In [1]:
```
import torch
import torch.nn.functional as F
```

In [2]:
```
torch.manual_seed(0)
```

Out[2]:
```
<torch._C.Generator at 0x7f7cd811d8b0>
```

In [3]:
```
# define net

class BobNet(torch.nn.Module):

    def __init__(self, n_in, n_out):
        super(BobNet, self).__init__()
        self.predict = torch.nn.Linear(n_in, n_out)

    def forward(self, x):
        x = F.leaky_relu(self.predict(x))
        return x
```

In [4]:
```
# spawn model

bob = BobNet(n_in=2, n_out=2)
print(bob)
```

```
BobNet(
  (predict): Linear(in_features=2, out_features=2, bias=True)
)
```

In [5]:
```
# define optim and loss

optimizer = torch.optim.Adam(bob.parameters(), lr=0.01)
loss_fn = torch.nn.MSELoss()
```

In [6]:
```
# data

x = torch.Tensor([1.0, 2.0])
y = torch.Tensor([2.0, 4.0])


# training loop

for i in range(1000):

    # predict
    out = bob(x)
    loss = loss_fn(out, y)

    if i%100 == 0:
        print(f"loss at step {i:3d}: {loss.item():2.2f}")

    # reset, calculate and apply gradient
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
```

```
loss at step   0: 9.21
loss at step 100: 0.66
loss at step 200: 0.00
loss at step 300: 0.00
loss at step 400: 0.00
loss at step 500: 0.00
loss at step 600: 0.00
loss at step 700: 0.00
loss at step 800: 0.00
loss at step 900: 0.00
```