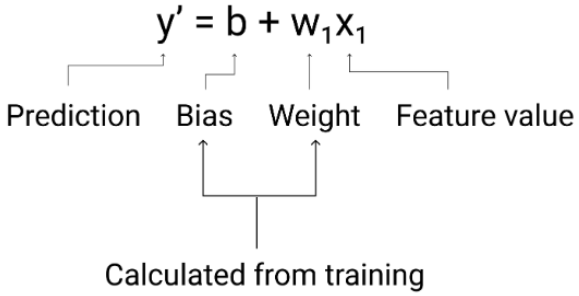
Machine Learning Google Developer

线性回归



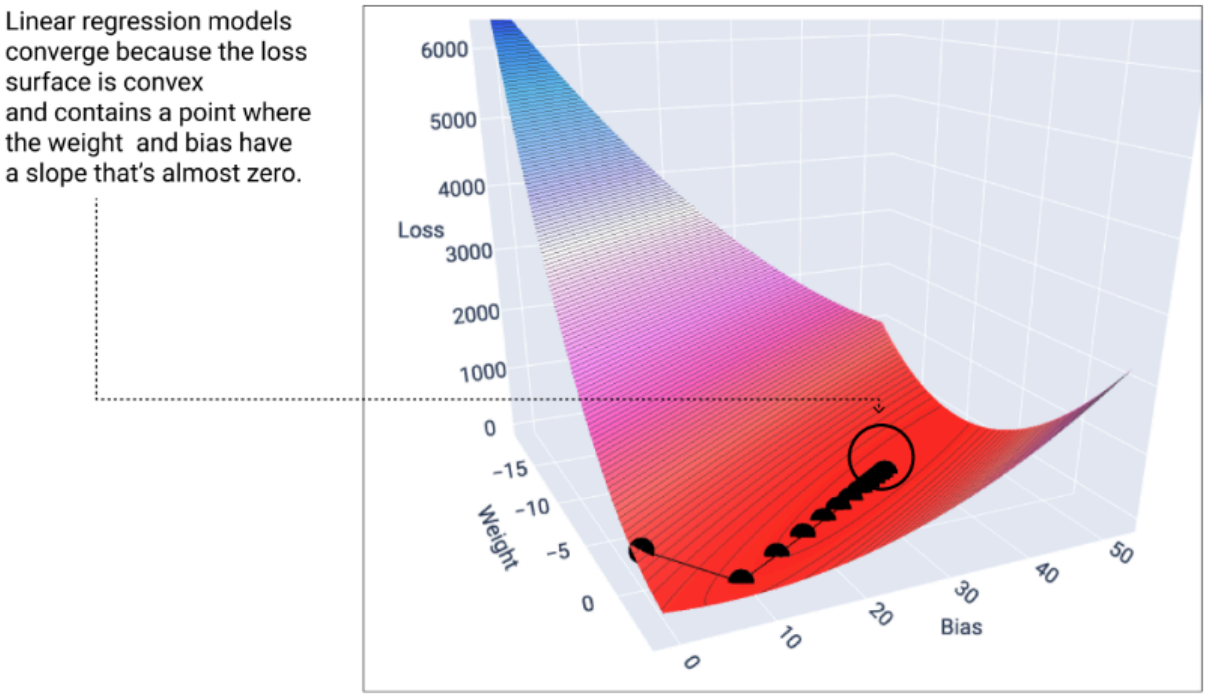
在训练期间，模型会计算可产生最佳结果的权重和偏差。

在线性回归中，有四种主要的损失函数，如下表所示。

损失类型	定义	公式
L_1 损失	预测值与实际值之间差异的绝对值的总和。	$\sum \lvert actual\ value - predicted\ value \rvert$
平均绝对误差 (MAE)	一组示例的 L_1 损失的平均值。	$\frac{1}{N} \sum \lvert actual\ value - predicted\ value \rvert$
L_2 损失	预测值与实际值之间的平方差的总和。	$\sum (actual\ value - predicted\ value)^2$
均方误差 (MSE)	一组示例的 L_2 损失的平均值。	$\frac{1}{N} \sum (actual\ value - predicted\ value)^2$

损失是一个数值指标，用于描述模型的**预测**有多大偏差。损失函数用于衡量模型预测与实际标签之间的距离。训练模型的目标是尽可能降低损失，将其降至最低值。与 L1 损失函数相比，L2 损失函数对离群值的惩罚更高。

梯度下降法是一种迭代过程，用于找到可最大限度地减少损失的最佳权重和偏差。

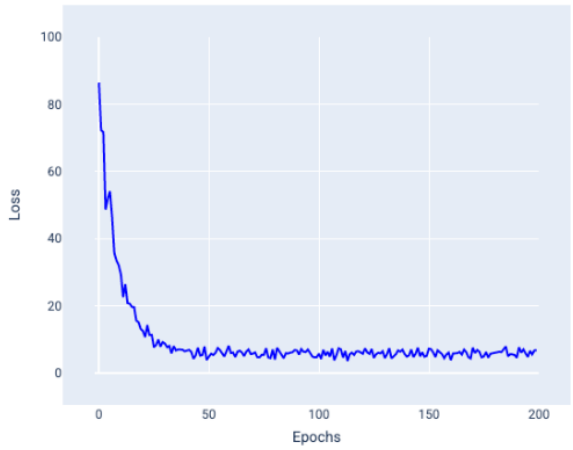


线性模型的损失函数始终会生成凸函数。因此，当线性回归模型收敛时，我们就知道该模型找到了可产生最小损失的权重和偏差。

超参数：三个常见的——学习率，批量大小，epoch. 参数是模型在训练期间计算的值，是模型的一部分，而超参数是你控制的值。

- 将学习率翻倍可能会导致学习率过大，从而导致权重“跳来跳去”，增加收敛所需的时间。

随机梯度下降法 (SGD)：随机梯度下降法每次迭代只使用一个示例（批量大小为 1）。在足够的迭代次数下，SGD 会起作用，但噪声很大。“噪声”是指训练期间的变化，会导致在迭代过程中损失增加而不是减少。“随机”一词表示每个批次包含的一个示例是随机选择的。



随着模型使用 SGD 更新其权重和偏差，损失略有波动，导致损失图表中出现噪声。使用随机梯度下降法可能会在整个损失曲线中产生噪声，而不仅仅是在收敛附近。

较大的批处理大小有助于减少数据中存在离群值的负面影响。

小批次随机梯度下降法 (mini-batch SGD)：小批次随机梯度下降法是全批次梯度下降法和 SGD 之间的折衷方案。对于N个数据点，批处理大小可以是任何大于 1 且小于N的数字。模型会随机选择每个批处理中包含的示例，对其梯度求平均值，然后每迭代一次更新权重和偏差。

逻辑回归

S 型函数

您可能想知道逻辑回归模型如何确保其输出表示概率，始终输出介于 0 到 1 之间的值。由于 会发生一系列函数，这些函数称为逻辑函数 其输出具有相同的特征。标准逻辑函数， 也称为 **S 型函数**（*sigmoid* 表示“s 形”），其 公式：

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

图 1 显示了 sigmoid 函数的相应图表。

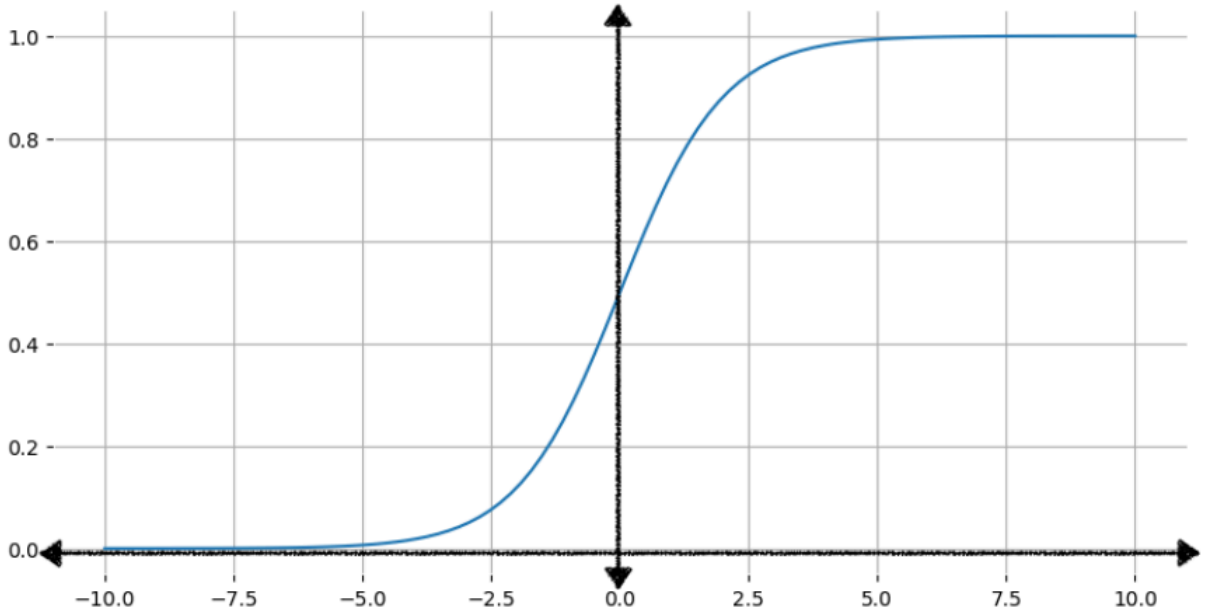


***图 1.** S 型函数的图形。曲线接近 0 因为 x 值减少到负无穷大，而 1 则等于 x 值越接近无穷大。*

逻辑回归 模型的训练过程与 **线性回归** 两个关键区别：

- 逻辑回归模型使用 **对数损失函数**作为损失函数 而不是**平方损失函数**。
- 应用正则化 是防止 **过拟合**。（例如，L2，早停）

$$\text{Log Loss} = \sum_{(x,y)\in D} -y \log(y') - (1 - y) \log(1 - y')$$

分类

混淆矩阵

概率得分不是现实情况，也不是**标准答案**。二元分类器的每个输出都有四种可能的结果。在垃圾邮件分类器示例中，如果您将标准答案作为列，将模型的预测结果作为行，则会得到以下表格（称为**混淆矩阵**）：

	实际正例	实际负例
预测为正例	真正例 (TP)：被正确分类为垃圾邮件的垃圾邮件。这些是系统自动发送到“垃圾邮件”文件夹的垃圾邮件。	假正例 (FP)：非垃圾邮件被误分类为垃圾邮件。这些是最终进入“垃圾邮件”文件夹的正常电子邮件。
预测为负	假负例 (FN)：垃圾邮件被错误分类为非垃圾邮件。这些是垃圾邮件过滤器未捕获的垃圾邮件，并已进入收件箱。	真负例 (TN)：非垃圾邮件被正确分类为非垃圾邮件。 这些是直接发送到收件箱的正规电子邮件。

请注意，每行中的总数表示所有预测正例 (TP + FP) 和所有预测负例 (FN + TN)，无论其有效性如何。与此同时，每个列中的总和会显示所有真实正例 (TP + FN) 和所有真实负例 (FP + TN)，而不会考虑模型分类。

如果实际正例的总数与实际负例的总数不接近，则表示数据集**不平衡**。不平衡数据集的实例可能是包含数千个云彩照片，而您感兴趣的罕见云类型（例如卷云云）仅出现几次。

随着阈值的提高，模型预测的正例（包括真正例和假正例）总数可能会减少。阈值为 0.9999 的垃圾邮件分类器仅在认为某封电子邮件分类为垃圾邮件的可能性至少为 99.99% 时，才会将某封电子邮件标记为垃圾邮件，这意味着极不可能错误地为合法电子邮件添加标签，但也很可能会错过实际的垃圾邮件。

$$\text{Accuracy} = \frac{\text{correct classifications}}{\text{total classifications}} = \frac{TP + TN}{TP + TN + FP + FN}$$

准确性是指所有分类（无论是正类还是负类）正确分类的比例。在数据集平衡且两个类别中的示例数量相近的情况下，可以用作衡量模型质量的粗略指标。因此，它通常是默认评估指标。

对于严重不均衡的数据集（其中一个类别出现的频率非常低，例如 1%），如果模型 100% 都预测为负类，则准确率得分为 99%，尽管该模型毫无用处。

$$\text{Recall (or TPR)} = \frac{\text{correctly classified actual positives}}{\text{all actual positives}} = \frac{TP}{TP + FN}$$

召回率 (TPR)，即所有实际正例被正确分类为正例的比例；召回率的另一个名称是**检测概率**

$$\text{FPR} = \frac{\text{incorrectly classified actual negatives}}{\text{all actual negatives}} = \frac{FP}{FP + TN}$$

假正例率 (FPR) 是指被*错误*地归类为正例的所有实际负例所占的比例，也称为**误报概率**。

$$\text{Precision} = \frac{\text{correctly classified actual positives}}{\text{everything classified as positive}} = \frac{TP}{TP + FP}$$

精确率是指模型所有正分类分类中实际正分类的比例

召回率 （真正例率）	在假负例比假正例开销更高时使用。
假正例率	在假正例比假负例开销更高时使用。
精确率	当正例预测的准确性非常重要时，请使用此方法。

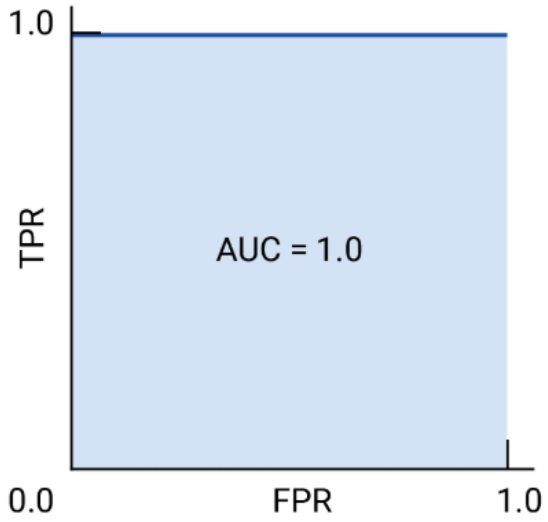


图 1. 假设的完美模型的 ROC 和 AUC。

ROC 曲线的绘制方法是计算召回率 (TPR) 和假正例率 (FPR) 值。

ROC 曲线下面积 (AUC) 表示模型的概率，

AUC = 0.5 表示模型的性能与随机猜测相同，即模型没有预测能力。

神经网络

结点，层

常用激活函数：Sigmoid, Tanh

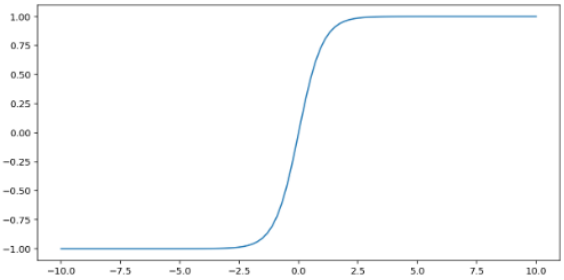


Figure 5. Plot of the tanh function.

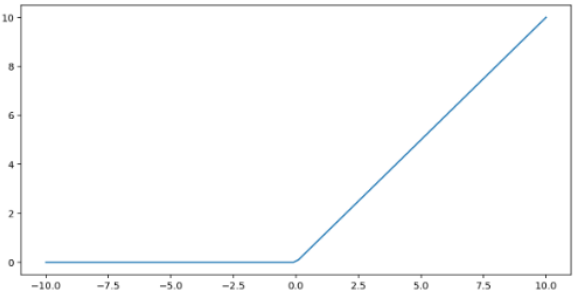


Figure 6. Plot of the ReLU function.

ReLU 作为激活函数的效果通常优于 S 型函数或 tanh 等平滑函数，因为它在神经网络训练期间不易受到梯度消失问题的影响。与这些函数相比，ReLU 的计算也更容易。
较低神经网络层的梯度可能会变得很小，在深度网络中计算这些梯度可能涉及许多小项乘法。

在较低层的梯度接近0时，梯度消失了，这使得训练速度非常慢，或者根本无法训练。

可以考虑Dropout

嵌入

低位嵌入解决 1. 高维导致独热难以使用 2. 向量间缺少有意义关系 的问题。

嵌入**embedding**是 你可以将空间转换为相对低维的空间，有助于更轻松地对大型语言模型进行机器学习 **特征向量**。理想情况下，嵌入会捕获 将含义更相似的输入放在更接近的位置， 集中在嵌入空间中。例如，良好的嵌入会将 字词“汽车”靠近“车库”比“大象”更专业我们可以训练嵌入， 并在不同模型中重复使用。

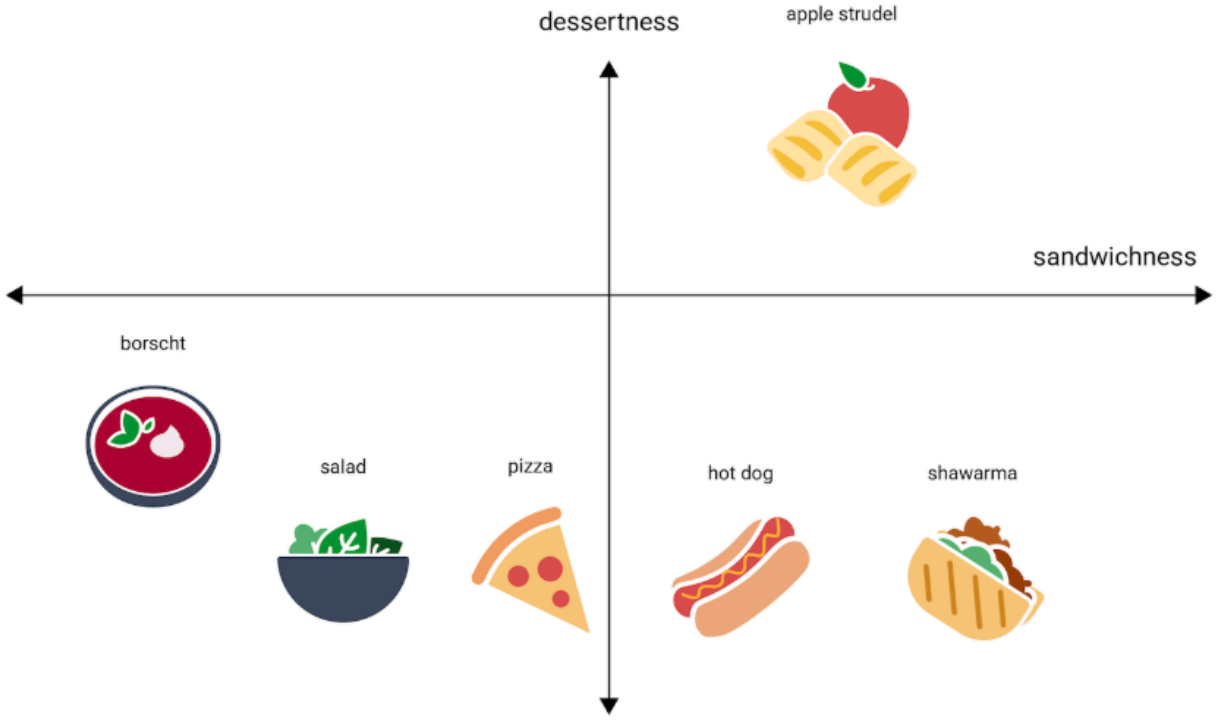


图 4. 同时以“三明治”和“三明治”的形式标出食物和“dessertness”等字词。

在嵌入中，可以计算任意两个项之间的距离 从数学角度 可以解释为两者之间的相对相似度 项。两个距离很近的事物，例如 shawarma 和 hot dog 与图 4 中的两个对象之间的距离， 例如 apple strudel 和 borscht。

word2vec 嵌入层 Embedding Projector 工具

嵌入层是可以自己构造的，比如在输入层后加一个结点数为d的embedding layer。

文本情景下的嵌入解决了静态文本嵌入在不同语境下含义不同的限制。ELMo，BERT，Transformer (self-attention)

BERT (Bidirectional Encoder Representations from Transformers) 使用了transformer框架，因而依赖self-attention。使用encoder部分，用于生成优质的文本表达，而不只是为了分类。Bidirectional， 在非监督学习时用了masking

大语言模型

A **language model** estimates the probability of a **token** or sequence of tokens occurring within a longer sequence of tokens. A token could be a word, a subword (a subset of a word), or even a single character. **token 词元**

N-gram 语言模型：N grams时用于构建语言模型的已排序的文字序列。N是一个序列中词语的数量。比如对于2-gram, bigram；假如我们有you are very nice,那么2-gram就是 you are, are very, very nice. 当3-gram，就是you are very, are very nice.

如果给定两个词语作为输出，那么3gram语言模型就可以预测第三个词语的概率。

当n数量比较小的时候，我们缺乏足够多的文本去预测，因而更长的N可以比更短的N更多的信息。但是这也导致N很大的时候，我们通常只有极少数的实例出现过这N个词语。**如果训练文本足够足够大， 那么N越大通常效果越好。**

RNN比N-gram提供更多的信息，RNN是基于一个词元序列训练的神经网络。同时，如果很长的文本用于训练RNN，通常会发生梯度消失的问题。

LLMs大语言模型，它存储远远多于RNN的参数，它可以收集更遥远的文本。

什么是Transformer？

完整的 Transformer 由 encoder 和 decoder组成，encoder将输入文本转换成中间表达，是一个巨大的神经网络；decoder把中间表达转化为有用的文本，也是一个巨大的神经网络。

当然部分的transformer也是可以用的，只有encoder的架构可以预测 输入序列中的 某个词元；或者创造足够负责的embedding，从而在另一个系统中作为输入（比如最终用于分类）

Decoder-only的框架可以从已经生成出的文本中，生成新词元。比如续写。

什么是Self-attention?

"How much does each other token of input affect the interpretation of this token?"

这里的自己指的就是input序列。因为有些注意力机制会去研究输入和输出序列中的词语关系。但是自注意力只关心输入序列中的关系。假如我们把每个词元理解为一个单词，那么对于一个输入的完整句子，比如：

```
The animal didn't cross the street because it was too tired.
```

这里的11个词语，每个都会关心剩下的10个词语。“这10个对自己的影响有多大？”

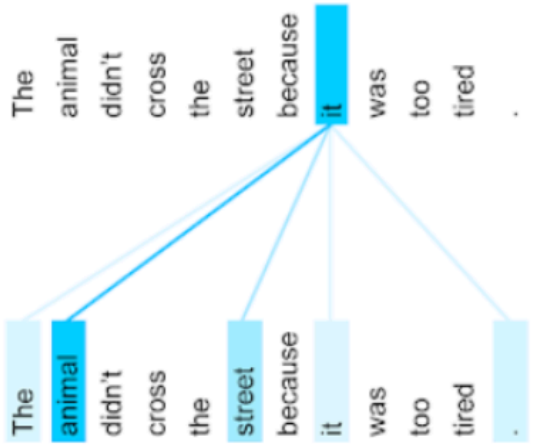


Figure 3. *Self-attention for the pronoun it. From Transformer: A Novel Neural Network Architecture for Language Understanding.*

Bidirectional的注意力是指，会对词语前后文都检验关系，计算相关分值。unidirectional只考虑一侧的词语。对于生成一整个序列的表达时，双边非常有用。但是对于一个词一个词来生成序列时，单边自注意力是需要的。因此，encoder通常双边，decoder单边。

什么是多头自注意力？

每个自注意力层通常由多个**自注意力头**组成。层的输出是不同 head 输出的数学运算（例如加权平均值或点积）。

由于每个自注意力层都初始化为随机值，因此不同的头可以学习被注意到的每个字词与附近字词之间的不同关系。例如，上一部分介绍的自注意力层侧重于确定人称代词 **it** 所指的名词。不过，其他自注意力层可能会学习每个词与每个其他词的语法相关性，或学习其他互动。

Transformer 包含多个自注意力层，并且每个自注意力层还包含多个自注意力头，因此大 O 函数实际上是：O(N^2 · S · D)

其中：

- S 是自注意力层的数量。
- D 是每个层的 head 数量。

蒸馏，微调 （蒸馏的效果会变差，但是效率更高，更轻量化）

A foundation LLM can serve as a platform rather than a solution.

Transforming a foundation LLM into a solution that meets specific needs requires **Find-tuning**. A secondary process called **distillation** generates a smaller version of the fine-tuned model.

知识蒸馏（Distillation）及其应用解析

知识蒸馏是一种常见的模型压缩技术，其中 **大模型（教师模型，Teacher Model）** 通过 **批量推理（Bulk Inference）** 生成标签数据，而 **小模型（学生模型，Student Model）** 则利用这些标签数据进行训练，以实现更高效的部署和推理。

工作原理

1. **教师模型生成数据标签**：使用计算资源较大的教师模型对数据进行推理（预测），得到一批标注的数据。
2. **学生模型训练**：利用这些数据训练一个规模更小、计算成本更低的学生模型。
3. **高效推理**：训练好的学生模型可用于在线服务，例如在实时系统中进行预测，而无需使用庞大的教师模型。