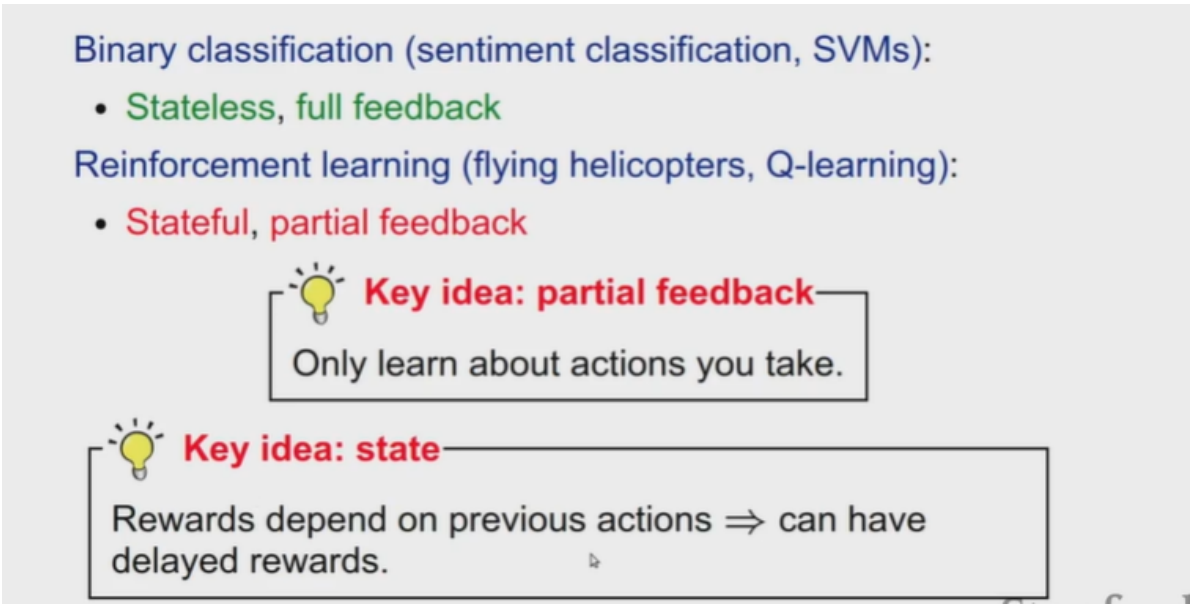
Summary:

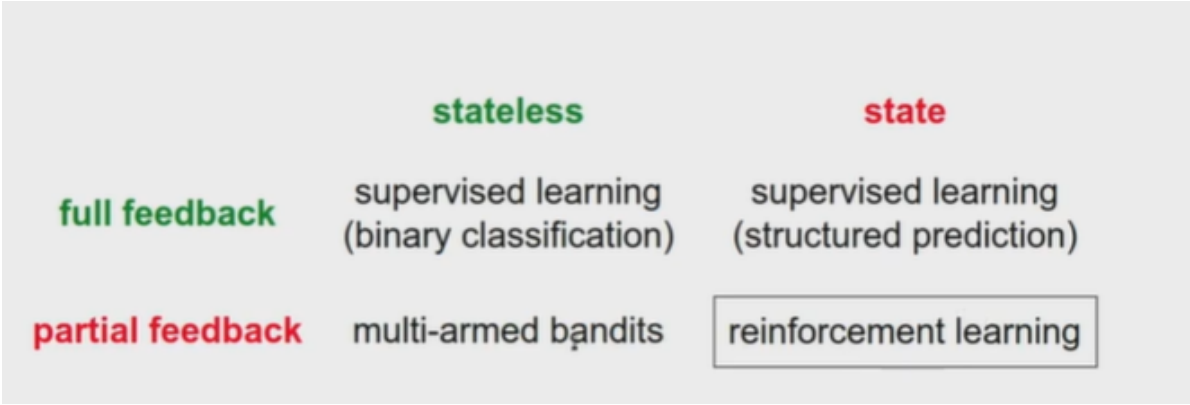
Epsilon-greedy: balance the exploration/exploitation tradeoff

Function approximation: can generalize to unseen states

Online setting: learn and take actions in the real world

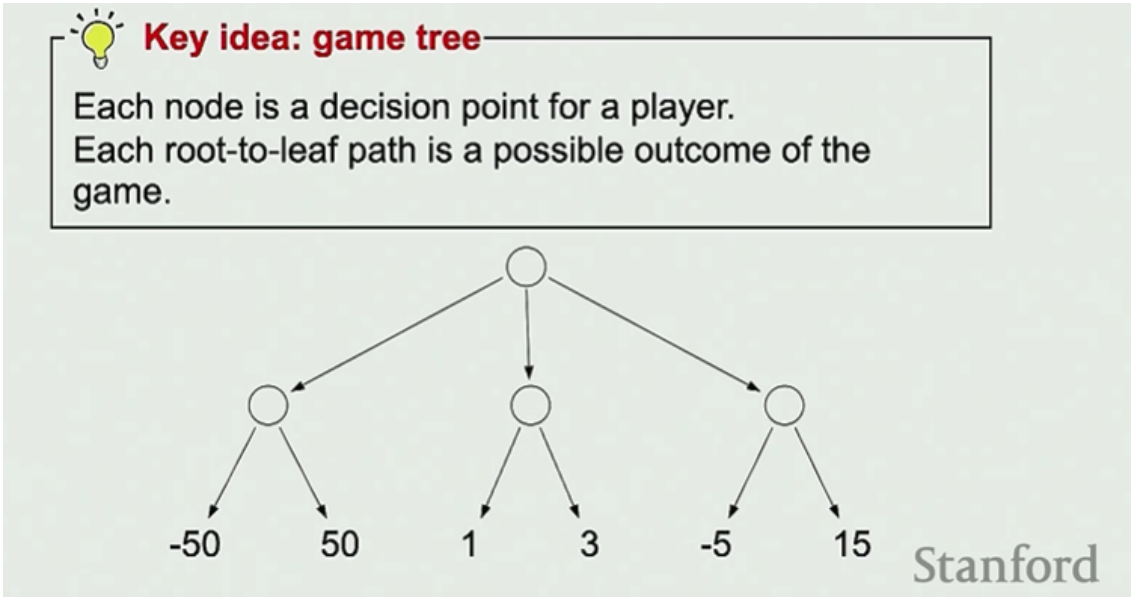
Bootstrapping（自举）: update towards target that depends on estimate rather than just raw data





multi-armed bandits https://zhuanlan.zhihu.com/p/125658638

- L9 Game Playing 1 - Minimax & Alpha-beta Pruning



two-player zero-sum（零和）game:

s_{start} starting state

Actions(s) possible actions from state s

Succ(s,a) resulting state if choose action a in state s

IsEnd(s) whether s is an end state(game over)

Utility(s):agent`s utility for end state s

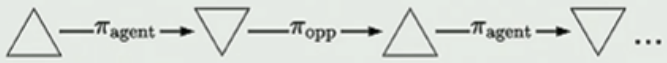
$Player(s)\in Players$ player who controls state s

Characteristics of games:

1. All the utility is at the end state
2. Different players in control at different states

Minimax recurrence

No analogy in MDPs:



$$V_{\text{minmax}}(s) = \begin{cases} \text{Utility}(s) & \text{IsEnd}(s) \\ \max_{a \in \text{Actions}(s)} V_{\text{minmax}}(\text{Succ}(s,a)) & \text{Player}(s) = \text{agent} \\ \min_{a \in \text{Actions}(s)} V_{\text{minmax}}(\text{Succ}(s,a)) & \text{Player}(s) = \text{opp} \end{cases}$$

Recurrences produces policies:

$V_{\text{exptmax}} \Rightarrow \pi_{\text{exptmax}(7)}, \pi_7$ (some opponent)

$V_{\text{minmax}} \Rightarrow \pi_{\text{max}}, \pi_{\text{min}}$

Play policies against each other:

	π_{min}	π_7
π_{max}	$V(\pi_{\text{max}}, \pi_{\text{min}})$	$V(\pi_{\text{max}}, \pi_7)$
$\pi_{\text{exptmax}(7)}$	$V(\pi_{\text{exptmax}(7)}, \pi_{\text{min}})$	$V(\pi_{\text{exptmax}(7)}, \pi_7)$

Proposition:

1.best against minimax opponent

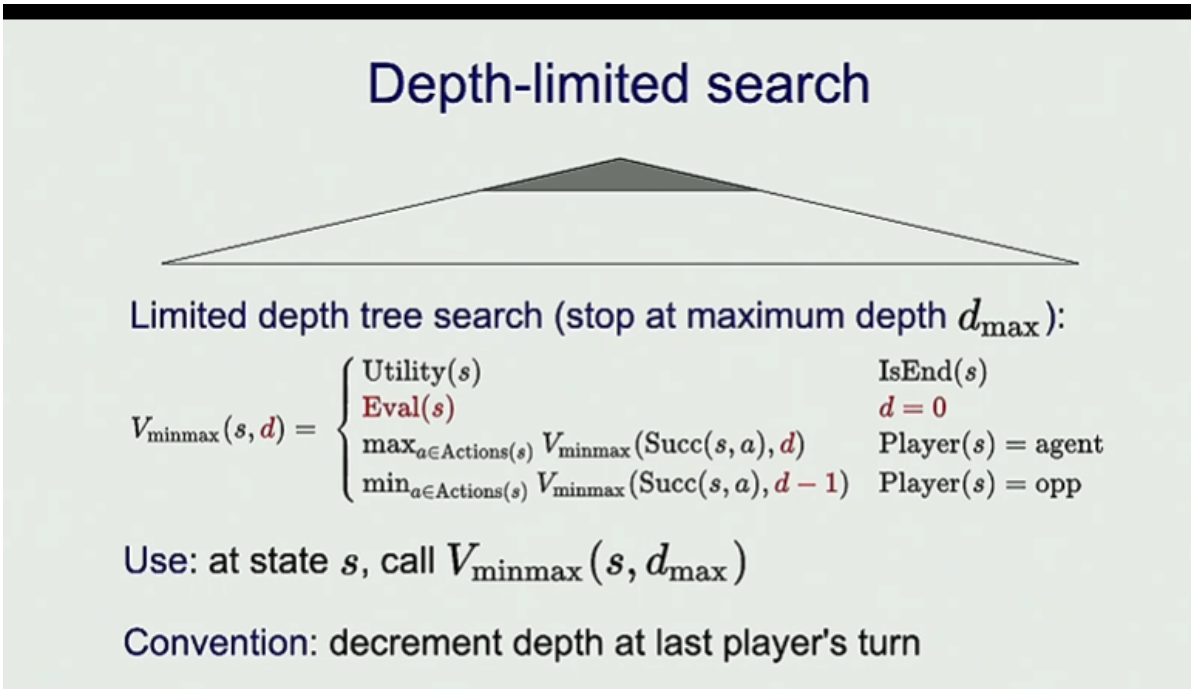
$V(\pi_{max}, \pi_{min}) \geq V(\pi_{agent}, \pi_{min})$ for all π_{agent}

2.lower bound against any opponent

$V(\pi_{max}, \pi_{min}) \leq V(\pi_{max}, \pi_{opp})$ for all π_{opp}

3.not optimal if opponent is known

$V(\pi_{max}, \pi_7) \leq V(\pi_{exptmax(7)}, \pi_7)$ for all π_7



Evaluation function Eval(s) : a (possibly very weak) estimate of the value $V_{minmax}(s)$

Depth-limited exhaustive search:$O(b^{2d})$time

Eval(s) attempts to estimate$V_{minmax}(s)$ using domain knowledge

No guarantees(unlike A*) on the error from approximation

Alpha-beta pruning:

optimal path: is the path that minimax policies take. Values of all nodes on path are the same

a_s lowerbound on max node s

b_s upperbound on min mode s

Prune a node if its interval doesn`t have non-trivial overlap with every ancestor (store $\alpha_s=max*{s^{\prime}\leq s}a_s$ and $\beta_s=min*{s^{\prime}\leq s}b_s$)

Pruning depends on order of actions

Worst ordering:$O(b^{2d})$ time

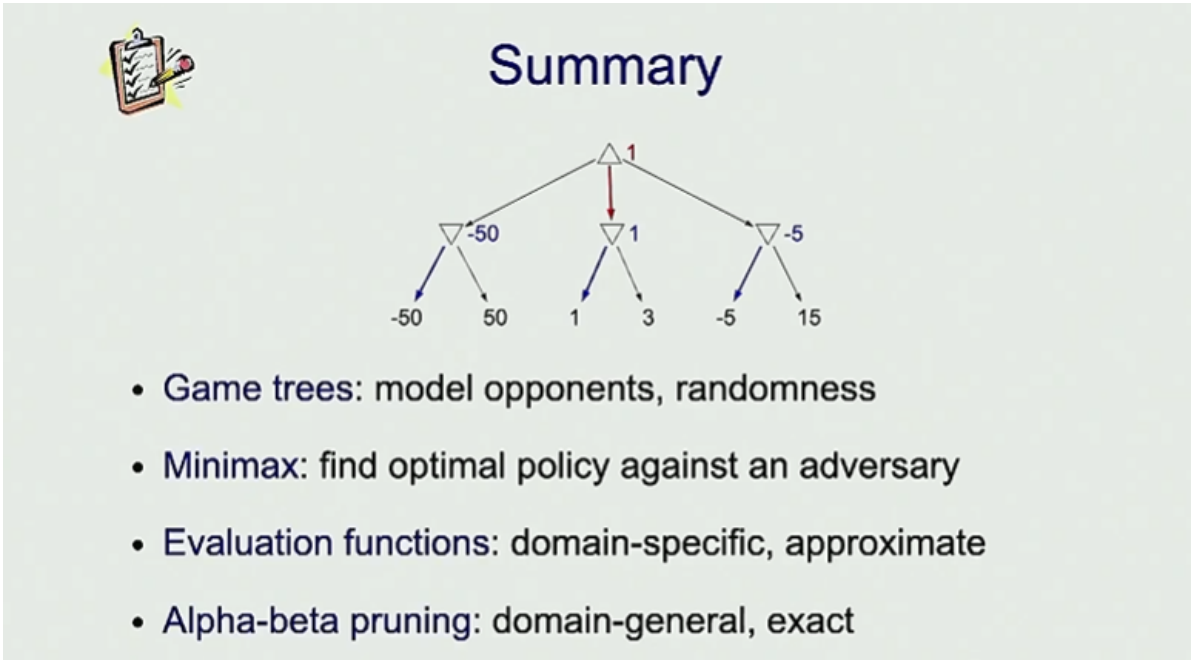
Best ordering:$O(b^d)$ time

Random ordering:$O(b^{1.5d})$ time

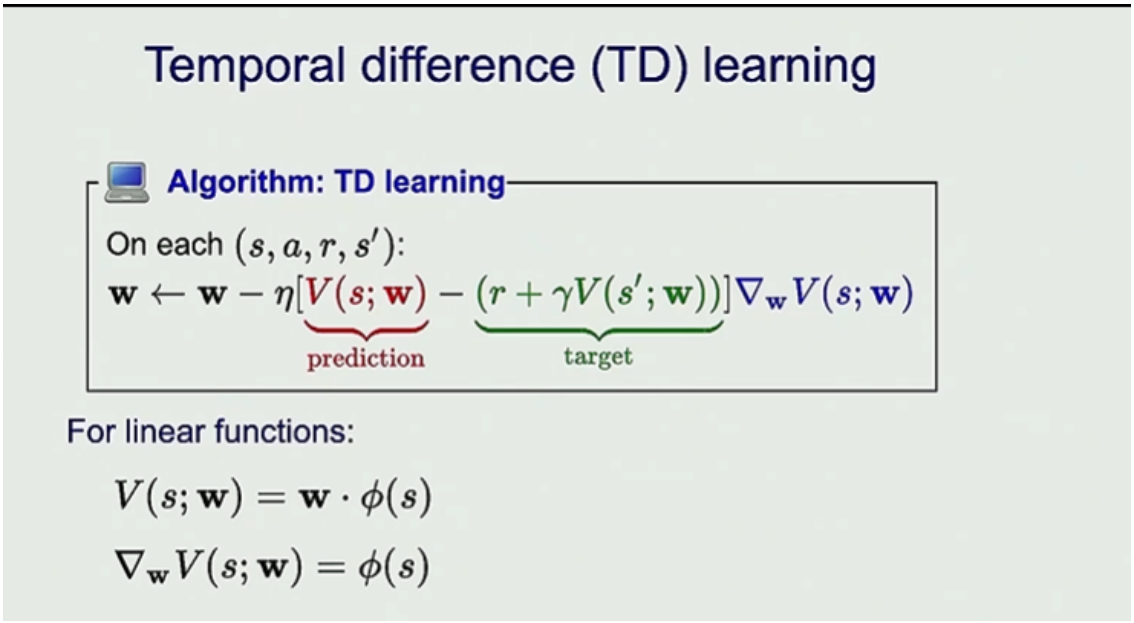
In practice, can use evaluation function Eval(s):

Max nodes: order successors by decreasing Eval(s)

Min nodes: order successors by increasing Eval(s)



- L10 Game Playing 2 - Temporal Difference Learning



Q-learning vs TD learning:

Q-learning

1. operate on $\hat Q_{opt}(s,a,;w)$
2. off-policy value is based on estimate of optimal policy
3. to use, don`t need to know MDP transitions $T(s,a,s^{\prime})$

TD learning

1. operate on $\hat V_{\pi}(s;w)$
2. on-policy value is based on exploration policy (usually based on $\hat V_{\pi}$)
3. to use, need to know rules of the game Succ(s,a)

Single-move simultaneous game:

players={A,B}

Actions:possible actions

V(a,b): A`s utility if A chooses action a, B chooses b (Let V be payoff matrix)

Pure strategy: a single action $a\in Actions$

Mixed strategy: a probability distribution $0\leq\pi(a)\leq1$ for $a\in Actions$

game evaluation: The value of the game if player a follows π_A and player B follows π_B is

$V(\pi_A,\pi_B)=\sum_{a,b}\pi_A(a)\pi_B(b)V(a,b)$

minimax theorem (von Neumann Theorem):

For every simultaneous two-player zero-sum game with a finite number of actions:

$max*{\pi_A} min*{\pi_B}V(\pi_A,\pi_B)=min*{\pi_B}max*{\pi_A}V(\pi_A,\pi_B)$

where π_A,π_B range over mixed strategies

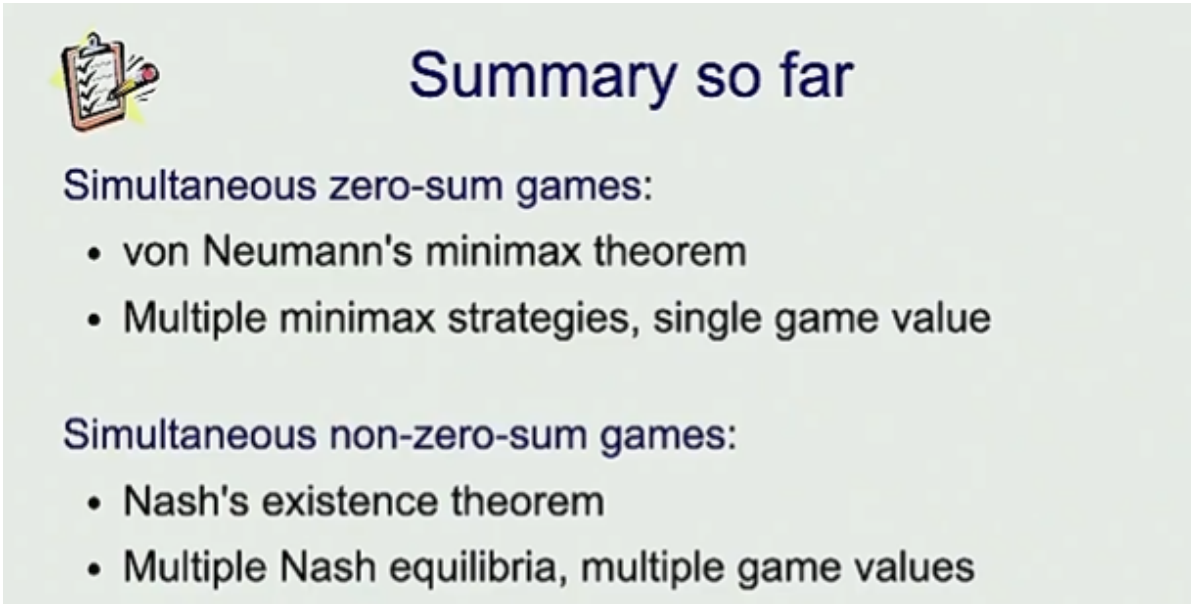
Upshot:revealing your optimal mixed strategy doesn`t hurt you

Nash equailibrium: A Nash equilibruim is $(\pi^_A,\pi^_B)$ such that no player has an incentive to change his/her strategy:

$V_A(\pi^_A,\pi^B)\geq V_A(\pi_A,\pi^*B)$ for all π_A

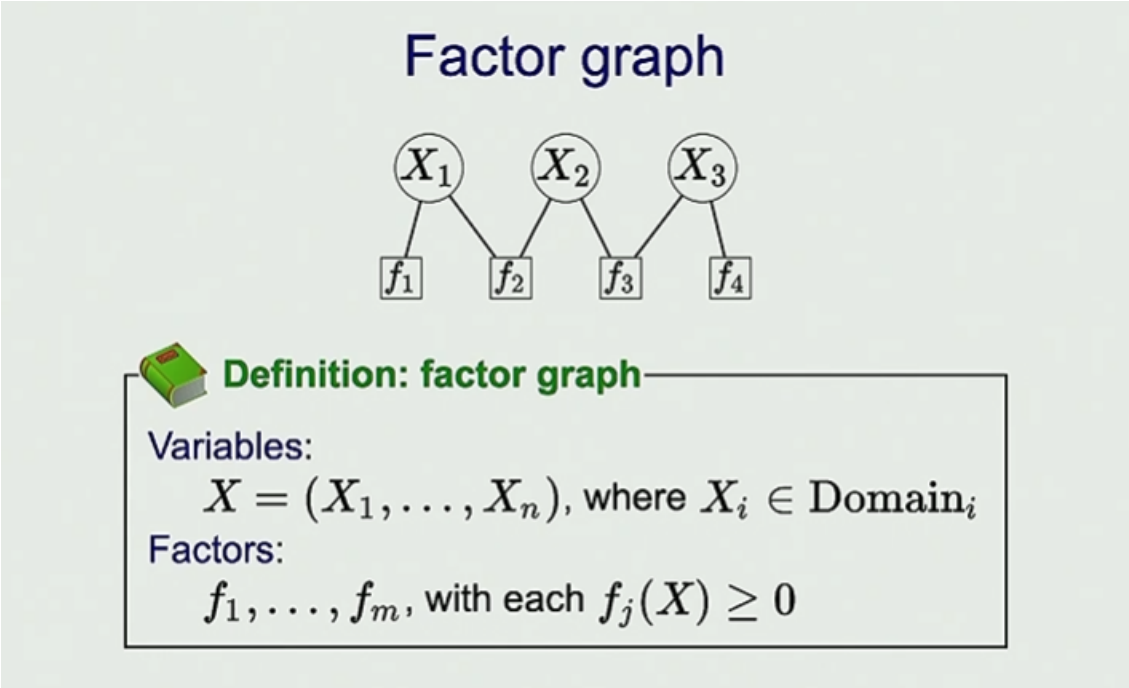
$V_B(\pi^_A,\pi^_B)\geq V_B(\pi_A,\pi_B)$ for all π_B

Nash`s existence theorem: In any finite-player game with finite number of actions, there exists at least one Nash equilibrium



- L11 Factor Graph 1 - Constraint Satisfaction Problems

  Constraint Satisfaction Problem (CSP)



scope and arity:

scope of a factor f_j : set of variables it depends on

Arity （参数数量）of f_j is the number of variables in the scope

Each assignment $x=(x_1,...,x_n)$ has a weight:

$Weight(x)=\prod^m_{j=1}f_j(x)$

Objective: $argmax_xWeight(x)$

A CSP is a factor graph where all the factors are constraints:

$f_j(x)\in {0,1}$ for all j=1,...,m

The constraint is satisfied if $f_j(x)=1$

An assignment is consistent if Weight(x)=1 (i.e., all constraints are satisfied)

Dependent factors: Let $D(x,X_i)$ be set of factors depending on X_i and x but not on unassigned variables. Here , x refers to partial assignment

BackTracking Search Algorithm:

Backtrack($x,w,Domains$):

If x is complete assignment: update best and return

Choose unassigned VARIABLE X_i (Choose variable that has the fewest consistent values/most constrainted)

Order VALUES $Domain_i$ of chosen X_i

For each value v in that order: (choose least constrained value)

$\delta \gets \prod_{f_j \in D(x,X_i)}f_j(x\cup{X_i:v})$

If $\delta=0$:continue

$Domains^{\prime}\gets Domains$ via LOOKAHEAD

Backtrack$(x\cup {X_i:v},w\delta,Domains^{\prime})$

Forward Checking (one-step lookahead)

1.After assigning a variable X_i eliminate inconsistent values from the domains f X_i`s neighbors

2.If any domain becomes empty, don`t recure

3.When unassigned X_i,restore neighbor`s domains

Arc consistency

A variable X_i is arc consistent with respect to X_j if for each $x_i\in Domain_i$,there exists $X_j\in Domain_j$ such that $f({X_i:x_i,X_j:x_j})\neq 0$ for all factors f whose scope contains X_i and X_j

Enforce arc consistency:

$EnforceArcConsistency(X_i,X_j)$: Remove values from $Domain_i$ to make X_i arc consistent with respect to X_j

Algorithm AC-3: repeatedly enforce arc consistency on all variables

Add X_j to set.

While set is non-empty:

Remove any X_k from set

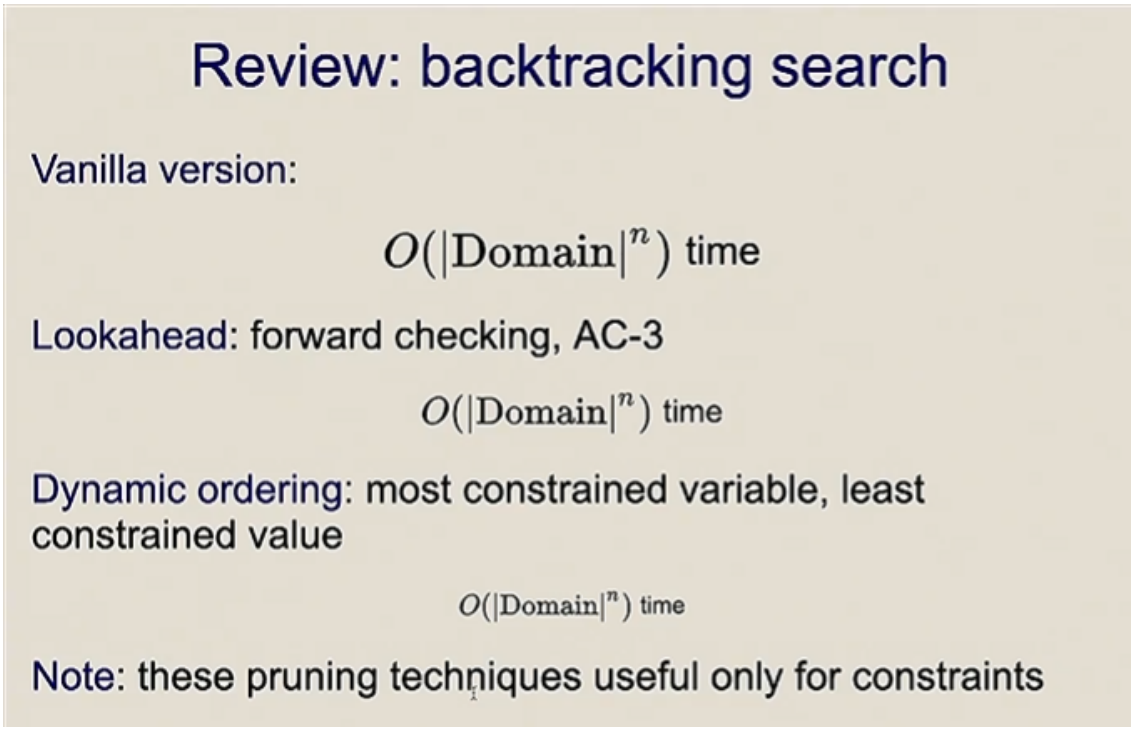
For all neighbors X_l of X_k:

Enforce arc consistency on X_l w.r.t. X_k

If $Domain_l$ changed, add X_l to set

(w.r.t.=with respect to)

- L12 Factor Graph 2 - Conditional Independence



Greedy search

Partial assignment $x\gets {}$

For each $i=1,...,n$:

Extennd:

Compute weight of each $x_v=x \cup{X_i:v}$

Prune:

$x\gets x_v$ with highest weight

Note: Not guaranteed to find optimal assignment

Beam search

Idea: keep $\leq K$ candidate list C of partial assignments

Algorithm:

Initialize $C\gets[{}]$

For each i=1,...,n:

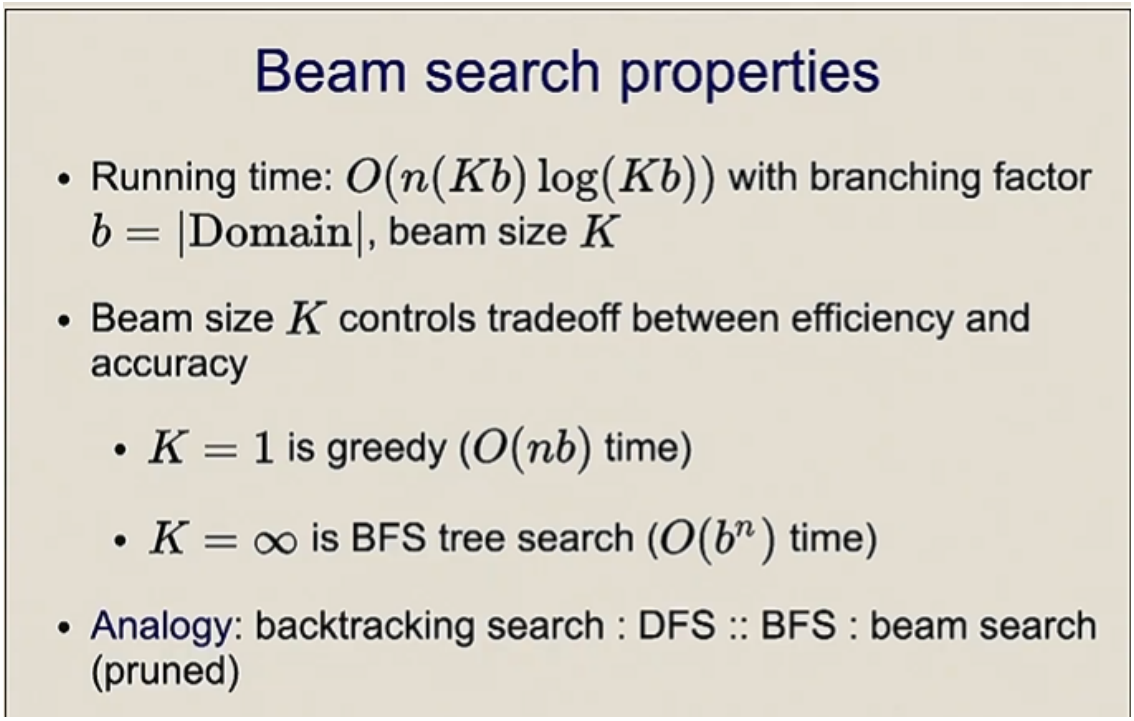
Extend:

$C^{\prime} \gets {x\cup{X_i:v}:x\in C,v_{Domain_i}}$

Prune:

$C \gets K$elements of $C^{\prime}$ with highest weights

Note: Not guaranteed to find optimal assignment!



Iterated Conditional Modes (ICM)

locality: When evaluating possible re-assignments to X_i, only need to consider the factors that depend on X_i

Algorithm:

Initialize x to a random complete assignment

Loop through i=1,...,n until convergence:

Compute weight of $x_v=x\cup{X_i:v}$ for each v

$x \gets x_v$ with highest weight

Gibbs sampling

Randomness: Sample an assignment with probability proportinal to its weight.

Algorithm:

Intialize x to a random complete assignment

Loop through i=1,...,n until convergence:

Compute weight of $x_v=x\cup {X_i:v}$ for each v

$x \gets x_v$ with highest weight

Independence:

Let A and B be a partitioning of variables X

We say A and B are independent if there are no edges between A and B

In symbols:$A \perp !!! \perp B$

Conditioning:

1. To condition on a variable $X_i=v$ consider all factors $f_1,...f_k$ that depends on X_i
2. Remove X_i and $f_i,...,f_k$
3. Add $g_j(x)=f_j(x\cup{X_i:v})$ for $j=1,...,k$

Conditional Independence:

1. Let A,B,C be a partitining of the variables
2. We say A and B are conditionally independent given C if conditioning on C produces a graph in which A and B are independent
3. In symbols: $A \perp !!! \perp B\space|\space C$

Equivolent: Every path between A and B through C

Markov blanket:

Let $A\subseteq X$ be a subset of variables

Define MarkovBlanket(A) be the neighbors of A that are not in A.

So:

Let $C=MarkovBlanket(A)$

Let B be $X\backslash(A\cup C)$

Then$A \perp !!! \perp B\space|\space C$

Elmination:

To elminate a variable X_i , consider all factors $f_1,...,f_k$ that depend on X_i

Remove X_i and $f_1,...,f_k$

Add $f{new}(x)=max\prod^k{j=1}f_j(x)$

Algorithm:variable elimination:

For i=1,...,n:

Eliminate X_i (produces new factor $f_{new,i}$)

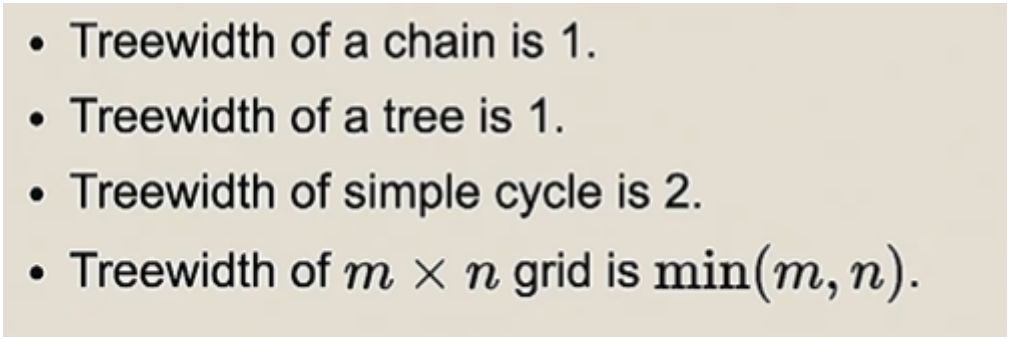
For i=n,...,1:

Set X_i to the maximizing value in $f_{new,i}$

Let max-arity be the maximum arity of any $f_{new,i}$

Running time: $O(n\cdot |Domain|^{max-arity+1})$

Treen width: the tree width of a factor graph is the maximum arity of any factor created by variable elimination with the best variable ordering



- L13 Bayesian Network 1 - Inference

  Bayesian network

  Let $X=(X_1,...X_n)$ be random variables.

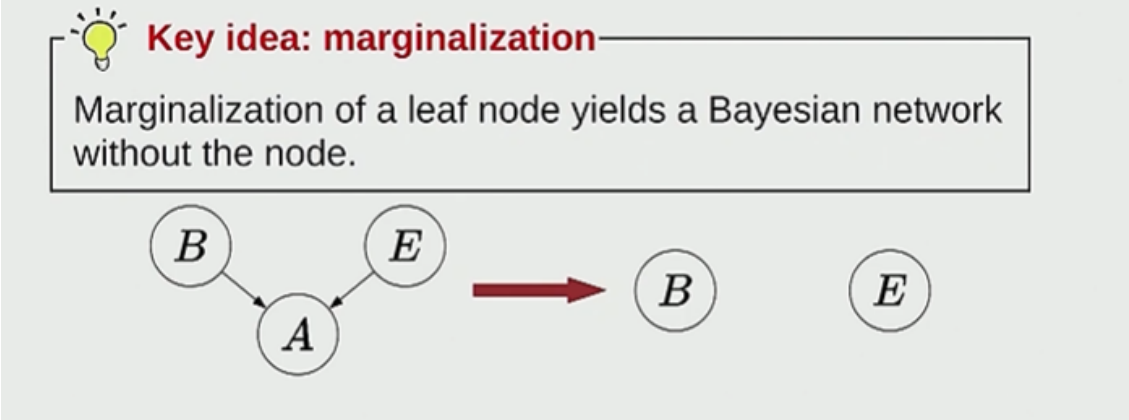
  A Bayesian network is a direct acyclic graph (DAG) that specifies a joint distribution over X as a product of local conditional distributions,one for each node:

  $P(X_1=x_1,...,X_n=x_n)\overset{def}{=}\prod ^n{i=1}p(x_i|x{Parents(i)})$

  Locally normalized

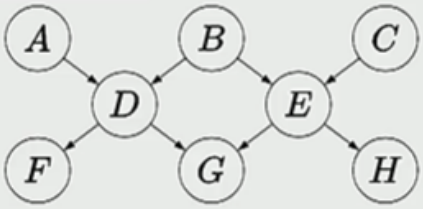
  All factors (local conditional distributions) satify:

  $\sum{x_i}p(x_i|x{Parents(i)})=1$ for each $x_{Parents(i)}$



Key idea: local conditional distributions

Local conditional distributions (factors) are the true conditional distributions.

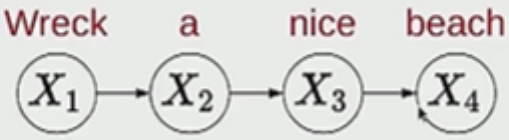


$$\underbrace{\mathbb{P}(D = d \mid A = a, B = b)}_{\text{from probabilistic inference}} = \underbrace{p(d \mid a, b)}_{\text{by definition}}$$

Probabilistic program: A randomized program that sets the random variables.

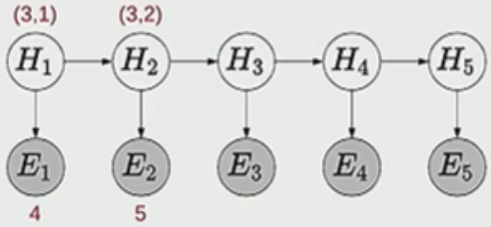
Probabilistic program: Markov model

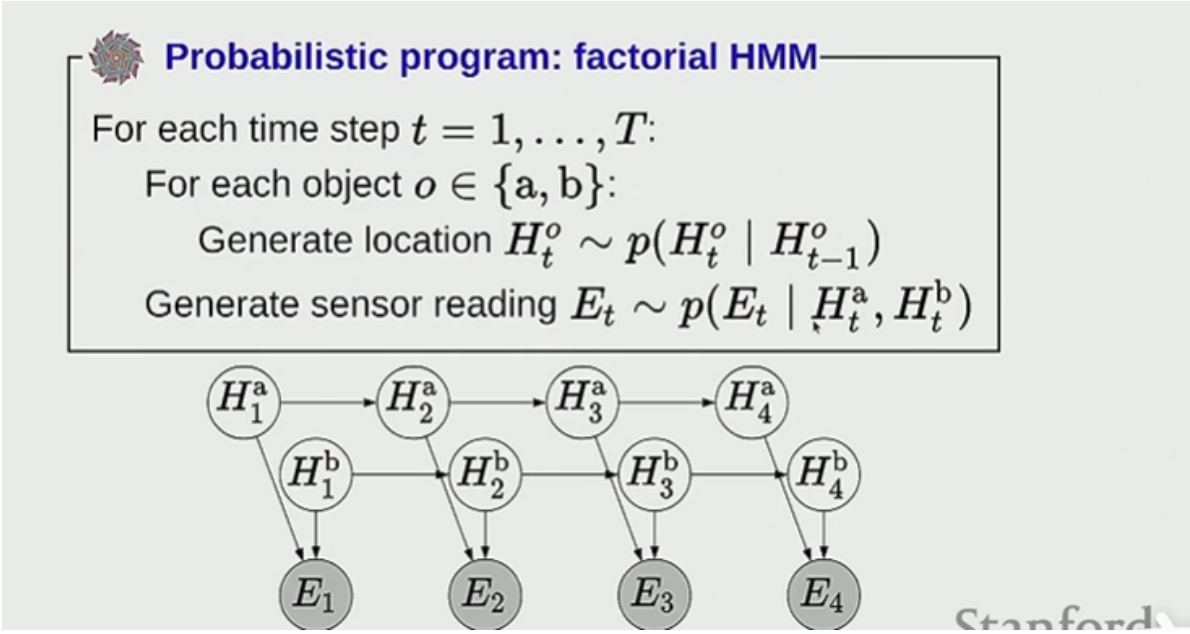
For each position $i = 1, 2, \ldots, n$:
Generate word $X_i \sim p(X_i \mid X_{i-1})$

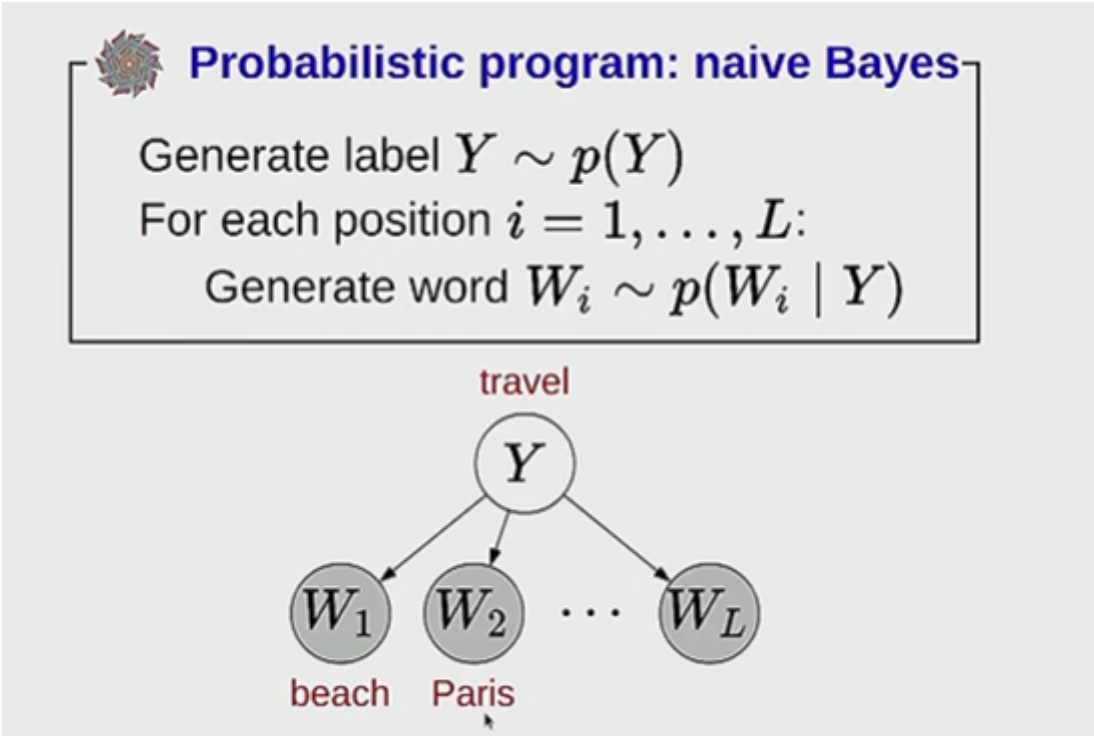


Probabilistic program: hidden Markov model (HMM)

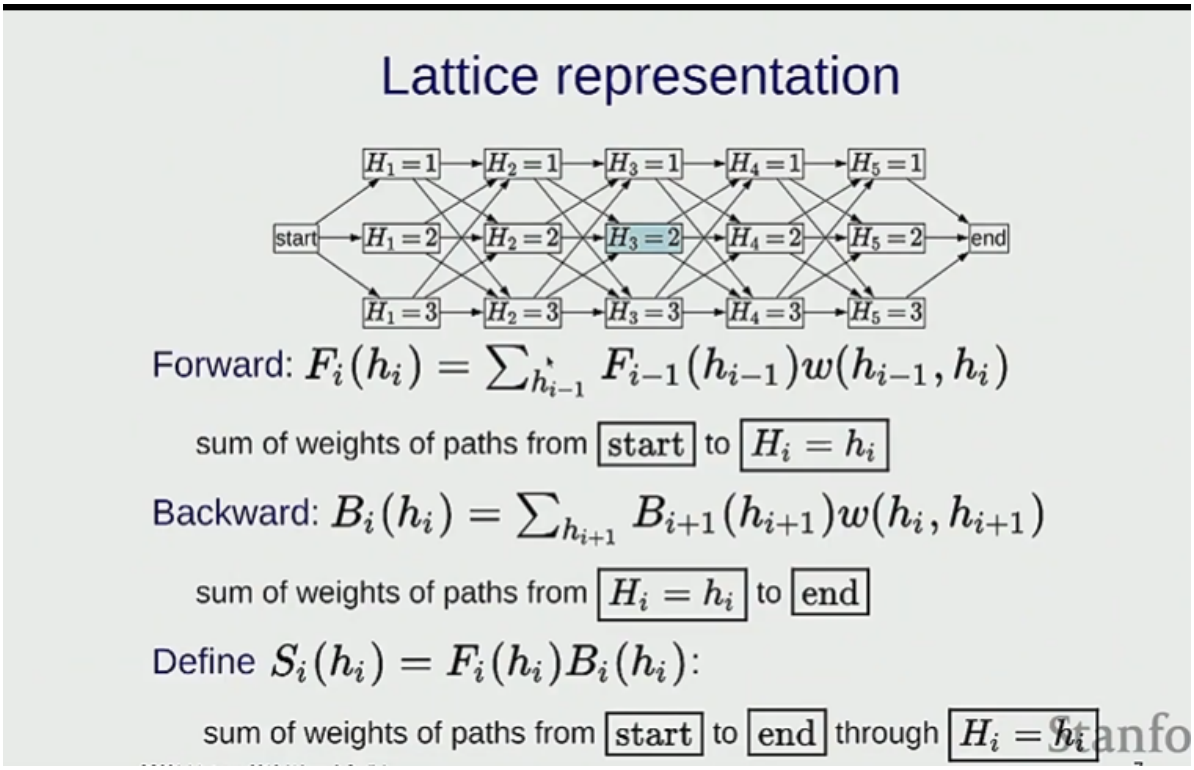
For each time step $t = 1, \ldots, T$:
Generate object location $H_t \sim p(H_t \mid H_{t-1})$
Generate sensor reading $E_t \sim p(E_t \mid H_t)$

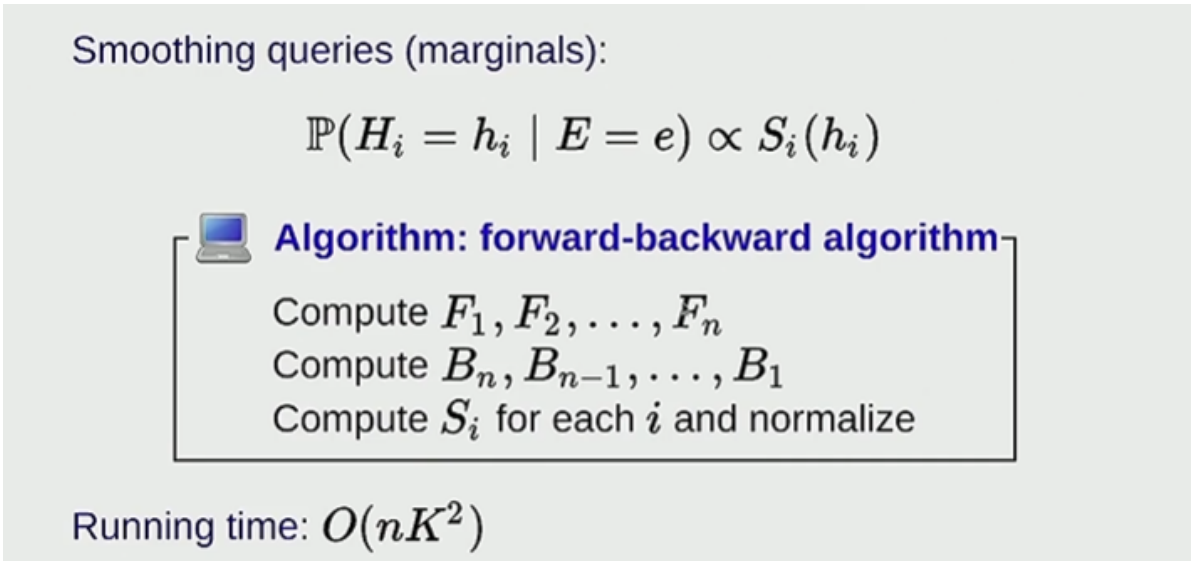






- L14 Bayesian Network 2 - Forward & Backward





(n time steps, K values of H_i, based on Hidden Markov models)

Algorithm:partical filtering

Initialize $C\gets [{}]$

For each i=1,...,n:

Propose(extend):

$C^{\prime}\gets {h\cup{H_i:h_i}:h\in C,h_i\sim p(h_i|h_{i-1})}$

Reweight:

Compute weights $w(h)=p(e_i|h_i)$ for $h\in C^{\prime}$

Resample(prune):

$C\gets K$ elements drawn independently from $\propto w(h)$

Gibbs sampling Algorithm:

Setup: Weight(x)

Initialize x to a random complete assignment

Loop through i=1,...n until convergence:

Compute weight of $x\cup {X_i:v}$ for each v

Choose $x\cup {X_i:v}$ with probability proption (prop.) to weight

probabilistic interpretation:

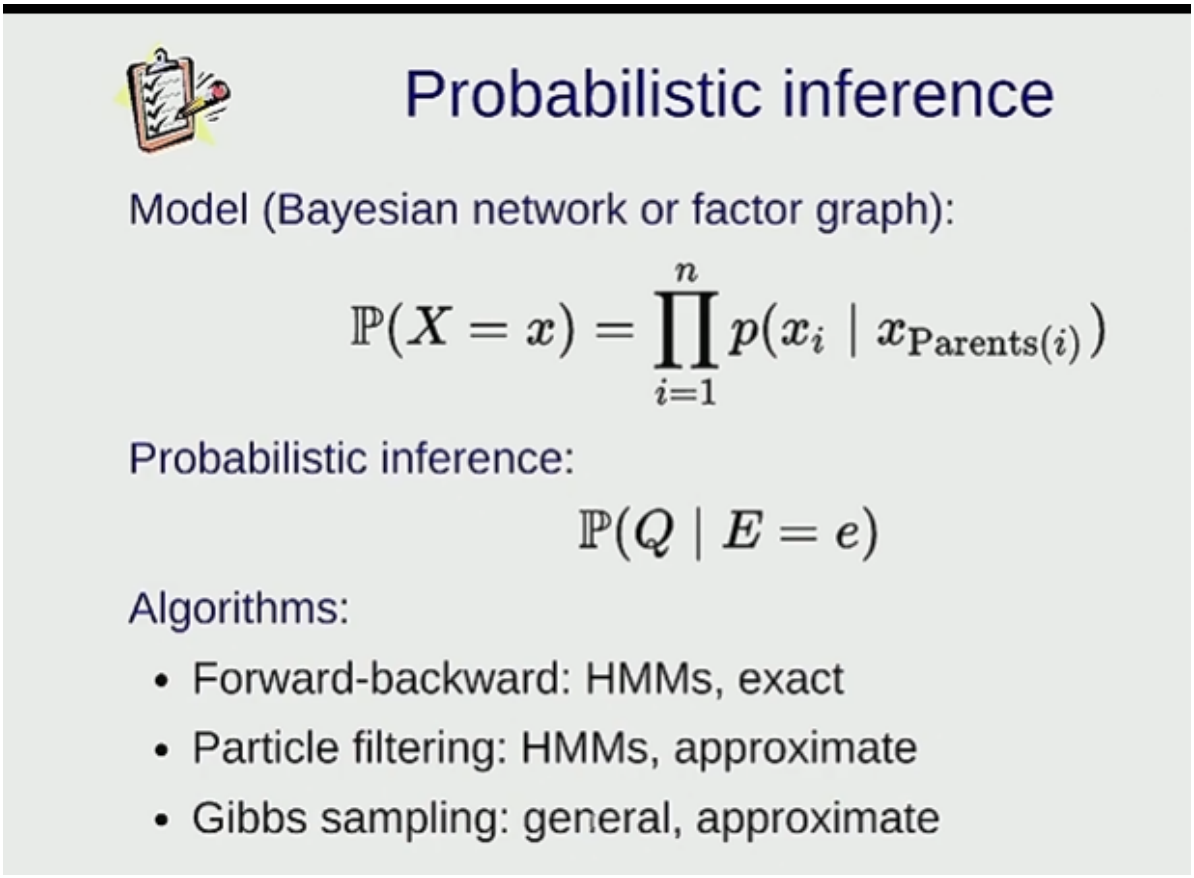
Setup: $P(X=x)\propto Weight(x)$

Intialize x to a random complete assignment

Loop through i=1,...,n until convergence:

Set $X_i=v$ with prob. $P(X_i=v|X{-i}=x{-i})$

Note: X_{-i} denotes all variables except X_i



(e for evidence)

- L15 Bayesian Network 3 - Maximum Likelihood

Supervised learning

Parameter sharing: the local conditional distributions of different variables use the same parameters

General case

Bayesian network: variables $X_1, \ldots, X_n$

Parameters: collection of distributions $\theta = \{p_d : d \in D\}$
(e.g., $D = \{\textbf{start}, \textbf{trans}, \textbf{emit}\}$)

Each variable X_i is generated from distribution p_{d_i}:

$$\mathbb{P}(X_1 = x_1, \ldots, X_n = x_n) = \prod_{i=1}^{n} p_{d_i}(x_i \mid x_{\text{Parents}(i)})$$

Parameter sharing: d_i could be same for multiple i

General case: learning algorithm

Input: training examples $\mathcal{D}_{\text{train}}$ of full assignments

Output: parameters $\theta = \{p_d : d \in D\}$

Algorithm: maximum likelihood for Bayesian networks

Count:
  For each $x \in \mathcal{D}_{\text{train}}$:
    For each variable x_i:
      Increment $\text{count}_{d_i}(x_{\text{Parents}(i)}, x_i)$

Normalize:
  For each d and local assignment $x_{\text{Parents}(i)}$:
    Set $p_d(x_i \mid x_{\text{Parents}(i)}) \propto \text{count}_d(x_{\text{Parents}(i)}, x_i)$

Maximum likelihood objective:

$max_{\theta}\prod_{x\in D_{train}}P(X=x;\theta)$

Laplace smooothing:

For each distribution d and partial assignment

$(x_{Parents(i)},x_i)$, add λ to $count_d(x_{Parents(i)},x_i)$

Then normalize to get probability estimates

Interpretation: hallucinate （幻想）λ occurrences of each local assignment

Larger λ ->more smoothing -> probabilities closer to uniform

Expectation Maximization(EM)

Intuition: generalization of the K-means algorithm

Variables : H is hidden, E=e is observed

Algorithm:

Initialize θ

E-step:

Compute $q(h)=P(H=h|E=e;\theta)$ for each h (use any probabilistic inference algorithm)

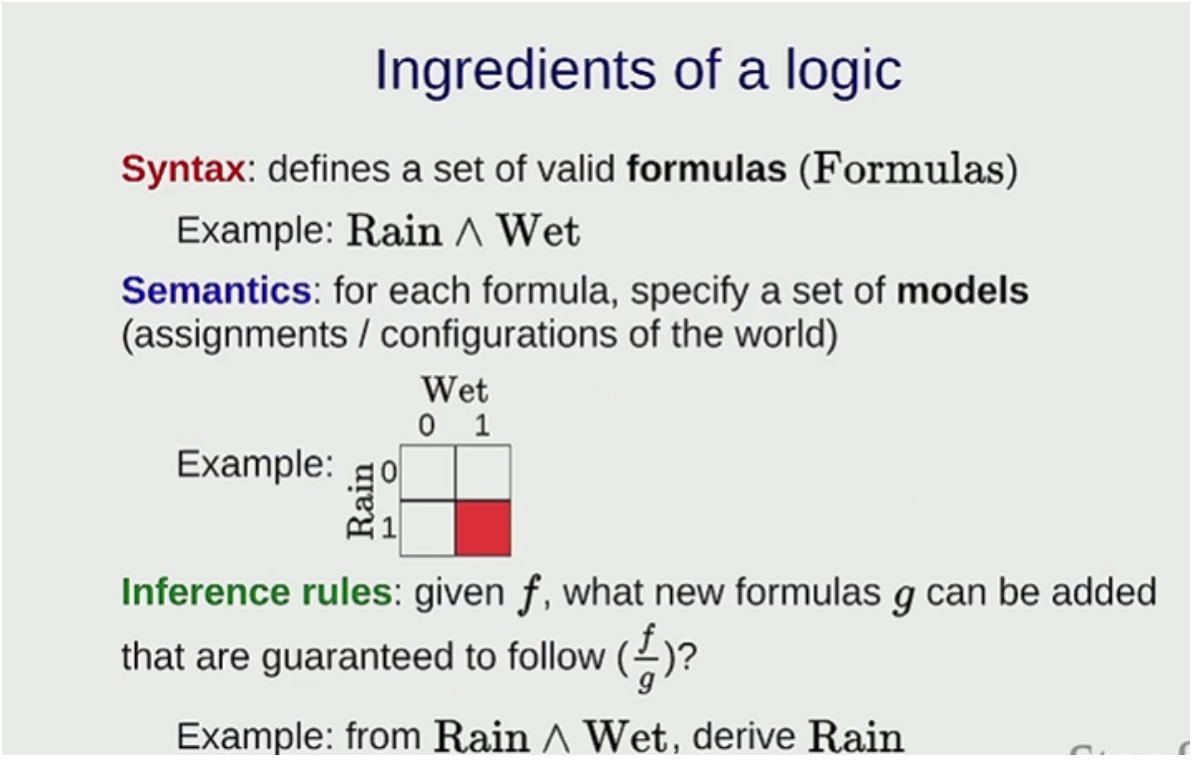
Create weighted points (h,e) with weight q(h)

M-step:

Compute maximum likelihood (just count and normalize) to get θ

Repeat until convergence

- L16 Logic 1 - Propositional Logic



Syntax: what are valid expressions in the language?

Semantics: what do these expressions mean?

Different syntax, same semantics (5):

$$2 + 3 \Leftrightarrow 3 + 2$$

Same syntax, different semantics (1 versus 1.5):

$$3 \ / \ 2 \text{ (Python 2.7)} \not\Leftrightarrow 3 \ / \ 2 \text{ (Python 3)}$$

Propositional symbols (atomic formulas): A, B, C

Logical connectives: $\neg, \wedge, \vee, \rightarrow, \leftrightarrow$

Build up formulas recursively—if f and g are formulas, so are the following:

- Negation: $\neg f$
- Conjunction: $f \wedge g$
- Disjunction: $f \vee g$
- Implication: $f \rightarrow g$
- Biconditional: $f \leftrightarrow g$

Note: Formulas by themselves are just symbols (syntax). No meaning yet (semantics)!

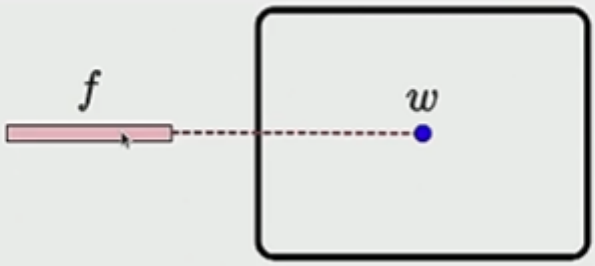
disjunction=逻辑或

A model w in propositional logic is an assignment of truth values to propositional symbols

Definition: interpretation function

Let f be a formula.
Let w be a model.
An **interpretation function** $\mathcal{I}(f, w)$ returns:

- true (1) (say that w satisfies f)
- false (0) (say that w does not satisfy f)



Base case:

- For a propositional symbol p (e.g., A, B, C):
  $\mathcal{I}(p, w) = w(p)$

Recursive case:

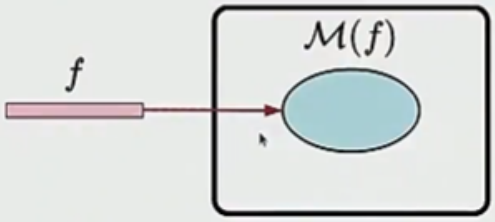
- For any two formulas f and g, define:

$\mathcal{I}(f,w)$	$\mathcal{I}(g,w)$	$\mathcal{I}(\neg f,w)$	$\mathcal{I}(f \wedge g,w)$	$\mathcal{I}(f \vee g,w)$	$\mathcal{I}(f \rightarrow g,w)$	$\mathcal{I}(f \leftrightarrow g,w)$
0	0	1	0	0	1	1
0	1	1	0	1	1	0
1	0	0	0	1	0	0
1	1	0	1	1	1	1

So far: each formula f and model w has an interpretation
$\mathcal{I}(f, w) \in \{0, 1\}$

Definition: models

Let $\mathcal{M}(f)$ be the set of **models** w for which
$\mathcal{I}(f, w) = 1$.



A formula compactly represents a set of models

A knowledge base KB is a set of formulas representing their conjunction/intersection:

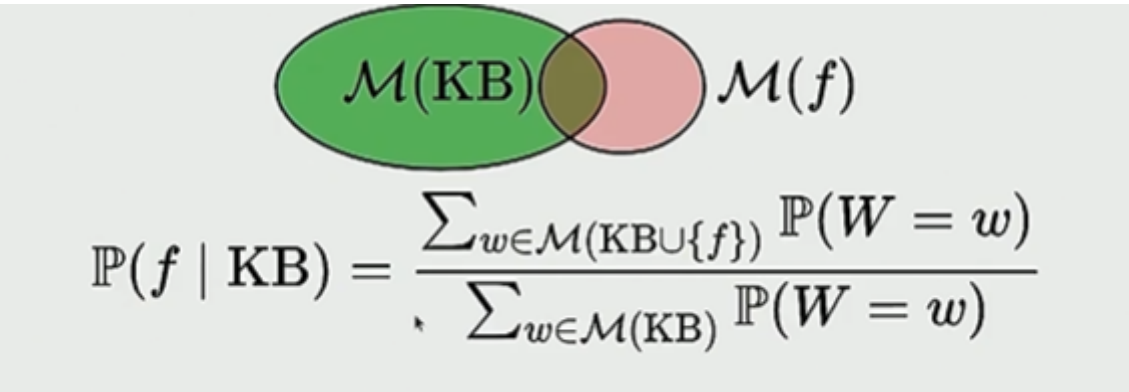
$M(KB)=\bigcap_{f\in KB} M(f)$

Intuition: KB specifies constraints on the world.M(KB) is the set of all worlds satisfying those constraints

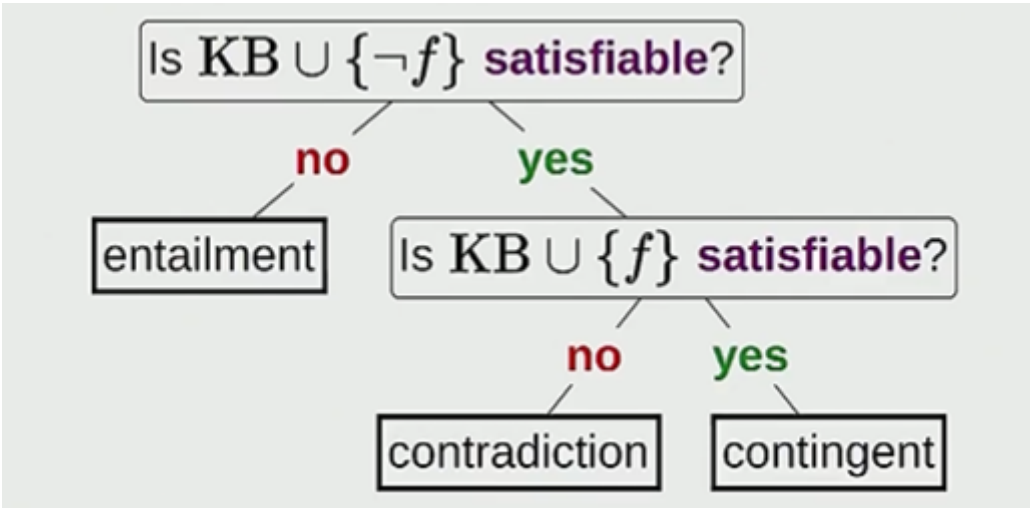
entailment:KB entails f (written $KB\models f$) if $M(KB)\subseteq M(f)$

contradiction: KB contradicts f if $M(KB)\cap M(f)=\emptyset$ or if KB entails $\neg f$

contingency:相交



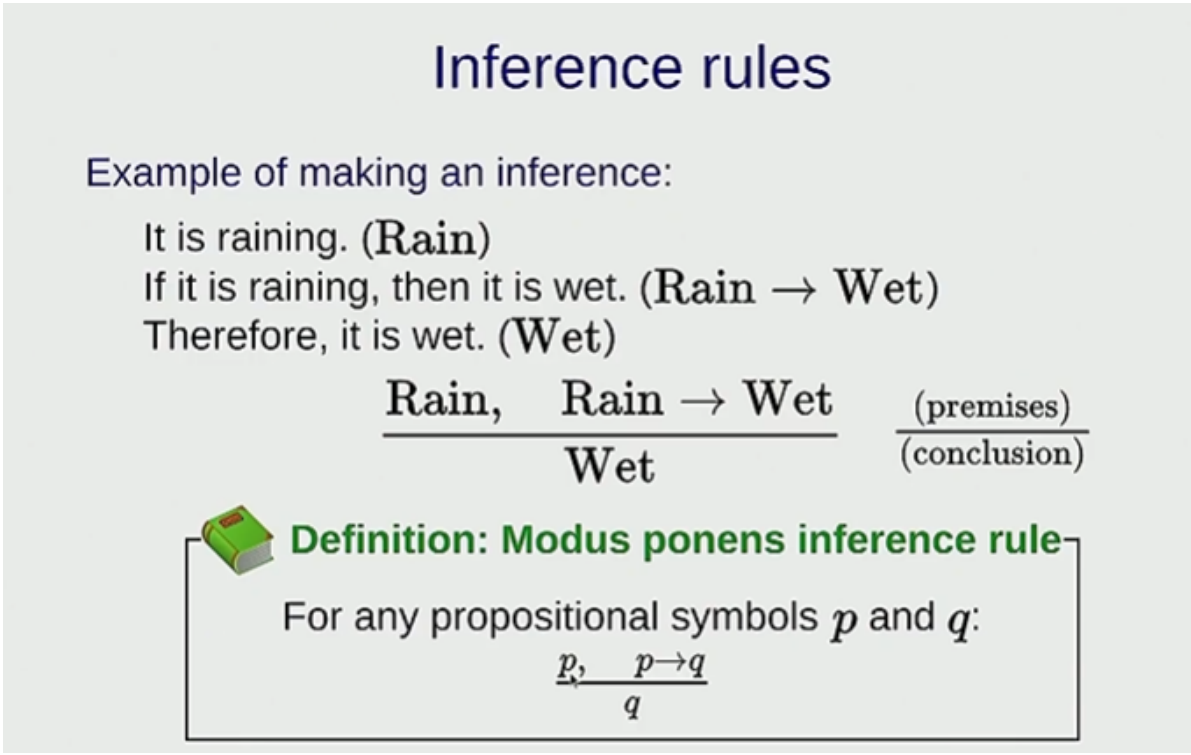
satisfiability: a knowledge base KB is satisfiable if $M(KB)\neq \emptyset$



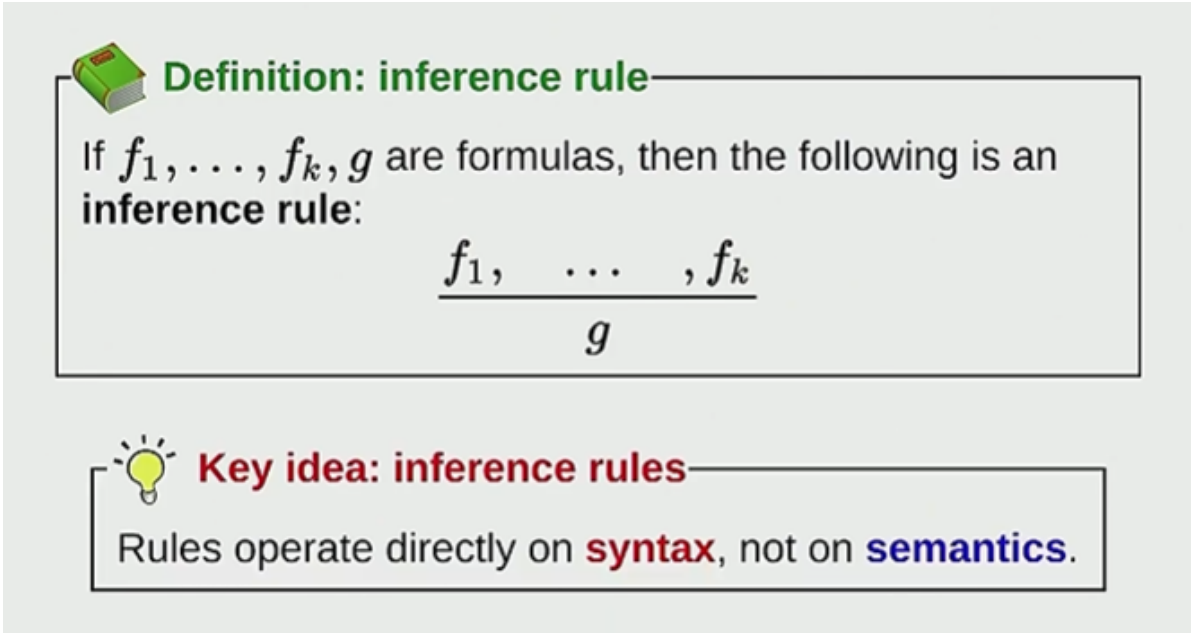
model checking:

Input: knowledge base KB

Output: exists satisfying model ($M(KB)\neq \emptyset$)?



model ponens:肯定前件



Algorithm: forward inference

Input:set of inference rules Rules

Repeat until no changes to KB:

Choose set of formulas $f_1,...,f_k\in KB$

If matching rule$\frac{f_1,...,f_k}{g}$ exists:

Add g to KB

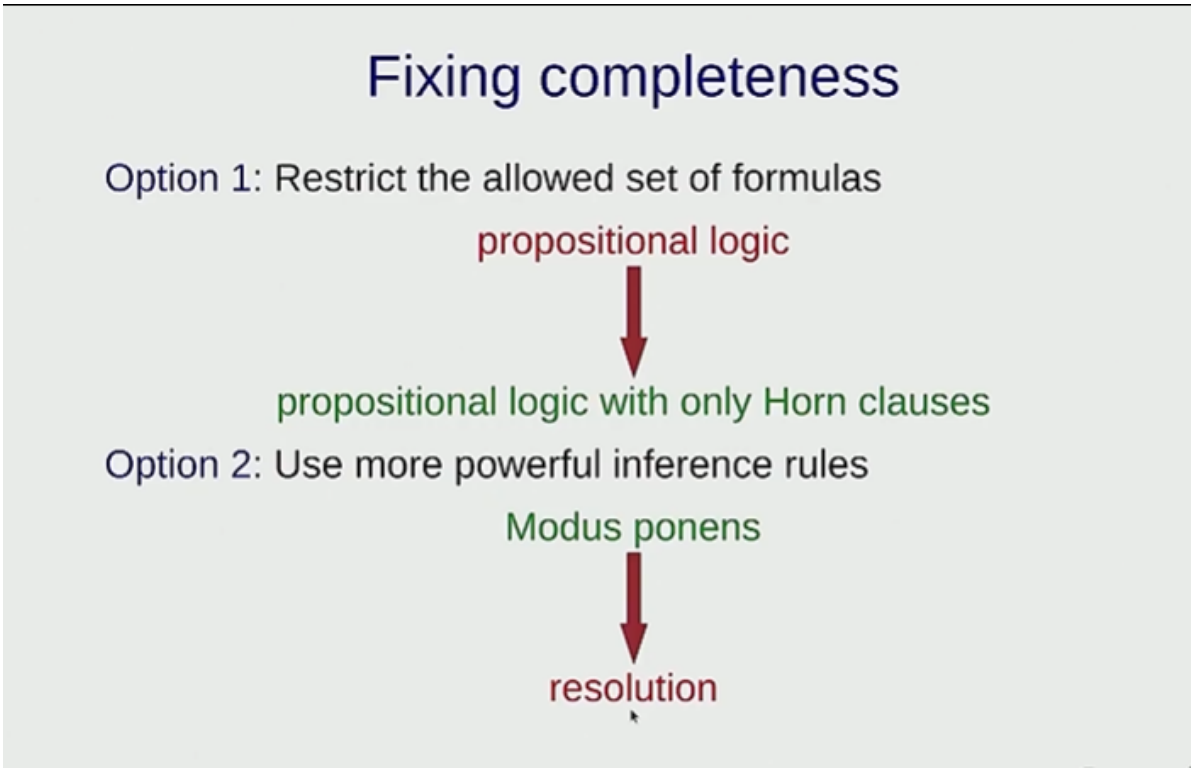
derivation: KB derives/proves f $(KB \vdash f)$ if f eventually gets added to KB

soundness: A set of inference rules Rules is sound if:

${f:KB\vdash f}\subseteq{f:KB\models f} $

soundness: nothing but the truth

completeness: whole truth



Definite clause:

A definite clause has the following form:

$(p_1\wedge...\wedge p_k)\to q$

where $p_1,,...,p_k,q$ are propositional（命题的） symbols

Intuition: if $p_1,...,p_k$ holds, then q holds

Horn clause:

A horn clause is either

a definite clause ($p_1\wedge ... \wedge p_k \to q$)

a goal clause ($p_1\wedge ...\wedge p_k\to false$)

modus ponens can also be written as:

$\frac{p_1,...,p_k,(p_1\wedge...\wedge p_k)\to q}{q}$

Theorem: Modus ponens on Horn clauses

Modus ponens is complete with respect to Horn clauses:

Suppose KB contains only Horn clauses and p is an entailed propositional symbol

Then applying modus ponens will derive p

Upshot: $KB \models p$ (entailment) is the same as $KB\vdash p$ (derivation)

- L17 Logic 2 - First-order Logic

  $A\to C \leftrightarrow \neg A \cup C $

  Literal: either p or $\neg p$,where p is a propsitional symbol

  Clause: disjunction of literals

  Horn clauses: at most one positive literal

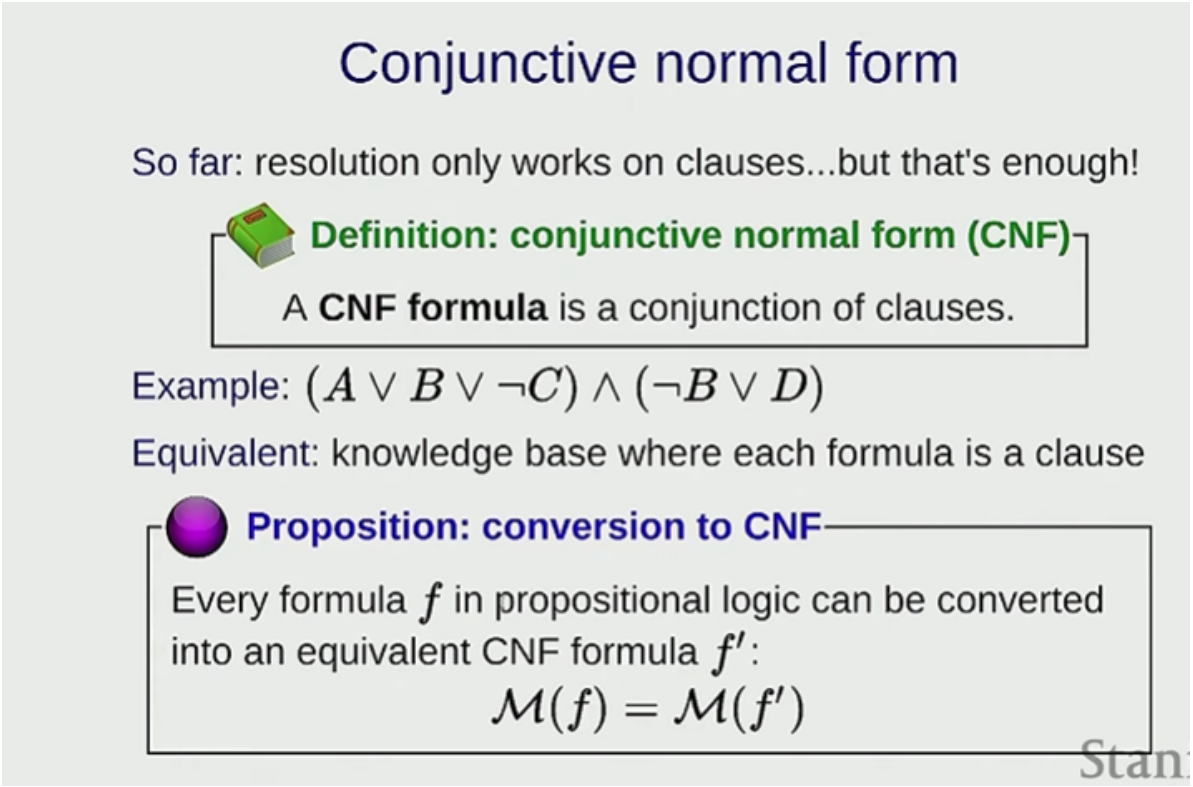
  Modus ponens (rewritten)

  $\frac {A,\neg A\cup C} {C}$

  Intuition: cancel out A and $\neg A$

  Resolution inference rule:

  $\frac {f_1\cup...\cup f_n\cup p,\neg p\cup g_1\cup ...\cup g_m}{f_1\cup...\cup f_n\cup g_1\cup ...\cup g_m}$



Conversion to CNF: general

Conversion rules:

- Eliminate $\leftrightarrow$: $\frac{f \leftrightarrow g}{(f \rightarrow g) \wedge (g \rightarrow f)}$
- Eliminate $\rightarrow$: $\frac{f \rightarrow g}{\neg f \vee g}$
- Move $\neg$ inwards: $\frac{\neg(f \wedge g)}{\neg f \vee \neg g}$
- Move $\neg$ inwards: $\frac{\neg(f \vee g)}{\neg f \wedge \neg g}$
- Eliminate double negation: $\frac{\neg\neg f}{f}$
- Distribute $\vee$ over $\wedge$: $\frac{f \vee (g \wedge h)}{(f \vee g) \wedge (f \vee h)}$

$KB\models f \leftrightarrow KB\cup {\neg f} is \space unsatisfiable$

Algorithm: reslution-based inference:

1. Add $\neg f$ into KB
2. Convert all formulas into CNF
3. Repeatly apply resolution rule
4. Return entailment if derive false

Summary

Horn clauses	any clauses
modus ponens	resolution
linear time	exponential time
less expressive	more expressive

definite clause (first-order logic):

A definite clause has the following form:

$\forall x_1...\forall x_n(a_1\cap...\cap a_k)\to b$

for variables $x_1,...,x_n$ and atomic formulas $a_1,...,a_k,b$ (which contain those variables)

modus ponens (first-order logic):

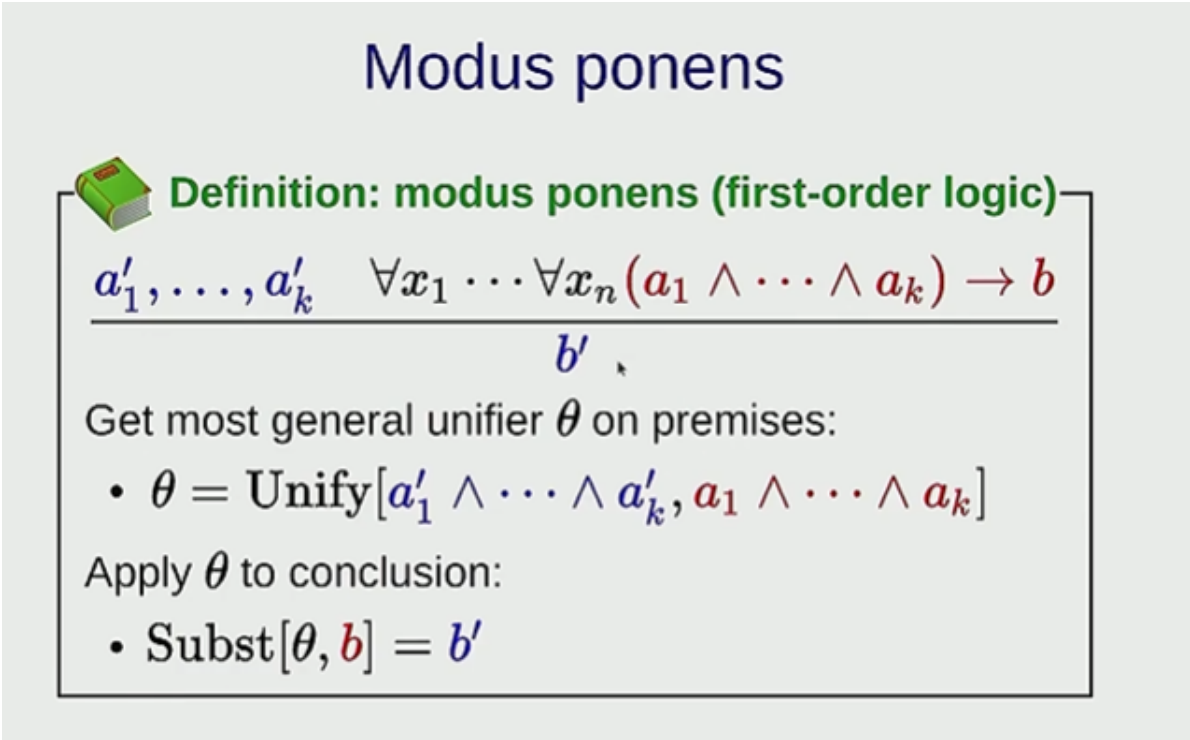
$\frac {a_1,...,a_k\space \forall x_1...\forall x_n(a_1\cap...\cap a_k)\to b}{b}$

A substitution θ is a mapping from variables to terms. $Subst[\theta,f]$ returns the result of performing substitution θ on f

Unification takes two formulas f and g and returns a substitution θ which is the most general unifier:

$Unify[f,g]=\theta$ such that $Subst[\theta,f]=Subst[\theta,g]$ or 'fail' if no such θ exists

Apply unification and subsititution to modus ponens:



Modus ponens is complete for first-order logic with only Horn clauses

Theorem:semi-devidability

First-order logic (even restricted to only Horn clauses) is semi-decidable

If $KB \models f$, forward inference on complete inference rules will prove f in finite time

If $KB \nvDash f$, no algorithm can show this in finite time

Definition: resolution rule (first-order logic)

$$\frac{f_1 \vee \cdots \vee f_n \vee p, \quad \neg q \vee g_1 \vee \cdots \vee g_m}{\text{Subst}[\theta, f_1 \vee \cdots \vee f_n \vee g_1 \vee \cdots \vee g_m]}$$

where $\theta = \text{Unify}[p, q]$.

Summary

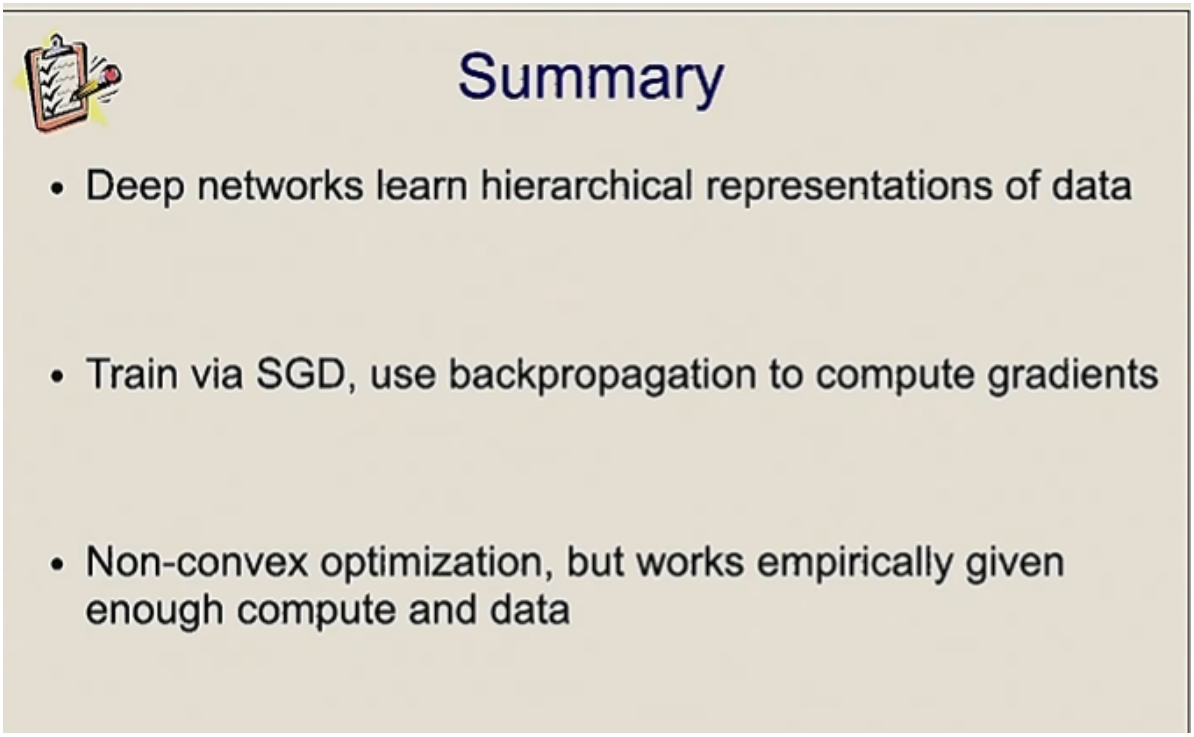
Propositional logic	First-order logic
model checking	n/a
⇐ propositionalization	
modus ponens (Horn clauses)	modus ponens++ (Horn clauses)
resolution (general)	resolution++ (general)

++: unification and substitution

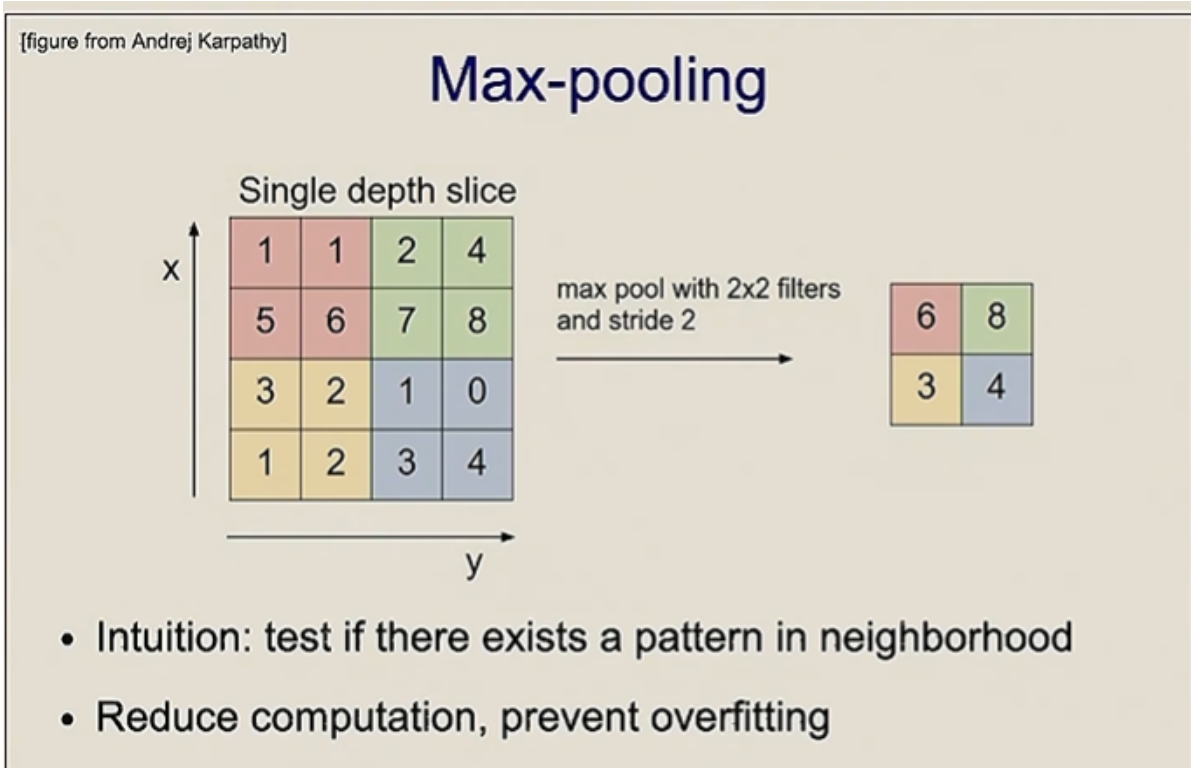
Key idea: variables in first-order logic

Variables yield compact knowledge representations.

- L18 Introduction to Deep Learning

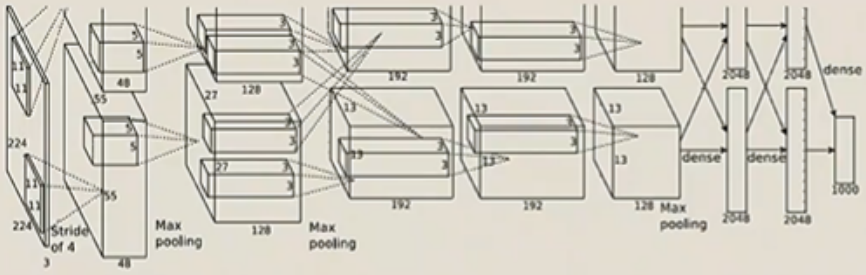


convex optimization 凸优化



[Krizhevsky et al., 2012]

AlexNet

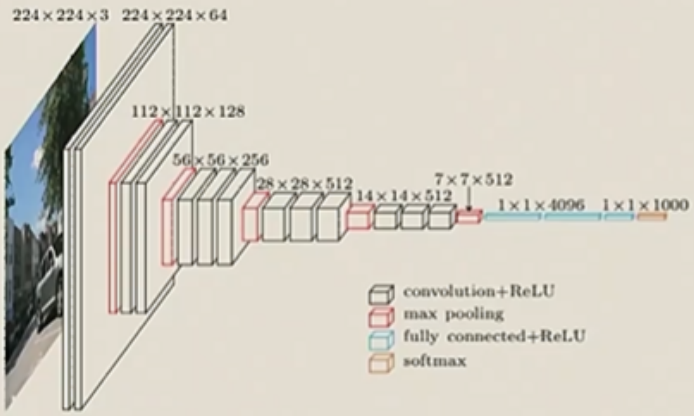


- Non-linearity: use RelU ($\max(z, 0)$) instead of logistic
- Data augmentation: translate, horizontal reflection, vary intensity, dropout (guard against overfitting)
- Computation: parallelize across two GPUs (6 days)
- Results on ImageNet: 16.4% error (next best was 25.8%)

[Simoyan/Zisserman, 2014]

[image credit: Davi Frossard]

VGGNet

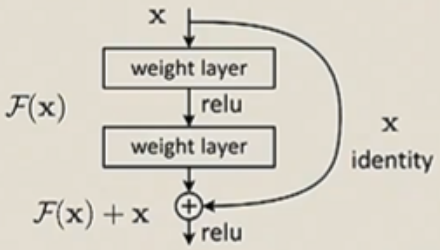


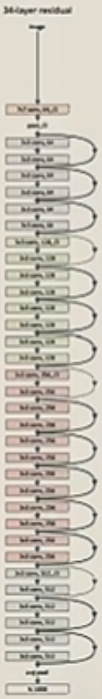
- Architecture: deeper but smaller filters; uniform
- Computation: 4 GPUs for 2-3 weeks
- Results on ImageNet: 7.3% error (AlexNet: 16.4%)

[He et al. 2015]

Residual networks

$$x \mapsto \sigma(Wx) + x$$



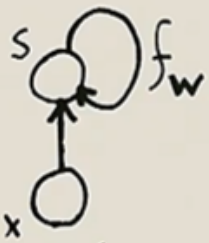


- Key idea: make it easy to learn the identity (good inductive bias)
- Enables training 152 layer networks
- Results on ImageNet: 3.6% error

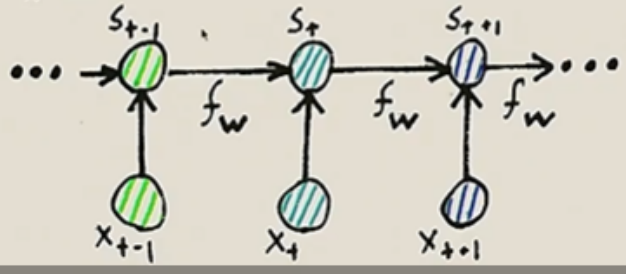
Recurrent neural networks

Formula $s_t = f_W(s_{t-1}, x_t)$

Network

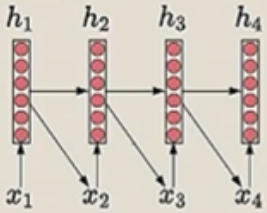


Computation Graph



x_t: hidden layer of time instant t

Recurrent neural networks



$$h_1 = \text{Encode}(x_1)$$
$$x_2 \sim \text{Decode}(h_1)$$
$$h_2 = \text{Encode}(h_1, x_2)$$
$$x_3 \sim \text{Decode}(h_2)$$
$$h_3 = \text{Encode}(h_2, x_3)$$
$$x_4 \sim \text{Decode}(h_3)$$
$$h_4 = \text{Encode}(h_3, x_4)$$

Update context vector:

$$h_t = \text{Encode}(h_{t-1}, x_t)$$

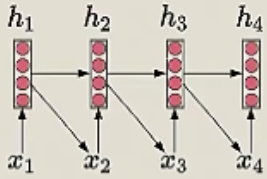
Predict next character:

$$x_{t+1} = \text{Decode}(h_t)$$

context h_t compresses $x_1,...,x_t$

[Elman, 1990]

Simple recurrent network



$$\text{Encode}(h_{t-1}, x_t) = \sigma(V h_{t-1} + W x_t) = h_t$$

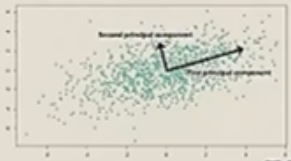
$$\text{Decode}(h_t) \sim \text{softmax}(W' h_t) = p(x_{t+1})$$

Summary

- Recurrent neural networks: model sequences (non-linear version of Kalman filter or HMM)
- Logic intuition: learning a program with a for loop (reduce)
- LSTMs mitigate the vanishing gradient problem
- Attention-based models: when only part of input is relevant at a time
- Newer models with "external memory": memory networks, neural Turing machines

Principal component analysis

Input: points $x_1, \ldots, x_n$



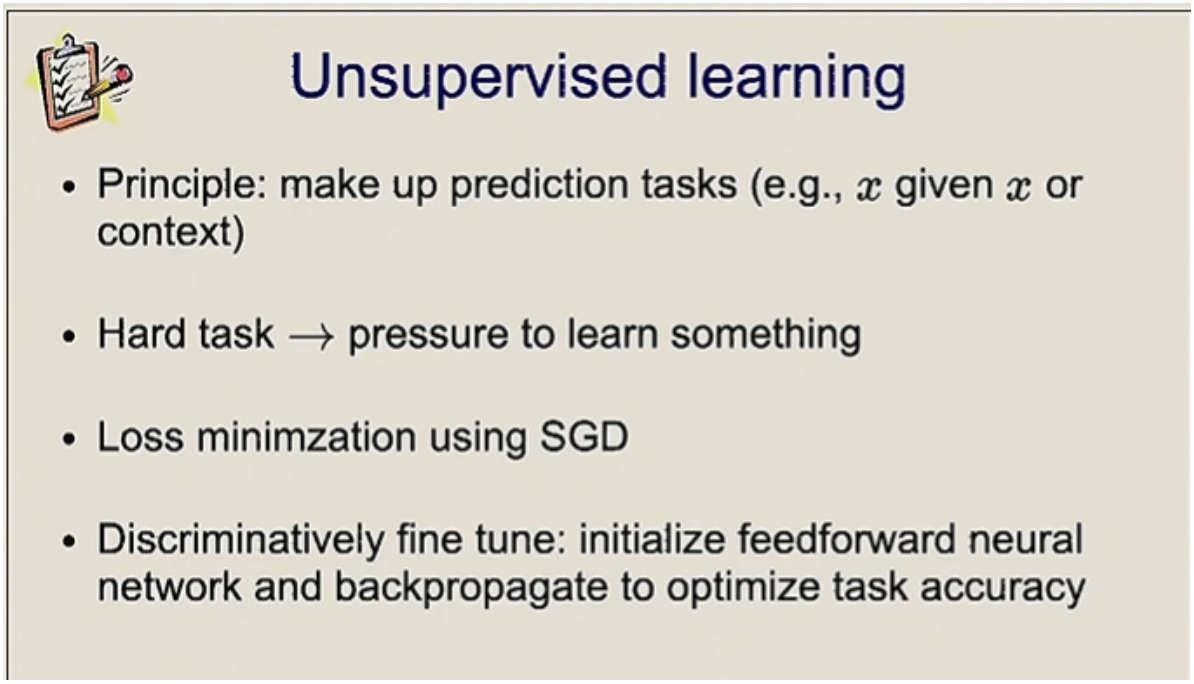
$$\text{Encode}(x) = U^\top x \qquad \text{Decode}(h) = U h$$

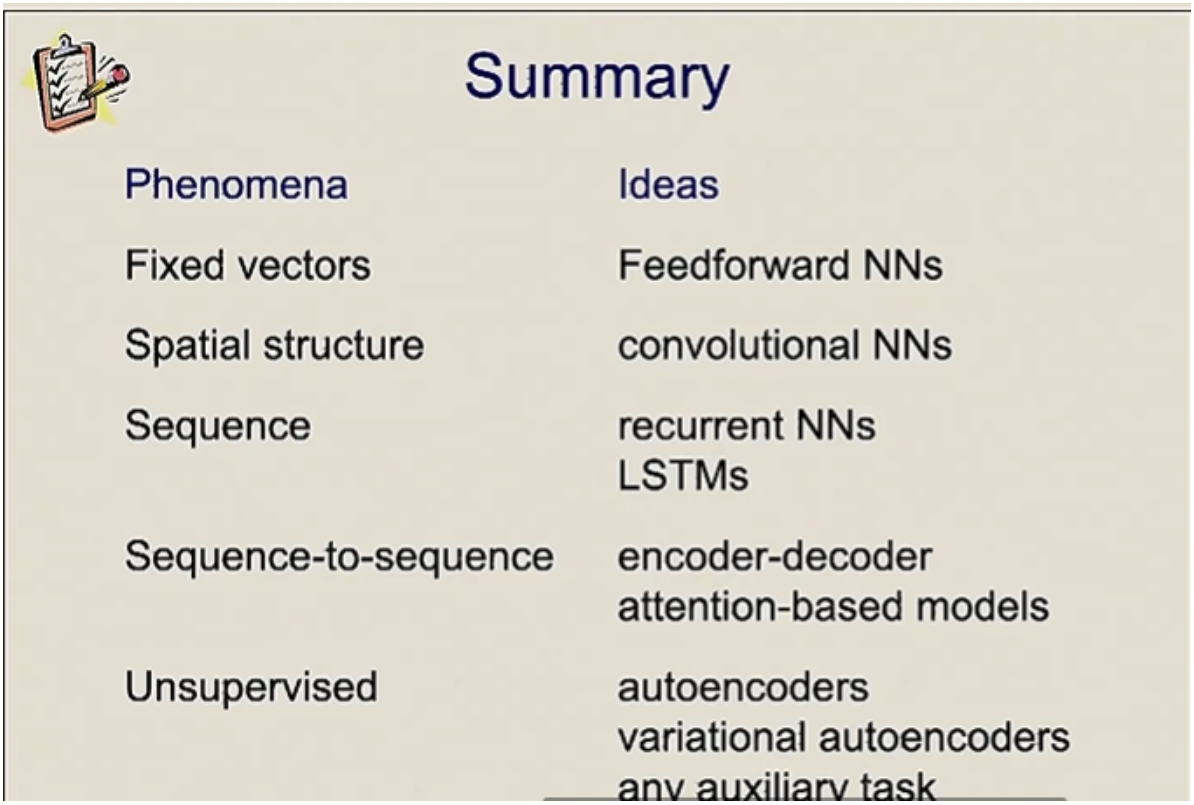
(assume x_i's are mean zero and U is orthogonal)

PCA objective:

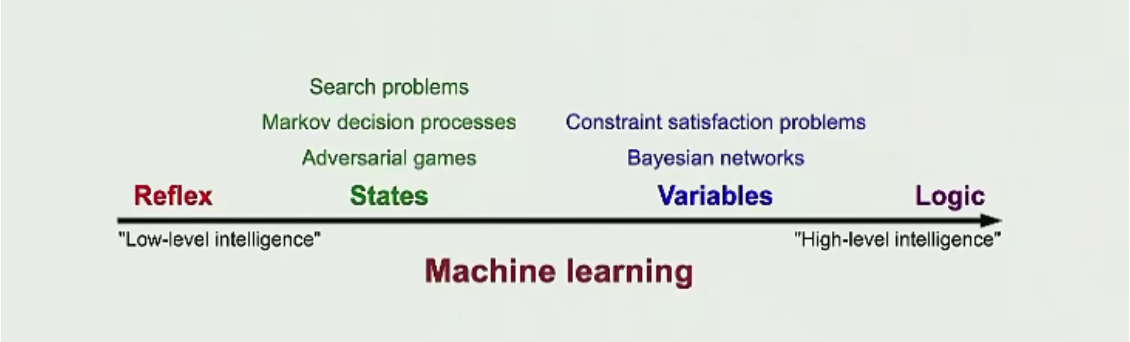
$$\text{minimize} \sum_{i=1}^{n} \|x_i - \text{Decode}(\text{Encode}(x_i))\|^2$$

orthogonal 正交的 正交矩阵满足$AA^T=单位阵E$



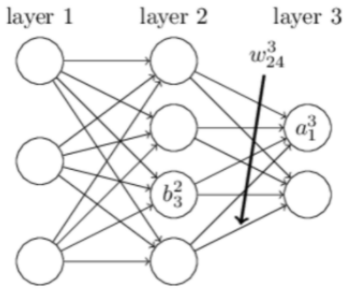


- L19 Conclusion



其他杂项

- 零基础入门DL https://www.zybuluo.com/hanbingtao/note/433855
- 反向传播概念与公式推导https://blog.csdn.net/u014313009/article/details/51039334



上图是一个三层人工神经网络，layer1至layer3分别是输入层、隐藏层和输出层。如图，先定义一些变量：

w^l_{jk}表示第$(l-1)$层的第k个神经元连接到第l层的第j个神经元的权重；

b^l_j表示第l层的第j个神经元的偏置；

z^l_j表示第l层的第j个神经元的输入，即：

$$z^l_j = \sum_k w^l_{jk} a^{l-1}_k + b^l_j$$

a^l_j表示第l层的第j个神经元的输出，即：

$$a^l_j = \sigma\left(\sum_k w^l_{jk} a^{l-1}_k + b^l_j\right)$$

其中σ表示激活函数。

4. 反向传播算法伪代码

- **输入训练集**
- **对于训练集中的每个样本x，设置输入层（Input layer）对应的激活值a^1：**
  - **前向传播：**

$$z^l = w^l a^{l-1} + b^l, \quad a^l = \sigma(z^l)$$

  - **计算输出层产生的错误：**

$$\delta^L = \nabla_a C \odot \sigma'(z^L)$$

  - **反向传播错误：**

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l)$$

- **使用梯度下降（gradient descent），训练参数：**

$$w^l \to w^l - \frac{\eta}{m} \sum_x \delta^{x,l} (a^{x,l-1})^T$$

$$b^l \to b^l - \frac{\eta}{m} \sum_x \delta^{x,l}$$

- 常见的六大聚类(clusting)算法https://blog.csdn.net/weixin_42056745/article/details/101287231

K-means 均值漂移 基于密度（DBSCAN） 用高斯混合模型（GMM）的最大期望（EM）凝聚层次图团体检测(Graph Community Detection)

- 贪心算法https://baike.baidu.com/item/%E8%B4%AA%E5%BF%83%E7%AE%97%E6%B3%95/5411800?fr=aladdin

利用贪心法求解的问题应具备如下2个特征。

1、贪心选择性质

一个问题的整体最优解可通过一系列局部的最优解的选择达到，并且每次的选择可以依赖以前作出的选择，但不依赖于后面要作出的选择。这就是贪心选择性质。对于一个具体问题，要确定它是否具有贪心选择性质，必须证明每一步所作的贪心选择最终导致问题的整体最优解。

2、最优子结构性质

当一个问题的最优解包含其子问题的最优解时，称此问题具有最优子结构性质。问题的最优子结构性质是该问题可用贪心法求解的关键所在。在实际应用中，至于什么问题具有什么样的贪心选择性质是不确定的，需要具体问题具体分析。

- CLIP相关https://www.zhihu.com/question/438649654
- FID https://baijiahao.baidu.com/s?id=1647349368499780367&wfr=spider&for=pc 对于使用了GAN的network来说，FID越低效果越好
- 神经网络输出尺寸的基本运算方法（以CNN为例） http://www.51zixue.net/deeplearning/589.html
- 前向传播与反向传播https://blog.csdn.net/bitcarmanlee/article/details/78819025
- focal loss https://www.cnblogs.com/king-lps/p/9497836.html
- curriculum learning 从易到难的学习 https://zhuanlan.zhihu.com/p/81455004
- Ensemble learning 集成学习

  https://blog.csdn.net/qq_36330643/article/details/77621232

  https://zhuanlan.zhihu.com/p/27689464

- pytorch dataloader 中的shuffle

  https://blog.csdn.net/qq_20200047/article/details/105671374

  https://blog.csdn.net/qq_35248792/article/details/109510917

- 多层感知机MLP

  https://blog.csdn.net/fg13821267836/article/details/93405572

  https://blog.csdn.net/flykinghg/article/details/72677273

  https://www.cnblogs.com/QinHaotong/p/9571122.html