

# 第一讲

## cv和dl简史

- Alexnet网络进行图像分类：深度学习走向主流
- 卷积层、池化、全连接层

## cs231n概述

- dl基础
- 感知与理解视觉世界

传统视觉模型（如 CNN、RNN）在处理长序列数据（如视频帧序列、图像 - 文本跨模态数据）时存在明显局限：CNN 依赖局部卷积核，难以捕捉长距离特征关联；RNN 按时序逐帧处理，易出现“长序列梯度消失”，且并行计算效率低。

Attention 机制与 Transformer 的出现，正是为了突破这些局限——**Attention 机制通过“动态分配注意力权重”，让模型聚焦对任务更重要的信息；Transformer 则基于 Attention 机制构建全局依赖建模能力，同时支持高效并行计算**，成为处理长序列、跨模态视觉任务的核心架构。

### Attention 机制：“聚焦关键信息”的核心逻辑

#### 1. 核心原理

Attention 机制模拟人类视觉的“选择性关注”行为（如观察图像时优先关注物体主体而非背景），其本质是通过计算“查询 (Query, Q)”与“键 (Key, K)”的相似度，为每个“值 (Value, V)”分配权重，最终输出加权后的特征表示，公式可简化为：

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

Q：模型当前任务的“查询”（如视频理解中当前帧的特征）；

- K、V：待匹配的“键 - 值对”（如视频中所有帧的特征，K 与 Q 维度一致，V 为实际需加权的特征）；

$$\frac{1}{\sqrt{d_k}}$$

：防止  $(QK^T)$  数值过大导致 Softmax 输出趋于极端；

- Softmax：将相似度转化为 0-1 之间的权重，确保权重总和为 1。

#### 2. 视觉任务中的作用

在计算机视觉中，Attention 机制主要用于“筛选关键特征”，典型场景包括：

视频理解：为不同帧分配权重（如动作发生的帧权重更高，背景帧权重更低），避免无关帧干扰；

图像分割：为像素分配权重（如分割“猫”时，优先关注猫的轮廓、纹理像素，弱化背景像素）；

跨模态任务：如“图像 - 文本匹配”（CLIP 模型），计算图像特征 (Q) 与文本特征 (K) 的相似度，聚焦匹配度最高的图文信息。

### Transformer：基于 Attention 的全局建模架构

#### 1. 核心结构

Transformer 最早由 Vaswani 等人于 2017 年提出，最初用于自然语言处理（NLP），后被广泛适配到计算机视觉领域。其核心由“编码器（Encoder）”和“解码器（Decoder）”组成，但在视觉任务中（如图像分类、视频理解），常以“编码器”为核心，结构特点包括：

多头注意力 (Multi-Head Attention)：并行执行多个 Attention 计算 (“多头”), 每个头聚焦不同维度的特征关联 (如一个头关注物体形状, 另一个头关注颜色), 最后将多头结果拼接, 提升特征表达的丰富性;

位置编码 (Positional Encoding)：由于 Transformer 无时序 / 空间顺序感知能力, 通过添加 “位置编码”(如正弦余弦函数生成的向量), 为图像像素、视频帧赋予空间 / 时序位置信息;

前馈神经网络 (Feed-Forward Network, FFN)：对每个注意力输出的特征进行独立非线性变换, 增强模型的拟合能力;

残差连接 (Residual Connection) 与层归一化 (Layer Normalization)：缓解深层网络的梯度消失问题, 加速训练收敛。

## 2. 在视觉任务中的适配与应用

课程中明确提到, Transformer 已成为 “超越多层感知机” 的关键模型, 在计算机视觉中的典型应用包括:

视频分类与理解：处理视频的长帧序列，通过全局 Attention 捕捉帧间动作关联 (如 “跑步” 动作中不同帧的肢体位置变化);

视觉 - 语言模型 (VLM)：如 CLIP 模型, 通过 Transformer 编码器分别处理图像与文本特征, 再通过 Attention 计算图文相似度, 实现 “文本检索图像”“图像分类” 等跨模态任务;

图像分割与目标检测：如 DETR 模型, 用 Transformer 直接建模图像全局特征与目标框的关联, 替代传统的锚框 (Anchor) 机制, 简化检测流程。

- 生成式与交互式视觉智能
- 以人为本的应用与意义

# 第二讲：使用线性分类器进行图像分类

## 最近邻分类器

- k近邻：k和距离是超参数（每种都试试，挑最好的）
- 训练集和测试集（训练集可以k折交叉验证选择超参数）

## 线性分类器

- 损失函数、优化
- softmax多分类（最大损失为无穷、相等初始化损失为 $\log$ （类别数））
- 多分类svm

# 第三讲

## 正则化

- 偏好更简单的模型

## 优化

实际应用中：应始终使用解析梯度，但要用数值梯度检查实现是否正确。这称为梯度检查。

# 第四讲

## 神经网络

- 激活函数：得到非线性效果
- 矩阵的反向传播

## 作业1代码

### knn

- argsort()：返回排列下标
- bincount()：返回下标数字出现个数
- argmax()：返回最大值下标
- numpy处理不同维度矩阵相减会使用广播机制
- 数学拆解+np广播机制：

```
x_square = np.sum(x**2, axis=1, keepdims=True)
x_train_square = np.sum(self.X_train**2, axis=1, keepdims=True)
cross_term = np.dot(x, self.X_train.T)
dists = np.sqrt(x_square - 2 * cross_term + x_train_square.T)
```

- 矩阵计算可以加速：避免py循环的检查、可以并行计算
- np.array\_split()：分割样本用于交叉验证
- ```
for k in k_choices:
    k_to_accuracies[k] = []
    for i in range(num_folds):
        x_val = x_train_folds[i]
        y_val = y_train_folds[i]
        x_tr = np.vstack(x_train_folds[:i] + x_train_folds[i+1:])
        y_tr = np.hstack(y_train_folds[:i] + y_train_folds[i+1:])

        classifier.train(x_tr, y_tr)
        dists = classifier.compute_distances_no_loops(x_val)
        y_val_pred = classifier.predict_labels(dists, k=k)

        num_correct = np.sum(y_val_pred == y_val)
        accuracy = float(num_correct) / len(y_val)
        k_to_accuracies[k].append(accuracy)
```

- vstack()垂直拼接、hstack()水平拼接
- 后续还要可视化准确率和标准差进行k值选择，最后用最好的k值训练整个模型

# softmax

- 图像可以中心化（减去均值）

```
# split the data into train, val, and test sets. In addition we will
# create a small development set as a subset of the training data;
# we can use this for development so our code runs faster.
num_training = 49000
num_validation = 1000
num_test = 1000
num_dev = 500

# Our validation set will be num_validation points from the original
# training set.
mask = range(num_training, num_training + num_validation)
x_val = x_train[mask]
y_val = y_train[mask]

# Our training set will be the first num_train points from the original
# training set.
mask = range(num_training)
x_train = x_train[mask]
y_train = y_train[mask]

# We will also make a development set, which is a small subset of
# the training set.
mask = np.random.choice(num_training, num_dev, replace=False)
x_dev = x_train[mask]
y_dev = y_train[mask]

# We use the first num_test points of the original test set as our
# test set.
mask = range(num_test)
x_test = x_test[mask]
y_test = y_test[mask]
```

- 建立一个开发集（小），模型调试、超参数调整、梯度检查等操作时，用开发集计算损失 / 梯度的速度更快（无需遍历全量训练数据）。能大幅缩短代码运行时间，快速验证梯度实现是否正确、超参数是否合理，提升开发迭代效率。

- 数值梯度的计算如下，很慢：

| current W:                                                                          | $W + h$ (third dim):                                                                                 | gradient dW:                                                 |
|-------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------|--------------------------------------------------------------|
| [0.34,<br>-1.11,<br>0.78,<br>0.12,<br>0.55,<br>2.81,<br>-3.1,<br>-1.5,<br>0.33,...] | [0.34,<br>-1.11,<br>0.78 + <b>0.0001</b> ,<br>0.12,<br>0.55,<br>2.81,<br>-3.1,<br>-1.5,<br>0.33,...] | [-2.5,<br>0.6,<br>?,<br>?,<br>?,<br>?,<br>?,<br>?,<br>?,...] |

loss 1.25347      loss 1.25347

| current W:                                                                          | $W + h$ (third dim):                                                                                 | gradient dW:                                                                                                                                                  |
|-------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------|
| [0.34,<br>-1.11,<br>0.78,<br>0.12,<br>0.55,<br>2.81,<br>-3.1,<br>-1.5,<br>0.33,...] | [0.34,<br>-1.11,<br>0.78 + <b>0.0001</b> ,<br>0.12,<br>0.55,<br>2.81,<br>-3.1,<br>-1.5,<br>0.33,...] | [-2.5,<br>0.6,<br><b>0</b> ,<br>?,<br>$(1.25347 - 1.25347)/0.0001 = 0$ ,<br>$\frac{df(x)}{dx} = \lim_{h \rightarrow 0} \frac{f(x + h) - f(x)}{h}$ ,<br>?,...] |

loss 1.25347      loss 1.25347

- 实际操作：

实际应用中：应始终使用解析梯度，但要用数值梯度检查实现是否正确。这称为梯度检查。

- ```
def softmax_loss_naive(W, X, y, reg):
    """  

    Softmax loss function, naive implementation (with loops)  

    Inputs have dimension D, there are C classes, and we operate on minibatches  

    of N examples.
```

```

Inputs:
- W: A numpy array of shape (D, C) containing weights.
- X: A numpy array of shape (N, D) containing a minibatch of data.
- y: A numpy array of shape (N,) containing training labels; y[i] = c means
  that X[i] has label c, where 0 <= c < C.
- reg: (float) regularization strength

Returns a tuple of:
- loss as single float
- gradient with respect to weights W; an array of same shape as W
"""

# Initialize the loss and gradient to zero.
loss = 0.0
dw = np.zeros_like(W)

# compute the loss and the gradient
num_classes = W.shape[1]
num_train = X.shape[0]
for i in range(num_train):
    scores = X[i].dot(W)

    # compute the probabilities in numerically stable way
    scores -= np.max(scores)
    p = np.exp(scores)
    p /= p.sum() # normalize
    logp = np.log(p)

    loss -= logp[y[i]] # negative log probability is the loss
    #####对中间变量得分的梯度进行修改#####
    p[y[i]] -= 1 # 对中间变量得分的梯度进行修改
    dw += np.outer(X[i], p) # 链式求导, 计算梯度
    #####
    # normalized hinge loss plus regularization
loss = loss / num_train + reg * np.sum(W * W)

#####
# TODO: #
# Compute the gradient of the loss function and store it dw. #
# Rather than first computing the loss and then computing the derivative, #
# it may be simpler to compute the derivative at the same time that the #
# loss is being computed. As a result you may need to modify some of the #
# code above to compute the gradient. #
#####
dw = dw / num_train + 2 * reg * W # 加上正则化项的梯度

return loss, dw

```

- 上述为softmax损失计算，同时返回梯度

- np.zeros\_like()、np.reshape()、np.outer()
- 梯度检查：

```
# Numerically compute the gradient along several randomly chosen dimensions, and
# compare them with your analytically computed gradient. The numbers should match
# almost exactly along all dimensions.
from cs231n.gradient_check import grad_check_sparse
f = lambda w: softmax_loss_naive(w, x_dev, y_dev, 0.0)[0]
grad_numerical = grad_check_sparse(f, w, grad)
```

- lambda：可f(w)，w为传入参数
- 向量化技巧：

```
num_train = x.shape[0]
scores = x.dot(w) # (N, C)
scores -= np.max(scores, axis=1, keepdims=True)
p = np.exp(scores)
p /= p.sum(axis=1, keepdims=True)
logp = np.log(p)

loss = -np.sum(logp[np.arange(num_train), y]) # 很关键，记住这个技巧
loss += reg * np.sum(w * w)
```

- 安全检查：可视化训练过程中的损失变化趋势
- 使用验证集选取超参数学习率和正则化强度，还要对不同组合进行可视化，多样可视化

## 神经网络

np.prod(): 计算形状维度相乘结果

```
x = np.linspace(-0.1, 0.5, num=input_size).reshape(num_inputs, *input_shape) # *解包出数据形状
```

前向传播、反向传播代码

```
out = np.maximum(0, x) # ReLU
```

整合线性层与ReLU层，方便代码编写

这里再复习一遍softmax损失代码编写

### TwoLayerNet:

- 仿射层 - ReLU激活层 - 仿射层 - softmax层
- 注意：该类不实现梯度下降算法；而是与一个独立的Solver（求解器）对象交互，由Solver对象负责执行优化过程。
- 模型的可学习参数存储在字典self.params中，字典将参数名称映射到numpy数组。
- init初始化：参数准备和reg（正则化强度）

- `loss()`: 要有测试模式和训练模式，测试前向传播，训练反向传播（利用前面写过的函数来实现，从而模块化，解耦合）
- `save()`、`load()`: 保存和加载模型参数

## solver:

- `Solver` (求解器) 封装了训练分类模型所需的全部逻辑。  
该类使用 `optim.py` 中定义的不同更新规则执行随机梯度下降 (SGD) 优化。

`Solver` 同时接收训练数据和验证数据及标签，因此能定期检查模型在训练集和验证集上的分类准确率，以便监测过拟合现象。

训练模型的流程：

1. 构造 `Solver` 实例，传入模型、数据集以及各类配置参数（学习率、批次大小等）；
2. 调用 `train()` 方法执行优化过程，完成模型训练。

`train()` 方法返回后，`model.params` 中将存储训练过程中在验证集上表现最佳的参数。

此外，`Solver` 实例的变量包含以下训练记录：

- `solver.loss_history`: 训练过程中所有损失值的列表；
- `solver.train_acc_history`: 每个 epoch (训练轮次) 模型在训练集上的准确率列表；
- `solver.val_acc_history`: 每个 epoch 模型在验证集上的准确率列表。

使用示例：

```
data = {
    'x_train': # 训练数据
    'y_train': # 训练标签
    'x_val': # 验证数据
    'y_val': # 验证标签
}
model = MyAwesomeModel(hidden_size=100, reg=10) # 初始化自定义模型
solver = Solver(model, data,
                 update_rule='sgd', # 优化更新规则 (如SGD)
                 optim_config={
                     'learning_rate': 1e-4, # 学习率 (必选参数)
                 },
                 lr_decay=0.95, # 学习率衰减系数
                 num_epochs=5, # 训练轮数
                 batch_size=200, # 批次大小
                 print_every=100) # 每迭代100次打印一次日志
solver.train() # 启动训练
```

## 训练调试：

- 使用我们上面提供的默认参数，你的模型在验证集上的准确率应该能达到约 0.36 (36%)。这个结果其实并不理想。想要定位问题所在，有两种实用策略：
  - 绘制优化过程中的曲线：通过绘制训练过程中的损失函数曲线，以及训练集和验证集的准确率曲线，可以直观观察模型的训练状态（比如是否收敛、是否过拟合 / 欠拟合）；
  - 可视化第一层学到的权重：在大多数基于视觉数据训练的神经网络中，第一层的权重通常会呈现出可识别的结构（比如边缘、纹理等简单视觉特征）。通过可视化这些权重，能判断模型是否真正学到了有效信息。

## 超参数调优

- 问题出在哪里？观察上述可视化结果，我们发现损失函数大致呈线性下降趋势 —— 这通常表明学习率可能过低。此外，训练准确率与验证准确率之间几乎没有差距，说明当前使用的模型容量不足，需要增大模型规模。但需注意：若模型过大，可能会出现更严重的过拟合现象，具体表现为训练准确率与验证准确率的差距显著增大。

#### 调优说明

超参数调优以及培养“参数如何影响最终性能”的直觉，是神经网络应用中的重要环节，因此需要大量实践。请你尝试调整以下各类超参数的不同取值：

- 隐藏层大小 (`hidden layer size`)
- 学习率 (`learning rate`)
- 训练轮数 (`number of training epochs`)
- 正则化强度 (`regularization strength`)

你也可以考虑调整学习率衰减 (`learning rate decay`)，但使用默认值通常也能获得较好的性能。

#### 预期效果

目标是在验证集上实现超过 48% 的分类准确率。我们优化后的最佳网络在验证集上的准确率可超过 52%。

#### 实验任务

本次练习的核心目标是：使用全连接神经网络在 `CIFAR-10` 数据集上获得尽可能优的性能（52% 可作为参考基准）。你可以自由实现各类优化技巧，例如：

- 采用 `PCA`（主成分分析）进行降维
- 添加 `Dropout`（随机失活）层
- 为优化器 (`solver`) 增加自定义功能
- 其他有效优化手段

## 图像特征

HOG 特征通过聚合局部区域的梯度方向分布，有效捕捉图像的纹理和边缘结构，对光照变化和局部形变具有一定鲁棒性，因此在目标检测和图像分类中被广泛应用。

HSV 色调直方图能有效表征图像的颜色分布，且相比 RGB 空间更符合人类对颜色的感知（例如对光照强度变化更鲁棒）。在特征融合中，它与 HOG 特征结合可同时捕捉图像的颜色和纹理信息，提升分类性能。

## 多层全连接神经网络

- 可任意指定隐藏层数

该网络包含任意数量的隐藏层、`ReLU`非线性激活函数和`softmax`损失函数。

还可选择实现`dropout`（丢弃法）和`batch/layer normalization`（批量/层归一化）。

对于一个具有  $L$  层的网络，其结构为：

{仿射层 - [批量/层归一化] - `ReLU` - [`dropout`] }  $\times (L-1)$  - 仿射层 - `softmax`

其中批量/层归一化和`dropout`为可选组件，{...}块会重复  $L-1$  次。

可学习参数存储在 `self.params` 字典中，将通过 `Solver` 类进行训练。

输入参数：

- `hidden_dims`: 整数列表，指定每个隐藏层的大小。
- `input_dim`: 整数，指定输入数据的维度（默认 $3 \times 32 \times 32$ 对应CIFAR-10图像）。
- `num_classes`: 整数，指定待分类的类别数。
- `dropout_keep_ratio`: 0到1之间的标量，指定dropout保留率。若`dropout_keep_ratio=1`，则网络不使用dropout。
- `normalization`: 网络使用的归一化类型。可选值为"batchnorm"（批量归一化）、"layernorm"（层归一化）或None（不使用归一化，默认值）。
- `reg`: 标量，指定L2正则化强度。
- `weight_scale`: 标量，指定权重随机初始化的标准差。
- `dtype`: numpy数据类型对象；所有计算将使用该数据类型。`float32`速度更快但精度较低，数值梯度检查时应使用`float64`。
- `seed`: 若不为None，则将该随机种子传入dropout层，使dropout层具有确定性，便于模型的梯度检查。

层归一化与批量归一化的核心区别是：它不依赖批次数据，而是对单个样本的特征维度进行归一化，不需要跟踪移动均值 / 方差，因此参数字典初始为空（后续也无需存储额外状态变量）。

合理性检查（有梯度检查）（优化规则也要进行梯度检查）：

作为另一个合理性检查（sanity check），请确保你的网络能够在一个仅包含 50 张图像 的小数据集上过拟合（overfit）。

首先，我们将尝试一个三层神经网络，其中每个隐藏层都有 100 个神经元（units）。

在下面的代码单元中，请调整学习率（learning rate）和权重初始化尺度（weight initialization scale），使得模型能够在 20 个训练轮次（epochs）内达到 100% 的训练准确率（training accuracy）。

现在，尝试使用一个\*\*五层神经网络\*\*，其中每一层都有 \*\*100 个神经元（units）\*\*，在 \*\*50 个训练样本\*\*上实现\*\*过拟合（overfit）\*\*。

同样地，你需要调整学习率（learning rate）和权重初始化尺度（weight initialization scale），但你应该能够在 20 个训练轮次（epochs）内让模型达到 100% 的训练准确率（training accuracy）。

## 第五讲：CNN

HOG：方向梯度直方图

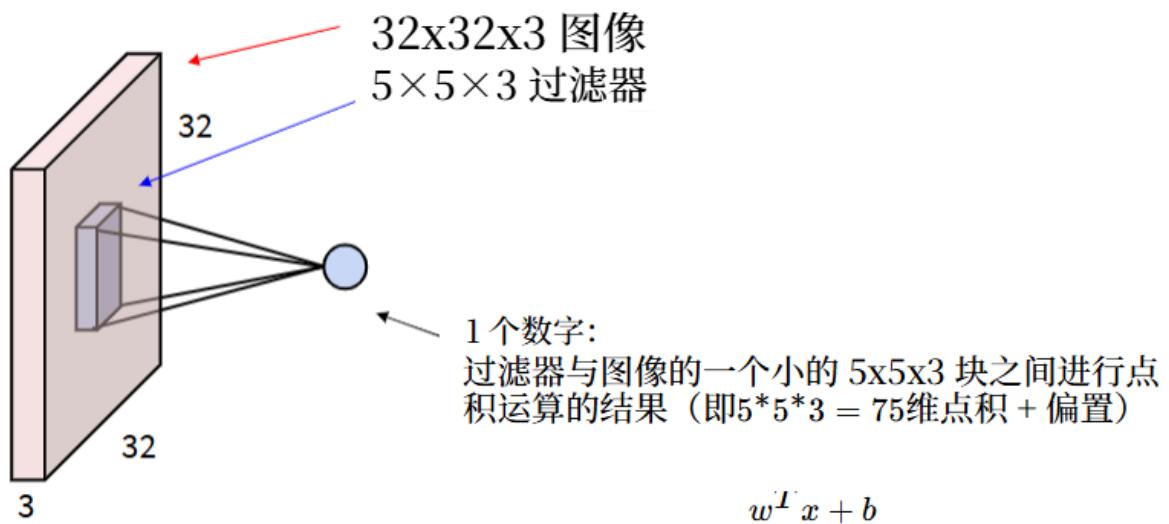
卷积和池化算子在提取特征的同时会保留二维图像结构

全连接层在最后构成一个多层次感知机（MLP）来预测分数

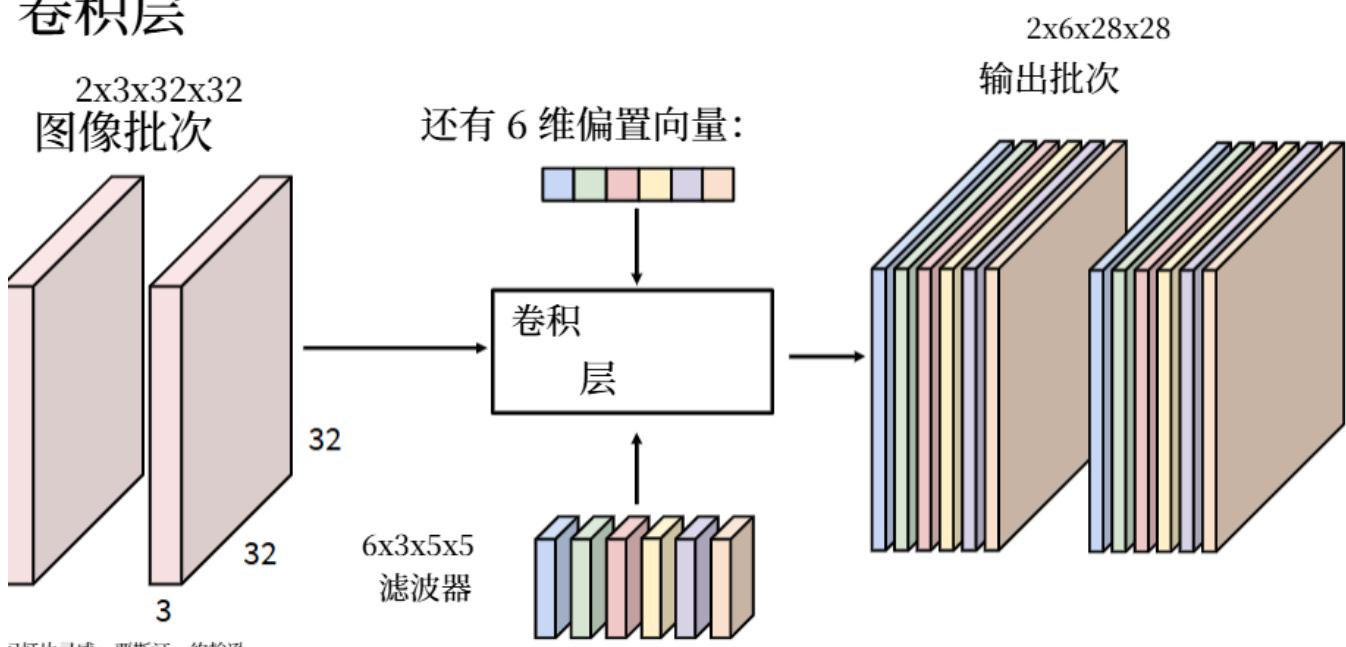
通过反向传播+梯度下降进行端到端训练

之前：卷积神经网络主导；现在：transformer主导

## 卷积层



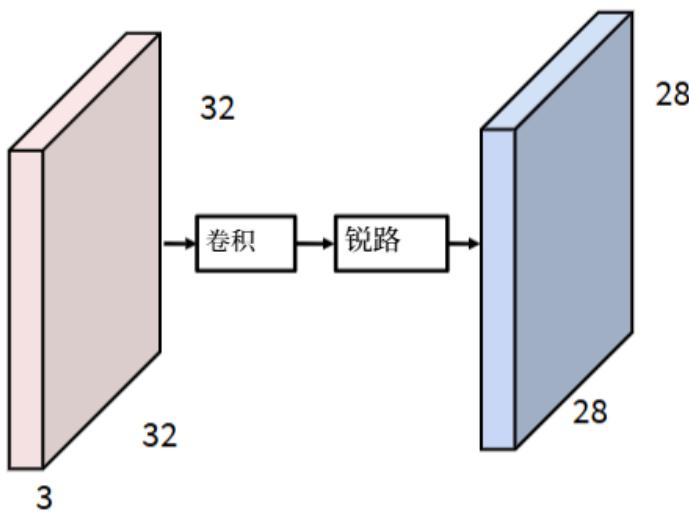
## 卷积层



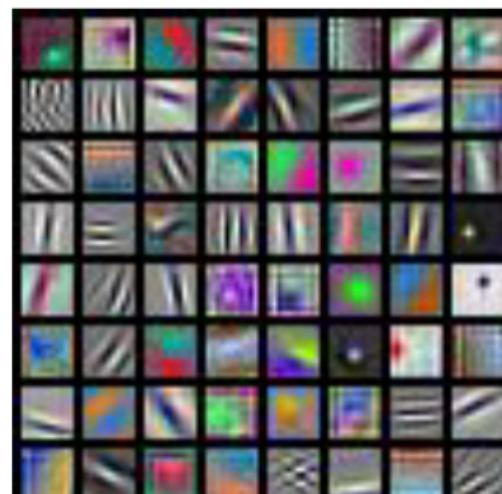
幻灯片灵感：贾斯汀·约翰逊

- 卷积神经网络是一种具有带激活函数的卷积层的神经网络！

## 卷积滤波器学到了什么？

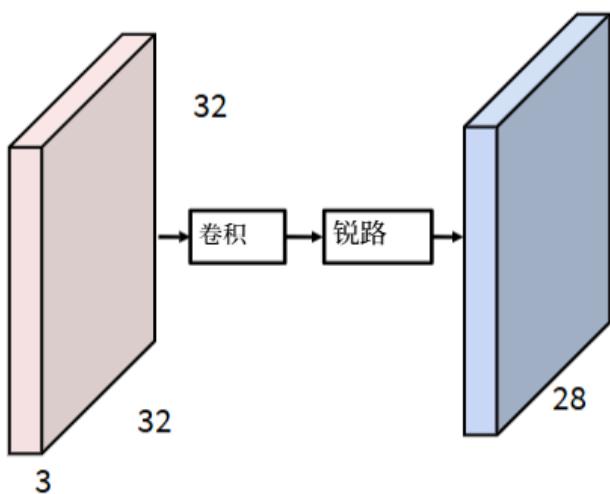


第一层卷积滤波器：局部图像模板  
(通常学习定向边缘、对比色)

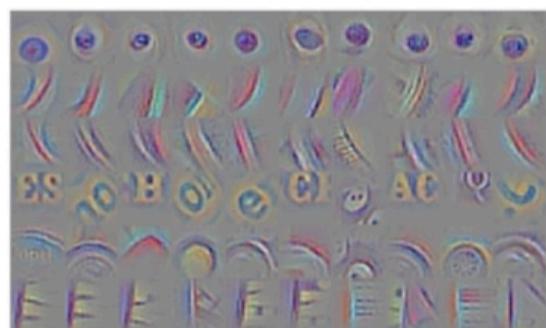


AlexNet: 64 个过滤器，每个为  $3 \times 11 \times 11$

## 卷积滤波器学到了什么？



更深的卷积层：更难可视化  
倾向于学习更大的结构，例如眼睛、字母

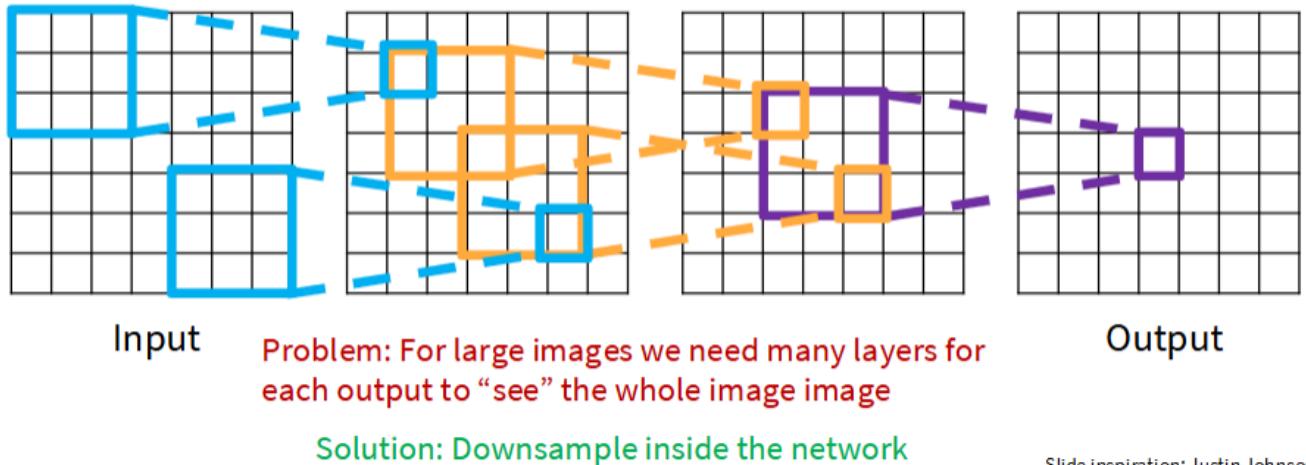


$6^{th}$  层 卷积层 来自一个 ImageNet 模型

来自 [Springenberg 等人, ICLR 2015] 的可视化

## Receptive Fields

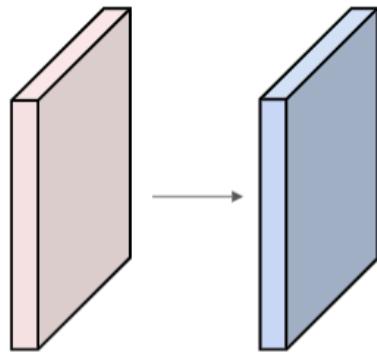
Each successive convolution adds  $K - 1$  to the receptive field size  
With  $L$  layers the receptive field size is  $1 + L * (K - 1)$



# Convolution Example

Input volume: **3 x 32 x 32**

10 **5x5** filters with stride 1, pad 2



Output volume size: **10 x 32 x 32**

Number of learnable parameters: 760

Number of multiply-add operations: **768,000**

**10\*32\*32 = 10,240 outputs**

Each output is the inner product of two **3x5x5** tensors (75 elems)

Total =  $75 \times 10240 = \mathbf{768K}$

## Convolution Summary

**Input:**  $C_{in} \times H \times W$

**Hyperparameters:**

- **Kernel size:**  $K_H \times K_W$
- **Number filters:**  $C_{out}$
- **Padding:** P
- **Stride:** S

**Weight matrix:**  $C_{out} \times C_{in} \times K_H \times K_W$   
giving  $C_{out}$  filters of size  $C_{in} \times K_H \times K_W$

**Bias vector:**  $C_{out}$

**Output size:**  $C_{out} \times H' \times W'$  where:

- $H' = (H - K + 2P) / S + 1$
- $W' = (W - K + 2P) / S + 1$

**Common settings:**

$K_H = K_W$  (Small square filters)

$P = (K - 1) / 2$  ("Same" padding)

$C_{in}, C_{out} = 32, 64, 128, 256$  (powers of 2)

$K = 3, P = 1, S = 1$  (3x3 conv)

$K = 5, P = 2, S = 1$  (5x5 conv)

$K = 1, P = 0, S = 1$  (1x1 conv)

$K = 3, P = 1, S = 2$  (Downsample by 2)

# Pooling Summary

**Input:**  $C \times H \times W$

**Hyperparameters:**

- **Kernel size:** K
- **Stride:** S
- **Pooling function:** max, avg

**Output size:**  $C \times H' \times W'$  where:

- $H' = (H - K) / S + 1$
- $W' = (W - K) / S + 1$

**No learnable parameters**

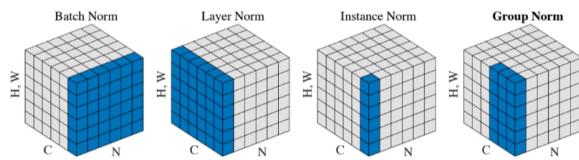
**Common setting:**

$\text{max}, K=2, S=2 \Rightarrow \text{Gives } 2x \text{ downsampling}$

- 卷积和平移操作可交换，说明图像的特征不依赖于他们在图像中的位置

## 第六讲：CNN训练与架构

### Other Normalization Layers



You will implement some of these in assignment 2!

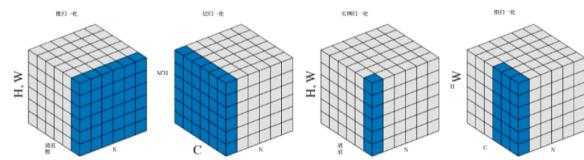
Wu and He, "Group Normalization", ECCV 2018

Stanford CS231n 10<sup>th</sup> Anniversary

Lecture 6 - 12

April 17, 2025

### 其他归一化层



你将在作业 2 中实现其中的一些！

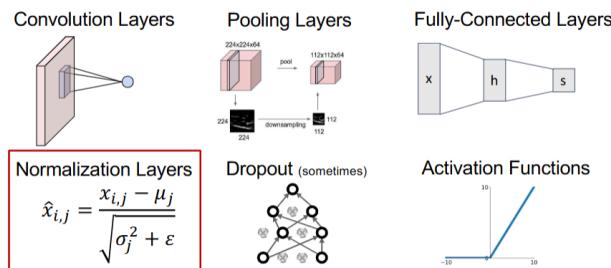
吴和何,《组归一化》, ECCV 2018

斯坦福大学 CS231n 十周年纪念

第 6 讲 - 12

2025 年 4 月 17 日

### Components of CNNs

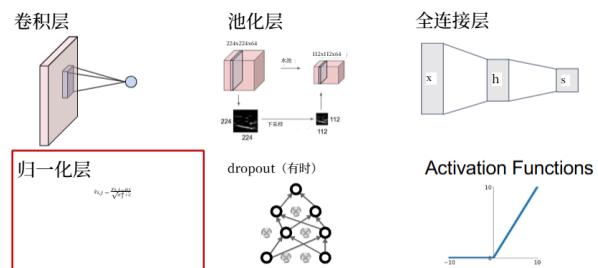


Stanford CS231n 10<sup>th</sup> Anniversary

Lecture 6 - 13

April 17, 2025

### CNN 的组成部分



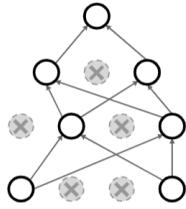
斯坦福大学 CS231n 10<sup>th</sup>周年纪念

第 6 讲 - 13

激活函数

## Regularization: Dropout

How can this possibly be a good idea?



Another interpretation:

Dropout is training a large **ensemble** of models (that share parameters).

Each binary mask is one model

An FC layer with 4096 units has  $2^{4096} \sim 10^{1233}$  possible masks!

Only  $\sim 10^{82}$  atoms in the universe...

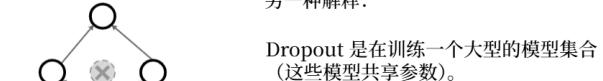
Stanford CS231n 10<sup>th</sup> Anniversary

Lecture 6 - 17 April 17, 2025

## 正则化: dropout

这怎么可能是个好主意呢?

另一种解释:



Dropout 是在训练一个大型的模型集合 (这些模型共享参数)。

每个二进制掩码都是一个模型

一个具有 4096 个单元的全连接层有  $2^{4096}$  次方到 10 的 1233 次方种可能的掩码!

宇宙中只有约  $10^{82}$  个原子……

斯坦福大学 CS231n 10<sup>th</sup>周年纪念

第 6 讲 - 17

2025 年 4 月 17 日

## Dropout: Test time

```
def predict(X):
    # ensembled forward pass
    H1 = np.maximum(0, np.dot(W1, X) + b1) * p # NOTE: scale the activations
    H2 = np.maximum(0, np.dot(W2, H1) + b2) * p # NOTE: scale the activations
    out = np.dot(W3, H2) + b3
```

At test time all neurons are active always

=> We must scale the activations so that for each neuron:  
output at test time = expected output at training time

Stanford CS231n 10<sup>th</sup> Anniversary

Lecture 6 - 18 April 17, 2025

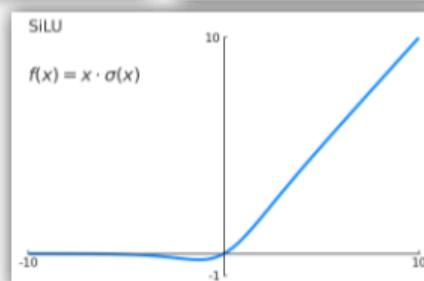
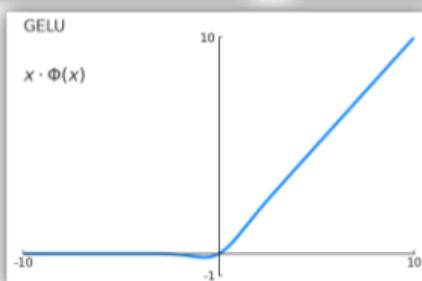
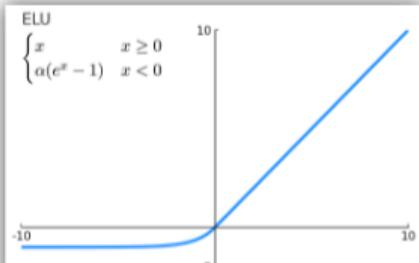
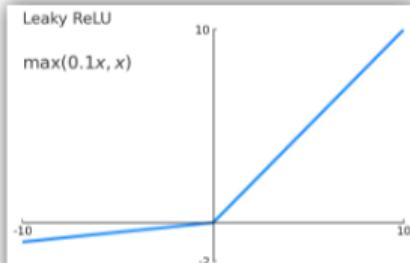
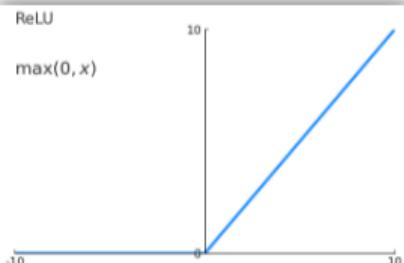
## Dropout: 测试时

```
def predict(X):
    # ensembled forward pass
    H1 = np.maximum(0, np.dot(W1, X) + b1) * p # NOTE: scale the activations
    H2 = np.maximum(0, np.dot(W2, H1) + b2) * p # NOTE: scale the activations
    out = np.dot(W3, H2) + b3
```

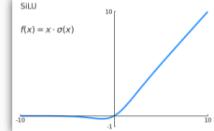
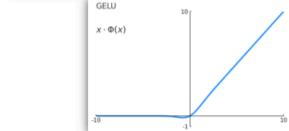
在测试时，所有神经元始终处于激活状态

=> 我们必须对激活值进行缩放，以便对于每个神经元：  
测试时的输出 = 训练时的预期输出

## Activation Function Zoo



## Activation Function Zoo



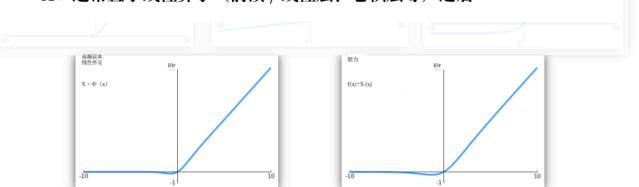
Stanford CS231n 10<sup>th</sup> Anniversary

Lecture 6 - 30 April 17, 2025

## 激活函数集合

问：激活函数在卷积神经网络中用于何处？

A: 通常置于线性算子（前馈 / 线性层、卷积层等）之后



斯坦福大学 CS231n 10<sup>th</sup>周年纪念

第 6 讲 - 30

2025 年 4 月 17 日

## Case Study: VGGNet

[Simonyan and Zisserman, 2014]

Q: Why use smaller filters? (3x3 conv)

Stack of three 3x3 conv (stride 1) layers has same effective receptive field as one 7x7 conv layer

But deeper, more non-linearities

And fewer parameters:  $3 * (3^2 C^2)$  vs.  $7^2 C^2$  for C channels per layer



Stanford CS231n 10th Anniversary

Lecture 6 - 43

April 17, 2025

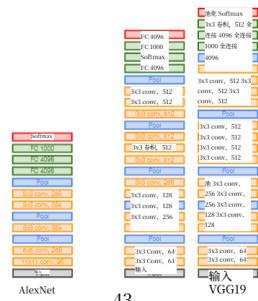
## 案例研究: VGG 网络

[西蒙尼扬和齐瑟曼, 2014]

问: 为什么使用更小的过滤器? (3x3 卷积)

三个 3x3 卷积 (步长为 1) 层的堆叠与一个 7x7 卷积层具有相同的有效感受野

但更深层次、更多的非线性以及更少的参数:  $3 * (3^2 C^2)$  对比每一层 C 个通道的  $7^2 C^2$



AlexNet

43

VGG16

2025 年 4 月 17 日

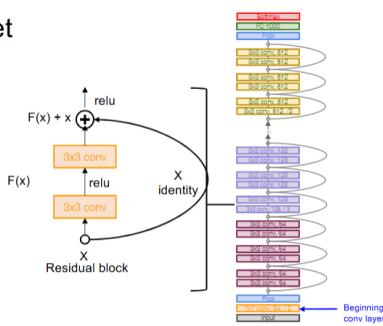
- VGGNet->小滤波器、深网络：更少参数、更多非线性
- ResNet->普通卷积神经网络上堆叠更深的层，效果不好，并非过拟合。而是更深更难优化，一种构造性解决方案是复制较浅模型中的已学习层，并将额外层设置为恒等映射。解决方案：使用网络层来拟合残差映射，而非直接尝试拟合期望的基础映射。

## Case Study: ResNet

[He et al., 2015]

Full ResNet architecture:

- Stack residual blocks
- Every residual block has two 3x3 conv layers
- Periodically, double # of filters and downsample spatially using stride 2 (/2 in each dimension)
- Additional conv layer at the beginning (stem)



Stanford CS231n 10th Anniversary

Lecture 6 - 55

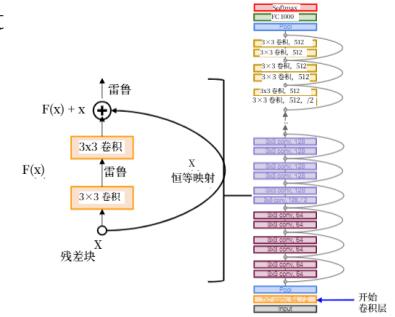
April 17, 2025

## 案例研究: ResNet

[何等人, 2015]

完整的 ResNet 架构:

- 堆叠残差块
- 每个残差块都有两个 3x3 卷积层
- 定期将滤波器数量加倍，并使用步长 2 进行空间下采样（每个维度缩小 1/2）
- 开头（主干）处额外的卷积层



Stanford CS231n 10th Anniversary

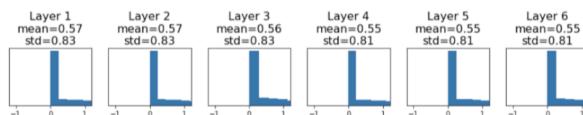
Lecture 6 - 55

April 17, 2025

斯坦福大学 CS231n 10th 周年纪念 第 6 讲 - 55 2025 年 4 月 17 日

One solution: Kaiming / MSRA Initialization

```
dims = [4096] * 7 ReLU correction: std = sqrt(2 / Din) "Just right": Activations are nicely scaled for all layers!
hs = []
x = np.random.randn(16, dims[0])
for Din, Dout in zip(dims[:-1], dims[1:]):
    W = np.random.randn(Din, Dout) * np.sqrt(2/Din)
    x = np.maximum(0, x.dot(W))
    hs.append(x)
```



He et al., “Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification”, ICCV 2015

Stanford CS231n 10th Anniversary

Lecture 6 - 65

April 17, 2025

斯坦福大学 CS231n 10th 周年纪念 第 6 讲 - 65 2025 年 4 月 17 日

## TLDR for Image Normalization: center and scale for each channel

- Subtract per-channel mean and Divide by per-channel std (almost all modern models) (stats along each channel = 3 numbers)
- Requires pre-computing means and std for each pixel channel (given your dataset)

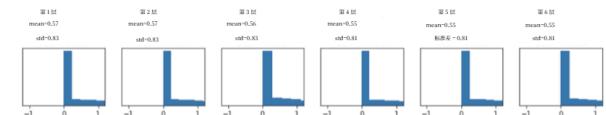
```
norm_pixel[i,j,c] = (pixel[i,j,c] - np.mean(pixel[:, :, c])) / np.std(pixel[:, :, c])
```

(正则化) 数据增强：裁剪、缩放、颜色抖动

- 迁移学习看视频
- 超参数选择（很重要）

## 一种解决方案: Kaiming / MSRA 初始化

```
dims = [4096] * 7 ReLU correction: std = sqrt(2 / Din) "恰到好处": 所有层的激活值都得到了很好的缩放!
hs = []
x = np.random.randn(16, dims[0])
for Din, Dout in zip(dims[:-1], dims[1:]):
    W = np.random.randn(Din, Dout) * np.sqrt(2/Din)
    x = np.maximum(0, x.dot(W))
    hs.append(x)
```



He 等人,《深入研究激活层: 在 ImageNet 分类上超越人类水平》, ICCV 2015

Stanford CS231n 10th Anniversary

Lecture 6 - 65

April 17, 2025

斯坦福大学 CS231n 10th 周年纪念 第 6 讲 - 65 2025 年 4 月 17 日

图像归一化的简而言之：对每个通道进行中心化和缩放

- 减去每个通道的均值并除以每个通道的标准差（几乎所有现代模型都是如此）（每个通道的统计量 = 3 个数值）
- 需要为每个像素通道预先计算均值和标准差（基于你的数据集）

```
norm_pixel[i,j,c] = (pixel[i,j,c] - np.mean(pixel[:, :, c])) / np.std(pixel[:, :, c])
```

## Choosing Hyperparameters

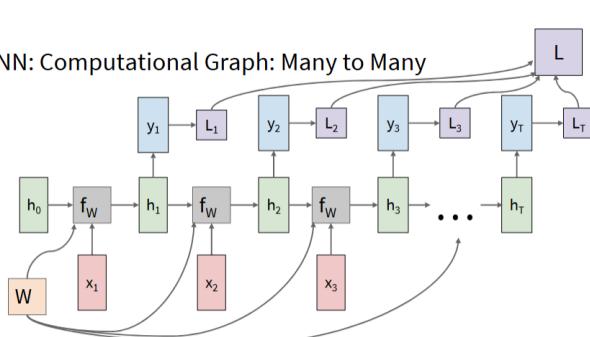
- Step 1:** Check initial loss
- Step 2:** Overfit a small sample
- Step 3:** Find LR that makes loss go down
- Step 4:** Coarse grid, train for ~1-5 epochs
- Step 5:** Refine grid, train longer
- Step 6:** Look at loss and accuracy curves
- Step 7:** GOTO step 5

## 选择超参数

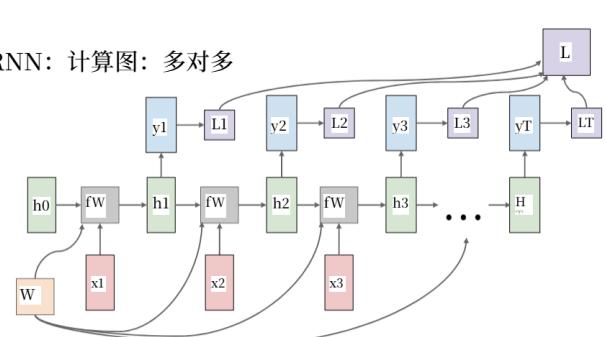
- 步骤 1: 检查初始损失
- 步骤 2: 过拟合小样本
- 步骤 3: 找到能使损失下降的学习率 (LR)
- 步骤 4: 粗略网格, 训练约 1-5 个轮次
- 第 5 步: 优化网格, 延长训练时间
- 第 6 步: 查看损失和准确率曲线
- 步骤 7: 转到步骤 5

# 第七讲：RNN

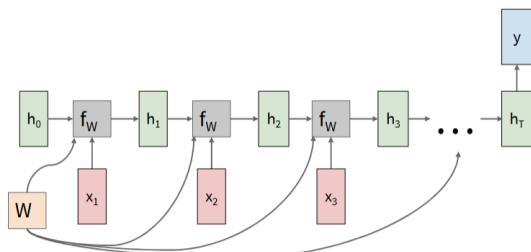
RNN: Computational Graph: Many to Many



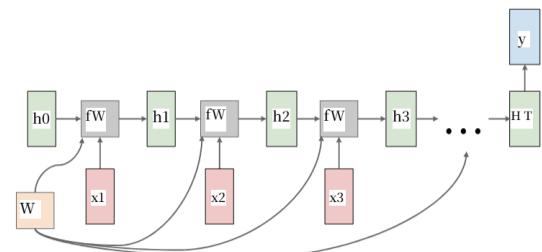
RNN: 计算图：多对多



RNN: Computational Graph: Many to One



RNN: 计算图：多对一



## RNN tradeoffs

### RNN Advantages:

- Can process any length of the input (no context length)
- Computation for step  $t$  can (in theory) use information from many steps back
- Model size does not increase for longer input
- The same weights are applied on every timestep, so there is symmetry in how inputs are processed.

### RNN Disadvantages:

- Recurrent computation is slow
- In practice, difficult to access information from many steps back

## RNN 的权衡

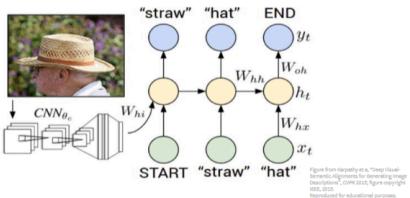
### RNN 的优势:

- 可以处理任意长度的输入 (无上下文长度限制)
- 理论上, 第  $t$  步的计算可以使用许多步骤之前的信息
- 模型大小不会因输入变长而增加
- 相同的权重应用于每个时间步, 因此在输入的处理方式上存在对称性。

### RNN 的缺点:

- 循环计算速度慢
- 实际上, 难以获取多步之前的信息

## Image Captioning



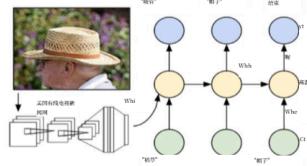
Explain Images with Multimodal Recurrent Neural Networks, Mao et al.  
Deep Visual-Semantic Alignments for Generating Image Descriptions, Karpathy and Fei-Fei  
Show and Tell: A Neural Image Caption Generator, Vinyals et al.  
Long-term Recurrent Convolutional Networks for Visual Recognition and Description, Donahue et al.  
Learning a Recurrent Visual Representation for Image Caption Generation, Chen and Zitnick

Stanford CS231n 10<sup>th</sup> Anniversary

Lecture 7 - 85

April 22, 2025

## 图像 captioning



图中来自 Karpathy 等人。用于生成图像描述的深度视觉语义对齐。

展示了与讲述：一种视觉语义对齐生成模型，Vinyals 等人。

（用于视觉识别与描述的长期循环卷积网络），多伦多等人。

学习用于图像描述生成的循环视觉表示，Chen 和 Zitnick

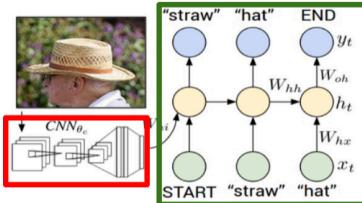
© 2013 IEEE. 保留所有权利。2013 年 4 月 22 日，为纪念斯坦福大学 CS231n 十周年。

斯坦福大学 CS231n 十周年纪念

第 7 讲 - 85

2025 年 4 月 22 日

## Recurrent Neural Network



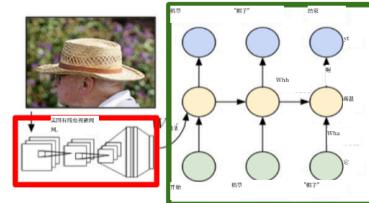
### Convolutional Neural Network

Stanford CS231n 10<sup>th</sup> Anniversary

Lecture 7 - 86

April 22, 2025

## 循环神经网络



### 卷积神经网络

斯坦福大学 CS231n 十周年纪念

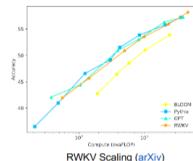
第 7 讲 - 86

2025 年 4 月 22 日

- 深度RNN是单元加深，不是时间加深

## Modern RNNs

- Sometimes called "state space models"
  - Hidden state
- Main advantages:
  - Unlimited context length
  - Compute scales linearly with sequence length



## SIMPLIFIED STATE SPACE LAYERS FOR SEQUENCE MODELING

Mamba: Linear-Time Sequence Modeling with Selective State Spaces

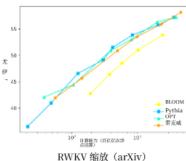
Stanford CS231n 10<sup>th</sup> Anniversary

Lecture 7 - 131

April 22, 2025

## 现代循环神经网络

- 有时被称为“状态空间模型”  
隐藏状态
- 主要优势：  
无限制的上下文长度  
计算量随序列长度呈线性增长



用于序列建模的简化状态空间层

Mamba: 基于选择性状态空间层的线性时间序列建模

斯坦福大学 CS231n 十周年纪念

第 7 讲 - 131

2025 年 4 月 22 日

## Summary

- RNNs allow a lot of flexibility in architecture design
- Vanilla RNNs are simple but don't work very well
- More complex variants (e.g. LSTMs, Mamba) can introduce ways to selectively pass information forward
- Backward flow of gradients in RNN can explode or vanish. Exploding is controlled with gradient clipping. Backpropagation through time is often needed.
- Better/simpler architectures are a hot topic of current research, as well as new paradigms for reasoning over sequences

Stanford CS231n 10<sup>th</sup> Anniversary

Lecture 7 - 132

April 22, 2025

## 摘要

- 循环神经网络 (RNNs) 在架构设计上具有很大的灵活性
- vanilla 循环神经网络虽然简单，但效果不太好
- 更复杂的变体 (如 LSTMs、Mamba) 可以引入选择性传递信息的方法
- 循环神经网络中梯度的反向传播可能会出现爆炸或消失的情况。梯度裁剪可以控制梯度爆炸。通常需要通过时间的反向传播。
- 更优 / 更简洁的架构是当前研究的热门话题，新的序列推理范式也是如此。

斯坦福大学 CS231n 十周年纪念

第 7 讲 - 132

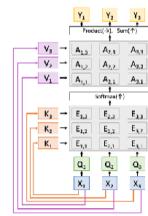
2025 年 4 月 22 日

# 第八讲：attention and transformers

- 看attention模块

## Today: Attention + Transformers

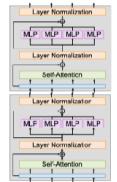
**Attention:** A new primitive that operates on sets of vectors



Transformers are used everywhere today!

But they developed as an offshoot of RNNs so let's start there

**Transformer:** A neural network architecture that uses attention everywhere



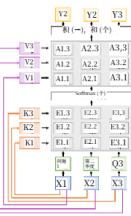
Stanford CS231n 10<sup>th</sup> Anniversary

Lecture 8 - 5

April 24, 2025

## 今天：注意力机制 + Transformer 模型

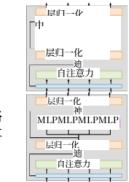
注意力：一种对向量集进行操作的新基元



如今，Transformer 模型的应用无处不在！

但它们是作为循环神经网络的分支发展而来的，所以让我们从这里开始讲起。

Transformer：一种处处使用注意力机制的神经网络架构

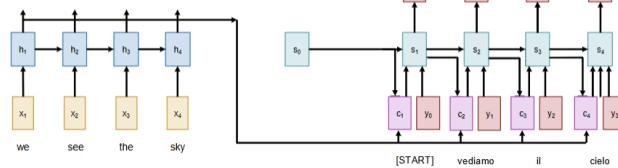


第 8 讲 - 2025 年 4 月 24 日

## Sequence to Sequence with RNNs and Attention

Use a different context vector in each timestep of decoder

- Input sequence not bottlenecked through single vector
- At each timestep of decoder, context vector "looks at" different parts of the input sequence



Bahdanau et al., "Neural machine translation by jointly learning to align and translate", ICLR 2015

Stanford CS231n 10<sup>th</sup> Anniversary

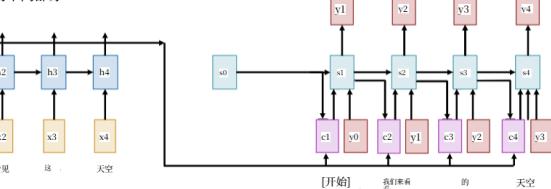
Lecture 8 - 23

April 24, 2025

## 使用循环神经网络和注意力机制的序列到序列模型

在解码器的每个时间步使用不同的上下文向量

- 输入序列不会通过单个向量形成瓶颈
- 在解码器的每个时间步，上下文向量“关注”输入序列的不同部分



Bahdanau 等人,《通过结合学习对齐和翻译的神经机器翻译》, 国际学习研讨会, 2015 年

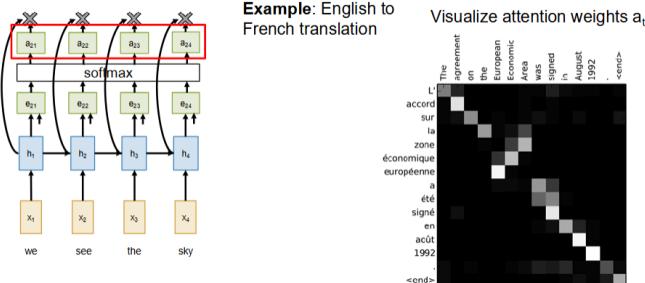
斯坦福大学 CS231n 10th 周年纪念

第 8 讲 - 23

2025 年 4 月 24 日

## Sequence to Sequence with RNNs and Attention

**Example:** English to French translation



Bahdanau et al., "Neural machine translation by jointly learning to align and translate", ICLR 2015

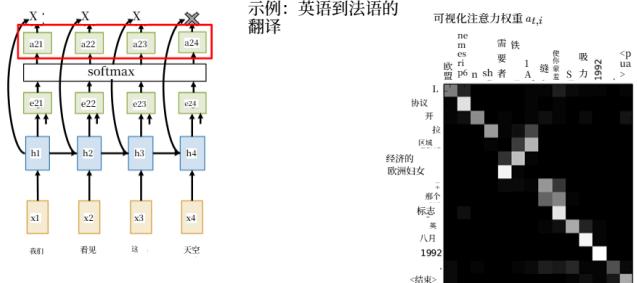
Stanford CS231n 10<sup>th</sup> Anniversary

Lecture 8 - 24

April 24, 2025

## 基于循环神经网络和注意力机制的序列到序列模型

**示例：** 英语到法语的翻译



Bahdanau 等人,《通过结合学习对齐和翻译的神经机器翻译》, 国际学习研讨会, 2015 年

斯坦福大学 CS231n 10th 周年纪念

第 8 讲 - 24

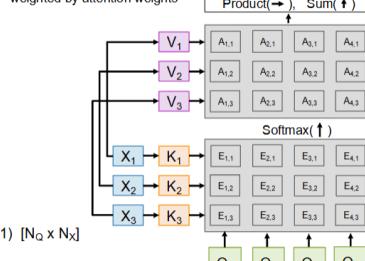
2025 年 4 月 24 日

## Attention Layer

**Inputs:**

Query vector:  $Q$  [ $N_Q \times D_Q$ ]  
Data vectors:  $X$  [ $N_X \times D_X$ ]  
Key matrix:  $W_K$  [ $D_X \times D_Q$ ]  
Value matrix:  $W_V$  [ $D_X \times D_V$ ]

Each output is a linear combination of all values, weighted by attention weights



**Computation:**

Keys:  $K = XW_K$  [ $N_X \times D_Q$ ]  
Values:  $V = XW_V$  [ $N_X \times D_V$ ]  
Similarities:  $E = QK^T / \sqrt{D_Q}$  [ $N_Q \times N_X$ ]  
 $E_{ij} = Q_i \cdot K_j / \sqrt{D_Q}$   
Attention weights:  $A = \text{softmax}(E, \text{dim}=1)$  [ $N_Q \times N_X$ ]  
Output vector:  $Y = AV$  [ $N_Q \times D_V$ ]  
 $Y = \sum_i A_{ij} V_j$

Stanford CS231n 10<sup>th</sup> Anniversary

Lecture 8 - 45

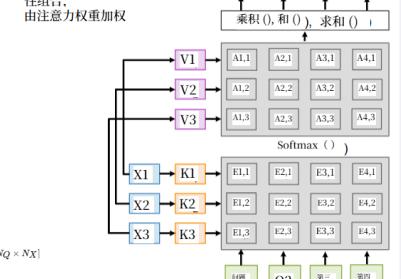
April 24, 2025

## 注意力层

输入：

查询向量:  $Q$  [ $N_Q \times D_Q$ ]  
数据向量:  $X$  [ $N_X \times D_X$ ]  
关键矩阵:  $W_K$  [ $D_X \times D_Q$ ]  
值矩阵:  $W_V$  [ $D_X \times D_V$ ]

每个输出都是所有值的线性组合，由注意力权重加权



**计算:**

Keys:  $K = XW_K$  [ $N_X \times D_Q$ ]  
Values:  $V = XW_V$  [ $N_X \times D_V$ ]  
 $E_{ij} = \frac{Q_i \cdot K_j}{\sqrt{D_Q}}$  [ $N_Q \times N_X$ ]  
Attention weights:  $A = \text{softmax}(E, \text{dim}=1)$  [ $N_Q \times N_X$ ]  
输出:  $Y = AV$  [ $N_Q \times D_V$ ]

斯坦福大学 CS231n 10th 周年纪念

第 8 讲 - 45

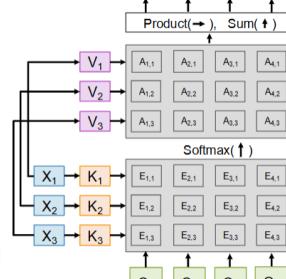
2025 年 4 月 24 日

## Cross-Attention Layer

**Inputs:**

Query vector:  $Q$  [ $N_Q \times D_Q$ ]  
Data vectors:  $X$  [ $N_X \times D_X$ ]  
Key matrix:  $W_K$  [ $D_X \times D_Q$ ]  
Value matrix:  $W_V$  [ $D_X \times D_V$ ]

Each query produces one output, which is a mix of information in the data vectors



**Computation:**

Keys:  $K = XW_K$  [ $N_X \times D_Q$ ]  
Values:  $V = XW_V$  [ $N_X \times D_V$ ]  
Similarities:  $E = QK^T / \sqrt{D_Q}$  [ $N_Q \times N_X$ ]  
 $E_{ij} = Q_i \cdot K_j / \sqrt{D_Q}$   
Attention weights:  $A = \text{softmax}(E, \text{dim}=1)$  [ $N_Q \times N_X$ ]  
Output vector:  $Y = AV$  [ $N_Q \times D_V$ ]  
 $Y = \sum_i A_{ij} V_j$

Stanford CS231n 10<sup>th</sup> Anniversary

Lecture 8 - 46

April 24, 2025

## 交叉注意力层

输入：

查询向量:  $Q$  [ $N_Q \times D_Q$ ]  
数据向量:  $X$  [ $N_X \times D_X$ ]  
关键矩阵:  $W_K$  [ $D_X \times D_Q$ ]  
值矩阵:  $W_V$  [ $D_X \times D_V$ ]

每个查询生成一个输出，该输出是数据向量中信息的混合体。

计算:  
Keys:  $K = XW_K$  [ $N_X \times D_Q$ ]  
Values:  $V = XW_V$  [ $N_X \times D_V$ ]  
 $E_{ij} = \frac{Q_i \cdot K_j}{\sqrt{D_Q}}$  [ $N_Q \times N_X$ ]  
Attention weights:  $A = \text{softmax}(E, \text{dim}=1)$  [ $N_Q \times N_X$ ]  
输出:  $Y = AV$  [ $N_Q \times D_V$ ]

斯坦福大学 CS231n 10th 周年纪念

第 8 讲 - 46

2025 年 4 月 24 日

## Self-Attention Layer

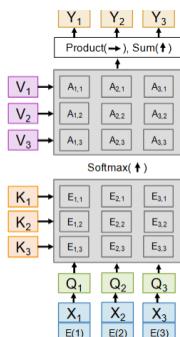
**Inputs:**  
Input vectors:  $X [N \times D_{in}]$   
Key matrix:  $W_K [D_{in} \times D_{out}]$   
Value matrix:  $W_V [D_{in} \times D_{out}]$   
Query matrix:  $W_Q [D_{in} \times D_{out}]$

**Computation:**  
Queries:  $Q = XW_Q [N \times D_{out}]$   
Keys:  $K = XW_K [N \times D_{out}]$   
Values:  $V = XW_V [N \times D_{out}]$   
Similarities:  $E = QK^\top / \sqrt{D_Q} [N \times N]$   
 $E_{ij} = Q_i K_j / \sqrt{D_Q}$

Attention weights:  $A = \text{softmax}(E, \text{dim}=1) [N \times N]$   
Output vector:  $Y = AV [N \times D_{out}]$   
 $Y = \sum_j A_{ij} V_j$

**Problem:** Self-Attention does not know the order of the sequence

**Solution:** Add positional encoding to each input; this is a vector that is a fixed function of the index



Stanford CS231n 10<sup>th</sup> Anniversary

Lecture 8 - 59

April 24, 2025

## 自注意力层

**输入:**  
输入向量:  $X [N \times D_{in}]$   
关键矩阵:  $W_K [D_{in} \times D_{out}]$   
值矩阵:  $W_V [D_{in} \times D_{out}]$   
查询矩阵:  $W_Q [D_{in} \times D_{out}]$

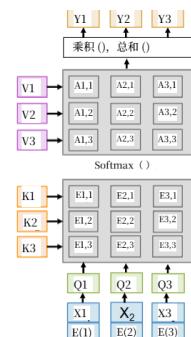
**解决方案:** 添加位置对每个输入进行编码; 这是一个固定的向量索引的函数

$Keys : K = XW_K [N \times D_{out}]$   
 $Values : V = XW_V [N \times D_{out}]$   
 $Similarities : E = QK^\top / \sqrt{D_Q} [N \times N]$   
 $E_{ij} = Q_i K_j / \sqrt{D_Q}$

$Attentionweights : A = \text{softmax}(E, \text{dim}=1) [N \times N]$

输出:  $Y = AV [N \times D_{out}]$

$Y = \sum_j A_{ij} V_j$



斯坦福大学 CS231n 10<sup>th</sup> 周年纪念讲座 8 -

59

2025 年 4 月 24 日

## Masked Self-Attention Layer

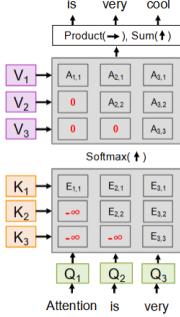
Don't let vectors "look ahead" in the sequence

**Inputs:**  
Input vectors:  $X [N \times D_{in}]$   
Key matrix:  $W_K [D_{in} \times D_{out}]$   
Value matrix:  $W_V [D_{in} \times D_{out}]$   
Query matrix:  $W_Q [D_{in} \times D_{out}]$

Override similarities with -inf; this controls which inputs each vector is allowed to look at.

**Computation:**  
Queries:  $Q = XW_Q [N \times D_{out}]$   
Keys:  $K = XW_K [N \times D_{out}]$   
Values:  $V = XW_V [N \times D_{out}]$   
Similarities:  $E = QK^\top / \sqrt{D_Q} [N \times N]$   
 $E_{ij} = Q_i K_j / \sqrt{D_Q}$

Used for language modeling where you want to predict the next word



Stanford CS231n 10<sup>th</sup> Anniversary

Lecture 8 - 61

April 24, 2025

## 掩码自注意力层

不要让向量在序列中“向前看”

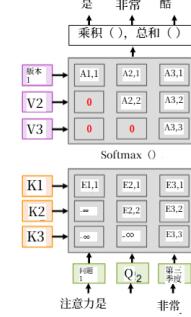
**输入:**  
输入向量:  $X [N \times D_{in}]$   
关键矩阵:  $W_K [D_{in} \times D_{out}]$   
值矩阵:  $W_V [D_{in} \times D_{out}]$   
查询矩阵:  $W_Q [D_{in} \times D_{out}]$

用负无穷覆盖相似度; 这控制着每个向量被允许查看哪些输入。

**计算:**

$Queries : Q = XW_Q [N \times D_{out}]$   
 $Keys : K = XW_K [N \times D_{out}]$   
 $Values : V = XW_V [N \times D_{out}]$   
 $Similarities : E = QK^\top / \sqrt{D_Q} [N \times N]$   
 $Attentionweights : A = \text{softmax}(E, \text{dim}=1) [N \times N]$   
输出:  $Y = AV [N \times D_{out}]$

$Y = \sum_j A_{ij} V_j$



斯坦福大学 CS231n 10<sup>th</sup>周年纪念

第 8 讲 - 61

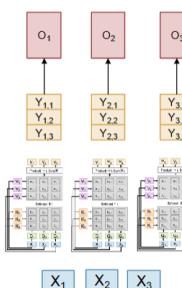
2025 年 4 月 24 日

## Multiheaded Self-Attention Layer

Run H copies of Self-Attention in parallel

**Inputs:**  
Input vectors:  $X [N \times D]$   
Key matrix:  $W_K [D \times HD_H]$   
Value matrix:  $W_V [D \times HD_H]$   
Query matrix:  $W_Q [D \times HD_H]$   
Output matrix:  $W_O [HD_H \times D]$

In practice, compute all H heads in parallel using batched matrix multiply operations.



**Computation:**  
Queries:  $Q = XW_Q [H \times N \times D_H]$   
Keys:  $K = XW_K [H \times N \times D_H]$   
Values:  $V = XW_V [H \times N \times D_H]$   
Similarities:  $E = QK^\top / \sqrt{D_H} [H \times N \times N]$   
Attention weights:  $A = \text{softmax}(E, \text{dim}=2) [H \times N \times N]$   
Head outputs:  $Y = AV [H \times N \times D_H] \Rightarrow [N \times HD_H]$   
Outputs:  $O = YW_O [N \times D]$

Used everywhere in practice.

Stanford CS231n 10<sup>th</sup> Anniversary

Lecture 8 - 67

April 24, 2025

## 多头自注意力层

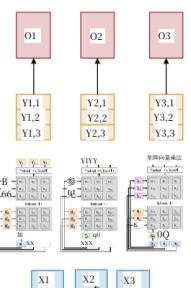
并行运行 H 个自注意力副本

**输入:**  
输入向量:  $X [N \times D]$   
键矩阵:  $W_K [D \times HD_H]$   
值矩阵:  $W_V [D \times HD_H]$   
查询矩阵:  $W_Q [D \times HD_H]$   
输出矩阵:  $W_O [HD_H \times D]$

在实际应用中，并行计算所有 H 个注意力头，使用批处理矩阵乘法运算。

**计算:**  
Queries:  $Q = XW_Q [H \times N \times D_H]$   
Keys:  $K = XW_K [H \times N \times D_H]$   
Values:  $V = XW_V [H \times N \times D_H]$   
Similarities:  $E = QK^\top / \sqrt{D_H} [H \times N \times N]$   
Attentionweights:  $A = \text{softmax}(E, \text{dim}=2) [H \times N \times N]$   
Headoutputs:  $Y = AV [H \times N \times D_H] \Rightarrow [N \times HD_H]$   
Outputs:  $O = YW_O [N \times D]$

在实践中被广泛使用。



斯坦福大学 CS231n 10<sup>th</sup>周年纪念

67

2025 年 4 月 24 日

## Self-Attention is Four Matrix Multiplications!

**Flash Attention**  
algorithm computes 2x3 at the same time without storing the full attention weights

1. QKV Projection  
 $[N \times D] \rightarrow [N \times 3HD_H] \Rightarrow [N \times 3HD_H]$   
Split and reshape to get  $Q, K, V$  each of shape  $[H \times N \times D_H]$
2. QK Similarity  
 $[H \times N \times D_H] [H \times D_H \times N] \Rightarrow [H \times N \times N]$
3. V-Weighting  
 $[H \times N \times N] [H \times N \times D_H] \Rightarrow [H \times N \times D_H]$   
Reshape to  $[N \times HD_H]$
4. Output Projection  
 $[N \times HD_H] [HD_H \times D] \Rightarrow [N \times D]$

If  $N=100K, H=64$  then  $H \times N \times N$  attention weights take 1.192 TB! GPUs don't have that much memory...

Q: How much memory does this take as the number of vectors N increases?

A:  $O(N^3)$  with Flash Attention

Attention weights:  $A = \text{softmax}(E, \text{dim}=2) [H \times N \times N]$

Head outputs:  $Y = AV [H \times N \times D_H] \Rightarrow [N \times HD_H]$

Outputs:  $O = YW_O [N \times D]$

Stanford CS231n 10<sup>th</sup> Anniversary

Lecture 8 - 78

April 24, 2025

## 自注意力就是四次矩阵乘法!

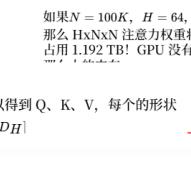
**Flash Attention** 算法在存储完整注意矩阵的情况下同时计算 2x3!

使大 N 成为可能

**计算:**  
Queries:  $Q = XW_Q [H \times N \times D_H]$   
Keys:  $K = XW_K [H \times N \times D_H]$   
Values:  $V = XW_V [H \times N \times D_H]$   
Similarities:  $E = QK^\top / \sqrt{D_H} [H \times N \times N]$   
Attentionweights:  $A = \text{softmax}(E, \text{dim}=2) [H \times N \times N]$   
Headoutputs:  $Y = AV [H \times N \times D_H] \Rightarrow [N \times HD_H]$   
Outputs:  $O = YW_O [N \times D]$

随着向量数量 N 的增加，这需要多少内存？

A: 使用 Flash Attention 时为 O(N)



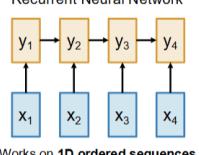
斯坦福大学 CS231n 10<sup>th</sup>周年纪念

第 8 讲 - 78

2025 年 4 月 24 日

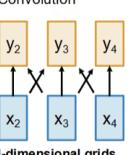
## Three Ways of Processing Sequences

Recurrent Neural Network



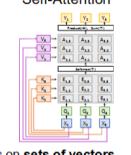
(+) Theoretically good at long sequences:  $O(N)$  compute and memory for a sequence of length N  
(-) Not parallelizable. Need to compute hidden states sequentially

Convolution



(-) Bad for long sequences: need to stack many layers to build up large receptive fields  
(+) Parallelizable, outputs can be computed in parallel

Self-Attention



(+) Great for long sequences; each output depends directly on all inputs  
(+) Highly parallel, it's just 4 matmuls  
(-) Expensive:  $O(N^2)$  compute,  $O(N)$  memory for sequence of length N

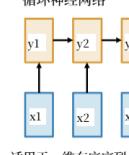
Stanford CS231n 10<sup>th</sup> Anniversary

Lecture 8 - 82

April 24, 2025

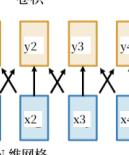
## 序列处理的三种方式

循环神经网络



适用于 1 维有序序列 (+) 理论上慢  
长序列: 对于长度为 N 的序列, 计算和内存复杂度为  $O(N)$  (-) 不可并行化。需要按顺序计算隐藏状态

卷积



(-) 对长序列不好: 需要堆叠许多层才能构建大的感受野  
(+) 可并行化, 输出可并行计算

自注意力



适用于向量集 (+) 非常适合长序列; 每个输出直接依赖于所有输入 (+) 高度并行, 仅需 4 次矩阵乘法 (-) 成本高:  $O(N^2)$  计算, 长度为 N 的序列需要  $O(N)$  的内存

• transformers

斯坦福大学 CS231n 10<sup>th</sup>周年纪念

第 8 讲 - 82

2025 年 4 月 24 日

## The Transformer

### Transformer Block

**Input:** Set of vectors x  
**Output:** Set of vectors y

Self-Attention is the only interaction between vectors  
LayerNorm and MLP work on each vector independently

Highly scalable and parallelizable, most of the compute is just 6 matmuls:  
4 from Self-Attention  
2 from MLP

Vaswani et al. "Attention is all you need." NeurIPS 2017

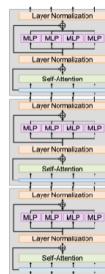
A **Transformer** is just a stack of identical Transformer blocks!

They have not changed much since 2017... but have gotten a lot bigger

Original: [Vaswani et al, 2017]  
12 blocks,  $D=1024$ ,  $H=16$ ,  $N=512$   
213M params

GPT-2: [Radford et al, 2019]  
48 blocks,  $D=1600$ ,  $H=25$ ,  $N=1024$   
1.5B params

GPT-3: [Brown et al, 2020]  
96 blocks,  $D=12288$ ,  $H=96$ ,  $N=2048$   
175B params



## Transformer 模型

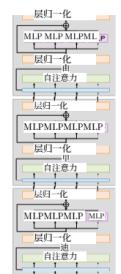
### Transformer 块

输入: 向量集 x  
输出: 向量集 y

自注意力是向量之间唯一的交互方式  
LayerNorm 和 MLP 独立作用于每个向量

GPT-2: [雷德福德等人, 2019] 48 块,  $D = 1600$ ,  $H = 25$ ,  $N = 1024$  15 亿参数

GPT-3: [Brown et al, 2020] 96 块,  $D = 12288$ ,  $H = 96$ ,  $N = 2048$  175B 参数



Stanford CS231n 10<sup>th</sup> Anniversary

Lecture 8 - 95

April 24, 2025

斯坦福大学 CS231n 十周年纪念

第 8 讲 - 95

2025 年 4 月 24 日

## Transformers for Language Modeling (LLM)

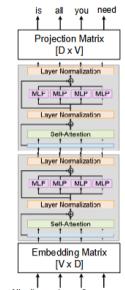
Learn an embedding matrix at the start of the model to convert words into vectors.

Given vocab size V and model dimension D, it's a lookup table of shape  $[V \times D]$

Use masked attention inside each transformer block so each token can only see the ones before it

At the end, learn a projection matrix of shape  $[D \times V]$  to project each D-dim vector to a V-dim vector of scores for each element of the vocabulary.

Train to predict next token using softmax + cross-entropy loss



Stanford CS231n 10<sup>th</sup> Anniversary

Lecture 8 - 99

April 24, 2025

## 用于语言建模的 Transformer (大语言模型)

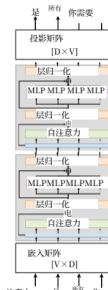
在模型开始时学习一个嵌入矩阵，将单词转换为向量。

给定词汇量 V 和模型维度 D, 它是一个形状为  $[V \times D]$  的查找表。

在每个 Transformer 块中使用掩码注意力，这样每个标记只能看到它前面的标记。

最后，学习一个形状为  $[D \times V]$  的投影矩阵，将每个 D 维向量投影到一个 V 维的词汇表中每个元素的得分向量上。

使用 softmax + 交叉熵损失来训练预测下一个 token

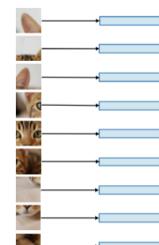
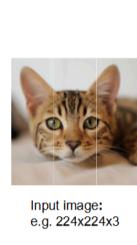


斯坦福大学 CS231n 十周年纪念

第 8 讲 - 99

2025 年 4 月 24 日

## Vision Transformers (ViT)



Input image:  
e.g. 224x224x3

Break into patches  
e.g. 16x16x3

Q: Any other way to describe this operation?

A: 16x16 conv with stride 16, 3 input channels, D output channels

Lecture 8 - 104

April 24, 2025

## 视觉 Transformer (ViT)



问: 还有其他方式描述这个操作吗?

A: 16x16 卷积, 步长 16, 3 个输入通道, D 个输出通道

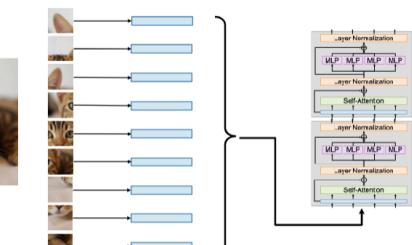
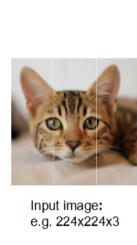
输入图像示例:  
224x224x3

斯坦福大学 CS231n 十周年纪念

第 8 讲 - 104

2025 年 4 月 24 日

## Vision Transformers (ViT)



Input image:  
e.g. 224x224x3

Break into patches  
e.g. 16x16x3

Average pool NxD vectors to 1xD, apply a linear layer  $D \Rightarrow C$  to predict class scores

Transformer gives an output vector per patch

Don't use any masking; each image patch can look at all other image patches

Use positional encoding to tell the transformer the 2D position of each patch

D-dim vector per patch are the input vectors to the Transformer

Lecture 8 - 105

April 24, 2025

## 视觉 Transformer (ViT)



分割成补丁, 例如 16x16x3

展平并应用  $768 \Rightarrow D$  的线性变换

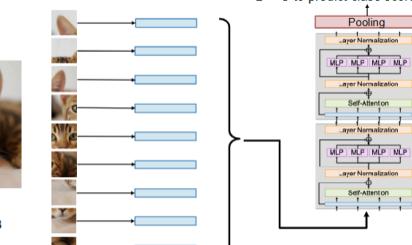
每个补丁的  $D$  维向量是 Transformer 的输出向量

斯坦福大学 CS231n 十周年纪念

第 8 讲 - 105

2025 年 4 月 24 日

## Vision Transformers (ViT)



Input image:  
e.g. 224x224x3

Break into patches  
e.g. 16x16x3

Average pool NxD vectors to 1xD, apply a linear layer  $D \Rightarrow C$  to predict class scores

Transformer gives an output vector per patch

Don't use any masking; each image patch can look at all other image patches

Use positional encoding to tell the transformer the 2D position of each patch

D-dim vector per patch are the input vectors to the Transformer

Lecture 8 - 109

April 24, 2025

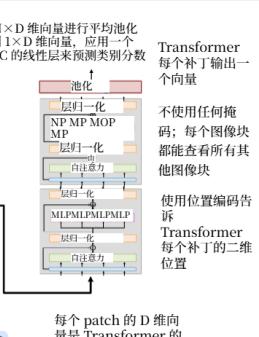
## 视觉 Transformer (ViT)



斯坦福大学 CS231n 十周年纪念

第 8 讲 - 109

2025 年 4 月 24 日



## Pre-Norm Transformer

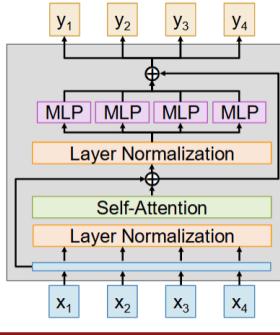
Layer normalization is outside the residual connections

Kind of weird, the model can't actually learn the identity function

**Solution:** Move layer normalization before the Self-Attention and MLP, inside the residual connections. Training is more stable.

Baevski & Auli, "Adaptive Input Representations for Neural Language Modeling", arXiv 2018

Stanford CS231n 10<sup>th</sup> Anniversary



Lecture 8 - 112 April 24, 2025

## 前置归一化 Transformer

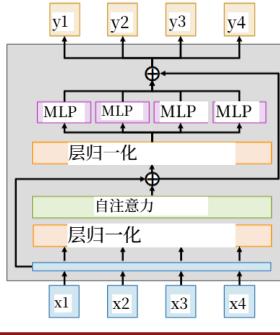
层归一化位于残差连接之外

有点奇怪，这个模型实际上无法学习恒等函数

解决方案：将层归一化移至自注意力和多层感知器之前，置于残差连接内部。训练会更稳定。

巴歌斯斯基和奥利·“用于神经语言建模的适应输入表示”，arXiv 2018

斯坦福大学 CS231n 十周年纪念



第 8 讲 - 112

2025 年 4 月 24 日

## RMSNorm

Replace Layer Normalization with Root-Mean-Square Normalization (RMSNorm)

**Input:**  $x$  [shape D]  
**Output:**  $y$  [shape D]  
**Weight:**  $\gamma$  [shape D]

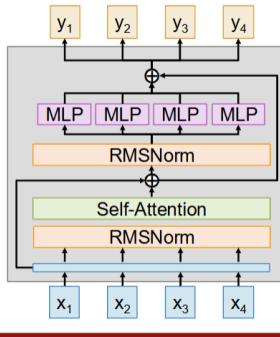
$$y_i = \frac{x_i}{\text{RMS}(x)} * \gamma_i$$

$$\text{RMS}(x) = \sqrt{\varepsilon + \frac{1}{N} \sum_{i=1}^N x_i^2}$$

Training is a bit more stable

Zhang and Sennrich, "Root Mean Square Layer Normalization", NeurIPS 2019

Stanford CS231n 10<sup>th</sup> Anniversary



Lecture 8 - 113 April 24, 2025

## RMSNorm

用均方根归一化  
(RMSNorm) 替代层归一化

**输入:**  $x$  [形状 D]  
**输出:**  $y$  [形状 D]  
**权重:**  $\gamma$  [形状 D]

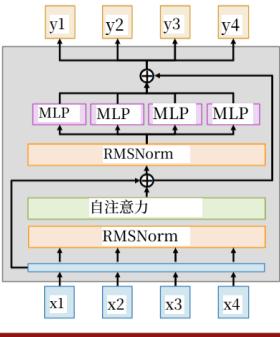
$$y_i = \frac{x_i}{\text{RMS}(x)} * \gamma_i$$

$$\text{RMS}(x) = \sqrt{\varepsilon + \frac{1}{N} \sum_{i=1}^N x_i^2}$$

训练稍微稳定一些

李和孙里奇,《均方根归一化》,神经信息处理系统大会 2019

斯坦福大学 CS231n 十周年纪念



第 8 讲 - 113

2025 年 4 月 24 日

## SwiGLU MLP

### Classic MLP:

**Input:**  $X$  [N x D]  
**Weights:**  $W_1$  [D x 4D]  
 $W_2$  [4D x D]  
**Output:**  $Y = \sigma(XW_1)W_2$  [N x D]

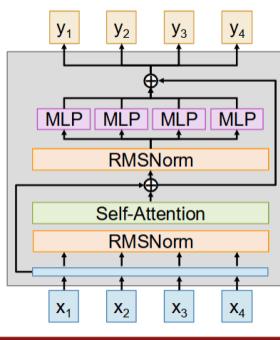
### SwiGLU MLP:

**Input:**  $X$  [N x D]  
**Weights:**  $W_1$ ,  $W_2$  [D x H]  
 $W_3$  [H x D]  
**Output:**  $Y = (\sigma(XW_1) \otimes XW_2)W_3$

Setting H = 8D/3 keeps same total params

Shazeer, "GLU Variants Improve Transformers", 2020

Stanford CS231n 10<sup>th</sup> Anniversary



Lecture 8 - 116 April 24, 2025

## SwiGLU 多层感知机

经典多层感知机:

**输入:**  $X$  [N x D]  
**权重:**  $W_1$  [D x 4D]

$$\text{Output: } Y = \sigma(XW_1)W_2[N \times D]$$

SwiGLU 多层感知机:

**输入:**  $X$  [N x D]  
**权重:**  $W_1$ ,  $W_2$  [D x 高度 × 宽度]

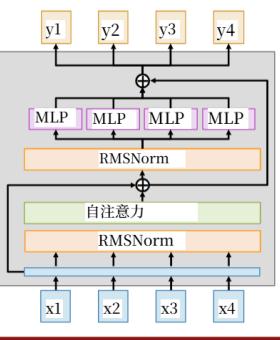
**输出:**

Setting H = 8D/3 保持

我们不会解释为什么这些架构似乎能奏效；我们将它们的成功，以及其他一切，都归功于神的仁慈。

沙泽尔,《GLU 变体改进 Transformer》, 2020 年

斯坦福大学 CS231n 10th 周年纪念



第 8 讲 - 116

2025 年 4 月 24 日

## Mixture of Experts (MoE)

Learn E separate sets of MLP weights in each block; each MLP is an *expert*

$$W_1 : [D \times 4D] \Rightarrow [E \times D \times 4D]$$

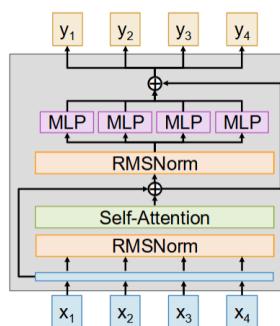
$$W_2 : [4D \times D] \Rightarrow [E \times 4D \times D]$$

Each token gets *routed* to A < E of the experts. These are the *active experts*.

Increases params by E,  
But only increases compute by A

All of the biggest LLMs today (e.g., GPT4o, GPT4.5, Claude 3.7, Gemini 2.5 Pro, etc.) almost certainly use MoE and have > 1T params; but they don't publish details anymore

Stanford CS231n 10<sup>th</sup> Anniversary



Lecture 8 - 119 April 24, 2025

## 专家混合模型 (MoE)

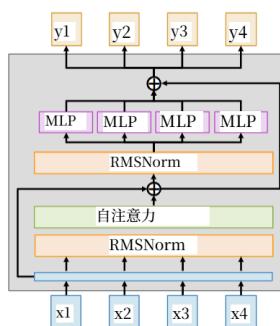
在每个块中学习 E 组独立的 MLP 权重;  
每个 MLP 都是一个专家,  $W_1 : [D \times 4D]$   
 $\Rightarrow [E \times D \times 4D]$ ,  $W_2 : [4D \times D] \Rightarrow [E \times 4D \times D]$

每个令牌都被路由到专家的  $A < E$ 。这是活跃的专家。

参数增加了 E 倍。  
但计算量增加 A

如今所有最大的语言模型（例如 GPT4o, GPT4.5, Claude 3.7, Gemini 2.5 Pro 等）几乎肯定都使用了混合专家模型 (MoE) 和  $have > 1T$  参数，但它们不再公布细节了。

斯坦福大学 CS231n 10th 周年纪念



第 8 讲 - 119

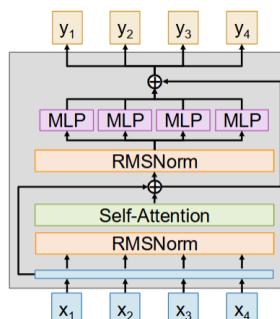
2025 年 4 月 24 日

## Tweaking Transformers

The Transformer architecture has not changed much since 2017.

- **Pre-Norm:** Move normalization inside residual
- **RMSNorm:** Different normalization layer
- **SwiGLU:** Different MLP architecture
- **Mixture of Experts (MoE):** Learn E different MLPs, use A < E of them per token. Massively increase params, modest increase to compute cost.

Stanford CS231n 10<sup>th</sup> Anniversary



Lecture 8 - 120 April 24, 2025

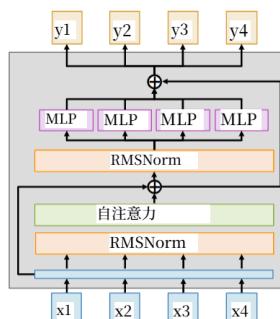
## 调整 Transformer 模型

自 2017 年以来, Transformer 架构没有太大变化。

- 有一些变化已变得普遍:
  - 前置归一化：将归一化移入残差连接中

RMSNorm: 不同的归一化层 -  
SwiGLU: 不同的 MLP 架构  
- 专家混合模型 (MoE): 学习 E 个不同的多层感知器，每个标记使用 A < E 个。大幅增加参数，计算成本适度增加。

斯坦福大学 CS231n 10th 周年纪念



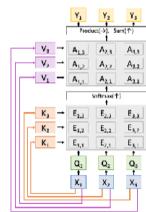
第 8 讲 - 120

2025 年 4 月 24 日

# summary

## Summary: Attention + Transformers

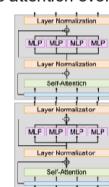
**Attention:** A new primitive that operates on sets of vectors



Transformers are the backbone of all large AI models today!

Used for language, vision, speech, ...

**Transformer:** A neural network architecture that uses attention everywhere



Stanford CS231n 10<sup>th</sup> Anniversary

Lecture 8 - 121

April 24, 2025

## 摘要：注意力机制 + Transformer 模型

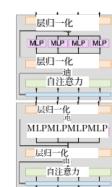
注意力：一种对向量集进行操作的新基元



Transformer 是当今所有大型 AI 模型的支柱！

用于语言、视觉、语音等领域……

**Transformer:** 一种在各处都使用注意力机制的神经网络架构



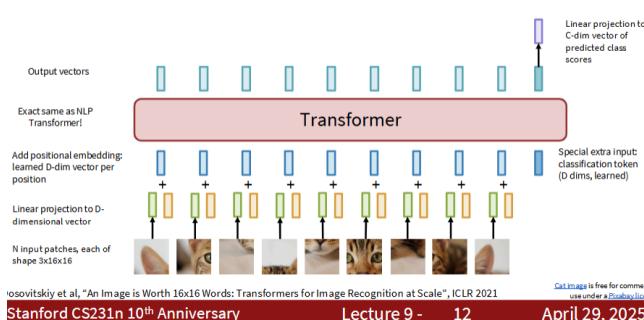
第 8 讲 - 121

2025 年 4 月 24 日

# 第九讲：检测、分割、可视化与理解

## 分割

### Vision Transformers (ViT)

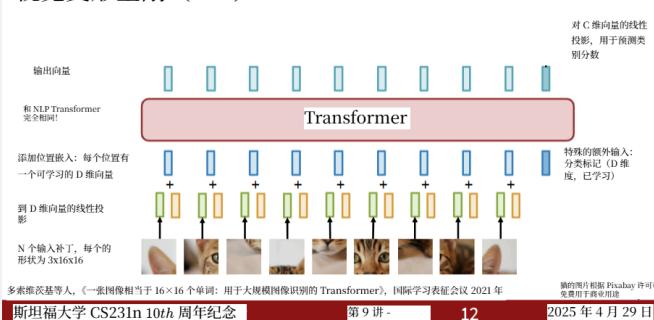


Stanford CS231n 10<sup>th</sup> Anniversary

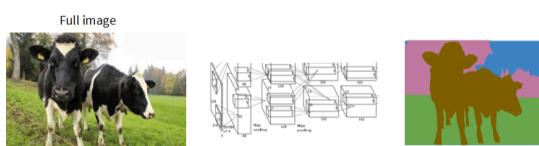
Lecture 9 - 12

April 29, 2025

### 视觉变形金刚 (ViT)



### Semantic Segmentation Idea: Convolution



An intuitive idea: encode the entire image with conv net, and do semantic segmentation on top.

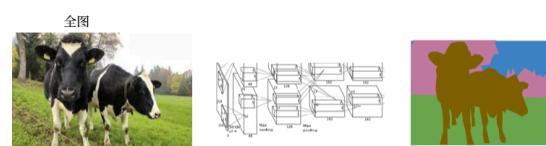
Problem: classification architectures often reduce feature spatial sizes to go deeper, but semantic segmentation requires the output size to be the same as input size.

Stanford CS231n 10<sup>th</sup> Anniversary

Lecture 9 - 40

April 29, 2025

### 语义分割思路：卷积



一个直观的想法是: 用卷积网络对整个图像进行编码, 然后在此基础上进行语义分割。

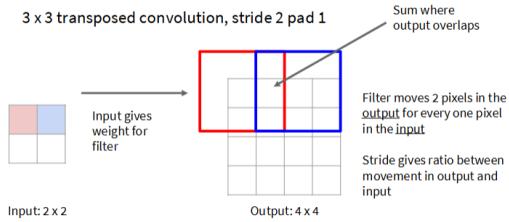
问题: 分类架构通常会缩小特征的空间尺寸以实现更深层次的网络, 但语义分割需要输出尺寸与输入尺寸相同。

斯坦福大学 CS231n 10<sup>th</sup> 周年纪念

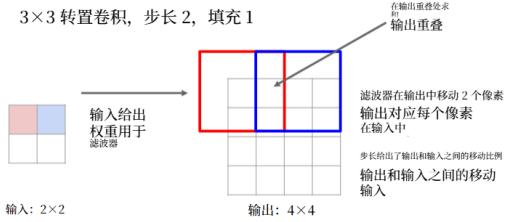
第 9 讲 - 40

2025 年 4 月 29 日

## Learnable Upsampling: Transposed Convolution



## 可学习上采样：转置卷积



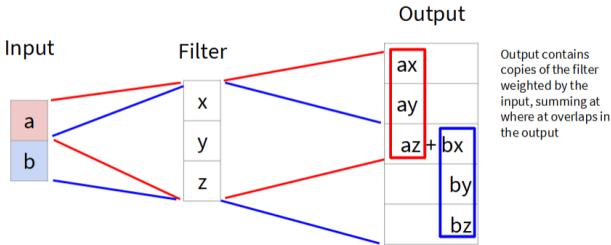
Stanford CS231n 10<sup>th</sup> Anniversary

Lecture 9 - 57 April 29, 2025

斯坦福大学 CS231n 10th 周年纪念

第 9 讲 - 57 2025 年 4 月 29 日

## Learnable Upsampling: 1D Example



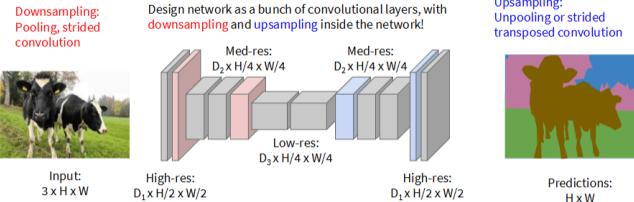
Stanford CS231n 10<sup>th</sup> Anniversary

Lecture 9 - 58 April 29, 2025

斯坦福大学 CS231n 10th 周年纪念

第 9 讲 - 58 2025 年 4 月 29 日

## Semantic Segmentation Idea: Fully Convolutional



Long, Shelhamer 和 Darrell, "Fully Convolutional Networks for Semantic Segmentation", CVPR 2015  
Noh 等人, "Learning Deconvolution Network for Semantic Segmentation", ICCV 2015

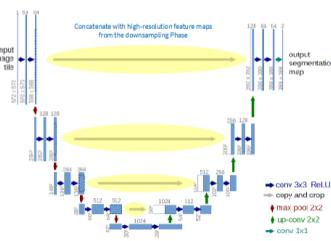
Stanford CS231n 10<sup>th</sup> Anniversary

Lecture 9 - 59 April 29, 2025

斯坦福大学 CS231n 10th 周年纪念

第 9 讲 - 59 2025 年 4 月 29 日

## U-Net



Stanford CS231n 10<sup>th</sup> Anniversary

Lecture 9 - 63 April 29, 2025

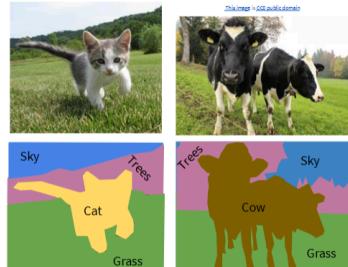
斯坦福大学 CS231n 10th 周年纪念

第 9 讲 - 63 2025 年 4 月 29 日

## Semantic Segmentation

Label each pixel in the image with a category label

Don't differentiate instances, only care about pixels



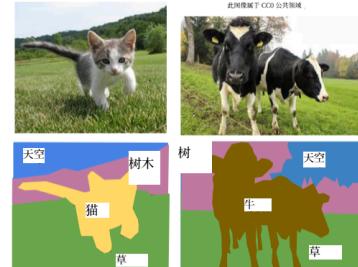
Stanford CS231n 10<sup>th</sup> Anniversary

Lecture 9 - 65 April 29, 2025

## 语义分割

用类别标签标记图像中的每个像素

不区分实例，只关注像素

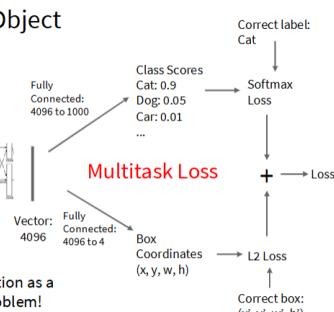
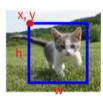


斯坦福大学 CS231n 10th 周年纪念

第 9 讲 - 65 2025 年 4 月 29 日

# 检测

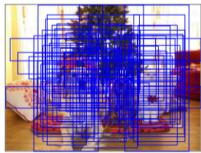
## Object Detection: Single Object (Classification + Localization)

Stanford CS231n 10<sup>th</sup> Anniversary

Lecture 9 - 70 April 29, 2025

## Object Detection: Multiple Objects

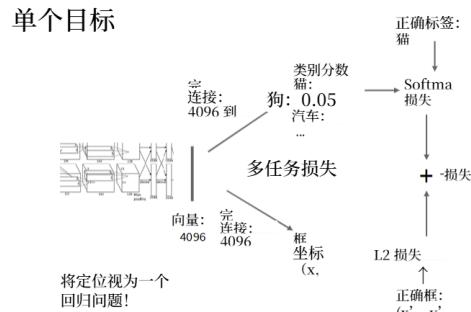
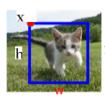
Apply a CNN to many different crops of the image, CNN classifies each crop as object or background



Problem: Need to apply CNN to huge number of locations, scales, and aspect ratios, very computationally expensive!

Dog? NO  
Cat? YES  
Background? NO

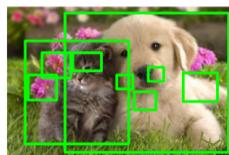
## 目标检测：单个目标 (分类 + 定位)

斯坦福大学 CS231n 10<sup>th</sup> 周年纪念

第 9 讲 - 70 2025 年 4 月 29 日

## Region Proposals: Selective Search

- Find “blobby” image regions that are likely to contain objects
- Relatively fast to run; e.g. Selective Search gives 2000 region proposals in a few seconds on CPU

Stanford CS231n 10<sup>th</sup> Anniversary

Lecture 9 - 78 April 29, 2025

## 区域提议：选择性搜索

- 找到可能包含物体的“模糊状”图像区域 ●  
运行速度相对较快；例如，选择性搜索在 CPU 上几秒钟内就能生成 2000 个区域提案

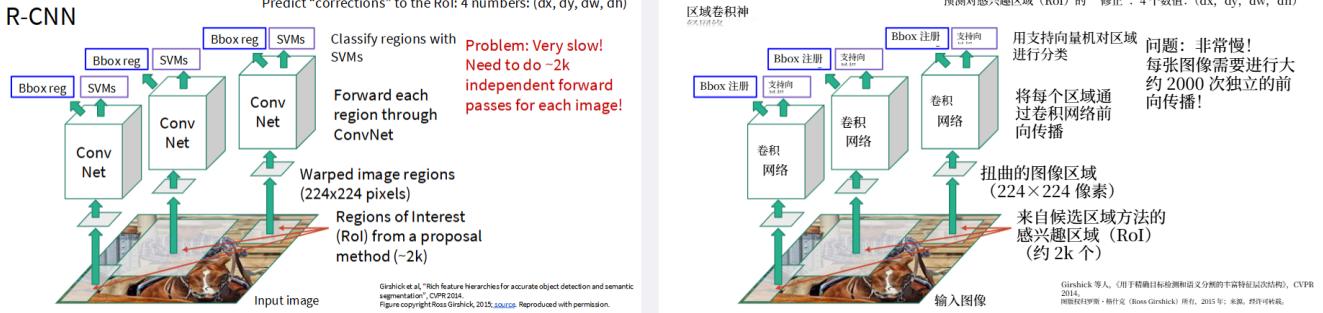


ConvNet (ImageNet-pre-trained): 指预训练的卷积神经网络。

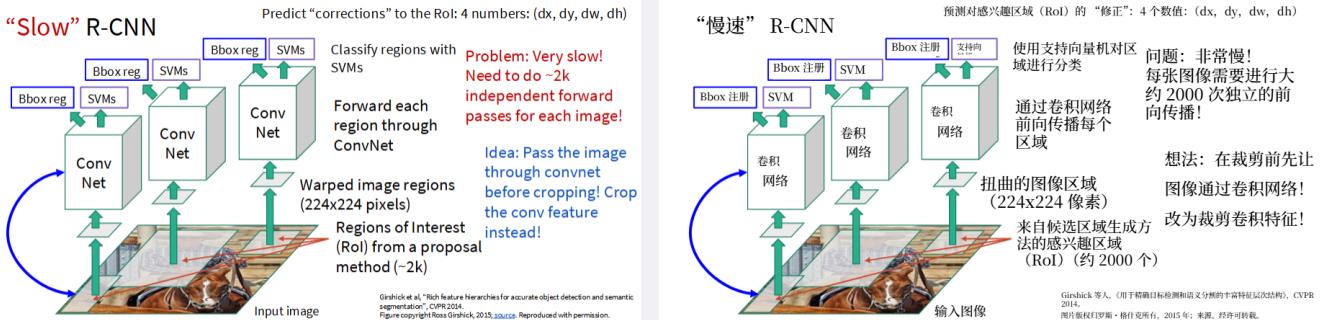
其中：

“ConvNet”是核心特征提取器（如 AlexNet、VGG 等经典 CNN 结构）；

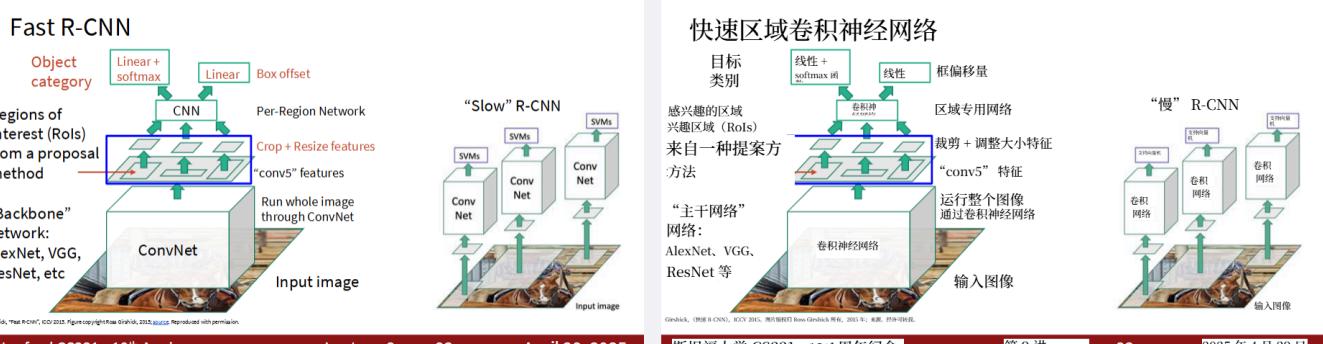
“ImageNet-pre-trained”表示该卷积网络已在 ImageNet 数据集（含 1000 个类别、数百万图像的大规模图像分类数据集）上完成训练，已学习到通用的图像特征（如边缘、纹理、局部形状等），可用于迁移学习。

Stanford CS231n 10<sup>th</sup> Anniversary

Lecture 9 - 85 April 29, 2025

Stanford CS231n 10<sup>th</sup> Anniversary

Lecture 9 - 86 April 29, 2025

Stanford CS231n 10<sup>th</sup> Anniversary

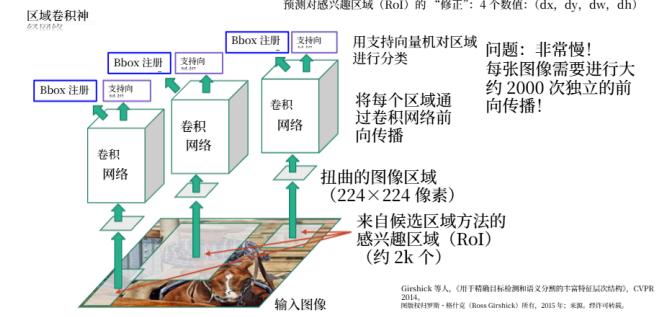
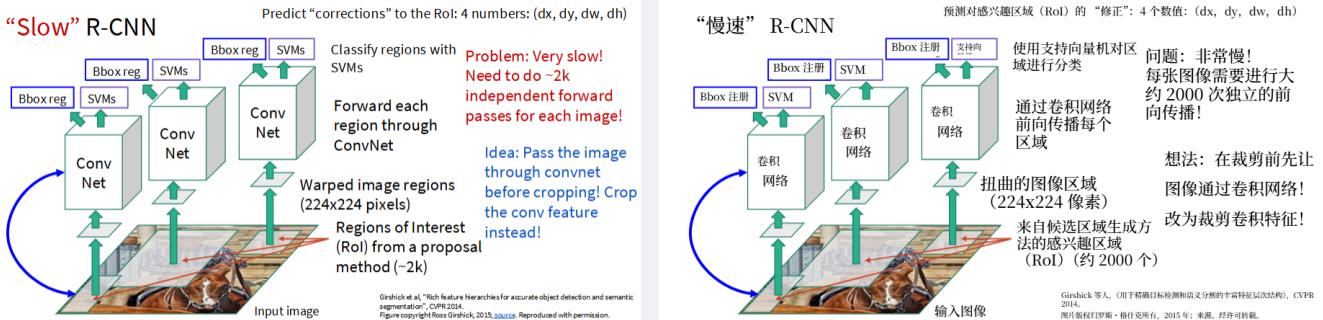
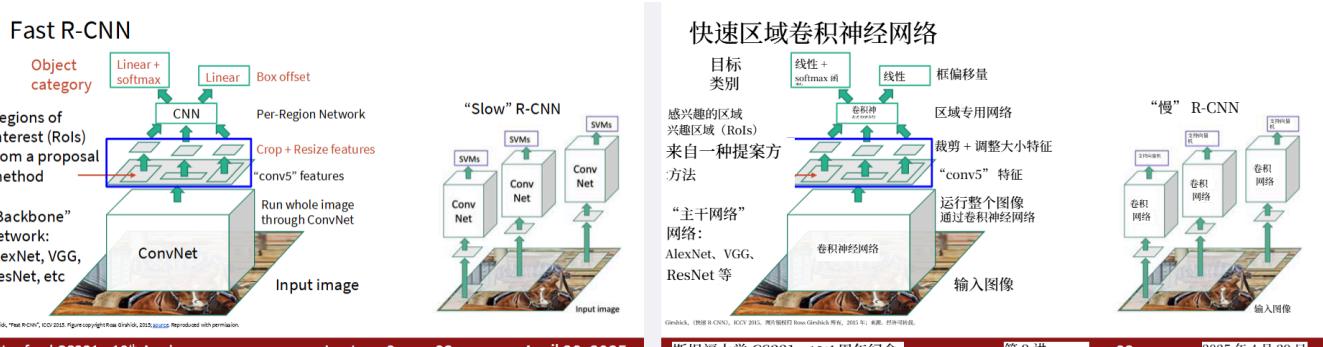
Lecture 9 - 92 April 29, 2025

Stanford CS231n 10<sup>th</sup> Anniversary

Lecture 9 - 96 April 29, 2025

Stanford CS231n 10<sup>th</sup> Anniversary

Lecture 9 - 98 April 29, 2025

斯坦福大学 CS231n 10<sup>th</sup>周年纪念 第 9 讲 - 85 2025 年 4 月 29 日斯坦福大学 CS231n 10<sup>th</sup>周年纪念 第 9 讲 - 86 2025 年 4 月 29 日斯坦福大学 CS231n 10<sup>th</sup>周年纪念 第 9 讲 - 92 2025 年 4 月 29 日Stanford CS231n 10<sup>th</sup> Anniversary

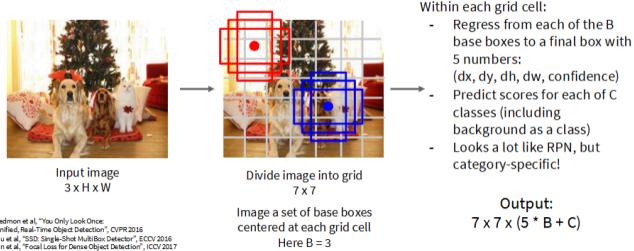
Lecture 9 - 96 April 29, 2025

Stanford CS231n 10<sup>th</sup> Anniversary

Lecture 9 - 98 April 29, 2025

斯坦福大学 CS231n 10<sup>th</sup>周年纪念 第 9 讲 - 96 2025 年 4 月 29 日

## Single-Stage Object Detectors: YOLO / SSD / RetinaNet

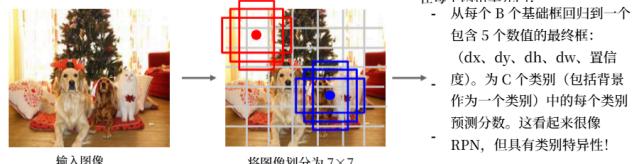


Stanford CS231n 10th Anniversary

Lecture 9 - 99

April 29, 2025

## 单阶段目标检测器：YOLO / SSD / RetinaNet



Output:

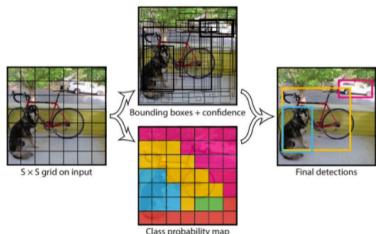
斯坦福大学 CS231n 十周年纪念

第九讲 -

99

2025 年 4 月 29 日

## YOLO (You Only Look Once) real-time object detection

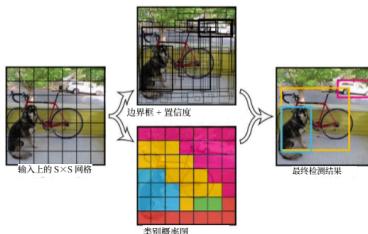


Stanford CS231n 10th Anniversary

Lecture 9 - 100

April 29, 2025

## YOLO (你只看一次) 实时目标检测



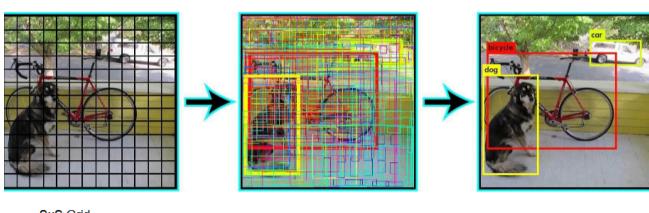
斯坦福大学 CS231n 10th 周年纪念

第 9 讲 -

100

2025 年 4 月 29 日

## YOLO

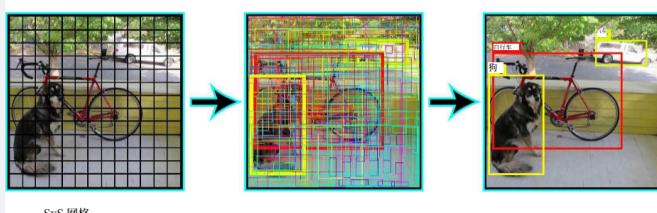


Stanford CS231n 10th Anniversary

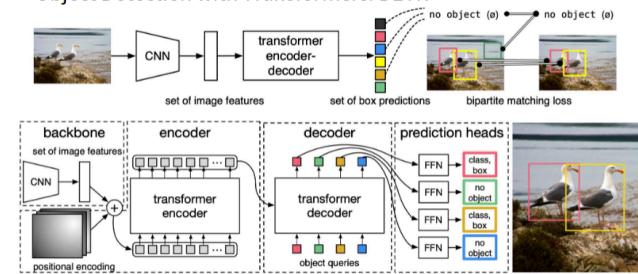
Lecture 9 - 106

April 29, 2025

## YOLO



## Object Detection with Transformers: DETR

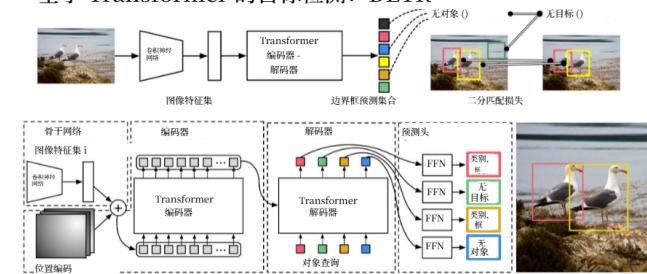


Stanford CS231n 10th Anniversary

Lecture 9 - 112

April 29, 2025

## 基于 Transformer 的目标检测：DETR



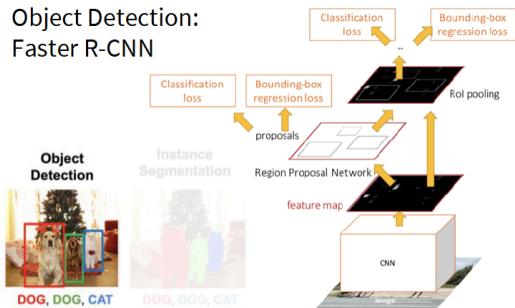
斯坦福大学 CS231n 10th 周年纪念

第 9 讲 -

112

2025 年 4 月 29 日

## Object Detection: Faster R-CNN

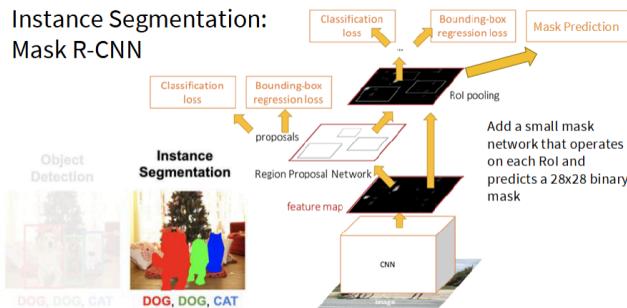


Stanford CS231n 10<sup>th</sup> Anniversary

Lecture 9 - 114

April 29, 2025

## Instance Segmentation: Mask R-CNN



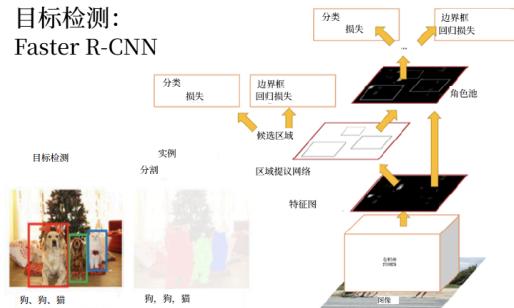
He et al., "Mask R-CNN", ICCV 2017

Stanford CS231n 10<sup>th</sup> Anniversary

Lecture 9 - 115

April 29, 2025

## 目标检测： Faster R-CNN

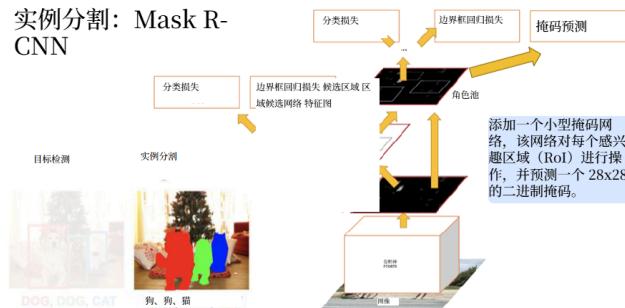


斯坦福大学 CS231n 10<sup>th</sup> 周年纪念

第 9 讲 - 114

2025 年 4 月 29 日

## 实例分割：Mask R-CNN



何等人, (Mask R-CNN), ICCV 2017

斯坦福大学 CS231n 10<sup>th</sup> 周年纪念

第 9 讲 - 115

2025 年 4 月 29 日

# 可视化与理解

- 第一层：可视化卷积核
- 哪些像素点重要：通过反向传播实现显著性检测；前向传播：计算概率
- 最后一个卷积层：类激活映射 (CAM)
- 任意层：梯度加权类激活映射 (Grad-CAM)
- 可视化ViT特征

## Object Detection: Lots of variables ...

Backbone Network	"Meta-Architecture"
VGG16	Two-stage: Faster R-CNN
ResNet-101	Single-stage: YOLO / SSD
Inception V2	Hybrid: R-FCN
Inception V3	Image Size
Inception ResNet	# Region Proposals
MobileNet	...

Takeaways
Faster R-CNN is slower but more accurate
SSD is much faster but not as accurate
Bigger / Deeper backbones work better

Huang et al., "Speed/accuracy trade-offs for modern convolutional object detectors", CVPR 2017

Zou et al., "Object Detection in 20 Years: A Survey", arXiv 2019

I-FCN: 带行人 (R-FCN: 带行人和车辆的全连接网络), 对比训练模型在 2016 年度图像识别竞赛上取得了第二名。YOLO: 一个单阶段的端到端模型，直接从输入图像到输出 bounding box 和类别的全连接神经网络，ICML 2015  
Inception V3: Search 等人, (重新设计 Inception 架构用于计算机视觉), arXiv 2016  
Inception ResNet: Szegedy 等人, (Inception-V2, Inception-ResNet 和深度残差连接对神经元学习的影响), arXiv 2016  
MobileNet: Howard 等人, (深度卷积神经网络为移动设备而设计), arXiv 2017

Stanford CS231n 10<sup>th</sup> Anniversary

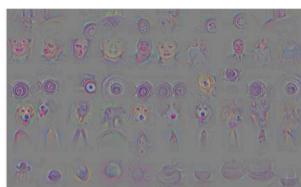
Lecture 9 - 174

April 29, 2025

## Intermediate features via (guided) backprop



Maximally activating patches  
(Each row is a different neuron)



Guided Backprop

## 目标检测：有很多变量……

骨干网络	"元架构"
网络	两阶段: Faster R-CNN
VGG16	单阶段: YOLO / SSD
ResNet-101	混合式: R-FCN
Inception V2	图像尺寸
Inception V3	区域提议
Inception ResNet	移动

要点	Faster R-CNN 速度较慢但精度更高
SSD 速度快得多, 但准确性不高	
更大 / 更深的骨干网络效果更好	

黄等人, "现代卷积目标检测器的速度 / 精度权衡", CVPR 2017

邹等人, (20 年目标检测综述), arXiv 2019

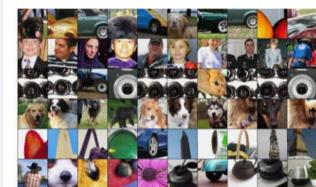
I-FCN: 行人 (R-FCN: 行人和车辆的全连接网络), 对比训练模型在 2016 年度图像识别竞赛上取得了第二名。YOLO: 一个单阶段的端到端模型，直接从输入图像到输出 bounding box 和类别的全连接神经网络，ICML 2015  
Inception V3: Search 等人, (重新设计 Inception 架构用于计算机视觉), arXiv 2016  
Inception ResNet: Szegedy 等人, (Inception-V2, Inception-ResNet 和深度残差连接对神经元学习的影响), arXiv 2016  
MobileNet: Howard 等人, (深度卷积神经网络为移动设备而设计), arXiv 2017

斯坦福大学 CS231n 10<sup>th</sup> 周年纪念

第 9 讲 - 174

2025 年 4 月 29 日

## 通过（引导式）反向传播获得的中间特征



最大激活补丁  
(每行是一个不同的神经元)



引导反向传播

Stanford CS231n 10<sup>th</sup> Anniversary

Lecture 9 - 178

April 29, 2025

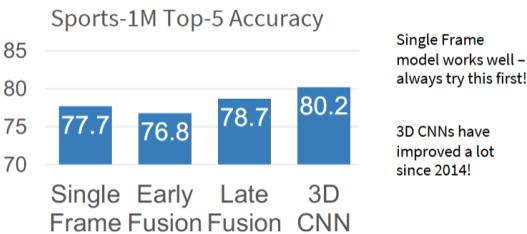
斯坦福大学 CS231n 10<sup>th</sup> 周年纪念

第 9 讲 - 178

2025 年 4 月 29 日

# 第十讲：视频理解

## Early Fusion vs Late Fusion vs 3D CNN



Karpathy et al., "Large-scale Video Classification with Convolutional Neural Networks", CVPR 2014

Stanford CS231n 10<sup>th</sup> Anniversary

Lecture 10 - 39 May 1, 2025

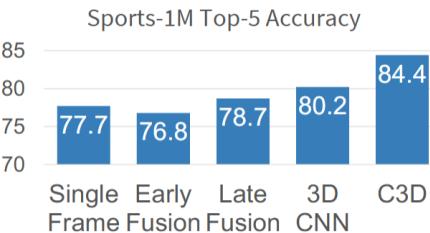
## 早期融合 vs 晚期融合 vs 3D 卷积神经网络



斯坦福大学 CS231n 10<sup>th</sup> 周年纪念

第 10 讲 - 39 2025 年 5 月 1 日

## Early Fusion vs Late Fusion vs 3D CNN



Karpathy et al., "Large-scale Video Classification with Convolutional Neural Networks", CVPR 2014

Stanford CS231n 10<sup>th</sup> Anniversary

Lecture 10 - 42 May 1, 2025

## 早期融合 vs 晚期融合 vs 3D 卷积神经网络

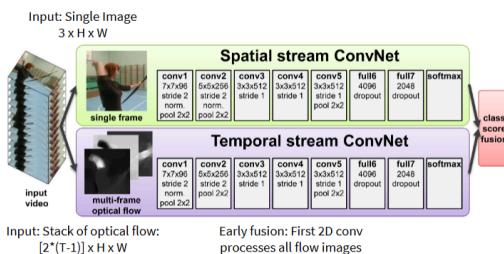


Karpathy et al., "Large-scale Video Classification with Convolutional Neural Networks", CVPR 2014

斯坦福大学 CS231n 10<sup>th</sup> 周年纪念

第 10 讲 - 42 2025 年 5 月 1 日

## Separating Motion and Appearance: Two-Stream Networks

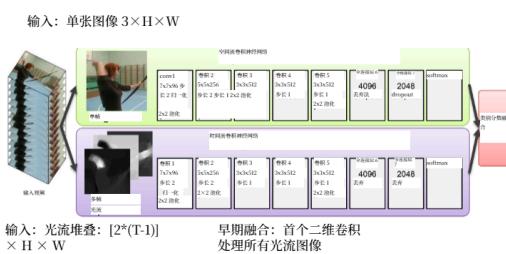


Simonyan and Zisserman, "Two-stream convolutional networks for action recognition in videos", NeurIPS 2014

Stanford CS231n 10<sup>th</sup> Anniversary

Lecture 10 - 47 May 1, 2025

## 分离运动与外观：双流网络

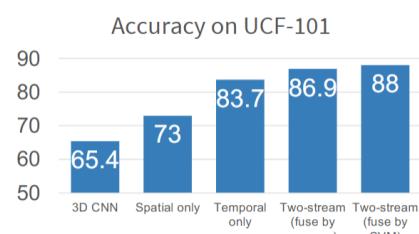


Simonyan et Zisserman, "用于视频动作识别的双流卷积网络", NeurIPS 2014

斯坦福大学 CS231n 10<sup>th</sup> 周年纪念

第 10 讲 - 47 2025 年 5 月 1 日

## Separating Motion and Appearance: Two-Stream Networks

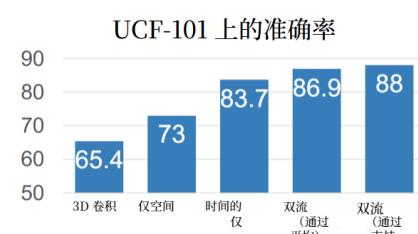


Simonyan and Zisserman, "Two-stream convolutional networks for action recognition in videos", NeurIPS 2014

Stanford CS231n 10<sup>th</sup> Anniversary

Lecture 10 - 48 May 1, 2025

## 分离运动与外观：双流网络



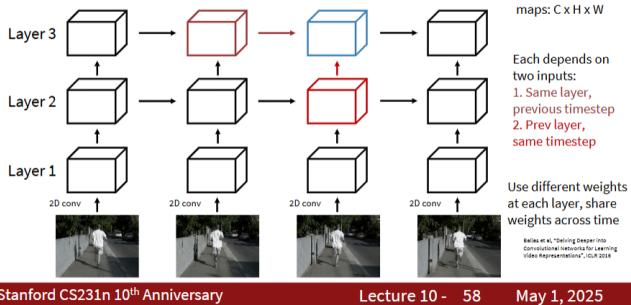
Simonyan et Zisserman, "用于视频动作识别的双流卷积网络", NeurIPS 2014

斯坦福大学 CS231n 10<sup>th</sup> 周年纪念

第 10 讲 - 48 2025 年 5 月 1 日

- 长时序结构建模方法：引入RNN（但速度慢，无法并行）

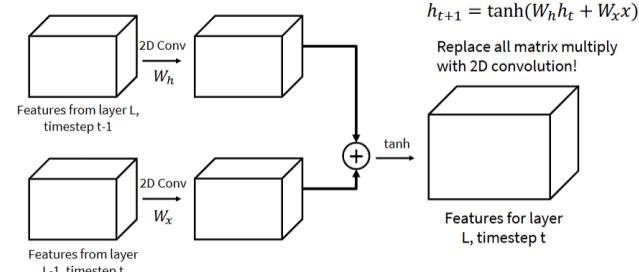
## Recurrent Convolutional Network



Stanford CS231n 10<sup>th</sup> Anniversary

Lecture 10 - 58 May 1, 2025

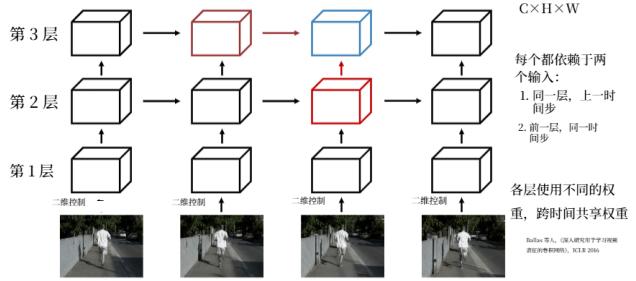
## Recurrent Convolutional Network



Stanford CS231n 10<sup>th</sup> Anniversary

Lecture 10 - 61 May 1, 2025

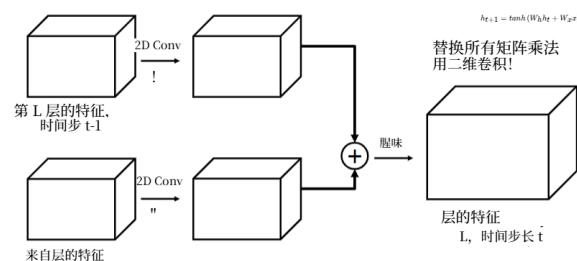
## 循环卷积网络



斯坦福大学 CS231n 10周年纪念

第 10 讲 - 58 2025 年 5 月 1 日

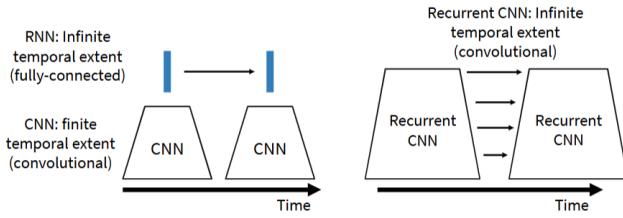
## 循环卷积网络



斯坦福大学 CS231n 10周年纪念

第 10 讲 - 61 2025 年 5 月 1 日

## Modeling long-term temporal structure

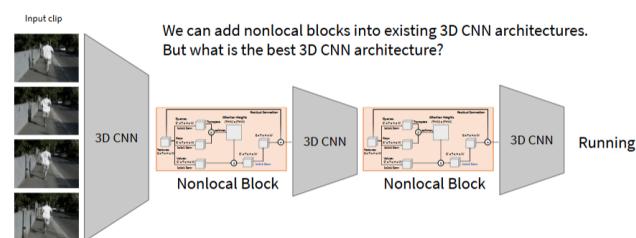


Stanford CS231n 10<sup>th</sup> Anniversary

Lecture 10 - 62 May 1, 2025

- 引入自注意力

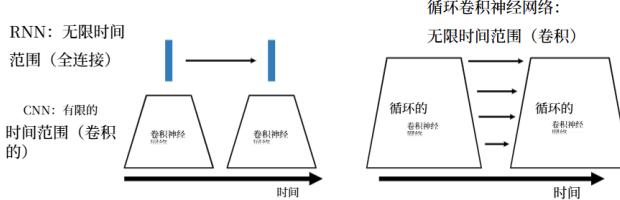
## Spatio-Temporal Self-Attention (Nonlocal Block)



Stanford CS231n 10<sup>th</sup> Anniversary

Lecture 10 - 71 May 1, 2025

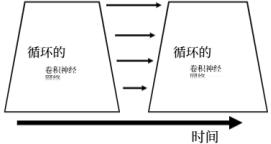
## 建模长期时间结构



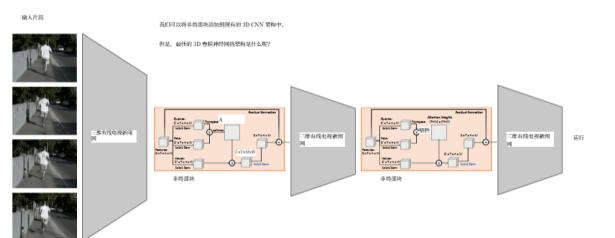
斯坦福大学 CS231n 10周年纪念

第 10 讲 - 62 2025 年 5 月 1 日

循环卷积神经网络：  
无限时间范围（卷积）



时空自注意力 (全局观)



斯坦福大学 CS231n 10周年纪念

第 10 讲 - 71 2025 年 5 月 1 日

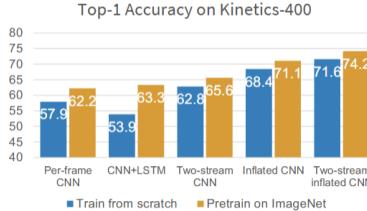
## Inflating 2D Networks to 3D (I3D)

There has been a lot of work on architectures for images.  
Can we reuse image architectures for video?

Idea: take a 2D CNN architecture.

Replace each  $2D K_h \times K_w$  conv/pool layer with a  $3D K_t \times K_h \times K_w$  version

Can use weights of 2D conv to initialize 3D conv: copy  $K_t$  times in space and divide by  $K_t$   
This gives the same result as 2D conv given "constant" video input



Carreira and Zisserman, "Quo Vadis, Action Recognition? A New Model and the Kinetics Dataset", CVPR 2017

Stanford CS231n 10th Anniversary

Lecture 10 - 76 May 1, 2025

## 将 2D 网络扩展到 3D (I3D)

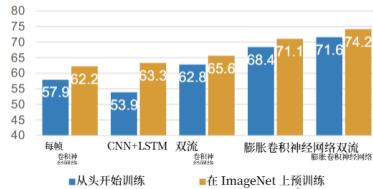
在图像架构方面已经有了大量的研究。  
我们可以将图像架构重用于视频吗?

思路: 采用二维卷积神经网络架构。

将每个  $2D K_h \times K_w$  卷积 / 池化层  
替换为 3D 的  $K_t \times K_h \times K_w$  版本

可以使用二维卷积的权重来初始化三维卷积: 在空间上复制  $K_t$  次  
空间并除以  $K_t$   
在“恒定”视频输入的情况下, 这会得到与二维卷积相同的结果。

Kinetics-400 上的 Top-1 准确率

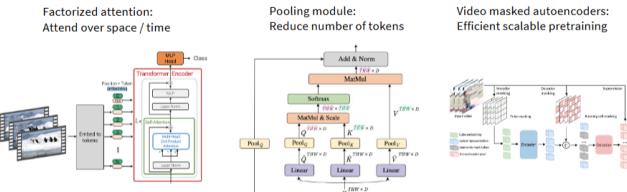


Carreira 和 Zisserman, "Quo Vadis, Action Recognition? A New Model and the Kinetics Dataset", CVPR 2017

斯坦福大学 CS231n 10th 周年纪念

第 10 讲 - 76 2025 年 5 月 1 日

## Vision Transformers for Video



Bertasius et al., "Is Space-Time Attention All You Need for Video Understanding?", ICML 2021

Arnab 等人, "MVIT: A Video Vision Transformer", ICCV 2021

Neimark 等人, "Video Transformer Network", ICCV 2021

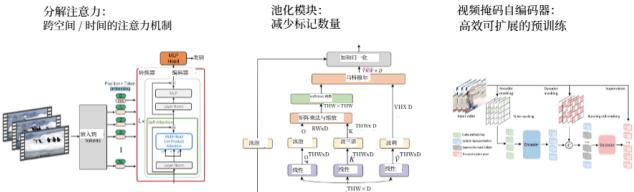
Ean et al., "Multi-Scale Vision Transformers", IJCV 2021

Li et al., "MVITv2: Improved Multi-Scale Vision Transformers for Classification and Detection", CVPR 2022

Wang et al., "VideoMAE V2: Scaling Video Masked Autoencoders with Dual Making", CVPR 2023

Video masked autoencoders:  
Efficient scalable pretraining

## 用于视频的视觉 Transformer



Bertasius 等人, 《时空注意力是视频理解的全部所需求?》, ICML 2021

Arnab 等人, 《MVIT: 视觉视频 Transformer》, ICCV 2021

Neimark 等人, 《视频 Transformer 网络》, ICCV 2021

Fan 等人, 《多尺度视觉 Transformer》, ICCV 2021

Li 等人, 《MVITv2: 用于分类和检测的改进型多尺度视觉 Transformer》, ICCV 2022

Feichtenhofer 等人, 《作为时空学习的视频的自编码器》, 视觉信息处理系统大会 2022

Wang 等人, VideoMAE V2: 通过双重掩码扩展现有模型的自编码器, CVPR 2023

童等人, VideoMAE: 基于自编码器的用于自监督视频数据的有效学习者, NeurIPS 2022

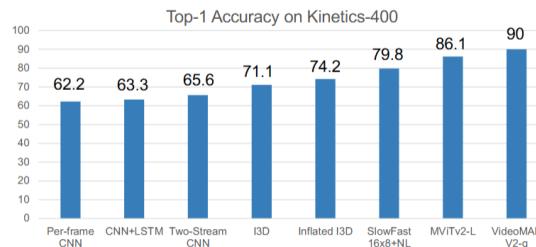
Feichtenhofer 等人, 作为时空学习的视频的自编码器, 视觉信息处理系统大会 2022

Stanford CS231n 10th Anniversary

Lecture 10 - 77 May 1, 2025

第 10 讲 - 77 2025 年 5 月 1 日

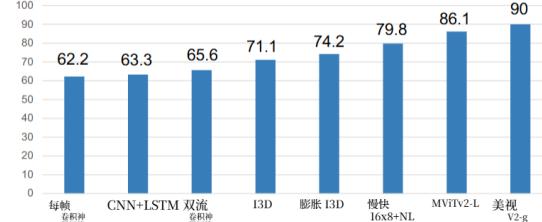
## Vision Transformers for Video



Stanford CS231n 10th Anniversary

Lecture 10 - 78 May 1, 2025

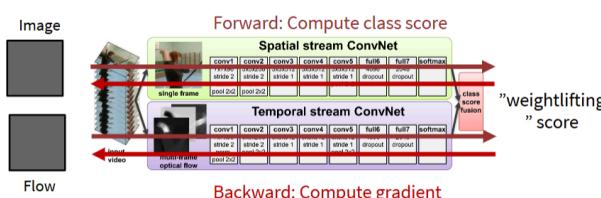
Kinetics-400 数据集上的 Top-1 准确率



斯坦福大学 CS231n 10th 周年纪念

第 10 讲 - 78 2025 年 5 月 1 日

## Visualizing Video Models



Add a term to encourage spatially smooth flow; tune penalty to pick out “slow” vs “fast” motion

Figure credit: Simonyan 和 Zisserman, "Three-dimensional convolutional networks for action recognition in videos", NeurIPS 2014; Feichtenhofer 等人, "What have we learned from deep representations for action recognition?", CVPR 2018; Feichtenhofer 等人, "Deep insights into convolutional networks for video recognition?", IJCV 2019.

Stanford CS231n 10th Anniversary

Lecture 10 - 79 May 1, 2025

## 可视化视频模型



添加一个项以促进空间上平滑的流动;  
调整惩罚项以区分“慢速”与“快速”运动

Figure credit: Simonyan 和 Zisserman, "用于视频动作识别的深度卷积网络", NeurIPS 2014; Feichtenhofer 等人, "从深度学习中我们学到了什么?", CVPR 2018; Feichtenhofer 等人, "对于视频识别的深度卷积神经网络的深入见解", IJCV 2019.

斯坦福大学 CS231n 10th 周年纪念

第 10 讲 - 79 2025 年 5 月 1 日

## # 斯坦福CS231n第十讲: 视频理解总结

本讲围绕视频理解展开, 从视频数据特性、核心任务、主流模型架构到前沿应用方向, 系统梳理了视频理解领域的关键技术与发展脉络, 同时对比了不同方法的优劣及性能表现。

### ## 一、视频理解基础与核心挑战

#### ### 1. 视频的本质与表示

视频可视为“2D图像+时间维度”的组合，以4D张量（ $T \times 3 \times H \times W$ 或 $3 \times T \times H \times W$ ）表示，其中T为帧数、3对应RGB通道、H和W为空间分辨率。其核心任务区别于图像分类（识别物体），聚焦于动作识别（如游泳、跑步等）。

### ### 2. 核心挑战：数据规模庞大

视频数据量远超图像，例如30帧/秒的非压缩视频，标清（640×480）每分钟约1.5GB，高清（1920×1080）每分钟约10GB。解决方案是基于短片段训练：降低帧率和空间分辨率（如 $T=16$ 、 $H=W=112$ ，5帧/秒下仅3.2秒，588KB），训练时用低帧率短片段，测试时对多个片段预测结果取平均。

## ## 二、视频分类核心模型架构

### ### 1. 基于2D CNN的扩展方法

- \*\*单帧CNN\*\*：独立处理视频帧，测试时平均预测概率，是视频分类的强基准模型，性能意外优秀（如在Sports-1M数据集上Top-5准确率达84.4%）。
- \*\*晚期融合（Late Fusion）\*\*：用2D CNN提取每帧高级特征后，通过全连接层拼接或时空池化组合特征，再输入分类器。缺点是难以捕捉帧间低级运动信息。
- \*\*早期融合（Early Fusion）\*\*：将视频张量重塑为 $3T \times H \times W$ ，用第一层2D卷积融合时间信息，后续为标准2D CNN。问题是仅一层时间处理可能不足以捕捉复杂时序关系，且无时间平移不变性，需为不同时间位置的相同运动学习独立滤波器。

### ### 2. 3D CNN方法

- \*\*核心思想\*\*：用3D卷积和池化操作，在网络层级中逐步融合时空信息，每层输出为4D张量（ $D \times T \times H \times W$ ），实现“慢融合”（空间和时间维度均逐步构建感受野）。
- \*\*优势\*\*：具备时间平移不变性，滤波器可在时间维度滑动，能捕捉任意时空位置的运动模式（如蓝橙颜色过渡）。
- \*\*经典模型C3D\*\*：被称为“3D CNN中的VGG”，采用 $3 \times 3 \times 3$ 卷积和 $2 \times 2 \times 2$ 池化（第一层池化为 $1 \times 2 \times 2$ ），预训练于Sports-1M数据集，常作为视频特征提取器。缺点是计算成本极高（39.5 GFLOP，是VGG-16的2.9倍）。
- \*\*性能对比\*\*：在Sports-1M数据集上，3D CNN（80.2%）优于早期融合（77.7%）和晚期融合（76.8%），且近年性能提升显著。

### ### 3. 双流网络（Two-Stream Networks）

- \*\*核心思路\*\*：分离外观和运动信息分别建模，再融合结果。
  - 空间流：输入单帧图像，用2D CNN提取外观特征。
  - 时间流：输入光流堆叠（ $2 \times (T-1) \times H \times W$ ，包含水平和垂直位移信息），用2D CNN提取运动特征。
- \*\*性能\*\*：在UCF-101数据集上，双流融合（SVM融合准确率88%、平均融合86.9%）显著优于单一空间流（73%）、时间流（65.4%）和3D CNN。

### ### 4. 长时时序结构建模方法

- \*\*CNN+RNN/LSTM\*\*：用2D/3D CNN提取局部片段特征，再通过RNN/LSTM处理序列特征，支持“多对一”（视频级输出）或“多对多”（帧级输出）任务。可固定CNN参数作为特征提取器以节省内存，结合了CNN的局部时空特征捕捉能力和RNN的全局时序依赖建模能力。
- \*\*循环卷积网络（Recurrent Convolutional Network）\*\*：用2D卷积替代RNN中的矩阵乘法，每层特征图（ $C \times H \times W$ ）同时依赖前一时刻同层特征和当前时刻前一层特征，权重跨时间共享，兼具CNN的空间建模能力和RNN的长时依赖捕捉能力，且为卷积操作而非全连接，效率更高。
- \*\*时空自注意力（Nonlocal Block）\*\*：在3D CNN中插入非局部块，通过 $1 \times 1 \times 1$ 卷积生成查询（Queries）、键（Keys）和值（Values），计算全局时空注意力权重，捕捉长距离时空依赖。可作为残差连接插入现有3D CNN架构，提升长时结构建模能力。

### ### 5. 2D网络3D扩展（I3D）与视觉Transformer

- \*\*I3D核心思想\*\*：复用成熟2D CNN架构，将2D卷积/池化层替换为3D版本（如 $2D \ 3 \times 3 \rightarrow 3D \ Kt \times 3 \times 3$ ），用2D权重初始化3D权重（复制 $Kt$ 次并除以 $Kt$ ），确保对“恒定视频”输入与2D CNN输出一致。在Kinetics-400数据集上，预训练于ImageNet的膨胀CNN性能（74.2%）优于双流网络（71.1%）和CNN+LSTM（65.6%）。
- \*\*视频视觉Transformer\*\*：采用因子化注意力（分别关注空间/时间维度）、池化模块减少token数量，结合视频掩码自编码器实现高效预训练。代表性模型如MViTv2-L、VideoMAE V2，在Kinetics-400数据集上Top-1准确率达86.1%，显著超越传统3D CNN（如I3D 74.2%）。

## ## 三、视频理解的进阶任务与应用

### ### 1. 进阶任务

- \*\*时序动作定位\*\*: 对长视频识别不同动作对应的帧区间，可借鉴**Faster R-CNN**架构，先生成时序候选区域再分类。
- \*\*时空检测\*\*: 在长视频中同时定位空间中的人物及对应的时间区间，分类其动作（如AVA数据集的“举杯→喝酒”“看手机→接电话”等原子动作）。

### ### 2. 前沿应用方向

- \*\*视听融合视频理解\*\*: 结合视觉和音频信息，实现音频源分离（如分离多说话者声音、乐器声音）、跨域动作识别、视听掩码自编码预训练等，代表工作包括**visualvoice**、**Audio-Visual MAE**等。
- \*\*高效视频理解\*\*: 针对长视频优化，采用流式评估、显著片段采样（**SCSampler**）、音频预览机制（用音频特征指导图像采样）等方法提升效率，代表模型如**MoViNets**、**X3D**、**AdaMML**。
- \*\*第一视角视频理解\*\*: 基于摄像头和麦克风阵列，构建视听对话图，预测第一视角与他人的交互关系（如“说话对象”“倾听对象”）。
- \*\*视频+大语言模型（LLMs）\*\*: 通过视觉编码器提取视频时空特征，与LLM对齐融合，实现视频问答、详细视频理解（如**Video-ChatGPT**、**VideoLLaMA3**）。

## ## 四、关键结论与后续内容

1. \*\*模型选择优先级\*\*: 单帧CNN作为基准，3D CNN/双流网络提升性能，I3D和Transformer模型在现代数据集上表现最优。
2. \*\*核心权衡\*\*: 模型性能与计算成本（如3D CNN性能优但耗资源，Transformer高效且性能领先）、短片段建模与长时依赖捕捉（RNN/注意力机制解决长时问题）。
3. \*\*后续内容\*\*: 下一讲将聚焦大规模分布式训练，为视频理解模型的高效训练提供支撑。

# 第十一讲：大规模分布式训练

## # 斯坦福CS231n第11讲：大规模分布式训练总结

本讲聚焦大规模分布式训练技术，以Llama3-405B模型为核心案例，围绕GPU硬件特性、多GPU并行训练策略、内存优化及性能评估展开详细讲解，核心目标是通过合理的技术方案实现超大规模模型的高效训练。

## ## 一、核心背景与硬件基础

### ### 1. 案例模型与行业对比

- \*\*Llama3优势\*\*: Meta 2024年开源的大语言模型，公开了模型架构、训练细节等关键信息（区别于GPT-4的闭源策略），其405B参数版本是本讲分布式训练的核心案例；Llama4于2025年4月发布初始模型，但暂未公开技术论文。
- \*\*硬件核心\*\*: NVIDIA H100 GPU: 作为当前主流训练芯片，本质是通用并行处理器（原用于图形处理），关键参数包括：80GB HBM内存、3352 GB/sec内存带宽、132个启用的流式多处理器（SM）、50MB L2缓存；每个SM含128个FP32核心（256 FLOP/周期）和4个张量核心（4096 FLOP/周期，支持混合精度计算）。
- \*\*GPU性能演进\*\*: 自2013年以来，GPU计算能力实现千倍提升（如B200的FP32张量核心性能达83.3 TFLOP/sec），且支持多GPU协同训练，为大规模模型提供硬件支撑。
- \*\*其他训练芯片\*\*: Google TPU v5p (459 TFLOP/sec BF16、95GB内存)、AMD MI325X (1300 TFLOP/sec BF16、256GB内存)、AWS Trainium2 (667 TFLOP/sec BF16、96GB内存)，各有适配场景。

### ### 2. 大规模GPU集群架构（以Meta Llama3集群为例）

- 层级结构：单服务器含8个H100 GPU (GPU间带宽900 GB/sec) → 1个机架含2台服务器 (16个GPU) → 1个Pod含192个机架 (3072个GPU, GPU间带宽50 GB/sec) → 1个集群含8个Pod (24576个GPU)。
- 集群性能：总计1.875 PB GPU内存、415M FP32核心、13M张量核心，总算力达24.3 EFLOP/sec ( $24.3 \times 10^{18}$  FLOP/sec)，可支撑超大规模模型训练。

## ## 二、多GPU并行训练核心策略

模型训练的核心是对张量（维度为Batch×Sequence×Dim）和网络层（L层）进行拆分，四大基础并行策略及衍生方案如下：

### ### 1. 数据并行 (DP)

- \*\*核心逻辑\*\*: 将训练数据的批次（Batch）拆分到M个GPU，每个GPU持有完整模型副本，独立计算局部梯度后聚合平均，最终同步更新权重。

- 数学原理：损失函数和梯度具有线性可加性，全局梯度为所有GPU局部梯度的平均值（公式： $\frac{1}{M} \sum_{i=1}^M$  局部梯度）。
- \*\*执行步骤\*\*：GPU初始化完整模型→加载局部批次数据→前向计算损失→反向计算局部梯度→聚合梯度→更新各自模型权重（梯度计算与聚合可并行优化）。
- \*\*局限\*\*：模型大小受单GPU内存限制（如10B参数模型需80GB内存存储参数、梯度及优化器状态），无法支撑超大规模模型。

### ### 2. 完全分片数据并行 (FSDP)

- \*\*核心改进\*\*：解决DP的内存瓶颈，将模型权重、梯度及优化器状态分片存储在多个GPU上，每个GPU仅持有部分参数。

- \*\*关键流程\*\*：

1. 前向计算前，参数所属GPU将该层权重广播至所有GPU；
2. 所有GPU完成该层前向计算后，删除局部权重副本（优化：保留最后一层权重避免重复传输）；
3. 反向计算前，参数所属GPU再次广播权重；
4. 各GPU计算局部梯度并发送至参数所属GPU，由所属GPU聚合梯度并更新权重。

- \*\*优势\*\*：突破单GPU内存限制，如100B参数模型（需800GB存储）拆分到80个GPU后，单GPU仅需10GB内存。

### ### 3. 混合分片数据并行 (HSDP)

- \*\*核心逻辑\*\*：结合DP与FSDP，将N个GPU划分为M个组，每组K个GPU执行FSDP（分片存储模型），组间执行DP（拆分数据）。

- \*\*通信优化\*\*：组内GPU通信频繁（前向/反向需传输权重和梯度），优先部署在同一节点/Pod；组间仅需传输梯度，容忍较慢通信速度。
- \*\*示例\*\*：2组×4个GPU，每组内4个GPU分片存储模型，两组间拆分数据并行训练。

### ### 4. 激活检查点 (Activation Checkpointing)

- \*\*背景\*\*：即使通过FSDP拆分模型，Llama3-405B等大模型的激活值仍会占用大量内存（如FFN层隐藏激活值需63GB）。

- \*\*核心思想\*\*：不存储所有层激活值，反向计算时按需重新计算，通过少量计算开销换取内存节省。

- \*\*性能权衡\*\*：

- 全存储激活值：O(N)计算、O(N)内存；
- 全重新计算：O(N²)计算、O(1)内存；
- 间隔检查点：每C层存储一次激活值，实现O(N²/C)计算与O(C)内存的平衡（如取√N个检查点，达成O(N/√N)计算、O(√N)内存）。

### ### 5. 上下文并行 (CP)

- \*\*适用场景\*\*：主要针对Transformer模型的长序列处理，拆分序列 (Sequence) 维度，多个GPU协同处理单个长序列。

- \*\*并行细节\*\*：

- 归一化、残差连接：无权重，可直接并行；
- MLP层：各GPU持有权重副本，类似DP同步梯度；
- 自注意力层：最复杂，可通过“环形注意力”（拆分注意力矩阵块，支持超长序列）或“Ulysses”（按注意力头并行，实现简单但并行度受限）优化。
- \*\*案例\*\*：Llama3-405B训练第二阶段，序列长度从8192扩展至131072时，采用16路上下文并行（单GPU处理8192序列长度）。

### ### 6. 流水线并行 (PP)

- \*\*核心逻辑\*\*：拆分模型的层 (L层) 维度，不同GPU负责不同层的计算，通过传输层间激活值实现流水线训练。

- \*\*核心问题\*\*：原始方案存在严重空闲（GPU需等待前一层输出，N路PP的最大MFU仅1/N）。

- \*\*优化方案\*\*：引入微批次 (Microbatches)，多个微批次流水线并行执行，减少GPU空闲时间。示例：4路PP+4个微批次，MFU从25%提升至约57.1%。

### ### 7. 张量并行 (TP)

- \*\*核心逻辑\*\*：拆分线性层的权重维度 (Dim)，利用块矩阵乘法实现并行计算。

- \*\*执行方式\*\*：4路TP中，权重矩阵拆分为4个块 (1×4)，每个GPU计算输入与对应权重块的乘积，得到输出块。

- \*\*优化技巧\*\*：连续两层分别按行、列分片，可避免中间输出的聚合通信，提升效率。例如第一层按列拆分权重，第二层按行拆分，中间输出无需传输即可直接用于下一层计算。

### ### 8. 多维并行 (ND Parallelism)

- \*\*核心思想\*\*：超大规模模型（如Llama3-405B）需结合TP、CP、PP、DP四种策略，将GPU组织为4D网格，按维度分配不同并行任务。
- \*\*案例\*\*：Llama3-405B使用8192个GPU训练时，采用8路TP、1路CP、16路PP、64路DP，实现16M tokens/批次的高效训练，BF16 MFU达43%。

## ## 三、性能评估与优化目标

### ### 1. 硬件浮点运算利用率 (HFU)

- \*\*定义\*\*：实际矩阵乘法性能与GPU理论峰值性能的比值 (H100 16位矩阵乘法理论峰值989.4 TFLOP/sec)。
- \*\*特点\*\*：仅衡量核心计算性能，不包含激活检查点、数据预处理等辅助操作；大规模矩阵乘法可实现约80%的HFU。

### ### 2. 模型浮点运算利用率 (MFU)

- \*\*定义\*\*：衡量GPU理论峰值算力中用于“有效模型计算”（前向+反向矩阵乘法）的比例，是分布式训练的核心优化目标。
- \*\*计算步骤\*\*：
  1. 计算理论浮点运算量 ( $FLOP_{theoretical}$ )：前向矩阵乘法运算量+2×前向运算量（反向近似）；
  2. 计算理论时间 ( $t_{theoretical}$ ) =  $FLOP_{theoretical} / GPU$ 理论算力；
  3. 测量实际时间 ( $t_{actual}$ )：完整迭代（数据加载+前向+反向+优化器更新）的时间；
  4.  $MFU = t_{theoretical} / t_{actual}$ 。
- \*\*评估标准\*\*：MFU>30%为良好，>40%为优秀。例如PaLM（540B参数）达46.2%，Llama3-405B（8192 GPU）达43%。
- \*\*注意事项\*\*：新一代GPU（如A100→H100）的算力提升速度（3.1倍）远超内存带宽提升（2.1倍），可能导致MFU下降。

## ## 四、核心总结与实践建议

1. \*\*硬件与集群\*\*：GPU是大规模训练的核心，通过层级化集群架构（服务器→机架→Pod→集群）实现算力聚合，支撑超大规模模型训练。
2. \*\*并行策略选择\*\*：
  - 1B参数以下、≤128 GPU：优先数据并行；
  - 1B~50B参数、≤256 GPU：结合FSDP+激活检查点；
  - >256 GPU、>50B参数或序列长度>16K：采用HSDP或多维并行 (TP+CP+PP+DP)。
3. \*\*优化核心\*\*：以最大化MFU为目标，合理设置批次大小（充分利用GPU内存）、并行维度配比、激活检查点间隔等参数。
4. \*\*关键工具\*\*：依赖FSDP（模型分片）、激活检查点（内存优化）、多维并行（算力扩展）等技术，突破单GPU的内存和算力限制，实现千亿参数级模型的高效训练。

## How to train on lots of GPUs

A model with L layers operates on tensors of shape (Batch, Sequence, Dim)

**Data Parallelism (DP)**  
Split on Batch dimension

**Context Parallelism (CP)**  
Split on Sequence dimension

**Pipeline Parallelism (PP)**  
Split on L dimension

**Tensor Parallelism (TP)**  
Split on Dim dimension

## 如何在大量 GPU 上进行训练

一个具有 L 层的模型对形状为 (Batch, Sequence, Dim) 的张量进行操作

**数据并行 (DP)**  
按批次维度拆分

**上下文并行 (CP)**  
按序列维度拆分

**流水线并行 (PP)**  
按 L 维度拆分

**张量并行 (TP)**  
按 Dim 维度拆分

## How to train on lots of GPUs

HSDP + Activation checkpointing can take you a long way!

Scaling recipe:

1. Use **data parallelism** up to ~128 GPUs, models with ~1B params
2. Always set per-GPU batch size to max out GPU memory
3. If your model is >1B params, consider **FSDP**
4. Add **activation checkpointing** to fit larger batches per GPU
5. If you have >256 GPUs, consider **HSDP**
6. If you have >1K GPUs, models >50B params, or sequence lengths > 16K then use more advanced strategies (CP, PP, TP)

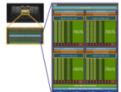
**Problem:** Lots of knobs to tune! How should we set them?

**Solution:** Maximize Model Flops Utilization (MFU)

### • 全部都要用

## Summary: Large-Scale Distributed Training

A GPU is a parallel processor with hundreds of cores



Split up the computation along different axes  
Consider a model with many Layers, operating on tensors of shape (Batch, Seq, Dim)

- Data Parallel (DP): Split on Batch
- Context Parallel (CP): Split on Seq
- Pipeline Parallel (PP): Split on Layers
- Tensor Parallel (TP): Split on Dim

Activation Checkpointing saves memory by recomputing during backward

Tune parallelism recipe to maximize **Model Flops Utilization (MFU)**

Stanford CS231n 10<sup>th</sup> Anniversary

Lecture 11 - 147

May 6, 2025

# 作业2代码

## BN

### # 批量归一化 (Batch Normalization)

使深度网络更易于训练的一种方法是采用更复杂的优化算法，例如随机梯度下降+动量（SGD+momentum）、RMSProp 或 Adam。另一种策略是通过修改网络架构来降低训练难度。2015年，文献[1]提出的\*\*批量归一化 (Batch Normalization) \*\*便是这一思路下的代表性方法。

要理解批量归一化的目标，首先需要明确：机器学习方法在处理\*\*特征无相关性、均值为0且方差为1\*\*的输入数据时，通常能取得更优性能。在训练神经网络时，我们可以在将数据输入网络前进行预处理，显式地对特征去相关——这能确保网络第一层接收的数据分布符合理想特性。然而，即便对输入数据做了预处理，网络深层的激活值仍可能不再满足“无相关性、零均值、单位方差”的条件（因为这些激活值是前层网络的输出结果）。更严重的是，在训练过程中，随着各层权重的更新，每一层特征的分布都会发生\*\*偏移\*\*。

文献[1]的作者推测，深度神经网络内部特征分布的这种动态偏移（即“内部协变量偏移”）可能是导致深度网络训练困难的关键原因。为解决这一问题，他们提出在网络中插入\*\*批量归一化层\*\*：

- \*\*训练阶段\*\*：该层利用当前批次（minibatch）的数据，估算每个特征的均值和标准差，并用这些估算值对当前批次的特征进行\*\*中心化（减去均值）\*\* 和 \*\*归一化（除以标准差）\*\* 处理；同时，会维护这些均值和标准差的\*\*移动平均值（running average）\*\*。
- \*\*测试阶段\*\*：不再使用单批次数据计算统计量，而是直接采用训练过程中维护的移动平均值对特征进行中心化和归一化。

需要注意的是，这种归一化策略可能会\*\*降低网络的表征能力\*\*——因为在某些情况下，特定层的特征保持非零均值或非单位方差可能是最优的。为此，批量归一化层为每个特征维度引入了\*\*可学习的偏移参数 (shift parameter)\*\* 和缩放参数 (scale parameter)\*\*，以保留网络的表征灵活性。

[1] Sergey Ioffe 和 Christian Szegedy, 《批量归一化：通过减少内部协变量偏移加速深度网络训练》，国际机器学习大会（ICML）2015年。

## 如何在大量 GPU 上进行训练

HSDP + 激活检查点能带你走很远！

扩展方案：

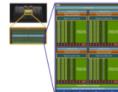
1. 使用数据并行，最多可支持约 128 个 GPU，模型参数约为 10 亿
2. 始终将每个 GPU 的批处理大小设置为最大化 GPU 内存的大小
3. 如果你的模型参数超过 10 亿，考虑使用 FSDP
4. 添加激活检查点以在每个 GPU 上容纳更大的批次
5. 如果您有超过 256 个 GPU，请考虑使用 HSDP
6. 如果你有超过 1K 个 GPU、models > 50 B 参数，或者序列长度 > 16 K 超过 16K，那么请使用更高级的策略 (CP, PP, TP)

问题：有很多需要调整的参数！我们应该如何设置它们？

解决方案：最大化模型浮点运算利用率 (MFU)

## 摘要：大规模分布式训练

GPU 是一种拥有数百个核心的并行处理器



一个 GPU 集群有大约 10K 个 GPU



沿不同轴拆分计算  
考虑一个具有许多层的模型，它处理形状为 (Batch, Seq, Dim) 的张量。

- 数据并行 (DP): 按批次拆分 -  
• 上下文并行 (CP): 按序列拆分  
• 流水线并行 (PP): 按层拆分  
• 索量并行 (TP): 按维度拆分

激活检查点通过在反向传播过程中重新计算节省内存

调整并行策略以最大化模型浮点运算利用率 (MFU)

斯坦福大学 CS231n 十周年纪念

第 11 讲 - 147

2025 年 5 月 6 日

### 关键术语补充说明（贴合CS231n课程语境）

1. \*\*Internal Covariate Shift (内部协变量偏移) \*\*:

课程中重点强调的核心概念，指训练过程中网络各层输入数据的分布随权重更新而持续变化的现象。这种偏移会导致后层网络需要不断适应新的分布，不仅减慢训练速度，还可能导致梯度消失/爆炸。

2. \*\*Running Average (移动平均值) \*\*:

批量归一化中用于测试阶段的统计量估算方法。训练时，对每个批次的均值/标准差采用“指数移动平均”（而非简单算术平均）进行累积，公式通常为：

```
`running_mean = momentum * running_mean + (1 - momentum) * batch_mean`
```

(CS231n作业中默认`momentum=0.99`或`0.9`），目的是降低单批次数据噪声对测试结果的影响。

3. \*\*Learnable Shift and Scale Parameters (可学习偏移与缩放参数) \*\*:

对应公式中的` $\beta$ `（偏移）和` $\gamma$ `（缩放），是批量归一化层的核心参数。若不引入这两个参数，归一化会强制特征分布固定，可能破坏前层学到的有用表征；通过学习` $\beta$ `和` $\gamma$ `，网络可自主调整：`output =  $\gamma$  \* normalized\_feature +  $\beta$ `，既保留了归一化的稳定性，又保留了特征分布的灵活性（极端情况下，` $\gamma=\sqrt{\text{方差}}$ `、` $\beta=\text{均值}$ `可完全抵消归一化效果）。

```
eps = bn_param.get("eps", 1e-5)
```

从`bn\_param`中获取"eps"参数，用于数值稳定性（避免计算标准差时除以零）。

如果`bn\_param`中未提供"eps"，则使用默认值`1e-5`（即`0.00001`）。

作用：在计算`1 / sqrt(var + eps)`时，`eps`确保分母不会因`var`接近`0`而过大，防止数值溢出或不稳定。

下面详细推导批量归一化反向传播中核心的梯度公式，从单个元素的梯度逐步推导到最终的  $\frac{\partial L}{\partial x_i}$ ，并说明简化思路。

## 符号定义

- $x_i$ : 输入数据的第  $i$  个样本 (批量大小为  $N$ , 即  $i = 1, 2, \dots, N$ )
- $\mu$ : 批量均值,  $\mu = \frac{1}{N} \sum_{k=1}^N x_k$
- $v$ : 批量方差,  $v = \frac{1}{N} \sum_{k=1}^N (x_k - \mu)^2$
- $\sigma$ : 标准差,  $\sigma = \sqrt{v + \epsilon}$  ( $\epsilon$  为防止除零的小常数)
- $y_i$ : 归一化后的值,  $y_i = \frac{x_i - \mu}{\sigma}$
- 上游梯度:  $\frac{\partial L}{\partial y_i} = dy_i$  (即代码中的 `dout[i]`)
- 目标: 求  $\frac{\partial L}{\partial x_i}$  (即 `dx[i]`)

## 分步梯度推导

根据链式法则,  $x_i$  对损失  $L$  的梯度需考虑所有依赖路径:

$$\frac{\partial L}{\partial x_i} = \frac{\partial L}{\partial y_i} \cdot \frac{\partial y_i}{\partial x_i} + \sum_{k=1}^N \frac{\partial L}{\partial y_k} \cdot \frac{\partial y_k}{\partial x_i} + \sum_{k=1}^N \frac{\partial L}{\partial \mu} \cdot \frac{\partial \mu}{\partial x_i} + \sum_{k=1}^N \frac{\partial L}{\partial \sigma} \cdot \frac{\partial \sigma}{\partial x_i} + \sum_{k=1}^N \frac{\partial L}{\partial v} \cdot \frac{\partial v}{\partial x_i}$$

(解释:  $x_i$  影响  $y_i$ 、 $\mu$ 、 $v$ , 而  $\mu$  和  $v$  又影响所有  $y_k$ , 因此需汇总所有路径的梯度)

### 1. 基础局部梯度

$$(1) \quad \frac{\partial \mu}{\partial x_i}$$

$\mu = \frac{1}{N}(x_1 + x_2 + \dots + x_N)$ , 直接求导:

$$\boxed{\frac{\partial \mu}{\partial x_i} = \frac{1}{N}}$$

$$(2) \quad \frac{\partial v}{\partial x_i}$$

$v = \frac{1}{N} \sum_{k=1}^N (x_k - \mu)^2$ , 展开后对  $x_i$  求导:

- 第一项:  $(x_i - \mu)^2$  对  $x_i$  的导数为  $2(x_i - \mu) \cdot \frac{\partial(x_i - \mu)}{\partial x_i}$

其中  $\frac{\partial(x_i - \mu)}{\partial x_i} = 1 - \frac{\partial \mu}{\partial x_i} = 1 - \frac{1}{N}$  (因  $\mu$  依赖  $x_i$ )

- 其他项 ( $k \neq i$ ):  $(x_k - \mu)^2$  对  $x_i$  的导数为  $2(x_k - \mu) \cdot (-\frac{\partial \mu}{\partial x_i}) = -2(x_k - \mu) \cdot \frac{1}{N}$

汇总后简化 (利用  $\sum_{k=1}^N (x_k - \mu) = 0$ , 因均值的定义) :

$$\frac{\partial v}{\partial x_i} = \frac{2}{N} (x_i - \mu) \cdot \left(1 - \frac{1}{N}\right) + \frac{2}{N} \cdot \frac{1}{N} \sum_{k \neq i} -(x_k - \mu) = \frac{2}{N^2} (x_i - \mu) \cdot (N-1) - \frac{2}{N^2} \sum_{k \neq i} (x_k - \mu)$$

进一步简化 (因  $\sum_{k \neq i} (x_k - \mu) = -(x_i - \mu)$ ) :

$$\boxed{\frac{\partial v}{\partial x_i} = \frac{2}{N} (x_i - \mu)}$$

(3)  $\frac{\partial \sigma}{\partial v}$

$\sigma = \sqrt{v + \epsilon} = (v + \epsilon)^{1/2}$ , 求导:

$$\boxed{\frac{\partial \sigma}{\partial v} = \frac{1}{2} (v + \epsilon)^{-1/2} = \frac{1}{2\sigma}}$$

(4)  $\frac{\partial y_k}{\partial \mu}$  和  $\frac{\partial y_k}{\partial \sigma}$

$y_k = \frac{x_k - \mu}{\sigma}$ , 分别对  $\mu$  和  $\sigma$  求导:

$$\boxed{\frac{\partial y_k}{\partial \mu} = -\frac{1}{\sigma}} \quad (\text{因分子对 } \mu \text{ 求导为 } -1)$$

$$\boxed{\frac{\partial y_k}{\partial \sigma} = -\frac{x_k - \mu}{\sigma^2}} \quad (\text{因分母对 } \sigma \text{ 求导为 } -(x_k - \mu) \cdot \sigma^{-2})$$

(5)  $\frac{\partial y_i}{\partial x_i}$

$y_i$  直接依赖  $x_i$ , 求导:

$$\frac{\partial y_i}{\partial x_i} = \frac{1}{\sigma} \cdot \frac{\partial (x_i - \mu)}{\partial x_i} = \frac{1}{\sigma} \cdot \left(1 - \frac{1}{N}\right) \quad (\text{利用 } \frac{\partial \mu}{\partial x_i} = \frac{1}{N})$$

$$\boxed{\frac{\partial y_i}{\partial x_i} = \frac{N-1}{N\sigma}}$$

## 2. 汇总所有路径的梯度

将上述局部梯度代入链式法则公式, 逐步合并:

(1) 来自  $y_i$  直接依赖  $x_i$  的梯度

$$\frac{\partial L}{\partial y_i} \cdot \frac{\partial y_i}{\partial x_i} = dy_i \cdot \frac{N-1}{N\sigma}$$

## (2) 来自所有 $y_k$ 依赖 $\mu$ 的梯度

$$\sum_{k=1}^N dy_k \cdot \frac{\partial y_k}{\partial \mu} \cdot \frac{\partial \mu}{\partial x_i} = \sum_{k=1}^N dy_k \cdot \left(-\frac{1}{\sigma}\right) \cdot \frac{1}{N} = -\frac{1}{N\sigma} \sum_{k=1}^N dy_k$$

## (3) 来自所有 $y_k$ 依赖 $\sigma$ 和 $v$ 的梯度

$$\sum_{k=1}^N dy_k \cdot \frac{\partial y_k}{\partial \sigma} \cdot \frac{\partial \sigma}{\partial v} \cdot \frac{\partial v}{\partial x_i}$$

代入局部梯度：

$$= \sum_{k=1}^N dy_k \cdot \left(-\frac{x_k - \mu}{\sigma^2}\right) \cdot \frac{1}{2\sigma} \cdot \frac{2}{N} (x_i - \mu)$$

化简（约去 2）：

$$= -\frac{(x_i - \mu)}{N\sigma^3} \sum_{k=1}^N dy_k \cdot (x_k - \mu)$$

## 3. 最终简化公式

将三部分梯度相加，得到  $\frac{\partial L}{\partial x_i}$ ：

\$

$$\boxed{\frac{\partial L}{\partial x_i} = \frac{1}{\sigma} \left( dy_i \cdot \frac{N-1}{N} - \frac{1}{N} \sum_{k=1}^N dy_k - \frac{(x_i - \mu)}{N\sigma^2} \sum_{k=1}^N dy_k \cdot (x_k - \mu) \right)}$$

## 向量化实现优势

上述公式可通过向量运算快速实现：

- 预计算  $\sum dy_k$  (记为 `sum_dy`) 和  $\sum dy_k \cdot (x_k - \mu)$  (记为 `sum_dy_xmu`)
- 利用广播机制对所有  $x_i$  一次性计算，避免循环，大幅提升效率。

这就是简化版反向传播 (`batchnorm_backward_alt`) 的核心数学基础，与原始实现结果一致但计算步骤更少。

\$\$\$



编辑 分享 ...

这里的“这样”指的是\*\*批量归一化的简化版反向传播实现\*\*，它之所以“更好、更快”，核心源于\*\*数学推导的优化\*\*和\*\*计算效率的提升\*\*，具体可从以下维度分析：

### 一、“更快”的底层逻辑：减少冗余计算

简化版反向传播通过\*\*合并中间梯度的推导步骤\*\*，避免了重复计算和不必要的内存开销：

1. \*\*数学公式的合并简化\*\*：

原始实现中需要分别计算 `dvar`、`dmean` 等中间梯度，再通过多个分支汇总得到 `dx`。而简化版通过\*\*单步推导直接得到 `dx` 的紧凑表达式\*\*，减少了中间变量的存储和运算次数。

例如，原始实现中对 `σ`（标准差）的幂次（如 `σ<sup>3</sup>`）需要多次计算，简化版通过代数变形将其合并为更简洁的形式，降低了浮点运算的复杂度。

## 2. \*\*向量化运算的高效性\*\*:

简化版的推导更贴合\*\*向量化计算的特性\*\*，可以充分利用 NumPy 等框架的矩阵运算优化（如广播机制、SIMD 指令加速）。相比原始实现中对多个中间变量的分步处理，简化版的向量运算更“紧凑”，减少了 CPU/GPU 指令的调用次数，从而提升运行速度。

## ### 二、“更好”的深层原因：稳定性与可维护性

### 1. \*\*数值稳定性的提升\*\*:

简化版通过\*\*更简洁的公式推导\*\*，减少了因多次浮点运算（如重复开根号、幂次计算）导致的数值误差累积。例如，原始实现中 `(sample\_var + eps) \*\* -1.5` 这类操作在简化版中被合并为更稳定的形式，降低了因数值精度问题导致的梯度异常风险。

### 2. \*\*代码可读性与可维护性\*\*:

简化版的推导基于\*\*更直观的链式法则拆解\*\*（从单个元素 `x\_i` 到整体梯度的推导），代码逻辑更清晰，便于后续的调试、扩展和维护。相比之下，原始实现的中间变量较多，逻辑分支更复杂，容易在修改时引入 bug。

### ### 三、与原始实现的对比验证

尽管实现方式不同，但简化版和原始版的\*\*数学原理完全等价\*\*，因此两者计算的梯度结果几乎完全一致（误差在 $10^{-10}$ 量级以内）。这意味着简化版在保证结果正确性的前提下，实现了效率的提升，属于“既快又准”的优化。

综上，批量归一化的简化版反向传播通过\*\*数学推导的优化\*\*和\*\*计算流程的精简\*\*，实现了“更快”的运行效率和“更好”的数值稳定性与可维护性，是深度学习框架中批量归一化层的典型优化方向。

## 有无BN的可视化

### ### 问题1：

描述本实验的结果。权重初始化尺度对含/不含批量归一化（BN）的模型影响有何不同？原因是什么？

### ### 答案：

本实验揭示了权重初始化尺度对含/不含批量归一化（BN）模型的显著差异影响，核心结论如下：

#### 1. \*\*对不含BN的影响\*\*

- 有效尺度范围狭窄：仅当权重尺度适中时（例如  $(10^{-2} \sim 10^{-1})$  左右），模型才能正常训练。
- 极端尺度下训练失效：
  - 尺度过小时（如  $<10^{-3}$ ）：初始激活值趋近于0，导致梯度消失。模型无法学习，训练损失停滞不前，准确率始终偏低。
  - 尺度过大时（如  $>10^{-1}$ ）：初始激活值急剧膨胀，引发梯度爆炸。训练过程不稳定，损失可能发散或持续处于高位。

#### 2. \*\*对含BN的影响\*\*

- 鲁棒尺度范围宽广：在实验测试的全部尺度区间 ( $10^{-4} \sim 1$ ) 内，模型均能稳定训练。即便使用极小或极大的权重尺度，训练损失仍能平稳下降，准确率可达到较高水平。
- 对尺度敏感性低：模型性能（训练/验证准确率、损失）随权重尺度变化的波动极小，表明BN将模型训练与初始化尺度解耦。

#### 3. \*\*差异的根本原因\*\*

- 不含BN的模型：权重尺度不当会导致各层输入分布剧烈漂移（即内部协变量偏移）。极端的激活值会破坏梯度传播过程，使网络无法正常训练。
- 含BN的模型：BN通过在每个批次中对层输入进行归一化，将其调整为稳定分布（均值 $\approx 0$ 、方差 $\approx 1$ ）。再通过缩放（Y）和偏移（ $\beta$ ）操作抵消极端权重尺度的影响，确保梯度始终处于合理范围。这一机制稳定了训练过程，降低了模型对权重初始化的依赖。

### ### 问题2：

描述本实验的结果。这对批量归一化和批量大小之间的关系意味着什么？为什么会观察到这种关系？

### ### 答案：

实验结果及分析如下：

#### 1. \*\*实验结果\*\*

- \*\*训练准确率\*\*：含批量归一化（BN）的模型中，随着批量大小增大（从5到50），训练准确率显著提升。批量大小为50的BN模型(`with\_norm50`)训练准确率最高，而批量大小为5的BN模型(`with\_norm5`)最低。无BN的基准模型（批量大小5）表现优于所有BN模型。
  - \*\*验证准确率\*\*：BN模型整体优于基准模型。在BN模型中，批量大小50和10的模型验证准确率更高且更稳定，而批量大小5的模型验证准确率波动更大，最终精度更低。

## 2. \*\*对批量归一化与批量大小关系的启示\*\*

在一定范围内，批量归一化的性能与批量大小呈\*\*正相关\*\*。更大的批量大小有助于BN模型获得更好的训练效果，且在多数情况下能提升验证性能。

## 3. \*\*现象背后的原因\*\*

批量归一化依赖\*\*批次统计量（均值和方差）\*\* 对层输入进行归一化。更大的批量大小能提供更具代表性、更稳定的统计量估计，减少归一化过程中的噪声，从而使梯度传播更稳定，模型学习更充分。相反，小批量（如5）会导致批次统计量噪声较大，削弱BN稳定训练的能力，造成收敛速度变慢、精度下降。无BN的基准模型不依赖批次统计量，因此其性能不受批量大小的直接影响（与BN模型的规律不同）。

### ### 层归一化 (Layer Normalization)

批量归一化 (Batch Normalization) 已被证实能有效降低网络训练难度，但它对批量大小的依赖性，使其在复杂网络中适用性受限——这类网络常因硬件限制而无法使用较大的输入批量。

为缓解这一问题，研究人员提出了多种批量归一化的替代方案，层归一化 (Layer Normalization) [2] 便是其中之一。与“基于批量维度进行归一化”不同，层归一化的归一化维度是\*\*特征维度\*\*。具体来说，使用层归一化时，单个数据样本对应的特征向量，会根据该向量内部所有元素的统计信息（均值、方差）进行归一化处理。

[2] 参考文献: Ba, Jimmy Lei, Jamie Ryan Kiros, and Geoffrey E. Hinton. "Layer Normalization." *stat* 1050 (2016): 21.

### ### 补充说明

1. \*\*核心差异\*\*：批量归一化是对“同一特征在不同样本上的取值”做归一化（跨样本、同特征），而层归一化是对“同一样本在不同特征上的取值”做归一化（跨特征、同样本）。
2. \*\*优势适配场景\*\*：层归一化不依赖批量大小，即使批量为1（如在线学习、生成模型采样阶段）仍能稳定工作，尤其适合硬件资源有限、批量大小受限的复杂网络。
3. \*\*文献背景\*\*：该方法由谷歌大脑团队于2016年提出，初衷是解决循环神经网络 (RNN) 训练中批量归一化适配性差的问题，后续也被广泛应用于Transformer等深度学习模型中。

### ### 问题翻译

#### \*\*内联问题3:\*\*

以下哪些数据预处理步骤与批量归一化类似，哪些与层归一化类似？

- 对数据集中的每张图像进行缩放，使图像内每一行像素的RGB通道之和为1。
- 对数据集中的每张图像进行缩放，使图像内所有像素的RGB通道之和为1。
- 从数据集中的每张图像中减去数据集的平均图像。
- 根据给定阈值将所有RGB值设为0或1。

#### \*\*答案:\*\*

[填写内容]

### ### 答案

- \*\*与批量归一化类似\*\*：从数据集中的每张图像中减去数据集的平均图像 (\*Subtracting the mean image of the dataset from each image in the dataset\*)。

解释：批量归一化是基于\*\*整个批量的统计量\*\*（均值、方差）对每个样本的特征进行归一化，类似地，“减去数据集的平均图像”是基于\*\*整个数据集的统计量（平均图像）\*\*对每个样本（图像）进行处理。

- \*\*与层归一化类似\*\*：对数据集中的每张图像进行缩放，使图像内每一行像素的RGB通道之和为1 (\*Scaling each image in the dataset, so that the RGB channels for each row of pixels within an image sums up to 1\*)。

解释：层归一化是基于\*\*单个样本的特征维度统计量\*\*对该样本的特征进行归一化，类似地，“使图像内每一行像素的RGB通道之和为1”是基于\*\*单个图像内的行维度（局部特征维度）统计量\*\*对该行的RGB通道进行处理。

- 其余步骤不类似：

- “使图像内所有像素的RGB通道之和为1”是基于\*\*单个样本的全局统计量\*\*，但操作粒度与层归一化的“特征维度”不完全对齐；
- “根据阈值二值化RGB值”属于离散化操作，与归一化的统计量调整逻辑无关。

(chatgpt) 我们逐条分析这些操作与 \*\*批量归一化 (Batch Normalization, BN)\*\* 和 \*\*层归一化 (Layer Normalization, LN)\*\* 的相似性。

---

### ◆ 回顾两种归一化的区别

归一化方法	计算均值/方差的范围
特点	
Batch Normalization	在一个batch中，对同一通道的所有样本取均值与方差（即 across batch & spatial dims, per channel）
Layer Normalization	在单一样本内，在所有通道/特征取均值与方差 与batch大小无关，每个样本独立

---

### ◆ 选项分析

#### 1 「对数据集中的每张图像进行缩放，使图像内每一行像素的RGB通道之和为1」

- \* 操作范围：单张图像内的\*\*每一行\*\*
- \* 归一化的统计量来自\*\*单一样本（每张图）内部\*\*
- \*\*更接近层归一化 (Layer Normalization) \*\*

> 理由：每行相当于图像的一个“层”，在样本内部进行局部归一化，不依赖batch。

---

#### 2 「对数据集中的每张图像进行缩放，使图像内所有像素的RGB通道之和为1」

- \* 操作范围：单张图像内的\*\*全局像素\*\*
- \* 归一化的统计量仍然是从\*\*单一样本内部\*\*计算
- \*\*更接近层归一化 (Layer Normalization) \*\*

> 理由：与LN类似，对每个样本整体归一化，不参考其他样本。

---

#### 3 「从数据集中的每张图像中减去数据集的平均图像」

- \* 操作范围：基于\*\*整个数据集\*\*统计的均值
- \* 每张图像都减去\*\*全局均值图像\*\*
- \*\*更接近批量归一化 (Batch Normalization) \*\*

> 理由：这相当于使用\*\*所有样本的统计信息\*\*来规范化每个样本，符合BN的思想（BN利用batch统计量）。

---

#### 4 「根据给定阈值将所有RGB值设为0或1」

\* 操作范围：非归一化操作，而是\*\*离散化 / 二值化\*\*

✗ \*\*既不像Batch Normalization也不像Layer Normalization\*\*

> 理由：这是阈值化（thresholding），与均值方差归一无关。

---

### ✓ 总结对照表

数据预处理步骤	类似于	理由
每行像素RGB通道之和为1	层归一化 (LN)	每张图内部局部归一化
整张图像RGB通道之和为1	层归一化 (LN)	每张图内部全局归一化
减去数据集平均图像	批量归一化 (BN)	使用全体样本统计信息
RGB值设为0或1	都不像	离散化操作，不是归一化

---

如果想更贴近类比关系，可以记：

> \* “\*\*跨样本求统计量\*\*” → 类似 \*\*BatchNorm\*\*

> \* “\*\*在单样本内部求统计量\*\*” → 类似 \*\*LayerNorm\*\*

\*\*层归一化：实现部分（Layer Normalization: Implementation）\*\*

现在你将实现层归一化（Layer Normalization）。

这一步相对比较直接，因为在概念上，它的实现几乎与批量归一化（Batch Normalization）相同。

不过有一个重要区别：

对于层归一化，我们\*\*不需要维护滑动平均的均值与方差\*\*（moving moments），

而且\*\*测试阶段与训练阶段完全相同\*\*—均值和方差都是针对每个样本（datapoint）直接计算的。

---

### 你需要完成的任务：

1. 在 `cs231n/layers.py` 中，

实现层归一化的前向传播函数 `layernorm\_forward`。

✓ 实现后，运行下方的测试单元（cell），检查你的结果。

---

2. 在 `cs231n/layers.py` 中，

实现层归一化的反向传播函数 `layernorm\_backward`。

实现后，运行第二个测试单元，检查结果是否正确。

---

3. 修改 `cs231n/classifiers/fc\_net.py` 文件，  
使其在构造函数中当 `normalization` 参数被设置为 `"layernorm"` 时，  
在每个 ReLU 激活函数 (ReLU nonlinearity) \*\*之前\*\* 插入一个层归一化层。

完成后，运行第三个测试单元，以执行关于层归一化的批量大小实验 (batch size experiment)。

---

💡 \*\*总结：\*\*

- \* 层归一化 (LayerNorm) 与批量归一化 (BatchNorm) 非常相似；
- \* 不同点：不依赖批量统计量，不使用滑动平均；
- \* 实现时关键是对每个样本的特征维度计算均值和方差；
- \* 测试阶段与训练阶段一致。

这与归一化的目标和参数的作用直接相关：

#### ### 1. 均值和标准差的参数个数：与\*\*样本数量 (N)\*\* 相关

- \*\*原因\*\*：均值和标准差是\*\*针对每个样本单独计算的统计量\*\*（层归一化的核心逻辑）。  
层归一化的目的是让单个样本的特征分布更稳定，因此需要为每个样本（共N个样本）计算其自身的特征均值和标准差（每个样本对应1个均值和1个标准差）。

例如，输入数据形状为(N, D)时，均值和标准差的形状均为(N, 1)（每个样本1个值），总个数均为N。

#### ### 2. Gamma和Beta的参数个数：与\*\*特征维度 (D)\*\* 相关

- \*\*原因\*\*：Gamma和Beta是\*\*针对每个特征维度的可学习参数\*\*，用于恢复归一化后的数据表达能力。  
归一化操作会消除特征原有的尺度信息（均值为0、方差为1），因此需要为每个特征维度（共D个特征）分配一个Gamma（缩放）和Beta（偏移），以允许模型自主学习最优的特征尺度和偏移量。

例如，输入数据形状为(N, D)时，Gamma和Beta的形状均为(D,)，总个数均为D。

简言之：

- 均值/标准差是\*\*按样本计算的统计量\*\*，数量随样本数 (N) 变化；
- Gamma/Beta是\*\*按特征维度设计的可学习参数\*\*，数量随特征维度 (D) 固定。

#### ### 问题4：

层归一化在哪些情况下可能效果不佳？原因是什么？

(选项：在极深的网络中使用、特征维度非常小、使用高正则化项)

#### ### 答案：

层归一化最可能在\*\*特征维度非常小\*\*的情况下效果不佳，其余两种情况并非主要问题，具体分析如下：

#### #### 1. 核心不佳场景：特征维度非常小

- \*\*原因\*\*：层归一化依赖单个样本的特征维度计算均值和方差（统计量基于特征维度）。若特征维度极小（如D=2、3），有限的特征数量会导致统计量估计极不稳定、噪声大。

- 具体影响：不稳定的均值/方差会让归一化后的特征分布波动剧烈，反而破坏输入数据的原有规律，导致梯度传播混乱，模型无法有效学习。

#### #### 2. 非主要问题场景

- \*\*在极深的网络中使用\*\*：层归一化本身能稳定各层输入分布，缓解深层网络的梯度消失/爆炸问题，反而适合极深网络（如Transformer的深层编码器/解码器），不会因网络深而效果变差。
- \*\*使用高正则化项\*\*：正则化（如L2正则）的作用是惩罚大权重、防止过拟合，与层归一化的“稳定训练”逻辑不冲突。高正则化项可能会降低模型整体拟合能力，但并非针对层归一化的特异性影响，不会导致其单独效果不佳。

## dropout

- 除以p保证期望输出正确
- 简言之，Iteration 是“一次参数更新”，Epoch 是“一次完整的训练集遍历”，前者是后者的组成单元。
  - lr\_decay: 学习率衰减的标量；每个 epoch 后学习率乘以该值。
  - batch\_size: 训练期间用于计算损失和梯度的小批次大小。
  - num\_epochs: 训练期间运行的 epoch 数。

## CNN

同样，前向传播和反向传播都要检查

快速实现调api

(封装) 卷积激活池化

参数初始化

写好loss()的两种模式

损失、梯度检查

过拟合小数据集（多个epoch，迭代次数少）

#### 核心逻辑与目的

过拟合原理：少量数据的特征有限，模型（即使是简单的三层卷积网）能“死记硬背”所有训练样本的模式，导致在训练集上几乎无错误，但无法泛化到未见过的验证集。

验证价值：

若能成功过拟合，说明模型的前向 / 反向传播、参数更新逻辑完全正确（模型有足够的能力拟合数据）。

若无法过拟合（训练准确率低），则需排查：学习率设置不当、梯度计算错误、网络容量不足（如滤波器数量过少）等问题。

```

(Iteration 27 / 30) loss: 0.123754
(Iteration 28 / 30) loss: 0.172242
(Epoch 14 / 15) train acc: 0.990000; val_acc: 0.222000
(Iteration 29 / 30) loss: 0.123923
(Iteration 30 / 30) loss: 0.070897
(Epoch 15 / 15) train acc: 0.990000; val_acc: 0.218000

[16]: # Print final training accuracy.
print(
    "Small data training accuracy:",
    solver.check_accuracy(small_data['X_train'], small_data['y_train'])
)
Small data training accuracy: 0.81

[17]: # Print final validation accuracy.
print(
    "Small data validation accuracy:",
    solver.check_accuracy(small_data['X_val'], small_data['y_val'])
)
Small data validation accuracy: 0.248

```

- 训练过程中展示的是当前epoch的准确率，而判断最终过拟合需要使用最后训练完成的参数对整个数据共同进行计算准确率

训练一个epoch，使用完整数据集，迭代次数多

核心目标：验证网络在完整训练流程下的性能，确保前向传播、反向传播、参数更新等环节无错误（若准确率低于 40%，需排查模型实现、学习率、正则化等问题）。

空间批归一化（卷积中用）：沿C通道特征维度

空间组归一化：沿通道维度分组。简言之，组归一化是一种兼顾“批归一化对卷积层的适配性”和“层归一化对小批量的鲁棒性”的技术，通过通道分组实现了在小批量场景下的稳定归一化，是卷积网络归一化的重要替代方案。

## pytorch

### 基本使用

数据加载：

transform为一串数据操作单元

sampler.SubsetRandomSampler()按批次抽取

每次迭代loader时，这 1000 个验证样本会被随机打乱顺序后组成批量（batch）

```

import torchvision.datasets as dset
import torchvision.transforms as T

transform = T.Compose([
    T.ToTensor(),
    T.Normalize((0.4914, 0.4822, 0.4465), (0.2023, 0.1994, 0.2010))
])

cifar10_train = dset.CIFAR10('./cs231n/datasets', train=True, download=True,
                             transform=transform)
loader_train = DataLoader(cifar10_train, batch_size=64,

```

```

        sampler=sampler.SubsetRandomSampler(range(NUM_TRAIN)))

cifar10_val = dset.CIFAR10('./cs231n/datasets', train=True, download=True,
                           transform=transform)
loader_val = DataLoader(cifar10_val, batch_size=64,
                        sampler=sampler.SubsetRandomSampler(range(NUM_TRAIN, 50000)))

cifar10_test = dset.CIFAR10('./cs231n/datasets', train=False, download=True,
                           transform=transform)
loader_test = DataLoader(cifar10_test, batch_size=64)

```

卷积

权重初始化要: w.requires\_grad = True

```
import torch.nn.functional as F
```

准确率计算:

```

def check_accuracy_part2(loader, model_fn, params):
    """
    Check the accuracy of a classification model.

    Inputs:
    - loader: A DataLoader for the data split we want to check
    - model_fn: A function that performs the forward pass of the model,
      with the signature scores = model_fn(x, params)
    - params: List of PyTorch Tensors giving parameters of the model

    Returns: Nothing, but prints the accuracy of the model
    """
    split = 'val' if loader.dataset.train else 'test'
    print('Checking accuracy on the %s set' % split)
    num_correct, num_samples = 0, 0
    with torch.no_grad():
        for x, y in loader:
            x = x.to(device=device, dtype=dtype) # move to device, e.g. GPU
            y = y.to(device=device, dtype=torch.int64)
            scores = model_fn(x, params)
            _, preds = scores.max(1)
            num_correct += (preds == y).sum()
            num_samples += preds.size(0)
    acc = float(num_correct) / num_samples
    print('Got %d / %d correct (%.2f%%)' % (num_correct, num_samples, 100 * acc))

```

模型训练: 自动跑一个epoch

```

def train_part2(model_fn, params, learning_rate):
    """
    在 CIFAR-10 数据集上训练模型。
    """

```

输入:

- **model\_fn**: 执行模型前向传播的 Python 函数。  
函数签名为 `scores = model_fn(x, params)`, 其中 `x` 是图像数据的 PyTorch 张量, `params` 是给出模型权重的 PyTorch 张量列表, `scores` 是形状为 (`N, C`) 的 PyTorch 张量, 给出 `x` 中每个元素的分类得分。
- **params**: 给出模型权重的 PyTorch 张量列表
- **learning\_rate**: Python 标量, 给出 SGD 优化器使用的学习率

返回: 无返回值

```

"""
# 遍历训练集数据加载器, t 为迭代次数(从 0 开始), (x, y) 为批量数据和标签
for t, (x, y) in enumerate(loader_train):
    # 将数据迁移到合适的设备(GPU 或 CPU)
    x = x.to(device=device, dtype=dtype)
    y = y.to(device=device, dtype=torch.long)

    # 前向传播: 计算分类得分和损失
    scores = model_fn(x, params)
    loss = F.cross_entropy(scores, y)

    # 反向传播: PyTorch 会自动识别计算图中 requires_grad=True 的张量,
    # 通过反向传播计算损失相对于这些张量的梯度, 并将梯度存储在每个张量的 .grad 属性中。
    loss.backward()

    # 更新参数。我们不希望通过参数更新过程进行反向传播,
    # 因此将更新过程包裹在 torch.no_grad() 上下文管理器中, 避免构建计算图。
    with torch.no_grad():
        for w in params:
            # SGD 参数更新: 权重 = 权重 - 学习率 * 梯度
            w -= learning_rate * w.grad

            # 反向传播后手动清零梯度(避免梯度累计)
            w.grad.zero_()

    # 每 print_every 次迭代, 打印损失并评估验证集准确率
    if t % print_every == 0:
        print('Iteration %d, loss = %.4f' % (t, loss.item()))
        check_accuracy_part2(loader_val, model_fn, params)
        print()

```

## nn.Module

`torch.optim` 提供求解

- 模型设计: `self.fc1 = nn.Linear(input_size, hidden_size)` 这样之后自动包含 weight 和 bias

```

class TwoLayerFC(nn.Module):
    def __init__(self, input_size, hidden_size, num_classes):
        super().__init__()
        # 将层对象分配给类属性
        self.fc1 = nn.Linear(input_size, hidden_size)
        # nn.init 包含便捷的初始化方法
        # http://pytorch.org/docs/master/nn.html#torch-nn-init
        nn.init.kaiming_normal_(self.fc1.weight)

```

```

        self.fc2 = nn.Linear(hidden_size, num_classes)
        nn.init.kaiming_normal_(self.fc2.weight)

    def forward(self, x):
        # forward 方法始终定义网络连接方式
        x = flatten(x)
        scores = self.fc2(F.relu(self.fc1(x)))
        return scores

def test_TwoLayerFC():
    input_size = 50
    # 批量大小 64, 特征维度 50 的零张量
    x = torch.zeros((64, input_size), dtype=dtype)
    model = TwoLayerFC(input_size, 42, 10)
    scores = model(x)
    print(scores.size()) # 你应该看到 [64, 10]
test_TwoLayerFC()

```

- 模型训练

```

def train_part34(model, optimizer, epochs=1):
    """
    使用 PyTorch Module API 在 CIFAR-10 数据集上训练模型。
    输入:
    - model: 待训练的 PyTorch Module 模型
    - optimizer: 用于训练模型的 Optimizer (优化器) 对象
    - epochs: (可选) Python 整数, 指定训练的轮次
    返回: 无返回值, 但会在训练过程中打印模型准确率
    """

    # 将模型参数迁移到指定设备 (CPU 或 GPU)
    model = model.to(device=device)
    # 遍历训练轮次
    for e in range(epochs):
        # 遍历训练集数据加载器, t 为迭代次数, (x, y) 为批量数据和标签
        for t, (x, y) in enumerate(loader_train):
            # 将模型设置为训练模式
            model.train()
            # 将输入数据迁移到指定设备, 并转换为指定数据类型
            x = x.to(device=device, dtype=dtype)
            # 将标签迁移到指定设备, 转换为长整数类型
            y = y.to(device=device, dtype=torch.long)

            # 前向传播: 计算分类得分
            scores = model(x)
            # 计算交叉熵损失
            loss = F.cross_entropy(scores, y)

            # 清零优化器要更新的所有变量的梯度
            optimizer.zero_grad()

            # 反向传播: 计算损失相对于模型每个参数的梯度

```

```
loss.backward()

# 使用反向传播计算的梯度，实际更新模型参数
optimizer.step()

# 每 print_every 次迭代，打印损失并评估验证集准确率
if t % print_every == 0:
    print('Iteration %d, loss = %.4f' % (t, loss.item()))
    check_accuracy_part34(loader_val, model)
    print()
```

- 最终操作

```
hidden_layer_size = 4000
learning_rate = 1e-2
model = TwoLayerFC(3 * 32 * 32, hidden_layer_size, 10)
optimizer = optim.SGD(model.parameters(), lr=learning_rate)

train_part34(model, optimizer)
```

## PyTorch Sequential API

前面介绍了 PyTorch Module API，它允许你定义任意可学习层及其连接方式。

对于像“堆叠前馈层”这样的简单模型，你仍需执行三个步骤：继承 nn.Module、在 `init` 中为类属性分配层、在 `forward()` 中逐个调用每层。有没有更便捷的方法？

幸运的是，PyTorch 提供了一个名为 nn.Sequential 的容器模块，它将上述步骤合并为一步。它不如 nn.Module 灵活——无法定义超出前馈堆叠的复杂拓扑结构，但足以满足许多使用场景。

```
# 我们需要将 `flatten` 函数包装在一个模块中，以便在 nn.Sequential 中堆叠使用
class Flatten(nn.Module):
    def forward(self, x):
        return flatten(x)

hidden_layer_size = 4000 # 隐藏层大小
learning_rate = 1e-2      # 学习率

model = nn.Sequential(
    Flatten(), # 展平层：将图像张量转换为一维特征
    nn.Linear(3 * 32 * 32, hidden_layer_size), # 第一层全连接层
    nn.ReLU(), # ReLU激活函数
    nn.Linear(hidden_layer_size, 10), # 第二层全连接层（输出层）
)
# 可以在 optim.SGD 中使用 Nesterov 动量
optimizer = optim.SGD(model.parameters(), lr=learning_rate,
                      momentum=0.9, nesterov=True)

# 训练模型
train_part34(model, optimizer)
#####
```

```

model = nn.Sequential(
    nn.Conv2d(3, channel_1, kernel_size=5, padding=2),
    nn.ReLU(),
    nn.Conv2d(channel_1, channel_2, kernel_size=3, padding=1),
    nn.ReLU(),
    Flatten(),
    nn.Linear(channel_2*32*32, 10)
)
optimizer = optim.SGD(model.parameters(), lr=learning_rate,
                      momentum=0.9, nesterov=True)

```

## RNN

检查数据：了解数据可以更好的编写代码

解耦合：先写单层，再多层嵌套

- 词嵌入的前向传播

```

def word_embedding_forward(x, w):
    """Forward pass for word embeddings.

    We operate on minibatches of size N where
    each sequence has length T. We assume a vocabulary of V words, assigning each
    word to a vector of dimension D.

    Inputs:
    - x: Integer array of shape (N, T) giving indices of words. Each element idx
      of x must be in the range 0 <= idx < V.
    - w: Weight matrix of shape (V, D) giving word vectors for all words.

    Returns a tuple of:
    - out: Array of shape (N, T, D) giving word vectors for all input words.
    """
    out = None
    #########################################################################
    # TODO: Implement the forward pass for word embeddings.                #
    #                                                                       #
    # HINT: This can be done in one line using Pytorch's array indexing.   #
    #########################################################################
    out = w[x]
    #########################################################################
    #                                                                       #
    # END OF YOUR CODE                                                       #
    #########################################################################
    return out

```

- `sample` 方法（测试时生成描述）

# 第十二讲：自监督学习

Summary: pretext tasks from image transformations

- Pretext tasks focus on “visual common sense”, e.g., predict rotations, inpainting, rearrangement, and colorization.
- The models are forced to learn good features about natural images, e.g., semantic representation of an object category, in order to solve the pretext tasks.
- We often do not care about the performance of these pretext tasks, but rather how useful the learned features are for downstream tasks (classification, detection, segmentation).
- Problems: 1) coming up with individual pretext tasks is tedious, and 2) the learned representations may not be general.

摘要：来自图像变换的 pretext 任务

- pretext 任务侧重于“视觉常识”，例如预测旋转、图像修复、重新排列和上色。
- 为了解决 pretext 任务，模型被迫学习自然图像的良好特征，例如物体类别的语义表示。
- 我们通常并不关心这些 pretext 任务的表现，而是关注所学到的特征对下游任务（分类、检测、分割）有多大用处。
- 问题：1) 设计单独的 pretext 任务十分繁琐，2) 所学的表征可能缺乏通用性。

进而引出对比表示学习，提高通用性

## (二) 对比表征学习

- 核心思路：通过对正负样本对，让模型学习“同类相近、异类远离”的特征，采用 InfoNCE 损失函数（互信息的下界）。
- 代表性模型：
  - SimCLR：通过数据增强生成正样本对，使用非线性投影头提升特征灵活性，需大批次训练（依赖 TPU）。
  - MoCo (v1/v2)：引入动量编码器和样本队列，解耦批次大小与负样本数量，MoCo v2 融合 SimCLR 的强数据增强和非线性投影头，效率更优。
  - CPC (对比预测编码)：面向序列数据，通过自回归模型总结上下文，预测未来序列特征，适用于音频、图像补丁序列等。
  - DINO：无标签自蒸馏框架，通过学生网络与动量更新的教师网络对比学习，自动学习类别特异性特征，支持无监督目标分割。

## Summary: Contrastive Representation Learning

A general formulation for contrastive learning:

$$\text{score}(f(x), f(x^+)) >> \text{score}(f(x), f(x^-))$$

InfoNCE loss: N-way classification among positive and negative samples

$$L = -\mathbb{E}_X \left[ \log \frac{\exp(s(f(x), f(x^+)))}{\exp(s(f(x), f(x^+)) + \sum_{j=1}^{N-1} \exp(s(f(x), f(x_j^-)))} \right]$$

Commonly known as the InfoNCE loss (van den Oord et al., 2018)

A lower bound on the mutual information between  $f(x)$  and  $f(x^+)$

$$MI[f(x), f(x^+)] - \log(N) \geq -L$$

## 摘要：对比表示学习

对比学习的通用公式：

$$\text{score}(f(x), f(x^+)) >> \text{score}(f(x), f(x^-))$$

InfoNCE 损失：正负样本间的 N 分类

$$L = -\mathbb{E}_X \left[ \log \frac{\exp(s(f(x), f(x^+)))}{\exp(s(f(x), f(x^+)) + \sum_{j=1}^{N-1} \exp(s(f(x), f(x_j^-)))} \right]$$

通常被称为 InfoNCE 损失 (van den Oord 等人, 2018)

$f(x)$  与  $f(x^+)$  之间互信息的下界

$$MI[f(x), f(x^+)] - \log(N) \geq -L$$

Stanford CS231n 10<sup>th</sup> Anniversary

Lecture 12 - 107

May 8, 2025

斯坦福大学 CS231n 十周年纪念

第 12 讲 - 107

2025 年 5 月 8 日

- 对比表征学习比单一图像变换任务更具通用性，InfoNCE 损失是核心支撑。

# 第十三讲：生成模型（一）

## Generative vs Discriminative Models

**Discriminative Model:**  
Learn a probability distribution  $p(y|x)$

→ Assign labels to data  
Feature learning (with labels)

**Generative Model:**  
Learn a probability distribution  $p(x)$

→ Detect outliers  
Feature learning (without labels)  
**Sample** to generate new data

**Conditional Generative Model:** Learn  $p(x|y)$

→ Assign labels while rejecting outliers  
**Sample** to generate data from labels

Stanford CS231n 10<sup>th</sup> Anniversary

Lecture 13 - 35

May 15, 2025

## Generative vs Discriminative Models

**Discriminative Model:**  
Learn a probability distribution  $p(y|x)$

→ Assign labels to data  
Feature learning (with labels)

**Generative Model:**  
Learn a probability distribution  $p(x)$

→ Detect outliers  
Feature learning (without labels)  
**Sample** to generate new data

**Conditional Generative Model:** Learn  $p(x|y)$

→ Assign labels while rejecting outliers  
**Sample** to generate data from labels

Stanford CS231n 10<sup>th</sup> Anniversary

Lecture 13 - 35

May 15, 2025

## Generative vs Discriminative Models

**Discriminative Model:**  
Learn a probability distribution  $p(y|x)$

**Generative Model:**  
Learn a probability distribution  $p(x)$

**Conditional Generative Model:** Learn  $p(x|y)$

"Generative models" means either of these; conditional generative models are most common in practice

Stanford CS231n 10<sup>th</sup> Anniversary

Lecture 13 - 36

May 15, 2025

## 生成模型与判别模型

判别模型：  
学习概率分布  $p(y|x)$

生成模型：  
学习概率分布  $p(x)$

条件生成式  
模型：学习  $p(x|y)$

"生成模型" 指的是上述两种情况中的任意一种；条件生成模型在实际应用中最为常见。

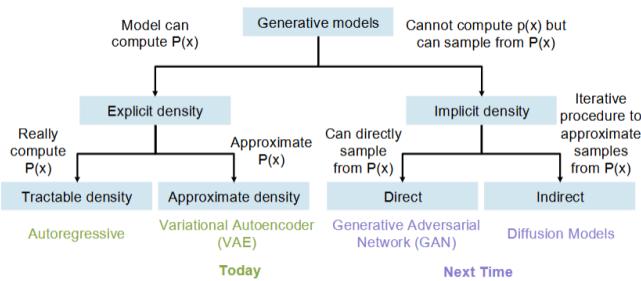
图片来源：伊恩·达雷尔。  
(生成对抗网络教程), 2017

Stanford CS231n 10<sup>th</sup> Anniversary

第 13 讲 - 36

2025 年 5 月 15 日

## Taxonomy of Generative Models

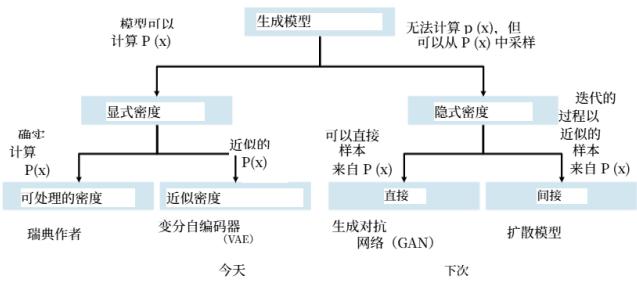


Stanford CS231n 10<sup>th</sup> Anniversary

Lecture 13 - 47

May 15, 2025

## 生成模型分类法



斯坦福大学 CS231n 十周年纪念

第 13 讲 - 47

2025 年 5 月 15 日

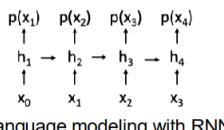
# 自回归模型

## Autoregressive Models

**Goal:** Write down an explicit function for  $p(x) = f(x, W)$

Assume  $x$  is a sequence:  $x = (x_1, x_2, \dots, x_T)$

We have already seen this!



Use the chain rule of probability:

$$\begin{aligned} p(x_1) &= p(x_2) \\ h_1 \rightarrow h_2 \rightarrow h_3 \rightarrow h_4 &= p(x_1, x_2, x_3, \dots, x_T) \\ \uparrow &= p(x_1)p(x_2 | x_1)p(x_3 | x_1, x_2) \dots \\ x_0 &= \prod_{t=1}^T p(x_t | x_1, \dots, x_{t-1}) \\ x_1 & x_2 & x_3 & x_4 \end{aligned}$$

Language modeling with RNN

Stanford CS231n 10<sup>th</sup> Anniversary

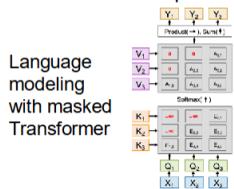
Lecture 13 - 55

May 15, 2025

## LLMs are Autoregressive Models

**Goal:** Write down an explicit function for  $p(x) = f(x, W)$

Assume  $x$  is a sequence:  $x = (x_1, x_2, \dots, x_T)$



Use the chain rule of probability:

$$\begin{aligned} p(x) &= p(x_1, x_2, x_3, \dots, x_T) \\ &= p(x_1)p(x_2 | x_1)p(x_3 | x_1, x_2) \dots \\ &= \prod_{t=1}^T p(x_t | x_1, \dots, x_{t-1}) \end{aligned}$$

Stanford CS231n 10<sup>th</sup> Anniversary

Lecture 13 - 56

May 15, 2025

## Autoregressive Models of Images

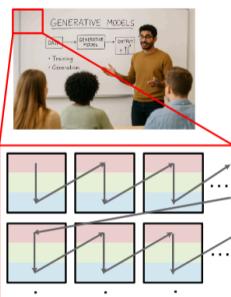
Treat an image as a sequence of 8-bit subpixel values (scanline order)

Predict each subpixel as a classification among 256 values [0...255]

Model with an RNN or Transformer

**Problem:** Too expensive. 1024x1024 image is a sequence of 3M subpixels

**Solution** (jumping ahead): Model as sequence of tiles, not sequence of subpixels



Stanford CS231n 10<sup>th</sup> Anniversary

Lecture 13 - 59

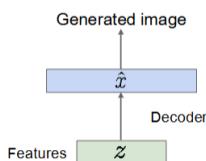
May 15, 2025

# 变分自编码器

## 引入问题

### (Non-Variational) Autoencoders

If we could generate new  $z$ , could use the decoder to generate images



**Problem:** Generating new  $z$  is not any easier than generating new  $x$

**Solution:** What if we force all  $z$  to come from a known distribution?

Stanford CS231n 10<sup>th</sup> Anniversary

Lecture 13 - 70

May 15, 2025

变分：不可求，用一些公式来近似转换

## 自回归模型

目标：为  $p(x) = f(x, W)$  写下一个明确的函数

假设  $x$  是一个序列：

$x = (x_1, x_2, \dots, x_T)$

我们已经见过这个了！

$$\begin{aligned} p(x_1) &= p(x_2) \\ h_1 \rightarrow h_2 \rightarrow h_3 \rightarrow h_4 &= p(x_1, x_2, x_3, \dots, x_T) \\ \uparrow &= p(x_1)p(x_2 | x_1)p(x_3 | x_1, x_2) \dots \\ x_0 &= \prod_{t=1}^T p(x_t | x_1, \dots, x_{t-1}) \\ x_1 & x_2 & x_3 & x_4 \end{aligned}$$

使用循环神经网络进行语言建模

```
{\begin{aligned} p(x) &= p(\text{left}(x_{-1}), x_{-2}, x_{-3}, \dots, \\ x_{-T}\text{right}) \\ &\quad \&= p(\text{left}(x_{-1})\text{right})p(\text{left}(x_{-2})| \\ x_{-1}\text{right})p(\text{left}(x_{-3})|x_{-1}, x_{-2}\text{right}) \dots \\\&\quad \&= \prod_{t=1}^T p(\text{left}(x_{-t})|x_{-1}, \dots, x_{-t-1}\text{right}) \end{aligned}}
```

使用概率的链式法则：

斯坦福大学 CS231n 十周年纪念

第 13 讲 - 55

2025 年 5 月 15 日

## 大语言模型是自回归模型

目标：为  $p(x) = f(x, W)$  写下一个明确的函数

假设  $x$  是一个序列：

$x = (x_1, x_2, \dots, x_T)$



$$\begin{aligned} p(x) &= p(x_1, x_2, x_3, \dots, x_T) \\ &= p(x_1)p(x_2 | x_1)p(x_3 | x_1, x_2) \dots \\ &= \prod_{t=1}^T p(x_t | x_1, \dots, x_{t-1}) \end{aligned}$$

使用概率的链式法则：

斯坦福大学 CS231n 十周年纪念

第 13 讲 - 56

2025 年 5 月 15 日

## 图像的自回归模型

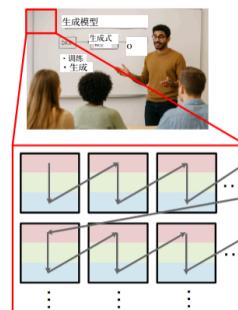
将图像视为 8 位子像素值的序列（按扫描线顺序）

将每个子像素预测为 256 个值 [0...255] 中的一个分类

使用循环神经网络 (RNN) 或 Transformer 构建模型

问题：成本过高。1024x1024 的图像是一个包含 300 万个子像素的序列

解决方案（提前说明）：将其建模为瓦片序列，而非子像素序列



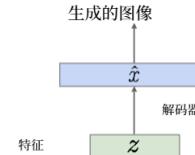
斯坦福大学 CS231n 十周年纪念

第 13 讲 - 59

2025 年 5 月 15 日

## (非变分) 自编码器

如果我们能够生成新的  $z$ ，就可以使用解码器来生成图像



问题：生成新的  $z$  并不比生成新的  $x$  容易

解决方案：如果我们强制所有  $z$  都来自一个已知分布会怎样？

Stanford CS231n 10<sup>th</sup> Anniversary

Lecture 13 - 70

May 15, 2025

第 13 讲 - 70

2025 年 5 月 15 日

# 第十四讲：生成模型（二）

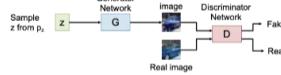
## 生成对抗网络 (GANs)

### Generative Adversarial Networks: Summary

Jointly train Generator and Discriminator with a minimax game

- Pros:
- Simple formulation
  - Very good image quality

- Cons:
- No loss curve to look at
  - Unstable training
  - Hard to scale to big models + data



These were the go-to generative models from ~2016 – 2021

### 生成对抗网络：总结

通过极小极大博弈联合训练生成器和判别器

优点: - 公式简单 - 图像质量非常好

缺点:

- 没有可参考的损失曲线 - 训练不稳定
- 难以扩展到大型模型和数据



这些是 2016 年至 2021 年左右主流的生成模型。

Stanford CS231n 10<sup>th</sup> Anniversary

Lecture 14 - 35

May 20, 2025

斯坦福大学 CS231n 第十周年纪念讲座 14 - 35

2025 年 5 月 20 日

## 扩散模型

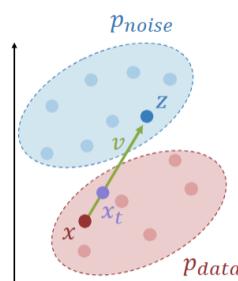
### Diffusion Models: Rectified Flow

Suppose we have a simple  $p_{noise}$  (e.g. Gaussian) and samples from  $p_{data}$

On each training iteration, sample:  
 $z \sim p_{noise}$   $x \sim p_{data}$   $t \sim Uniform[0, 1]$

Set  $x_t = (1-t)x + tz$ ,  $v = z - x$

Train a neural network to predict v:  
 $L = \|f_\theta(x_t, t) - v\|_2^2$



Liu et al., "Flow Straight and Fast: Learning to Generate and Transfer Data with Rectified Flow", 2022  
Lipman et al., "Flow Matching for Generative Modeling", 2022

Stanford CS231n 10<sup>th</sup> Anniversary

Lecture 14 - 45

May 20, 2025

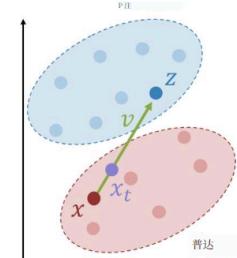
### 扩散模型：修正流

假设我们有一个简单的 noise (例如高斯分布), 以及来自  $p_{data}$  的样本。

在每次训练迭代中, 采样:  $z \sim p_{noise}$   
 $x \sim p_{data}$   $t \sim Uniform[0, 1]$

Set  $x_t = (1-t)x + tz$ ,  $v = z - x$

训练一个神经网络来预测 v:  
 $L = \|f_\theta(x_t, t) - v\|_2^2$



Liu 等人, "Flow Straight and Fast: 学习通过整流流生成和传输数据", 2022 年; Lipman 等人, "用于生成建模的流匹配", 2022 年

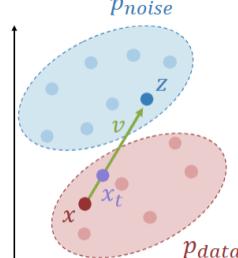
斯坦福大学 CS231n 十周年纪念讲座 14 - 45

2025 年 5 月 20 日

### Diffusion Models: Rectified Flow

Core training loop is just a few lines of code!

```
for x in dataset:  
    z = torch.randn_like(x)  
    t = random.uniform(0, 1)  
    xt = (1 - t) * x + t * z  
    v = model(xt, t)  
    loss = (z - x - v).square().sum()
```



Liu et al., "Flow Straight and Fast: Learning to Generate and Transfer Data with Rectified Flow", 2022  
Lipman et al., "Flow Matching for Generative Modeling", 2022

Stanford CS231n 10<sup>th</sup> Anniversary

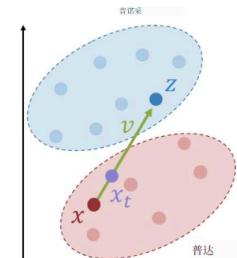
Lecture 14 - 46

May 20, 2025

### 扩散模型：整流流

核心训练循环只需几行代码!

```
for x in dataset:  
    z = torch.randn_like(x)  
    t = random.uniform(0, 1)  
    xt = (1 - t) * x + t * z  
    v = model(xt, t)  
    loss = (z - x - v).square().sum()
```



Liu 等人, "Flow Straight and Fast: 学习通过整流流生成和传输数据", 2022 年; Lipman 等人, "用于生成建模的流匹配", 2022 年

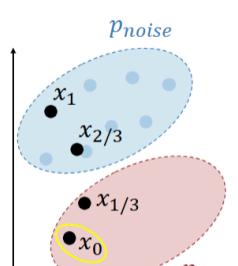
斯坦福大学 CS231n 十周年纪念讲座 14 - 46

2025 年 5 月 20 日

### Rectified Flow: Sampling

Choose number of steps T (often T=50)

```
Sample x ~ p_{noise}  
For t in [1, 1 - 1/T, 1 - 2/T, ..., 0]:  
    Evaluate v_t = f_\theta(x_t, t)  
    Step x = x - v_t/T  
Return x  
  
sample = torch.randn(x_shape)  
for t in torch.linspace(1, 0, num_steps):  
    v = model(sample, t)  
    sample = sample - v / num_steps
```



Liu et al., "Flow Straight and Fast: Learning to Generate and Transfer Data with Rectified Flow", 2022  
Lipman et al., "Flow Matching for Generative Modeling", 2022

Stanford CS231n 10<sup>th</sup> Anniversary

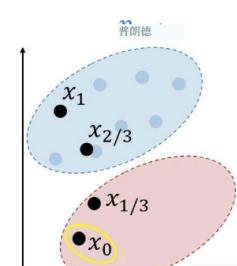
Lecture 14 - 56

May 20, 2025

### 修正流：采样

选择步数 T (通常为  $T = 50$ )

```
Sample x ~ p_{noise}  
For t in [1, 1 - 1/T, 1 - 2/T, ..., 0]:  
    Evaluate v_t = f_\theta(x_t, t)  
    Step x = x - v_t/T  
Return x  
  
sample = torch.randn(x_shape)  
for t in torch.linspace(1, 0, num_steps):  
    v = model(sample, t)  
    sample = sample - v / num_steps
```



Liu 等人, "Flow Straight and Fast: 学习通过整流流生成和传输数据", 2022 年; Lipman 等人, "用于生成建模的流匹配", 2022 年

斯坦福大学 CS231n 十周年纪念讲座 14 - 56

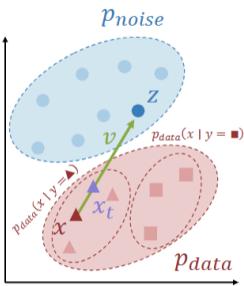
2025 年 5 月 20 日

## Conditional Rectified Flow

### Training

```
for (x, y) in dataset:
    z = torch.randn_like(x)
    t = random.uniform(0, 1)
    xt = (1 - t) * x + t * z
    v = model(xt, y, t)
    loss = (z - x - v).square().sum()
```

Can we control how much we "emphasize" the conditioning y?



### Sampling

```
y = user_input()
sample = torch.randn(x_shape)
for t in torch.linspace(1, 0, num_steps):
    v = model(sample, y, t)
    sample = sample - v / num_steps
```

Liu et al., 'Flow Straight and Fast: Learning to Generate and Transfer Data with Rectified Flow', 2022  
Lipman et al., 'Flow Matching for Generative Modeling', 2022

Stanford CS231n 10th Anniversary

Lecture 14 - 61

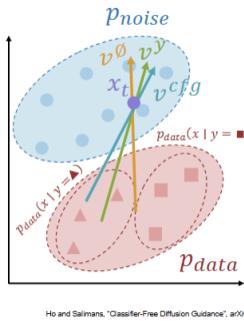
May 20, 2025

## Classifier-Free Guidance (CFG)

### Training

```
for (x, y) in dataset:
    z = torch.randn_like(x)
    t = random.uniform(0, 1)
    xt = (1 - t) * x + t * z
    if random.random() < 0.5: y = y_null
    v = model(xt, y, t)
    loss = (z - x - v).square().sum()
```

Can we control how much we "emphasize" the conditioning y?



Randomly drop y during training.

Now the same model is conditional and unconditional!

Consider a noisy  $x_t$ :

$v^\theta = f_\theta(x_t, y_0, t)$  points toward  $p(x)$

$v^y = f_\theta(x_t, y, t)$  points toward  $p(x|y)$

$v^{cfg} = (1+w)v^y - wv^\theta$  points more toward  $p(x|y)$

During sampling, step according to  $v^{cfg}$

Stanford CS231n 10th Anniversary

Lecture 14 - 67

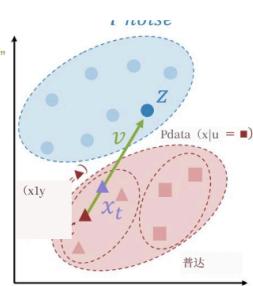
May 20, 2025

## 条件修正流

### Training

```
for (x, y) in dataset:
    z = torch.randn_like(x)
    t = random.uniform(0, 1)
    xt = (1 - t) * x + t * z
    v = model(xt, y, t)
    loss = (z - x - v).square().sum()
```

Can we control how much we "emphasize" the conditioning y?



### Sampling

```
y = user_input()
sample = torch.randn(x_shape)
for t in torch.linspace(1, 0, num_steps):
    v = model(sample, y, t)
    sample = sample - v / num_steps
```

刘等人,《Flow Straight and Fast: Learning to Generate and Transfer Data with Rectified Flow》, 2022 年; 利普曼等人,《Flow Matching for Generative Modeling》2025 年 5 月 20 日

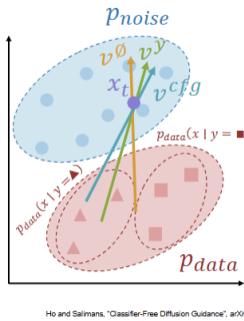
斯坦福大学 CS231n 十周年纪念讲座 14 -

## Classifier-Free Guidance (CFG)

### Training

```
for (x, y) in dataset:
    z = torch.randn_like(x)
    t = random.uniform(0, 1)
    xt = (1 - t) * x + t * z
    if random.random() < 0.5: y = y_null
    v = model(xt, y, t)
    loss = (z - x - v).square().sum()
```

Can we control how much we "emphasize" the conditioning y?



Randomly drop y during training.

Now the same model is conditional and unconditional!

Consider a noisy  $x_t$ :

$v^\theta = f_\theta(x_t, y_0, t)$  points toward  $p(x)$

$v^y = f_\theta(x_t, y, t)$  points toward  $p(x|y)$

$v^{cfg} = (1+w)v^y - wv^\theta$  points more toward  $p(x|y)$

During sampling, step according to  $v^{cfg}$

Stanford CS231n 10th Anniversary

Lecture 14 - 67

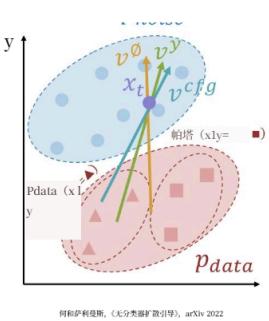
May 20, 2025

## 无分类器引导 (CFG)

### 训练

```
for (x, y) in dataset:
    z = torch.randn_like(x)
    t = random.uniform(0, 1)
    xt = (1 - t) * x + t * z
    if random.random() < 0.5: y = y_null
    v = model(xt, y, t)
    loss = (z - x - v).square().sum()
```

我们能否控制对条件 y 的“强调”程度?



训练期间随机丢弃 y。

现在，同一个模型既支持条件模式也支持无条件模式！想

象一个带噪声的  $x_t$ ,  $v = f_\theta(x_t, y, t)$  指向  $p(x)$ 。

$v^y = f_\theta(x_t, y, t)$  指向  $p(x|y)$   $v^{cfg} = (1+w)v^y - wv^\theta$  更指

向  $p(x|y)$  采样期间，按照  $v^{cfg}$  进行步骤操作

何和萨利曼斯,《无分类器扩散引导》，arXiv 2022

斯坦福大学 CS231n 十周年纪念讲座 14 - 67

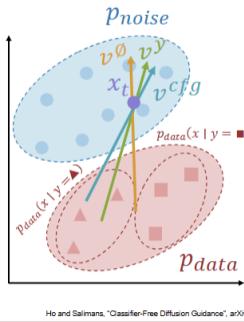
2025 年 5 月 20 日

## Classifier-Free Guidance (CFG)

### Training

```
for (x, y) in dataset:
    z = torch.randn_like(x)
    t = random.uniform(0, 1)
    xt = (1 - t) * x + t * z
    if random.random() < 0.5: y = y_null
    v = model(xt, y, t)
    loss = (z - x - v).square().sum()
```

Can we control how much we "emphasize" the conditioning y?



Sampling

y = user\_input()

sample = torch.randn(x\_shape)

for t in torch.linspace(1, 0, num\_steps):

    vy = model(sample, y, t)

    vθ = model(sample, y\_null, t)

    v = (1 + w) \* vy - w \* vθ

    sample = sample - v / num\_steps

Ho 和 Salimans, 'Classifier-Free Diffusion Guidance', arXiv 2022

Stanford CS231n 10th Anniversary

Lecture 14 - 68

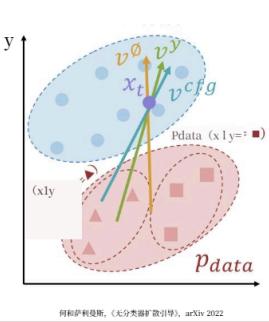
May 20, 2025

## 无分类器引导 (CFG)

### 训练

```
for (x, y) in dataset:
    z = torch.randn_like(x)
    t = random.uniform(0, 1)
    xt = (1 - t) * x + t * z
    if random.random() < 0.5: y = y_null
    v = model(xt, y, t)
    loss = (z - x - v).square().sum()
```

我们能否控制对条件 y 的“强调”程度?



采样

y = user\_input()  
sample = torch.randn(x\_shape)  
for t in torch.linspace(1, 0, num\_steps):  
    vy = model(sample, y, t)  
    vθ = model(sample, y\_null, t)  
    v = (1 + w) \* vy - w \* vθ  
    sample = sample - v / num\_steps

Ho 和 Salimans, '无分类器扩散引导', arXiv 2022

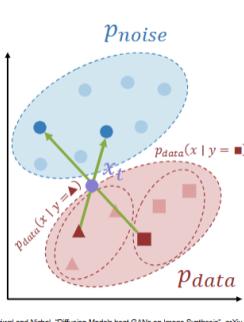
斯坦福大学 CS231n 十周年纪念讲座 14 - 68 2025 年 5 月 20 日

## Optimal Prediction

### Training

```
for (x, y) in dataset:
    z = torch.randn_like(x)
    t = random.uniform(0, 1)
    xt = (1 - t) * x + t * z
    if random.random() < 0.5: y = y_null
    v = model(xt, y, t)
    loss = (z - x - v).square().sum()
```

Q: What is the optimal prediction for the network?



There may be many pairs (x, z) that give the same  $x_t$ ; network must average over them

Full noise ( $t=1$ ) is easy: optimal  $v$  is mean of  $p_{data}$   
No noise ( $t=0$ ) is easy: optimal  $v$  is mean of  $p_{noise}$   
Middle noise is hardest, most ambiguous  
But we give equal weight to all noise levels!  
Solution: Use a non-uniform noise schedule

Stanford CS231n 10th Anniversary

Lecture 14 - 79

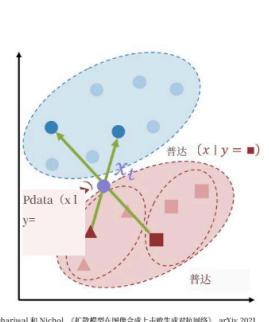
May 20, 2025

## 最优预测

### 训练

```
for (x,y) in dataset:
    z = torch.randn_like(x)
    t = random.uniform(0, 1)
    xt = (1 - t) * x + t * z
    if random.random() < 0.5: y = y_null
    v = model(xt, y, t)
    loss = (z - x - v).square().sum()
```

问：该网络的最佳预测是什么？



完全噪声( $t=1$ )简单：最优  $v$  是  $p_{data}$  的平均值。无噪声( $t=0$ )很简单：最优  $v$  是  $p_{noise}$  的平均值。  
中等噪声最难处理，最模糊  
但我们对所有噪声水平赋予同等权重！解决方案：  
使用非均匀噪声调度

Dhariwal 和 Nichol, 'Diffusion Models beat GANs on Image Synthesis', arXiv 2021  
Ho 和 Salimans, '无分类器扩散引导', arXiv 2022

斯坦福大学 CS231n 10th 周年纪念讲座 14 - 79

2025 年 5 月 20 日

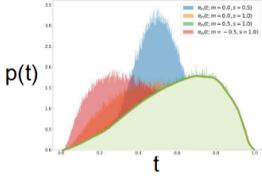
## Noise Schedules

### Training

```
for (x, y) in dataset:
    z = torch.randn_like(x)
    t = torch.randn(), sigmoid()
    xt = (1 - t) * x + t * z
    if random.random() < 0.5: y = y_null
    v = model(xt, y, t)
    loss = (z - x - v).square().sum()
```

There may be many pairs  $(x, z)$  that give the same  $x_t$ ; network must average over them

Full noise ( $t=1$ ) is easy: optimal  $v$  is mean of  $P_{data}$   
No noise ( $t=0$ ) is easy: optimal  $v$  is mean of  $P_{noise}$   
Middle noise is hardest, most ambiguous  
But we give equal weight to all noise levels!  
Solution: Use a non-uniform noise schedule



Put more emphasis on middle noise

Common choice: logit-normal sampling

For high-res data, often shift to higher noise to account for pixel correlations

Egger et al., "Scaling Rectified Flow Transformers for High-Resolution Image Synthesis", arXiv 2024

Stanford CS231n 10<sup>th</sup> Anniversary

Lecture 14 - 82

May 20, 2025

## Diffusion: Rectified Flow

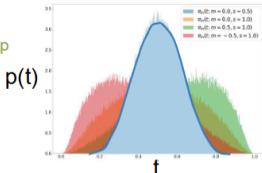
### Training

```
for (x, y) in dataset:
    z = torch.randn_like(x)
    t = torch.randn(), sigmoid()
    xt = (1 - t) * x + t * z
    if random.random() < 0.5: y = y_null
    v = model(xt, y, t)
    loss = (z - x - v).square().sum()
```

### Sampling

```
y = user_input()
sample = torch.randn(x.shape)
for t in torch.linspace(1, 0, num_steps):
    vy = model(sample, y, t)
    v0 = model(sample, y_null, t)
    v = (1 + w) * vy - w * v0
    sample = sample - v / num_steps
```

Simple and scalable setup for many generative modeling problems!



Put more emphasis on middle noise

Common choice: logit-normal sampling

For high-res data, often shift to higher noise to account for pixel correlations

Egger et al., "Scaling Rectified Flow Transformers for High-Resolution Image Synthesis", arXiv 2024

Stanford CS231n 10<sup>th</sup> Anniversary

Lecture 14 - 83

May 20, 2025

## Diffusion: Rectified Flow

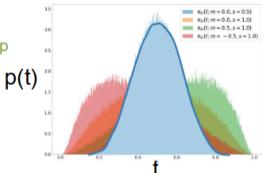
### Training

```
for (x, y) in dataset:
    z = torch.randn_like(x)
    t = torch.randn(), sigmoid()
    xt = (1 - t) * x + t * z
    if random.random() < 0.5: y = y_null
    v = model(xt, y, t)
    loss = (z - x - v).square().sum()
```

### Sampling

```
y = user_input()
sample = torch.randn(x.shape)
for t in torch.linspace(1, 0, num_steps):
    vy = model(sample, y, t)
    v0 = model(sample, y_null, t)
    v = (1 + w) * vy - w * v0
    sample = sample - v / num_steps
```

Simple and scalable setup for many generative modeling problems!



Put more emphasis on middle noise

Common choice: logit-normal sampling

For high-res data, often shift to higher noise to account for pixel correlations

Egger et al., "Scaling Rectified Flow Transformers for High-Resolution Image Synthesis", arXiv 2024

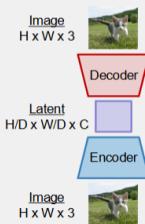
Stanford CS231n 10<sup>th</sup> Anniversary

Lecture 14 - 84

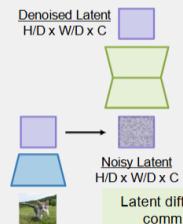
May 20, 2025

## Latent Diffusion Models (LDMs)

Train **encoder** + **decoder** to convert images to **latents**



Train **diffusion model** to remove noise from **latents** (**Encoder** is frozen)



After training:

Sample random latent

Iteratively apply diffusion model to remove noise

run **decoder** to get **image**

Latent diffusion is the most common form today

Stanford CS231n 10<sup>th</sup> Anniversary

Lecture 14 - 92

May 20, 2025

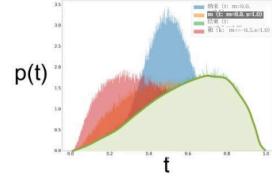
## 噪声调度

### 训练

```
for (x, y) in dataset:
    z = torch.randn_like(x)
    t = torch.randn(), sigmoid()
    xt = (1 - t) * x + t * z
    if random.random() < 0.5: y = y_null
    v = model(xt, y, t)
    loss = (z - x - v).square().sum()
```

可能存在许多对  $(x, z)$  会产生相同的  $x_t$ ; 网络必须对它们进行平均

全噪声 ( $t=1$ ) 调度: 最优  $v$  是  $P_{data}$  的平均值  
无噪音 ( $t=0$ ) 调度: 最优  $v$  是  $P_{noise}$  的平均值  
中等噪声最难处理, 也最模糊  
但我们对所有噪声水平赋予同等权重! 解决方案: 使用非均匀噪声调度



更加强调中间噪声

常见选择: logit - 正态采样

对于高分辨率数据, 通常会转向更高的噪声明以解决像素相关性问题

Egger 9人, "Scaling Rectified Flow Transformer 用于高分辨率图像合成", arXiv 2024

斯坦福大学 CS231n 10周年纪念

第 14 讲 - 82

2025 年 5 月 20 日

## 扩散: 整流流

### 训练

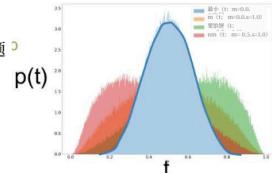
```
for (x, y) in dataset:
    z = torch.randn_like(x)
    t = torch.randn(), sigmoid()
    xt = (1 - t) * x + t * z
    if random.random() < 0.5: y = y_null
    v = model(xt, y, t)
    loss = (z - x - v).square().sum()
```

### 采样

```
y = user_input()
sample = torch.randn(x.shape)
for t in torch.linspace(1, 0, num_steps):
    vy = model(sample, y, t)
    v0 = model(sample, y_null, t)
    v = (1 + w) * vy - w * v0
    sample = sample - v / num_steps
```

适用于许多生成建模问题

p(t)



更加强调中间噪声 常见选择: logit - 正态采样

对于高分辨率数据, 通常会转向更高的噪声明以考虑像素相关性

Egger 9人, "Scaling Rectified Flow Transformer 用于高分辨率图像合成", arXiv 2024

斯坦福大学 CS231n 十周年纪念

第 14 讲 - 83 2025 年 5 月 20 日

## 扩散: 修正流

### 训练

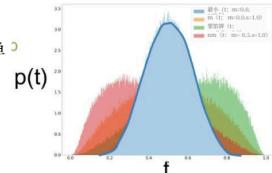
```
for (x, y) in dataset:
    z = torch.randn_like(x)
    t = torch.randn(), sigmoid()
    xt = (1 - t) * x + t * z
    if random.random() < 0.5: y = y_null
    v = model(xt, y, t)
    loss = (z - x - v).square().sum()
```

### 采样

```
y = user_input()
sample = torch.randn(x.shape)
for t in torch.linspace(1, 0, num_steps):
    vy = model(sample, y, t)
    v0 = model(sample, y_null, t)
    v = (1 + w) * vy - w * v0
    sample = sample - v / num_steps
```

许多生成建模问题的简单

p(t)



问题: 直接应用在高分辨率数据上不起作用

更强调中间噪声

常见选择: logit - 正态采样

对于高分辨率数据, 通常会转向更高的噪声明以考虑像素相关性

Rombach 等人, "IP 残差流 Transformer 用于高分辨率图像合成", arXiv 2024

斯坦福大学 CS231n 十周年纪念

第 14 讲 - 84

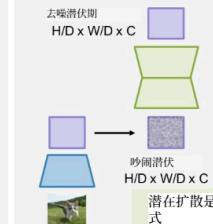
2025 年 5 月 20 日

## 潜在扩散模型 (LDMs)

训练编码器 + 解码器, 将图像转换为潜变量



训练扩散模型以去除潜在变量中的噪声 (编码器被冻结)



训练后:



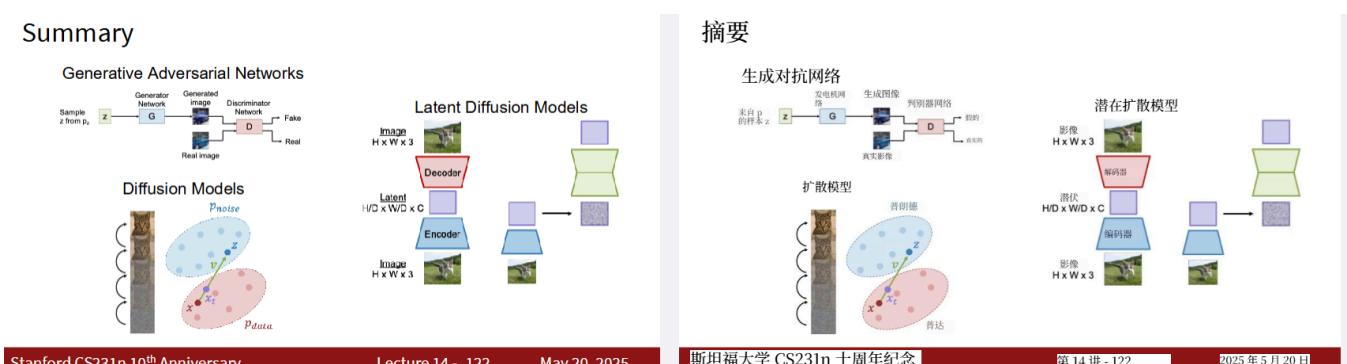
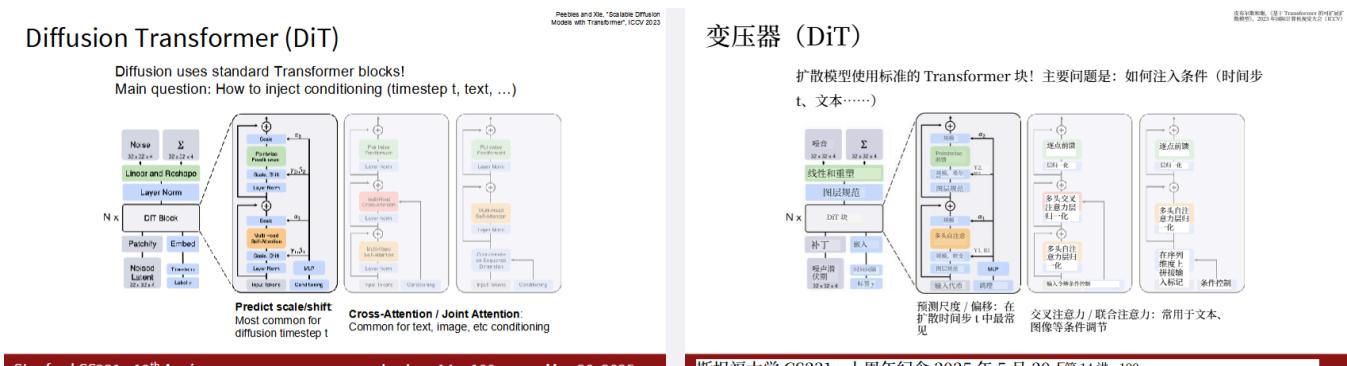
Stanford CS231n 10<sup>th</sup> Anniversary

Lecture 14 - 92

May 20, 2025

斯坦福大学 CS231n 十周年纪念

第 14 讲 - 92 2025 年 5 月 20 日

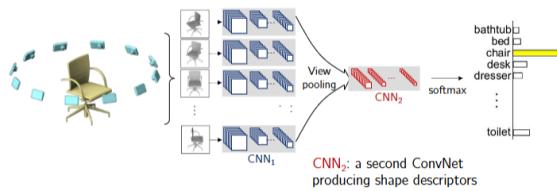


# 第十五讲：3D视觉

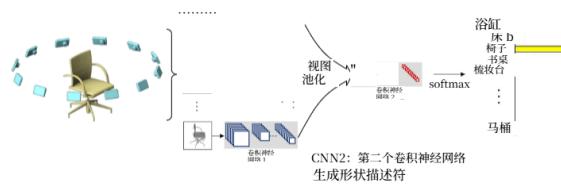
3D几何表示方法：

- 显式表示：点云、网格、参数化
- 隐式表示

## Multi-View CNN



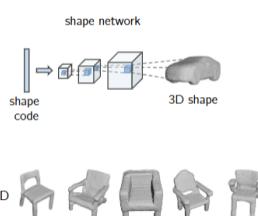
## 多视角卷积神经网络



Su et al. ICCV 2015

苏等人。ICCV 2015

## 3D-GANs



Wu et al. NeurIPS 2016

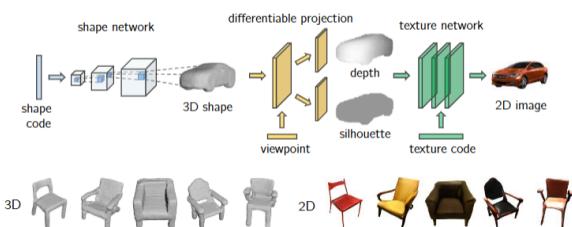
## 3D 赢



吴等人。《神经信息处理系统会议》2016年

苏等人。ICCV 2015

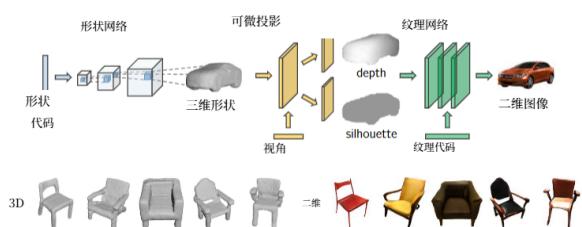
## Visual Object Networks (Geometry + Rendering)



Wu et al. NeurIPS 2016

Zhu et al. NeurIPS 2018

## 视觉对象网络（几何 + 渲染）

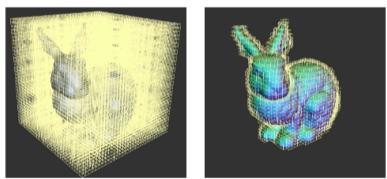


吴等人。《神经信息处理系统会议》2016年

苏等人。ICCV 2015

## Octave Tree Representations

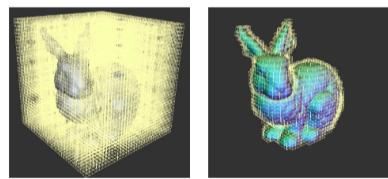
- Store the sparse surface signals
- Constrain the computation near the surface



Slide Credit: Hao Su

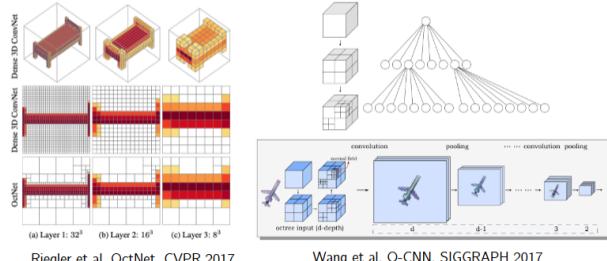
## 八度树表示

- 存储稀疏表面信号
- 将计算限制在表面附近

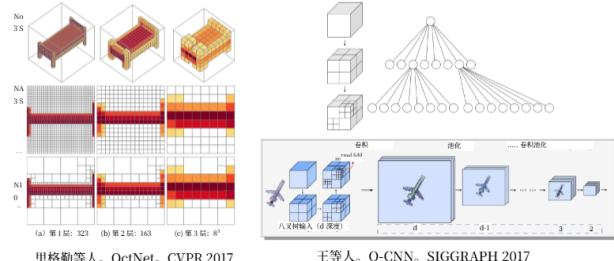


幻灯片来源：郝苏

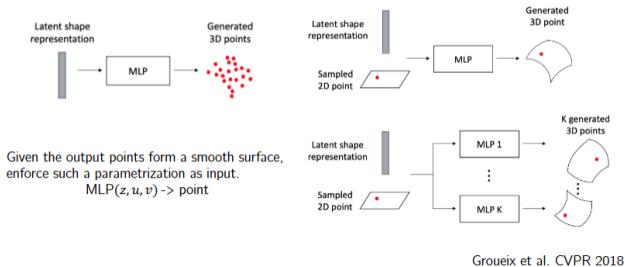
## Octree: Recursively Partition the Space



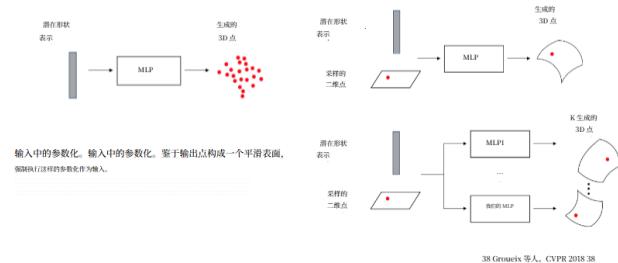
## 八叉树：递归划分空间



## Parametric Decoder: AtlasNet



## 参数解码器：AtlasNet



# 第十六讲：多模态基础模型

How do identify a model as a Foundation?

### Always see with foundation models:

- general / robust to many different tasks

### Often see with foundation models:

- Large # params
- Large amount of data
- Self-supervised pre-training objective

如何将一个模型认定为基础模型？

### 基础模型的共性：

- 通用 / 对多种不同任务具有稳健性

### 基础模型通常具有以下特点：

- 大量参数数量
- 大量数据
- 自监督预训练目标

We will focus on multimodal (vision) foundation models

Language	Classification	LM + Vision	And More!	Chaining
ELMo BERT GPT T5	CLIP CoCa	LLaVA Flamingo GPT-4V Gemini Molmo	Segment Anything Whisper Dalle Stable Diffusion Imagen	LMs + CLIP Visual Programming

我们将专注于多模态（视觉）基础模型

Language	Classification	LM + Vision	And More!	Chaining
ELMo BERT GPT T5	CLIP CoCa	LLaVA Flamingo GPT-4V Gemini Molmo	Segment Anything Whisper Dalle Stable Diffusion Imagen	LMs + CLIP Visual Programming

Ranjay Krishna

Lecture 16 - 8

May 27, 2025

奎师那

第 16 讲 - 8

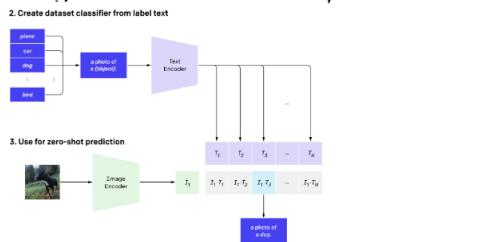
2025 年 5 月 27 日

## (分类) clip: 无需人工标注 (参数规模大、训练数据规模大)

图像编码和文本解码器都有

对比学习：拉进相同、远离不同

That's it! Now, you can use CLIP as a foundation model for image classification for any dataset

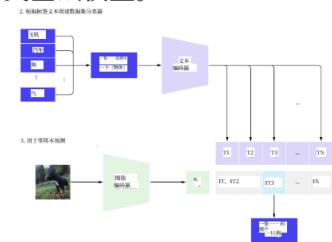


Ranjay Krishna

Lecture 16 - 32

May 27, 2025

就是这样！现在，你可以将 CLIP 用作任何数据集的图像分类基础模型。



奎师那

第 16 讲 - 32

2025 年 5 月 27 日

Why does CLIP perform so well?

How can no labels beat labels??

Scale!

为什么 CLIP 表现如此出色？

为什么无标签能战胜有标签??

规模！

Ranjay Krishna

Lecture 16 - 39

May 27, 2025

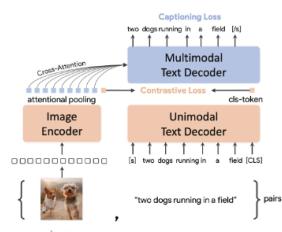
奎师那

第 16 讲 - 39

2025 年 5 月 27 日

## (分类) coca: clip改进

CoCa added a decoder with a captioning loss



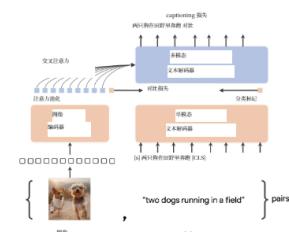
"Contrastive Captioners are Image-Text Foundation Models", 2022

Ranjay Krishna

Lecture 16 - 43

May 27, 2025

CoCa 添加了一个带有 captioning loss 的解码器



"对比 captioner 是图文基础模型", 2022 年

奎师那

第 16 讲 - 43

2025 年 5 月 27 日

## Advantages of CLIP-style models

1. Dot product is super efficient
  - a. Easy to train (enables scaling)
  - b. Fast inference, e.g., retrieval over 5B images
2. Open-vocabulary (zero-shot generalization)
3. Can be chained with other models (CuPL)  
[we will discuss this later today]

Ranjay Krishna

Lecture 16 -

May 27, 2025

## CLIP 风格模型的优势

1. 点积效率极高
  - a. 易于训练 (支持扩展)
  - b. 快速推理, 例如, 对 50 亿张图像进行检索
2. 开放词汇 (零样本泛化)
3. 可以与其他模型 (CuPL) 链接【我们今天晚些时候会讨论这一点】

奎师那

第 16 讲 -

2025 年 5 月 27 日

## Disadvantages of CLIP-style models

1. Rely too heavily on batch size to learn concepts
2. Image-level captions are insufficient supervision
3. You can't know everything in your 5B dataset



It's extremely important to be intentional about data collection and filtering

Ranjay Krishna

Lecture 16 -

May 27, 2025

## CLIP 风格模型的缺点

1. 过度依赖批量大小来学习概念
2. 图像级别的标题作为监督信息是不够的
3. 你不可能了解 50 亿数据集里的所有内容



有意识地进行数据收集和筛选是极其重要的。

奎师那

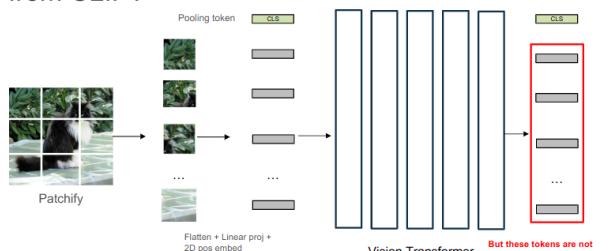
第 16 讲 -

2025 年 5 月 27 日

# (图文->文) llava：为大型语言模型增加视觉信息

比如使用clip来提取

## What features should we use from CLIP?

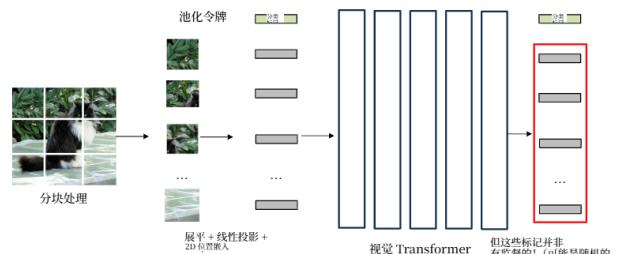


Ranjay Krishna

Lecture 16 - 65

May 27, 2025

## 我们应该使用 CLIP 的哪些特征?

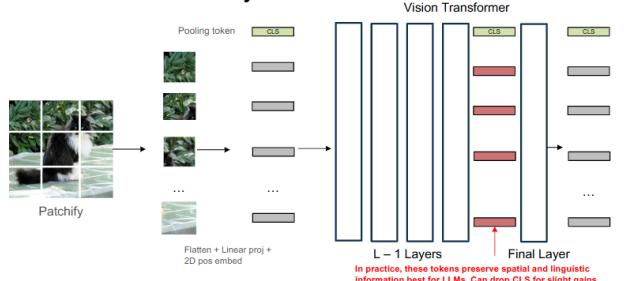


奎师那

第 16 讲 - 65

2025 年 5 月 27 日

## Use Penultimate Layer!

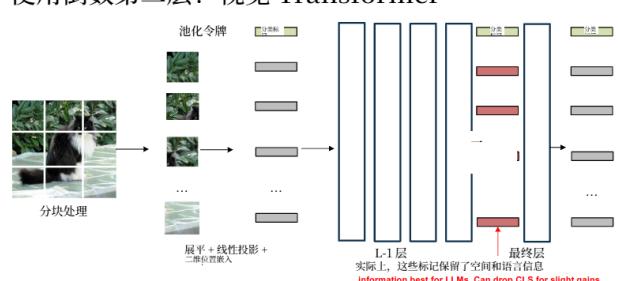


Ranjay Krishna

Lecture 16 - 66

May 27, 2025

## 使用倒数第二层！视觉 Transformer



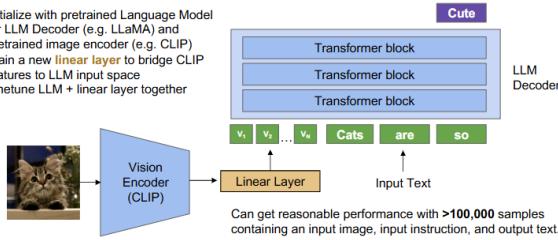
奎师那

第 16 讲 - 66

2025 年 5 月 27 日

## LLaVA – Overall Architecture + Training Recipe

1. Initialize with pretrained Language Model for LLM Decoder (e.g. LLaMA) and pretrained image encoder (e.g. CLIP)
2. Train a new **linear layer** to bridge CLIP features to LLM input space
3. Finetune LLM + linear layer together



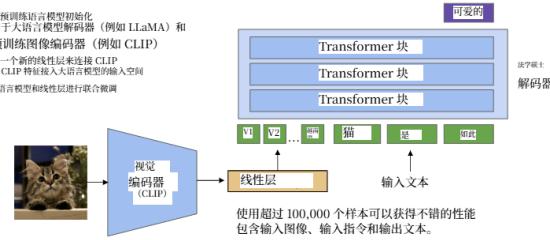
Ranjay Krishna

Lecture 16 - 67

May 27, 2025

## LLaVA —— 整体架构 + 训练方法

1. 使用预训练的语音模型初始化  
用大语言模型解码器（例如LLaMA）和  
预训练图像编码器（例如CLIP）
2. 训练一个新的线性层来连接CLIP  
将CLIP特征接入大语言模型的输入空间
3. 对大语言模型和线性层进行联合微调

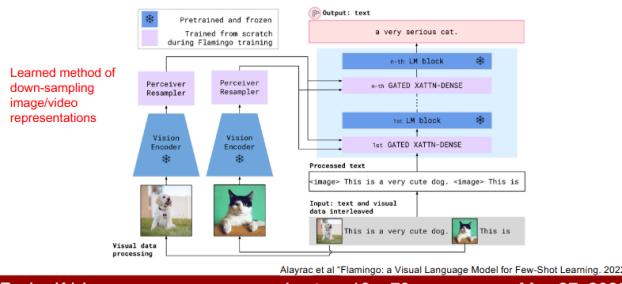


奎师那

第 16 讲 - 67

2025 年 5 月 27 日

## Flamingo full architecture

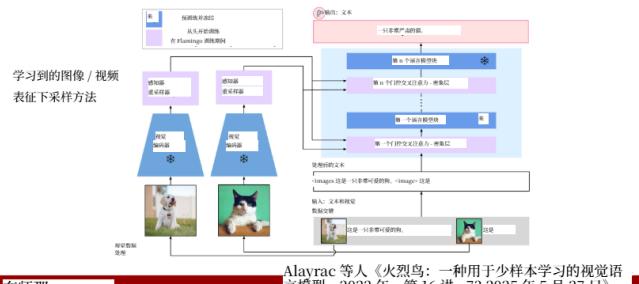


Ranjay Krishna

Lecture 16 - 73

May 27, 2025

## Flamingo 完整架构



奎师那

Alayrac 等人.《火烈鸟：一种用于少样本学习的视觉语言模型》。2022 年。第 16 讲 - 73 2025 年 5 月 27 日

模型对未见过的东西进行分类

### Solution: chaining

1. Get an LLM to generate a description.
2. Classify using the description

"A **marimba** is a large wooden percussion instrument that looks like a xylophone."  
"A **viaduct** is a bridge composed of several spans supported by piers or pillars."  
"A **papillon** is a small, spaniel-type dog with a long, silky coat and fringed ears."  
"A **lorikeet** is a small to medium-sized parrot with a brightly colored plumage."



Ranjay Krishna

Lecture 16 - 134

May 27, 2025

### 解决方案：链式法

1. 让大语言模型生成一段描述。
2. 使用该描述进行分类

"森林里有一座木制的高架桥，看起来很古老。"  
"这座桥是由许多个不同的拱形组成的，连接两侧的山峰以跨越一条河。"  
"这座桥很大，桥孔很大，看起来很坚固，桥面上有行人走来走去。"  
"这座桥的颜色是深棕色的，看起来很坚固，桥面上有行人走来走去。"

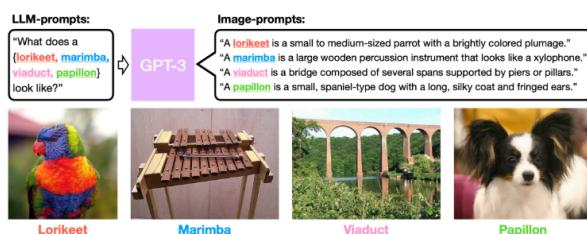


普拉特等人.《鸭嘴兽长什么样？为零样本图像分类生成定制提示词》，2023 年。

第 16 讲 - 134

2025 年 5 月 27 日

## CuPL (CUstomized Prompts via Language models)



Ranjay Krishna

Lecture 16 - 135

May 27, 2025

## CuPL (通过语言模型实现的定制化提示词)



普拉特等人.《鸭嘴兽长什么样？为零样本图像分类生成定制提示词》，2023 年。

第 16 讲 - 135

2025 年 5 月 27 日

# 第十七讲：机器人学习

强化学习和普通的监督学习有什么不同：

- 随机性：奖励和状态转换可能是随机的

- 信用分配：奖励( $r_t$ )可能不直接依赖于动作( $a_t$ )
- 不可微性：无法通过环境进行反向传播；无法计算奖励  $r_t$  对动作  $a_t$  的导数 ( $\frac{dr_t}{da_t}$ )
- 非平稳性：智能体的体验取决于其行为方式

## Model Learning & Model-Based Planning

Learn a model of the world's state transition function  $P(s_{t+1}|s_t, a_t)$  and then use planning through the model to make decisions



Model might not be accurate enough.

1. Execute the first action
2. Obtain new state
3. Re-optimize the action sequence using gradient descent

**Key:** GPU for parallel sampling / gradient descent

**Key question:** what should be the form of  $s_t$ ?

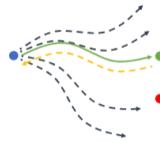
Yunzhu Li

Lecture 17 - 55

May 29, 2025

## 模型学习与基于模型的规划

学习世界状态转移函数  $P(s_{t+1}|s_t, a_t)$  的模型，然后通过该模型进行规划以做出决策



模型可能不够准确。

1. 执行第一个动作
2. 读取新状态
3. 使用梯度下降重新优化动作序列

**关键：**用于并行采样 / 梯度下降的 GPU

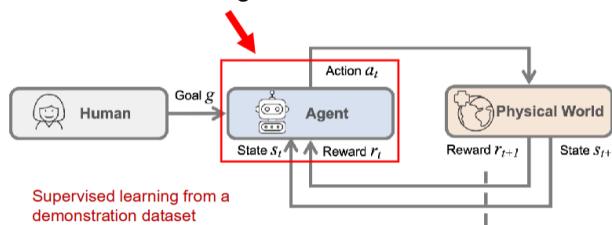
**关键问题：**  $s_t$  应该是什么形式？

李云珠

第 17 讲 - 55

2025 年 5 月 29 日

## Imitation Learning

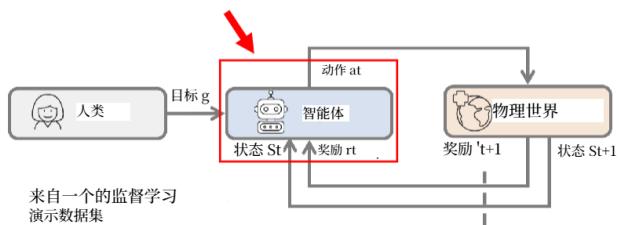


Yunzhu Li

Lecture 17 - 69

May 29, 2025

## 模仿学习



Yunzhu Li

Lecture 17 - 82

May 29, 2025

## Robotic Foundation Models

- What is a Robotic Foundation Model?
  - No explicit representation of states / transition functions
  - A policy that maps (observation/state, goal) to action
- Current Foundational Vision-and-Language Models
  - The output may **not** always be **perfect**.
  - It will always generate something **reasonable**.
- Robotic Foundation Models
  - The synthesized action may **not** always be **optimal**.
  - The generated trajectory will always be **beautiful** and **reasonable**.
- Different names
  - Vision-Language-Action Models (VLAs), Large behavior models (LBMs)

## 机器人基础模型

- 什么是机器人基础模型？
  - 没有对状态 / 转移函数的显式表示
  - 一种将 (观察 / 状态, 目标) 映射到动作的策略

- 当前的基础视觉 - 语言模型
  - 输出可能并不总是完美的。
  - 它总会生成一些合理的内容。

- 机器人基础模型
  - 合成的动作可能并不总是最优的。
  - 生成的轨迹将始终美观且合理。

- 不同名称
  - 视觉 - 语言 - 动作模型 (VLAs)、大型行为模型 (LBMs)

Yunzhu Li

Lecture 17 - 82

May 29, 2025

李云珠

第 17 讲 - 82

2025 年 5 月 29 日

# 作业3代码

## transformer captioning

- 下三角掩码可屏蔽未来时间步
- vit：图像分类的transformer

为什么最后要池化？

全局信息的综合利用

图像的语义信息分布在多个图像块中，平均池化能综合所有图像块的特征，捕捉全局的空间和语义依赖关系，从而更准确地完成分类任务。

简言之，池化操作是 ViT 从“图像块序列”到“单向量分类特征”的关键桥梁，让模型能适配分类任务的输出格式，同时充分利用所有图像块的信息。

- 用其他模型提取的图像特征可以直接作为解码器的k、v输入，不用经过编码器的位置编码等操作

# 自监督学习

自监督学习方法之所以近期备受关注，核心原因在于其训练出的模型在其他数据集（即模型未训练过的新数据集）上也能保持良好性能。

对比学习：simclr

近期，[SimCLR](#)（简单对比学习框架）提出了一种新架构，它通过**对比学习**（contrastive learning）学习优质的视觉表征。对比学习的核心目标是：让相似图像的表征向量相近，让不同图像的表征向量相异。正如我们将在本笔记本中看到的，这种简单思路能让模型在完全不使用标签的情况下，达到出人意料的训练效果。

具体来说，对于数据集中的每张图像，SimCLR 会生成该图像的两个不同增强视图，这对视图被称为**正样本对**（positive pair）。随后，模型会被引导为这对样本生成相似的表征向量。

目标：最大化正样本对的一致性

向量化实现损失函数来提速

- simclr：
  - 训练完成后，我们丢弃投影头 ( $g$ )，仅使用编码器 ( $f$ ) 及其输出的表征 ( $h$ ) 来执行下游任务（如分类）。
  - 在训练好的 SimCLR 模型基础上微调一层，用于分类任务，并将其性能与基准模型（未使用自监督学习）进行比较。
  - simclr+微调：微调线性层以完成分类任务，测试表征向量的效果，我们将从 SimCLR 模型中移除投影头，并添加一个线性层，通过微调来完成简单的分类任务。线性层之前的所有层都会被冻结，仅训练最终线性层的权重。

## DDPMs

- 前向加噪、反向去噪
- 使用CLIP的文本编码器

特性	nn.Sequential	nn.ModuleList
核心功能	自动按顺序执行模块	存储模块，手动控制执行
适用结构	线性无分支网络	复杂架构（跳跃连接、分支、动态逻辑）
forward逻辑	无需手动编写	必须手动遍历/索引执行
模块交互	仅支持“前输出→后输入”的简单交互	支持任意模块间交互（拼接、残差等）

## CLIP

- 不同模态，采用对比学习目标
- autoreload 2自动重载所有已导入的模块
- zip压缩为元组，enumerate将索引迭代
- 直接打印模型看模型结构
- 分类：设定["a person", "an animal", "food", "a landscape", "other"]文本即可利用CLIP进行图像分类
- Image Retrieval：使用文本进行图像检索

## DINO

---

使用 SimCLR 和 CLIP 等传统对比学习方法训练的模型需要极大的批量大小 (batch sizes)。这使得它们的计算成本高昂，且可及性受限。后续研究 (如 BYOL) 提出了一种替代方案，通过采用师生框架 (student-teacher framework)，避免了对大量负样本的需求。该方法表现出了出人意料的优异性能，随后被 DINO 模型采用。与 SimCLR 类似，DINO 的训练目标是最大化来自同一张图像不同视图 (views) 的两个向量之间的一致性。但与 SimCLR 不同的是，DINO 使用两个训练方式不同的独立编码器。学生网络 (student network) 通过反向传播更新参数，以匹配教师网络 (teacher network) 的输出。教师网络不通过反向传播更新，而是利用学生网络权重的指数移动平均 (EMA) 来更新自身权重。这意味着教师模型的进化速度更慢，能为学生网络提供一个稳定的学习目标。

- DINO 特征能够提供效果出色的分割线索

## 实用操作

---

使用远程服务器、python环境迁移：

- 先打包为.yml，记得删除文件里的prefix本地路径，方便用于其他地方
- 先在服务器上用.yml配好环境，再用pycharm建立ssh连接
- 环境不行？手动安装：在autodl窗口中输入下方内容，同时重启内核，即可手动安装

```
conda activate cs231n
conda install ipykernel -y
python -m ipykernel install --user --name=cs231n --display-name "conda env:cs231n"
```

- 注意数据集先别传，用filezilla传
- 有时可能jupyter中没更新，要重新加载模块
- 不要用pip，用conda安装环境