7. Deep Generative Models

המודלים שהוצגו בפרקים הקודמים הינם מודלים דיסקרימינטיביים, קרי הם מאומנים לבצע פעולות על בסיס דאטה נתון, אך לא יכולים ליצור פיסות מידע או דוגמאות חדשות בעצמם. בניגוד אליהם קיימים מודלים גנרטיביים, המסוגלים ליצור פיסות מידע חדשות על בסיס הדוגמאות שנלמדו. באופן פורמלי, בהינתן אוסף דוגמאות $X \in \mathbb{R}^{n\times d}$ ואוסף תגיות $Y \in \mathbb{R}^n$, מודל דיסקרימנטיבי מאומן לשערך את ההסתברות $\Pr(y|x)$. מודל גנרטיבי לעומת זאת לומד את ההסתברות $\Pr(x,y)$ (או את $\Pr(x)$ במקרה שהתגיות אינן נתונות), כאשר x,y הן צמד נתון של דוגמה ו-label, מתוכן ניתן לייצר דוגמאות חדשות.

ישנם שני סוגים עיקריים של מודלים גנרטיביים: סוג אחד של מודלים מאומן למצוא באופן מפורש את פונקציית הפילוג של הדאטה הנתון, ובעזרת הפילוג לייצר דוגמאות חדשות (על ידי דגימה מההתפלגות שנלמדה). סוג שני של מודלים אינו עוסק בשערוך הפילוג של הדאטה המקורי, אלא מסוגל לייצר דוגמאות חדשות בדרכים אחרות. בפרק זה נדון במודלים הפופולריים בתחום – VAE, ו-GANs, השייכים לסוג הראשון והשני של המודלים הגנרטיביים.

7.1 Variational AutoEncoder (VAE)

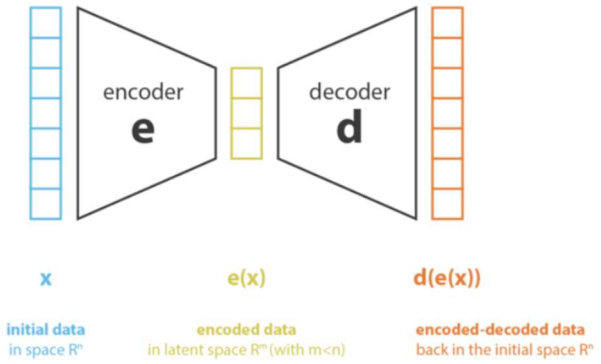
המודל הראשון הינו VAE, וכדי להבין כיצד ניתן בעזרתו לייצר מידע חדש, יש להסביר קודם מהם Autoencoders, כיצד הם עובדים ומה החסרונות שלהם. Autoencoder הינו רשת המאומנת לבצע הורדת ממד לדאטה, תוך איבוד מינימלי של מידע. בכדי להבין את המבנה ואופי הפעולה שלו, נקדים מעט על הורדת ממד באופן כללי.

7.1.1 Dimensionality Reduction

במקרים רבים, הדאטה אותו רוצים לנתח הוא בעל ממד גבוה, כלומר, לכל דגימה יש מספר רב של מאפיינים (features). לרוב, לא כל המאפיינים 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 תמונת הפנים שלו, לא כל הפיקסלים בתמונת הפנים יהיו בעלי אותה חשיבות לצורך החיזוי. כיוון שקשה לנתח דאטה מממד גבוה ולבנות מודלים עבור דאטה כזה, במקרים רבים מנסים להוריד את הממד של הדאטה תוך איבוד מידע מינימלי עד כמה שניתן. 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 המאפיינים המשמעותיים של הדאטה.

הייצוג של הדאטה בממד נמוך נקרא הייצוג הלטנטי (חבוי) או הקוד הלטנטי, וכאמור, יותר קל לבנות מודלים למשימות שונות על סמך הייצוג הלטנטי של הדאטה מאשר לעבוד עם הדאטה המקורי. בכדי לקבל ייצוג לטנטי איכותי, ניתן לאמן אותו באמצעות מנגנון הנקרא decoder, הבוחן את יכולת השחזור של הדאטה מהייצוג הלטנטי שלו. 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: פיסת מידע המיוצגת על ידי וקטור המאפיינים $x \in \mathbb{R}^n$ עובר דרך encoder, שמטרתו להפיק מהדאטה את הייצוג הלטנטי שלו $e(x) \in \mathbb{R}^m$, כאשר $m < n$. לאחר מכן התוצאה מוכנסת ל-decoder בכדי לשחזר את הדאטה המקורי, ולבסוף מתקבל וקטור $d(e(x)) \in \mathbb{R}^n$. אם מתקיים השיוויון $x = d(e(x))$ אז למעשה לא אבד 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 הדאטה המקורי מהייצוג שלו בממד הנמוך בדיוק טוב מספיק, כנראה שהייצוג הלטנטי הצליח להפיק את המאפיינים המשמעותיים של הדאטה המקורי.



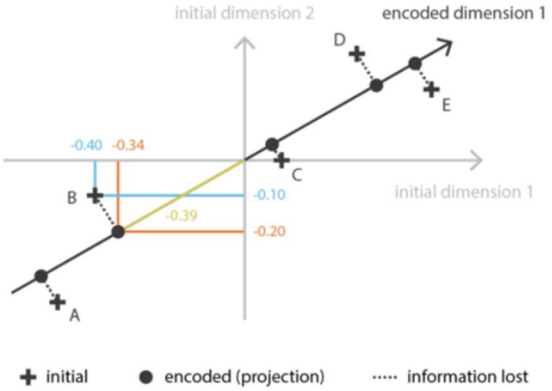
איור 7.1 ארכיטקטורת encoder ו-decoder.

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, ומביאים 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 $m < n$, כך שהמאפיינים של הייצוג הלטנטי של הדוגמאות המקוריות יהיו אורתוגונליים. תהליך זה נקרא גם feature decorrelation, והמטרה שלו היא למזער את המרחק האוקלידי בין הדאטה המקורי לדאטה המשוחזר, בצורה לינארית גם כן, מהייצוג החדש במרחב ה-m- ממדי.



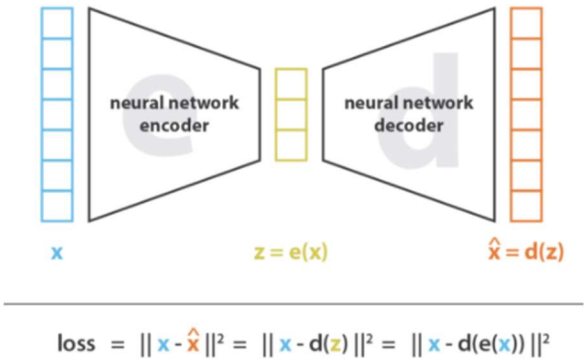
Point	Initial	Encoded	Decoded
A	(-0.50, -0.40)	-0.63	(-0.54, -0.33)
B	(-0.40, -0.10)	-0.39	(-0.34, -0.20)
C	(0.10, 0.00)	0.09	(0.07 0.04)
D	(0.30, 0.30)	0.41	(0.35, 0.21)
E	(0.50, 0.20)	0.53	(0.46, 0.27)

איור 7.2 דוגמה להורדת ממד בשיטת PCA.

במונחים של encoder-decoder, ניתן להראות כי אלגוריתם PCA מחפש את הזוג של encoder-decoder שמקיימים שני תנאים: א. ה-encoder מבצע טרנספורמציה לינארית על הדאטה כך שהמאפיינים החדשים (בממד נמוך) של הדאטה יהיו אורתוגונליים. ב. ה-decoder הלינארי המתאים יביא לשגיאה מינימלית במונחים של מרחק אוקלידי בין הדאטה 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:

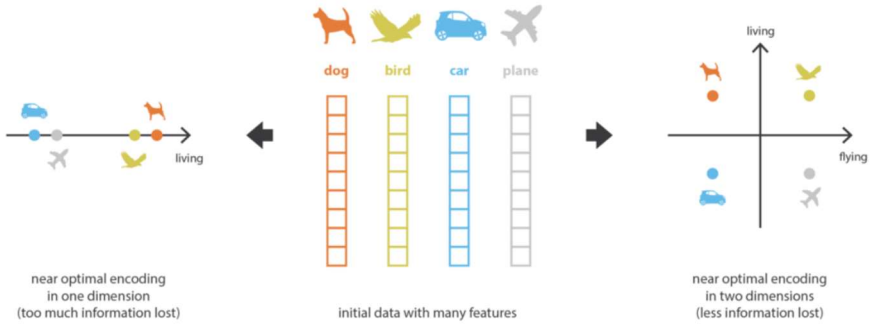


איור 7.3 Autoencoder – שימוש ברשתות נוירונים עבור הורדת הממד והשחזור.

באופן הזה, הארכיטקטורה יוצרת לדאטה צוואר בקבוק מידעי (information bottleneck), 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 הפעלה (activation functions)

לא לינאריות, ניתן לראות כי ה-autoencoder יחפש טרנספורמציה לינארית של הדאטה, שבאמצעותו ניתן לשחזרו באופן לינארי גם כן. בדומה ל-PCA, גם רשת כזו תחפש להוריד את הממד באמצעות טרנספורמציות לינאריות של המאפיינים המקוריים אך הייצוג בממד נמוך המופק על ידה לא יהיה בהכרח זהה לזה של PCA, כיוון שלהבדיל מ-PCA המאפיינים החדשים (לאחר הורדת ממד) עשויים לצאת לא אורתוגונליים (-קורלציה שונה מ-0).

כעת נניח שהרשתות של ה-encoder וה-decoder הן רשתות עמוקות ומשתמשות בפונקציות הפעלה לא לינאריות. במקרה כזה, ככל שהארכיטקטורה של הרשתות מורכבת יותר, כך רשת ה-encoder יכולה להוריד יותר ממדים תוך יכולת באמצעות ה-decoder לבצע שחזור ללא כל 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 כל איבוד מידע. עם זאת, הפחתת ממד דרסטית שכזו עלולה לגרום לדאטה המשוחזר לאבד את המבנה שלו. לכן יש חשיבות גדולה בבחירת מספר הממדים של המרחב הלטנטי, כך שמצד אחד אכן יתבצע ניפוי של מאפיינים פחות משמעותיים ומצד שני המידע עדיין יהיה בעל משמעות למשימות downstream שונות. כדי להמחיש את המתואר לעיל, ניקח לדוגמה מערכת שמקבלת תמונות של כלב, ציפור, מכונית ומטוס ומנסה למצוא את הפרמטרים העיקריים המבחינים ביניהם:



איור 7.4 דוגמה לשימוש ב-Autoencoder.

לפריטים אלו יש הרבה מאפיינים, וקשה לבנות מודל שמבחין ביניהם על סמך כל המאפיינים. רשת נוירונים מורכבת מספיק מאפשרת לבנות ייצוג 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 הסמנטי של הדוגמאות ולא באמת ניתן להבין את ההפרדה ביניהן. לעומת זאת ניתן להוריד את הממד של תמונות אלו לדו-ממד ולהתייחס רק לפרמטרים "חי" ו"עף", וכך לקבל הבחנה יותר ברורה בין הדוגמאות. כמובן שהפרדה זו היא הרבה יותר פשוטה מאשר הסתכלות על כל הפרמטרים (-הפיקסלים) של הדוגמאות. דוגמה זו מראה את החשיבות שיש בבחירת הממדים של ה-encoder.

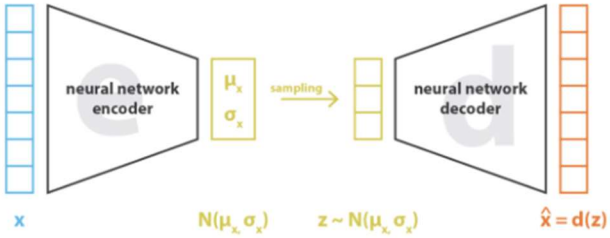
7.1.3 Variational AutoEncoders (VAE)

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 עליו אומן. AE מאומן לייצג את הדאטה בממד נמוך, שלוקח בחשבון את המאפיינים העיקריים, ולאחר מכן משחזר את התוצאה לממד המקורי. אולם, מנגנון זה אינו מאפשר להשפיע על האופן בו הדאטה מיוצג במרחב הלטנטי. אם יוגרל וקטור כלשהו מהמרחב הלטנטי – קרוב לוודאי שהוא לא יהווה ייצוג שדומה לדאטה המקורי. אם היינו מכניסים אותו ל-decoder, סביר להניח שהתוצאה לא תהיה דומה בכלל לדאטה המקורי. למשל אם AE אומן על אוסף של תמונות של כלבים ודוגמים באקראי 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 AutoEncoder (VAE). בשונה מ-AE שלוקח דאטה ורק בונה לו ייצוג מממד נמוך, VAE קובע התפלגות פריורית למרחב הלטנטי z – למשל התפלגות נורמלית עם תוחלת 0 ומטריצת covariance $\mathbb{I}$. בהינתן התפלגות זו, רשת ה-encoder מאומנת לקבל דאטה x ולהוציא פרמטרים של התפלגות פוסטריורית $z|x$, מתוך מטרה למזער כמה שניתן את המרחק 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 שתייצג את ערך הפיקסל בתמונה המשוחזרת.

באופן הזה, הלמידה דואגת לא רק להורדת הממד של הדאטה, 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 שמשמש ליצירת דאטה חדש לאחר תהליך האימון, או במילים אחרות, הוא הופך את המערכת למודל גנרטיבי.



איור 7.5 ארכיטקטורה של VAE.

לאחר שהוצג המבנה הכללי של VAE, ניתן לתאר את תהליך האימון, ולשם כך נפריד בשלב זה בין שני החלקים של ה-VAE. ה-encoder מאמן רשת שמקבלת דוגמאות מסט האימון, ומנסה להפיק מהן פרמטרים של התפלגות $z|x$ הקרובים כמה שניתן לפרמטרים של ההתפלגות הפריורית z, שכאמור נקבעה מראש. מההתפלגות הנלמדת הזו דוגמים וקטורים לטנטיים חדשים ומעבירים אותם ל-decoder. ה-decoder מבצע את הפעולה ההפוכה – לוקח וקטור שנדגם מהמרחב הלטנטי $z|x$, ומייצר באמצעותו דוגמה חדשה שאמורה להיות דומה 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 $p(X;\theta)$, כלומר למקסם את הנראות המרבית של סט 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 והפלט של ה-encoder הוא בעל התפלגות $q(z|x;\lambda)$ (מה ההסתברות לקבל את z בהינתן x בכניסה), וננסה לייצג את ההתפלגות הזו בעזרת רשת נוירונים עם סט פרמטרים λ. 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 $p(z|x), q(z)$, שנקרא 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 Gradient Descent (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 חוק 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:

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

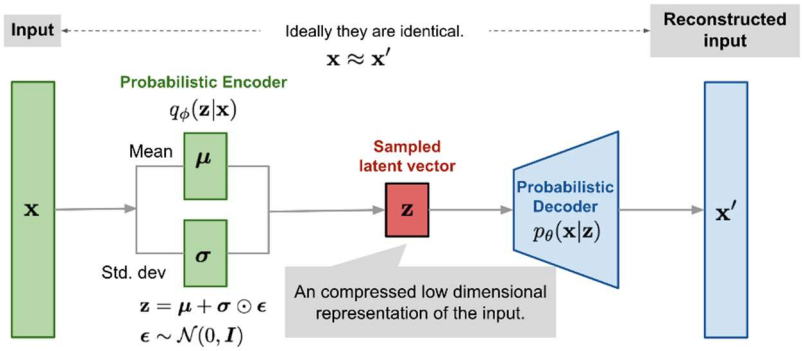
כדי לחשב את התוחלת ניתן פשוט לדגום דוגמאות מההתפלגות $z|x\sim N\big(\mu_\theta(x),\sigma_\theta(x)\big)$ ולקבל:

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

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 של VAE לבין המוצא שלו, וגם למזעור המרחק בין ההתפלגות הפריורית של z לבין ההתפלגות $z|x$ שבמוצא ה-encoder.



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

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

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 אלו, מעבירים אותו ב-decoder ומקבלים את $\mu_\theta, \sigma_\theta$. לאחר התהליך ניתן להציב את הפרמטרים המתקבלים ב-ELBO ולחשב את ערך פונקציית המחיר. ניתן לשים לב שה-ELBO מורכב משני איברים – האיבר הראשון משערך את הדמיון 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 ערך הדגימה של 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 התפשטות הערך ברשת.

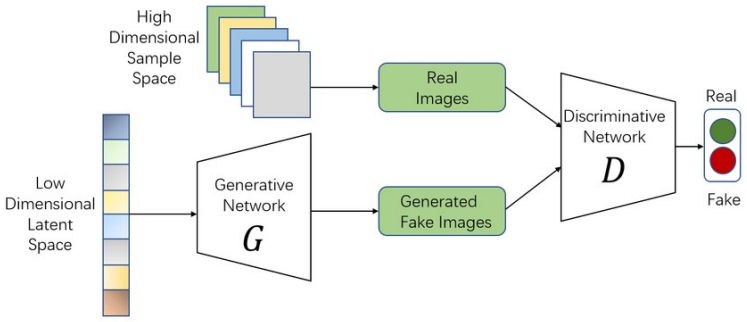
7.2 Generative Adversarial Networks (GANs)

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 (על ידי מציאת הפרמטרים הממקסמים את הנראות המירבית של סט האימון), אלא יוצרים דאטה באופן אחר. הרעיון הוא לאמן שתי רשתות במקביל – רשת אחת שלומדת לייצר דוגמאות, ורשת שניה שלומדת להבחין בין דוגמה אמיתית מסט האימון לבין תמונה סינתטית 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 סינתטי 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 כי קיימים מאות רבות של וריאנטים שונים של GAN שהוצעו מאז, ועדיין תחום זה פעיל מאוד מבחינה מחקרית.

כאמור, GAN מבוסס על שני אלמנטים מרכזיים – רשת שיוצרת דאטה (generator) ורשת שמכריעה האם הדאטה הזה סינתטי או אמיתי (discriminator), כאשר האימון נעשה על שתי הרשתות יחד. ה-discriminator מקבל כקלט הן את הדוגמאות האמיתיות והן את הפלט של ה-generator, כדי ללמוד להבחין בין דאטה אמיתי לבין דאטה סינתטי. ה-generator מייצר דוגמאות ומקבל פידבק מה-discriminator וכך לומד לייצר דוגמאות שנראות אמיתיות. 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 שהפלט שלו הוא ההסתברות שהקלט הינו דוגמה אמיתית, ונסמן ב-$D(x)$ את ההסתברות הזו. כדי לאמן את ה-discriminator נרצה להשיג שני דברים: א. למקסם את $D(x)$ עבור x מסט האימון, כלומר, לטעות כמה שפחות בזיהוי דאטה אמיתי. ב. למזער את $D(x)$ עבור דאטה סינתטי, כלומר, לזהות נכון כמה שיותר דוגמאות סינתטיות שיוצרו על ידי ה-generator. באופן דומה נרצה לאמן את ה-generator כך שהדגימות שהוא מייצר תהיינה כמה שיותר דומות לדוגמאות אמיתיות, כלומר ה-generator מעוניין לגרום ל-discriminator להוציא ערכים כמה שיותר גבוהים עבור הדאטה הסינתטי שהוא מייצר. בשביל לאמן יחד את שני חלקי המודל, נבנה פונקציית מחיר בעלת שני איברים, באופן הבא:

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

נסביר את הביטוי המתקבל: ה-discriminator מעוניין למקסם את פונקציית המחיר, כך שהערך של $D(x)$ 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 יחשוב ש-$G(z)$ הוא דאטה אמיתי.

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 הנראות אותנטיות. האימון של ה-generator נעשה באמצעות Gradient Descent (מזעור פונקציית המחיר ביחס ל-θ_g), והאימון של ה-discriminator נעשה באמצעות Gradient Ascent (מקסום פונקציית המחיר ביחס ל-ϕ_D). האימון מתבצע במשך מספר מסוים של Epochs, כאשר כאמור מאמנים לסירוגין את G ו-D. בפועל דוגמים mini-batch בגודל m מסט האימון $(x_1, \dots, x_m)$ ו-m דגימות של רעש $(z_1, \dots, z_m)$, ומכניסים את הקלט ל-G. הגרדיאנט של פונקציית המחיר לפי הפרמטרים של 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 מסט 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.

הערכים האופטימליים של G ו-D:

כזכור, פונקציית המחיר הינה:

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

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

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

כדי להביא את הביטוי הזה למקסימום, נרצה למקסם את האינטגרד עבור 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 את הביטוי האחרון ונשווה ל-0 בכדי למצוא את הערך האופטימלי של $D(x)$ עבור x נתון:

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

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

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

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

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 המתקבל הינו המרחק בין ההתפלגויות p_r ו-p_g, והוא נקרא 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 $\mathcal{D}_{JS}$ בין p_r לבין p_g עד כדי קבוע, ובאופן מפורש:

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

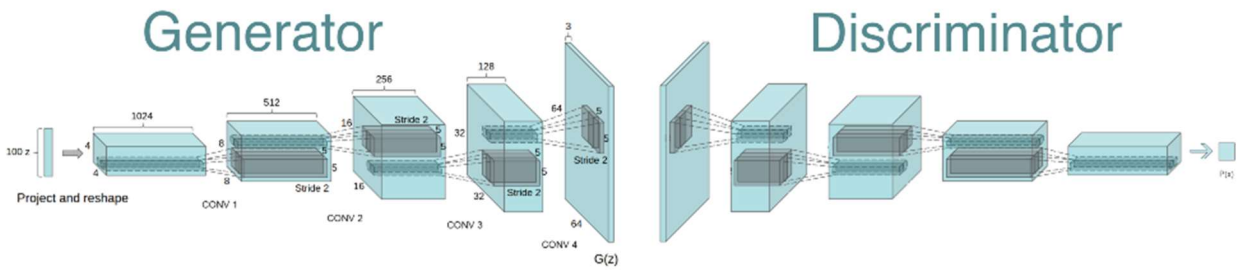
כאשר G אופטימלי ומתקיים $p_g(x)=p_r(x)$, אז המרחק בין ההתפלגויות שווה 0, כלומר $\mathcal{D}_{JS}\big(p_r,p_g\big)=0$, ולכן מתקבל:

$$V\big(G_{opt},D_{opt}\big)=-\log 4$$

יש משמעות גדולה לביטוי שהתקבל – ככל שנצליח למזער יותר את $\mathcal{D}_{JS}\big(p_r,p_g\big)$, כך נצליח לקבל GAN יותר טוב.

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 ולבנות בעזרתן 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 סינתטית. DCGAN הומצא ב-2015 ומאז פותחו רשתות שמייצרות תמונות יותר איכותיות הן מבחינת הרזולוציה והן מבחינת הדמיון שלהן לתמונות אמיתיות, אך החשיבות של המאמר נעוצה בשימוש ברשתות קונבולוציה עבור GAN שמיועד לדומיין של תמונות.



איור 7.8 ארכיטקטורת DCGAN.

7.2.3 Conditional GAN (cGAN)

לעיתים מודל גנרטיבי נדרש לייצר דוגמה בעלת מאפיין ספציפי ולא רק דוגמה שנראית אותנטית. למשל, עבור אוסף 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 (או בקיצור cGAN), 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. סט האימון במקרה זה בנויה מזוגות של תמונות והסקיצות שלהן.

שיטת Pix2Pix משתמשת בארכיטקטורה של GAN אך במקום לדגום את וקטור z מהתפלגות כלשהי, 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 מסט 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 סינתטית. 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.

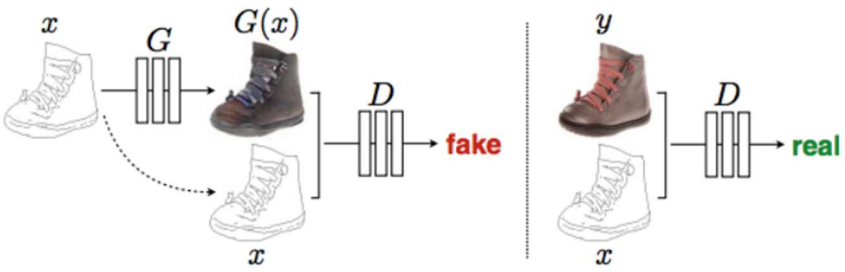
כעת נסמן תמונה אמיתית השייכת לסקיצה ב-y, ונרשום את פונקציית המחיר כשני חלקים נפרדים – cross entropy רגיל של GAN ומרחק אוקלידי L_1 בין תמונת המקור לבין הפלט:

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

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

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

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



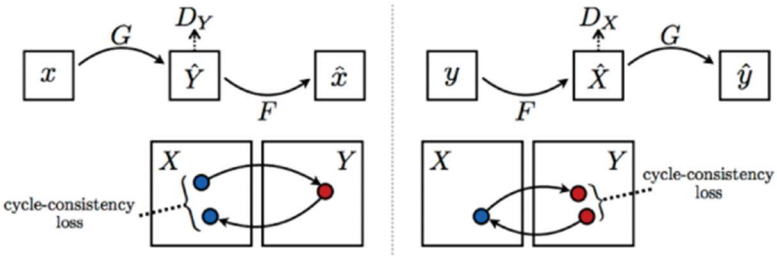
איור 7.9 ארכיטקטורת Pix2Pix - Image-to-Image Translation.

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 מתחומים 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-generator השני 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:

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

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

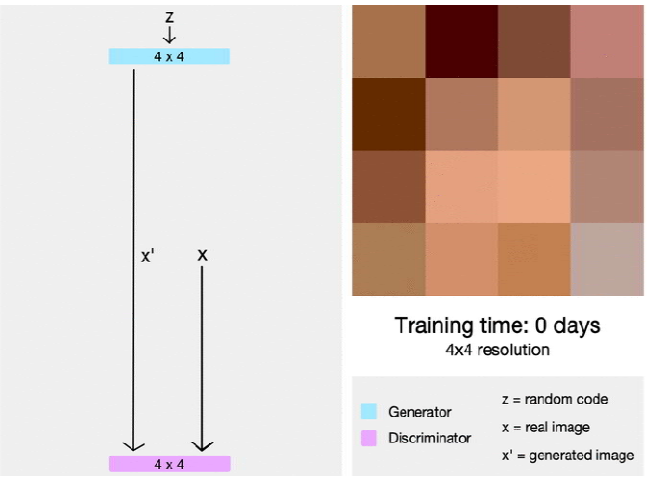


איור 7.10 ארכיטקטורת CycleGAN.

7.2.6 Progressively Growing GAN (ProGAN)

כאמור לעיל, עבור דומיין של תמונות, הגיוני להשתמש ברשתות קונבולוציה עבור יצירת תמונות חדשות, וזה הרעיון הבסיסי שמאחורי DCGAN. למרות היכולת המרשימה של DCGAN ביצירה של תמונה באיכות גבוהה, יכולת זאת מוגבלת לתמונות בגודל מסוים. ככל שהרזולוציה של תמונה גבוהה יותר, כך יותר קל להבחין אם תמונה זו אמיתית או נוצרה על ידי רשת גנרטיבית. בעוד ש-DCGAN מצליח ליצור תמונות שנראות אותנטיות בגדלים של $32 \times 32, 64 \times 64$, ואפילו 128×128, הוא מתקשה ביצירת תמונות ברזולוציות גבוהות יותר, כמו למשל רזולוציה של 256×256. ProGAN בא לתת מענה לכך, והוא היה ה-GAN הראשון שפרץ את מחסום הרזולוציה והצליח ליצור תמונות איכותיות מאוד (במאמר המקורי של ProGAN – עד רזולוציה של 1024×1024) בלי שיהיה ניתן להבחין שתמונות אלה סינתטיות. אמנם עוד לפני ProGAN היו GANs שהצליחו ליצור תמונה בעלת רזולוציה גבוהה מתמונה אחרת ברזולוציה גבוהה (pix2pix), אך זו משימה אחרת, מכיוון שבשבילה צריך רק ללמוד לשנות תכונות של תמונת קלט, ולא לייצר תמונה חדשה לגמרי מאפס.

הרעיון העיקרי מאחורי ProGAN, שהוצע ב-2017 על ידי חוקרים מחברת Nvidia, הינו לייצר תמונות ברזולוציה הולכת וגדלה בצורה הדרגתית. כלומר, במקום לנסות לאמן את כל השכבות של ה-generator בבת אחת, כפי שנעשה בכל ה-GANs לפני כן, ניתן לאמן אותו לייצר תמונות ברזולוציה משתנה – בהתחלה הוא מתאמן לייצר תמונות ברזולוציות מאוד נמוכה (4×4), לאחר מכן המשיכו ליצירת תמונות ברזולוציה 8×8, אחר כך 16×16, וכך הלאה עד יצירה של תמונה ברזולוציה של 1024×1024.



איור 7.11 ארכיטקטורת ProGAN.

כדי לאמן GAN לייצר תמונות בגודל 4×4, התמונות מסט האימון הוקטנו לגודל זה (down-sampling). אחרי שה-GAN לומד לייצר תמונות בגודל 4×4, מוסיפים לו עוד שכבה המאפשרת להכפיל את גודל התמונות המיוצרות, קרי ליצור תמונות בגודל 8×8. יש לציין שהאימון של הרשת עם השכבה הנוספת מתחיל עם המשקלים שאומנו קודם לכן, אך לא "מקפיאים" אותם, כלומר הם מעודכנים גם כן תוך כדי אימון הרשת בשביל ליצור תמונה ברזולוציה כפולה. הגדלה הדרגתית של הרזולוציה מאלצת את הרשתות להתמקד תחילה בפרטים "הגסים" של התמונה (דפוסים בתמונה מטושטשת מאוד). לאחר מכן הרשת "לומדת" לבצע up-sampling (להכפיל את הרזולוציה) של התמונות המטושטשות האלה. תהליך זה משפר את איכות התמונה הסופית כיוון שבאופן זה הסבירות שהרשת תלמד דפוסים שגויים קטנה משמעותית.

7.2.7 StyleGAN

StyleGAN, שיצא בשלהי שנת 2018, מציע גרסה משודרגת של ProGAN, עם דגש על רשת ה-generator. מחברי המאמר שמו לב כי היתרון הפוטנציאלי של שכבות ProGAN המייצרות תמונה בצורה הדרגתית נובע מיכולתן לשלוט בתכונות (מאפיינים) ויזואליות שונות של התמונה, אם משתמשים בהן כראוי. ככל שהשכבה והרזולוציה נמוכה יותר, כך התכונות שהיא משפיעה עליהן גסות יותר.

למעשה, StyleGAN הינו ה-GAN הראשון שנותן יכולת לשלוט במאפיינים ויזואליים (אומנם לא בצורה מלאה) של התמונה הנוצרת. מחברי StyleGAN חילקו את התכונות הוויזואליות של תמונה ל-3 סוגים:

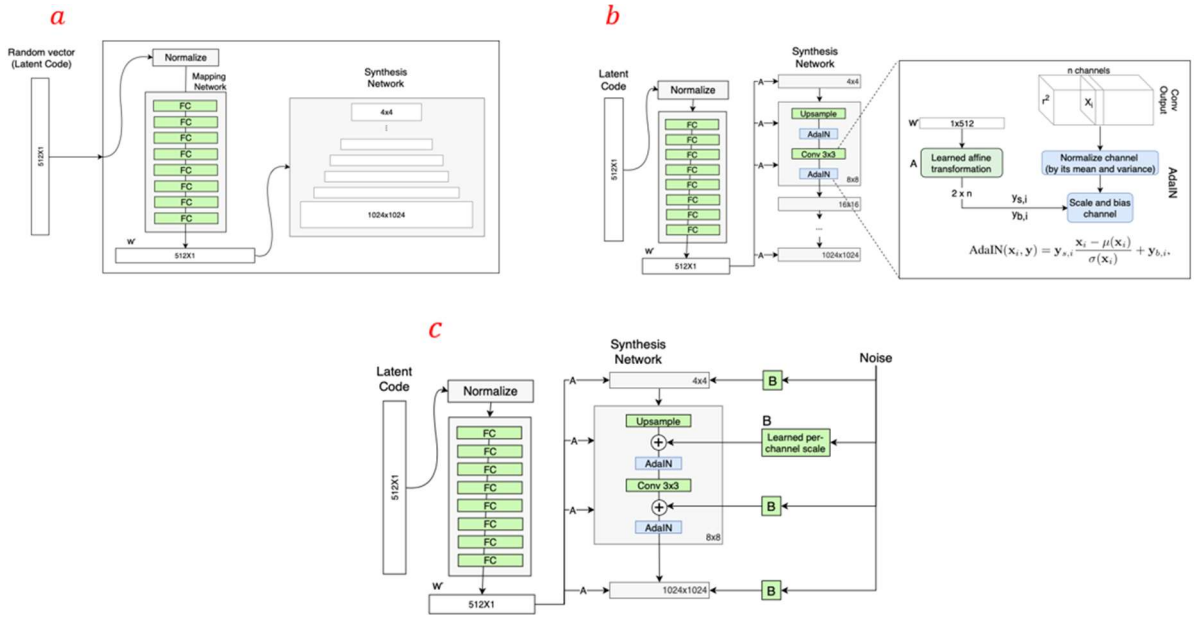
- **גס:** משפיע על תנוחה, סגנון שיער כללי, צורת פנים וכו'.
- **אמצעית:** משפיעה על תווי פנים עדינים יותר, סגנון שיער, עיניים פקוחות/עצומות וכדו'.
- **רזולוציה דקה**: משפיעה על צבע (עיניים/שיער/עור) ועל שאר תכונות המיקרו של תמונה.

כדי להעניק ל-StyleGAN את היכולות האלו, נדרשים מספר שינויים ביחס לארכיטקטורה של ProGAN (נתאר רק את שלושת השינויים החשובים ביותר כאן):

- **הוספת רשת מיפוי:** מטרת רשת המיפוי היא קידוד וקטור הקלט לווקטור ביניים w (הנקרא וקטור סגנון) אשר האיברים השונים שלו שולטים בתכונות ויזואליות שונות של התמונה הנוצרת. זהו תהליך לא טריוויאלי מכיוון שהיכולת של הרשת לשלוט בתכונות ויזואליות באמצעות וקטור הקלט הינה מוגבלת. הסיבה לכך טמונה בעובדה שווקטור הקלט נאלץ "לעקוב אחר צפיפות ההסתברות של סט האימון" שגורם לתופעה הנקראת feature entanglement (FE) (-ערבוב מאפיינים). FE בין תכונות צבע השיער והמגדר יכול להופיע אם למשל בסט האימון יש מגמה כללית של גברים עם שיער קצר ונשים בעלות שיער ארוך. במקרה זה הרשת תלמד שגברים יכולים להיות בעלי שיער קצר בלבד ולהיפך אצל נשים. כתוצאה מכך, אם "נשחק" עם רכיבי וקטור הקלט כדי לייצר תמונה של גבר בעל שיער ארוך, בסופו של דבר מגדרו ישתנה גם כן ונקבל תמונה של אישה.
  רשת המיפוי שהתווספה לארכיטקטורה הופכת את וקטור הקלט לווקטור ביניים w שאינו צריך לעקוב אחר התפלגות של סט האימון, וכך יש פחות ערבוב המאפיינים. במילים אחרות, רשת זו מאפשרת את היכולת לשלוט במאפיינים ויזואליים של התמונה הנוצרת באמצעות שינוי רכיביו של וקטור w. רשת המיפוי מורכבת משמונה שכבות FC וגודל הפלט שלה זהה לגודל הקלט.
- **החלפת BN ב-AdalN:** רשתות הקונבולוציה של ה-generator, שנועדו ליצירת תמונות ברזולוציות שונות משתמשות במנגנון שנקרא AdaIN (במקום Batch Normalization). בשונה מ-BN, הפרמטרים של

הממוצע ושל השונות בגישת AdaIN נלמדים מווקטור הסגנון w (הם בעצם טרנספורמציה לינארית של w עם משקלים נלמדים). להבדיל מ-AdaIN, במנגנון BN סטנדרטי פרמטרים אלו נלמדים כמו המשקלים האחרים ולא תלויים במוצא של שכבה כלשהי.

- **ויתור על אקראיות של וקטור קלט:** ב-StyleGAN וקטור הקלט אינו וקטור המוגרל מהתפלגות גאוסית אלא וקטור דטרמיניסטי עם רכיבים נלמדים. וקטורי הרעש מתווספים ישירות לפלטים של ערוצי קונבולוציה ברשתות ה-generator כאשר העוצמה שלהן נלמדת לכל ערוץ בנפרד. שימוש בוקטור קלט דטרמיניסטי במקום בוקטור אקראי מקל ככל הנראה על הפרדת המאפיינים על ידי רשת המיפוי (יותר קל לעשות זאת על וקטור קבוע מאשר להתאים את משקלי רשת המיפוי לווקטורי כניסה אקראיים).



איור 7.12 השינויים העיקריים בארכיטקטורת StyleGAN. a) הוספת רשת מיפוי. b) שימוש ב-AdalN. c) שימוש בקלט דטרמיניסטי.

יש עוד כמה שינויים יותר מינוריים ב-StyleGAN יחסית ל-ProGAN, כמו שינוי של היפר פרמטרים של הרשתות, פונקציית מחיר וכו'. התוצאות הן לא פחות ממרשימות – StyleGAN יוצר תמונות שנראות ממש אמיתיות ובנוסף מקנה יכולת לשלוט בחלק מהתכונות החזותיות של התמונות.

איור 7.13 תמונות שיוצרו באמצעות StyleGAN.

7.2.8 Wasserstein GAN

אחד סוגי ה-GAN החשובים ביותר הינו Wasserstein GAN, והוא נוגע בבעיה שיש בפונקציית המחיר בה משתמשים הרבה וריאנטים של GAN-ים. כאמור, תהליך הלמידה של הרשת המייצרת דאטה – ה-generator – נעשה באמצעות משוב המתקבל מה-discriminator. בעוד שה-discriminator מאומן להבחין בין דאטה אמיתי לדאטה סינתטי 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 ה-generator עוד לא מאומן, הדוגמאות הסינתטיות שהוא מייצר אינן דומות כלל 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 ה"ידע" העובר ל-generator בעזרת הגרדיאנט של פונקציית המחיר (loss) שלו, התלוי בערכים אותם מוציא ה-discriminator. כדי להבין מדוע תהליך העברת המידע באופן הזה בעייתי, 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 האמיתי נמצאים ב"משטח בעל ממד נמוך" בתוך מרחב בעל ממד גבוה של הדאטה המקורי. באופן פורמלי יותר, משטח זה נקרא יריעה (manifold), וההשערה שתוארה מעלה מהווה הנחת יסוד בתחום הנקרא למידת יריעות (manifold learning). מכיוון שמדובר במשטחים בעלי ממד נמוך בתוך מרחב בעל ממד גבוה, קיימת סבירות גבוהה שלא יהיה שום חיתוך בין המשטח בו "חי" הדאטה האמיתי לבין זה של הדאטה הסינתטי (לכל הפחות בתחילת תהליך האימון של ה-GAN), ויתרה מכך, המרחק בין משטחים אלה עשוי להיות די גדול. מכך נובע שה-discriminator D עשוי ללמוד להבחין בין הדאטה האמיתי לסינתטי בקלות, כיוון שבמרחב מממד גבוה יש מרחק גדול בין יריעה אמיתית לבין היריעה של הדאטה הסינתטי. בנוסף, D כנראה ייתן לדוגמאות סינתטיות ציונים (score) ממש קרובים לאפס כי אכן קל מאוד למצוא "משטח הפרדה" בין שתי היריעות – זה של הדוגמאות האמיתיות וזה של הסינתטיות, 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 היריעות שתואר מעלה) לדוגמאות המיוצרות על ידי ה-generator, ה-generator פשוט לא יצליח לשפר את איכות התמונות שהוא מייצר. 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" מה-discriminator ל-generator.

יש מספר לא קטן של שיטות הבאות לשפר את תהליך האימון של GAN, אך אף אחת מהן אינה מטפלת בבעיה זו באמצעות שינוי של פונקציית המחיר. השיטות הבולטות הן:

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

כפי שהוסבר, הבעיה של המרחק בין היריעות 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 הסינתטי 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 ו-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 ו-p_g. זו למעשה הבעיה המהותית ביותר עם פונקציית המחיר המקורית של ה-generator, שעדכון המשקלים לא משפיע כמעט על $\mathcal{D}_{JS}$, כיוון שמראש ההתפלגויות p_r ו-p_g רחוקות אחת מהשנייה.

Wasserstein GAN בא להתמודד עם בעיה זו, ולשם כך הוא משתמש בפונקציית מחיר אחרת, בה עדכון המשקלים של ה-generator משתקף גם במרחק בין ההתפלגויות p_r ו-p_g. פונקציית המחיר החדשה מבוססת על מרחק הנקרא Earth Mover (EM), המהווה מקרה פרטי של מרחק וסרשטיין המסומן ב-$\mathcal{D}_W$. 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 0.1_{kg} כל אחת המפוזרות באופן הבא: 6 משקולות (0.6_{kg}) בנקודה $x = 0$, ו-4 משקולות (0.4_{kg}) בנקודה $x = 1$. כעת נרצה להזיז את המשקולות כך שתהיינה מפוזרות באופן הבא: בנקודה $x = 4$ יהיה משקל של 0.3_{kg}, בנקודה $x = 5$ יהיה משקל של 0.5_{kg}, ושאר המשקולות (0.2_{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:

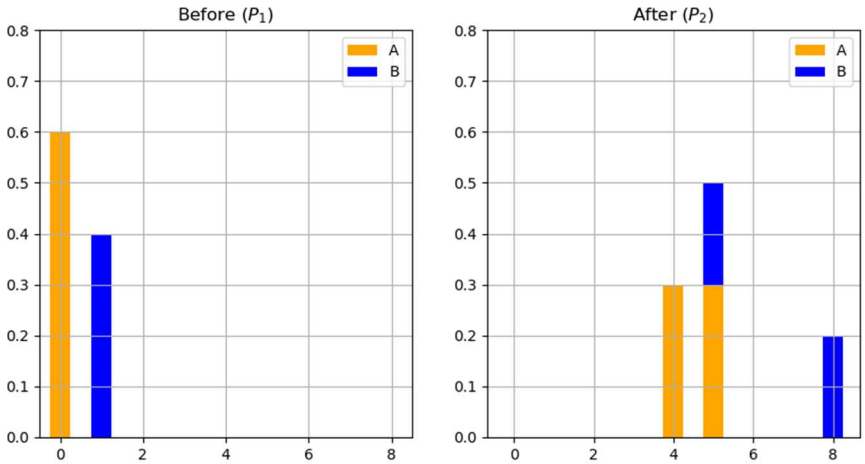
0.3_{kg} מועברים מ-$x = 0$ ל-$x = 4$, כאשר המאמץ הנדרש לכך הינו $(4-0)\cdot 0.3 = 1.2$.

0.3_{kg} מועברים מ-$x = 0$ ל-$x = 5$, כאשר המאמץ הנדרש לכך הינו $(5-0)\cdot 0.3 = 1.5$.

0.2_{kg} מועברים מ-$x = 1$ ל-$x = 5$, כאשר המאמץ הנדרש לכך הינו $(5-1)\cdot 0.2 = 0.8$.

0.2_{kg} מועברים מ-$x = 1$ ל-$x = 8$, כאשר המאמץ הנדרש לכך הינו $(8-1)\cdot 0.2 = 1.4$.

סך המאמץ המינימלי שווה במקרה הזה ל: $1.2 + 1.5 + 0.8 + 1.4 = 4.9$.



איור 7.14 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 אדמה 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 GAN. נציין כי מרחק $\mathcal{D}_W$ בין מידות ההסתברות מתחשב בתכונות של הקבוצות עליהן מידות אלו מוגדרות בצורה מפורשת, על ידי התחשבות במרחקים בין בקבוצות אלו. תכונה זו היא למעשה בדיוק מה שצריך בשביל למדוד את המרחק בין ההתפלגות האמיתית של דאטה p_r לבין התפלגות של הדאטה הסינתטי p_g. מרחק EM ידע לשערך בצורה טובה את המרחק בין היריעות שבהן "חיות" שתי ההתפלגויות, כלומר אם מזיזים את היריעה של הדאטה הסינתטי, נוכל לדעת בעזרת מרחק EM עד כמה השתנה המרחק בין היריעות. נציין שזה לא קורה כאשר משתמשים בפונקציית 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 K-ליפשיץ רציפה (כלומר, פונקציה רציפה עם קצב השתנות החסום על ידי K). כעת נניח ש-$f(w)$ הינה פונקציית K-ליפשיץ רציפה 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 לסינתטית, אלא 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.

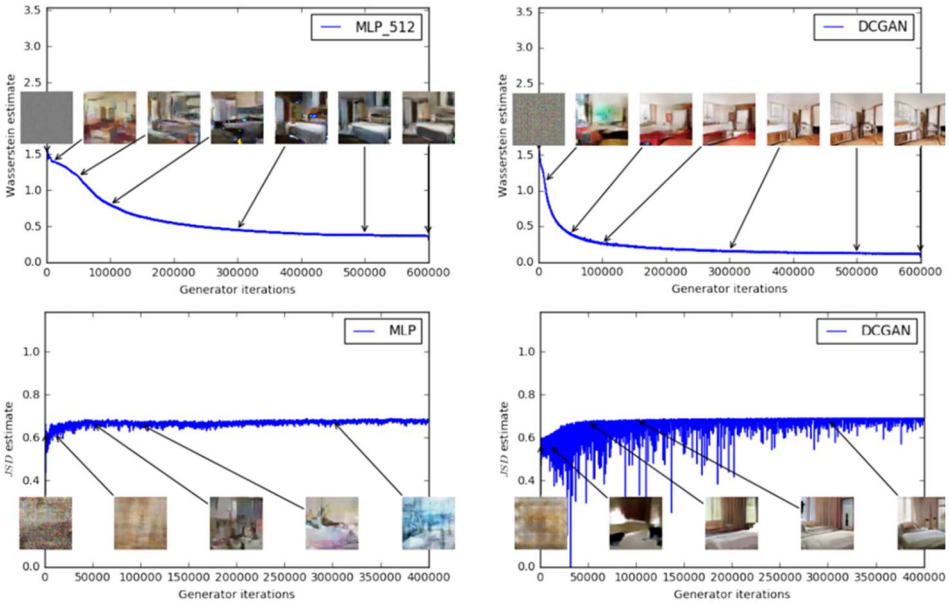
כאמור, תנאי הכרחי לשימוש במרחק זה בפונקציית המחיר הינו שהפונקציה תהיה K-ליפשיץ רציפה. מסתבר שקיום תנאי זה אינו משימה קלה כלל. כדי להבטיח את קיומו, המאמר המקורי הציע לבצע קטימה של משקלי ה-discriminator לטווח סופי מסוים, נניח $[-0.01,0.01]$. ניתן להראות כי קטימה זו מבטיחה את ש-f_w תהיה K-ליפשיץ רציפה. 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 שיאט 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 ה-generator לבין ערך של פונקציית לוס תהיה הרבה יותר בולטת מאשר ב-GAN רגיל בעל אותה ארכיטקטורה. ניתן להמחיש זאת היטב באמצעות גרפים הבוחנים את היחס בין D_W לבין D_{JS}:



איור 7.15 שערוך מרחק $\mathcal{D}_W$ בין p_r ל-p_g כפונקציה של מספר האיטרציות (בגרפים העליונים), לעומת שערוך מרחק $\mathcal{D}_{JS}$ בין p_r ל-p_g כפונקציה של מספר האיטרציות (בגרפים התחתונים).

ניתן לראות בבירור כי ככל שאיכות התמונות שה-generator מייצר עולה, כך $\mathcal{D}_W$ הולך וקטן, ואילו מרחק $\mathcal{D}_{JS}$ לא מראה שום סימן של ירידה. הצלחה זו נובעת מהשינוי בפונקציית המחיר, שגרם לאימון להיות יותר יעיל, והביא לכך שהדוגמאות הסינתטיות תהיינה דומות הרבה יותר לדאטה המקורי.

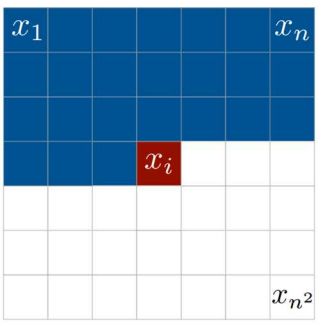
נקודה נוספת שיכולה להסביר את ההצלחה היחסית של השימוש ב-$\mathcal{D}_W$ נובעת מכך שמטריקת וסרשטיין חלשה יחסית למטריקת JS, וננסה להבהיר נקודה זו.

באופן אידיאלי, כאשר אנו מאמנים מודל היינו רוצים להיות בטוחים שאם ננהג בצורה נאותה ובכל צעד נעדכן המודל בדיוק על פי הוראות הגרדיאנט, נסיים את האימון בנקודה כמעט אופטימלית. אולם בפועל זה לא תמיד כך, כיוון שישנן בעיות שעבורן מטריקות מסוימות יגיעו לנקודה זו ואחרות לא. ניקח לדוגמה שני אנשים שעומדים על סף תהום ורוצים להגיע לעמק. האחד מודד את הגובה ומתקדם על פיו, ולכן הוא יגיע למטה בקלות יחסית. האחר מתעניין במיקומו על ציר צפון דרום, ולכן הוא עשוי להיתקל בקשיים במהלך הירידה, וגם אם הוא אכן יגיע למטה, זה בהכרח יהיה בתהליך איטי יותר. באופן דומה, כאשר לוקחים זוג מטריקות, באופן פורמלי ניתן להגדיר שאם התכנסות של סדרת התפלגויות תחת מטריקה אחת גוררת התכנסות של הסדרה תחת מטריקה אחרת, אזי המטריקה הראשונה חזקה יותר מהמטריקה השנייה. העובדה ש-$\mathcal{D}_W$ חלש יותר מ-$\mathcal{D}_{JS}$ בעצם אומרת שיתכן ויש בעיות שעבורן תתקבל תוצאה אופטימלית עבור $\mathcal{D}_W$ אך לא עבור $\mathcal{D}_{JS}$.

7.3 Auto-Regressive Generative Models

משפחה נוספת של מודלים גנרטיביים נקראת Auto-Regressive Generative Models, ובדומה ל-VAE גם מודלים אלו מוצאים התפלגות מפורשת של מרחב מסוים ובעזרת התפלגות זו מייצרים דאטה חדש. עם זאת, בעוד VAE מוצא קירוב להתפלגות של המרחב הלטנטי, שיטות AR מנסות לחשב במדויק התפלגות מסוימת, וממנה לדגום ולייצר דאטה חדש.

תמונה x בגודל $n \times n$ היא למעשה רצף של n^2 פיקסלים. כאשר רוצים ליצור תמונה, ניתן ליצור כל פעם כל פיקסל באופן כזה שהוא יהיה תלוי בכל הפיקסלים שלפניו.



איור 7.15 תמונה כרצף של פיקסלים.

כל פיקסל הוא בעל התפלגות מותנית:

$$p(x_i|x_1 \dots x_{i-1})$$

כאשר כל פיקסל מורכב משלושה צבעים (RGB), לכן ההסתברות המדויקת היא:

$$p(x_{i,R}|\mathrm{x}_{<i})p(x_{i,G}|\mathrm{x}_{<i}, x_{i,R})p(x_{i,B}|\mathrm{x}_{<i}, x_{i,R}, x_{i,G})$$

כל התמונה השלמה היא מכפלת ההסתברויות המותנות:

$$p(x) = \prod_{i=1}^{n^2} p(x_i) = \prod_{i=1}^{n^2} p(x_i|x_1 \dots x_{i-1})$$

הביטוי $p(x)$ הוא ההסתברות של דאטה מסוים לייצג תמונה אמיתית, לכן נרצה למקסם את הביטוי הזה כדי לקבל מודל שמייצג תמונות שנראות אותנטיות עד כמה שניתן.

7.3.1 PixelRNN

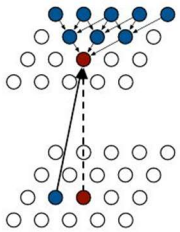
אפשרות אחת לחשב את $p(x)$ היא להשתמש ברכיבי זיכרון כמו LSTM עבור כל פיקסל. באופן טבעי היינו רוצים לקשר כל פיקסל לשכנים שלו:

$$\text{Hidden State } (i,j) = f\big(\text{Hidden State } (i-1,j), \text{Hidden State } (i,j-1)\big)$$

הבעיה בחישוב זה היא הזמן שלוקח לבצע אותו. כיוון שכל פיקסל דורש לדעת את הפיקסל שלפניו – לא ניתן לבצע אימון מקבילי לרכיבי ה-LSTM. כדי להתגבר על בעיה זו הוצעו כמה שיטות שנועדו לאפשר חישוב מקבילי.

<u>Row LSTM</u>

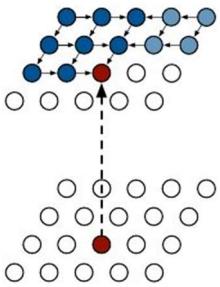
במקום להשתמש במצב החבוי של הפיקסל הקודם, ניתן להשתמש רק בשורה שמעל הפיקסל אותו רוצים לחשב. שורה זו בעצמה מחושבת לפני כן על ידי השורה שמעליה, ובכך למעשה לכל פיקסל יש receptive field של משולש. בשיטה זו ניתן לחשב באופן מקבילי כל שורה בנפרד, אך יש לכך מחיר של איבוד הקשר בין פיקסלים באותה שורה (loss context).



איור 7.16 Row LSTM – כל פיקסל מחושב על ידי $k \geq 3$ פיקסלים בשורה שמעליו.

<u>Diagonal BiLSTM</u>

כדי לאפשר גם חישוב מקבילי וגם שמירה על קשר עם כל הפיקסלים, ניתן להשתמש ברכיבי זיכרון דו כיווניים. בכל שלב מחשבים את רכיבי הזיכרון משני הצדדים של כל שורה, וכך כל פיקסל מחושב גם בעזרת הפיקסל שלידו וגם על ידי זה שמעליו. באופן הזה ה-receptive field גדול יותר ואין loss context, אך החישוב יותר איטי מהשיטה הקודמת, כיוון שהשורות לא מחושבות בפעם אחת אלא כל פעם שני פיקסלים.

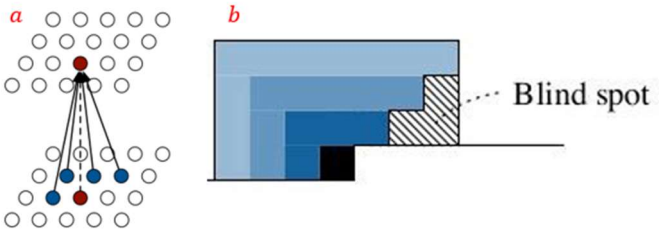


איור 7.17 Diagonal BLSTM – כל פיקסל מחושב על ידי $k \geq 3$ פיקסלים בשורה שמעליו.

כדי לשפר את השיטות שמשתמשות ברכיבי זיכרון ניתן להוסיף עוד שכבות, כמו למשל Residual blocks שעוזרים להאיץ את ההתכנסות ו-Masked convolutions כדי להפריד את התלות של הערוצים השונים של כל פיקסל.

7.3.2 PixelCNN

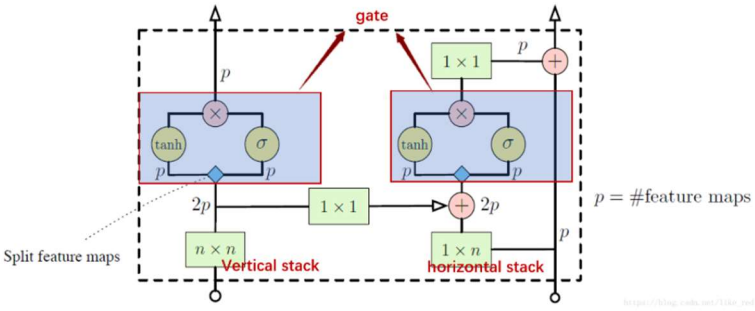
החיסרון העיקרי של PixelRNN נובע מהאימון האיטי שלו. במקום רכיבי זיכרון ניתן להשתמש ברשת קונבולוציה, ובכך להאיץ את תהליך הלמידה ולהגדיל את ה-receptive field. גם בשיטה זו מתחילים מהפיקסל הפינתי, רק כעת הלמידה היא לא בעזרת רכיבי זיכרון אלא באמצעות שכבות קונבולוציה. היתרון של שיטה זו על פני PixelRNN מתבטא בקיצור משמעותי של תהליך האימון, אך התוצאות פחות טובות. חיסרון נוסף בשיטה זו נובע מהמבנה של הפילטרים ו ה-receptive field – כל פיקסל מתבסס על שלושה פיקסלים שמעליו, והם בתורם כל אחד תלוי בשלושה פיקסלים בשורה שמעל. מבנה זה מנתק את התלות בין פיקסלים קרובים יחסית אך אינם ב-receptive field.



איור 7.18 a) receptive field של PixelCNN. b) החיסרון של PixelCNN – ניתוק בין פיקסלים יחסית קרובים.

7.3.3 Gated PixelCNN

בכדי להתגבר על בעיות אלו – ביצועים לא מספיק טובים והתעלמות מפיקסלים יחסית קרובים שאינם ב-receptive field – נעשה שימוש ברכיב זיכרון הדומה ל-LSTM, המשלב את רשתות הקונבולוציה בתוך RNN.



איור 7.19 שכבה של Gated PixelCNN.

כל רכיב זיכרון בנוי משני חלקים – horizontal stack and vertical stack, כאשר כל אחד מהם הוא למעשה שכבת קונבולוציה. ה-vertical stack בנוי מזיכרון של כל השורות שהיו עד כה בתמונה, וה-horizontal stack הוא פילטר יחיד על הקלט הנוכחי. ה-horizontal stack עובר דרך שער של אקטיבציות לא ליניאריות ובנוסף מתחבר ל-vertical stack, כאשר גם החיבור ביניהם עובר דרך שער של אקטיבציות לא ליניאריות. לפני כל כניסה של stack לתוך שער, הפילטרים מתפצלים – חצי עוברים דרך tanh וחצי דרך סיגמואיד. בסך הכל המוצא של כל שער הינו:

$$y = \tanh(w_f * x) \odot \sigma(w_g * x)$$

7.3.4 PixelCnn++

שיפור אחר של PixelCNN הוצע על ידי OpenAI, והוא מבוסס על מספר מודיפיקציות:

- שכבת ה-SoftMax שקובעת את צבע הפיקסל צורכת הרבה זיכרון, כיוון שיש הרבה צבעים אפשריים. בנוסף, היא גורמת לגרדיאנט להתאפס מהר. כדי להתגבר על כך ניתן לבצע דיסקרטיזציה לצבעים, ולאפשר טווח צבעים קטן יותר. באופן הזה קל יותר לקבוע את ערכו של כל פיקסל, ובנוסף תהליך האימון יותר יעיל.
- במקום לבצע בכל פיקסל את ההתניה על כל צבע בנפרד (כפי שהראינו בפתיחה), ניתן לבצע את ההתניה על כל הצבעים יחד.
- אחד האתגרים של PixelCNN הוא היכולת המוגבלת למצוא תלויות בין פיקסלים רחוקים. כדי להתגבר על כך ניתן לבצע down sampling, ובכך להפחית את מספר הפיקסלים בכל פילטר, מה שמאפשר לשמור את הקשרים בין פיקסלים בשורות רחוקות.

- בדומה ל-U-Net, ניתן לבצע חיבורים בעזרת Residual blocks ולשמור על יציבות במהלך הלמידה.
- שימוש ב-Dropout לצורף רגולריזציה והימנעות מ-fitting.

7. References

VAE:

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

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

https://lilianweng.github.io/lil-log/2018/08/12/from-autoencoder-to-beta-vae.html

GANs:

https://arxiv.org/abs/1406.2661

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

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

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

https://arxiv.org/abs/1710.10196

https://arxiv.org/abs/1812.04948

https://towardsdatascience.com/explained-a-style-based-generator-architecture-for-gans-generating-and-tuning-realistic-6cb2be0f431

https://arxiv.org/abs/1701.07875

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