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)

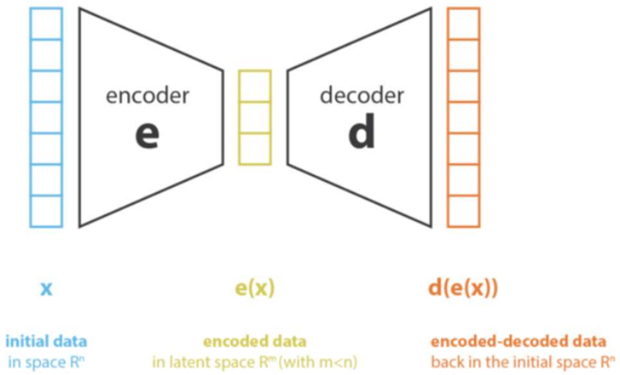
המודל הראשון הינו VAE, וכדי להבין אותו היטב יש להסביר קודם מהם Autoencoders, כיצד הוא עובד ומה החסרונות שלו.

7.1.1 Dimensionality Reduction

במקרים רבים, הדאטה אותו רוצים לנתח הוא בעל מימד 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 ממימד גבוה ולבנות מודלים עבור דאטה כזה, הרבה פעמים מנסים להוריד את המימד של הדאטה תוך איבוד מינימלי של מידע. בתהליך הורדת המימד מנסים לקבל ייצוג חדש של הדאטה בעל מימד יותר נמוך, כאשר הייצוג הזה מורכב מהמאפיינים הכי משמעותיים של הדאטה. יש מגוון שיטות להורדת המימד כאשר הרעיון המשותף לכולן הוא לייצג את הדאטה במימד 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 למימד המקורי, ולבסוף מתקבל $d\big(e(x)\big) \in \mathbb{R}^n$. אם לאחר התהליך מתקיים $x = d\big(e(x)\big)$ אז למעשה לא נאבד שום מידע בתהליך, אך אם לעומת זאת $x \neq d\big(e(x)\big)$ אז מידע מסוים אבד עקב הורדת המימד ולא היה ניתן לשחזר אותו במלואו בפענוח. באופן אינטואיטיבי, אם אנו מצליחים לשחזר את הקלט המקורי מהייצוג של במימד נמוך בדיוק טוב מספיק, כנראה שהייצוג במימד נמוך הצליח להפיק את הפיצ'רים המשמעותיים של הדאטה המקורי.



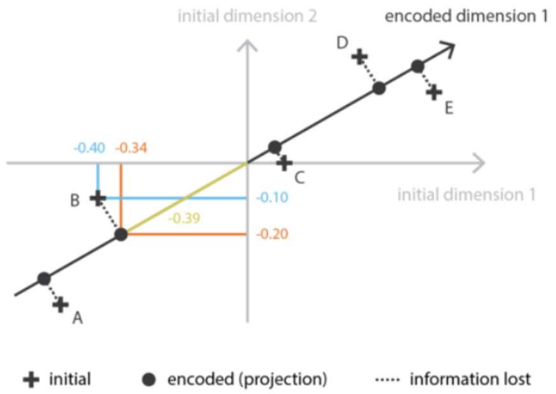
איור 7.1 ארכיטקטורת encoder ו-decoder.

כאמור, המטרה העיקרית של השיטות להורדת מימד 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 המימד באופן הבא:

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

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

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



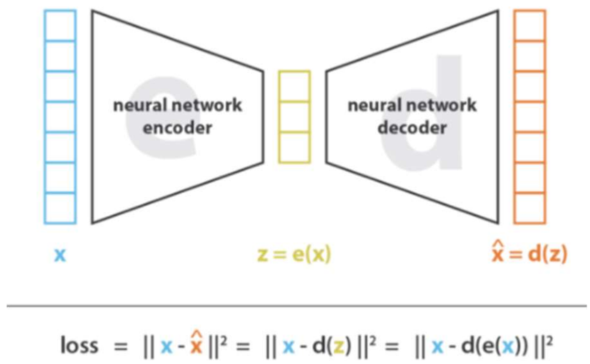
Point	Initial	Encoded	Decoded
A	(-0.50, -0.40)	-0.63	(-0.54, -0.33)
B	(-0.40, -0.10)	-0.39	(-0.34, -0.20)
C	(0.10, 0.00)	0.09	(0.07 0.04)
D	(0.30, 0.30)	0.41	(0.35, 0.21)
E	(0.50, 0.20)	0.53	(0.46, 0.27)

איור 7.2 דוגמה להורדת מימד בשיטת PCA.

במונחים של encoder-decoder, ניתן להראות כי אלגוריתם PCA מחפש את ה-encoder שמבצע טרנספורמציה לינארית על הדאטה לבסיס אורתוגונלי במימד 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:

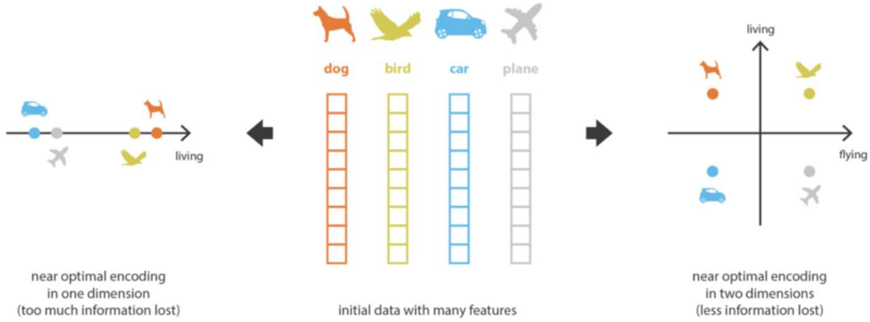


איור 7.3 Autoencoder – שימוש ברשתות נוירונים עבור הורדת המימד והשחזור.

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 המימד באמצעות טרנספורמציות לינאריות של הפיצ'רים המקוריים אך הייצוג במימד נמוך המופק על ידה

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

כעת נניח שהרשתות הן עמוקות ומשתמשות באקטיבציות לא לינאריות. במקרה כזה, ככל שהארכיטקטורה מורכבת יותר, כך הרשת יכולה להוריד יותר מימדים 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 מימד של כל דאטה לחד-מימד ללא איבוד מידע. עם זאת, הפחתת מימד דרסטית שכזו יכולה לגרום לדאטה המשוחזר לאבד את המבנה שלו. לכן יש חשיבות גדולה בבחירת מספר המימדים 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-מימדי, אבל הוא גורם לאיבוד המבנה של הדוגמאות ולא באמת ניתן להבין את ההפרדה ביניהן. לעומת זאת ניתן להוריד את המימד לדו-מימד 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 המימדים של ה-encoder.

7.1.3 Variational AutoEncoders (VAE)

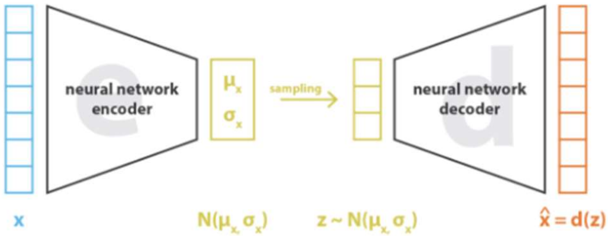
ניתן לקחת את ה-AE ולהפוך אותו למודל גנרטיבי, כלומר מודל שמסוגל לייצר בעצמו דוגמאות חדשות שאכן מתפלגות כמו הפילוג של הדאטה המקורי. אם מדובר בדומיין של תמונות למשל, אז נרצה שהמודל יהיה מסוגל לייצר תמונות שנראות אותנטיות ביחס לדאטה סט עליו אומן. הרשתות של ה-AE מאומנות לייצג את הדאטה במימד נמוך, שלוקח בחשבון את הפיצ'רים העיקריים, ולאחר מכן לשחזר את התוצאה למימד 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 סט 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 ממימד 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 המימד, 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 במימד 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 מסט 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 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 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-$q(z|x;\lambda)$:

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

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

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

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:

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

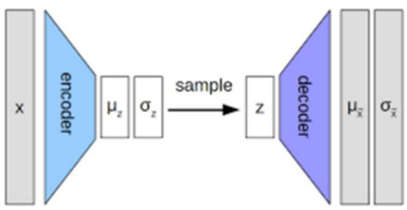
כדי לחשב את התוחלת ניתן פשוט לדגום דוגמאות מההתפלגות $z|x\sim N\big(\mu_\theta(x),\sigma_\theta(x)\big)$ ולקבל:

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

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 סט 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.

IWAE

VaDE – מחליפים את ההתפלגות של z בהתפלגות MoG.

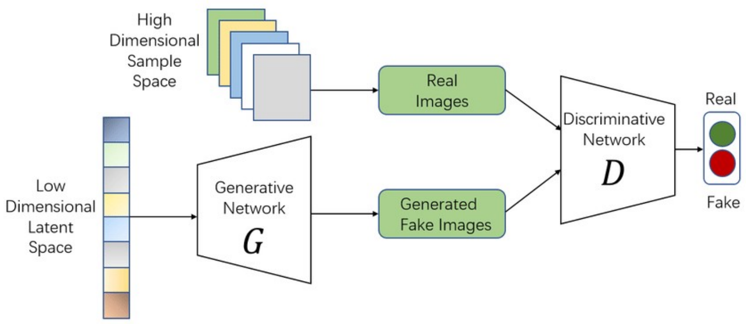
7.2 Generative Adversarial Networks (GANs)

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 מסט 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 מאומנת 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 עובר תהליך אימון על דאטה אמיתי כדי לדעת להבחין בין דאטה אמיתי לבין דאטה סינטטי, וה-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 בינארי ($y = 1$ עבור דוגמה אמיתית, ו-$y = 0$ עבור דוגמה סינטטית), והפלט שלו הוא ההסתברות שהקלט הינו דוגמה אמיתית, כאשר נסמן ב-$D(x)$ את ההסתברות הזו. כדי לאמן את ה-discriminator נרצה למקסם את $D(x)$ (=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, ולכן נרצה להביא למקסימום את ה-cross entropy של ה-generator כאשר $y = 0$ (=למצוא את ה-generator שהכי פחות מזייף, כלומר מייצר דאטה כמה שיותר אמיתי):

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

ניתן לשים לב כי הקלט של ה-GAN הינו וקטור של רעש אקראי, כאשר המטרה של ה-generator הינה ללמוד ליצור דוגמאות אותנטיות מהרעש שבכניסה. בדרך כלל הרעש הינו גאוסי בעל תוחלת אפס ומטריצת covariance $\mathbb{I}$, אך קיימים גם GAN-ים עם קלט המפולג 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 וביטול סימן המינוס:

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

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 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 וקבוע ביחס ל-θ_g. לכן נשאר רק לבדוק את הביטוי השני, שמחפש את ה-generator שמייצר הכי טוב דאטה שנראה אמיתי:

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

כאמור, המטרה היא לאמן את G בעזרת D (במצבו הנוכחי), כדי שיהיה מסוגל ליצור דוגמאות המסווגות. מציאת האופטימום נעשית בעזרת Gradient Descent/Gradient Ascent, במשך מספר מסוים של Epochs. 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 היא:

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

וכאשר מאמנים את ה-discriminator, הגרדיאנט נראה כך:

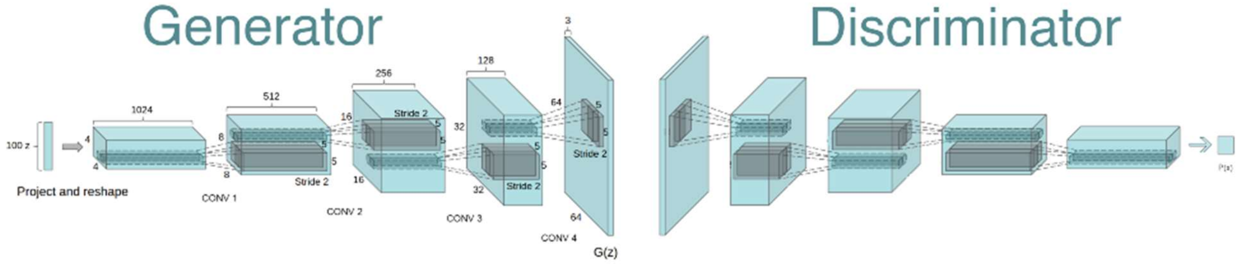
$$\nabla_\phi V\big(G_\theta,D_\phi\big)=\frac{1}{m}\nabla_\phi\sum_{i=1}^{m}\log D_\phi(x_i)+\log\Big(1-D_\phi\big(G_\theta(z_i)\big)\Big)$$

נהוג לבצע מודיפיקציה קטנה על הביטוי של ה-generator. כיוון שבהתחלה הדגימות שמיוצרות על ידי ה-generator לא דומות לחלוטין לאלו מסט האימון, ה-discriminator מזהה אותן בקלות כמזויפות. הביטוי $D\big(G(z)\big)$ מקבל ערכים מאוד קרובים ל-0, וממילא גם הביטוי $\mathbb{E}_{z\sim Noise}\log\Big(1-D\big(G(z)\big)\Big)$ שואף 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 על 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 הביטוי החדש עובד יותר טוב נומרית ומצליח לשפר את ה-generator יותר טוב.

לאחר שהוסבר המבנה הכללי של GAN, נעבור לסקור מספר ארכיטקטורות של מודלי GAN.

7.2.2 Deep Convolutional GAN (DCGAN)

כפי שהוסבר בפרק של רשתות קונבולוציה, 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.

7.2.4 Pix2Pix

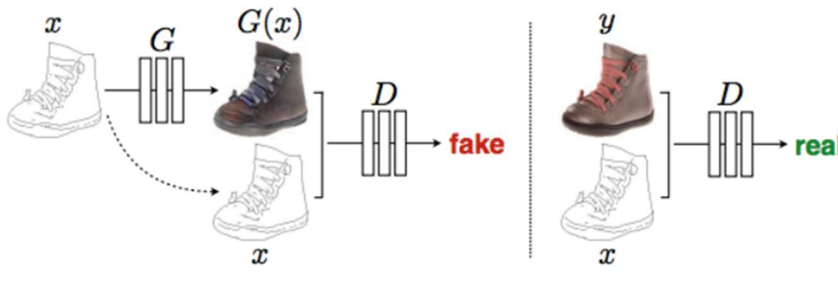
כפי שראינו, ה-GAN הפשוט מסוגל לייצר דוגמאות חדשות בעזרת וקטור אקראי z, המוגרל מהתפלגות מסוימת (בדרך כלל התפלגות נורמלית סטנדרטית, אך זה לא מוכרח). ישנן גישות נוספות ליצור דאטה חדש, כמו למשל ייצור תמונה חדשה על בסיס קווי מתאר כלליים שלה. שיטת Pix2Pix משתמשת בארכיטקטורה של GAN אך במקום לדגום את וקטור z 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, אך ה-discriminator כן משתנה – במקום לקבל תמונה ולבצע עליה סיווג בינארי, הוא מקבל זוג תמונות – את הסקיצה ואת התמונה הסינטטית, ועליו לקבוע האם התמונה הסינטטית 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 יסווג אותן כאמיתיות, וגם למזער את המרחק בין התמונה שנוצרת לבין תמונה אמיתית השייכת לסקיצה. נסמן תמונה אמיתית השייכת לסקיצה ב-y, ונרשום את פונקציית המחיר כשני חלקים נפרדים – cross entropy רגיל של GAN והפרש אוקלידי בין תמונת המקור לבין הפלט:

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

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

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

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



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

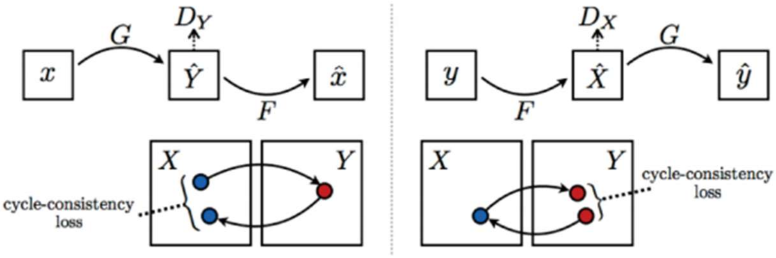
7.2.5 CycleGAN

ב-Pix2Pix הדאטה המקורי הגיע בזוגות – סקיצה ואיתה תמונה אמיתית. זוגות של תמונות זה לא דבר כל כך זמין, ולכן שיפרו את האלגוריתם כך שיוכל לקבל שתי תמונות x,y שאינן תואמות ולבצע השלכה מאחת לשנייה. הארכיטקטורה עבור המשימה הזו מורכבת משני generators – בהתחלה מכניסים את x ל-G שמנסה להפוך אותו ל-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 שמסווג האם התמונה שהתקבלה אמיתית או לא. 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 בתהליך 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 $x \approx F\big(G(x)\big)$. התהליך של השוואת הכניסה

למוצא נקרא cycle-consistency, והוא מוסיף עוד איבר לפונקציית המחיר, שמטרתו למזער עד כמה שניתן את ההפרש בין התמונה המקורית לתמונה המשוחזרת:

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

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

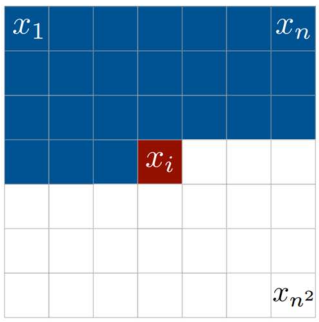


איור 7.10 ארכיטקטורת CycleGAN.

7.3 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 7.11 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

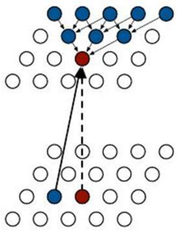
אפשרות אחת לחשב את $p(x)$ היא להשתמש ברכיבי זיכרון כמו LSTM עבור כל פיקסל. באופן טבעי היינו רוצים לקשר כל פיקסל לשכנים שלו:

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

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.

Row LSTM

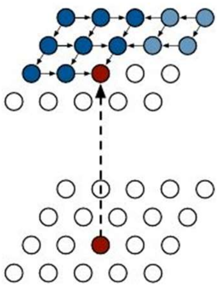
במקום להשתמש במצב החבוי של הפיקסל הקודם, ניתן להשתמש רק בשורה שמעל הפיקסל אותו רוצים לחשב. שורה זו בעצמה מחושבת לפני כן על ידי השורה שמעליה, ובכך למעשה לכל פיקסל יש receptive field של משולש. בשיטה זו ניתן לחשב באופן מקבילי כל שורה בנפרד, אך יש לכך מחיר של איבוד הקשר בין פיקסלים באותה שורה (loss context).



איור 7.12 Row LSTM – כל פיקסל מחושב על ידי $k \geq 3$ פיקסלים בשורה שמעליו.

Diagonal BiLSTM

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.

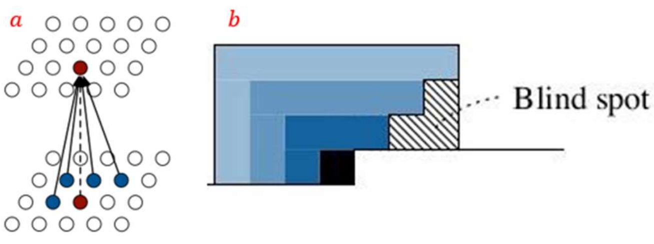


איור 7.13 Diagonal BLSTM – כל פיקסל מחושב על ידי $k \geq 3$ פיקסלים בשורה שמעליו.

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

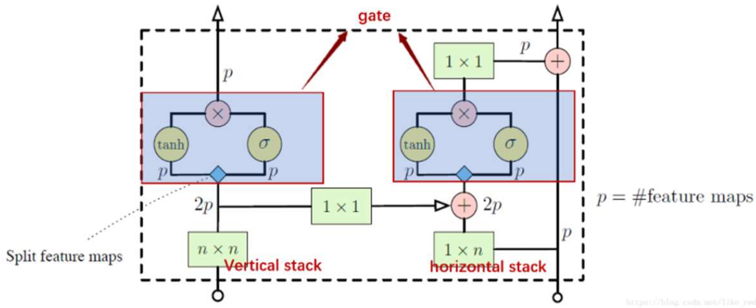
החיסרון העיקרי של PixelRNN נובע מהאימון האיטי שלו. במקום רכיבי זיכרון ניתן להשתמש ברשת קונבולוציה, ובכך להאיץ את תהליך הלמידה ולהגדיל את ה-receptive field. גם בשיטה זו מתחילים מהפיקסל הפינתי, רק כעת הלמידה היא לא בעזרת רכיבי זיכרון אלא באמצעות שכבות קונבולוציה. היתרון של שיטה זו על פני PixelRNN מתבטא 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 7.14 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 7.15 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 לינאריות ובנוסף מתחבר ל-vertical stack, כאשר גם החיבור ביניהם עובר דרך שער של אקטיבציות לא לינאריות. לפני כל כניסה של stack לתוך שער, הפילטרים מתפצלים – חצי עוברים דרך tanh וחצי דרך סיגמואיד. בסך הכל המוצא של כל שער הינו:

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

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 לצורך רגולריזציה והימנעות מ-fitting.

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