The goal of this lab is to give first hand experience with deriving autograd formulas and adding them to PyTorch.

The deliverable will be a stack of PRs (that won't be merged into master) containing all the code for the different sections below. You should add both your mentor as well as @albanD as a reviewer for this PR. The task will be considered finished when the PR is accepted by the reviewers and you can just close it at that time.

Considered Function

For this lab, we are going to consider the following attention function

```python
import torch

def attn(q, k, v):
    x = torch.matmul(q, k.transpose(0, 1))
    a = torch.tanh(x)
    o = torch.matmul(a, v)
    return o, a

q = torch.rand(2, 3)
k = torch.rand(2, 3)
v = torch.rand(2, 4)
o, a = attn(q, k ,v)
```

The goal is to implement this function as a native operator in PyTorch and provide an efficient autograd formula for it. You can expect that all inputs are 2D Tensors (you should have the proper error handling for this) where the first dimension is always the same and the second dimension for q and k are the same.

I) Derive gradient formula

For the function above, you should derive on paper the gradient formula. You can write the final formula in a markdown file.

II) Write custom Function and gradcheck

Based on the formula above, you should implement the forward/backward inside a custom autograd Function to be able to verify that the computed gradient is correct. You can write this in a small standalone script in `test/` that defines the custom Function and run gradcheck/gradgradcheck on it. To make sure that your script actually runs in the CI, don't forget to add it to `test/run_test.py`

III) Write native composite function and OpInfo

Now that you know that the formula is correct, you can implement a brand new native function in native_functions.yaml as a CompositeImplicit function. You can then add an OpInfo entry (in common_methods_invocations.py) to be able to test the new operator.

IV) Write custom formula in derivatives.yaml

Once testing is working fine with the Composite version, you can replace it with an Explicit version and add an entry in "derivatives.yaml" to specify this gradient.