第二章　深度学习运行全流程

作业布置 & 参考资料

- 实践作业：
  - 1. 自己尝试着运行代码，看看每一个变量都是什么，长什么样子
  - 2. 将模型放到GPU上训练(现在的代码只是用cpu训练的)
  - 3. 使用不同的学习率lr，看看学习率对模型训练效果的影响
  - 4. 尝试给模型换一个优化器 optimizer
- 参考资料：
  - 1. 推荐资料，PyTorch官方文档
  - 2. 工具推荐，GPU状态监控工具，nvidia-smi命令行指令；训练可视化工具TensorBoard（TensorBoard最全使用教程：看这篇就够了）

运行环境配置

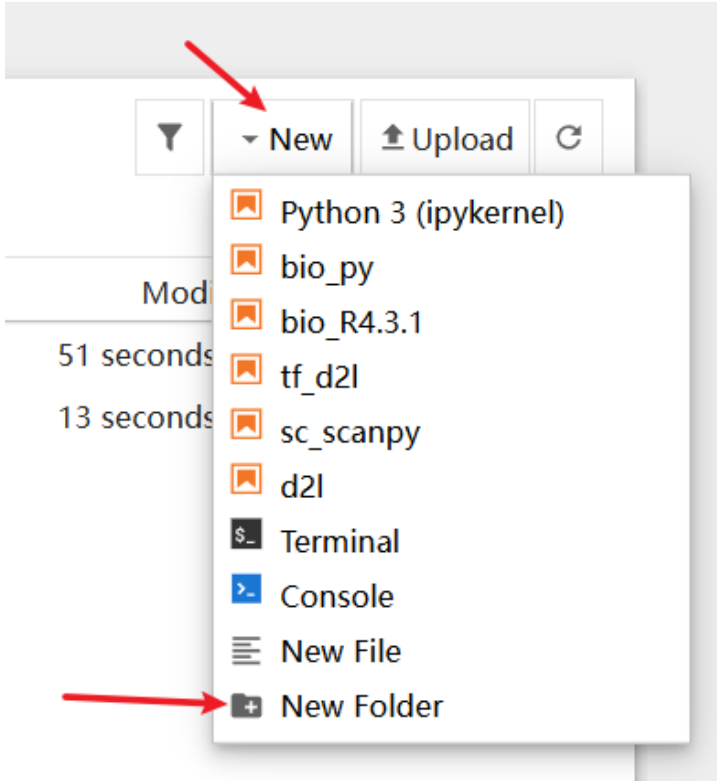
环境配置

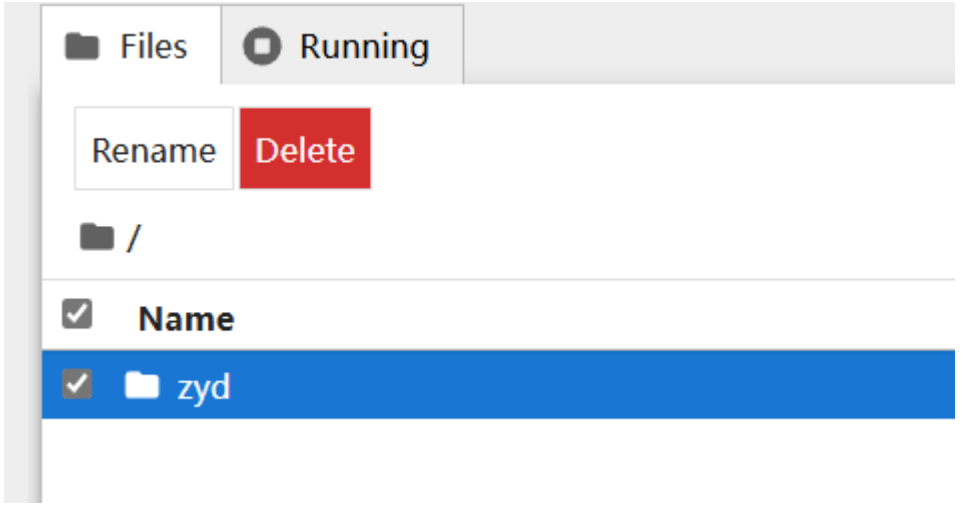
- conda create -n deeplearning python=3.11 -y
- conda activate deeplearning
- pip install torch==2.5.1 torchvision==0.20.1 torchaudio==2.5.1 --index-url https://download.pytorch.org/whl/cu121
- pip install ipykernel

- 有gpu服务器的可以自己配置，没有的用jupyter

运行环境账号

- 校园网
  - http://122.205.75.103:8888/
  - 密码: carrot

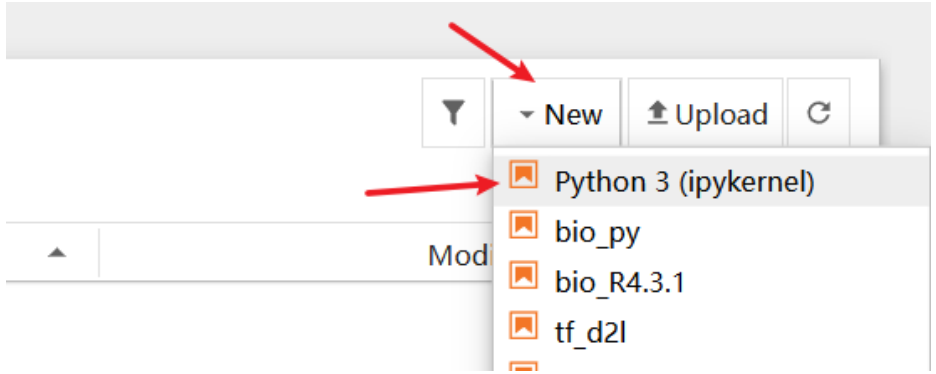




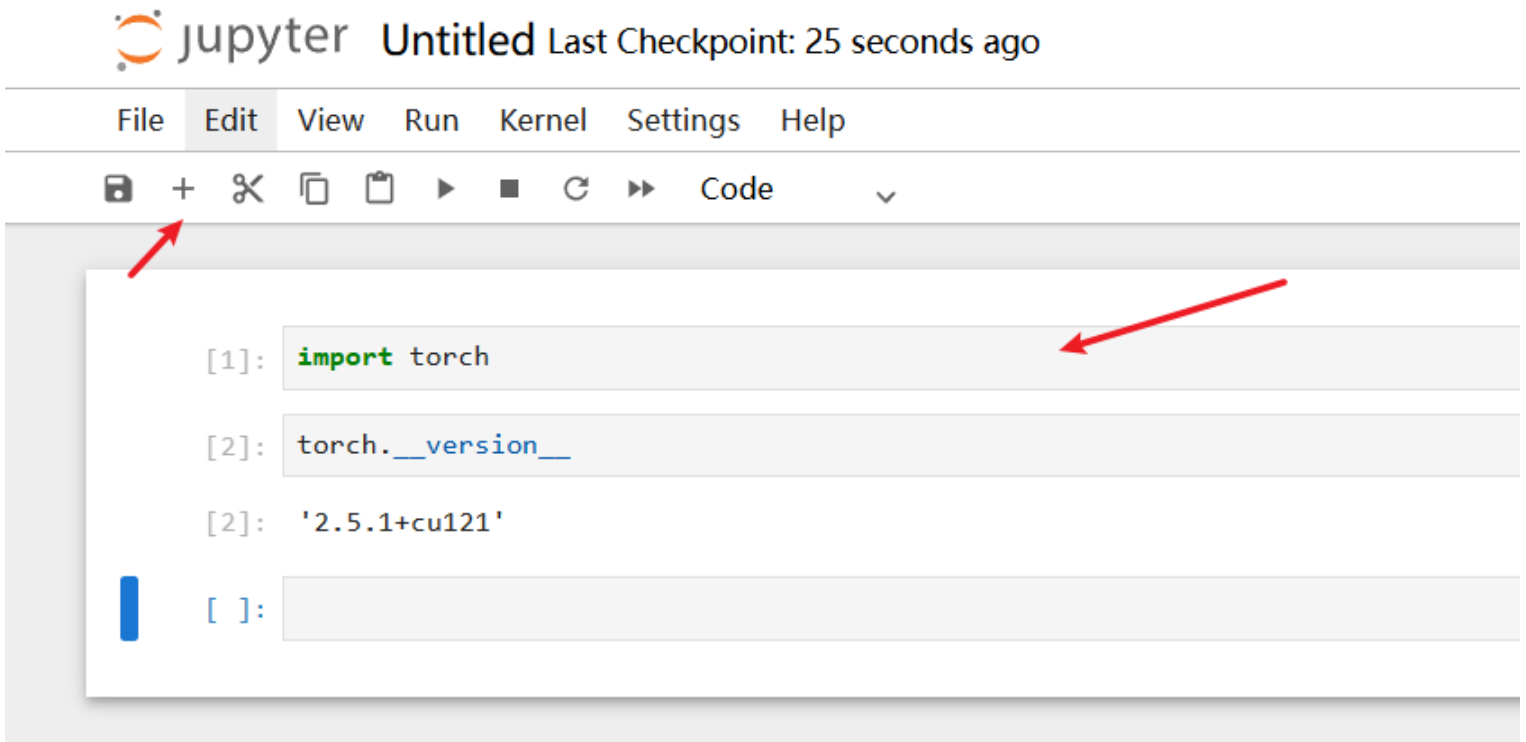
每个人在自己的目录里面写代码

创建属于自己的工作目录

创建ipynb文件



每个人都在 ipynb里面写代码，测试，看代码输出，方便debug

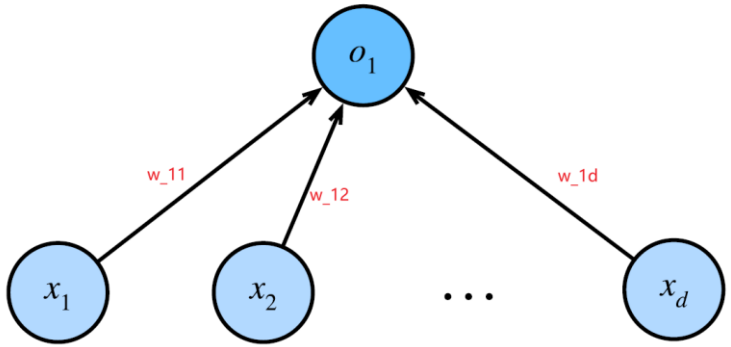


目标

- 1. 让完全不懂深度学习的同学对深度学习有一个直观感性的理解
- 2. 了解深度学习模型的训练数据处理流程(Dataset对象，Dataloader对象)
- 3. 了解模型是如何构建的
- 4. 直观理解深度学习模型的反向传播和梯度优化
- 5. 了解 深度学习模型整个训练过程，从载入数据，到定义loss，训练模型，保存模型，测试集评估
- 6. 理解什么是epoch，什么是step

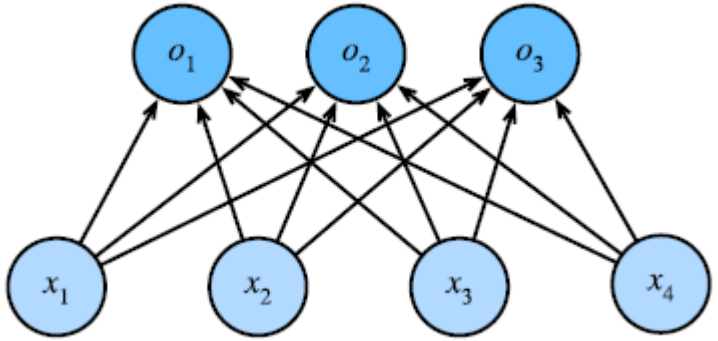
直观理解深度学习模型

一个神经元

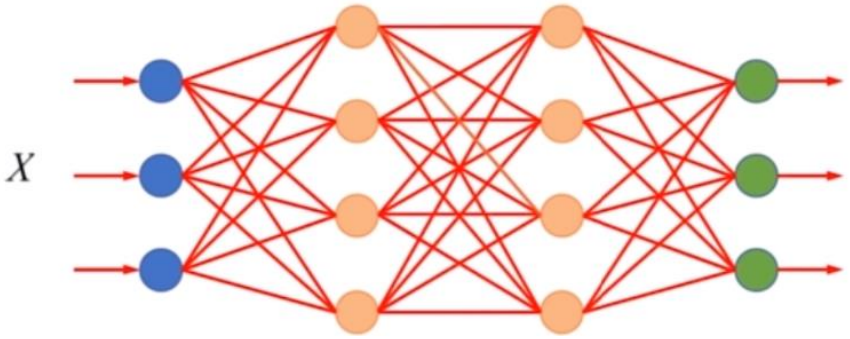


$o1 = w_{11} * x_1 + w_{12} * x_2 + ... + w_{1d} * x_d$

多个神经元组成一个层 layer

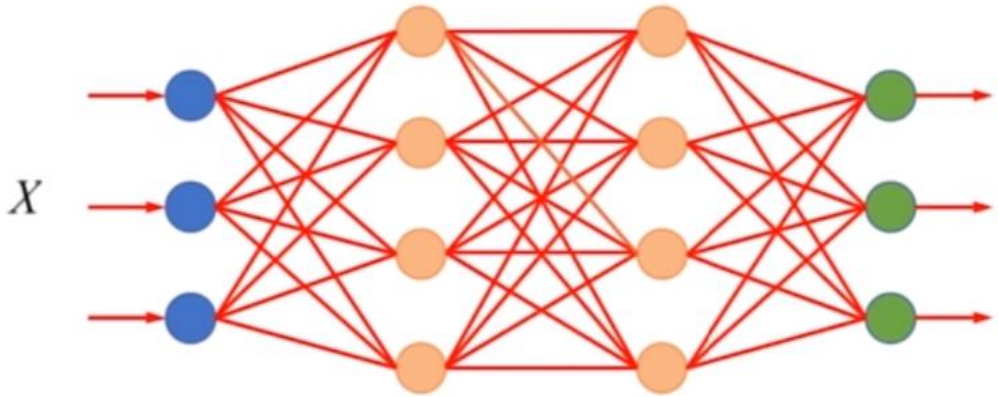


$$o_1 = x_1 w_{11} + x_2 w_{12} + x_3 w_{13} + x_4 w_{14} + b_1,$$
$$o_2 = x_1 w_{21} + x_2 w_{22} + x_3 w_{23} + x_4 w_{24} + b_2,$$
$$o_3 = x_1 w_{31} + x_2 w_{32} + x_3 w_{33} + x_4 w_{34} + b_3.$$



多个层组成一个神经网络

直观理解深度学习模型

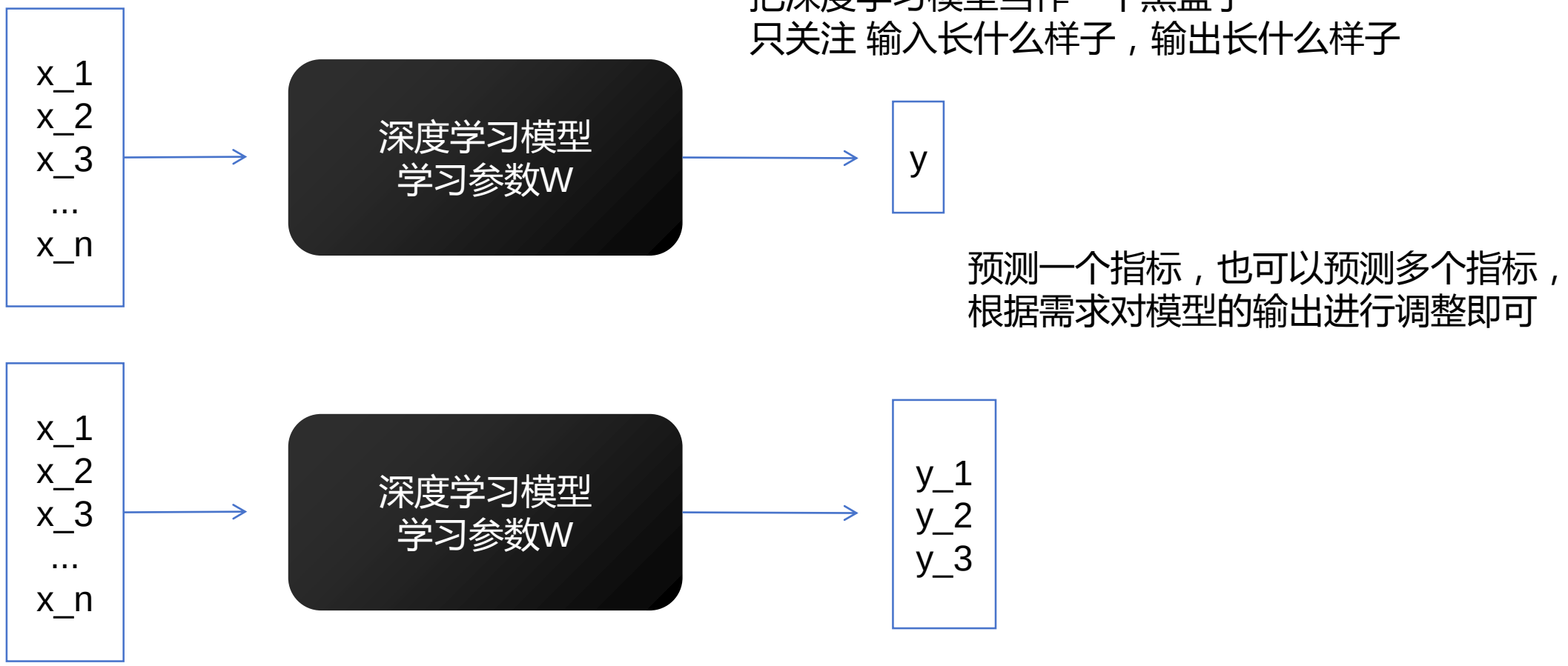


深度学习的目标，就是得到学习参数w

我们可以把这个过程当作一个线性回归的过程

$y = w_1 * x_1 + w_2 * x_2 + w_3 * x_3$

直观理解深度学习模型

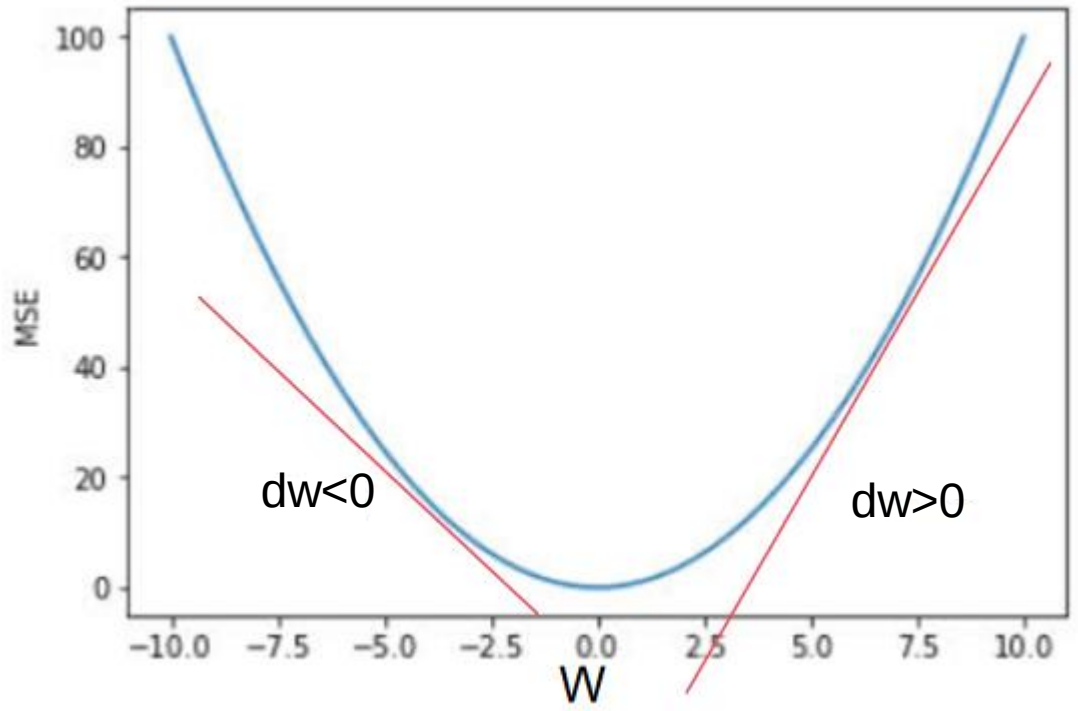

把深度学习模型当作一个黑盒子
只关注 输入长什么样子，输出长什么样子
x_1
x_2
x_3
...
x_n
深度学习模型
学习参数W
y
预测一个指标，也可以预测多个指标，
根据需求对模型的输出进行调整即可
x_1
x_2
x_3
...
x_n
深度学习模型
学习参数W
y_1
y_2
y_3

直观理解深度学习模型

y’ = f(X,W)

$$MSE = \frac{\sum_{i=1}^{n}(f(x)-y)^2}{n}$$

- **深度学习模型是如何学习参数W的?**
  - **损失函数 loss (均方误差MSE loss)**
  - **随机梯度下降 SGD**
- **我希望得到一个W，随便输入一个样本X，f(X,W)计算得到的y’和真实的y差异很小，这个差异用MSE衡量。**



固定一个样本X，W和MSE的函数曲线
关于W求导得到d_w
W_new = W - lr * d_w
这是针对一个样本的，叫做梯度下降，比较慢
随机取多个X(batch_size个)求loss的均值，对这个loss_avg求关于W的梯度，更新学习参数W，叫随机梯度下降

直观理解深度学习模型

房价预测的案例

是否有厨房	面积(m2)	多少个厕所	最近地铁站距离	房价
1	230	4	3000	500
0	90	1	100	100
1	180	1	200	300
1	270	2	5000	600
0	130	1	300	100
1	300	4	100	800
0	50	0	20	80

定义 损失函数为 MSE

$$MSE = \frac{\sum_{i=1}^{n}(f(x)-y)^2}{n}$$

让模型学习每一个样本，使用随机梯度下降SGD，更新模型的参数W，让模型输入任何X，预测得到的y' 和 真实y差距很小

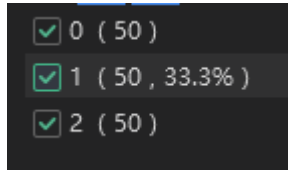
训练深度学习模型的要素

- 数据集
  - 训练集
  - 验证集
  - 测试集
- 模型结构
- 损失函数
  - 回归可以用 MSE
  - 分类可以用交叉熵
- 模型参数更新的算法 SGD

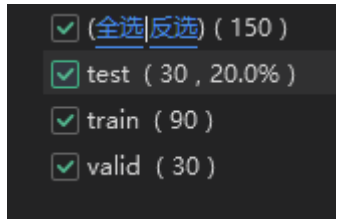
案例数据集描述

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species	label	split
1	5.1	3.5	1.4	0.2	setosa	0	train
2	4.9	3	1.4	0.2	setosa	0	train
3	4.7	3.2	1.3	0.2	setosa	0	train
4	4.6	3.1	1.5	0.2	setosa	0	train
5	5	3.6	1.4	0.2	setosa	0	train
6	5.4	3.9	1.7	0.4	setosa	0	train
7	4.6	3.4	1.4	0.3	setosa	0	train
8	5	3.4	1.5	0.2	setosa	0	train
9	4.4	2.9	1.4	0.2	setosa	0	train
10	4.9	3.1	1.5	0.1	setosa	0	train
11	5.4	3.7	1.5	0.2	setosa	0	train
12	4.8	3.4	1.6	0.2	setosa	0	train
13	4.8	3	1.4	0.1	setosa	0	train
14	4.3	3	1.1	0.1	setosa	0	train
15	5.8	4	1.2	0.2	setosa	0	train
16	5.7	4.4	1.5	0.4	setosa	0	train
17	5.4	3.9	1.3	0.4	setosa	0	train
18	5.1	3.5	1.4	0.3	setosa	0	train
19	5.7	3.8	1.7	0.3	setosa	0	train
20	5.1	3.8	1.5	0.3	setosa	0	train
21	5.4	3.4	1.7	0.2	setosa	0	train
22	5.1	3.7	1.5	0.4	setosa	0	train
23	4.6	3.6	1	0.2	setosa	0	train
24	5.1	3.3	1.7	0.5	setosa	0	train
25	4.8	3.4	1.9	0.2	setosa	0	train
26	5	3	1.6	0.2	setosa	0	train
27	5	3.4	1.6	0.4	setosa	0	train
28	5.2	3.5	1.5	0.2	setosa	0	train
29	5.2	3.4	1.4	0.2	setosa	0	train
30	4.7	3.2	1.6	0.2	setosa	0	train
31	4.8	3.1	1.6	0.2	setosa	0	valid
32	5.4	3.4	1.5	0.4	setosa	0	valid
33	5.2	4.1	1.5	0.1	setosa	0	valid

label



split



目标

- 训练一个深度学习模型

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species	label	split
1	5.1	3.5	1.4	0.2	setosa	0	train
2	4.9	3	1.4	0.2	setosa	0	train
3	4.7	3.2	1.3	0.2	setosa	0	train
4	4.6	3.1	1.5	0.2	setosa	0	train
5	5	3.6	1.4	0.2	setosa	0	train
6	5.4	3.9	1.7	0.4	setosa	0	train

X: [batch_size, 4]

Y: [batch_size, 3]

y_0就是这个样本属于第0类的概率

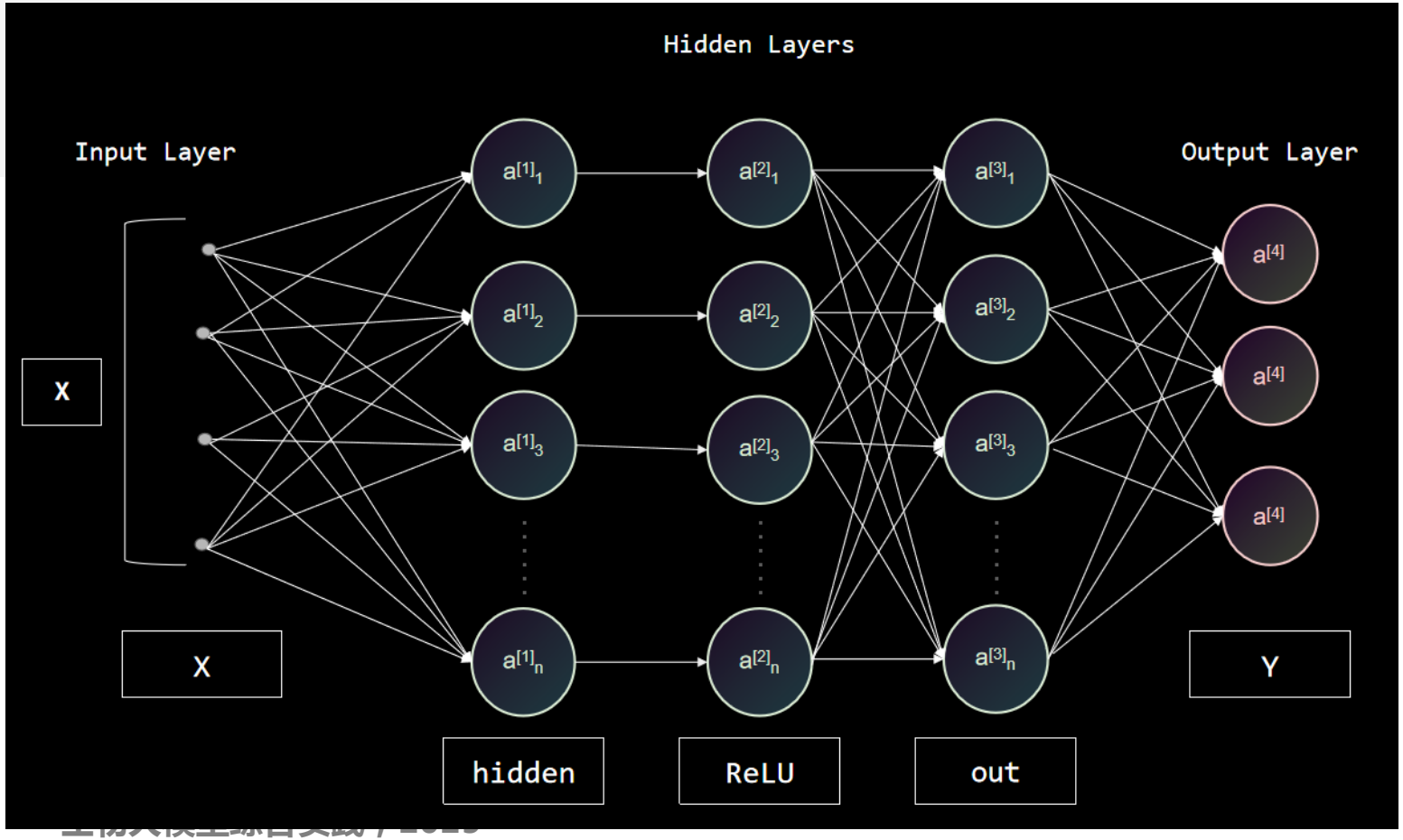
模型定义

```python
class Iris_classifier(torch.nn.Module):
    def __init__(self, n_feature, n_hidden, n_output):
        super(Iris_classifier, self).__init__()
        self.hidden = torch.nn.Linear(n_feature, n_hidden)   # 定义隐藏层网络
        self.relu = torch.nn.ReLU()
        self.out = torch.nn.Linear(n_hidden, n_output)   # 定义输出层网络

    def forward(self, x):
        x = self.relu(self.hidden(x))
        x = self.out(x)
        return x
```

ReLU

$$f(x) = \max(0, x)$$

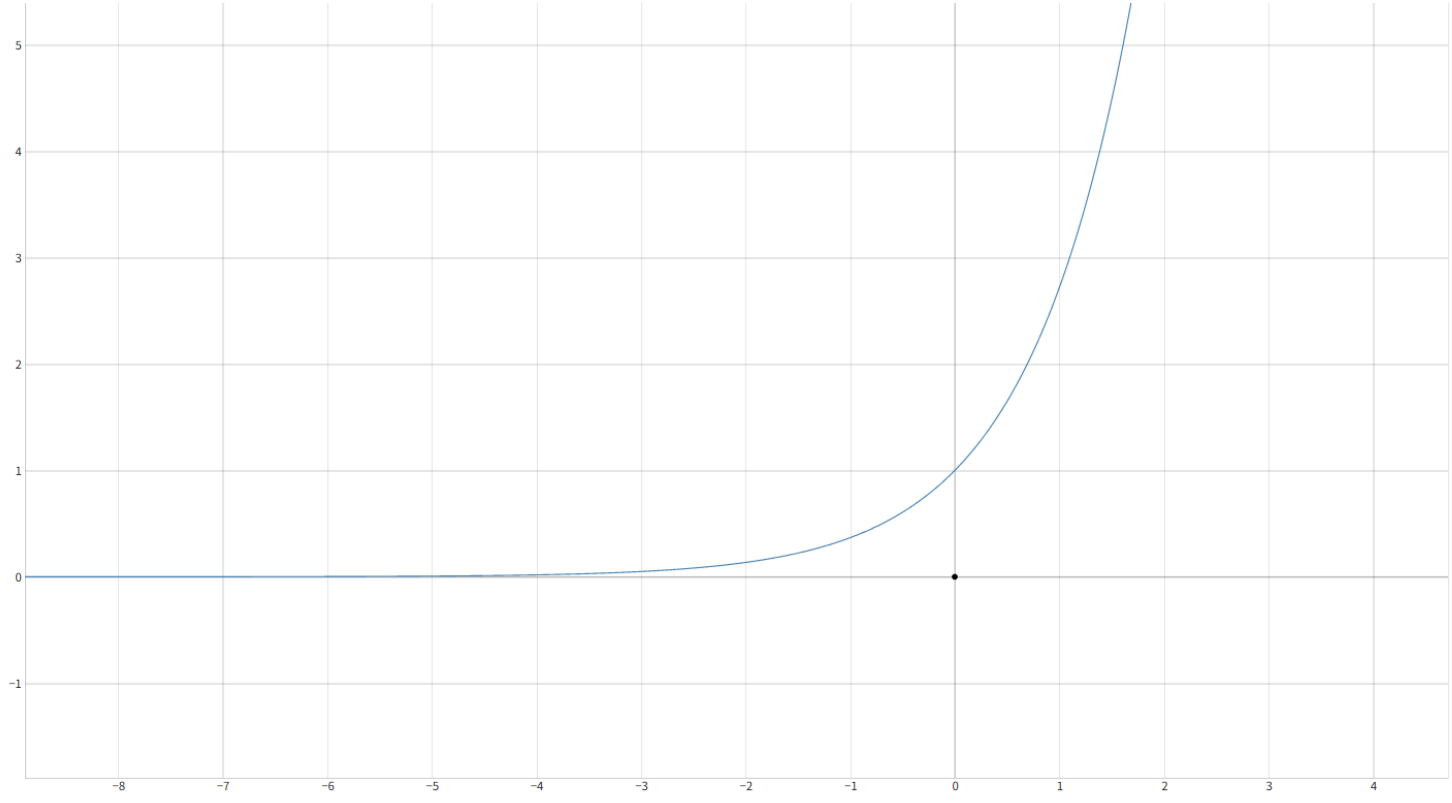


Softmax操作

- 模型输出的Y $[y_0, y_1, y_2]$ 称为 logits 值域$[-\infty, +\infty]$
- 如何将logits变成概率?
  - softmax

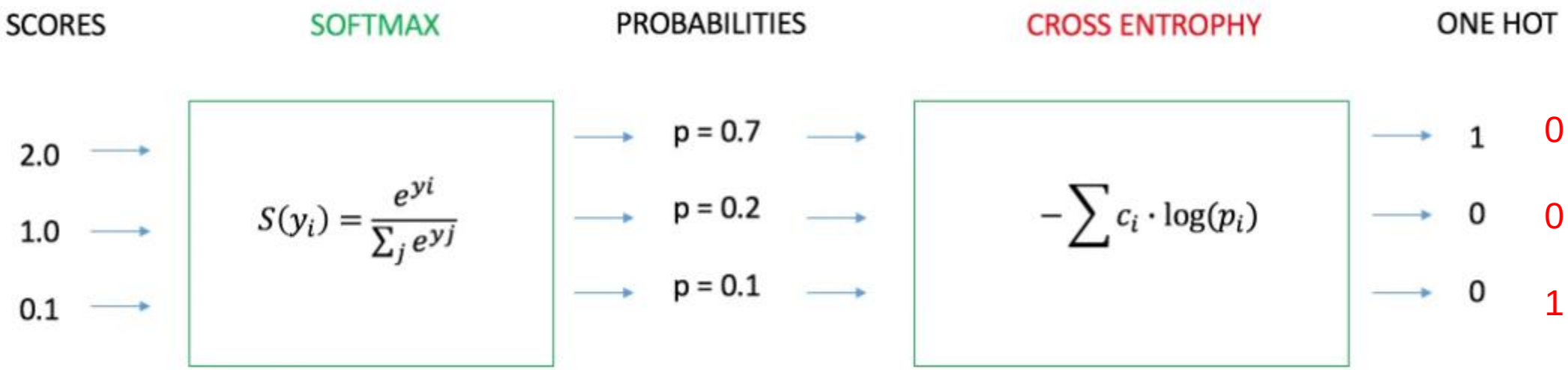
$$\hat{\mathbf{y}} = \mathrm{softmax}(\mathbf{o}) \quad \text{其中} \quad \hat{y}_j = \frac{\exp(o_j)}{\sum_k \exp(o_k)}$$

通过softmax，就能够将logits转化为该样本属于 哪一个品种的概率，用于计算loss



损失函数定义

分类问题，用**交叉熵损失**



-(1 * log(0.7) + 0 * log(0.2) + 0 * log(0.1)) = 0.3566749439387324 5 预测比较正确的时候

-(0 * log(0.7) + 0 * log(0.2) + 1 * log(0.1)) = 2.302585092994045 5 预测比较错误的时候

训练数据载入对象 Dataset

```python
class Iris_Dataset(torch.utils.data.Dataset):
    def __init__(self,df):
        # 初始化数据和标签
        self.df = df

    def __len__(self):
        # 返回数据集大小
        return self.df.shape[0]

    def __getitem__(self, idx):
        # 根据索引返回数据和标签
        x = self.df[["Sepal.Length","Sepal.Width","Petal.Length","Petal.Length"]].iloc[idx].to_numpy()
        y = self.df.iloc[idx]["label"].item()
        return x,y
```

df_test

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species	label	split
0	5.0	3.5	1.3	0.3	setosa	0	test
1	4.5	2.3	1.3	0.3	setosa	0	test
2	4.4	3.2	1.3	0.2	setosa	0	test
3	5.0	3.5	1.6	0.6	setosa	0	test
4	5.1	3.8	1.9	0.4	setosa	0	test
5	4.8	3.0	1.4	0.3	setosa	0	test
6	5.1	3.8	1.6	0.2	setosa	0	test
7	4.6	3.2	1.4	0.2	setosa	0	test
8	5.3	3.7	1.5	0.2	setosa	0	test
9	5.0	3.3	1.4	0.2	setosa	0	test

```python
test_ds = Iris_Dataset(df_test)
test_ds[0],test_ds[1]
```

```
((array([5. , 3.5, 1.3, 0.3]), 0), (array([4.5, 2.3, 1.3, 0.3]), 0))
```

训练数据载入对象 Dataloader

```
test_ds = Iris_Dataset(df_test)
test_dl = torch.utils.data.DataLoader(test_ds, shuffle = False, batch_size = 14)
```

```
for X,Y in test_dl:
    print(X.shape)
    print(Y.shape)
    print("#"*30)
```

```
torch.Size([14, 4])
torch.Size([14])
##############################
torch.Size([14, 4])
torch.Size([14])
##############################
torch.Size([2, 4])
torch.Size([2])
##############################
```

dataloader 就是对 dataset 进行进一步包装，能够控制是否要打乱样本顺序，控制batch_size的大小

训练集 验证集的dataloader设置shuffle为True，测试集设置为False

df_test 一共有30个样本
设置batch_size 为14的话，一共就有3个step

为什么要用dataset,dataloader包装一下? 为了规范

模型训练流程_超参数设置和各种对象定义

```
# 超参数设置
n_feature = 4 # 输入X的特征数目为4
n_hidden = 16 # 隐藏层的神经元数目
n_output = 3 # 输出3种类别的logits

batch_size = 8
learning_rate = 0.01

num_epoch = 100
```

```
# 定义网络对象
net = Iris_classifier(n_feature = n_feature,n_hidden = n_hidden,n_output=n_output)

# 定义训练集 和 验证集 的DataLoader对象
train_ds = Iris_Dataset(df_train)
valid_ds = Iris_Dataset(df_valid)
train_dl = torch.utils.data.DataLoader(train_ds, shuffle = True, batch_size = batch_size)
valid_dl = torch.utils.data.DataLoader(valid_ds, shuffle = True, batch_size = batch_size)

# 定义损失函数
criterion = torch.nn.CrossEntropyLoss()
# 定义优化器
optimizer = torch.optim.SGD(net.parameters(), lr=learning_rate)
```

模型训练流程_使用未经过训练的模型预测测试集

训练开始之前对测试集数据进行评估

```
test_ds = Iris_Dataset(df_test)
test_dl = torch.utils.data.DataLoader(test_ds, shuffle = False, batch_size = 14)
pred = []
real = []
with torch.no_grad():
    for X,Y in test_dl:
        X = X.float()
        Y = Y.long()
        y_pred = net(X).argmax(-1)
        real.append(Y.numpy())
        pred.append(y_pred.numpy())
real = np.concatenate(real)
pred = np.concatenate(pred)
print(f"还没有训练的时候,测试集的准确率:{get_accuracy(pred,real)}")
```

```
还没有训练的时候,测试集的准确率:0.3333333333333333
```

模型训练流程_训练逻辑

```
save_metrics_csv = "./metrics.csv"
with open(save_metrics_csv,"w") as f:
    f.write(f"{','.join(['epoch', 'train_loss', 'valid_loss', 'train_accuracy', 'valid_accuracy'])}\n")
for epoch in range(num_epoch):
    train_loss = []
    train_pred = []
    train_real = []
    valid_loss = []
    valid_pred = []
    valid_real = []
    for X,Y in train_dl:
        X = X.float()
        Y = Y.long()
        optimizer.zero_grad()
        y_pred = net(X)
        loss = criterion(y_pred,Y)
        loss.backward()
        optimizer.step()
        train_loss.append(loss.item())
        train_pred.append(y_pred.argmax(dim = 1).numpy())
        train_real.append(Y.numpy())
    with torch.no_grad():
        # 一个epoch训练完成以后，在验证集上测试一下
        for X,Y in valid_dl:
            X = X.float()
            Y = Y.long()
            y_pred = net(X)
            loss = criterion(y_pred,Y)
            valid_loss.append(loss.item())
            valid_pred.append(y_pred.argmax(-1).numpy())
            valid_real.append(Y.numpy())
    # 根据训练过程保存的数据计算指标，打印并且记录
    get_metrics(train_loss,valid_loss,train_pred,valid_pred,train_real,valid_real)
```

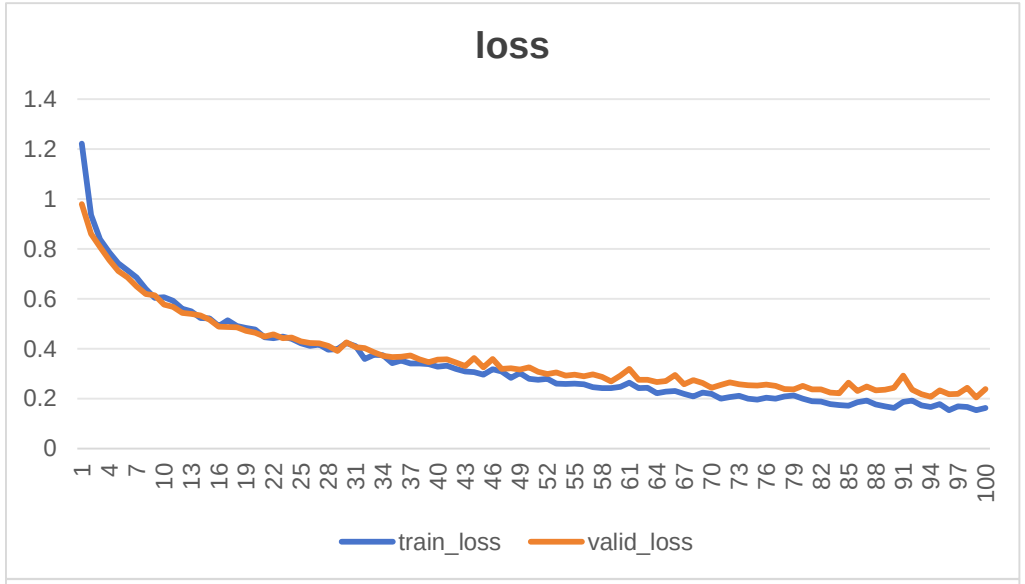
```
test_ds = Iris_Dataset(df_test)
test_dl = torch.utils.data.DataLoader(test_ds, shuffle = False, batch_size = 14)
pred = []
real = []
with torch.no_grad():
    for X,Y in test_dl:
        X = X.float()
        Y = Y.long()
        y_pred = net(X).argmax(-1)
        real.append(Y.numpy())
        pred.append(y_pred.numpy())
real = np.concatenate(real)
pred = np.concatenate(pred)
print(f"模型训练后,测试集的准确率:{get_accuracy(pred,real)}")
```

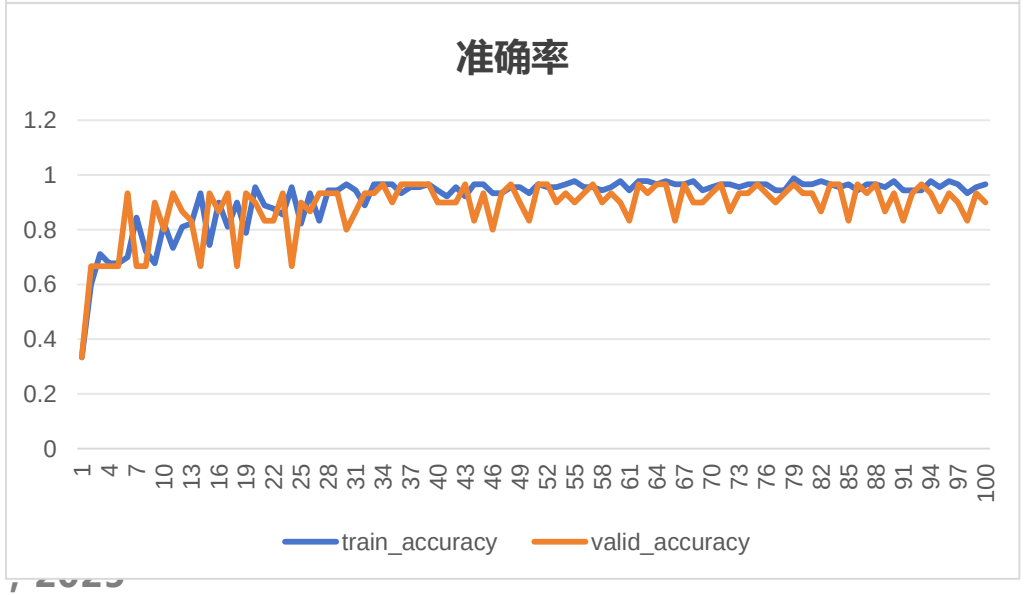
```
模型训练后,测试集的准确率:0.9666666666666667
```

模型训练流程_训练指标可视化

Delimiter: ,

	epoch	train_loss	valid_loss	train_accuracy	valid_accuracy
1	0	1.222166230281194	0.9801111221313477	0.3333333333333333	0.3333333333333333
2	1	0.9364276627699534	0.8601184338331223	0.6	0.6666666666666666
3	2	0.8391327361265818	0.8075671494007111	0.7111111111111111	0.6666666666666666
4	3	0.7862534274657568	0.7547780871391296	0.6777777777777778	0.6666666666666666
5	4	0.7421350280443827	0.7123619467020035	0.6777777777777778	0.6666666666666666
6	5	0.7138084471225739	0.6856103539466858	0.7	0.9333333333333333
7	6	0.6855766077836355	0.6498307883739471	0.8444444444444444	0.6666666666666666
8	7	0.6404783030351003	0.6198966354131699	0.7222222222222222	0.6666666666666666
9	8	0.603975348174572	0.614330604672432	0.6777777777777778	0.9
10	9	0.6071022798617681	0.577733501791954	0.8222222222222222	0.8
11	10	0.5928880572319031	0.5679693296551704	0.7333333333333333	0.9333333333333333
12	11	0.5599714989463488	0.543851837515831	0.8111111111111111	0.8666666666666667
13	12	0.5496179560820261	0.5394100025296211	0.8222222222222222	0.8333333333333334
14	13	0.5238898992538452	0.5337138548493385	0.9333333333333333	0.6666666666666666
15	14	0.521927073597908	0.5163911730051041	0.7444444444444445	0.9333333333333333
16	15	0.49145840605099994	0.4882262796163559	0.9	0.8666666666666667
17	16	0.513971596956253	0.4874909296631813	0.8111111111111111	0.9333333333333333
18	17	0.49155804763237637	0.48570484668016434	0.9	0.6666666666666666
19	18	0.48374559730291367	0.4723152667284012	0.7888888888888889	0.9333333333333333
20	19	0.47737079362074536	0.4647085443139076	0.9555555555555556	0.9





模型训练流程_保存模型和载入模型

保存训练后的模型 ¶

[15]:
```
pth_path = './model_iris_classifier.pth'
# 保存模型参数
torch.save(net.state_dict(), pth_path)
```

载入模型进行预测

[16]:
```
pth_path = './model_iris_classifier.pth'
n_feature = 4 # 输入X的特征数目为4
n_hidden = 16 # 隐藏层的神经元数目
n_output = 3 # 输出3种类别的logits
model = Iris_classifier(n_feature,n_hidden,n_output) # 创建一个新的模型实例，Net()是模型的结构
model.load_state_dict(torch.load(pth_path,weights_only=True))
model.eval()
```

[16]:
```
Iris_classifier(
  (hidden): Linear(in_features=4, out_features=16, bias=True)
  (relu): ReLU()
  (out): Linear(in_features=16, out_features=3, bias=True)
)
```

如果我们要将模型放到cpu/gpu上运行

```
test_ds = Iris_Dataset(df_test)
test_dl = torch.utils.data.DataLoader(test_ds, shuffle = False, batch_size = 14)
pred = []
real = []
with torch.no_grad():
    for X,Y in test_dl:
        X = X.float()
        Y = Y.long()
        y_pred = model(X).argmax(-1)
        real.append(Y.numpy())
        pred.append(y_pred.numpy())
real = np.concatenate(real)
pred = np.concatenate(pred)
print(f"模型训练后,测试集的准确率:{get_accuracy(pred,real)}")
```

```
模型训练后,测试集的准确率:0.9666666666666667
```

cpu

gpu

```
test_ds = Iris_Dataset(df_test)
test_dl = torch.utils.data.DataLoader(test_ds, shuffle = False, batch_size = 14)
model.to("cuda")
pred = []
real = []
for X,Y in test_dl:
    X = X.float()
    Y = Y.long()
    X = X.to("cuda")
    Y = Y.to("cuda")
    y_pred = model(X).argmax(-1)
    real.append(Y.cpu().numpy())
    pred.append(y_pred.cpu().numpy())
real = np.concatenate(real)
pred = np.concatenate(pred)
print(f"测试集的准确率:{get_accuracy(pred,real)}")
```

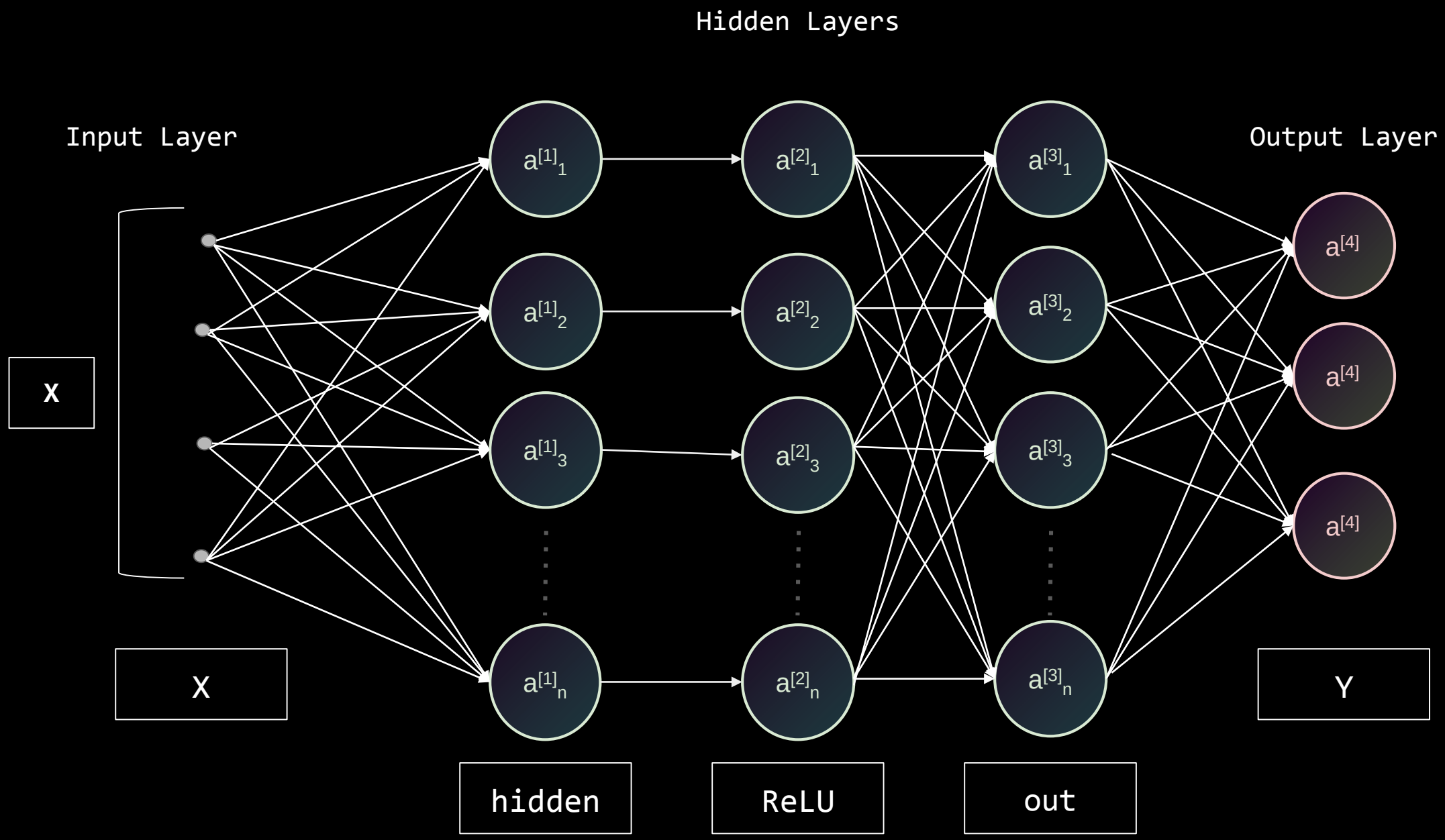
```
测试集的准确率:0.9666666666666667
```

为什么要 .cpu().numpy()
将数据从gpu中迁移到内存里，节约显存

课程总结

- 1. 了解到什么是损失函数，知道了深度学习模型训练的原理(模型如何根据样本更新参数)
- 2. 知道了 dataset 和 dataloader都有什么用 (其实我们也可以自己写这个数据生成的逻辑，不过用别人包装好的更加规范)
- 3. 知道了训练深度学习模型的一个通用的流程(定义模型，定义损失函数，定义优化器，定义dataset和dataloader，将这些“组件”拼成一个训练流程)
- 4. 知道在训练过程中如何计算梯度(loss.backward())，如何更新模型参数(optimizer.step())
- 5. 知道了如何在训练过程计算并保存各种评估指标(比如准确率，记录loss等)
- 6. 知道了如何保存模型，载入模型，如何将模型放到gpu上训练和推理

ipynb使用演示


Hidden Layers
Input Layer
Output Layer
x
X
Y
$a^{[1]}_1$
$a^{[1]}_2$
$a^{[1]}_3$
$a^{[1]}_n$
$a^{[2]}_1$
$a^{[2]}_2$
$a^{[2]}_3$
$a^{[2]}_n$
$a^{[3]}_1$
$a^{[3]}_2$
$a^{[3]}_3$
$a^{[3]}_n$
$a^{[4]}$
$a^{[4]}$
$a^{[4]}$
hidden
ReLU
out