CME 216, ME 343 - Winter 2021

Eric Darve, ICME

Stanford University

Automatic differentiation using TensorFlow

Let's explore how we can use TensorFlow to compute derivatives of functions using a special technique called **Automatic differentiation.**

In fact, we have already seen this with the backpropagation algorithm.

This allowed computing

$$\frac{\partial L}{\partial W_{ij}}$$

efficiently.

But TensorFlow (PyTorch as well) offers much more general capabilities to compute derivatives.

In fact it can differentiate any function using automatic differentiation.

$$y(x) = 4x + x^3$$

Let's compute dy/dx using automatic differentiation in TensorFlow.

Declare that x is an independent variable:

```
x = tf.Variable(1.0)
```

Define the operations that will require differentiation:

```
with tf.GradientTape() as g:
    y = 4*x + x**3
```

Differentiate!

```
dy_dx = g.gradient(y, x)
print('dy/dx = ', dy_dx)
```

```
Output: dy/dx =  tf.Tensor(7.0, shape=(), dtype=float32)
```

```python
x = tf.Variable(1.0)
with tf.GradientTape() as g:
    y = 4*x + x**3

dy_dx = g.gradient(y, x)
```

All the Python code in these slides can be found on the github repository:

[TensorFlow AD.ipynb]

We can differentiate with respect to multiple variables.

Take

$$y = x_1 + x_2^2, \quad x_1 = 1, \quad x_2 = 2$$

Compute:

$$\frac{\partial y}{\partial x_1} = 1$$

$$\frac{\partial y}{\partial x_2} = 2x_2 = 4$$

```python
x1 = tf.Variable(1.0)
x2 = tf.Variable(2.0)

with tf.GradientTape() as g:
    y = x1 + x2**2

dy_dx1, dy_dx2 = g.gradient(y, [x1,x2])
print('dy/dx1 = ', dy_dx1.numpy(), '; dy/dx2 = ', dy_dx2.numpy())

Output: dy/dx1 =  1.0 ; dy/dx2 =  4.0
```

`dy_dx1.numpy()` prints the numerical content of the tensor as a `numpy` variable.

It's important to distinguish between `tf.Variable` and constant values which are not independent variables.

```python
# Independent variable
x0 = tf.Variable(2.0, name='x0')
# A tf.constant is not a variable
c1 = tf.constant(-2.0, name='c1')
# Constant because we specify trainable=False
c2 = tf.Variable(-1.0, name='c2', trainable=False)
# variable + tensor returns a tensor. So c3 is not a tf.Variable.
c3 = tf.Variable(1.0, name='c3') + 1.0
# A variable but not used to compute y
x4 = tf.Variable(0., name='x4')
```

When you differentiate with respect to a constant rather than a `tf.Variable`, TF returns `None`.

```
with tf.GradientTape() as g:
    z = x0 + c1
    y = z**2 + (c2**3) + 4*c3

grad = g.gradient(y, [x0, c1, c2, c3, x4])

for dy_dxi in grad:
    print(dy_dxi)
```

Output

```
tf.Tensor(0.0, shape=(), dtype=float32)
None
None
None
None
```

In some cases, you need TF to interpret a `tf.Tensor` as an independent variable.

Consider

```
x = tf.constant(-3.)

with tf.GradientTape() as g:
    y = x**4

print(g.gradient(y, x))
```

What is the output?

What is the output?

```
None
```

Instead use `watch`:

```
x = tf.constant(-3.)

with tf.GradientTape() as g:
    g.watch(x)
    y = x**4

print(g.gradient(y, x))  # 4x^3 = 4 (-27) = -108

Out: tf.Tensor(-108.0, shape=(), dtype=float32)
```

Tensor of variables can be used as input.

```
x = tf.Variable([1, -3.0])
with tf.GradientTape() as g:
    y = tf.math.reduce_sum(x**2)
```

Tensor of variables can be used as input.

```
x = tf.Variable([1, -3.0])
with tf.GradientTape() as g:
    y = tf.math.reduce_sum(x**2)
```

See tf.math for all tensor operations supported by TF. For example, `tf.math.reduce_sum`.

In that case, you compute the gradient with respect to each variable in the tensor.

```
print(x.numpy())
print(y.numpy()) # x[0]**2 + x[1]**2 = 1 + 9 = 10
print(g.gradient(y, x))  # (2x[0], 2x[1]) = (2,-6)
Out:
[ 1. -3.]
10.0
tf.Tensor([ 2. -6.], shape=(2,), dtype=float32)
```

As an extension, when the dependent variable (or target) is a vector, `gradient` returns the sum of the gradients for each component.

```
x = tf.Variable(-1.)
with tf.GradientTape() as g:
    y = [2*x,x**4]

print([y[i].numpy() for i in range(2)]) # [-2,1]
print(g.gradient(y, x))  # 2 + 4x^3 = 2 - 4 = -2
Out:
[-2.0, 1.0]
tf.Tensor(-2.0, shape=(), dtype=float32)
```

Let's consider a more complicated example where we want to differentiate two different functions.

Moreover, the input will be a vector of `tf.Variable`.

```python
x = tf.Variable([1, -3.0])
with tf.GradientTape() as g:
    y = 2*x
    z = y**2

print(x.numpy())
print(y.numpy())
print(g.gradient(y, x))
print(g.gradient(z, x))
```

Fail!

Fail!

After calling `gradient`, the resources for `g` are deleted.

`GradientTape` is not persistent.

We need to explicitly tell TF not to free the resources.

```python
x = tf.Variable([1, -3.0])
with tf.GradientTape(persistent=True) as g:
    y = 2*x
    z = y**2

print(x.numpy())
print(y.numpy())
print(g.gradient(y, x))
print(g.gradient(z, x))
del g # release resources
```

What should the output be?

What should the output be?

`y=2*x`: a tensor of size 2.

`y[0] = 2*x[0]; y[1] = 2*x[1];`

`g.gradient(y, x)`:

Gradient of `y[0]`: `[2. 0.]`

Gradient of `y[1]`: `[0. 2.]`

Because `y` is a tensor, you have to sum the gradient of `y[0]` and `y[1]`.

```
Out: tf.Tensor([2. 2.], shape=(2,), dtype=float32)
```

What is `g.gradient(z, x)`?

What is `g.gradient(z, x)`?

`z[0]=4*x[0]**2; z[1]=4*x[1]**2` with
`x = tf.Variable([1., -3.])`

`g.gradient(z, x)`: `[8*x[0] + 0., 0. + 8*x[1]]`
which is `[8., -24.]`

```
Out: tf.Tensor([  8. -24.], shape=(2,), dtype=float32)
```

Plotting the derivative is very easy!

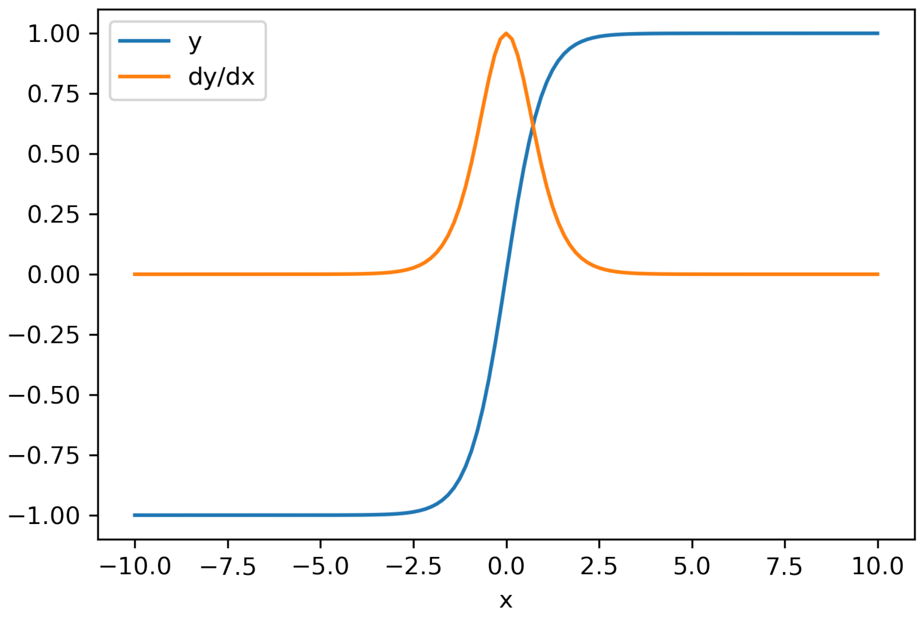
```python
x = tf.linspace(-10.0, 10.0, 129) # A tf.Tensor, not a tf.Variable

with tf.GradientTape() as g:
    g.watch(x)
    y = tf.math.tanh(x)

dy_dx = g.gradient(y, x)
```

`y` is a vector such that `y[i] = tanh(x[i])`.

When computing the gradient of `y` with respect to `x[i]` we need to differentiate all the components of `y` with respect to `x[i]` and add these derivatives together.

Since `y[i] = tanh(x[i])`, the `i` component of `g.gradient(y, x)` is simply $\tanh'(x_i)$.

```
plt.plot(x, y, label='y')
plt.plot(x, dy_dx, label='dy/dx')
```



More information on Automatic Differentiation

and

Advanced Automatic Differentiation

Everything on Gradient Tape