7. Deep Generative Models

המודלים שהוצגו בפרקים הקודמים הינם מודלים דיסקרימנטיביים, קרי הם מוציאים פלט על בסיס מידע נתון, אך לא יכולים ליצור מידע חדש בעצמם. בניגוד אליהם ישנם מודלים גנרטיביים, שלא רק לומדים להכליל את הדאטה הנלמד גם עבור דוגמאות חדשות, אלא יכולים גם להבין את מה שהם ראו וליצור מידע חדש על בסיס הדוגמאות שנלמדו. ישנם שני סוגים עיקריים מודלים גנרטיביים – מודלים המוצאים באופן מפורש את פונקציית הפילוג של הדאטה הנתון ובעזרת הפילוג מייצרות דוגמאות חדשות, ומודלים שלא יודעים לחשב בפירוש את הפילוג אלא מייצרים דוגמאות חדשות בדרכים אחרות. בפרק זה נדון במודלים הפופולריים בתחום – VAE, GANs ו-Auto-regressive models (PixelCNN and PixelRNN).

יתרונות של VAE: קל לאימון, בהינתן x קל למצוא את z, וההתפלגות של z נתונה בצורה מפורשת.

יתרונות של GAN: התמונות יוצאות באיכות גבוהה, מתאים להרבה דומיינים.

7.1 Variational AutoEncoder (VAE)

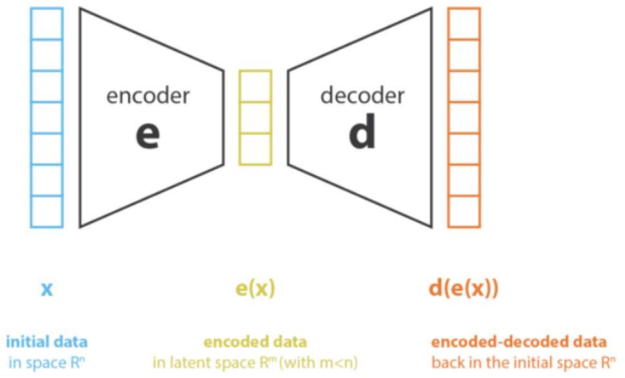
המודל הראשון הינו VAE, וכדי להבין אותו היטב יש להסביר קודם מהם Autoencoders, כיצד הוא עובד ומה החסרונות שלו.

7.1.1 Dimensionality Reduction

במקרים רבים, הדאטה אותו רוצים לנתח הוא בעל ממד גבוה, כלומר, לכל דגימה יש מספר רב של פיצ'רים, כאשר בדרך כלל לא כל הפיצ'רים משמעותיים באותה מידה. לדוגמה – מחיר מניה של חברה מסוימת מושפע ממספר רב של גורמים, אך ככל הנראה גובה ההכנסות של החברה משפיע על מחיר המניה הרבה יותר מאשר הגיל הממוצע של העובדים. דוגמה נוספת – במשימת חיזוי גיל של אדם על פי הפנים שלו, לא כל הפיקסלים בתמונת הפנים יהיו בעלי אותה חשיבות לצורך החיזוי. כיוון שקשה לנתח דאטה מממד גבוה ולבנות מודלים עבור דאטה כזה, הרבה פעמים מנסים להוריד את הממד של הדאטה תוך איבוד מינימלי של מידע. בתהליך הורדת הממד מנסים לקבל ייצוג חדש של הדאטה בעל ממד יותר נמוך, כאשר הייצוג הזה מורכב מהמאפיינים הכי משמעותיים של הדאטה. יש מגוון שיטות להורדת הממד כאשר הרעיון המשותף לכולן הוא לייצג את הדאטה בממד נמוך יותר, בו באים לידי ביטוי רק הפיצ'רים המשמעותיים יותר.

הייצוג החדש של הדאטה נקרא הייצוג הלטנטי או הקוד הלטנטי, כאשר יותר קל לעבוד איתו במשימות שונות על הדאטה מאשר עם הדאטה המקורי. בכדי לקבל ייצוג לטנטי איכותי, ניתן לאמן אותו באמצעות decoder הבוחן את יכולת השחזור של הדאטה. ככל שניתן לשחזר בצורה מדויקת יותר את הדאטה מהייצוג הלטנטי, כלומר אובדן המידע בתהליך הוא קטן יותר, כך הקוד הלטנטי אכן מייצג בצורה אמינה את הדאטה המקורי.

תהליך האימון הוא דו שלבי: דאטה $x \in \mathbb{R}^n$ עובר דרך encoder, ולאחריו מתקבל $e(x) \in \mathbb{R}^m$, כאשר $m < n$. לאחר מכן התוצאה מוכנסת ל-decoder בכדי להחזיר אותה לממד המקורי, ולבסוף מתקבל $d(e(x)) \in \mathbb{R}^n$. אם לאחר התהליך מתקיים $x = d(e(x))$ אז למעשה לא נאבד שום מידע בתהליך, אך אם לעומת זאת $x \neq d(e(x))$ אז מידע מסוים אבד עקב הורדת הממד ולא היה ניתן לשחזר אותו במלואו בפענוח. באופן אינטואיטיבי, אם אנו מצליחים לשחזר את הקלט המקורי מהייצוג של בממד נמוך בדיוק טוב מספיק, כנראה שהייצוג בממד נמוך הצליח להפיק את הפיצ'רים המשמעותיים של הדאטה המקורי.



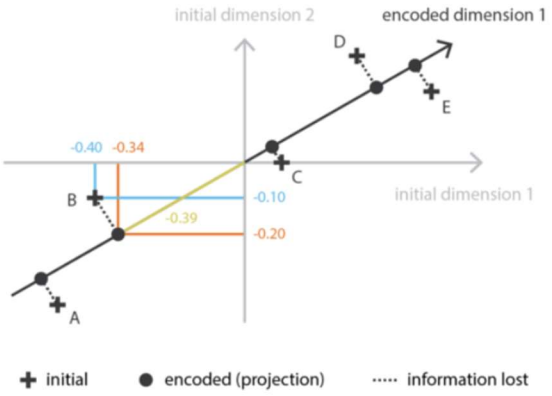
איור 7.1 ארכיטקטורת encoder ו-decoder.

כאמור, המטרה העיקרית של השיטות להורדת ממד הינה לקבל ייצוג לטנטי איכותי עד כמה שניתן. הדרך לעשות זאת היא לאמן את זוג ה-encoder-decoder השומרים על מקסימום מידע בעת הקידוד, וממילא מביאים למינימום את שגיאת שחזור בעת הפענוח. אם נסמן בהתאמה E ו-D את כל הזוגות של encoder-decoder האפשריים, ניתן לנסח את בעיית הורדת הממד באופן הבא:

$$(e^*, d^*) = \underset{(e,d)\in E\times D}{\arg\min}\ \epsilon\Big(x, d\big(e(x)\big)\Big)$$

כאשר $\epsilon\Big(x, d\big(e(x)\big)\Big)$ הוא שגיאת השחזור שבין הדאטה המקורי לבין הדאטה המשוחזר.

אחת השיטות השימושיות להורדת ממד שאפשר להסתכל עליה בצורה הזו היא Principal Components Analysis (PCA). בשיטה זו מטילים (בצורה לינארית) דאטה מממד n לממד m על ידי מציאת בסיס אורתוגונלי במרחב ה-m ממדי בו המרחק האוקלידי בין הדאטה המקורי לדאטה המשוחזר מהייצוג החדש הוא מינימלי.



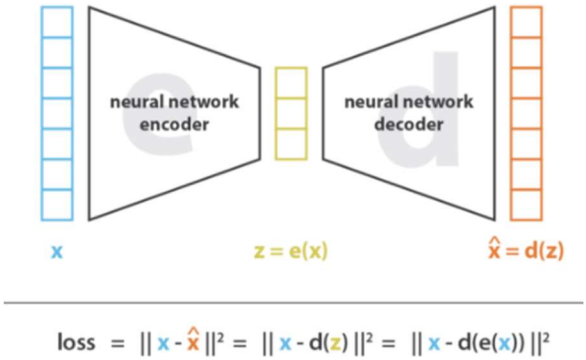
Point	Initial	Encoded	Decoded
A	(-0.50, -0.40)	-0.63	(-0.54, -0.33)
B	(-0.40, -0.10)	-0.39	(-0.34, -0.20)
C	(0.10, 0.00)	0.09	(0.07 0.04)
D	(0.30, 0.30)	0.41	(0.35, 0.21)
E	(0.50, 0.20)	0.53	(0.46, 0.27)

איור 7.2 דוגמה להורדת ממד בשיטת PCA.

במונחים של encoder-decoder, ניתן להראות כי אלגוריתם PCA מחפש את ה-encoder שמבצע טרנספורמציה לינארית על הדאטה לבסיס אורתוגונלי בממד נמוך יותר, שיחד עם decoder מתאים יביא לשגיאה מינימלית במונחים של מרחק אוקלידי בין הייצוג המקורי לבין זה המשוחזר מהייצוג החדש. ניתן להוכיח שה-encoder האופטימלי מכיל את הווקטורים העצמיים של מטריצת ה-covariance של מטריצת ה-design, וה-decoder הוא השחלוף של ה-encoder.

7.1.2 Autoencoders (AE)

ניתן לקחת את המבנה של ה-encoder-decoder המתואר בפרק הקודם ולהשתמש ברשת נוירונים עבור בניית הייצוג החדש ועבור השחזור. מבנה זה נקרא Autoencoder:

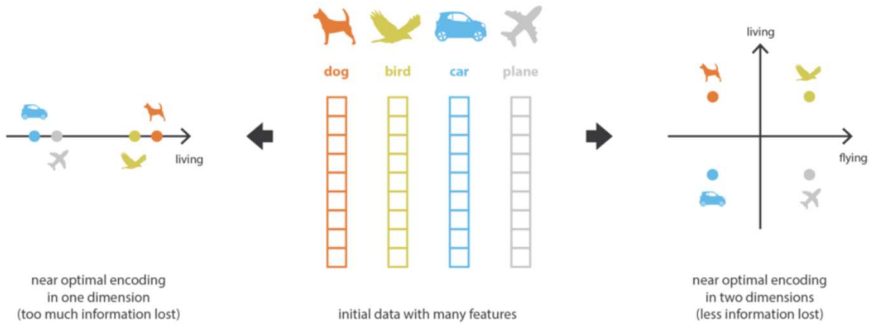


איור 7.3 Autoencoder – שימוש ברשתות נוירונים עבור הורדת הממד והשחזור.

באופן הזה, הארכיטקטורה יוצרת צוואר בקבוק לדאטה, שמבטיח שרק המאפיינים החשובים של הדאטה, שבאמצעותם ניתן לשחזר אותה בדיוק טוב, ישמשו לייצוג במרחב הלטנטי. במקרה הפשוט בו בכל רשת יש רק שכבה חבויה אחת והיא לא משתמשת בפונקציות אקטיבציה לא לינאריות, ניתן לראות כי ה-autoencoder יחפש טרנספורמציה לינארית של הדאטה באמצעותו ניתן לשחזרו באופן לינארי גם כן. בדומה ל-PCA, גם רשת כזו תחפש להוריד את הממד באמצעות טרנספורמציות לינאריות של הפיצ'רים המקוריים אך הייצוג בממד נמוך המופק על ידה

לא יהיה בהכרח זהה לזה של PCA, כיוון שלהבדיל מ-PCA הפיצ'רים החדשים (לאחר הורדת ממד) עשויים לצאת לא אורתוגונליים (-קורלציה שונה מ-0).

כעת נניח שהרשתות הן עמוקות ומשתמשות באקטיבציות לא ליניאריות. במקרה כזה, ככל שהארכיטקטורה מורכבת יותר, כך הרשת יכולה להוריד יותר ממדים תוך יכולת לבצע שחזור ללא איבוד מידע. באופן תיאורטי, אם ל-encoder ול-decoder יש מספיק דרגות חופש (למשל מספיק שכבות ברשת נוירונים), ניתן להפחית ממד של כל דאטה לחד-ממד ללא איבוד מידע. עם זאת, הפחתת ממד דרסטית שכזו יכולה לגרום לדאטה המשוחזר לאבד את המבנה שלו. לכן יש חשיבות גדולה בבחירת מספר הממדים שבתהליך, כך שמצד אחד אכן יתבצע ניפוי של פרמטרים פחות משמעותיים ומצד שני המידע עדיין יהיה בעל משמעות למשימות downstream שונות. ניקח לדוגמה מערכת שמקבלת כלב, ציפור, מכונית ומטוס ומנסה למצוא את הפרמטרים העיקריים המבחינים ביניהם:



איור 7.4 דוגמה לשימוש ב-Autoencoder.

לפריטים אלו יש הרבה פיצ'רים, וקשה לבנות מודל שמבחין ביניהם על סמך כל הפיצ'רים. מעבר ברשת נוירונים יכול להביא לייצוג של כל הדוגמאות על קו ישר, כך שככל שפרט מסוים נמצא יותר ימינה, כך הוא יותר "חי". באופן הזה אמנם מתקבל ייצוג חד-ממדי, אבל הוא גורם לאיבוד המבנה של הדוגמאות ולא באמת ניתן להבין את ההפרדה ביניהן. לעומת זאת ניתן להוריד את הממד לדו-ממד ולהתייחס רק לפרמטרים "חי" ו"עף", וכך לקבל הבחנה יותר ברורה בין הדוגמאות, וכמובן שהפרדה זו היא הרבה יותר פשוטה מאשר הסתכלות על כל הפרמטרים של הדוגמאות. דוגמה זו מראה את החשיבות שיש בבחירת הממדים של ה-encoder.

7.1.3 Variational AutoEncoders (VAE)

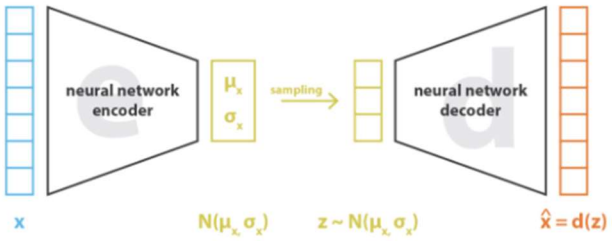
ניתן לקחת את ה-AE ולהפוך אותו למודל גנרטיבי, כלומר מודל שמסוגל לייצר בעצמו דוגמאות חדשות שאכן מתפלגות כמו הפילוג של הדאטה המקורי. אם מדובר בדומיין של תמונות למשל, אז נרצה שהמודל יהיה מסוגל לייצר תמונות שנראות אותנטיות ביחס לדאטה-סט עליו אומן. הרשתות של ה-AE מאומנות לייצג את הדאטה בממד נמוך, שלוקח בחשבון את הפיצ'רים העיקריים, ולאחר מכן לשחזר את התוצאה לממד המקורי, אך הן אינן מתייחסות לאופן בו הדאטה מיוצג במרחב הלטנטי. אם יוגרל וקטור כלשהו מהמרחב הלטנטי – קרוב לוודאי שהוא לא יהווה ייצוג שקשור לדאטה המקורי, כך שאם היינו מכניסים אותו ל-decoder, סביר שהתוצאה לא תהיה דומה בכלל לדאטה המקורי. למשל אם AE אומן על אוסף של תמונות של כלבים ודוגמים וקטור מהמרחב הלטנטי שלו, הסיכוי לקבל תמונת כלב כלשהו לאחר השחזור של ה-decoder הינו אפסי.

כדי להתמודד עם בעיה זו, ניתן להשתמש ב-Variational AutoEncoders (VAE). בשונה מ-AE שלוקח דאטה ובונה לו ייצוג מממד נמוך, VAE קובע התפלגות פריורית למרחב הלטנטי z – למשל התפלגות נורמלית עם תוחלת 0 ומטריצת covariance $\mathbb{I}$. בהינתן התפלגות זו, ה-encoder מאמן רשת המקבלת דאטה x ומוציאה פרמטרים של התפלגות פוסטריורית $z|x$, מתוך מטרה למזער כמה שניתן את ההפרש בין ההתפלגויות z ו-$z|x$. לאחר מכן דוגמים וקטורים מההתפלגות הפוסטריורית $z|x$ (הנתונה על ידי הפרמטרים המחושבים ב-encoder), ומעבירים אותם דרך ה-decoder כדי לייצר פרמטרים של ההתפלגות $x|z$. חשוב להבהיר שאם הדאטה המקורי הוא תמונה המורכבת מאוסף של פיקסלים, אזי במוצא יתקבל $x|z$ לכל פיקסל בנפרד ומההתפלגות הזו דוגמים נקודה והיא תהיה ערך הפיקסל בתמונה המשוחזרת.

באופן הזה, הלמידה דואגת לא רק להורדת הממד, אלא גם להתפלגות המושרית על המרחב הלטנטי. כאשר ההתפלגות המותנית במוצא $x|z$ טובה, קרי קרובה להתפלגות המקורית של x, ניתן בעזרתה גם ליצור דוגמאות חדשות, ובעצם מתקבל מודל גנרטיבי.

כאמור, ה-encoder מנסה לייצג את הדאטה המקורי באמצעות התפלגות בממד נמוך יותר, למשל התפלגות נורמלית עם תוחלת ומטריצת covariance: $z \sim p(z|x) = N(\mu_x, \sigma_x)$. חשוב לשים לב להבדל בתפקיד של ה-decoder –

בעוד שב-AE הוא נועד לתהליך האימון בלבד ובפועל מה שחשוב זה הייצוג הלטנטי, ב-VAE ה-decoder חשוב לא פחות מאשר הייצוג הלטנטי, כיוון שהוא זה שהופך את המערכת למודל גנרטיבי.



איור 7.5 ארכיטקטורה של VAE.

לאחר שהוצג המבנה הכללי של VAE, ניתן לתאר את תהליך הלמידה, ולשם כך נפריד בשלב זה בין שני החלקים של ה-VAE. ה-encoder מאמן רשת שמקבלת דוגמאות מקבוצת האימון, ומנסה להפיק מהן פרמטרים של התפלגות $z|x$ הקרובים כמה שניתן להתפלגות פריורית z, שכאמור נקבעה מראש. מההתפלגות הנלמדת הזו דוגמים וקטורים חדשים ומעבירים ל-decoder. ה-decoder מבצע את הפעולה ההפוכה – לוקח וקטור שנדגם מהמרחב הלטנטי $z|x$, ומייצר באמצעותו דוגמה חדשה הדומה לדאטה המקורי. תהליך האימון יהיה כזה שימזער את השגיאה של שני חלקי ה-VAE – גם $x|z$ שבמוצא יהיה כמה שיותר קרוב ל-x המקורי, וגם ההתפלגות $z|x$ תהיה כמה שיותר קרובה להתפלגות הפריורית z.

נתאר באופן פורמלי את בעיית האופטימיזציה ש-VAE מנסה לפתור. נסמן את הווקטורים של המרחב הלטנטי ב-z, את הפרמטרים של ה-decoder ב-θ, ואת הפרמטרים של ה-encoder ב-λ. כדי למצוא את הפרמטרים האופטימליים של שתי הרשתות, נרצה להביא למקסימום את $p(\hat{x} = x; \theta)$, כלומר למקסם את הנראות המרבית של קבוצת האימון תחת θ. כיוון שפונקציית log מונוטונית, נוכל לקחת את לוג ההסתברות:

$$L(\theta) = \log p(x; \theta)$$

אם נביא למקסימום את הביטוי הזה, נקבל את ה-θ האופטימלי. כיוון שלא ניתן לחשב במפורש את $p(x; \theta)$, יש להשתמש בקירוב. נניח וה-encoder הוא בעל התפלגות מסוימת $q(z|x; \lambda)$ (מה ההסתברות לקבל את z בהינתן x בכניסה). כעת ניתן לחלק ולהכפיל את $L(\theta)$ ב-$q(z; \lambda)$:

$$\log p(x; \theta) = \log \sum_z p(x, z; \theta) = \log \sum_z q(z; \lambda) \frac{p(x, z; \theta)}{q(z; \lambda)} \geq \sum_z q(z; \lambda) \log \frac{p(x, z_i; \theta)}{q(z; \lambda)}$$

כאשר אי השוויון האחרון נובע מאי-שוויון ינסן, והביטוי שמימין לאי השיוויון נקרא Evidence Lower BOund ($ELBO(\theta, \lambda)$). ניתן להוכיח שההפרש בין ה-ELBO לבין הערך שלפני הקירוב הוא המרחק בין שתי ההתפלגויות $p(z|x), q(z)$, והוא נקרא Kullback–Leibler divergence ומסומן ב-$\mathcal{D}_{KL}$:

$$\log p(x; \theta) = ELBO(\theta, \lambda) + \mathcal{D}_{KL}(q(z; \lambda) \| p(z|x; \theta))$$

אם שתי ההתפלגויות זהות, אזי מרחק $\mathcal{D}_{KL}$ ביניהן הוא 0 ומתקבל שוויון: $\log p(x; \theta) = ELBO(\theta, \lambda)$. כזכור, אנחנו מחפשים למקסם את פונקציית המחיר $\log p(x; \theta)$, וכעת בעזרת הקירוב ניתן לרשום:

$$L(\theta) = \log p(x; \theta) \geq ELBO(\theta, \lambda)$$

$$\rightarrow \theta_{ML} = \arg\max_\theta L(\theta) = \arg\max_\theta \max_\lambda ELBO(\theta, \lambda)$$

כעת ניתן בעזרת שיטת GD למצוא את האופטימום של הביטוי, וממנו להפיק את הפרמטרים האופטימליים של ה-encoder ושל ה-decoder. נפתח יותר את ה-$ELBO(\theta, \lambda)$ עבור VAE, ביחס לשתי התפלגויות:

$p(x|z; \theta)$ – ההסתברות ש-decoder עם סט פרמטרים θ יוציא x בהינתן z.

$q(z|x; \lambda)$ –ההסתברות ש-encoder עם סט פרמטרים λ יוציא את z_i בהינתן x בכניסה

לפי הגדרה:

$$ELBO(\theta,\lambda) = \sum_z q(z|x;\lambda)\log p(x,z;\theta) - \sum_z q(z|x;\lambda)\log q(z|x;\lambda)$$

את הביטוי $\log p(x,z;\theta)$ ניתן לפתוח לפי בייס $p(x,z) = p(x|z)\cdot p(z)$:

$$= \sum_z q(z|x;\lambda)(\log p(x|z;\theta) + \log p(z;\theta)) - \sum_z q(z|x;\lambda)\log q(z|x;\lambda)$$

$$= \sum_z q(z|x;\lambda)\log p(x|z;\theta) - \sum_z q(z|x;\lambda)(\log q(z|x;\lambda) - \log p(z;\theta))$$

$$= \sum_z q(z|x;\lambda)\log p(x|z;\theta) - \sum_z q(z|x;\lambda)\frac{\log q(z|x;\lambda)}{\log p(z;\theta)}$$

הביטוי השני לפי הגדרה שווה ל-$\mathcal{D}_{KL}(q(z|x;\lambda)\|p(z;\theta))$, לכן מתקבל:

$$= \sum_z q(z|x;\lambda)\log p(x|z;\theta) - \mathcal{D}_{KL}(q(z|x;\lambda)\|p(z))$$

הביטוי הראשון הוא בדיוק התוחלת של $\log p(x|z;\theta)$. תחת ההנחה ש-z מתפלג נורמלית, ניתן לרשום:

$$= \mathbb{E}_{q(z|x;\lambda)}\log N\big(x;\mu_\theta(z),\sigma_\theta(z)\big) - \mathcal{D}_{KL}\big(N\big(\mu_\lambda(x),\sigma_\lambda(x)\big)\|N(0,I)\big)$$

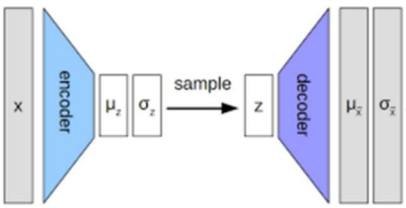
כדי לחשב את התוחלת ניתן פשוט לדגום דוגמאות מההתפלגות $z|x\sim N\big(\mu_\theta(x),\sigma_\theta(x)\big)$ ולקבל:

$$\mathbb{E}_{q(z|x;\lambda)}\log N\big(x;\mu_\theta(z),\sigma_\theta(z)\big) \approx \log N\big(x;\mu_\theta(z),\sigma_\theta(z)\big)$$

ועבור הביטוי השני יש נוסחה סגורה:

$$\mathcal{D}_{KL}(N(\mu,\sigma^2)\|N(0,I)) = \frac{1}{2}(\mu^2 + \sigma^2 - \log\sigma^2)$$

כעת משיש בידינו נוסחה לחישוב פונקציית המחיר, נוכל לבצע את תהליך הלמידה. יש לשים לב שפונקציית המחיר המקורית הייתה תלויה רק ב-θ, אך באופן שפיתחנו אותה היא למעשה דואגת גם למזעור ההפרש בין הכניסה למוצא, וגם למזעור ההפרש בין ההתפלגות הפריורית z לבין ההתפלגות $z|x$ שבמוצא ה-encoder.



$$x_t \to \mu_\lambda(x_t),\Sigma_\lambda(x_t) \to z_t\sim N\big(\mu_\lambda(x_t),\Sigma_\lambda(x_t)\big) \to \mu_\theta(z_t),\Sigma_\theta(z_t)$$

$$ELBO = \sum_t \log N\big(x_t;\mu_\theta(z_t),\Sigma_\theta(z_t)\big) - \mathcal{D}_{KL}(N\big(\mu_\lambda(x_t),\Sigma_\lambda(x_t)\big)||N(0,I)$$

איור 7.6 תהליך הלמידה של VAE.

כאשר נתון אוסף דוגמאות x, ניתן להעביר כל דוגמה x_t ב-encoder ולקבל עבורה את $\mu_\lambda,\sigma_\lambda$. לאחר מכן דוגמים וקטור לטנטי z מההתפלגות עם פרמטרים, מעבירים אותו ב-decoder ומקבלים את μ_θ,σ_θ. לאחר התהליך ניתן להציב את הפרמטרים המתקבלים ב-ELBO ולחשב את ה-Loss. ניתן לשים לב שה-ELBO מורכב משני איברים – האיבר הראשון מחשב את היחס בין הדוגמה שבכניסה לבין ההתפלגות שמתקבלת במוצא, והאיבר השני מבצע רגולריזציה להתפלגות הפריורית במרחב הלטנטי. הרגולריזציה גורמת לכך שההתפלגות במרחב הלטנטי $z|x$ תהיה קרובה עד כמה שניתן להתפלגות הפריורית z. אם ההתפלגות במרחב הלטנטי קרובה להתפלגות הפריורית, אז ניתן בעזרת ה-decoder ליצור דוגמאות חדשות, ובמובן הזה ה-VAE הוא מודל גנרטיבי.

הדגימה של z מההתפלגות במרחב הלטנטי יוצרת קושי בחישוב הגרדיאנט של ה-ELBO, לכן בדרך כלל מבצעים Reparameterization trick – דוגמים z_0 מהתפלגות נורמלית סטנדרטית, ואז כדי לקבל את z משתמשים בפרמטרים של ה-encoder: $z = z_0\sigma_\lambda(x) + \mu_\lambda(x)$. בגישה הזו כל התהליך נהיה דטרמיניסטי – מגרילים z_0 מראש ואז רק נשאר לחשב באופן סכמתי את ה-forward-backward.

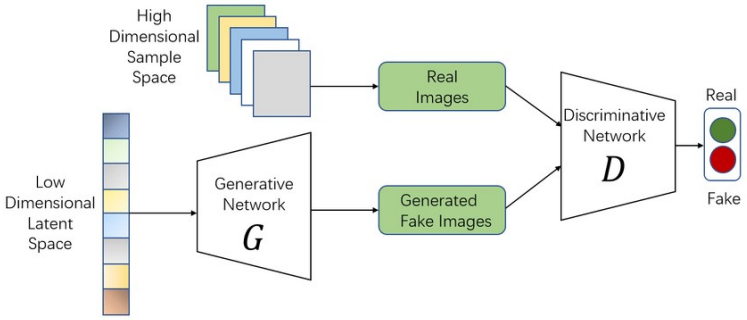
7.2 Generative Adversarial Networks (GANs)

גישה אחרת של מודל גנרטיבי נקראת Generative Adversarial Networks או בקיצור GANs, ובשונה מ-VAE בגישה זו לא מנסים לשערך התפלגות של דאטה בצורה מפורשת, אלא יוצרים דאטה באופן אחר. הרעיון הוא לאמן שתי רשתות במקביל – רשת אחת שלומדת לייצר דוגמאות, ורשת שניה שלומדת להבחין בין דוגמה אמיתית מקבוצת האימון לבין תמונה סינטטית שנוצרה על ידי הרשת הראשונה. הרשת הראשונה מאומנת ליצור דוגמאות שיגרמו לרשת השנייה לחשוב שהן אמיתיות, בזמן שהמטרה של הרשת השנייה היא לא לתת לרשת הראשונה לבלבל אותה. באופן הזה הרשת הראשונה מהווה למעשה מודל גנרטיבי, שלאחר שלב האימון היא מסוגלת לייצר דאטה סינטטי שלא ניתן להבין בינו לבין דאטה אמיתי.

7.2.1 Generator and Discriminator

בפרק זה נסביר את המבנה של ה-GAN הקלאסי שהומצא בשנת 2014 על ידי Ian Goodfellow ושותפיו. נציין שקיימות מאות רבות של וריאנטים שונים של GAN שהוצעו מאז, ועדיין תחום זה הוא פעיל מאוד מבחינה מחקרית.

כאמור, GAN מבוסס על שני אלמנטים מרכזיים – רשת שיוצרת דאטה (generator) ורשת שמכריעה האם הדאטה הזה סינטטי או אמיתי (discriminator), כאשר האימון נעשה על שתי הרשתות יחד. ה-discriminator מקבל כקלט את דוגמאות אמיתיות והן את הפלט של ה-generator, כדי ללמוד להבחין בין דאטה אמיתי לבין דאטה סינטטי, וה-generator מייצר דוגמאות ומקבל פידבק מה-discriminator וכך לומד לייצר דוגמאות שנראות אמיתיות. יש להדגיש שה-generator לא רואה דוגמאות אמיתיות לכל אורך האימון, אלא מפיק את המידע רק על בסיס המשוב מה-discriminator. נסמן את ה-generator ב-G ואת הdiscriminator ב-D, ונקבל את הסכמה הבאה:



איור 7.7 ארכיטקטורת GAN.

ה-discriminator D הוא למעשה מסווג רך שהפלט שלו הוא ההסתברות שהקלט הינו דוגמה אמיתית, כאשר נסמן ב-$D(x)$ את ההסתברות הזו (ניתן כמובן להעביר את התוצאה ב-argmax ולקבל $y = 1$ עבור דוגמה אמיתית, ו-$y = 0$ עבור דוגמה סינטטית). כדי לאמן את ה-discriminator נרצה למקסם את $D(x)$ (כלומר, למצוא את ה-discriminator שטועה כמה שפחות בזיהוי דאטה אמיתי), ובכדי לעשות זו ננסה להביא למינימום את ה-cross entropy עבור $y = 1$:

$$\min_D\{-y\log D(x) - (1-y)\log\big(1 - D(x)\big)\} = \min_D\{-y\log D(x)\}$$

באופן דומה נרצה לאמן את ה-generator כך שהדאטה שהוא מייצר יהיה כמה שיותר דומה לאמיתי, כלומר ה-generator מעוניין לגרום ל-discriminator להוציא ערכים כמה שיותר גבוהים עבור הדאטה הסינטטי שהוא מייצר. לשם כך, ה-generator יהיה מעוניין למקסם את הנוסחה שלעיל עבור $y = 0$ (כיוון שאלו המקרים שבו הדאטה הוא לא אמיתי, ועבורם נרצה שה-discriminator ייתן ערך כמה שיותר גבוה):

$$\max_G\left\{-(1-y)\log\Big(1 - D\big(G(z)\big)\Big)\right\} = \max_G\left\{-\log\Big(1 - D\big(G(z)\big)\Big)\right\}$$

ניתן לשים לב כי הקלט של ה-GAN הינו וקטור של רעש אקראי, כאשר המטרה של ה-generator הינה ללמוד ליצור דוגמאות אותנטיות מווקטור זה. בדרך כלל הרעש הינו גאוסי בעל תוחלת אפס ומטריצת covariance $\mathbb{I}$ (שונות 1 לכל משתנה ושונות משותפת 0 לכל זוג משתנים), אך קיימים גם GAN-ים עם קלט המפולג באופן אחר.

אם מחברים את שני האילוצים האלה מקבלים את פונקציית המחיר של ה-GAN:

$$V(D,G) = \min_D \max_G -\mathbb{E}_{x\sim Data} \log D(x) - \mathbb{E}_{z\sim Noise} \log\Big(1 - D\big(G(z)\big)\Big)$$

באופן שקול ניתן להפוך את האילוצים, ולאחר ביטול סימן המינוס מתקבלת בעיית האופטימיזציה הבאה:

$$V(D,G) = \min_G \max_D \mathbb{E}_{x\sim Data} \log D(x) + \mathbb{E}_{z\sim Noise} \log\Big(1 - D\big(G(z)\big)\Big)$$

ה-discriminator מעוניין למקסם את פונקציית המחיר, כך ש-$D(x)$ יהיה כמה שיותר קרוב ל-1 ו-$D\big(G(z)\big)$ יהיה כמה שיותר קרוב ל-0. ה-generator לעומת זאת רוצה להביא למינימום את פונקציית המחיר, כך ש-$D\big(G(z)\big)$ יהיה כמה שיותר קרוב ל-1, כלומר ה-discriminator חושב ש-$G(z)$ הוא דאטה אמיתי. בטרמינולוגיה של תורת המשחקים ניתן להסתכל על תהליך אימון של GAN בתור משחק סכום אפס של שני שחקנים שלא משתפים פעולה, כלומר כאשר אחד מנצח, השני בהכרח מפסיד. כמובן שהשחקן הראשון כאן הוא G והשני הוא D.

כעת האימון נעשה באופן איטרטיבי, כאשר פעם אחת מקבעים את G ומאמנים את D, ופעם אחת מקבעים את D ומאמנים את G. אם מקבעים את G, אז למעשה מאמנים מסווג בינארי, כאשר מחפשים את האופטימום התלוי בוקטור הפרמטרים ϕ_d:

$$\max_{\phi_d} \mathbb{E}_{x\sim Data} \log D_{\phi_d}(x) + \mathbb{E}_{z\sim Noise} \log\left(1 - D_{\phi_d}\left(G_{\theta_g}(z)\right)\right)$$

אם לעומת זאת מקבעים את D, אז ניתן להתעלם מהאיבר הראשון כיוון שהוא פונקציה של ϕ_d בלבד וקבוע ביחס ל-θ_g. לכן נשאר רק לבדוק את הביטוי השני, שמחפש את ה-generator שמייצר דאטה שנראה אמיתי בצורה הטובה ביותר:

$$\min_{\theta_g} \mathbb{E}_{z\sim Noise} \log\left(1 - D_{\phi_d}\left(G_{\theta_g}(z)\right)\right)$$

כאמור, המטרה היא לאמן את G בעזרת D (במצבו הנוכחי), כדי שיהיה מסוגל ליצור דוגמאות המסווגות. האימון של ה-generator נעשה באמצעות Gradient Ascent (מקסום פונקציית המחיר ביחס ל-θ_g), והאימון של ה-discriminator נעשה באמצעות Gradient Descent (מזעור פונקציית המחיר ביחס ל-ϕ_D). האימון מתבצע במשך מספר מסוים של Epochs, כאשר מאמנים לסירוגין את G ו-D. בפועל דוגמים mini-batch בגודל m מהדאטה-סט של האימון $(x_1, \dots, x_m)$ ו-m דגימות מרעש נורמלי $(z_1, \dots, z_m)$, ומכניסים את הקלט ל-G. הגרדיאנט של פונקציית המחיר לפי ה-generator במהלך האימון מחושב באופן הבא:

$$\nabla_\theta V\big(G_\theta, D_\phi\big) = \frac{1}{m} \nabla_\theta \sum_{i=1}^{m} \log\Big(1 - D_\phi\big(G_\theta(z_i)\big)\Big)$$

וכאשר מאמנים את ה-discriminator, הגרדיאנט נראה כך:

$$\nabla_\phi V\big(G_\theta, D_\phi\big) = \frac{1}{m} \nabla_\phi \sum_{i=1}^{m} \log D_\phi(x_i) + \log\Big(1 - D_\phi\big(G_\theta(z_i)\big)\Big)$$

נהוג לבצע מודיפיקציה קטנה על פונקציית המטרה של ה-generator. כיוון שבהתחלה הדגימות המיוצרות על ידי ה-generator לא דומות לחלוטין לאלו מקבוצת האימון, ה-discriminator מזהה אותן בקלות כמזויפות. כתוצאה מכך הביטוי $D\big(G(z)\big)$ מקבל ערכים מאוד קרובים ל-0, וממילא גם הביטוי $\mathbb{E}_{z\sim Noise} \log\Big(1 - D\big(G(z)\big)\Big)$ קרוב ל-0. עניין זה גורם לכך שהגרדיאנט של ה-generator גם יהיה מאוד קטן, ולכן כמעט ולא מתבצע שיפור ב-generator. לכן במקום לחפש מינימום של הביטוי $\mathbb{E}_{z\sim Noise} \log\Big(1 - D\big(G(z)\big)\Big)$ מחפשים מינימום לביטוי $-\mathbb{E}_{z\sim Noise} \log\Big(D\big(G(z)\big)\Big)$. הביטויים לא שווים לגמרי אך שניהם מובילים לאותו פתרון של בעיית האופטימיזציה אותה הם מייצגים, והביטוי החדש עובד יותר טוב נומרית ומצליח לשפר את ה-generator בצורה יעילה יותר.

Optimal values

כזכור, פונקציית המחיר הינה:

$$V(D,G) = \min_G \max_D \mathbb{E}_{x\sim Data} \log D(x) + \mathbb{E}_{z\sim Noise} \log\Big(1 - D\big(G(z)\big)\Big)$$

נרצה לחשב מה הערך האופטימלי של ה-Discriminator, ועבורו לחשב את הערך של פונקציית המחיר. לשם הנוחות נסמן את התפלגות הדאטה האמיתי ב-p_r, ואת התפלגות הדאטה הסינטטי המיוצר על ידי ה-generator ב-p_g. עבור G קבוע, ניתן לרשום את פונקציית המחיר כך:

$$V(D,G) = \int_x p_r(x) \log D(x) + p_g(x) \log\big(1 - D(x)\big)\, dx$$

הביטוי החשוב הוא האינטגרד עצמו, וניתן להתעלם מחישוב האינטגרל כיוון שעוברים על כל ערכי x האפשריים. לכן הפונקציה לה מעוניינים למצוא אופטימום הינה:

$$f\big(D(x)\big) = p_r(x) \log D(x) + p_g(x) \log\big(1 - D(x)\big)$$

נגזור ונשווה ל-0:

$$\frac{\partial f\big(D(x)\big)}{\partial D(x)} = \frac{p_r(x)}{D(x)} - \frac{p_g(x)}{1 - D(x)} = 0$$

$$\rightarrow p_r(x)\big(1 - D(x)\big) - p_g(x) D(x) = 0$$

$$D(x)_{opt} = \frac{p_r(x)}{p_r(x) + p_g(x)}$$

הביטוי שהתקבל הינו ה-discriminator האופטימלי עבור generator קבוע. נשים לב שעבור המקרה בו ה-GAN מצליח לייצר דוגמאות שנראות אמיתיות לחלוטין, כלומר $p_g(x) \approx p_r(x)$, אז מתקיים $D(x) \approx \frac{1}{2}$. הסתברות זו משמעותה שה-discriminator לא יודע להחליט לגבי הקלט המתקבל, והוא קובע שההסתברות שהקלט אמיתי זהה לכך שהקלט סינטטי.

כעת נבחן מהו ערך פונקציית המחיר כאשר D אופטימלי:

$$V(G,D) = \mathbb{E}_{x\sim Data} \log D(x) + \mathbb{E}_{z\sim Noise} \log\Big(1 - D\big(G(z)\big)\Big)$$

$$= \mathbb{E}_{x\sim Data} \log\left(\frac{p_r(x)}{p_r(x) + p_g(x)}\right) + \mathbb{E}_{z\sim Noise} \log\left(1 - \left(\frac{p_r(x)}{p_r(x) + p_g(x)}\right)\right)$$

$$= \mathbb{E}_{x\sim Data} \log\left(\frac{p_r(x)}{p_r(x) + p_g(x)}\right) + \mathbb{E}_{z\sim Noise} \log\left(\frac{p_g(x)}{p_r(x) + p_g(x)}\right)$$

$$= \mathbb{E}_{x\sim Data} \log\left(\frac{p_r(x)}{\frac{\big(p_r(x) + p_g(x)\big)}{2}}\right) + \mathbb{E}_{z\sim Noise} \log\left(\frac{p_g(x)}{\frac{\big(p_r(x) + p_g(x)\big)}{2}}\right) - \log 4$$

הביטוי שמתקבל, לפי הגדרה, הינו מרחק הנקרא Jensen-Shannon divergence ומסומן ב-$\mathcal{D}_{JS}$. מרחק זה הינו גרסה סימטרית של Kullback–Leibler divergence ($\mathcal{D}_{KL}$), ועבור שתי התפלגויות P, Q הוא מוגדר באופן הבא:

$$\mathcal{D}_{JS} = \frac{1}{2}\mathcal{D}_{KL}(P||M) + \frac{1}{2}\mathcal{D}_{KL}(Q||M), M = \frac{1}{2}(P + Q)$$

קיבלנו שעבור D אופטימלי, פונקציית המחיר שווה למרחק זה עד כדי קבוע, ובאופן מפורש:

$$V(G, D_{opt}) = \mathcal{D}_{JS}(p_r, p_g) - \log 4$$

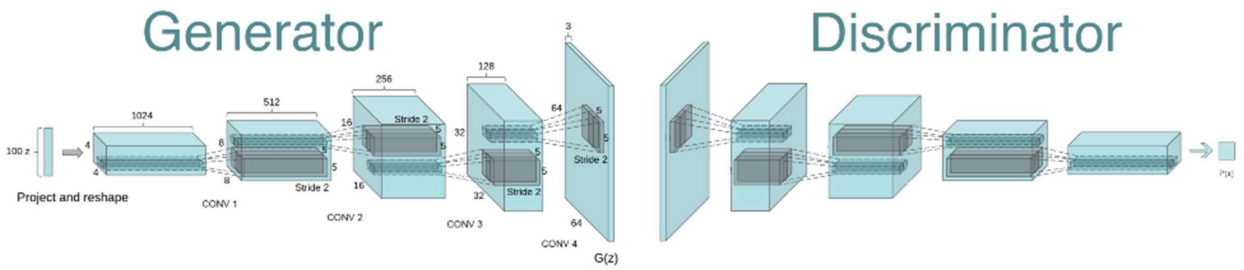
כאשר G אופטימלי ומתקיים $p_g(x) \approx p_r(x)$, אז המרחק בין ההתפלגויות שווה 0, ומתקבל:

$$V(G_{opt}, D_{opt}) = \mathcal{D}_{JS}(p_r, p_g) - \log 4$$

יש משמעות גדולה לביטוי שהתקבל, כיוון שהוא למעשה אומר מה צריך לעשות בכדי לקבל את GAN אופטימלי – להביא למינימום את $\mathcal{D}_{JS}(p_r, p_g)$.

7.2.2 Deep Convolutional GAN (DCGAN)

כפי שהוסבר בפרק 5, רשתות קונבולוציה יעילות יותר בדומיין של תמונות מאשר רשתות FC. לכן היה טבעי לקחת רשתות קונבולוציה ולהשתמש בהן בתור generator ו-discriminator עבור דומיין של תמונות. ה-generator מקבל וקטור אקראי ומעביר אותו דרך רשת קונבולוציה על מנת ליצור תמונה, וה-discriminator מקבל תמונה ומעביר אותו דרך רשת קונבולוציה שעושה סיווג בינארי אם התמונה אמיתית או סינטטית.



איור 7.8 ארכיטקטורת DCGAN.

7.2.3 Conditional GAN (cGAN)

לעיתים מודל גנרטיבי נדרש לייצר דוגמה בעלת מאפיין ספציפי ולא רק דוגמה שנראית אוטנטית. למשל אם נתון דאטה של תמונות המייצגות את הספרות מ-0 עד 9, ונרצה שה-GAN ייצר תמונה של ספרה מסוימת. במקרים אלו, בנוסף לווקטור הכניסה z, ה-GAN מקבל תנאי נוסף על הפלט אותו הוא צריך לייצר, כמו למשל ספרה ספציפית אותה רוצים לקבל. GAN כזה נקרא conditional GAN, ופונקציית המחיר שלו דומה מאוד לפונקציית המחיר של GAN רגיל למעט העובדה שהביטויים הופכים להיות מותנים:

$$\mathcal{L}_c(D, G) = \min_G \max_D \mathbb{E}_{x\sim Data} \log D(x|\boldsymbol{y}) + \mathbb{E}_{z\sim Noise} \log\Big(1 - D\big(G(z|\boldsymbol{y})\big)\Big)$$

7.2.4 Pix2Pix

כפי שראינו, ה-GAN הקלאסי שתואר לעיל מסוגל לייצר דוגמאות חדשות מווקטור אקראי z, המוגרל מהתפלגות מסוימת (בדרך כלל התפלגות גאוסית סטנדרטית, אך זה לא מוכרח). ישנן גישות נוספות ליצור דאטה חדש, כמו למשל ייצור תמונה חדשה על בסיס קווי מתאר כלליים שלה. קבוצת האימון במקרה זה בנויה מזוגות של תמונות והסקיצות שלהן.

שיטת Pix2Pix משתמשת בארכיטקטורה של GAN אך במקום לדגום את וקטור z מהתפלגות כלשהיא, Pix2Pix מקבלת סקיצה של תמונה בתור קלט, וה-generator לומד להפוך את הסקיצה לתמונה אמיתית. הארכיטקטורה של ה-generator נשארת ללא שינוי ביחס למה שתואר קודם לכן (פרט להתאמה למבנה הקלט), אך ה-discriminator כן משתנה – במקום לקבל תמונה ולבצע עליה סיווג בינארי, הוא מקבל זוג תמונות – את הסקיצה ואת התמונה (פעם תמונה מקבוצת האימון המתאימה לסקיצה S ופעם זאת שמיוצרת על ידי ה-generator על בסיס S). על ה-discriminator לקבוע האם התמונה היא אכן תמונה אמיתית של הסקיצה או תמונה סינטטית. ווריאציה זו של ה-GAN משנה גם את פונקציית המחיר – כעת ה-generator צריך ללמוד שני דברים – גם ליצור תמונות טובות כך שה-discriminator יאמין שהן אמיתיות, וגם למזער את המרחק בין התמונה שנוצרת לבין תמונה אמיתית השייכת לסקיצה.

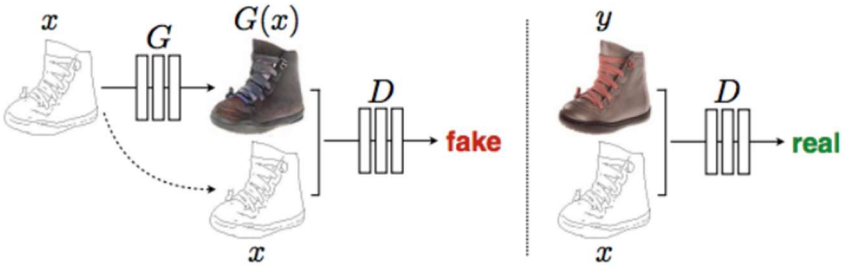
כעת נסמן תמונה אמיתית השייכת לסקיצה ב-y, ונרשום את פונקציית המחיר כשני חלקים נפרדים – cross entropy רגיל של GAN ומרחק אוקלידי L_1 בין תמונת המקור לבין הפלט:

$$V(D, G) = \min_G \max_D \mathbb{E}_{x,y}\left(\log D(x, y) + \log\Big(1 - D\big(x, G(x)\big)\Big)\right)$$

$$\mathcal{L}_{L1}(G) = \min_{\theta_g} \mathbb{E}_{x,y} \lambda \|G(x) - y\|_1$$

$$\mathcal{L}(G, D) = \min_{G} \max_{D} V(D, G) + \mathcal{L}_{L1}(G)$$

ניתן להסתכל על pix2pix בתור GAN הממפה תמונה לתמונה (image-to-image translation). נציין שבמקרה זה הקלט והפלט pix2pix שייכים לדומיינים שונים (סקיצה ותמונה רגילה).



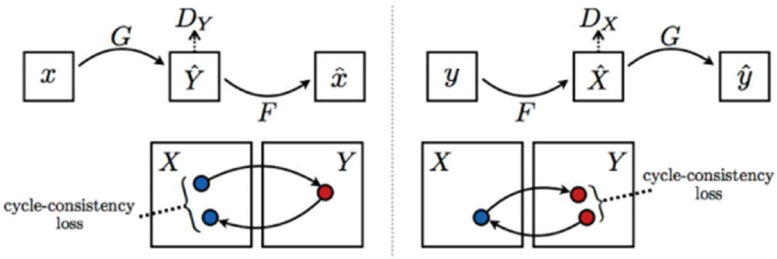
איור 7.9 ארכיטקטורת Pix2Pix - Image-to-Image Translation.

7.2.5 CycleGAN

ב-Pix2Pix הדאטה המקורי הגיע בזוגות – סקיצה ואיתה תמונה אמיתית. זוגות של תמונות זה לא דבר כל כך זמין, ולכן שיפרו את תהליך האימון כך שיהיה ניתן לבצע אותו על שני סטים של דאטה מדומיינים שונים. הארכיטקטורה עבור המשימה הזו מורכבת משני generators – בהתחלה מכניסים דוגמה מהדומיין הראשון x ל-G, שמנסה להפוך אותו לדוגמה מהדומיין השני y, והפלט נכנס ל-F שמנסה לשחזר את המקור x. המוצא של ה-G נכנס לא רק ל-F אלא גם ל-discriminator D_y שנועד לזהות האם התמונה שהתקבלה הינה אמיתית או לא (עבור הדומיין של y). ניתן לבצע את התהליך הזה באופן דואלי עבור y – מכניסים את y ל-F על מנת לקבל את x ואת המוצא מכניסים ל-discriminator D_x בכדי לבצע סיווג בינארי ול-G על מנת לנסות לשחזר את המקור. ה-generator השני F נועד לשפר את תהליך הלמידה – לאחר ש-x הופך ל-y דרך G, ניתן לקבל חזרה את x אם נעביר את y דרך F מתוך ציפייה לקבל $F(G(x)) \approx x$. התהליך של השוואת הכניסה למוצא נקרא cycle-consistency, והוא מוסיף עוד איבר לפונקציית המחיר, שמטרתו למזער עד כמה שניתן את המרחק בין התמונה המקורית לתמונה המשוחזרת:

$$V(D_x, D_y, G, F) = \mathcal{L}_{GAN}(G, D_y, x, y) + \mathcal{L}_{GAN}(F, D_x, x, y)$$

$$+\lambda \left(\mathbb{E}_x \|F(G(x)) - x\|_1 + \mathbb{E}_y \|G(F(y)) - y\|_1 \right)$$



איור 7.10 ארכיטקטורת CycleGAN.

7.2.8 Wasserstein GAN

אחד ה-GANs היסודיים ביותר הינו וסרשטיין גאן (Wasserstein GAN), והוא נוגע בבעיה שיש בפונקציית המחיר בה משתמשות הרבה רשתות גנרטיביות אחרות. כאמור, תהליך הלמידה של הרשת המייצרת דאטה – ה-generator – נעשה באמצעות משוב המתקבל מה-discriminator. בעוד שה-discriminator מאומן להבחין בין דאטה אמיתי לדאטה סינטטי הן בעזרת דאטה אמיתי והן בעזרת דאטה שה-generator מייצר, ה-generator לא מסתמך על דוגמאות אמיתיות אלא רק על המשוב מה-discriminator. משום כך, בתחילת הלמידה, כאשר הגנרטור עוד לא מאומן, הדאטה הסינטטי שהוא מייצר אינו דומה כולל לדאטה האמיתי, וה-discriminator מבחין בקלות ביניהם. במילים אחרות, בתחילת תהליך הלמידה ה-discriminator טוב יותר מאשר ה-generator. פער זה יוצר בעיה בתהליך ההשתפרות של ה-generator, כיוון שהשיפור מתבסס על הידע העובר בעזרת הגרדיאנט של פונקציית המחיר (loss) שלו, התלוי בערכים אותם מוציא ה-discriminator. כדי להבין מדוע תהליך העברת המידע באופן

הזה בעייתית, יש להרחיב מעט על תהליך היצירה של הדאטה על ידי ה-generator ואיך ה-discriminator מסתכל על דאטה זה.

ההנחה היסודית ברוב המודלים הגנרטיביים, ובפרט ב-GANs, הינה שהדאטה הרב ממדי (למשל תמונות) "חי" במשטח מממד נמוך בתוכו. אפשר להסתכל על משטח בתור הכללה של תת-מרחב וקטורי מממד נמוך הנפרס על ידי תת-קבוצה של וקטורי בסיס של מרחב וקטורי מממד גבוה יותר. גם המשטח נוצר מתת-קבוצה של וקטורי הבסיס של "מרחב האם", אך ההבדל בינו לבין תת-מרחב וקטורי מתבטא בכך שלמשטח עשויה להיות צורה מאוד מורכבת יחסית לתת-מרחב וקטורי. משתמע מכך שניתן לייצר דאטה רב ממדי על ידי טרנספורמציה של וקטור ממרחב בעל ממד נמוך (וקטור לטנטי). למשל, ניתן בעזרת רשת נוירונים לייצר תמונה בגודל $64 \times 64 \times 3 > 12k$ פיקסלים מווקטור באורך 100 בלבד. זאת אומרת, שגם התפלגות התמונות של הרשת הגנרטיבית וגם ההתפלגות של הדאטה אמיתי נמצאים ב"משטח בעלי ממד נמוך" בתוך מרחב בעל ממד גבוה של הדאטה האמיתי. באופן פורמלי יותר, משטח זה נקרא מניפולד (manifold), וההשערה שתוארה מעלה מהווה הנחת יסוד בתחום הנקרא למידת מניפולד (manifold learning). מכיוון שמדובר במשטחים בעלי ממד נמוך בתוך מרחב בעל ממד גבוה, קיימת סבירות גבוהה שלא יהיה שום חיתוך ביניהם, ויתרה מכך, המרחק ביניהם יהיה די גדול. מכך נובע שה-discriminator D עשוי ללמוד להבחין בין הדאטה האמיתי לסינטטי בקלות, כיוון שבמרחב מממד גבוה יש מרחק גדול בין מניפולד אמיתי לבין מניפולד שיוצר באופן סינטטי, כאשר שניהם הינם משטחים מממד יותר נמוך. בנוסף, D כנראה ייתן לדוגמאות סינטטיות ציונים (score) ממש קרובים לאפס כי אכן קל מאוד למצוא "משטח הפרדה" בין שני המניפולדים – זה של הדוגמאות האמיתיות וזה של הסינטטיות, כיוון שהם נוטים להיות רחוקים מאוד אחד מהשני.

רקע זה מסייע להבין מדוע הפער שיש בין ה-generator וה-discriminator מבחינת אופי הלמידה מהווה בעיה. כאמור, ה-generator מעדכן את המשקלים שלו על סמך הציונים שהוא מקבל מה-discriminator (דרך פונקציית המחיר של ב-GAN). אבל אם ה-discriminator כל הזמן מוציא ציונים מאוד נמוכים (עקב מרחק גדול בין המניפולדים שתואר מעלה) לדוגמאות המיוצרות על ידי ה-generator, ה-generator פשוט לא יצליח לשפר את איכות התמונות. במילים פשוטות, D "פשוט הרבה יותר מדי טוב יחסית ל-G". אתגר זה בא לידי ביטוי גם בצורה של פונקציית המחיר, שלא מאפשרת "העברה יעילה של ידע" מהדיסקרימנטור לגנרטור.

יש מספר לא קטן של שיטות הבאות לשפר את תהליך האימון של GAN כאשר הבולטות הן:

- התאמת פיצ'רים (feature matching).
- minibatch discrimination.
- virtual batch normalization.
- מיצוע היסטורי.

כפי שהוסבר, הבעיה של מרחק המניפולדים משתקפת במבנה של פונקציית המחיר, וכיוון שכך, ניתן לנסות ולפתור את הבעיה מהשורש על ידי שימוש בפונקציית מחיר יותר מתאימה. לשם כך ראשית נסמן את התפלגות הדאטה האמיתי ב-p_r, ואת התפלגות הדאטה הסינטטי המיוצר על ידי ה-generator ב-p_g. לעיל הראינו שפונקציית המחיר האופטימלית הממזערת את המרחק בין ההתפלגויות p_r, p_g, מתוארת על ידי $\mathcal{D}_{JS}$ – Jensen-Shannon divergence.

ניתן להוכיח כי מרחק $\mathcal{D}_{JS}$ בין ההתפלגויות p_r, p_g לא רגיש לשינוים ב-p_g כאשר המשטחים שבהם "חיים" p_r, p_g רחוקים אחד מהשני. כלומר, מרחק D_{JS} כמעט ולא ישתנה אחרי עדכון המשקלים של ה-generator, וממילא לא ישקף את המרחק המעודכן בין שתי ההתפלגויות p_r, p_g. זו למשעה הבעיה המהותית ביותר עם פונקציית המחיר המקורית של ה-generator, שעדכון המשקלים לא משפיע כמעט על $\mathcal{D}_{JS}$, כיוון שמראש ההתפלגויות p_r, p_g רחוקות אחת מהשנייה.

Wasserstein GAN בא להתמודד עם בעיה זו, ולהציע פונקציית מחיר אחרת שעבורה עדכון המשקלים ישתקף גם במרחק בין ההתפלגויות p_r, p_g. פונקציית המחיר החדשה מבוססת על מרחק הנקרא Earth Mover (EM), המהווה מקרה פרטי של מרחק וסרשטיין ($\mathcal{D}_W$). מרחק וסרשטיין מסדר $p \geq 1$ בין שתי מידות הסתברות μ, ν על מרחב M מוגדר באופן הבא:

$$W_p(\mu,\nu) = \inf_{\gamma \in \Gamma(\mu,\nu)} \mathbb{E}_{(x,y)\sim\gamma}[\|x-y\|] = \left(\inf_{\gamma \in \Gamma(\mu,\nu)} \int_{M\times M} d(x,y)^p d\gamma(x,y) \right)^{\frac{1}{p}}$$

כאשר $\Gamma(\mu,\nu)$ הן כל מידות הסתברות על מרחב המכפלה (product space) של M עם עצמו (זהו למעשה מרחב המכיל את כל הזוגות האפשריים של האלמנטים מ-M) עם פונקציות שוליות (marginal) השוות ל-μ, ν בהתאמה. תחת סימן האינטגרל יש את המרחק האוקלידי מסדר p בין הנקודות. מרחק EM הינו מקרה פרטי של מרחק וסרשטיין, כאשר $p = 1$, ובאופן מפורש:

$$EM = W_1(\mu,\nu) = \inf_{\gamma\in\Gamma(\mu,\nu)} \int_{M\times M} d(x,y)d\gamma(x,y)$$

הגדרה זו נראית מאוד מסובכת וננסה לתת עבורה אינטואיציה, ולהבין מדוע עבור $p = 1$, מרחק וסרשטיין נקרא EM. לשם הפשטות נניח שהמרחב M הינו חד ממדי, כלומר קו ישר, ועליו עשר משקולות של 1_{kg} כל אחת המפוזרות באופן הבא: 6 משקולות (6_{kg}) בנקודה $x = 0$, ו-4 משקולות (4_{kg}) בנקודה $x = 1$. כעת נרצה להזיז את המשקולות כך שתהיינה מפוזרות באופן הבא: בנקודה $x = 4$ יהיה משקל של 3_{kg}, בנקודה $x = 5$ יהיה משקל של 5_{kg}, ושאר המשקולות (2_{kg}) יהיו בנקודה $x = 8$.

כמובן שיש הרבה דרכים לבצע את הזזת המשקולות, ונרצה למצוא את הדרך היעילה ביותר. לשם כך נגדיר מאמץ כמכפלה של משקל במרחק אותו מזיזים את המשקל (בפיזיקה מושג זה נקרא עבודה - כוח המופעל על גוף לאורך מסלול). בדוגמה המובאת, המאמץ המינימלי מתקבל על ידי הזזת המשקולות באופן הבא:

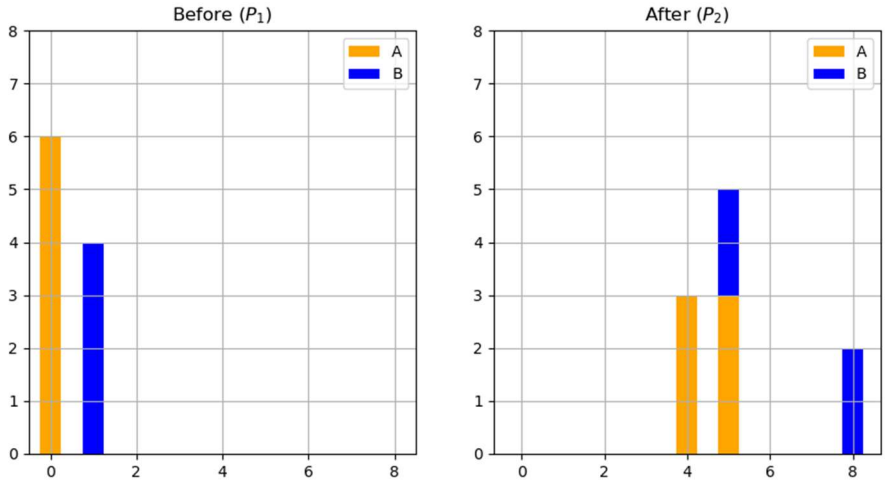
3_{kg} מועברים מ-$x = 0$ ל-$x = 4$, כאשר המאמץ הדרוש לכך הינו $(4 - 0) \cdot 3 = 12$.

3_{kg} מועברים מ-$x = 0$ ל-$x = 5$, כאשר המאמץ הדרוש לכך הינו $(5 - 0) \cdot 3 = 15$.

2_{kg} מועברים מ-$x = 1$ ל-$x = 5$, כאשר המאמץ הדרוש לכך הינו $(5 - 1) \cdot 2 = 8$.

2_{kg} מועברים מ-$x = 1$ ל-$x = 8$, כאשר המאמץ הדרוש לכך הינו $(8 - 1) \cdot 2 = 14$.

סך המאמץ המינימלי שווה במקרה הזה ל: $12 + 15 + 8 + 14 = 49$.



איור 7.11 העברת משקולות באופן אופטימלי. P_1 מייצג את המצב ההתחלתי, ו-P_2 הינו המצב לאחר הזזת המשקולות.

כעת, במקום להסתכל על משקלים, נתייחס להתפלגויות p_1, p_2, המוגדרות באופן הבא:

$$p_1(x) = \begin{cases} 0.6, x = 0 \\ 0.4, x = 1 \\ 0, else \end{cases}, p_2(x) = \begin{cases} 0.3, x = 4 \\ 0.5, x = 5 \\ 0.2, x = 8 \\ 0, else \end{cases}$$

השאלה כיצד ניתן להעביר מסה הסתברותית מ-p_1 כך שתתקבל ההתפלגות p_2, שקולה לדוגמה של הזזת המשקולות. מרחק EM בין שתי התפלגויות p_1, p_2 מוגדר להיות ה"מאמץ" המינימלי הנדרש בשביל להעביר את המסה ההסתברותית מ-p_1 ל-p_2, או במילים אחרות – מרחק EM מגדיר מהי כמות ה"עבודה" (מאמץ) המינימלית הנדרשת בשביל להפוך p_1 ל-p_2. אם נחזור לדוגמה של המשקולות, נוכל להבין מדוע $\mathcal{D}_w$ עבור $p = 1$ נקרא מרחק Earth Mover – מרחק בין שתי התפלגויות שקול לכמה מאמץ נדרש להעביר כמות אדמה במשקל מסוים כדי לעבור מחלוקה מסוימת של האדמה לחלוקה אחרת. באופן יותר פורמלי – מידת ההסתברות על מרחב המכפלה בנוסחה של מרחק EM מתארת את האופן שבו אנחנו מעבירים את המסה ההסתברותית (משקל מסוים של אדמה), כאשר הביטוי $\gamma(x,y)$ מתאר כמה מסה הסתברותית מועברת מנקודה x לנקודה y.

לאחר שהוסבר מהו מרחק וסרשטיין $\mathcal{D}_W$, ומהו מרחק EM, ניתן להבין כיצד אפשר להשתמש במושגים אלו עבור פונקציית מחיר של רשת גנרטיבית. עבור מרחק בין מידות ההסתברות, $\mathcal{D}_W$ מתחשב בתכונות של הקבוצות עליהן מידות אלו מוגדרות בצורה מפורשת, על ידי התחשבות במרחק בין הנקודות שלהם. תכונה זו היא למעשה בדיוק מה שצריך בשביל למדוד את המרחק בין ההתפלגות האמיתית של דאטה p_r לבין התפלגות של הדאטה הסינטטי p_g.

מרחק EM ידע לשערך בצורה הטובה ביותר את המרחק בין המניפולדים שבין שתי ההתפלגויות, ואם מזיזים את המניפולד של הדאטה הסינטטי, נוכל לדעת בעזרת מרחק EM עד כמה השתנה המרחק בין המניפולדים. מידע זה לא היה ידוע לנו באמצעות פונקציית המחיר המקורית הנמדדת באמצעות $\mathcal{D}_{JS}$. כעת, בעזרת פונקציית המחיר החדשה המבוססת על מרחק EM, ניתן לדעת עד כמה עדכון המשקלים מקרב או מרחיק את p_g מ-p_r.

באופן תיאורטי זה מצוין, אך עדיין זה לא מספיק, כיוון שצריך למצוא דרך לחשב את $\mathcal{D}_W$, או לכל הפחות את המקרה הפרטי שלו עבור $p = 1$, כלומר את מרחק EM. במקור מרחק זה מוגדר כבעיית אופטימיזציה של מידות הסתברות על מרחב המכפלה, וצריך למצוא דרך להשתמש בו כפונקציית מחיר. בשביל לבצע זאת, ניתן להשתמש בצורה דואלית של $\mathcal{D}_W$ עבור $p = 1$ – שיוויון RK (Rubinstein-Kantorovich), לפיו ניתן לחשב את $\mathcal{D}_W, p = 1$ באופן הבא:

$$W(p_r, p_g) = \frac{1}{K} sup_{||f||_L} E_{x\sim p}[f(x)] - E_{x\sim p_s}[f(x)]$$

כאשר $f(x)$ הינה פונקציית ליפשיץ מסדר K (כלומר, פונקציה רציפה עם קצב השתנות החסום על ידי K). כעת נניח ש-$f(w)$ הינה פונקציית K-ליפשיץ המתארת discriminator בעל סט הפרמטרים w. ה-discriminator מחשב באופן מקורב את המרחק בין ההתפלגויות באופן הבא:

$$L(p(r), p(g)) = W(p(r), p(g)) = \max_{w\in W} \mathbb{E}_{x\sim p_r}[f_w(x)] - \mathbb{E}_{z\sim p_r(z)}[f_w(g_\theta(z))]$$

פונקציית מחיר זו מודדת את המרחק $\mathcal{D}_W$ בין ההתפלגויות p_r, p_g, וככל שפונקציה זו תקבל ערכים יותר נמוכים ככה ה-generator יצליח לייצר דוגמאות שמתפלגות באופן יותר דומה לדאטה המקורי. בשונה מ-GAN קלאסי בו ה-discriminator מוציא הסתברות עד כמה הדוגמה אותה הוא מקבל אמיתית, פה ה-discriminator לא מאומן להבחין בין דוגמא אמיתית לסינטטית, אלא הוא מאומן ללמוד פונקציית K-ליפשיץ רציפה המודדת את $\mathcal{D}_W$ בין ההתפלגויות p_r, p_g. ה-generator לעומת זאת מאומן למזער את $L(p_r, p_g)$ (כאשר רק האיבר השני שתלוי ב-g_θ), וככל שפונקציית המחיר הולכת וקטנה, כך p_g מתקרב יותר ל-p_r.

כאמור, תנאי הכרחי לשימוש במרחק זה בפונקציית המחיר הינו שהפונקציה תהיה K-ליפשיץ. מסתבר שקיום זה אינו משימה קלה כלל. בכדי לשמור על רציפות, המאמר המקורי הציע לבצע קטימה לטווח סופי מסוים, נניח $[-0.01, 0.01]$, וניתן להראות כי קטימה זו מבטיחה את ש-f_w תהיה K-ליפשיץ רציפה. אולם, כמו שכותבי המאמר מודים בעצמם, ביצוע קטימה בכדי לדאוג לקיום תנאי ליפשיץ יכול לגרום לבעיות אחרות. למעשה, כאשר חלון הקטימה של המשקלים צר מדי, הגרדיאנטים של Wasserstein GAN עלולים להתאפס, מה שיעצור את תהליך הלמידה. מצד שני, כאשר חלון זה רחב מדי, ההתכנסות עלולה להיות מאוד איטית. נציין שיש עוד מספר דרכים לכפות על f_w להיות ליפשיץ-רציפה למשל gradient penalty.

האימון של Wasserstein GAN דומה לאימון של ה-GAN המקורי, למעט שני הבדלים עיקריים:

א. קיצוץ טווח המשקלים על מנת לשמור על רציפות-ליפשיץ.
ב. פונקציית ממחיר המסתמכת על $\mathcal{D}_W$ במקום על $\mathcal{D}_{JS}$.

תהליך הלמידה מתבצע באופן הבא – לאחר כל עדכון משקלים של ה-discriminator (באמצעות gradient ascent), מקצצים את טווח המשקלים. לאחר מכן מבצעים עדכון רגיל של משקלי ה-generator תוך ביצוע של איטרציה של gradient descent. Wasserstein GAN מצליח לגרום לכך שהקורלציה בין איכות התמונה הנוצרת על ידי הגנרטור לבין ערך של פונקציית לוס תהיה הרבה יותר בולטת מאשר ב-GAN רגיל בעל אותה ארכיטקטורה. הצלחה זו נובעת מהשינוי בפונקציית המחיר, שגרם לאימון להיות יותר יעיל, והביא לכך שהדאטה הסינטטי יהיה דומה הרבה יותר לדאטה המקורי.

7. References

VAE:

https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73

https://jaan.io/what-is-variational-autoencoder-vae-tutorial/

GANs:

https://arxiv.org/abs/1406.2661

https://arxiv.org/pdf/1511.06434.pdf

https://phillipi.github.io/pix2pix/

https://junyanz.github.io/CycleGAN/

AR models:

https://arxiv.org/abs/1601.06759

https://arxiv.org/abs/1606.05328

https://arxiv.org/pdf/1701.05517.pdf

https://towardsdatascience.com/auto-regressive-generative-models-pixelrnn-pixelcnn-32d192911173

https://wiki.math.uwaterloo.ca/statwiki/index.php?title=STAT946F17/Conditional_Image_Generation_with_PixelCNN_Decoders#Gated_PixelCNN