Homework 2 Writeup

Due: Friday I think

This is Generative AI with Cody Watson. I am Jack Bosco. I made a LaTeX class for doing homework assignements, very cool! Also, a lot of the LaTeX code is 'borrowed' from Lu Kefu. He has great taste for document formatting.

1 Model Architecture

There are two classes, Generator and Discriminator, which are key components of a Generative Adversarial Network (GAN). GANs are a type of neural network architecture used for generating new data that resembles the input data.

Generator: This class is responsible for generating new data. It takes a latent vector (a random noise) as input and outputs an image. The architecture of the generator model is a series of linear layers (nn.Linear), each followed by a LeakyReLU activation function (nn.LeakyReLU) and batch normalization (nn.BatchNorm1d). The final layer is a Tanh activation function (nn.Tanh), which scales the output to be between -1 and 1.

Discriminator: This class is responsible for classifying whether a given image is real (from the dataset) or fake (generated by the generator). It takes an image as input and outputs a single value representing the probability that the image is real. The architecture of the discriminator model is a series of linear layers (nn.Linear), each followed by a LeakyReLU activation function (nn.LeakyReLU). The final layer is a Sigmoid activation function (nn.Sigmoid), which scales the output to be between 0 and 1, representing the probability.

Both classes inherit from nn.Module, which is a base class for all neural network modules in PyTorch. This means they both must implement a forward method that defines how to compute the output using the layers defined in the __init__ method.

2 Training Process

Training a GAN involves training the generator and discriminator models simultaneously. To train the GAN, I used the following processes:

Algorithm 1 GAN Training

procedure TRAIN DISCRIMINATOR(D, G loss_fn, $D_{\text{optimizer}}$)
    fake_images $\leftarrow G$(random_noise)
    real_images $\leftarrow$ images from dataloader
    predicted $\leftarrow$ mapping of real_images to D(real_images), fake_images to D(fake_images)
    targets $\leftarrow$ mapping of real_images to 1, fake_images to 0
    $D_{\text{loss}} \leftarrow$ loss_fn(predicted, targets)
    use $D_{\text{optimizer}}$ to update discriminator weights with D_{loss}
end procedure
procedure TRAIN GENERATOR(D, G, loss_fn, $G_{\text{optimizer}}$)
    fake_images $\leftarrow G$(random_noise)
    predicted $\leftarrow D$(fake_images)
    targets $\leftarrow$ mapping of fake_images to 1
    $G_{\text{loss}} \leftarrow$ loss_fn(predicted, targets)
    use $G_{\text{optimizer}}$ to update generator weights with G_{loss}
end procedure
procedure TRAIN GAN(D, G, loss_fn, $D_{\text{optimizer}}$, $G_{\text{optimizer}}$)
    for for each epoch **do**
        for for each batch in dataloader **do**
            Train Discriminator(D, G, loss_fn, $D_{\text{optimizer}}$)
            Train Generator(D, G, loss_fn, $G_{\text{optimizer}}$)
        end for
        Report on training progress
    end for
end procedure

3 Issues

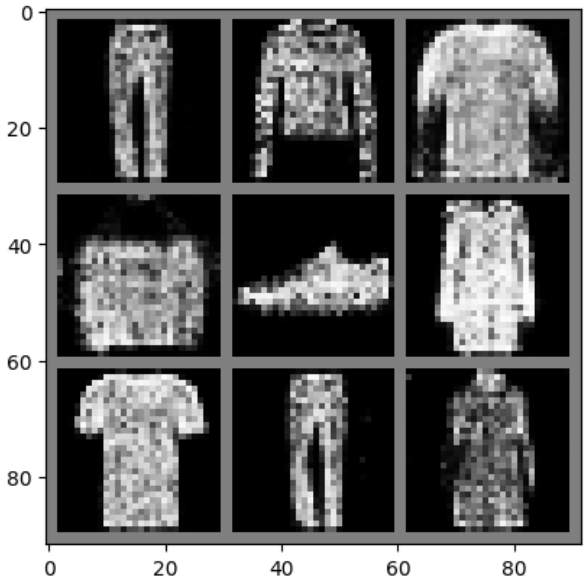
The main issue I encountered was making the generator produce realistic images. Weight collapse was not an issue since I used the Adams optimizer to add hyperamaters such as weight decay and epsilon. Thise parameters made the GAN more stable, since there was a balance between exploration and exploitation. Ultimately, after some hyperparameter tuning, I was able to generate images that were satisfiably recognizable.

The hyperparemeter values I chose were:

Hyper	**Generator**	**Discriminator**
Learning rate η	0.0002	0.0001
Batch size	64	64
Epsilon ϵ	$2e^{-6}$	$2e^{-5}$
Weight decay λ	e^{-12}	e^{-10}

4 Evaluation

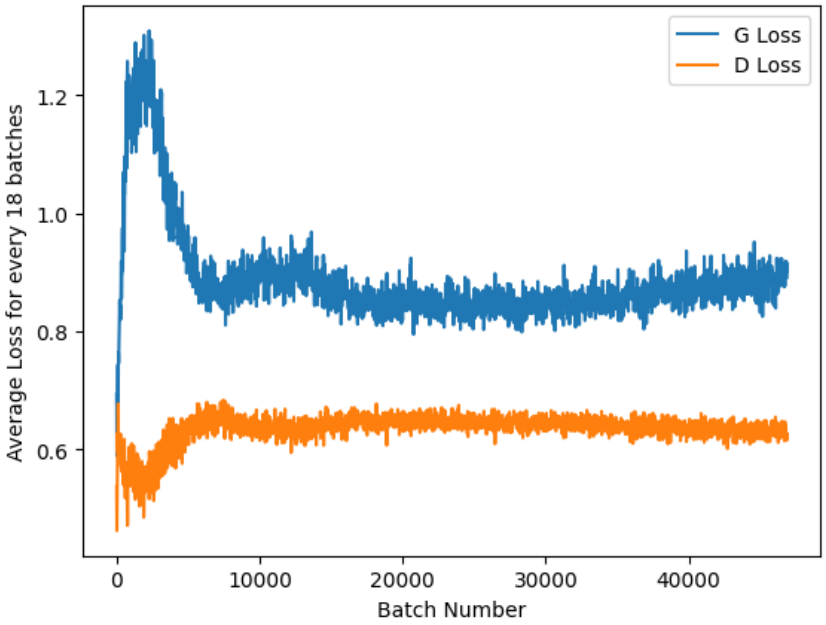
1. Quality of the generated images: Are the generated images recognizable as clothing items?
   Although blurry, I can tell the images are clothing items. Check out the images below:



   Here, you can make out the clothing items by their general shape, although the pixels are somewhat grainy.

2. Diversity of the generated images: Does the model generate a variety of clothing items?
   Yup, I made sure to set epsilon to a non-zero value, so the model should generate a variety of clothing items. In the above images, there are two t-shirts, two pairs of pants, and three long-sleeve shirts or dresses. There is also a purse (the square-shaped image).

3. Training stability: How stable are the loss curves? Are there signs of mode collapse?
   The models' loss curves (D for discriminator, G for generator) are shown below:



The loss curves are pretty stable. The generator's loss is consistently higher than the discriminator's loss, which makes sense because the generator has to train more weights than the descriminator. when the generator's loss spikes and the discriminator's loss drops, it indicates the generator is producing poor images. However, the loss curves quickly converge. There is partial mode collapse, since the generator is biased towards shirts and dresses (as seen in the generated images). Shirts and dresses are similar and more uniform in shape than other clothing items, so the generator is more likely to fool the discriminator with them. This leads to a partial mode collapse.