深度学习

第一章 概述

略

第二章 机器学习基础

2.1 数据准备和预处理

数据集由一个个样本组成，每个样本由特征组成，通常将一个样本表示成一个向量$x \in R^n$

此部分分为以下四步：数据收集->数据清洗->特征工程->数据集划分

数据收集要具有代表性和多样性；清洗要注意处理缺失值、异常值、噪声、重复数据；特征工程要注意标准化，并去掉无关特征；数据集划分为训练集，验证集（用于模型选择和超参数调节）和测试集（最终测试，只能用一次）。

2.2 模型构建

选择合适的模型：即学习算法和超参数配置

选择损失函数：通过计算模型预测值$\hat{y}$与真实值y之间的平方差来衡量误差，以下两个损失函数均为最小化目标

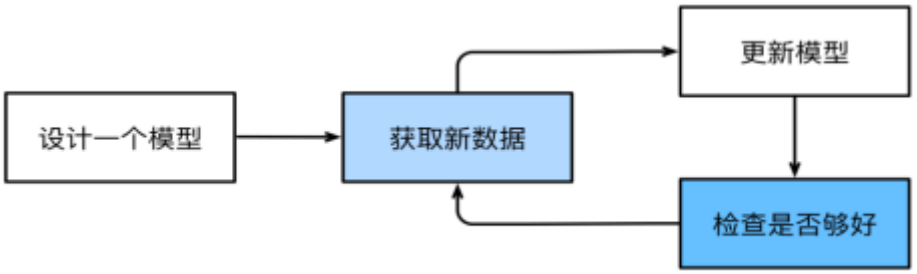
- 均方误差：$Loss = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$
- 二分类交叉熵误差(一般用于分类问题)：$Loss = -\frac{1}{n}\sum_{i=1}^{n}\left[y_i \log\left(\hat{y}_i\right) + (1-y_i)\log\left(1-\hat{y}_i\right)\right]$

参数：模型中需要从数据中学习的内部变量；

超参数：用于管理机器学习模型的外部配置变量，如网络层数、宽度、学习率等，深度学习领域的“调参”，就是指超参数调优；

2.3 模型训练

训练过程如下：



2.4 测试与泛化

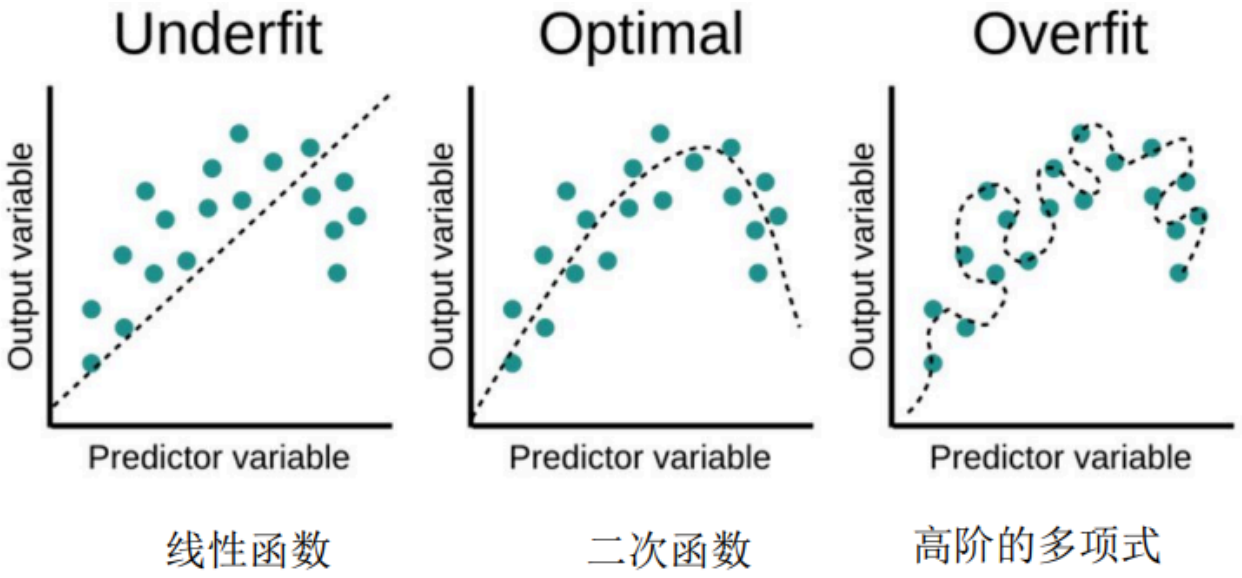
泛化：在未观测到的输入上表现良好的能力称为泛化能力。

独立同分布假设：数据集中的每个样本都是彼此相互独立的，训练集和测试集是同分布的，即采样自相同的分布，从而使模型在训练阶段学到的规律能够在测试数据上有效泛化，而不会因为分布差异而误导训练过程。

过拟合：模型不能在训练集上获得足够低的训练误差，对训练样本的一般性质都没有学到

欠拟合：训练误差和和测试误差之间的差距太大

模型复杂度：模型的容量，是指其拟合各种函数的能力，线性模型的容量是线性函数空间，深度神经网络的容量是非线性函数空间，如果用线性模型很难拟合非线性函数，用容量高的模型可能造成过拟合，如下图，对于二次函数而言，一次函数欠拟合，高阶多项式过拟合。



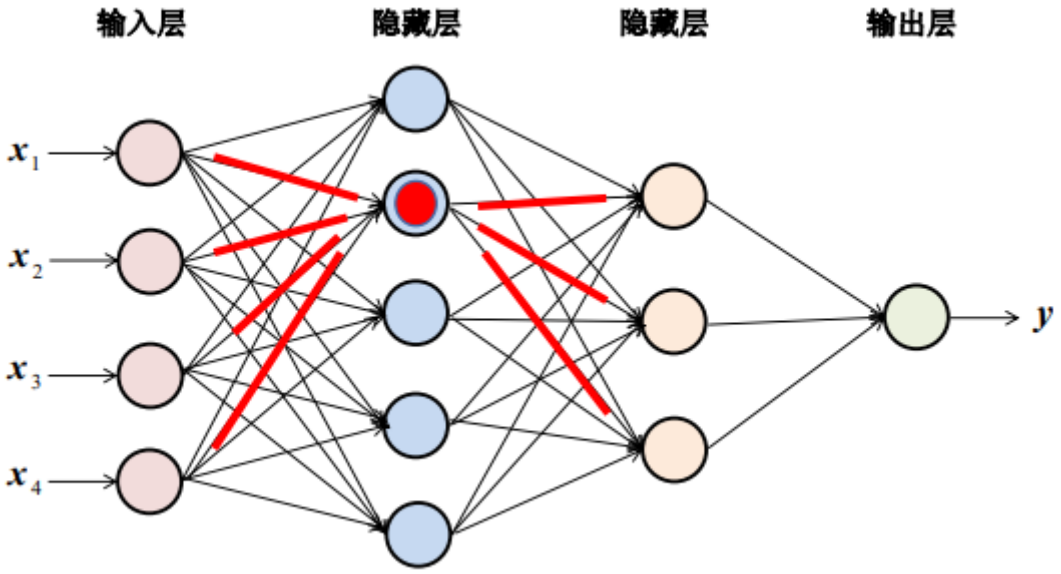
影响模型泛化的因素

参数的数量（自由度）：参数很大时，容量很大，模型往往更容易过拟合；

参数的取值范围：当参数的取值范围较大时，容量更大，模型可能更容易过拟合；

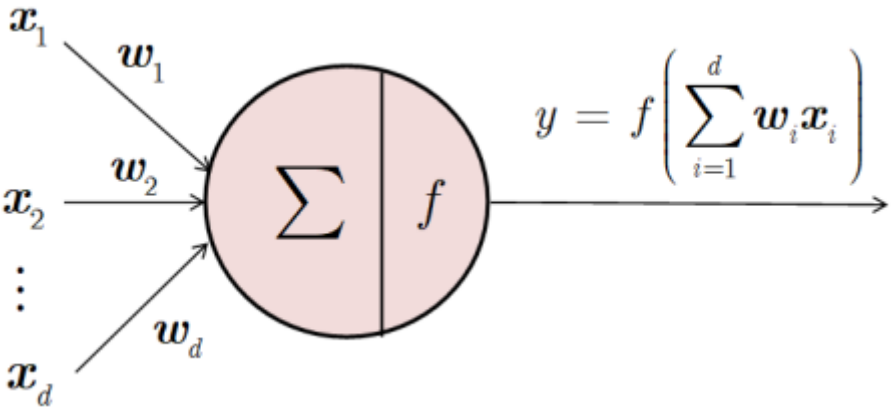
训练样本的数量：样本数越多，越不容易发生过拟合。

第三章 全连接网络



（注：层数只统计隐藏层和输出层）

人工神经元结构如下：



3.1 隐藏层

向量输入为$x \in R^{d_0}$，隐藏层输出为$h_1 \in R^{d_i}$，则第一个权重矩阵为$W_1 \in R^{d_1 \cdot d_0}$ ，偏置为$b_1 \in R^{d_1}$，则第一层输出为$\sigma(h_1 = W_1 \cdot x + b_1)$，外层的$\sigma$为激活函数。多个隐藏层可以实现对输入特征的多层次抽象， 从而更好地处理数据，并不是隐藏层越多越好：层数变多导致计算成本爆炸式增长，且存在边际效用递减规律，甚至过拟合。

3.2 激活函数

神经元输入与输出之间的一种函数变换，目的是为了加入**非线性**因素，增强模型的表达能力。激活函数的性质为：非线性函数、连续并可导，要尽可能的简单。若不引入激活函数相当于简单线性网络，线性函数和线性函数的复合仍然是线性函数。

ReLU：$ReLU(x) = max(x, 0)$；优点是计算简单，计算速度快；缺点是当输入为负数时，输出及导数都为零，不会更新权重。

Softplus：$Softplus（x）= log(1 + e^x)$, 与 ReLU 相比，SoftPlus具有平滑性，但是由于涉及指数运算，数值计算上存在不稳定性。

Sigmoid：$Sigmoid = \frac{1}{1+e^{-x}}$，是0-1阈值函数的一个常用近似适用于将预测概率作为输出的模型或需要门控机制的模型（二分类和LSTM等）。其导数等于$Sigmoid \cdot (1 - Sigmoid)$容易计算，但是对于大部分输入，sigmoid的导数接近为0，反向传播时让大部分信息流消失，容易导致梯度消失。

tanh：tanh(x) = $\frac{1-e^{-2x}}{1+e^{-2x}}$，将输入压缩到-1~1范围内，与Sigmoid类似，其导数也存在梯度消失等问题。

GeLU：常用于GPT、BERT等大语言模型，表达式较复杂。

3.3 万能逼近定理

一个具有线性输出层和“挤压”性质的激活函数（例如sigmoid）的隐藏层的两层前馈神经网络，只要给予网络足够数量的隐藏单元（即网络足够宽），它可以以任意的精度来近似从一个有限维空间到另一个有限维空间的任何Borel可测函数。

3.4 损失函数

均方误差损失MSE：见上；

平均绝对误差损失MAE：$\frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$，也称为L1损失，相比于MSE，MAE对异常值不太敏感；因为没有二次项。

平滑L1损失 Huber：当误差较大时采用MAE，误差较小时采用MSE，避免了MSE对异常值过于敏感，MAE在0处不可导的问题。

$$\ell_i = \begin{cases} |y_i - \hat{y}_i| - \frac{\beta}{2}, & if|y_i - \hat{y}_i| \geq \beta \\ \frac{1}{2\beta}(y_i - \hat{y}_i)^2, & if|y_i - \hat{y}_i| < \beta \end{cases}$$

交叉熵损失：$-\frac{1}{n}\sum_{i=1}^{n}\sum_{i=1}^{d} y_{i,j}\log(\hat{y}_{i,j})$，也可以表示为$-\frac{1}{n}\sum_{i=1}^{n} log(\hat{y_{i,r}})$，因为在独热编码情境下，只有正确类别的$y_{i,r}$才为1，其余均为0。

Softmax 函数：$Softmax(h)_i = \frac{e^{h_i}}{\sum_j^k e^{h_j}}$ ，为每一个标签给出一个概率形式。

KL散度：使用交叉熵作为损失函数和使用KL散度作为 损失函数在指导训练时是等价的。

3.5 正则化

指让学习算法降低泛化误差，而不仅仅是训练误差。

权重衰减：l_2正则化:在损失函数基础上加上$\frac{\lambda}{2}\cdot\sum||W||_F^2$，其中矩阵的F范数的平方是指矩阵所有元素的平方和，从而避免矩阵中某个权重过大使得x_i的微小扰动对输出影响很大。

Dropout:训练过程中，通过随机丢弃一部分神经元（同时丢弃其对应的连接边）来避免过拟合的做法,设置一个固定的概率 p 来判断是否需要丢弃，靠近输入的地方丢弃概率较小，测试阶段，一般不丢弃神经元，所有神经元都激活。Dropout减少了神经元之间的依赖性，输出层的计算也无法过度依赖任何一个神经元，从而减少过拟合。

DropConnect：训练阶段随机丢弃神经元之间的链接，与Dropout类似。

BatchNorm：数据预处理时，一般会进行标准化处理，使其平均值为0，方差为1；其是对同一批次内某一特征进行规范化处理，比如该批次内有n个样本，某个样本经过一个隐藏层后变成隐藏层中的h个特征，则对这n个样本的该特征进行$\frac{x_i-\mu}{\sigma}$操作，然后进行激活操作，即$h = \sigma(BN(Wx + b)$。

避免了极端大小的输出出现，即使是对于sigmoid，也可以缓解梯度消失，减小了不同层之间参数的依赖关系，使模型更鲁棒。

LayerNorm：与BatchNorm不同，LayerNorm式对同一个样本在特征维度上做归一化，对于其多个特征，进行求均值方差等操作进行归一化。BatchNorm需要保留训练阶段的滑动平均（多个batch内的均值的耦合信息，更新公式为running_mean←(1-α)running_mean+$\alpha\mu_B$）。而LayerNorm则不需要。

提前终止：如果我们返回使验证集误差最低的参数设置，就可以获得更好的模型（因此，有希望获得更好的测试误差）

数据增强：由于数据量有限因此可以对图像截取其中一部分，翻转、平移、旋转、缩放，随机改变图像的亮度、对比度、色调，添加噪声等操作，添加到数据集中。在输入层中随机生成一些假样本注入少量噪声，向隐藏层施加噪声也是可行的。

第四章 反向传播

4.1 导数基础

对于矩阵形式的导数，其求导结果如下：

$$x=\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \qquad y=\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}$$

$$\frac{\partial y}{\partial x}=\begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_1}{\partial x_2} & \cdots & \frac{\partial y_1}{\partial x_n} \\ \frac{\partial y_2}{\partial x_1} & \frac{\partial y_2}{\partial x_2} & \cdots & \frac{\partial y_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial y_m}{\partial x_1} & \frac{\partial y_m}{\partial x_2} & \cdots & \frac{\partial y_m}{\partial x_n} \end{bmatrix}$$

4.2 标量形式的反向传播

考虑流经输入层、隐藏层、输出层的信息都是标量的简化网络：

$$\begin{gathered} h_1=\sigma(\omega_1\cdot x+b_1) \\ h_2=\sigma(\omega_2\cdot h_1+b_2) \\ h_3=\sigma(\omega_3\cdot h_2+b_3) \\ \cdots\cdots \\ y=h_L=\sigma(\omega_L\cdot h_{L-1}+b_L) \end{gathered}$$

则最后的目标可以表示为$y=f(\omega,b)$在随机梯度下降中，每次迭代随机选择一个小批量样本，计算每个样本的梯度，然后取平均，这里以单个样本为例，我们的目标是求解$\frac{\delta f}{\omega_i}$和$\frac{\delta f}{b_i}$，那么我们需要保存每一个$z_i=\omega_i\cdot h_{i-1}+b_i$，那么反向传播过程中，导数的计算公式如下：

$$\frac{\delta f}{\delta h_i}=\frac{\delta f}{\delta h_L}\cdot\frac{\delta h_L}{\delta z_L}\cdot\frac{\delta z_L}{\delta h_{L-1}}\cdot\frac{\delta h_{L-1}}{\delta z_{L-1}}\cdots\frac{\delta z_{i+1}}{\delta h_i}$$

那么，可以得出如下的导数计算公式：

$$\frac{\delta f}{\delta \omega_i} = \frac{\delta f}{\delta h_i} \cdot \frac{\delta h_i}{\delta z_i} \cdot h_{i-1}$$
$$\frac{\delta f}{\delta \omega_i} = \frac{\delta f}{\delta h_i} \cdot \frac{\delta h_i}{\delta z_i}$$

在前向传播过程中，记录每一个h_i和z_i的值，那么反向传播过程如下：

$$\text{for } k = L, L-1, \cdots, 1$$
$$\frac{\partial f}{\partial w_k} = \frac{\partial f}{\partial h_k}\sigma'(z_k)h_{k-1}$$
$$\frac{\partial f}{\partial b_k} = \frac{\partial f}{\partial h_k}\sigma'(z_k)$$
$$\frac{\partial f}{\partial h_{k-1}} = \frac{\partial f}{\partial h_k}\sigma'(z_k)w_k$$
$$\text{end for}$$

向量/矩阵形式的矩阵传播涉及到矩阵求导运算，在矩阵论中一般每经过一次求导都要进行一次转置，感兴趣的同学可以自行查阅，考试不涉及。

4.3 梯度消失和梯度爆炸

将递归形式的梯度计算写成连乘形式,如果$\sigma'(z_k)$或ω_k较小，那么经过累乘后梯度值可能会以指数形式衰减并趋近于零，导致梯度消失，sigmoid函数的导数对于大多数输入均为0，即使最大也只有0.25，所以会导致梯度消失。

改善梯度消失的方法：采用ReLU，其导数在输入为+时，为1；或者采用BN对每一层进行标准化，使其输出在和范围合理内，从而即使是对于sigmoid，将其输入标椎化在0附近，其导数不会接近0；或者ResNet的旁支连接。

深层网络中的梯度在传播过程中也可能因链式法则的连乘效应而迅速增长，甚至呈指数级增长，导致网络参数更新过大，网络不稳定。

解决梯度爆炸的方法：采用正则化，避免ω_i过大；BN标准化；或者梯度截断。

第五章 卷积网络

5.1 卷积

卷积网络是一种专门用来处理具有类似网格结构的数据的神经网络，核心思想是利用卷积操作来提取输入数据的局部特征。

卷积与互相关函数的区别：卷积是对称运算，卷积核的左上角与输入图像的右上角做乘后进行求和，是对称的，而互相关是对应位置直接做乘后求和，由于卷积核是从数据中学习到的，因此无论执行的是卷积 运算还是互相关运算，卷积层的输出都不受影响。

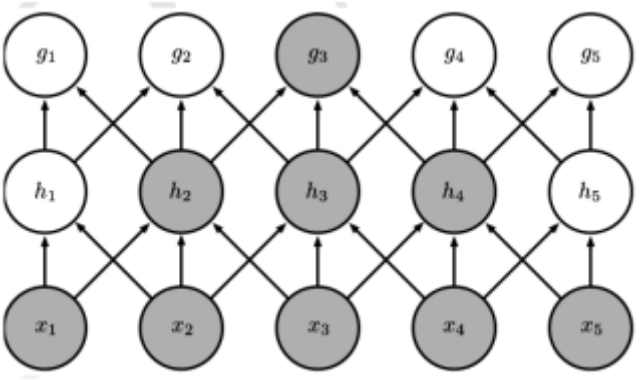
5.2 卷积层

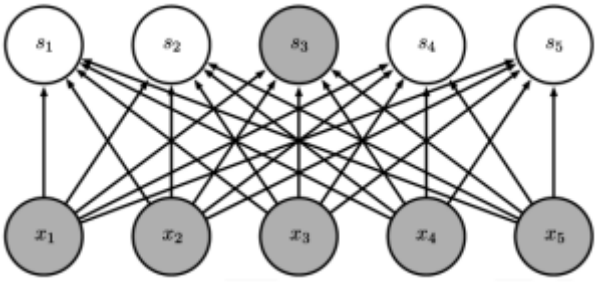
卷积层对输入和卷积核权重进行互相关运算，并加上偏置，卷积层的两个被训练的参数是卷积核权重和偏置，卷积层是一种特殊的全连接层。

5.3 卷积的特性

稀疏连接，参数共享，平移不变性。

稀疏连接：卷积网络中，每个输出只和部分输入产生交互，当卷积核大小远小于输入时，输出和输入之间的连接是稀疏的，而全连接网络的每一层中，每一个输出和所有输入都产生交互。感受野：每个神经元关心的输入神经元范围；只要网络够深，很小的感受野也能检测到很大的输入范围。从下图可以发现，我们可以通过多层的卷积层，在稀疏连接的基础上实现与全连接相同的感受野。





参数共享：输出与输入之间的卷积映射对所有输出共享相同的参数，即每一层只使用一个卷积核；以上图为例，如果x->h时，忽略边界部分的连接缺失，每一个h的感受野为3，则我们需要3×5个参数，如果采用参数共享则只需要3个参数，全连接则需要5×5个参数。降低网络的训练难度，降低模型复杂度，防止过拟合，而由于同样的模式（例如鸟嘴模式）可能会出现在图像的不同区域，不管鸟嘴出现在哪里，检测鸟嘴的卷积核都应该检测出来，过滤器的参数不应随扫描区域的变化而变化，即参数共享。

平移不变性：参数共享使得卷积网络具有平移不变性；先卷积，再平移 = 先平移，再卷积。

5.4 填充和步幅

记输入图像大小为 $h \times w$，卷积核大小为 $k \times k$，步长为 s，填充为 p。

则输出形状为，$H = [\frac{h+2p-k}{s}] + 1$，$W = [\frac{w+2p-k}{s}] + 1$。

5.5 池化层

最大池化，平均池化等，重叠池化：池化窗口大小>步幅，部分元素被多个池化滑动窗口提取；池化无需参数学习。

作用：减小输出的宽度和高度，也叫降采样，从而减少后续层参数量，降低计算复杂度，降低过拟合的风险；具有一定程度的平移不变性：对输入的微小位置变化具有一定鲁棒性；降低网络对位置的敏感性。

最大池化提取最显著特征，平均池化保留图像的整体结构信息，同时减小噪声影响。

5.6 通道

图像往往包括多个通道，比如RGB图像包含三个通道。

对于多通道的图像，其卷积核数 = 输入通道数 × 输出通道数，在当前神经网络架构中，随着神经网络层数的加深， 我们常会减小宽度和高度，增加输出通道数，多通道作用是获取图像的多方面特征表示，每个通道对应的是我们想要提取的一个特征。

1×1卷积核的输出中的每个元素是输入中同一位置的所有通道元素的线性加权组合，可以看做是通道层面的全连接操作。

作用：实现信息的跨通道交互和整合，调整通道数：可以减少通道数，减少后续卷积层的参数量，也可增加通道数。

1×1卷积核的应用：深度可分离卷积，采用卷积核数 = 输入通道数，减少参数量。

多通道下的池化操作在每个输入通道上单独运算，不需要在通道层面进行汇总，输入通道数和输出通道数相等。

5.8 经典卷积网络结构

LeNet：第一个卷积神经网络；

AlexNet：使用ReLU，使用dropout，使用GPU；

VGG：使用了小卷积核，证明了深且窄的卷积（即3×3）比浅且宽的卷积更有效，利于训练更深的网络；

NiN：每个NiN块包括一个普通卷积层和两个1×1的卷积层，NiN块后接最大池化，最后使用全局平均汇聚层，输出通道数等于标签类别数，每个通道内做全局平均池化，即对每个通道内所有元素求平均，不用全连接；

GoogLeNet：每个Inception中由四条并行路径组成，采用不同的卷积方式，最后合并通道；作为该Inception提取的特征。

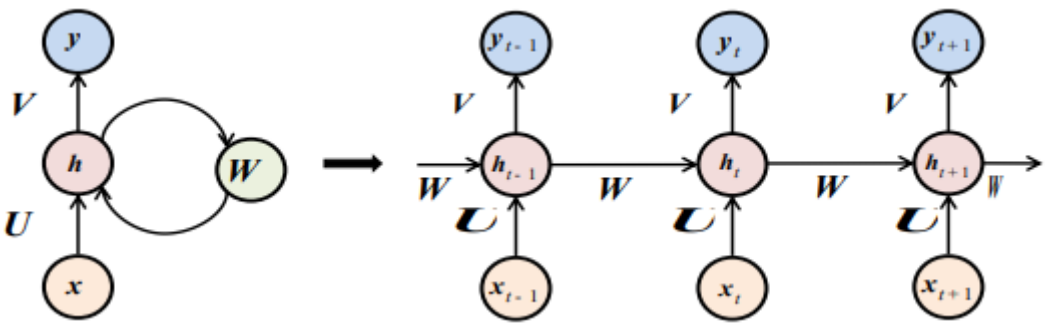
ResNet：引入跨层连接，如果底层已经学习到了主要特征，那么高层只需学习主要特征之外的附加特征即可，学习这些附加特征更加容易。

残差的好处：信息可通过旁支更快地向前传播，误差可以更方便地反向传播；更容易地学习到增量变化；缓解梯度消失问题，从而可以训练更深的网络，提高模型性能和稳定性；最差情况下令参数为0，什么都不学，只传播原信息，解决了随着网络层数的增加，网络性能下降的问题。

第六章 循环网络

处理序列，自然语言处理，挖掘数据中的序列特性；

6.1 循环神经网络结构



上述的带有两个箭头的W矩阵非常不清晰，其真实含义是，对于此时的隐藏层h_t，其输入由上一个输入对应的隐藏层h_{t-1}和输入x共同决定，其权重矩阵分别为W和U；因此隐藏层的表达式如下：

$$h_t = \sigma(U \cdot x + W \cdot h_{t-1} + b_h)$$

常用双曲正切作为激活函数，而时刻t处的输出依赖于时刻t处隐状态，其表达式如下：

$$y_t = \sigma(V \cdot h_t + b_y) \quad or \quad V \cdot h_t + b_y$$

通过引入隐状态，输出层只依赖于当前时刻的隐状态，而不需要显示依赖之前所有输入。不同时刻共享相同的隐藏层权重和偏置、输出层权重和偏置：在所有时间步学习单一模型使得模型能够扩展到不同长度的样本；减少参数量，简化训练难度，提高泛化性，能够处理相同信息出现在不同位置的情况。

输入输出编码：使用标量，容易使得模型误以为数值上接近的词有相近含义；独热编码则使得所有词向量的距离为$\sqrt{2}$，但任意两个向量内积为0， 表明该编码下的词彼此之间不含任何相似信息；Word2vec把一个词映射到一个低维稠密向量，从而使得语义相似的词具有相近的词向量，但其本质上是通过训练得到一个词向量的查询表，每个词对应的向量是固定的，无法解决一词多义问题。

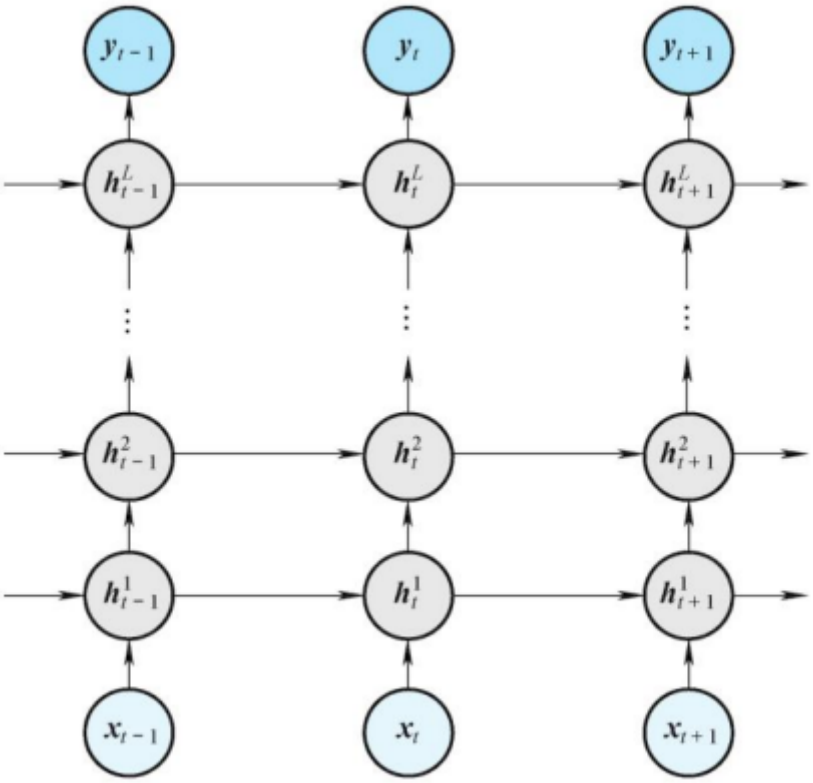
上下文词嵌入：使得词嵌入向量随着其所在上下文语境的不同而改变，可以通过BERT或Transformer编码器的输出作为词嵌入向量结果，也可以对独热编码进行一次线性变换，即$W \cdot x$其中的W是可学习矩阵，NLP中，输入一般使用词嵌入，样本真实输出一般使用独热编码。

损失函数：采用softmax表示某时刻输出每一个单词的概率，并采不同时刻交叉熵的平均值作为损失函数。

深度循环网络

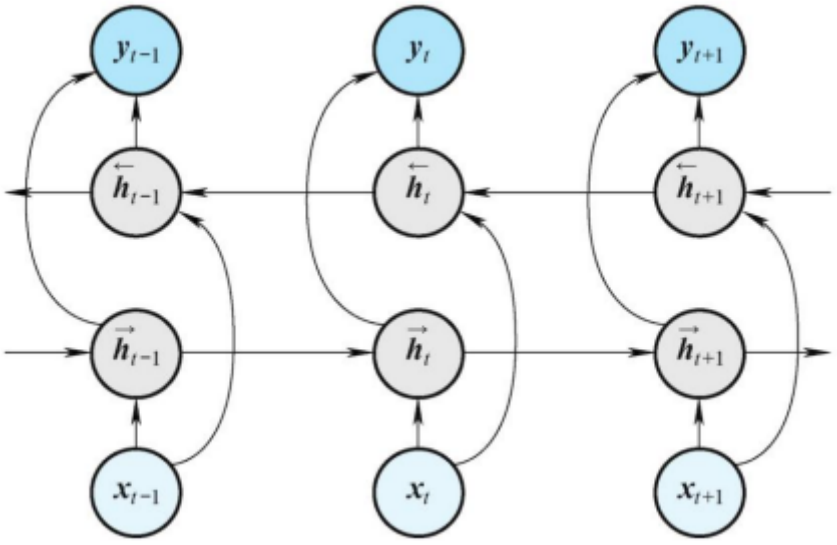
拥有多个隐藏层，每个隐状态都连续地传递到当前层的下一个时间步和下一 层的当前时间步。

$$h_t^1 = \sigma\left(U^1 \cdot x_t + W^1 \cdot h_{t-1}^1 + b_h^1\right)$$
$$h_t^2 = \sigma\left(U^2 \cdot h_t^1 + W^2 \cdot h_{t-1}^2 + b_h^2\right)$$
$$\cdots\cdots$$
$$y = V \cdot h_t^L + b_y$$



双向循环网络

循环网络只允许某隐状态读取自己前面的信息，但双向循环网络可以使之读取后面的，当输出某状态时，单向的只有前面的信息，而双向已经看完了整个句子。



$$\overrightarrow{h}_t = \sigma\left(\overrightarrow{U} \cdot x_t + \overrightarrow{W} \cdot \overrightarrow{h}_{t-1} + \overrightarrow{b}_h\right)$$

$$\overleftarrow{h}_t = \sigma\left(\overleftarrow{U} \cdot x_t + \overleftarrow{W} \cdot \overleftarrow{h}_{t+1} + \overleftarrow{b}_h\right)$$

$$y_t = \overrightarrow{V} \cdot \overrightarrow{h}_t + \overleftarrow{V} \cdot \overleftarrow{h}_t + b_y$$

梯度消失和爆炸

在RNN中，由于权重共享，使用反向传播 计算梯度时会连续乘以相同的权重矩阵，若其特征值小于1，则会导致梯度消息，若其大于1，则会导致梯度爆炸，解决方法之一是梯度截断。这就是NLP中经典的长距离依赖问题，当相关信息距离比较近时，RNN容易学习到，反之不容易，RNN实际上只能学习到短周期的依赖关系。

6.2 门控循环网络

由于隐状态是一个有损映射，不能保留完全的历史信息，因此引入重置门、更新门等概念，能够更好地控制隐状态的更新，从而捕捉时间序列中时间步距离较大的依赖关系，解决传统循环神经网络难以捕捉序列长期依赖关系的问题。模型有可学习的机制捕捉何时更新隐状态，以及应该何时重置隐状态，保留重要的关键词。

重置门

引入一个t时刻的向量r_t，决定了在结合当前输入和历史信息时需要保留多少历史信息：

$$\tilde{h}_t = \tanh(U \cdot x + W \cdot [r_t \odot h_{t-1}] + b_h)$$

其中r_t与上一个隐状态逐个元素相乘，当r趋近于1时保留更多历史信息，反之更关注当前输入，从而有助于 捕获序列中的短期依赖关系。

更新门

用于计算当前隐状态，决定了在计算时的候选隐状态和前一时刻隐状态的占比多少：

$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot \tilde{h}_t$$

当z_t趋于0时，更多使用候选隐状态，而当其趋于1时，更多应用历史信息，有助于 捕获序列中的长期依赖关系。

学习重置门和更新门

$$z_t = sigmoid(U_z x_t + W_z h_{t-1} + b_z)$$
$$r_t = sigmoid(U_r x_t + W_r h_{t-1} + b_r)$$

通过sigmoid将其范围约束在（0，1）内。

当$r_t \to 0, z_t \to 0$时，时刻t的隐状态h_t倾向于只接收当前输入x_t，忽略历史信息，**有助于捕获序列中的短期依赖关系**

当$r_t \to 1, z_t \to 0$时，时刻t的隐状态h_t倾向于同时考虑前一时刻隐状态h_{t-1}和当前输入x_t

当$z_t \to 1$时，时刻t的隐状态h_t倾向于只接收前一时刻隐状态h_{t-1}，忽略当前输入x_t，**有助于捕获序列中的长期依赖关系**

重置门和更新门可以有效避免梯度爆炸和消失，因为其可以截断隐状态的递归关系；同时通过z_t的引入类似ResNet的旁支连接，有效缓解梯度消失。

6.3 长短期记忆网络

LSTM采用更复杂的门控形式，引入了遗忘门f_t，输入门i_t，输出门o_t：

$$f_t = sigmoid(U_f x_t + W_f h_{t-1} + b_f)$$
$$i_t = sigmoid(U_i x_t + W_i h_{t-1} + b_i)$$
$$o_t = sigmoid(U_o x_t + W_o h_{t-1} + b_o)$$

引入记忆单元c_t以及新记忆单元$\tilde{c}_t$：

$$\tilde{c}_t = tanh(U \cdot x_t + W \cdot h_{t-1} + b_h)$$
$$c_t = f_t \cdot c_{t-1} + i_o \cdot \tilde{c}_t$$

隐状态由输入门控制：

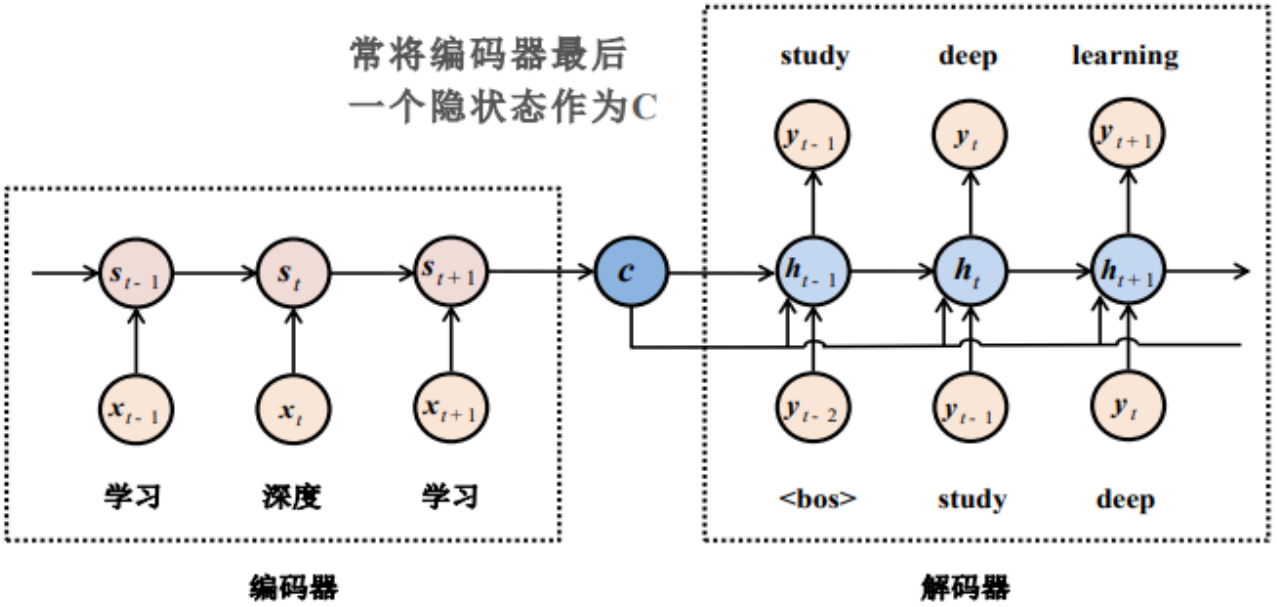
$$h_t = o_t \cdot tanh(c_t)$$

最终输出与原式一致：

$$y_t = V \cdot h_t + b_y$$

6.4 编码器-解码器框架

这里的编码器-解码器是基于RNN的，简单RNN用于同步任务，因此对于每一个输入都有一个对应的输出，而编码器则是仅仅利用每一个隐状态作为下一个隐状态的输入，将最后一个隐状态输出作为C，而这个C作为输入的一部分，进入解码器，解码器的初始输入仅为C，而其他隐状态h_t的输入由C，上一个的输出y_{t-1}，上一个隐状态h_{t-1}共同构成，其中，上一个的输出y_{t-1}作为x_{t-1}进行输入。



不足：编码器RNN输出的 C 的维度太小而难以适当地概括一个长序列，源句子前面单词的语义信息几乎没有被编码到 C 中；并非所有输入词元都对解码某个词元有用，但在每个解码步骤中仍使用相同编码的上下文特征C，因此解码不同词元时，把注意力集中在更相关的输入词元上。

第七章 优化算法

优化算法解决的问题：训练神经网络，减小训练误差。

7.1 非凸函数曲面

对于定义在凸集上的凸函数，其任意局部极小点就是全局极小点，容易优化得到最优解。

局部极小值：梯度为零且在邻域内是最小值的点，当优化算法迭代点接近局部极小值点时，随着目标函数的梯度接近于零，算法可能 仅得到局部最优解，而非全局最优解。

鞍点：函数的梯度为零，但既不是局部极大值点也不是局部极小值点的点。表示其可能是某个横截面的极小值点但是另一个横截面的极大值点。

Hession矩阵可以判断临界点或驻点是局部极小值，鞍点或局部极大值。特征根都为正时，为局部极小值，有正有负为鞍点，都为负时为局部极大值。对于高维问题，特征值有正有负的可能性远大于特征值都为正或都为负的概率，所以鞍点比局部极小值点更有可能出现。而添加随机扰动可以逃离鞍点。

7.2 梯度下降法

对于函数$f(x)$，其导数为$f'(x)$，则有：

$$f(x+\epsilon)=f(x)+\epsilon\cdot f'(x)+o(\epsilon^2)$$

所以若使得$\epsilon=-\eta f'(x)$ ，则有：

$$f(x-\eta f'(x))=f(x)-\eta\cdot f'(x)^2+o(\eta f'(x)^2)$$

当η足够小时，后面的高阶无穷小可以忽略，所以$f(x-\eta f'(x))<f(x)$，可以通过此方法有效使得损失函数下降。

对于向量形式，只需要转化为对应的向量形式，如$x=[x_1,x_2,.....x_n]^T$那么$\nabla f=[\frac{\delta f}{\delta x_1},\frac{\delta f}{\delta x_2},.....,\frac{\delta f}{\delta x_n}]$，然后按照向量形式将$\epsilon$代入然后进行迭代即可。

7.3 随机梯度下降法

由于深度学习中目标函数往往为训练集中所有样本误差的平均值，如果采用所有数据进行训练，那么每次迭代需要遍历所有样本，计算复杂度较高，所以随机梯度下降采用随机抽取一个样本的梯度作为完整梯度的近似，进行训练。

Epoch = 所有训练样本各被用于一次训练（一次前向 + 反向传播）：

1 Batch GD

- 一次性用 全部 N 个样本
- 只更新 1 次参数

2 SGD（batch size = 1）

- 每个样本各用 1 次
- 更新 N 次参数

3 Mini-batch GD（batch size = B）

- 每次用 B 个样本
- 更新 N/B 次参数

而SGD的随机抽取只是更新N次时，样本使用的顺序不一样，而不是有放回的随机抽取。小批量SGD每次抽取B个样本，将其梯度平均值作为更新使用的数据，所以在一个Epoch内需要更新$\frac{N}{B}$次。

初始化：

随机初始化

- 将优化变量初始化为随机值，通常使用高斯分布或均匀分布生成随机值
- PyTorch API：torch.nn.init.uniform_、torch.nn.init.normal_

Xavier初始化

- 对于m个输入和n个输出的全连接层，可以初始化为$\left(-\sqrt{\frac{6}{m+n}},\sqrt{\frac{6}{m+n}}\right)$中的均匀分布，或均值为0方差为$\frac{2}{m+n}$的高斯分布
- PyTorch API：torch.nn.init.xavier_uniform_、torch.nn.init.xavier_normal_

He初始化

- 对于具有m个输入的使用ReLU的全连接层，可以将优化变量初始化为$\left(-\sqrt{\frac{3}{m}},\sqrt{\frac{3}{m}}\right)$中的均匀分布，或均值为0方差为$\frac{1}{m}$的高斯分布
- PyTorch API：torch.nn.init.kaiming_uniform_、torch.nn.init.kaiming_normal_

学习率：

初始时，迭代点远离最优解，大幅度的更新是可以接受的；快收敛时，需要降低学习率，避免打转。

7.4 动量法

$$m^k = \beta \cdot m^{k-1} - \nabla f$$
$$x = x - \eta \cdot m^k$$

其中，m相当于速度，而β相当于一个衰减权重，这表示受到阻力，而梯度相当于受力，也就是当前速度受到阻力和受力的共同影响，梯度下降法相当于$\beta = 0$。

优势：减轻了梯度法“之”字形的震荡，即某一向量在接近极小值点时，可能存在有些特征不需要大幅度变化而有些特征需要大幅度变化的请客，引入历史速度，当历史速度与当前梯度方向一致时，加速收敛，反之衰减震荡。

重球法：将梯度转化为x_k和x_{k-1}的差，与动量法是等效的。

指数移动平均：m速度项相当于历史梯度的加权求和，而不是加权平均。其系数之和约为$\frac{1}{1-\beta}$，那么为了解决此问题，我们只需要在动量法的梯度项上乘$1-\beta$即可解决。

Nesterov加速法：没有在原地计算梯度进行更新，而是沿着原方向再走一步后计算梯度，进行更新，相当于在动量法的基础上多“向前看了一步”。

7.5 自适应学习率算法

既然单一的学习率无法同时兼顾不同方向的收敛性和收敛速度，那么就在不同方向使用不同的学习率。

不同的参数使用不同的学习率。对于历史梯度普遍较大的参数，减小其学习率减缓该方向的更新；历史梯度普遍较小的参数，增大学习率加快该方向的更新。

AdaGrad

$$g^k = \frac{1}{|B_k|}\sum_{i\in B_k}\nabla f_i(x^k)$$

$$v_i^k = \sum_{t=0}^{k}(g_i^t)^2 = v_i^{k-1} + (g_i^k)^2$$

$$x_i^{k+1} = x_i^k - \frac{\eta}{\sqrt{v_i^k}+\varepsilon}g_i^k$$

AdaGrad存在的问题是v几乎线性增长导致学习率衰减过快，所以引入历史信息衰减，使之对最新梯度的反应更快。

RMSProp

$$g^k = \frac{1}{|B_k|}\sum_{i\in B_k}\nabla f_i(x^k)$$

$$v^k = \beta v^{k-1} + (1-\beta)(g^k)^2$$

$$x^{k+1} = x^k - \frac{\eta}{\sqrt{v^k}+\varepsilon}\cdot g^k$$

Adam

$$g^k = \frac{1}{|B_k|}\sum_{i\in B_k}\nabla f_i(x^k)$$

$$v^k = \beta_2 v^{k-1} + (1-\beta_2)(g^k)^2$$

$$m^k = \beta_1 m^{k-1} + (1-\beta_1)g^k$$

$$x^{k+1} = x^k - \frac{\eta}{\sqrt{v^k}+\varepsilon}\cdot m^k$$

v是衰减系数，衰减系数与RMSProp类似，更关注近期信息，同时引入动量法，动量法中也采用指数移动平均，对历史速度采用加权平均，更新时进行自适应权重调整。

由于β_1和β_2设置的大小不同，而第一次迭代的时候往往v_0和m_0都是0，就会导致v_1和m_1有极大差距，所以对二者进行标准化：

Adam偏差修正

$$g^k = \frac{1}{|B_k|}\sum_{i\in B_k}\nabla f_i(x^k)$$

$$v^k = \beta_2 v^{k-1} + (1-\beta_2)(g^k)^2$$

$$m^k = \beta_1 m^{k-1} + (1-\beta_1)g^k$$

$$\hat{m}^k = \frac{m^k}{1-\beta_1^k},\quad \hat{v}^k = \frac{v^k}{1-\beta_2^k}$$

$$x^{k+1} = x^k - \frac{\eta}{\sqrt{\hat{v}^k}+\varepsilon}\cdot \hat{m}^k$$

AdamW

即权重衰减，再梯度计算时加入正则项，所以此时还需要加入$min[f(x)+\frac{\lambda}{2}\cdot||x||^2]$，那么在梯度计算时，则变为$g+\lambda\cdot x$；

$$g^k = \frac{1}{|B_k|}\sum_{i\in B_k}\nabla f_i(x^k)$$

$$v^k = \beta_2 v^{k-1} + (1-\beta_2)(g^k)^2$$

$$m^k = \beta_1 m^{k-1} + (1-\beta_1)g^k$$

$$\hat{m}^k = \frac{m^k}{1-\beta_1^k}, \quad \hat{v}^k = \frac{v^k}{1-\beta_2^k}$$

$$x^{k+1} = x^k - \eta\left(\frac{1}{\sqrt{\hat{v}^k}+\varepsilon}\cdot\hat{m}^k + \lambda x^k\right)$$

$$= (1-\eta\lambda)x^k - \frac{\eta}{\sqrt{\hat{v}^k}+\varepsilon}\cdot\hat{m}^k$$

7.6 批量规范化和层规范化作用

将网络信息流的值规范化到合理的范围内，缓解梯度消失和梯度爆炸，使得网络更容易被训练。

第八章 注意力与Transformer

8.2 注意力机制组件

注意力：是指序列中两个元素之间的相关性程度；允许神经网络在处理输入数据时集中注意力于关键信息，忽略不相关信息。注意力权重满足概率分布要求：非负，和为1。

Q，K，V是重要的，分别表示查询，键，值；解释如下：

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

对于Q，其表示一个查询向量，我们先假设其为行向量，而K的每行则表示一个对应的键，$Q\cdot K^T$即是Q向量与所有键K的内积所构成的向量，举个例子，$K=[k_1^T,k_2^T]^T$，那么$Q\cdot K^T$得到的应当是一个匹配度向量，对于点积，只有当向量方向相同时，其内积越大，所以得出的$Q\cdot K^T=[p_1,p_2]$，分别表示Q向量与k_1,k_2的匹配程度，经过softmax后转化为概率，然后乘键对应的值，即可得出分配好概率的结果，V与K是同型矩阵，K的某行和V的某行有如下性质，K的某行表示对应V的特征向量，而V的某行则是该特征对应的信息，得出匹配度后，我们对每一行V进行加权，即可得到最终的注意力，有着与Q更匹配的键K的V获得更多权重。公式中的$\sqrt{d_k}$表示查询和键的长度，除以这个避免键长度过长导致的内积过大，以至于输入softmax后在指数函数放大下获得的概率不均等。最终上面公式的输出应当与原有的V大小保持一致，其结果每一行即一个新的词元，应当耦合所有词元的信息，每个词元所占权重由注意力决定。

掩蔽softmax

给定一个范围，我们只希望它考虑这个范围内的键，比如只取K矩阵的某些行，则在进行softmax的时候，其他地方不会纳入考虑，直接取0，然后该范围内的匹配度被输入softmax转化为概率。避免模型偷看后面的正确答案。

8.3 自注意力

查询、键和值来自同一组输入，多用于文本数据；

对于一组词元，将其按行排列得到从上到下的$x_1,.....,x_n$，那么其Q，K，V，都等于一个权重矩阵W^Q,W^K,W^V与其自身的X相乘，这些权重矩阵是**需要学习的参数**，然后按照原来的公式进行分析，即可实现自注意力转化得到的新的信息矩阵，原X中的每一个词元是孤立的，经过注意力后，我们把上下文相关联。

劣势是复杂度高为O(n^2)，而优势在于可以并行计算。

8.4 多头注意力

当给定查询、键和值时，我们希望模型可以从不同角度学习到不同的行为和知识，然后将不同的行为和知识组合起来。例如处理自然语言时，一个头可能专注于语法结构，另一个头可能关注语义关系。

对于一组词元，将其按行排列得到 $x_1,...,x_n$，每个词元的维度为 d，整体输入可以表示为矩阵 $X \in R^{n\times d}$。首先，通过需要学习的权重矩阵 W^Q,W^K,W^V 对 X 进行线性变换，得到查询 $Q=XW^Q$、键 $K=XW^K$ 和值 $V=XW^V$。在多头注意力中，设注意力头数为 h，每个头会在特征维度上得到 d/h 维的子矩阵，将 Q,K,V 通过全连接层做线性变化（此全连接层的本质功能是解耦，避免q，k，v给每一个注意力头的信息完全一致导致每个注意力头学习到一样的权矩阵），将 Q,K,V 拆分为 h 个子矩阵分别作为各个注意力头的输入。每个注意力头独立计算后，得到 n 行 d/h 列的输出矩阵，将所有头的输出在特征维度上拼接，得到 n 行 d 列的矩阵，最后通过输出线性变换矩阵 W^O 进行融合，得到多头自注意力模块的最终输出。经过处理后，输出矩阵中的每个词元表示融合了序列上下文信息，不再是原来孤立的表示。

8.5 位置编码

（怎么用的，为什么使用）

自注意力计算中没有考虑序列中词元的位置信息，即不同词元的前后顺序关系，为了使用序列的顺序信息，可以在输入表示中添加位置编码来注入位置信息，好处是既能使用序列中的位置信息，又能保留并行计算的高效性。

对于输入 $X \in R^{n\times d}$，固定位置编码方式如下：

$$P_{i,2k}=sin(\frac{i}{10000^{\frac{2k}{d}}})$$

$$P_{i,2k+1}=cos(\frac{i}{10000^{\frac{2k}{d}}})$$

偶数位置用正弦，奇数位置用余弦。最终输入注意力头时，需要输入$X=X+P$用来表示信息。

旋转位置编码方式有不同，注意力权重只与查询、键及其相对位置有关，只在attention计算时在Q，K内积中参与运算。

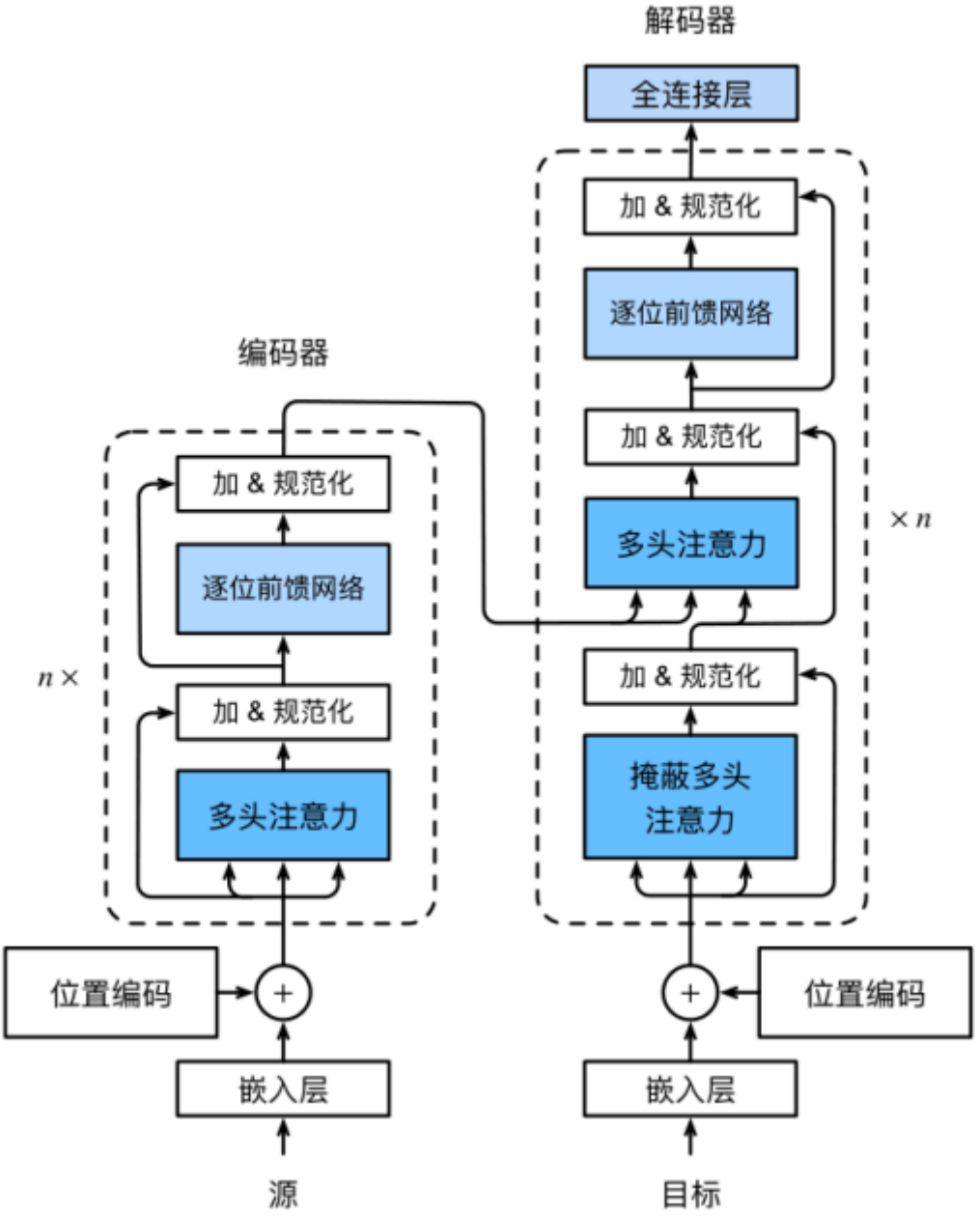
8.6 注意力 & CNN & RNN

CNN只考虑了局部信息，默认局部信息是需要重点关注的；注意力考虑了全局信息，通过学习得到哪些信息需要重点关注；CNN是简化版的注意力。

RNN当前时刻的计算依赖前一时刻，无法并行计算，序列较长时，RNN存在梯度消失和梯度爆炸问题，不能很好解决长距离依赖关系，注意力使得序列中任意两个词之间的依赖关系可以直接被建模，更好地解决长距离依赖关系。

8.7 Transformer

接下来解释每一个结构



8.7.1 输入和位置编码

首先将源，也就是输入的自然语言句子，切割为Token：词元，文本中的基本单元，通常为词或短语，即Tokenization。有词粒度（dog、dogs等会被视作不同的词，导致词表庞大，应用中可能出现不在词表中的词），字符粒度（蕴含语义信息过少），子词粒度（比如boy/girl，s；这时s是一个子词，boys和girls并不是两个新的词语，而是boy/girl加上复数的词缀，词表尽可能小、具有语义信息、 更好地处理同前缀同后缀的词、较好处理未见过词）。

嵌入层：将输入序列（源）的token转换为词嵌入向量，加上位置编码向量，作为编码器的输入X。

8.7.2 编码器

第一个子层：输入到多头注意力模块并引入残差连接后，进行Layernorm：基于每 一层的每个token的特征向量进行规范化。

$$X_A = Layernorm(X + Attention(X))$$

第二个子层：逐位置前馈网络FNN，将上一个输出X_A的每一行的每个位置上的词元向量$x_i \in R^d$，独立送入一个两层的全连接网络，第一层扩大维度，激活，比如隐藏层宽度可以变成词元向量的4倍，然后再经过一层恢复原大小：

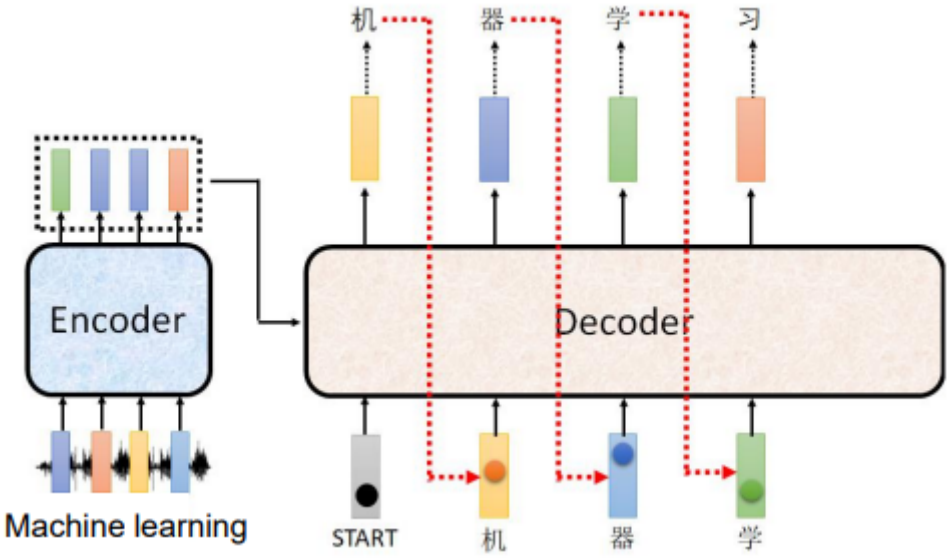
$$W_2(GeLU(W_1 \cdot x_i + b_1)) + b_2$$

其中W和b对于所有词元参数共享，但是对于不同的编码器参数不共享。

最终输出X_B表示如下：

$$X_B = Layernorm(PositionFFN(X_A) + X_A)$$

8.7.3 解码器



负责生成目标语言序列，对于编码器的输出，其会输入到解码器中，推理时，按顺序生成每一个词元；将上一时刻的预测输出作为下一时刻的输入，预测下一个输出词；**自回归**允许在生成下一个输出时，可以看到之前的所有输入，最开始的输入为初始词向量{}随后在翻译过程中，逐步填充，在翻译"器"时，输入的词向量为{，“机”}。

输入和位置编码：输出序列的词嵌入向量加上位置编码，输入到解码器中，最开始为{}；训练时，使用真实输出作为当前时刻的输入，避免前一刻输出错误导致一步错步步错。

第一个子层：掩蔽多头自注意力，训练时，所有的正确答案输入给解码器，解码每个词时只能使用之前生成的词，不能偷看后面的正确答案，掩蔽注意力将后面的正确答案掩蔽，确保每个输出仅依赖于前面已生成的输出词元：

$$O = V \cdot softmax(\frac{V \cdot Q^T \cdot M}{\sqrt{d_k}})$$

其中M为上三角0-1掩码矩阵，其本质是一个列变换，即，对于$V \cdot Q^T$，经过M后，第一列只允许其保留第一列信息，第二列则变为第一第二列之和，以此类推，确保掩码的实现，最终左乘V得到自注意力后的Z。

第二个子层：编码器－解码器注意力层；Q来自前一个解码器子层的输出，K和V来自编码器最后一层的输出；交叉注意力构建起了编码器到解码器 之间的信息桥梁。

第三个子层：逐位置前馈网络。

最终经过全连接层将解码器的输出维度映射到词表大小，并做softmax变换，判断每个位置输出哪一个词的概率最大，然后输出，加入到目标中。

Transformer	
性能强大的原因	**缺点**
自注意力机制（Self-Attention Mechanism）	数据要求高
并行计算	
层次化表示	解释性差
位置编码（Positional Encoding）	
大规模训练数据	处理长序列时计算开销高
优化技巧	

8.8 ViT

Step1：对于尺寸为 $224 \times 224 \times 3$ 的输入图像，将输入图像划分成 196 个 $16 \times 16 \times 3$的图像块，将每一个图像块扁平化为向量（长度为 $16 \times 16 \times 3 = 768$），线性投影得到编码向量，将所有 196 个编码向量放在一起形成输入序列。

Step2：定义一个类别嵌入向量，长度也为 768，放在序列开头，组成一个197个编码向量的输入序列。

Step3：输入Transformer

Step 4：使用Transformer模型处理输入，使用了PreNorm模式。

Step 5：在 Transformer 模型的输出端构建一个分类网络（即全连接网络），将类别嵌入向量对应的输出作为分类网络的输入。

8.10 混合专家模型

MoE通过多个专家的协同工作来提升模型的预测效果。每个专家都专注于特定的领域知识，让不同的输入选择最合适的专家

第九章 大语言模型

特点：参数量大，训练数据多，泛化能力强。

9.1 预训练+微调范式

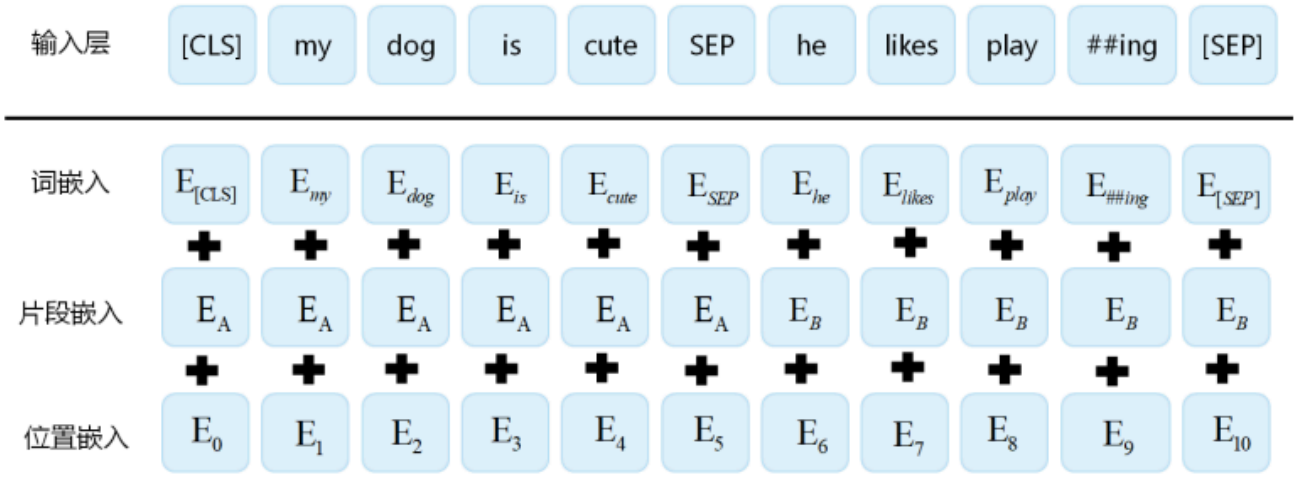
预训练：在海量无标注语料上自监督地训练一个模型，形成一种通用的语言理解能力，微调：针对特定任务，使用高质量标注数据集进一步训练预训练好的模型，使其适应特定任务。

自监督学习：利用大型语料库的现有文本序列，自监督地从海量文本数据中学习，不需要昂贵的标签标注。即对于某句或某段，再某句中挖出一个词语，在某段中挖出一个句子，让模型预测被挖出的地方的填充内容，即让模型做填空题。

9.2 BERT模型架构

输入表示层

BERT的输入层由词嵌入、片段嵌入和位置嵌入三部分加和而成；词嵌入会包含特殊的首位字符[CLS]，每个句子之间则是分割表示符[SEP]，片段嵌入在同一个句子在同一个嵌入的句子中形式一致。位置嵌入是可学习的参数。



BERT只使用Transformer编码器部分，同时具有双向的编码器结构，其特点是包括双向的多头自注意力模块；可以同时利用词左右两边的信息（这里表意不明，其本质就是Transformer的编码器，因为LLM历史中采用了M上三角矩阵掩码避免其观察到后续信息，所以这样写，但是Bert其实还是Encoder）。

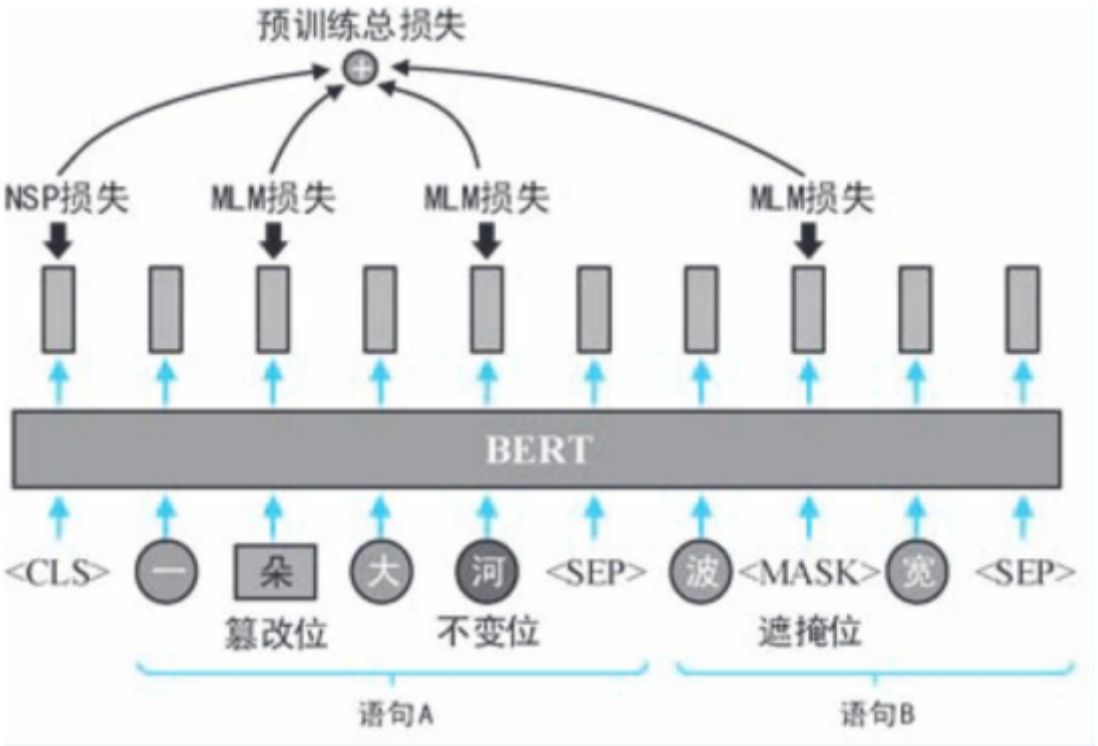
9.2.1 BERT预训练：MLM

随机掩蔽词元，使用上下文词元自监督地预测掩蔽词元（以输入我爱南开大【mask】为例，其本质是一个n行d列的矩阵，经过Encoder后输出的仍然是一个n行d列的矩阵，mask部分会输出被掩盖部分的预测值，而我们此时计算损失函数不考虑前面我爱南开大这几个字的loss，只考虑mask部分的损失）；掩码模式分别为遮挡，篡改，不变。在经过Encoder后得到的矩阵会被输入到一个全连接层，经过Softmax对被掩盖的字进行预测，在训练时，训练全连接层和Encoder的参数。

9.2.2 BERT预训练：NSP

下句预测可以建模句子之间的关系，对于由[SEP]隔开的句子，标签为判断第二局是否为第一句合适的下一句。

9.2.3 BERT预训练



NSP和MLM共同作用指导BERT的预训练。损失函数是二者之和。

9.2.4 BERT微调

在BERT的基础上增加一个全连接输出层，BERT主体采用之前预训练好的参数，而新增加的全连接部分随机初始化，然后用高质量的特定任务的标注数据统一训练整个模型。全参数微调，微调BERT中的所有参数：计算及存储资源大。

LORA

在预训练模型参数中加入旁路，该旁路由低秩矩阵A和B组成，例如对于$\mathbb{R}^{n\times d}$的输入和输出的一个Encoder，其权矩阵应该为$W\in\mathbb{R}^{n\times n}$，那么LORA在训练中则冻结当前$W$，转而通过旁路连接，训练$A\in\mathbb{R}^{n\times r},B\in\mathbb{R}^{r\times n}$，并最终的$W_1=W+AB$作为微调后得出的权重。

9.2.5 BERT原理解释

BERT 的本质贡献不是某个具体任务，而是学到了一套“上下文相关的词表示函数”。这个函数会根据词所处的上下文，动态地产生不同的向量表示；下游任务只需要“拿来用”这些向量，就能解决分类、匹配、问答等问题。

9.3 GPT模型架构

GPT使用Transformer的解码器部分，GPT去掉了编-解码注意力模块，仅保留掩蔽多头自注意力和逐位前馈网络两个模块。

9.3.1 GPT预训练

预测下一个词元；自回归语言模型：根据上文内容预测下一个词，这样逐个生成词汇的语言模型。

9.3.2 GPT1微调

GPT1采用了预训练+微调范式；

多选问答任务（阅读理解）：给定一段语料z、一个问题q、一个候选答案集合 {a1 ,…, ad }，选择最佳答案。

自然语言推理：输入两个句子，判断这两个句子逻辑上是否相关；将两个句子作为整体输入GPT，接收最后一个输出特征，进行线性变换和概率化。

9.3.3 GPT2、3微调

提示学习：不显著改变预训练语言模型结构和参数的情况下，通过向输入中增加 “提示”，将下游任务改造为文本生成任务，从而实现上下游任务统一的一种学习方法。

零样本学习：不微调直接用；

小样本学习：依次给样本使其进行学习；

确保大语言模型的行为与人类价值观、人类真实意图和社会伦理相一致上下文学习：不需要调整模型参数，仅通过几条下游任务的示例就能理解任务并给出满意的回答，提升模型的小样本学习能力。因为其实模型已经有能力回答问题了，但是他不知道我们需要什么答案，我们只需要给模型提供少量引导性的范例。

9.3.4 ChatGPT：SFT和RLHF

监督微调（SFT）：人类提供少量提问和答案；

基于人类反馈的强化学习（RLHF）：不直接给答案，而是告诉机器现有的答案 是好还是不好，或者为模型的多个输出进 行打分或排序。

9.4 大模型中的若干概念

人类对齐：确保大语言模型的行为与人类价值观、人类真实意图和社会伦理相一致；

思维链：一种高级提示策略，向大语言模型提供诸如 “Let’s think step by step.” 这样的诱导性指令；

涌现能力：在小型模型中不存在但在大模型中出现的能力；

扩展定律：

2020年，OpenAI团队建立了大语言模型性能与三个主要因素——模型规模（N）、数据规模（D）和计算算力（C）之间的幂律关系（L为交叉熵损失）

$$
\begin{aligned}
L(N) &= \left(\frac{N_c}{N}\right)^{\alpha_N}, \quad \alpha_N \sim 0.076, N_c \sim 8.8 \times 10^{13} \\
L(D) &= \left(\frac{D_c}{D}\right)^{\alpha_D}, \quad \alpha_D \sim 0.095, D_c \sim 5.4 \times 10^{13} \\
L(C) &= \left(\frac{C_c}{C}\right)^{\alpha_C}, \quad \alpha_C \sim 0.050, C_c \sim 3.1 \times 10^{8}
\end{aligned}
$$

模型越大、数据越多、算力越足，模型的表现就越好，而且这种提升是可预测的，不会突然遇到无法 逾越的瓶颈；

蒸馏：将复杂模型（称为教师模型）包含的知识迁移到简单模型（称为学生模型）中，实现模型压缩；

量化与剪枝：量化：从浮点数映射到整数；剪枝：在尽可能不损失模型性能的情况下，消减参数；

多模态：能够处理和整合多种模态信息（比如文本、图像和音频）的大语言模型；

幻象：模型在生成过程中可能会出现与事实不符的信息。