2020_jax_AD

October 15, 2020

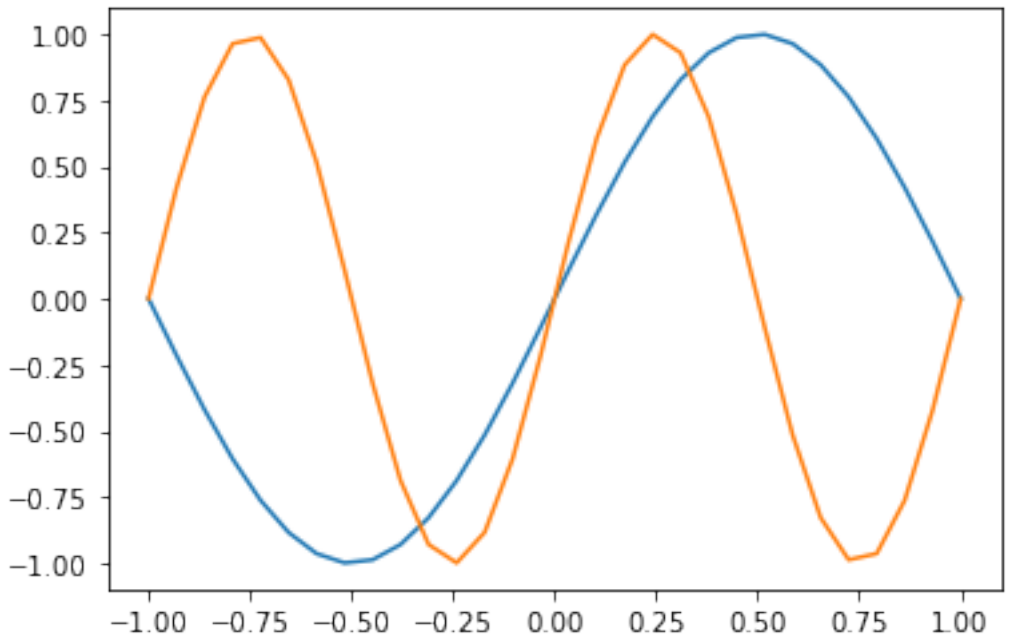
[1]:
```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import cm
import jax.numpy as jnp
from jax import grad, vmap, random
```

[2]:
```
# %env JAX_PLATFORM_NAME=cpu
# If we uncomment the magic env line above, you won't
# see the error of not finding a GPU/TPU  when you first call JAX (below)
```


In ordinary numpy we can define an array X and evaluate a numpy native function on X

[3]:
```
Xn=np.linspace(-1
y1=np.sin(Xn*(np.
y2=np.sin(Xn*(2*n
plt.plot(Xn,y1)
plt.plot(Xn,y2)
```


[3]:
```
[<matplotlib.lines.Line2D at 0x7fc9ad79b340>]
```



In JAX numpy we can do the same

[4]:
```
def wsin(w, xin):
    return jnp.array(jnp.sin(w*xin))
Xj=jnp.linspace(-1,1,30)

plt.plot(Xj, wsin(1., np.pi*Xj))
plt.plot(Xj, wsin(2., np.pi*Xj))
```

```
/opt/anaconda3/lib/python3.8/site-packages/jax/lib/xla_bridge.py:130:
UserWarning: No GPU/TPU found, falling back to CPU.
  warnings.warn('No GPU/TPU found, falling back to CPU.')
```

[4]:
```
[<matplotlib.lines.Line2D at 0x7fc9ae0b7c10>]
```


[5]:
```
print(type(Xn), '\n', type(Xj))
```

```
<class 'numpy.ndarray'>
 <class 'jax.interpreters.xla.DeviceArray'>
```

JAX creates a different type, suitable for executing on different platforms.

0.0.1 JAX numpy can give us derivatives of programs

But now, we can do something in JAX numpy that wasn't possible in numpy. We can ask for the derivative of the function we have just defined. There are two arguments in $y = \text{wsin}(w, x)$, w and x. We can define the partial derivatives $(\partial/\partial x)\text{wsin}(w, x)$ and $(\partial/\partial w)\text{wsin}(w, x)$

[6]:
```
d_wsin_x = grad(wsin, argnums=1)
d_wsin_w = grad(wsin, argnums=0)
```

0.0.2 Check:

Verify that these following numbers are correct

[7]:
```
print(wsin(1.,0.), d_wsin_x(1., 0.), d_wsin_w(1.,0.))
print(wsin(3.,0.), d_wsin_x(3., 0.), d_wsin_w(3.,0.))
```

```
0.0 1.0 0.0
0.0 3.0 0.0
```


[8]:
```
# As before, wsin is an array returns an array
xvals = jnp.linspace(-1,1,5)
print(xvals,'\n',
```



```
[-1.  -0.5  0.   0.5  1. ]
 [ 8.742278e-08 -1.000000e+00  0.000000e+00  1.000000e+00 -8.742278e-08]
```


0.0.3 Vectorising:

The grad function does not work on an array. Check for yourself: Try to execute both d_wsin_w(xvals) and d_wsin_x(xvals). You will see

TypeError: Gradient only defined for scalar-output functions. Output had shape: (5,).

[9]:
```
# uncomment this
# d_wsin_w(1.,xvals)
```

0.1 vmap

vmap vectorises the grad of function.
Don't worry about the syntax in_axes yet.

[10]:
```
grad_wsin_w = vmap(d_wsin_w, in_axes=(None, 0))
grad_wsin_x = vmap(d_wsin_x, in_axes=(None, 0))
print(grad_wsin_w(1., np.pi*xvals))
print(grad_wsin_x(1., np.pi*xvals))
```

```
[ 3.1415927e+00  6.8661691e-08  0.0000000e+00 -6.8661691e-08
 -3.1415927e+00]
[-1.000000e+00 -4.371139e-08  1.000000e+00 -4.371139e-08 -1.000000e+00]
```

You should verify by inspection that the output makes sense.

[11]:
```
# Having done this, you can now evaluate the gradient on an array
# and plot the returned array of values
X_plot = np.linspace(-1,1,20)
plt.plot(np.pi*X_plot, wsin(1., np.pi*X_plot))
plt.plot(np.pi*X_plot, grad_wsin_x(1., np.pi*X_plot))
```

[11]:
```
[<matplotlib.lines.Line2D at 0x7fc9ae8b1e50>]
```


0.1.1 Note:

We never called the cosine function. Calling the grad operation on sin enabled the automatic differentiation (AD) routines within JAX to compute (d/dx) sin (x) = cos (x).

0.1.2 Your turn:

- Plot (d/dx) sin(w x) for different values of w.
- Plot (d/dw) sin(w x) for different values of w.

[12]:
```
def sigmoid(w, xin):
    return jnp.array(1/(1+jnp.exp(-w*xin)))
```

[13]:
```
X1 = np.linspace(-6,6,40)
plt.plot(X1, sigmoid(2.,X1))
```

[13]: [<matplotlib.lines.Line2D at 0x7fc9aef75d00>]

0.1.3 Your turn:

define grad_sigmoid in exactly the same way that we defined grad_wsin_x and grad_wsin_w above. First define the scalar version using **grad** with partial derivatives with respect to w and x captured by the arguments **argnums** set to 0 or 1. Then **vmap** the grad with the "in_axes" argument.