מבוא לרשתות עצביות - 67103

2 בינואר 2018

מרצה: רענן פטאל ודני לישנסקי

איני לוקחת אחריות על מה שכתוב כאן, so tread lightly
אין המרצה קשור לסיכום זה בשום דרך.
הערות יתקבלו בברכה - *noga.rotman@gmail.com*. אהבתם? יש עוד!
http://www.cs.huji.ac.il/~nogar02/integrali

תוכן עניינים

0.1 מנהלות

60 אחוז בחינה, 40 אחוז תרגילים (על סמך שלושה תרגילים - בזוגות) בפייתון. בשיעור נוסף נעבור על החבילה שנשתמש בה בתרגילים - TensorFlow של גוגל. משתדלים שיהיה כמה שפחות עבודה שחורה ויותר התמקדות בחומר.

אימייל: raananf@cs.huji.ac.il / danix@cs.huji.ac.il

שעות קבלה: ימי שלישי 13-12. למי שזה לא נוח, ניתן לפנות במייל ולקבוע שעה אחרת.

ספר: זהו תחום מאוד חדש מבחינת קורסים, אז "אין לנו ספר טוב להציע לכם". קישור אונליין לספר אופציונלי http://www.deeplearningbook.org, אך הוא לא עוקב אחרי כל הקורס. וכמו כן נדבר על מאמרים מהשנים האחרונות, אפשר להעזר גם בהם.

השנה רענן רוצה לא לפרסם את השקפים, כי הוא מאמין שמבחינה פדגוגית זה לא דבר נכון, עדיף לנסות לסכם.

0.2 הקדמה

תזכורת ממבוא למידה: בלמידה יש שני סוגי מודלים:

- מודלים דיסקרימינטיבים - ידועים גם כקלאסיפיירים. כאן ממדלים את boundry בין המחלקות, אבל לא באמת מתמקצעים במידול של כל אחד מהקלאסים (לא יודעים איך למשל להפיק אינסטנסיאציה מאחד הקלאסים).
- מודלים גנרטיביים - ממדלים את ההתפלגות של מחלקות אינדיבידואליות. ניתן לדגום מההתפלגות.

artificial neural networks מודלו בהשראת המודל הביולוגי. אלו מודלים דיסקרימינטיביים. כזכור, לנוירונים ביולוגים, יש קלט ופלט. הנוירונים לא מגיבים בצורה לינארית: יש להם סף של אקטיבציה, מתעלמים מגירוי חלש, וכאשר סכום הגירויים עולה מעל סף מסויים, הנוירון "נותן את הירייה שלו הלאה", כלומר נעשית אקטיבציה של הנוירון. הקישוריות ביניהם היא די מורכבת (ונראית רנדומלית).

המודלים המתמטיים שהתחילו לעסוק איתם (ואנחנו עדיין מתעסקים עם המודלים שהופיעו לראשונה בשנות החמישים) הוא המודל הקלאסי של היחידה הבסיסית:

אינפוטים, מקושרים למשקלות, פונקציה סוכמת, ולבסוף מגיעים לפונקצית אקטיבציה (**להכניס ציור**). קל מאוד לתכנת את המודל הזה.

השאלה היא כיצד מחברים את היחידות הללו לכדי רשת נוירונים מלאה. כזכור, רשת נוירונים תכיל שכבת קלט יחידה, מספר כלשהוא של שכבות "חבויות", ושכבת פלט יחידה. השאלה הזו מעסיקה והרבה מאוד חוקרים בשנים האחרונות, כמובן ולכל בעיה יתכן ורשת אחרת תתאים. לאט לאט נכיר את הארכיטקטורות שמשתמשים בהם היום, ומדוע הן עובדות ביחס למה שהיה בעבר.

חשוב לזכור כי רשת נוירונים היא בעצם פונקציה של הרבה מאוד משתנים, הרבה יותר ממה שראינו במודלים אחרים של למידה. מן הסתם המשמעות היא שנצטרך הרבה מאוד מידע. הרעיון של deep neural networks הוא ארכיטקטורה בה יש הרבה מאוד רשתות חבויות (באלכסנט, אחד הרשתות הראשונות מסוג זה, היו 5-6 שכבות, וברשתות ידועות היום יש סדר גודל של 20), וזהו רעיון מאוד פופולרי בשנים האחרונות.

0.2.1 קצת היסטוריה

המודלים הראשונים נולדו בשנות החמישים. העובדה שאנחנו משתמשים בהם עדיין היום מראים את החוזק של מודלים אלו.

לפני אמצע שנות החמישים, השימוש במחשבים היה למשימות חישוב מאוד פשוטות: חישוב מסלולי טילים, אינטגרלים, טלפוניה, קידוד, וכו'. חשבו עליהם כמעין extension של מחשבון כיס.

ב-1956 נפגשו כמה חוקרים בולטים בכנס, והצהרו שאולי ניתן להשתמש במחשבים כדי להתחקות אחרי אינטליגנציה אנושית. למשל, לפתור מבוף, להבין טקסט, שליטה ברובוטים. בשנת 1958, Frank Rosenblatt הציע את הרעיון של רשת נוירונים (רעיון שנקרא אז connectionism).

נעשתה השקעה מאוד גדולה, אך בשל בעיות, חלקן טכניות (המחשבים באותן שנים לא התאימו למשימות, היה חסר מידע לאמן את הרשתות, מורכבות החישוב), המחקר בתחום זה עבר ל"שינת חורף" מ- 1974 עד 1980.מושג שאנחנו אמורים להכיר הוא הפרספטרון, שהוא למעשה רשת נוירונים בעלת נוירון יחיד. IBM אפילו בנתה מכונה שממממשת אותו. עוד דבר שתרם להפסקת התפתחות המחקר - הספר של Minsky and Papert ביקרו את הפרספטרון וטען שאינו יכול לפתור אפילו בעיות פשוטות (למשל XOR).

בשנות ה-80 חלה תחייה מחודשת. כל מיני רשתות קטנות התחילו לתת תוצאות. הופיעו רשתות חדשות - Hopfield, אך אלו לא תפסו ממש עד היום, ונדבר עליהן בהמשך. כמו כן, ההתפתחות של ה-back-propogarion לחישוב הגרדיאנט, שיטה אותה נלמד מאוחר יותר ומשתמשים בה עד היום, ועזרה בחישובים של הרשתות הללו.

בשנת 1987 נכנס התחום לשינת חורף נוספת, וזאת עקב מחשבים מאוד מהירים שהפכו את התחום ללא רלוונטי, שכן היה קל יותר לממש בתוכנה את הפונקציות של רשתות הנוירונים.

משנת 1993 עד 2000, חוק מור חל על החוזקה של המחשבים, שרק הלכו ונהיו יותר ויותר מהירים. בתקופה זו AI זכה לכמה הישגים, למשל המכונה שניצחה את קספרוב בשח, אם כי בעולם העסקי המשיכו לא להאמין ביכולת של AI לפתור בעיות.

וכאן אנחנו מגיעים לימינו אנו. כידוע, החל משנת 2010 בערך חלה התקדמות אדירה בתחום של למידה. הופעת ה-deep learning גרמה לשיפור משמעותי בתוצאות של רשתות נוירונים. כמות המידע הזמינה לנו,התקדמויות טכנולוגיות (כח החישוב של GPUs), וכמו כן נסיונות שונים עם ארכיטקטורה של רשתות עזרו (אותם נראה בהמשך הקורס), עזרו בהשגת מטרות חדשות ולפתור בעיות שלהן הקדישו שנים של לימוד באופן פשוט יחסי. היום באמת אפשר, ללא הבנה מעמיקה של רשתות נוירונים, למדל בעיה ולפתור אותה בעזרת רשת נוירונים, וזו עוד סיבה בגינה הנושא מאוד מבוקש גם בתעשיה.

חלק I

חלק תיאורתי

1 Convolutional Neural Networks - CNN

רשת מסוג מסויים שהצליחה "לנצח" הרבה ארכיטקטורות אחרות. רשתות אלו מנסות למדל את מה שקורה ב־visual cortex. שתי הנחות:

- איזור השפעה: כל נוירון רגיש רק לסביבה מאוד קטנה (פיקסל אחד והסביבה שלו), הנקראת בביולוגיה receptive field.
- התמונות כגירוי, כסיגנל, מתנהגים בצורה סטציונרית במרחב. כלומר, מבחינה סטטיסטית, כשמתסכלים על איזורים קטנים של התמונה, אין משהו מיוחד באיזור אחד של התמונה שלא מופיע באיזור אחר (בצורה גסה, האובייקט בדר"כ מופיע באמצע התמונה). הנחה זו לא לגמרי מדוייקת, ובאמת בשלב מסויים הופיעו גם שכבות אחרות שמתעלמות מההנחה הזו. לפי ההנחה, אפשר לשכפל את הנוירונים, כלומר להשתמש בנוירון שעובד בצורה זהה ־ כלומר עם אותם משקולות. הנחה זו נקראית translation invariance.

הדבר העיקרי שרשתות אלו הכניסו היא ההנחה השניה ־ כי הנחה זו מקטינה לנו מאוד את דרגות החופש מבחינת משקולות. כיצד בונים?

- שמים את הנוירונים על שריג רגולרי
- משתמשים באותם משקלות בשריג: parameter sharing.

צריך לזכור כי יש הרבה נוירונים, וכל איזור זוכה לאותו ניתוח, אבל זה לא אומר שהתוצאות לכל איזור הן זהות! נזכור גם כי הפוקוס של החלונות הם בהבדל של פיקסל ולכן יש חפיפה של נוירונים על כל פיקסל.
כאשר מפעילים פעולות כאלו עם כל ההזזות, סכימה של weighted averaged הופכת לקונבולוציה, ומכאן השם. נוודא כי הטענה הזו נכונה:

$$I(x,y),\ w(s,t)$$

$$N(x,y)=\sum_{s,t} I(x+s,y+t)\cdot w(s,t)$$

$$\int I(x+s)\,w(s)\,ds \overbrace{=}^{t=-s} \int I(x-t)\,w(-t)\,dt$$

$$\overline{w(x)}=w(-x),\ \overline{I}(x)=I(-x)\Rightarrow \int \overline{I}(t-x)\,\overline{w}(t)\,dt$$

1.1 ארכיטקטורה טיפוסית

העולם של רשתות קונבולוציה גדל, אך עושה רושם שיש הרבה משותף לרשתות אלו: יש ארכיטקטורת אב שנראה שכל הרשתות שקיימות היום משתמשות באיזה וריאנט שלה. ארגיטקטורה זו כוללת את המרכיבים הבאים בסדר שבה הם מופיעים. אל דאגה, נדבר על כל אחד מהמרכיבים הללו באריכות בהמשך:

- קונבולוציה ־ אופורטורים ILT עם restricted spacial support (כלומר פילטרים קומפקטיים). נשים לב כי הרזולוציה נשארת כפי שהיתה.
- bias ־ הוספה של activation offset. גם כאן הרזולוציה נשארת כפי שהיתה
- Pointwise non-linearity: מידול של האקטיבציה. יש כאן כמה אופציות, למשל RelU.
- Pooling/striding: מורידים רזולוציה לתמונה (בפעם הראשונה בתהליך)
- שכבות שהן fully connected: כל הנוירונים מחוברים זה לזה.

$$I \xrightarrow{Conv} I \times f \xrightarrow{bias} I \times f + b \xrightarrow{RelU} RelU\left(I \times f + b\right)$$

לאחר מכן מבצעים Pooling, ואז ממשיכים לחזרה על התהליך מספר פעמים. בסוף התהליך הרזולוציה קטנה מאוד. כאן אנחנו "זורקים" את ההנחה ההנחה השניה (מאפשרים לנוירונים להיות שונים זה מזה), עושים לתמונה reshape לוקטור, ואז מכפילים את התוצאה במטריצה (לאו דווקא ריבועית, לפעמים מלבנית כדי להקטין את המרחב) ומוסיפים bias. שוב מפעילים RelU, ושוב אותו תהליך... יתכן שהוקטור כאן קטן אם השתמשנו במטריצה מלבנית, והוקטור הסופי הוא ה־prediction. אנחנו שוב נדבר על דברים אלה בהרחבה בהמשך.

האם המיקום של ה־reshape אם נעשה בשלב המתואר למעלה או בהמשך? לא! הרשת תלמד את אותו הדבר, ונקבל מטריצה עם פרמוטציה של השורות.

1.1.1 שכבת הקונבולוציה

קונבולוציה היא טרנספורמציה לינארית שהיא translation-invariant (אמ"מ). מכיוון והיא טרנספורמציה לינארית, ניתן לכתוב אותה כמטריצה. פילטרים קומפקטיים ־ המטריצה של הקונבולוציה מציגה את הפילטרים בכל הזזה שלהם ־ ולכן היא באמת translation invariant:

$$L\left(f\left(t+s\right)\right)\left(t\right) = L\left(f\left(t\right)\right)\left(t+s\right)$$

אם נזיז את סדר השורות ונכפול בה, נקבל עדיין את אותה התוצאה. במקרה זה מספר הפרמטרים קטן מאוד ממטריצה כללית, ולכן מקבלים כאן שיפור ביעילות. בדוגמא בשיעור מקבלים 5×3 (כאשר 5 הוא מספר הכניסות שאינן אפס בדוגמא זו) מול N^2:

$$\begin{pmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{pmatrix}$$

31.10.2017

להשלים כאן חלק מהמחברת כאשר המחשב השתגע לי.

דוגמא ־ הארכיטקטורה של AlexNet, הרשת הראשונה שהצליחה להשיג יתרון משמעותי על ImageNet (הרבה מאוד מאמרים מתייחסים למאמר המתאר את הרשת הזו).

נשים לב כי בעוד הפילטרים קומפקטיים ב־x ו־y, הם אינם קופמפקטיים בציר ה־z, ואפשר לחשוב עליהם בציר הזה כdot product (אין מושג של קרבה, בניגוד לצירים האחרים). כל פעם שעושים קונבולוציה של פילטר כזה עם התמונה, מקבלים תמונה בעומק 1, מהסיבה שהפילטר והאות היו מאותו העומק, ואנחנו מזניחים את המקרים של השפות של התמונה (במקום לחשב את השפות ואז יוצאים מהrange, מחשבים את הקונבולוציה עבור פיקסלים שהם בתוך מסגרת שבה לא מקבלים תוצאות מחוץ לשפה). סטודנט מציע: במקום להשתמש בפילטר חד ערוצי ולעשות קונבולוציה רק עם ה־red, אחר כך עם ה־blue, וכו'. היום ב־TensorFlow מתחילים לעבוד על פעולות אקזוטיות כאלה כדי להוריד את מספר דרגות החופש במודל. נשים לב כי הפילטרים האלה לבעיה שלנו לא עוזרים (אנחנו לא יודעים על אינטראקציה בין ערוצים), ולכן בעוד שיש בדיקות בנוגע לבעיות אחרות, אבל במקרה של recognition אנחנו לא נתעסק במקרה הזה כי לא נקבל תוצאות טובות.

דוגמא בשקופית 11 של התוצאות מ־AlexNet. בשכבת הקונבולוציה נלמדו 96 פילטרים שנקראים Gabor filters (לא הבסיס של פוריה, אלא קוסינוס וסינוס). הטרנספורם פוריה שלהם (כשקופית) מתאים כל אחד לתדר אחר. זו האינפורמציה היחידה שהולכת לעבור מהתמונה הלאה, ולכן זו בעצם המהות של התמונה.

1.1.2 Pointwise Non-linearity and Bias Layer

שתי פעולות נקודתיות בשקופית יחידה (12).

במקור, תפקיד פונקצית האקטיבציה הזה: להתחקות אחר ה־activation potential. מקובל להשתמש בכמה פונקציות: סיגמואיד \ Logistic function ־ היא פונקציה מאוד גזירה, והיתה נוחה לשימוש בעבר. היום משתמשים הרבה ב־ReLU - rectified linear unit. הסטורציה לא צריכה להיות סביב אפס, ה־bias יכול להזיז את הגרף של איזו פונקצית אקטיבציה שנבחר לאן שנרצה.

> שימו ♡!
> רענן משתמש הרבה פעמים במונח RelU כשהוא מתייחס לאיזושהיא פונקצית אקטיביציה, לאו דווקא RelU (זאת כנראה כי היום במחקר יש הרבה מאוד שימוש בפונקציה הזו), למשל בשקופית 13 יש שימוש בפונקצית הטנגנס ההיפרבולי. מכיוון ואני מתעדת את מה שהוא אומר, אני גם כותבת הרבה פעמים RelU כשהכוונה היא לפונקציה אחרת.

זהו המקום היחיד בתוך כל הרשת של פעולה לא לינארית! קונבולוציה לינארית, הטלה לינארית, ושות'. למה זה טוב? כל המטריצות האחרות לבסוף שוות ערך למטריצה אחת יחידה מאוד דקה (בגלל הלינאריות). ביחס

לקלסיפיקציה זה לא הצליח לעזור הרבה כי כל דרגות החופש הללו הן redundant. הרכבה של פונקציות אפיניות היא גם פונקציה אפינית, אך כאשר מכניסים פעולה לא לינארית, גם אם פשוטה ביותר, מגדילה לנו מאוד את המרחב הפונקציות שרשתות נוירונים יכולות לבטא (נראה את התיאוריה בהמשך).

נשים לב כי הערכים השליליים ממופים לערכים לא שליליים. זה נותן לתמונה יכולת להביע נורמות \ נוכחות של דוגמא. למה? כלומר, מה היה קורה לו היינו מקבלים גם ערכים שליליים? לא היינו יכולים למדוד את התגובה של פילטר מסויים בתמונה, כי אם סתם נסכום ערכים שהם חיוביים ושליליים, נקבל כל הזמן ערכים סביב האפס (בשל האופי של הנתונים שלנו - כי ל־edge detectors יש נטיה כזו). הטענה היא שהרכיב הזה, שמאפס את השליליים, פחות או יותר שקול לנורמה. לכן סכימה של השכבה הבאה תיתן לנו יכולת להעריך כמה העצם היה נוכח בשכבה הקודמת, כלומר כמה תגובה יש לפילטר בשכבה מסויימת בשכבה הבאה על ידי סכימה. זו עוד תכונה שהשכבה זו מוסיפה לרשת שלנו. עוד מעט נראה תגובה שבה הדבר הזה קורה.

אנחנו נדבר בהמשך על אופטימיזציות של רשתות, ועל בעיות שעולות, אבל בשלב הזה נעלה בעיה אח, והיא: לערכים שאנחנו מקבלים יש נגזרות נמוכות. בדר"כ מאתחלים את כל הביאסים והפילטרים לערכים רנדומיים. יתכן כי באופן רנדומי נקבל פילטרים שמקבל ערכים שליליים כל הזמן. ואז ה־ReLU יאפס את כל התגובה לתמונה, ולכן אם לא יהיו שינויים הרשת תחשוב שהגענו לנקודה "טובה". במקרה זה הנוירונים "מתים" - אם נוציא אותם נקבל את אותה למידה, ולכן בזבזנו חישוב. על מנת להמנע מזאת, יש פונקציה נוספת שנקראת leakyRelU, ובה מקבלים ערכים שליליים קטנים מאוד, ואז במקרה הזה לאט לאט יהיה פעפוע לאיזור של ערכים חיוביים אותם אנחנו מחפשים. כלומר, איזורי הקיטום יכולים להיות בעיתיים מבחינת האופטימיזציה, ולכן בשלב האופטימיזציה משנים קצת את הפונקציה כך שהיא לא אי שלילית.

1.1.3 דוגמאות!

(שקופית 13) איזה כיף שיש מימוש של TensorFlow ברשת!

http://playground.tensorflow.org

זהו אתר שמריץ רשת ומאמן אותה, ונותנת אפשרות לשחק עם רשתות מאוד פשוטות כדי לקבל אינטואיציה איך דברים עובדים. מומלץ בחום.

כאשר מורידים את הגמישות ועוברים לפונקצית אקטיבציה לינארית (שקופית 14), מאבדים הרבה מאוד מהמידע, והתוצאה נראית שונה לגמרי! זה משקף את אי היכולת של אופרטורים לינאריים להתמודד עם בעיות יומיומיות.

שקופית 15: הקלט הוא תמונה, בה יש או ריבועים או מלבנים, והם יכולים להיות או חיוביים או שליליים. אנחנו רוצים שהרשת תספור את מספר הריבועים בלבד, תוך כדי התעלמות מהמלבנים. הפלט: מספר שמשקף את מספר הריבועים החיובים והשליליים.

הרשת היא מאוד קטנה, הכי קטנה שרענן יכל לחשוב עליה:

- שני פילטרים 5×5
- RelU
- סכום התמונה הראשונה והשניה, ועל שני המספרים האלה מפעיל מטריצה מלאה, כלומר בסה"כ:

$$(1)\ SumRelU\ (I * f_1 + c_1) \cdot a_1$$
$$(2)\ SumRelU\ (I * f_2 + c_2) \cdot a_2$$
$$\Rightarrow (1) + (2) + b$$

- Loss של L_2

לפי התוצאות: הרשת מצליחה! בהרצה אחת, התקבל a_1, a_2 חיובי (נובע מהרנדומיות בשלב ההתחלתי), bias שלילי. בהרצה שניה קיבלנו bias פחות או יותר אפס. מה שזה אומר, שכדי לקבל ערכים בגרסא הראשונה, השכבה האחרונה יש לה משהו שתורם חיובית לספרה, וה־bias השלילי בא לפצות על הערכים החיוביים האלה. העוצמות פרופוציונליות לעצמים שיש, וגם יש נטיה לרעש שצריך לבטל מהסכום. זה בדיוק מה שאנחנו רואים! בהרצה השניה ה־a_1, a_2 אחד שלילי ואחד חיובי, ולכן ה־bias היה קרוב לאפס.

עכשיו נחליף את ה־RelU בפונקצית אקטיבציה לינארית. התוצאה בשקופית 16 - התוצאות השתנו באופן דרסטי. כל פילטר פעם מגיב חיובית ופעם שלילית, ולכן נשארים סביב האפס (כי הוא לא מאמין לסכומים של

הפילטרים, ולכן נותן ערך קבוע שמרן בהנתן תמונות שונות שממזער את ערך L_2). כלומר, הפרדה לינארית לא טובה במקרים האלה.

דבר אחרון - מחליפים את ה־RelU בפונקצית ABS - ערך מוחלט, שאיננה פונקציה לינארית, ולכן לא סובלים מהבעיה שראינו בוריאנט הקודם של הבעיה. והנה, ההפסה והדיוק שלנו חוזרים לערכים דומים לזה של ה־RelU. נוכל לשאול, מדוע לא משתמשים בפונקציה הזו במקום ה־RelU? נראה זאת בהמשך, אבל אינטואיטיבית - היא מכניסה יותר מידי "קשקוש", יותר מידי רעש.

Poolying Layer 1.1.4

הורדת רזולוציה מרחבית. מקובל להשתמש בפי שתיים בכל ציר, כלומר בוחרים פיקסל אחד מתוך כל ארבעה. כיצד בוחרים? יש כמה שיטות:

- stride (subsampling)
- average pooling - בא לעזור לקונבולוציה בכך שלמעשה עושה קונבולוציה משלו בעזרת פילטר של במקרה שבדוגמא רבע רבע רבע רבע.
- max pooling - די פופולרי

בבעית הרקוגניציה הקלאסית, אנחנו רוצים להמיר את המידע המרחבי במידע סמנטי, ולכן מעבירים תגובות מרחביות ללייבלים, או epitoms. לאחר שדגמנו את התמונה המקורית והורדנו את המידע המרחבי, והפעלנו פילטרים, זה מתאים לסקאלות יותר ויותר גדולות של המרחב המקומי. תכונה זו מרשה ל־classifier לחפש צורות במספר סקאלות. או במילים אחרות, כל נוירון מושפע מסביבה די גדולה של התמונה. תכונה זו מאפשרת לנו באיזה שהוא שלב לתת הכרעה לגבי מה היה בכל התמונה - זה מתאפשר על ידי השלב של ה־pooling.

Max pooling הוא החזק ביותר מבחינת match-oriented, כלומר הוא חזק ביותר במובן של ה־translation invariant (המקסימום המקומי שומר על עצמו - אם כי לא לחלוטין). כלומר, הרשת לא תתבלבל אם העצם נמצא במיקום מסויים או חמישה פיקסלים ימינה.

07.11.2017

בשבוע שעבר דיברנו על אפשרויות שונות לשכבת הפולינג - stride, average pooling ו־max pooling (המחפש את התגובה הבולטת ביותר מתוך הערכים הנתונים, בדוגמא שעליה דיברנו - המספר המקסימלי מבין הרביעיה). כאמור ל־max pooling יש יותר נטיה לשמור (בניגוד לאופציות אחרות) את התכונה של ה־translation invariant (לא תמיד, תלוי בערכים הספציפיים שמקבלים אחרי הזזה).

ראינו שהלייבלים, או מה שהפילטרים דוגמים מהתמונה epitoms הם "דברים" שמאוד בולטים בתמונה, ולבסוף באמת הדברים הללו נבנים לכדי עצמים. האיזור שאנחנו מקבלים הוא ביחס מסויים לתמונה, וכשחוזרים לתמונה המקורית, אנחנו מקבלים סביבות שהולכות וגדלות.

עוד דרך לתאר את האינטואיציה הזו - בעזרת העובדה שאנחנו עובדים ברזולוציות שונות, אפשר לקבל אפקטיבית דטקטורים של פיצ'רים ברזולוציות יותר קטנות. נדבר קצת על ההרכבות של השכבות וננסה להבין מה הולך שם. נגיד שה־receptive field הכולל של רמה n מורכב מכל הפילטרים עד רמה n. ניקח בדוגמא ארבעה פיצ'ים לאורך ציר y, אלכסון, ציר x ואלכסון הפוך, כך שבשלושת הפילטרים הראשונים הכחול בחוץ והאדום בפנים, ובפילטר האחרון האדום בחוץ. בדוגמא נותנים ערכים כך שנקבל אוריאנטציה של מעגלית. הפילטר שירכיב את הפילטרים האלה אנחנו מקבלים בעצם פיצ'ר דידקטור לצורה של חצי מעגל, כך שהאדום בפנים והכחול בחוץ. בדוגמא בשקופית, מקבלים תתי חלקים המתאימים לאותו עצם - בעזרת יכולת בחירה מאוד גבוהה לרשת, מקבלים הרבה צורות שהרשת יכולה לחפש בתמונה - במקום ליצר פיצ'ר לכל קלאס, היא מיצרת מעין "אבני לגו" מהן ניתן לבנות את ה־patterns הרלוונטיות. זו האינטואציה - כעת נפתח זאת באופן מתמטי:

$$\begin{aligned}
&\overbrace{g}^{Layer2} * \overbrace{\overbrace{(I*f)}^{convolution-layer1}}^{pooling}\overbrace{[2h]} \\
&= \sum_t g(t) \sum_s I(2(n-t)-s) f(s) \\
&= \sum_t \sum_s I(2n-2t-s) f(s) g(t) \\
&\overbrace{=}^{2t+s=:m \Rightarrow s=m-2t} \sum_m I(2n-m) \sum_t f(m-2t) g(t) \; (*)
\end{aligned}$$

זה כבר נראה כמעט כמו קונבולוציה, מלבד האלמנט של ה־2. בעצם מקבלים "ריווח" באופן הבא:

$$\begin{aligned}
f &= f_1 f_2 f_3 f_4 f_5 f_6 \\
\uparrow g &= g_1 0 g_2 0 g_3 0
\end{aligned}$$

כאשר האיבר השני נקרא ה־up sampling של g, וזהו בעצם ניפוח על ידי הוספת אפסים. תחת סימון זה נקבל כי הביטוי שמצאנו שווה ל:

$$(*) =\downarrow (I * (f* \uparrow g))$$

דוגמא קונקרטית:

$$f = [1, -1]\,,\ g = [1, 1]$$
$$\Rightarrow\uparrow g = [1, 0, 1, 0] \Rightarrow\uparrow g * f = [1, -1, 1, -1]$$

שאלה: אם אנחנו עושים קונבולציה של פילטר 3 על 3 שוב ושוב, מה נקבל?

$$\underbrace{\underbrace{f * f}_{5} * f * ...}_{7}$$

צריך קצת לשכנע את עצמנו כאן מתמטית, אבל בשכבות המאוד עמוקות ברשת, מקבלים שה־receprive field הוא בגודל כל התמונה (או כמעט כולה).

דוגמאות בשקופיות, שוב מראות לנו כי הפולינג מעורבב עם הקונבולוציה בעצם משיג לנו את התכונה שבכל שכבה אנחנו מקבלים פאטצ'ים יותר ויותר גדולים ומורכבים.

1.1.5 Fully-Connected layer

האינטואיציה - לכל איזור בתמונה השכבה יכולה להחליט שהוא מייצר רגישות אחרת.

שכבה זו בעצם מקבילה לפעולה המתמטית של כפל במטריצה כללית כלשהיא. כל פלט של כפל זה היא מכפלה פנימית של השורה של המטריצה עם כל התמונה - "בדרך שהשכבה רוצה לנתח אותה", שכן המטריצה לא חייבת להיות מטריצת קונבולוציה או כל מטריצה אחרת.

ב־imagenet למשל, השלב בו מתחילים להשתמש ב־fully connected layer הוא ב־64×64 (מתחילים מ־224×224).

עיקר היכולת הייחודית היא יצירת תלויות קבועות ביחס לקטגוריה, למשל השמיים למטה, האדמה למטה, וכו'. מי שנתן את הלייבל לתמונה הסתכל באמצע, ולכן הקטגוריה נקבלת לפי המרכז, כדי שהרשת תתייחס לזה (ולא נגיד לעצם בצד שמאל) צריך את השכבה הזו.

נשים לב כי בזמן ששכבות הקונבולוציה משרתות את כל הקלאסים, השורות של המטריצה "מתמחות" בקלאס מסויים. החלק הראשון של הרשת, המכיל את שכבות הקונבולוציה, Relu, pooling וכו' נקרא feature learning, והחלק השני של הרשת נקרא קלאסיפיקציה, ומכיל את השכבות הבאות:

- flatten
- fully conncted
- ולבסוף - softmax

נשים לב כי החלוקה הזו לא ברורה לחלוטין במעבר, אבל מאוד ברורה בקצוות הקיצוניים של שני החלקים.

1.1.6 Local Response Normalization Layer

לא כל הרשתות משתמשות בשכבה זו, לכן לא נכביר עליה במילים (אם כי אלכסנט כן משתמשת בה) - נירמול של כל הריספונסים. מה שבעצם מנסים לעשות - אם יש פיקסל בו כל הנוירונים הגיבו מאוד חלש, נרצה להגביר את זה, וההפך. למעשה, זהו נסיון להתמודד עם עוצמות הבהירות הכלליות של התמונה (לעזור לאינווריאנטיות של הרשת לבהירות של התמונה). שכבה זו, כאשר היא מופיעה, מופיעה בהתחלת הרשת.

$$\hat{R}_i(x) = R_i(x) \Big/ \left(c + \alpha \sum_j R_j(x)\right)^{\beta}$$

כאשר c, α, β הם קבועים, i זה הנוירון ו־x הוא הפיקסל. שימו לב כי אם c לא יהיה ממש אפסילוני (כלומר לא קטן) זה יהיה הגורם הדומיננטי, ואז פונקצית הנרמול לא באמת תעשה את עבודתה (לא תנרמל).

1.1.7 דיון על Incremental Coding

בשבוע שעבר נשאלה שאלה מאוד טובה - למה הרשת מורידה את המימד המרחבי בצורה הדרגתית לרמת הקטגוריות, ולא "במכה אחת"? זה הרי מיקר את כל התהליך.
אם אנחנו רוצים לרדת מרמת התמונה לשני קלאסים, מה יכול לעשות את זה? Fully connected (עם צירוף של RelU או משהו בסגנון על התוצאה של הדבר הזה). זה אומר שיש פרספטרון לכל אחד מהקלאסים, כל אחד מצביע על קלאס אחר. מה המגבלות שקיבלנו?

- אם למשל אנחנו רוצים למצוא חתול, איפה החתול יהיה? התוצאה שלנו היא לא transition invariant!
- תמונות השייכות לקטגוריה ספציפית לעיתים מתחלקות לתתי קטגוריות. למשל, ישנם כלבים גדולים וקטנים, עם שיער קצר וארוך, ועוד ועו'. במקרה הזה, מבנה הרשת מקשה עלינו למדל את הקטגוריה בצורה טובה, כי אנחנו מקבלים תיאור אחד ספציפי שלאו דווקא מתאים לכל תת הקטגוריות (בניגוד למבנה מרובה שכבות, בו יש מספר דרכים יחודיות ושונות זו מזו "להדליק" קטגוריה אחת).

1.2 רשתות In Practice

1.2.1 Class Prediction Loss

דבר שחשוב להבין - לקלאסים אין מידול גיאומטרי ברור (לא ניתן למספר על ידי ערך סקלארי - כלומר לכפות proximity). מה שנהוג לעשות - יצוג של כל קטגוריה על ידי וקטור היחידה e_k . כזכור, המרחק בין הוקטורים הללו הוא קבוע והוא 1. באופן אידאלי היינו רוצים שהרשת תפלוט את אחד מהוקטורים האלה כפלט, אבל רב הסיכויים שזה לא יקרה, ואז צריך להבין איזו קטגוריה מתאימה.
הרשת שדיברנו עליה עד עכשיו פולטת מספרים, ונרצה שיהיו כמה שיותר קרובות לוקטורי היחידה. בשלב הזה מקובל למפות את המספרים האלו כך שיהיו בתחום של $[0,1]$. זה מתחיל יותר להקל עלינו ועל הרשת. משתמשים ברגרסיה לוגית:

$$q = \frac{1}{(1+e^{-t})}$$

המטריקה המקובלת בה למדוד את ה-loss, כלומר תוצאת הפרידיקציה מול התוצאה האמיתית, היא ה-cross enthropy בין ההתפלגויות q ו-p:

$$H(p,q) = H(p) + D_{KL}(p\|q)$$
$$E(p) = -\sum p(x)\, logp(x)$$

כאשר האנטרופיה (הביטוי השני) מספרת לנו על מספר הביטים המינימלי שצריך כדי לקדד אות x מהתפלגות p. ה-cross enthropy מדברת על מספר הביטים המינימלי שצריך כדי לקדד אות מהתפלגות אחת כשהיא באמת מגיעה מהתפלגות שניה. כאשר שתי ההתפלגויות שוות, נקבל את האנטרופיה. עוד על הנורמה הזו - בקורס תורת האינפורמציה. זו פונקציה מאוד סבירה למקסם אותה על מנת שהפרדיקציה תהיה מתאימה! נקבל:

$$H(y,q) = -y_i logq_i - (1-y_i)\, log\,(1-q_i)$$

מקובל להשתמש בנוסחא הזו כאשר יש שתי קטגוריות, או מספר קטגוריות במקרה שהקטגוריות לא אקסלוסיביות (ואז לבדוק זוג זוג). כאשר יש אקסלוסיביות (האם העצם בתמונה חתול או כלב?) משתמשים בדר"כ בפונקציה הבאה:

$$H(y,q) = -\sum_i y_i logp_i,\ p_i = \frac{q_i}{\sum_i q_i}$$

וכאן יש קישור ל-$softmax$ שצריך להשלים.
עוד פונקצית loss אחת די טריוויאלית - L_2:

$$\min \|N(x) - y\|^2$$

מסתבר שמקבלים תוצאה יותר טובה מפונקצית הפסד ש"מענישה" יותר במקרה בו אתה בטוח בטעות.
אנחנו מגדירים את ה-loss בעזרת פונקציה אנליטית. עם זאת, את ה-accuracy אנחנו מחשבים לפי התוצאה שקיבלנו מהרשת. השגיאה הזו מבטאת האם זיהינו נכון או לא. כלומר, הדיוק של הרשת לא מחושב לפי ה-loss.
כמובן שמאמנים על מספר דוגמאות, ו-loss נסכם על הרבה דוגמאות.

Regression Loss 1.2.2

ישנן רשתות שלא צריכות להכריע בנוגע לקטגוריה, אלא צריכות לתת פרדיקציה בנוגע לאיזה שהוא מספר - למשל, מהו הגובה של האיש בתמונה. במקרה זה, הרשת כולה היא בעצם בעית רגרסיה, וה־loss הוא regression loss. במקרה זה באמת משתמשים יותר בנורמת L_2 (גם נורמות אחרות סבירות - L_1 למשל ועוד).

שימו לב שעבור רשתות מזנים שונים משתמשים ב־loss שונה. נראה בהמשך הקורס דוגמאות של רשתות משולבות, ואז יש להן מספר פונקציות loss.

אימון רשתות 1.2.3

במקרה של הרשתות שדיברנו עליהן, מהם הפרמטרים? משקלות של הפילטרים, ה־bias. למשל שכבת ה־$RelU$ חסרת פרמטרים. נסמן את הפרמטרים ב־θ. האימון מחפש את θ המינימלי עבור הפרדיקציה:

$$\min_{\theta} loss = \min_{\theta} -\sum_{n}\sum_{i} y_i^n logp_i^n / \sum_{i} \left(N_\theta\left(x_i\right) - y_i\right)^2$$

כאשר i - מספר הדוגמאות.

איך ממזערים? שיטה קלאסית היא gradient descent

Gradient Descent Optimization 1.2.4

אנחנו מתחילים מנקודה כלשהיא שנובעת מכל מיני יוריסטיקות שונות - x. בהנתן פונקצית היעד $f\left(x\right)$ ופונקצית ה־loss, נרצה למצוא באיזה כיוון "ללכת" על מנת להגיע למינימום של הפונקציה. אנחנו רוצים לבדוק את ההתנהגות המקומית של הפונקציה, אבל התנהגות מקומית לאו דוקא מצביעה על התנהגות כללית. לכן אנחנו נגביל את עצמנו לסביבת ε קטנה:

$$\min_{||z||^2=\varepsilon} f\left(x+z\right) - f\left(x\right) (*)$$

איך נעשה את זה? נשתמש בטיילור!

$$(*) = \min_{||z||^2=\varepsilon} f\left(x\right) + z\nabla f\left(x\right) - f\left(x\right) + O\left(\overbrace{||z||^2}^{\varepsilon}\right)$$

$$= \min_{||z||^2=\varepsilon} z\overbrace{\nabla f\left(x\right)}^{taylor} (**)$$

לפי משפט כופלי לגראנג':

$$\Rightarrow G\left(x\right) = C,\ \nabla F\left(x\right) = \nabla G\left(x\right)$$

ולכן צריך לפתור את $(**)$ צריך לפתור את מערכת המשוואות הבאה:

$$\nabla f\left(x\right) = \lambda 2 \cdot z$$

$$||z||^2 = \varepsilon,\ z = \pm\frac{\nabla f}{||\nabla f||}\sqrt{\varepsilon}$$

ומכאן נקבל את תהליך ה־gradient descent:

$$x^{n+1} = x^n - \varepsilon \cdot \nabla f\left(x^n\right)$$

כאשר x^0 הוא ניחוש התחלתי רנדומי.

יהיה לנו עוד שיעור שלם על אסטרטגיות של אימון. בשקופית 28 יש משוואה של מינימום כללי שאולי כדאי להשלים לכאן.

$$\theta^{(k+1)} = \theta^k ...$$

SGD - Stochastic Gradient Descent - לא מבצעים את החישוב על כל הדוגמאות, אלא על מספר קטן שלהם. סט כזה של דוגמאות נקרא batch. בדר"כ הגודל שלהם תלוי הזיכרון שיש לנו על ה־GPU, בדרך כלל מדובר בגודל של 4 עד 500 דוגמאות ל־batch.

Back Propogation (חוק השרשרת) 1.2.5

הרשת בנויה מהרכבה של מספר פונקציות, כאשר:

$$L^i_{\theta_i}(1) = L^i(I_i\theta_i)$$

התוצאה של השכבה ה־i של הרשת.
.נזכר בפורמליזם של חוק השרשרת:

$$\frac{\partial}{\partial x}(f(g(x))) = \frac{\partial f}{\partial x}(g(x)) \cdot \frac{\partial g}{\partial x}(x)$$

נשתמש בכלל השרשרת ונקבל:

$$N_{\overline{\theta}}(I) = L^3_{\theta_3}\left(L^2_{\theta_2}\left(L^1_{\theta_1}(I)\right)\right)$$

$$\frac{\partial}{\partial x}(f(g(x))) = \overbrace{\frac{\partial}{\partial x}f(g(x))}^{vector}\overbrace{\frac{\partial g}{\partial x}(x)}^{matrix}$$

$$BP5\frac{\partial N}{\partial \theta_1}(I) = \overbrace{\overbrace{\frac{\partial L^3_{\theta_3}}{\partial I}}^{Loss}\left(L^2_{\theta_2}\left(L^1_{\theta_1}(I)\right)\right) \cdot \overbrace{\frac{\partial L^2_{\theta_2}}{\partial I}\left(L^1_{\theta_1}(I)\right)}^{BP1}}^{BP2} \cdot \frac{\partial L^1_{\theta_1}}{\partial \theta_1}(I)$$

$$BP4\frac{\partial N}{\partial \theta_2}(I) = \overbrace{\frac{\partial L^3_{\theta_3}}{\partial I}}^{Loss}\left(L^2_{\theta_2}\left(L^1_{\theta_1}(I)\right)\right) \cdot \frac{\partial L^2_{\theta_2}}{\partial \theta_2}\left(L^1_{\theta_1}(I)\right)$$

$$BP3\frac{\partial N}{\partial \theta_3}(I) = \overbrace{\frac{\partial L^3_{\theta_3}}{\partial \theta_3}}^{Loss}\left(L^2_{\theta_2}\left(L^1_{\theta_1}(I)\right)\right)$$

וכשאנחנו רוצים למזער את ה־loss נקבל:

$$\overrightarrow{\theta_{h+1}} = \overrightarrow{\theta_h} - \frac{\partial}{\partial \overrightarrow{\theta}}\left(L^2_{\theta_2}\left(L^1_{\theta_1}(x)\right)\right)$$

אבל "אנחנו לא רוצים לעבוד נכון, אנחנו רוצים לעבוד חכם!". מה נעשה? נשים לב שהביטוי הראשון מופיע בשלושת הנגזרות. בהנתן התמונה, אנחנו רוצים לחשב אותה בכל אחת מהרמות של הרשת ו"לשמור בצד" ורק כשמגיעים לשכבה האחרונה אפשר לחשב את הביטוי הזה. כך נקבל את סדר החישוב המופיע למעלה, ובעצם את אלגוריתם ה־ BP.

דיפרנסיאציה של שכבה יחידה - שקופית 30, להשלים לכאן. נשים לב כי:

$$\frac{\partial N}{\partial \theta_1}(I) = \overbrace{\overbrace{\frac{\partial L^3_{\theta_3}}{\partial x}\left(\overbrace{L^2_{\theta_2}\left(L^1_{\theta_1}(I)\right)}^{BP2}\right)}^{vector} \cdot \overbrace{\frac{\partial L^2_{\theta_2}}{\partial x}\left(\overbrace{L^1_{\theta_1}(I)}^{BP1}\right)}^{matrix}}^{matrix} \cdot \overbrace{\frac{\partial L^1_{\theta_1}}{\partial \theta_1}}^{matrix}$$

החישוב נעשה באופן כזה, שכאשר מגיעים לרמה האחרונה ביותר, מכפילים את הוקטור שקיבלנו במטריצה, ואז במטריצה נוספת. אם נצליח לבצע את הכפל של הוקטור במטריצה באופן שלא נצטרך להכפיל את כל המטריצה, אז נצליח לבצע זאת בצורה יעילה (לינארית). ואכן לא מפתיע שניתן לעשות זאת - לכל פעולה בצורה שמתאימה לה, תחת ההנחות על המטריצה המתאימות לכל פעולה (מטריצה אלכסונית, למשל). למשל, הבה נגזור את שכבת ה־RelU:

$$RelU(x) = \begin{cases} x & if\ x \geq 0 \\ 0 & if\ x < 0 \end{cases} \Rightarrow RelU'(x) = \begin{cases} 1 & if\ x \geq 0 \\ 0 & if\ x < 0 \end{cases}$$

זו הנגזרת של הפונקציה פר רכיב אחד. נזכור כי $RelU$ מבצעת פעולה ביחס לפיקסל מסוים, כלומר לכל אינפוט אחר נקבל 0. דהינו, נקבל מטריצה אלכסונית:

$$J = RelU\left(I\right)$$

$$\nabla RelU\left[I\right] = \begin{bmatrix} 1 & 0 & \cdots & \cdots & 0 \\ 0 & \ddots & \ddots & 0 & \vdots \\ \vdots & \ddots & 0 & \ddots & \vdots \\ \vdots & 0 & \ddots & \ddots & 0 \\ 0 & \cdots & \cdots & 0 & 1 \end{bmatrix}$$

ולכן נקבל:

$$\underbrace{\frac{\partial L^2_{\theta_2}}{\partial x} \cdot \left(L^1_{\theta_1}\left(x\right)\right)}_{vector} \cdot \overset{RelU}{[]} = \frac{\partial}{\partial x} RelU\left[x\right]$$

בשיעורי בית: השלם את אותו תהליך לשכבות הקונבולוציה והפולינג.

הערה כללית לכל הדיון מעלה: נשים לב כי כל שכבה ניתן לגזור בשתי דרכים: או לפי הפרמטרים θ_i, או לפי המשתנה x. אלו שתי האופציות היחידות.

וכאן סיימנו את הדיון על CNN. יעלה תרגיל בנושא למודל להגשה.

1.2.6 דיון סיום

החומר כאן - כדאי לדעת, לא למבחן.

מה היה ה-state of the art לפני הרשתות שדיברנו עליהם - deformable parts model: מציאת עצמים בעזרת היסטוגרמות (הסבר מפורט בשקופית 32).

עוד מודל מקובל ב-2011 - למידה בעזרת SVM. ברשתות נוירונים אנחנו לומדים את הפיצ'רים, ב-SVM התהליך קצת הפוך - מתחילים בפיצ'רים שאנחנו מאמינים בהם, עושים למידה בהתאם להם. אם ב-SVM בחרנו את הפיצ'רים הנכונים, נגיע לתוצאה נכונה עם מעט דאטה, אבל אחרת אנחנו בבעיה. בעולם רשתות הנוירונים צריך הרבה מאוד דאטה, אבל יש בו הרבה יותר דרגות חופש, וכך המודל המתקבל הוא יותר אקספרסיבי.

1.2.7 Tensorflow - הכנה לתרגיל הבית!

זוהי ספריה של גוגל שמספקת לנו את הפונקציות המתמטיות למימוש רשתות תקשורת. היא ממומשת על ארבע שפות (סי, פייתון,...). יש לה תשתית מאוד טובה בסי שרצה מאוד מהר על GPU. Tensorflow היא אחת הספריות הכי מקובלות היום לשימושים האלה.

איך עובד באופן כללי? מעבירים את המידע פעם יחידה! מתחילים עם session:

```
sess=tf.Session()
sess.run(pull1, feed - disct = {x : input - batch})
```

ניתן להסתכל על כל התהליך כגרף, וכך באמת נראה המימוש. הקודקודים בגרף הם:

- משתנים - פרמטרים של המודל
- placeholders - קבועים\ערכים התחלתיים
- אופרטורים - אופרטורים מתמטיים המגדירים את הרשת (Conv ,Pool, וכו' - ה-RelU הוא חלק מקודקוד האופרטור אליו הוא שייך)

הקשתות הן התלויות הפונקציונליות בין הקודקודים (מהו האינפוט לאופרטורים, למשל)..

יש אפשרות לראות את הגרף שיצרנו בעזרת Tensorboard.

המשתנים הבסיסיים הם טנזורים (מטריצות ארבע מימדיות) - תמונות ופילטרים.

לאחר מכן עברנו על הקוד שנתחיל ממנו שמגדיר את הרשת. הקוד בפייתון, שבה נצטרך להשתמש בתרגיל. יש טוטוריאל על הקוד הזה באתר של טנזורפלואו, יהיה קישור בתרגיל.

2 תיאוריה של רשתות נוירונים

אחד הדברים שחוזרים בשנים האחרונות הוא, שלמרות שמשתמשים ברשתות נוירונים לפתרון כל מיני בעיות, הבסיס התיאורתי חסר מאוד. אנחנו נעשה טעימות לשתי שאלות במסגרת החלק הזה של הקורס. בסמסטר הבא יש קורס של עמית דניאלי שנכנס לתחום הזה יותר לעומק למי שמעוניין.

2.1 מה אפשר לבטא בעזרת רשתות נוירונים

2.1.1 משפט בסיסי

יש משפט די עתיק יומין, אשר שופך אור על הכוח האקספרסיבי של רשת נוירונים שטוחה (בעלת שכבה hidden יחידה). למעשה המשפט מסביר לנו כמה רחבה מחלקת הפונקציות שניתן לבטא בעזרת רשת שכזו:

משפט 2.1 (Cybenko 89)תהי σ פונקציה רציפה מונוטוני, כאשר $\sigma(-\infty)=0$ וכמו כן $\sigma(\infty)=1$ (למשל סיגמואיד). אזי המשפחה:

$$f(x)=\sum \alpha_i \sigma(w_i \cdot x+b_i)$$

היא קבוצה צפופה במרחב הפונקציות הרציפות $C([0,1]^n)$ ביחס לנורמת הסופרימום:

$$d(f,g)=sup|f(x)-g(x)|$$

או באופן יותר מדויק, תהי $g(x)\in C([0,1])$. אזי ניתן לקרב אותה טוב ככל שנרצה $\varepsilon >$:

$$\sup_{x\in[0,1]}|f(x)-g(x)|<\varepsilon$$

וכאשר $m\to\infty$:

$$f(x)=\sum_{i=1}^{m}\alpha_i\sigma(x\cdot w_i+b_i)$$

נשים לב שהסכימה היא בדיוק על הנוירונים בשכבה ה־hidden היחידה. כלומר, צריך הרבה נוירונים בשביל להביע פונקציות, אבל כשאפשר מקבלים הרבה מאוד!

כאשר מסתכלים על הגרף של פונקציה, המשפט בעצם מאפשר לנו להשתמש בהרבה מאוד אבני בנין - למשל לקחת את הפונקציה ולכווץ אותה, לשקף אותה, וכו' - כאשר α מטפל ב"גובה", ו־w מטפל במתיחה\כיווץ .

ישנו משפט של Hornik מ־91 שמרחיבה את המשפט הזה לכל σ חסומה.

אבל, כצפוי, החופש הזה בא במחיר מסוים - no free lunch!

2.1.2 סקירה קצרה

לפני המחיר, נדבר קצת על ההיסטוריה של machine learning.

- מערכות expert היו הראשונות (מה שלומדים בקורס הבסיסי של machine learning). "שיחקו" שם עם הפרמטרים ידנית.
- לאחר מכן, ML "traditional": מודלים מותאמים לבעיה. מאות בין אלפי פרמטרים שהיה צריך "לתפור" אותם. לצורך זה היה צריך קצת דאטה בשביל ה־training.
- Deep learning - הארכיטקטורות גנריות. כאן אין צורך לבחור במודל סטטיסטי לבעיה. הרבה פרמטרים עד המון פרמטרים (מיליונים עד עשרות מיליונים), ויש צורך בהמון (המון!) מידע ל־training.

ה־tradeoff כאן מאוד ברור - ככל שהבנו את הבעיה הינו צריכים הרבה פחות מידע, והיום ב־deep learning אין צורך ממש להבין את הבעיה, אבל יש צורך בהמון פרמטרים והמון מידע לאימון.

2.1.3 מה התפקיד של ה־Deep?

כאן נניח ש־σ זה RelU. אפשר לבנות את פונקצית ה"כובע" ־ פונקציה פחות ה־$x - mirror$ שלה, על ידי רשת קטנה באופן הבא:

$$\sigma\left(2\sigma\left(x\right)-4\sigma\left(x-0.5\right)\right)$$

הגדרה 2.2 נגיד שפונקציה היא t־שן־מסור (t-sawtooth) אם היא פונקציה אפינית ב־t חלקים.

למשל, σ היא 2-sawtooth, פונקצית הכובע היא $4-$sawtooth.
למה זה מעניין אותנו? כי אנחנו הולכים להשתמש ברשת הזו כאבן בנין לבנית רשת עמוקה. נשים לב לגרף המתקבל לאחר הרבה איטרציות ־ נראה כמו מסור, כאשר כל איטרציה מכפילה את מספר השיניים. אבל, אולי יש טעות בשקופית, רענן יבדוק ונקוה לפתרון בשיעור הבא.
כדי להמנע מהבעיה הפוטנציאלית, נשתמש בפונקציה נוספת:

$$\sigma\left(2x\right)+\sigma\left(2x-1\right)$$

פונקציה זו מכווצת בפקטור שתיים, מסיתה ב־1, וסוכמת את שתיהן. למעשה כאשר פונקציה זו מקבל "שן" רחבה, היא הופכת אותה לשתי שניים צרות.
נבנה רשת שמשתמשת בפונקצית הכובע, ואז משתמשת בפונקציה השניה בחזרות שלה.
נשים לב שניתן לבטא רשת של שרשורים שכאלה ברשת שטוחה, אבל זה ידרוש הרבה יותר עבודה קשה.

$$\sigma\left(\sigma\left(x\right)-2\sigma\left(x-1\right)+2\sigma\left(x+2\right)-2\sigma\left(x+3\right)\right)$$

בדוגמא זו, רשת בעומק d עם $O\left(d\right)$ נוירונים המייצרת פונקציה של 2^d שיני מסור.מכיוון וכל שן מסור דורשת C נוירונים, כדי לקרב אותה בעזרת ברשת שטוחה, נצטרך $C\cdot 2^d$ נוירונים. זהו סימן ראשון שיש משהו שהעומק נותן לנו, שקשה להשיג ברשתות שטוחות.

למה 2.3 אם f היא $a-$sawtooth, ו־g היא $b-$sawtooth, אזי לפונקציה $f+g$ היא לכל היותר $(a+b)-$ sawtooth, ו־$f\left(g\right)$ היא לכל היותר $ab-$sawtooth.

הוכחה: בציורים, נפנופי ידיים ואינטואיציות.
כאשר נקודות האי גזירות של f ו־g לא מתלכדות, לכל היותר פונקצית הסכימה תקבל "עוד" נקודת אי גזירות, כלומר מקסימום של $a+b$ שיני מסור.
במקרה של ההרכבה, סורקים בכל קטע רציף של g בודקים את כל תחום ההגדרה של f, ולכן רואים b פעמים a שיני מסור, כלומר לכל היותר $a\cdot b$ שיני מסור. ■

קצת תזכורות מלינארית, ואז נעבור לדוגמא יותר אנליטית.
יהיו $V,U,W\in\mathbb{R}^n$, $n>2$ (ל־n אחר הפתרון טריוויאלי). כשיש לנו איזשהוא וקטור V ממימד גבוה, ואנחנו רוצים לקרב אותו על ידי קומבינציה לינארית של שני וקטורים אחרים U ו־W:

$$\min_{a,b}\left\|\bar{V}+a\overline{U}+b\overline{W}\right\|^2$$

אזי גוזרים ומשווים לאפס:

$$\begin{aligned}0&=\frac{\partial}{\partial a}\left(V+aU+bW\right)^2\left(V+aU+bW\right)\\&=U^T\left(V+aU+bW\right)\left(V+aU+bW\right)^TU\\&=2U^T\left(V+aU+bW\right)=2\left\langle U,\overbrace{V+aU+bW}^{\text{the error}}\right\rangle\end{aligned}$$

דהיינו מקבלים שגיאה שהיא ניצבת לכל אחד מהצירים שאנחנו יכולים לקרב. במילים אחרות, אנחנו חיים בתת מרחב לינארי שנפרש על ידי U ו־W, ואנחנו מחפשים לקרב נקודה ב־V על המישור הזה. אז אנחנו מחפשים את הנקודה עבורה וקטור השגיאה (או השארית) ניצבת גם ל־U וגם ל־W.

תזכורת נוספת - בהנתן $U \perp V \perp W$ אורתוגונליים:

$$\min_{a',c'} \|aU + bV + cW + a'U + c'W\|$$

אזי $a' = -a,\ c' = c$ נקבל קיזוז מכל מה שחורג באופן אורתונורמלי למישור שנפרש. קיבלנו בדיוק את אותו התנאי. כלומר, במידה וכל הוקטורים אורתוגונליים, נוכל לקבל פתרון קל (השארית היא בדיוק U). לכן, יש לנו אינטרס להפוך וקטורים שלנו לאורתוגונליים.

בעזרת האבחנה מעלה, נעבור לניתוח אנליטי עם דוגמא נוספת. נרצה באופן דומה לקרב פונקציות רציפות. נסתכל על פונקציות שמוגדרות בתחום $[-1,1]$ או $[-\varepsilon,\varepsilon]$. הפונקציה הראשונה היא 1, השניה x, השלישית x^2. נרצה שהם יהיו אורתוגונליים. האם 1 אורתוגונלי ל-x? לא, כי אחת סימטרית והשניה אנטי סימטריות:

$$\langle 1, x\rangle = \int_{-1}^{1} 1 \cdot x = 1 - 1 = 0$$
$$\langle x, x^2\rangle = \int_{-1}^{1} x^3 dx = \frac{x^4}{4}|_{-1}^{1} = \frac{1}{4} - \frac{1}{4} = 0$$
$$\langle 1, x^2\rangle = \int_{-1}^{1} x^2 = \frac{x^3}{3}|_{-1}^{1} = \frac{1}{3} - \left(-\frac{1}{3}\right) = \frac{2}{3}$$

כלומר חסר לנו אורתוגונליות של הזוג האחרון. איך נתקן אותן כך שנוכל לפתור את הבעיה שלנו בצורה קלה? נשתמש בתהליך גרהם-שמידט!

$$x^2 \Rightarrow x^2 + something \times \overbrace{\langle x, x^2\rangle}^{=0} - \underbrace{\frac{\overbrace{\langle 1, x^2\rangle}^{\frac{2}{3}}}{\|1\|}}_{\sqrt{2}} \cdot \frac{1}{\|1\|} = x^2 - \frac{1}{3}$$

אזי כעת יש לנו שלושה וקטורים אורתוגונליים. 28.11.2017

שיעור שצריך להשלים! משלימה מתוך הקלטות של מתן ארגמן[1]. מכיוון וחלקו חוזר על השיעור הקודם, אני אשלים לתוך ההוכחה הקודמת ונמשיך משם. 05.12..2017

קצת תיקון מהשיעור הקודם:

ראינו שרשת בעומק d צריכה 2^d נוירונים על מנת לקרב רשת שטוחה. אבל מה בכיוון השני?

$$f(x) = \sum_{i=1} \alpha_i \sigma(w_i x + b_i),\ \alpha_i > 0$$

כאשר $\alpha_i > 0$הנחה מקלה. בשלב הראשון אנחנו נעשה את ה"טריק" של ההזזה כדי להקל עלינו:

$$h_2 = \alpha_1 \sigma(w_1 x + b_1)$$

$$x$$

כך ממשיכים בשלבים לחשב את האיברים (ציור להעתיק מהכיתה).

אם היינו רוצים לוותר על ההנחה המקלה, מה היינו יכולים לעשות? איך היינו יכולים לטפל בα_i השליליים? נחלק לשניים - את סכום החיוביים אנחנו יודעים לפתור. לגבי הסכום של השליליים - נבנה אותם כמו איברים חיוביים, נמבור אותם כמו את האיברים החיוביים, ובסוף נוסיף מינוס. כך נגיע רק ל-$3n$ נוירונים!

2.1.4 בעית האימון היא NP-קשה

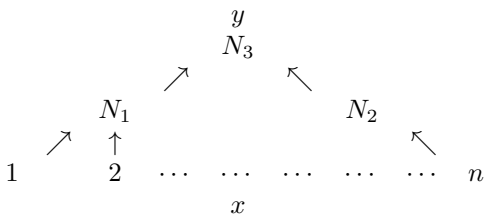
הגענו למשפט האחרון, המראה שבעית הלמידה היא בעיה NP-קשה. נראה זאת עבור רשת בעלת שלושה נוירונים. נוכיח את הטענה דרך בעיה דומה - בעית ה-$solvability$, אז נעשה רדוקציה (יאי!).

[1] מתן תזכה למצוות, או פתרון לבעית $P\ vs.\ NP$, לבחירתך.

בעית ה־$solvability$ בהנתן סט אימון בינארי $\{x_i, y_i\}$, כאשר n הוא אורך הוקטורים, ויש $o(n)$ דוגמאות בסט האימון, קבע האם קיימים משקולות וביאסים המשכפלים את הבעיה.

$$f_k(x) = \begin{cases} 1 & if\ \sum_{i=1}^{n} a_i^k x_i > b_k \\ -1 & otherwise \end{cases}$$

כאשר:



אבחנות גיאומטריות:

- הדוגמאות הן x קודקודים של ההיפר־קוביה ה־n מימדית
- N_1 ו־N_2 מגדירים מישור שהוא $(n-1)$ מימדי, כאשר כל מה שמעליו חיובי, כל מה שמתחתיו שלילי. שניהם ביחד מגדירים איזור שהוא בעצם $quadrant$ (רביע? לבדוק מינוח מתמטי).
- N_3 לא יכול לתת ערך 1 גם ל־$(-1,-1)$ וגם ל־$(1,1)$, כלומר תיוג זה לא אפשרי, אם כי אפשר לתת את התיוג -1 לשני

השאלה היא, על כן, בהנתן $\{x_i y_i\}$האם קיים:

1. מישור מפריד אחד בין 1 ל־-1
2. יש שני מישורים המפרידים ביניהם, כלומר רביע

האבחנה הגיאומטרית השלישית בעצם מראה לנו שהחלוקה לא יכולה להיות בין שני רביעונים מנוגדים זה לזה (אם הרביעונים לא מנוגדים שניהם ביחד בעצם מגדירים מישור מפריד), ולכן אלו שתי האופציות היחידות.
אם אחנו יודעים לעשות זאת בזמן לינארי, אזי הבעיה NP־קשה.
איך נראה את זה? נתחיל מהתמקדות במקרה 2. לאחר מכן נראה שמקרה 1 לא יכול להתקיים, ונסיים.

המקרה בו קיימים שני מישורים מפרידים היא NP־קשה Set-Splitting: בהנתן קבוצה סופי x_i ב־S, ואוסף של תתי קבוצות c_i ב־C, קבע האם קיימים קבוצות זרות S_1, S_2 כך ש־$S_1 \bigcup S_2 = S$ ו־c_i ב־C שלא מוכלת ב־S_1 או ב־ S_2.
זוהי בעיה NP־קשה ידועה. נשתמש בה לרדוקציה לעולם שלנו. איך נעשה את זה?
דוגמא בציור ־ נעביר את הקבוצות לקוביה ה־n מימדית, כלומר ל־$training\ set$, באופן הבא:

- שמים בהתחלה את הראשית כדוגמא חיובית
- מספר הקבוצות S הוא גודל המימד של ה־$training\ set$
- לכל i, לכל s_i יצר וקטור e_i, והוסף אותו כנקודה בסט האימון עם התיוג -1
- לכל i, לכל קבוצה c_i קבע וקטור $(s_1 \in c_i, s_2 \in c_i, ..., s_n \in c_i)$ במרחב המתויגת עם $+1$

נשם לב כי מההגדרה הזו לא יתכנו קבוצות שהם סינגלטונים (אזי יהיו מתויגות פעם אחד עם תיוג חיובי ופעם שניה עם תיוג שלילי, מצב שאיננו אפשרי).
בהנתן צביעה $S_1\,(/S_2)$ קיים מישור $P_1\,(/P_2)$ כך ש:

$$a_1x_1 + ... + a_nx_n = -\frac{1}{2},\ where\ a_i = -1\ if\ s_i \in S_1,\ and\ a_i = n\ if\ s_i \notin S_1$$

מקבלים:

- דוגמאת הראשית היא תמיד חיובית

- כל נקודה s_i היא או ב־S_1 או ב־S_2, ועל כן יש $(-1,1)$ או $(1,-1)$ (הנקודות הפוכות במישור, הרי שתיהן זרות זו לזו)
- כל c_i לא מוכל ב־S_1 ולא מוכל ב־S_2 , לכן קיימת קואורדינטה s_k שלא ב־S_1 (וקואורדינטה אחרת שלא ב־S_2), ולכן היא ערך חיובי באינדקס $a_k = n$, ולכן הדוגמא מקבלת תיוג חיובי בכל המישורים.

לכן, כיוון אחד מוכח.
בכיוון השני, נראה כי אם Solvable אז SS.
בהנתן שני המישורים $P_1\ (/P_2)$, קיימת צביעה $S_1\ (/S_2)$, אשר מוגדרת באופן הבא:

- הנקודות המופרדות מהראשית (המתויגת חיובית) הן ב־S_1 (כלומר שם כל הדוגמאות המתויגות שלילית):

$$S_1 = \left\{ s_i | \, sign\left(P_1\left(e_i\right)\right) \neq sign\left(P_1\left(\overline{0}\right)\right) \right\}$$

- באופן דומה, S_2 היא קבוצת כל הנקודות המתוייגות חיובית:

$$S_1 = \left\{ s_i | \, sign\left(P_2\left(e_i\right)\right) \neq sign\left(P_2\left(\overline{0}\right)\right) \right\}$$

מדוע $S_1 \bigcup S_2 = S$? e_i מתוייגות כשליליות ו־$\overline{0}$ מתוייג חיובי, כלומר יש להם תיוג שונה. N_3 לא יכלה לעשות את זה אם לא כל הדוגמאות היו ב־$S_1 \bigcup S_2$.
ההגדרות האלה לא מיצגות קבוצות שהן זרות, כי יכולים להופיע איברים בשתי הקבוצות. את הבעיה הזו נפתור בצורה שרירותית ־ כל דוגמא כזו נוריד מאחת הקבוצות.
כעת כל מה שניתן לנו לעשות זה להראות כי S_1 ו־S_2 לא מכילות שום תת־קבוצה c. נוכיח בשלילה כי קיים כזה $c = \{s_1, s_2, ...\}$, והוא מוכל ב־S_i. נניח בה"כ כי הוא מוכל ב־S_1. אזי לכל אחת מ־$s_i \in c$, $P_1\left(e_i\right) < 0$.
יהיו $c \in C,\ c \subseteq S$. נרצה לבדוק מה הפולינום נותן על הקבוצה הזו, ונרצה להראות כי הוא לא מפריד אותה מהראשית, כלומר c לא מוכלת לא ב־S_1 ולא ב־S_2. נניח בשלילה כי c מוכלת ב־S_1. אזי:

$$\forall i \ s.a. \ s_i \in c, \ P_1\left(e_i\right) < 0$$

מצד שני, אנחנו יודעים כי $P_1\left('c'\right) > 0$, כאשר $'c' = \sum_{s_i \in c} e_i > 0$, כלומר $P\left(\sum_{s_i \in c} e_i\right) > 0$. דהיינו קיימים a_i, b כך ש:

$$\sum_{s_i \in c} a_i e_i + b > 0$$
$$\Rightarrow \forall i \ s.a. \ s_i \in c, \ a_i + b < 0, \ \sum_i a_i + b > 0$$

זה יהיה סתירה אם b חיובי, כי אז:

$$a_i + b < 0 \Rightarrow \sum a_i + \sum b < 0$$

כמובן ו־b חיובי, כי $P\left(\overline{0}\right) > 0$, ולכן סיימנו כאן..
איך מבטלים את המקרה הראשון שדנו בו? מוסיפים נקודות ורואים שזה לא אפשרי (קצת נפנופי ידיים). אפשר אולי גם לטעון שניתן להשליך אותה לבעיה השניה, אבל זה דורש עוד מחשבה.

מאימון ל־solvability זו בעיה יותר קשה, כי צריך להביא את הפתרון עצמו.

כשמדברים על רשתות שמשתמשים בהם בת'כלס, ההוכחות האלו לא בדיוק מתאימות, ויש סדרה של מאמרים שעושים את ההשלכה לרשתות שמשתמשים בהם היום. עמית דניאלי ידבר על כך בקורס שהוזכר בסמסטר הבא. בפרקטיקה, למרות שהבעיה קשה, אנחנו פותרים אותה "טוב למדי".
סיימנו את הפרק של התיאוריה! על שאר הפרקים אחראי דני.

חלק II
חלק מעשי

3 נושא חדש!

3.1 שיטות ויזואליזציה לרשתות עמוקות

אנחנו ממשיכים עם רשתות עמוקות, אבל נדבר על דברים יותר אמפיריים (לפחות בהרצאה הזו).
רשתות עמוקות, למשל CNNs, שיפרו דרמטית את התוצאות בבעיות קלאסיות בראיה חישובית. למשל, בבעית זיהוי פנים, בעיות קלסיפיקציה, ועוד. במבחן הטיורינג של תוצאות כבר יש ניצחון 0 יש בעיות בהן הגיעו לחמישה אחוזי שגיאה, מול מיצוע של 10־5 אחוז אצל בני אנוש.
למרות הניצחון הזה, לא תמיד אפשר להבין מדוע הביצועים הם כל כך טובים. זה לא מצב מספק מבחינה אינטלקטואלית ־ אנחנו רוצים לדעת למה משהו פועל. כמו כן, אנחנו רוצים להבין כיצד אפשר לשפר עוד, ולכן צריך להבין מדוע הרשתות הללו עובדות עבור מקרים מסויימים, ופחות טוב עבור מקרים אחרים.
כאן נכנסת החשיבות של ויזואליזציה של רשתות. קשה מאוד להבין רשתות עמוקות ־ למשל ברזנט יש 150 שכבות. בעזרת ויזואליזציה אנחנו מקווים להבין יותר לעומק את הרשתות שאנחנו בודקים
נראה היום עבודות שנחשבות קלאסיות בנושא ויזואלי זציה של רשתות. על הדרך נכיר כל מיני קונספטים שהשתמשו בהם מאוחר יותר למטרות אחרות, כך שנלמד דברים נוספים.

3.1.1 Visualizing and Understanding Convolutional Networks, Zeiler & Fergus, 2014

עבודה ראשונה שנסקור. זוהי עבודה מאוד חשובה ומצוטטת מאוד.
שאלות:

- מה לומדת רשתות CNN?
- איזה חלק של המודל הוא החשוב ביותר לביצועים?
- כמה הפיצ'רים שהרשתות האלו מחשבים הם אוניברסליים?

הגישה של המאמר: גלגול אחורה של התוצאות! תהליך זה ידוע גם בשם דה־קונבולוציה.
על תוצאות הקונבולוציה מפעילים Relu, על זה Max Pooling, וכו' ־ הקונפיגורציה הקלאסית. איך מגלגלים את התהליך הזה אחורה?
מתחילים מ־Max Unpooling: ממפים את הערך המקסימלי בחזרה. על המפה בגודל שמתקבל מפעילים Relu[2], וכך מקבלים הטלה של פיצ'רים מהשכבה הזו בחזרה לשכבה היותר מפורטת. ממשיכים בדרך זו וכך פיצ'רים מפעפעים לתמונה. המטרה כאן היא לא לקבל את התמונה המקורית, שכן היא נתונה לנו, אלא לראות איזה חלק מהתמונה גרם לאקטיבציה חזקה, וכן אילו נוירונים ויזואלים מתקבלים. במודל זה קוראים למעבר בין רמה ברשת המקורית ולבין הרמה בתהליך הגלגול לאחור בשם switch.
הם מוסיפים לרשת הקונבולוציה הרגילה שכבה ששומרת, עבור שלב ה־Unpooling, מהיכן במטריצה הגיע הערך הכי גדול, ולפי שכבה זו יודעים היכן למקם את הערך הכי גדול.
נניח שאנחנו רוצים לעשות ויזואליזציה ברמה כלשהיא k.

- בהנתן תמונה ספציפית נותנים לה לפעפע ברשת עד לרמה ה־k.
- בוחרים פיצ'ר יחיד במפה כל פעם, ולוקחים את האקטיבציה החזקה ביותר.
- עוברים לתוך רשת הדה־קונבולוציה בעזרת ה־switches.

כמובן ורוצים לעשות את התהליך על יותר מתמונה אחת, אלא עבור סט תמונות (למשל סט הולידציה ב־ImageNet, שהיא בגודל של 50k). בוחרים מהן את התמונות שיוצרות את האקטיבציות הכי חזקות, ומסתכלים על כמה מהן ולא על אחד יחידה.
בשקופיות ־ דוגמאות של פיצ'רים משכבות שונות. כשמתקדמים מעבר לרמה הראשונה כל פיצ'ר גדל ברמת הסיבוכיות מעבר לשכבה הקודמת, ולכן הוזואליזציה קצת יותר מסובכת, והם מתחילים לבטא תכונות יותר מורכבות. הוויזואליזציה עצמה נראית קצת מבלבלת, אבל כשמסתכלים על הויזואליציה מול התמונה, אפשר להבין מה הפיצ'ר מייצג ־ למשל, פרצופים של כלבים דומים (למשל ברמה 4), פרחים וכדומה.

[2] ולא את ההפוך של ה$Relu$!

זו שיטה יחסית פשוטה, אבל גם עם כלי כזה אפשר להכניס שיפורים בארכיטקטורה שיגרמו לה לעבוד יותר טוב על המשימה שלשמה היא נוצרה. למשל, אם מסתכלים על הפיצ'רים בשכבה הראשונה בדוגמא, ניתן לראות שלא כולם מנוצלים היטב. ברור שלא צריך הרבה פילטרים לזהות איזור ש"לא קורה בו הרבה". אפשר לראות תדרים נמוכים ותדרים גבוהים בדוגמא שלנו, אבל חסרים תדרי ביניים, וכאמור לא כולם מנוצלים. אבחנה זו גרמה להם להקטין את הפילטרים בשכבה הראשונה מ־11×11 ל־7×7. בביצועים[3]. הם גם שיחקו עם "קצב הדגימה". ברמה השניה, אפשר לראות שיש כל מיני מבנים שלא מופיעים במידע עצמו. זו תופעה שנקראית aliases: תדרים גבוהים שנדגמו יותר מידי כך ש"מתחזים" לתדרים יותר נמוכים. בעיה זו נראתה בעזרת הויזואליזציה! כדי לתקן את השגיאות הללו, המחברים של המאמר בחרו להקטין את הגודל של צעד הקונבולוציה ברמה הראשונה מ־4 ל־2. שני השינויים הללו גרמו לשיפור של 1.7% בביצועים, מספר משמעותי לבעיה שהם בדקו (AlexNet).

Occlusion Sensitivity ויזואליזציה על תמונה ספציפית - מה יש בה ואיפה שגורם לזיהוי נכון או שגוי?
התהליך: משתמשים בחלון זז על התמונה ש"מכסה" כל פעם חלק אחר של התמונה, ובודקים את ה־feature map מה גרם לירידה החזקה ביותר לפיצ'ר הכי דומיננטי. למשל בדוגמא, ה"חלון" על הפנים של הכלב גרם ל־response הכי נמוך.
טריק נוסף - מיפוי של איפה צריך "להסתיר" כדי לפגוע הכי הרבה בוקטור ההסתברות של הקלאסיפיקציה. למשל בתמונה השלישית, פרצופים הם חשובים לאקטיבציה, אבל הם לא רלוונטים לקלאסיפיקציה של סוג הכלב, ורואים את המקרה הזה במיפוי של וקטור ההסתברות, ולא במיפוי מהסוג הראשון. גם בתמונה השניה, ההסתרה של הגלגל בתמונה של המכונית גורמת להכי הרבה טעויות בקלאסיפיקציה, ולא הפנים של הנהג או הטקסט על הרכב.
מיפוי שלישי - איזה קלאס הוא הכי סביר הסתברותית: אם מסתירים את הגוף מקבלים סוג כלב אחד, אם מסתירים את הראש סוג אחר של כלב, וכו'.

ג'נרליזציה של פיצ'רים האם הפיצ'רים שנלמדים הם ספציפיים ל־training set נתון?
בדקו בעזרת training images מדאטה סט אחר, ושכבות $1-7$ של CNN שאומנה על ImageNet, אימון קלאסיפייר softmax.

3.1.2 ויזואליזציה ע"י אופטימיזציה

מאמר: Deep Inside Convolutional Networks - Simonyan, Vedaldi and Zisserman 2014. שיטה זו מבצעת ויזואליזציה על ידי אופטימיזציה על האינפוט. עושים את ה־back propogation עד התמונה עצמה, ואז משנים על מנת לגרום למשל לערך של פיצ'ר מסויים להיות הכי גבוה.

$$\arg\max_I S_C(I) - \lambda \|I\|_2^2$$

$S_C(I)$ הוא ה־score של הקלאס C, כפי שחושב על ידי השכבה האחרונה של רשת שחושבה עבור תמונה I. הביטוי השני הוא לשם רגולריזציה (למנוע רעש - במקרה זה, "בואו לא ניתן לערכים להשתולל").
דוגמאות בשקופית בכיתה - קיבלנו מעין דוגמאות שחוזרות על עצמו כמה פעמים, למשל פנים של כלב דלמטי, משקולות, וכו'. אם מאתחלים את התמונה הראשונית עם הגרלה של רעש רנדומי, אז נקבל כל פעם ויזואליזציה אחרת, ולא את ה־pattern הספציפי הזה (ההתכנסות היא למינימום לוקאלי ולא גלובלי).
אפשר גם בהנתן תמונה וקלאס, לרשום rank של הפיקסלים של I שהשפיעו על הציון של הקלאס, $S_C(I)$, ולחשב את הנגזרת בעזרת מעבר אחד של back propogation.
הקונספט החשוב כאן שאח"כ השתמשו בו במקרים אחרים לקבלת אפקטים מאוד מוצלחים - אפשר לעצב את האינפוט בכל מיני אופנים.

3.1.3 CAM - Class Activation Layer

מתוך המאמר Learning Deep Features for Discriminative Localization.
מאמנים קלסיפייר שאמור לזהות מתוך k הסקאלרים מה הקלאס, ומסכמים ביחד כדי לקבל את ההסתברות (העיפו את ה־fully connected והחליפו במנגנון הזה).
בהנתן המשקולות שהתקבלו, אפשר לכפול כל אחד מהערוצים לפני המיצוע ברשתות, לחבר ביחד, ולקבל Class Activation Layer.
באופן יותר פורמלי:
יש לנו Global Average Pooling Layer, או GAP:

$$F_k = \sum_{x,y} f_k(x,y)$$

[3] נשאלה השאלה, מדוע השינוי כזה? הרי זה לא ברור מעליו מהאבחנה. לא ברור, וזה עוד חלק בספרות שבו אין עדיין הסבר אינטואיטיבי.

אינפוט ל־softmax:

$$S_c = \sum_k w_k^C F_k = \sum_k w_k^C \sum_{x,y} f_k(x,y) = \sum_{x,y} \sum_k w_k^C f_k(x,y)$$

נקרא לתוצאה הזו ה־CAM - Class Activation Map:

$$= \sum M_C(x,y)$$

ובשורה תחתונה:

$$M_C(x,y) = \sum_k w_k^C f_k(x,y)$$

דוגמאות בשקופיות (מהמאמר?). המחברים בדקו על כמה רשתות, ויש אובדן מידע אבל הוא "לא כזה גדול".
לאחר שמקבלים את מפות האקטיבציה, אפשר למצוא קלסיפיקציה וטריקים של threshholding אפשר למצוא את האובייקט בתמונה בצורה מאוד פשוטה.

3.1.4 עוד שיטה בעזרת אופטימיזציה

Understanding Neaural NEtworks Through Deep Visualization - Yosinski et all 2015.
בשיטה זו, בכל איטירציה של אופטימיזציה, מחשבים את הגרדיאנט של האקטיבציה:

$$X \leftarrow r_\theta\left(x + \eta \frac{\partial a_i}{\partial x}\right)$$

כאשר r_θ הוא אופרטור שיכול להיות אחד מהאופציות הבאות: מקטינים בצורה מלאכותית את התמונה, משתמשים ב־blur גאוסיאני, קליפינג עם נורמה קטנה ועוד אופציה. לפי דני, כשכותבים כמה אופציות במאמר זה אומר שאף אחת מהן לא מנצחת תמיד, ואכן במאמר מראים דוגמאות שבהן אופרטורים שונים עובדים יותר טוב.
דוגמאות במצגת. אפשר לראות מבחינה איכותית שהדברים דומים להמ שראינו קודם

3.1.5 Inceptionism

רעיון מאוד דומה שלא מוסבר כל כך, דני מצא אותו בבלוג של גוגלנט. הפרטים המדוייקים לא ברורים, אבל "התמונות נחמדות". הקוד אמור להיות בגיטהאב.
הרעיון: מזינים את התמונה למערכת, עושים feed foward, לוקחים אקטיבציות יותר גבוהות עבור אותה תמונה ברמה מסויימת, ואז עושים אופטימיזציה שמטרתה להקצין את התכונות הללו, שאולי לא היו בולטות בתמונה המקורית, אבל יהיו יותר נראים בתמונה שתתקבל. נשים לב שכאן לא הכל מעשה אדם, אלא הרשת משנה את התמונה. תלוי באיזה רמה עושים את זה, מקבלים אפקטים שונים. ברמות ראשונות מקבלים הגברה של edges (נראה כמו brush strokes - משיכות מכחול). כמובן שהתוצאות תלויות במה שהרשת ראתה בזמן האימון.

3.1.6 קל לרמות רשתות עמוקות

ראינו שהשתמשנו עד כה בכל מני רגולריזציות. הסיבה לכך היתה כי תהליך אופטימיזציה עלול להתכנס לכל מיני רעשים - דוגמאות בשקופיות, למרות שהרעש גדול הרשת מוצאת קלסיפיקציה שלא נראית לעין בביטחון די גבוה. אפשר ליצור תמונות "מבלבלות" כאלה על ידי יצירת רעש בתמונה קיימת פר פיקסל, או שימוש בפונקציות יוצרות:

$$I = \sum_k w_f f^k(\theta_k)$$

כאשר f יכולה להיות פונקצית סינוס למשל, קומבינציה של גרדיאנטים, ועוד.
זהו לקח שאנחנו צריכים לזכור - נוח מאוד לחשוב שרשתות עושות את העבודה "כמונו", עם שימוש באותה אינטואיציה. עם זאת, זה לא נכון!
בימים הבאים יפורסם תרגיל לשחק עם הדברים האלה.

4 סקירת ארכיטקטורות מפורסמות

בשיעור זה נסקור ארכיטקטורות שונות. לא "באמת" חייבים לדעת את כולן - הרבה מאמרים פשוט לוקחים ארכיטקטורה כלשהיא ועושים לה fine tuning להתאים לבעיה ספציפית. עם זאת, יש פרטים שחשוב לדעת, במיוחד אם אנחנו נרצה להשתמש ברשתות לבעיות בהמשך.

4.1 "התותחים הכבדים"

4.1.1 AlexNet 2012

ארכיטקטורה זו זכתה באתגר ImageNet בשנת 2012, תחרות שנתית ידועה . ה־top 5 error שלה הוא 15.3% (הבא בתור היה 26.2%, פתרון זה לא היה מבוסס על רשתות עמוקות), כלומר פה היתה קפיצת מדרגה בדיוק.

ספציפית מה שהם עשו פה (**להכניס תמונה מהמאמר**) נראה קצת מסובך, כי במקור הם אימנו את הרשת הזו על 2 GPUs בעלי זיכרון קטן יחסית, ולכן חלק מהבחירות בארכיטקטורה זו נבעו מבעיות טכניות - בהתחלה שני ה־GPUs לא רואים אחד את השני ואז לעיתים הפילטרים לא באמת מסתכלים על כל ה־range שלהם. מימושים מודרניים יותר של AlexNet מניחים ריצה על GPU אחד, לו יש מספיק זיכרון (חומרה הזמינה היום), ויתרה מזו, המגבלות הטכניות של שימוש במספר GPUs "מוחבאות" בתוך ה־framework ולא באים לידי ביטוי בארכיטקטורה עצמה .על כן אנחנו נתעלם מה"דו זרועיות" של הארכיטקטורה הזו ונתייחס רק לאחת ויחידה.

אז איזה שכבות יש לנו כאן? לפי סדר הופעתם משמאל לימין:

C1 N1 MP1 C2 N2 MP2 C3 C4 C5 MP5 FC6 FC7 FC8

כאשרC הן שכבות קונבולוציה, MP = max pooling ,FC=fully connected ,ו־N היא local normalization layer, שכבה המנרמלת את מה שהתקבל מהקונבולוציה ומה־RelU. המחברים מצאו כי שכבה זו עוזרת לביצועים. בהמשך שכבת הנורמליזציה הזו ירדה.. בסך הכל קיבלנו חמש שכבות קונבולוציה, 5 שכבות max pooling, ושלוש שכבות fully connected.

עוד חידוש ברשת הזו - לפני כן היה שימוש באקטיבציה בדר"כ רק ב־tanh, כאן היה השימוש הראשון במקום הפונקציה הזו ב־RelU והמחברים הראו עד כמה הפונקציה הזו עוזרת בבעיות מסוג אלו. רשת זו השתמשה בעוד "טריקים" שאומצו בשמחה על ידי הקהילה, למשל רעיון ה־dropout שראינו בתרגיל הראשון.

לרשת יש 650 מיליון נוירונים, 60 מיליון פרמטרים. על כן צריך הרבה דאטה כדי לאמן את הרשת הזו - המחברים השתמשו ב־1.2 מיליון תמונות מ־1000 קלאסים שונים.

כמו כן, הרשת משתמשים ב־local response normalization. עוד טריקים: data augmentation, כאמור dropouts, variable learning rate ו־weight decay.

4.1.2 ZFNet 2013

בשיעור הקודם ראינו ויזואליזציות ואת הטענות של זיילר ופרגוס. הם השתמשו בתובנות האלה לשפר את AlexNet, ואכן הצליחו להקטין את אחוז הטעות ל־11.7% (ואכן הצליחו לזכות באותו אתגר בשנת 2013). עם זאת, הם הסתמכו בעיקר על fine tuning ולא על רעיונות חדשים ולכן רשת זו לא מעניינת כמו ה־AlexNet.

4.1.3 VGG Net 2014

רשת זו לא זכתה באתגר ב־2014, אבל הסתמכו עליה בהמשך פעמים רבות ולכן חשוב להכיר אותה. אנשים הראו שיש לרשת זו יכולת הכללה טובה.

גם כאן יש ירידה משמעותית באחוז הטעות באתגר - 7.3% (אבל כאמור לא זכתה).

יש לרשת שתי גרסאות - אחת עם 16 שכבות, השניה עם 19שכבות, אבל העמוקה יותר לא משפרת בהרבה ומסבכת את החישוב, לכן נדבר בעיקר על הרשת הראשונה. מדובר ב־CNN עם פילטרים 3×3 עם סטריידים של 1, שכבות max pooling 2×2עם סטרייד 2 , ובסה"כ מדובר ב־144/138 מיליון פרמטרים.

אפקטיבית גם למשימות קלאסיפיקציה וגם משימות לוקאליזציה.

מדובר בשלוש שכבות ובכל שכבה יש קרנלים של 3×3 בסטרייד 1 יש את אותו receptive fields כשכבת 7×7 יחידה. אבל, זוהי רשת מאוד עמוקה, ובעזרת האי לינאריות היא מקבלת הרבה מאוד כח!

איזה שכבות יש כאן? כתוב מטה בסדר הפוך, מקווה להפוך בהמשך כדי שזה יהיה יותר קריא (כרגע קראו מימין לשמאל).

Softmax, 3 FC, Pooling, 3 Conv, Pooling, 3 Conv, Pooling, 2 Conv, Pooling, 2 Conv, Pooling, 2 Conv

אחד הדברים שהראו באותו מאמר או במאמר אחר (??): הפיצ'רים בשכבות הראשונות (ה־fully connected) טובים ל־visual tasks. למה מסתכלים על הרמה זו ספציפית? הירידה ל־1000 (שכבה ראשונה\אחרונה של FC) הוא עדיין פיצ'ר ש"לא יודע מה אתם הולכים לחשב". יש הרבה עבודות שמשתמשות בפיצ'רים של רשת זו ולא מסתכלים בכלל על ה־FC אלא רק על התוצאות ש של הקונבולוציות, כי בשכבות ה־FC אנחנו "זורקים" את המידע המרחבי.

מי כן ניצב באתגר ב־2014? GoogLeNet!

4.1.4 GoogLeNet 2014

זוכת האתגר לשנת 2014. כבר הזכרנו אותה בשיעור קודם. רשת זו השתמשה ברעיון ה־inception שהופיע במאמר קודם. ניצחה את VGG באחוז אחד.

כשמחפשים פרטים על הרשת הזו, יש מקומות שכותבים שיש בה 100 שכבות, ויש כאלה שכותבים שיש 22 שכבות. זה תלוי בדרך בה סופרים את השכבות. למרות שזו רשת מאוד עמוקה (לעומת 8 השכבות של AlexNet למשל), יש בה פי 12 **פחות** פרמטרים מ־AlexNet.

מכאן הארכיטקטורות לא נכנסות בשקף אחד! היא מאוד מאוד (מאוד!) עמוקה. אחד המקומות שבהם נחסכו הפרמטרים היא בסוף הרשת, בשכבת ה־average pooling. 90% או יותר מהרשת זה ה־FC. פעולה זו לוקחת את כל אחת מהשכבות ומחשבת את ה־average, וכך הופכת את התמונות לוקטור חד מימדי, וזה חוסך הרבה מאוד פרמטרים. מה שמגיע ל־average pool מגיע stack של $7\times7\times1024$, ושם זה כבר חוסך בערך מיליון פרמטרים, כי מקבלים כאאוטפוט וקטור של 1024.

מהו הקסם של ה־inception module, ומה הקסם מאחוריו?

בכל שלב של הרשת יש המון החלטות: האם זו שכבה קונבולוציה או פולינג? אם קונבולוציה, מה גודל הקונבולוציה? צריך בכל רמה לעשות את ההחלטה הזו, וההחלטה הזו פוגעת בכלליות של הרשת. אם היינו יכולים לקחת שילוב של כולם, או לתת לתהליך לבחור את הארכיטקטורה הכי טובה, אזי היינו בוודאי משפרים את התוצאות. וזו המוטיבציה לרעיון: הבה נריץ מספר אופציות במקביל! כלומר **ציור**

הערה: בעזרת קונבולוציה 1×1 לקחת קומבינציה לינארית של הערך של הפיצ'רים השונים. פה יש הרבה בלבול בטרמינולוגיה: לפעמים פיצ'ר מתכוון לכל הבלוק, לפעמים למפה, ולפעמים לעמודה. צריך לשים לב לא להתבלבל, ולנסות להבין לפי הקונטקסט. כאן הפיצ'רים הם של כל נוירון (מפה?), ומטילים אותו למרחב פיצ'רים אחר שיכול להיות יותר גדול או יותר קטן בהתאם לכמה פילטרים של 1×1 קיימים. זה לא רק טרנספורמציה לינארית על האינפוט, אלא גם הפעלה של RelU שמכניסה את הלא־לינאריות לתמונה.

filter concatenation - עושים stacking בעומק. במודול הזה ה־MP מופעל בסטרייד אחד, ולכן התוצאה היא בעלת אותו מספר נוירונים בגובה וברוחב, וכמו כן (מכיוון ואין לינאריות), מספר הערוצים נשאר קבוע. בקונבולוציות מספר הערוצים גדל, ולכן בסה"כ בשכבה זו העומק גדל, והעלות החישובית להפעלת כל הקונבולוציות האלה גדלה.

מה הפתרון של גוגל נט לעזור לעניין הקטנת העומק שהתקבלו מכל שכבות הקונבולוציה? נכניס עוד שכבות קונבולוציה![4] לצורך זה מוסיפים עוד שכבות קונבולוציה 1×1 לפני שכבות הקונבולוציה (לפני כי זה יותר זול חישובית), ושכבת 1×1 אחרי ה־max pooling.

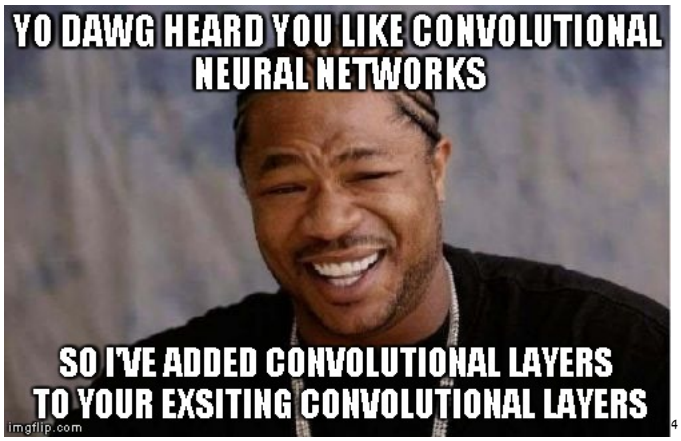
דוגמא מספרית בשקופית מתוך קורס של סטאנפורד - Case study:GoogLeNet.

עוד שקופית של הארכיטקטורה ב־horizontal (אם יהיה לי העתק אכניס כאן), כשב"אמצע" יש כל מיני stacked inception modules ש"יוצאים החוצה". מה זה? המחברים אמרו שקשה לחשב את הloss רק בסוף. אז הם הכניסו עוד 2 נקודות חישוב (כל פעם אחרי 3 אינספשנים כאלה). בסוף בסוף הרשת המאומנת מסתכלים רק על האאוטפוט הסופי, אבל בזמן אימון משתמשים בשתי הנקודות הנוספות.

4.1.5 Microsoft ResNet 2015

Deep Residual Learning for Image Recognition, He at al., (late 2015). זכתה בשנת 2015 עם שגיאה של 3.6%!! התקדמות מאוד מהירה בשנים ספורות, ודיוק טוב יותר מזה של בן אדם ממוצע. לרשת זו יש 152 שכבות, אבל עם סיבוכיות נמוכה מזו של VGG.

אבחנה של המחברים: למרות שבתיאוריה רשתות מאוד עמוקות אמורות לעבוד יותר טוב, הלכה למעשה מקבלים מהן דיוק פחות טוב, יש קושי לאמן אותן טוב. זוהי לא בעיה של אוברפיטינג, אלא של אופטימיזציה. אם



[4]

עושים פלוטינג של שגיאת האימון, רואים שהשגיאה לא יורדת ב־50 שכבות מול 20 שכבות (במאמר יש ממש פלוט של הטענה הזו). הרעיון של רשת זו מתבסס על האבחנה. הרעיון: במקום לאמן לחשב את $H(x)$ פונקציה לא טריוויאלית, יותר קל לאמן לחשב פונקצית שארית $F(x)$, כאשר $F(x) = H(x) - x$. זהו רעיון שבשיטות נומריות ובתחומים שונים השתמשו הרבה, אבל המחברים הביאו את הרעיון הזה לרשתות נוירונים בפעם הראשונה. לשם כך, הם בונים בלוקים שלומדים את השארי: **ציור**. דני לא ראה בשום מקום הסבר ריגורוזי למה ההתנהגות היא כזו. ב"נפנוף ידיים", על ידי הוספת המעקף (skip or shortcut) אפשר ללמוד את ה־identity, עדיין מקבלים את כל מה שמקבלים קודם, אבל לא שוכחים את האינפורציה שהיתה לך בכניסה, היא עדיין זמינה. כלומר היה ותהליך האימון גרם לטשטוש משהו באינפוט, הוא עדיין קיים. את כל הבלוקים ניתן לכתוב כך שכל המימדים המרחביים נשארים אותו דבר, וכך אפשר לחשב פונקציות מאוד מורכבות תוך כדי שההתכנסות תהיה יותר קלה.

ציור של רזנט בהשוואה ל־VGG. יש כאן מצבים בהם מספר הערוצים משתנה (לפי הצבע), שם במקום $F(x) + x$, יש כפל במטריצה $F(x) + w \cdot x$, זה קורה בקישורים ה"מקוקווים", הקוים הישרים מעבירים את המידע כמצופה. אין פה רשת FC מסיבית, רב השכבות הן 3×3 ולכן מקבלים מספר פרמטרים קטן.

הרבה ארכיטקטורות מאז הצגת המאמר הזה הכניסו את צעד המעקף שהוצג לראשונה כאן. לקונספט זה יש הרבה שימושים כיום, למשל יש ורסיה של גוגלנט עם צעד מעקף.

אלו הם ה"תותחים הכבדים", הקלאסיים. נראה עכשיו כמה ארכיטקטורות מעניינות אחרות (שלאו דווקא מסתמכות על CNN).

4.2 קונספטים נוספים

4.2.1 Siamese Neural Networks

מאמר -
Siamese Neural Networks for One-shot Image Recognition (Koch, Zemel, Salakhutdinov, ICML 2015).

יש לנו מידע training מאוד מצומצם, דוגמא אחת לכל class, וצריך ללמוד טוב ממידע זה. כאנטיתזה ל־MNIST, כאן משתמשים בדאטה סט בשם Omniglot (צריך להכניס ציור). אנחנו רוצים ללמד את המודל איך לחשב מרחק בין כל אחת מן התמונות, כך שמכל 20 התמונות, המרחק המינימלי יהיה מהאימג' המיועד.

איך מאמנים את הרשת?

יש שתי רשתות שיש להן את אותם המשקולות בדיוק, ומזינים לכל אחת פריט אחר ותמונה נוספת, והרשת צריכה להגיד אם זוג הפריטים שייכים לאותה קטגוריה או לא. כשמאמנים את הרשת אפשר להראות הרבה דאטה (לא one shot, קצת "מרמים" פה), אבל לא בזמן הריצה. את ה־ground truth אנחנו יודעים: כל זוג שייך לאותו הקלאס או לא. בזמן הריצה, נותנים תמונה וכמה אופציות, והרשת צריכה להגיד איזה מהפריטים הנוספים הוא באותה מחלקה כמו התמונה שניתנה. שיטה זו רלוונטית גם לקבצי קול.

איך העסק הזה עובד? יש שתי רשתות, שתי "זרועות", רשת קונבולוציה "די רגילה", כל אחת מוציאה feature vector ומחשבים את ההפרש:

$$p = \sigma\left(\sum_j \alpha_j |h^{(j)}_{1,L-1} - h^{(2)}_{2,L-1}|\right)$$

מאמנים את הרשת בעזרת משהו שנקרא constrastive loss (נוסחא ארוכה עם לוגים כדי לאפס כש"צריך" ופרמטר λ לרגולריזציה). באמצעות מספיק זוגות של same ו־different אפשר לאמן את הרשת כך שההסתברות p תהיה טובה.

איך הגיעו למספרים שהגיעו? אין נוסחאת קסם, "ניסו וזה מה שהצליח".

איך מייצרים דוגמאות same (יותר קשה) ו־different (יותר קל)? הרי צריך מידע לאימון שיכיל גם וגם (אחרת תהיה לרשת נטיה לא מציאותית). לוקחים תמונה ועושים לה אוגמנטציה, והן same, ותמונות אחרי אוגמנטציה מקבוצה אחרת הם different.

2/1/2018

פרק חדש! דני יחזור לסיים את הנושא שלו עוד שבועיים, בינתיים נמשיך הלאה[5].

4.3 מודלים גנרטיביים

למעשה דברים שדני דיבר עליהם בשיעור קודם הם מודלים גנרטיביים, אך הם הוצגו בצורה שונה.
נזכר במה שראינו בשיעור קודם - ההבדל בין המודלים:

- מודלים דיסקרימינטיביים - מידול הגבול בין קלאסים. מודלים אלו לא צריכים לאפיין את הקבוצות השונות - כלבים או חתולים, אלא לדעת להפריד ביניהם.

[5] אולי אסדר את הסדר בהתאם בסיכום

- מוגלים גנרטיביים - מידול ההתפלגויות של הקלאסים השונים. במודלים אלו ניתן ליצר דוגמא בעלת הסתברות מסויימת - זו בעצם פעולת הדגימה: בהנתן $p(x)$, למצוא x מתאים.

שימושים של מודלים גנרטיביים:

- יצירת תמונות על ידי דגימה $I_i \sim p(I)$
- איטרפולציה ב־multi-modal domains
- שחזור תמונות בעזרת חוק בייס:

$$P(I|I_c) = \frac{P(I_c|I)}{P(I_c)} P(I) \, \alpha P(I_c|I) P(I)$$

למעשה זו הדרך שבה מבטלים טשטוש \ רעש וכו', שכן קל הרבה יותר למדל את $P(I_c|I)$. דוגמאות מספרית - נתחיל עם רעש:

$$I_c = I + \eta$$

כאשר כאן יש הנחה שהרעש מתפלג נורמלי: $\eta \sim \mathcal{N}\left(o, \sigma^2\right)$. אז:

$$P(I_c|I) = \frac{1}{z} e^{-\frac{\eta^T 2}{2\sigma^2}} = \frac{1}{z} e^{\frac{(I_c - I)^T (I_c - I)}{2\sigma^2}}$$

דוגמא שניה - טשטוש (+רעש):

$$I_c = I \cdot K + \eta$$
$$P(I_c|I) = e^{-\frac{(I_c - I \cdot k)^T (I_c - I \cdot K)}{2\sigma^2}}$$

4.3.1 הקושי בהתאמת התפלגות

באופן מסורתי, כאשר אנחנו ניגשים לבעיה הזו, נבחר משפחה של התפלגויות P_θ, למשל, מודלים גאוסיים:

$$P_\theta(x) = \frac{1}{\sqrt{\pi}} e^{-\frac{(x-\mu)^2 \sum (x-\mu)}{2}}, \; \theta = \left\{\mu, \sum\right\}$$

(הנוסחא מעלה ממש לא נכונה, צריך לתקן!) ונתאים אותה בעזרת maximum likelihood estimator:

$$\max_\theta \prod P_\theta(x_i)$$

נגיד בהנתן $f(I) \geq 0$,$\int f_\theta(I) < \alpha$:

$$P_\theta(I) = \frac{f_0(I)}{\int f_\theta(I) \, dI} = \frac{1}{Z_\theta} f_0(I)$$

נשים לב שעל מנת לחשב את הביטוי, צריך לבצע כאן את האינטגרציה הרבה מאוד פעמים (למעשה הפעולה היקרה היא מה שמסומן מעלה ב־Z_θ, שנקראית גם partition func.), וזו פעולה מאוד יקרה. כאן הביטוי הוא ללא נרמול, ולכן הבעיה. ננסה להבין כיצד ניתן לנרמל וכך לקצר את התהליך.

4.3.2 בחירת דגימה מתוך התפלגות

נשאלת השאלה, כיצד ניתן ליצר x מתוך $P(x)$? זהו תחום מאוד בסיסי בתחום, וחשוב מאוד להכיר אותו. על כן נקדיש לו לא מעט זמן, ונראה דוגמאות בהמשך השיעור (או בשיעור הבא בהתאם להספק). נפרמל מעט יותר את הניסוח שלנו:
"לדגום" - בהנתן $P(x)$, נרצה ליצר סדרה $\{x_i\}_{i=1}^{k/\infty}$, כאשר הסיכוי לדגום כל x_i הוא $p(x_i)$.

משתנה נורמלי חד מימדי נניח כי קיים $x \sim U[0,1]$[6], ולכן מתוך $P(x,y,z) = U(x) U(y) U(z)$.
אם נרצה לקבל משתנה שהוא חד מימדי אך גאוסיאני (נורמלי), נשתמש במשפט הגבול המרכזי ונקבל: $g = sum\{u_i\}$.

[6] מעשית משתמשים ב־$seed$, למשל משתמשים בתאריך כ־$seed$ או בטריק דומה.

משתנה נורמלי רב מימדי נניח שיודעים ליצר $x \sim \mathcal{N}\left(0, \sigma^2\right)$. איך ניצר דוגמאות ממשתנה רב מימדי?

$$\overline{x} = (x_1, x_2, ...)\,,\ x \sim \mathcal{N}\left(0, \sigma^2\right)$$

זו אפשרות סבירה, אבל במקרה זה מדובר מהתפלגות מאוד ספציפית:

$$P(x) = e^{-\frac{x^T x}{2\sigma^2}},\ x \sim \mathcal{N}\left(0, I \cdot \sigma^2\right)$$

כיצד נשנה למקרה כללי יותר? תחילה, נבצע הזזה כדי לתקן את ה־variance:

$$\overline{x} = (x_1 + \mu_1, x_2 + \mu_2, ...)$$

נבצע עוד שינוי:

$$y = A \cdot \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} + \begin{bmatrix} \mu_1 \\ \vdots \\ \mu_n \end{bmatrix},\ \overline{y} = A\overline{x} + \overline{y},\ \overline{x} = A^{-1}\overline{y} - A^{-1}\eta$$

$$P(y) = \frac{1}{z}e^{-\frac{x^T x}{2\sigma^2}} = \frac{1}{z}e^{-\frac{\left(A^{-1}(\overline{y}-\mu)\right)^T A^{-1}(\overline{y}-\mu)}{2\sigma^2}} = \frac{1}{z}e^{-\frac{(y-\mu)^T A^{-T} A^{-1}(y-\mu)}{2\sigma^2}}$$

ולכן:

$$y \sim \mathcal{N}\left(\mu, AA^T \cdot \sigma^2\right),\ \Sigma^{-1} = \frac{A^{-T}A^{-1}}{\sigma^2}$$

כאן אנחנו רואים כמה זה מורכב לעשות את התהליך עבור כל משפחה! על כן, אנו נלמד שיטות גנריות לעשות זאת.

התפלגות חד מימדית כללית תהי $P(x)$, נניח לנוחיותינו כי $P(x) > 0$ באיזשהוא אינטרוול $x \in [-1, 1]$. פונקצית ההתסברות המצטברת נתונה על ידי:

$$F(x) = \int_{-\infty}^{\infty} P(s)\, ds$$

והיא מונוטונית עולה, ועל כן F^{-1} קיימת. נשאל מהי:

$$(*) P_v\left(z < F^{-1}(u) < z + \varepsilon\right),\ u \sim U[0, 1]$$

כלומר, מהי הסתברות ליפול בין z ל־$z + \varepsilon$, דהיינו מי נמצא באינטרוול:

$$(*) = \int_z^{z+\varepsilon} p'(s)\, ds \xrightarrow{\varepsilon \to 0} \int_z^{z+\varepsilon} p'(z) + o(\varepsilon) = p'(z) \cdot \varepsilon + o\left(\varepsilon^2\right) = (**)$$

נחשב:

$$(*) = P_v\left(F(z) \le u \le F(z + \varepsilon)\right)$$

וזה כבר קל ־ מה הסיכוי שמשתנה יוניפורמי נופל בקטע מסויים? אורך הקטע!

$$= F(z + \varepsilon) - F(z) \overset{taylor}{=} F(z) + \varepsilon \cdot F'(z) + O\left(\varepsilon^2\right) - F(z) = \varepsilon \cdot F'(z) + o\left(\varepsilon^2\right) = \varepsilon \cdot p(z) + o\left(\varepsilon^2\right)$$
$$\overset{(**)}{\Rightarrow} p' = p$$

התפלגות רב מימדית כללית אנחנו לא יודעים לעשות :(לכן נעשה איזשהוא tradeoff: בעוד שיצור דגימות בלתי תלויות קשה, נשתמש בדגימות תלויות בעזרת שרשראות מרקוב. הטענה היא שמתישהו השרשרת מתכנסת ל־x־אים שמגיעים מההתפלגות הרצויה, האמיתית, אבל הן תלויות אחת בשניה.

4.3.3 שרשראות מרקוב מונטי קרלו $(MCMC)$

מבצעים צעדים קטנים תלויים:

$$x^{n+1} \sim T\left(x|x^n\right),\ s.t.\ x^\infty \sim p\left(x\right),\ \text{even if } x^0 \sim q\left(x\right) \text{or } x^0 = const$$

כאשר T היא ה־transition matrix. כמובן שמבחינה אלגוריתמית זה לא קשה לעשות.
T היא מטריצה סטוכסטית (כלומר כל עמודה מייצגת התפלגות), והיא חייבת לקיים את התנאים הבאים:

- לא פריקה irreducible: לכל x, y חייב להיות n כך ש:

$$\left(T^n\right)\left(x|y\right) > 0$$

כלומר היא אי שלילית.

- לא מחזורי - aperiodic:

$$\gcd\left\{n|\left(T^n\right)\left(x|y\right) > 0\right\} = 1$$

על מנת להיות מטריצה ארגודית (ergodic) - יש לה התפלגות סטציונרית יחידה (גם נקראת equalibrium distribution).
כלומר פונקצית ההסתברות של $p\left(x\right)$ מתכנסת להתפלגות יחידה (והיא הסטציונרית).
לאחר ששני התנאים הללו התקיימו, עדיין יש לנו בעיות:

- אין לנו שליטה לאיזו התפלגות סטציונרית אנחנו מתכנסים
- short mixing time - n בו x^n לא תלוי ב־x^0 קטן

כדי לכוון את ההתפלגות הסטציונרית תהיה זו שאנחנו רוצים, אנחנו משתמשים ב־detailed balance:

$$T\left(x|y\right)P\left(y\right) = T\left(y|x\right)P\left(x\right)$$

כאשר T לא מחזורי ולא פריק, ורוצים לדעת מהיא ההתפלגות הסטציונרית $p\left(x\right)$. נראה כי אם לכל x, y:

$$T\left(x|y\right)P\left(y\right) = T\left(y|x\right)P\left(x\right)$$

אזי P היא ההתפלגות הסטציונרית של T.

$$P'\left(y\right) = \sum_x T\left(y,x\right)P\left(x\right) = \sum_x T\left(x,y\right)\cdot P\left(y\right) \overset{T(x,y)=P(x|y)}{=} P\left(y\right)\cdot\overbrace{\sum_x T\left(x|y\right)}^{=1} = P\left(y\right)$$

כלומר קיבלנו כי במקרה זה $P'\left(y\right) = P\left(y\right)$, ולכן אם ה־detailed balance מתקיים, סיימנו!
נראה שתי בניות קנוניות, שבהנתן P, גורמות להתקיימות שלושת התנאים (השניים הקודמים + detailed balance), דהיינו נוכל להשתמש בכל אחת מהבניות הללו.

Gibbs Sampler לכל ציר:

$$T\left(y|x\right) = P\left(y_i, x_{-i}|x\right) = \frac{p\left(y_i, x_{-i}\right)}{\int p\left(y, x_{-i}\right)dy}$$