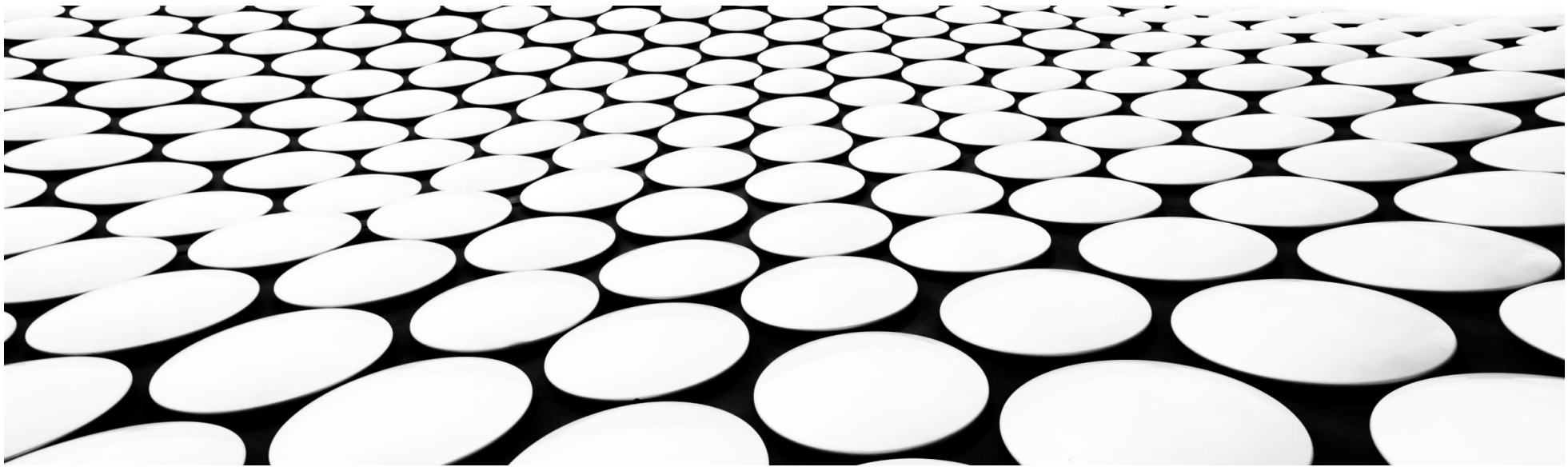
深度学习

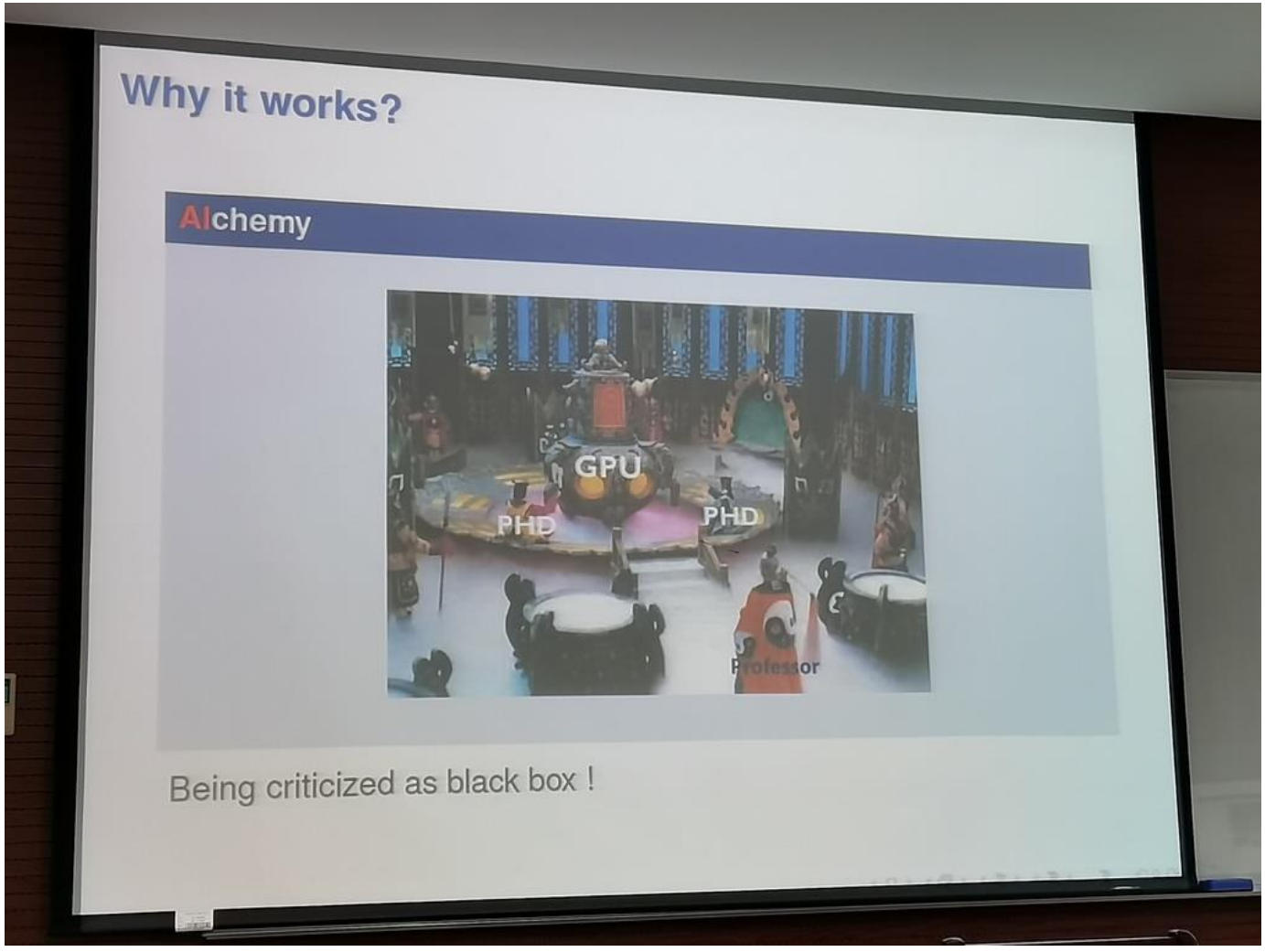
邱怡轩



今天的主题

- 神经网络训练实践


专业深度学习调试
caffe安装 10元
CNN 5元/层
RNN 8元/层
weibo.com/zhouzh2012


Why it works?
Alchemy
GPU
PHD
PHD
Professor
Being criticized as black box !

深度学习·炼丹入门

李沐

深度学习（Deep Learning）话题下的优秀答主

所谓炼丹，就是将大量灵材使用丹炉将其凝炼成丹。练成的灵丹蕴含灵材的大部分特性，方便携带，容易吸收。高级仙丹在炼制中更是能吸收天地灵气从而引发天地异象。

深度学习的模型训练就是炼丹。把精选原始数据，按照神经网络的规定法则通过计算框架提炼，从而得到一个远小于数据数倍的模型。一个好的模型不仅能抓取数据中的模式，更是能提取更加一般化规则从而可以用来预测新的数据。

精选评论（1）

知乎用户 2016-11-19

卷积三钱、全联二钱，盗梦半钱，极池一分。四味和匀，先大火煮沸，再以文火慢炖七日七夜。择良辰停火取药，滤去西梯西、弃参等杂质，以叉八六泰格拉送服。可以开天窍、晓阴阳、发混蒙、妙不可言。

208 查看回复

Ali Rahimi 的演讲

https://www.bilibili.com/video/BV1BW411Y78t
11:00-13:18

“炼金术”

- 为什么深度学习被戏称为炼金术?
- 正面含义:
  - 有用
  - 催生新的技术和问题
- 负面含义:
  - 系统性的科学认识还不足
  - 过度包装、泛化

当前目标

- 吸取实践中积累的经验
- 尽可能去解释背后的原因和动机

实践技巧

- 数据预处理
- 参数初始化
- 激活函数
- 特殊隐藏层
- 正则化方法
- 计算环境

数据预处理

实践

1. 对数据做中心化
   - 按观测进行平均（每个观测减去观测平均值）
   - 按图形通道进行平均（每个通道减去所有通道的平均值）

原因

1. 一些权重和激活函数对输入数据的数值范围比较敏感

参数初始化

实践

1. 对于以零为中心的激活函数使用 Xavier 初始化方法
2. 对于 ReLU 使用 Kaiming 初始化方法

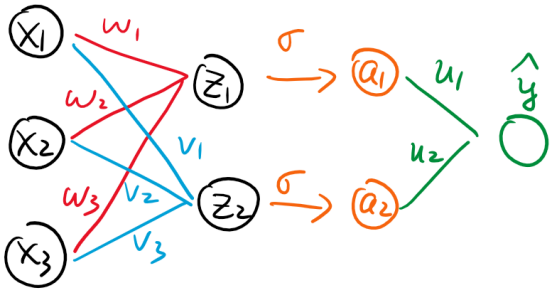
原因

1. 让输出的方法约等于输入的方差
2. 针对 ReLU 进行系数修正

简单初始化

- 对于简单的网络，可以用固定的分布随机初始化
- 例如 $Unif(-0.01, 0.01)$ 和 $N(0, 0.01^2)$
- 不能全部初始化为0!

对称初始化



If $w_1 = v_1$, $w_2 = v_2$, $w_3 = v_3$, $u_1 = u_2$, then $z_1 = z_2$, $a_1 = a_2$

$$u_1 = u_2 \Rightarrow \frac{dL}{da_1} = \frac{dL}{da_2}, \quad \frac{dL}{du_1} = \frac{dL}{du_2}$$

$$\frac{dL}{dz_1} = \frac{dL}{da_1} \cdot \sigma'(z_1) = \frac{dL}{da_2} \cdot \sigma'(z_2) = \frac{dL}{dz_2}$$

$$\frac{dL}{dw_1} = \frac{dL}{dz_1} \cdot \frac{dz_1}{dw_1} = \frac{dL}{dz_1} \cdot x_1 = \frac{dL}{dz_2} \cdot x_1 = \frac{dL}{dv_1}$$

...

$$\frac{dL}{dw_i} = \frac{dL}{dv_i}$$

结论

- 不光是不能将参数初始化为0
- 即使是对称也不行
- 需要使用随机初始化

初始化影响

- 对于高维、深层的网络，考虑 Xavier 和 Kaiming 初始化
- 见 `weight_initialization.ipynb`

激活函数

实践

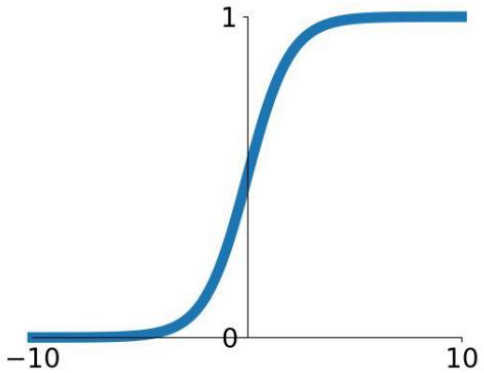
- 默认使用 ReLU
- 尝试使用 Leaky ReLU、ELU 等 ReLU 的变种
- 尽量避免 Sigmoid 和 Tanh（输出层除外）

原因

- 梯度饱和
- 以零为中心
- 计算复杂度

Sigmoid

- x 在 $[-5,5]$ 之外几乎是平坦取值
- 导数“饱和”了



回顾 BP

- $a = \sigma(z)$
- $\frac{\mathrm{d}l}{\mathrm{d}z} = \frac{\mathrm{d}l}{\mathrm{d}a} \circ \sigma'(z)$
- 当局部导数接近0时，下游导数也几乎为0

回顾 BP

- $a = \sigma(z)$

上游导数

$$\frac{\mathrm{d}l}{\mathrm{d}z} = \frac{\mathrm{d}l}{\mathrm{d}a} \circ \sigma'(z)$$

下游导数　　局部导数

- 当局部导数接近0时，下游导数也几乎为0

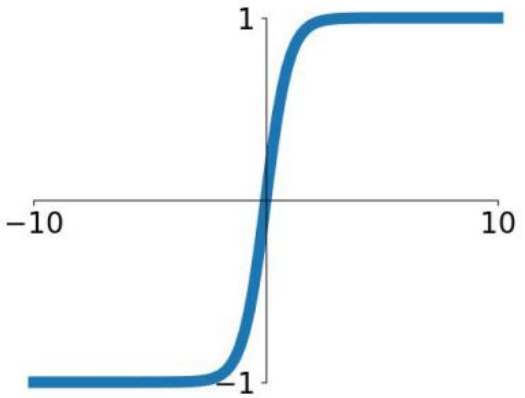
Sigmoid

- Sigmoid 的输出范围是 $[0,1]$
- 不是关于0对称的
- 如果输入永远是正的会发生什么?

- $z = Wx + b, \quad x \geq 0$
- $\frac{\mathrm{d}l}{\mathrm{d}W} = \frac{\mathrm{d}l}{\mathrm{d}z} x^T = \left[\frac{\mathrm{d}l}{\mathrm{d}a} \circ \overbrace{\sigma'(z)}^{\geq 0}\right] \overbrace{x^T}^{\geq 0}$
- W 导数的符号几乎被上游导数的符号制约

Tanh

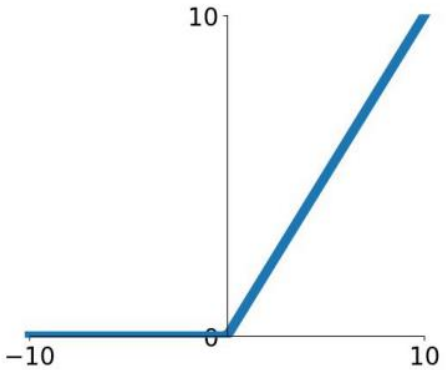
- Tanh 解决了输出范围中心化的问题
- 但导数饱和的问题依然存在



tanh(x)

ReLU

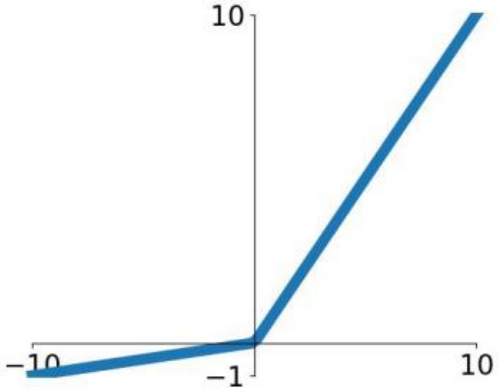
- 好处：
  - 在 >0 的区域永远有梯度
  - 计算简单
- 不足：
  - 值域非以0为中心
  - <0 时导数为0



ReLU
(Rectified Linear Unit)

Leaky ReLU

- 在所有区域都有梯度
- 计算简单
- 输出有正有负

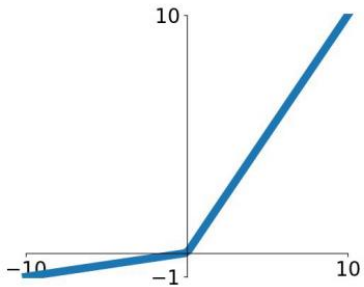


Leaky ReLU

$$f(x) = \max(0.01x, x)$$

PReLU

- 让 Leaky ReLU 中的系数变成可学习参数
- 反向传播时需计算 α 的导数并更新取值



Leaky ReLU

$$f(x) = \max(0.01x, x)$$

Parametric Rectifier (PReLU)

$$f(x) = \max(\alpha x, x)$$

特殊隐藏层

实践

1. 使用归一化层
2. 使用残差网络

原因

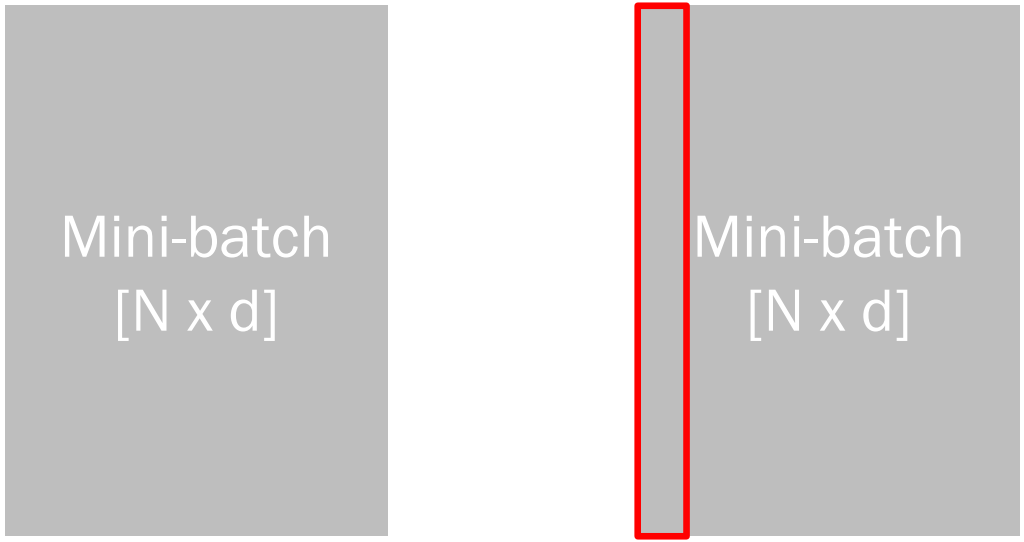
1. 保持一定的尺度不变性，有利于优化
2. 以线性函数为基础，拟合残差

归一化层

- 批量归一化（Batch normalization）
- 层归一化（Layer normalization）

批量归一化

- 类似于数据预处理中的中心化和标准化



计算每一列的均值和方差

批量归一化

- x_{ij} 表示第 j 个神经元的第 i 个观测
- $\mu_j = \frac{1}{N}\sum_{i=1}^{N} x_{ij}$, $\sigma_j^2 = \frac{1}{N}\sum_{i=1}^{N}(x_{ij} - \mu_j)^2$
- $\hat{x}_{ij} = \frac{x_{ij} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$ 防止变为常数
- 加入可学习参数向量 β, γ
- $z_{ij} = \gamma_j \hat{x}_{ij} + \beta_j$

计算每一列的均值和方差

与初始化相通

思考题

- BN 归一化的参数依赖于 mini-batch 的选择
- 对单个观测作预测时怎么计算?

- 计算所有观测的均值和方差
- 或在训练时计算均值和方差的滑动平均 adam
- 这里可能出现不一致性!

计算每一列的均值和方差

PyTorch

- https://pytorch.org/docs/stable/nn.html#normalization-layers

- PyTorch 中模型可以设置状态
- `model.train()` 用于训练
- `model.eval()` 用于预测

Batch Normalization for
fully-connected networks

$x: N \times D$

Normalize ↓

$\mu, \sigma: 1 \times D$

$\gamma, \beta: 1 \times D$

$y = \gamma(x-\mu)/\sigma+\beta$ 中心化

Batch Normalization for
convolutional networks
(Spatial Batchnorm, BatchNorm2D)

tensor

$x: N \times C \times H \times W$

Normalize ↓ ↓ ↓

$\mu, \sigma: 1 \times C \times 1 \times 1$

$\gamma, \beta: 1 \times C \times 1 \times 1$

$y = \gamma(x-\mu)/\sigma+\beta$

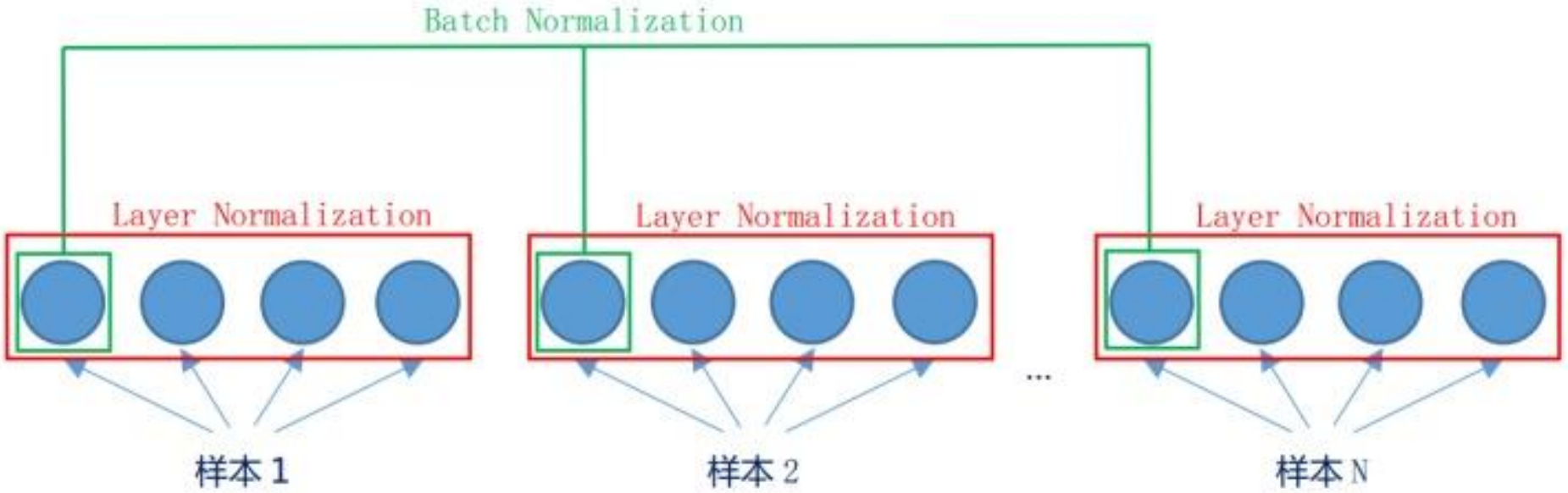
每个通道计算

层归一化

- 对每个观测的所有变量做归一化

批量归一化：
计算每一列的均值和方差

层归一化：
计算每一行的均值和方差


Batch Normalization
Layer Normalization
Layer Normalization
Layer Normalization
...
样本 1
样本 2
样本 N

残差网络

- 在早期的深度学习模型中出现反常的现象
- 深度网络比浅层网络的训练误差更大

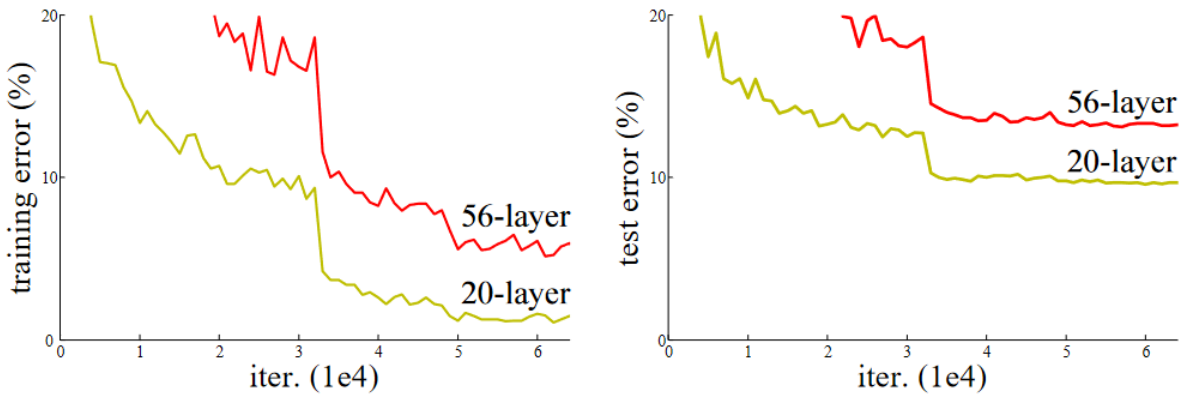


Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer "plain" networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.

残差网络

- 可能的原因是深层网络的目标函数更复杂、更难以优化
- 而线性函数是非常稳定的结构
- 残差网络的思想就是先用线性函数去拟合目标，再用非线性神经网络去拟合残差

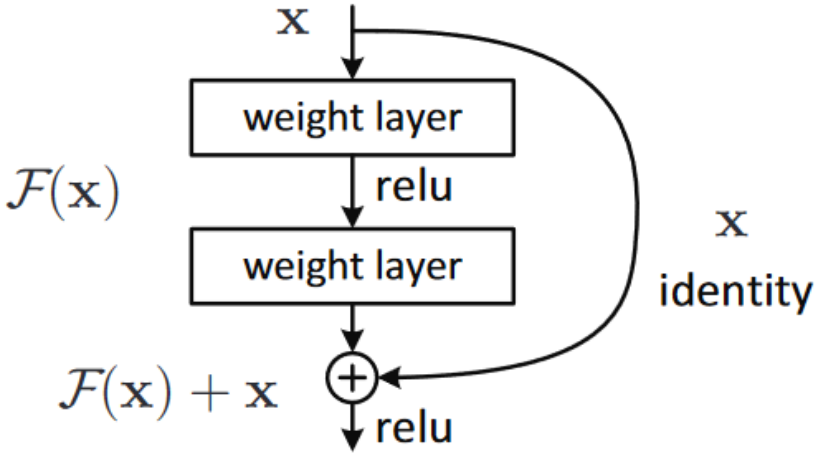


Figure 2. Residual learning: a building block.

残差网络

- 实现非常简单
- 但要注意维度的匹配

```
p = 3
d = 10
fc1 = nn.Linear(in_features=p, out_features=d)
fc2 = nn.Linear(in_features=d, out_features=p)

n = 5
x = torch.randn(n, p)

z1 = fc1(x)
a1 = torch.relu(z1)
z2 = fc2(a1)
z2_2 = x + z2
a2 = torch.relu(z2_2)
```

延伸阅读

- 李沐《精读论文》系列
- https://www.bilibili.com/video/BV1Fb4y1h73E
- https://www.bilibili.com/video/BV1P3411y7nn

延伸阅读

- 何恺明，2023年“未来科学大奖”
- https://www.bilibili.com/video/BV1Su4y1n7cw

正则化方法

实践

1. 目标函数中加入正则项
2. 使用提前停止、丢弃法等机制

原因

1. 减小过拟合风险
2. 增强模型泛化能力

模型泛化

- 神经网络往往具有非常多的参数
- 根据通用近似定理，按照一定的规则增加参数可以获得更好的拟合效果
- 但同时也增加了过拟合的风险
- 我们不希望网络只是记忆已知的数据
- 而是有一定的泛化能力

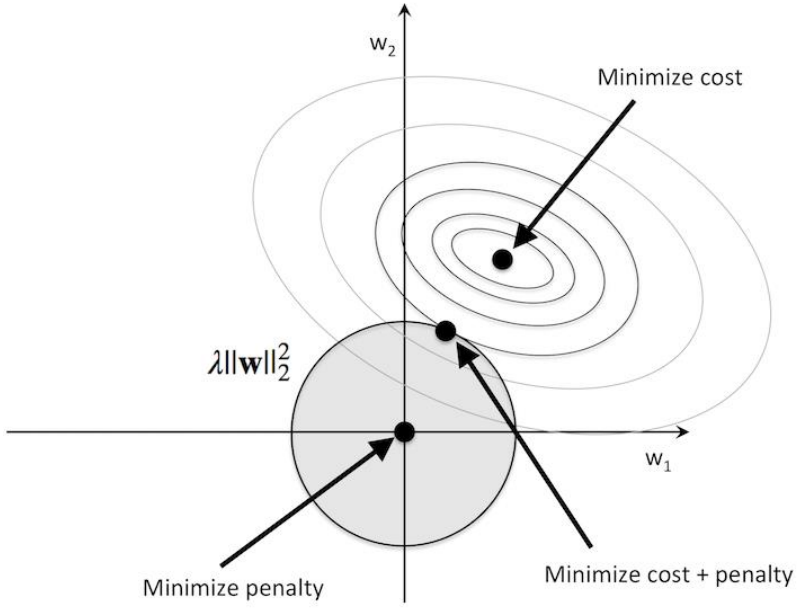
正则化

- 正则化可以认为是对模型的参数加入一些约束条件，使模型的复杂度降低

- 显式正则化：
  - 损失函数加入正则项
- 隐式正则化：
  - 提前停止（Early stopping）
  - 丢弃法（Dropout）

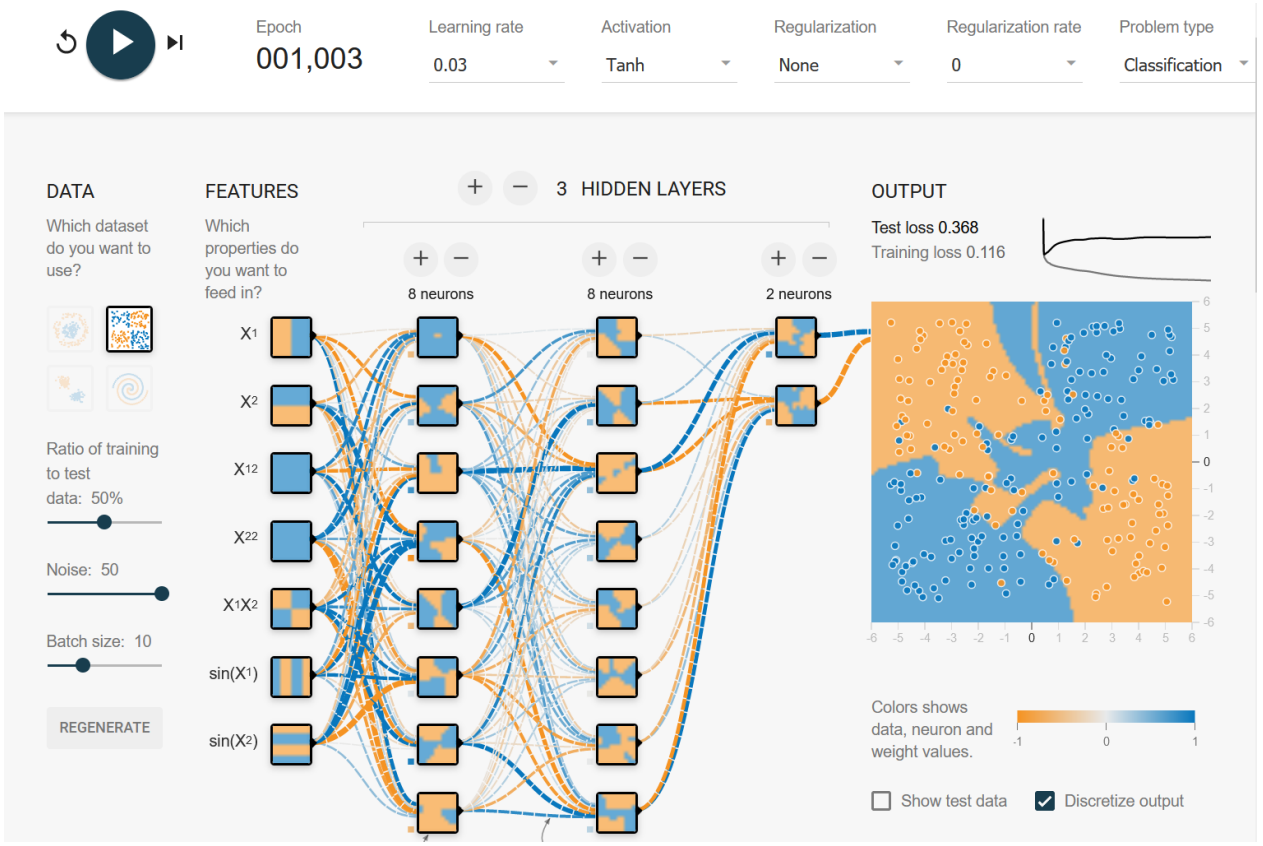
正则项

$$\theta^* = \arg\min_{\theta} \frac{1}{N} \sum_{n=1}^{N} \mathcal{L}\left(y^{(n)}, f(\mathbf{x}^{(n)}, \theta)\right) + \lambda \ell_p(\theta)$$



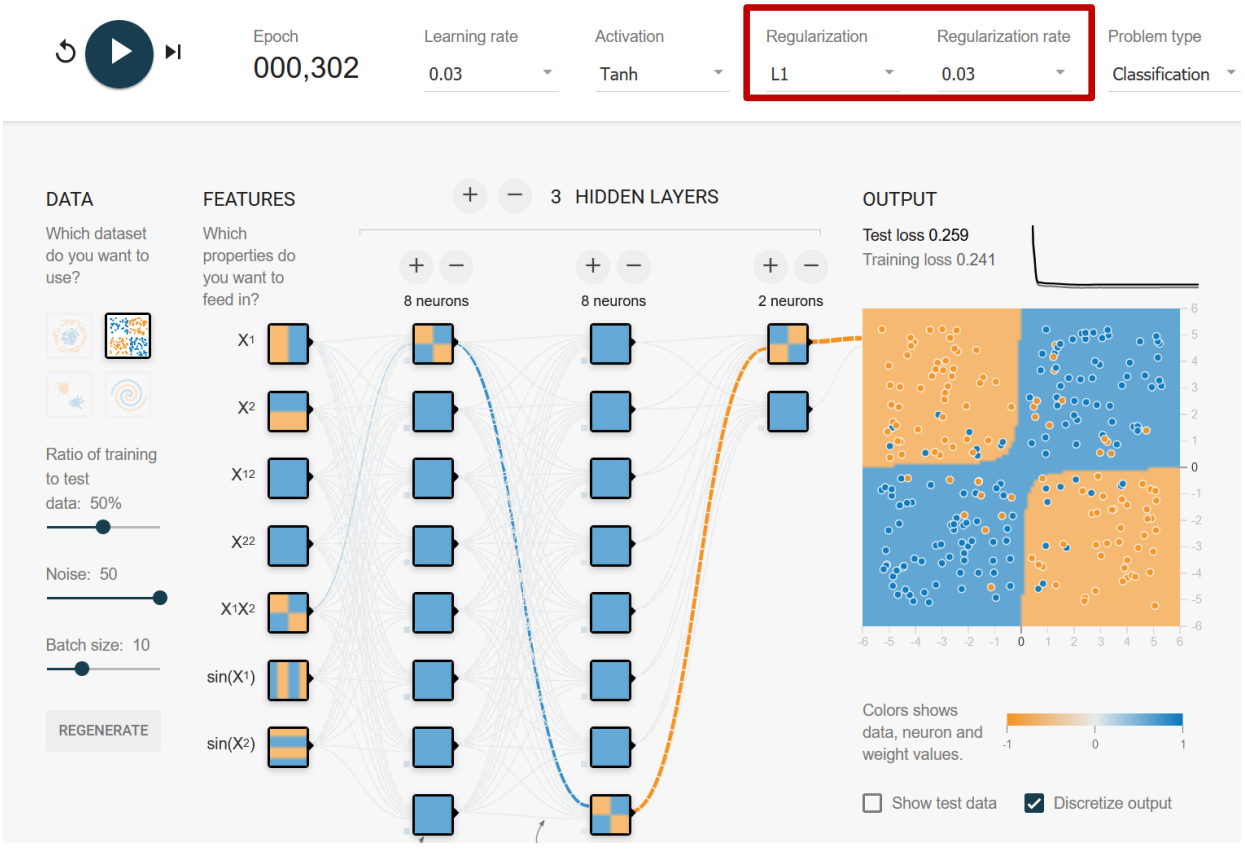
演示

- https://playground.tensorflow.org



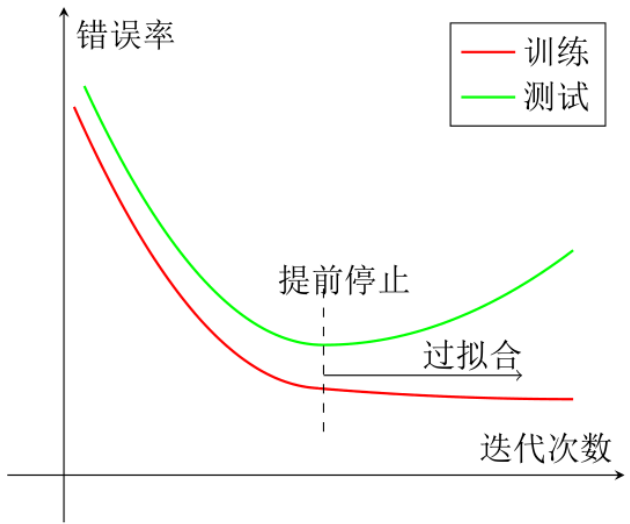
演示

- https://playground.tensorflow.org



提前停止

- 训练时划分出一个验证集，当验证集误差率不再下降时就停止迭代

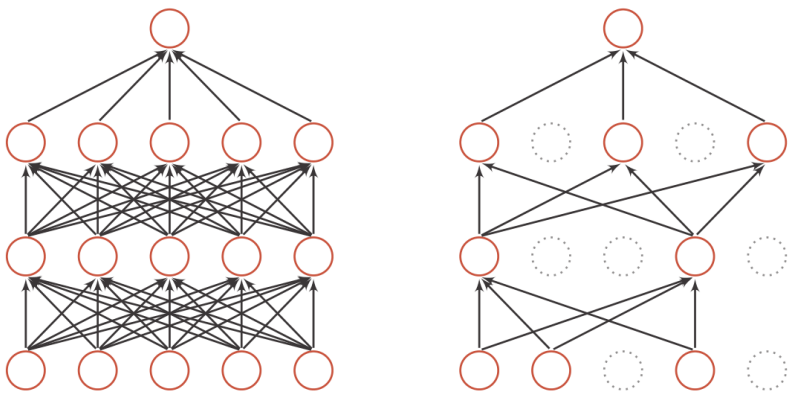


丢弃法

- 当数据通过隐藏层时，随机丢弃一些神经元
- $y = \sigma(Wd(x) + b)$

$$d(\mathbf{x}) = \begin{cases} \mathbf{m} \odot \mathbf{x} & \text{当训练阶段时} \\ p\mathbf{x} & \text{当测试阶段时} \end{cases}$$

- $m \in \{0,1\}^d$ 利用 Bernoulli 分布生成



(a) 标准网络　　(b) Dropout 后的网络

PyTorch

- https://pytorch.org/docs/stable/nn.html#dropout-layers

- PyTorch 中对于 Dropout 同样要注意设置模型状态
- `model.train()` 用于训练
- `model.eval()` 用于预测