שימושים נוספים של מקודדים עצמיים

עקרון הפעולה של המקודד העצמי אשר הכרנו בפרק הקודם תקף גם כאשר אוסף הנתונים המוזן אליו מורכב יותר מאשר נקודות במרחב דו מימדי. בפרק זה נאמן מקודדים עצמיים לדחיסת תמונות מאוסף הנתונים Fashion-MNIST, ולשחזור תמונות מושחתות, כדוגמה לשניים מהשימושים האפשריים של ארכיטקטורה זו.

בשלב הראשון, נאפשר למקודד ולמפרש ללמוד תלויות לא ליניארית על ידי תוספת של פונקציות אקטיבציה לאחר האגרגציה הליניארית, ראו זאת בקטע הקוד הבא, ושימו לב גם שבשלב ראשון אנו משטחים את טנזור הקלט, שכן batch של N תמונות מאוסף הנתונים מגיע כטנזור בגודל $N\times1\times28\times28$, בעוד שהשכבה הליניארית מצפה לקלט בעל המימדים $N\times784$.

```
class Encoder(nn.Module):
    def __init__(self,latent_dim):
        super().__init__()
        self.linear = nn.Linear(784,latent_dim)
        self.relu   = nn.ReLU()
    def forward(self, image):
      flattned         = image.flatten(start_dim=1)
      compressed_image = self.relu(self.linear(flattned))
      return compressed_image
```

בבואנו לכתוב את המפרש, עלינו לתת את הדעת לצורת הפלט הרצויה ממנו. ברצוננו לשחזר את התמונות המוזנות אל הרשת, אשר בזמן טעינתן עברו נרמול: ערך כל פיקסל הוא מספר ממשי בטווח בין 0 ל-1. על כן, נשתמש באקטיבציית הסיגמואיד המייצרת פלט בטווח הדרוש. כמו כן, עלינו להפוך את פעולת השיטוח, ולהחזיר פלט בעל אותם מימדים כקלט המקודד. ראו כיצד אילוצים אלו באים לידי ביטוי בקוד.

```
class Decoder(nn.Module):
    def __init__(self,latent_dim):
        super().__init__()
        self.linear    = nn.Linear(latent_dim,784)
        self.sigmoid   = nn.Sigmoid()
    def forward(self, compressed_image):
      decoded             = self.linear(compressed_image)
      decoded             = self.sigmoid(decoded)
      reconstructed_image = decoded.reshape(-1,1,28,28)
      return reconstructed_image
```

כעת נוכל לבחור ערך קטן עבור מימד המרחב החבוי, לחבר בטור מקודד ומפרש, ולאמנם יחדיו כמקודד עצמי. למשל, בשורה הבאה אנו מגדירים מקודד עצמי בעל מרחב חבוי עשר מימדי (זכרו שמימד הקלט הוא 28X28).

```
autoencoder = nn.Sequential(Encoder(10),Decoder(10))
```

לצורך אימון המקודד העצמי נעביר דרכו batch של תמונות ונשווה את התוצאה לתמונות המקוריות. פונקציית ההפסד תהיה שוב `nn.MSELoss`, כאשר בהקשר הנוכחי היא קרויה "מחיר השחזור" (reconstruction loss) מסיבות ברורות. תוצאות אימון מוצלח מופיעות באיור הבא, בו מאוירות תמונות המקור ופלט המקודד העצמי של זוג תמונות מאוסף נתוני הבדיקה.

ראו כי השחזור אינו מושלם, שכן הנתונים עברו דרך צוואר בקבוק אשר גודלו כ-13% מגודל הנתונים המקוריים.

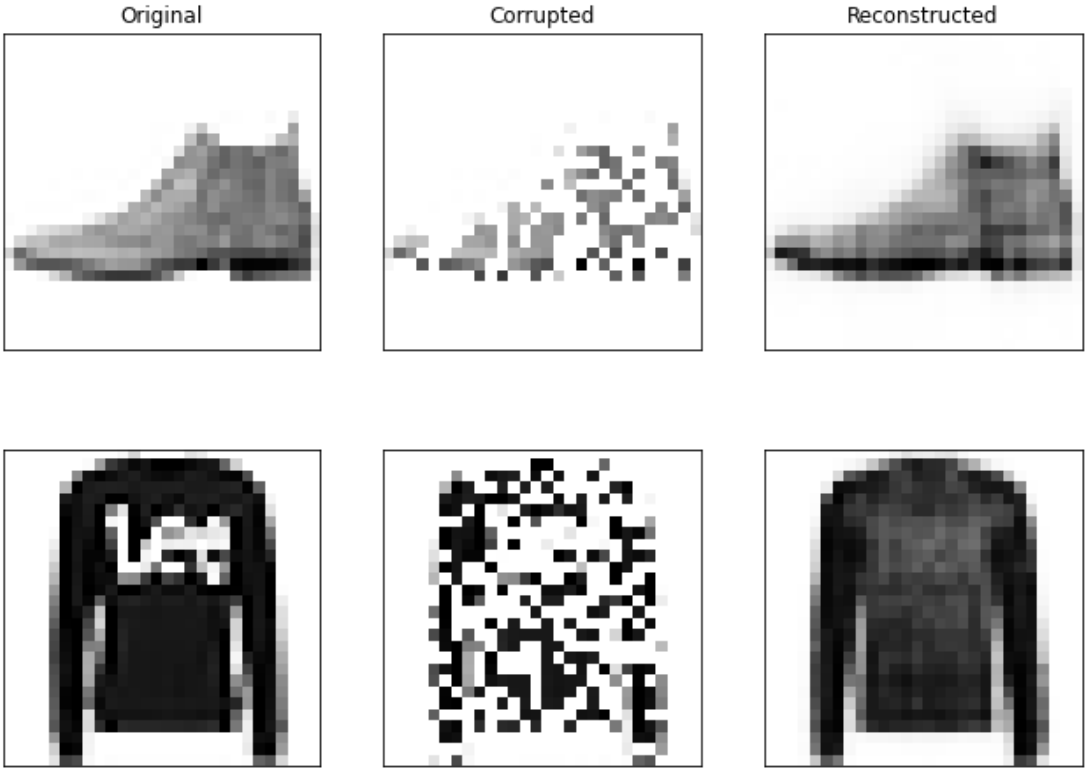
אם יהיה ברצוננו להשתמש במקודד לצורך משימות המשך, למשל עבור אימון מסווג על סמך הייצוג החבוי, נוכל לחלץ אותו בנקל, שכן הוא המודול הראשון באובייקט `autoencoder`.

```
encoder = autoencoder[0]
```

כעת על בסיס הפלט של האובייקט `encoder`, נוכל לבנות רשת סיווג חדשה, ולאמן את הפרמטרים של השכבות החדשות בלבד.

מקודד עצמי לניקוי רעש

באותה ארכיטקטורת רשת כנ"ל נוכל להשתמש גם לצורך אימון מקודד עצמי המקבל תמונות אשר עברו השחתה ומשחזר אותן. רשת המשמשת למטרה זו נקראת באנגלית Denoising Auto-Encoder (DAE). לפני שנדון בפרטי המימוש, ראו באיור להלן שחזורים של DAE מאומן. הקלט לרשת הוא התמונה האמצעית, אשר חצי מהפיקסלים שלה אופסו באקראי. הרשת מחזירה את הפלט מימין, מבלי לראות אף פעם את התמונה המקורית (שמאל).



ישנם מספר בודד של שינויים הדרושים בבניית המקודד העצמי ואימונו בכדי לקבל תוצאה זו. ראשית, נרצה ליצור יתירות במרחב החבוי, בכדי להתמודד עם אי הודאות המובנית בנתונים רועשים. לצורך זה נבחר את מימד המרחב החבוי להיות גדול במעט ממימד הנתונים המקוריים.

```
latent_dim = int(784 * 1.2)
DAE = nn.Sequential(Encoder(latent_dim), Decoder(latent_dim))
```

שנית, עלינו להשחית את הנתונים למטרות האימון. על כן, לאחר טעינת batch של תמונות למשתנה `original_imgs`, נעביר אותו דרך שכבת Dropout אשר תאפס בכל תמונה חצי מהפיקסלים באקראי.

```
corruptor          = nn.Dropout()
corrupted_imgs     = corruptor(original_imgs)
```

לבסוף, בתוך לולאת האימון נזכור להזין לרשת את התמונות המושחתות, אך את הפלט נשווה לתמונות המקוריות. מחיר השחזור יתגמל מודלים אשר ממלאים את החסר בתמונות המושחתות, כפי שאנו רוצים.

```
reconstructed = DAE(corrupted_imgs)
loss          = MSELoss(reconstructed,original_imgs)
```

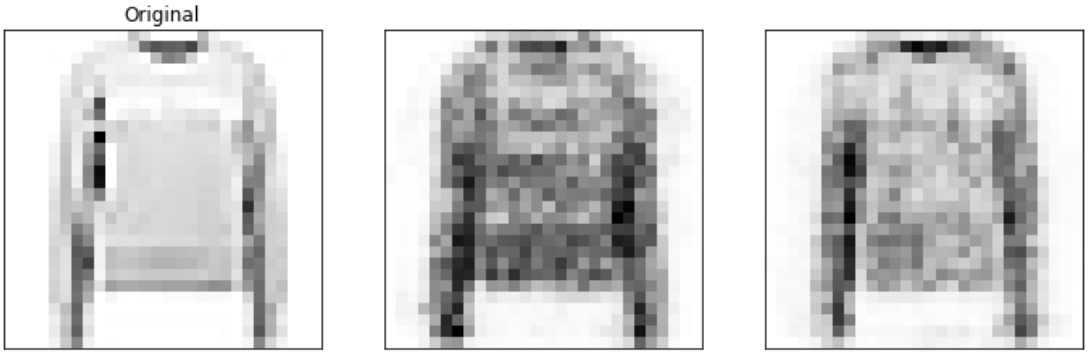
לאחר שינוי זה כל שנשאר הוא להפעיל את אלגוריתם האימון.

דגימת נתונים חדשים

שימוש נוסף של המקודד העצמי הוא למטרת ייצור דגימות מלאכותיות חדשות הדומות לתמונות הקלט. לאחר האימון, אנו יכולים להזין ייצוג חבוי חדש לתוך **המפרש** המאומן, ולקבל תמונה משוחזרת המתאימה לייצוג חבוי זה, וזאת למרות שהוא אינו מייצג אף תמונה מקורית. ראו דוגמה לכך בקטע הקוד הבא.

```
encoder = DAE[0]
decoder = DAE[1]
num_samples = 2
with torch.no_grad():
  encoded = encoder(one_img)
  perturbation = torch.randn((num_samples,*encoded.size()))
  perturbed_latent = encoded*(1+perturbation)
  new_samples = decoder(perturbed_latent)
```

ראו כיצד אנו מוסיפים לקידוד של תמונה מקורית רעש אקראי ומכניסים את התוצאה למפרש. שתי דגימות חדשות המתקבלות בצורה זו, לצד תמונת המקור, מופיעות באיור הבא.



דוגמה זו היא הפעם הראשונה בה אנו נתקלים ברשת נוירונים המייצרת נתונים חדשים. רשתות המשמשות למטרות אלו מהוות **מודלים גנרטיביים**, ועבורן קיימות ארכיטקטורות רשת מוצלחות במיוחד כגון GAN (Generative Adversarial Network), בהן לא נעסוק בקורס זה.

ככל רשת נוירונים, גם מקודדים עצמיים יוכלו ללמוד ייצוגים חבויים מוצלחים על ידי העמקת הרשת. בבואנו להוסיף שכבות למקודד לא ניתקל בכל בעיה, למעשה נוכל אפילו להשתמש ברשת שאומנה למטרה אחרת כגון ResNet, להוריד לה את ראש הסיווג ולהשתמש במחלץ המאפיינים שלה בתור המקודד. עם זאת, עבור המפרש נידרש לבצע פעולה הפוכה: עיבוד הנתונים באופן הדרגתי בחזרה מהייצוג החבוי לתמונה. עם אתגר זה נתמודד בפרק הבא.

שאלות לתרגול

1. אמנו מספר מקודדים עצמיים לניקוי רעש על אוסף הנתונים Fashion-MNIST הנבדלים זה מזה בגודל המרחב החבוי. השוו את התוצאות ומצאו את הערך האופטימלי. האם מצאתם DAE מוצלח בעל מימד חבוי קטן ממימד הנתונים המקוריים? נסו להסביר תוצאה זו על סמך יתירות המידע הקיימת באוסף הנתונים עצמו.
2. חזרו על השאלה הקודמת עם אוסף הנתונים MNIST. מה תוכלו להגיד על ההבדל בין אוספי הנתונים?

הערות:

1. בכדי לטעון את MNIST תוכלו להשתמש בפונקציה `torchvision.datasets.MNIST`.
2. שימו לב שאוספי הנתונים זהים מכל הבחינות (גודל התמונה, מספר דגימות, מספר מחלקות וכו') מלבד התוכן.