למידה עמוקה

Natural Language Tool-Kit

ספריה המאפשרת לנו לבצע את הפעולות הבאות:

-Tokenization

מודל בסיסי המאפשר שבירת המשפט למילים תוך שימוש במפרידים (delimiters) כך שהוא שם גם את המפרידים כמילה בשל עצמה ויודע להבדיל בין נקודה בסוף משפט לבין נקודה ששייכת למילה כמו MR. בנוסף גם הופך את don't ל "n't","do"

-Stemming

כאשר נרצה להתייחס למילים ששונות בכתיבתן אך השורש שלהן זהה. (walking, walk, walks…) Stemming פועל על כל מילה במשפט בנפרד לכן, לפני הפעלת stemming צריך קודם לעשות tokenization למשפט.

- **אלגוריתמים ב stemming:**
  - Nltk.stem.porter import **Porterstemmer**
  - Nltk.stem.lancaster import **Lancasterstemmer**
  - Nltk.stem import **Snowballstemmer.**

Part Of Speech- POS

interjection	מילים המציינות רגשות רגעיים (Ouch! Wow! Yikes!)
Conjunction	מילים המקשרות בין חלקי המשפט (and, but, yet)
Nouns	שם עצם A noun is a person, place, or thing Some examples of a person are: sister, friend, Alex, Stephanie, you, me, dog Examples of places are: house, beach, New York, playground, the store Things (can be physical things or ideas): chair, pencil, thoughts, memories, and knowledge.
Verb	פועל used to describe things that nouns do! An example of a verb would be “run”. Run is a word to describe someone or something who is moving faster than a walking speed. Other examples of verbs include: sing, watch, play, sleep, study, walk, and think.
pronoun	עצם

מודל הנותן לכל מילה במשפט פירוש (אנוטציה) תחבירית לחלקי הדיבר במשפט.

יודע גם להבחין בין מילים שכתובות אותו דבר אבל יכולות להיות גם פועל וגם של עצם כמו show

- **HMM - Hidden Markov Model**

  המטרה: סיווג של המילים במשפט ל part of speech שלהם.

  ההנחה של מרקוב- הסיווג של מילה ל part of speech שלה במשפט תלויה ב parts of speech של המילה שקדמה לה.

יש לנו dataset שבו יש המון מילים עם ההסתברות שלהן להופיע בתור כל אחד מה parts of speech. אנחנו נראה שלפי המילים והמילים שקדמו להן נחשב את ההסתברות שכל מילה תהיה ב part of speech הנכון לפי הקודמות.

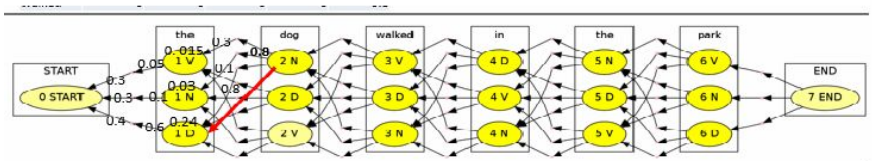
יש 2 טבלאות:

- מחזיק אוצ"מ וההסתברות של כל מילה להיות ב part of speech שלה.
- מחזיק הסתברויות של לעבור בין כל part of speech ל part of speech הבא.

כל מילה תמופה ל part of speech שלה לפי המקסימום בין הכפלה של ההסתברות לפי טבלה 1 כפול ההסתברות בטבלה 2.

- **Viterbi**

בשונה ממרקוב המתייחס עבור כל צמד מילים במשפט (מה ההסתברות שהמילה תהיה part of speech מסוים כפול ההסתברות שנעבור בין ה part of speech הקודם ל part of speech שנבחר), אלגוריתם זה מחשב את "המסלול" הכי "משתלם" מתחילת המשפט עד סופו לפי ההסתברויות שחושבו עד לסוף המשפט (בכל התקדמות במילה מתייחסים לכל המצבים הקודמים מההתחלה).



Transition D	END	N	START	V	
D	0.1	0	0.1	0.4	0.4
END	0	1	0.2	0	0.1
N	0.8	0	0.1	0.3	0.3
V	0.1	0	0.6	0.3	0.2

Emission D	END	N	START	V	
<END>	0	1	0	0	0
<START>	0	0	0	1	0
dog	0.1	0	0.8	0	0.05
in	0.3	0	0	0	0
park	0	0	0.1	0	0.1
the	0.6	0	0.1	0	0.05
walked	0	0	0	0	0.8

Lemmatization

בניגוד לStemming , הוא יודע להתאים מילים בעלות אותה משמעות אך משורשים שונים לפי ה part of speech של המילה והמילה עצמה לdatabase שמכניסים לו.

- Wordnet Lemmatizer - מ NLTK משתמש ברשימה קצרה של תגיות דיבור ולכן נעשה שימוש בפונקציה קצרה כדי לבצע את ההמרה.

```
my_text = "Whoever eats many cookies is regretting doing so"
>>> lemed
['Whoever', 'eat', 'many', 'cooky', 'be', 'regret', 'do', 'so']
```

Chunking

הינה שיטה לניתוח משפט (או סט של משפטים), לנתחים שונים - קבוצות.

אנחנו נגדיר את הקבוצות השונות ע"י ביטויים רגולריים - ex.{DT?<JJ>} :PR ובעזרת זה נוכל לפרק את המשפט לתבניות שהגדרנו.

- NLTK - ניתן ליצור "עץ" של החלקי דיבר של המשפט.

למה זה טוב? לענות על שאלות על תוכן המשפט ניתוח.

Bi-Grams

חיפוש צמדי מילים בטקסט וספירה שלהם יעיל ל: תיוג מילים לPOS, חילוץ פיצ'רים וכ'ו..

יש גם הרחבה לזה: Tri-Grams - אותו דבר עם שלושיות.

- באופן כללי קוראים לשיטה זו: **N-Grams**.

שימוש ב N-Grams ליצירת טקסט:

- המטרה: יצירת טקסט חדש שדומה לקיים.
- ביצוע: על ידי האלגוריתמים -
  - חילוק המידע ל Tri-Grams.- שמירה על השכיחות של קבוצת המילים.
  - נתחיל מ2 מילים וכל פעם נדגום את המילה הבאה בהינתן ה 2 מילים הקודמות לפי חלוקת הנתונים.

בסופו של דבר ההשלמה של הטקסט תתבסס על השכיחות של מילה להופיע לאחר 2 המילים שכבר רשמנו.

Context Free Grammar- CFG

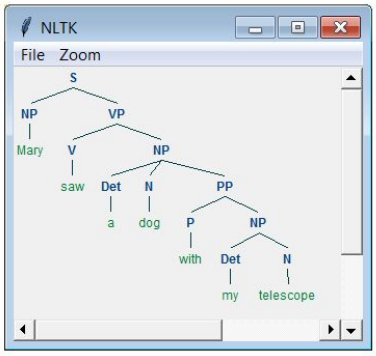
כמו באוטומטים 2, שפות חסרות הקשר מורכבות מ:

- **T -** אוצר מילים של טרמינלים. המילים עצמן "dog, bird"
- **V -** אוצר מילים לא של הטרמינלים. Adj , noun ...
- **P -** כללי מעבר.
- **S -** ההתחלה.

לפי הגדרה מראש של המשתנים הנ"ל לבנות עץ של חלקי הדיבר של המשפט שנכניס.

CFG Parser

בהינתן משפט ה Parser יפרק את המשפט לעץ של המשפט לפי ה grammer שהוגדר ע"י פונקציות המעברים (גם היא הוגדרה על ידינו) :




```
>>> grammar1 = nltk.CFG.fromstring("""
S -> NP VP
VP -> V NP | V NP PP
PP -> P NP
V -> "saw" | "ate" | "walked"
NP -> "John" | "Mary" | "Bob" | Det N | Det N P
Det -> "a" | "an" | "the" | "my"
N -> "man" | "dog" | "cat" | "telescope" | "park'
P -> "in" | "on" | "by" | "with" """)
```

הפלט שנקבל יכול להכיל בתוכו כמות מסוימת של ייצוגים שונים של פירוק המשפט.

CYK

האלגוריתם נועד כדי למצוא עץ אשר מתאר את המבנה הדקדוקי של המשפט בהסתברות הגבוהה ביותר.

סיבוכיות: $O(n^3|G|)$.

- **לפני שימוש:** עלינו להמיר את הכללי המעבר לצורה הנורמלית של חומסקי.
- **השימוש:** בהינתן משפט וכללי מעבר בצורה הנורמלית של חומסקי - ניצור טבלה (בגודל המילים) בצורה כך שהשורה הכי תחתונה תייצג כל מילה.

לאחר מכן בכל צא מעל נרשום את כללי המעבר שמביאים את המילה, באיטרציה הבאה נבדוק זוגות, לאחר מכן שלשות וכל הלאה.. עד שנגיע לאלכסון שייצג את המבנה של המשפט.
דגש: משתמשים כל פעם ב2 כללי מעבר בלבד. (לכן אם רוצים שלושייה ניקח את כלל המעבר שמביא לאחד מהמילים בשילוב עם הכלל מעבר שמביא ל2 האחרים וכן הלאה).

CoreNLP

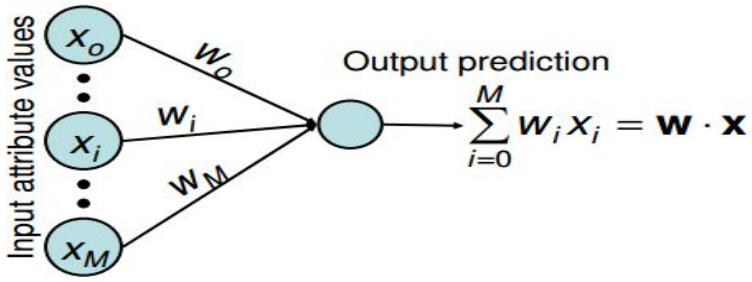
ספריה בJAVA חזקה יותר מ NLTK מכילה את הפיצ'רים:

- ניתוח תלות.
- דקדוק גדול ומובנה.
- מציאת מילים שמדברות על אותה ישות בטקסט.

Sentiment Analysis

לדעת אם תוכן הטקסט הוא חיובי או שלילי.
יש המון שיטות למימוש.

Linear Regression - המודל הליניארי הפשוט



הפונקציה הליניארית: $y = wx_i + b$

פונקציית loss:

$$J(w,b) = \frac{1}{2m}\sum_{i=1}^{m}(wx_i + b - y_i)^2$$

פתרון זה לא עובד כשיש לנו המון דאטא -

$$W = (X^T X)^{-1} XT$$

החישוב שלו יקר ולכן נעשה שימוש בGD וכו..

איך הגענו ל loss?

כדי להגיע למודל מדוייק שנותן לנו את הייצוג של כל y_i אנחנו נניח כי בנוסף למודל יש גם רעש, ז"א אפסילון כלשהו שהמודל טיפה יטעה בנוגע לערך של y_i. בצורה פורמלית:

$y_i = h(x_i) + \varepsilon_i = wx_i + b + \varepsilon_i = y' + \varepsilon_i$ כאשר אנו יודעים: $\varepsilon_i \sim N(0, \sigma^2)$ (מתפלג נורמלי)

(ε_i - הוספת רעש) ו $h(x_i)$ היא הפונקציה של המודל.

$y_i - h(x_i)$

$$P(y_i | x_i; w, b) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(y_i - (wx_i + b))^2}{2\sigma^2}}$$

(מה ההסתברות שנקבל את y_i בהינתן שהפונקציה שלנו)

MLE- Maximum Likelihood Estimation

אנחנו נרצה למצוא את w, b אשר ימקסמו לנו את ההסתברות שהמודל יחזיר ערך כמה שיותר קרוב לערך y_i המקורי בהינתן x_i בטעות של לכל היותר ε.

מניחים כי הדאטא שלנו הוא הכי סביר שניתן לקבל, מכאן אנו מניחים שהמודל יעבור טוב גם על דאטא אחר.

ההנחה : אנו רוצים למצוא מודל שממקסם את ההסתברות לדאטא שנתון לנו.

מקסום ה-y זה כפל של ההסתברות הנ"ל.

למצוא מקסימום ל ε זה קשה לכן נעזר ב LOG (כי הוא פונקציה מונוטונית).

במקום למקסם את ההסתברות, נצמצם את ה LOG.

Gradient Descent

כדי לתקן את השגיאה שקיבלנו מפונקציית ה loss שלנו נבחר α (learning rate) נעשה לאחר כל הרצה צעד בכיוון הופכי לנגזרת ונתקן את w ואת b עד שנגיע לנקודת המינימום.

גרדיאנט לפונקציה ה loss הלינארית: (נגזרת לפי w ונגזרת לפי b)

$$\nabla(J) = (\frac{1}{m}\sum_{i=1}^{m}(wx_i + b - y_i)x_i, \frac{1}{m}\sum_{i=1}^{m}(wx_i + b - y_i))$$

התיקון של w ו b:

$$\text{Update w to w-}\alpha\frac{1}{m}\sum_{i=1}^{m} x_i(h(x_i) - y_i)$$

$$\text{Update b to b-}\alpha\frac{1}{m}\sum_{i=1}^{m} 1 \cdot (h(x_i) - y_i)$$

כיצד לשפר את המודל?

1. להוסיף עוד פיצ'רים.
2. הוספת פיצ'ר בריבוע. (במקום מודל ליניארי יהיה פרבולה)

Regularization

שאנחנו במצב של overfitting נרצה לשים אילוצים על המודל בשביל למנוע מהמודל ללמוד את הtrain טוב מדי, הגבלת המודל על ידי כך שיגרום למודל לשלם על כל פיצ'ר שהוא משתמש.

- **Lasso-** $\lambda\|W\|$ הוספה ל **Loss** של סכום כל המשקולות בערך מוחלט.
  אגרסיבי יותר כלפי משקולות קטנות יותר, ולכן מאפס הרבה מהם בסופו של דבר.
  בעצם מוריד משקולות שאין להן כמעט השפעה על חישוב ה y .
- **Ridge Regression -** $\frac{\lambda}{2}\|W\|_2{}^2$ הוספה של של כל המשקולות בריבוע - יבדוק שהמשקולות לא יהיו גבוהות מידי. יכול להיות הרבה משקולות קטנות אך הן לא מתאפסות.

נשים לב: עבור $W > 0$ הנגזרת היא λ ועבור $W < 0$ היא $-\lambda$ וזה מה שדוחף את כל המשקולות להיות 0.

פתרונות ל overfitting:

1. להשתמש בירידה המתואמת. (לא קיים ב TF).
2. ניתן להשתמש באופטימיזציה שונה כמו ADAM.
3. להפחית את קצב הלמידה או לצמצם באופן ליניארי.

Early Stopping

שיטה נוספת רגולריזציה.
כדי לא להגיע ל overfitting , ברגע שנראה המרחק בין ה validation ל train גדל (באחוז השגיאה) נדרוש הפסקה יזומה של תהליך הלמידה ושימוש במודל במצב זה.

BGD, SGD, MB-GD

- **BGD -** משתמש בכל ה DATA כדי לחשב את הגרדיאנט. (יכול להיות קשה לחישוב).
- **SGD -** מבצע חישוב על דוגמא בודדת אך הצעדים יותר "רועשים" כי אנו עושים כל פעם צעד לכל דוגמה.
- **MiniBatch-GD -** מחשבים כל פעם על קבוצה דוגמאות וכל הנגזרת יותר מדוייקת ולא צורכת הרבה זיכרון.

Normalization

נרצה שכל הדאטא שלנו יהיה מנורמל כך שממוצע הערכים יהיה 0 וסטיית תקן 1.
קריטי לרגולריזציה.
הערה: מי שקיבל את המשקל הכי גדול אז הוא הכי משפיע.
לא מנרמלים לייבלים - יכול לצאת חסר משמעות.

Classification

Logistic Regression

המודל של linear regression לא יתן חלוקה נכונה לקלאסים לכן נעשה שימוש ב Logistic.
הפונקציה הלוגיסטית: $g(z) = \frac{1}{1+e^{-z}}$
פונקציית הפרדיקציה: $h(x) = \frac{1}{1+e^{-(xW+b)}}$
בעצם כאשר הפונקציה היא 1 אנחנו בטוחים כי הערך הוא 1.
באופן כללי אנחנו רוצים:

$$p(y_i = 1|x_i; w, b) = h(x_i)$$
$$p(y_i = 0|x_i; w, b) = 1 - h(x_i)$$

סה"כ:

$$p(y_i|x_i; w, b) = h(x_i)^{y_i}\big(1 - h(x_i)\big)^{1-y_i}$$

פונקציית LOSS:

$$J(w, b) = -\frac{1}{m}\sum_{i=1}^{m}\left(y_i(log(h(x_i))) + (1 - y_i)log(1 - h(x_i))\right)$$

פונקציית הגרדיאנט:

$$\frac{1}{m}\sum_{i=0}^{n} x_i(h(x_i) - y_i)$$

- **ירידת שיפוע ברגרסיה לוגיסטית:**
בוחרים w,b רנדומליים וקצב למידה וכמו לינארית נחזר על הפעולה עד להתכנסות.

נשים לב: כי החישוב של הגרדיאנט נעשה באותה צורה כמו במודל הליניארי השוני ביניהם הוא בחישוב ה $h(x)$.

בלי קשר: Bag of words

מודל המשמש לעיבוד שפות טבעיות ושליפת מידע מטקסט, תוך התעלמות מדקדוק וסדר המילים אך נותן משקל למספר הופעות של מילה בטקסט.
מודל זה משמש בדרך כלל לשיטות סיווג מסמכים או תדירות הופעת מילה היא הדרך לסיווג טקסט.

מה זה סיווג טוב?

נבדוק Recall & precision:

Confusion Matrix	Classified as Positive	Classified as Negative
Really Positive	True Positive	False Negative
Really Negative	False Positive	True Negative

איך נבדוק Accuracy?

Trues לחלק בכל הדגימות
או להסתכל על זה- true pos + true nag לחלק לסה"כ. $(true\ pos\ + true\ nag\ )/all\ data$

הRecall

זה יהיה true pos לחלק ל true pos + false neg, מה מוגדר כחיובי לחלק במה שבאמת חיובי.
$true\ pos\ /\ true\ pos\ +\ trueneg$

הPrecision

זה יהיה true pos לחלק ל true pos + false pos) (classified as pos
הערה : כשיש לנו משהו ממש חשוב שנרצה תוצאה מדויקת נעשה שימוש ב recall וב precision
$true\ pos\ /\ true\ pos\ +\ false\ pos$

הF-measure

זה השילוב של recall ושל precision.
$2 * p * r\ /\ p + r$
(כאשר p זה precision ו r זה recall)

Imbalance Data

פתרונות למידע לא מאוזן:

1. תת דגימה Under-sampling:
   a. התעלמות מחלק מהמידע (לא טוב).
   b. בכל Epoch דגימה באופן רנדומלי של דאטא מתוך קלאס גדול.
2. יתר דגימה Over-Sampling:
   a. שימוש באותו דאטא הרבה פעמים (רק ב train).
3. התערבות בפונקציית loss:
   a. הפסד משוכלל - (weighted loss) הכפלת ה loss של כל קלאס. (מחוץ ל log).
   b. שימוש ב cross_entropy_with_logits בקוד.

(שניהם יעשו $\frac{total\ number\ of\ samples}{number\ of\ samples\ in\ class\ i}$)

4. העלאת או הורדת סף החיזוי - הסף הקלסיפיקציה לקאסים.

Multiple Class

כאשר יש לנו מספר רב של קלאסים יש צורך לעבוד בשיטה שונה.

- **One hot:** יצירת מטריצה עבור כל הלייבלים כאשר כל וקטור מייצג לייבל בו הקלאס שהלייבל מתייחס אליו הוא 1 וכל השאר אפסים.
- **שכבת הפעלה SoftMax:** מנרמלת את התוצאה של הלוגיסטיק ריגרשן עבור כל קלאס על מנת להציג את תוצאת הדיגמה במרווח הסתברות בין 0 ל1.

$$h(y=i|x) = \frac{e^{x^T W_i + b_i}}{\sum_{j=1}^{k} e^{x^T W_j + b_j}}$$

- **Cross Entropy:** המודל הרגיל עם MSE לא נותן ענישה מספיק קשה לטעות בחיזוי ולכן צריך לעשות שימוש ב Binary Cross Entropy אשר מעניש הרבה יותר על טעות.
בגדול: cross entropy נותן גרדיאנט עם שיפוע גדול יותר עבור חיזוי לא נכון.
למה לא נשתמש ב בעיית רגרסיה? כי בבעיית רגרסיה המידע הוא רציף ולכן טעות על החיזוי לא אמורה לעלות למודל הרבה. לעומת בעיית קלסיפיקציה שם ניבוי לקלאס לא נכון אמור לעלות הרבה למודל שלנו.

$$L = -\frac{1}{m}\sum[y_i ln(\hat{y}_i) + (1-y_i)ln(1-\hat{y}_i)]$$

- **Adam optimizer:** שיטה נוספת לעדכון אשר מוסיפה את הנגזרת לממוצע של הלמידה וגם מחלק את הממוצע של הלמידה במה שלמדנו, החילוק מונע קפיצות גדולות וממוצע גורם לכך שיש ירידה יותר חלקה לכיוון המינימום.

Numerical Issues

בעיות שיכולות לקרות בTF.
אם הפונקציה של Softmax הופכת ל 0 או 1 זה יכול ליצור בעיה ב log של ה Cross entropy.
ואם xW+b גדול מידי אז החזקה של e יכולה להיות אינסוף.
לכן נעשה שימוש בפונקציה אחרת שמשלבת בין ה2:
()tf.nn.softmax_cross _entropy_with_logits_v2

$$= \frac{1}{k} y_i \sum_i (L - l_i + log(\sum_{j=1}^{k} e^{l_j - L})))$$

MLP - Adding a Hidden Layer

כדי לשפר את המודל אמרנו שניתן להוסיף עוד לייבלים או להגדיל אותך בריבוע.
במקום זה נוכל להוסיף שכבת ביניים למודל אשר לומדת על הפיצ'רים, כל פיצ'ר הוא צירוף ליניארי של ה input פיצ'ר שלנו שכל אחד מהם מקבל משקולות.

לשים לב: אם נאתחל את כל המשקולות לאפס\אותו דבר אז כל נוירון יעשו את אותו דבר.
פתרון: נאתחל כל נוירון עם ערכי משקולות אקראיים.

Activation layers

בין כל שני שכבות חייבת להיות לנו פונקציית אקטיבציה.
סוגים:

–Logistic function

–Step function

–Tanh function

–ReLU

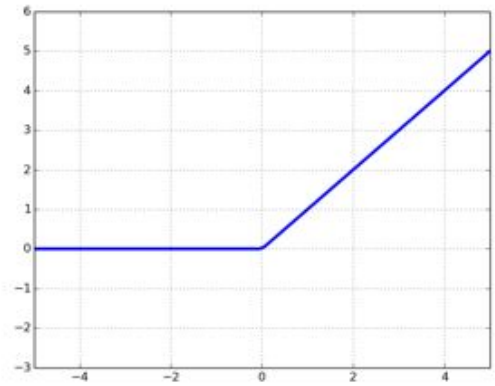
–Leaky ReLU

–ELU

–(Swish, SELU)

ReLU

- $ReLU(x) = \begin{cases} x & if\ x > 0 \\ 0\ otherwise \end{cases}$



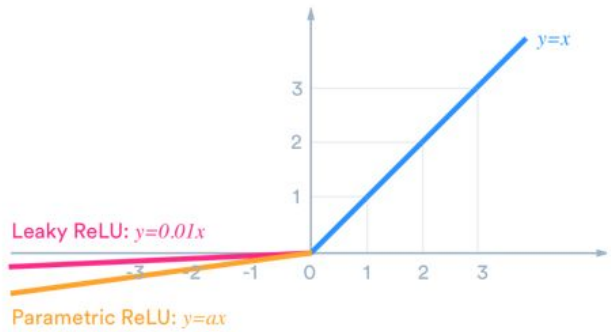
נועד על מנת להקל על חישוב הגרדיאנט ולמנוע בעיה של Vanishing gradient.

Leaking ReLU

פותר את הבעיה שאם הנגזרת היא 0 אז עדיין על הצעד לכיוון הנגזרת, יש השפעה על ה loss.

הנגזרת אף פעם לא תתאפס, בכל צעד יגדיל בטיפה.

- $ReLU(x) = \begin{cases} x & if\ x > 0 \\ \alpha x & otherwise \end{cases}$



Backpropagation

המטרה: לעדכן את המשקולות ואת הbias לאחר חישוב הטעות שלנו. רוצים לצמצם את ה loss.

לאחר חישוב הגרדיאנט נלך אחורה בנגזרת של הפונקציה ונעדכן כל משקל bais. בצורה הבאה:

$$w := w - \alpha \frac{\partial Error}{\partial w} \Big|_{(x1,x2\ldots,y1,y2,..w1,w2,\ldots,b1,\ldots)}$$

(בצורה דומה את ה bias)

Batch Normalization

נרמול הoutput של כל שכבה.

מה נעשה אם ה training error גבוהה מאוד?

Train error גבוהה מידי	Test error גבוה מידי, וה train error נמוך מידי (overfitting)
נמשיך עם האימון יכול להיות שהוא ירד ועצרנו אותו באמצע.	נוסיף עוד data.
נגרום למודל להיות מורכב יותר על ידי הוספת שכבות או להגדיל כל שכבה.	Early stopping.
הורדה או הגדלה של ה learning rate.	רגולריזציה: LASSO RIDGE DROPOUT
שימוש ב ADAM.	נוריד נוירונים.

CNN - Convolutional Neural Network

Convolution Layers:

משקולות: בניגוד למשקולות הרגילים פה יש שימוש ב Kernel שהוא מטריצה $K_n x K_m$.

ה Kernel משותפים עבור כל הנוירונים בשכבה. מאפשר יצירת מודל מורכב בקלות מסוימת.
שכבה זאת מנצלת את המבנה של ה input ע"י כך שהיא מוצאת את היחס בין כל 2 פיקסלים צמודים.
דרך פעולה:
בכל שכבה ניקח את הפיצ'רים שנוצרו בשכבה הקודמת (מטריצות של פיקסלים) ונכפול אותם ב kernel .
נסכום את התוצאות אחרי המכפלה ונשמור כפיצ'ר חדש.
כך נוציא הרבה פיצ'רים (תכונות מהתמונה במטריצות).

פרמטרים להגדרת Convolution Layers:

1. גודל ה kernel.
2. מספר ה kernels.
3. Stride: כמה צעדים רוצים לזוז בכל כיוון בביצוע הקונבולוציה.
4. Padding: הוספת מסגרת לתמונה בהתאם לתוצאה שרוצים. (בדרך כלל 0-ים)
   a. Valid - לא מוסיפים מסגרת. (מקטין לנו את המטריצה לאחר הקונבולוציה)
   b. Same - מוסיפים אפסים כדי להישאר באותו גודל מטריצה.
   c. Full - הוספת אפסים כדי שגודל ה output יהיה גדול יותר מה input.

נקודות נוספות:

1. הוספת bias לכל אחת מהמטריצות בפלט.
2. לאחר שכבת הקונבולוציה מגיעה שכבת הפעלה.

Max pooling

דרך נוספת להקטנת הקלט.
בכל פעם נסתכל על תת המטרציה בגודל ה kernel ו"נשלוף" את האיבר המקסימלי.
נבצע את זה לאחר קונבולוציה אחת לפחות.

נשים לב: כאשר לוקחים את המקסימום אנו לוקחים את החלקים הרלוונטיים שמרכיבים את התמונה או הפיצ'ר שאנחנו רוצים לזהות.

(Dropout (Regularization

נועד כדי למנוע overfitting בכך שמבטל נוריונים בכמות מסוימת בשכבה (כל פעם נוירונים אחרים).

סיבות לשימוש:

1. מונע overfitting על ידי כך שזרימת המידע משתנה בין איטרציות.
2. מניעת אבטלה סמויה של נוירונים (הרשת לא תסתמך על משקולות גדולים).
3. מעלה את סיבוכיות הרשת.

Deep Reinforcement Learning

MDP: S,A,R,T

MDP - Markov decision process

- **States = S -** רשימת המצבים שאליהם יכול להגיע הסוכן.
- **Action = A -** הפעולות שהסוכן יכול לבצע.
- **Reward = R -** פונקציה שבדרך כלל קשורה ל state מסוים או ל state + action. לפיה מחושב התגמול של הסוכן.

- **Transition = T -** פונקציית הסתברות להגעה מהנקודה בה הסוכן נמצא לנקודה אחרת.

Discount Factor

ערך (קטן מ 1) שנועד להוריד את הערך של נקודות המתקבלות בשלבים מאוחרים יותר של המשחק.
כל שלב (חוץ מהראשון) הערך יוכפל בנקודות שהסוכן קיבל עבור אותו שלב (קח יקטין את מס' הנקודות לאותו שלב) ובנוסף יוכפל עבור הסכימה של כל הנקודות עד לאותו שלב.
המטרה- נקודות שנאספות בשלב מוקדם יותר יתועדפו על פני נקודות בשלב מאוחר יותר.

(Model Free (MDP

- אנחנו מקבלים את המצב הנוכחי
- אבל לא מקבלים את T ואז R מהנחת מרקוב.
- אנחנו מכירים את כל הצעדים שאנחנו יכולים לעשות Actions.

https://stats.stackexchange.com/questions/407230/what-is-the-difference-between-policy-based-on-policy-value-based-off-policy#:~:text=Policy%2Dbased%20vs.&text=In%20Policy%2Dbased%20methods%20we,a%20mix%20of%20the%20two.

Off Policy

אם פעולת העדכון ערך שלנו מנסה לעשות עדכון ערך בצורה חמדנית לדוגמה, זה יכול להיות שונה מה policy שהוגדר בפועל ולכן אלגוריתמים כאלה הם off policy

Q-Learning Algorithm - Basic model:

אלגוריתם שהוא off policy, לכן זה אומר שאנחנו לא יודעים את העולם שלנו וצריך "לחקור".

Q-Values:

$Q_{\Pi}(s,a)$ עבור זוג (state, action) - תוחלת (ממוצע) הרווח של הסוכן אם הוא נמצא ב state =s ויבצע את פעולה Action=a בהתאם למדיניות Pi שהוגדר לסוכן (לך אל המקסימלי מבין כל האפשרויות).

Q-Learning - תהליך הלימוד:

מתחילים מערכים רנדומליים (או אפסים) ולאט לאט לומדים מה הערכים האופטימליים לכל state.
לאחר ביצוע action וקבלת ה reward וסקירה של המצב החדש ה- Q learning מעדכן את ה Q value של המצב הנוכחי באופן חלקי, עדכון זה נעשה באמצעות קצב הלמידה α (בדומה ל SGD).
ה Q learning מוסיף את הדלתא בין הערך הישן לערך החדש הצפוי.

$$Q(s_t,a_t) \leftarrow \underbrace{Q(s_t,a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \left(\overbrace{\underbrace{r_{t+1}}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1},a)}_{\text{estimate of optimal future value}}}^{\text{learned value}} - \underbrace{Q(s_t,a_t)}_{\text{old value}} \right)$$

Wikipe

הסבר: כל מצב מעדכן את הערך שלנו ב"צעד" לכיוון הפעולה הבאה שתיתן את התוצאה המקסימלית וימשיך לעדכן כל פעם עד שנגיע למצב שבו ה Q value של נקודת ההתחלה יהיה הערך המקסימלי שניתן יהיה להגיע אליו לאחר שלמדנו את הערך של כל יתר המצבים.

אנחנו לא מקבלים את ערכי ה-Q ולכן צריך ללמוד אותם:

שיטות למידה:

- **Exploration -** ביצוע צעדים שיכולים להיות לא אופטימליים אבל יכולים ללמד את הסוכן דברים חדשים.
  - **שימוש בשיטה אפסילון גרידי -** בהסתברות אפסילון כלשהי הסוכן יעשה צעד רנדומלי על מנת לחקור את השטח ולהכיר עוד מסלולים. ככל שהאלגוריתם מכיר יותר כך האפסילון שלנו קטן על מנת שיפסיק לחקור בהמשך.
- **Exploitation -** ביצוע פעולות על סמך פעולות שכבר בוצעו בהתאם למדיניות.

Q-Learning Algorithm - Regression model:

מה קורה שיש הרבה מצבים states:
כבר לא נוכל להחזיק טבלת Q_vals בגודל הזה לכן נהפוך את הבעיה לבעיית רגרסיה כל שבהינתן מצב נעשה נעשה חיזוי של ה Q value הטוב ביותר לפי הצעדים שניתן לעשות.

מה זה אומר? כל נוירון ייצג צעד שאנחנו יכולים לעשות ב state הנוכחי, יעשה על הפעולה linear regression ויפלוט חיזוי לערך של הצעד בכיוון. לאחר מכן נבחר את המקסימלי ונתקדם.
ההבדל הוא שאת הלייבים אנחנו נלמד תוך כדי תנועה ולא נדע בהתחלה את ה"לייב" הנכון.

על מנת להשיג את הערך ש Q האמיתי:
- נבצע פעולה (שממקסמת את Q או Explore)
- נקבל את r עבור הפעולה
- נקבל state חדש.
- נחשב את ערכי ה Q של ה state, עבור כל אחת מהפעולות האפשריות.
  כאשר q_n מציין את ה Q המקסימלי שניתן לצבור.
- כעת ערך ה Q האמיתי עבור צמד הפעולות הקודם הוא : $r + \gamma \cdot q_n$.

שיפורים למודל:
- להפוך את הבעיה מ linear regression לבעיה של Deep CNN.
- נעשה שימוש ב 2 שכבות קונבולוציה עבור סט של פריימים עבור כל מצב ע"י שימוש ב lstm בשביל לקבל את התזוזה. כאשר שכבת ה output היא כמות ה actions שניתן לעשות .
- נתחיל לשחק במצב רנדומלי עבור X משחקים ראשונים (עדכון Q)
- ע"מ לשפר את מהירות הלימוד נשמור את ההיסטוריה של הצעדים שנבחרו בעבר.
- נזין באצים של ההיסטוריה ששמרנו באופן רנדומלי.
- נשחק המון משחקים באופן אסינכרוני ונשלב את תוצאות ה loss שהתקבלו.

On Policy RL

מה שהיה עד כה זה OFF POLICY.
ON POLICY זה אומר שאנחנו עושים שימוש ב POLICY האמיתי, הוא בודק באמת מה נעשה בפעם הבאה ונשתמש בפעולה הזאת. (לא נעשה את ה max של הצעד הבא בעצם).

$$Q(s_t, a_t) \leftarrow \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \left(\overbrace{\underbrace{r_{t+1}}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\boxed{Q(s_{t+1}, a_{t+1})}}_{\text{estimate of optimal future value}}}^{\text{learned value}} - \underbrace{Q(s_t, a_t)}_{\text{old value}} \right)$$

SARSA

State action reward state action
גרסת ה ON POLICY של Q learning ההבדל היחיד הוא בכך שהוא בוחר תחילה את הצעד הבא שלו ורק ח"כ מחשב את העדכון.

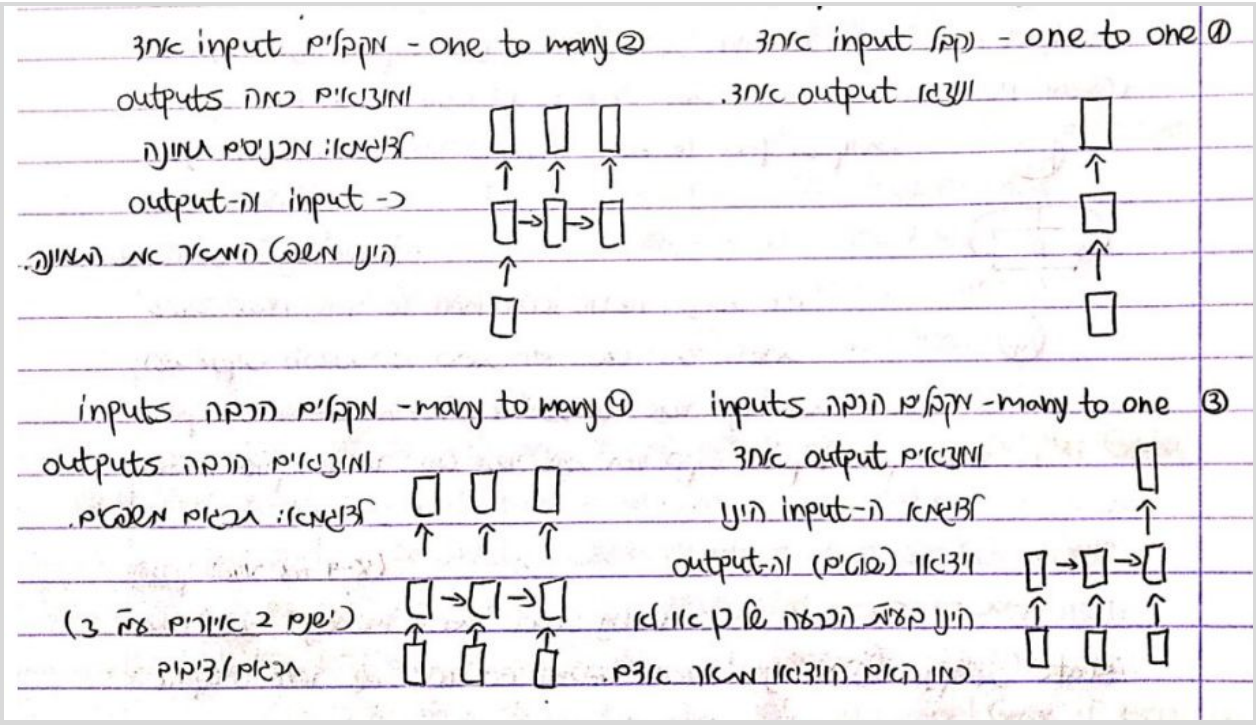
Value\Policy Based

לעומת Q learning ו SARSA שמחשבות את הצעד הבא שלהן על בסיס ה value שהוא יניב להן, Policy Based Methods מנסות ללמוד את החוקיות של כל צעד ולשפר את החוקיות לצעד הבא.
נעשה זאת ע"י כך שהרשת נוירונים שתנבע הצעד הבא ע"פ ההסתברות שהוא יניב בהמשך את הערך הגבוהה ביותר.
מטרה: מקסום של תוחלת הרווח שניתן לעשות על ידי הסתברות לכל צעד.

RNN

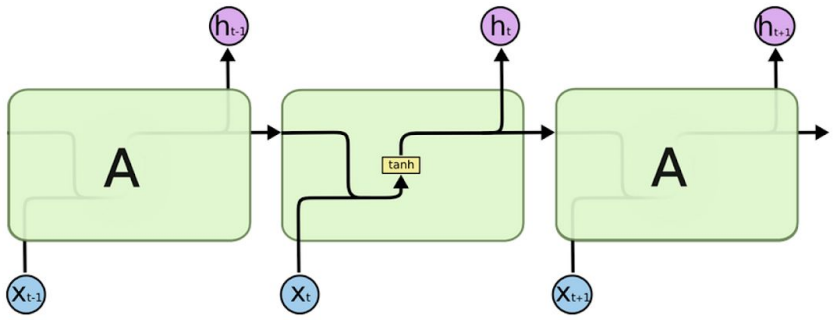
עיבוד משפטים - Sequences

סוגים של Sequences:



RNN

רשת מעגלית שבה כל שכבה פולטת את הפלט שלה לשני כיוונים.
כיוון 1: פלט.
כיוון 2: כקלט לשכבה הבאה.
בצורה זו מכריחים את המודל להתייחס לסדר המילים במשפט.



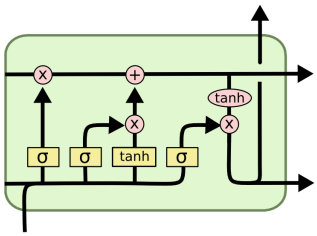
שימוש: הכנסת ה input לתוך פונקציית tanh.

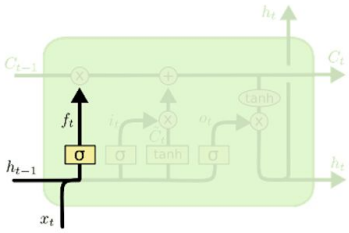
בעיה: חישוב גרדיאנט - Vanishing Gradient
בגלל שגודל הרשת תלוי בכמות המילים זה יוצר רשת ענקית שבמהלכה חישוב הגרדיאנט עד המילה הראשונה עלול להיעלם או להתפוצץ.

פתרון:
LSTM
הפלט עובר גם לעצמו וגם לשכבה הבאה.
יש תוספת של state שנשאר ב LSTM והרעיון הוא הוא שבתוך ה state ה data יכול לזרום ללא שינוי כך שאנו יכולים לשנות משהו קטן בהתחלה ולראות בסוף איך הוא משפיע ספציפית על ההתחלה.

מבנה:

- **Forget gate:**



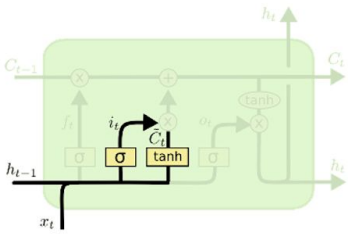


Forget Gate:

$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$

שער המאפשר לשכוח מידע או חלק ממנו. (state)
עובד על ידי Logistic regression : הפעלת פונקציית סיגמואיד על הכפלת המשקולות על הקלט והפלט מהשכבה הקודמת והוספת biasה.
אם מקבלים 1 = שמור את המידע, אם מקבלים 0 = שכח את המידע הקודם.

- **Input gate:**



Input Gate:

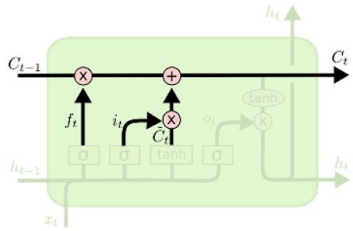
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

כאשר נרצה להכניס מידע לתוך ה state נשאל 2 שאלות:

1. האם נרצה לזכור? (כמו ה forget gate)
2. מה נרצה לזכור? חישוב על ידי ה tanh.

- **Flow Gate:**

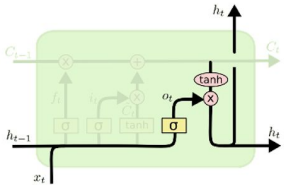


Flow:

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

עושה עדכון ל state בין מה שהיה לבין מה קלט החדש.

- **Output gate:**

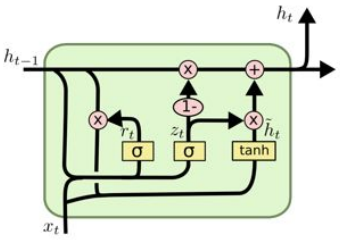


Output Gate:

$$o_t = \sigma\left(W_o\ [h_{t-1}, x_t]\ +\ b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

שולט באיזה אינפורמציה נשלח לרמה הבאה.

GRU



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

שיטה עוד שיטה כמו LSTM רק שהוא שוכח את ה state הקודם ושומר את הבא באותה פעולה.
(forget ו input מחוברים). תופס פחות מקום אבל ביצועים פחות טובים מ LSTM.

Seq2Seq

שימוש ב LSTM כדי לעשות תרגום בשיטת: Encoder Decoder.

הרחבה לזה:

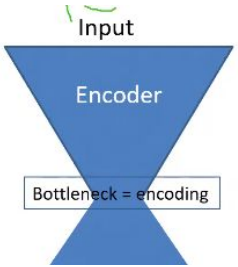
Attention

המודל של ה Decoder מקבל לא רק את הפלט הסופי של ה Encoder אלה גם מכל שלב את ה output ונותן "תשומת לב" לחלקים הרלוונטיים לו.

Auto Encoder

מטרה: כאשר יש לנו המון DATA שהוא unsupervised ז"א אין לנו לייבלים. הקלט שלנו הוא הלייבל ה"נכון" שמולו אנחנו משווים את התוצאה שקיבלנו.

יש לנו את החלק הצר לשם אנחנו צריכים ל"דחוס" את הקלט ואז לפענח אותו ולראות שקיבלנו תוצאה זהה לקלט.



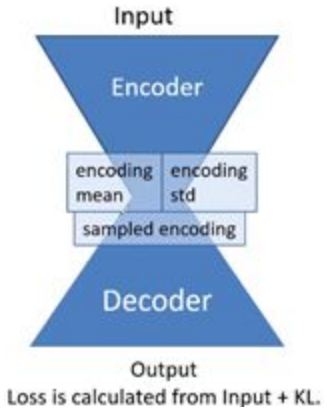
ה LOSS - זה בעצם ההשוואה לקלט המקורי ולבחון את ההבדל. (MSE)
בעצם רואים כמה טוב הצלחנו לשחזר את הקלט בהינתן "תקציר" של הקלט שיצרנו.
שימושים: הורדת רעש וכו..
תהליך העבודה:

- **ה encoder** : יכול להיות כל סוג של רשת.
- **צוואר בקבוק:** פירוק הקלט לתכונות החשובות.
- **ה decoder :** הרכבה מחדש של הקלט.
  - יכול להיות fully connected.
  - יכול להיות Deconvolution.

בעיה של Semi supervise learning :
ניקח את כל ה DATA שיש שהוא ללא לייב וניקח כמות מתויגת של DATA וננסה להעביר את הקלטים הלא מתוייגים לתוך ה encoder כך שבאמצע הוא ינסה לשייך ללייב הנכון (לפי הDATA המתויג שלנו).

VAE -variational autoencoder

במקום למפות כל קלט לווקטור קבוע (בצוואר בקבוק) נרצה למפות אותו לפי ההסתברות כך שהדבר היחיד ששונה הוא שצוואר הקבוק שהיה וקטור בודד מוחלף ל 2 וקטורים האחד ממוצע והשני של סטיית תקן.
כך שבכל פעם שאנחנו צריכים "קלט" ל decoder אנחנו בוחרים דגימה ממרחב ההסתברות.



כיצד נוכל לגזור כדי לבדוק?
Sampling - בעצם כאשר אנחנו לוקחים דגימה במקום לקחת אותה בצורה ישירה ממרחב ההסתברות ניקח את הממוצע + השונות עם טיפה רעש ($\varepsilon *$)
בשביל למנוע שהשונות שלנו תהיה 0 - נוסיף ל loss את מדד KL מה שיגרור שהתוצאה "תימשך" לכיוון בו השונות = 1 והתוחלת = 0 . נשאר במצב גאוסיאני.

Word Embedding

הרעיון הכללי: ייצוג של מילה בצורת וקטור.

Very Basic Word Embedding

הרעיון: במקום לעשות את ה one hot הרגיל נעשה כך:
לכל מילה ניתן אינדקס i , וניצור מטריצת שכנויות לכל מילה (על סמך "משפחות מילים") כך שבכל תא $j \neq i$ נכניס את ההסתברות של המילה ה i להיות דומה\קרובה למילה ה j.
לאחר שיצרנו את המטריצה, נרצה לחפש שני וקטורים שהמכפלה שלהם יוצרת מטריצה זהה (או ממש דומה) למטריצת השכנויות שלנו.

- **דגש:** גם אם יש לנו חלקים אינפורמציה (ז"א תאים ריקים) נחפש מטריצה שדומה כמה שיותר לתאים שכן יש לנו עליהם אינפורמציה.

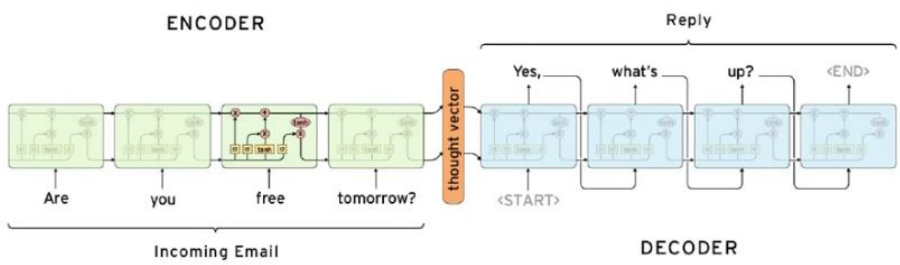
בצורה הזו נקבל ייצוג נכון יותר לכל מילה לפי משמעות כך שגם מילים שונות בעלות אותה משמעות יקבלו וקטורים דומים.
לפי הדוגמה: בהינתן משפחות מילים שקרובות במשמעות, ניצור וקטור לכל מילה ובכל תא j נכניס את מספר הפעמים שהמילה עם האינדקס j הופיעה באותה משפחה.
לאחר מכן נחלק בכמות הכוללת של המילים שיש לנו כך נקבל את ההסתברות.

Skip Gram Word2vec

הרעיון: בהינתן רשת שמנסה לנבות סביבה של מילה, השכבה לפני האחרונה תיתן לנו את הייצוג הווקטורי של המילה שהכנסו כקלט - כך נוכל לקבל את הוקטור שהרשת חושבת שהוא ייצוג נכון של המילה.
קושי: להבין ניגודים במשפט.
דרך מימוש: Spacy - ספרייה שדרכה ניתן לעשות word2vec.
כיצד לבדוק תוצאות: שימוש בזווית של כל וקטור במרחב האוקלידי כדי לראות שהם נוטים לכיוונים דומים.

Sentence 2 Vec From Translation

הרעיון: יצירת רשת של lstm לתרגום (שיטת ה encoder decoder שלה) ואימון הרשת על המון דוגמאות.
כל דוגמה כזאת ניתן לקחת את הייצוג הווקטורי של המילה מאמצע הרשת (thought vector).
כך נרכיב לנו מאגר של וקטורים, ובהינתן מילה נהפוך גם אותה לייצוג וקטורי ונשווה מול היתר כדי למצוא את הווקטור הכי דומה מהמאגר וכך גם את המשמעות.

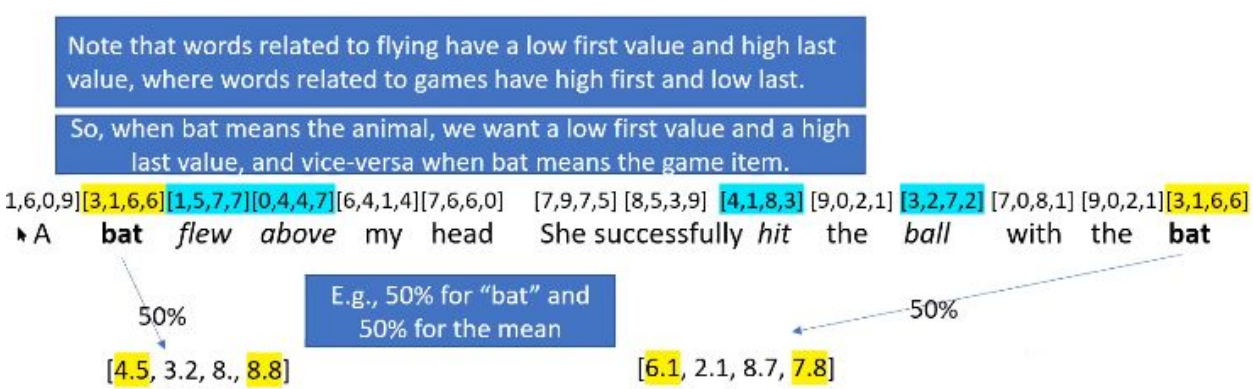


Word Embeddings With Context

הבעיה: בשיטות הקודמות כשיצרנו וקטור למילה יכולנו ליצור וקטור אחד למילה בעלת שתי משמעויות (לדוג bat)
פתרון:
ניסיון ראשון: נבנה וקטור מייצג עבור המילה בעזרת הנוסחה: כאשר כל תא בו יהיה סכימה של 50 אחוז מערך המילה הדומה + סכימה של כל הערכים במשפט במיקום i כפול החלק היחסי להשלמת ה 50 אחוז הנותרים.

$$Value = position\ i = Query[i] \cdot 0.5 + \Sigma_{Other\ Words} KeyWord_j[i] \frac{0.5}{|total\ words| - 1}$$

דוגמה:



למה זה לא טוב? אנחנו נותנים משקל גם למילים שלא קשורות למילה שאותה אנחנו מחפשים.

ניסיון שני:

Weightless Self-Attention

הנחת הבסיס היא: אנחנו מקבלים את ה embedding של כל מילה מ Word2Vec.
מטרה: במקום לתת אותו אחוז יחס לכל מילה, נבדוק את הזווית בין הווקטורים וניתן משקל לפי המרחק ביניהן, ככה שהווקטורים עם זווית קטנה יותר יקבלו יותר יחס (יותר דומים) ועם זווית גדולה יקבלו פחות.

- **Cos Similarity + SoftMax :** בעצם Dot product של וקטורים לחלק לנורמאל שלהם ולאחר מכן SoftMax על מנת לנרמל את התוצאות להסתברות.

$$Query = The\ word\ we\ want\ to\ create\ it's\ embedding$$
$$Key = All\ the\ words\ we\ have\ in\ the\ sentence$$
$$Value = The\ final\ vector\ after\ all\ the\ calculations$$

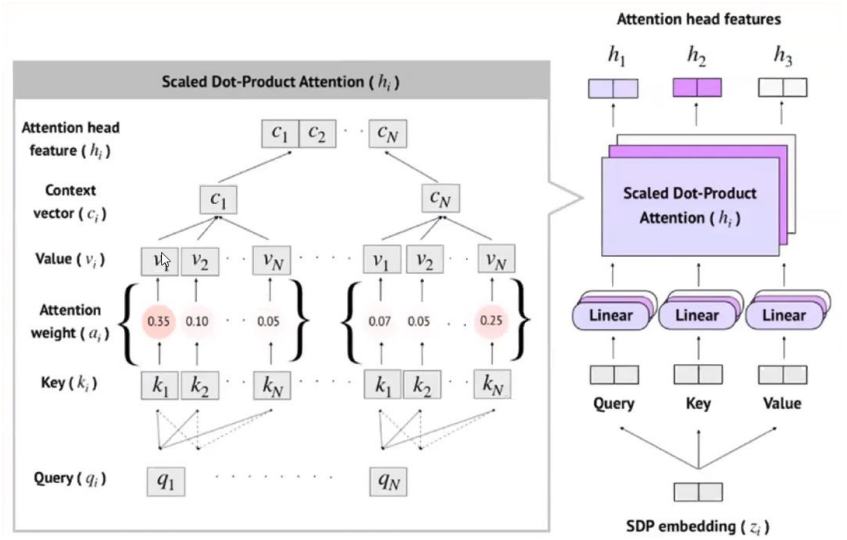
השלבים:

1. **חישוב Cos Similarity -** נקבל את הווקטור של הזוויות של כל Key מה-Query.
   נקבל את הוקטור: Cosine.
2. **נרמול SoftMax -** על כל Cosine נעשה SM על מנת להכניס את התוצאות להסתברות בין 0 ל1.
   נקבל את הוקטור: Attention.
3. **סכימה סופית -** ניקח כל משקל i מ Attention ונכפיל בערך במיקום ה i של כל וקטור Key ונסכום למיקום ה i בוקטור התוצאה שלנו - Final Embedding.

Adding Weights Self-Attention

הנחה: כאן אנחנו לא מקבלים את Embedding של כל מילה אלה צורת One Hot שלה.
מטרה: ללמוד את הווקטור של כל מילה בנוסף במקום לתת אותו אחוז יחס לכל מילה, נבדוק את הזווית בין הווקטורים וניתן משקל לפי המרחק ביניהן, ככה שהווקטורים עם זווית קטנה יותר יקבלו יותר יחס (יותר דומים) ועם זווית גדולה יקבלו פחות.
התהליך כמו מקודם בדיוק.

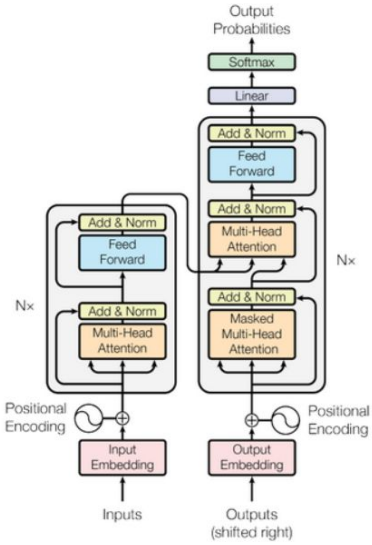
Multi-Head Attention



הרעיון: במקום לעשות את הפעולות הקודמות על מילה אחת כל פעם אנחנו נעשה פשוט במקביל על כל המילים ביחד. זאת על מנת להגיע ליצוג מלא של כל המשפט.

Transformer

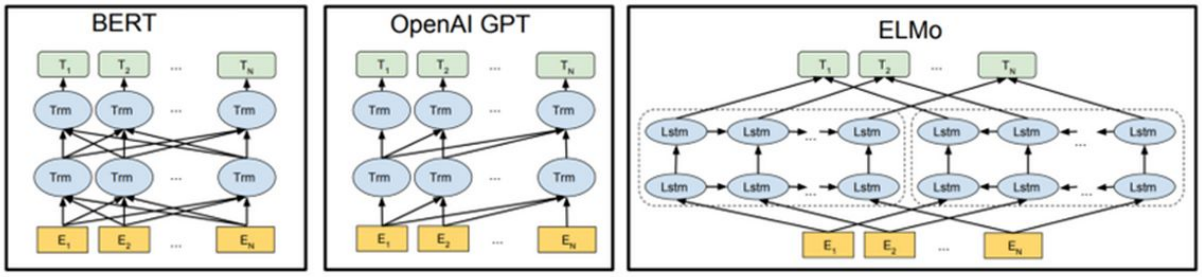
מטרה: לקבל embedding ולהוציא embedding "טוב יותר" או לתרגום לדוגמה.
Skip Connection - כאשר יש לנו "דילוג" בתוך המודל מעל פעולות מסוימות (כמו לדוגמה כאן מעל ה multi head attention).
מטרה: כאשר יש רשת עמוקה מאוד קשה לאמן ולכן נרצה לדלג על חלק מהשכבות על מנת "להתעלם" מהן בשלב זה ולאמן את השכבה שלפניה, ברגע ששכבה זו "מאומנת" מספיק, מפסיקים את הדילוג ומתחילים לקבל את הפלט של השכבה שהתעלמנו ממנה ולאמן אותה.
באיור: מעל multi head attention יש קפיצה, ז"א הקלט נכנס ישר ל Add and Norm, ברגע שזה מאומן מספיק נתחיל להכניס את הקלט מ MHA.



Positional Encoding - תוספת לקלט כך שנותן לכל מילה יחסות של מקום בתוך המשפט כדי ליצור הבדלה בין משפטים עם אותם מילים אבל בסדר שונה ומשמעות שונה.
Feed forward - רשת FF פשוטה שסה"כ לוקחת את וקטור ה Attention שיוצא לנו והופכת אותו לצורה שיותר "קלה" לשכבה הבאה.

דוגמה לפעולה:
ב Encoder אנחנו יכולים לקבל מילים באנגלית להעביר אותם דרך הרשת ונקבל בסוף וקטורים המייצגים כל מילה במשפט באנגלית שהכנסנו.
ב Decoder יעשה פחות או יותר את אותם שלבים בהתחלה רק על מילים משפה אחרת.
ה MHA האמצעי יקבע את היחס של כל וקטור (שזה מילה בעצם) ביחס לשאר הוקטורים, הפלט יהיה Attention וקטור של כל המילים לאחר מכן נעביר את יתר השכבות ולבסוף SoftMax יתן לנו את ההסתברות למילה הבאה.

BERT



מטרה: די דומה ל Auto encoder, הרעיון הוא לקבל משפטים עם מילים מוחבאות או חסרות נעביר אותם דרך Transformers ולעשות להם embedding ולאחר מכן לחזות את מה המילה שאמורה להיות שם.

דגשים:

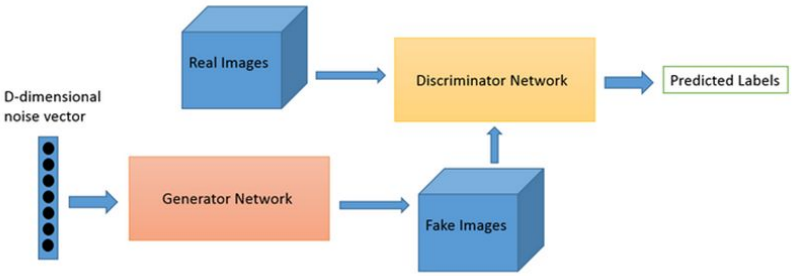
- הוא בעל אוצר מילים של סה"כ 30,000.
- האורך המקסימלי של הטוקנים שלנו 512.
- כל ה Transformers בכל שכבר זה אותו אחד עם אותם משקולות.

אימון unsupervised:

- אנחנו לוקחים 15% מהמילים בקורפוס (?) - מסמנים אותם.
  בניבוי אנחנו בודקים מול המילים המסומנות.
  - 80% נחליף אותם ב mask כלשהו.
  - 10% נחליף בטוקן רנדומלי.
  - 10% להשאיר כמו שהוא.
- המודל בסוף צריך להוציא את המשפט המקורי.
- ה Loss נמדד רק אל מול הטוקנים המסומנים.

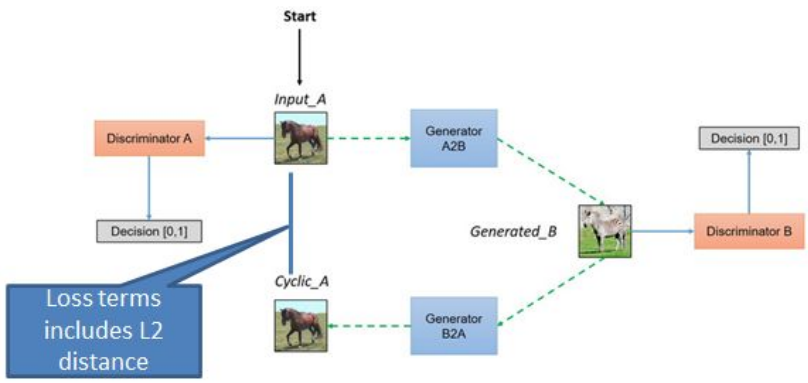
GAN

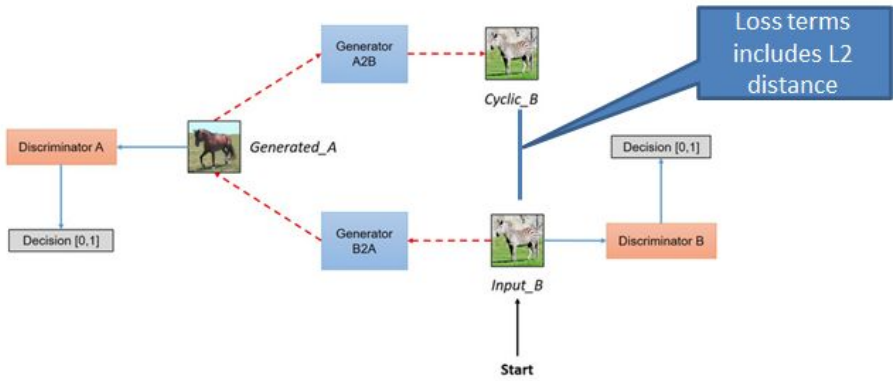
מטרה: יצירת תמונות שיראו אמיתיות.
נבנה מחולל שמייצר בהתחלה "רעש" ואז נבחן מול classifier (Discriminator) שמקבל תמונות מהמחולל וממאגר תמונות אמיתי וצריך להבחין מי אמיתי ומי לא.
בסוף רוצים שהוא יאמין שגם התמונות המזויפות הן אמיתיות.



דגש: נרצה לתת "פור" קטן ל Discriminator אך השאיפה היא שה Generator & Discriminator יתאמנו בצורה מתואמת.

Cycle GAN





נשתמש ב 2 Generators ו 2 Discriminators ע"מ להפוך סוס לזברה.
הרעיון: נלמד איך להפוך סוס לזברה ולהבדיל בין תמונה אמיתית או מפוברקת, כך נוכל ללמד את ה generator ליצור זברה על גבי הסוס.
ונעשה אותו דבר גם לכיוון השני כי כנראה זה יותר יעיל.

<u>שאלות מבחן בעל פה של עמוס:</u>

שאל אותנו על gan, q value ,gradient decent, Adam optimizer, reinforced learning

אותנו הוא שאל על autoencoder, cnn, הפונ שמאחדת את הsoftmax והcross entropy
Semi supervised learning auto encoders
שאל אותי על LSTM, RNN וBERT
שאל על הכל ADAM למה לעשות שימוש בSOFTMAX WITH CROSS ENTROPY על ATTNTION ועל self attention ועל word2vec
הוא שאל אותנו נגיד למה משתמשים ברשתות בDeepRL ולא בטבלה
שאל על QVALUE
שאל רעיון כללי של reinforcment learning
מזה batch normalization, במה אנחנו כופלים אותו.
מזהvae, איך מחושב הלוס של vae
מה זה הממוצע והסטיית תקן בvae
RL- q-learning, exploration/explotation, auto encoder
LSTM גודל קלט

אותנו הוא שאל על part of speech,nlp,rnn(lstm),gan