CS5260 Tutorial 2: Generative Adversarial Network

Xiangyu Peng

Jan 27, 2023

Comments on Assignment 1

6 marks in total:

- Submission on time: 1 mark
  - Not following the submission instructions: -0.5 (see main.ipynb)
- Function 1: 3 marks
- Function 2: 2 marks

- Function 1: 3 marks
  - Cannot run (throw an error): -1.5
    - input image requires grad
    - Input of nn.CrossEntropyLoss() should be one of the following:
      - logits of shape [B, C] and label (index) of shape [B] (pytorch < 1.13)
      - logits of shape [C] and label (index) is a scaler (pytorch >= 1.13)
      - logits of shape [B, C] and label (probability) of shape [B, C] (pytorch >= 1.13)
      - logits of shape [C] and label (probability) of shape [C] (pytorch >= 1.13)
  - Should not call softmax() before criterion: -1
    - as CrossEntropyLoss already does that
  - the result is not a signed one: -0.5

```python
def create_adversarial_pattern(image, label, model, criterion):
    """Create signed gradient of an image

    Args:
        image (Tensor): image; shape=[C, H, W]
        label (int): label of the image
        model: pre-trained model
        criterion: criterion for computing loss

    Return:
        signed_grad: signed gradient of input image

    Hints:
        1. Pay attention to the input shape of model
        2. Pay attention to the label shape for calculating loss
        3. use loss.backward() or torch.autograd.grad() to get gradient of a tensor
    """

    signed_grad = None
    model.eval()

    ################################
    # Your code starts here
    ################################
    x = image.unsqueeze(0).requires_grad_(True)
    logits = model(x)
    loss = criterion(logits, torch.LongTensor([label]))

    use_backward = True
    if use_backward:
        loss.backward()
        x_grad = x.grad
    else:
        x_grad = torch.autograd.grad(outputs=[loss], inputs=[x])[0]
    signed_grad = torch.sign(x_grad).squeeze(0)

    ################################
    # Your code ends here
    ################################
    assert signed_grad.shape == image.shape
    return signed_grad
```

- Function 2: 2 marks
  - correctly implement the function: +1
  - clip within [0, 1]: +1

```python
def create_adversarial_sample(image, eps, perturbation):
    """ Create an adversarial sample by adding noise to an image
    """
    adv_image = None

    ################################
    # Your code starts here
    ################################

    adv_image = image + eps * perturbation
    adv_image.clamp_(0, 1)

    ################################
    # Your code ends here
    ################################
    assert adv_image.shape == image.shape
    return adv_image
```

Recap: GAN

- Generative Adversarial Networks are composed of two models
  - Generator: it aims to generate new data similar to the expected one
    - e.g. a human art forger who creates fake works of art
  - Discriminator: it recognizes if an input data is 'real' — belongs to the original dataset — or if it is 'fake' — generated by a forger
    - e.g. an art expert, who detects artworks as truthful or fraud
- How do these two models interact?
  - The Generator has an adversary: the Discriminator
  - The Generator (forger) learns how to create data in such a way that the Discriminator isn't able to distinguish it as fake anymore
  - The competition between these two models is what improves their knowledge, until the Generator succeeds in creating realistic data

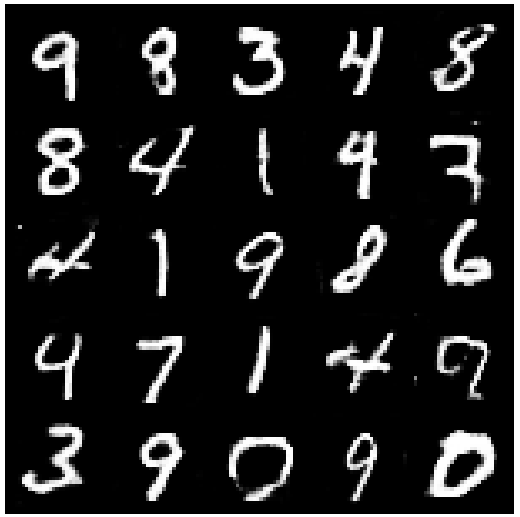
Recap: GAN

- Both generator and discriminator are being modeled with neural networks
  - A gradient-based optimization algorithm can be used to train a GAN
- The fundamental steps to train a GAN:
  1. Sample a noise set and a real-data set, each with size m
  2. Train the Discriminator on this data batch
  3. Sample a different noise subset with size m
  4. Train the Generator on this data batch
  5. Repeat from Step 1

Training data: MNIST

real

fake



Tasks

- model is given
- just implement the training code

```
"""
TODO (1): Sample a batch of noise samples from Gaussian distribution.
Hints:
    1. Use torch.randn() for sampling.
    2. Assign these samples to variable z, which is in accordance with Algorithm 1. z should be a tensor with shape [B, C], where
        B is the batch size and C is the latent dimension.
"""
```

```
"""
TODO (2): Compute loss for the discriminator, denoted as loss_d
Hints:
    1. You only need to implment the forward pass of discriminator. The fake images and backward pass are already prepared.
    2. In training of the discriminator, we don't want the gradient flow back to the generator.
        Use imgs_fake.detach() to achieve this.
    3. loss_d has two parts: loss for the real images L_real and loss for the fake images L_fake. Use the given criterion
        and gt_real/gt_fake to compute L_real/L_fake.
    4. Requirement: compute loss_d as loss_d = (L_real + L_fake) / 2
"""
```

```
"""
TODO (3): Compute the loss for the generator, denoted as loss_g
Hints:
    1. You only need to implment the forward pass. The backward pass is already given.
    2. Use the given criterion to compute loss_g
"""
```

Google Colab

- 1 GPU is sufficient for this assignment; training time should be no longer than 30 min
- Official website: https://colab.research.google.com/
- Video tutorial: https://www.youtube.com/watch?v=RLYoEyIHL6A

Q & A