```
from fastai.basics import *
```

```
def update_prev_grad(p, mom, dampening=False, grad_avg=None, **kwargs):
    "Keeps track of the previous gradient, should be one of last cbs. "
    return {'prev_grad': p.grad.data}
```

```
def n_avg_grad(p,lr,nmom=None,n_avg=None,prev_grad=None,**kwags):
    if n_avg is None:
        prev_grad=torch.zeros_like(p.grad.data)
        n_avg = p.grad.data-prev_grad
    else:
        n_avg = (1-nmom)*n_avg+nmom*(p.grad.data-prev_grad)
    return {'n_avg': n_avg,'prev_grad':prev_grad}
```

```
def n_average_sqr_grad(p,nmom,sqr_mom, prev_grad=None, dampening=True, sqr_avg=None, **kwa
    if sqr_avg is None: sqr_avg = torch.zeros_like(p.grad.data)
    damp = 1-sqr_mom if dampening else 1.
    grad = (2-nmom)*p.grad.data+(nmom-1)*prev_grad
    sqr_avg.mul_(sqr_mom).addcmul_(grad,grad, value=damp)
    return {'sqr_avg': sqr_avg}
```

```
def adan_step(p,lr,grad_avg=None,nmom=None,n_avg=None,sqr_avg=None,
              eps=None,**kwargs):
    p.data.addcdiv_(grad_avg+(1-nmom)*n_avg,
                    (sqr_avg).sqrt() + eps,
                    value = -lr)
```

```
def Adan(params, lr, mom=0.9, sqr_mom=0.99,nmom=0.9, eps=1e-5, wd=0.01, decouple_wd=True):
    "A `Optimizer` for Adam with `lr`, `mom`, `sqr_mom`, `eps` and `params`"
    cbs = [weight_decay] if decouple_wd else [l2_reg]
    cbs += [partial(average_grad, dampening=True),n_avg_grad, n_average_sqr_grad,adan_step
    return Optimizer(params, cbs, lr=lr,nmom=nmom, mom=mom, sqr_mom=sqr_mom, eps=eps, wd=w
```

```
l=nn.Linear(4,4)
opt=Adan(l.parameters(),0.01)
print(l.weight)
inp=torch.tensor([.1,.2,.3,.4])
F.mse_loss(l(inp),torch.tensor([1.,2.,3.,4.])).backward()
opt.step()
```

```
F.mse_loss(l(inp),torch.tensor([1.,2.,3.,4.])).backward()
opt.step()
```

```
Parameter containing:
tensor([[ 0.4984,  0.2108,  0.3309, -0.1065],
        [-0.4451,  0.3669, -0.2573,  0.1675],
        [-0.3011, -0.4368, -0.3770,  0.4079],
        [-0.0182,  0.3828,  0.4397, -0.0060]], requires_grad=True)
```

```
l.weight
```

```
Parameter containing:
tensor([[ 0.5296,  0.2421,  0.3622, -0.0752],
        [-0.4137,  0.3981, -0.2259,  0.1988],
        [-0.2698, -0.4054, -0.3455,  0.4392],
        [ 0.0132,  0.4141,  0.4709,  0.0254]], requires_grad=True)
```