**ארכיטקטורות**

**AutoEncoders**

<https://www.youtube.com/watch?v=E28CVTbNoSA&t=1184s>

**הקדמה**

בעידן המידע בו אנו חיים, כמות הנתונים שנצברת מדי יום מציבה אתגרים והזדמנויות חדשים לפני מדעני הנתונים והמחקר. במרכזם של אתגרים אלו עומד הצורך למצוא דרכים יעילות לדחוס, לנתח ולפרש את הנתונים האדירים הללו. בהקשר זה, אחת הטכנולוגיות המרכזיות המשמשות היום היא ארכיטקטורת ה-AutoEncoders, שפותחה בשנות ה-80 ומאפשרת דחיסת נתונים באופן אוטומטי ויעיל. היא מתמקדת בלמידת רפרזנטציות מצומצמות ועמוקות של הנתונים, באמצעות מודל שמחלק את עצמו לשני חלקים: Encoder, האחראי להפחתת ממדי הנתונים, ו-Decoder, שתפקידו לשחזר את הנתונים מהרפרזנטציה המצומצמת.

חשוב לציין כי בAutoEncoders, הקלט הוא גם הפלט והObjective שיפורט בהמשך חל בין הקלט לבין עצמו. למעשה במהלך תהליך הלמידה המטרה לצמצם ככל הניתן את הקלט למרחב לטנטי כלשהו ועם זאת לשמר את התלויות המהותיות בדאטא כך שבסופו של דבר הרפרזנטציה הנלמדת מהווה תמצות וזיקוק של הדאטא ולאחר מכן משמת לטובת הרכבתו מחדש.

**שימושים**

ארכיטקטורת AutoEncoders מתמקדת במספר שימושים עיקריים המבוססים על הייצוג הלטנטי:

1. דחיסה (Compression) – מאפשר דחיסת נתונים על ידי הפחתת מימד הקלט, דרך הלמידה של רפרזנטציות נמוכות יותר מבלי לאבד מידע מהותי. זהו שימוש חשוב במיוחד כאשר רוצים לצמצם את נפח הנתונים לפני עיבודם או שימורם.
2. ניקוי רעשים (Denoising): דרך הטמעת שכבת רעש במהלך האימון, AutoEncoders יכולים ללמוד לשחזר את הקלט המקורי מתוך קלט רועש, ובכך להפחית או להסיר את הרעש מהנתונים עבור דאטא חדש במהלך תהליך הinference.
3. ייצוג דליל (Sparse Representation): אפשרות נוספת היא למידת ייצוג דליל שבו רוב הערכים הם אפסים.
4. יצירת נתונים (Data generation): בהינתן וה-AutoEncoder כבר מאומן כראוי, על ידי ידיעת התפלגות הווקטור במימד הלטנטי ניתן לבצע הרכבה (reconstruction) על מנת לקבל דאטא מג׳ונרט.

**תיאור הארכיטקטורה**

התמונה מציגה את ארכיטקטורת ה-AutoEncoder, המורכבת משלושה חלקים עיקריים:

1. **Encoder**: החלק הראשון של הרשת, שבו התמונה המקורית מוזנת. ה-Encoder מתמקד בכך שהוא לוקח את הנתונים הממדיים הגבוהים ודוחס אותם לתוך מרחב ממדי נמוך יותר, הנקרא מרחב הלטנטי. הEncoder מכיל סט משקולות נלמדות.
2. **Bottleneck**: זהו החלק המרכזי של ה-AutoEncoder, כאן למעשה מוצג הדאטא בצורתו הדחוסה, המידע מיוצג על ידי וקטור קטן יותר של תכונות, המכונה גם "הייצוג במרחב הלטנטי".
3. **Decoder**: בחלק הזה של הרשת, המידע שנדחס על ידי ה-Encoder מוזן כעת ל-Decoder, שמטרתו לשחזר את התמונה המקורית מהייצוג הדחוס שלה. התמונה שמופקת כאן מכונה "התמונה המשוחזרת". הDecoder מכיל סט משקולות נלמדות.

A diagram of a blue rectangular object

Description automatically generated

**פורמלית**

מבחינה פורמלית, המשוואה המוצגת מבטאת את הObjective שבה אנו שואפים למזער במהלך האימון של מודל AutoEncoder. המטרה היא למצוא את הערכים האופטימליים של המשקולות , שבהם ההבדל בין נתוני הקלט ​ לבין תמונת הקלט המשוחזרת , המיוצרת על ידי המודל, יהיה הקטן ביותר. הפונקציה מחושבת כסכום של חצי הנורמה האוקלידית בריבוע, עבור כל נקודת נתונים בסט האימון. המשקולות מייצגות את הEncoder, והמשקולות ​ מייצגות את הDecode.

חשוב לציין כי הObjective המוצג כאן למעשה מייצג linear AutoEncoder מהסיבה שהפעולות המוצגות כאן הן פעולות של כפל מטריציוני (טרנספורמציה לינארית) ולכן הAutoEncoder המשתמש בObjective זה הוא לינארי.

עם זאת, בהינתן פונקציות לא לינאריות המוגדרות עבור הEncoder והDecoder בהתאמה או לחילופין אסופה של פונקציות לא לינאריות המצויות בין קומפוננטות בEncoder והDecoder נראה כי נקבל Non-linear AutoEncoder באופן בו הייצוג הלטנטי שהתקבל מסוגל להכיל תלויות שאינן לינאריות וכמו כן אופן ההרכבה בהתאם יכול להיות לא לינארי.

מבחינה פורמלית ניתן לתאר את ה Non-linear AutoEncoderבאופן הבא:

רפרזנטציה דלילה:

כאמור, תיארנו את אחד השימושים של AutoEncoder ככזה המאפשר ללמוד רפרזנטציה דלילה (Sparse Representation) באופן בו הווקטור המצוי במרחב הלטנטי מכיל ברובו אפסים. לצורך ביצוע פעולה זו על פניו היינו רוצים להשתמש בפונקציית הObjective הבאה:

*nnz מייצג non-zeros entries כלומר היינו רוצים פונקציית Objective שמטרתה למנמם ככל הניתן את מספר הסקלארים בווקטור הלטנטי שאינם אפס.*

*מהסיבה שזו דרישה בעייתית לאופטימיזציה נוכל לבצע רדוקציה לדרישה ולתאר אותה באופן הבא:*

מבלי להיכנס להוכחה שמתארת את השקילות, אנחנו יכולים לראות כאן כי רגולריזציה מסוג L1 שקולה לאופטימיזציה שרצינו לבצע על ידי מינמום הnnz ולכן Objective זה עתיד ליצור ווקטור לטנטי דליל.

כמו כן אנחנו יכולים לראות כי c הוא היפרפרמטר שניתן לקביעה במהלך תהליך האימון כיאה לObjectives מהסוג הזה.

**דוגמא לשימוש – Denoising AutoEncoder**

כאמור, כפי שציינו מקודם אחד השימושים של AutoEncoder הוא לטובת ניקוי רעשים.

האופן בו אנו יכולים לרתום את הAutoEncoder לניקוי רעשים הוא באופן הבא:

שלב 1 – הכנת דאטא מלוכלך

בהינתן דאטא כלשהו, נוסיף לו רעש בהתפלגות גאוסיינית . לרוב כאשר אנו מוסיפים רעש בהתפלגות גאוסיינית, ערך ממוצע ההתפלגות הוא 0 וערך סטיית התקן משתנה לפי הצורך. על ידי הוספת רעש בצורה זו נקבל כי רוב הפיצ׳רים בדאטא יקבלו ערכי שינויי קטנים מהסיבה שהם ממוקמים במרכז ההתפלגות. עם זאת יהיו כאלו שיקבלו ערכים גדולים יותר תחת ההנחה שהערכים שיתווספו יהיו ממוקמים בשולי ההתפלגות.

לאחר שהוספנו לדאטא רעש גאוסייני קיבלנו דאטא מורעש שנוצר כתוצאה מהדאטא המקורי.

שלב 2 – שלב האימון

בשלב זה אנחנו מאמנים את הAutoEncoder באופן בו הקלט לAutoEncoder הוא הדאטא המורעש. בתהליך האימון אנחנו מעבירים את הדאטא המורעש לאורך הEncoder והDecoder כך שלסוף אנו מודדים את הLoss בין הדאטא המקורי לפני ההרעשה לבין הדאטא המנוקה לכאורה. אנחנו ממשיכים בפעולה באופן איטרטיבי עד להתכנסות.

ניתן לראות את הObjective המתואר פורמלית במשוואה הבאה:

שלב 3 – שלב הInference

בInference קיבלנו על פניו מודל מאומן ולכן כל שעלינו לעשות הוא להכניס דאטא מלוכלך והרשת עתידה לספק לנו את הדאטא מנוקה מהסיבה שהיא למדה רפרזנטציות של תהליך הניקוי.

A group of numbers on a black background

Description automatically generated

**דוגמא לארכיטקטורה**

התמונה מתארת את ארכיטקטורת AutoEncoder עם מרכיבי ביניים שמרחיבים את התהליך הפשוט של Encoder והDecoder.

בחלק השמאלי של האיור, אנו רואים את הEncoder, שמתחיל בקלט של תמונה בגודל 28x28x1. התמונה עוברת דרך מספר שכבות קונבולוציה (Conv1 ו-Conv2), כאשר כל שכבה מבצעת סינון של הנתונים ומפחיתה את הממדים דרך פעולת ה-stride. לאחר שכבות הקונבולוציה, הנתונים מועברים לשכבת Flatten לקבלת וקטור ישר אחד, ואז מועברים לשכבת חיבור מלאה (FC) לקבלת הייצוג הווקטורי במרחב הלטנטי (Latent space).

בחלק הימני של האיור, אנו רואים את הDecoder שמנסה לשחזר את התמונה המקורית מהייצוג הלטנטי. הוא מתחיל משכבת חיבור מלאה (FC) ולאחר מכן משתמש בפעולת Reshape כדי להחזיר את הנתונים למבנה מרחבי שמתאים לפעולות קונבולוציוניות. שתי שכבות קונבולוציוניות הפוכות (DeConv1 ו-DeConv2), שכל אחת מהן מבצעת Upsampling, נמצאות בפעולה כדי להרחיב את הממדים ולהגיע לממד התמונה המקורי של 28x28x1.

לסיכום ניתן לומר כי אמנם AutoEncoder אמנם מוגדר בצורה כללית על ידי Encoder, Decoder אך גם למבנים אלו ישנן מרכיבי ביניים שמשתנים בין כל AutoEncoder למשנהו.

A diagram of a diagram of a complex structure

Description automatically generated with medium confidence

**Probabilistic AutoEncoders**

העיקרון של AutoEncoders הסתברותיים מאוד דומה לעיקרון של AutoEncoders עליהם דיברנו עד כה. עם זאת, השוני כאן הוא שהAutoEncoders הללו מכילים פונקציות הסתברותיות המוגדרות על סמך התפלגות מסוימת.

הכוונה היא שהווקטור במרחב הלטנטי וכמו כן הפלט מוגדרים על פי הקלט שנכנס אל הפונקציה, מועבר דרך estimator המוגדר על פי התפלגות, ורק לאחר מכן מתקבל ערך המשתנה במרחב הלטנטי וערך המשתנה בפלט.

בתמונה ניתן לראות כי המבנה הארכיטקטורי של הAutoEncoder ההסתברותי זהה לחלוטין למבנה של הAutoEncoder אך עם זאת ערכי הווקטור במרחב הלטנטי מוכתבים על פי התפלגויות עבור כל אחד מהאלמנטים בווקטור הלטנטי. הפלט ההסתברותי שמוכתב על ידי ההתפלגות הוא גם אלמנט הניתן ללמידה במהלך תהליך האימון. בהתאם לסוג ההתפלגות, הערכים המכתיבים את ההתפלגות (למשל μ, σ עבור התפלגות נורמלית), עתידים להשתנות במהלך תהליך האימון כדי שלאחר ההרכבה בDecoder של מהווקטור הלטנטי נקבל פלט שדומה ככל הניתן לקלט איתו התחלנו.

A diagram of a machine learning

Description automatically generated

מכיוון שהערכים בווקטור הלטנטי מוכתבים על ידי התפלגויות, הרשת מסוגלת לייצר פלטים מורכבים ולא דטרמיניסטיים, דבר שמאפשר גיוון רחב יותר בנתונים שהרשת מסוגלת להתמודד איתם ולשחזר. היכולת לקבוע את ההתפלגות של כל אלמנט בווקטור הלטנטי מאפשרת גישה יותר גמישה ומותאמת אישית לטיב הנתונים, ובכך משפרת את איכות הלמידה והשחזור של הרשת.

**Probabilistic AutoEncoders Based Normal Distribution Example**

לצורך ההמחשה נניח והקלט שלנו הוא בגודל 8 ומוגדר . כמו כן נניח והווקטור במרחב הלטנטי הוא מגודל 2 ומוגדר .

אנחנו מתחילים עם וקטור בגודל 8 ולאחר העברתו בשכבות הEncoder אנחנו מקבלים כי הפלט של השכבה האחרונה בEncoder (הכתומה) מכיל את ערכי ההתפלגות. אלו הם ערכים נלמדים במהלך תהליך האימון. לאחר מכן אנו דוגמים מן ההתפלגות ומייצרים את הווקטור הלטנטי (השכבה הירוקה) בגודל 2. הווקטור הזה בתורו עובר דרך הDecoder עד שהוא מגיע לשכבה האחרונה של הDecoder שמכילה 8 זוגות ערכים μ σ כגודל הקלט המקורי. לבסוף לאחר דגימת ההתפלגויות עבור כל אחד מהאלמנטים בווקטור אנו מקבלים את הפלט בגודל 8.

A diagram of a graph

Description automatically generated

A blue heart with white numbers

Description automatically generated

**המוטיבציה לשימוש בProbabilistic AutoEncoders**

על פניו, היינו יכולים להמשיך ולהשתמש בAutoEncoders, לקבל ייצוגים לטנטיים ולהרכיב אותם לכדי ערך הפלט כמו למשל שהדגמנו בשימוש בDenoising AutoEncoder.

עם זאת, אם אנחנו מסתכלים על הייצוג של הווקטורים במרחב הלטנטי נדמה כי הם לא מספרים לנו סיפור מעניין במיוחד. אוסף של מספרים שהתקבלו מתהליך של עדכון משקולות כחלק מBackpropagation. אם מסתכלים למשל על מספרים המיוצגים אל אותם וקטורים לטנטיים נראה כי הם למעשה אסופה של וקטורים שנלמדו על סמך סט האימון ואינם לקוחים משום התפלגות.

A blue flower with white numbers

Description automatically generatedלכן, באמצעות שימוש בAutoEncoders הסתברותיים אנו יכולים לאפיין את הווקטורים הלטנטיים כלקוחים מתוך הסתברות רציפה כלשהי. כך בהינתן ווקטור לטנטי אותו נרצה לייצר תחת ההנחה שנרצה לקבל פלט הגיוני, נוכל לדגום מן ההתפלגות באזורים המיוחסים לאותו פלט ובכך לקבל תוצר הקרוב למה שציפינו.

למעשה, באמצעות שיטה זו אנחנו יכולים לייצר דאטא מלאכותי באמצעות ידיעת התלות בין הקלט, הווקטור הלטנטי וההתפלגות. זו למעשה ההקדמה לVAE וליכולת הג׳נרטיבית שהאכיטקטורה מאפשרת לנו עליה נרחיב כעת.

**Variational AutoEncoder – VAE**

<https://www.youtube.com/watch?v=DWVlEw0D3gA&t=1447s>

**הקדמה**

Variational Autoencoders, או בקיצור VAE, מהווים חידוש בתחום של למידת מכונה בשנים האחרונות. מדובר בסוג של רשתות נוירונים שמאפשרות גישה חדשה ליצירת רפרזנטציה מצומצמת של נתונים. עיקר ההבדל בין VAE ל-Autoencoders הקלאסי הוא בהנחה הבסיסית שעל פיה המרחב הלטנטי אינו מרחב דטרמיניסטי אלא מרחב הסתברותי, מה שמאפשר ל-VAE ליצור דגימות חדשות ומגוונות באמצעות חישוב של התפלגויות.

בVAE, ה-Encoder מוגדר כפונקציה סטטיסטית המשקפת את הפרמטרים של התפלגות המרחב הלטנטי, וה-Decoder לוקח דגימות מהמרחב הלטנטי ומשחזר את הנתונים.

המטרה בלמידה היא למקסם את פונקציה הLoss של ה-VAE. פונקציה זו נועדה להתמודד עם שני גורמים: האחד הוא השגת דחיסה יעילה של הנתונים ומיקסום הlog likelihood, והשני הוא קידוד הנתונים לכדי התפלגות נורמלית שדומה להתפלגות המקורית – למעשה רצון לקירוב ההתפלגויות.

**הסבר הObjective בVAE**

את המשוואה שמייצגת את הObjective של VAE ניתן לחלק ל2 חלקים:

**חלק 1 – Maximum log likelihood –**

אנו משתמשים ב-log likelihood כפונקציית loss בחלק הראשון של ה-Objective. הסיבה לכך היא שפונקציית ה-log likelihood מודדת את הסיכוי שהנתונים המשוחזרים יהיו דומים לנתונים המקוריים . מיקסום ה-log likelihood מתרחש כאשר הסיכוי לשחזור מדויק של התמונה המקורית הוא הגבוה ביותר. במילים אחרות, ככל שהדמיון בין ל-*יהיה* גבוה יותר, כך ערך ה-log likelihood גבוה יותר, והמודל יתקרב יותר למטרה שלו.

**חלק 2 – קירוב ההתפלגויות –**

בחלק השני של ה-Objective, אנו משתמשים ב-Kullback-Leibler divergence כדי לקרב את ההתפלגות של הוקטור הלטנטי להתפלגות נורמלית רב-ממדית . הסיבה לכך היא שדגימה מהתפלגות נורמלית פשוטה יחסית, ומאפשרת יצירת תמונות חדשות בקלות.

כמו כן, חשוב לציין שאנחנו לא יכולים לדגום ישירות מההתפלגות של כתלות בוהסיבה לכך היא שהתפלגות זו עשויה להיות מורכבת וקשה לדגימה. לכן, אנו משתמשים ב-KL Divergence כדי למצוא התפלגות קרובה (התפלגות נורמלית) שקל יותר לדגום ממנה.

**פירוט על המשמעות של קירוב ההתפלגויות**

קירוב ההתפלגויות במרחב הלטנטי הוא פעולה מרכזית בפעילות של Variational Autoencoders. המנגנון הזה מאפשר למודלים להתמודד עם אחד האתגרים הבסיסיים בלמידת מכונה: יצירת והבנה של התפלגויות מורכבות של נתונים. ב-VAEs, נעשה שימוש בהתפלגות נורמלית רב-ממדית כתיקן כדי להתמודד עם המרחב הלטנטי של הנתונים. תהליך זה מאפשר לנו לגשר בין הפערים הנובעים מהמרחב המקורי של הנתונים לבין המרחב שאנו יכולים לנהל ולשלוט בו מבחינה חישובית.

הקושי לדגום ישירות מהתפלגות המורכבת של ​ נובע מהאופי הלא דטרמיניסטי והלא ליניארי של רוב הנתונים בעולם האמיתי. נתונים אלו יכולים להיות בעלי תכונות של רעש, פיזור ואינטראקציות דינמיות שאינן מתוארות היטב על ידי התפלגויות פשוטות. דגימה מהתפלגות כזו יכולה להיות מורכבת מאוד ולדרוש כמות גדולה של חישובים. בניגוד לכך, התפלגות נורמלית רב-ממדית היא יחסית פשוטה להבנה ולדגימה, מכיוון שכל אלמנטים בהתפלגות זו נתונים לתיאור סטטיסטי בסיסי.

בשימוש ב-KL Divergence כחלק מה-Objective ב-VAEs, אנו מצליחים למצוא התפלגות נורמלית שהיא הקרובה ביותר להתפלגות המקורית המורכבת. זהו למעשה קירוב שמאפשר לנו ליצור פתרון חישובי יעיל וממוקד. כשאנו מבצעים דגימה מהתפלגות הנורמלית המקורבת, אנו מוצאים פתרון שהוא אמנם לא מושלם, אבל הוא מקנה לנו הבנה ובקרה טובה יותר על התהליך הכולל.

המשמעות העמוקה של תהליך זה היא שאנו יכולים ליצור מודלים שמנצלים את המידע הלטנטי העמוק של הנתונים, ולא רק את המאפיינים הפניים שלהם. זה מאפשר ל-VAE להיות לא רק כלי לדחיסת נתונים, אלא גם כלי ליצירת נתונים חדשים באופן שמחקה את המציאות בצורה טבעית ומשכנעת.

A diagram of a graph

Description automatically generatedA diagram of a graph

Description automatically generated

בשתי התמונות אנו יכולים לראות במרחב הלטנטי 2 התפלגויות. ההתפלגות השחורה היא זו המתארת את התפלגות הוקטור הלטנטי כתלות ב- ומוגדרת **.** כמו כן ההתפלגות האדומה מייצגת את ההתפלגות הנורמלית הרב מימדית המוגדרת **.**

בתמונה השמאלית ניתן לראות כי ההתפלגויות במרחק הלטנטי מרוחקות זו מזו באופן יחסי דבר הגורם לקבלת פרדיקציות רחוקות זו מזו. עם זאת, כאשר אנו מקרבים את ההתפלגויות כפי שניתן לראות בתמונה הימנית באמצעות כלילת רכיב הנורמליזציה הזה בפונקציתת הObjective ניתן לראות כי גם התפלגויות הפרדיקציות מתקרבות זו לזו לתיאור טוב יותר של המציאות ביחס לקלט **.**

(בעמודים 10,11,12 יש הסברים נלווים להבנה טובה יותר של הנושא – להמשיך רציף אפשר לעבור לעמוד 13)

**המשמעות של Maximum Likelihood Estimation**

<https://www.youtube.com/watch?v=Pwgpl9mKars>

ננצל את הבמה על מנת לדון בחשיבות של MLE כפונקציית loss עבור מודלי למידת מכונה בכלל ועבור VAEs ומדידת הReconstruction Loss בין ל-*בפרט.*

כאשר אנו מדברים על הlikelihood, הנראות, אנו למעשה מדברים על הנראות של הפרדיקציות על מול הלייבלים האמיתיים. הנראות יכולה להשתקף במספר אופנים אך המשותף לכל האופנים הללו הוא שהם כולם מציינים מידה של התאמה בין פרדיקציות המודל לנתונים. ככל שהפרדיקציות שהמודל מייצר מתאימות יותר לתיוגים או לנתונים האמיתיים שאנו נתקלים בהם, כך הנראות תהיה גבוהה יותר, ולכן המודל נחשב למדויק יותר.

פונקציית log loss (נקראת גם NLL) מודדת את הפער בין התפלגויות ההסתברויות לבין התיוג האמיתי בבעיות סיווג. ניתן לחשב אותה באמצעות הנוסחה הבאה:

נשים לב כי המשוואה מכילה שני חלקים:

– מבטא את הענישה על הערך החזוי כאשר התווית האמיתית היא 1

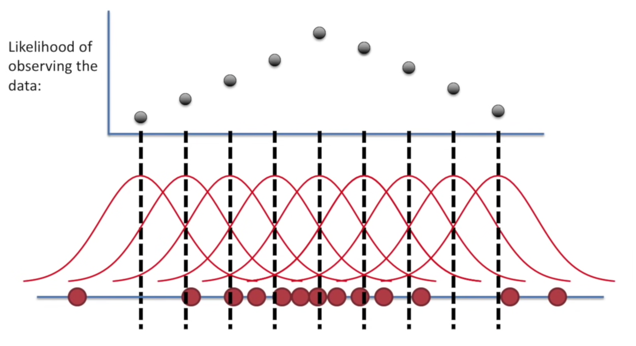
– מבטא את הענישה על הערך החזוי כאשר התווית האמיתית היא 0

שני החלקים הללו נדרשים כדי להבטיח שהloss יתואר באופן מדויק לשתי האפשרויות הבינאריות של התיוג, ורק החלק הרלוונטי יתווסף לחישוב ההפסד עבור כל תצפית. תחת ההנחה שאחד משני חלקי המשוואה יהיה שווה ל-0 ולכן תהיה התעלמות מהחלק שאינו רלוונטי לתיוג הנתון.

ניתן להרחיב את התיאור באותו אופן למספר מחלקות ולא רק לקלסיפיקציה בינארית:

דוגמא:

נניח ויש לנו 3 מחלקות [A,B,C] ונניח שעבור איטרציה מסוימת התיוג הוא A כלומר הווקטור הוא [1,0,0]. כמו כן נניח והתפלגות ההסתברויות היא [0.7,0.2,0.1] אז נקבל כי:



**Maximization of Maximum Log Likelihood in VAE**

עד כה תיארנו את הLikelihood, הנראות, עבור בעיות קלסיפיקציה (בינארית וmulti-class) אך כעת נרצה לתאר את מושג הנראות שישמש אותנו בVAEs.

במודלים כמו VAE, המטרה היא מיקסום הlog likelihood של הפלט המשוחזר בהינתן המודל בהשוואה לקלט. לכן ניתן לתאר את הObjective באופן הבא:

ניתן לפשט את הביטוי באופן בו אנחנו יכולים להתייחס לחלק ממנו כאל חלקים קבועים שאינם משתנים ולכן נקבל:

דוגמא:

ניתן כעת דוגמא לחישוב של Reconstruction Loss וכמו כן נסביר את החשיבות של מקסימיזציה של ערך זה ואיך הוא בא לידי ביטוי מבחינה מתמטית.

נניח ויש לנו אופטימיזציה עבור פיקסל יחיד המכיל ערך יחיד וכמו כן נניח ואנחנו מספקים 2 פרדיקציות. נסכם כעת את הנתונים:

Original Pixel Value – **0.5**

Reconstruction Pixel Value Case 1 – **0.6**

2Reconstruction Pixel Value Case – **0.52**

נבצע כעת חישוב עבור הערך המוגדר כמינוס ערכי הReconstruction Loss () עבור כל אחד מהמקרים:

כך, על ידי מיקסום ה-log likelihood ב-VAEs ומינימיזציה של ה-Reconstruction Loss, אנו למעשה מבטיחים שהתמונות המשוחזרות מתקרבות ככל האפשר לתמונות המקוריות, דבר המאפשר שיפור ניכר בדיוק השחזור ובאיכות המודל הכללית.

* נקודה חשובה היא שהטרמינולוגיה (יש לומר המבלבלת) היא שאנו רוצים לדחוף למיקסום הנראות (Maximize Likelihood) – הגדרה שקולה היא שאנחנו רוצים לדחוף למינמום ערך הNegative Log Loss. למעשה יש שקילות בין מיקסום הנראות לבין מינמום ערכי הNLL.

**פירוט על KL Divergence**

<https://www.youtube.com/watch?v=SxGYPqCgJWM>

KL Divergence הוא מדד מתמטי המודד את המרחק בין שתי התפלגויות הסתברותיות.

בהינתן שתי התפלגויות P, Q ועבור משתנה מקרי X=x נראה כי המשוואה מוגדרת באופן הבא:

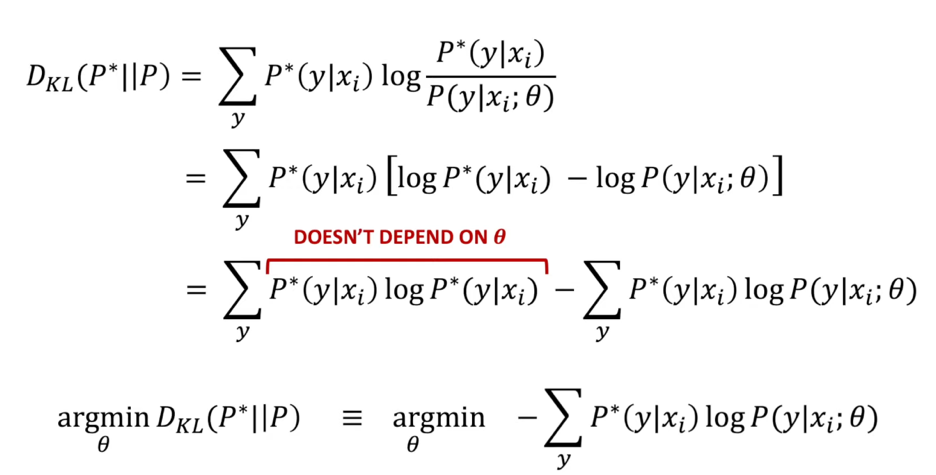
באופן עקרוני P, Q יכולות להיות 2 התפלגויות כלשהן אך לטובת התקשרות לעולם המושגים שלנו ולטובת ההסבר נוכל לדמות את P להתפלגות האמיתית (הOracle) וQ להתפלגות ההסתברויות כפלט של מודל כלשהו.

נוכל להבחין ב2 נקודות מעניינות:

1. – עבור כל ערכי הדגימות בהינתן וערכי הסתברות של התפלגות הפרדיקציות שווה לערכי התפלגות האמיתית נקבל כי log = 0 ולכן מדד ה KL Divergence יהיה 0 כלומר ההתפלגויות זהות.
2. כאשר ערך בהתפלגות האמיתית הוא 0, תרומתו למדד KL Divergence היא אפס, מכיוון שהמונה בחישוב הופך לאפס. זה מדגיש שרק הנקודות שבהן ההתפלגות האמיתית מציגה ערכים חיוביים תורמות לחישוב ה-Divergence, ומאפשרת התמקדות באי-התאמות המשמעותיות בין ההתפלגויות.

שקילות KL Divergence and Log Loss:

השקילות בין פונקציית הlog loss למדד KL Divergence נובעת מהמבנה המתמטי הדומה של שניהם. שתי הפונקציות מודדות את ההבדל בין התפלגויות הסתברותיות, אך מתוך זוויות שונות. פונקציית הlog loss מחשבת את הInformation loss כאשר מודל מייצג נכון או לא נכון נתוני קלט, בעוד ש-KL Divergence מודדת את ה"מרחק" הכולל בין ההתפלגות המודלית לבין ההתפלגות האמיתית. השקילות נכונה מכיוון שבשתי המקרים, אנו מחפשים למזער את הטעויות בחיזוי של המודל ביחס למציאות הקיימת. כאשר ההפסד הלוגריתמי ממוזער, ההתפלגות שהמודל מציע קרובה יותר להתפלגות האמיתית, מה שמוביל למינימום של KL Divergence ולכן לשקילות בין שתי המדדים. הפיתוח המתמטי מצוי בתמונה הבאה:



חשוב לציין כי בתמונה P\* היא התפלגות הOracle (אנו ציינו בתור P) והתפלגות ההסתברויות היא P (אנו ציינו כQ)

**VAE Likelihood**

אז תיארנו את הObjective בVAE ככזה התלוי בשני חלקים. על החלק השני המודד את המרחק בין ההתפלגויות הרחבנו בפירוט בעמודים הקודמים.

כעת נרצה להתמקד בחלק הראשון של הObjective המוגדר .

נשאלת למעשה השאלה איך ניתן לחשב אותו. על מנת לענות על השאלה הזו נרצה לפשט את הבעיה ולומר כי נניח ולווקטור הלטנטי h היו שני ייצוגים וקטורים בלבד: Vec A, Vec B.

אזי היינו יכולים לומר כי התיאור הבא מתאר את ההסתברות:

כמו כן בהינתן ולווקטור הלטנטי h היו n ייצוגים ווקטורים היינו יכולים להכליל ולומר כי החישוב מתואר:

עם זאת מאחר וh הוא ייצוג לטנטי מתוך מרחב כלשהו ממימד גבוה אזי מתקיים כי הוא יכול להכיל אינסוף ערכים ולכן יש צורך לחשב את ההסתברות באופן הבא תחת ההנחה כי היא רציפה:

הסבר פשוט לאופן חישוב ההסתברות בצורה רציפה:

כשכתבתי את הסיכום, מצאתי את אופן חישוב ההתפלגות למבלבל מעט ולכן החלטתי לכלול גם דוגמא פשוטה יותר שתמחיש את העקרון המנחה באופן חישוב זה.

נניח ואנו רוצים לצלם תמונה של גינה המוגדרת התלויה בזמן כלשהו time. אילו לזמן היו שני ערכים בלבד היינו יכולים לתאר את ההסתברות בתור:

עם זאת, מאחר וזמן הוא משתנה מקרי רציף אזי נצטרך לחשוב על חישוב ההסתברות בצורה רציפה:

על פניו היינו רוצים לתחום את החישוב ב כלומר תחימה וחישוב הסתברות על סמך מסגרת זמן כלשהי. אך למעשה באופן בו אנו מחשבים את ההסתברות בצורה על כל המרחב אנו למעשה יוצרים חישוב רציף עבור כל רגע בזמן.

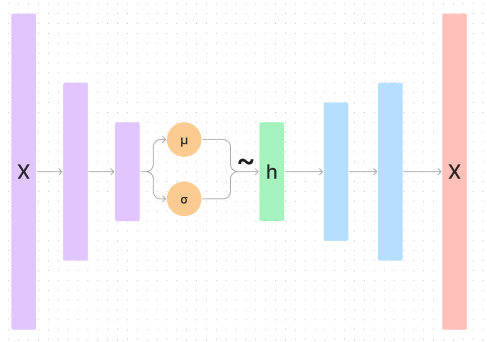
**בעייתיות בחישוב ההסתברות בהגדרת הObjective**

כאמור, הגדרנו באופן הבא את ההסתברות:  **. נשים לב שבאופן חישוב זה אנחנו מגדירים את ההסתברות להיות תלויה באופן חישוב רציף המוגדר על כל המרחב ועל כל הערכים שהווקטור h יכול לקבל, על זה מתבצעת האינטגרציה. מבחינה פרקטית וחישובית דבר זה אינו ישים מהסיבה שהמרחב הוא מרחב רציף המוגדר** .

לפיכך נרצה להגדיר את ההסתברות להיות ככזו שאינה מחושבת על פני כל המרחב אלא למעשה נדגמת מתוך התפלגות נורמלית. ערכי ההתפלגות הנורמלית מוגדרים להיות ככאלו המחושבים על סמך רשמת נוירונים שהמטרה שלה זה למצוא את ערכי ההתפלגות המספקים רפרזנטציות התואמות לתצפיות. נשים לב כי ערכי ההסתברות של הEncoder מוגדרים באופן הבא:

*באופן זה ניתן לשים לב כי אנחנו יכולים להגדיר את ה*Objectiv*e עבור ה*VAE *כולו באופן מעט שונה התלוי ב* *כך ש**כשלעצמו נדגם מהתפלגות נורמלית ולכן אין צורך בחישוב האינטגרציה:*

*נשים לב כי אנו למעשה משערכים את האינטגרל באופן בו אנו דוגמים בכל פעם את* *מתוך התפלגות גאוסיינית המורכבת מאוסף של הנלמדים באמצעות רשת נוירונים. ההנחה היא שאומנם אנו דוגמים בכל פעם דגימה יחידה מתוך ההתפלגות הנלמדת אך מאחר והתהליך מתבצע פעמים רבות, בתוחלת אנחנו נקבל קירוב לערך האינטגרל.*

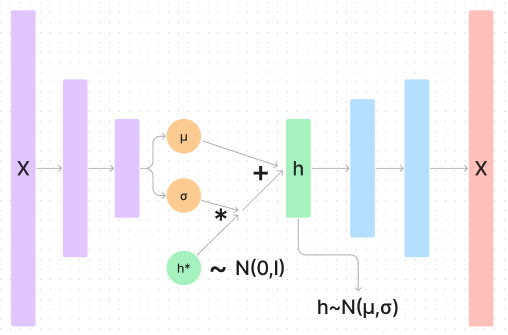
**

**בעיית חישוב הגראדיינט ו- Reparameterization Trick**

*אז אמנם פתרנו את הבעיה של חישוב באינטגרל באמצעות דגימה מהתפלגות גאוסיינית שבתוחלת מתכנסת לתיאור ההתפלגות האמיתית אך כעת אנחנו נתקלים בבעיה נוספת.*

*הבעיה הנוספת היא שאנחנו דוגמים מהתפלגות נלמדת כלשהו במהלך הפרופוגציה לאורך הרשת – דבר המוביל לבעיה בעת חישוב הגראדיינט לטובת עדכון המשקולות.* *בפועל, מאחר ופעולת הדגימה היא פעולה רנדומית היא לא נחשבת לפעולה חלקה מתמטית ולפיכך הנגזרת אינה מוגדרת.*

*הפתרון לבעיה הוא שימוש ב*Reparameterization Trick *באופן בו אנו נדגום תחילה מתוך גאוסיין ״טיפש״ שלא מכיל שום ערכים נלמדים ואל ערך הדגימה נוסיף בצורה אריתמטית את ערכי*

**

*ניתן לראות כי מתבצעת דגימה ולאחר מכן יש הכפלה ב והוספת לקבלת ערך דגימה המשקף דגימה מתוך כפי שהיינו רוצים.*

*כמו כן, מאחר ופעולות הכפל ופעולות החיבור הינן פעולות אריתמטיות אזי גזירות ולפיכך ניתן לחשב עבורן גראדיינט ולפעפע את ערכיו לאורך הרשת בתהליך ה* Backpropagation

**אופן ביצוע ה Inference**

*כאמור, חלק מה*Objective *כולל קירוב התפלגויות בין באמצעות* KL Divergence. *המשמעות של הדבר היא שבעת דגימה בשלב ה* Inference *ניתן לדגום ווקטור לטנטי* ***h*** *מתוך גאוסיין ״טיפש״ תחת ההנחה כי הרפרזנטציות כבר נלמדו במהלך תהליך האימון, ולכן המודל מסוגל להפיק פלטים משמעותיים גם מנקודות שבהן לא נראו דוגמאות במפורש במהלך האימון. זהו מרכיב מרכזי ביכולת הגנרטיבית של המודל, שמאפשר לו ליצור על פניו דגימות חדשות שלא נראו מעולם מתוך התפלגות שקרובה להתפלגות המקורית.*

**Generative Adversarial Network – GAN**

<https://www.youtube.com/watch?v=DWVlEw0D3gA>

*עד כה דיברנו על* VAE *כמודל ג׳נרטיבי המסוגל לדגום וקטור לטנטי מתוך התפלגות, להעביר את* *הווקטור דרך* Decoder *המכיל שכבות נלמדות ולייצר מהן דגימות חדשות הלקוחות מההתפלגות.*

*כשאנו מדברים על* GAN*, או* Generative Adversarial Network*, אנו מתייחסים למודל ג'נרטיבי שפותח בשנת 2014.* GAN *מורכב משני רכיבים עיקריים (*Generator, Discriminator*) שמתחרים זה בזה במהלך האימון. השניים יוצרים דינמיקה שבה כל אחד מהם משפר את ביצועיו בתגובה לשינויים של השני. זוהי טכניקה יעילה מאוד לייצור דגימות חדשות שנראות כאילו הן לקוחות מהתפלגות הנתונים המקורית.*

1. Generator – *הגנרטור הוא רכיב האחראי לייצור הדגימות החדשות. המטרה של הגנרטור היא ללמוד כיצד לייצר דגימות שנראות כמה שיותר אמינות ואמיתיות, כלומר דומות לדגימות מהתפלגות הנתונים האמיתית שממנה הוא מנסה ללמוד. הגנרטור מקבל כקלט וקטור לטנטי של רעש אקראי (לרוב מתוך התפלגות נורמלית סטנדרטית) ומשתמש ברשת נוירונים כדי להמיר את הרעש הזה לדגימה שמדמה דגימה אמיתית*
2. Discriminator – *הדיסקרימינטור הוא הרכיב השני ב-*GAN*, ותפקידו להבחין בין דגימות אמיתיות לבין דגימות שהופקו על ידי הגנרטור. זהו למעשה* Classifier *שמנסה לקבוע אם דגימה נתונה היא "אמיתית" (לקוחה מהתפלגות הנתונים המקורית) או "מזויפת" (נוצרה על ידי הגנרטור). הדיסקרימינטור גם כן מבוסס על רשת נוירונים ומשתפר במהלך האימון ככל שהוא נחשף יותר ויותר לדגימות משני הסוגים.*

*במהלך האימון, הגנרטור והדיסקרימינטור מתחרים זה בזה באופן בו הגנרטור מנסה להונות את הדיסקרימינטור על ידי יצירת דגימות הדומות כמה שיותר לאמיתיות, והדיסקרימינטור משתפר ביכולתו להבחין בין אמיתי למזויף. המטרה הכללית היא להגיע לשיווי משקל, שבו הגנרטור יוצר דגימות שהדיסקרימינטור לא יכול להבחין בהן מדגימות אמיתיות, מה שמעיד על כך שהגנרטור למד היטב את התפלגות הנתונים.*

**אופן פעולת ה Generator**

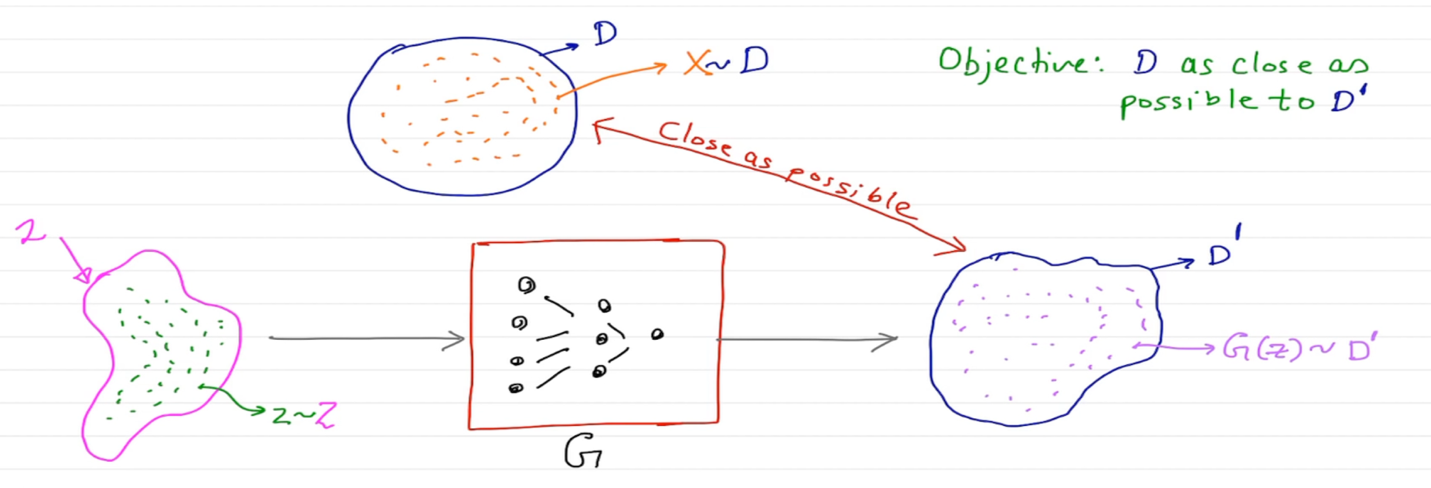
על מנת להבין את אופן פעולת הגנרטור, נרצה לתאר את ההתפלגויות המעורבות בפעולת הגנרטור:

1. Z – ניתן להסתכל על Z כעל מרחב לטנטי המתקבל מהתפלגות רנדומית (למשל גאוסיין)
2. D – ההתפלגות המקורית שמתארת את התצפיות העולם האמיתי
3. D’ – ההתפלגות המתקבלת כתוצאה מהעברת סך כל הווקטורים הלטנטיים בגנרטור לאחר תהליך האימון

כדי להמחיש את ההתפלגויות ניתן דוגמאות ממשיות לכל אחת מההתפלגויות על בסיס MNIST:

1. Z – ההתפלגות מתארת ווקטורים ממימד כלשהו הקטן ממימד התמונות בMNIST
2. D – התמונות בMNIST, כל תמונה x~D לקוחה מתוך ההתפלגות המתוארת כ-Dataset כולו
3. D’ – התמונות שהתקבלו כתוצאה מהעברת סך הווקטורים הלטנטיים בגנרטור. למעשה מתקבלות תמונות שדומות לתמונות הלקוחות מההתפלגות המקורית אך לא זהות אליהן לחלוטין

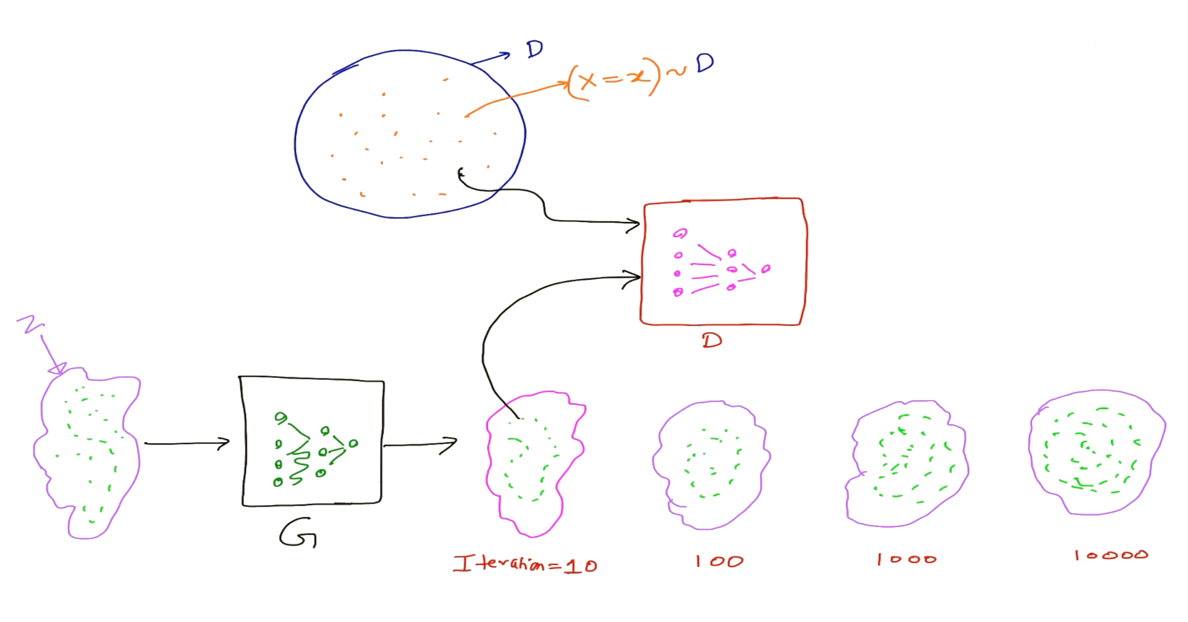
בתרשים הבא ניתן לראות את אופן פעולת רכיב הגרנרטור ב-GAN:

**

*תחילה ניתן לראות כי אנו דוגמים מתוך ההתפלגות הרנדומית. לאחר מכן* z *עובר לאורך כל שכבות הגנרטור כך שמתקבל המהווה דגימה מתוך ההתפלגות D’ שהגנרטור מייצר באופן פעולתו. המטרה היא להביא לקירוב בין ההתפלגויות D’ ל D דבר שיעיד על כך שהגנרטור מצליח לייצר דגימות אשר נראות כלקוחות מתוך ההתפלגות האמיתית ובכך יצליח להערים על הדיסקרימינטור כפי שיפורט בהרחבה בהמשך.*

**Overview על אופן פעולת GAN לאורך האיטרציות**

לאחר שתיארנו את אופן פעולת הגנרטור והאופן בו הוא מנסה לייצר התפלגות שדומה ככל הניתן להתפלגות המקורית, נרצה להדגיש כי התהליך הזה מתבצע בצורה איטרטיבית לאורך תהליך האימון. כמפורט בתמונה, תחילה ההתפלגות שמתהווה על ידי הגנרטור היא התפלגות שרחוקה מההתפלגות המקורית. לאורך תהליך הלמידה עם ההתקדמות באיטרציות, הגנרטור לומד רפרזנטציות מדויקות יותר של ההתפלגות המקורית בהתאם להחלטות הדיסקרימינטור.



נרצה לפרוט מספר נקודות שיובילו להעמקת ההבנה על האופן בו הארכיטקטורה פועלת:

1. מהסיבה שבעת תהליך האימון קיימים בפנינו דגימות מההתפלגות המקורית וכמו כן, קיימת גם דגימות שג׳ונרטו על ידי הגנרטור, אנו יכולים לתייג את הדגימות מההתפלגות המקורית כאמיתיות (Real) ואת הדגימות שיוצרו על ידי הגנרטור כמזויפות (Fake)
2. בשלב הראשון אנו רוצים לאמן את הדיסקרימינטור לסווג את הדגימות לאמיתיות ומזויפות. למען האמת זה משימת קלסיפיקציה קלה מהסיבה שהגנרטור מפיק דגימות שאינן קרובות כלל להתפלגות המקורית מהסיבה שהוא עדיין לא עבר תהליך אימון. לאחר סיום שלב זה יש לנו דיסקרימינטור שיודע לסווג את הדאטא בצורה דיי משכנעת
3. בשלב הבא אנחנו מתחילים באיטרציות האימון עבור הגנרטור באופן בו הדגימות שמיוצרות על ידו נראות יותר ויותר ככאלו הלקוחות מתוך ההתפלגות המקורית ככל שמתקדמים באיטרציות. האופן בו הגנרטור לומד הוא על סמך הloss שמחושב לאחר קלסיפיקציית הדיסקרימינטור, ככל שהדיסקרימנטור יהיה מסוגל להבחין בקלות רבה יותר בדגימות המזויפות שנוצרו על ידי הגנרטור, כך צעד העדכון בBackpropagation יהיה חזק יותר
4. נמשיך את תהליך האימון לאורך האיטרציות ובכך נקרב את התפלגות הגנרטור D’ להתפלגות המקורית D. (כפי שרואים בתמונה היא הולכת ומתעגלת)

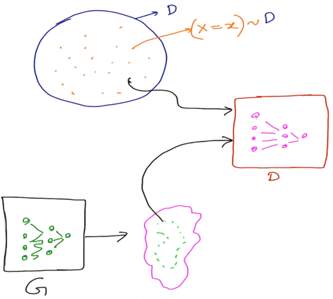
נקודות חשובות לשים אליהן לב:

1. האימון של הדיסקרימינטור ושל הגנרטור יתבצע בנפרד באופן בו הדיסקרימינטור יאומן בנפרד בעוד הגנרטור מקובע, והגנרטור יאומן בנפרד בעוד הדיסקרימינטור מקובע.
2. המטרה לאורך תהליך האימון הוא שהדיסקרימינטור יספק ערכי פרדיקציה זהים עבור שתי המחלקות – האמיתית והמזויפת. ערכי הפרדיקציה יעמדו על 0.5 בקירוב ויעידו על כך שהגנרטור מצליח לייצר דגימות מתוך ההתפלגות שיצר שדומות מאוד להתפלגות המקורית.

**Discriminator Loss Function and Objective**

ה Loss Function של הדיסקרימינטור הוא Binary Cross Entropy המשמש לרוב כ Loss קלאסי לבעיות קלסיפיקציה בינארית.

ב Binary Cross Entropy נראה כי מייצג את התיוג האמיתי ו מייצג את ערך הפרדיקציה שבמקרה שלנו מקורה בפרדיקציית הדיסקרימינטור.



במקרה שלנו הדיסקרימינטור מספק פרדיקציות עבור אחד משני מצבים:

1. הדיסקרימינטור מספק פרדיקציות עבור דגימות הלקוחות מההתפלגות האמיתית – במקרה זה ערך הוא 1



1. הדיסקרימינטור מספק פרדיקציות עבור דגימות הלקוחות מההתפלגות הגנרטור – במקרה זה ערך הוא 0

* חשוב לצין כי כעת כשנרשום D הכוונה היא לדיסקרימינטור

לפיכך, ניתן לתאר את פונקציית ה Loss במקרה הזה ככזו התלויה בשני מקרים אלו:

מקרה 1 – הפרדיקציות לקוחות מההתפלגות האמיתית -

עבור מקרה זה נקבל כי למעשה פונקציית ה Loss מוגדר כ באופן בו ערך הפרדיקציה הוא ערך ההחזרה של הדיסקרימינטור על x הלקוח מתוך ההתפלגות המקורית. כשנטעין במשוואה נקבל:

מקרה 2 – הפרדיקציות לקוחות מהתפלגות הגנרטור -

עבור מקרה זה נקבל כי למעשה פונקציית ה Loss מוגדר כ באופן בו ערך הפרדיקציה הוא ערך ההחזרה של הדיסקרימינטור על z הלקוח מתוך התפלגות הגנרטור. כשנטעין במשוואה נקבל:

נראה כי קיבלנו ביטוי ל Objective של הדיסקרימינטור המורכב מ2 ביטויים. נראה כי המטרה של הדיסקרימינטור הוא למקסם את ערך שני הביטויים על מנת להיות אופטימלי.

הסיבה למקסימיזציה נעוצה ברצון למנמם את ה Loss באמצעות מקסימיזציה של ה Objective, למעשה הדיסקרימינטור צריך לספק ערכי 1 עבור כל הערכים הלקוחים מההתפלגות האמיתית וערכי 0 עבור כל הערכים הלקוחים מהתפלגות הגנרטור דבר היוביל לתרחיש הבא:

נשים לב כי כל ערך אחר שאינו 1 עבור פרדיקציית הדיסקרימינטור עבור ההתפלגות המקורית ו-0 עבור פרדיקציית הדיסקרימינטור עבור התפלגות הגנרטור הייתה מובילה לכדי ערך Loss גדול מ-0.

סך הכל כי ה-Objective של הדיסקרימינטור מוגדר להיות:

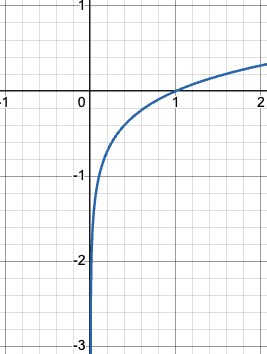
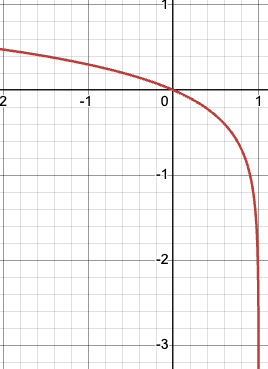
**משמעות גרפית של ה-Discriminator Objective**

לאחר שתיארנו בצורה מילולית וסיפקנו מוטיבציה למיקסום ה-Objective כתנאי הכרחי למינימיזציית ערך ה-loss והתכנסות הדיסקרימינטור נרצה כעת לספק תיאור גרפי.

log D(x)

log (1-D(G(z)))

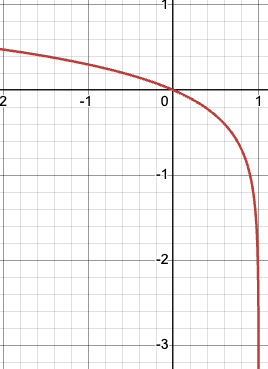


הגרפים המצויים לשמאל מתארים את הפונקציות המתמטיות הקשורות בObjective.

ניתן לראות כי הגרף הימני מתאר את הביטוי התלוי בערכי פרדיקציית הדיסקרימינטור עבור ההתפלגות המקורית. מהסיבה שהפונקציה מונוטונית עולה וחסומה בין 0 ל-1 נראה כי היא מקבל ערך מקסימלי כאשר ערך הפרדיקציה הוא 1. כמו כן הגרף השמאלי מתאר את הביטוי התלוי בערכי הפרדיקציה עבור התפלגות הגנרטור. מהסיבה שהפונקציה מונוטונית יורדת חסומה בין 0 ל-1 אזי היא מקבלת ערך מקסימלי כאשר הפרדיקציה 0.

לפיכך, על סמך התצוגה הגרפית ועל סמך התיאור המילולי ניתן לומר כי סיפוק פרדיקציות של 1 עבור ההתפלגות המקורית ו-0 להתפלגות הגנרטור תוביל להשאפת ערכי הפרדיקציה אל עבר הנקודות הירוקות ובכך תוביל למיקסום Objective הדיסקרימינטור.

**Genrator Loss Function and Objective**

לאחר שתיארנו את ה-Objective עבור הדיסקרימינטור נרצה לתאר את



log (1-D(G(z)))

ה-Objective עבור הגנרטור. על סמך אותו הגיון שהנחה אותנו למקסם את

ה-Objective של הדיסקרימינטור, נראה כי הרצון כעת הוא למנמם את

ה-Objective של הגנרטור. הנחת היסוד היא שהמטרה של הגנרטור היא להערים על הדיסקרימינטור באופן בו הדיסקרימינטור יתבלבל ויחשוב שהדגימות שמיוצרות על ידי הגנרטור הן אמיתיות ולכן ייתן להן ערכי פרדיקציה גבוהים ושואפים ל-1. מטעינת 1 בערך המשוואה נקבל את הביטוי הבא:

נשים לב כי במקרה בו הדיסקרימינטור טועה בפרדיקציה שלו על דגימה מתוך התפלגות הגנרטור הוא יספק פרדיקציה שתוביל להתאפסות הערך שמצוי בתוך ה-log ולכן ערך הלוג ישאף ל. במילים אחרות, ככל שהגנרטור יצליח להערים יותר על הדיסקרימינטור כך הוא יוביל לערכי פרדיקציות שמתאימים להתפלגות המקורית ולכן מטרת הגנרטור היא למנמם את ערכי ה-Objective.

לסיכום, ניתן לומר כי ה-Objective של הגנרטור מוגדר להיות:

**GAN Objective**

לאחר שפירטנו על ה-Objective של הדיסקרימינטור והגנרטור בנפרד נרצה לתאר את ה-Objective המשותף שלהם – ה-Objective הכולל של GAN אשר מוגדר להיות מבחינה לא פורמלית:

מבחינה מילולית כפי שתואר, מטרת הדיסקרימינטור היא מיקסום ה-Objective ומטרת הגנרטור היא מיקסום ה-Objective ולכן ה-Objective הכללי של GAN מתייחס ל-Objective של שני הרכיבים.

במאמר המקורי עם זאת ה-Objective תואר באופן הבא:

ניתן לראות כי הכתיב מעט שונה אבל המשמעות זהה ודומה למשמעות ב-Objective הכולל שתואר.

כמו כן נשים לב לשקילות כלומר הפונקציה V מתארת את ביטויי התוחלות המופיע בצד ימין.

כעת נסתכל על שני הביטויים הנוספים:

1. – הביטוי מייצג את התוחלת של הלוגריתם של הסיכוי שהדיסקרימינטור יחשוב שדגימה x שנלקחה מההתפלגות של הנתונים האמיתית היא אמיתית.
2. – הביטוי מייצג את התוחלת של הלוגריתם של ההסתברות שהדיסקרימינטור לא יצליח לזהות את שהדגימה שהגנרטור ייצר מהווקטור z הנלקח מההתפלגות הרנדומית היא מזויפת.

* הנקודה החשובה לקחת היא שכל תכליתו של ה-Objective היא משחק ה-min max שקיים בין הדיסקרימינטור לגנרטור באופן בו מטרת הגנרטור היא להערים על הדיסקרימינטור באופן בו הוא יתקשה לזהות את הדגימות שיצר מההתפלגות הרנדומית ויחשוב שאלו נוצרו מההתפלגות המקורית. מצד שני, מטרת הדיסקרימינטור היא לגלות מה הן הדגימות המזויפות שיוצרו על ידי הגנרטור.
* בתהליך איטרטיבי של שיפור הדיסקרימינטור והגנרטור באופן נפרד, השיפור הנפרד של כל אחד מהם מוביל לשיפור גם ביכולותיו של האחר כך שהמטרה הסופית היא שיפור משמעותי ביכולותיו של הגנרטור באופן בו הוא בסופו של דבר יצליח לייצר דגימות הדומות ככל הניתן להתפלגות המקורית.

**Optimal Discriminator**

אז כאמור, כפי שהזכרנו מספר פעמים ישנו קרב מתמיד בין הדיסקרימינטור לגנרטור. נרצה להגדיר מושג חדש: דיסקרימינטור אופטימלי. הדיסקרימינטור אופטימלי הוא הדיסקרימינטור אליו מגיעים בהינתן גנרטור מקובע. יש לציין כי השם כאן מעט מבלבל אבל האופטימלית של הדיסקרימינטור היא ביחס ל-GAN כמודל מאומן ולא ביחס לדיסקרימינטור כשלעצמו כאופטימלי. כלומר, אין הדבר שהדיסקרימינטור אופטימלי מהסיבה שהסיווג שלו מושלם.

ניגש להגדרה המתמטית, בהינתן גנרטור מקובע, דיסקרימינטור מקובע יקיים:

לפני שנצלול להוכחה המתמטית נרצה דווקא לספק מוטיבציה שתסביר את התלות של דיסקרימינטור אופטימלי בקיום התלות.

לטובת יישור הקו נתאר את שני הביטויים הבאים:

1. – צפיפות ההסתברות של הדיסקרימינטור עבור הדגימות האמיתיות
2. – צפיפות ההסתברות של הדיסקרימינטור עבור דגימות שיוצרו על ידי הגנרטור

עבור מצב בו נקבל כי כלומר עבור הדגימה הספציפית הדיסקרימינטור האופטימלי משערך כי בהסתברות של 80% הדגימה שייכת להתפלגות האמיתית. ככל שהערך קרוב יותר ל-1 ככה הדיסקרימינטור מצביע על כך שהדגימה שייכת להתפלגות האמיתית וככל שהערך קרוב יותר ל-0 כך הדיסקרימינטור מצביע על כך שהדגימה מזויפת.

באופן כללי נרצה לקבל עבור רוב הדגימות, דבר שיצביע על כך שהדיסקרימנטור ״מבולבל״ והוא נוטה לשערך את הדגימות האמיתיות והמזויפות באותו האופן דבר שירמז על כך שהגנרטור הצליח להערים על הדיסקרימינטור ומכאן שהגנרטור הצליח ללמוד רפרזנטציות נכונות של ההתפלגות המקורית.

***הוכחה מתמטית***

השאלה הנשאלת היא בהינתן גנרטור מקובע, עבור אילו ערכים מקבל ערך מקסימלי. ופורמלית:

עד כה אין חידוש, למעשה תיארנו את המשוואה המוכרת והשינוי היחידי שביצענו הוא לקיחת ערכי argmax מהסיבה שמה שמעניין אותנו זה אילו ערכים ממקסמים את הביטוי.

מהגדרת התוחלת נקבל: ולכן לאחר שנעשה פלאג לביטוי במשוואה נקבל:

גם כאן, אין חידוש ושום דבר מורכב, בסך הכל עשינו פלאג להגדרת התוחלת למשוואת המתארת את הפונקציה V.

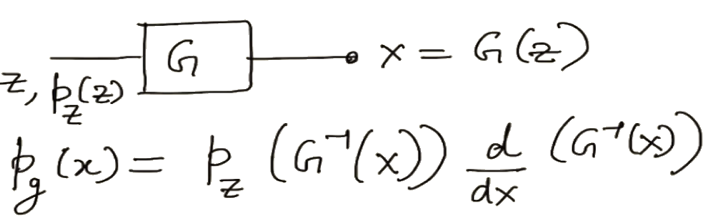
**למה 1**

על מנת להמשיך ולהתקדם בהוכחה נרצה להוכיח את השקילות הבאה על מנת לקבל ביטוי שתלוי ב-x בלבד ואינו תלוי ב-z.

השקילות הזו מתקיימת הודות לקונספט הנקרא ״החלפת משתנה״ והוא מוגדר באופן הבא:

בהינתן משתנה מקרי X המוגדר ניתן לחשב את פונקציית צפיפות ההסתברות של משתנה כלשהו אחר Y התלוי בX ומוגדר להיות באופן הבא: .

הסיבה לכך היא מאחר ש-f היא פונקציה הפיכה אזי הפעלתה על איברים מהטווח תוביל לקבלת איברים מהתחום. במקרה שלנו, למעשה נותן ערך ששקול ל-x כלשהו מהתחום.



במקרה שלנו, z מייצג את הווקטור שנלקח מההתפלגות הרנדומית בעלת פונקציית צפיפות ההסתברות ומועבר אל הגנרטור לצורך קבלה של דגימה מתוך התפלגות הגנרטור.

מהסיבה שהגנרטור הוא בסופו של דבר רשת נוירונים ולפיכך על פניו ניתנת לפישוט כפונקציה הפיכה (מבחינה תיאורטית ניתן להרצה מהסוף להתחלה) אזי ניתן להחיל את הקונספט של ״החלפת משתנה״ ובכך לומר כי .

* דבר שחשוב לציין הוא ש הוא לא x הלקוח מתוך ההתפלגות המקורית אלא דגימה שיוצרה על ידי הגנרטור. אנו מסוגלים לעשות את הקירוב הזה תחת ההנחה שבסוף תהליך האימון התפלגות הגנרטור וההתפלגות המקורית קרובות באופן יחסי.

כעת מה שנרצה לעשות, באמצעות ההבחנה שביצענו נרצה לצאת מ ולהגיע לביטוי השקול המוגדר כ

כמה דגשים שחשוב לשים אליהם לב:

1. המעבר בין ל מתאפשר בעקבות ההגדרה ש *ומכאן*
2. המעבר בין ל מתאפשר בעקבות ההגדרה ש
3. המעבר בין ל מגיע בעקבות כלל השרשרת. שהרי בהכללה

הדבר הבא שנרצה לעשות הוא לקחת את שני הביטויים ולהגיע באמצעותם אל השקילות הסופית:

1. .

מאחר החלק הימני של הביטוי אינו תלוי ב וכמו כן מאחר ופעולת הכפל הינה פעולה קומוטטיבית אזי ניתן לכתוב את הביטוי בצורה שונה באופן הבא:

נוכל לשים לב כי החלק האדום שקול ל ולכן סך הכל נקבל:

**סוף למה 1**

לאחר שהוכחנו את הלמה נרצה להמשיך ולהתקדם בהוכחת הדיסקרימינטור האופטימלי.

הביטוי הבא הוא ביטוי שאנו כבר מכירים ומתאר את החישוב של התוחלות:

נחליף את ב (זו השקילות שהוכחנו בלמה 1) ונקבל:

מאחר ומדובר באינטגרל מעל x ניתן לכנס את האינטגרלים ולקבל:

על מנת לקבל ערך עבור הדיסקרימינטור האופטימלי נרצה לבדוק מה הערך שממקסם את הערך שמצוי בתוך האינטגרל: . לשם כך נרצה לגזור את הביטוי הזה ונקבל:

* חשוב לציין כי יש לבצע בדיקה באמצעות נגזרת כפולה כדי לוודא שאכן מדובר בנקודת מקסימום ולא בנקודת במינימום.

לאחר שנכנס איברים נקבל את הביטוי הבא שכבר מוכר לנו:

סך הכל קיבלנו כי מיקסום ה-Objective מוביל לכדי דיסקרימינטור אופטימלי המקיים את הביטוי.

**Optimal Generator**

אז לאחר שדיברנו על הדיסקרימינטור האופטימלי נרצה לדבר כעת על הגנרטור האופטימלי. יש לציין שאופטימליות הגנרטור היא אף יותר חשובה בסופו של דבר מהסיבה שבפעול in inference time הוא המודל שייצר את הדגימות.

כפי שקיבענו את הגנרטור בתהליך ההסקה כלפי הדיסקרימינטור האופטימלי, כאן נקבע את הדיסקרימינטור באופן בו הדיסקרימינטור שנקבע הוא הדיסקרימינטור האופטימלי.

כאמור, בגנרטור יש רצון למנמם את ה-Objective ולכן המטרה היא למצוא גנרטור המקיים:

מהסתכלות על ביטוי התוחלות (בשילוב הלמה) נקבל:

כעת נעשה פלאג ל אל תוך המשוואה ונקבל:

על מנת להמשיך בפישוטים ולהגיע להתכנסויות בהמשך נרצה לבצע טריק מתמטי והוא הוספה והחסרה של לכל אחד מחלקי המשוואה:

* מה שמופיע בירוק הוא ההוספה וההחסרה של הביטוי תוך כינוס איברים

נבצע עוד כינוסים מתמטיים ונקבל:

נרצה לפרק את הביטוי ל3 אינטגרלים נפרדים לטובת המשך העבודה:

* פירוק האינטגרל לחלקים מתרחשת בעקבות תכונת לינאריות האינטגרל:
* כמו כן באינטגרל הראשון ישנה הוצאה של -log2 החוצה מהסיבה שהוא אינו תלוי ב-x

נמשיך ונבצע פישוט, נשים לב כי עבור 3 הרכיבים, רכיב 1 הוא ביטוי קבוע סקלארי. רכיבים 2,3 למעשה הם ביטוי של KL Divergence מהסיבה שהם מהצורה

המשמעות היותר עמוקה של

* המעבר בין ל מתקיים כתוצאה מהגדרת פונקציית צפיפות ההסתברות שהרי מתקיים כי
* המעבר עבור רכיבים 2,3 מתקיים באמצעות השקילות

לאחר שאכן זיהינו כי אכן מדובר ב-KL Divergence נוכל לכתוב את הביטוי באופן הבא:

כמו כן הביטוי הוא לא אחר מאשר JSD ולכן נקבל את הביטוי:

JSD (Jensen Shannon Divergence) הוא מדד המודד את המרחק בין שתי התפלגויות באופן סימטרי עליו יפורט מעט יותר בהרחבה בהמשך. מה שחשוב לנו לדעת לרגע זה הוא ש-JSD אכן מוגדר להיות וכמו כן עבור מצב בו P=Q אזי JSD=0.

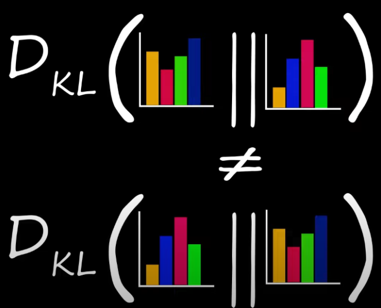
במקרה שלנו, מהסיבה ואנו מנסים להגיע לביטוי מינימלי עבור הגנרטור וכמו כן על ידי הגעה לביטוי מינימלי עבור הגנרטור רוצים להוביל למצב בו אזי במקרה שלנו JSD=0 ולכן נקבל:

סך הכל נקבל כי עבור מצב בו JSD=0 הערך התיאורטי המינימלי שאנו מקבלים עבור הגנרטור המינימלי הוא -log4. עם זאת, השורה התחתונה היא לאו דווקא הערך הזה אלא העובדה שהוא מתקבל בעקבות העובדה שהובלנו למצב בו כלומר הגנרטור האופטימלי שאף לקירוב ההסתברויות. מהסיבה ש-JSD יכול לקבל רק ערכים חיוביים, כל מצב אחר בו יוביל ל-

כמו כן אנחנו יכולים לבצע הסתכלות מפרספקטיבה שונה על הערך שקיבלנו – דווקא מזווית הדיסקרימינטור. אם מסתכלים על הערך של ניתן לראות כי עבור מצב בו אזי מתקבל הביטוי הבא:

כלומר ההסתכלות היא שקולה משני הכיוונים, הגנרטור שואף למצב בו כלומר ההתפלגויות שוות עבור הדאטא האמיתי והדאטא שנוצר כתוצאת מהתפלגות הגנרטור. במידה והוא יצליח להגיע למצב זה, הוא יוכל להערים על הדיסקרימינטור ויגרום לו לפלוט ערכי פרדיקציה השווים ל- עבור כל דגימה ומכאן נקבל את המסקנה שהגנרטור מייצר דגימות טובות שנראות כאילו לקוחות מההתפלגות האמיתית.

**JSD (Jensen Shannon Divergence)**



עד כה כאשר דיברנו על מדד למדידת מרחק בין שתי התפלגויות דיברנו על KL Divergence. עם זאת, נקודה חשובה שחשוב לשים אליה לב היא העובדה ש-KL Divergence הוא מדד שאינו סימטרי. כלומר בהינתן שתי התפלגויות כלשהן P, Q אזי מתקיים עבורן כי: כלומר ערך ההחזרה של המדד הוא שונה כתלות באיזו התפלגות אנו משווים לאיזו התפלגות.

מדד JSD לוקח את העובדה הזו בחשבון והוא מציג ערך אחיד למרחק בין שתי התפלגויות באופן בו ישנה קומוטטיביות במיקומי ההתפלגויות בנוסחת המדד. האופן בו JSD עושה זאת הוא באמצעות לקיחת ממוצע ערכי ה-KL Divergence עבור כל אחד מההתפלגויות. מתמטית:

דוגמא:

לטובת הדוגמא נגדיר ומכאן:

נסתכל על ה-Probability Distribution של 2 ההתפלגויות P,Q:

מכאן:

כעת נסתכל על KL Divergence – ונחשב:

לסיום, נבצע מיצוע של שני הערכים ונקבל את הערך של JSD:

*גם כאן מבחינה מתמטית, נשים לב כי עבור מצב בו* P=Q *כלומר ההתפלגויות זהות, אזי נקבל מצב בו ערכי ה-*KL Divergence *יהיו 0 ומכאן נקבל כי* JSD *יקבל ערך של 0.*

**Transformers**

<https://www.youtube.com/watch?v=eMlx5fFNoYc&t=297s>

<https://www.youtube.com/watch?v=-QH8fRhqFHM&t=1434s>

הטרנספורמר היא אולי אחת הארכיטקטורות המשמעותיות ביותר בכמה השנים האחרונות. במאמר בשם Attention Is All You Need תיארו בשנת 2017 חוקרים מגוגל את הארכיטקטורה שבאמצעות מנגנון ה-Attention שבה מסוגלת להתמודד עם Sequential Data ללא תופעות לוואי ובעיות שרווחו באותה תקופה לגבי מודלים כמו RNN, LSTM ודומיהם.

מאז, הטרנספורמר שולב במספר ארכיטקטורות שמהוות כיום SOTA בתחום השפה (NLP), תמונה ואף וידאו מה שהופך אותו לבסיס למודלים מתקדמים בתחומים אלו אשר הולכים ומתפתחים בקצב מסחרר כמו GPT, CLIP, Stable Diffusion ועוד הרבה נוספים. בין היתר, הארכיטקטורה מאפשרת יצירת Autoregressive Model לטובת יצירת תוכן ג׳נרטיבי של טקסט, יצירת קישור בין תמונות וטקסט או כל שני סוגי דאטא והבאתו לכדי ייצוג ווקטורי אחיד ועוד.

כפי שצוין, הקפיצה המחשבתית שהביא הטרנספורמר מבוססת בעיקר על היכולת להתמודד עם תלויות ארוכות טווח בין אלמנטים בדאטא, בזכות מנגנון ה-Attention המאפשר למודל להתמקד באזורים המגלים תלויות רלוונטיות בדאטא שמודלים אחרים התקשו בזיהויים.

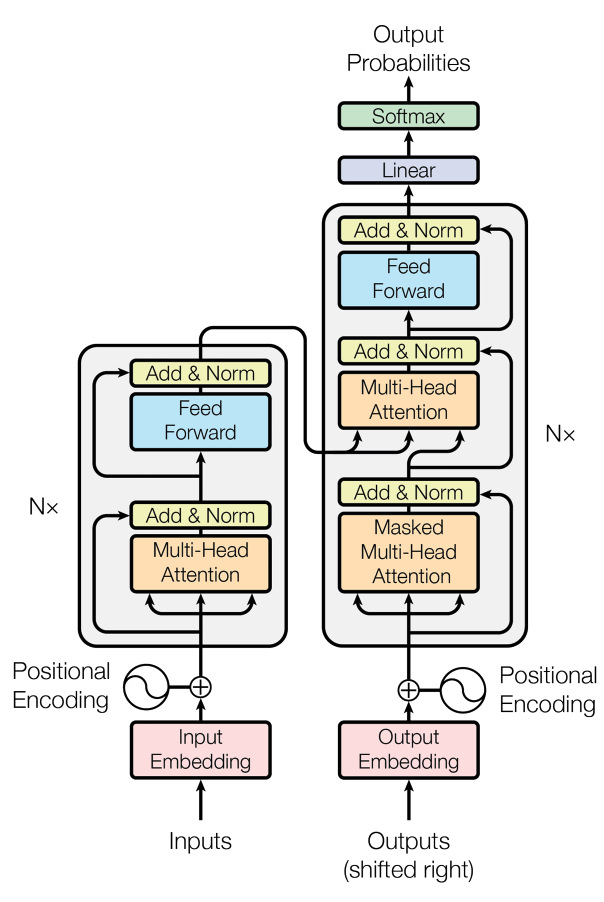
חשוב לציין כי הארכיטקטורה מורכבת יחסית ובנויה ממספר לא קטן של קומפוננטות ולכן תחילה יתוארו חלק מהקומפוננטות ב-High Level לטובת הבנת החשיבות הכללית שלהן ולאט לאט לאורך הסיכום אני אתאר In Depth יותר את הקומפוננטות.

*חשוב לציין כי מאחר והארכיטקטורה יחסית מורכבת לא אוכל לפרט לפרטי פרטים קונספטים יותר יסודיים למען שמירה על הרצף אך עם זאת אגע בנושאים המצויים בארכיטקטורה וחשובים לטובת הבנתה.*

*כמו כן, לטובת הפשטות, תיאור הארכיטקטורה יתרחש תחילה עבור* Inference *ורק לאחר מכן יתואר איך מתבצע תהליך האימון בטרנספורמר.*

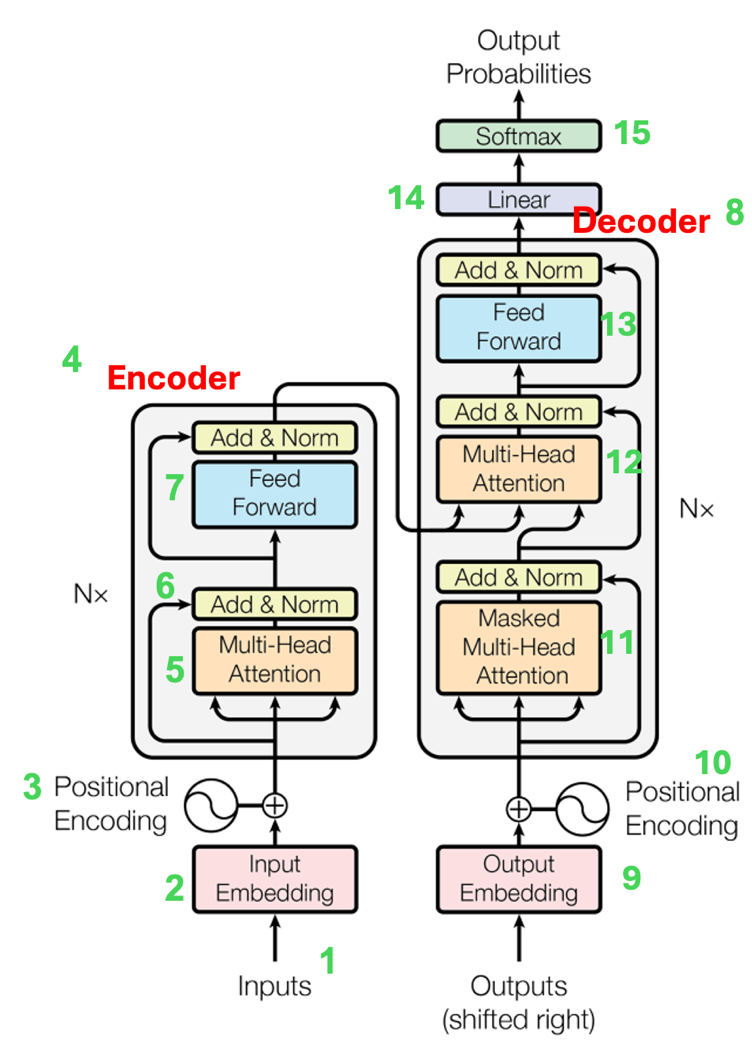
*הדוגמא בה נשתמש היא דוגמא לטרנספורמר אוטו-רגרסיבי כך שבהינתן פרומפט יוצר את התשובה על סמך פרדיקציה על המילה הבאה ברצף עד לכדי השלמת התשובה המלאה.*

**ארכיטקטורת ה-Transformer**

**

**Decoder**

**Encoder**

**Transformer Overview**

נרצה כעת לתת תיאור High Level לקומפוננטות:

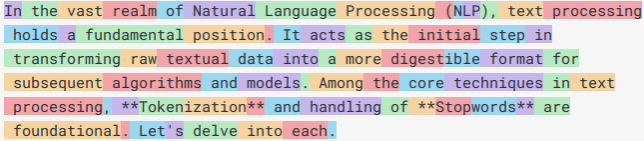
1. Inputs – הפרומפט ההתחלתי של המודל המורכב מטוקנים שמקורם במילים.
2. Input Embedding – המרה של הטוקנים לייצוגים ווקטורים מספריים.
3. Positional Encoding – הוספה ווקטורית המייצגת את מיקום הטוקן בפרומפט.
4. Encoder – הרכיב בארכיטקטורה שמטרתו לקודד את המידע בפרומפט לכדי ייצוג מספרי מזוקק ומרוכז שמועבר אל ה-Decoder. בפועל אין Encoder יחיד אלא n כאלו, כל Encoder הנסמנו באות i מעביר את הפלט שלו ל-Encoder ה i+1. לבסוף ה-Encoder ה-n יעביר את הפלט הסופי אל כל אחד מה-Decoders כקלט.
5. Multi-Head Attention – הרכיב החשוב בארכיטקטורה – מורכב ממספר תתי יחידות הנקראות Self Attention. מטרתה של כל אחד מתתי היחידות הללו זה למצוא טוקנים בעלי משמעות סמנטית והשפעה זה על זה
6. Add & Norm – בשכבה זו מוסיפים את הערך של ה-Residual Connection לערך הפלט משכבת ה-Multi Head Attention (Add). כמו כן מבצעים נורמליזציה על הערך הסופי שמתקבל (Norm).
7. Feed Forward Neural Network – רשת נוירונים נלמדת המקבלת כקלט את פלט שכבת ה-Attention. מכילה שכבות לא לינאריות שמסייעות בלמידת הרפרזנטציות עבור המודל.
8. Decoder – הרכיב האוטו-רגרסיבי בארכיטקטורה שמטרתו לייצר בצורה איטרטיבית ורציפה Probability Distribution over the corpus לטובת בחירת המילה הבאה המתאימה ביותר. כמו ב-Encoder גם כאן יש n שכבות של Decoder באופן בו פלט ה-Encoder ה-n נכנס אל כל אחד מה-Decoders ה-i ופלט כל Decoder ה-i מועבר אל עבר ה-Decoder הi+1 כך שלבסוף נקבל כי ה-Decoder ה-n הוא זה שיוצר את הפרדיקציה הסופית.
9. Output Embedding – ייצוג ווקטורי מספרי של כל המילים שיוצרו עד כה על ידי ה-Decoder.
10. Positional Encoding – הוספה ווקטורית המייצגת את מיקום הטוקן בפרומפט.
11. Masked Multi-Head Attention – שכבת Attention ממוסכת. השוני של משכבת Attention רגילה היא שהיא מוודאה שפרדיקציות עבור טוקן במיקום מסוים יהיו קאוזליות ועל סמך טוקני עבר בלבד ולא על סמך טוקני עתיד שיכולים להטות את תוצאות הפרדיקציות.
12. Multi-Head Attention – שכבת Attention נוספת הפעם ב-Decoder שמקבלת כקלט את הפלט של ה-Encoder ה-i וכמו כן את הפלט של Masked Multi-Head Attention לאחר נורמליזציה.
13. Feed Forward Neural Network – רשת נוירונים נלמדת המקבלת כקלט את פלט שכבת ה-Attention. מכילה שכבות לא לינאריות שמסייעות בלמידת הרפרזנטציות עבור המודל.
14. Linear – שכבה לינארית לטובת
15. SoftMax – ד

**Model’s Input**

הקלט של הטרנספומר יכול להיות מגוון אך לטובת המחשת הארכיטקטורה אתמקד בקלט טקסטואלי. אז למעשה מאחר והקלט הוא טקסטואלי, קרי פרומפט, אזי מורכב ממילים. המילים האלו בתהליך טוקניזציה הופכים להיות טוקנים.

טוקנים הם למעשה פיסת טקסט (לעיתים גם מילה) שמייצגת יחידה תפקודית לטובת המודל.

למשל את המילה working אפשר לפצל ל-2 טוקנים: work + ing.

דוגמאות ל-Tokenizations:

Unbelievable: "un", "believ", "able"

Transformation: "trans", "form", "ation"

Intercontinental: "inter", "continent", "al"

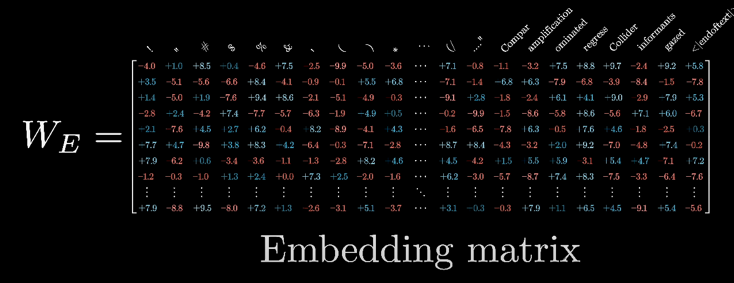
Misunderstanding: "mis", "under", "stand", "ing”

באופן כללי נוכל לתאר את התהליך באופן הבא:

כלומר הטוקנים הנשלחים אל המודל הן כלל המילים המופיעות בפרומפט.

**Input Embedding**

לאחר שייצרנו את הטוקנים, אנו רוצים לקבל ייצוגים ווקטוריים מספריים. דבר זה נעשה באופן בו מתרחש מיפוי בין כל טוקן לבין unique id באמצעות look up table ולאחר מכן שליפה של הייצוג הווקטורי במיקום ה-index מתוך ה-Embedding Matrix כייצוג הווקטורי המתאר את הטוקן.



נניח וה-token הוא door. ולפי ה-look up table ישנו מיפוי בין door לאינדקס 54323. כעת נלך אל ה-Embedding Matrix ונשלוף את הווקטור במיקום 54323.

חשוב לציין כי בעבר שיטות Embedding נפוצות היו למשל Word2Vec, Glove, אך כיום כאשר אנו משתמשים בארכיטקטורת הטרנספורמר, ה-Embedding Matrix כלולה בתוך המודל ולמעשה הייצוגים הווקטוריים נלמדים במהלך תהליך האימון של המודל. על קצה המזלג, למעשה מתחילים בייצוגים וקטוריים רנדומליים או מבוססים על ייצוגים וקטוריים קודמים ובמהלך תהליך האימון נדחפים גראדיינטים כחלק מה-Backpropagation ומעדכנים את הייצוגים הוקטוריים של הטוקנים.

בפועל, למעשה ה-Embedding Matrix היא למעשה סט משקולות נלמד שמתעדכן ומשתנה לאורך תהליך האימון ומטרתו ללמוד רפרזנטציות לגבי הטקסט שבסופו של דבר מגולמות בייצוגיים וקטוריים עבור הטוקנים.

מימד ווקטור ה-Embedding שתואר במאמר הוא 512 אך עם זאת במודלי שפה מתקדמים יותר כמו GPT3, המימד גדול בהרבה ועומד על 12288.

**Positional Encoding**

<https://www.youtube.com/watch?v=T3OT8kqoqjc>

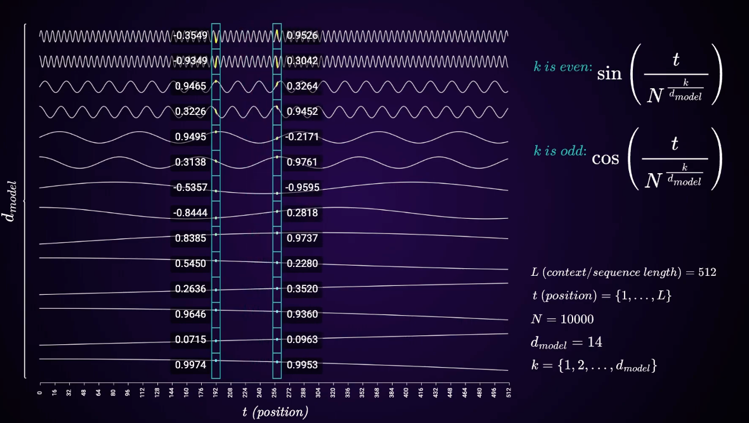
ל-Positional Encoding יש משמעות גדולה בטרנספורמרים והסיבה לכך נעוצה באופי המודל ובעיקר בשכבת ה-Attention. בעוד שלמודלים שמותאמים לעבוד בצורה ישירה עם Sequential Data יש יכולת עצמאית לייחס משמעות למיקומי הקלט לטרנספורמר המשתמש ב-Attention אין את היכולת הזו.

כפי שנראה בהמשך לשכבת ה-Attention איך יכולת להבדיל לגבי המיקומים של הטוקנים ברצף כך למשל, ללא Positional Encoding’ המשפטים “I walked to the park” , “The park walk to I” יהיו זהים לחלוטין בעיני הטרנספורמר על אף שלנו כבני אדם קל להבחין כי הם שונים בתכלית. לכן, יש צורך בקידוד המידע הזה בצורה ייעודית, כאן Positional Encoding נכנס לתמונה.

ה-Positional Encoding הוא ערך ווקטורי ממימד ה-Embedding שמתווסף Element-Wise אל הווקטור המספרי המייצג את הטוקן. ה-Positional Encoding צריך לקיים כמה תכונות עיקריות בניהן:

1. פונקציה חד-חד ערכית ועל – הפונקציה המשמשת ליצירת הווקטור שיתווסף ל-Embedding צריכה להיות כזו היוצרת ווקטורים יחודיים ולכן בהינתן position של מילה, יווצר ווקטור יחיד עובר position זה.
2. דטרמיניסטי – עבור ערך position יווצר בכל פעם אותו ערך ווקטורי יחיד (לא יכול להיות מצב שעבור position מסוים יווצר פעם אחד ווקטור a ופעם אחת ווקטור b)
3. מדיד אמפירית – עבור כל שני ווקטורים הלקוחים מההתפלגות, ניתן לתאר את המרחק בניהם באמצעות כלל מסוים כלשהו

השיטה המקובלת לבצע Positional Encoding זה באמצעות Cosine and Sine:

נראה כי t מייצג את ה-position של הטוקן הפרומפט, k מייצג את ה-position של הסקלר בווקטור ה-Embedding, N מייצג מספר גדול אותו אנחנו מעלים בחזקה, d\_model מייצג את מימד ה-Embedding איתו אנחנו עובדים.

סך הכל עבור כל סקלר בווקטור ה-Embedding יש פורמולה ייחודית ליצירתו.

לאחר שיצרנו את ווקטור ה-Positional Encoding אנו מוסיפים אותו Element-Wise לכל אחד מהאלמנטים בווקטור ה-Embedding כך שכעת יש קידוד של האינפורמציה המייצגת את מיקומים הטוקנים לאורך המשפט באופן בו שכבת ה-Attention והטרנספורמר באופן כללי יוכל ללמוד את הרפרזנטציות הסדרתיות.

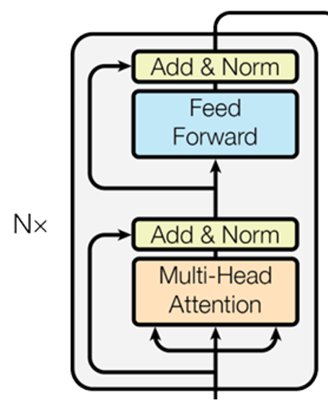
שיטה נוספת המקובלת לייצוג הווקטור שתואר היא באמצעות Color Map באופן בו ציר ה-x מייצג את המיקום של הטוקן במשפט וציר ה-y מייצג את האלמנט ה-i בווקטור אותנו נוסיף לייצוג ה-Embedding של הטוקן.

צבע ה-cell מתאר את הערך שיתווסף עבור אותו אלמנט כאשר צבע אדום מייצג ערך גבוה וצבע כחול מייצג ערך נמוך.

**Encoder**

רכיב ה-Encoder במודל הטרנספורמר הוא אחד משני חלקי הבסיס של הארכיטקטורה, השני הוא ה-Decoder. כל חלק במודל כולל מספר שכבות של Encoders או Decoders, שכל אחת מהן תורמת לעיבוד הנתונים בשלבים רבים ומורכבים.

בפועל, ה-Encoder מתחיל את פעולתו על ידי קבלת סדרת קלט, שבדרך כלל מורכבת מווקטורים של טוקנים שהומרו מטקסט לייצוג ווקטורי מספרי. מטרתו העיקרית של ה-Encoder היא להפוך את הסדרה הזו לייצוגים עשירים ומרוכבים יותר, שיכולים להכיל מידע רב על ההקשרים שבין האלמנטים השונים בטקסט. זה מאפשר ל-decoder לפענח את המידע ביעילות רבה יותר בהמשך.

הטכניקה המרכזית שבה משתמש ה-Encoder היא מנגנון ה-Attention, שמאפשר לו להתמקד בחלקים הרלוונטיים ביותר של הטקסט בזמן עיבודו. כל שכבת encoder כוללת שני רכיבים עיקריים:

1. Multi-Head Attention
2. Feed Forward Neural Network.

ה-Multi-Head Attention מסייע בהבנת הקשרים המרחביים בין המילים בטקסט, כאשר מספר ראשים של Attention עובדים במקביל לחפש אינטראקציות שונות. ה-Feed Forward Neural Network, שממוקם אחרי כל שכבת Attention, משמש להפעלת תהליכים נוספים של למידה על המידע שנאסף, ולהגברת ההבנה של ההקשרים הטקסטואליים.

חשוב לציין שעל אף שבתמונה נראה כי ה-Feed Forward Neural Network מופיע ככללותו רק לאחר ה- Multi-Head Attentionבפועל הוא מופיע אחרי כל Self Attention כך שבהינתן Encoder עם k שכבות של Self Attention, אזי יהיו בו k שכבות של Feed Forward Neural Network.

כמו כן לאחר כל Multi-Head Attention ולאחר כל Feed Forward Neural Network ישנה שכבת של Add & Norm לטובת הוספת ה-Residual Connection וביצוע נורמליזציה של ערך הקומפוננטה.

במהלך העיבוד, כל שכבת Encoder מוסיפה ידע ומורכבות לייצוג הטקסט, מה שמאפשר בסופו של תהליך לייצר וקטורים עשירים שמשקפים את המשמעות העמוקה וההקשרים המורכבים של הטקסט המקורי.

כמו כן, פרט שחשוב לציין הוא שבפועל אין Encoder יחיד אלא n כאלו, כל Encoder הנסמנו באות i מעביר את הפלט שלו ל-Encoder ה i+1. לבסוף ה-Encoder ה-n יעביר את הפלט הסופי אל כל אחד מה-Decoders כקלט.

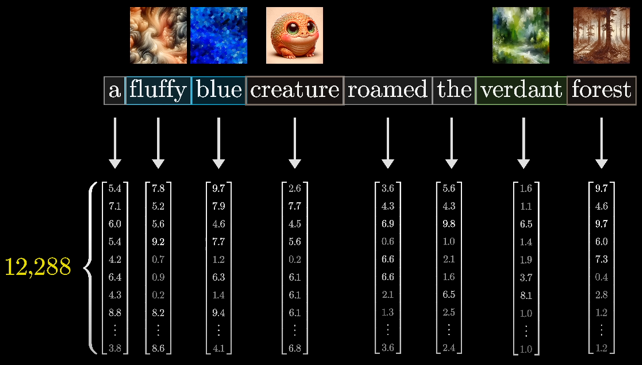
בארכיטקטורה המקורית תוארה ארכיטקטורת הטרנספורמר ככזו המכילה 6 שכבות Encoders אך עם זאת במודלים גדולים יותר למשל כמו GPT3, המספר הזה גדל משמעותית ועומד על בין 24 ל-96.

**Self-Attention**

לפני שנדבר על Multi-Head Attention חשוב קודם לכן לדבר על Self-Attention הרכיב שהוא למעשה גולת הכותרת של הטרנספורמר ופתח את הדלת לכל החידושים האחרונים ב-GenAI.

עד אז, מודלים ל-Sequential Data נתקלו בבעיות רבות כמו Exploding/Vanishing Gradients וכמו כן התקשו למצוא להתמודד עם תלויות מורכבות ברצפים ארוכים.

במנגנון ה-Attention למעשה תיאורטית אין הגבלה לגבי אורך הקונטקסט ובאופן עקרוני אורך הרצף יכול להיות אינסופי, ההגבלה בפועל היא משיקולי זמן ריצה ומשאבים.



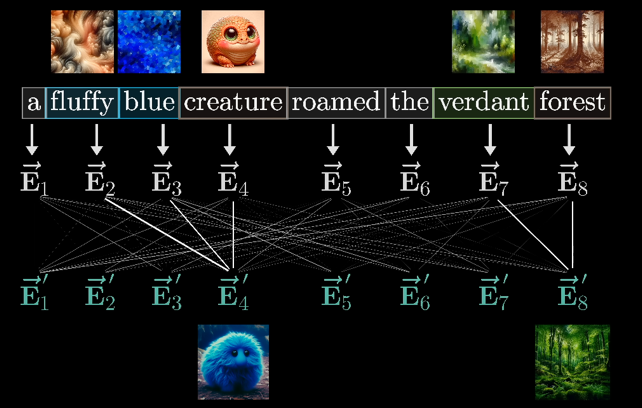
על מנת להסביר את מנגנון ה-Attention נפתח דווקא בדוגמא שמציגה את העיקרון. נניח ואנו ניצבים אל מול המשפט הבא: a fluffy blue creature roamed the verdant forest

בתור בני אדם, ישר קופצות לנו לעין התלויות הבאות:

1. Fluffy, blue הן מילות תואר המתארות את creature
2. Verdant היא מילת תואר המתארת את forest
3. Roamed הוא פועל המתאר את פעולת ה-creature
4. Forest מתאר את המיקום בו ה-creature roamed

כלומר, בתור בני אדם שיודעים לפרש שפה, מיד אנחנו מבחינים בתלויות שגורמות לכך שאנחנו מוצאים הגיון במשפט ויודעים לספוג את האינפורמציה שבו. יכולת זו היא יכולת שאנחנו רוצים גם שהמודל יפתח, שידע למצוא תלויות ברצף ולתרגם אותן לכדי רפרזנטציות שהוא לומד על הקונטקסט בפרט ועל האינפורמציה המובאת לידי ביטוי בשפה אנושית ככלל.

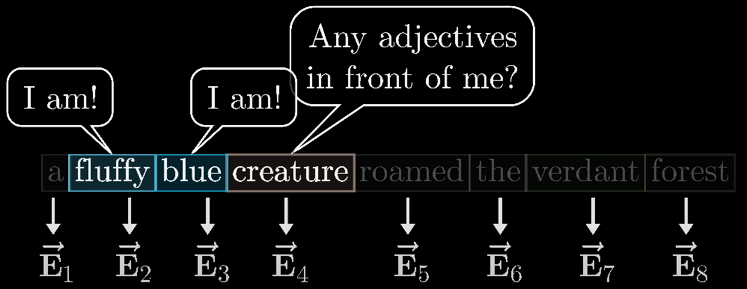
ממבט נוסף בתמונה חשוב לי לציין גם מה אנחנו רואים, למעשה המתואר הוא אסופה של ווקטורים המייצגים את הטוקנים (שבדוגמא זו נתייחס אליהם כמילים). כפי שתואר, אנחנו כבר בתוך ה-Encoder ולכן הווקטורים הללו כבר עברו PE (Positional Encoding) ועתידים כעת להיכנס אל עבר הקומפוננטה של ה-Self Attention.



באופן כללי ניתן לומר כי המודל מתחיל תחילה מהייצוגים הווקטוריים הלבנים שלמעשה מייצגים את הערך שנשלף מתוך ה-Embedding Matrix בתוספת PE והמטרה שלו במהלך תהליך ה-Attention הוא למצוא את הייצוגים הווקטוריים הכחולים שמזקקים את המשמעות הסמנטית ואת התלויות בין המילים.

כך למשל ניתן לראות למשל כי E4 הוא ייצוג ווקטורי של המילה creature שספגה לתוכה את המשמעות הסמנטית של 2 המילים הקודמות לה, fluffy and blue.

בדוגמא זו התמקדנו בתלות בין המילים creature, fluffy, blue כך ש-creature הוא שם עצם ו-fluffy, blue הן מילות תואר וכך גם עבור forest, verdant. עם זאת חשוב לציין שזו דוגמא בלבד והיא ניתנה רק כי לנו בתור בני אדם קל לתפוס אותה. ניתן לחשוב על כל שכבת Attention שכזו התופסת תלויות שכאלו בטקסט וזה גם מה שנעשה לשם הפשטות, אך בפועל נתפסות תלויות ורפרזנטציות על ידי המודל שבפועל לנו כבני אדם אין יכולת להבחין בהן. עם זאת, לטובת המשך ההסברים נמשיך לדבוק בדוגמא זו מהסיבה שהיא מתיישבת בצורה טובה עם ההיגיון האנושי.

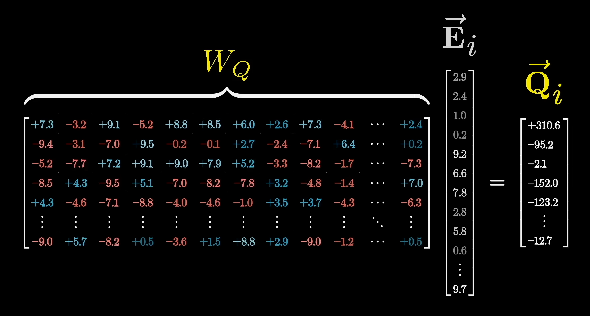
אז כאמור, ההתנהגות של המודל אותה אנו רוצים להשיג היא כזו היודעת לתפוס תלויות ברצף. למעשה עבור הדוגמא שלנו אנו יכולים לדמות זאת כאילו כל מילה שואלת: ״האם יש מילים לפני שחשובות לי לצורך הגדרתי?״ אם התשובה לכך היא כן, אזי המודל צריך לתכלל זאת ולשנות את הייצוג הווקטורי של המילה השואלת בהתאם.

פה בדיוק נכנס לתמונה מנגנון ה-Attention, זה בדיוק אופן הפעולה שלו ובאופן עבודתו הוא ממדל בדיוק את ההתנהגות הזו אותה אנו רוצים להשיג. כעת נסקור את האופן שבו הוא עושה זאת.

מבחינה פשטנית, אנחנו רוצים רכיב במנגנון שיודע לשאול את השאלה (Query), רכיב במנגנון שיודע לספק תשובה לשאלה (Key) ורכיב במנגנון שיודע לבצע את ההוספה לווקטור במצב בו התשובה לשאלה חיובית (Value).

**Query**

בהינתן רצף טקסטואלי באורך n המורכב מ- טוקנים (מילים במקרה שלנו), אנחנו רוצים שיהיה לנו אלמנט שאנחנו יכולים להפעיל על כל ווקטור שיוביל אותנו לשאלה הרצויה עבור הטוקן.

**

*האלמנט במקרה הזה הוא . האלמנט הזה הוא מטריצה בעלת משקולות נלמדות. אני אוהב להסתכל עלייה כמו קאוצ׳רית, היא כביכול מנחה את הווקטור לשאול את השאלה על ידי כך שמתבצעת מכפלה מטריציונית המתוארת .*

*חשוב לציין כי הולכת ומשתפרת לאורך תהליך האימון אך כעת מאחר וציינו שאנחנו מתארים את המנגנון ב-*Inference *אנו נתייחס אליה כאל קאוצ׳רית מנוסה שיודעת להוביל לשאלות הנכונות לכל ווקטור.*

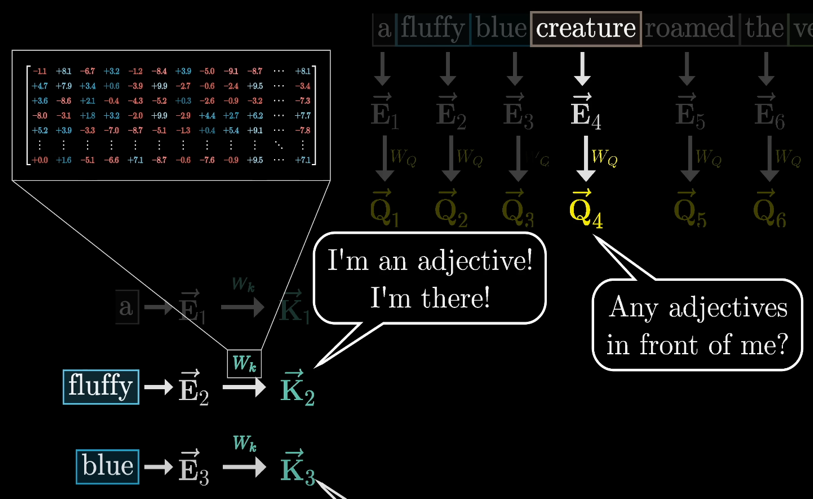


בדוגמא שלנו ווקטור ה-Embedding הוא בגודל של 12288x1. עם זאת המטריצה היא בגדול של 128x12288 ולכן מהמכפלה המטריציונית מתקבל ווקטור ממימד 128x1.

ניתן לחשוב על כך שבעצם המטריצה הנלמדת ממפה את הווקטור מהמימד הגבוה לווקטור אחר במימד נמוך בהרבה שממצה את השאלה הנשאלת עבור הווקטור.

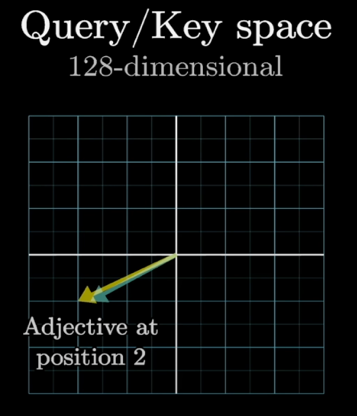
למעשה אנו מקבלים מימד נמוך יותר שהוא המימד אליו ממופות השאלות וכפי שנראה תכף שם הן גם מקבלות מענה על ידי ווקטור מאותו מימד. לפי המענה לשאלות ממימד זה המודל יידע האם לבצע עדכון חזק לווקטור הטוקן או עדכון חלש.

**Key**



אנחנו יכולים להסתכל על ה-Key כעל ווקטור מענה עבור ה-Query. לכל טוקן נבצע מכפלה באלמנט נלמד נוסף בשם . זו למעשה מטריצה נלמדת שמכילה משקולות אותן אנו כופלים בכל ווקטור  לקבלת תשובה המוגדרת באופן: כלומר לכל ווקטור מתבצעת מכפלה מטריציונית במטריצת המשקולות הנלמדת לקבל תשובה לשאלה .

גם כאן חשוב לציין כי במהלך תהליך האימון המשקולות של המטריצה משתנות בהתאם לדאטא אך גם כאן מאחר ואנו מדברים על Inference נניח וערכי המטריצה כבר מכוילים באופן מספק.

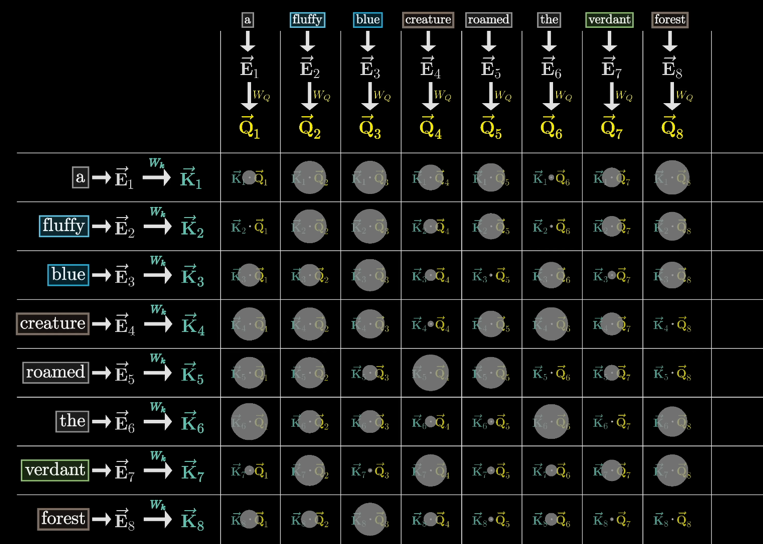


סך הכל, לכל ווקטור Embedding אנו מחשבים את המכפלה המטריציונית שלו עם שתי מטריצות משקולות לקבלת שני ווקטורים ממימד נמוך יותר .

מבחינה גרפית ניתן לחשוב על התלות בין ל- כעל ניסיון להגיע לחפיפה מושלמת בין הווקטורים במידה ויש התאמה טובה בין ה-Key ל-Query עבור שני ווקטורים. הסיבה לכך היא שכפי שתכף נראה אנו מחשבים dot-product בין שני הווקטורים באופן הבא:

כלומר לכל זוג ווקטורים i, אנחנו מחשבים מכפלה dot-product המסתכמת לסקלר. בהנחה והסקלר גדול אזי קיים דמיון רב יותר בין הווקטורים, בהנחה והסקלר קטן יותר קיים דמיון פחות יותר בין הווקטורים.

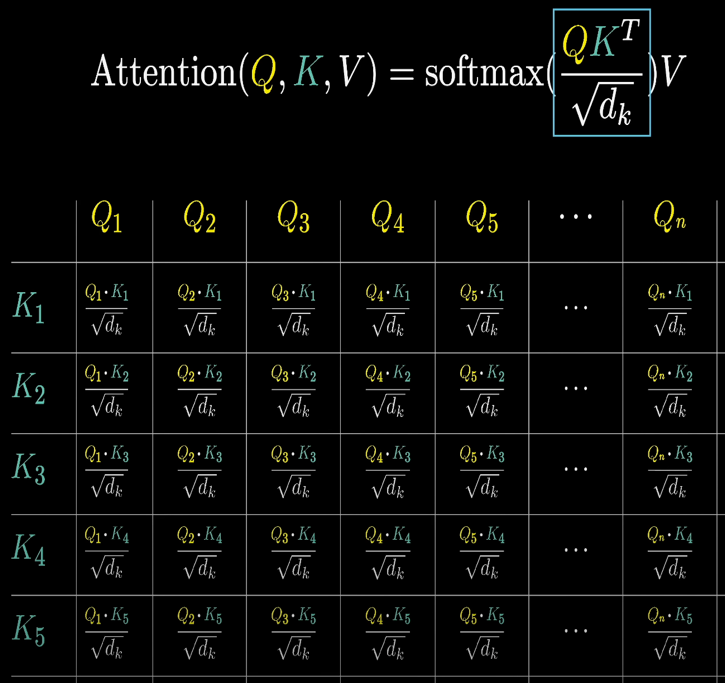
הסיבה לכך נעוצה באופן בו מכפלת dot-product מחושבת. בהינתן שני ווקטורים a,b, נוכל להגדיר את המכפלת dot-product באופן הבא: . בהינתן הגדרה זו נשים לב כי הביטוי מקבל ערך מקסימלי כאשר a=b מהסיבה ש תמיד מקבל ערך חיובי.



בתמונה ניתן לראות את ה-Key Value Matrix באופן בו לכל תא במטריצה יש עיגול שמייצג את ערך מכפלת ה-dot-product באופן בו עיגולים גדולים מייצגים ערך גבוה כלומר הלימה בין ה-Key ל-Query וערכים קטנים מייצגים חוסר התאמה בין ה-Key ל-Query כלומר מצב בו יש התאמה קטנה בין הזוג הסדור של הטוקנים. התאמה גדולה כאמור מייצגת מצב בו יש Attention ותלות בין שני הטוקנים והתאמה קטנה מייצגת מצב בו אין Attention ותלות בין הטוקנים.

כפי שנראה תכף, ערך ה-dot-product עבור כל זוג סדור של טוקנים ישמש עבורנו לטובת עדכון הייצוגים הווקטוריים של כל אחד מהטוקנים ב-inference ויוביל לחיזוי עבור המילה הבאה שהכי מתאימה להשלמת הרצף.

**הגדרת ה-Attention**



כפי שהוצג במאמר המקורי, הגדרת ה-Attention מוגדרת באופן הבא:

הסימון היחידי המוכר לנו הוא המייצג כפי שדובר את ה-Key Value Matrix.

עם זאת, עד כה דיברנו על שני רכיבים מתוך הגדרת ה-Attention שהם Query, Key. לפני שנתמקד ברכיב השלישי שהוא ה-Value המיוצג באות V, נרצה לתאר 3 גורמים נוספים:

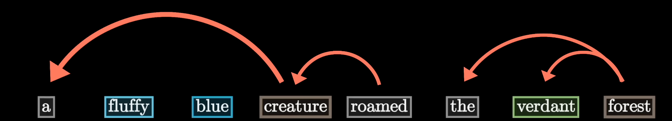
1. חלוקה ב-
2. Masking
3. SoftMax

חלוקה ב-

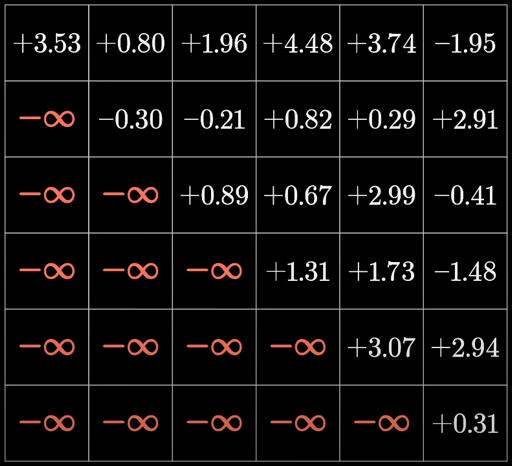
החלוקה ב- היא straight forward חלוקה של כל תא במטריצה ב- כלומר בשורש המימד הווקטורי. הסיבה לכך היא רצון לשמור על יציבות נומרית ולהימנע מלעבוד עם מספרים גדולים שיכולים לגרום לקושי בעדכון המשקולות ולמידת הרפרזנטציות של המודל.

Masking

שימוש ב-Masking מונע מהמודל ללמוד רפרזנטציות עבור טוקנים באמצעות טוקני עתיד במהלך תהליך האימון או במהלך תהליך ה-Inference בהנחה ואנחנו משתמשים ב-Cross Attention כשאנו באים לפתור בעיית תרגום למשל.



אם נסתכל למשל על התא המסומן באדום בתמונה העליונה, נראה כי זו למעשה המילה a שמפיקה Query אך את ה-Key מפיקה המילה creature שהיא מילת עתיד. למעשה, המילה a לא חשופה בכלל למילה creature ולכן נרצה למסך את כל האלכסון התחתון שמייצג מילות עבר שמקבלות מענה על ידי מילות עתיד.



האופן שבו נבצע את ה-Masking הוא על ידי השמת ערך של מינוס אינסוף בכל התאים שמצויים במשולש התחתון מתחת לאלכסון הראשי. של המטריצה.

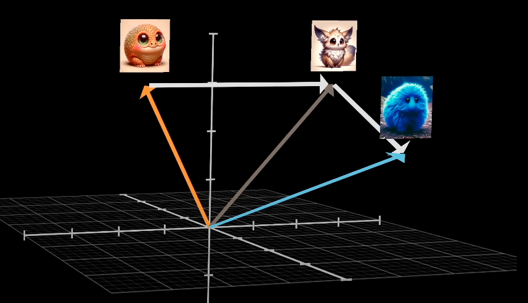
באופן זה הערכים שנותרו מייצגים אך ורק מילות נוכחיות שמתבססות אך ורק על מילות עבר לצורך חישוב ה-Attention.

חשוב לציין כי Masking אינו הכרחי תמיד ויש ארכיטקטורות שממשות את המנגנון הזה ויש ארכיטקטורות שאינן מממשות את המנגנון הזה.

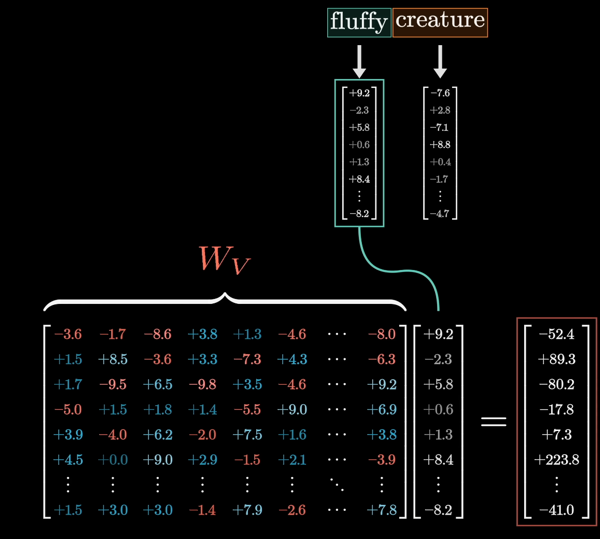
SoftMax

לאחר החישוב של , חלוקה ב- וביצוע Masking שכאמור אופציונלי בלבד, כעת יש חישוב של SoftMax על כל עמודות המטריצה בנפרד.

באופן זה אנו מקבלים כי עבור כל עמודה, המייצגת טוקן, אנו מקבלים ערכי הסתברות בין 0 ל-1 המסתכמים ל-1. כלומר אנו מקבלים Probability Distribution עבור ה-Attention לכל טוקן ביחס לטוקני העבר שלו. התא שמכיל את הערך הגבוה ביותר הוא התא שמייצג את הטוקן בעל ההשפעה הגדולה ביותר עבור הטוקן הנוכחי.

**Value**

אז אמנם דיברנו על ה-Query וה-Key כרכיבים מהותיים במנגנון ה-Attention שבאמצעותם נמדדת התלות בין הטוקנים לאורך הרצף. עם זאת, לאחר שהמנגנון מכמת את התלות בין המילים ברצף יש לבצע שינוי בייצוגים הוקטוריים של הטוקנים על מנת שיגלמו ייצוגים חדשים התלויים בתלויות של הטוקנים שקדמו להם. כאן נכנס לתמונה ווקטור ה-Value.

לכל טוקן נבצע מכפלה במטריצה נלמדת נוספת בשם . זו למעשה מטריצה נלמדת שמכילה משקולות אותן אנו כופלים בכל ווקטור לקבלת תשובה המוגדרת באופן: . אני אוהב להסתכל על המטריצה הנלמדת כמו תהליך של חליטת תה באופן בו המטריצה הנלמדת יודעת למצות מהווקטור את מהות השינויים שיובילו לעדכון האפקטיבי ביותר של הייצוג הווקטורי של הטוקן הבא.

למעשה ניתן לחשוב על כמו תמצית הפרחים, על כעל המים הזורמים על תמצית הפרחים (המים יודעים איך לחלוט ולמצות) ונשפכים אל הכוס באופן בו התה כשלעצמו הוא הווקטור V שמייצג את המיצוי של הרפרזנטציות הרלוונטיות שיובילו לכך שהטוקן הבא יושפע מהטוקן שקדם לו באופן נכון ואפקטיבי.

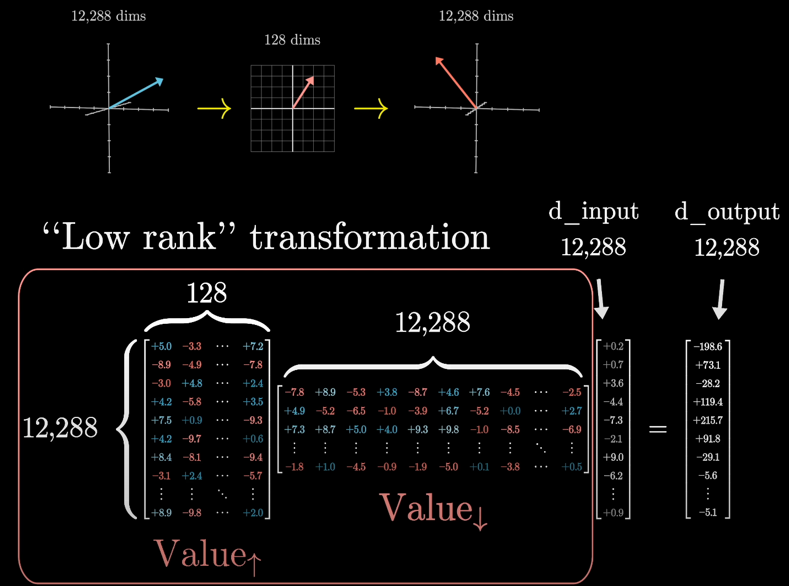


מתיאור התהליך נשים לב כי כל אחד מהייצוגים הוקטוריים מוכפל ב- לקבלת . לאחר מכן, מתבצעת מכפלה של ב- לקבלת הערך הממושקל של השינוי שכל טוקן קודם מוסיף לטוקן הנוכחי.

חשוב לציין כי עבור , ניתן להתייחס אליהם בתור ווקטורים המוכפלים dot-product אך ניתן להתייחס אליהם גם כאסופת וקטורים, קרי – מטריצה ולכן לייצג מתמטית את הפעולה שאנחנו עושים כ- *שזו בדיוק הגדרת ה-*Attention.

לכן, עבור כל ווקטור באמצעות פעולת ה-Attention אנו למעשה יוצרים ווקטור שקול בגודלו שסימונו הוא שמייצג את השינוי שצריך לבצע בייצוג הווקטורי של הטוקן על מנת לתכלל את הטוקנים שקדמו לו והשפעתם על הייצוג הווקטורי שלו וכל זאת בהתאם לערכי ה-Query, Key, Value.

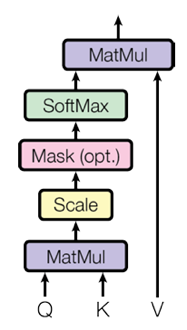
בדוגמא שלנו ניתן לראות כי הטוקן הנוכחי הוא המייצג את המילה creature, ומכפלת הdot-product של fluffy, blue, נתנה ערכים של 0.42, 0.58 בהתאמה. לפיכך, הם משפיעים על המילה creature ולכן צריך לשנות את הייצוג הווקטורי של creature לפי היחסים של תוצאת ה-SoftMax שמובילה לכדי יצירת ווקטור שמוכפל בווקטור V ליצירת שמתווסף element-wise אל .



נקודה חשובה שחשוב לציין היא גודל המטריצה הנלמדת . בעוד שאמרנו שגם וגם הן בגדול של 12288x128, המטריצה היא בגודל 12288x12288. דבר שמוביל למורכבות מהסיבה שזה הרבה פרמטרים נלמדים שיכולים לעקב את תהליך האימון את תהליך ה-Inference.

הפתרון לכך הוא לפרק את המטריצה לשתי מטריצות קטנות יותר:

1. בגודל 12288x128
2. בגודל 128x12288

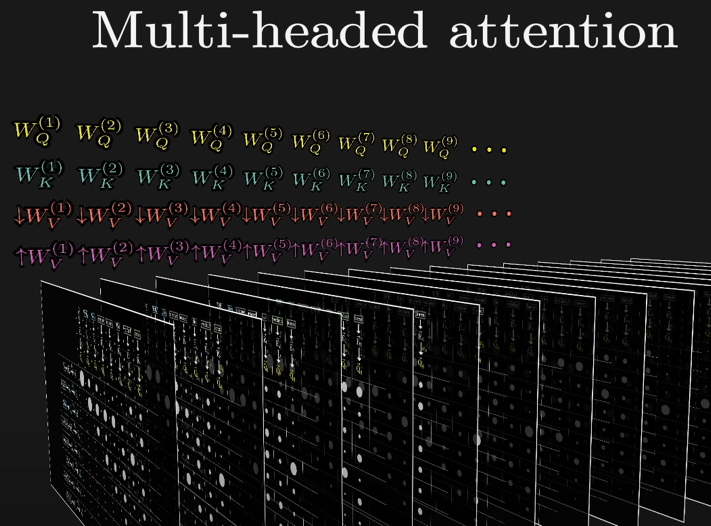
ניתן לשים לב כי תחילה מתבצעת מכפלה של עם הייצוג הווקטורי של הטוקן לקבלת מיפוי למימד נמוך ולאחר מכן מתבצעת מכפלה של עם הייצוג הווקטורי מהמימד הנמוך לקבלת ווקטור השקול למימד של הטוקן. באופן זה אנו מקבלים כי ומכאן ניתן לראות כי הגודל של שווה לפעמיים הגודל של / .

לסיכום, ניתן לומר כי התמונה בצד שמאל ממחישה את השלבים שתוארו לגבי הגדרת ה-Attention: .

כמו כן כפי שתואר נשים לב כי שלב ה-Masking הוא אופציונלי ולא הכרחי למשל כמו בסוגי Attention אחרים כמו Cross Attention בו אנו עוסקים בסוגי דאטא מתחומים שונים או למשל ברצפי טקסט משפות שונות.

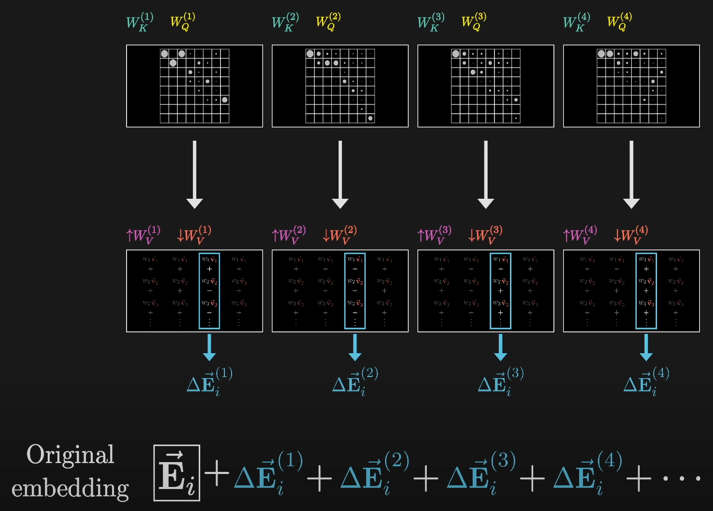
**Multi Head Attention**

עד כה דיברנו על Self-Attention ועל התפקיד שלו במציאת תלויות לאורך הרצף. בדוגמא שהבאנו, ראינו כי שכבת ה-Self-Attention מטרתה למצוא תארים לפני שמות עצם, אך כאמור זו הייתה רק דוגמא והתלויות שנמצאות בפועל יכולות להיות תלויות שאנחנו כבני אדם בכלל לא יכולים להבחין בהן.

עם זאת, אנו תיארנו תלות אחת בלבד אך יכולות להיות רבות ומגוונות ולכן לא נרצה להסתפק בשכבת Attention יחידה אלא נרצה שיהיה לנו מספר ראשי Attention שהמטרה שלהם למצוא תלויות שונות ומגוונות בדאטא ובדיוק לשם כך נכנס לתמונה מנגנון ה-Multi Head Attention שמורכב ממספר ראשים.

לכל אחד מהראשים יש את כל אחת ממטריצות המשקולות , , , וכל אחד מהראשים מפיק בסופו של דבר את שלושת הווקטורים מהמימד הנמוך: Q, K, V.

למעשה מה שאנחנו מקבלים זה שכל ראש Attention למעשה מציע את השינוי שלו לייצוג ה-Embedding של ווקטור הטוקן בהתאם לסט הערכים הנלמדים שמצויים בו. כל ראש למעשה מסיים עם הצעות לשינוי שנסמנן ב- כפלט של שכבת ה-Attention הספציפית.



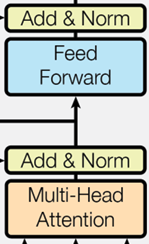
האופן בו מתבצע העדכון של הייצוג ה-Embedding בסופו של דבר עבור ערך הווקטור של הטוקן הוא על ידי סכימה של כל השינויים שהוצעו בכל אחד מראשי ה-Attention אשר יתווספו אל הערך הווקטורי המקורי של הטוקן.

בתמונה ניתן לראות כי עבור ייצוג ווקטורי ישנה הוספה של כל באופן בו המספר הקטן העליון הוא אינדקס של ראש ה-Attention באופן בו הוא מציין את השינוי שהראש רוצה לבצע בווקטור הטוקן כתלות בדפוס אותו הוא אומן למצוא.

היתרון המשמעותי ב-Multi Head Attention היא היכולת שלו לרוץ בצורה מקבילה על GPU דבר שמבחינה הנדסית יכול להוביל לזמני ריצה מהירים ול-Inference במהירות מהירה. בעוד שלמודל יש מספר רב של פרמטרים, הרצה שלו במקביל מובילה לכך שזמן הריצה יכול להיחתך אף ב-2 עד 3 סדרי גודל מהסיבה שלא קיימת תלות משורשרת בין הפלט של ראש Attention כלשהו לקלט של ראש Attention שבא אחריו ולכן כולם יכולים לרוץ במקביל, לסכם את תוצאות עבור כל ראש אל ווקטור הטוקן ולהמשיך אל עבר החישוב הבא.

**Add & Norm**

קומפוננטת ה-Add & Norm מופיעה מספר פעמים בארכיטקטורה. כמו כן, היא מופיעה בין היתר לאחר שכבות ה-Multi Head Attention ולאחר שכבות ה-Feed Forward Neural Network.



Add

בשכבת ה-Add & Norm מתבצע הוספת ערך ה-Residual Connection מהשכבה הקודמת (Add). אופן פעולת ה-Add עובדת באופן הבא:

נסמן ב-f את הטרנספורמציה שמתבצעת על הקלט המקורי למשל Feed Forward או Multi Head Attention. אזי הפלט יהיה שווה ל- כלומר הוספה ישירה של ערך הקלט אל ערך הפלט לאחר הטרנספורמציה.

בארכיטקטורת הטרנספורמר הערך מתווסף בצורה ישירה ובמלואו אך ישנן ארכיטקטורות בהן חלק קטן יותר מתווסף למשל אם כופלים את x במספר כלשהו בין 0 ל-1.

Norm

למעשה מדובר כאן ב-Batch Normalization באופן בו הנורמליזציה מתרחשת בנפרד עבור כל עמודה. הנוסחה המתמטית לתיאור ה- Batch Normalizationהיא:

למעשה עבור כל עמודה מחושב ערכי הממוצע והשונות ולאחר מכן עבור כל ערך בעמודה יש החסרה של ערך הממוצע וחלוקה בערך השונות ועוד אפסילון כדי למנוע חלוקה ב-0.

לבסוף, יש הכנסה של הערך אל עבר הערך הסופי של הסקלר בעמודה, לאחר מניפולציה באמצעות שני פרמטרים נלמדים .

עצם העובדה ש- נלמדים מאפשרת למודל לשלוט על גמישות הנורמליזציה ולבצע אותה באופן חלקי בטווח הרציף עבור הערכים אותם יכולים הפרמטרים הללו לקבל. כמו כן, אחת הסיבות שיש בפרמטרים הללו שימוש באופן נלמד היא כדי לאפשר מצב בו המודל למעשה כלל לא ישתמש בפרמטרים ובכך נפחית את ה-bias שעלול להיווצר ולגרום ל-overfitting. נראה כי עבור מצב בו מקבלים למעשה זהות ל- ולכן למודל יש אפשרות לבצע טרנספורמציה ישירה.

סך הכל, נראה כי שכבת ה- Add & Normמופיעה פעמים רבות בארכיטקטורה ומהווה תפקיד חשוב באופן למידת המודל ולמידת הרפרזנטציות. היא דואגת למניעת Exploding / Vanishing Gradients ומובילה להתכנסות בצורה יעילה יותר.

**אילוסטרציית הרצה נומרית עבור Inference**

כעת נרצה לבצע אילוסטרציה מספרית עבור Inference בארכיטקטורה. האילוסטרציה תתמקד בפרומפט המכיל 3 מילים שיהוו גם כטוקנים ולהם נרצה לחזות את המילה הרביעית. הארכיטקטורה בה נבחר היא כזו בה הייצוג ה-Embedding הוא ממימד 3, ישנם 2 ראשים ב-Multi head Attention וכמו כן יש 2 Encoders ו-2 Decoders.

Inputs

ה-Inputs הוא הפרומפט “I love eating” המורכב מהמילים: “I”, “love”, “eating”. מאחר ואנו מבצעים inference אזי כבר קיימת Embedding Matrix שהתקבלה לאחר תהליך האימון ולכן נניח כי האינדקס במטריצה המתאים לכל מילה הוא: “I” = 1, “love” = 2, “eating” = 3.

Inputs Embedding

על סמך האינדקסים שהבאנו מה-look up table עבור כל מילה ניגש כעת אל ה-Embedding Matrix ונשלוף את הייצוגים הוקטוריים המתאימים לכל אחת מהמילים. נניח והערכים הם:

Positional Encoding

ניזכר בנוסחת ה-PE ונציב עבור כל אחד מהייצוגים הוקטוריים את ערכים במשוואת הבאות:

*תזכורת:*

*t – מייצג את ה-*position *של הטוקן (במקרה שלנו מילה) בפרומפט*

*k – מייצג את ה-position של הסקלר בווקטור ה-*Embedding

N – *מייצג מספר גדול אותו אנו מעלים בחזקה (10000)*

*d\_model – מייצג את מימד ה-*Embedding  *איתו אנו עובדים (במקרה שלנו 3)*

*כמו כן, חשוב לציין כי הקלט לפונקציות Sin, Cos הוא ברדיאנים ולא במעלות*

Position 0: “I”

Overall Positional Vector: [0, 1, 0.002]

Position 1: “love”

Overall Positional Vector: [0, 0.99, 0.002]

Position 2: “eating”

Overall Positional Vector: [0.9, 0.99, 0.004]

סך הכל נראה כי לאחר חיבור הווקטורים שמייצגים את ה-PE בצורת element-wise אל הווקטורים המקוריים שהגיעו מה-Embedding Matrix נקבל: