正则表达式（python）:

e.g.1: ^(http|https):\/\/[^\s/$.?#].[^\s]\*$

这个正则表达式用于匹配URL。具体来说，它的工作原理如下：

1. ^(http|https)：匹配字符串的开头，然后匹配"http"或"https"。
2. :\/\/：匹配"://"。
3. [^\s/$.?#]：匹配任何不是空白字符、斜杠、点、问号或井号的字符。
4. .[^\s]\*$：匹配点之后的任何非空白字符，直到字符串的末尾。

综合起来，这个正则表达式匹配以"http://"或"https://"开头，后跟一个非空白字符，然后是任何非空白字符的字符串，即一个URL。

Pytorch中的detach()，detach()会将变量分离出计算图，将detach()的变量进行计算无法通过后面的数值计算其梯度或计算组成该变量的变量的梯度，如果detach后的变量与别的参数一起计算，比如说weight，那可以对weight求梯度，detach就只是个数值，不计算梯度。

特征选择—sklearn Kbest

随机梯度下降—SGD

Adam

Pytorch的nn.Crossentropy（）这个计算loss的函数的一些说明：实际上已经包含了nn.logsoftmax()和nn.NLLLoss()，使用范例：loss=nn.Crossentropy(prediction,labels)，其中prediction是多个预测向量组成的矩阵，每个向量是一条样本的预测，labels是一个向量，记录这每个样本的标签，即对应着预测矩阵中预测向量的索引，将预测向量做logsoftmax，然后索引对应的向量里面的值乘1，其他的乘0，全部加起来作为该向量的loss，比如一个预测向量是

[ 6.6132e-01,

5.8612e-01,

-1.7432e+00,

3.4200e-01,

-1.0279e+00,

-8.5446e-01,

3.6296e-01,

2.2967e+00,

-3.9669e-01,

-4.0955e-01,

4.4718e-01]

然后labels里面这个预测向量的标签是9，整个的loss就是

0\*10+1\*log（-4.0955e-01/sum（整个向量））。

CNN—卷积，池化—接全连接层-softmax，卷积-相当于多个核对图像多个部分进行筛查，每个核对应图像不同特征，池化—降维，应用案例alphar-go，相当于每个卷积后，卷积到深层，相当于一个卷积核看原始图片上的某个特征，与围棋类似，但是围棋不能使用max-pooling，图像用max-pooling不会损失特征信息，但是围棋会。

训练集、验证集的区别

训练集-更新网络参数（不是超参数），利用正向传播和反向传播确定梯度，利用optimizer·更新参数（常见优化器有SGD随机梯度下降和Adam）（训练集默认用mini-batch training 来更新参数）

验证集-计算valid\_loss，判断是否提前停止train，更新超参数（scheduler比如退火算法更新学习率，动态下降）等

训练轮数（epoch）-每轮过一次整个数据集，epoch越大，学到的特征越多，但是可能过拟合

批量大小（batch\_size）-进行mini\_batch training时，在一个epoch将整个数据集按照batch\_size分为多份，分批进行梯度下降

Pytorch-dataset:

要定义getitem()来返回输入和标签

要定义getlen()

t-SNE:

一种数据的非线性可视化工具，与pca不同，t-SNE注重数据之间的相关性，在supervised-learning和semi-supervised-learning中，t-SNE的表现形式不太一样，不过主要是，通过原数据寻找最好的Wij组合，然后根据Wij组合寻找最佳的从原始数据X到变换后数据Z的映射，然后就可以将X可视化为Z。

PCA（主成分分析）：特征降维的工具，过程：将数据对应矩阵标准化后求协方差，得到协方差矩阵，再由协方差矩阵求特征值与特征向量，设定K为需要的特征数量，K可由方差解释率计算，即主成分的特征值和占总特征值和的比例，来选择，然后将对应的特征向量cat，得到pca映射矩阵，然后将原始数据与映射矩阵相乘，得到预测点，预测点与原始数据点求欧几里得距离，可以得到偏移的程度

交叉验证—

最常用的是K折交叉验证，将训练集与验证集的总量分为K份（一般是均分称为K-fold），依次计算第i份为验证集的acc或交并比等（i=0,1,2,…,k）,然后计算这些性能指标的平均值作为最终的性能指标或取其中最高的作为最终的性能指标。（好处，能够综合考虑每种训练集和验证集划分的影响）

K-fold cross validation的使用方法：

将训练集和验证集放在同一个文件夹，使用sklearn.model\_selection下的方法StraitifiedKFold或KFold，第一个步骤—将数据集与验证集分成K份，使用dataloader和KFold方法，

Skf=KFold(n\_splits=num\_splits)

For fold,(train\_idx,val\_idx) in enumerate(skf(datasetlist or dir list , y\_hat(label)):

#定义train\_dataloader和valid\_dataloader

#接下来就在不同epoch下train()，并calculate sum\_acc or sum\_loss 等

在各个不同的训练集和验证集组合下训练后，对sum\_acc和sum\_loss求平均，得到最终性能指标

留一交叉验证（leave-one-out）

K折交叉验证重复，K折交叉验证嵌套（涉及每一折之间共享参数，K-fold交叉验证并没有实现每一折之间互相影响，都是相互独立的）

K折交叉验证的好处主要是能够让每个数据集都有机会成为训练集和验证集，可以减小数据集和验证集之间的偏差，减小数据划分带来的偏差，减少数据倾斜的问题。

应用K-fold会train得更久，举个例子就是，原来只要train num\_epoches次就行，应用K-fold之后就要train num\_epoches\*K次，要在每个fold上都train。

Ensemble（集成学习方法）：

Bagging:N笔training data 取N’笔放回抽样，得到多个dataset，使用网络分别对这几个dataset进行learning，将输出的结果进行average or voting.（会使得得到的模型的variance比较小），当model很复杂容易overfit时，会使用Bagging.

e.g：Random Forest：在选择每个branch的时候，都随机选择其中一部分feature，然后在训练的时候选择OOB验证，最后训练出来的model得出的结果进行average or voting predict.

OOB验证（Out-of-bag validation）：随机有放回地从原始数据集中抽取多个子集形成多个bootstrap，由此来训练多个弱学习器（例如decision-tree），使用out-of-bag data（袋外数据）对学习器性能进行评估（作为验证集），然后平均各个学习器的输出性能

bootstrap（重复随机有放回抽样）

boosting：思想是通过迭代多个弱学习器，每次迭代都修正前一个学习器的错误

e.g : Adaboost : 增加数据权重u （dataset），目标函数：L=∑ui\*loss，在原始的training data上训练出弱学习器，计算错误率，利用权重变更参数d改变权重使得错误率增加到0.5（0.5是便于后面数学推到计算权重变更）（错误样本权重乘以变更参数d，正确样本权重除以变更参数d）根据数学推导，其中，然后再根据新权重训练弱学习器，为了简便计算，加ln得，，经过多个epoch，得到多个f弱学习器，然后输出.其中，设，training不断maximizing 就能够使得testing的error rate decresing，可以通过minimizing adaboost的upper-bound来实现

e.g gradient boosting：定义一个与g(x)有关的的Loss function，新的弱学习器f，由Loss function对g(x)求导而得来，与梯度下降类似，然后为便于计算，定义Loss function为,对g(x)求导，得到新的函数，由计算得， 其中，weight就是adaboost的weight，就得到，用负梯度的方法找到f后，固定f找，即把代入Loss function，然后对进行求导，经过数学推导.

Adaboost和Gradient Boost的区别：1.优化方法上，前者是根据前一学习器错误率调整权重来得到新的学习器，后者是根据优化损失函数，得到负梯度一次获得新学习器。2.权重分配，前者是根据前一个学习器的错误率来确定权重，后者是为每个学习器定义一个权重，这个权重适配该学习器，通过对损失函数优化确定。3.前者主要是对错误分类样本的权重修正模型，后者是根据残差来修正模型。4.对学习器的选择，前者这能选择简单的分类器，如决策树，后者可以选择回归和分类问题，可以是复杂度较高的问题。

Plus：在Gradient Boosting中，训练新的弱学习器和更新模型参数（新的弱学习器的权重）：

新的弱学习器，是根据损失函数，求得负梯度的表达式，举例比如用的是MSE，负梯度的表达式是y^hat-g(x)（也就是残差，但是如果损失函数不是MSE负梯度就不是残差了），新的弱学习器实际上就是这个负梯度，但是为了确定新的弱学习器的回归模型或预测模型，我们将以往的输入和由输入得到的负梯度组成数据集，然后通过决策树或其他线性网络拟合新的弱学习器（不过拟合），这样我们就得到了新的弱学习器。

更新模型（权重参数）：将权重和新的学习器相乘与原来的串行学习器相加，得到新的模型，然后代入损失函数，对损失函数求这个新权重参数的导数，令其为零（这是理论上的，实际中，就是用SGD Adam等来计算权重参数的最佳值，使得损失函数最小化）。

e.g.：stacking：训练多个基本模型，然后另外训练一个元模型（Meta-Model）来结合基本模型的预测结果得到输出，来提高整体的性能。

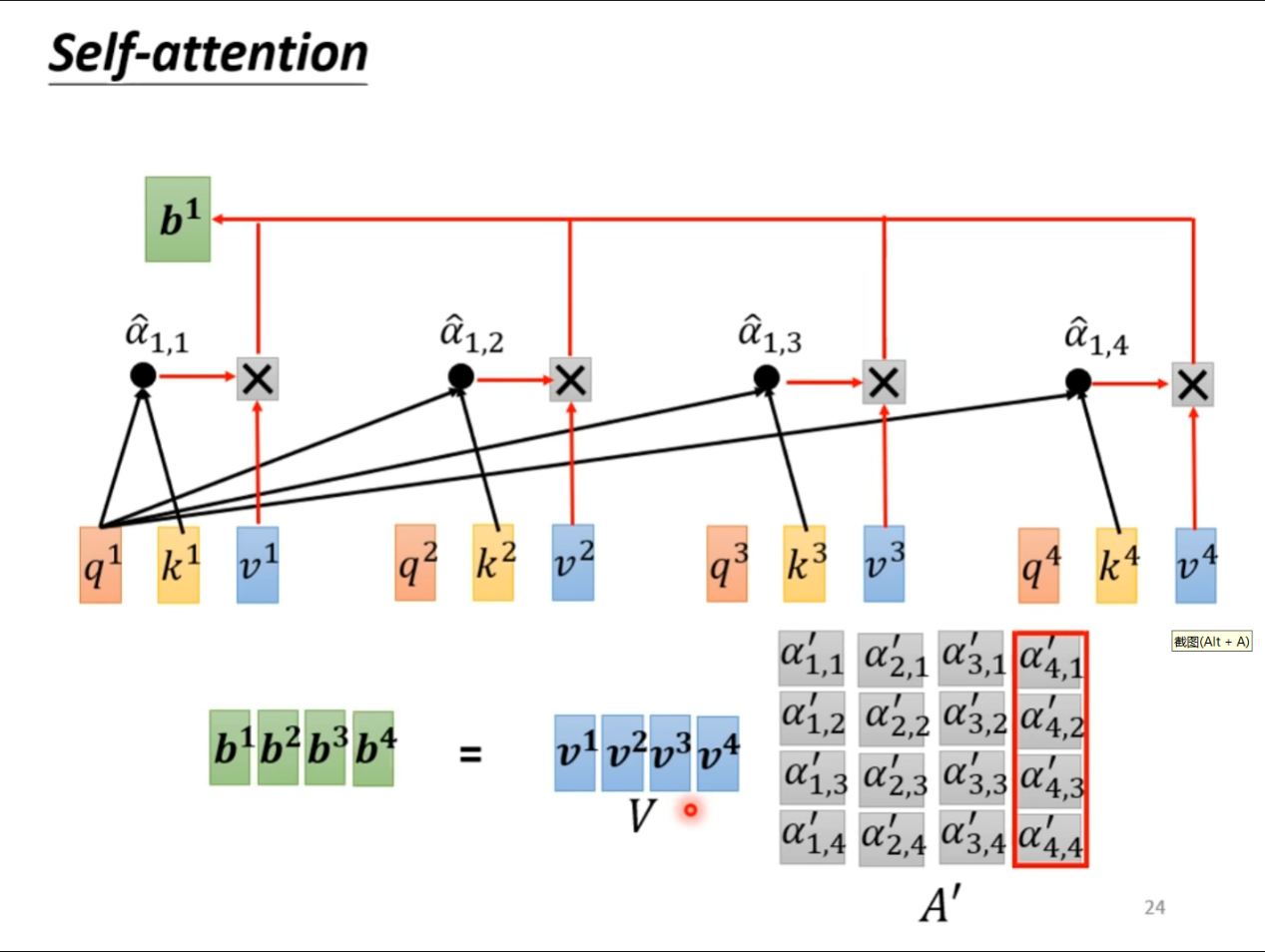
具体的步骤：1.将数据划分为两份（或多份）不重叠的子集，将第一部分子集用于训练基本模型，另一部分用于训练模型预测元模型。2.基本模型训练可采用多种网络（各种算法：决策树、支持向量机、神经网络等等）然后使用一部分的数据子集。3.元模型的训练使用另一部分数据子集，将该子集的数据标签，与子集输入基本模型得到的输出组成新的数据集用来训练该元模型，该元模型可以是简单的机器学习模型，目标是集成基本模型的预测结果来获得更准确的最终预测。

可对集成学习使用K-fold交叉验证，每折对应一个学习器或一个基本模型。

Self-attention：

Self-attention的优势是，输入为任意长度都行，简而言之就是对全局的特征都会考虑到，但是缺点就是对局部特征没办法做到精细的考虑。

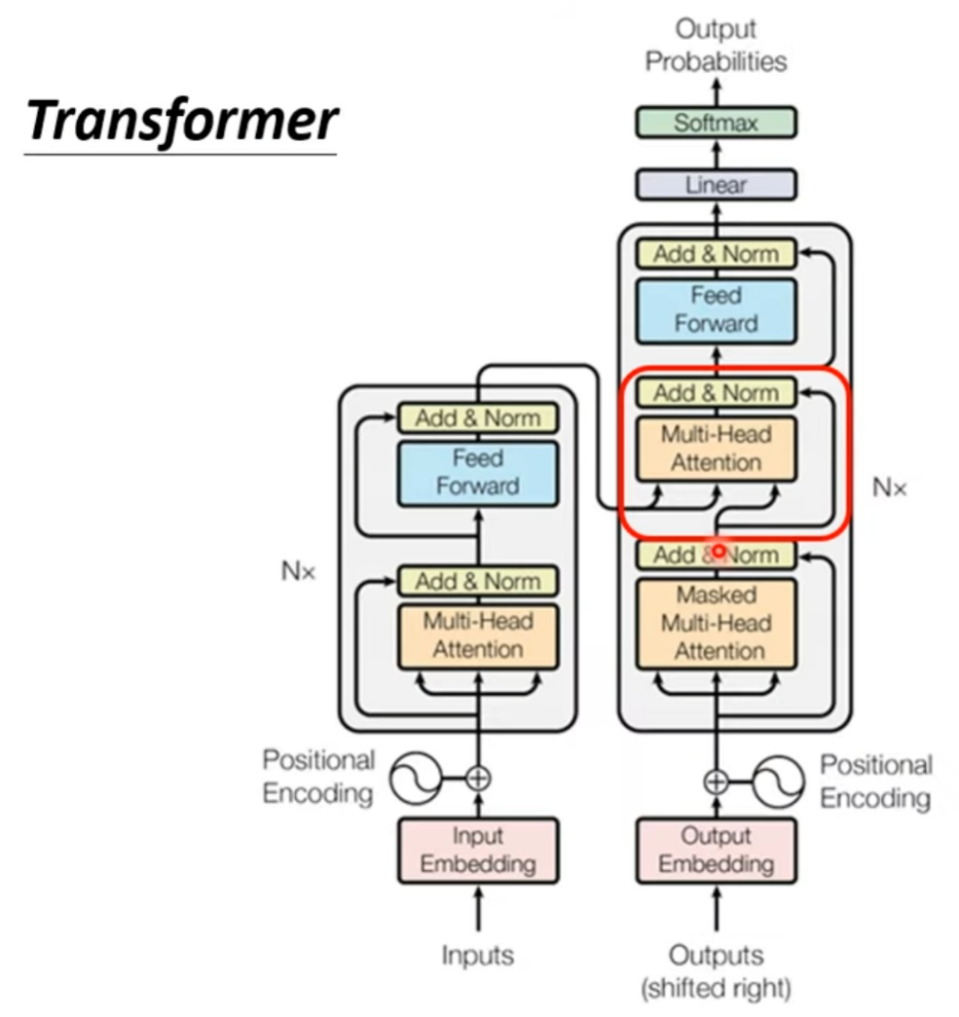
Self-attention的原理就是，对每个输入经过Wq，Wk，Wv矩阵变换成多个向量，q，k，v，然后各个q和每个输入的k相乘得到多个向量的相关程度，然后得到的每个相关程度再乘以各自的v，所有的都相乘后得到一个b向量，这个b向量即输出，b里面的每个元素对应每个输入。如下图



Self-attention有一种叫multi-head self-attention就是有一个W对q进行重新计算，形成两个新的q向量，然后两个q向量会分别形成两个b向量，两个b向量再经过一个W得到一个b向量即为输出。

常常将self-attention放在transformer里面作为一个组件。Transformer

Transformer：个人理解，transformer就是将一个self-attention作为组件，然后配套residual（残差）和线性（linear relu），再加上layer-norm,还有cross-attention,word-embedding这些。然后transformer的改进型是conformer是将transformer和convolution结合在一起，可以对局部变量有更好的考量。其中decoder有一个masked multi-head attention的意思是，在每个输入输出bi的时候，只考虑输入之前的k。



Word-embedding:

1. create a vocabulary
2. one-hot encoding
3. 获得词向量（word vector）--word2vec（skip-gram，cbow），glove，fasttext
4. 上面的几种方法，可以分为两大类，一类是count-based，另一类是prediction-based

Skip-gram和cbow是prediction-based，glove是count-based和prediction-based结合

Glove详解[GloVe最全面、最深度的解析\_glove详解\_Xu\_Wave的博客-CSDN博客](https://blog.csdn.net/qq_22795223/article/details/105737651?ops_request_misc=%257B%2522request%255Fid%2522%253A%2522169563983316800182787698%2522%252C%2522scm%2522%253A%252220140713.130102334..%2522%257D&request_id=169563983316800182787698&biz_id=0&utm_medium=distribute.pc_search_result.none-task-blog-2~all~top_positive~default-1-105737651-null-null.142%5ev94%5econtrol&utm_term=GloVe&spm=1018.2226.3001.4187)