PyTorch Concepts

Summer School on Generative Models

Morten Hannemose, mohan@dtu.dk

August 13th, 2019

Overview

- Fully connected example
- zero_grad(), backward() and step()
- detach()
- no_grad()

Fully connected sample network

```
class Net(nn.Module):
    def __init__(self):
        super(net, self).__init__()
        self.classifier = nn.Sequential(
                nn.Linear(28*28, 512),
                nn.ReLU(),
                nn.Linear(512, 256),
                nn.ReLU(),
                nn.Linear(256, 1),
                nn.Sigmoid())

    def forward(self, x):
        x = x.view(x.size(0), -1)
        x = self.classifier(x)
        return x
```

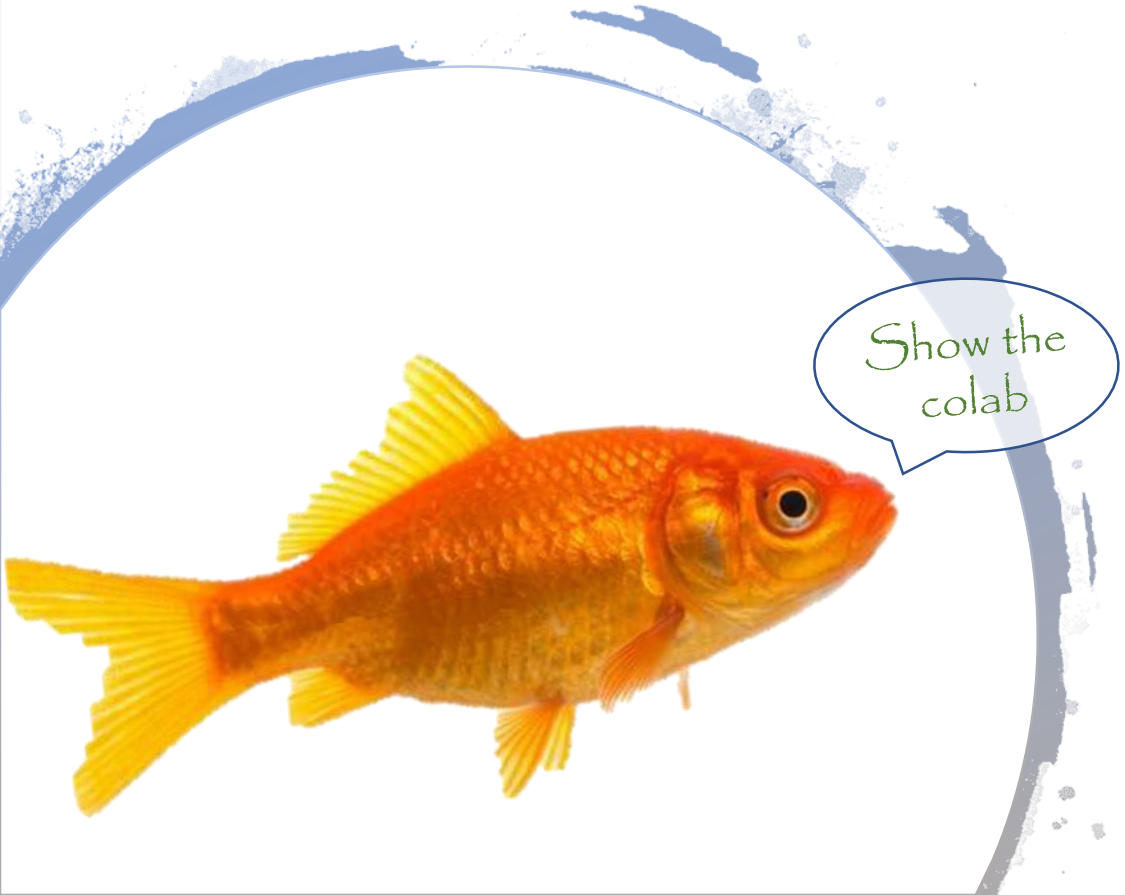
Training a network

- ```
  opt.zero_grad()
  loss = …
  loss.backward()
  opt.step()
  ```

- Each weight has an associated **gradient buffer**
  - `zero_grad()` writes zeros to it
  - `backward()` adds the current gradient to the buffer
  - `step()` takes a step in the buffer's direction
  ```

detach

- detach()
  - Stops .backward() from differentiating back through it
  - From documentation:
    - Returns a new Tensor, detached from the current graph.
    - The result will never require gradient.

torch.no_grad()

- Used for cases where you will not need to backprop through the computation

- ```
  with torch.no_grad():
      y = f(x)
  ```

- Good for inference
  - Computation uses less memory
  ```

http://bit.ly/2MdZsXI

Exercise time



- Generate MNIST digits using a vanilla GAN
- Additional tasks:
  - Implement another loss
  - Convert your network to a DCGAN
  - Show images of an interpolation in latent space
  - Generate images from FashionMNIST
  - Convert your architecture into a cGAN
  - Extra: Create a cGAN model to convert from SVHN to MNIST