digital whisper
Technologic Papers

רשתות נוירונים למשוענממים בלבד

מאת ספיר פדרובסקי

הקדמה

כולנו מכירים את ChatGPT, חלקנו משתמשים בזיהוי פנים בטלפון, מחזיקים תעודת זהות ביומטרית, מנסים לזהות פוגענים עם מודלים חכמים או מג'נרטים איזו תמונה מוזרה ב-midjourney. או בקיצור, אני לא חושבת שיש צורך בימים אלו להסביר את ההייפ סביב מכונות חכמות.

במאמר הזה אני אתמקד בבסיס של רשתות נוירונים (שהן הבסיס לכל הדוגמאות שנתתי בשורה הראשונה, ולעוד הרבה יכולות אחרות), מהפן התיאורטי והמתמטי, לאחר מכן אני אסביר על כמה מתקפות נגד רשתות נוירונים, גם כאן, מכיוון יותר תיאורטי ולבסוף קצת הגנות.

הסיבה שבחרתי לגשת לנושא הזה מהפן התיאורטי ולא המעשי, מעבר לזה שיש לי חיבה לא מוסברת למתמטיקה, היא שהיום אפשר לבנות מודל ב-40 שורות (הנה). המודל הזה מצליח ב-97% לנבא את הדוגמא שנעבוד איתה לאורך כל המאמר:

```
import tensorflow as tf import tensorflow_datasets as tfds
(ds_train, ds_test), ds_info = tfds.load(
    'mnist',
    split=['train', 'test'],
    shuffle_files=True,
    as_supervised=True,
    with_info=True,
)
def normalize_img(image, label):
  """Normalizes images: `uint8` -> `float32`."""   return tf.cast(image, tf.float32) / 255., label

ds_train = ds_train.map(
    normalize_img, num_parallel_calls=tf.data.AUTOTUNE)
ds_train = ds_train.cache()
ds_train = ds_train.shuffle(ds_info.splits['train'].num_examples)
ds_train = ds_train.batch(128)
ds_train = ds_train.prefetch(tf.data.AUTOTUNE)
ds_test = ds_test.map(
    normalize_img, num_parallel_calls=tf.data.AUTOTUNE)
ds_test = ds_test.batch(128)
ds_test = ds_test.cache()
ds_test = ds_test.prefetch(tf.data.AUTOTUNE)
model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dense(10)
])
model.compile(
    optimizer=tf.keras.optimizers.Adam(0.001),
```

```
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
    metrics=[tf.keras.metrics.SparseCategoricalAccuracy()],
)

model.fit(
    ds_train,
    epochs=6,
    validation_data=ds_test,
)
```

ובשביל זה לא צריך מאמר. אבל כדי לשפר את המודל, או לבנות דברים מסובכים יותר, צריך להבין מה הולך שם. וגם, מה כיף בזה? צריך לסבול קצת.

חשוב להגיד, הטכנולוגיה שאציג במאמר היא כבר old news, אבל היא בהחלט מהווה את הבסיס למה שקורה היום.

בנוסף, אני אתמקד ברשתות המשמשות לזיהוי תמונה, אבל כמובן ש-Input יכול להיות גם טקסט או שמע או כל דבר שמומר לווקטור מספרי.

משום שאני מתמקדת בזיהוי תמונה, אני אוסיף קצת חלקים הקשורים לעבודה עם תמונה כמטריצה ופעולות שניתן לבצע על מטריצות של תמונות.

- המאמר כנראה מתאים לאנשים סקרנים ואולי קצת משועממים שיודעים טיפה מתמטיקה.

כמה מילים על רשת נוירונים

הקונספט של רשת נוירונים היא בעצם שכבות של נוירונים, כאשר כל שכבה מקושרת לשכבה הבאה באמצעות משקולות שקובעות את מידת ההשפעה שיש לנוירונים בשכבה X לנוירונים שהם קשורים אליהם בשכבה X+1.

האם זה באמת דומה לאיך שהמוח האנושי עובד? אין לי מושג אבל יש הטוענים שקצת. לרשת יש שכבת Input, זאת אומרת, ה-Data שהרשת קיבלה, ושכבת Output, שהיא בעצם הפלט של הרשת לגבי ה-Input שהיא קיבלה. (זדוני או לא זדוני, שור נמר או חתול, ישבן של קורגי או כיכר לחם - ממליצה לחפש באינטרנט בחום).

#התחלנו

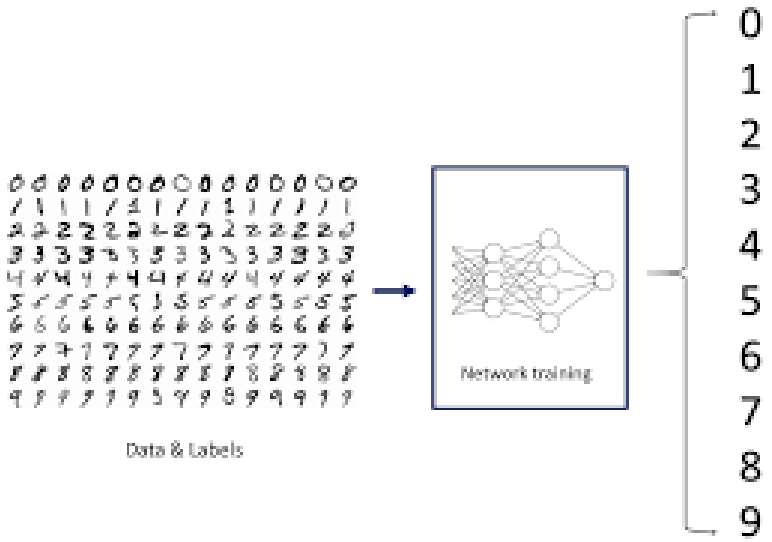
לצורך הדוגמא אנחנו נדבר על רשת נוירונים שתפקידה לקבל תמונה של ספרה שנכתבה ביד אדם (זה בעצם ה-Input של הרשת), ה-Output של הרשת הוא מה המודל חושב שהספרה הכתובה בתמונה.

נשתמש במאגר MNIST המכיל 60,000 דוגמאות של ספרות שנכתבו ביד אדם בצורות שונות.

ה-Input וה-Output שלנו יהיו:

1. קבלת תמונה של ספרה
2. החזרת הספרה שהוא חושב שנמצאת בתמונה

משהו כזה:



הבנו מה אמורה להיות שכבת ה-Input, מה זה אומר בעצם? המידע שהרשת שלנו מקבלת. המידע יכול להיות כל דבר, אבל הוא כמובן יוצג בצורה מספרית.

נמשיך עם הדוגמא, נחשוב על תמונה.

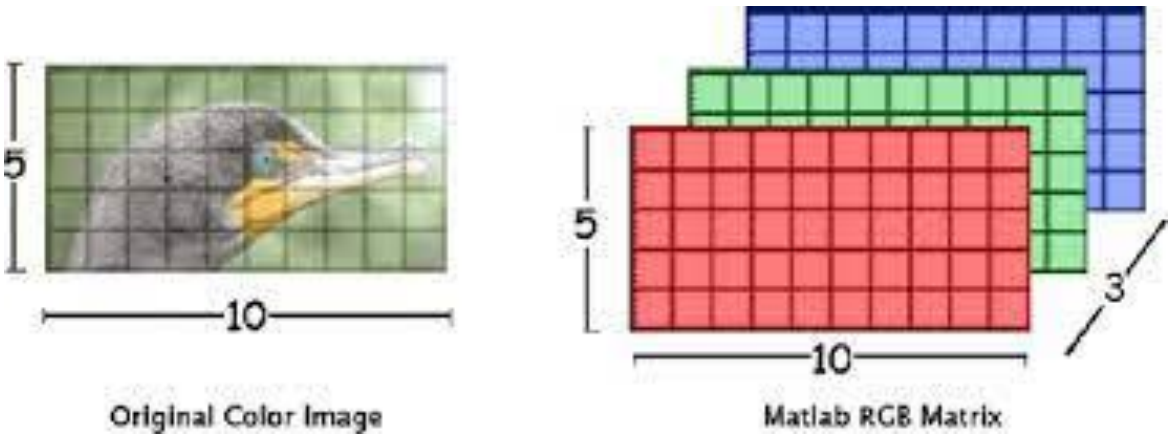
(בדרך כלל זה)

תמונה המיוצגת באמצעות מטריצה של MXNX3 כאשר 3 זה ערכי ה-RGB. מה זה אומר? אנחנו בעצם מייצגים את הפיקסלים בהתבסס על RGB - אדום, ירוק וכחול. אם היינו משתמשים בתמונה שחור לבן, היינו יכולים לייצג את הפיקסלים בצורה חד מימדית בין 0 (שחור) ל 255 (לבן)וזה היה מספיק.

מספר השורות והעמודות במטריצה היו מתייחסים לגודל התמונה.

אך משום שכיום רוב התמונות מכילות יותר משחור ולבן, נוהגים לייצג אותם באמצעות מטריצות 3MXNX (כך שבכל בלוק יהיה את אותו מספר שורות ועמודות, ומספרים בין 0 ל 255) שבעצם ייצגו את הטווח בין הצבעים אדום ירוק וכחול והשילוב שלהם ייצור תמונה.

הנה תמונה כדי להבין את הקונספט:



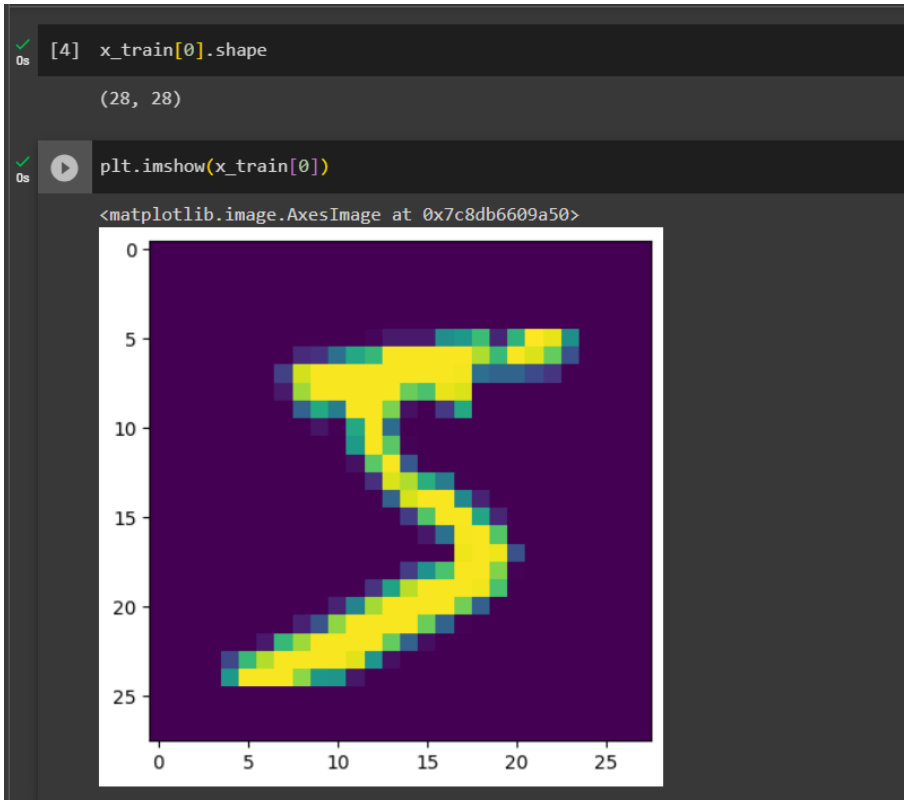
אז אם כך, איך יראה בעצם ה-Input שלנו?

בדרך כלל מה שנעשה זה להפוך את המטרייצת MXNX3 הזו לווקטור ארוך אחד.

(חזרה לדוגמא שלנו)

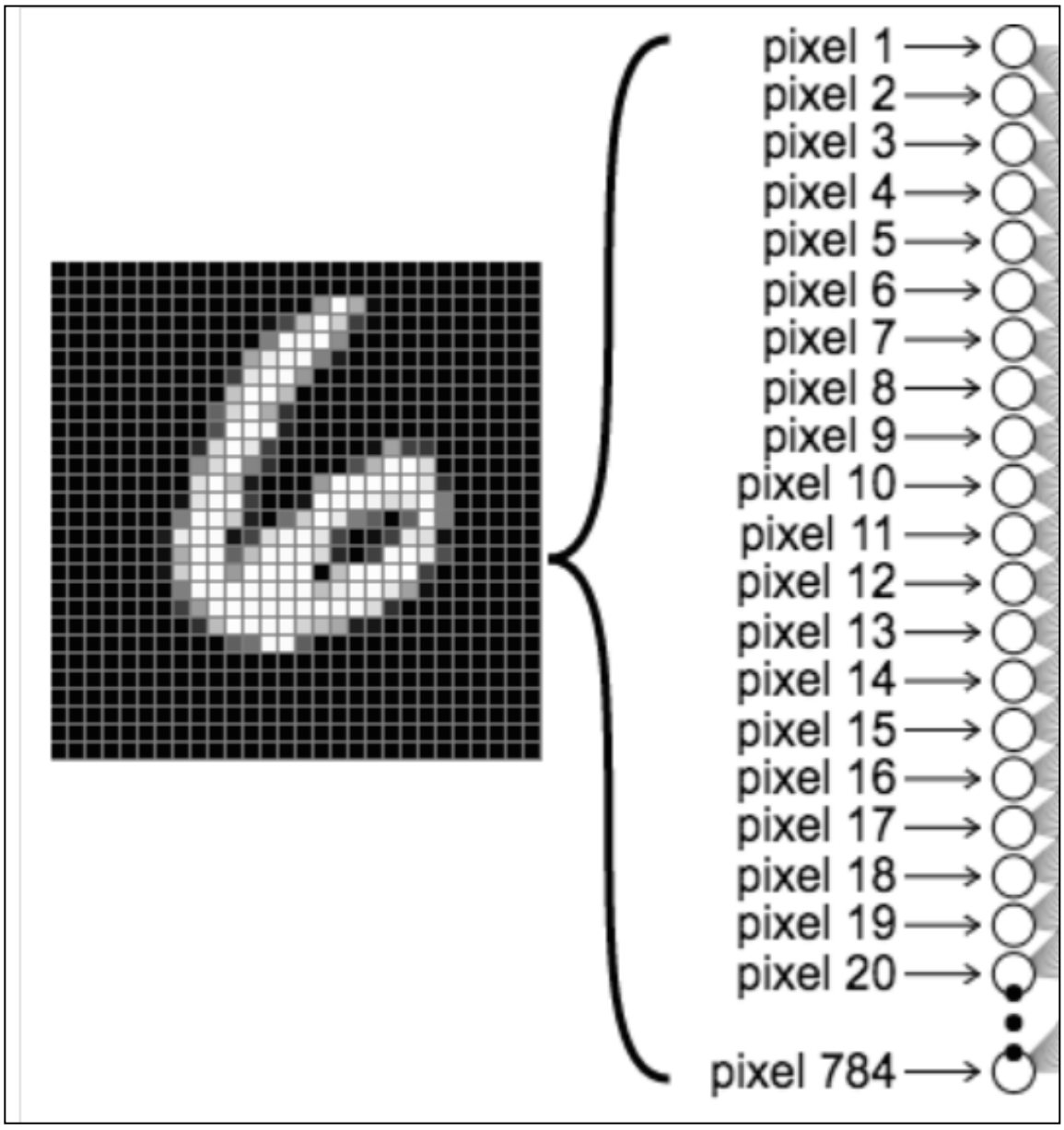
משום שמדובר סך הכל בתמונות של ספרות, אין סיבה שנצטרך צבעים, ולכן אנחנו מדברים על תמונת שחור לבן בגודל 28x28 (כי זה הגודל של התמונות ב-MNIST).

איך זה בעצם נראה?



(התמונה היא שחור לבן אבל משום שאני על dark Mode קיבלנו אפקט קצת צבעוני). זאת אומרת של-Input הזה אנחנו מצפים ל-Output שהוא הספרה 5.

ואיך זה יראה כ-Input של רשת?

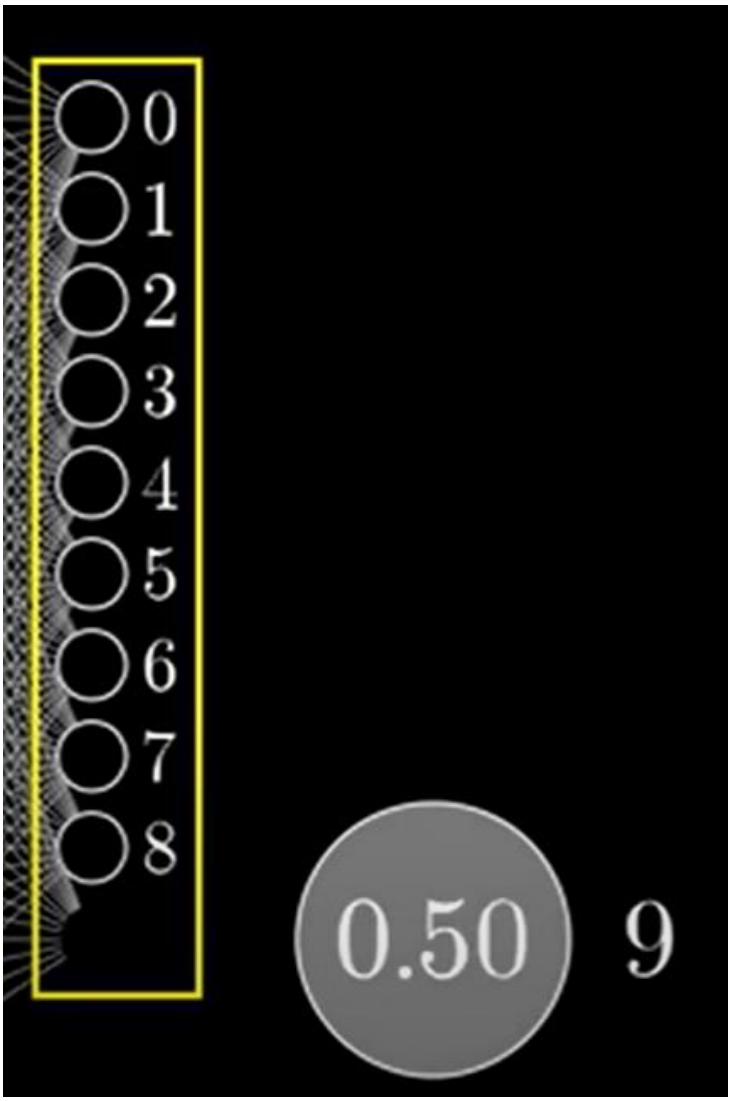


זאת אומרת שיהיה לנו וקטור אחד באורך 784 (28x28) שכל מספר בו יהיה בין 0 ל-255 וייצג פיקסל. כפי שהסברנו קודם, רשת נוירונים מורכבת משכבות שונות של נוירונים. אז בשכבה הראשונה, ה-Input, כל נוירון בעצם מייצג פיקסל בתמונה.

נקפוץ רגע לסוף, מה יוצא לנו מכל הדבר הזה?

בתרחיש שלנו, המודל צריך לפלוט החוצה ספרה שהוא חושב שהיא הספרה שבתמונה. איך זה נראה בפועל? פשוט מאוד, עשרה נוירונים! הראשון מייצג 0 והאחרון מייצג 9 וכך כל השאר בהתאמה לספרות שביניהם.

ואיך נדע מה המודל החזיר? הפלט יראה משהו כזה:



כל אחד מה-נוירונים בשכבה האחרונה, יכיל מספר בין 0 ל-1. ככל שהמספר גדול יותר, כך המודל יותר בטוח שזו הספרה שבתמונה. במודל מושלם - אם הספרה בתמונה היא 5. כל הנוירונים פרט לנוירון שמייצג את הספרה 5 יהיו 0, והנוירון המייצג את הספרה 5 יהיה 1.

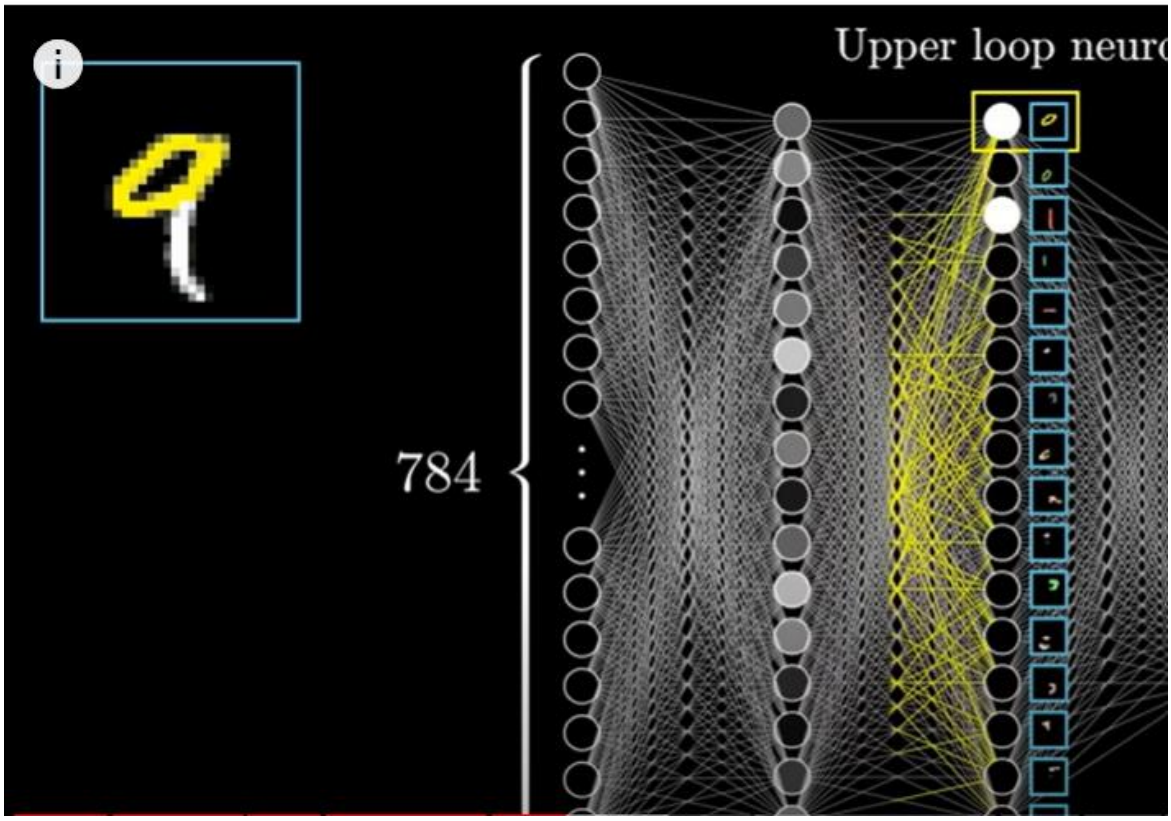
אבל יכול להיות שרמת הודאות של המודל שלנו לא תהיה עד כדי כך מוצלחת, ולכן במקרה שאין תשובה ודאית, הנוירון עם המספר הגבוה ביותר יבחר.

ולבסוף - יש לנו את ה "בפנוכו", כל השכבות הפנימית של הרשת. שנקראות גם "hidden layers" ונדבר עליהן בהמשך. אבל בגדול, כל שכבה שכזו תשפיע על הבאה בתור, ונוכל לשחק איתן ולעצב אותן בדרכים שונות כדי לשפר את המודל שלנו.

מה הקטע של השכבות? (הסבר לא נכון אבל עוזר)

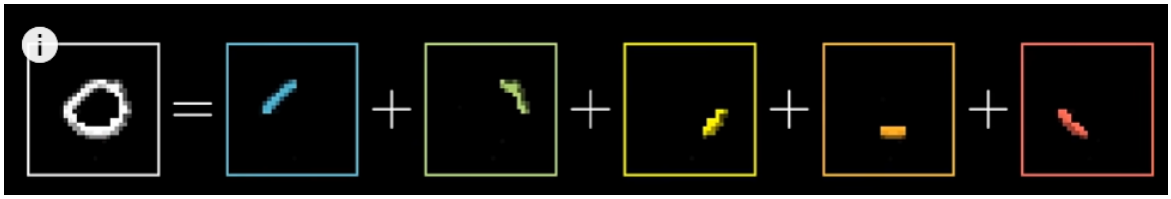
ההסבר הזה הוא ממש לא נכון, אבל לפעמים זה נחמד שיש משהו לדמיין. לצורך העניין בואו נפרק את הספרה 9. נוכל להסכים שהיא בדרך כלל מורכבת מעיגול וקו:

כעת, בואו נדמיין משהו כזה, נניח שכל נוירון בשכבה האחת לפני אחרונה שלנו מייצג "חתיכה" מספרה (עיגול, קו ישר, קו מעוגל, קו אופקי...). במקרה שלנו ברגע שהרשת תראה את הספרה 9, הנוירונים שמייצגים את החתיכות שלה "ידלקו" (זאת אומרת שערכם יהיה מספר קרוב ל-1):

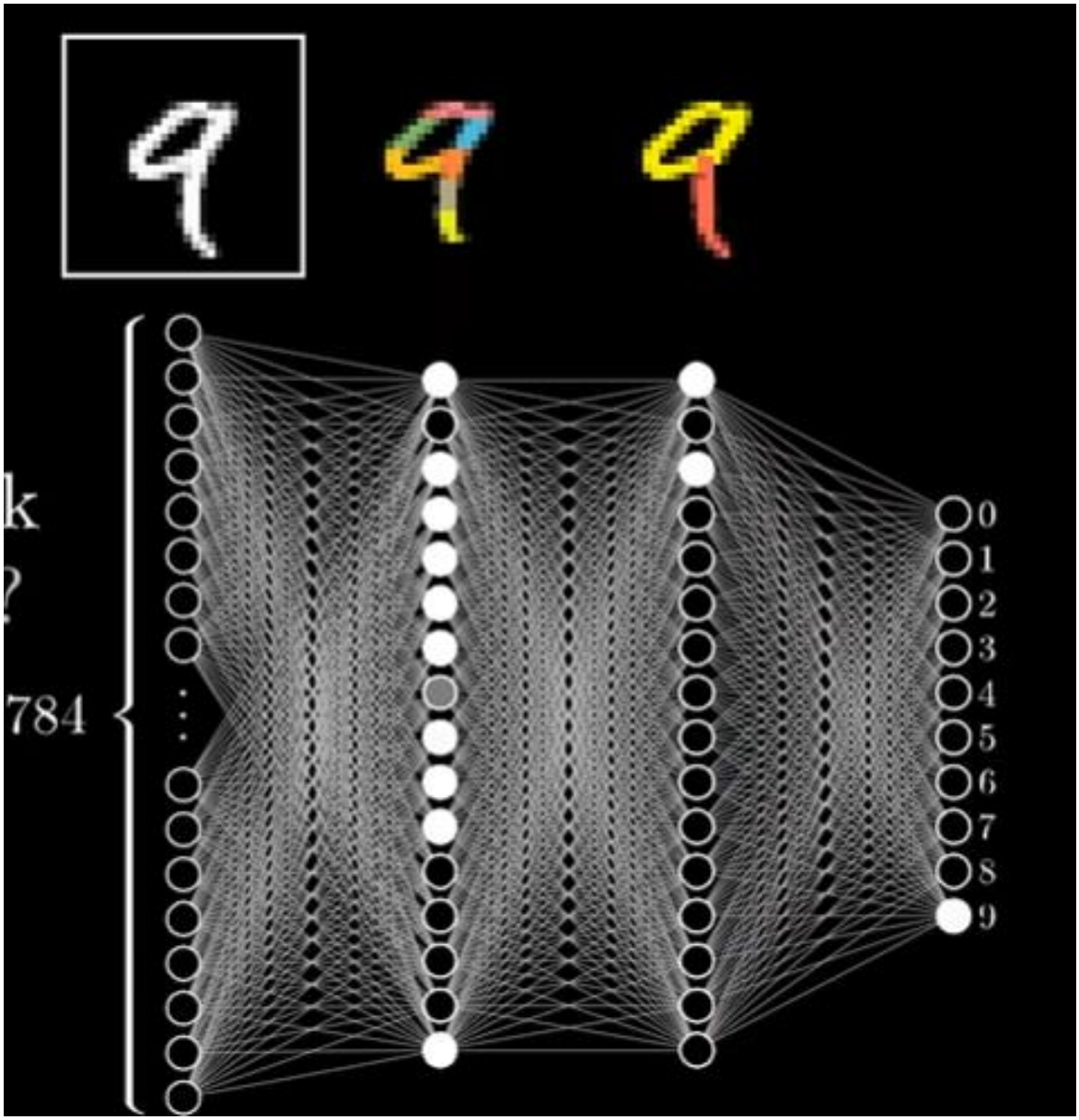


אם כך המצב, כל מה שהרשת תצטרך כדי לעבור מהשכבה הזו לשכבה האחרונה היא ללמוד אילו קומבינציות יוצרים אלו ספרות. פשוט!

אבל... איך הרשת מזהה את העיגול של ה-9? אז אולי השכבה שלפני, יודעת "להדליק" את כל הנוירונים שיוצרים עיגול:



ואז הרשת תראה בערך ככה:



כפי שציינתי הדבר הזה אפילו לא קרוב לבאמת להסביר את הסיטואציה, אבל זה עוזר קונספטואלית ואנחנו כאן כדי להכיר את הבסיס אז לדעתי זה בסדר ☺

הנפשות הפועלות

1. שכבות - לפחות Input, Output ושכבה אחת באמצע (בדרך כלל זה כמובן יותר משכבה אחת)
2. נוירונים, בכל שכבה יכול להיות מספר שונה של נוירונים
3. משקלים - כפי שראינו בתמונות, כל שכבת נוירונים מחוברת לבאה בתור באמצעות קווים, הקווים האלו מייצגים משקלים,מטרת המשקלים היא לייצג את מידת ההשפעה של הנוירון הנוכחי על הנוירון המחובר אליו בשכבה הבא. מיד נרחיב עליהם

Forward Propagation

נתחיל מהשלב הראשון, שנקרא גם Forward Propagation:

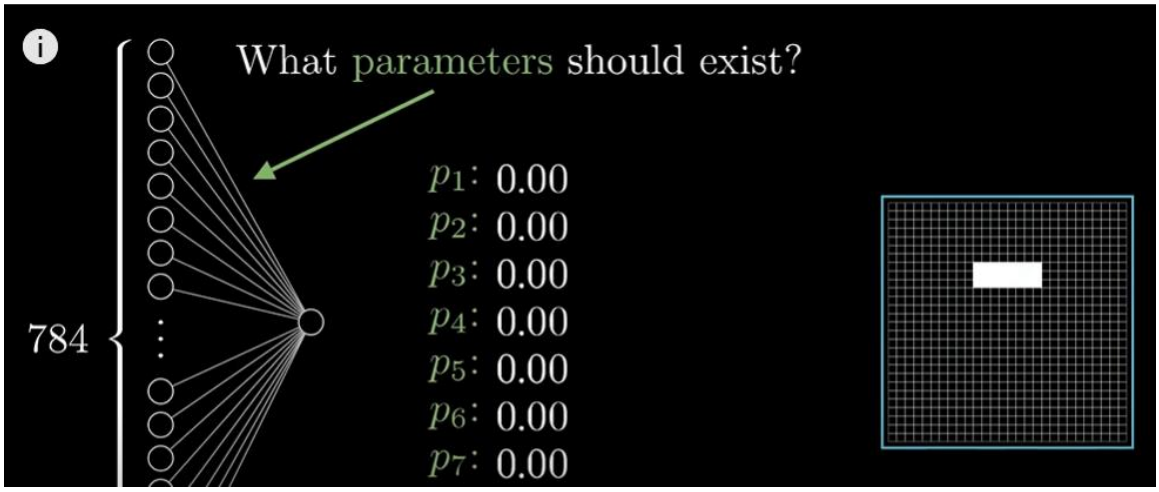
- הרשת תתחיל עם ערכי משקולות אקראיים
- הפעולות הבאות יבוצעו על נוירון אחד לצורך הדגמה, אך יש לזכור שהן מבוצעות על כל נוירון ברשת!

נדמיין את המצב הבא:

בשכבה הראשונה קיבלנו תמונה של ספרה בגודל 28X28, אז יש לנו רשת ששכבת ה-Input שלה מכילה 784 נוירונים.

כעת נתמקד בנוירון בודד בשכבה השנייה:

נניח שמטרת הנוירון הזה היא להבין האם קיים קו אופקי באזור מסוים.



כפי שניתן לראות בתמונה, הנוירון בשכבה השנייה שלנו מקושר באמצעות קווים להרבה נוירונים מהשכבה הראשונה. מה שנעשה, זה בעצם לבחור משקל (מספר, יכול להיות שלילי או חיובי). כל קו המקשר בין נוירון משכבה X לנוירון משכבה X+1 הוא בעצם בעל משקל כלשהוא.

המשקולות יעזרו לנו לזהות האם הקו האופקי שלנו אכן קיים. איך?

אם למשל ניתן משקלים חיוביים רק לנוירונים שמייצגים את הפיקסלים באזור זה, ולשאר המשקלים נשים מספר קרוב ל- 0 מלמטה, נוכל לזהות את האזור המבוקש. איך?

הפעולה שאנו עושים בין כל שכבה היא כזו (אל תשכחו שהדוגמא היא רק על קשרים לנוירון אחד, אבל היא תתבצע על כל נוירונים בשכבה X+1 שמחובר לנוירון בשכבה X):

נסכום את החישוב הבא: נוירון * משקל (הפעולה היא כפל).

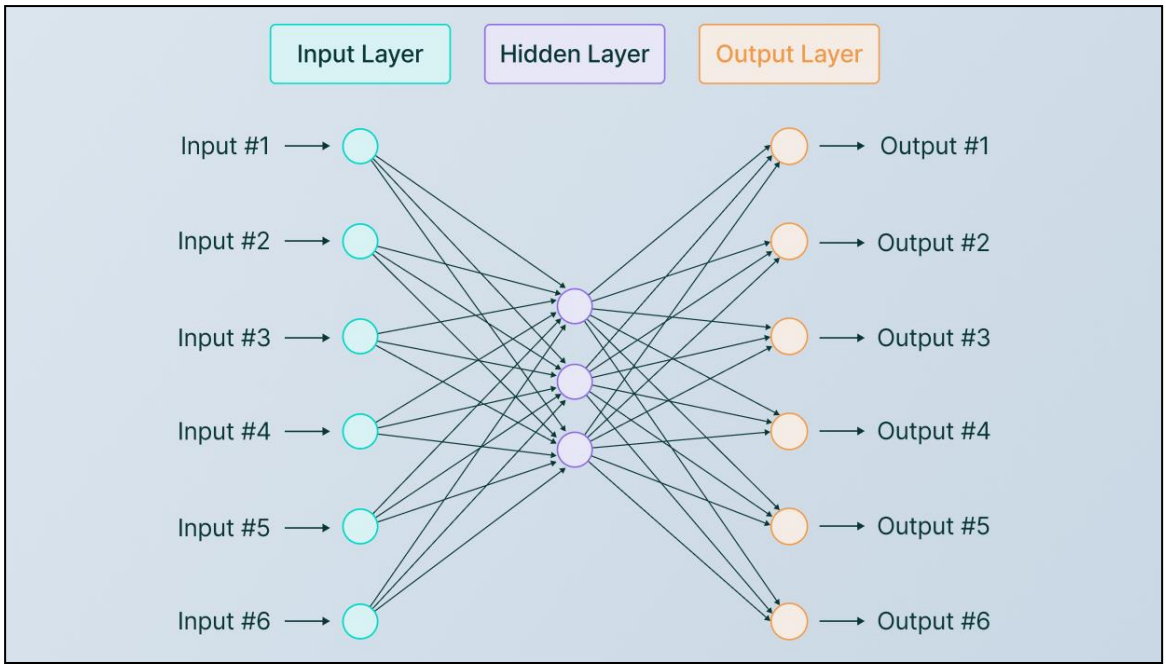
עד כה יש לנו את הנוסחה הבאה:

$$\sum \ (x_1 * \ w_1) + (x_2 * \ w_2) + .. + (x_n * \ w_n)$$

- הסימון - X מייצג את הנוירון במקום ה-i
- הסימון - W מייצג את המשקל במקום ה-i שמחבר בין הנוירון X במקום ה-i לנוירון בשכבה הבאה

נוכל להבין שלנוירון יכולות להיות הרבה משקולות שיוצאות ממנו, שכן כל משקולת מקשרת אותו לנוירון אחר בשכבה הבאה, הנוירון יכול להיות קשור לכולם, חלקם ואפילו בטכניקות מסוימות לשיפור המודל - לאף נוירון בשכבה הבאה.

עוד תמונה להמחשה:



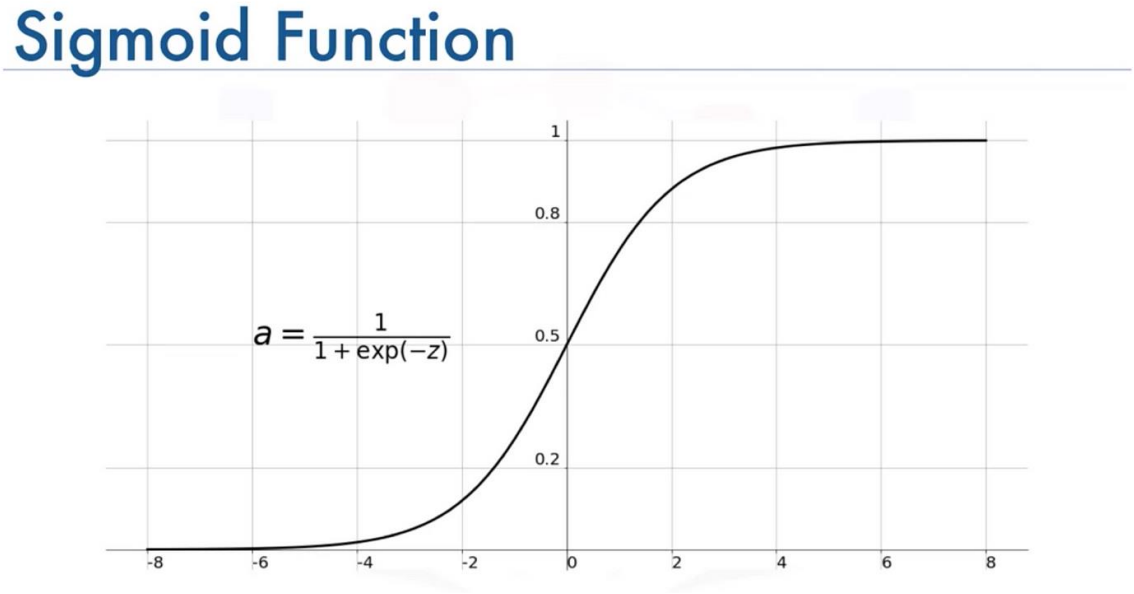
נמשיך, על הסכום שלנו אנחנו הולכים להפעיל פונקציה, שתקרא גם פונקציית אקטיבציה.

הסיבה שאנחנו עושים את זה היא כדי לקחת את התוצאה שלנו ולהמיר אותה לערך בין 0 ל-1.

סיבה נוספת היא, כדי להפוך את הפונקציה שלנו ל-לא לינארית, לשם כך, נפעיל עליה פונקציה לא לינארית.

הסיבה שאנחנו רוצים שהפונקציה שלנו תהיה לא לינארית היא כי כך היא תוכל להתאים את עצמה הרבה יותר טוב. במידה והפונקציה שלנו תהיה לניארית, כל המודל בסוף יהיה לינארי, שכן שרשור של לינארי עם לינארי זה לינארי גם כן.

יש המון פונקציות אקטיבציה כמו ReLU, Tanh, softmax, אנחנו נתייחס ל-Sigmoid:



- תהיה מסומנת בהרבה נוסחאות כך: σ

הפונקציה הזו בעצם תוחמת כל ערך בין מינוס אינסוף לפלוס אינסוף לערכים בין 0 ל-1.

ככל שהמספר יותר שלילי כך הוא יהיה קרוב יותר ל-0, וככל שהוא יותר גדול כך הוא יהיה קרוב יותר ל-1.

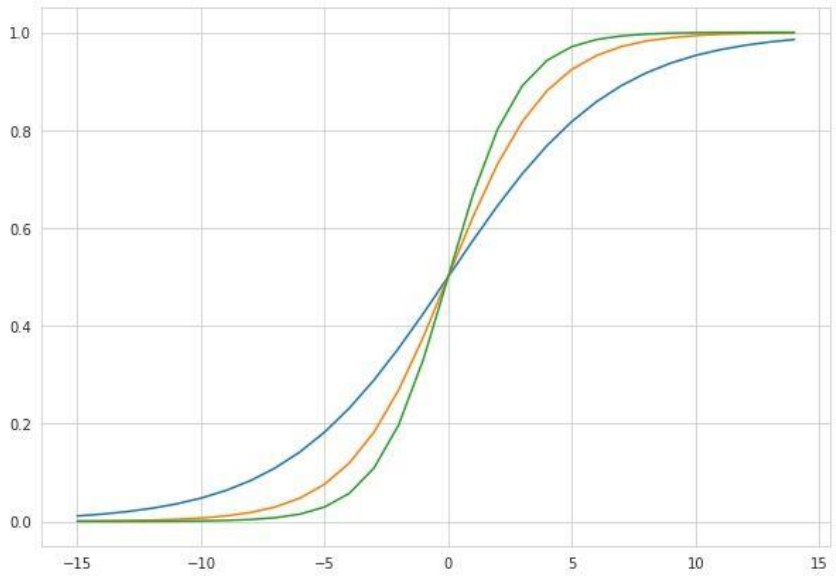
לכן, באמצעות הפעלה הפונקציה הזו נוכל בעצם לדעת כמה חיובי סכום המשקולות והנוירונים.

הנה שוב הפונקציה:

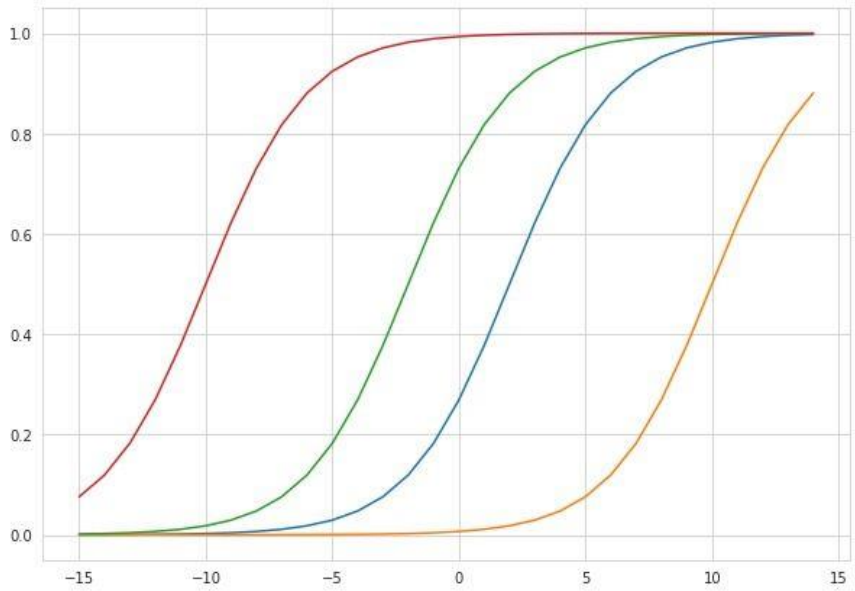
$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

במקרה שלנו הX- הוא הסכום שחישבנו קודם (משקל כפול נוירון) - כמעט.

למה כמעט? באמצעות שינוי ערך המשקלים שלנו נוכל להזיז את הפונקציה בצורה כזו:



זאת אומרת שאנחנו משנים רק את חדות העקומה שלנו, אבל מה אם נרצה להזיז אותה בצורה כזו?



אה! אז בשביל זה יש לנו bias. מדובר בעצם בערך קבוע שנוסיף לסכום שלנו (לפני הפעלת פונקציית האקטיבציה), ובאמצעותו נוכל להזיז את הפונקציה שלנו ימינה ושמאלה.

אגב, למה בכלל שנרצה להזיז את הפונקציה? בלי ה-bias, הנוירון "ידלק" אם הערכים שלנו גדולים מ-0. אבל מה אם לא אכפת לנו מ-0? מה אם נרצה שהנוירון "ידלק" רק אם הערכים גדולים מ-10? פה נכנס השימוש ב-bias בעצם.

אז הנוסחה הסופית שלנו היא:

Step 1

$$y = w_1 * x_1 + w_2 * x_2 + .. + w_n * x_n + bias$$

$$y = \Sigma_{i=1}^{n}(w_i * x_i) + b$$

Step 2

$$z = y_{pred} = \frac{1}{1 + e^{-(w_1 * x_1 + w_2 * x_2 + .. + w_n * x_n + b)}}$$

[הרציניים אוהבים להציג את זה עם כפל מטריצות, בחרתי לחסוך לכם את התענוג הזה]

כל זה, היא רק הפעולה הראשונה, והדגמנו אותה רק עבור נוירון אחד.

הדבר הזה מתבצע בין כל הנוירונים המקושרים ברשת! לכל נוירון יהיה bias משלו, ומשקלים משלו.

אז לצורך העניין נגיד שהרשת שלנו מורכבת מ-784 נוירונים בשכבה הראשונה, ועוד 2 שכבות פנימיות עם 16 נוירונים בכל אחת, ושכבה אחרונה עם 10 נוירונים. ונניח גם שכל הנוירונים בשכבה X מחוברים לכל הנוירונים בשכבה X+1 (זה כמובן לא תמיד המצב) גודל הרשת יהיה כזה:

784 x 16 + 16

נוירונים בשכבה הראשונה כפול נוירונים בשכבה השנייה בתוספת ה-Bias לכל נוירון בשכבה השנייה:

+

16 x 16 + 16

נוירונים בשכבה השנייה כפול נוירונים בשכבה השלישית בתוספת ה-Bias לכל נוירון בשכבה השלישית:

+

16 x 10 + 10

נוירונים בשכבה השלישית כפול נוירונים בשכבה האחרונה בתוספת ה-Bias לכל נוירון בשכבה האחרונה.

והנה בשביל הרשת הכי פשוטה בעולם שהיום עובדת באיזה הצלחה של 99 אחוז קיבלנו רשת בגודל 13,002. מטורף, לא?

עם כל משקל כזה, וכל bias, נוכל לשחק כדי להתאים את הרשת שלנו ולקבל את התוצאה המדויקת ביותר.

אגב - השלב השני מסובך יותר ☺

כפי שראינו, לחלק הראשון קראו Forward Propagation, משום שבחלק הזה התחלנו מה-Input וחישבנו את ה-Output, הלכנו "קדימה" ברשת שלנו. עד כה, המכונה שלנו לא למדה כלום, היא רק חישבה דברים. החלק השני נקרא, Backward Propagation, ואני אשאיר לכם להבין למה. לפני שנצלול ליופי המתמטי הזה, כדאי שנכיר כמה דברים:

איך נראה בעצם תהליך בניית הרשת?

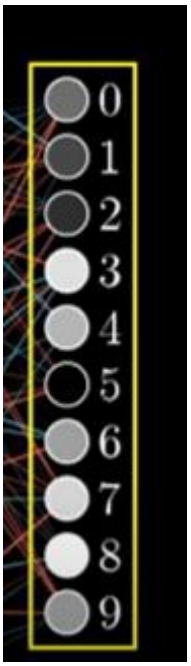
כפי שציינתי, אנחנו נגדיר את מספר השכבות, כמות הנוירונים בכל שכבה והחיבורים בין כל שכבה. כמו כן אנחנו מעבירים את שכבת ה-Input אז אנחנו יודעים מה יהיו ערכי הנוירונים בה בכל Input. אבל מעבר לזה, אנחנו לא שולטים על כלום.

זאת אומרת, תחילה, המודל נותן ערכים רנדומליים לנוירונים והמשקולות בשכבות השונות.

אז נניח והתחלנו לרוץ, יש לנו שכבות, יש נוירונים, יש משקלים, נתנו למודל וקטור מספרי המיצג תמונה של הספרה 3, ונתנו לו להריץ את הפונקציה שהסברנו קודם (תהליך ה-Forward Propagation).

אנחנו מצפים שהנוירון בשכבת ה-Output המייצג את הספרה 3 יהיה בעל ערך קרוב ל-1 ושאר הנוירונים יהיו בעלי ערך קרוב ל-0, נכון?

אבל למה שזה יקרה אם הרשת מלאה בערכים רנדומליים? אז זה לא יקרה. אנחנו נקבל איזו שטות מבולבלת.



עכשיו שאלת השאלות, איך המודל יודע שהוא טעה? ואיך הוא משתפר (לומד)?

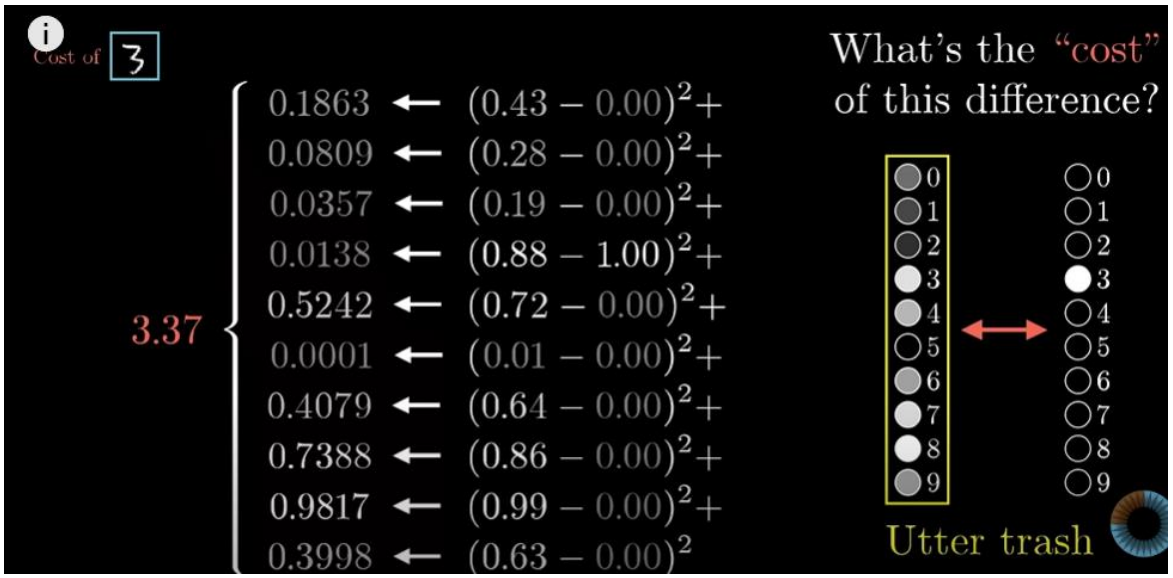
אנחנו מגדירים משהו שנקרא Cost function, מטרת הפונקציה הזו היא להגיד למודל שהוא טעה.

- אם יצא לכם ללמוד או לקרוא פעם בתחום, אתם בטח אומרים, רגע זה לא loss function?

אז, loss function מתייחסת למקרה בודד (הפרש הטעות בריבוע), Cost function היא פונקצית חישוב הטעות הכללית. מיד הכל יתברר.

איך היא נראית?

ניקח את הערכים שהמודל שלנו הוציא (שטות כלשהי), וניקח את הערך שהוא היה אמור להוציא (0 בכל הנוירונים ו-1 בנוירון שמיצג את הספרה 3), נחשב את סכום הריבועים של ההפרש ביניהם:



אם הרשת הצליחה לסווג את הספרה בצורה טובה, הסכום הזה יתקרב ל-0, ככל שהרשת תוציא תוצאה לא טובה, הסכום הזה יגדל.

ולכן - אנחנו רוצים שהערך של פונקצית ה-cost יהיה כמה שיותר נמוך!

את הסכום הזה, אנו מחשבים עבור כל ספרה באימון.

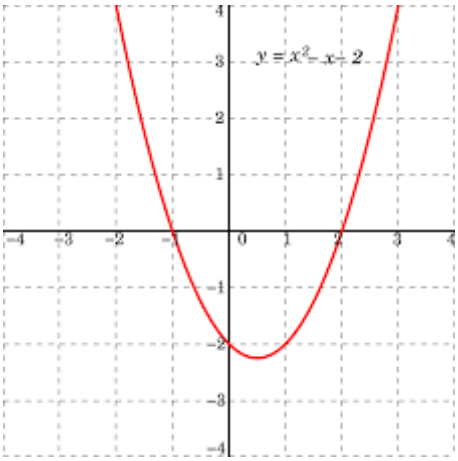
אז אם אנחנו מאמינים את המודל שלנו על 40,000 תמונות, יהיו לנו 40,000 סכומים שכאלו שמייצגים את ההצלחה של המודל בכל פעם. מה אנחנו עושים איתם?

מחשבים את הממוצע שלהם. זו היא האינדיקציה של המודל לכמה הוא היה גרוע.

אז המודל עכשיו רץ על 40,000 תמונות, עם ערכים אקראיים במשקולות וב-bias, עכשיו הוא יודע שהוא גרוע, מה הוא עושה עם המידע הזה?

כדי להבין את קונספט ההשתפרות, בואו נחשוב פשוט.

נניח וזו פונקצית ה-Cost שלנו, פונקציה מחייכת פשוטה, מקבלים X ומחזירים Y.



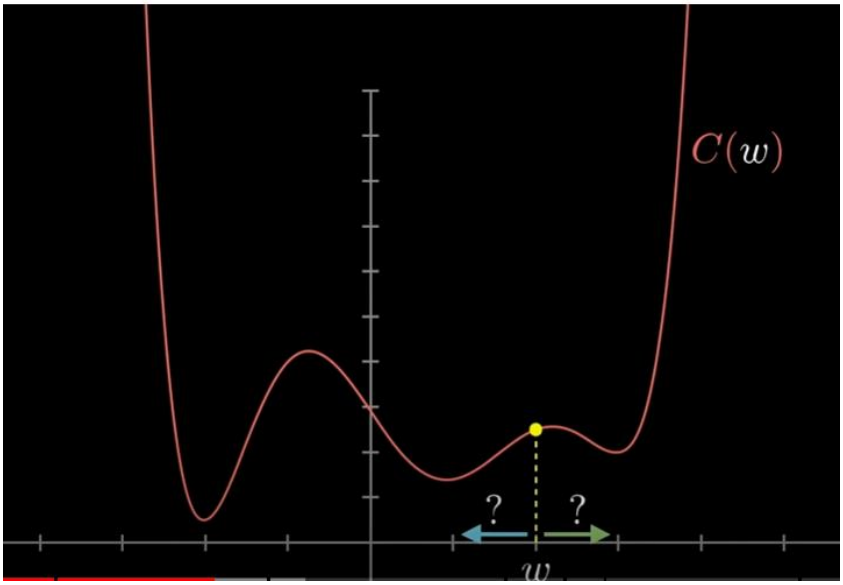
אם נסתכל על הפונקציה הזו, בידיעה שהתוצאה של פונקצית ה-cost שלנו צריכה להיות כמה שיותר נמוכה (כי זה מה שמעיד על מודל יותר מדויק), הרי ברור לחלוטין מה אנחנו צריכים לעשות כאן!

איך מוצאים את ערך ה-X שיחזיר את ה-Y הכי קטן? נקודת מינימום!

(וכאן כולנו יכולים להפסיק להתלונן שמה שלמדנו בתואר לא עוזר בכלום, כי הנה היא, נגזרת!)

- חשוב לציין שהפונקציה שלנו יכולה להיות הרבה יותר מסובכת מהפונקציה הזו, ואם למדנו מתמטיקה אנחנו יודעים שלפונקציה יכולות להיות נקודות קיצון לוקאליות, שהן לא הנמוכות ביותר לפונקציה.

אז מה בעצם אנחנו עושים מאחורי הקלעים? הרצנו את המודל פעם אחת על כל הדוגמאות וקיבלנו פונקציית cost, נניח שזו הפונקציה שלנו:

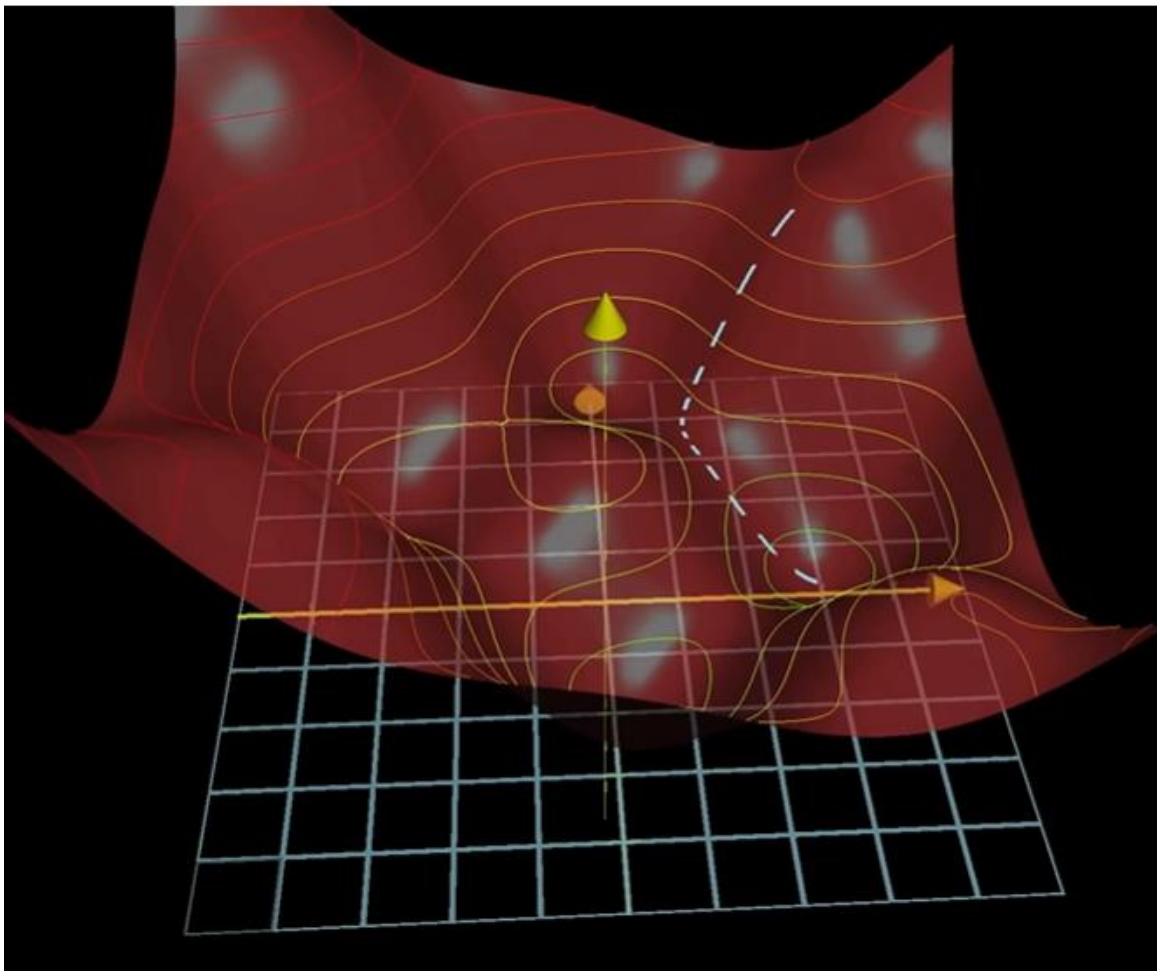


כעת, עלינו להבין לאן עלינו לזוז על מנת להקטין את הערך של הפונקציה.

כיצד? אם נוכל לחשב את השיפוע (אהמ אהמ... נגזרת) בנקודה בה אנו נמצאים, נוכל לדעת לאיזה כיוון עלינו לזוז ! (זאת אומרת, האם עלינו להגדיל או להקטין).

אם נעשה את הפעולה הזו שוב ושוב, וכל פעם נזוז מטה בהתאם לשיפוע (מימין או משמאל), בסוף נגיע לנקודת מינימום לוקאלית של הפונקציה.

רק נקודה פצפונת בנוגע לזה, הפונקצית cost שלנו תראה פחות כמו בדוגמא ויותר.. ככה (וגם זה לא, כי בדוגמא שלנו למשל יש לה 13,002 מימדים... ☹

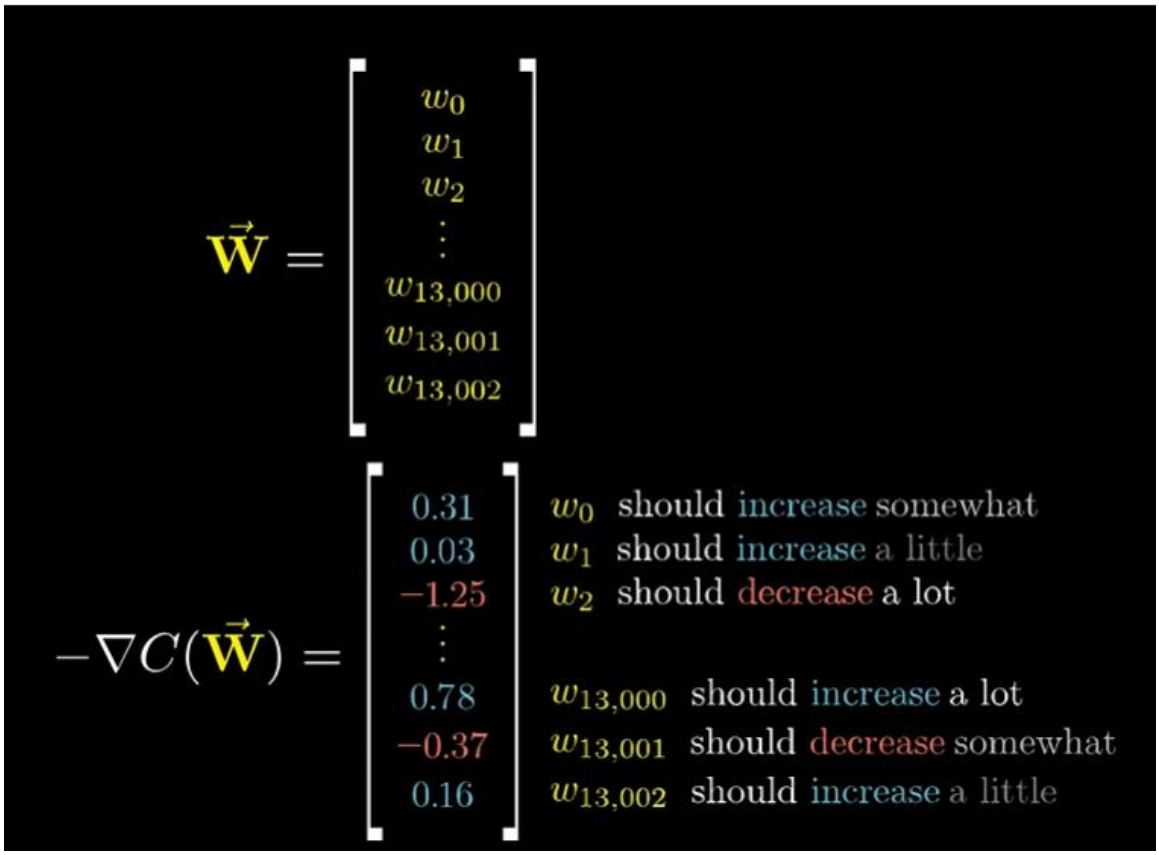


אך הקונספט הוא זהה, פשוט הדרך למציאת המינימום היא שונה, והיא נקראת Gradient Descent. בעצם "למידה של הרשת" מתרגם להקטנת ערך התוצאה של פונקציית ה-cost.

בסופו של דבר, Gradient Descent מחזיר לנו וקטור מספרי, שבו נשתמש כדי לשנות את ערכי המשקולות שלנו ברשת. לאחר שנשנה אותן, נריץ את כל הסיפור הזה שוב, ונשנה אותן שוב, ושוב, ושוב..

עד שנגיע לנקודה שבה פונקציית ה-cost היא מינימלית.

דוגמא קטנה:



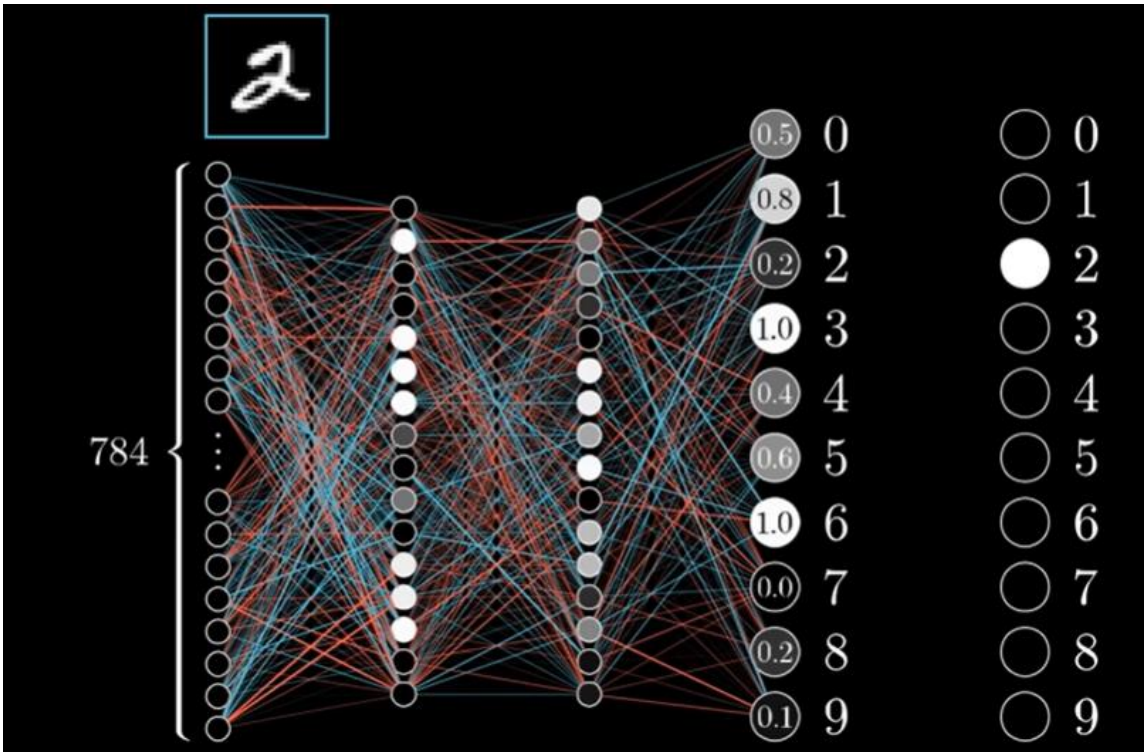
כל התהליך הזה, של חישוב ה-Gradient (הוקטור) ועדכון המשקולות, נקרא Backward Propagation.

לעומת תהליך Forward Propagation, שמתחיל בשכבת ה-Input ומסתיים בשכבת ה-Output. התהליך הזה מתחיל עם הערך שקיבלנו בשכבת ה-Output, ואז "הולך אחרונית" ומעדכן את המשקלים ברשת.

בקיצור הבנתם את הקונספט של השמות.

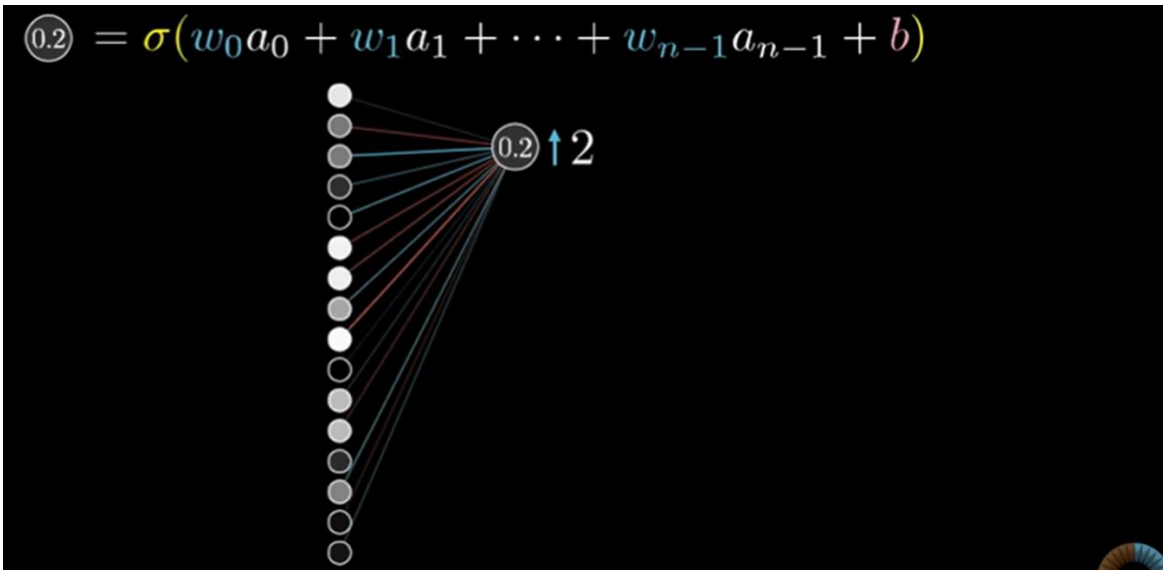
בואו נבין טוב יותר את התהליך הזה, קודם כל בלי מתמטיקה

נניח ויש לנו תמונה של הספרה 2, ביצענו Forward Propagation וקיבלנו תוצאה כלשהי:



במקרה הזה, נוכל לראות שהערך בנוירון שמייצג 2 צריך לגדול משמעותית (כרגע הוא 0.2 ואנחנו רוצים שיהיה 1) ושאר הערכים צריכים להתקרב ל-0, זאת אומרת שהיינו רוצים להקטין אותם.

נסתכל רק על הנוירון שמייצג את הספרה 2:



נוכל לראות גם את הנוסחה שבאמצעותה אנחנו מחשבים את הנוירון (Sigmoid של סכום כפל המשקולות והנוירונים בתוספת ה-bias).

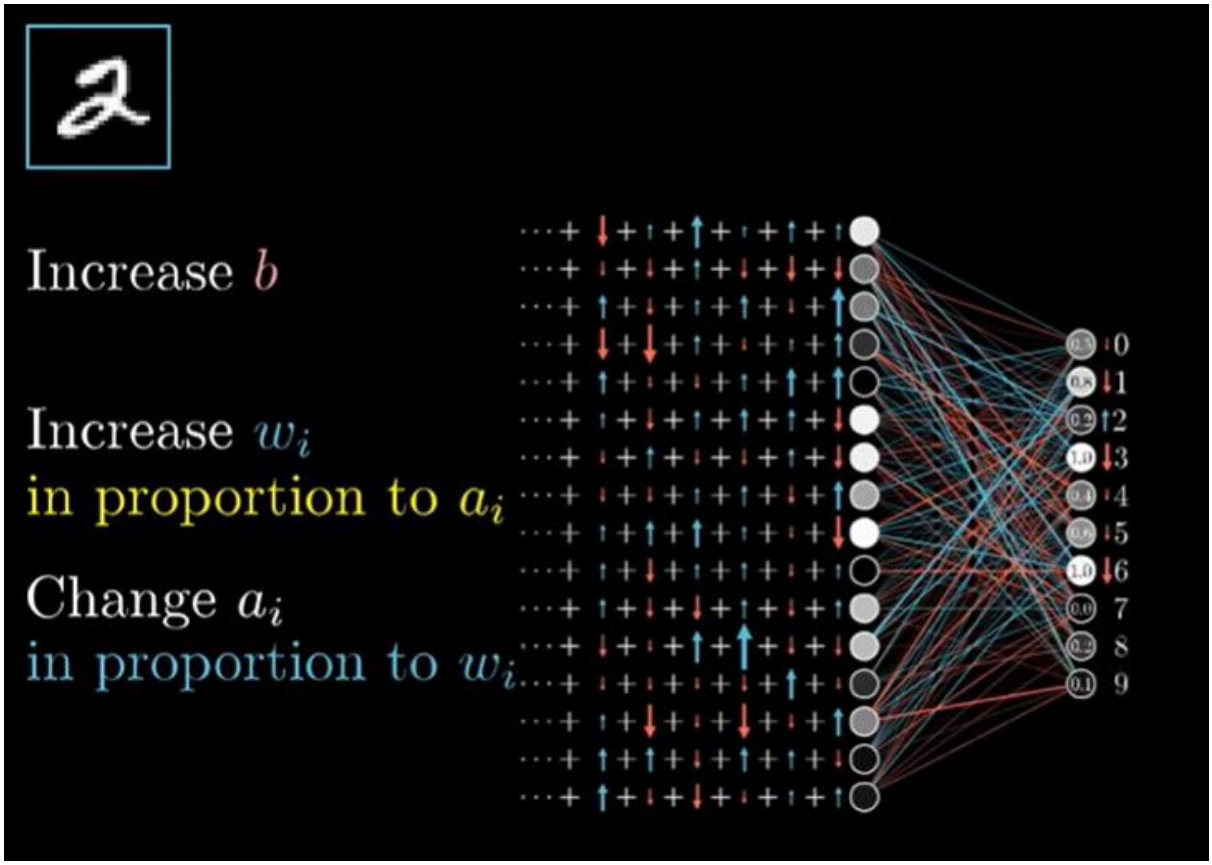
מה נוכל לשנות כאן כדי להגדיל את התוצאה שלנו?

1. את המשקולות המחברות את הנוירונים בשכבה הקודם לנוירון שלנו
2. את ה-bias של הנוירון שלנו
3. ללכת עוד שכבה אחת אחורה, לשנות בה את המשקולות או ה-bias, מה שישנה את הנוירונים של השכבה האחת לפני אחרונה פה

נוכל לראות בתמונה שהנוירון הראשון בשכבה האחת לפני אחרונה הוא מאוד בהיר, מה שאומר שיש לו ערך יחסית גבוה, לכן אם נגדיל את המשקל שמחבר בינו לבין הנוירון שלנו, נוכל להשפיע משמעותית(יחסית) על הערך של הנוירון שלנו. באותה מידה, נוכל להקטין את המשקולות שמחברות נוירונים נמוכים (שחורים) ובכך למנוע את הירידה של הערך בנוירון שלנו. (כי אנחנו נותנים לו פחות משקל, פחות משמעות ויכולת השפעה).

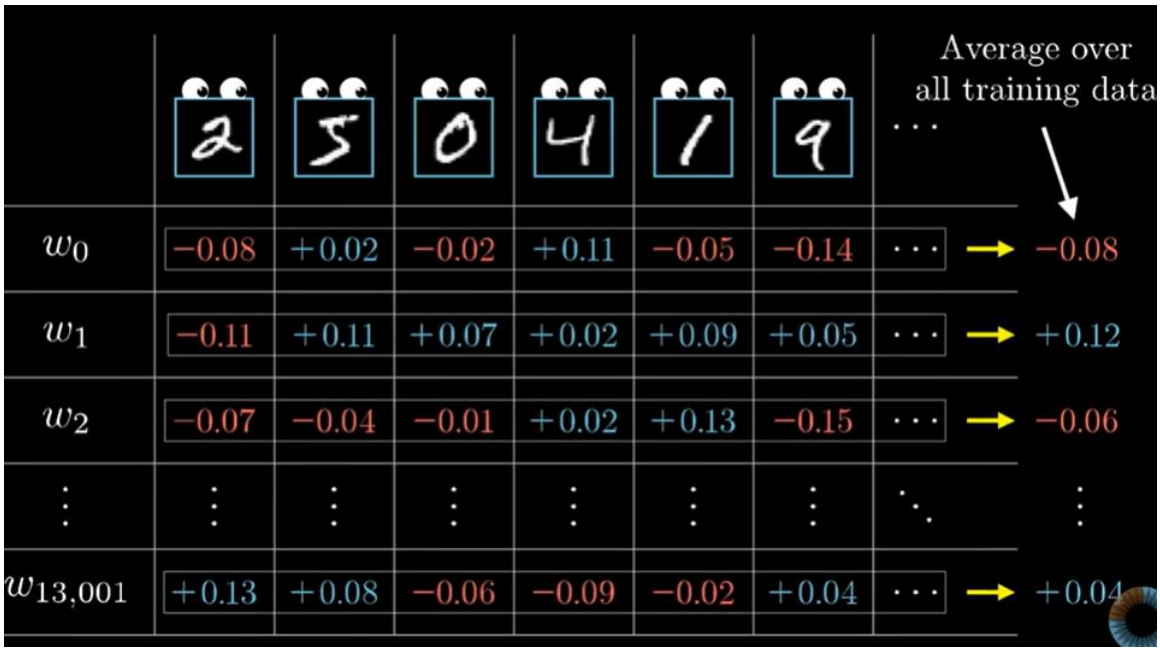
חשוב לזכור, שאחרי שנחליט מה הדרך הכי טובה להגדיל את הערך בנוירון הזה, נצטרך למצוא את הדרך הכי טובה למזער את הערכים בשאר הנוירונים.

להמחשה:



ושוב, כל התהליך הזה, הוא רק על דוגמא אחת באימון, אבל האימון שלנו מכיל 40,000 דוגמאות.

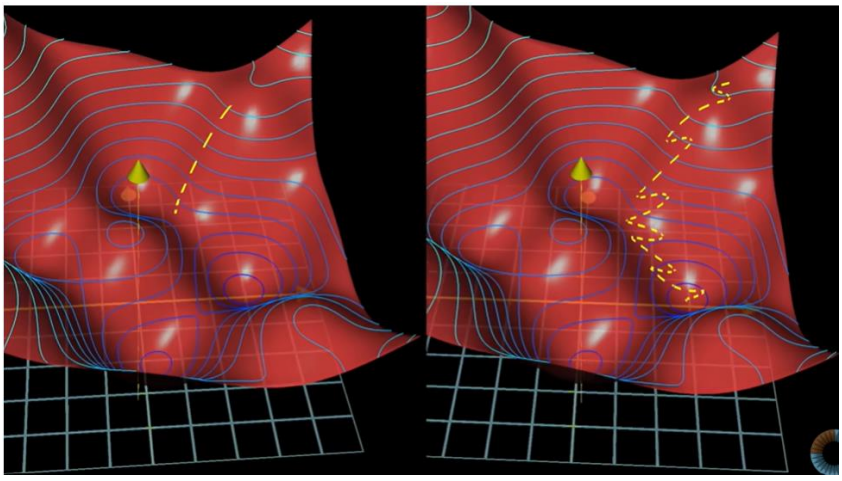
אחרי שעשינו את התהליך על כל הדוגמאות, נעשה ממוצע לכל משקל, וזה יהיה הערך שבו נשתמש כדי לשנות את המשקלים שלנו ברשת!



זה בגדול הוקטור Gradient שלנו. משום שתהליך החישוב הזה לוקח הרבה זמן, נהוג להשתמש בשיטה בשם stochastic gradient descent, שבעצם מחלקת את ה-data שלנו ל-batch-ים, שהם בעצם קבוצות קטנות יותר של דוגמאות אימון.

בכל פעם התהליך הזה מתבצע על כל batch ומעדכן את המשקלים בהתאם. הפעולה הזו משפיעה על החישוב בכך שבמקום שנרד למינימום לאט, אך בצורה מדויקת, נעשה פסיעות קטנות ומהירות כלפי המינימום.

- דמיינו איש שיורד מהר בצורה מחושבת ומדויקת לעומת איש שיכור שמתנדנד לו בהר עד למטה.



צד ימין זה stochastic.

כעת, בואו ננסה להבין קצת את הרעיון של החישוב שנעשה כדי ליצור את הוקטור הזה

אז יש לנו את פונקצית ה-cost שהיא ממוצע של ההפרש בין התוצאה שיצאה למודל לתוצאה המצופה בריבוע. להלן:

$$C = MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

MSE - mean squared error

עכשיו, כדי להבין מה המשקלים וה-bias הטובים ביותר לרשת שלנו, נצטרך כיצד הערך של פונקצית ה-cost משתנה בהתאם למשקולות ול-bias שלנו.

וכאן מתחיל חישוב ה-gradient. בדוגמא שלנו נחשב את ה-gradient של פונקצית ה-cost שתיוצג כ-C ביחס למשקולת אחת שתיוצג כ-wi. (החישוב הזה בעצם יתבצע עבור כל משקולת וכל bias).

לצורך החישוב אנחנו נשתמש בנגזרות חלקיות, ובכלל השרשרת המאפשר למצוא נגזרת של פונקציה המורכבת ממספר פונקציות אחרות:

$$\frac{\partial C}{\partial w_i} = \frac{\partial C}{\partial \hat{y}} \times \frac{\partial \hat{y}}{\partial z} \times \frac{\partial z}{\partial w_i}$$

אז מה יש לנו כאן? פונקציית ה-cost מיוצגת כ-C. המשקל הנוכחי מיוצג כ-wi הערך שהפונקציה חזתה (הערך הנוירון הסופי) מיוצג כ-$\hat{y}$.

סכום המשקולות כפול הנוירונים ועוד הbias שהובילו לנוירון שלנו מיוצג כ-Z (לפני פונקציית האקטיבציה). כעת, עלינו למצוא את שלושת הגרדיאנטים האלו:

$$\frac{\partial C}{\partial \hat{y}} = ? \qquad \frac{\partial \hat{y}}{\partial z} = ? \qquad \frac{\partial z}{\partial w_1} = ?$$

נתחיל מהראשון, הגרדיאנט של C ביחס לערך שהמודל חזה ($\hat{y}$):

$$\frac{\partial C}{\partial \hat{y}} = \frac{\partial}{\partial \hat{y}}\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 = 2 \times \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)$$

עד פה מדובר על נגזרת קלאסית, כעת, למטרות נוחות, נסדר את זה קצת.

נגיד ש-Y זה הוקטור $[y_1 \dots y_n]$ ו-$\hat{y}$ זה הוקטור $[\widehat{y_1} \dots \widehat{y_n}]$. ואז נוכל להציג את הגרדיאנט בצורה מופשטת יותר:

$$\frac{\partial C}{\partial \hat{y}} = \frac{2}{n} \times sum(y - \hat{y})$$

כעת נעבור ל-gradient של היחס שהמודל חזה ($\hat{y}$-y) לעומת Z:

$$\begin{aligned}\frac{\partial \hat{y}}{\partial z} &= \frac{\partial}{\partial z}\sigma(z) \\ &= \frac{\partial}{\partial z}\left(\frac{1}{1+e^{-z}}\right) \\ &= \frac{e^{-z}}{(1+e^{-z})^2} \\ &= \frac{1}{(1+e^{-z})} \times \frac{e^{-z}}{(1+e^{-z})} \\ &= \frac{1}{(1+e^{-z})} \times \left(1 - \frac{1}{(1+e^{-z})}\right) \\ &= \sigma(z) \times (1-\sigma(z))\end{aligned}$$

למה זה נכון? אמרנו ש-Z הוא בעצם הסכום של המשקולות כפול הנוירונים ועוד ה-bias בלי הפעלת פונקצית האקטיבציה.

אמרנו גם ש-$\hat{y}$ הוא הערך שנחזה, הערך הזה הוא בעצם הערך שיוצא אם ניקח את Z ונפעיל עליו את פונקצית האקטיבציה. זה ההסבר לשורה הראשונה. בשורה השניה פשוט שינינו את הסמל של פונקצית sigmoind ובמקומו ממש כתבנו את הפונקציה.

משם המשכנו עם חוקי נגזרת בסיסיים ומתמטיקה של תיכון, אז הכל בסדר. ונשאר לנו רק הגרדיאנט של Z ביחס למשקל הנוכחי wi:

$$\begin{aligned}\frac{\partial z}{\partial w_i} &= \frac{\partial}{\partial w_i}(z) \\ &= \frac{\partial}{\partial w_i}\sum_{i=1}^{n}(x_i.w_i + b) \\ &= x_i\end{aligned}$$

גם פה פשוט הצבנו במקום Z את מה שהוא מייצג (סכום המשקולות כפול הנוירונים ועוד ה-bias) והשתמשנו בכללי גזירה רגילים.

לבסוף הגענו לנוסחה הבאה שמייצגת את הגרדיאנט של פונקצית ה-cost ביחס למשקל הנוכחי:

$$\frac{\partial C}{\partial w_i} = \frac{2}{n} \times sum(y - \hat{y}) \times \sigma(z) \times (1-\sigma(z)) \times x_i$$

וזהו, בכל היופי הזה אנחנו יכולים רק לעדכן משקולת אחת אחרי אימון אחד, אז בכל אימון אנו עושים את התהליך הזה עבור כל המשקולות. תענוג!

מתקפות

רשתות מאוד מעניינות היום הן רשתות קונבולוציה CNN.

באמצעות רשתות אלו מתבצע זיהוי ויזואלי, זאת אומרת

1. מכוניות אוטונומיות
2. תעודות זהות ביומטריות
3. זיהוי הפנים שלך באייפון
4. ניתוח צילומי רנטגן

וכו...

אני חושבת שברור כמה כוח יכול להיות לבן אדם שמסוגל להטעות רשתות שכאלו.

מה הכוונה בלהטעות?

האם אני יכולה לקחת תמונה של מעבר חציה עם הולך רגל, לשנות אותה כך שלעין אנושית זה עדיין יראה כמו מעבר חציה עם הולך רגל, אבל עבור המודל זה יראה כמו מעבר חציה ריק?

מה ההשלכות של יכולות הטעיה שכאלו? והאם בכלל ניתן לעשות את זה?

אז התקפות על רשתות נוירונים מתחלקות ל-2 קטגוריות עיקריות:

1. מקרה בו יש לנו את ארכיטקטורת הרשת - ערכי המשקולות, שכבת ה-Input ושכבת ה-Output, לסוג הזה נקרא whitebox
2. מקרה בו יש לנו רק את שכבת ה-Input ושכבת ה-Output, לסוג הזה נקרא blackbox ועליו אנחנו לא נדבר במאמר זה

כמו כן, כאשר אנחנו תוקפים רשת יכולות להיות לנו מטרות שונות, למשל:

1. הטעיה, לא אכפת לי מה המודל יחזה, רק שהוא לא יזהה את הדבר הנכון. למשל, נתתי למודל תמונה של הספרה 3, לא אכפת לי איזה ספרה הוא יזהה, העיקר שהוא לא יזהה את הספרה 3.
2. הטעיה ליעד מכון, במקרה זה אני רוצה שהוא יזהה ספציפית את הספרה 8.

באופן כללי, הקונספט של התקפות על רשתות נוירונים מתמקד בהוספת רעש, שלא נראה לעין אנושית, אבל לחלוטין מבלבל את המודל.

הנה כמה רעיונות נחמדים שלפחות לשנת 2019 עוד עבדו.

כאן נוכל לראות איך באמצעות משקפיים הצליחו לגרום למודל להאמין שריס וויתרספון היא ראסל קרואו.

Figure 3: An impersonation using frames. Left: Actress Reese Witherspoon (by Eva Rinaldi / CC BY-SA / cropped from https://goo.gl/a2sCdc). Image classified correctly with probability 1. Middle: Perturbing frames to impersonate (actor) Russel Crowe. Right: The target (by Eva Rinaldi / CC BY-SA / cropped from https://goo.gl/AO7QYu).

כאן נוכל לראות איך באמצעות העובדה שהבינו מה המאפיינים הבולטים שמזהים את האדם הצליחו להסתיר את זהותו באמצעות הסתרה שלהם:

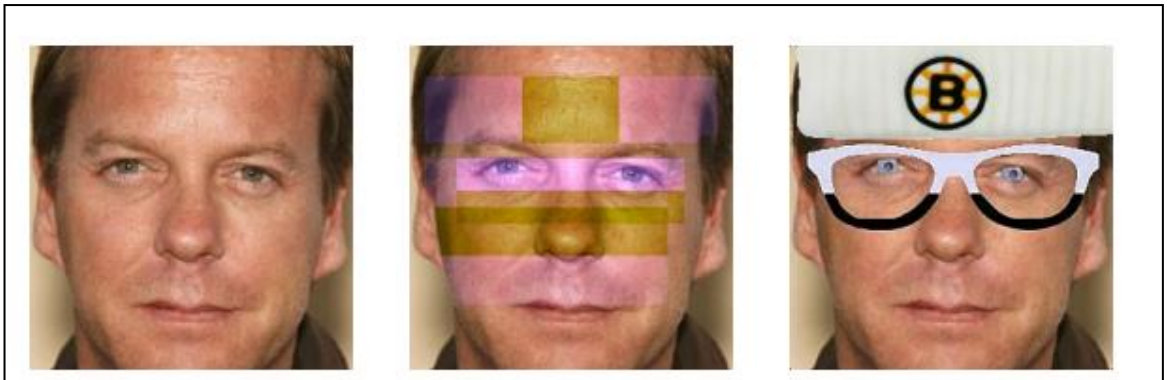


Figure 6: An example of an invisibility attack. Left: original image of actor Kiefer Sutherland. Middle: Invisibility by perturbing pixels that overlay the face. Right: Invisibility with the use of accessories.

FGSM - fast gradient sign method

עכשיו כשאנחנו מבינים כיצד מחושב גרדיאנט, המתקפה הזו ממש פשוטה להבנה!

השלבים לביצוע המתקפה הם כאלו:

1. נחשב את פונקצית ה-cost
2. נחשב את הגרדיאנט, אבל הפעם, במקום לחשב אותו ביחס למשקל wi, נחשב אותו ביחס לתווית ה-Input
3. לפי הגרדיאנט שחישבנו, נזיז את הפיקסלים של תמונת ה-Input מעט, במקום להזיז אותם לכיוון cost מינימלי, נזיז אותם לכיוון cost מקסימלי

זאת אומרת ששינינו כמה דברים:

1. במקום להשתמש במשקל בחישוב הגרדיאנט השתמשנו בתווית ה-Input
2. במקום להגיע לפונקצית cost מינימלית אנו מנסים להגיע לפונקצית cost מקסימלית
3. לאחר שמצאנו את הגרדיאנט, נשתמש בו כדי לשנות את הפיקסלים של תמונת ה-Input (ולא את המשקלים)

אז הנוסחה תראה בעצם כך:

$$x^{adv} = x + \varepsilon \cdot \mathrm{sign}(\nabla_x J(x, y_{true})),$$

where

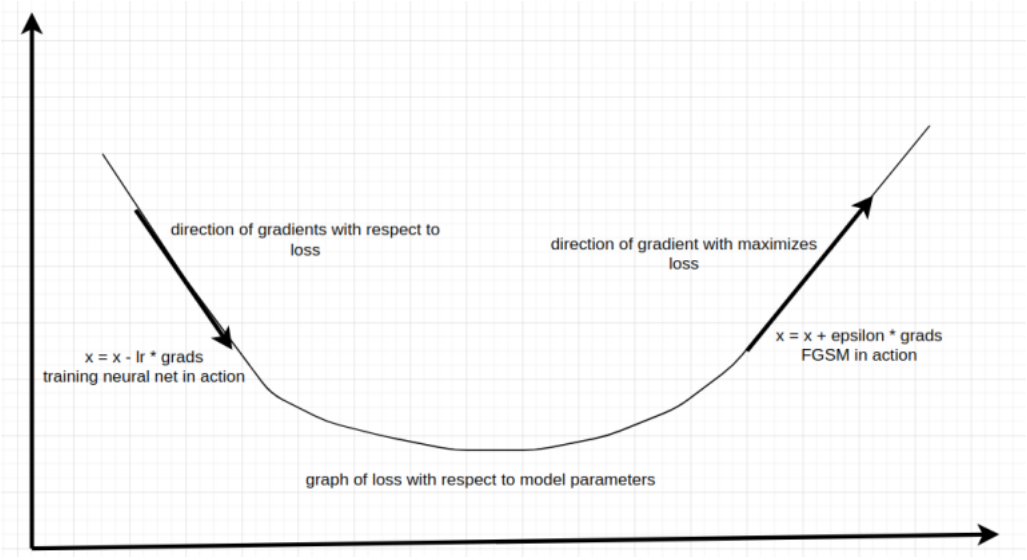
x is the input (clean) image,

x^{adv} is the perturbed adversarial image,

J is the classification loss function,

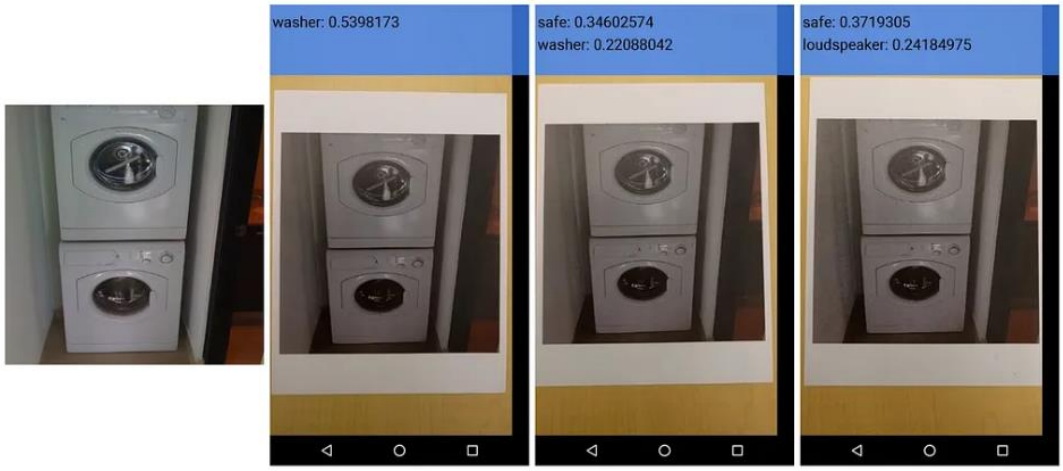
y_{true} is true label for the input x.

זאת אומרת, לקחנו את הכיוון של הוקטור שקיבלנו בגרדיאנט (שחושב לפי תווית ה-Input ופונקצית ה-cost), הכפלנו בערך אפסילון (ערך קטן רנדומלי) והוספנו לפיקסלים של התמונה.



[בצד ימין אפשר לראות את הפעולה של המתקפה לעומת צד שמאל בו נוכל לראות אימון רגיל]

הנה דוגמא נחמדה לתמונות, בכל פעם תוכלו לראות את התווית שהמודל זיהה ואת אחוז הודאות, השינוי שנעשה הוא הגדלת האפסילון:



(a) Image from dataset (b) Clean image (c) Adv. image, $\epsilon = 4$ (d) Adv. image, $\epsilon = 8$

מתקפה זו היא מסוג one-shot שכן ביצענו רק חישוב אחד, לכן היא גם מהירה יותר אך פחות מוצלחת ממתקפות אחרות.

מתקפה נוספת היא T-FGSM

במתקפה זו ה-T מסמל target והיא עובדת בדומה למתקפה הקודמת, אך הפעם, במקום לחשב את הגרדיאנט ביחס לפונקצית ה-cost ותווית ה-Input, נחשב אותו ביחס לפונקצית ה-cost ותווית ה-output, ואת הסימן שלו נחסר מתמונת המקור.

נניח שוקטור ה-Output שלנו יהיה בגודל 10 כאשר הנוירון המיצג את הספרה 3 יכיל את הערך 1 ושאר הנוירונים יכילו את הערך 0. ניקח את הוקטור הזה ונציב אותו בנוסחה:

$$x^{adv} = x - \varepsilon \cdot \mathtt{sign}\left(\nabla_x J(x, y_{target})\right),$$

where

y_{target} is the target label for the adversarial attack.

חשוב לשים לב לסימן החיסור של X באפסילון לעומת החיבור בנוסחה הקודמת, פעולה זו הגיונית אינטואיטיבית שכן אנו מחשבים את הכיוון ביחס לתוצאה ונרצה שהתמונה פחות תתאים לתוצאה.

והמתקפה האחרונה שנדבר עליה מסוג זה היא I-FGSM **כאשר ה-**I **מסמן** iterative

מתקפה מסוג זה נמצאת תחת הקטגוריה של Iterative attacks וזו משום שהחישוב לא נעשה בפעם אחת כמו במתקפות שהצגתי עד כה, אלא מתבצעות מספר איטרציות שבהן נעדכן את התמונה המקורית:

$$x_0^{adv} = x, \quad x_{t+1}^{adv} = x_t^{adv} + \alpha \cdot \mathrm{sign}\left(\nabla_x J(x_t^{adv}, y)\right).$$

הנוסחה זהה למתקפת FGSM הראשונה שהצגתי רק שהפעם התמונה המקורית תתעדכן t פעמים.

למתקפות איטרציה יש שיעורי הצלחה גבוהים יותר ממתקפות one shot.

הגנות

רוב המודלים שונים זה מזה, אך הרבה פעמים הם אומנו על אותם data-set-ים. זאת משום שכדי להכין data-set לוקח הרבה זמן ומשאבים ולפעמים זה בכלל לא אפשרי כי אין לך גישה אישית ל-data שאתה צריך.

לכן, הרבה פעמים ניתן לשטות במודלים שונים עם אותה התמונה. עם השנים, פותחו מספר דרכים שמנסות להגן על המודלים.

אחת הדרכים המוכרות להגנה מפני מתקפות כאלה היא להכניס את התמונות המטעות אל תוך שלב האימון, התמונות יכנסו עם תווית נכונה, ולמרות שהפיקסלים בהן שונו כדי להטעות את המודל, אנחנו מלמדים אותו לשנות את המשקלים שלו כך שיוכלו לזהות נכון גם את התמונות האלו.

האימון הזה יכול לקחת בין 3 ל-30 פעמים יותר זמן.

דחיסת פיצ'רים היא שיטה נוספת בה משתמשים כדי לנסות למנוע מתמונות להטעות את המודל.

השיטה מורכבת מ-3 שלבים:

1. בשלב הראשון נבצע את התהליך הרגיל ונראה מה המודל מזהה עבור התמונה
2. בשלב השני נפחית את עומק הצבע בתמונה, כך נפחית את מספר הצבעים השונים שכל פיקסל יכול לייצג

כדי להפחית את עומק הצבעים, אנחנו בעצם לוקחים אזורים עם צבעים דומים ומחליפים אותם בצבע אחד שמייצג את האזור(סוג של ממוצע)

דוגמא: נניח שאנחנו עובדים עם תמונה צבעונית, אז כפי שהסברנו בהתחלה יש לנו מטריצת 3MXNX שמייצגת RGB, כל פיקסל נע בין 0 ל-255.

נניח שאנחנו רוצים לדעת כיצד יראה אדום בהיר. אז אדום בהיר מורכב מ-100% אדום, 80% ירוק ו-79.6% כחול, ייצוג ה-RGB שלו בעצם יראה כך RGB(255,204,203)

נחזור לאלגוריתם להפחתת עומק הצבעים:

נחלק את הצבעים לאזורים, למשל חלוקת הצבע האדום:

R(0-31), R(32-63), R(64-95), R(96-127), R(128-159), R(160-191), R(192-223), R(224-255)

וכעת, נניח שיש לנו צבע C שהייצוג שלו הוא (3,34,189), אז הוא שייך לאזורים הבאים:

R(0-31)

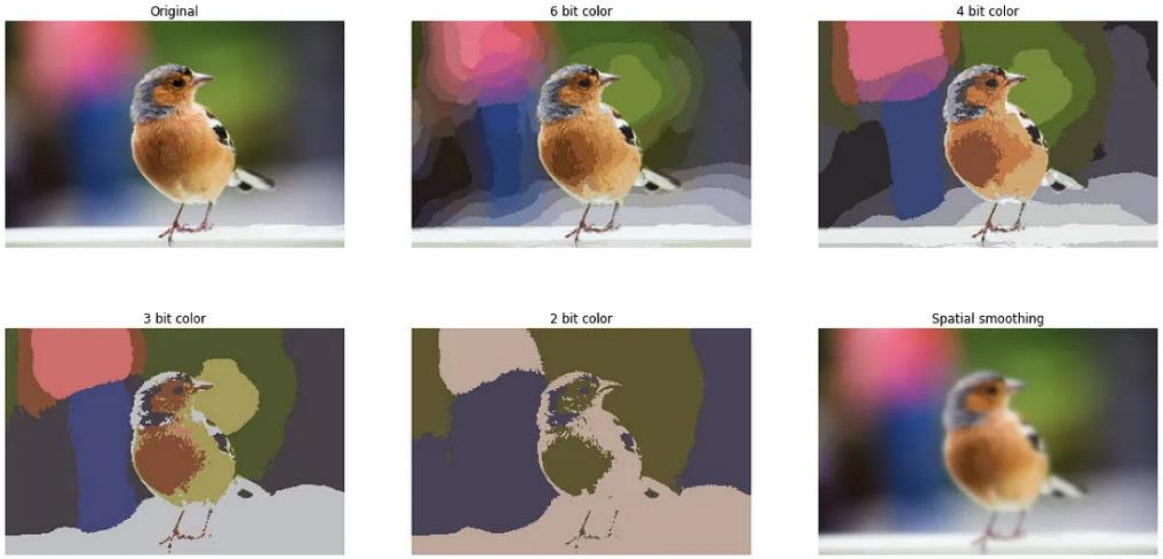
G(32-63)

B(160-191)

כעת, אם נעשה ממוצע לכל אזור, כעת C יהיה (16,48,178), וכך יוצגו גם כל הצבעים שבטווחים האלו. המטרה של פעולה זו היא להגביל את מרחב התמרון שיהיה לתוקף עם המשחק של הפיקסלים.

3. בשלב השלישי נבצע פעולה בשם spatial smoothing שבקיצור זה סוג של ממוצע עם כלל השכנים שאמור לעזור לצמצם רעשים קטנים.

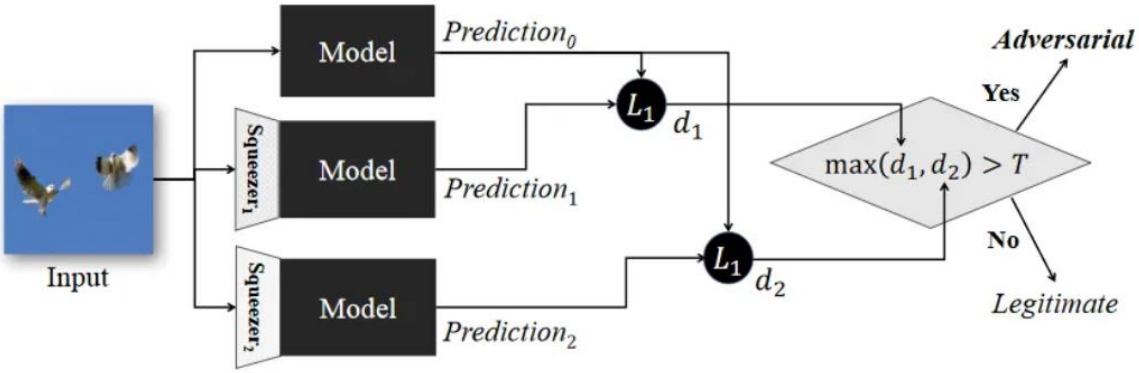
נוכל לראות את הצמצום של הצבעים וכן את הפלט של spatial smoothing כאן:



לאחר מכן, אנחנו מחשבים שני דברים:

1. את המרחק בין מה שהמודל חזה בתמונה המקורית לבין מה שהוא חזה בתמונה שיצרנו בשלב 2
2. את המרחק בין מה שהמודל חזה בתמונה המקורית לבין מה שהוא חזה בתמונה שיצרנו בשלב 3

אם אחד המרחקים חורג מאיזה שהוא סף שהוגדר, אז התמונה מסווגת כמטעה (או זדונית). והנה התרשים של התהליך הזה:



סיכום

קצת קשה לסכם משום שמדובר רק בתמצית מהחומר.

מאוד מעניין לראות כיצד דברים שנראים על טבעיים או פשוט לא ברור כיצד הם עובדים מקבלים הסברים הגיוניים.

אני חושבת שהמתמטיקה מאחורי האלגוריתמים האלו היא פשוט גאונית ומרתקת, ככל שאני מתעמקת בה יותר, כך אני סקרנית יותר, מקווה שעכשיו גם אתם

על המחבר

@sapirxfed עושה retweet לכל דבר שקשור ל-AD או AAD.

אוהבת לכתוב קוד, מנסה למצוא חולשות, ומעריצה שרופה של DigitalWhisper!

- ולפעמים אוהבת מתמטיקה

תודות

תודה לכל מי שהסכים לקרוא ולתת הערות למאמר הזה ☺

ביביליוגרפיה

סדרת סרטונים (רוב התמונות במאמר לקוחות מסדרה זו) שמסביר את כל הקונספט של Backward and Forward Propagation:

https://www.youtube.com/watch?v=aircAruvnKk&list=PLZHQObOWTQDNU6R1_67000Dx_ZCJB-3pi

מאמרים על מתקפות והגנות ברשתות נוירונים:

https://towardsdatascience.com/breaking-neural-networks-with-adversarial-attacks-f4290a9a45aa

https://towardsdatascience.com/fooling-neural-networks-with-adversarial-examples-8afd36258a03

https://arxiv.org/abs/1412.6572

המאמר "Adversarial Examples in the Physical World":

https://users.cs.northwestern.edu/~srutib/