第二次汇报-jax内容学习（含下次规划）-1022-1023

下次学习内容

- [ ] 熟练使用flax框架、代码敲一遍框架全部代码
- [ ] https://github.com/ikostrikov/jaxrl
- [ ] 想跟着这个过一遍，不知道怎么样

学习内容22-23

- jax特性
- 特性jax.numpy、jax.grad、random
- pytree数据格式学习
- 简单MLP流程熟悉

jax特性

1. jnp.arange(size) 创建的数组**不可改变内容**
2. jax.jit() 将方法更改为纯函数方法，加速函数的执行
3. jax.grad() 对函数进行求导，默认输入参数为自变量
4. grad和jit可以组合

常见库使用方法

PYTHON

```python
import jax
import jax.numpy as jnp

from jax import grad, jit, vmap, pmap
from jax import random
```

首先理解random随机数的使用

jax.random与numpy中随机数的使用区别

jax.numpy与numpy的数组使用几乎可以完美适配，**但jax中数组值不可更改。**

```python
# In JAX we have to deal with immutable arrays
x = jnp.arange(size)
print(x)
x[index] = value
print(x)
```

```
[0 1 2 3 4 5 6 7 8 9]
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[12], line 4
      2 x = jnp.arange(size)
      3 print(x)
----> 4 x[index] = value
      5 print(x)

File ~/anaconda3/lib/python3.11/site-packages/jax/_src/numpy/array_methods.py:587, in _unimplemented_setitem(self, i, x)
    582 def _unimplemented_setitem(self, i, x):
    583   msg = ("'{}' object does not support item assignment. JAX arrays are "
    584          "immutable. Instead of ``x[idx] = y``, use ``x = x.at[idx].set(y)`` "
    585          "or another .at[] method: "
    586          "https://jax.readthedocs.io/en/latest/_autosummary/jax.numpy.ndarray.at.html")
--> 587   raise TypeError(msg.format(type(self)))
```

grad

求导函数，可以自动求导

jit

这是一个比较重要的内容，类似于将一个函数实现某一定的功能，转化为一个纯函数（只进行输入，操作，输出）从而实现函数运行过程中的加速功能。
使用时要将需要改变的参数显示的传入才可以

pytree数据格式

pytree是一种jax中的数据格式，主要应用与列表，元组，字典中数据的提取，其中**对于类中的定义数据无法提取**所以对于类的中的数据我们需要扁平化处理，才可以操作使用
并且将每一个提取到的基础数据作为叶子节点

```
# A contrived example for pedagogical purposes
# (if your mind needs to attach some semantics to parse this - treat it as model params)
pytree_example = [
        [1, 'a', object()],
        (1, (2, 3), ()),
        [1, {'k1': 2, 'k2': (3, 4)}, 5],
        {'a': 2, 'b': (2, 3)},
        jnp.array([1, 2, 3]),
]

# Let's see how many leaves they have:
for pytree in pytree_example:
        leaves = jax.tree.leaves(pytree)    # handy little function
        print(f"{repr(pytree):<45} has {len(leaves)} leaves: {leaves}")
```

```
[1, 'a', <object object at 0x78b3ac996e80>]   has 3 leaves: [1, 'a', <object object at 0x78b3ac996e80>]
(1, (2, 3), ())                               has 3 leaves: [1, 2, 3]
[1, {'k1': 2, 'k2': (3, 4)}, 5]               has 5 leaves: [1, 2, 3, 4, 5]
{'a': 2, 'b': (2, 3)}                         has 3 leaves: [2, 2, 3]
Array([1, 2, 3], dtype=int32)                 has 1 leaves: [Array([1, 2, 3], dtype=int32)]
```

像对于lies，dict，tuple中的数据都会被一一取出

其中数据为取出，因为array算一个类

另外，对于pytree数据格式shi'y时数据一定要对齐。

```
# PyTrees need to have the same structure if we are to apply tree_multimap!
another_list_of_lists = deepcopy(list_of_lists)
another_list_of_lists.append([23])
print(jax.tree.map(lambda x, y: x+y, list_of_lists, another_list_of_lists))
```

数据未对齐

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-15-60f0bc744e02> in <cell line: 4>()
      2 another_list_of_lists = deepcopy(list_of_lists)
      3 another_list_of_lists.append([23])
----> 4 print(jax.tree.map(lambda x, y: x+y, list_of_lists, another_list_of_lists))

2 frames
/usr/local/lib/python3.10/dist-packages/jax/_src/tree_util.py in <listcomp>(.0)
    341   """Alias of :func:`jax.tree.map`."""
    342   leaves, treedef = tree_flatten(tree, is_leaf)
--> 343   all_leaves = [leaves] + [treedef.flatten_up_to(r) for r in rest]
    344   return treedef.unflatten(f(*xs) for xs in zip(*all_leaves))
    345

ValueError: List arity mismatch: 5 != 4; list: [{'a': 3}, [1, 2, 3], [1, 2], [1, 2, 3, 4], [23]].
```

```
#为了实现类中数据的pytree操作我们需要扁平化(flatten)和非扁平化处理(unflatten):将有效数据从类中提取出来
def flatten_MyContainer(container):传入类
    """Returns an iterable over container contents, and aux data."""
    flat_contents = [container.a, container.b, container.c]

    # we don't want the name to appear as a child, so it is auxiliary data.
    # auxiliary data is usually a description of the structure of a node,
    # e.g., the keys of a dict -- anything that isn't a node's children.
    aux_data = container.name

    return flat_contents, aux_data

def unflatten_MyContainer(aux_data, flat_contents):
    """Converts aux data and the flat contents into a MyContainer."""
    return MyContainer(aux_data, *flat_contents)

# Register a custom PyTree node
jax.tree_util.register_pytree_node(MyContainer, flatten_MyContainer, unflatten_MyContainer)
```

提取类中有效数据

使用要点：

1. 在tree.map中只有同类型的数据才可以相加
2. 对于pytree数据格式使用时数据一定要对齐。
3. 为了实现类中数据的pytree操作我们需要扁平化(flatten)和非扁平化处理(unflatten):将有效数据从类中提取出来

最最最简单的MLP编写

首先神经网络的编写流程：

1. 初始化参数 kernel、baise | 激活函数|损失函数的确定

2. 向前传播（输入参数得到输出的步骤）
3. 通过比对预测值与实际值的偏差，来确定梯度，从而确定如何优化 | 损失函数
4. 更新参数，多次数据输入，模型收敛

初始化参数：

初始化参数

```
def init_mlp_params(layer_widths):
        params = []

        # Allocate weights and biases (model parameters)
        # Notice: we're not using JAX's PRNG here - doesn't matter for this simple example
        for n_in, n_out in zip(layer_widths[:-1], layer_widths[1:]): #创建全连接层权重，以及偏置值biase
                params.append(
                        dict(weights=np.random.normal(size=(n_in, n_out)) * np.sqrt(2/n_in),
                                biases=np.ones(shape=(n_out,))
                )
        )

        return params

# Instantiate a single input - single output, 3 layer (2 hidden layers) deep MLP
params = init_mlp_params([1, 128, 128, 1])
```

得到预测值的过程：

```
def forward(params, x):
        *hidden, last = params

        for layer in hidden:
                x = jax.nn.relu(jnp.dot(x, layer['weights']) + layer['biases'])

        return jnp.dot(x, last['weights']) + last['biases']
```

损失函数：

损失函数：评判预测值与实际值的误差有多大

```
def loss_fn(params, x, y):
        return jnp.mean((forward(params, x) - y) ** 2)    # MSE loss
```

预测值　　实际值

更新参数：

```
此函数已使用jit加速，请把他当作一个纯函数使用
@jit  # notice how we do jit only at the highest level - XLA will have plenty of space to optimize
def update(params, x, y):

    # Note that grads is a pytree with the same structure as params.
    # grad is one of the many JAX functions that has built-in support for pytrees!
    grads = jax.grad(loss_fn)(params, x, y)   此处求的是梯度，也就是收敛方向

    # Task: analyze grads and make sure it has the same structure as params

    # SGD update
    注意使用了tree，数据一定要对齐                    根据梯度方向，更新参数
    return jax.tree.map(
        lambda p, g: p - lr * g, params, grads   # for every leaf i.e. for every param of MLP
    )
```

训练：

```
xs = np.random.normal(size=(128, 1))
ys = xs ** 2   # let's learn how to regress a parabola

# Task experiment a bit with other functions (polynomials, sin, etc.)

num_epochs = 500
for _ in range(num_epochs):
    params = update(params, xs, ys)   # again our lovely pattern

plt.scatter(xs, ys)
plt.scatter(xs, forward(params, xs), label='Model prediction')
plt.legend();
```