CI-0148 Aprendizaje Máquina
Prof. Pablo Sauma Chacón

Laboratorio 6

Objetivo: Familiarizar al estudiante con la creación y el uso de redes neuronales utilizando la biblioteca Torch. Así como la construcción de redes neuronales sofisticadas, como es el caso de redes generativas adversarias/antagónica.

Enunciado: Lea las instrucciones del documento y resuelva el enunciado en Python. Al inicio de su documento adjunte su nombre y carnet como un comentario. Este trabajo es de carácter individual, tampoco se aceptarán códigos que no hayan sido desarrollados por su persona.

Para este laboratorio se trabajará con el set de datos MNIST, que contiene 60000 imágenes de dígitos (0-9) escritos a mano en tamaño 28x28 (x1 dado que se maneja en escala de grises). La biblioteca `torchvision` permite importa con facilidad este conjunto de datos por medio del comando:

```
train = torchvision.datasets.MNIST(".", download=True)
```

y permite extraer sus datos x y y utilizando:

```
x = train.data.float()
y = train.targets
```

Note que deberá cambiar la forma del tensor de imágenes adecuadamente. Además su profesor le brinda una función que permite imprimir (y guardar) tensores de dimensiones Channels x Row x Column (donde channels puede ser 3 para RGB o 1 para grayscale).

```
def plot_image(img,save=False,name=None):
        cmap = 'gray' if img.shape[0]==1 else None
        data = (img.detach()*(255  if img.max()<=1 else 1)).permute((1,2,0)).numpy().astype(np.uint8)
        plt.figure()
        plt.imshow(data,cmap=cmap)
        plt.show()
        if save:
        if img.shape[0]==1:
                plt.imsave(name, data.squeeze(), cmap=cmap)
        else:
                plt.imsave(name, data, cmap=cmap)
```

Su objetivo es, para un dígito de su elección, crear un sistema que logre crear "falsificaciones" de dicho dígito. Para ello deberá implementar las clases **`Classifier`** y **`Generator`** que corresponden a redes neuronales utilizando los módulos de Torch. La primera red se encargará de clasificar imágenes:

1. Implemente la clase `Classifier` con una arquitectura que le permita clasificar imágenes de 1x28x28 como pertenecientes a un dígito específico o no (clasificación binaria).

   a. Puede utilizar una arquitectura convolucional para ello o, dado el tamaño de las imágenes, utilizar una arquitectura densa.
   b. Puede probar con múltiples arquitecturas hasta utilizar una que lo satisfaga.
   c. De utilizar una arquitectura completamente densa, agregar `dropout` en medio de sus capas puede ayudarle a reducir el *overfitting* de su red.

2. Implemente un método `train_classifier(opt, model, x_true, x_false, accuracy=None, max_iters=100, batch_size=1000))` que reciba el optimizador del clasificador, el clasificador, el conjunto de datos reales, el conjunto de datos falsos, así como opcionalmente una precisión deseada, un máximo de iteraciones y un tamaño de bache. Dentro de dicho método, debe ejecutar un máximo de `max_iters` iteraciones de aprendizaje, tomando `batch_size` elementos de los datos de entrenamiento positivos (donde el clasificador debe predecir 1) y `batch_size` elementos de los datos de entrenamiento negativos (donde el clasificador debe predecir 0). Si `accuracy` es un valor numérico (en lugar de `None`), entonces el entrenamiento se detiene al alcanzar `max_iters` o si la precisión del modelo alcanzó el valor deseado.

3. Implemente la clase `Generator` con una arquitectura que le permita generar imágenes de 1x28x28 que simulen ser un dígito específico escrito a mano.
   a. Puede leer sobre capas convolucionales transpuestas, sin embargo, para este ejercicio se pueden utilizar capas densas sencillas dado el tamaño de la salida.
   b. La capa de entrada de esta red recibe un arreglo de valores aleatorios, pero el tamaño del mismo queda a su decisión. Considere que una entrada demasiado grande puede generarle mucho ruido a lo interno de su generador, mientras que una entrada muy pequeña puede hacer que su generador sea muy predecible.

4. Implemente un método `train_generator(opt, generator, classifier, accuracy=None, max_iters=100, batch_size=1000)` que obedece la misma función de parámetros que `train_classifier`.
   a. Tome en consideración que al entrenar el generador, no requiere datos de entrada dado que los mismos son solo tensores de valores aleatorios.
   b. Recuerde que solo requiere el optimizador del generador, dado que el clasificador no se actualiza en esta etapa (pero las imágenes generadas por el generador son enviadas al clasificador para obtener su resultado y se utiliza la función de pérdida con el ideal de que el clasificador las clasifique como un positivo).

5. Escriba un código que permita ejecutar 50 iteraciones del ciclo GAN:
   a. Inicie con un conjunto de imágenes verdaderas extraídas de MNIST y un conjunto de imágenes de ruido.
   b. Entrene inicialmente el clasificador para que aprenda a distinguir los dígitos reales de los falsos.
   c. Luego entrene al generador para que aprenda a “engañar” al clasificador.

d. Luego genere un bache de nuevas imágenes “falsas” para enviar al clasificador en su próxima ronda de entrenamiento.
    i. Recuerde que en este caso si es necesario hacerle `.detach()` a los tensores/imágenes generadas, para que no se calculen sus respectivas propagaciones hacia atrás.
e. De estás imágenes, tome una al azar y grafíquela en pantalla, o guárdela como un archivo con nombre “`epoch_n.png`” donde n es el número de iteración.

6. ¿Qué observa de las imágenes? ¿Se converge al dígito deseado? ¿Qué modificaciones tuvo que realizar a las arquitecturas de sus redes para lograr el resultado deseado? ¿Observa algún comportamiento al exceder cierta cantidad de iteraciones del algoritmo?

7. (Opcional +10 puntos extra) Si lo desea, puede realizar un experimento utilizando GAN para trabajar con imágenes más complejas, como lo es la creación de Pokemon. Para ello se le otorga un conjunto de datos con las imágenes de los 151 Pokemon originales de Pokemon Red-Blue. Note que las imágenes varían en tamaño y son a color. Por este motivo, resulta ideal ajustar todas las imágenes a un mismo tamaño, y dado que las imágenes más grandes llegan a 55x55 sería bueno reducir su dimensionalidad a un tamaño más manejable como 32x32 (aunque se pierda calidad). Su profesor le otorga el siguiente código que le permitirá simplificar la carga de dichas imágenes, permitiéndole elegir si desea enfrentar el problema en RGB o escala de grises, así como el tamaño del resize. Además le proporciona ejemplos de resultados obtenidos tras 1000 iteraciones de GAN. Tome en consideración que este experimento consume mucho tiempo y recursos, por lo que podría considerar incluir un algoritmo de guardado y carga en su programa, de manera que pueda pausar el algoritmo si lo desea.

```
def loader(path,total = 151,action='pad',gray=True,resize=None):
    if not os.path.exists(path):
         print("Invalid path")
         return None
    pimages = []
    minsize = None
    maxsize = None
    for i in range(1,total+1):
         pkmn = path+"/"+str(i)+".png"
         if not os.path.exists(pkmn):
                 print("Failed to find",pkmn)
                 continue
         pimages.append( PIL.Image.open(pkmn).convert('RGB') )
         minsize = min(pimages[-1].size[0], minsize) if minsize else pimages[-1].size[0]
         maxsize = max(pimages[-1].size[0], maxsize) if maxsize else pimages[-1].size[0]
    if action=='min':
         for i in range(len(pimages)):
                 if pimages[i].size[0]==minsize: continue
                 pimages[i] = pimages[i].resize((minsize,minsize), resample=PIL.Image.LANCZOS)
    elif action=='max':
         for i in range(len(pimages)):
                 if pimages[i].size[0]==maxsize: continue
                 pimages[i] = pimages[i].resize((maxsize,maxsize), resample=PIL.Image.NEAREST)
    elif action=='pad':
         for i in range(len(pimages)):
```

```
                if pimages[i].size[0]==maxsize: continue
                newimg = PIL.Image.new(pimages[i].mode, (maxsize, maxsize), (255, 255, 255))
                delta = (maxsize - pimages[i].size[0])//2
                newimg.paste(pimages[i], (delta, delta))
                pimages[i] = newimg
    images = []
    for i,pkmn in enumerate(pimages):
          if resize: pkmn = pkmn.resize(resize,PIL.Image.LANCZOS)
          image = np.array(pkmn, dtype=np.float32) / 255
          image = np.moveaxis(image, 2, 0)
          if gray:
                  image = np.array([image.mean(axis=0)])
          images.append(image)
    return np.array(images)
```

** Ejemplos producidos por el profesor: