

HUMAN LANGUAGE TECHNOLOGIES
ACADEMIC YEAR 2019-2020
UNIVERSITY OF PISA
Luca Corbucci

Contents

1 Lesson 1 - Language Modeling	12
1.1 Introduzione	12
1.2 N-Gram	13
1.3 Shannon's Game - Generate new Text	18
1.4 Smoothing	19
1.5 Zipf's Law	21
1.6 Valutare un language Model	22
2 Lesson 2 - Representation of Words	29
2.1 Vector Space Model	30
2.2 Modeling Word Similarity	33
2.3 Word Embedding	34
2.3.1 Word2Vec	35
2.3.2 CBoW	41
2.3.3 Alternative a Word2Vec	42
2.4 Co-Occurrence Count	43
2.4.1 Considerazioni	44
2.5 Embedding in Neural Networks	45
2.6 Limiti	45
3 Vector Semantics and Embedding - Approfondimento da SLP Capitolo 6	47

3.1	Introduzione	47
3.2	Come lo rappresentiamo il significato di una parola?	48
3.3	Vector Semantic	50
3.4	Parole e Vettori	52
3.4.1	TF-IDF	53
3.5	Word2Vec	54
3.5.1	Classificatore	56
3.5.2	Learning	56
3.6	Approfondimento su Glove	57
4	Lesson 3 - Tokenization	62
4.1	Regular Expression	62
4.2	Tokenization	63
4.2.1	Words	63
4.3	Unix Tool per la Tokenizzazione	64
4.4	Trasformazione del testo	65
4.4.1	Word Normalization	65
4.4.2	Case Folding	66
4.4.3	Lemmatization	66
4.5	Sentence Segmentation	67
5	Lesson 4 - Text Classification	69
5.1	Classificazione	69
5.2	Naive Bayes	71
5.2.1	Training del classificatore Naive Bayes	73
5.2.2	Generative Model	74
5.3	Naive Bayes as a Language Model	75
5.4	Esempio di Naive Bayes	76
5.4.1	Learning Phase	76
5.4.2	Testing Phase	77
5.5	Multiclass Classification	79
5.6	Test set e Cross Validation	80

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

5.7 Conclusion	82
6 Lesson 5 - Neural Network Classification	84
6.1 Introduzione	84
6.2 Linear versus Non Linear Algorithms	85
6.3 Perceptron	86
6.3.1 Funzionamento del Perceptron	88
6.3.2 Perceptron come single layer neural network	91
6.4 MultiLayer Perceptron	92
6.4.1 Architettura della Neural Network	93
6.5 Softmax Classifier	97
6.5.1 Logistic Regression	97
6.5.2 Multinomial Logistic Regression	99
6.6 Traditional NLP vs Word Vectors	101
6.7 Sperimentare con Keras	101
6.7.1 1 Layer Perceptron	101
6.7.2 Perceptron con Word Embedding - Glove	106
7 Lesson 6 - Part of Speech Tagging	108
7.1 Markov Chains	108
7.2 Hidden Markov Models	110
7.2.1 Calcolo della Likelihood	114
7.2.2 Decoding: Viterbi Algorithm	117
7.2.3 Training dell'HMM	119
7.3 HMM per Part of Speech Tagging	120
7.3.1 Esempio di disambiguazione	123
8 Lesson 7 - Sequence Tagging	126
8.1 Discriminative e Generative Models	128
8.1.1 Logistic Regression	129
8.1.2 Maximum Entropy Markov Model	129
8.2 Problemi con i classificatori	131

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro *Speech and Language Processing* e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

8.3	Notebook: POS Tagging	132
8.3.1	Creazione del modello	135
8.3.2	Maximum Entropy Pos Tagging	137
8.4	Named Entity Recognition	138
8.4.1	Usare NER	139
8.4.2	Usare NER come sequence labeling	142
9	Lesson 8 - Parsing	144
9.1	Constituency Grammar	147
9.2	Dependency Grammar	151
9.3	Dependency Parsing	152
9.3.1	Transition Based	153
9.3.2	Oracolo	157
9.4	Non Projective Transitions	161
9.4.1	Parsing con swap	163
9.5	Learning Phase	164
9.5.1	CoNLL-X	165
9.5.2	Problemi con gli Oracoli	167
9.6	Dependency Parser con le Neural network	168
9.7	Graph Based Dependency Parsing	169
10	Lesson 9 - Universal Dependency	175
11	Lesson 10 - Opinion Mining e Sentiment Analysis	179
11.0.1	Il caso Cambridge Analytica	181
11.1	Sentiment Analysis	183
11.1.1	Un modello per la Sentiment Analysis	184
11.1.2	Direct Opinion	184
11.1.3	Feature Based Sentiment Analysis	187
11.1.4	Aspect based Sentiment Analysis	188

12 Lesson 11 - Sentiment Classification	191
12.1 Sentiment Analysis con i Tweet	192
12.2 Deep Learning per la sentiment analysis	194
12.3 Conclusione	197
13 Lesson 12 - CNN per NLP	198
13.1 Introduzione	198
13.2 CNN per NLP	200
13.2.1 Pooling Layer	206
13.2.2 Channels	207
13.2.3 Possibili utilizzi delle CNN in NLP	208
13.3 Implementazione di una CNN per NLP	209
14 Lesson 13 - Introduction to TensorFlow	213
14.1 Introduction	213
14.2 TensorFlow Computation Graph	215
14.3 Automatic Differentiation	216
15 Lesson 14 - Recurrent Neural Networks	224
15.1 Introduzione alle RNN	224
15.1.1 Tipologie di RNN	228
15.1.2 Training di RNN	232
15.1.3 Vanishing Gradient	235
15.1.4 Exploding Gradient	236
15.2 LSTM	237
15.2.1 Confronto	244
15.3 Implementare il POS Tagger usando RNN	245
15.3.1 Creiamo i layer	245
15.3.2 GRU	247
16 Lesson 15 - Machine Translation: Introduction	253
16.1 Machine Translation	253

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

16.2 Storia della Machine Translation	255
16.3 Somiglianze e variazioni nei linguaggi	256
16.3.1 Morphology	256
16.3.2 Ordinamento della traduzione	258
16.3.3 Segmentation Variation	258
16.3.4 Inferential Load	258
16.3.5 Lexical Divergences	259
16.4 Approcci classici per la Machine Translation	260
17 Lesson 16 - Statistical Machine Translation	266
17.1 Faithfulness e Fluency	267
17.1.1 Formulazione	268
17.1.2 Fluency $P(E)$	269
17.1.3 Faithfulness $P(F E)$: Phrase-Based Translation Model	270
17.1.4 Osservazioni su Fluency e Faithfulness	273
17.1.5 Training dei parametri del Phrase Based translation model	274
17.2 Allineamento in Machine Translation	275
17.2.1 IBM Model 1	277
17.3 Training dei modelli di allineamento	280
17.3.1 Esempio di EM	283
17.4 Phrase Based Translation Model	285
17.4.1 Phrase Alignment	285
17.5 Decoding per il Phrase Based Statistical MT	288
17.6 Valutare un sistema di Machine Translation	294
17.6.1 Automatic Evaluation della Machine Translation .	296
17.7 Syntactic Models for MT	299
18 Lesson 17 - Neural Machine Translation	301
18.1 Sequence To Sequence Model	302
18.1.1 Eseguire il training del Sequence To Sequence Model	305
18.1.2 Beam Search Decoding	307

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

18.1.3 Pro e contro di NMT e di SMT	309
18.2 Attention	312
18.2.1 Funzionamento dell'Attention	313
18.2.2 Equazioni dell'attention	319
18.2.3 Considerazioni sull'Attention	320
18.2.4 Altri utilizzi dell'attention	321
18.3 Multilingual Translation	324
18.3.1 Zero-shot Translation	325
18.3.2 Visualizzare la traduzione	326
18.4 Conclusione	327
19 Transformer e Bert	328
19.1 Introduzione	328
19.2 Il transformer	329
19.2.1 Transformer Model	331
19.2.2 Word Embedding	334
19.3 Self Attention	336
19.3.1 Perchè usare la Self Attention?	343
19.3.2 Multi Head Attention	343
19.4 Positional Encoding	347
19.5 Residual Connections e Layer Normalization	349
19.6 Il decoder	350
19.6.1 Output del Decoder	352
19.7 Training	354
19.8 Conclusione - Machine Translation	356
19.9 Bert	357
19.9.1 Problemi con i modelli precedenti	357
19.9.2 Una possibile soluzione	358
19.9.3 Rappresentazione dell'input	360
19.9.4 Two Step Development	361
19.9.5 Architettura del Modello	366
19.9.6 Conclusioni su Bert	371

19.10 Evoluzioni di Bert	372
19.10.1 RoBERTa	372
19.10.2 XLNet	373
19.10.3 ALBERT	373
19.10.4 T5	374
19.11 Distillation	374
19.12 Conclusioni	375
20 Syntax Probe	377
20.1 Introduzione	377
20.2 Observational vs constructive evidence	382
20.2.1 Trovare una distance metric per il parse tree	386
20.2.2 Reconstructed trees and depth	387
20.3 Conclusioni	389
20.4 Bert Sentiment - Notebook	389
20.4.1 Introduzione	389
20.4.2 Training del modello	390
20.4.3 Valutare i risultati del modello	391
21 Pytorch	393
21.1 Introduzione	393
21.1.1 Autograd	394
21.1.2 Training Data	395
21.1.3 Regression	395
21.2 Tutorial	400
21.2.1 Fine Tuning	406
21.2.2 Clean Code	407
22 Question Answering	409
22.1 Introduzione	409
22.1.1 Open Domain QA	411
22.2 Trec QA (2001-2006)	412

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

22.3 Answer Selection	415
22.3.1 L'approccio utilizzato	416
22.4 Reading Comprehension	421
22.5 SQuAD	422
22.6 Conclusione	423
23 Reading Comprehension	424
23.1 Reading Comprehension	425
23.1.1 SQuAD 1.1	425
23.2 Stanford Attentive Reader	428
23.3 SQuAD 2.0	432
23.3.1 Limitazioni di SQuAD	435
23.4 Open Domain Question Answering	436
23.4.1 DrQA	437
23.4.2 Document Retriever	438
23.5 Stato dell'arte del QA nel 2020	438
23.5.1 La compezione	438
23.5.2 Il vincitore	441
24 Machine Reading	442
24.1 bAbl Dataset	445
24.2 Memory Networks	447
25 chatbot	459
25.1 Storia	460
25.2 Generation Based Models	463
25.3 Retrieval Based Models	464
25.4 Dialog Control	465
25.5 Challenges	466
25.6 Diversity	467
25.6.1 Coherent Personality	467
25.6.2 Speaker Embeddings	468

25.7 Evaluation	469
25.8 Advanced Features	469
25.8.1 Alexa Prize Socialbot	470
26 DialogFlow e Facebook Blender	476
26.1 Template Based with DialogFlow	476
26.1.1 Esempio di utilizzo di DialogFlow	480
26.2 Facebook BlenderBot	485
26.2.1 Skills	486
26.2.2 Conclusione	486
27 Trends and Future of NLP	487
27.1 Introduzione	487
27.1.1 Applicazioni	488
27.1.2 Attention	488
27.2 Deep NLP	489
27.3 Inference from Memory	490
27.4 Limiti di Deep NLP	492
27.5 Conclusione	492

Chapter 1

Lesson 1 - Language Modeling

1.1 Introduzione

Supponiamo di avere varie possibili frasi che sono differenti tra loro in termini di parole presenti al loro interno oppure in termini dell'ordine delle parole presenti al loro interno. La presenza di una parola o di un'altra o anche il diverso ordine delle parole può far sì che la frase assuma un senso completamente differente. Prese varie frasi vorremmo poter capire quale ha più senso in quel contesto, per farlo vorremmo calcolare la probabilità di ottenere quella specifica frase andando a considerare le parole presenti all'interno della frase. In alternativa potremmo anche calcolare la probabilità di una singola parola considerando le parole che la precedono nella frase. Il modello che assegna probabilità ad una sequenza di parole è chiamato Language Model LMs.

Il Language Model più semplice è l'N-Gram Model dove la N indica il numero di parole che stiamo considerando ogni volta, ad esempio il bigram model indica che stiamo considerando la probabilità di frasi formate da 2 parole.

L'N-gram model lo potremo utilizzare sia per stimare la prossima parola che abbiamo in un N-gram date le parole precedenti e lo possiamo anche usare per assegnare una probabilità all'intera sequenza. In generale si utilizza semplicemente il termine N-gram per indicare entrambi gli utilizzi che abbiamo appena elencato.

1.2 N-Gram

Consideriamo un primo caso, abbiamo una frase e vogliamo calcolare la probabilità di avere una certa parola come prossima parola della frase. Un metodo naïve sarebbe quello di andare a considerare un corpus abbastanza grande e contare quante volte compare la frase che abbiamo, poi nello stesso corpus contiamo anche quante volte quella frase compare seguita dalla parola che stiamo considerando. In questo modo otteniamo la probabilità di avere quella parola alla fine della frase considerando il numero di volte che la frase compare seguita dalla parola diviso il numero delle volte che compare quella frase senza considerare l'ultima parola:

$$P(\text{the} | \text{its water is so transparent that}) = \\ \frac{C(\text{its water is so transparent that the})}{C(\text{its water is so transparent that})}$$

Con un ragionamento simile se volessimo considerare la probabilità di ottenere una specifica frase dovremmo considerare la lunghezza della frase e calcolare tutte le possibili frasi che abbiamo di quella lunghezza. Una volta calcolato questo dovremmo vedere quante volte la frase che stiamo considerando compare in questo set che abbiamo generato. Poi potremo calcolare la probabilità della frase facendo la divisione come prima.

Chiaramente tutto questo non è fattibile e quindi possiamo considerare un metodo differente e più intelligente per questo calcolo della probabilità. Qualche notazione prima di tutto:

- Se scriviamo w_1^4 vuol dire che stiamo considerando le parole comprese tra 1 e 4
- La formula della chain rule che utilizzeremo nel calcolo di questa probabilità dipende dalla definizione di probabilità condizionale. In particolare abbiamo

$$P(B|A) = \frac{P(A \cap B)}{P(A)}$$

e lo possiamo riscrivere come

$$P(A \cap B) = P(A)P(B|A)$$

. Questa la possiamo estendere a più variabili contemporaneamente e quindi otteniamo ad esempio

$$P(x_1, x_2, \dots, x_n) = P(x_1)P(x_2|x_1)P(x_3|x_1, x_2), \dots, P(x_n|x_1, \dots, x_{n-1})$$

Quindi se per esempio vogliamo stimare la probabilità di avere una certa frase o la probabilità di avere una parola che segua una frase, possiamo utilizzare la probabilità delle singole parole utilizzando poi la chain rule. Ad esempio applicando la chain rule alle parole otteniamo:

$$\begin{aligned} P(w_1^n) &= P(w_1)P(w_2|w_1)P(w_3|w_1^2)\dots P(w_n|w_1^{n-1}) \\ &= \prod_{k=1}^n P(w_k|w_1^{k-1}) \end{aligned}$$

Il problema di questo sistema e della Chain Rule è che noi non sappiamo calcolare la probabilità di avere una parola dato tutto l'elenco di parole che la precedono e quindi si deve pensare ad un sistema differente in cui andiamo a considerare un numero minore di parole. Quello che si fa è approssimare il numero di parole che dobbiamo considerare per calcolare la probabilità di una parola. Quindi invece di considerare tutte le parole che precedono quella che abbiamo preso, consideriamo solamente poche parole precedenti, ad esempio nel bigram model consideriamo solamente quella precedente. Quindi ad esempio:

$$P(\text{the}|\text{Walden Pond's water is so transparent that})$$

diventa:

$$P(\text{the}|\text{that})$$

Quindi in pratica l'approssimazione che facciamo è la seguente:

$$P(w_n|w_1^{n-1}) \approx P(w_n|w_{n-1})$$

Markov Assumption

L'assunzione di Markov mi permette di dire che la probabilità di avere una certa parola dipende solamente dalla parola precedente. I Markov Models in particolare sono modelli probabilistici che mi permettono di fare delle previsioni senza guardare troppo nel passato. Questa idea la possiamo generalizzare con il bigram model (consideriamo solamente la parola precedente) e con l'n-gram model (consideriamo le n-1 parole precedenti). Quindi ora l'approssimazione della probabilità condizionale la possiamo scrivere come:

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

$$P(w_n|w_1^{n-1}) \approx P(w_n|w_{n-N+1}^{n-1})$$

Se poi consideriamo il caso del bigram possiamo dire che la probabilità di trovare una certa frase dipende dal prodotto di tutte le probabilità condizionali precedenti:

$$P(w_1^n) \approx \prod_{k=1}^n P(w_k|w_{k-1})$$

Come possiamo fare a fare una stima della probabilità (ad esempio) di ognuno dei bigram che stiamo considerando? Ad esempio per calcolare la probabilità di $P(y|x)$ ovvero che y segue x possiamo contare il numero di volte che nel nostro testo abbiamo xy e dividerlo per il numero delle volte che nel testo compare x . Questa divisione serve per normalizzare e fare in modo che la probabilità che stiamo cercando sia compresa tra 0 e 1.

Quando utilizziamo la MLE otteniamo dei parametri che massimizzano la likelihood del training set T dato il modello M. Questa MLE non sempre ci restituisce dei risultati che sono interessanti, pensiamo al caso in cui abbiamo un corpus con 1 milione di parole e la parola cinese compare 400 volte, ora con l'MLE possiamo stimare la presenza di questa parola dividendo il numero di occorrenze per il numero totale di parole. Alla fine otteniamo una probabilità che è molto bassa (0.004). Ora se noi prendiamo un altro testo e vogliamo la probabilità di ottenere la parola cinese se prendiamo una parola a caso dal testo, possiamo utilizzare nuovamente la probabilità che è stata calcolata al passo precedente, quindi diciamo che la probabilità è 0.004. Ovviamente questa potrebbe essere una pessima stima perchè noi qua abbiamo due differenti testi e quindi dovremmo considerare di cosa parla il secondo prima di dire che la probabilità di ottenere una parola è la stessa. Quello che sappiamo

di sicuro è che la probabilità che noi assegnamo (0.004) è sicuramente la migliore stima possibile al momento.

Consideriamo un altro esempio, abbiamo un altro corpus che è il Berkeley Restaurant Project ovvero un corpus con 9332 parole. Quello che noi possiamo fare è andare a considerare 8 parole presenti nel corpus e poi contare il numero di occorrenze in un bigram:

	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

All'interno della matrice i valori che sono 0 sono quelli in cui il bigram non compare mai. Data la matrice presente sopra quello che possiamo fare poi è trasformarla in una matrice di probabilità, per farlo dobbiamo normalizzare e quindi dividere ognuno dei conteggi che abbiamo ottenuto fino ad ora per il conteggio delle singole parole nel corpus. Per esempio "Chinese food" lo troviamo 82 volte e chinese da solo 158 volte, questo vuol dire che per normalizzare calcoliamo $82/158 = 0.52$.

	i	want	to	eat	chinese	food	lunch	spend
i	0.002	0.33	0	0.0036	0	0	0	0.00079
want	0.0022	0	0.66	0.0011	0.0065	0.0065	0.0054	0.0011
to	0.00083	0	0.0017	0.28	0.00083	0	0.0025	0.087
eat	0	0	0.0027	0	0.021	0.0027	0.056	0
chinese	0.0063	0	0	0	0	0.52	0.0063	0
food	0.014	0	0.014	0	0.00092	0.0037	0	0
lunch	0.0059	0	0	0	0	0.0029	0	0
spend	0.0036	0	0.0036	0	0	0	0	0

Questo metodo non è tanto usato con i bigram quanto invece con trigram o 4-gram, ogni volta che aumentiamo la N aumenta anche il

numero di parole precedenti che dobbiamo considerare e questo comporta che dovremo assumere altre informazioni aggiuntive per calcolare la probabilità.

Un problema che potremmo avere riguarda l'underflow, questo infatti è dovuto al fatto che noi abbiamo da moltiplicare varie probabilità che sono tutte minori di 1 e quindi a mano a mano che moltiplichiamo valori piccoli otteniamo valori sempre più piccoli che si avvicinano a 0. Quindi rischiamo di andare in underflow e il problema lo possiamo risolvere se utilizziamo il logaritmo delle probabilità. Quindi la nostra moltiplicazione di probabilità diventa il logaritmo di una moltiplicazione che quindi possiamo scrivere come somma di logarimi delle varie probabilità. In questo modo riusciamo ad evitare l'underflow:

$$\log(p_1 \cdot p_2 \cdot p_3 \cdot p_4) = \log(p_1) + \log(p_2) + \log(p_3) + \log(p_4)$$

Un problema che si può presentare quando utilizziamo l'n-gram model è che rischiamo di andare in overfitting perché il modello funziona bene solamente quando lo utilizziamo su dei testi che sono simili a quelli con cui abbiamo fatto il training.

In pratica nella vita reale questo non accade questo utilizzare un modello probabilistico di questo genere non è possibile.

In generale dobbiamo eseguire il training di un language model su un corpus di grandi dimensioni per stimare dei parametri validi da poter essere utilizzati. Il modello che creiamo poi lo possiamo andare a valutare se consideriamo la capacità di predire su un set di dati che è differente rispetto a quello di training.

1.3 Shannon's Game - Generate new Text

Vorremmo utilizzare le probabilità che abbiamo generato fino a questo punto per generare del nuovo testo e non solo per fare delle previsioni

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

sulle possibili parole o su una frase. In particolare possiamo adottare il seguente algoritmo per generare del nuovo testo basandoci sulla probabilità:

- Scegliamo un bigram iniziale $\langle s \rangle, w$, prendiamo quello che ha la probabilità più alta
- Scegli un altro bigram (w, x) sempre basandoci sulla probabilità
- Ripeti il secondo step fino a quando non generi la fine della stringa ovvero $\langle /s \rangle$
- Unire insieme tutte le stringhe generate

Un esempio a cui possiamo pensare se consideriamo la generazione di nuovo testo è il caso dei testi di Shakespeare. In particolare quello che si vede in questa situazione è che preso il corpus che rappresenta i testi di Shakespeare e prodotti i vari bigram (o trigram o 4-gram) noi riusciamo a generare dei testi che più o meno somigliano a quelli di Shakespeare perché per la generazione ci basiamo sulla probabilità che le frasi che generiamo siano presenti nell'opera originale.

1.4 Smoothing

Precedentemente abbiamo creato la matrice relativa ai bigram in cui abbiamo inserito un conteggio relativo alla presenza dei vari bigram all'interno del testo.

La matrice però era abbastanza sparsa perché molti dei bigram non erano visibili nel testo originale e quindi assegnavamo probabilità 0 a quel bigram. Questo comporta che se troviamo il bigram nel momento in cui applichiamo il modello a dati che non abbiamo mai visto otteniamo una probabilità per quel bigram pari a 0 e quindi si azzerà anche la probabilità di avere quella frase.

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

Per risolvere questo problema è stato pensato un sistema di regolarizzazione che consiste nell'andare a non assegnare il valore 0 alle probabilità. Questo prende il nome di smoothing, le probabilità infatti vengono modificate e togliamo del valore dalle probabilità che sono maggiori di 0 per darle alle probabilità che sono 0. È necessario fare questo spostamento dei valori in modo da mantenere una distribuzione dei dati che integrata produca una somma pari a 1.

Il metodo più semplice per eseguire lo smoothing è il Laplace Smoothing che consiste nell'aggiungere 1 a tutti i conteggi dei vari bigram prima di eseguire la normalizzazione. Nella formula oltre ad aggiungere 1 al numeratore aggiungiamo anche V al denominatore e questo mi indica il numero di osservazioni extra che svolgiamo. Si tratta di un metodo che non funziona perfettamente se vogliamo utilizzarlo in modelli n-gram recenti ma comunque è utile per capire il funzionamento del sistema.

$$P_{\text{Laplace}}(w_i) = \frac{c_i + 1}{N + V}$$

Una cosa che possiamo fare e che potrebbe tornare piuttosto utile è utilizzare l'adjusted count c^* (chiamato anche Reconstructed count), questo è più semplice da confrontare con l'MLE e possiamo trasformarlo in una probabilità semplicemente normalizzando con N. Questo reconstructed o adjusted count può essere calcolato in questo modo:

$$c_i^* = (c_i + 1) \frac{N}{N + V}$$

e comporta anche grandi cambiamenti in termini di conteggi di bigram.

Il Laplace Smoothing, anche se non viene usato per gli n-gram model viene comunque utilizzato per regolarizzare altri modelli di NLP, specialmente in situazioni dove il numero degli 0 non è eccessivo.

Una alternativa al Laplace Smoothing consiste nell'aggiungere una K dove la K è un numero frazionario quindi minore di 1, ad esempio $k = 0.01$.

Una soluzione ancora migliore invece consiste nell'utilizzare la Bayesian Prior Smoothing in cui la formula diventa la seguente:

$$P_{Prior}(w_i | w_{i-1}) = \frac{C(w_{i-1}, w_i) + P(w_i)}{C(w_{i-1}) + 1}$$

Qui non andiamo a sommare 1 come abbiamo fatto in precedenza ma sommiamo la probabilità della parola w_i .

1.5 Zipf's Law

La zipf's Law mi dice che, dato un set di eventi, ci sono un numero piccolo di eventi che avvengono con una frequenza molto alta e ci sono un numero alto di eventi che avvengono con una frequenza bassa, gli eventi che accadono spesso li possiamo studiare più facilmente e sarà anche più facile capirne le statistiche, al contrario quelli che accadono di rado sono più difficili da comprendere. In particolare quando consideriamo la Zipf's Law avremo una serie di parole e possiamo calcolare la frequenza di ognuna di esse e poi ad ognuna di queste parole assegnamo un rank che dipende da quanto quella parola è rara. Il coefficiente che viene restituito nella Zipf's Law è proporzionale a $f * r$ dove la f è la frequenza delle parole mentre la r indica il rank. I problemi che possono sorgere sono simili a

quelli che abbiamo con la classica probabilità, in particolare anche qua rischiamo di andare in underfitting perchè moltiplichiamo valori che sono minori di 0 e quindi per risolvere il problema utilizziamo il logaritmo.

Questo comporta che la nostra stima sarà sparsa ovvero avremo una serie di 0 all'interno della matrice in cui contiamo le occorrenze dei bigram. Il problema in questo caso è che noi non sappiamo quando questi 0 sono effettivamente degli 0 o quando sono dovuti ad eventi che hanno semplicemente una bassa frequenza.

Quello che possiamo fare per differenziare questi tipi di eventi è andare a stimare la likelihood di questi eventi che non abbiamo visto e che quindi contribuiscono con uno 0 nella matrice.

Per quanto riguarda gli N-gram, ci sono dei progetti e delle librerie che possono essere utilizzate e in particolare Google ha rilasciato anche un enorme dataset di N-gram.

1.6 Valutare un language Model

Per valutare un language model abbiamo due possibilità, il primo tipo di valutazione viene chiamato extrinsic evaluation ed è una metrica che ci permette di provare il nostro modello in una condizione "reale". In questo caso noi inseriamo il nostro modello in una applicazione e misuriamo quanto il nostro modello funziona bene sui dati che abbiamo a disposizione. Questa tipologia di test ci restituisce delle stime accurate ma ha un costo troppo alto e quindi non può essere utilizzato.

L'alternativa viene chiamata intrinsic evaluation e in questo caso siamo indipendenti rispetto all'applicazione finale del nostro modello. In particolare quello che facciamo in questo caso è avere un set di dati che suddividiamo in training set, development set e test set (dividiamo dando l'80% dei dati al training set, poi 10% agli altri due set). Quello che facciamo quindi è eseguire il training del nostro modello sul training set e poi facciamo il tuning dei nostri iperparametri sul development set.

Alla fine il test del modello viene eseguito sul test set. Quindi se per esempio abbiamo due modelli che differiscono per i parametri utilizzati consideriamo il risultato del test sul test set e poi scegliamo di utilizzare il modello che mi fornisce un risultato migliore.

È importante non utilizzare i dati del test set all'interno del training set perché altrimenti quando testiamo il nostro modello sul test set andremo a fare un test su dati che abbiamo già visto e questo causerà dei risultati migliori di quanto dovrebbero essere che quindi non saranno accurati.

Perplexity

Quando dobbiamo considerare più di un modello e dobbiamo valutarne le performance quello che si fa in pratica non è utilizzare direttamente la probabilità che viene restituita ma si utilizza una metrica differente che viene chiamata perplexity.

Intuitivamente possiamo definire la Perplexity utilizzando l'idea di sorpresa ovvero quanto è sorpreso il language model quando vede il test set. Più il modello è sorpreso e più la probabilità di quella frase è bassa, al contrario meno il modello è sorpreso e più aumenta la probabilità. In generale possiamo dire che la perplexity mi permette di capire quanto bene il nostro modello è in grado di fissare i dati del test set.

Per il calcolo della Perplexity prendiamo il test set $W = w_1w_2\dots w_n$ e ne calcoliamo la probabilità, poi calcoliamo l'inverso di questa probabilità e la normalizziamo considerando il numero totale di parole che abbiamo nel corpus.

$$\begin{aligned} \text{PP}(W) &= P(w_1w_2\dots w_N)^{-\frac{1}{N}} \\ &= \sqrt[N]{\frac{1}{P(w_1w_2\dots w_N)}} \end{aligned}$$

Partendo dalla formula precedente poi possiamo applicare la Chain Rule e ottenere la seguente formula:

$$\text{PP}(W) = \sqrt[N]{\prod_{i=1}^N \frac{1}{P(w_i|w_1 \dots w_{i-1})}}$$

che diventa ancora più semplice nel caso in cui scegliamo di utilizzare i bigram.

$$\text{PP}(W) = \sqrt[N]{\prod_{i=1}^N \frac{1}{P(w_i|w_{i-1})}}$$

Quando utilizziamo la perplexity come metro di paragone tra vari language model il nostro obiettivo è trovare il modello che mi restituisce la perplexity più bassa ovvero il modello che mi restituisce la probabilità più alta su dati che non ho ancora visto.

Un esempio, prendiamo un corpus con 10 possibili numeri, la probabilità di ottenere ognuno di questi numeri è 1/10. Quello che facciamo per calcolare la perplexity di una sequenza lunga N e composta da questi numeri è moltiplicare la probabilità dei vari numeri e poi invertirla e normalizzarla usando N, in questo caso otteniamo una perplexity che è uguale a 10:

$$\begin{aligned}
 \text{PP}(W) &= P(w_1 w_2 \dots w_N)^{-\frac{1}{N}} \\
 &= \left(\frac{1}{10}\right)^{-\frac{1}{N}} \\
 &= \frac{1}{10}^{-1} \\
 &= 10
 \end{aligned}$$

Quello che possiamo dire riguardo alla perplexity è che più il nostro modello è "informato" e più abbiamo una bassa perplexity, in particolare consideriamo il training del modello su 38 milioni di parole e poi facciamo la valutazione su 1.5 milioni di parole otteniamo questi valori per la perplexity che quindi dipende moltissimo dal tipo di N che scegliamo nel nostro N-gram:

	Unigram	Bigram	Trigram
Perplexity	962	170	109

Notare inoltre che più aggiungiamo conoscenza riguardo al test set più la perplexity diventa artificialmente minore rispetto ad un altro modello che è meno informato e questo può essere un problema perché poi in realtà la probabilità di quel modello con quel test set potrebbe non essere effettivamente minore. Quindi è importante considerare che due language model possono essere confrontati sullo stesso test set solamente se utilizzano lo stesso vocabolario. Notare anche che un miglioramento della perplexity, che come detto è una intrinsic evaluation non sempre corrisponde ad un miglioramento in termini della extrinsic evaluation.

Unknown Words

Una cosa che dobbiamo tenere in considerazione sono le parole che potremmo trovare nel test set e che non abbiamo mai visto in precedenza nel training set. Queste parole non viste in precedenza sono chiamate unknown oppure out of vocabulary. Possiamo trovarci in due differenti situazioni:

- Abbiamo un vocabolario fissato iniziale e quando consideriamo il training set sostituiamo le parole che non sono presenti nel vocabolario con la parola $<UNK>$. Poi la probabilità di ottenere $<UNK>$ viene stimata come se fosse una qualsiasi altra parola che troviamo normalmente nel training set.
- Non abbiamo un vocabolario iniziale fissato ma consideriamo il nostro training set e da quello creiamo il nostro vocabolario. Quello che facciamo è contare il numero di occorrenze delle parole e poi sostituiamo quelle parole che occorrono meno di n volte con la parola UNK e poi le trattiamo come se fossero parole normali quando facciamo il training del modello.

Per evitare che il nostro language model assegna probabilità 0 a tutti questi eventi che sono poco probabili possiamo utilizzare delle tecniche di smoothing che tolgano un po' di probabilità agli eventi più frequenti per "donarle" agli eventi meno frequenti.

Smoothing: Backoff and Interpolation

Abbiamo già visto alcuni metodi che possiamo utilizzare per lo smoothing, ora vogliamo considerare altri due metodi che sono il Backoff e l'Interpolation.

Quando dobbiamo calcolare la probabilità di un certo trigram $P(w_n|w_{n-1}w_{n-2})$ ma quella probabilità è 0 perchè non abbiamo mai visto quel trigram allora possiamo cercare di calcolare comunque la probabilità approssimandola ovvero andando a calcolare la probabilità del bigram $P(w_n|w_{n-1})$

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

e se neanche questa ha un valore maggiore di 0 possiamo approssimarla ancora calcolando la probabilità dell'unigram $P(w_n)$. Questo sistema di approssimazione si chiama backoff e in pratica consiste nel cercare la probabilità dell'n-gram di ordine inferiore quando non riusciamo a calcolare la probabilità dell'n-gram iniziale. In pratica andiamo avanti ricorsivamente fino a che non troviamo una probabilità che abbia valore maggiore di 0.

Nel caso del Backoff dobbiamo cercare di diminuire la probabilità dei vari n-gram di ordine alto per poi assegnarla a quelli che hanno un ordine più basso, un po' come facevamo con l'add one smoothing. Questo tipo di Backoff con il discounting viene chiamato Katz Backoff, qua noi ci affidiamo ad una probabilità "discounted" P^* nel caso in cui abbiamo visto l'n-gram prima d'ora altrimenti andiamo ricorsivamente a calcolare la Katz probability per l'n-gram di lunghezza minore. Questo lo possiamo scrivere in formula in questo modo:

$$P_{katz}(w_n | w_{n-N+1}^{n-1}) = \begin{cases} P^*(w_n | w_{n-N+1}^{n-1}) & \text{if } C(w_{n-N+1}^n) > 1 \\ \alpha(w_{n-N+1}^{n-1}) P_{katz}(w_n | w_{n-N+2}^{n-1}) & \text{otherwise} \end{cases}$$

Solitamente il calcolo del Katz Backoff viene eseguito insieme al Good-Turing backoff.

Cosa facciamo quando troviamo una parola che non abbiamo mai visto in precedenza? Utilizziamo il metodo dell'OOV che abbiamo definito in precedenza, quindi usiamo $\langle UNK \rangle$.

Al contrario invece l'interpolation va a considerare più probabilità alla volta, in particolare se abbiamo un trigram con una certa probabilità, nell'interpolation consideriamo anche il bigram e l'unigram e uniamo tutte queste probabilità in un'unica probabilità finale. Per calcolare questa probabilità finale utilizziamo la seguente formula:

$$\begin{aligned}\hat{P}(w_n | w_{n-2} w_{n-1}) &= \lambda_1 P(w_n | w_{n-2} w_{n-1}) \\ &\quad + \lambda_2 P(w_n | w_{n-1}) \\ &\quad + \lambda_3 P(w_n)\end{aligned}$$

e all'interno della formula abbiamo delle λ che fanno da pesi per la nostra formula. Queste λ sono dei parametri e c'è una regola che mi dice che devono sommarsi ad 1. In particolare per eseguire il training del parametro λ dobbiamo considerare un set di dati che chiamiamo Held-Out Data. Una volta che abbiamo eseguito il training del nostro modello sui dati di training andiamo poi a provare il modello sui dati Held-Out ed eseguiamo il tuning dei parametri λ scegliendo poi quelli che ci danno la probabilità più alta sul set Held-Out.

Problemi per N-gram

L'N-gram model di cui abbiamo parlato fino ad ora è in grado di comprendere solamente le dipendenze che derivano dalle parole vicine a quella che stiamo prendendo in considerazione. Ad esempio se stiamo usando un trigram noi andiamo a considerare solamente la dipendenza tra la parola in posizione N e le due parole precedenti $N - 1$ e $N - 2$.

Questo ovviamente non ci permette di cogliere delle dipendenze che sono più lontane all'interno del testo e questa è una delle principali limitazioni dei modelli N-gram. Per risolvere questo tipo di problemi ci sono altri modelli che possiamo utilizzare e che sono in grado di gestire dipendenze anche ad una più lunga distanza.

Chapter 2

Lesson 2 - Representation of Words

Quando abbiamo una certa frase possiamo dire che il significato delle parole che sono presenti all'interno della frase dipende dal contesto. Una certa parola infatti potrebbe avere un significato differente a seconda del contesto in cui viene posta.

In generale possiamo dire che se prendiamo due frasi, due parole che compaiono in due contesti che sono simili tra loro avranno anche un significato simile. Questo legame tra il modo in cui le parole sono distribuite e la somiglianza nel loro significato viene chiamato distributional hypothesis. L'idea è stata proposta quando ci si è accorti che le parole che sono sinonimi tendono a trovarsi in contesti che sono simili.

Una possibile soluzione al problema del significato delle parole consiste nell'utilizzare un dizionario in cui indichiamo per ogni parola il suo significato. Il problema di questo metodo è che una parola può avere un significato diverso in base alla frase dove si trova e soprattutto dovremmo stare di continuo ad aggiungere nuove parole oppure a aggiungere nuovi significati a quelle già esistenti. Una soluzione di questo genere è impossibile da mantenere.

2.1 Vector Space Model

Un metodo per lavorare con questi distributional models consiste nel creare una co-occurrence matrix dei testi che abbiamo a disposizione e delle parole presenti all'interno del testo. Questa soluzione è stata utilizzata in Information retrieval e viene chiamata Vector Space Model (VSM).

In alcuni casi, quando non siamo interessanti al numero di volte che una parola compare in un testo ma solamente al fatto che la parola compare nel testo, inseriamo all'interno della matrice solamente 1 se la parola è presente o 0 se non è presente.

Se abbiamo la nostra matrice e vogliamo trovare la similitudine tra due documenti o tra due parole possiamo utilizzare come metrica la cosine distance che mi va a misurare l'angolo tra i due vettori. La cosine distance consiste nel calcolo del prodotto vettoriale tra i due vettori. Il problema di questa metrica è che il prodotto scalare sarà maggiore se il vettore ha una lunghezza maggiore dove la lunghezza del vettore la misuriamo in questo modo:

$$|\mathbf{v}| = \sqrt{\sum_{i=1}^N v_i^2}$$

Un modo semplice per risolvere questo problema consiste nell'andare a normalizzare il risultato che otteniamo dividendolo per il prodotto della lunghezza dei due vettori.

Tf-Idf

Quando consideriamo la frequency matrix e cerchiamo di capire le somiglianze tra i vari vettori basandoci solamente su quella capiamo subito che con-

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

siderare solamente la frequenza non basta per capire quanto le parole sono veramente simili tra loro perchè la frequenza potrebbe essere una cosa non troppo discriminativa.

Noi vorremmo dare un peso maggiore alle parole che compaiono spesso insieme rispetto a quelle che compaiono solamente poche volte ma allo stesso tempo vorremmo anche rendere poco importanti le parole che sono troppo frequenti, ad esempio gli articoli, che non sono discriminativi.

L'algoritmo che viene utilizzato per calcolare queste somiglianze cercando di bilanciare tutte le richieste è chiamato tf-idf. Il tf-idf è il prodotto di due termini e ognuno dei due cattura le due richieste precedenti:

- Il termine tf indica la frequenza dei termini all'interno dei vari documenti. Lo possiamo calcolare come

$$tf_{t,d} = count(t, d)$$

- Il secondo termine lo usiamo per dare un peso maggiore alle parole che compaiono solamente in pochi documenti perchè questi mi permettono di discriminare meglio. Questo secondo termine è il df_t e mi conta il numero di documenti d in cui il termine t compare. Questo è utile perchè potremmo avere due termini che compaiono lo stesso numero di volte all'interno dei documenti che abbiamo però magari il primo termine compare solamente in un singolo documento mentre il secondo termine compare in tanti documenti differenti. Questo vuol dire che il primo termine avrà una importanza particolare per il documento in cui compare e se il nostro obiettivo è trovare il documento in cui si parla di quello specifico argomento, quel documento dovrebbe avere una importanza maggiore. Questo è utile perchè potremmo avere due termini che compaiono lo stesso numero di volte all'interno dei documenti che abbiamo però magari il primo termine compare solamente in un singolo documento

mentre il secondo termine compare in tanti documenti differenti. Questo vuol dire che il primo termine avrà una importanza particolare per il documento in cui compare e se il nostro obiettivo è trovare il documento in cui si parla di quello specifico argomento, quel documento dovrebbe avere una importanza maggiore.

Alla fine la formula del tfidf è la seguente:

$$w_{t,d} = tf_{t,d} * idf$$

dove in particolare $idf = \log_{10}(\frac{N}{df_t})$ e N indica il numero totale di documenti che abbiamo a disposizione.

Il tfidf può anche essere utilizzato per rappresentare documenti, questo è equivalente a usare un vettore per ognuna delle occorrenze del termine t , ovvero avremmo un vettore del tipo $v(t) = [0, idf_t, \dots, 0]$, il vettore di documenti in particolare diventa la somma dei vettori delle parole che compaiono all'interno del documento.

$$\begin{aligned} d &= \text{'you played a game'} \\ v(d) &= v(\text{'you'}) + v(\text{'played'}) + v(\text{'a'}) + v(\text{'game'}) \\ &= [0, tf_{\text{you},d} * idf_{\text{you}}, \dots, tf_{\text{played},d} * idf_{\text{played}}, \dots, tf_{\text{game},d} * idf_{\text{game}}, \dots, 0] \end{aligned}$$

Il risultato che otteniamo è una matrice che sarà abbastanza sparsa e in cui non riusciamo nemmeno a catturare le somiglianze (semantiche) tra i vari termini. Ad esempio qua abbiamo una somiglianza tra match e game però questa rappresentazione in cui utilizziamo il tfidf per assegnare i pesi all'interno della matrice non mi permette di identificare la somiglianza tra le due parole.

$d_1 = \text{'you played a game'}$
 $d_2 = \text{'you played a match'}$
 $d_3 = \text{'you played a trumpet'}$

$$v(d_1) = [0, w_{\text{played},d1}, w_{\text{game},d1}, 0, \dots, 0, 0, \dots, 0]$$

$$v(d_2) = [0, w_{\text{played},d2}, 0, \dots, w_{\text{match},d2}, 0, \dots, 0, 0, \dots, 0]$$

$$v(d_3) = [0, w_{\text{played},d3}, 0, \dots, 0, \dots, 0, w_{\text{trumpet},d3}, 0]$$

Il problema di questa rappresentazione in cui utilizziamo il tfidf per rappresentare i documenti è che non riusciamo ad avere una nozione di somiglianza ovvero due parole come ad esempio macchina e auto potrebbero non avere niente in comune. Questo è un problema che potremmo risolvere se riuscissimo a codificare la somiglianza delle parole direttamente all'interno del vettore.

2.2 Modeling Word Similarity

Come facciamo a modellare la similitudine di due parole in termini di significato? Vorremmo modellare il significato delle parole "embeddandolo" in un vector space dove i significati delle parole sono numeri. Questi vettori sono chiamati embedding.

Se fino ad ora avevamo rappresentato le parole utilizzando vettori lunghi e sparsi ora vogliamo passare ad una rappresentazione che invece deve essere densa perchè vogliamo evitare gli 0 e questo comporta anche che i vettori saranno più piccoli. Nei vettori sparsi infatti abbiamo il problema che i sinonimi non sono riconosciuti come tali e quindi non riusciamo a modellare la somiglianza delle parole. Questo fatto di avere vettori più piccoli e densi permette di utilizzarli come feature in un sistema di machine learning. Inoltre dato che in questo modo abbiamo meno parametri, riusciamo ad avere una generalizzazione migliore e riusciamo ad evitare l'overfitting.

Ci sono vari metodi per ottenere queste rappresentazioni dense del vettore:

- Abbiamo l'SVD.
- Abbiamo modelli ispirati dalle Neural Network come ad esempio word2vec, Glove e fastText.
- Abbiamo i Brown Cluster.

L'idea che permette di utilizzare questo sistema è quella delle Distributional Hypothesis: "Il significato di una parola è dato dalle parole che appaiono frequentemente vicino a quella parola". Si tratta di un'idea abbastanza vecchia che però è tornata di moda nel mondo dell'NLP. In pratica utilizziamo il contesto di una parola per capire il suo significato in quel contesto. Per rappresentare questa idea possiamo utilizzare un word-context Matrix ovvero per ognuna delle parole che abbiamo consideriamo il contesto e contiamo le frequenze della parola in ognuno dei contesti.

		Context words						
		...	cook	eat	...	changed	broke	...
Word s	cake	...	10	20	...	0	0	...
	steak	...	12	22	...	0	0	...
	engine	...	0	0	...	3	10	...
	tire	...	0	0	...	10	1	...

Ad esempio nella figura sopra possiamo vedere che abbiamo la parola eat che compare 20 volte in un contesto insieme alla parola cake.

2.3 Word Embedding

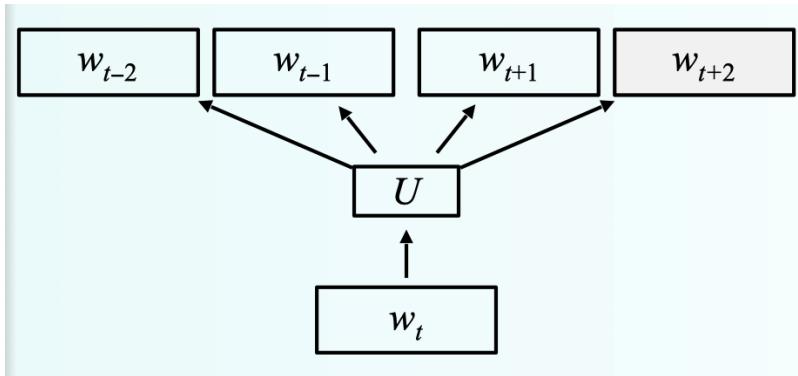
L'idea del Word Embedding è quella di produrre un vettore che sia denso e quindi più piccolo di quello che abbiamo all'inizio. In particolare quello

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

che succede è che noi abbiamo un vettore $v(t)$ che mi indica la presenza di una certa parola all'interno dei vari testi e poi quello che vogliamo fare è che tramite una funzione vogliamo diminuirne la dimensione. Quello che vorremmo è che il vettore risultato sia in grado di mantenere le informazioni più importanti del vettore di partenza e vorremmo anche che le parole simili tra loro siano vicine tra loro anche all'interno del vettore denso.

2.3.1 Word2Vec

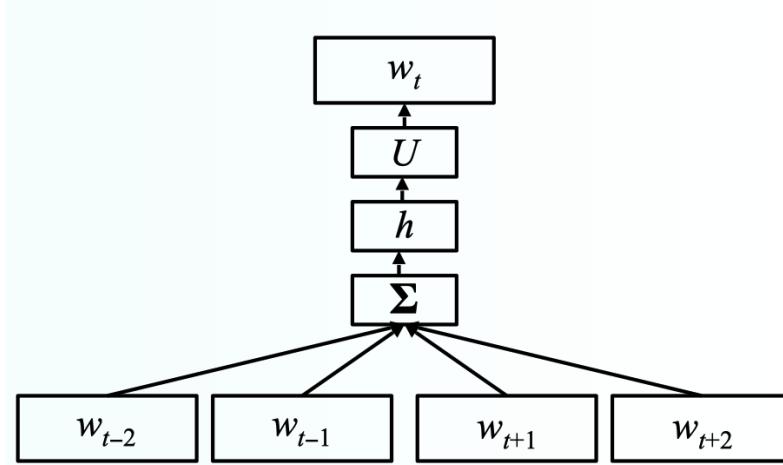
Word2Vec è un framework che permette la creazione di word vectors. L'intuizione di questo sistema è che invece di contare quante volte una parola compare insieme ad un'altra possiamo eseguire il training di un classificatore con un output che sarà binario e che sarà una previsione. Ci chiediamo in particolare se la parola w è solita comparire vicino ad un'altra parola. Non ci interessa il risultato della previsione ma ci interessano solamente i pesi che vengono utilizzati durante questo task. Questi pesi sono quelli che utilizzeremo come embedding. L'idea rivoluzionaria è che qua noi utilizziamo il testo per eseguire un task di training supervisionato di un classificatore, non c'è bisogno dell'intervento di qualcuno che esegua la labellizzazione del testo. Durante questo processo di previsione utilizziamo la logistic regression e non una neural network e questo semplifica di molto l'algoritmo rispetto a sistemi simili che però usano le neural network. L'algoritmo funziona in questo modo e possiamo utilizzare due architetture differenti per il modello, la prima che consideriamo è lo skip-gram:



- Prendiamo il nostro testo e per ogni parola trattiamo la coppia *parola, contesto* come un esempio positivo.
- Facciamo il random sampling di una parola differente e la consideriamo come un esempio negativo.
- Utilizziamo la logistic regression per eseguire il training di un classificatore che discrimini i due casi.
- Utilizziamo i pesi che abbiamo usato per la logistic regression come embedding.

Con lo skip-gram noi ci chiediamo, data la parola, se è possibile che quel contesto possa comparire con quella parola.

Nell'architettura CBoW invece facciamo il contrario, dato il contesto ci chiediamo se una certa parola può comparire all'interno di quel contesto o meno.



Solitamente la dimensione del contesto è di 5 parole prima e 5 parole dopo quella che stiamo considerando come centro, se scegliamo di avere un contesto più grande allora potremo catturare più semantica ma meno sintassi.

Consideriamo il classificatore che vorremmo utilizzare, come abbiamo detto dobbiamo abbiammo un certo contesto e una parola e vogliamo capire se quella parola fa parte del contesto oppure no. Quindi abbiamo una probabilità che è positiva e poi abbiamo la probabilità negativa.

Per calcolare la probabilità l'idea del modello Skip-Gram è quella di utilizzare la similarità, in particolare una parola è probabile che compaia vicino ad un'altra se i due embedding sono simili. Come possiamo fare a calcolare la similarity di due embedding? Si utilizza la cosine similarity, i due vettori sono simili se il loro prodotto è alto. Questo prodotto ovviamente non è un qualcosa che è compreso tra 0 e 1, quindi per trasformare il confronto in una probabilità si deve utilizzare la sigmoid function.

Nelle due formule qua sotto utilizziamo la sigmoid function e calcoliamo la cosine similarity tra una parola target t e una parola del contesto c in cui la parola target si potrebbe trovare.

$$P(+|t, c) = \frac{1}{1 + e^{-t \cdot c}}$$

$$\begin{aligned} P(-|t, c) &= 1 - P(+|t, c) \\ &= \frac{e^{-t \cdot c}}{1 + e^{-t \cdot c}} \end{aligned}$$

Qua sotto invece calcoliamo il prodotto delle varie probabilità calcolate con le due formule precedenti, in particolare dato che il contesto è formato da varie parole avremo varie probabilità calcolate con la sigmoid e la cosine similarity e le moltiplichiamo tra di loro in modo da ottenere la probabilità complessiva che quella parola target compare in quel contesto. Gli skip gram fanno una assunzione forte ma che risulta molto utile, infatti si considerano indipendenti tutte le parole che fanno parte del contesto e questo ci permette di moltiplicare le varie probabilità:

$$\begin{aligned} P(+|t, c_{1:k}) &= \prod_{i=1}^k \frac{1}{1 + e^{-t \cdot c_i}} \\ \log P(+|t, c_{1:k}) &= \sum_{i=1}^k \log \frac{1}{1 + e^{-t \cdot c_i}} \end{aligned}$$

Come funziona l'algoritmo di Learning

L'algoritmo di learning parte con un set iniziale di embedding, poi vengono analizzate le varie parole che fanno parte del testo e si va a modificare gli embedding in modo che l'embedding della parola che stiamo considerando sia il più simile possibile rispetto agli embedding delle parole che gli sono vicine.

Durante la fase di Learning abbiamo bisogno anche di esempi negativi, nel caso degli skip-gram abbiamo più esempi negativi che esempi positivi. Questi esempi negativi sono esempi in cui consideriamo delle parole che non sono corrette in quel contesto.

L'obiettivo dell'algoritmo di learning poi è il seguente:

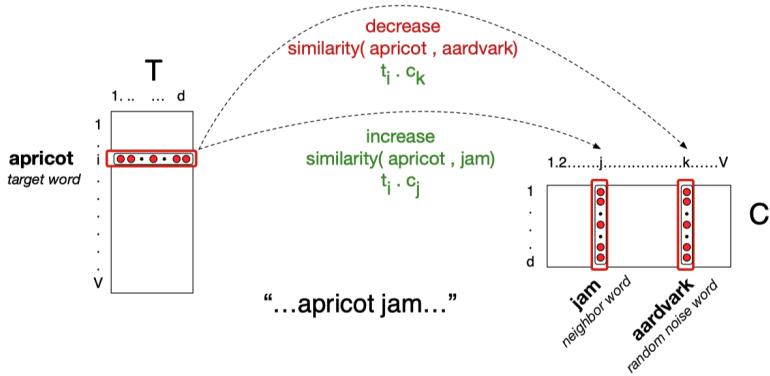
- Massimizzare la similarità della parola che stiamo considerando rispetto al contesto nel caso dell'esempio positivo.
- Minimizzare la similarità della parola che stiamo considerando rispetto al contesto nel caso dell'esempio negativo.

Questa formula la possiamo esprimere nel modo seguente:

$$L(\theta) = \sum_{(t,c) \in +} \log P(+|t, c) + \sum_{(t,c) \in -} \log P(-|t, c)$$

$$\begin{aligned}
 L(\theta) &= \log P(+|t, c) + \sum_{i=1}^k \log P(-|t, n_i) \\
 &= \log \sigma(c \cdot t) + \sum_{i=1}^k \log \sigma(-n_i \cdot t) \\
 &= \log \frac{1}{1+e^{-c \cdot t}} + \sum_{i=1}^k \log \frac{1}{1+e^{n_i \cdot t}}
 \end{aligned}$$

Possiamo eseguire il gradient descent per eseguire il training di questa funzione in modo da modificare iterativamente i parametri ovvero gli embedding e ottenere una massimizzazione della funzione. Lo skip-gram model esegue il learning di due embedding per ogni parola, uno è l'embedding della parola stessa e uno è l'embedding del contesto, questi sono memorizzati in una matrice. Nella matrice abbiamo che in ogni riga abbiamo l'embedding della parola nel vocabolario mentre in ogni colonna abbiamo l'embedding della colonna c_j rispetto alla parola j all'interno del vocabolario V.



In questo algoritmo noi partiamo con una matrice W di embedding delle parole e con una matrice C di embedding dei contesti che sono ran-

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

dom poi utilizziamo il training corpus e il gradient descent per modificare queste due matrici e massimizzare la funzione obiettivo:

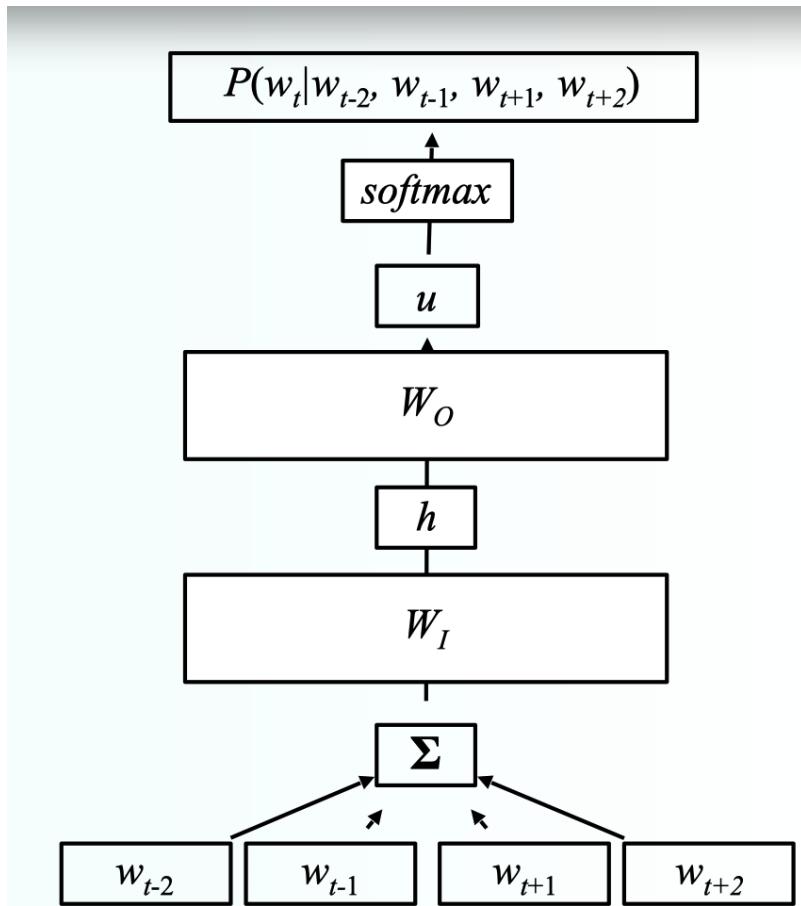
$$\log \frac{1}{1+e^{-c \cdot t}} + \sum_{i=1}^k \log \frac{1}{1+e^{n_i \cdot t}}$$

Una volta che abbiamo terminato la fase di learning per ogni parola avremo due vettori di embedding t_i e c_i e quello che possiamo fare è o considerare solamente il vettore di embedding t_i oppure considerarli entrambi facendo la somma $t_i + c_i$ oppure possiamo concatenare i due embedding.

In questo algoritmo anche la dimensione della finestra che consideriamo per ognuna delle parole che stiamo prendendo in considerazione è un parametro e possiamo eseguire il tuning di questo parametro utilizzando il dev set. In questo caso più aumentiamo la L e più aumenta la difficoltà nell'esecuzione della fase di training.

2.3.2 CBoW

Fino ad ora abbiamo parlato soprattutto del modello Skip-Gram ovvero del modello che cerca di capire se il contesto è adatto per una certa parola. Ora vediamo il CBoW ovvero il Continuous Bag of Words che è un modello in cui noi abbiamo il contesto e cerchiamo di capire quanto è possibile che una parola sia all'interno di un certo contesto.



In questo caso lo schema cambia e noi abbiamo delle parole che mi permettono di calcolare una media degli embedding del contesto che abbiamo in input, quindi h è questa media degli embedding. Poi consideriamo l'embedding della parola che abbiamo in input e lo confrontiamo con l'embedding h e applichiamo la softmax per ottenere una probabilità.

2.3.3 Alternative a Word2Vec

Una alternativa a Word2Vec si chiama Glove ed è un metodo basato sulle probabilità condizionali, la proporzione di queste probabilità condizionali potrebbe catturare i significati delle parole.

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

	$x = \text{solid}$	$x = \text{gas}$	$x = \text{water}$	$x = \text{random}$
$P(x \text{ice})$	large	small	large	small
$P(x \text{steam})$	small	large	large	small
$P(x \text{ice})/P(x \text{steam})$	large	small	~1	~1

Un'altra alternativa è fastText che è simile a CBoW e permette di utilizzare la media degli embedding delle parole in modo da ottenere una buona rappresentazione della frase che stiamo considerando.

2.4 Co-Occurrence Count

Torniamo un attimo al caso in cui abbiamo una matrice di dimensioni $|V| * |V|$ dove la $|V|$ indica la dimensione del nostro vocabolario. Si tratta di una matrice enorme in cui per ogni parola contiamo quante volte quella parola compare nello stesso contesto con un'altra parola (All'interno della matrice possiamo inserire i valori relativi al TfIdf).

Vorremmo eseguire una riduzione della dimensione della matrice in modo da avere una matrice più piccola come con gli embedding dello skip-gram e possiamo farlo utilizzando la PCA oppure l'SVD. Lavorando con questa matrice possiamo poi dare un peso maggiore ad alcuni delle parole che co-occorrono insieme ad altre basandomi sulle caratteristiche del corpus.

SVD

L'SVD è un metodo che mi permette di decomporre la matrice X di partenza di dimensione $m*n$ in tre matrici:

- U è la matrice ortogonale di dimensione $m * n$
- Σ è la matrice diagonale di dimensione $n*n$, i valori sulla diagonale sono $\sigma_1, \dots, \sigma_n$ e sono i singular value della matrice X di partenza.

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

- V è una matrice ortonormale di dimensione $n * n$,

Quello che possiamo fare è ordinare le righe delle matrice in base ai valori di σ e possiamo tenere solamente le K righe che corrispondono ai K valori più alti di σ .

Il problema di questo metodo è il costo, abbiamo infatti un costo che è quadratico e soprattutto la difficoltà di inserire nuove informazioni che arrivano da nuove parole o da nuovi documenti.

Hellinger Principal Component Analysis

In questo caso il funzionamento è differente, qua noi eseguiamo la PCA con la probability matrix delle co-occorrenze minimizzando l'errore di ricostruzione basandoci sulla Hellinger distance tra le varie parole.

2.4.1 Considerazioni

Ci sono alcuni studi riguardo a questi metodi di riduzione della dimensione della matrice e riguardo al Word2Vec:

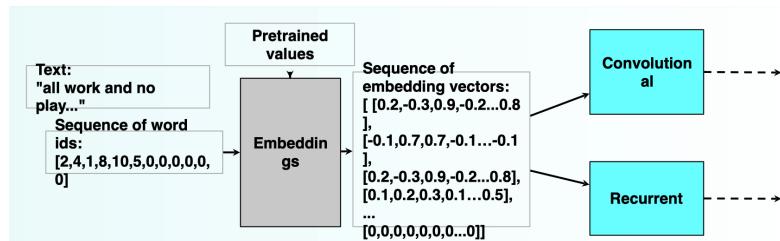
- Il Word2Vec calcolato eseguendo anche il sampling degli esempi negativi è molto simile alla fattorizzazione di una variante della matrice X in cui salvavamo i conteggi delle co-occorrenze. Il word2Vec è un procedimento che è lineare nella dimensione dell'input e in particolare l'algoritmo di training può anche essere parallelizzato, quando ci sono delle sovrapposizioni tra i contesti non è una cosa troppo importante e spesso la ignoriamo. Il modello possiamo adattarlo con nuovi dati quando ce ne sono.
- Non c'è un vincitore tra SVD, PCA, GloVe..., i parametri che vengono utilizzati hanno un ruolo fondamentale in questo caso.

Esistono degli embedding già calcolati, un esempio è quello di Google che comprende gli embedding di 100 miliardi di occorrenze di parole. Quello che determina la semantica catturata dall'embedding e anche il funzionamento (corretto o meno) dell'embedding è sicuramente l'insieme dei dati di training.

2.5 Embedding in Neural Networks

Solitamente quando abbiamo una rete neurale che lavora con dei testi, il primo layer che viene inserito nella rete è quello che utilizziamo per il calcolo dell'embedding delle parole presenti all'interno del testo. Questo layer consiste in una matrice W di dimensione $|V| * d$ dove V è la dimensione del dizionario e d è la dimensione dell'embedding space. Questa matrice W serve per mappare le parole nella "dense representation" ovvero per creare gli embedding e viene o inizializzata in modo random oppure con embedding per cui è già stato eseguito il training.

Durante la fase di Learning i pesi presenti all'interno di questa matrice possono essere modificati oppure tenuti fissi (questo è utile se usiamo dei pesi pre-trained).



2.6 Limiti

I principali problemi del word embedding:

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

- Le parole possono assumere vari significati, come possiamo fare a catturare tutti questi significati? Una possibile soluzione è ELMo che genera gli embedding di una parola basandosi sul contesto in cui compare.
- Sono sistemi che funzionano solamente con le parole
- Ci sono gli antinomi che spesso sembrano simili ma in realtà non lo sono e questo non va bene se vogliamo fare sentiment analysis o se abbiamo parole polisemiche.

Ci sono due tipologie di Word Embedding da considerare:

- Context Aware: in questo caso l'embedding della parola dipende dal contesto che stiamo considerando e questo è utile se vogliamo disambiguare delle parole o se dobbiamo classificare un documento.
- Sense Specific: è un word embedding specifico per la sentiment analysis, utilizza un corpus annotato.

Chapter 3

Vector Semantics and Embedding - Approfondimento da SLP Capitolo 6

3.1 Introduzione

Qualche definizione per iniziare:

- Distributional Hypothesis: le parole che compaiono nello stesso contesto tendono ad avere lo stesso significato. Con distributional hypothesis intendiamo il collegamento tra il modo in cui le parole sono distribuite e la similitudine delle parole.
- Embeddings: sono una rappresentazione del significato delle varie parole che abbiamo nel corpus.
- Vector Semantics: istanziano le ipotesi linguistiche eseguendo il

learning degli embedding direttamente dalla distribuzione degli embedding all'interno del testo.

Questa rappresentazione delle parole viene utilizzata in molti casi in NLP ed è anche il primo esempio di representation Learning ovvero un processo che esegue in automatico il learning di una rappresentazione del testo in input.

3.2 Come lo rappresentiamo il significato di una parola?

Noi vorremmo trovare il modo di rappresentare il significato delle parole, per farlo vorremmo un modello che ci permetta di capire questo significato in modo del tutto automatico. Il modello dovrebbe essere anche in grado di dirci se due parole sono sinonimi o contrari o ad esempio anche di dirci se una parola rappresenta qualcosa di allegro o di triste. In generale un modello di questo genere dovrebbe permetterci di fare inferenza in modo da poter risolvere dei task in cui è necessario capire il significato della frase che stiamo analizzando, questo è utile ad esempio nel question answering.

In generale ci interessa studiare la semantica delle varie parole, abbiamo alcune cose da considerare:

- Data una parola, ad esempio mouse, che chiamiamo lemma o anche citation form abbiamo possibili significati associati.
- Ad esempio potremmo considerare l'animale o anche il mouse del computer, questo vuol dire che abbiamo vari possibili word sense e che la parola è polysemous.
- Data una parola possiamo avere alcuni word sense che sono molto simili tra loro, questi sono i sinonimi. Possiamo considerare due

parole sinonimi se le possiamo usare una al posto dell'altra. In generale possiamo anche dire che non ci sarà sempre una corrispondenza precisa tra le due parole che vogliamo usare come sinonimi, infatti una parola potrebbe essere più adatta in alcuni contesti mentre un sinonimo potrebbe essere più adatto in un altro contesto.

- Una cosa molto importante è il concetto di Word Similarity, qua noi non stiamo più considerando i sinonimi ma la somiglianza tra le parole. Ad esempio cane e gatto non sono sinonimi ma sono parole simili dal punto di vista del significato nel senso che le utilizziamo in un contesto e in una frase che hanno un significato generale sicuramente simile tra loro. Questo è importante nei task di NLP.
- Un'altra feature interessante da considerare è la Word Relatedness, qua noi consideriamo quanto le parole sono connesse tra loro, ad esempio le parole caffè e tazza sono molto legate tra di loro. Partendo dalla definizione di relazione tra le parole possiamo dire che le parole appartengono allo stesso campo semantico. Il campo semantico è interessante anche quando vogliamo utilizzare LDA per creare dei gruppi di parole simili tra loro partendo dal testo.
- Una feature più generale del semantic field è il semantic frame in cui andiamo a considerare un gruppo di parole che denotano la partecipazione ad un certo evento, ad esempio se parliamo di una transazione bancaria avremo un semantic frame in cui utilizziamo dei verbi che sono tutti legati all'idea delle transazioni.
- Un altro aspetto da considerare è la connotation e il sentiment delle parole. Data una parola quello che potremmo fare è considerare la connotazione positiva/negativa della parola.

Tutti questi aspetti che abbiamo descritto fino ad ora vanno considerati quando lavoriamo con un modello di NLP perchè il nostro testo

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

che abbiamo in input va "trasformato" in embedding che devono tenere in considerazione queste feature.

3.3 Vector Semantic

Un modello che è perfettamente in grado di gestire tutti questi aspetti che abbiamo descritto è chiamato vector semantic.

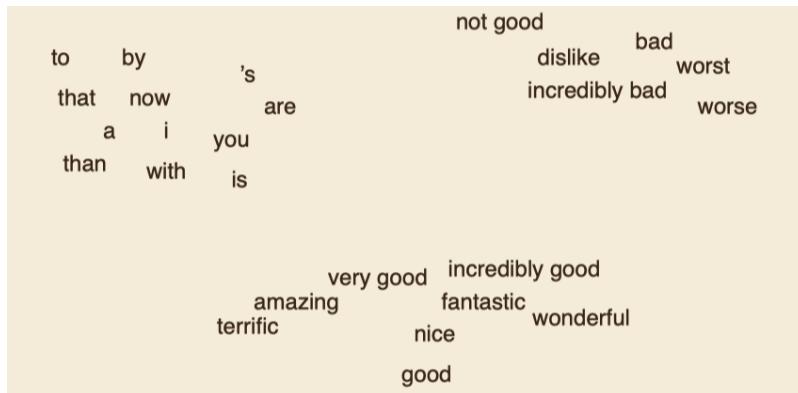
L'idea di base di questi vector semantic è che vorremmo definire una parola tramite l'ambiente in cui viene utilizzata andando a considerare i vari contesti in cui compare (i vari contesti in cui compare sono chiamati Distribuzioni di parole), le parole vicine e la grammatica. L'idea è che se 2 parole che occorrono in una distribuzione molto simile tra loro allora è altamente probabile che hanno anche lo stesso significato. Vediamo un esempio:

- (6.1) Ongchoi is delicious sauteed with garlic.
- (6.2) Ongchoi is superb over rice.
- (6.3) ...ongchoi leaves with salty sauces...

Supponiamo di non conoscere la parola Ongchoi ma abbiamo le frasi sopra, vedendo il contesto in cui compare la parola riusciamo a comprendere più o meno il possibile significato. Possiamo fare la stessa cosa da un punto di vista computazionale andando a contare le parole all'interno del contesto di Ongchoi. Vediamo che troviamo più o meno sempre le stesse parole in quel contesto e il fatto che quelle stesse parole compaiono ad esempio nel contesto di un'altra parola che può essere per esempio "Spinaci" ci permette di scoprire una similarità tra le Ongchoi e Spinaci.

I vector semantic combinano due intuizioni, l'idea della distribuzione delle parole (definire una parola considerando quali sono le altre parole che compaiono in quello stesso contesto) e l'intuizione di definire il sig-

nificato della parola w come un vettore ovvero come una lista di numeri, come un punto in uno spazio N-dimensionale.



L'idea del vector semantic è quella di rappresentare le parole in un piano come vediamo nella immagine sopra. I vettori che rappresentano le parole sono generalmente chiamati embedding perchè la parola è embeddata in uno specifico vector space.

Notare che nella figura sopra abbiamo una divisione tra le parole positive e le parole negative e questo è uno dei vantaggi del vector semantics perchè ci offre un modello per il significato delle parole che ci permette di implementare la somiglianza tra le parole.

Questi modelli di vector semantic sono anche molto pratici perchè possiamo eseguire il learning in modo rapido e automatico senza la necessità di avere una labellizzazione o un problema supervisionato.

Il risultato di tutte queste caratteristiche fa sì che i vector models possano essere utilizzati per rappresentare il significato delle parole in NLP, il modello più semplice utilizzato per creare questi vettori è il tf-idf in cui definiamo il significato della parola è definito in funzione delle parole che si trovano nelle vicinanze. Questo modello però comporta la creazione di vettori molto lunghi che sono sparsi e contengono molti 0. L'alternativa è il word2vec model, una famiglia di modelli che sono invece più corti e densi e che hanno anche interessanti proprietà semantiche.

3.4 Parole e Vettori

I vettori possono essere costruiti partendo da una co-occurrence matrix che è un modo per rappresentare quante volte due parole co-occorrono all'interno di uno stesso documento. Ci sono vari modi per costruire questa co-occurrence matrix:

- Term-Document Matrix: in questo caso abbiamo le righe della matrice che mi rappresentano le parole nel nostro vocabolario mentre invece le colonne rappresentano i vari documenti. Ogni cella della matrice rappresenta il numero di occorrenze di quella specifica parola che sono presenti all'interno di quel documento. Questa term-document matrix viene definita come parte del vector space model in information retrieval, in questo modello ciascuno dei vettori viene rappresentato come un vettore di contatori. Questa rappresentazione con un vettore di ciascun documento ci permette di rappresentare il vettore in uno spazio N-Dimensionale e possiamo anche notare che i documenti che sono simili tra loro conterranno più o meno le stesse parole e magari anche in quantità simili e quindi si troveranno in posizione simile all'interno dello spazio N-dimensionale. Questa tipologia di matrici è stata originariamente sviluppata per comprendere la somiglianza dei documenti all'interno di task di information retrieval.
- Words as vectors: abbiamo associato fino ad ora un vettore ad ogni documento, ora vogliamo associare ad ogni parola un vettore che possa rappresentare il significato di quella parola. In questo caso abbiamo un vettore per riga, in ogni riga contiamo la quantità di parole di quel tipo che sono presenti all'interno di ognuno dei documenti. Qua per la somiglianza vale un discorso simile a quello che abbiamo fatto prima con i vettori colonna, se due vettori riga sono simili allora vuol dire che le due parole sono simili perchè compaiono all'interno degli stessi documenti.

- Word-Word matrix o Term-Context Matrix: è più comune che si crei una matrice di dimensione $|V| * |V|$ in cui $|V|$ è la dimensione del dizionario, quindi abbiamo un numero di righe e di colonne pari alle parole nel dizionario. All'interno delle celle di questa matrice contiamo quante volte le due parole che stiamo considerando compaiono insieme. Il contesto in cui le parole devono comparire insieme può essere l'intero documento oppure, solitamente, solamente una piccola parte del documento, ad esempio 5 parole prima di quella che stiamo prendendo in considerazione e 5 dopo. In questo modo otteniamo una matrice di co-occorrenze in un certo contesto. Anche in questo caso possiamo avere una rappresentazione nello spazio di queste parole e possiamo capire quali sono quelle che sono simili tra di loro.

Per calcolare la distanza tra i vettori e quindi la differenza tra i vettori possiamo utilizzare una metrica che viene chiamata cosine similarity. In pratica quello che facciamo è calcolare il coseno dell'angolo che si forma tra le due parole che stiamo considerando.

$$\text{cosine}(\mathbf{v}, \mathbf{w}) = \frac{\mathbf{v} \cdot \mathbf{w}}{|\mathbf{v}| |\mathbf{w}|} = \frac{\sum_{i=1}^N v_i w_i}{\sqrt{\sum_{i=1}^N v_i^2} \sqrt{\sum_{i=1}^N w_i^2}}$$

All'interno della formula dobbiamo anche normalizzare e per farlo dividiamo per la lunghezza dei due vettori.

3.4.1 TF-IDF

Fino ad ora abbiamo creato i vettori relativi alle varie parole utilizzando il conteggio delle parole che compaiono insieme. Per come l'abbiamo

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

fatto fino ad ora però non stiamo considerando il tipo di contesto in cui si trovano le parole e non stiamo neanche considerando che ci sono delle parole che sono più frequenti di altre che ci potrebbero portare su una strada sbagliata. La soluzione quindi può essere l'utilizzo del Tf-idf, con questo algoritmo consideriamo il prodotto di due termini, ognuno cattura una cosa differente:

- Tf cattura la term frequency ovvero quanto il termine t compare all'interno del documento d.
- idf: con questo secondo fattore forniamo un peso più alto alle parole che occorrono solamente all'interno di pochi documenti. I termini che sono limitati a pochi documenti sono utili per discriminare questi documenti da tutti gli altri mentre i termini che occorrono in tutti i documenti non sono utili.

Un esempio di matrice in cui utilizziamo il Tf-Idf:

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	0.074	0	0.22	0.28
good	0	0	0	0
fool	0.019	0.021	0.0036	0.0083
wit	0.049	0.044	0.018	0.022

Utilizziamo il Tf-Idf soprattutto per calcolare la somiglianza tra le parole, si tratta di un tool utile per task come ad esempio la ricerca di differenti significati delle parole in differenti corpus. Il tf-idf può anche essere utilizzato per capire se due documenti sono simili tra loro, calcoliamo i vettori di ogni parola presente all'interno del documento, poi calcoliamo il centroide di questo vettore e poi usiamo la cosine similarity tra i due risultati che abbiamo ottenuto con il centroide.

3.5 Word2Vec

Fino ad ora abbiamo rappresentato le parole utilizzando dei vettori che sono più lunghi del dovuto e che hanno all'interno molti 0, perchè abbi-

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

amo visto il calcolo di questi vettori di parole rispetto alle parole stesse per comprendere le co-occorrenze o rispetto ai documenti per capire in quali documenti compaiono. Ora vorremmo trovare un modo che ci permetta di rappresentare le parole in vettori che devono essere corti e devono essere densi. I vettori densi funzionano meglio in NLP, non sappiamo bene perchè ma ci sono delle intuizioni:

- Dato che la dimensione di questi vettori è minore della dimensione dei vettori che abbiamo usato fino ad ora, possiamo usarli in un modello di ML perchè non saranno necessari troppi pesi.
- Non essendo troppo lunghi e non essendoci troppi pesi non andiamo in overfitting.
- Dato che i vettori sono densi possiamo catturare meglio i sinonimi rispetto ad un vettore sparso.

Uno dei metodi che possiamo utilizzare per creare questi embedding è word2Vec che è un algoritmo del 2013 che ha avuto una intuizione rivoluzionaria. Invece di contare quanto spesso la parola X compare vicino a Y l'algoritmo esegue il training di un classificatore che esegue una previsione e si chiede se è probabile che la parola X compaia vicino alla Y. Non siamo interessati alla risposta di questo problema di classificazione, siamo invece interessati ai pesi che otteniamo quando eseguiamo il learning di questo classificatore. Questi pesi che otteniamo sono quelli che chiamiamo word embedding.

L'intuizione rivoluzionaria quindi è stata quella di utilizzare il testo come se ci trovassimo in un problema di learning supervisionato perchè una certa parola X che troviamo vicino alla Y sarà una risposta corretta alla domanda "è possibile che X compaia vicino a Y?". Non abbiamo bisogno di labellare nulla per fare tutto questo.

Word2Vec funziona con un sistema molto più semplice di quello delle neural network infatti esegue il training di un classificatore logistic regression che classifica in modo binario. Il funzionamento è il seguente:

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

- La parola target e il suo vicinato vengono trattati come esempio positivo.
- Viene eseguito il sample di altre parole dal corpus in modo da avere anche esempi negativi.
- Viene usata la logistic regression per eseguire il training di un classificatore che distingue questi due casi, esempi positivi da esempi negativi.
- Utilizziamo i pesi di questo modello di logistic regression come embedding delle nostre parole.

3.5.1 Classificatore

Assumendo di avere una frase in cui abbiamo la parola che stiamo considerando e assumendo di utilizzare un context words di una certa dimensione il nostro obiettivo è eseguire il training di un classificatore che mi permette di capire se due parole sono simili tra loro o no. Noi per ognuna di queste parole abbiamo gli embedding che abbiamo calcolato considerando i pesi della Logistic Regression, qua nel classificatore quello che facciamo è considerare il prodotto tra i due vettori di embedding e poi utilizziamo la sigmoid function per generare la probabilità che quei due vettori siano simili tra loro.

Possiamo fare tutta questa classificazione delle parole che sono simili tra loro solamente se abbiamo gli embedding delle parole presenti all'interno del dizionario.

3.5.2 Learning

Come vengono creati gli embedding? Word2Vec parte con una generazione casuale degli embedding e poi iterativamente cerca di fare in modo che gli embedding delle parole che sono vicine a loro siano il più possibili

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

simili mentre gli embedding delle parole che sono lontane tra loro devono essere poco simili.

Supponendo di avere una frase ci focalizziamo su una parola presente all'interno della frase e prendiamo come esempi positivi le parole che sono nei dintorni mentre come esempi negativi parole a caso del corpus che non sono nei dintorni. Le parole casuali che usiamo negli esempi negativi sono chiamate noise word e sono scelte in base alla loro frequenza all'interno del corpus.

L'obiettivo dell'algoritmo di learning, dati gli embedding iniziali e dato il set iniziale di embedding consiste nel:

- Massimizzare la similarità tra la parola target e le parole prese dagli esempi positivi.
- Minimizzare la similarità della parola target con le parole prese dal gruppo degli esempi negativi.

Per eseguire il training di queste due funzioni obiettivo possiamo utilizzare lo stochastic gradient descent in modo da modificare in modo iterativo i parametri (ovvero gli embedding) e massimizzare le funzioni. Solitamente per ogni parola vengono memorizzati due embedding, uno è il target embedding e uno è il context embedding.

3.6 Approfondimento su Glove

I metodi che vengono utilizzati per creare i word vectors erano, fino all'introduzione di Glove, principalmente due:

- Global Matrix Factorization: un esempio è la Latent Semantic Analysis (LSA) che utilizza la low rank approximation per decomporre matrici molto grandi cercando di catturare le informazioni che riguardano un certo corpus. In LSA le righe della matrice

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

corrispondono alle parole mentre le colonne ai documenti del corpus. Il problema di questo metodo è che le parole molto comuni comportano dei problemi nel momento in cui vogliamo calcolare la somiglianza tra le parole.

- Local Context Window: questo è il caso del word2vec, il problema di questo metodo è che il training degli embedding viene fatto localmente ai singoli contesti e non viene considerata una visione più generale di tutto il corpus, quindi in pratica vanno a utilizzare male le statistiche del corpus. Allo stesso tempo però funzionano meglio per quanto riguarda il task di confronto di parole simili.

L'idea di Glove è di prendere il meglio di questi due sistemi.

Consideriamo il fatto che le statistiche riguardo la co-occorrenza delle parole in un corpus sono una fonte di informazioni fondamentale per tutti i task di unsupervised learning per imparare le word representation. Il problema è che non si sa bene in che modo generare della conoscenza che sia sensata partendo da queste matrici di co-occorrenza. L'idea dei ricercatori che hanno prodotto Glove quindi è quella di andare a costruire un modello per la creazione degli embedding che sia in grado di catturare le informazioni statistiche del corpus.

Cerchiamo di capire come Glove estrae informazioni interessanti dalle statistiche del corpus:

- Supponiamo di avere la nostra matrice con le co-occorrenze e poi possiamo definirci $P(k|w)$ che è la probabilità che la parola k capiti nel contesto di w.
- Prendiamo due parole, ghiaccio e vapore, la relazione tra queste due parole la possiamo analizzare andando a considerare il rapporto tra la probabilità di co-occorrenza di ognuna di queste due parole con K parole che sono chiamate probe words.

- Consideriamo ora la parola *solido*, questa sarà più legata a ghiaccio che a vapore, quindi avremo che $P(\text{solido}|\text{ghiaccio})$ sarà maggiore della $P(\text{solido}|\text{vapore})$.
- Possiamo anche notare che il rapporto tra $P(\text{solido}|\text{ghiaccio})$ e $P(\text{solido}|\text{vapore})$ sarà grande. Allo stesso tempo se prendiamo la parola *gas* avremo esattamente l'opposto e in particolare in questo caso il rapporto avrà un valore basso. Se la parola è legata ad entrambe oppure non è legata a nessuno invece avremo un numero vicino a 1.

Questo ragionamento dovrebbe quindi farci capire che invece di considerare le singole probabilità possiamo considerare il rapporto tra le probabilità per capire in che modo le parole sono legate tra loro e per estrarre quindi una qualche forma di conoscenza dalle statistiche del corpus.

Possiamo capire meglio il perchè dalla tabella sotto:

Probability and Ratio	$k = \text{solid}$	$k = \text{gas}$	$k = \text{water}$	$k = \text{fashion}$
$P(k \text{ice})$	1.9×10^{-4}	6.6×10^{-5}	3.0×10^{-3}	1.7×10^{-5}
$P(k \text{steam})$	2.2×10^{-5}	7.8×10^{-4}	2.2×10^{-3}	1.8×10^{-5}
$P(k \text{ice})/P(k \text{steam})$	8.9	8.5×10^{-2}	1.36	0.96

Considerando la tabella infatti vediamo che la parola *solid* è molto legata a *ice* ma non a *steam* perchè il rapporto tra le due probabilità è molto alto, allo stesso tempo *gas* fa esattamente il contrario e il rapporto tra le due probabilità è molto basso. La cosa importante però è che utilizzare questo rapporto tra le due probabilità mi permette di dare poco peso a quelle parole che non mi fanno discriminare perchè o non sono legate con *ice* e *steam* o sono legate con entrambi (in questo caso il rapporto è circa 1).

Come detto l'obiettivo di Glove è imparare degli embedding tali che il prodotto degli embedding sia uguale al logaritmo della probabilità della

co-occorrenza.

Vediamo nel dettaglio come funziona Glove dal punto di vista matematico:

- Abbiamo detto fino ad ora che dobbiamo calcolare la probabilità e poi il rapporto tra queste probabilità, assumiamo di avere una funzione definita in questo modo:

$$F(w_i, w_j, u_k) = \frac{P_{ik}}{P_{jk}}$$

All'interno della formula abbiamo w e u che sono due embedding layer che funzionano nello stesso modo ma si differenziano per l'inizializzazione, avere due livelli aiuta per ridurre l'overfitting.

- Possiamo riscrivere la formula del punto precedente in questo modo:

$$F(w_i - w_j, u_k) = \frac{P_{ik}}{P_{jk}}$$

- Ora vorremmo rendere il parametro della funzione F uno scalare e questo lo possiamo fare calcolando la trasposta e il prodotto tra i due parametri in questo modo:

$$F((w_i - w_j)^T * u_k) = \frac{P_{ik}}{P_{jk}}$$

- Questa F cosa può essere? Possiamo assumere delle proprietà ad esempio che sia un omomorfismo e quindi possiamo riscrivere la formula come:

$$F(w_i * u_k - w_j * u_k) = \frac{F(w_i * u_k)}{F(w_j * u_k)}$$

$$= \frac{P_{ik}}{P_{jk}}$$

Quindi abbiamo

$$\frac{F(w_i * u_k)}{F(w_j * u_k)} = \frac{P_{ik}}{P_{jk}}$$

E possiamo ricavare che $F(w_i * u_k) = P_{ik}$. In realtà questa ultima uguaglianza la possiamo ricavare considerando che il termine P_{ik} è moltiplicato per una costante c.

- Ora assumiamo che $F = \exp$ e quindi possiamo dire che: $\exp(w_i * u_k) = P_{ik} = \frac{X_{ik}}{X_i}$. Usando le proprietà dei logaritmi abbiamo quindi:

$$w_i * u_k = \log(X_{ik}) - \log(X_i)$$

$$w_i * u_k + \log(X_i) = \log(X_{ik})$$

- Facciamo una ulteriore assunzione e vogliamo inserire il bias all'interno di questa funzione quindi abbiamo $\log(X_i) = bw_i + bu_k$ quindi:

$$w_i * u_k + bw_i + bu_k = \log(X_{ik})$$

- Questa che abbiamo ottenuto è la cost function, in un caso ideale sarebbe 0, l'obiettivo è minimizzarla.

Con tutti questi calcoli quello che siamo riusciti a fare è ridurre il calcolo della loss function da 3 elementi (i, j, k) a due soli elementi che sono i e k . In questo modo riusciamo ad avere una loss function più semplice da calcolare.

Concludendo, il vantaggio di Glove è che non sfrutta solamente statistiche locali (informazioni locali dei contesti) ma incorpora statistiche globali (co-occorrenze delle parole).

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

Chapter 4

Lesson 3 - Tokenization

4.1 Regular Expression

Con regular expression (Regex) intendiamo un linguaggio formale per specificare stringhe di testo. Ad esempio possiamo utilizzarlo quando dobbiamo cercare un certo pattern (una stringa di un certo tipo) all'interno di un testo, alla fine mi vengono restituite tutte le occorrenze che matchano con la nostra query oppure solamente una (possiamo scegliere). Il testo in cui cerchiamo lo chiamiamo corpus, il corpus può essere formato da una serie di testi o può essere un singolo testo.

Quando lavoriamo con le Regex possono accadere vari tipi di errori, possiamo avere falsi positivi (potremmo matchare una stringa che non doveva essere matchata), falsi negativi (potremmo non matchare una stringa che in realtà andava matchata). In Natural language processing questi errori vanno sempre considerati e quando riusciamo a ridurne la presenza allora riusciamo a migliorare il nostro modello aumentando l'accuracy del modello (aumentiamo quindi la precision del modello ovvero minimizziamo i falsi positivi) e aumentando anche la recall (ovvero minimizziamo i falsi negativi).

Le regex sono comunque utilizzate e anche se utilizziamo dei clas-

sificatori che lavorano tramite Machine Learning possiamo comunque sfruttarle come feature dei classificatori.

4.2 Tokenization

4.2.1 Words

La prima cosa che dobbiamo chiederci quando lavoriamo con il testo è una definizione più precisa di parola. A seconda del tipo di problema che stiamo affrontando la definizione di parola può cambiare:

- Tra Cat e Cats cambia solamente il plurale ma il significato di base è fissato, dobbiamo considerarle come una sola parola o come due parole? In questo caso la parte in comune è detta stem e il set di parole che hanno in comune lo stem è chiamato Lemma. La s finale in questo caso è un wordform.
- Nella frase "I do uh main- mainly business data processing" abbiamo delle parole che non sono complete, magari stavo parlando e ho pronunciato solamente una parte della parola. Se sto convertendo in testo questa frase non le vorrei avere ma se invece voglio fare una previsione sulla prossima parola del discorso possono essere utili perché potrebbero significare che ci siamo fermati e quindi stiamo ricominciando la frase. Questi tipi di parole sono dette fillers.
- Anche il numero di parole (e cosa consideriamo parola può cambiare). Possiamo avere un dizionario di una certa dimensione ma avere allo stesso tempo un numero di parole nel testo che è maggiore perchè alcune sono ripetute. Possiamo avere la punteggiatura, in alcuni casi questa possiamo considerarla come parole in altre non serve.

4.3 Unix Tool per la Tokenizzazione

Una versione molto semplice della tokenizzazione di un testo è quella che possiamo ottenere con dei comandi unix. In particolare un comando che possiamo utilizzare è il comando *tr* che con alcune opzioni ci permette di tokenizzare il testo ovvero di dividerlo in tante parole. Una volta eseguita la tokenizzazione possiamo poi scriverla su un file e ordinarla in base al numero di occorrenze. Una cosa che possiamo fare per avere un ordinamento ancora migliore è non considerare le differenze tra parole in maiuscono e in minuscono e quindi contare le occorrenze. Eseguendo il comando `\it tr -sc 'A-Za-z' '\n' < sh.txt | tr A-Z a-z | sort | uniq -c` riusciamo ad ottenere la lista complessiva delle parole con le loro occorrenze ordinata per il numero di occorrenze.

Possibili problemi con la Tokenizzazione

Quando eseguiamo la tokenizzazione con sistemi "semplici" come quello che abbiamo visto nel caso del comando "*tr*" potremmo avere alcune difficoltà:

- La punteggiatura potrebbe essere importante ma non viene tenuta in considerazione. Lavorando con NLP una virgola o un punto possono significare molto nel discorso ma in realtà non vengono tenute in considerazione da *tr*. Un'altra cosa da considerare è che in alcuni casi la punteggiatura potrebbe essere proprio interna al nome, ad esempio *P.h.D..*
- Un'altra difficoltà sono quelle parole che sono composte, ad esempio New York. È una parola composta da due parole e se prese singolarmente non hanno molto senso.
- Un altro problema lo abbiamo ad esempio in italiano con la parola L'insieme in cui potremmo considerare una sola parola o due parole.

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

- Un altro problema ce l'abbiamo con la lingua cinese perchè non utilizzano spazi tra le varie parole e quindi diventa tutto molto più complesso. Nella lingua cinese abbiamo che ogni parola è in media lunga 2.4 caratteri e un algoritmo che viene utilizzato per la tokenizzazione è la text segmentation, in particolare uno viene chiamato Maximum Matching. Il maximum matching è un algoritmo greedy che può essere applicato a dei testi dato un vocabolario di parole. Il funzionamento è il seguente:
 - Abbiamo a disposizione un testo e un vocabolario, partiamo dalla prima lettera del testo.
 - Cerchiamo nel vocabolario la parola più lunga che matcha la parola che inizia con la lettera in cui siamo posizionati e la memorizziamo.
 - Ci posizioniamo alla fine della parola che abbiamo trovato.
 - Andiamo avanti iterativamente ripartendo dal punto 2.

L'algoritmo non funziona benissimo con molte lingue, ad esempio con l'inglese, ma funziona in modo accettabile in Cinese. Algoritmi migliori di questo sono ad esempio la probabilistic segmentation e i sequence model come ad esempio gli Hidden Markov Models.

- Altro problema lo abbiamo quando dobbiamo considerare le date o vogliamo indicare una certa quantità di denaro.

4.4 Trasformazione del testo

4.4.1 Word Normalization

Il task di Word Normalization consiste nel trasformare parole e token in un formato standard scelto per analizzare varie fonti di dati. Ad esempio potremmo scrivere USA o US, si sceglie una versione condivisa e si

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

usa sempre quella. Questa standardizzazione è utile quando si parla di information retrieval perchè magari in alcuni casi quando cerchiamo informazioni riguardanti gli USA vorremmo ottenere le informazioni anche se c'è scritto US.

Quello che si fa quindi è definire in modo esplicito una equivalenza tra le classi di termini, un'alternativa è eseguire una espansione asimmetrica che consiste nel memorizzare tutte le possibili estensioni di una certa parola.

Prima di lavorare quindi sui dati va eseguita la normalizzazione del nostro testo che consiste in tre fasi principali:

- Tokenizing
- Normalizing word formats
- Segmenting sentences

4.4.2 Case Folding

Questo è un altro tipo di normalizzazione in cui andiamo a modificare le parole per fare in modo che all'interno della parola ci siano solamente lettere minuscole. Questo tipo di normalizzazione può risultare utile in alcuni casi, come ad esempio nel settore dell'information retrieval o nella speech recognition in altri invece può essere un problema, ad esempio nella sentiment analysis. Nella sentiment analysis infatti quello che succede è che una parola che viene scritta in maiuscolo potrebbe avere una interpretazione differente rispetto alla parola che viene scritta in minuscolo (US nazione e us pronome ad esempio).

4.4.3 Lemmatization

La lemmatization è il task tramite il cui passiamo da una certa forma della parola che stiamo considerando ad una forma base. Questo può

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

valere con le parole che hanno un plurale ma può anche valere per i verbi che vengono coniugati, in questo caso preferiamo avere la forma base del verbo, ad esempio per is e are noi preferiamo avere la forma base del verbo essere be.

Gli algoritmi più complicati di Lemmatization includono l'utilizzo di morphological parsing che vuol dire studiare in che modo le parole sono create partendo da unità più piccole chiamate "morphemes" che sono le unità più piccole che permettono di ottenere la parola finale. Il morphemes centrale della parola viene chiamato stem e fornisce il significato prevalente della parola mentre gli affixes sono quelle parti aggiuntive della parola che vengono aggiunte allo stem e che cambiano il significato della parola.

L'algoritmo di Lemmatization più semplice è quello che viene chiamato Stemming in cui si riduce la parola alla sua radice. Questo dipende molto dal linguaggio che stiamo utilizzando. Uno degli algoritmi di Stemming più famosi è l'algoritmo di Porter che viene utilizzato per lo stemming della lingua inglese, ci sono varie regole che servono per ottenere la radice dalla parola che stiamo prendendo in considerazione.

In alcuni linguaggi la segmentazione può diventare un problema, ad esempio con la lingua Turca.

4.5 Sentence Segmentation

La Sentence Segmentation è un altro importante algoritmo utilizzato nel text processing, in particolare qua ci chiediamo quanto siano importanti all'interno del testo la punteggiatura e altri simboli come ad esempio punti esclamativi/interrogativi. Ci sono contesti in cui il punto è solo la fine della frase, altri in cui invece è parte di una parola e con la Sentence Segmentation cerchiamo di risolvere questo problema.

La prima cosa che si decide nella sentence tokenization è come gestire i punti presenti nel testo, questo lo possiamo decidere utilizzando

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

delle regole (ad esempio possiamo creare un decision tree) oppure possiamo utilizzare tecniche di Machine Learning in cui andiamo ad eseguire il training di un modello per fare in modo che riconosca le parole che contengono i punti e le distingua da quelle che invece hanno punti come fine del discorso. In particolare i decision tree non vengono creati a mano (solamente se abbiamo poche feature da considerare) ma sono creati utilizzando algoritmi di induzione.

Un esempio di questi algoritmi è il Punkt Sentence Splitter. Possiamo vedere qua un esempio per la divisione in frasi:

```
import nltk.data
splitter = nltk.data.load('tokenizers/punkt_english.pickle')
for line in file:
    for sent in splitter.tokenize(line.strip()):
        print sent
```

E un esempio per la tokenizzazione delle varie parole:

```
tokenizer = splitter._lang_vars.word_tokenize
print '\u'.join(tokenizer(sent))
```

Chapter 5

Lesson 4 - Text Classification

5.1 Classificazione

Un altro dei task che vorremmo eseguire dato un testo e delle classi di appartenenza è la classificazione. Consideriamo di avere un testo, un primo algoritmo di classificazione è il Naive Bayes che mi permette di assegnare una categoria che mi classifichi quel testo. Ad esempio potremmo avere una recensione e vorremmo poter capire se questa è positiva o negativa. Questo primo task di categorizzazione viene chiamato sentiment analysis. Potremmo avere una mail e vorremmo sapere se è spam o meno. In questi esempi abbiamo un classificatore binario perché abbiamo sempre due possibili classi, in generale possiamo avere anche più classi. Altre cose che potrebbero interessarci di un testo e per cui potremmo eseguire degli algoritmi di classificazione:

- Comprendere la lingua in cui è stato scritto il testo.
- Determinare l'autore di un testo.

- Determinare l'argomento principale di un testo, questo è importante ad esempio nell'information retrieval se vogliamo capire quali sono gli argomenti principali di un paper ad esempio.
- La classificazione può anche essere utile per i task come tokenization e disambiguation (se vogliamo capire se un punto è parte della parola o no ad esempio.)

L'obiettivo della classificazione è prendere una singola osservazione ed estrarre delle feature che potrebbero aiutarci a classificare l'osservazione in una delle possibili classi. Per questo task di classificazione una idea potrebbe essere quella di creare delle regole di classificazione, queste dovrebbero essere create dall'uomo e quindi potrebbero sia essere troppo complicate da creare e allo stesso tempo potrebbe non essere semplice aggiornarle nel tempo. Una volta create le regole potremmo utilizzare la presenza di parole all'interno del testo per fare in modo che il testo venga classificato con una certa classe. Solitamente l'accuracy che si ottiene è molto alta ma allo stesso tempo abbiamo problemi perché mantenere questo tipo di regole è molto costoso.

La soluzione migliore quindi è utilizzare il supervised machine learning, abbiamo i nostri dati e li abbiamo labellati, per ognuna delle frasi ad esempio abbiamo anche la classe di appartenenza e noi vogliamo creare poi un modello che mi permetta di generalizzare ad istanze non ancora viste.

Quello che vogliamo creare è un classificatore che è in pratica una funzione $f : d \rightarrow c$. La d è l'input del classificatore mentre l'output è la c che è una delle classi, l'insieme delle classi lo definiamo come $C = c_1, \dots, c_n$.

Un classificatore è di tipo probabilistico nel momento in cui, date le varie classi, ci permette di avere per ogni classe la probabilità di classificazione dell'input usando quella classe.

5.2 Naive Bayes

Il classificatore Naive Bayes è un classificatore molto semplice che prende un testo e lo classifica sfruttando la Bayes Rule.

Quello che si fa con questo classificatore è prendere il testo e creare una rappresentazione Bag Of Word nel senso che non consideriamo la posizione delle parole ma facciamo la lista delle parole presenti all'interno del testo contando anche il numero di parole presenti. Quello che si fa è prendere poi il testo e applicare la funzione di classificazione che considererà le parole presenti al suo interno e il numero di occorrenze per capire quale classe assegnare. Alla fine assegnerà la classe che ha la posterior probability più alta.

L'idea di base di questo algoritmo Naive Bayes nasce dalla formula di Bayes che mi permette di riscrivere una probabilità condizionale in questo modo:

$$P(B|A) = \frac{P(A|B)P(B)}{P(A)}$$

in pratica quello che facciamo è invertire le due condizioni della probabilità condizionale e spesso questo ci permette di avere una maggiore facilità nel calcolare la probabilità. Quello che facciamo quindi è calcolare queste varie probabilità per ogni documento e per ogni classe e poi per classificare ogni documento consideriamo la probabilità massima che abbiamo calcolato. Nel caso di documenti e classi possiamo scrivere la formula in questo modo:

$$P(c|d) = \frac{P(d|c)P(c)}{P(d)}$$

Dato che noi stiamo cercando la c che massimizza la formula sopra, la nostra formula per trovare la classe migliore per il documento corrente è la seguente:

$$c^* = \operatorname{argmax}_c P(d|c)P(c)$$

in questa formula in particolare abbiamo $P(d|c)$ che è la likelihood mentre la $P(c)$ è la prior distribution.

Utilizzando un metodo di learning supervisionato vogliamo eseguire il learning di un classificatore γ che preso in input un testo X mi restituisca una classe C .

Il learning method lo chiamiamo Γ , questo prende in input il training set e mi restituisce in output il classificatore che abbiamo creato γ . Una volta che abbiamo ottenuto γ possiamo applicarlo sul test set.

Fino ad ora abbiamo sempre scritto la formula del nostro classificatore come:

$$P(c|d) = \frac{P(d|c)P(c)}{P(d)}$$

Quello che vogliamo fare ora è considerare la d come un insieme di feature che fanno parte del documento d , in questo caso si tratta di un insieme di lettere. Quindi avremmo una cosa del tipo

$$P(c|d) = \frac{P(d|c)P(c)}{P(d)} = \frac{P(x_1, \dots, x_n|c)P(c)}{P(x_1, \dots, x_n)}$$

e quando andiamo a massimizzare come abbiamo visto possiamo ignorare il denominatore quindi avremmo da massimizzare:

$$\operatorname{argmax}_{c_j \in C} P(x_1, \dots, x_n|c_j)P(c_j)$$

Calcolare questa probabilità direttamente è troppo complicato quindi Naive Bayes fa due assunzioni:

- La prima assunzione riguarda il fatto che noi non consideriamo importante la posizione delle parole all'interno del testo, quindi la lista delle feature è qualcosa in cui codifichiamo solamente le parole e non la loro posizione.

- La seconda assunzione è quella tipica di Naive Bayes e mi permette di considerare le varie feature tutte indipendenti tra loro. Questo vuol dire che possiamo scrivere la formula andando ad eseguire il prodotto delle varie feature:

$$\operatorname{argmax}_{c_j \in C} P(x_1, \dots, x_n | c_j) P(c_j) = \operatorname{argmax}_{c_j \in C} P(x_1 | C_j) P(x_2 | C_j) P(x_n | C_j) P(c_j)$$

Noi qua prendiamo semplicemente tutte le parole che sono all'interno del testo e ci calcoliamo la probabilità corrispondente.

- La terza assunzione che facciamo è che la probabilità $P(C_j)$ la possiamo andare a calcolare se stimiamo la frequenza di questa probabilità all'interno degli esempi di training. Ad esempio se abbiamo 3 classi, 50 esempi di training e la classe c_i compare 20 volte allora la probabilità sarà $P(C_i) = \frac{20}{50}$

Abbiamo detto che la probabilità delle varie classi viene calcolata come moltiplicazione di probabilità. Eseguendo tutte queste moltiplicazioni potremmo andare in underflow e quindi la soluzione che viene adottata è quella di utilizzare il logaritmo della probabilità:

$$c_{NB} = \operatorname{argmax}_{c_j \in C} \log P(c_j) \sum_i \log P(x_i | c_j)$$

5.2.1 Training del classificatore Naive Bayes

Per eseguire il training del classificatore quello che dobbiamo fare è il calcolo delle probabilità $P(c)$ e $P(x_i | c)$.

Abbiamo già visto che per calcolare $P(c)$ la formula è la seguente:

$$P(c) = \frac{N_c}{N_{doc}}$$

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

Per calcolare la probabilità $P(x_i|c)$ invece dobbiamo considerare quante volte compare la parola x_i all'interno del bag of word classificata con la classe c . Quindi la formula diventa:

$$P(x_i|c_j) = \frac{N(X_i = x_i, C = c_j)}{N(C = c_j)}$$

Il problema di questa formula è che se troviamo nel test set una parola che non è mai utilizzata con una certa classe c_j allora vuol dire che la $P(x_i|c_j)$ sarà uguale a 0 e quindi sarà uguale a 0 anche la formula complessiva della probabilità che quel testo venga classificato in una certa classe. Questo è un problema che possiamo risolvere con il Laplace Smoothing:

$$P(x_i|c_j) = \frac{N(X_i = x_i, C = c_j) + 1}{N(C = c_j) + k}$$

Dove la k indica il numero di valori di X_i .

Un'altra forma di smoothing per evitare l'overfitting e per evitare di avere probabilità uguali a 0 anche se non dovrebbero esserlo è la Bayesian Unigram Prior:

$$P(x_i|c_j) = \frac{N(X_i = x_i, C = c_j) + mp_{i,k}}{N(C = c_j) + m}$$

dove la $mp_{i,k}$ indica la frazione di dati dove vale $X_i = x_{i,k}$.

5.2.2 Generative Model

L'algoritmo Naive Bayes possiamo anche utilizzarlo come generative model per generare del testo. Quello che si fa è scegliere una classe, poi possiamo generare del testo scegliendo tra le parole che fanno parte di quella classe considerando le probabilità di ciascuno dei termini che fanno parte della classe.

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

5.3 Naive Bayes as a Language Model

Il classificatore Naive Bayes può utilizzare qualsiasi tipo di feature, se però utilizziamo solamente le caratteristiche delle singole parole e usiamo tutte le parole che fanno parte del testo allora abbiamo un'alta similarità con il language model in cui abbiamo che ogni parola è associata ad una certa probabilità. Dato che ogni parola ha una sua probabilità, ogni frase avrà la sua probabilità, la possiamo calcolare semplicemente facendo la moltiplicazione delle singole probabilità delle varie parole.

Ad esempio se consideriamo questo Bayes Model in cui assegnamo le varie probabilità alle parole in base a due classi, positiva e negativa, abbiamo che ognuna delle due colonne è un language model:

w	P(w +)	P(w -)
I	0.1	0.2
love	0.1	0.001
this	0.01	0.01
fun	0.05	0.005
film	0.1	0.1
...

Quello che possiamo fare quindi presa una certa frase è andare a calcolare la probabilità che quella frase sia classificata come positiva o

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

come negativa.

$$\begin{aligned} P(\text{"I love this fun film"}|+) &= 0.1 \times 0.1 \times 0.01 \times 0.05 \times 0.1 = 0.0000005 \\ P(\text{"I love this fun film"}|-) &= 0.2 \times 0.001 \times 0.01 \times 0.005 \times 0.1 = .0000000010 \end{aligned}$$

In questo caso stiamo considerando solamente la likelihood e non consideriamo alcuna prior probability e quindi quello che otteniamo è un modello che mi fornisce un risultato che sarà differente da quello in cui calcoliamo anche la prior.

5.4 Esempio di Naive Bayes

5.4.1 Learning Phase

Abbiamo un set di dati di training che vogliamo utilizzare per eseguire il training del nostro modello. In particolare l'algoritmo funziona in questo modo:

- Contiamo per prima cosa il numero di documenti N_c presenti all'interno del nostro dataset che vengono classificati con ognuna delle classe c .
- Presa ciascuna delle possibili classi calcoliamo la prior probability $\frac{N_c}{Count(C)}$ dove $Count(C)$ indica il numero di documenti presenti nel dataset. Questa è la prior probability.
- Prendiamo tutti i documenti che vengono classificati con la classe c e per ognuno consideriamo ognuna delle parole presenti nel vocabolario e poi contiamo il numero delle occorrenze.
- Ora possiamo calcolare la probabilità condizionale quindi prendiamo ogni parola presente nel dizionario calcoliamo il numero di

occorrenze di quella parola in frasi classificate come c diviso il numero di frasi che sono classificate come c .

```

TrainMultinomialNB( $C, D$ )
 $V \leftarrow \text{ExtractVocabulary}(D)$ 
 $N \leftarrow \text{CountDocs}(D)$ 

for each  $c$  in  $C$ 
     $N_c \leftarrow \text{CountDocsInClass}(D, c)$ 
     $prior[c] \leftarrow N_c / \text{Count}(C)$ 
     $text_c \leftarrow \text{TextOfAllDocsInClass}(D, c)$ 
    for each  $t$  in  $V$ 
         $F_{tc} \leftarrow \text{CountOccurrencesOfTerm}(t, text_c)$ 
        for each  $t$  in  $V$ 
             $condprob[t][c] \leftarrow (F_{tc} + 1) / (\sum(F_{tc}) + 1)$ 

return  $V, prior, condprob$ 

```

5.4.2 Testing Phase

Per quanto riguarda la fase di test del modello, applichiamo il modello ai dati che abbiamo nel test set, in particolare calcoliamo la log prior e la log likelihood, sommando tutte le varie probabilità riusciamo ad avere la probabilità per ognuna delle classi possibili per ognuna delle frasi che analizziamo.

Quello che facciamo quindi è assegnare alla frase la classe che ha la probabilità maggiore.

Quando eseguiamo la fase di test è fondamentale che il test set sia differente rispetto al training set altrimenti avremmo una valutazione troppo ottimistica.

Abbiamo varie metriche per valutare il nostro modello:

- La prima metrica che consideriamo è l'accuracy della classificazione, contiamo semplicemente le frasi che sono classificate correttamente C e poi prendiamo il numero totale delle frasi N , questa accuracy è data da $\frac{C}{N}$

- Per valutare il nostro sistema poi possiamo creare una confusion matrix in cui consideriamo 4 casi:
 - Elemento classificato come positivo e realmente positivo (vero positivo).
 - Elemento classificato come positivo ma negativo (falso positivo).
 - Elemento classificato come negativo ma positivo (falso negativo).
 - Elemento classificato come negativo e realmente negativo (vero negativo).

		gold positive	gold negative	
		true positive	false positive	precision = $\frac{tp}{tp+fp}$
system output labels	system positive	false negative	true negative	
	system negative			recall = $\frac{tp}{tp+fn}$
				accuracy = $\frac{tp+tn}{tp+fp+tn+fn}$

Partendo da questa tabella riusciamo a considerare varie metriche:

- La precision misura la percentuale di elementi che sono classificati come positivi e che in realtà sono positivi.

$$\text{Precision} = \frac{\text{true positives}}{\text{true positives} + \text{false positives}}$$

- La recall misura la percentuale di elementi che sono classificati correttamente dal sistema.

$$\text{Recall} = \frac{\text{true positives}}{\text{true positives} + \text{false negatives}}$$

Quello che si fa è trovare una tradeoff tra precision e recall perché se una è troppo alta l'altra è troppo bassa e viceversa. Allora è stato inventato un sistema differente che consiste nell'utilizzare una misurazione chiamata F-Measure che le incorpora entrambi:

$$F_\beta = \frac{(\beta^2 + 1)PR}{\beta^2P + R}$$

5.5 Multiclass Classification

Fino ad ora abbiamo sempre considerato una classificazione con 2 sole classi ma solitamente le classi che consideriamo sono più di 2, ad esempio per la sentiment analysis abbiamo tre classi, positivo, negativo e neutro.

Di queste classi multiple abbiamo due versioni:

- Una prima versione è quella in cui ogni elemento può essere assegnato a più di una classe. In questo caso abbiamo una multi-label classification.
- Una seconda versione, più comune, è quella in cui un elemento può essere assegnato ad una sola classe. In questo caso abbiamo una multinomial classification.

Quando dobbiamo testare il nostro modello con dei dati di test possiamo creare anche in questo caso una confusion matrix. Per ottenere una matrice che mi permetta di giudicare il risultato ottenuto con questa confusion matrix possiamo combinare i valori in due modi:

- Macroaveraging: calcoliamo le performance delle singole classi e poi calcoliamo la media delle varie classi. La macroaveraging riflette le statistiche delle classi più piccole ed è meglio utilizzarla quando le performance delle varie classi sono equamente importanti.
- Microaveraging: In questo caso calcoliamo le performance del modello usando una singola confusion matrix, da questa confusion matrix calcoliamo la precision e la recall. Questa microaveraging è dominata dalle classi più frequenti.

Parlando sempre di multiclass classifier possiamo fornire un peso maggiore ai vari tipi di errori che possiamo fare. Ad esempio se abbiamo varie classi e un testo che parla di moda viene classificato come sport, questo errore potrebbe essere peggiore di classificare un testo che parla di moda come testo che parla di notizie. Questo sistema di ranking lo possiamo creare o manualmente oppure chiedendo dei feedback all'utente.

Una volta che abbiamo capito come gestire questi pesi possiamo cercare di fare di nuovo il tuning del nostro modello in modo da minimizzare il costo totale.

5.6 Test set e Cross Validation

La procedura di testing di modelli di classificazione del testo è simile alla procedura di testing dei language modelling. In particolare abbiamo i nostri dati che vengono suddivisi in:

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

- Training set: per eseguire il training del nostro modello. Dalla dimensione del training set solitamente dipende il risultato che otteniamo sul test set, più il set di training è grande e più i nostri risultati sono migliori.
- Dev Set: chiamato anche validation set, lo utilizziamo per il tuning dei parametri.
- Test Set: lo utilizziamo per capire le reali performance del nostro modello. Non vanno mischiati questi dati con il training set altrimenti avremmo un modello che esegue il tuning anche considerando il test set e poi esegue il test sugli stessi dati con cui ha fatto il tuning.

Chiaramente quando dividiamo il nostro set di dati in tre parti riduciamo in modo drastico il numero di sample che possono essere utilizzati per la fase di learning del modello e il risultato del modello potrebbe anche essere dipendente dalle particolari scelte che facciamo in base ai dati che finiscono nel training set.

Una possibile soluzione per questo problema è chiamata cross-validation, in questo caso manteniamo il test set ma il validation set non è più necessario. Nell'approccio basilare, chiamato K-fold CV il training set viene suddiviso in K set più piccoli

- Il training del modello viene effettuato su $k-1$ dei fold dei dati di training.
- Il modello risultato viene validato sui rimanenti dati (vengono usati questi metodi per calcolare le performance del modello)

Per la misura di performance definitiva di K-Cross validation viene poi fatta una media tra tutti i valori calcolati in precedenza. Questo approccio può essere costoso ma non spreca dati



5.7 Conclusione

Naive Bayes ha dimostrato di poter essere utilizzato in alcune situazioni e di fornire anche risultati accettabili, i pro sono i seguenti:

- È robusto rispetto alle feature irrilevanti perchè si cancellano tra di loro e il risultato finale non ne risente. Al contrario i decision tree possono soffrire dalla presenza di feature irrilevanti.
- Se abbiamo un set di dati con delle feature che hanno la stessa

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

importanza, Naive Bayes si comporta bene e non ne preferisce una all'altra, cosa che non succede con i Decision Tree.

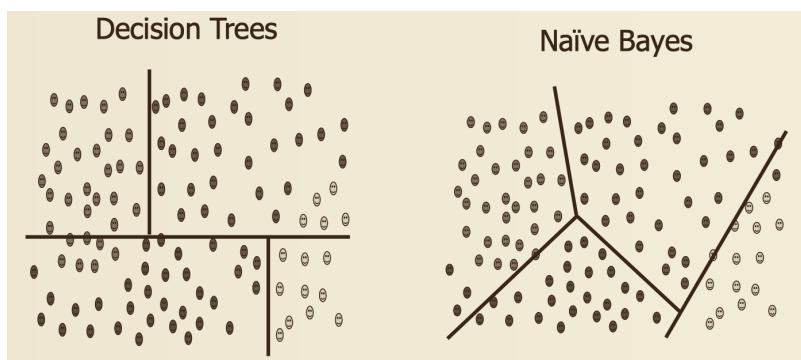
- Naive Bayes è una buona base di partenza per la classificazione del testo ma non la migliore.
- Nella formulazione di Naive Bayes avevamo assunto l'indipendenza delle varie parole tra loro quando abbiamo calcolato la probabilità di assegnare una classe ad un testo. Naive Bayes è ottimo se vale davvero l'indipendenza tra le classi.
- Naive Bayes è anche veloce perchè la fase di learning viene eseguita analizzando i dati solamente una volta e la fase di testing è lineare nel numero di attributi e nel numero di documenti all'interno della collezione.
- Naive Bayes può essere allenato in modo incrementale quando abbiamo nuovi dati.
- Non necessita di troppo spazio per lavorare.

Chapter 6

Lesson 5 - Neural Network Classification

6.1 Introduzione

Fino ad ora parlando di classificazione del testo abbiamo visto solamente il modello Naive Bayes che si comporta abbastanza bene ma che fa delle assunzioni molto forti che in alcuni casi possono diventare un problema.



Il modello Naive Bayes funziona meglio dei decision Tree perchè pur essendo un modello lineare permette di creare dei decision boundary che non sono soltanto paralleli o perpendicolari inoltre è un modello facile

da implementare che può scalare facilmente se aggiungiamo nuovi dati di training. È un modello efficiente e veloce che non richiede troppo spazio per funzionare ma con le sue assunzioni comporta dei problemi. Con Naive Bayes infatti ignoriamo completamente la posizione delle varie parole e poi le parole sono del tutto indipendenti tra loro e questo non va bene perchè in alcuni casi le parole sono molto dipendenti tra di loro e anche l'ordine delle parole può essere importante per la classificazione. Il fatto che le parole vengono considerate indipendenti poi fa sì che l'effetto della presenza di alcune parole che sono molto correlate venga contato per due volte e questo fa sì che il classificatore scelga un certo risultato basandosi proprio su questo conteggio errato.

Il problema di modelli come Naive Bayes ma anche come il decision tree è che sono modelli lineari che funzionano bene se i dati sono linearmente separabili mentre iniziano ad esserci dei problemi quando il dataset diventa più complesso e non è più linearmente separabile. Con un dataset non linearmente separabile avremmo bisogno di un algoritmo di classificazione non lineare che sia in grado di cogliere le feature di questi dati.

6.2 Linear versus Non Linear Algorithms

Abbiamo già nominato nel paragrafo precedente gli algoritmi lineari e quelli non lineari. Gli algoritmi lineari hanno le seguenti caratteristiche:

- Possono trovare un decision boundary che è lineare, quindi se sappiamo che i nostri dati sono linearly separable possiamo utilizzarli tranquillamente.
- Sono modelli semplici e usano pochi parametri.
- Se abbiamo dei dataset molto grandi, come ad esempio nel caso dell'NLP abbiamo dataset che solitamente non sono separabili e quindi non ci conviene utilizzare algoritmi lineari.

- Esempi di algoritmi lineari sono Naive Bayes e il Perceptron.

Gli algoritmi non lineari sono più complessi e hanno molti più parametri da prendere in considerazione ma le performance che otteniamo sono generalmente migliori. La distinzione tra algoritmi lineari e algoritmi non lineari possiamo applicarla anche alla multi class classification.

Tra i modelli che permettono di avere una divisione non lineare dei dati abbiamo le Neural Network. Le Neural network le possiamo vedere come un grafo al cui interno sono presenti delle computing units che prendono in input dei dati e restituiscono un output. L'architettura più semplice è quella delle feedforward neural network in cui i dati vanno solamente in una direzione. Spesso quando utilizziamo le neural network si utilizza la parola deep learning perchè queste reti neurali che creiamo sono spesso molto profonde e sono formate da molti livelli. Le neural network sono modelli molto più complessi e potenti dei classici algoritmi di classificazione lineare, quando le utilizziamo non andiamo neanche a indicare manualmente quali sono le feature da considerare perchè anche la scoperta delle feature viene effettuata dalla stessa neural network. Le Neural Network funzionano particolarmente bene se abbiamo tanti dati che possiamo utilizzare per eseguire il training del modello e per capire le feature del modello automaticamente.

6.3 Perceptron

Il perceptron è un algoritmo lineare per la classificazione, è uno dei più semplici e ci permette di eseguire la classificazione binaria. Possiamo considerare il perceptron come una rete neurale che però ha un singolo layer. Una delle cose positive è che l'aggiunta di nuovi dati è abbastanza facile e l'algoritmo quindi si evolve senza troppe difficoltà.

Il perceptron lo utilizziamo tramite un iperpiano che ci serve per dividere i nostri dati in due regioni. In particolare l'iperpiano lo definiamo

usando la seguente funzione:

$$wx + b = 0$$

Il nostro obiettivo è trovare l'iperpiano che mi divide meglio i due set di punti commettendo meno errori possibili, in particolare abbiamo che la funzione dell'iperpiano ha valore 0 sull'iperpiano stesso, ha valore minore di 0 sotto all'iperpiano e maggiore di 0 sopra all'iperpiano. Questo vuol dire che i punti li classifichiamo in base al valore che otteniamo considerando la loro posizione rispetto all'iperpiano.

Quando utilizziamo il Perceptron possiamo selezionare alcune possibili feature delle parole e rappresentare poi le parole utilizzando un one hot encoding, in questo caso si tratta di considerare delle feature che sono binarie.

word is capitalized (Trump)
word made of digits (2016)
all upper (USA)

tree = <0, 0, 0>
Dylan = <1, 0, 0>

Nell'esempio vediamo che tree ha 0 nella prima feature perchè non ha la prima lettera maiuscola mentre ha 1 nella prima feature della seconda parola perchè è una lettera maiuscola.

Per le feature che non sono binarie invece abbiamo ad esempio la rappresentazione Bag of Words nel senso che abbiamo il nostro testo e poi possiamo fare il conteggio delle parole che sono presenti all'interno del testo andando a creare un array in cui contiamo le occorrenze.

6.3.1 Funzionamento del Perceptron

Consideriamo l'algoritmo del Perceptron, il nostro input consiste in:

- Abbiamo un set di dati x_i .
- Abbiamo un set di classi, ad ognuno dei dati che abbiamo in input abbiamo assegnato una classe.

In questo caso noi stiamo considerando un classificatore binario quindi abbiamo che le classi possibili sono $+1, -1$. Il nostro obiettivo ora è trovare un decision boundary che deve essere lineare $wx + b$, in questa definizione dell'iperpiano abbiamo w e b che sono i parametri che ci permettono di dividere lo spazio in due sottospazi in modo da classificare producendo meno errore possibile.

La nostra classification rule quindi è la seguente:

$$y = \text{sign}(wx + b)$$

se $wx + b > 0$ allora $y = 1$ altrimenti $y = -1$. Per semplificare la formula $wx + b$ invece di tenere separati w e il bias b possiamo scriverli in un solo vettore W .

La parte di learning del perceptron funziona in questo modo: noi abbiamo i nostri parametri di partenza w e b , inizialmente possiamo scegliere un iperpiano random che prova a dividere i nostri dati.

Quello che si fa è prendere ognuno dei datapoint presenti all'interno del dataset, considerare la classificazione con l'iperpiano corrente e successivamente modificare i pesi dell'iperpiano in modo che la classifi-

cazione del punto sia corretta. Si va avanti con tutti i punti cercando di ottenere un iperpiano che me li classifichi bene il più possibile.

La learning rule che utilizziamo per modificare i pesi del perceptron è la seguente, preso un punto generiamo la corrispondente y_i^* e poi calcoliamo

$$\Delta w_i = \epsilon x_i (y_i - y_i^*)$$

questa Δ che abbiamo calcolato la utilizziamo per modificare i pesi in modo che il punto venga classificato correttamente (se prima era classificato in modo errato). Quando calcoliamo il Δ utilizziamo la ϵ che è il learning rate, questo mi serve per decidere quanto dell'errore va utilizzato nel momento in cui facciamo la correzione, l'altra cosa che utilizziamo è il x_i che è il punto che stiamo classificando male. Una volta calcolato il Δ possiamo calcolare il nuovo peso per quel punto specifico:

$$w_{i+1} = w_i + \Delta w_i$$

La correzione ci comporta uno spostamento dell'iperpiano che molto spesso è solamente una rotazione dell'iperpiano originale che ci possa permettere di separare meglio i dati.

Come derivare la delta rule

La delta Rule viene generata utilizzando il Gradient Descent, prendiamo la error function e calcoliamo il gradiente:

- Noi abbiamo una certa previsione $y^* = wx + b$, ammettiamo che questa sia errata rispetto alla previsione corretta.
- Calcoliamo l'errore

$$E = \frac{1}{2} \sum_n (y_n - y_n^*)^2$$

questo sarà un numero positivo e rappresenta la somma di tutte le istanze classificate male elevata poi al quadrato e divisa (per convenienza) per due.

- Ora calcoliamo la derivata di questo errore che abbiamo calcolato, possiamo utilizzare la chain rule, quindi calcoliamo prima la derivata dell'errore rispetto a y_n e poi la derivata rispetto a w_i :

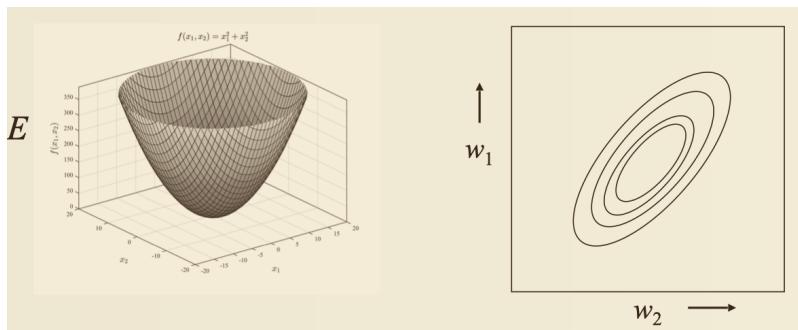
$$\frac{\nabla E}{\nabla W_i} = \frac{1}{2} \sum_n \frac{\nabla E_n}{\nabla y_n^*} \frac{\nabla y_n^*}{\nabla W_i} = - \sum_n (y_n - y_n^*) x_{n,i}$$

- Ora possiamo derivata la formula finale della delta:

$$\Delta w_i = -\epsilon \frac{\nabla E}{\nabla W_i}$$

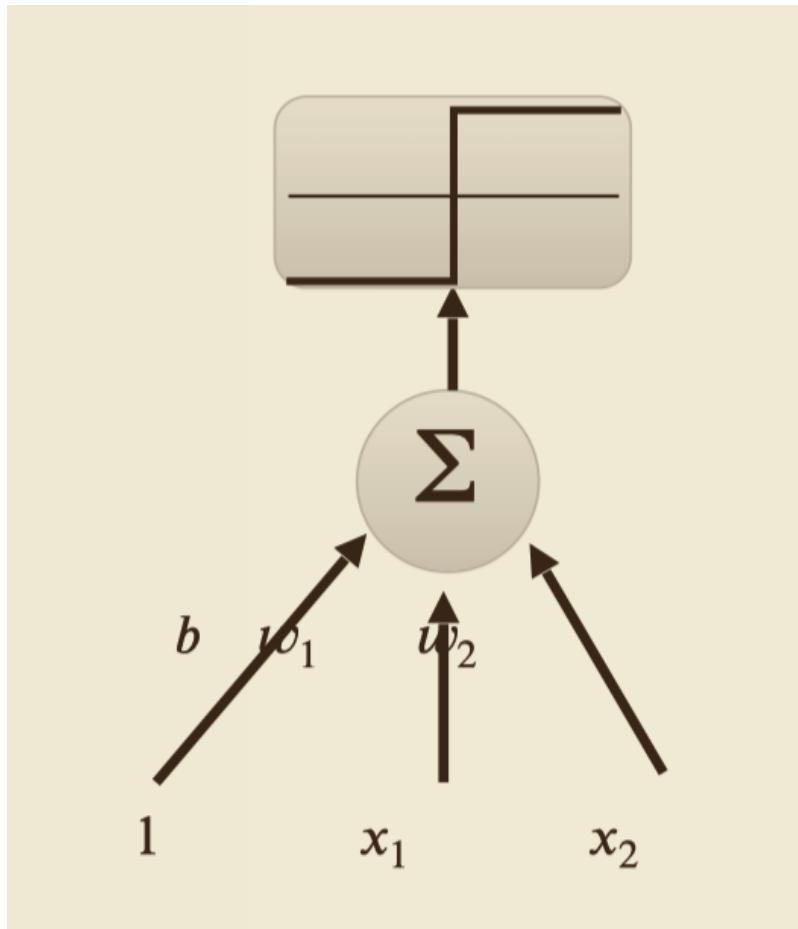
Tipicamente quello che si fa è considerare un solo esempio alla volta e calcolare la delta. L'aggiornamento andrebbe fatto in un ordine random in modo che possiamo convergere più velocemente.

Questo è un caso semplice di training utilizzando il gradient descent, l'obiettivo è cercare di minimizzare l'errore modificando i pesi in modo che l'errore diventi sempre minore.



6.3.2 Perceptron come single layer neural network

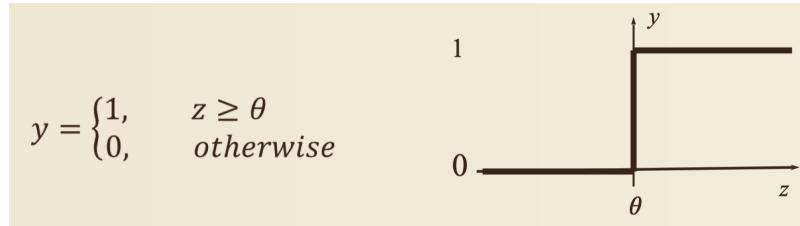
Come abbiamo già detto il perceptron può essere visto come una single layer neural network, questo perchè noi abbiamo in input una somma ovvero $wx + b$. Questa somma la diamo in pasto al nostro single layer ovvero al Perceptron che produce un output ovvero la classificazione del punto.



Per classificare in questo caso si prende la somma e si restituisce $+1$ se abbiamo la somma in input maggiore di un certo threshold e -1 se la somma in input è minore del threshold. La funzione che mi restituisce 1

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

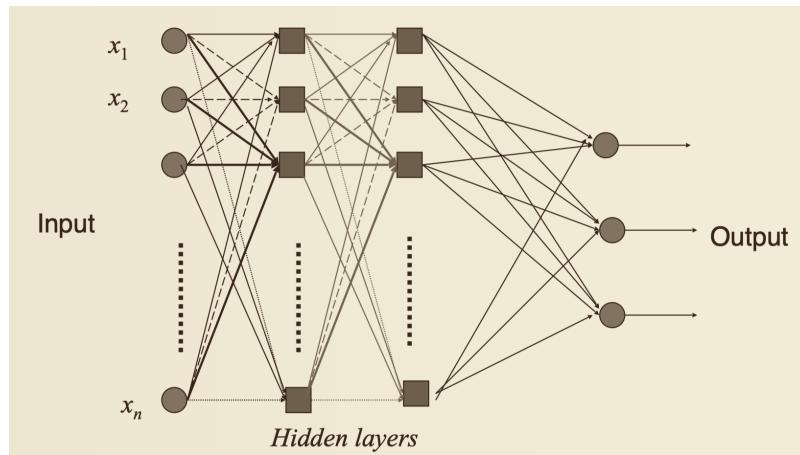
o -1 la possiamo definire e rappresentare graficamente in questo modo:



in questa rappresentazione della funzione abbiamo Θ che è il threshold.

6.4 MultiLayer Perceptron

Possiamo rendere il perceptron più complesso andando ad aggiungere in questa nostra rete neurale più di un singolo layer (aggiungiamo più perceptron layer).



Questa architettura ha delle proprietà interessanti:

- Non ci sono connessioni all'interno del singolo layer e ogni elemento presente in un layer è detto neurone.

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

- Non abbiamo una connessione diretta tra input e output layer.
- Ogni neurone del layer N è connesso con ogni neurone del layer $N+1$. Ogni layer fornisce dati solamente al layer successivo e tutti i dati che erano in input contribuiscono alla creazione dell'output del layer.
- Il numero di output non deve essere uguale al numero di input.
- Il numero di neuroni in ognuno dei layer è variabile.

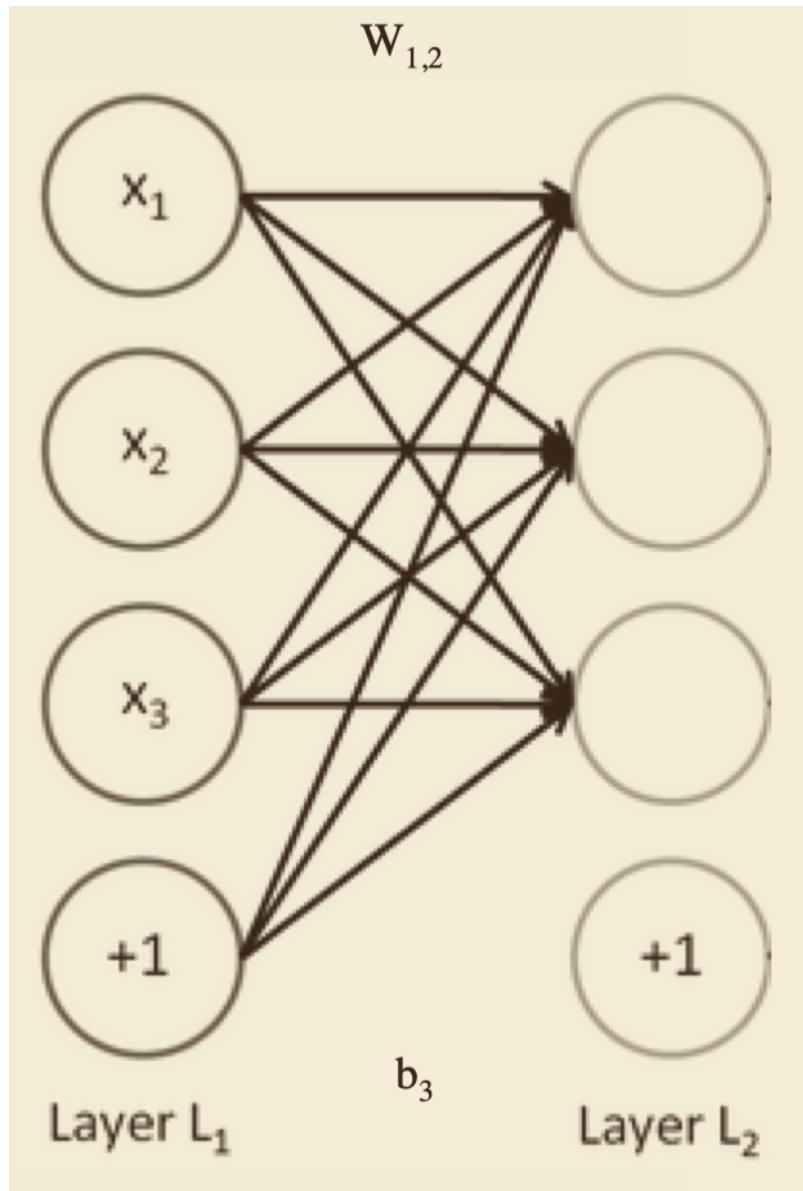
Il motivo per cui aggiungiamo i vari layer è aggiungere più complessità al nostro modello, ognuno di questi layer sarà in grado di gestire e riconoscere differenti feature presenti all'interno dei nostri dati andando quindi a creare un decision boundary che non sarà più lineare.

Questo che fino ad ora abbiamo chiamato Multi Layer Perceptron viene chiamato anche Deep neural network.

6.4.1 Architettura della Neural Network

Consideriamo una neural network e in particolare focalizziamoci su due layer completamente connessi:

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.



Qui quello che succede è che abbiamo per ognuno dei collegamenti tra il layer N e il layer $N + 1$ un peso w che moltiplichiamo per l'input relativo a quel collegamento aggiungendo poi il bias: $wx + b$. Il risultato di questa somma viene poi passato ad una activation function il cui

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

output sarà l'input del layer $N + 1$. L'activation function è la funzione che contribuisce a rendere non lineare il nostro modello, è necessario avere una funzione di questo genere all'interno della neural network perché altrimenti non riusciremmo ad ottenere una funzione non lineare per classificare i dati. Se non avessimo la funzione non lineare all'interno dei vari layer della rete neurale potremmo semplicemente combinare tutti i layer della rete neurale in un singolo layer perchè comunque la trasformazione che viene fatta è lineare.

Più layer aggiungiamo e meglio riusciamo ad approssimare le funzioni più complesse. C'è un teorema in particolare (Universal Approximation Theorem) che ci indica che con una rete neurale di 2 layer (input e hidden layer) possiamo approssimare qualsiasi funzione.

Prendendo questo teorema alla lettera avremmo un solo hidden layer in cui però ci sono tantissimi neuroni e questa soluzione non è molto pratica da utilizzare, questo è il motivo per cui si preferisce una deep neural network.

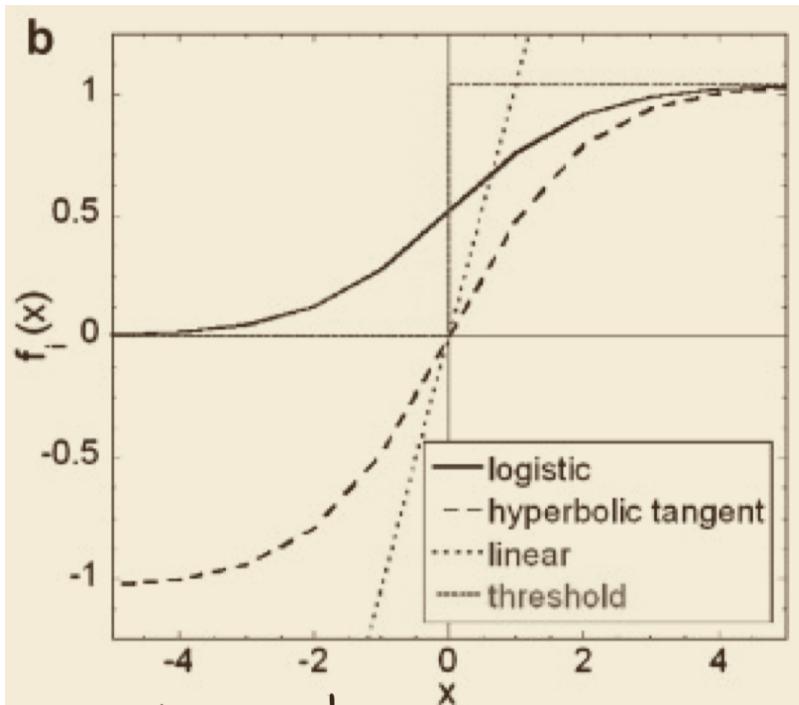
Nel caso del perceptron con il single layer l'Universal Approximation Theorem mi dice che possiamo approssimare la funzione e arrivare alla convergenza in tempo finito se i dati nel nostro dataset sono linearmente separabili. Anche in questo caso aggiungendo più layer possiamo diminuire la dimensione del singolo hidden layer.

Abbiamo varie possibili activation function che possiamo utilizzare:

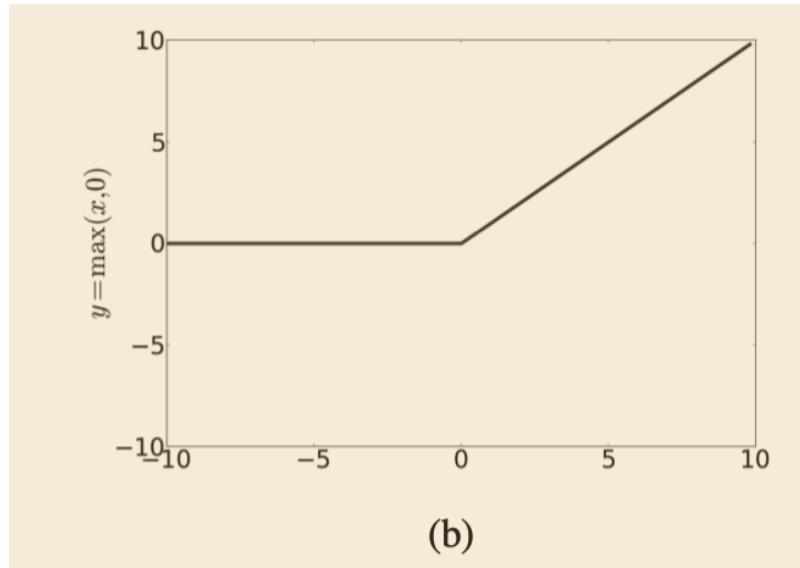
- Funzione lineare: la vediamo nel piano come una linea obliqua che divide il piano.
- Hyperbolic tangent
- Sigmoid function, è la funzione che, preso in input il valore me lo riduce in un range compreso tra 0 e 1.

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

La sigmoid function è anche differenziabile ed è utile che mi riduca l'output tra 0 e 1 anche per rendere meno dannosi gli outlier.



- Relu è l'activation function più semplice e probabilmente la più usata.



Se l'input è positivo restituisce lo stesso input come output, se invece è negativo allora restituisce 0:

$$y = \max(x, 0)$$

6.5 Softmax Classifier

6.5.1 Logistic Regression

Fino ad ora abbiamo sempre parlato di task di classificazione in cui distinguiamo tra varie possibili classi. Perchè non provare ad usare la logistic regression?

Avremmo bisogno di calcolare, dato un punto x , la probabilità che quel punto appartenga ad una certa classe y :

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

$$P(y|x) = \sum_{i=0}^N w_i f_i = w * f$$

Il problema è che il valore che noi calcoliamo wf è un qualcosa che potrebbe essere minore di 0 o compreso tra 0 e 1 ma anche maggiore di 0 quindi non è sicuramente una probabilità. Quello che possiamo fare per risolvere il problema e per ottenere una probabilità che poi possiamo utilizzare per classificare è usare la sigmoid function. Una volta eseguito il calcolo $w * f$ infatti possiamo applicare la sigmoid function che come abbiamo visto mi porta il valore risultante nel range 0, 1 quindi perfetto per la probabilità.

Quindi, se abbiamo un certo punto x come lo classifichiamo ammettendo di avere solamente due possibili classi?

$$y^* = \max_y P(y|x)$$

dobbiamo prendere il massimo delle varie probabilità che possiamo calcolare, quello che varia è il valore della y , se abbiamo due classi sarà ad esempio +1 o -1.

$$\begin{aligned}\hat{y} &= 1 \text{ iff } p(y=1|x) > p(y=-1|x) \\ \frac{p(y=1|x)}{p(y=-1|x)} &> 1 \\ \frac{p(y=1|x)}{1-p(y=1|x)} &> 1\end{aligned}$$

Se utilizziamo la sigmoid function riusciamo a riscrivere questa formula in questo modo:

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

$$\frac{\frac{e^{wx}}{1+e^{wx}}}{1 - \frac{e^{wx}}{1+e^{wx}}} = \frac{\frac{e^{wx}}{1+e^{wx}}}{\frac{1}{1+e^{wx}}} = e^{wx} > 1$$

in questo modo abbiamo la frazione tra due probabilità e poi facendo delle semplificazioni arriviamo al criterio utilizzabile per classificare un certo valore.

6.5.2 Multinomial Logistic Regression

Se abbiamo più di due sole classi dobbiamo utilizzare un sistema differente, non possiamo usare la sigmoid function come detto in precedenza ma dobbiamo usare la softmax function. La formula in questo caso è quella della distribuzione di probabilità per la classe C dati i dati X e diventa:

$$p(c|x) = \frac{e^{w_c c}}{\sum_{x^* \in C} e^{w_c x}} = softmax_c(w_c x)$$

in questo caso all'interno della formula dobbiamo eseguire la normalization perchè abbiamo bisogno di una distribuzione di probabilità.

Training e Classificazione

Ora che abbiamo la formula della Softmax che possiamo utilizzare per classificare un certo punto in una classe, possiamo vedere come si esegue il training del modello e la classificazione.

Noi abbiamo il nostro classificatore e lo vogliamo utilizzare per studiare il decision boundary, in questo caso però non vogliamo usare il segno dell'activation function per cercare di modificare i pesi ma vogliamo usare la softmax che abbiamo visto nel paragrafo precedente.

$$P(y|x) = \frac{e^{W_y x}}{\sum_c e^{w_c x}}$$

È importante che nella formula dividiamo per la somma di tutte le $e^{W_c x}$ perchè altrimenti non avremmo una probabilità di una classe, questo algoritmo è simile al perceptron ma all'interno utilizziamo una activation function che è la softmax. Se diciamo che $f_y = W_y x$ allora la formula diventa:

$$P(y|x) = \frac{e^{f_y}}{\sum_c e^{f_c}} = softmax(f_y)$$

Per ognuno degli esempi di training l'obiettivo è massimizzare la probabilità della classe che viene predetta (quella che ha probabilità massima) ovvero fare in modo che la y^* sia uguale alla y reale.

Invece di eseguire la massimizzazione della probabilità di $P(y|x)$ possiamo eseguire la minimizzazione di

$$-\log(P(y|x)) = -\log\left(\frac{e^{f_y}}{\sum_c f_c}\right)$$

Quando eseguiamo la minimizzazione della formula originale in cui aggiungiamo il meno davanti e applichiamo il logaritmo otteniamo sempre lo stesso risultato.

Quando abbiamo finito il training del modello possiamo calcolare la cross entropy.

Presi la distribuzione di probabilità reale p e la distribuzione di probabilità q che abbiamo trovato con il nostro modello, calcoliamo la cross entropy:

$$H(p, q) = \sum_c p(c) \log(q(c))$$

La distribuzione di probabilità è rappresentata da un vettore 1 hot encoding in cui abbiamo 1 nella posizione della classe reale e 0 altrove. L'obiettivo è massimizzare la likelihood che è esattamente la stessa cosa di minimizzare la cross entropy.

6.6 Traditional NLP vs Word Vectors

Nel Natural Language tradizionale abbiamo dei pesi $\Theta = [W_1, \dots, W_c] \in R^{Cd}$, il numero di questi parametri è abbastanza basso perchè dipende solo dalla quantità di categorie che abbiamo nel nostro problema di classificazione moltiplicato per la dimensione del vettore di parole che stiamo considerando.

Se invece vogliamo eseguire un task di classificazione utilizzando il deep learning e quindi utilizzando i Word Vectors dobbiamo eseguire non solo il training dei parametri che abbiamo indicato in precedenza ma anche il training di un set di parametri, uno per ogni elemento presente nel dizionario, quindi ora il set di parametri diventa: $\Theta = [W_1, \dots, W_c] \in R^{Cd+Vd}$.

6.7 Sperimentare con Keras

6.7.1 1 Layer Perceptron

Facciamo una prima prova con Keras, ammettiamo di voler creare un perceptron classifier con un solo layer. Ci sono alcuni parametri che possiamo decidere:

- Il numero di parole da considerare nel nostro modello
- La lunghezza massima delle frasi che dobbiamo classificare
- La dimensione dei batch che usiamo per fare il training

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

- La dimensione degli embedding (ricordare che avremmo ad esempio una rappresentazione iniziale per le nostre parole presenti nelle frasi e noi vogliamo però ottenere una rappresentazione più piccola, gli embedding)

Dal punto di vista del codice abbiamo:

```
from keras.models import Sequential
from keras.layers import Dense, Activation, Input,
from keras.layers import Flatten, Embedding

# Parameters
num_words = 5000
maxlen = 80 # max sentence length
batch_size = 32
embedding_dims = 50
```

Ora passiamo alla creazione vera e propria della nostra rete, i componenti sono i seguenti:

- Il primo layer del nostro modello è la 1 hot representation di ognuna delle parole, abbiamo bisogno di una rappresentazione dell'input, utilizziamo una matrice $5000 * 50$ e possiamo usare il parametro *embedding_initializer* che è necessario per inizializzare la matrice, noi in questo caso usiamo la matrice identità, non è un vero e proprio embedding ma è solamente una rappresentazione 1 hot encoding.
- Aggiungiamo poi il Flatten layer, qua noi prendiamo in input la matrice $5000 * 50$ e andiamo a prendere da questa matrice le colonne corrispondenti alle parole che abbiamo nella frase che stiamo prendendo in considerazione. Considerando che noi abbiamo frasi di massimo 80 parole quello che possiamo fare è prendere 80 vettori che quindi avranno al loro interno 5000 elementi (all'interno abbiamo una one hot representation). Una volta che abbiamo preso questi 80 vettori di 5000 elementi li concateniamo in un unico vettore che ha al suo interno $80 * 5000 = 40.000$ elementi.

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

- Aggiungiamo il layer dense che esegue il prodotto wx restituendo in output un singolo valore.
- Aggiungiamo il layer con l'activation function e in questo caso scegliamo la sigmoid activation function in modo da portare il valore in output tra 0 e 1.

Il codice per questo modello è il seguente:

```
model = Sequential()
# trick to build a one-hot representation
model.add(Embedding(5000, # size of vocabulary
                    50, # embeddings size
                    input_length=80, # max sentence length
                    embeddings_initializer='identity',
                    trainable=False)) # do not train the embeddings
model.add(Flatten())
model.add(Dense(1)) # single value
model.add(Activation('sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

Questo modello esegue due operazioni:

- Una operazione di forward
- Una operazione di backward, dobbiamo calcolare il gradiente dell'errore dall'output, l'errore è calcolato utilizzando una error fuction, in questo caso è la Binary Cross Entropy. Questa fase di backward viene eseguita in modo automatico dal modello usando una differenziazione automatica, non abbiamo bisogno di tutta la parte matematica se ci limitiamo ad "assemblare" le varie parti del modello. Durante la backpropagation oltre a calcolare le derivate aggiorniamo i pesi.

Per capire se stiamo lavorando bene dobbiamo anche aggiungere come parametro del nostro modello (quando lo compiliamo) un parametro che è *metrics* e indica il tipo di metrica che utilizziamo per calcolare le performance del modello, ad esempio in questo caso è *accuracy*.

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

Per caricare i dati che vengono utilizzati all'interno del modello e per creare i set di dati di training e di test possiamo utilizzare il seguente codice:

```
from keras import imdb
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=max_words)

# pad sequences to the same length
from keras.preprocessing import sequence
x_train = sequence.pad_sequences(x_train, maxlen=maxlen)
x_test = sequence.pad_sequences(x_test, maxlen=maxlen)

# vocabulary
word_index = imdb.get_word_index()
```

Per eseguire il training del modello:

```
model.fit(x_train, y_train,
           batch_size=32,
           epochs=10,
           validation_data=(x_test, y_test))
```

Per testare il modello:

```
predictions = model.predict(x_test)

errors = y_test - predictions.round().flatten()
accuracy = len(errors[errors==0])/len(errors)
```

Notare alcuni dettagli dal Jupyter Notebook (MLP):

- Quando carichiamo i dati dal nostro dataset e chiamiamo la funzione *load_data* utilizziamo come parametro *index_from = 0* questo serve perchè altrimenti il conteggio parte da 3.
- Quando creiamo il set di dati di training e di test utilizziamo la funzione *pad_sequence* perchè noi abbiamo limitato la lunghezza dei testi che stiamo considerando ma potremmo trovarci con alcune review che sono più corte, in questo caso viene appunto aggiunto il padding per fare in modo di avere sempre 80 parole in ognuna di queste review, il padding viene aggiunto all'inizio ma sarebbe meglio averlo alla fine.

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

- Le parole che sono presenti all'interno del nostro dataset vengono contate e poi ordinate in base al numero di occorrenze. Questo è comodo perché quando riduciamo il numero di parole presenti nel dataset possiamo mantenere nel dataset quelle più utilizzate. In questo modo anche nelle review che analizziamo avremo comunque le parole più importanti che rimangono.
- Il modello che abbiamo creato ha le seguenti caratteristiche:

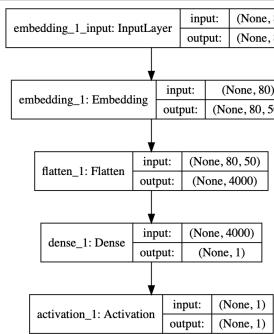
Layer (type)	Output Shape	Param #
<hr/>		
embedding_1 (Embedding)	(None, 80, 50)	250000
flatten_1 (Flatten)	(None, 4000)	0
dense_1 (Dense)	(None, 1)	4001
activation_1 (Activation)	(None, 1)	0
<hr/>		
Total params: 254,001		
Trainable params: 254,001		
Non-trainable params: 0		

Il layer embedding ha 250.000 parametri perchè al suo interno abbiamo una matrice di dimensioni $5000 * 50$ con 250.000 parametri. L'output di questo primo layer sarà una matrice di dimensione $80 * 50$. Il layer che esegue l'operazione flatten prende le varie colonne della matrice e le concatena in un'unica lista risultato. Nel layer dense assegnamo ad ognuno degli elementi dell'array che abbiamo preso in input (è un array di 4000 elementi perchè la matrice era $80 * 50$) assegnamo un peso differente, in tutto in questo layer avremo 4001 parametri, l'1 è dovuto al fatto che c'è anche il bias. L'output del dense level è un singolo numero che viene ottenuto moltiplicando i parametri con i dati che abbiamo nel nostro testo che stiamo analizzando, nel layer successivo verrà applicata la sigmoid function. In tutto i parametri del modello su cui dobbiamo fare il training sono 254.001.

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

Possiamo anche andare a mostrare graficamente l'architettura della neural network:

```
from IPython.display import SVG
from keras.utils.vis_utils import model_to_dot
SVG(model_to_dot(model, show_shapes=True, dpi=65).create(prog='dot', format='svg'))
```



6.7.2 Perceptron con Word Embedding - Glove

Una alternativa che abbiamo è quella di utilizzare un embedding pre-trained che è il glove embedding. In questo caso l'inizializzazione per l'embedding è differente perchè noi forniamo un embedding su cui è già stato eseguito il training.

Si tratta in questo caso di un modello più sofisticato rispetto al precedente perchè l'inizializzazione degli embedding ora non è più random come in precedenza ma partiamo con l'embedding di Glove, le parole che non vengono trovate nell'embedding di Glove avranno un peso random all'inizio.

Rispetto al modello precedente quello che facciamo è utilizzare la convoluzione infatti aggiungiamo il layer "GlobalMaxPool1D". Con la convoluzione quello che facciamo è prendere 5 elementi alla volta moltiplicarli con il vettore di pesi e ottenere un numero, poi ripetiamo la stessa cosa traslando dall'inizio del vettore alla fine. Quindi, il layer *max_pooling* prende in input una matrice 80×50 e restituisce in output 50 valori, poi il primo layer "Dense" fa una seconda trasformazione e mi

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

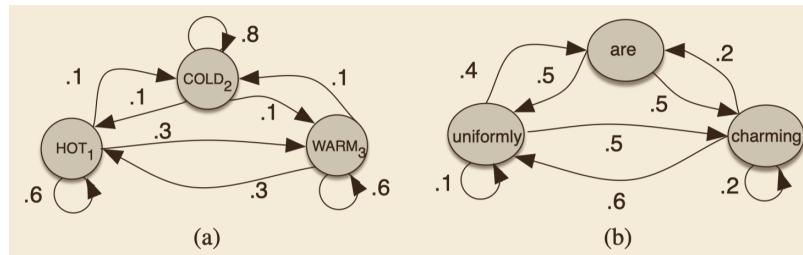
riduce questi 50 valori in soli 10 valori, l'ultimo layer Dense mi restituisce solamente un singolo valore.

Chapter 7

Lesson 6 - Part of Speech Tagging

7.1 Markov Chains

Il Markov chain è un modello stocastico che ci fornisce delle informazioni riguardanti la probabilità di particolari sequenze di random variables.



Più formalmente possiamo definire un Markov Chain in questo modo:

- Creiamo un grafo in cui abbiamo un set di N stati che rappresentano le varie random variables.
- I vari N stati sono collegati tra loro con archi a cui viene associata

una transition probability. Queste probabilità di passare da uno stato al successivo sono memorizzate in una probability matrix.

- Abbiamo degli stati iniziali da cui partiamo, a questi stati viene associata una distribuzione di probabilità che mi indica la probabilità di partire in ognuno degli stati. Qui avremmo: $\pi_i = P(q_1 = i)$ che quindi comporta che

$$\sum_{j=1}^N \pi_j = 1$$

Notare che i vari archi che escono da un nodo e che sono associati ad una certa probabilità devono avere come somma 1, lo stesso discorso vale per le probabilità di partenza che associamo ai vari nodi di partenza della rete.

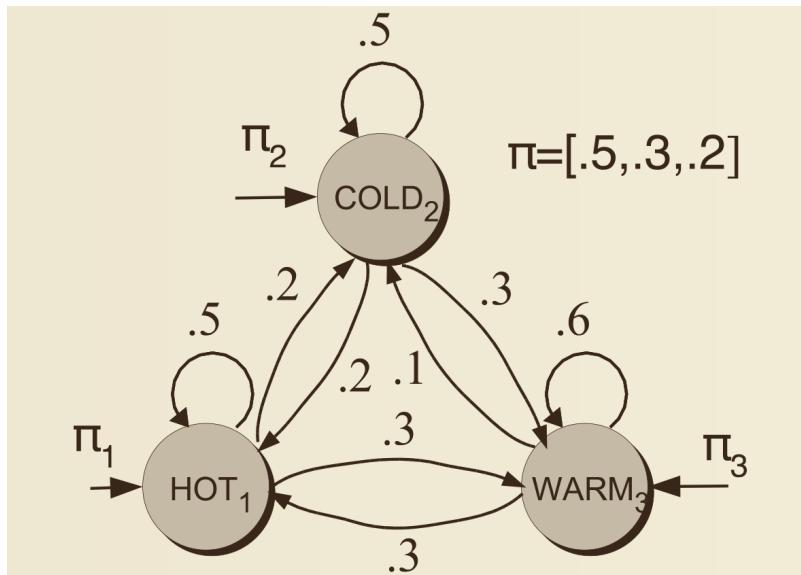
Una cosa importante da considerare è l'assunzione che viene fatta con i Markov Model, presa una sequenza di stati q_1, \dots, q_n abbiamo che quando vogliamo predire lo stato futuro conoscendo gli stati passati, dobbiamo prendere in considerazione solamente l'ultimo stato e non tutti quelli precedenti. Questo lo possiamo scrivere in formula e quindi abbiamo:

$$P(q_i = a | q_1, \dots, q_{i-1}) = P(q_i = a | q_{i-1})$$

ovvero la probabilità di raggiungere un certo stato dipende solamente dallo stato precedente.

Il Markov Chain può rappresentare un bigram language model con i vari archi che rappresentano la probabilità di trovare una certa parola dopo quella che stiamo considerando nello stato corrente.

Vediamo un esempio di Markov Chain:



vogliamo calcolare la probabilità di avere 4 giorni di caldo ovvero vogliamo calcolare la probabilità $P(3, 3, 3, 3)$. Per calcolarla partiamo consideriamo la probabilità di partire dallo stato 3 e poi di rimanerci, quindi abbiamo: $P(3, 3, 3, 3) = \pi_3 a_{33} a_{33} a_{33} = 0.2 * (0.6)^3$.

7.2 Hidden Markov Models

Fino ad ora abbiamo visto solamente le Markov Chains che sono utili quando dobbiamo calcolare la probabilità di una serie di eventi che sono osservabili. Molto spesso però gli eventi che ci interessano sono nascosti e non possiamo osservarli direttamente.

Prendiamo l'esempio in cui consideriamo una parola come output di un Hidden Markov Model, in questo caso noi conosciamo la parola finale ma non conosciamo il processo che mi porta alla generazione di questa parola perchè tutta la grammatica che c'è dietro è del tutto nascosta. La generazione di questa parola segue delle regole che noi non conosciamo e il Markov Chain non può aiutarci perchè è del tutto trasparente, quindi abbiamo bisogno di un modo per descrivere l'hidden state.

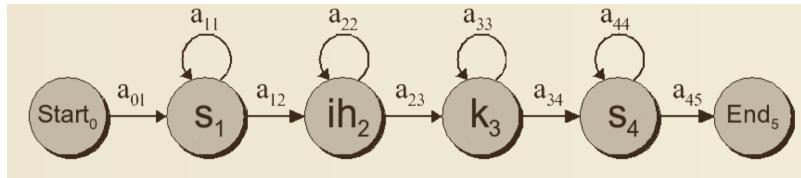
Un Hidden Markov Model ci permette di lavorare sia con eventi osservati (ad esempio le parole che sono in input) sia con eventi che sono nascosti (che vengono chiamati hidden, ad esempio potrebbero essere le part of speech tags di un testo) e viene specificato dai seguenti componenti:

- Abbiamo un set di N stati.
- Abbiamo una transition matrix con le varie probabilità di muoverci da uno stato all'altro.
- Abbiamo una sequenza di osservazioni prese da un dizionario V, anche queste osservazioni vengono generate in modo stocastico. Queste osservazioni sono quelle che io ho a disposizione all'inizio e che utilizzo per generare poi l'output che sto cercando.
- Abbiamo una sequenza di probabilità delle osservazioni, ognuna esprime la probabilità che l'osservazione i sia stata generata dallo stato q_i .
- Abbiamo una distribuzione di probabilità iniziale.

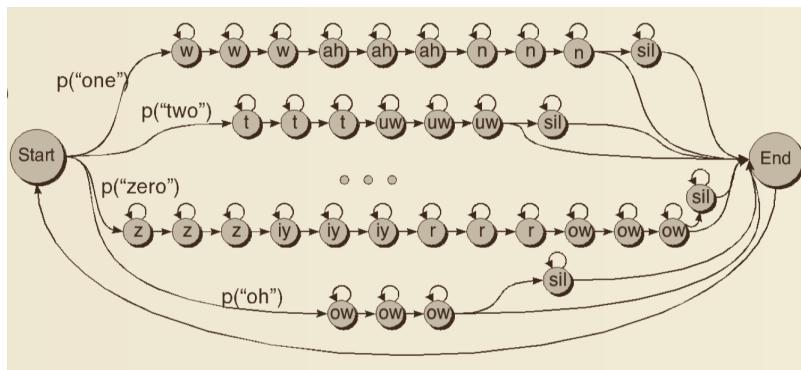
Un Hidden Markov model fa due assunzioni, la prima è la stessa che abbiamo detto prima con il Markov Model, la seconda è l'assunzione sull'indipendenza dell'osservazione in output o_i che dipende solamente dallo stato che produce l'osservazione e non da altri stati o da altre osservazioni:

$$P(o_t | O_1^{t-1}, q_1^t) = P(o_t | q_t)$$

Possiamo vedere un esempio di HMM per produrre la parola "six", in questo caso l'output è il suono che produciamo pronunciando la parola six e gli stati nascosti sono parti di questa parola.

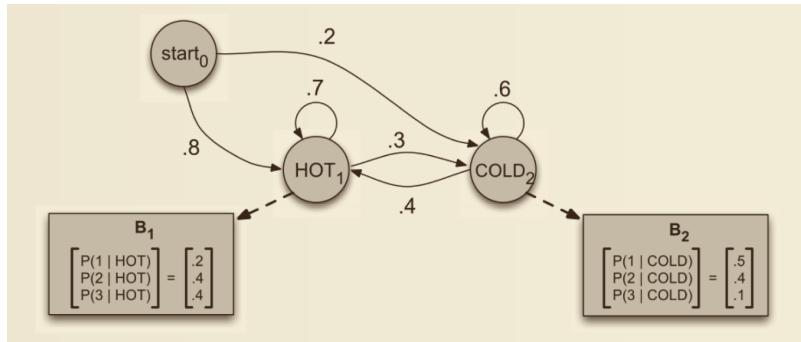


Nel seguente esempio invece possiamo vedere un esempio di HMM in cui abbiamo vari possibili suoni e per ognuno la probabilità di produrre quel suono (il suono indica una parola che stiamo pronunciando).



Quindi quello che si fa è considerare una serie di stati nascosti che producono quel suono.

Possiamo vedere un esempio più complesso in cui cerchiamo di capire la temperatura di una città basandoci sui dati riguardanti la vendita dei gelati (quanti gelati sono stati mangiati da una persona in ogni giorno dell'estate). Per questo esempio noi abbiamo bisogno di alcune osservazioni (ad esempio potremmo avere una lista del tipo 1, 2, 1, 3, 2, 3, 2) e vogliamo produrre una sequenza che mi indichi se la giornata relativa a quella osservazione è stata calda o fredda basandomi sul numero di gelati mangiati.



Nella figura abbiamo due probabilità che sono B_1 e B_2 che sono le probabilità di mangiare un determinato numero di gelati quando è caldo e quando è freddo.

Possiamo avere vari tipi di strutture per i Markov Models, in particolare abbiamo:

- Una organizzazione della struttura del modello che va da sinistra a destra
- Una organizzazione ergodica in cui il grafo deve convergere ad una certa stabilità ergodica.

Ci sono tre problemi principali da risolvere per gli HMM:

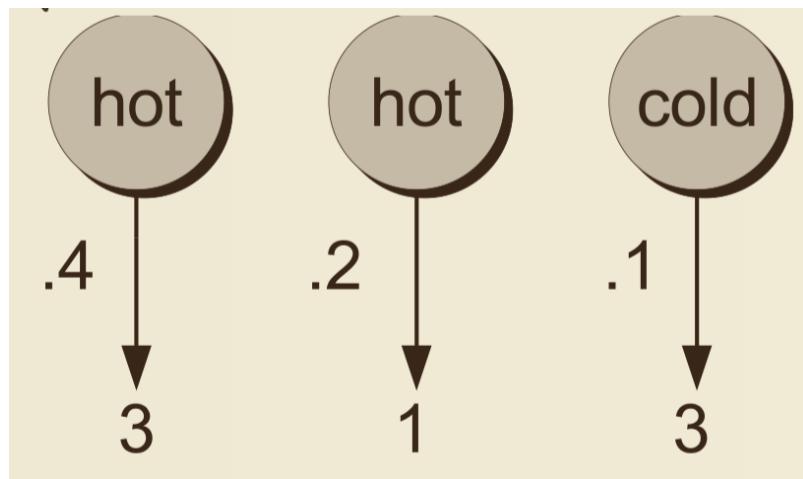
- Abbiamo la parte di valutazione in cui calcoliamo la Likelihood. Data la sequenza di osservazioni O e il modello HMM $\Theta = (A, B)$ dobbiamo calcolare efficientemente $P(O|\Theta)$.
- La fase di decoding invece consiste nel partire da una serie di osservazioni $O = (o_1, o_2, \dots, o_n)$ e dal modello HMM $\Theta = (A, B)$ per poi generare la migliore sequenza di stati nascosti corrispondente $Q = (q_1, \dots, q_n)$.
- La fase di learning consiste nell'aggiustare i parametri del modello ovvero i parametri A e B per massimizzare $P(O|\Theta)$

7.2.1 Calcolo della Likelihood

Quello che vogliamo fare nella prima parte di valutazione, data la nostra sequenza di osservazioni, è calcolare la likelihood di questa sequenza ovvero data la HMM $\lambda = (A, B)$ e la sequenza di osservazioni O vogliamo determinare $P(O|\lambda)$. Quello che dobbiamo fare è combinare la probabilità dell'emissione di un certo valore con la probabilità di spostarsi in un certo stato. Ad esempio potremmo voler calcolare $P(313|HHC)$. In una classica Markov Chain possiamo calcolare la probabilità di 313 semplicemente seguendo gli stati labellati con 313, nel caso degli Hidden Markov Model però è più complesso perché non conosciamo la sequenza di stati nascosti.

Per calcolare questa likelihood possiamo considerare come indipendenti le varie probabilità e quindi otteniamo:

$$P(313|HHC) = P(3|hot)P(1|hot)P(3|cold) = 0.4 * 0.2 * 0.1$$



In realtà per eseguire il calcolo completo dovremo anche considerare la probabilità di spostarsi da uno stato all'altro, quindi in realtà il calcolo completo diventa:

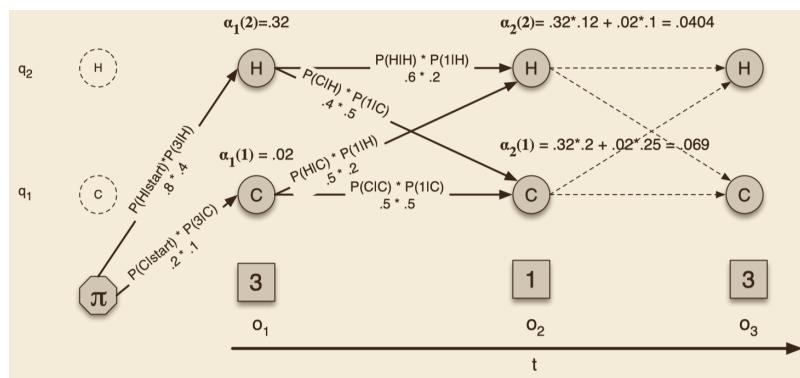
$$P(313|HHC) = P(3|hot)P(1|hot)P(3|cold)P(hot|start)P(hot|hot)P(cold|hot) = 0.4 *$$

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

Per avere una stima complessiva dobbiamo considerare tutte le possibili sequenze di H e C ovvero dato che abbiamo 2 sole possibili classi abbiamo $2^3 = 8$ possibili sequenze da considerare e per cui calcolare la probabilità.

Dato che vogliamo calcolare la probabilità di ottenere quella sequenza di osservazioni allora dobbiamo sommare le probabilità che otteniamo per ciascuna delle possibili sequenze di stati, quindi sommiamo 8 probabilità. Tutta questa operazione è chiaramente non fattibile perché il numero di probabilità che dobbiamo calcolare e poi sommare cresce in modo esponenziale, supponiamo di avere N stati possibili e una sequenza di T osservazioni allora avremmo da calcolare N^T probabilità. Questo vuol dire che questo algoritmo avrebbe una complessità $O(N^T)$.

Dobbiamo cercare di semplificare questo procedimento e per farlo possiamo fare alcune osservazioni, vediamo infatti che nelle varie sequenze di stati che consideriamo ci sono delle ripetizioni, possiamo evitare di ripetere lo stesso calcolo e possiamo ridurre in questo modo la complessità da esponenziale a polinomiale. Si tratta di una sorta di algoritmo di programmazione dinamica chiamato "Forward Algorithm" in cui utilizziamo una tabella per memorizzare i valori intermedi che calcoliamo e poi andiamo a calcolare la probabilità della sequenza di osservazioni.



L'algoritmo calcola le probabilità delle varie osservazioni sommando le probabilità dei singoli stati nascosti che possono generare la sequenza

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

finale di osservazioni, tutto il calcolo viene effettuato in modo molto efficiente perchè si riesce a comprimere ognuno di questi percorsi in un singolo "forward trellis".

Ogni cella di questo forward algorithm rappresenta un trellis $a_t(j)$ ovvero rappresenta la probabilità di trovarci nello stato j dopo aver visto le prime t osservazioni. Il valore di $a_t(j)$ viene calcolato sommando tutte le probabilità di ognuno di questi percorsi che possono portarci a questa cella, in ogni cella abbiamo:

$$\alpha_t(j) = P(o_1, \dots, o_t, q_t = j | \lambda)$$

Consideriamo la figura precedente e ammettiamo di voler calcolare $\alpha_2(2)$, ovvero la probabilità di essere al tempo 2 nello stato 2 sapendo che fino ad ora abbiamo generato l'osservazione 31. Quello che dobbiamo fare è estendere la probabilità α dello step 1 tramite due percorsi, il primo percorso è quello che va da C a H e questo contribuisce con la seguente formula:

$$\alpha_1(1) * P(H|C) * P(1|H)$$

ovvero consideriamo il valore che abbiamo calcolato al passaggio precedente che è $\alpha_1(1)$ e lo moltiplichiamo con la probabilità di muoversi da C ad H trovando un 1 nella serie di osservazioni. Il risultato di questo primo contributo è $0.02 * 0.2 * 0.5 = 0.02 * 0.1$.

Il secondo percorso che porta ad H è quello che va da H a H e quindi in questo caso abbiamo un calcolo simile ovvero:

$$\alpha_1(2) * P(H|H) * P(1|H)$$

questo ci fornisce il seguente risultato: $0.32 * 0.6 * 0.2 = 0.32 * 0.12$.

Ora che abbiamo i due risultati intermedi possiamo sommarli per ottenere $\alpha_2(2) = 0.32 * 0.12 + 0.02 * 0.1$.

In questo modo possiamo ridurre il numero di operazioni che svolgiamo per calcolare la somma finale di tutte le varie probabilità, quando

siamo arrivati negli stati finali sommiamo tutte le probabilità ottenute e abbiamo quindi trovato la probabilità della sequenza 313.

L'algoritmo che mi permette di fare tutto ciò è il forward algorithm e lo possiamo vedere nella seguente immagine:

```

function FORWARD(observations of len  $T$ , state-graph of len  $N$ ) returns forward-prob
    create a probability matrix forward[ $N+2, T$ ]
    for each state  $s$  from 1 to  $N$  do                                ; initialization step
         $forward[s, 1] \leftarrow a_{0,s} * b_s(o_1)$ 
    for each time step  $t$  from 2 to  $T$  do                ; recursion step
        for each state  $s$  from 1 to  $N$  do
             $forward[s, t] \leftarrow \sum_{s'=1}^N forward[s', t-1] * a_{s', s} * b_s(o_t)$ 
     $forward[q_F, T] \leftarrow \sum_{s=1}^N forward[s, T] * a_{s, q_F}$            ; termination step
    return forward[ $q_F, T$ ]

```

7.2.2 Decoding: Viterbi Algorithm

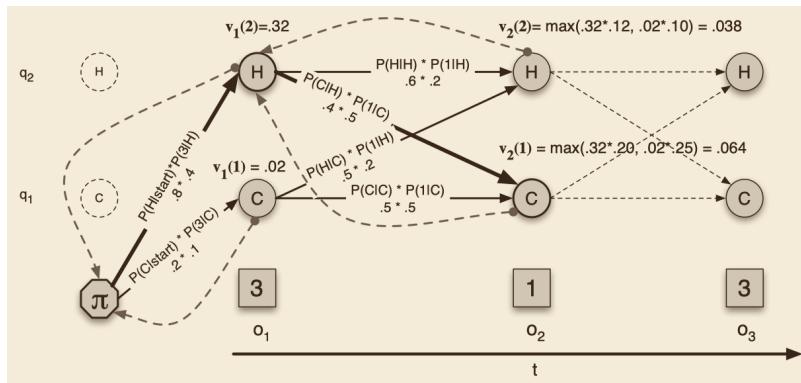
Come possiamo capire quanto è probabile una certa sequenza di osservazioni? L'obbiettivo della fase di decoding è trovare la migliore sequenza di hidden state basandoci sulle varie probabilità e scegliendo la probabilità più alta.

Teoricamente per il calcolo della probabilità di queste sequenze di hidden state dovremmo calcolare la probabilità di un certo hidden state date le osservazioni e quindi sarebbe necessario generare tutti i possibili hidden state e calcolare tutte le probabilità. Questo chiaramente è esponenziale e quindi non è fattibile specialmente se cresce il numero di stati possibili.

Si adotta quindi una strategia simile a quella che abbiamo utilizzato in precedenza con il forward algorithm. In questo caso l'algoritmo prende il nome di Viterbi Algorithm ed anche in questo caso abbiamo una sorta di algoritmo di programmazione dinamica. Anche qua abbiamo una sorta di grafo con i vari stati possibili e calcoliamo le varie

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

probabilità di muoversi da uno stato all'altro. Anche in questo caso ci muoviamo da sinistra a destra ma abbiamo una differenza, infatti ora per calcolare il valore associato ad una cella non facciamo la somma di tutti i possibili percorsi che arrivano in quel nodo del grafo ma prendiamo solamente la probabilità massima tra quelle dei vari percorsi che arrivano in quel nodo. La seconda differenza rispetto all'algoritmo forward è che ora siamo interessati al percorso di hidden state che ha associato la probabilità più alta e quindi è necessario mantenere all'interno del grafo un backpointer ovvero per ogni nodo abbiamo un puntatore al nodo che ho usato per arrivare in quel nodo.



Questi backpointers ci permettono di tornare dal nodo finale a quello iniziale seguendo il percorso che mi genera la probabilità massima.

L'algoritmo completo di Viterbi è il seguente:

```

function VITERBI(observations of len  $T$ , state-graph of len  $N$ ) returns best-path
    create a path probability matrix  $viterbi[N+2,T]$ 
    for each state  $s$  from 1 to  $N$  do ; initialization step
         $viterbi[s,1] \leftarrow a_{0,s} * b_s(o_1)$ 
         $backpointer[s,1] \leftarrow 0$ 
    for each time step  $t$  from 2 to  $T$  do ; recursion step
        for each state  $s$  from 1 to  $N$  do
             $viterbi[s,t] \leftarrow \max_{s'=1}^N viterbi[s',t-1] * a_{s',s} * b_s(o_t)$ 
             $backpointer[s,t] \leftarrow \operatorname{argmax}_{s'=1}^N viterbi[s',t-1] * a_{s',s}$ 
         $viterbi[q_F,T] \leftarrow \max_{s=1}^N viterbi[s,T] * a_{s,q_F}$  ; termination step
         $backpointer[q_F,T] \leftarrow \operatorname{argmax}_{s=1}^N viterbi[s,T] * a_{s,q_F}$  ; termination step
    return the backtrace path by following backpointers to states back in
          time from  $backpointer[q_F,T]$ 

```

7.2.3 Training dell'HMM

L'ultimo problema da risolvere in una HMM è la fase di Training, qua dobbiamo stimare e ottenere i migliori parametri possibili che, in questo caso, sono A e B. A è il parametro che mi indica la probabilità di spostarmi da uno stato all'altro mentre B indica la probabilità di trovarsi in un certo stato data l'osservazione.

Per eseguire il training di questo algoritmo possiamo utilizzare due metodi che sono il forward-backward o il Baum-Welch (un algoritmo che usa il metodo Expectation Maximization). In particolare Baum-Welch essendo un algoritmo EM è un algoritmo iterativo che calcola una stima iniziale della probabilità e poi utilizza questa stima per calcolare una stima ancora migliore fino a quando non otteniamo la migliore stima possibile. Per quanto riguarda la fase di massimizzazione quello che si fa è calcolare la forward probability per ognuna delle osservazioni e successivamente si divide questa probabilità tra i vari percorsi che hanno contribuito a questa forward probability. Per capire questo algoritmo è necessario definire una backward probability ovvero la probabilità di vedere l'osservazione del tempo $t + 1$ dato che al tempo t eravamo nello stato t .

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

7.3 HMM per Part of Speech Tagging

Con Part of Speech tagging intendiamo il processo di assegnare un marker ad ognuna delle parole nel testo che abbiamo in input. Il marker che vogliamo mi permette di classificare ciascuna parola "grammaticalmente" marcando ad esempio una parola come nome o come verbo.

Questa fase di tagging serve per disambiguare, molto spesso infatti abbiamo delle parole che sono ambigue e che potrebbero essere in alcuni casi un nome e in altri casi un verbo. L'obiettivo del Part of Speech Tagging è quello di disambiguare le varie parole assegnando il marker corretto. In alcuni casi però ci sono delle parole che sono sempre e comunque ambigue, l'idea in questo caso è quella di classificarle con il marker più frequente all'interno del corpus di training.

Gli algoritmi che eseguono la part of speech tagging sono classificatori che presa una certa osservazione considerano il contesto in cui si trova e assegnano il marker corrispondente. Questo task di classificazione è un task adeguato per un sequence classifier come lo sono ad esempio gli HMM, qua noi avremmo come osservazione una sequenza di parole che formano una frase e poi avremmo come output la sequenza di tag che meglio corrispondono alle parole osservate (ovvero la sequenza che ha la probabilità più alta). Più formalmente vorremmo il set di tag t_1, \dots, t_n tale che venga massimizzato $P(t_1, \dots, t_n | w_1, \dots, w_n)$ ovvero:

$$t_1^n = \operatorname{argmax}_{t_1^n} P(t_1^n | w_1^n)$$

dove la t_1^n è una sequenza di tag che massimizzano la sequenza di testo.

L'equazione $t_1^n = \operatorname{argmax}_{t_1^n} P(t_1^n | w_1^n)$ ci garantisce che abbiamo la migliore sequenza di tag, come possiamo fare a calcolare questo valore? L'idea è di utilizzare la Bayes Rule per trasformare questa probabilità in modo che sia più semplice da calcolare, calcolare direttamente la formula originale richiederebbe tra l'altro anche più dati.

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

Sapendo che

$$P(x|y) = \frac{P(y|x)P(x)}{P(y)}$$

allora possiamo dire che:

$$t_1^n = \operatorname{argmax}_{t_1^n} \frac{P(w_1^n|t_1^n)P(t_1^n)}{P(w_1^n)}$$

quindi in pratica abbiamo invertito la probabilità (ricordiamo che w_1^n è la sequenza di parole mentre la t_1^n indica la sequenza di tags). Dato che stiamo cercando la t_1^n che massimizza la nostra probabilità, il denominatore non contiene questo termine e quindi è solamente una costante che possiamo eliminare, quindi la nostra probabilità da massimizzare diventa:

$$t_1^n = \operatorname{argmax}_{t_1^n} P(w_1^n|t_1^n)P(t_1^n)$$

All'interno della formula sopra abbiamo la $P(w_1^n|t_1^n)$ che è la likelihood mentre la $P(t_1^n)$ indica la prior probability.

Ora si fanno due semplificazioni perchè vengono fatte due assunzioni, la prima assunzione è che la probabilità di trovare una parola dipende solamente dal suo tag ed è del tutto indipendente dalle parole che sono intorno e dai relativi tag, questa assunzione è quella fatta con Naive Bayes ovvero possiamo dire che:

$$P(w_1^n|t_1^n) \approx \prod_{i=1}^n P(w_i|t_i)$$

La seconda assunzione invece è la Markov assumption che mi dice che la probabilità di un tag dipende solamente dal tag precedente e non da tutti gli altri che sono ancora prima.

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

$$P(t_1^n) \approx \prod_{i=1}^n P(t_i|t_{i-1})$$

Quindi ora la nostra probabilità complessiva la possiamo riscrivere come:

$$t_1^n = argmax_{t_1^n} P(t_1^n|w_1^n) = argmax_{t_1^n} \prod_{i=1}^n P(w_i|t_i)P(t_i|t_{i-1})$$

all'interno di questa formula in particolare abbiamo che $P(w_i|t_i)$ è la probabilità di avere una certa osservazione dato che sono in un certo stato (emission probability) mentre la $P(t_i|t_{i-1})$ indica la probabilità di transire da un tag al successivo e la calcoliamo contando ogni quanto tempo eseguiamo questa transizione (transition probability).

La transition probability in particolare indica ad esempio la probabilità di trovare un nome dopo un articolo:

$$P(NOUN|DET) = \frac{C(DET, NOUN)}{C(DET)} = \frac{56,509}{116,454} = 0.49$$

in particolare all'interno della formula abbiamo che il numeratore viene calcolato contando le occorrenze di un articolo seguito dal nome mentre il denominatore conta il numero di volte che troviamo un articolo.

L'altra probabilità da calcolare è l'emission probability che è la probabilità che una certa parola sia taggata con un certo tag.

$$P(w_i | t_i) = \frac{C(t_i, w_i)}{C(t_i)}$$

Per calcolare questa probabilità dobbiamo contare il numero di volte che una certa parola viene marcata come verbo, ad esempio, dividendola per il numero di volte che le parole vengono segnate come verbi.

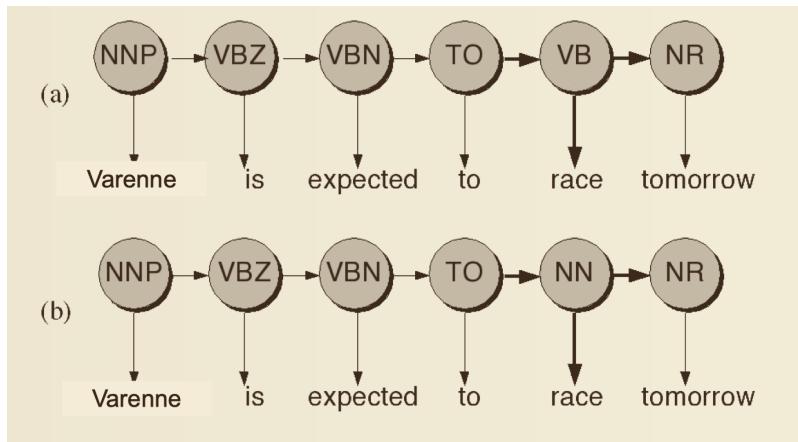
$$P(is | VERB) = \frac{C(VERB, is)}{C(VERB)} = \frac{10,073}{21.627} = 0.47$$

7.3.1 Esempio di disambiguazione

Consideriamo il seguente esempio, in particolare qua abbiamo una parola che è "race" che è ambigua perché in un caso è un verbo mentre nel secondo caso è un nome.

```
Varenne/NNP is/VBZ expected/VBN to/TO race/VB tomorrow/NR
People/NNS continue/VB to/TO inquire/VB the/DT reason/NN for/IN
the/DT race/NN for/IN outer/JJ space/NN
```

Vediamo come facciamo a disambiguare la parola "race", abbiamo due sequenze differenti relative alla stessa frase e calcoliamo la probabilità di ognuna di queste due, la differenza tra le due sequenze è che in una "race" viene indicata come verbo mentre nella seconda come nome.

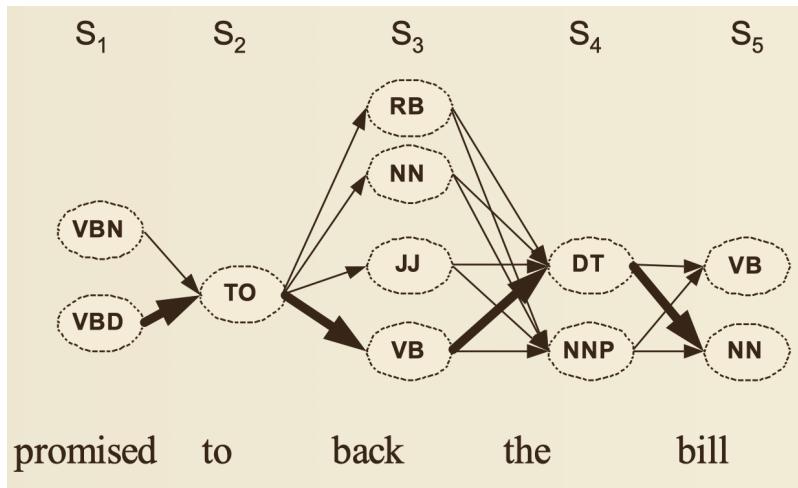


Lo score che otteniamo quindi è differente tra i due casi, noi consideriamo sia la transition probability sia l'emission probability:

$P(\text{NN} \text{TO}) = .00047$	Transition prob
$P(\text{VB} \text{TO}) = 0.83$	
$P(\text{race} \text{NN}) = 0.00057$	Emission prob
$P(\text{race} \text{VB}) = 0.00012$	
$P(\text{NR} \text{VB}) = 0.0027$	Transition prob
$P(\text{NR} \text{NN}) = 0.0012$	

e alla fine assegnamo la probabilità più alta.

Qua possiamo anche utilizzare l'algoritmo di Viterbi, nello specifico consideriamo un esempio differente con la frase "promised to back the bill". Back può avere varie possibili interpretazioni, quello che facciamo è creare il nostro grafo e inserire le possibili interpretazioni della parola che stiamo prendendo in considerazione:



Quindi quello che si fa è prima un calcolo forward in cui calcoliamo tutte le probabilità dei vari percorsi e arrivare alla fine e poi eseguiamo il passaggio backward e andiamo dal nodo finale del percorso con probabilità più alta fino al nodo iniziale recuperando il tag per tutte le varie parole della frase.

Chapter 8

Lesson 7 - Sequence Tagging

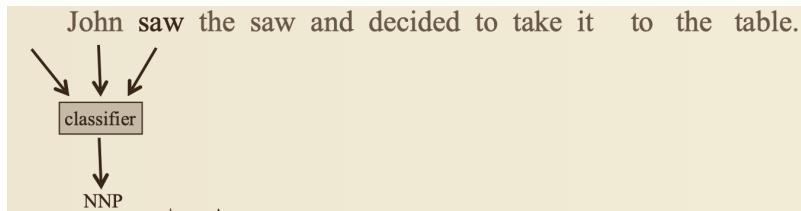
Fino ad ora abbiamo parlato di Part of Speech tagging e abbiamo detto che questa procedura può essere effettuata utilizzando un Hidden Markov Model. Allo stesso modo però questo tipo di classificazione può essere effettuata considerando il problema che dobbiamo risolvere e dividendolo. Per ognuna delle parole che abbiamo nel testo che stiamo prendendo in considerazione vorremmo poter applicare un classificatore:

$$\hat{t}_i \approx \operatorname{argmax}_{t_i} P(t_i | w_{i-k}, \dots, w_{i+k})$$

La probabilità che otteniamo per ognuna delle parole è la probabilità che quella parola in posizione i venga classificata in un certo modo considerando le K parole che la precedono e le K parole che la seguono. Alla fine tra tutte le probabilità che abbiamo calcolato consideriamo quella che ha il valore più alto.

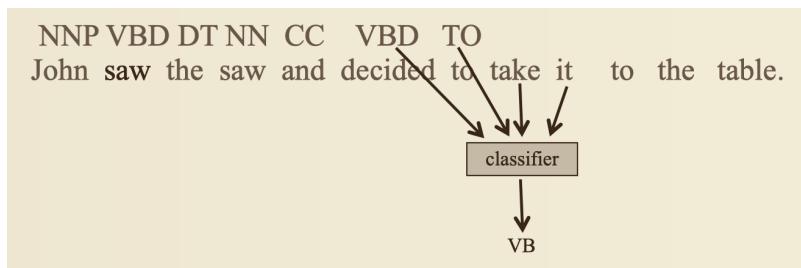
Un esempio, abbiamo il nostro classificatore e gli diamo in input la

parola che stiamo considerando, quella prima e quella dopo e il classificatore ci restituisce la classificazione corrispondente per quella parola:

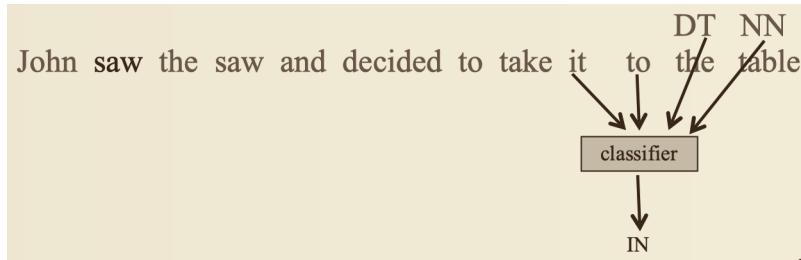


La stessa cosa la facciamo per tutte le parole che stanno nel testo andando da sinistra verso destra. Il classificatore è abbastanza intelligente da poter disambiguare nella maggioranza dei casi quelle parole che potrebbero avere vari possibili marker associati, ad esempio saw in questo caso è un nome e non un verbo. Ogni parte della sequenza viene classificata più o meno indipendentemente dato che consideriamo solamente una piccola finestra prima e dopo la parola target.

Possiamo migliorare la nostra classificazione, un metodo abbastanza semplice si chiama forward e consiste nell'utilizzare per ognuna delle parole del testo non solo la parola precedente come input ma anche la classificazione della parola precedente:



Un'alternativa a questo metodo si chiama backward, in questo caso partiamo dalla fine della frase e per ognuna delle parole utilizziamo anche la classificazione della parola successiva per classificare la parola target:



In questo caso ad esempio può essere utile sapere che dopo al "to" c'è un DT perchè in questo modo riusciamo a classificare "to" come un IN.

In alcuni casi potremmo ottenere una soluzione ancora migliore unendo le due strategie ovvero forward e poi backward.

8.1 Discriminative e Generative Models

Fino ad ora abbiamo visto soprattutto generative models perchè abbiamo studiato Naive Bayes e HMM. In questi modelli invece di produrre l'output dall'input possiamo provare a capire in che modo l'output viene generato ovvero vogliamo calcolare $P(X|Y)$ dove la X è il nostro input e la Y è l'output associato.

L'alternativa sono i discriminative models, qua abbiamo ad esempio la Max Entropy, la Logistic Regression, le CNN, la SVM e le Conditional Random Fields. Qua noi abbiamo un input X e vogliamo stimare l'output Y associato a quell'input. Quello che calcoliamo è $P(Y|X)$ e poi selezioniamo la classe che mi produce la probabilità più alta. Nella logistic Regression per esempio possiamo prendere in input il nostro dato X e produrre in output la classe C, quello che si fa è utilizzare dei pesi che vengono moltiplicati con i dati in input. Dobbiamo anche applicare la softmax perchè altrimenti non potremmo ottenere la probabilità. In questo caso possiamo anche avere più di due sole categorie.

8.1.1 Logistic Regression

Abbiamo parlato della logistic regression per eseguire il sequence tagging tramite classificazione. Utilizzando questo classificatore la prima cosa che dobbiamo fare è capire quali feature vogliamo considerare e poi stimare i pesi da usare nel nostro modello. Per ognuna delle parole presenti all'interno del nostro testo vogliamo estrarre alcune feature che possono essere rilevanti. Una volta che abbiamo capito quali feature utilizzare, per ognuna delle feature vogliamo utilizzare un peso che mi possa indicare quanto quella feature è realmente importante poi possiamo applicare la softmax per ottenere la probabilità.

È importante notare che la Logistic Regression non è un sequence model ma è un modello che mi classifica la singola osservazione.

8.1.2 Maximum Entropy Markov Model

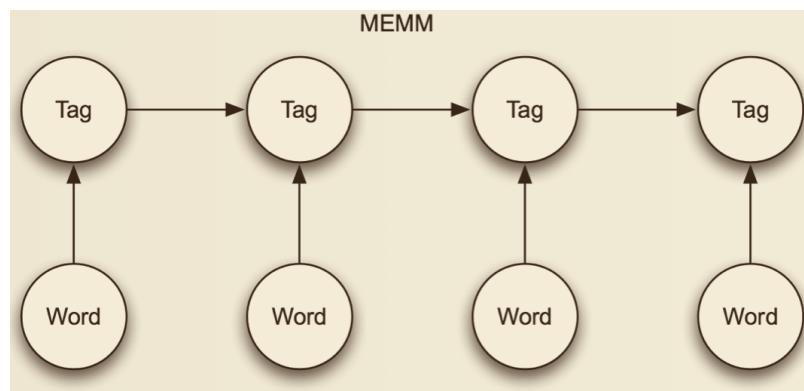
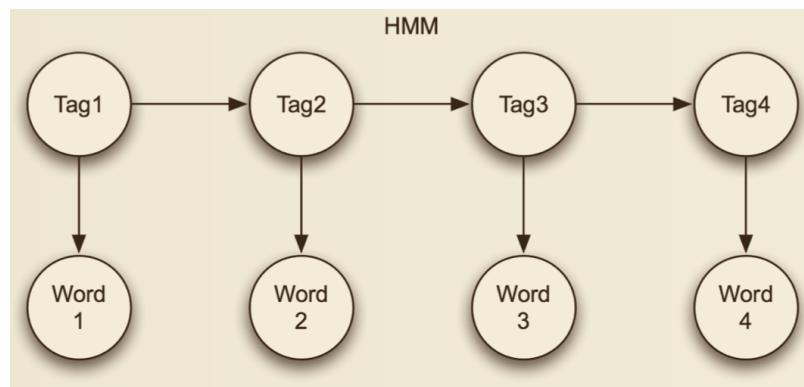
Parlando di sequence model abbiamo già visto gli Hidden markov Models che però hanno una limitazione, infatti non possiamo gestire parole non conosciute e non possiamo introdurre delle feature particolari all'interno del modello. Sarebbe molto più facile se potessimo riuscire ad utilizzare tutte le feature che vogliamo direttamente all'interno del nostro modello ma non è possibile usando direttamente l'HMM. Sappiamo però che la Logistic Regression mi permette di utilizzare tutte le feature che voglio per classificare una certa parola presente all'interno del testo, il problema è che la Logistic regression, come detto, non è un sequence model.

L'idea allora è quella di trasformare la logistic regression (che è un discriminative model) in un modello che possa lavorare su parole successive utilizzando ogni volta la classe assegnata alla parola precedente come feature delle classificazione della parola che stiamo prendendo in considerazione in quel momento. Quando applichiamo la logistic regression in questo modo (quindi come in un sequence model) allora abbiamo un modello che viene chiamato Maximum Entropy Markov Model o MEMM

(è un modello discriminativo).

Con il modello MEMM calcoliamo direttamente la posterior distribution $P(T|W)$ eseguendo il training che mi permetta di discriminare tra varie sequenze di tag. Questo sistema è differente rispetto al sistema che abbiamo visto nei HMM dove utilizzavamo la Bayes Rule per invertire la probabilità ovvero: $\text{argmax}P(T|W) = \text{argmax}P(W|T)P(T)$.

Graficamente possiamo notare la differenza nel momento in cui confrontiamo le seguenti figure:



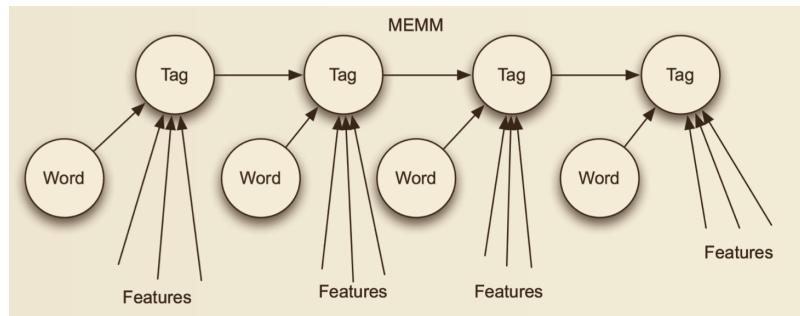
La seconda immagine considera il caso del MEMM e in questo caso le cose cambiano rispetto all'HMM perchè qua noi calcoliamo la probabilità che ognuna delle parole contribuisca ad ognuno dei tag. Nel modello

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

MEMM calcoliamo la posterior di ognuno degli stati considerando lo stato precedente e considerando l'osservazione attuale.

Utilizzando MEMM abbiamo anche la possibilità di fare la previsione dei tag ottenendo quindi la sequenza più probabile non soltanto in base al tag precedente e alla parola che stiamo considerando ma anche tenendo in considerazione:

- La parola in input w_i .
- I vicini della parola in input compresi tra w_{i-l} e w_{i+l} .
- I precedenti k tag



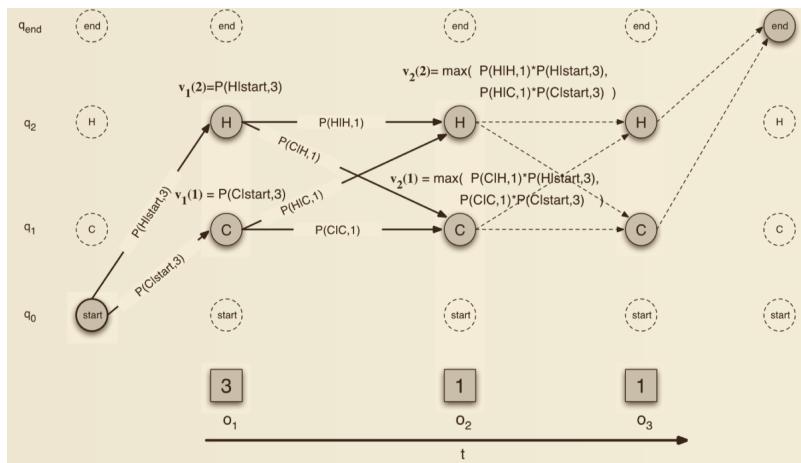
8.2 Problemi con i classificatori

Come possiamo fare a produrre la sequenza ottima T ? Il modo più semplice per eseguire questa trasformazione è creare un classificatore locale che sia in grado di classificare ogni parola partendo da sinistra e andando a destra. Questo è chiamato greedy decoding algorithm perchè scegliamo il miglior tag in modo greedy per ogni parola.

Il problema di questo classificatore greedy è che non possiamo utilizzare le decisioni future per modificare la risposta della classificazione della parola corrente. Allo stesso tempo però l'algoritmo greedy è molto veloce e in alcuni casi ha anche abbastanza accuracy.

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

L'alternativa all'algoritmo greedy è utilizzare MEMM insieme all'algoritmo di Viterbi in modo da trovare la sequenza di tag che sia ottima per l'intera frase e non solamente localmente. Utilizzando l'algoritmo di Viterbi con MEMM possiamo utilizzare la stessa idea sfruttata in precedenza e in particolare andiamo a considerare la probabilità solamente rispetto al tag precedente e alla parola che stiamo considerando, questo comporta che viene riempito un array $N * T$ e comporta anche che possiamo utilizzare Viterbi per prendere la probabilità più alta e da quella utilizzare l'algoritmo backward per tornare indietro trovando la sequenza corretta di tag.



In ogni caso la cosa migliore che possiamo fare è utilizzare un classificatore che prenda in considerazione la sequenza completa in modo da ottenere risultati migliori.

8.3 Notebook: POS Tagging

In questo notebook il dataset che viene utilizzato è il Brown Corpus, nello specifico si tratta della sezione "News" di questo corpus. Una delle prime cose che dobbiamo fare è scaricare l'insieme dei tag assegnati alle parole,

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

questi sono contenuti nell'Universal Tagset di UniversalDependencies.org che fornisce dati annotati in vari linguaggi.

Preso l'insieme delle parole presenti all'interno del corpus le annotiamo utilizzando l'insieme dei tag e quindi otteniamo delle coppie *<parola, tag>*.

```
from nltk.corpus import brown
nltk.download('universal_tagset')
brown_sents = brown.tagged_sents(categories='news',
                                    tagset='universal')
```

Dividiamo il nostro dataset in training set e in test set assegnando le ultime 1000 frasi al test set e le prime N frasi meno 1000 al training set.

Poi dobbiamo rappresentare le parole del nostro corpus e i tag associati alle parole con un singolo numero. Quindi prima di tutto creiamo un dizionario di tag e un dizionario di parole, poi associamo alle parole un numero e la stessa cosa la facciamo con i tag. Quello che otteniamo, per esempio per le parole del corpus, è un dizionario di questo genere con associazioni tra parole e numeri.

```
{u'sunbonnet': 3,
 u'Elevated': 4,
 u'narcotic': 5,
 u'four': 6,
 u'woods': 7,
 u'railing': 8,
 u'Until': 9,
 u'aggression': 10,
 u'marching': 11,
 u'looking': 12,
 u'eligible': 13,
 u'electricity': 14}
```

Quando creiamo il dizionario, anche se le parole del nostro corpus sono N e potremmo indicizzarle con numeri da 0 a N utilizziamo $N + 2$

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

numeri perchè teniamo lo 0 per indicare il token assegnato al padding e il numero 1 per indicare il token associato alle parole sconosciute.

La prossima cosa che dobbiamo fare è calcolare gli N-gram, quello che facciamo è considerare tutte le frasi del dataset e per ognuna delle frasi consideriamo ognuna delle parole presenti. L'N-gram lo formiamo considerando la parola e un numero di parole precedenti e successive che vengono indicate con il paramtetro *ctx_size*. Per ognuna di queste parole abbiamo un tag associato che viene aggiunto alla lista Y. Alla fine restituiamo due liste, una con tutti gli N-gram e una con i tag associati alla parola centrale di ciascuno degli N-gram. Notare che noi abbiamo la lista degli N-gram che è una lista di liste (perchè è una lista di N-gram ovvero una lista di liste di parole) mentre la lista dei tag è solamente una lista di tag.

```
import numpy as np

def build_ngrams(sentences, ctx_size, word_index, tag_index):
    """
    Turn sentences into n-grams of word indices, taken from word_index.
    :param sentences: a list of sentences.
    :param ctx_size: numer of tokens to consider before and after each word.
    :param word_index: dictionary of words.
    :param tag_index: dictionary of tags.
    :return: the list of n-grams and the list of corresponding labels.
    """
    padding_idx = 0
    unknown_idx = 1

    X = []
    Y = []

    for sentence in sentences:
        for n, (word, tag) in enumerate(sentence):
            word_indices = []
            for i in range(n - ctx_size, n + ctx_size + 1):
                # add padding
                if i < 0 or i >= len(sentence):
                    word_indices.append(padding_idx)
                    continue
                gram = sentence[i][0]
                # index of either word, lowercase word or unknown
                word_idx = word_index.get(gram) or word_index.get(gram.lower(),
                                                               unknown_idx)
                word_indices.append(word_idx)
            X.append(word_indices)
            Y.append(tag_index.get(tag))

    return X, Y
```

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

```

    word_indices.append(word_idx)

    X.append(word_indices)
    Y.append(tag_index[tag])

return np.asarray(X), np.asarray(Y)

```

Il prossimo step consiste nel trasformare gli output che abbiamo sui dati di training e sui dati di test (che ricordiamo, sono due array in cui in ogni posizione indichiamo il numero del tag che abbiamo assegnato alla parola centrale dell'N-gram associato) in un vettore 1 hot encoding in cui mettiamo 1 nella posizione corrispondente al tag associato all'n-gram e 0 in tutte le altre posizioni associate agli altri tag. Un esempio di questa trasformazione:

```
(4, array([0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0.]))
```

Qua noi diciamo che il tag associato all'n-gram è quello che troviamo in posizione 4 e quindi nell'array che viene restituito abbiamo un 1 in posizione 4 e 0 in tutte le altre posizioni. Per eseguire questa trasformazione utilizziamo la funzione *to_categorical* di keras:

```
y_train_cat = to_categorical(y_train, num_classes=n_tags)
y_test_cat = to_categorical(y_test, num_c)
```

8.3.1 Creazione del modello

Ora abbiamo i dati e li abbiamo trasformati in modo che il nostro modello di classificazione possa utilizzarli. Per il modello utilizziamo Keras, definiamo alcuni parametri come ad esempio la dimensione degli embedding, il numero di epoch per il training e la dimensione dei batch di dati per il training.

Nella rete neurale abbiamo i seguenti layer:

```

model = Sequential()
model.add(Embedding(n_words, EMBEDDING_DIM, input_length=window_size))
model.add(Flatten())
model.add(Dropout(0.2)) # optional
model.add(Dense(HIDDEN_SIZE, activation='relu'))
model.add(Dropout(0.2)) # optional

```

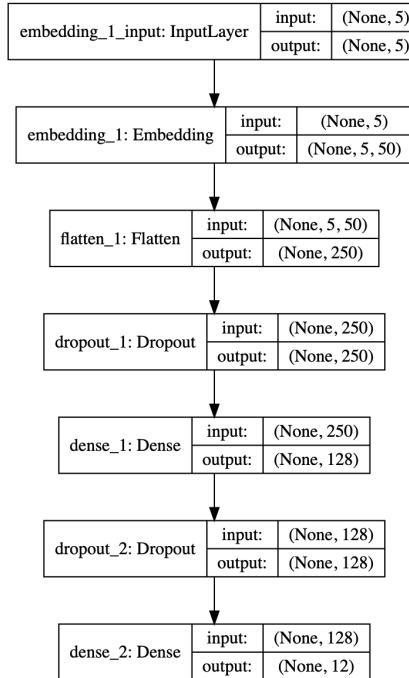
Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

```
model.add(Dense(n_tags, activation='softmax'))
model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['acc'])
```

Vediamoli nel dettaglio:

- Embedding: Ognuna delle frasi che stiamo considerando è formata dai vari N-gram, quello che vogliamo fare è creare gli embedding per ognuna delle parole che stanno nelle nostre frasi. In questo caso il classificatore avrà la sua rappresentazione delle parole e non utilizzeremo un pre-trained embedding. Questo layer trasforma i dati che sono una lista di numeri in una lista di vettori di dimensione 50. Ogni parola sarà rappresentata da un vettore di dimensione 50. In questo layer entrano gli N-gram uno alla volta.
- Flatten: questo insieme di vettori viene trasformato in un singolo vettore.
- Dropout: tra flatten e dense utilizziamo il layer del Dropout che mi serve per rimuovere alcuni dati dal nostro set, funziona in questo modo, per ogni dato viene generato un numero e se questo numero è maggiore di una certa probabilità che è stata scelta allora il dato viene rimosso altrimenti si mantiene.
- Dense: due layer, nel primo usiamo relu come activation function mentre nel secondo utilizziamo softmax. Il layer finale produce la distribuzione di probabilità per il prodotto di $w * x$ che abbiamo in input.

La rappresentazione della rete neurale è la seguente e da questa immagine sono chiare anche le dimensioni degli input e degli output:



Una volta completata la creazione del modello possiamo fare il training e possiamo provare il modello.

Per calcolare le performance del modello possiamo usare le metrics di sklearn.

8.3.2 Maximum Entropy Pos Tagging

Ora vogliamo utilizzare il Maximum Entropy Pos Tagging, in questo caso quindi possiamo anche utilizzare delle feature per classificare (le feature qua sono feature binarie). Per questo classificatore le feature che utilizziamo sono quelle del MaxentClassifier, nel caso del notebook ne consideriamo solamente una parte.

Per ogni parola presente all'interno delle nostre frasi consideriamo ciascuna delle feature e poi inseriamo in un dizionario di feature True se la feature è presente e False altrimenti.

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

Poi successivamente utilizziamo il nostro modello MaxEntropy Memory Model Tagger, prima lo possiamo definire e poi eseguiamo il training.

All'interno della definizione del modello dobbiamo anche definire la funzione di Viterbi che mi restituisce il path ottimo all'interno del grafo che viene prodotto dal MEMM.

8.4 Named Entity Recognition

Fino ad ora abbiamo sempre parlato di classificare testi o di prevedere la likelihood di un certo gruppo di parole e durante questi task andavamo a segnare le varie parole in base a regole grammaticali. Ci sono però nei testi delle parole che non possono essere segnate semplicemente basandosi su regole grammaticali perché potrebbero essere nomi di persone o di gruppi o di nazioni. Il riconoscere questi tipi di parole in un testo prende il nome di named entity recognition. Il riconoscimento di questi tipi di parole è un qualcosa che varia a seconda del contesto in cui stiamo lavorando, ci sono alcune parole che sono sempre valide, ad esempio nomi di persone o di nazioni, altre però sono specifiche. Prendiamo per esempio un testo di biologia, ci saranno delle parole specifiche relative a proteine che non troviamo in un testo che parla di economia ad esempio.

Nel campo della biologia è utile riconoscere un testo ed è particolarmente importante normalizzare le espressioni in modo da riconoscere che una certa frase si riferisce ad una proteina specifica anche se non si utilizza un ordine specifico nel nome di questa proteina.

Detto in modo più formale, definiamo named entity tutto quello che può essere definito con un nome proprio, a questi vanno aggiunte anche le cose che non sono entità di per sé, come ad esempio periodi storici o date specifiche che possono essere complicate da identificare (a differenza ad esempio di espressioni numeriche che possono essere più semplici).

Il task di Named Entity Recognition consiste nel trovare parti di testo che unite insieme costituiscono un nome proprio e poi classificare il tipo

di entità. Questo task diventa complicato per due motivi:

- Non sappiamo dove comincia e dove finisce l'entità, possiamo avere una frase di 3 parole che vogliamo sia taggata come entità perché magari rappresenta il nome proprio di una città. La difficoltà quindi sta proprio nel capire dove comincia e dove finisce l'entità.
- Un'altra difficoltà è causata dall'ambiguità del tipo di entità che troviamo. Ad esempio potremmo considerare la parola JFK che potrebbe voler indicare una persona ma anche un aeroporto.

Per restringere il campo delle possibili entità che vogliamo classificare potremmo creare ad esempio un elenco di questo genere:

Type	Tag	Sample Categories
People	PER	Individuals, fictional characters, small groups
Organizations	ORG	Companies, agencies, sport teams, parties, religious groups
Location	LOC	Physical extents, mountains, lakes, seas
Geo-political Entities	GPE	Countries, states, provinces, counties
Facility	FAC	Bridges, buildings, airports
Vehicles	VEH	Planes, trains, automobiles

Anche da questo elenco vediamo facilmente il problema dell'ambiguità perché la parola Italia potrebbe essere classificata sia come "Location" sia come "Geo Political Entities".

8.4.1 Usare NER

Per eseguire questo tipo di classificazione delle varie entità presenti all'interno di un testo possiamo utilizzare vari approcci differenti:

- Possiamo utilizzare un approccio Rule-Based in cui ci basiamo su una lista di nomi o su alcuni pattern che possiamo sfruttare per riconoscere le entità presenti all'interno di un discorso.

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

- Possiamo considerare l'approccio Machine Learning in cui andiamo a prendere i nostri dati di training che devono essere annotati e poi estraiamo da questi dati le feature più interessanti. Poi eseguiamo il training di un modello per replicare queste annotazioni anche sulle entità che non abbiamo ancora visto. Una soluzione può essere l'utilizzo di sequence classifier come ad esempio MEMM.

Come abbiamo detto il problema per il tagging delle parole presenti all'interno del testo è che qua non possiamo semplicemente classificare ogni parola perchè una certa entità potrebbe essere formata da 2 o 3 parole, ad esempio "Presidente Lotito". Una soluzione molto semplice a questo problema è l'utilizzo di un encoding chiamato IOB che funziona nel modo seguente:

- Nell'IOB tagging usiamo "B" come primo tag dell'entità che stiamo classificando.
- Usiamo "I" per indicare le parole interne all'entità.
- Usiamo "O" per indicare le parole esterne all'entità.

Consideriamo il caso in cui abbiamo N differenti categorie per il nostro named entity tagging, quello che otteniamo sono $2 * N + 1$ tags possibili (N per la B e N per la O), quindi in pratica raddoppiamo il numero di tag ma abbiamo una rappresentazione migliore.

L'alternativa è utilizzare un encoding differente che si chiama IO tagging, qua non consideriamo la B e questo potrebbe portarci al non poter distinguere tra due entità uguali che sono una vicina all'altra. Allo stesso tempo ci permette di avere solamente $N + 1$ tag invece dei $2 * N + 1$ dell'IOB. Dato che non capita spesso la situazione in cui abbiamo due entità uguali una vicino all'altra, IO potrebbe essere abbastanza funzionante.

Un esempio del tagging con IOB e con IO è il seguente:

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

Words	IOB Label	IO Label
American	B-ORG	I-ORG
Airlines	I-ORG	I-ORG
,	O	O
a	O	O
unit	O	O
of	O	O
AMR	B-ORG	I-ORG
Corp.	I-ORG	I-ORG
,	O	O
immediately	O	O
matched	O	O
the	O	O
move	O	O
,	O	O
spokesman	O	O
Tim	B-PER	I-PER
Wagner	I-PER	I-PER
said	O	O
.	O	O

Per eseguire il tagging delle entità possiamo utilizzare delle feature che possono aiutarci a prendere la decisione giusta. Prima di eseguire l'encoding con IOB ad esempio potrebbe essere utile effettuare una POS tagging per segnare ogni parola con la sua classificazione.

Questa fase del lavoro è la fase di feature engineer, queste feature sono importanti nel contesto di NER:

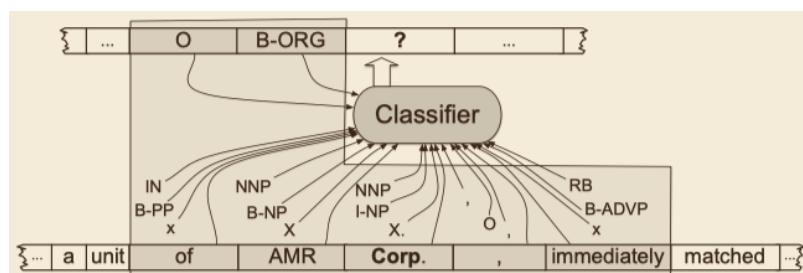
- Una feature che possiamo usare riguarda ad esempio la lettera maiuscola o minuscola che usiamo come prima lettera della parola.

- La presenza di una certa lettera davanti ad un'altra lettera ci può indicare di cosa stiamo parlando, ad esempio se trovo la parola "Via" è probabile che stiamo parlando di una location.
- Possiamo utilizzare liste di nomi di posti o anche liste di nomi di persone.

Nel corso degli anni poi lo stato dell'arte riguardo a questo tema si è espanso e ora si tende a considerare sempre più feature. Quando si sono cominciate ad utilizzare le Neural Network poi si è iniziato ad utilizzare il word embedding, quando si sono resi conto che questo sistema funzionava bene allora sono passati all'utilizzo di un word embedding a livello di caratteri. Si è passati quindi al character embedding, questo funziona bene ma non eccessivamente bene perché ha un costo molto alto perché dobbiamo andare carattere per carattere. Allora una soluzione differente consiste nell'inferire le feature che vengono considerate nella neural network sia utilizzando il word embedding sia utilizzando il character embedding. Utilizzando questo metodo possiamo avere risultati migliori rispetto all'utilizzo del singolo word embedding o del singolo character embedding.

8.4.2 Usare NER come sequence labeling

Consideriamo il seguente esempio:



Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

La parola su cui abbiamo il focus è "Corp.", consideriamo poi una finestra di due parole prima di "Corp." e due parole dopo. Nella figura le parti in grigio sono quelle per cui abbiamo a disposizione alcune features. Considerando queste features il classificatore può produrre una classificazione per la parola "Corp". Una volta che abbiamo classificato la parola "Corp" possiamo spostarci verso destra e possiamo considerare un altro gruppo di parole.

Questi classificatori individuali cercano di fare il massimo ma non ci permettono di ottenere il migliore assegnamento per l'intera sequenza. Una possibile soluzione consiste nell'applicare l'algoritmo di Viterbi che ci fornisce la migliore probabilità per l'intera sequenza. Possiamo utilizzare Viterbi in una HMM ad esempio oppure con MEMM.

Chapter 9

Lesson 8 - Parsing

Fino ad ora abbiamo parlato di Bag of Words, utili per l'information retrieval, di N-gram, usati per il POS tagging e per il NER, di sequences utili per la traduzione di testi. Vorremmo ora arrivare ad una comprensione del testo più "semantica" in cui riusciamo a comprendere il significato del testo e anche a capire quali sono le relazioni tra domande e risposte.

In particolare quello che vorremmo fare è ricostruire la struttura della frase in modo da comprenderne il significato. Le parole vengono composte in unità più grandi che vengono chiamate constituents, la sintassi si riferisce invece al modo in cui le parole sono messe insieme per creare i constituent o unità più grandi. La grammatica è il formalismo utilizzato per descrivere la sintassi del linguaggio.

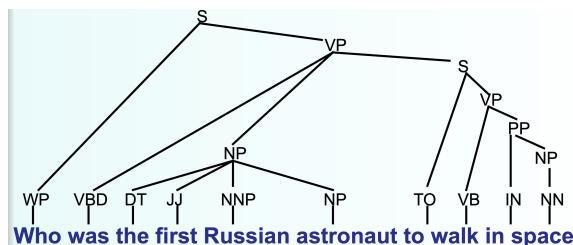
Abbiamo parlato più volte delle possibili ambiguità che possiamo trovare in un testo, si è parlato però solamente di ambiguità dal punto di vista della part of speech recognition, quindi ad esempio una parola che in un contesto potrebbe essere un nome o un verbo.

Un altro tipo di ambiguità è poi la structural ambiguity che capita quando tramite un parser possiamo assegnare più interpretazioni ad una certa frase. In particolare la structural ambiguity viene divisa in due tipi:

- Attachment Ambiguity: questo si verifica quando un determinato constituent può essere inserito in più di un punto del parse tree. Un esempio: "Gli scienziati hanno contato le balene dallo spazio", qua ci sono due possibili interpretazioni per la frase.
- Coordinator Ambiguity: in questo caso differenti parti di una frase possono essere uniti insieme con una congiunzione come ad esempio and e questo ci porta ad avere una ambiguità perché la parte di frase con l'and potrebbe rappresentare un singolo soggetto o magari due soggetti differenti.

I principali utilizzi pratici del parsing sono:

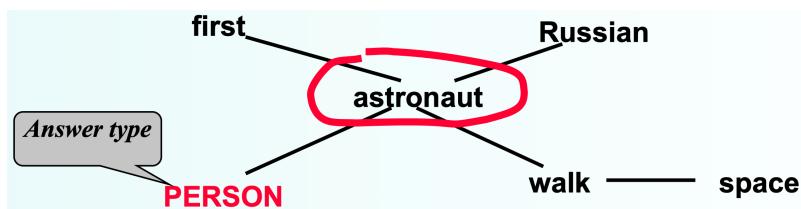
- Possiamo usarlo per la sentiment analysis, possiamo infatti migliorare l'analisi del testo utilizzando informazioni che arrivano dalla dependency parsing.
- Possiamo comprendere meglio le negazioni presenti all'interno di una frase.
- Possiamo riuscire a riassumere un testo andando a rilevare le parti più importanti.
- Possiamo utilizzare il parsing per il question answering, ad esempio con i chat bot.



Un esempio dell'utilizzo dei parse tree in chat bot che possono essere utilizzati per il question answering, possiamo vedere l'immagine

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

sopra che è la soluzione proposta da un gruppo in una challenge. Loro hanno utilizzato il parse tree per comprendere la frase, in particolare all'interno della frase hanno cercato quale fosse il focus, in questo caso è l'astronauta che è una persona e noi stiamo chiedendo informazioni riguardo a questa persona.



Se siamo capaci a parsare delle frasi allora possiamo anche disambiguare. Il parsing rientra in uno dei possibili problemi che possono essere risolti utilizzando una rappresentazione gerarchica e poi utilizzando metodi statistici per trovare la rappresentazione migliore.

Per quanto riguarda il parsing abbiamo in particolare due tecniche che possono essere adottate:

- Constituency Grammar
- Dependency Grammar

Breve divisione dei vari tipi di parser

- Constituency Parsing
- Dependency Parsing:
 - Transitional based: è il parser più veloce ed è anche competitivo rispetto agli altri.
 - Graph Based: se vogliamo ottenere una accuracy molto alta dobbiamo usare questo tipo di parser, altrimenti possiamo utilizzare il transitional based.

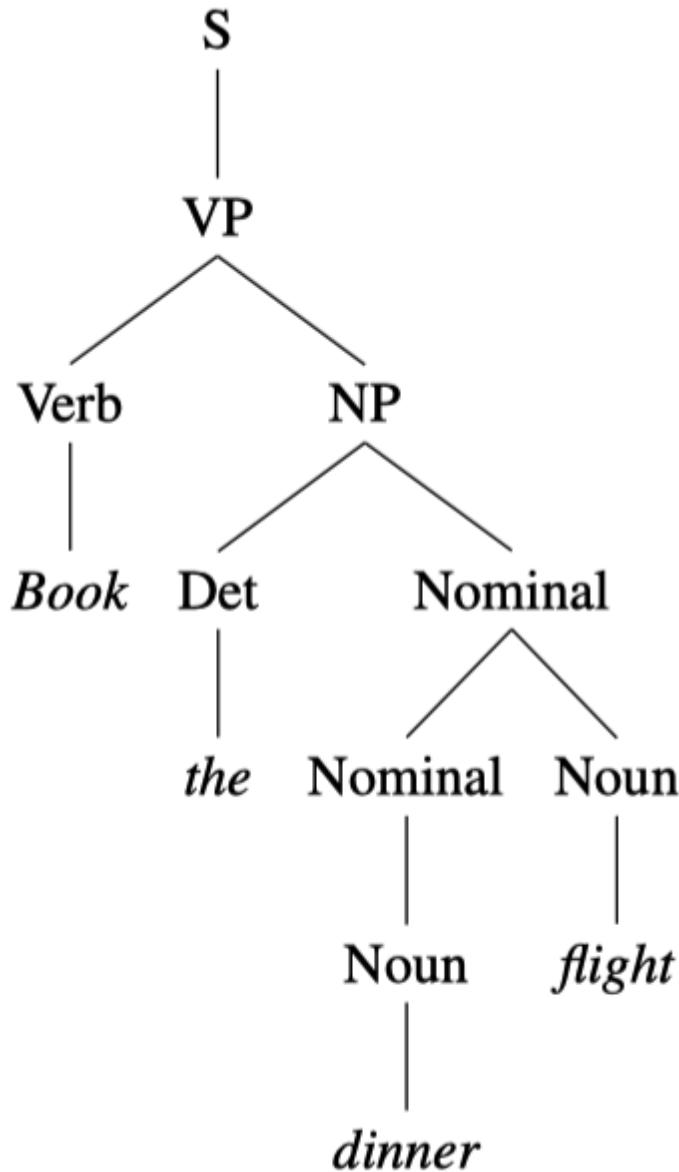
9.1 Constituency Grammar

Questa è una delle due tecniche che possono essere utilizzate per il parsing, per questa tecnica utilizziamo una context free grammar ovvero una grammatica definita da 4 parametri, in particolare utilizziamo delle production rule per il parsing delle varie frasi.

Utilizzando la grammatica libera dal contesto, definita coi seguenti parametri possiamo iniziare a generare il parse tree:

N	a set of non-terminal symbols (or variables)
Σ	a set of terminal symbols (disjoint from N)
R	a set of rules or productions, each of the form $A \rightarrow \beta [p]$, where A is a non-terminal,
	β is a string of symbols from the infinite set of strings $(\Sigma \cup N)^*$, and p is a number between 0 and 1 expressing $P(\beta A)$
S	a designated start symbol

Quello che si fa è creare delle production rules che vengono utilizzate per creare un parse tree di questo genere:



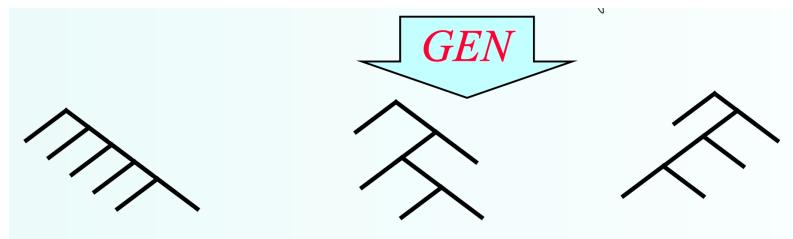
Per convenzione nel nodo root abbiamo la "S" che indica la frase intera, poi l'albero è la rappresentazione del parsing. Data la frase costruiamo il parse tree corrispondente, la soluzione più banale consiste nella

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

generazione di tutti i possibili alberi per quella frase ma chiaramente questo non è fattibile soprattutto se abbiamo una grammatica completa. L'alternativa più logica consiste nell'usare una "guida" che mi permetta di generare l'albero migliore per la frase che stiamo considerando basandoci sulla statistica.

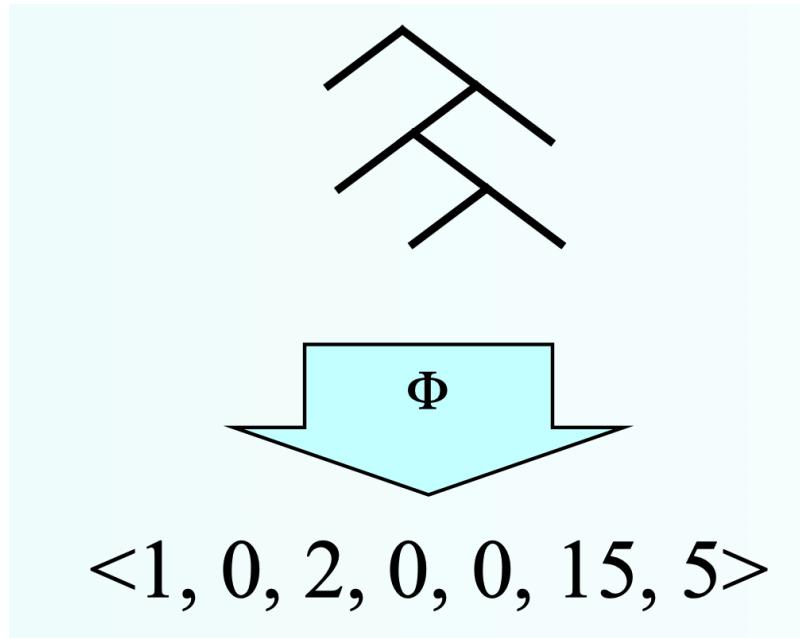
In un modello statistico per il constituency parsing abbiamo i seguenti componenti:

- GEN: è la funzione che va da una stringa ad un set di possibili candidati, nel nostro caso è rappresentata dalla grammatica che permette di generare il parse tree.



La funzione ci permette di generare vari possibili alberi utilizzando le production rule della grammatica.

- Abbiamo una funzione che permette di assegnare alla stringa che stiamo considerando un feature vector. Quello che si fa qua è prendere un sottoalbero e rappresentarlo con un vettore.



Queste feature che consideriamo dipendono dalla struttura dell’albero che stiamo creando, ad esempio una feature può essere il numero di volte che un determinato sottoalbero compare nell’albero complessivo. L’insieme di queste feature forma il feature vector.

- Abbiamo un array di parametri, uno per ognuna delle feature che stiamo considerando.

Definiti questi tre componenti il nostro obiettivo diventa quello di prendere un set di frasi X e un set di output Y (ad esempio potrebbero essere gli alberi corrispondenti alle frasi). Vogliamo eseguire il learning di una funzione $F : X \rightarrow Y$ che deve essere definita in modo tale che mi restituisca tra tutte le possibili soluzioni quella che ha lo score più alto:

$$F(x) = \operatorname{argmax}_{y \in \text{GEN}(x)} \phi(y) * W$$

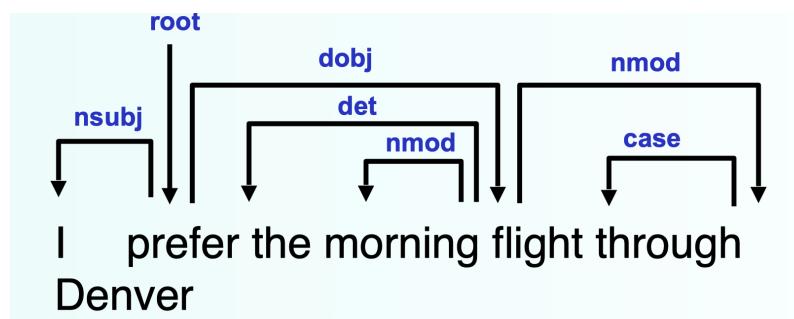
Alla fine sceglieremo l'albero a cui è associato lo score più alto tra i vari che abbiamo a disposizione.

9.2 Dependency Grammar

Si tratta della seconda delle due soluzioni che avevamo elencato in precedenza. In questa soluzione ci occupiamo di assegnare una struttura di dipendenze alle varie frasi che abbiamo a disposizione. Con struttura di dipendenze intendiamo l'elenco di quelle parole che sono dipendenti tra loro, quindi modificano o sono argomento di un'altra parola.

In questa soluzione i costituenti delle frasi e la struttura delle frasi non giocano un ruolo diretto, la struttura sintattica di una parola viene descritta solamente in termini di parole in una frase e in base alle relazioni che valgono tra le varie parole.

Possiamo rappresentare queste dipendenze tramite un grafo orientato con gli archi che connettono parti delle varie frasi:

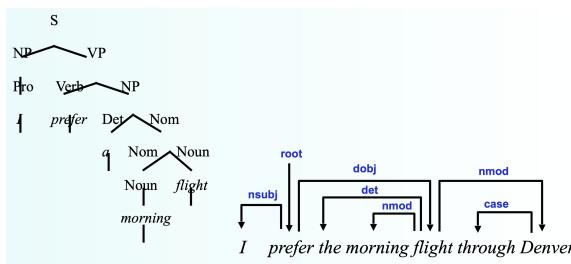


Quello che viene rappresentato sopra viene chiamato "typed dependency structure" perchè le descrizioni dei vari archi sono prese da una relazione di una grammatica.

Nell'esempio della figura sopra la parola "prefer" che è una delle più importanti della frase è legata (tramite una syntactical relation) alla parola "flight", allo stesso modo l'articolo "the" è legato a flight perchè è l'articolo determinativo corrispondente.

Uno dei vantaggi dell'utilizzo di questo metodo consiste nel fatto che qua riusciamo a mettere in relazioni le varie parti del discorso indipendentemente dalla loro posizione, quindi riusciamo a supportare anche quei linguaggi in cui la posizione delle parole all'interno delle frasi può variare, come ad esempio la lingua ceca.

Ora che abbiamo introdotto anche questo seconda rappresentazione possiamo fare un confronto tra il constituency tree e il dependency tree:



Il dependency tree sulla destra connette direttamente le varie parole mentre il constituency tree sulla sinistra fa uso di nodi intermedi tra le varie parole. Questo comporta che, se voglio cercare una relazione tra due parole presenti all'interno di questi due alberi sarà più facile utilizzare il dependency tree perché da ogni parola ho un collegamento diretto con le parole che sono dipendenti, utilizzando il constituency tree invece dovrei fare una visita dell'albero alla ricerca del nodo che è legato a quello che sto considerando. Un esempio, se vogliamo trovare il soggetto del verbo "prefer" possiamo trovare facilmente "I" con il dependency tree di destra mentre invece dobbiamo muoverci nell'albero andando verso l'alto fino a che non si trova "I" per fare la stessa cosa nel constituency tree.

9.3 Dependency Parsing

Il dependency parsing produce dei dependency tree in cui le parole sono collegate in dependency relations. Ognuna di queste dependency relation consiste in una "Head" e in un "Dependent" dove la "Head" consiste

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

nella parola centrale del discorso mentre il "Dependent" indica quello che dipende dalla "Head". Oltre a collegare Head con Dependent possiamo anche assegnare una grammatical relation che indica il ruolo che la dipendenza gioca rispetto alla Head a cui è collegata.

Per il dependency parsing possiamo utilizzare una strategia data driven e abbiamo due possibili soluzioni che possono essere adottate:

- Graph Based: proviamo i vari possibili parse tree e poi assegnamo uno score ad ognuno di questi alberi scegliendo poi il migliore. Sono solitamente delle soluzioni più lente ma che garantiscono risultati migliori.
- Transition Based: si tratta di un transition system che risulta più simile alle classiche soluzioni che vengono adottate nei compilatori. In questo caso quello che si fa è ottenere il parse tree andando a considerare una parola alla volta. Sono anche soluzioni molto veloci.

9.3.1 Transition Based

Un tipo di parsing che possiamo utilizzare è il Transition Based Shift Reduce. L'algoritmo utilizzato è semplice e ottimo da utilizzare anche dal punto di vista delle performance perché consiste semplicemente in una lettura da destra a sinistra di tutte le parole presenti all'interno della frase che stiamo considerando. L'algoritmo utilizza:

- Uno stack in cui inseriamo le parole che sono state parzialmente processate.
- Una lista di parole che devono essere parsate.
- Un set di label per i vari archi che abbiamo creato.

Gli elementi della lista di parole vengono progressivamente spostati nello stack e i primi due elementi dello stack vengono confrontati con le regole della grammatica che abbiamo definito. Quando troviamo un match sostituiamo gli elementi che sono stati matchati all'interno dello stack.

All'inizio del processo di parsing creiamo una configurazione iniziale con lo stack che contiene il nodo root, la lista di parole è inizializzata con l'elenco delle parole presenti all'interno della frase e abbiamo poi una lista di relazioni che, almeno inizialmente sarà vuota. L'obiettivo finale è avere una lista di relazioni piena, lo stack e la lista di parole vuote.

All'inizio partiamo dalla prima parola della lista, questa è attualmente il nostro focus e vogliamo trovare le dipendenze, la parola successiva sarà quella che prendiamo in considerazione. Partiamo dalla prima parola della nostra frase che quindi diventa la prima parola nello stack ("Top") e consideriamo la relazione con la seconda parola della frase che è la "Next". Abbiamo tre possibilità quando consideriamo due parole che sono vicine tra loro nella frase:

- LeftArc: in questo caso abbiamo una relazione tra le due parole, in particolare la prima nello stack (la "Top" della figura) deve essere legata con la seconda parola dello stack (la "next" della figura). Questa relazione è tale che la seconda parola dello stack sarà "Head" per la prima. Dopo aver messo questa relazione spostiamo la "Top" a sinistra mentre la "Next" rimane dove stava.

$$\frac{\langle \mathbf{S|s, n|B, A} \rangle}{\langle \mathbf{S, n|B, A \cup \{(n, s, r)\}} \rangle}$$

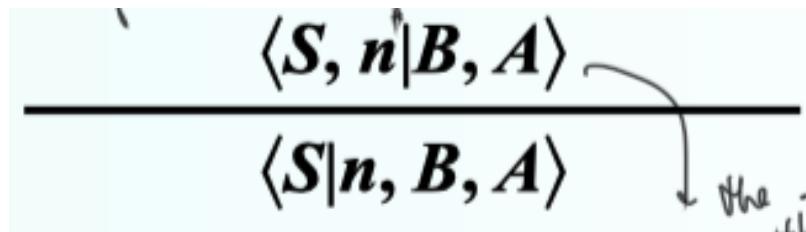
Come possiamo interpretare la formalizzazione:

- La parte sopra indica la situazione prima dell'applicazione della regola.
- La parte sotto indica la situazione dopo l'applicazione della regola.
- Stiamo considerando al s che è al top dello stack e la n che è al top del buffer. Quello che facciamo è creare un collegamento LeftArc tra n ed s .
- Dopo aver aggiunto questo collegamento vediamo che nel set delle relazioni ne viene aggiunta una nuova tra n e s , lo stack è vuoto e abbiamo il buffer che ha sempre la n come primo elemento.
- RightArc: in questo caso abbiamo una dipendenza che è l'opposto di quella LeftArc, quindi avremo un collegamento da destra verso sinistra e dopo aver inserito questo collegamento dovremo andare a muoverci verso destra.

$$\frac{\langle \mathbf{S}|s, \mathbf{n}|\mathbf{B}, A \rangle}{\langle \mathbf{S}, s|\mathbf{B}, A \cup \{(s, n, r)\} \rangle}$$

L'interpretazione della formula è speculare a quella che abbiamo visto per la LeftArc.

- Possiamo anche non aggiungere una dipendenza, se non c'è una dipendenza tra le due parole che stiamo considerando ci limitiamo solamente a muoverci verso destra.



	top	next
<i>LeftA</i>	He PP	saw VVD
	a DT	girl NN
	with IN	
	a DT	telescope NN
	.	SENT

	top	next
<i>LeftA</i>	saw VVD	a DT
	He PP	girl NN
		with IN
		a DT
		telescope NN
		.

	top	next
<i>Shift</i>	saw VVD	a DT
	He PP	girl NN
		with IN
		a DT
		telescope NN
		.

	top	next
	saw VVD	girl NN
	He PP	with IN
		a DT
		telescope NN
		.

	top	next
	saw VVD	with IN
	He PP	a DT
	girl NN	telescope NN
		.

Le regole che abbiamo visto per la transizione all'interno del set di parole da parsare sono chiamate "Nivre's Arc Standard" ma in generale ci sono anche regole alternative che possono essere utilizzate come ad esempio le "Attardi's Non Projective". Il problema principale dell'approccio

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

"Nivre's Arc Standard" è che dobbiamo prendere molto presto la decisione riguardo alle relazioni tra le parole e questo può comportare che vengono effettuati degli errori.

Più formalmente l'algoritmo che stiamo considerando per risolvere questo problema è il seguente:

```
Input Sentence: ( $w_1, w_2, \dots, w_n$ )
 $S = \langle \rangle$ 
 $B = \langle w_1, w_2, \dots, w_n \rangle$ 
while  $B \neq \langle \rangle$  do
     $x = \text{getContext}(S, B)$ 
     $y = \text{selectAction}(\text{model}, x)$ 
     $\text{performAction}(y, S, B)$ 
```

Un parser tradizionale ha bisogno di una grammatica per generare i possibili alberi, un parser induttivo invece non necessita di grammatica, possiamo infatti eseguire il training del parser partendo da un corpus annotato in cui consideriamo le azioni effettuate ad ogni step. In pratica durante la fase di learning consideriamo le azioni di shift e reduce che sono state effettuate e da queste riusciamo a creare un modello che è simile ad un classificatore.

Quello che otteniamo alla fine è il dependency graph che abbiamo già descritto in precedenza, un vincolo di questo grafo ogni nodo viene collegato con al più un altro nodo utilizzando un arco labellato.

9.3.2 Oracolo

Quando dobbiamo eseguire il training di un parser come possiamo fare a determinare l'azione corretta da applicare ad ogni step? Abbiamo bisogno di un oracolo che ci possa aiutare a scegliere ad ogni passaggio

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

l'azione migliore da effettuare. L'oracolo prende in input una configurazione e restituisce in output una operazione di transizione, per fare il training di un classificatore poi avremo bisogno di configurazioni assegnate ad una operazione di transizione. Se usiamo solamente l'albero, questo mi assegna ad ogni frase il corrispondente albero e non ci fornisce direttamente quello che ci serve, per questo abbiamo bisogno di un oracolo. Quello che vediamo qua è chiamato "Arc Standard Oracle".

Come funziona l'oracolo dato un parsing di riferimento e una configurazione?

- Si sceglie LeftArc se produce una corretta relazione di dipendenza Head-Dependent dato il parsing di riferimento e la configurazione corrente.
- Si sceglie RightArc se produce una corretta relazione di dipendenza Head-Dependent dato il parsing di riferimento e la configurazione corrente e se tutte le dipendenze della parola in cima allo stack sono già state assegnate. La seconda condizione mi dice che scegli RightArc se tutte le relazioni che escono dalla parola che stiamo considerando e che sono in R_p sono anche in R_c .
- Altrimenti si sceglie SHIFT.

Durante la fase di training l'oracolo ha accesso alle seguenti informazioni:

- La configurazione corrente di stack S e di relazioni di dipendenza R_c
- Un parsing di riferimento che consiste in un set di vertici e in un set di relazioni di dipendenza R_p .

Più formalmente possiamo descrivere la scelta fatta dall'oracolo in questo modo:

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

```

LEFTARC(r): if  $(S_1 \ r \ S_2) \in R_p$ 
RIGHTARC(r): if  $(S_2 \ r \ S_1) \in R_p$  and  $\forall r', w \text{ s.t. } (S_1 \ r' \ w) \in R_p \text{ then } (S_1 \ r' \ w) \in R_c$ 
SHIFT: otherwise

```

L'algoritmo dell'Oracolo è il seguente:

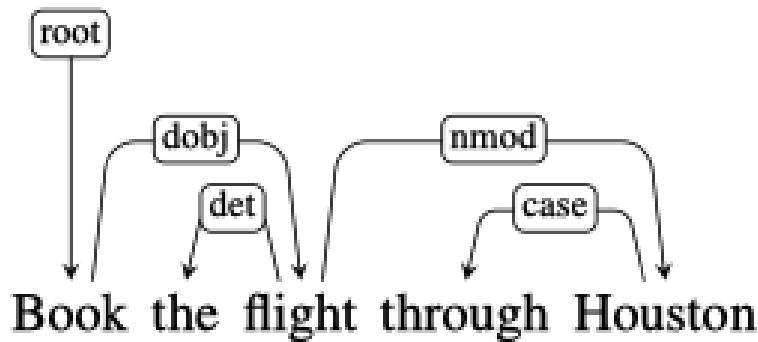
```

Emulate the parser
Input Sentence:  $(w_1, w_2, \dots, w_n)$ 
S = <>
B = <w1, w2, ..., wn>
while B != <> do
    if B[0] -r→ S[0] and all children of S[0] are attached
        perform Left-Arc,
    else if S[0] -r→ B[0] and all children of B[0] are
        attached
        perform Right-Arc,
    else
        perform Shift

```

Why?

Vediamo un esempio di funzionamento dell'oracolo:

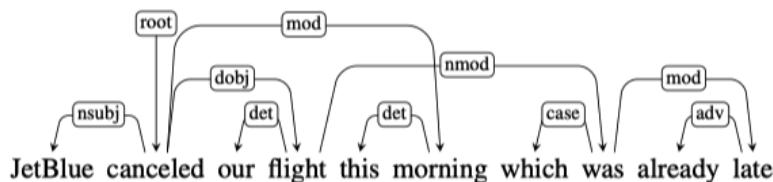


- Al primo passaggio non possiamo applicare LeftArc perché la relazione root-Book non è in R_p e non possiamo applicare RightArc perché Book non è stato ancora collegato alle sue dipendenze, quindi SHIFT.

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

- Quando il focus è su the possiamo usare LeftArc e colleghiamo quindi the alla sua Head che sarebbe Flight.
- Quando il focus è su Flight potremmo scegliere RightArc perchè abbiamo la dipendenza Book-Flight in R_p però siamo fermati dal fatto che non abbiamo in R_p anche la dipendenza Flight-Houston e quindi ci fermiamo e facciamo SHIFT.

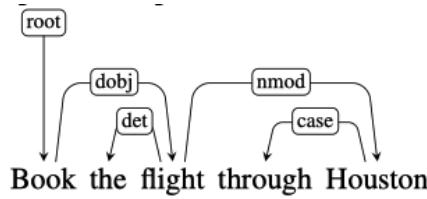
Un vincolo aggiuntivo che può essere derivato dall'ordine delle parole in input è l'idea della Projectivity e Non Projectivity. Con Projectivity intendiamo che data una coppia di parole che sono connesse con la relazione Head-Dependent, l'arco che connette è detto Projective se c'è un percorso per andare dalla Head a tutte le parole che sono comprese tra Head e Dependent. Esempio:



Qua abbiamo che l'arco tra flight e was è non projective perchè non c'è modo di arrivare a this e a morning, invece possiamo arrivare a which. Ad esempio invece tra was e late abbiamo un arco che è projective.

Quello visto fino ad ora è l'Arc Standard Algorithm che è capace di produrre solamente projective dependency tree ma non è in grado di produrne di non projective. È un algoritmo efficiente perchè è lineare e in un secondo riusciamo a processare centinaia di frasi.

Una alternativa all'Arc Standard Algorithm è l'Arc Eager che è in grado di riconoscere i RightArc prima rispetto all'Arc Standard.



Se consideriamo sempre il solito esempio di prima vediamo che il RightArc tra Book e Flight viene rimandato dalla Arc Standard mentre invece viene rilevato immediatamente dalla Arc Eager che cerca di collegare le parole alla Head corrispondente il prima possibile. Per fare questo l'Arc Eager definisce un nuovo operatore (Reduce) che si aggiunge a quelli visti in precedenza:

- LeftArc: Aggiunge una dipendenza tra la prima parola dell'input buffer e la parola che sta al top dello stack e poi svuota lo stack.
- RightArc: Aggiunge una dipendenza tra la parola al top dello stack e quella all'inizio dell'input buffer poi muove la parola all'inizio dell'input buffer nello stack.
- Shift: Rimuove la parola che sta all'inizio dell'input buffer e poi la mette nello stack.
- Reduce: Fa un pop dallo stack.

9.4 Non Projective Transitions

Vogliamo un sistema che permetta l'utilizzo anche di non projective arcs, possiamo utilizzare quello progettato dal Prof. Attardi.

Formalmente le regole che permettono il funzionamento di questo metodo sono le seguenti:

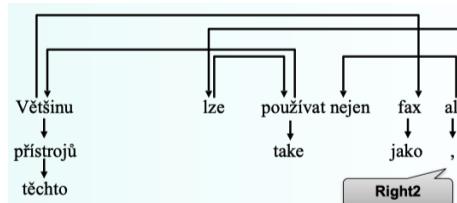
Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

Right2_r	$\frac{\langle S s_2 s_1, n B, A \rangle}{\langle S s_1, n B, A \cup \{(s_2, r, n)\} \rangle}$
Left2_r	$\frac{\langle S s_2 s_1, n B, A \rangle}{\langle S s_2, s_1 B, A \cup \{(n, r, s_2)\} \rangle}$
Right3_r	$\frac{\langle S s_3 s_2 s_1, n B, A \rangle}{\langle S s_2 s_1, n B, A \cup \{(s_3, r, n)\} \rangle}$
Left3_r	$\frac{\langle S s_3 s_2 s_1, n B, A \rangle}{\langle S s_3 s_2, s_1 B, A \cup \{(n, r, s_3)\} \rangle}$

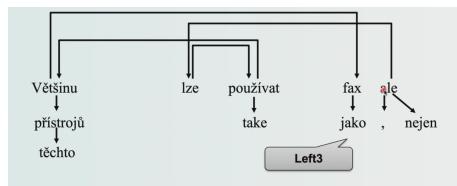
Vediamo un esempio di funzionamento di questo metodo, abbiamo un esempio di questo genere e possiamo iniziare a parsare:



Ad un certo punto arriviamo a questa situazione:

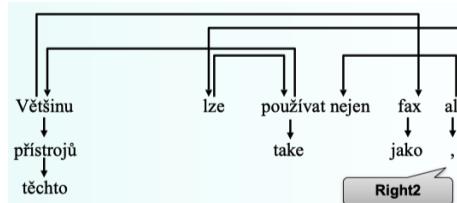


Abbiamo "ale" e "fax" che non sono ancora connesse ma possiamo usare la right2 rule per arrivare alla situazione seguente:



Da qua possiamo utilizzare la regola Left3 e arrivare qua:

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.



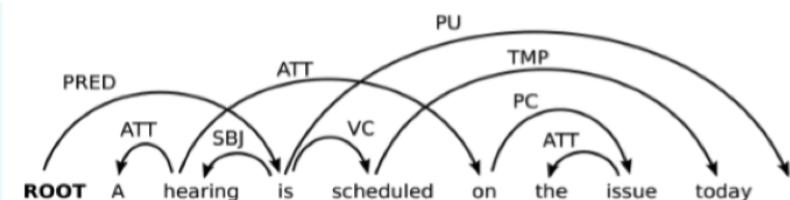
da qui in poi possiamo andare avanti con le regole normali.

Utilizzare un sistema di questo genere ha effettivamente dei benefici, abbiamo considerato il training con la lingua ceca, comunque abbiamo delle difficoltà se non ci sono troppi esempi.

9.4.1 Parsing con swap

Un'alternativa ancora consiste nell'eseguire il parsing "Swap" ovvero un parsing in cui prendiamo i vari token e poi possiamo modificarne l'ordine.

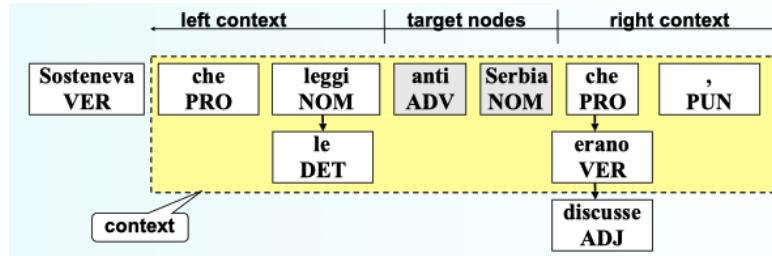
Prendiamo questo esempio:



Qua abbiamo la frase "A hearing is scheduled on the issue today", quando facciamo il parsing e siamo arrivati a parsare fino a "A hearing is scheduled" poi troviamo "on the issue today". Quello che possiamo fare arrivati a questo punto è spostare "scheduled" di nuovo nel buffer e poi dal buffer spostiamo nello stack on. Dobbiamo fare altre varie operazioni di swap per arrivare alla fine alla situazione in cui abbiamo la frase finale "riposizionata" ovvero "A hearing on the issue is scheduled today".

9.5 Learning Phase

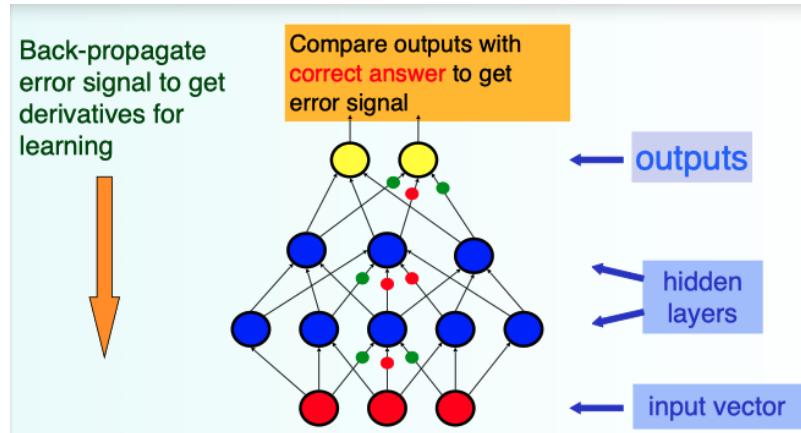
Durante la fase di learning abbiamo delle feature che possono essere utilizzate e che permettono di raggiungere il risultato finale.



Durante la fase di learning abbiamo due nodi target nel mezzo e dobbiamo decidere se questi sono connessi con il contesto di destra e con il contesto di sinistra. Il classificatore ci fornirà una rappresentazione di questo contesto, quindi quello che vogliamo fare è mantenere il contesto più piccolo possibile. All'interno del contesto non mettiamo tutte le parole che sono dipendenti, ad esempio "erano" è collegato con "discusse" ma "discusse" non è all'interno del contesto perché vengono mantenuti solamente i figli dei nodi e non i nipoti.

Per ognuna delle parole presenti in questo contesto vengono prese in considerazione le varie feature.

Durante la fase di training dobbiamo emulare le decisioni che vengono prese durante il parsing, consideriamo ogni frase e poi eseguiamo il training del classificatore. La fase di parsing è un classificatore un po' inusuale, non sarebbe un problema di classificazione ma in realtà lo consideriamo come tale. Un dependency parser che possiamo utilizzare nella fase di training è DeSR. Questo è un dependency parser transition based che supporta vari linguaggi e che è anche capace di gestire la non projectivity. Abbiamo a disposizione vari possibili classificatori, attualmente il Multilayer Perceptron è il migliore da utilizzare perchè possiamo utilizzarlo anche combinato con il word embedding e possiamo ottenere in questo modo un parser che utilizza il deep learning.



9.5.1 CoNLL-X

Queste attività sono state avviate nel 2006, l’obiettivo è quello di assegnare label alla dependency structure supportando tanti linguaggi e utilizzando dependency parser del tutto automatici. Abbiamo in input frasi tokenizzate e labellate con tag che indicano i token, i lemma, le POS e le dipendenze Head->Dependency. Ogni token viene rappresentato con 7 attributi, poi c’è anche una colonna extra che non viene utilizzata e che riguarda la projectivity. Per ogni frase abbiamo i lemma, le features sono morfologiche e abbiamo anche i dependence edge. I token sono numeri e nella seconda colonna abbiamo le varie parole.

N.	WORD	LEMMA	CPOS	POS	FEATS	HEAD	DEPREL
PHEAD	PDEPREL						
1	A	o	art	art	<artd>FIS	2	>N
2	direcção	direcção	n	n	FIS	4	SUBJ
3	já	já	adv	adv		4	ADVl
4	mostrou	mostrar	v	v-fin	PSI3SIIND	0	SIA
5	boa_vontade	boa_vontade	n	n	FIS	4	ACC
6	,	,	punc	punc		4	PUNC
7	mas	mas	conj	conj-c	<co-vfin>l<co-fmc>	4	CO
8	a	o	art	art	<artd>FIS	9	>N
9	greve	greve	n	n	FIS	10	SUBJ
10	prosegue	proseguir	v	v-fin	PRI3SIIND	4	CUT
11	em	em	prp	prp		10	ADVl
12	todas_as	todo_o	pron	pron-det	<quant>IFIP	13	>N
13	delegações	delegação	n	n	FIP	11	P<
14	de	de	prp	prp	<sam>	13	N<
15	o	o	art	art	<sam>l<artd>IMIS	16	>N
16	país	país	n	n	MIS	14	P<
17	.	.	punc	punc		4	PUNC

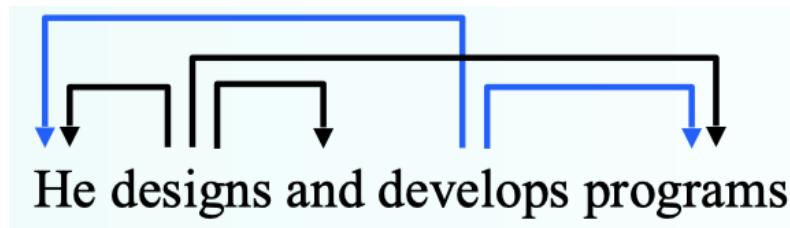
Ci sono due metriche che possono essere utilizzate per valutare il funzionamento di CoNLL:

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

- UAS (Unlabeled Attachment Score): indica la proporzione di token che sono stati assegnati alla Head corretta. Questo mi dice quanto è accurato il parser a creare gli archi.
- LAS (Labeled Attachment Score): indica la proporzione di token che sono assegnati sia alla Head corretta sia al label corretto. Questo prende in considerazione i link e gli archi, è un task più complesso. In alcune situazioni alcuni archi potrebbero non essere importanti.

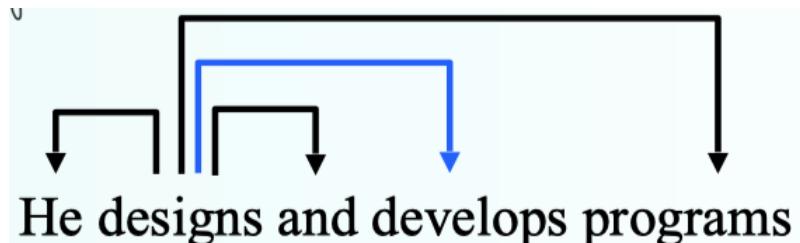
Ci sono alcuni problemi che vanno considerati, noi dobbiamo creare un albero e in questo albero deve esserci un nodo root. Ogni nodo dovrebbe essere connesso senza andare a creare dei cicli e ogni nodo dovrebbe avere solamente una Head. Con tutti questi vincoli abbiamo difficoltà ad esprimere alcuni fenomeni linguistici.

Poi ci sono alcuni casi anomali che vanno considerati:

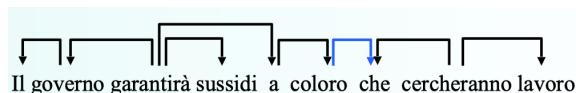


qua il problema è che abbiamo come dipendenza He-<Develops e He-<-Design con "He" che diventa il soggetto di questo verbo, in realtà qua noi dovremmo considera che "He" non è il soggetto dei due verbi presi separatamente ma è il soggetto di entrambi i verbi presi come congiunzione.

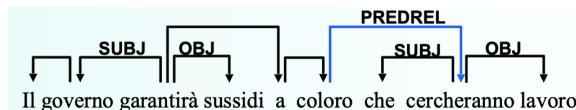
Per questo problema ci sono varie possibili soluzioni, una consiste nel connettere i due verbi, in questo modo riusciremmo comunque a ricostruire la frase navigando all'interno dell'albero.



Un problema simile l'abbiamo qua con una identificazione errata del soggetto del verbo cercheranno:



qua il problema può essere risolto quando vediamo che il verbo principale è "cercheranno" che in realtà è connesso con coloro che fa da soggetto.



9.5.2 Problemi con gli Oracoli

Gli oracoli in principio fanno la cosa giusta ma i parser non sono perfetti e quindi in alcuni casi potrebbero suggerire alcuni percorsi che non sono corretti e l'albero che otteniamo alla fine potrebbe non essere corretto. Nel momento in cui facciamo un errore con il parser, questo si troverà in uno stato che non ha mai visto durante la fase di training e quindi non saprà neanche come uscirne, questo causa una propagazione degli errori.

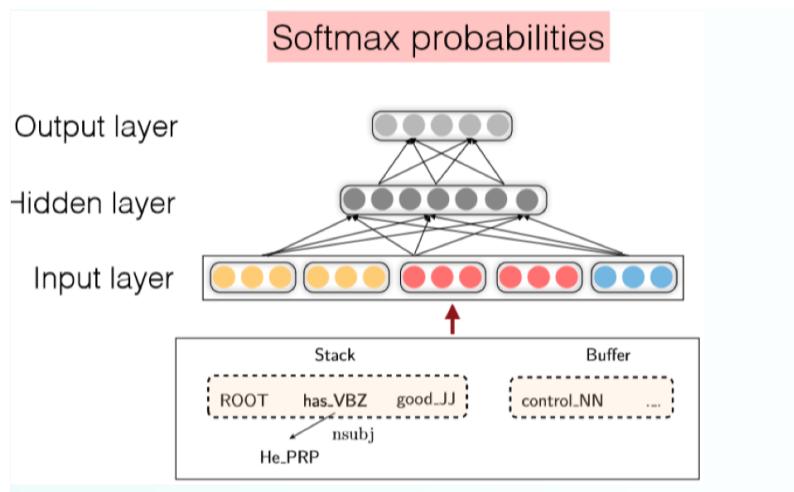
Con gli oracoli visti fino ad ora potremmo anche avere dei problemi per quel che riguarda le ambiguità.

Una possibile evoluzione degli oracoli che abbiamo visto fino ad ora consiste nell'utilizzo degli oracoli dinamici che sono oracoli che consid-

erano più di una azione alla volta e svolgono previsioni corrette in ogni possibile configurazione.

9.6 Dependency Parser con le Neural network

Nel paper "A fast and accurate dependency parser using NN" Chen e Manning presentano la loro soluzione che consiste nell'utilizzo delle neural network per il parsing. L'approccio utilizzato è simile a quello che abbiamo già descritto con un meccanismo transition based di shift e reduce. La differenza rispetto ai metodi visti fino ad ora sta nel classificatore. Invece di utilizzare feature che sono create manualmente utilizza degli embedding.



Nell'immagine sopra abbiamo uno schema del funzionamento di questo metodo, in particolare in input abbiamo lo stack e il buffer, come nei metodi visti fino ad ora, prendiamo ogni rappresentazione di questi token e la diamo in pasto alla neural network. In questo caso abbiamo

in input 3 parole nello stack e possiamo vedere che "He" è già stata collegata con "has".

In input abbiamo degli embedding che vengono concatenati per creare la rappresentazione dei dati in input. Tutti gli embedding sono creati a partire dal contesto e per ogni embedding abbiamo un vettore.

	Word	POS	dep
S1	good	JJ	0
S2	has	VBZ	0
B1	control	NN	0
lc(S1)	0	0	0
rc(S1)	0	0	0
lc(S2)	He	PRP	nsubj
rc(S2)	0	0	0

All'interno della neural network abbiamo un Dense Layer.

L'output layer della neural network è una softmax che assegna la probabilità alle varie possibili azioni.

Ci sono studi in corso su queste tematiche che comprendono l'utilizzo delle neural network per il dependency parsing, l'obiettivo è utilizzare reti neurali sempre più profonde con iperparametri migliori su cui abbiamo fatto meglio anche il tuning. C'è poi anche l'idea di utilizzare la Beam Search o le Bidirectional LSTM.

Per quanto riguarda lo stato dell'arte abbiamo che la soluzione migliore per il dependency parsing attualmente è un graph based parser, negli ultimi due anni si è iniziato ad utilizzare gli embedding.

9.7 Graph Based Dependency Parsing

L'approccio Graph Based per il dependency parsing ricerca, attraverso lo spazio dei possibili alberi che possiamo generare data una singola frase,

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

l’albero che massimizza lo score. Questi metodi codificano lo spazio in cui ricerchiamo l’albero come un grafo diretto e utilizzano metodi presi dalla teoria dei grafi per trovare la soluzione ottima.

Lo score complessivo che possiamo ottenere con questo metodo è funzione dello score delle singole parti dell’albero stesso.

Ci sono varie motivazioni che potrebbero portarci ad utilizzare un metodo di questo genere:

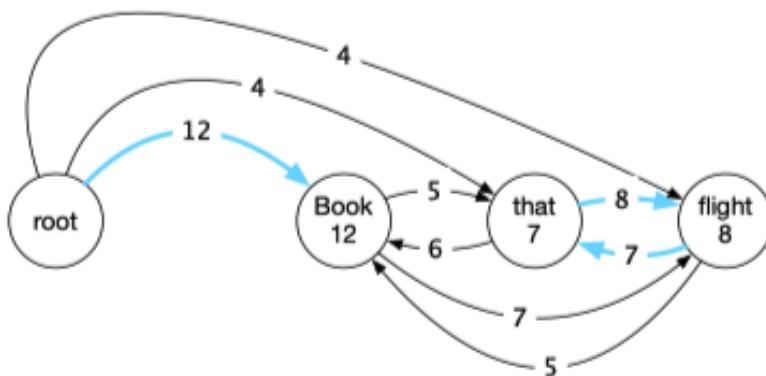
- Rispetto ai classici approcci transition based è un metodo capace di produrre anche alberi non projective. Avere alberi non projective non è un grande problema per lingue come l’inglese ma potrebbe essere un problema per altri linguaggi.
- L’accuracy è migliore rispetto a quella che abbiamo con i metodi transition based specialmente se le dipendenze che dobbiamo andare a trovare sono molto lontane tra loro nel testo. Le soluzioni transition based hanno un’ottima accuracy quando le dipendenze sono vicine tra loro ma poi le performance peggiorano a mano a mano che sono più lontane. I graph based risolvono il problema perché lo score viene deciso in base a tutto l’albero e non solamente in base a decisioni locali.

L’idea basilare di questo metodo basato sui grafi è quella di arrivare ad una situazione in cui possiamo generare un maximum spanning tree partendo da un grafo. Per generare il Maximum Spanning tree vediamo quali sono le proprietà di un Maximum Spanning Tree:

- Ogni vertice di uno spanning tree deve avere solamente un arco entrante.
- Il valore del peso degli archi che conta veramente è il peso degli archi che entrano in ognuno dei vertici, non è il valore assoluto degli score degli archi.

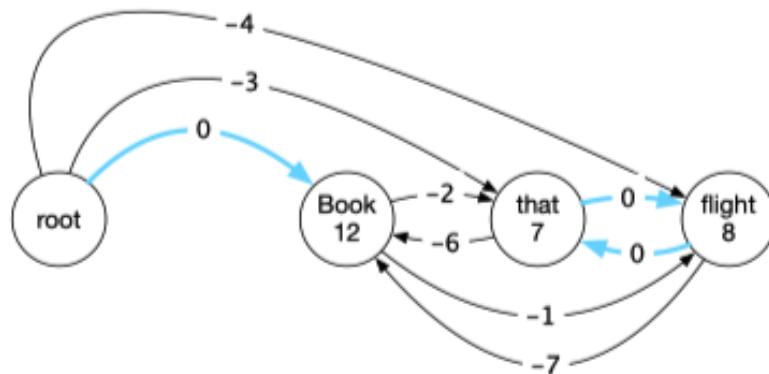
Come funziona l'algoritmo:

- Il primo step dell'algoritmo è semplice, noi partiamo con un grafo in cui abbiamo i nodi che rappresentano tutte le parole della frase che stiamo considerando e poi abbiamo gli archi che rappresentano le dipendenze. Per ogni arco che abbiamo dobbiamo assegnare un peso.
- Partendo con questo grafo quello che possiamo fare è scegliere per ogni vertice l'arco di peso maggiore e creare quindi lo spanning tree. Se è valido anche il vincolo del singolo arco entrante in ogni nodo abbiamo ottenuto un Maximum Spanning tree e finisce qua.
- Purtroppo però quello che succede è che spesso possiamo ottenere dei loop che fanno sì che non si possa ottenere un maximum spanning tree.

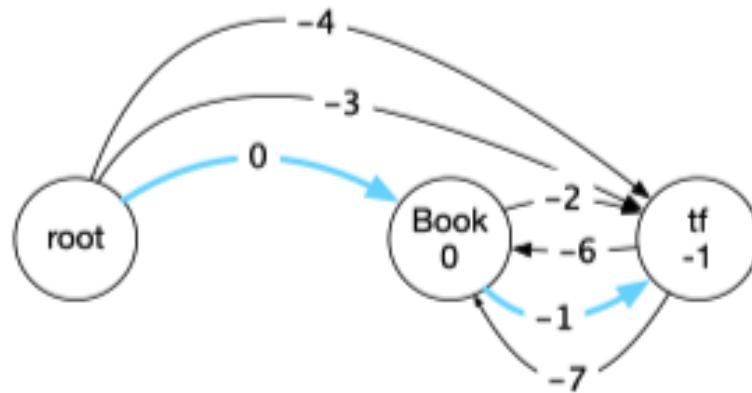


- Per risolvere il problema in questi casi quello che possiamo fare è eseguire un algoritmo che permette di eliminare questi loop:
 - Quello che si fa è considerare il massimo peso che abbiamo per ogni arco entrante nei nodi del grafo e poi sottraiamo questo

valore massimo a tutti gli archi (questo va fatto per ogni arco, quindi avremo un valore diverso per ogni arco).



- Fatto il primo passaggio collassiamo la coppia di nodi che formano il ciclo in un singolo nodo, i collegamenti che erano presenti verso i due nodi collassati ora vanno tutti verso il nuovo nodo collassato.
- Prendiamo tra tutti questi archi quello di peso maggiore, questo mi dice come raggiungo uno dei due nodi del loop.



- Una volta che ho identificato questo nodo posso eliminare il nodo collassato e rimettere i due che avevamo all'inizio, quello

che si può fare ora è vedere verso quale dei due nodi collassati va l'arco di peso maggiore, in questo caso va verso flight, questo mi permette di eliminare l'arco del ciclo che arriva al nodo flight e di tenere quello che va verso il nodo that.

- Il processo di eliminazione dei cicli può essere svolto varie volte fino a che non si arriva ad un grafo senza cicli che quindi possiamo trasformare in un maximum spanning tree.

Il problema di questo algoritmo è la complessità che è $O(n^2)$ questo perchè dobbiamo considerare tutte le possibili connessioni e dobbiamo ripetere il processo più volte.

Abbiamo anche la possibilità di utilizzare un graph based parser che utilizza le Neural Network, in questo caso abbiamo risultati che sono ottimi, abbiamo un accuracy del 94% che è una performance migliore rispetto a quelle che abbiamo visto fino ad ora. Il problema riguarda sempre la lentezza rispetto ad un transition based parser che usa le neural network.

Per il Graph based parser basato sulle Neural network abbiamo una architettura simile a quella che abbiamo visto fino nel caso transition base. Si parte anche qua con un primo layer che prende i vari embedding poi abbiamo vari layer. L'output produce due rappresentazioni nascoste, una per la testa e una per le dipendenze. Quello che vogliamo produrre è una previsione, vogliamo il miglior modo per assegnare. In particolare i layer sono:

- Abbiamo due Fully Connected Relu Layer
 - Uno rappresenta ogni nodo come dipendenza cercando di determinare il label corrispondente.
 - Uno rappresenta ogni token come head cercando di determinare il label delle sue dipendenze.

- Abbiamo due Relu Layer separati:
 - Uno rappresenta ogni token come dipendenza cercando di trovare la Head corrispondente.
 - Uno rappresenta ogni token come Head cercando di trovare le sue dipendenze.
- Abbiamo un layer di Self-Attention che cerca di calcolare lo score delle possibili Head per ognuna delle varie matrici di dipendenza. La matrice delle dipendenze è una matrice $n * n$ molto grande per le varie frasi. In pratica facciamo quello che avremmo dovuto fare in precedenza ma ora lo facciamo per tutte le frasi che stiamo considerando, l'idea è semplice ma è più complicata.

$$\begin{array}{cccc}
 s_i^{(arc)} & H^{(arc-head)} & W \oplus b & h_i^{(arc-dep)} \oplus 1 \\
 \begin{matrix} \text{---} \\ \text{---} \\ \text{---} \\ \text{---} \end{matrix} & \begin{matrix} \text{---} \\ \text{---} \\ \text{---} \\ \text{---} \end{matrix} & \begin{matrix} \text{---} \\ \text{---} \\ \text{---} \\ \text{---} \end{matrix} & \begin{matrix} \text{---} \\ \text{---} \\ \text{---} \\ \text{---} \end{matrix} \\
 \begin{matrix} \text{---} \\ \text{---} \\ \text{---} \\ \text{---} \end{matrix}^T = & \begin{matrix} \text{---} \\ \text{---} \\ \text{---} \\ \text{---} \end{matrix} \cdot & \begin{matrix} \text{---} \\ \text{---} \\ \text{---} \\ \text{---} \end{matrix} \cdot & \begin{matrix} \text{---} \\ \text{---} \\ \text{---} \\ \text{---} \end{matrix}^T \\
 s_i = H^{(arc-head)}(W h_i^{(arc-dep)} + b) & & & \\
 H^{(arc-head)} = [h_1^{(arc-head)} | \dots | h_n^{(arc-head)}] & & &
 \end{array}$$

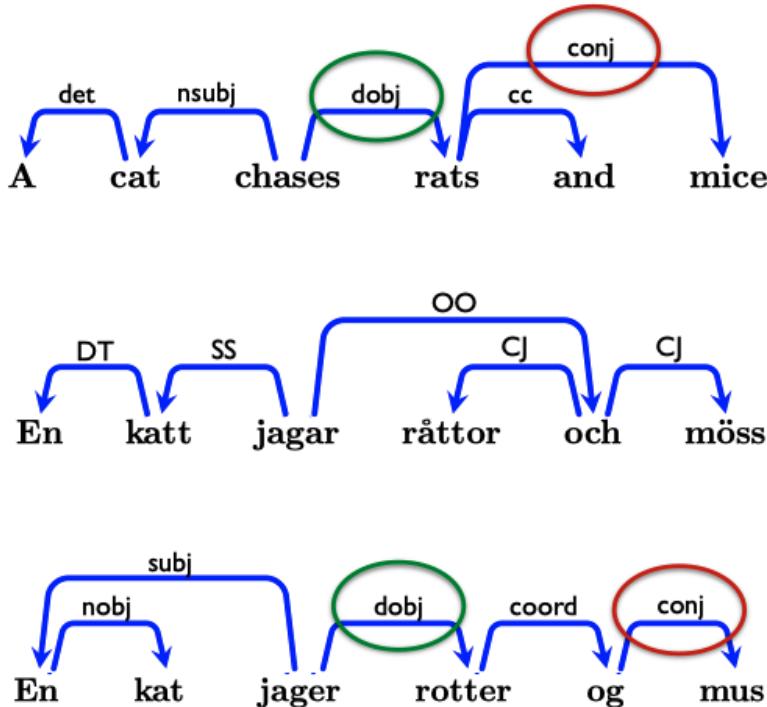
Chapter 10

Lesson 9 - Universal Dependency

L'Universal Dependency è un progetto che cerca di sviluppare un treebank di annotazioni che sia funzionante e consistente con vari linguaggi differenti. L'obiettivo è facilitare lo sviluppo di parser multilingua, supportare ricerche nel campo della NLP multilingua e della linguistica.

All'interno di questo progetto sono stati rilasciati vari treebanks, Stanford ha rilasciato quello delle Universal Dependencies, Google quello dei POS tags e poi ne è stato rilasciato uno per quel che riguarda le feature morfologiche delle varie parole delle frasi.

Attualmente c'è un forte interesse per un NLP che sia multilingua, allo stesso tempo ora il Natural Language Processing si basa ancora sulle annotazioni che possono variare tra i vari linguaggi.



Possiamo vedere nell'esempio qua sopra una stessa frase tradotta in vari linguaggi, abbiamo che le principali relazioni grammaticali sono sempre le stesse ma abbiamo però delle variazioni per quel che riguarda la grammatica concreta all'interno delle frasi.

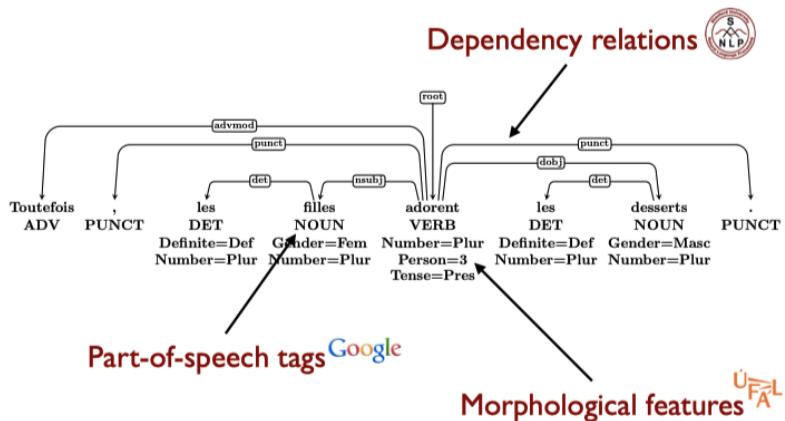
Questo è un problema perchè è difficile eseguire un confronto dei risultati ottenuti con linguaggi differenti, è difficile valutare il learning con varie lingue differenti, è difficile creare un sistema multilingua e fare quindi progressi in un parser universale che funzioni indipendentemente dalla lingua in cui è scritto il testo.

I principi basilari di Universal Dependency:

- Vogliamo massimizzare il parallelismo ma senza esagerare, non vanno annotate le stesse cose in modi differenti e non dobbiamo fare in modo che parti differenti sembrino uguali.

- Non vogliamo annotare le cose che non ci sono e dobbiamo permettere estensioni specifiche per i vari linguaggi.
- Una assunzione che possiamo fare è considerare che le "unità" che annotiamo sono le varie parole, questo vuol dire che il parsing non sarà ovvio, ad esempio una parola in italiano ("Diglielo") può diventare una frase in inglese ("Say that to him"). L'idea che si usa con le universal dependency è di dividere la parola originale in parti più piccole, la relazione è tra le varie parole.
- Si utilizzano le dipendenze, usate moltissimo nei sistemi NLP e disponibili in treebanks per molti linguaggi.
- A partire dalle annotazioni è possibile recuperare il testo originale.

Un esempio di Universal Dependency:



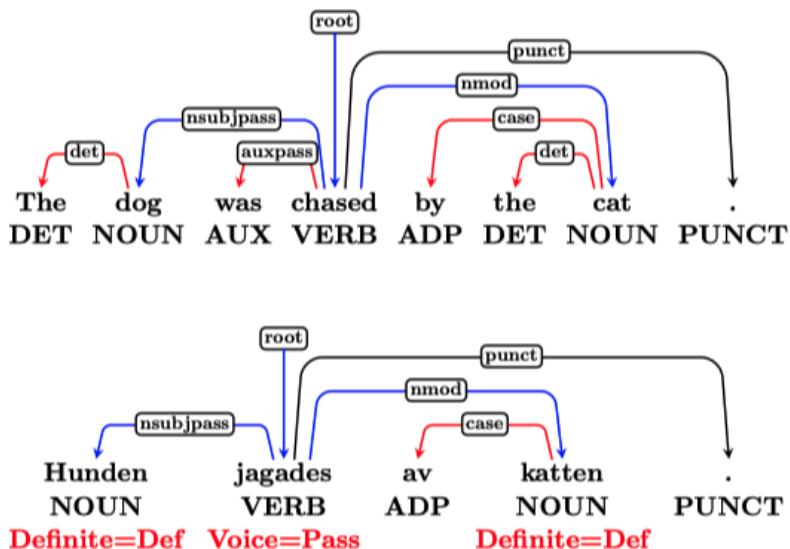
Cosa troviamo quindi all'interno di questa universal dependency:

- Abbiamo annotazioni morfologiche:
 - Lemma: per ogni parola scriviamo il lemma che rappresenta il contenuto "semantico" della parola.

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

- Abbiamo la Part of Speech tag associata con la parola. Ci sono 17 Universal pos tag basati sul Google Universal Tagset.
- Consideriamo anche le features che rappresentano proprietà lessicali e grammaticali associate con il lemma o con la forma particolare di quella parola.
- Abbiamo poi le annotazioni sintattiche:
 - Ci sono le dependency relations. Vengono utilizzate 42 relazioni grammaticali standard e in più è permesso specificare delle dipendenze specifiche per ogni linguaggio.
 - Alcune parole vengono collegate alla parola che vanno a modificare.
 - La punteggiatura viene collegata alla Head della frase.

Se abbiamo frasi differenti in differenti linguaggi la situazione è la stessa, qualsiasi analisi può essere effettuata su entrambi i linguaggi.



Chapter 11

Lesson 10 - Opinion Mining e Sentiment Analysis

Esistono due tipologie di informazioni testuali, fatti ed opinioni, con i fatti che spesso implicano anche le opinioni. I più recenti metodi di processing del testo, ad esempio per il text mining o per le ricerche nel web funzionano tramite informazioni derivanti dai fatti, c'è però un'altro campo in cui contano di più le opinioni ed è quello della sentiment analysis.

Con sentiment analysis o opinion mining indichiamo lo studio di opinioni, sentimenti ed emozioni presenti in un testo. Perchè ci interessiamo di opinion mining proprio ora? Le persone si fidano del web e dei social media quindi si possono utilizzare questi sistemi per influenzare il pensiero delle persone, è un problema molto delicato.

Le opinioni delle persone sono importanti perchè sono alla base del prendere decisioni, se in passato gli individui avevano delle opinioni che arrivavano da amici e famigliari ora le opinioni possono arrivare da social network, da forum, gruppi online, quindi le opinioni ora sono su larga

scala e non sono più focalizzate solamente su un piccolo gruppo di persone. Se sei un politico ad esempio potresti scrivere la tua opinione sul web e poi riceverai centinaia di reazioni che sono d'accordo con te o che non sono d'accordo con te.

Ci sono varie possibili applicazioni per la sentiment analysis:

- Il marketing è una parte importante del budget di una azienda, ci sono compagnie che utilizzano la sentiment analysis per capire se un loro prodotto viene apprezzato oppure no.
- I singoli individui potrebbero pensare di volersi fare una idea della qualità di un prodotto andando a guardare cosa gli altri pensano di quel prodotto.
- Sempre dal punto di vista del marketing si può pensare di voler fare una certa pubblicità ad un prodotto sui social network cercando di avere come target la persona che potenzialmente potrebbe essere più interessata a quel determinato prodotto.
- In generale possiamo anche pensare di voler ottenere l'opinione delle persone, potremmo voler creare un search engine che sia in grado di farmi capire cosa le persone pensano.

Questa della sentiment analysis è un'area che in alcuni casi si sovrappone alla social science. Ad esempio potremmo voler estrarre ed analizzare opinioni politiche che riguardano candidati e problematiche varie, oppure potremmo voler considerare vari differenti gruppi di persone e capire come cambia il pensiero tra questi gruppi. Le opinioni (specialmente le fake opinion) sono in grado di modificare il pensiero delle persone e soprattutto quello che succede sul web e le opinioni che la gente ha sul web è probabilmente legata alle opinioni che quelle stesse persone avranno nella vita reale, quindi vorremmo anche capire in che modo le opinioni sul web influenzano le opinioni nella vita reale.

11.0.1 Il caso Cambridge Analytica

Tutto è iniziato con l'idea di creare una misurazione dei tratti psicologici delle persone. Il problema era ottenere i dati per fare queste analisi quindi è stata lanciata una applicazione su Facebook da Michal Kosinski in cui si cercava di andare a delineare i tratti psicologici di una persona.

L'idea era di creare all'interno dell'applicazione un questionario in cui le persone potessero andare a rispondere a domande psicologiche. Alla fine l'utente riceveva un profilo della personalità e sceglie se condividere i dati di Facebook con l'applicazione. Con questo approccio Kosinski è riuscito a delineare una personalità psicologica delle varie persone che hanno svolto il test e poi ha confrontato i risultati con tutti i vari dati che riusciva a trovare online relativi a quella persona. Per quanto riguarda la psicologia della persona si è basato sui 5 grandi tratti della personalità di una persona (OCEAN) ovvero Openness to experience, Conscientiousness, Extraversion, Agreeableness, Neuroticism.

Il risultato di tutto questo studio è che mettendo in relazione i dati ottenuti dal questionario con i dati presi da facebook e relativi a pagine a cui la persona aveva messo il like o relativi ai post condivisi si è riusciti a trovare una correlazione tra tutte queste informazioni. Tutte queste informazioni prese singolarmente sono troppo povere per produrre una previsione accurata ma quando tante persone forniscono questi dati e tutti questi dati vengono combinati allora si riescono ad ottenere delle previsioni che sono molto accurate.

Dato che molte informazioni erano ottenute da Facebook e in particolare dai like (cosa legale fino a quel punto e in ogni caso chiedeva il consenso), Facebook minacciò Kosinski di denunciarlo e gli offrì un lavoro.

Tutto lo studio svolto da Kosinski partiva dai dati degli utenti e riusciva a delinearne un profilo psicologico, il problema è che tutto questo poteva anche portare ad un search engine delle persone. Era infatti possibile partire da una serie di caratteristiche psicologiche di una persona

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

e trovare quelle persone che avessero quelle caratteristiche. Allo stesso modo poteva essere possibile partire dai dati di Facebook di una persona per delinearne la psicologia. Questo era un problema perché in questo modo si andava a creare un search engine in cui potevamo andare a cercare tutte le persone con un determinato tipo di comportamento.

Questa reverse search è quella che è stata utilizzata ad esempio da Cambridge Analytica, normalmente durante una campagna pubblicitaria si invia lo stesso messaggio a tutti quanti, l'idea di Cambridge Analytica è stata quella di inviare messaggi differenti utilizzando delle metriche psicologiche.

In particolare erano stati in grado di creare un modello che potesse predire la personalità di ogni singolo adulto americano. Cambridge Analytica comprò dati riguardanti le persone da varie fonti, all'epoca la cosa era legale e li aggregò in un unico modello che potesse permettere di tracciare una impronta digitale di ogni persona con interessi, paure, necessità e anche indirizzi di residenza. In questo modo le campagne pubblicitarie potevano essere create in modo da avere un messaggio differente per ogni possibile votante, in base a vari criteri tra cui anche la posizione geografica. Il risultato è una campagna di pubblicità che ha più effetti rispetto ad una normale campagna di pubblicità.

Questo chiaramente era un problema per la privacy, negli USA ad esempio non avevano la possibilità per l'utente di bloccare l'utilizzo dei dati personali. Cambridge Analytica ha suddiviso gli americani in 32 tipi di personalità focalizzandosi solamente in 17 stati.

Alla fine Facebook è stata solamente l'arma per questa campagna e Cambridge Analytica non ha fatto niente di illegale, così come Kosinski ha solamente fatto vedere che il metodo esisteva senza però costruire realmente la "bomba" che poi ha portato all'uso di questi sistemi in campagne pubblicitarie.

11.1 Sentiment Analysis

La sentiment Analysis è diventato un argomento di ricerca molto importante negli ultimi anni con tante aziende che sono nate per lavorare su questi temi. È un argomento che tocca ogni aspetto della NLP e potenzialmente è la tecnologia più importante che possiamo ottenere dalla NLP ma non è ancora troppo semplice da effettuare soprattutto perché trovare dati e integrarli non è semplice.

Il problema della Sentiment Analysis può essere suddiviso in due parti:

- Dobbiamo poter definire cosa è un'opinione.
- Dobbiamo poter fare un "riassunto" delle opinioni, considerare una sola opinione non serve a nulla quindi dobbiamo considerare opinioni di più persone e poi saperle "riassumere".

Se consideriamo una frase, quali sono le parti che potrebbero interessarci in termini di sentiment analysis:

- Abbiamo il target dell'opinione ovvero l'oggetto o la persona riguardo cui stiamo esprimendo una opinione.
- Abbiamo un sentimento verso quella cosa/persona che può essere positivo o negativo.
- Abbiamo l'opinion holders che è la persona che esprime la sua opinione.
- Possiamo anche considerare il tempo in cui è stata espressa l'opinione, questo può essere importante perché possiamo vedere in che modo si modifica l'opinione di una persona durante il tempo.

Ci sono varie tipologie differenti di opinioni che possiamo considerare:

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

- Opinioni regolari: sono di due tipologie differenti e noi considereremo in modo particolare questo tipo di opinioni:
 - Abbiamo le opinioni dirette in cui noi indichiamo specificamente direttamente la nostra opinione riguardo ad un certo oggetto.
 - Ci sono le opinioni indirette in cui abbiamo un fatto che causa la mia opinione.
- Opinioni comparative: in questo caso noi facciamo un confronto tra varie entità.

11.1.1 Un modello per la Sentiment Analysis

Quando parliamo di sentiment analysis dobbiamo partire subito definendo un concetto base che è il concetto di entità, si tratta di una idea più generale rispetto a quella che abbiamo nella named entity recognition.

Una entità (chiamata anche object) è il prodotto o la persona o l'argomento per cui sto esprimendo una opinione. Data questa entità possiamo poi creare una gerarchia (rappresentabile in un albero) in cui abbiamo che l'entità è formata da componenti e sotto componenti, ognuno di essi ha anche degli attributi. Per rappresentare l'insieme di componenti e di attributi usiamo più genericamente la definizione di "aspects" (chiamati anche feature). L'opinione la posso poi esprimere riguardo ai nodi di questo albero o anche riguardo agli attributi.

11.1.2 Direct Opinion

Possiamo definire la direct opinion con una quintupla, dobbiamo considerare tutte le informazioni contenute in questa quintupla per poter produrre una previsione. La quintupla è composta da:

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

$$(o, f, s, h, t)$$

dove abbiamo i seguenti componenti:

- o è l'oggetto target.
- f è una feature specifica dell'oggetto o.
- h è l'opinion holder ovvero colui che sta esprimendo un'opinione.
(lo possiamo chiamare anche opinion source)
- t è il tempo in cui l'opinione è stata espressa.
- s è il sentiment, indica il valore dell'opinione di h riguardo alla feature f dell'oggetto o. s può essere positiva, negativa o neutrale.
Se usiamo un rating più complesso può anche avere un valore differente.

Partendo con un testo il nostro obiettivo è quello di convertire il testo non strutturato in un testo strutturato, per farlo dobbiamo estrarre le quintuple dal testo.

Quindi, partendo dal testo vorremmo estrarre per ognuna delle frasi del testo una quintupla del tipo (o, f, s, h, t) che mi possa rappresentare la corrispondente informazione presente nel testo. Mettendo insieme tutte le quintuple che abbiamo a disposizione per un certo testo l'obiettivo sarebbe quello di classificare il sentiment dell'intero documento. Con l'insieme delle quintuple quindi passiamo da un testo non strutturato ad un testo con dati strutturati che mi possono permettere un'analisi quantitativa e qualitativa.

Document Sentiment Classification

Vogliamo classificare un certo documento (ad esempio una recensione) basandoci sul sentiment espresso dall'opinion holder, le possibili classi sono "Positivo", "Negativo" e "Neutrale".

Per fare questa analisi possiamo considerare il nostro modello con la quintupla (o, f, s, h, t) e facciamo delle assunzioni, consideriamo infatti che ogni documento si focalizza su un singolo argomento e contiene opinioni di un singolo opinion holder. Inoltre consideriamo le opinioni su un singolo oggetto o.

Per eseguire la sentiment analysis è necessario saper fare una distinzione tra una frase oggettiva e una frase soggettiva, in particolare dovremmo saper distinguere tra:

- Frase soggettiva: una frase in cui riportiamo direttamente da un'opinione di chi scrive, non sempre è solamente una opinione solamente positiva o solo negativa, ad esempio "Penso siano arrivati ieri" è una frase soggettiva ma non ha una accezione positiva o negativa. Per le frasi che sono soggettive (quando esprimono un sentimento) dobbiamo essere in grado di fare una classificazione con le classi positiva e negativa. Un'altra cosa che dobbiamo considerare per le frasi soggettive è che non sempre la frase esprime una emozione di chi scrive, in alcuni casi potremmo semplicemente esprimere la nostra idea senza però indicare una emozione, possiamo notare la differenza se consideriamo queste due frasi: "Il volume di questo telefono è alto" è una frase in cui esprimiamo una opinione ma senza indicare la nostra emozione, se invece scrivo "Mi piace questo telefono" indichiamo anche l'emozione oltre all'opinione di apprezzamento.
- Frase oggettiva: è un qualcosa di vero e non viene riportato secondo l'opinione di chi scrive quella frase. È anche vero che le frasi oggettive possono però implicare una opinione anche se non viene

indicata in modo chiaro nella frase.

11.1.3 Feature Based Sentiment Analysis

Ammettiamo di avere una recensione di un telefono, avremmo che il target della sentiment analysis è questo telefono che è composto da vari componenti. Non possiamo accontentarci di utilizzare semplicemente una singola quintupla alla volta per ottenere una classificazione di tutto il documento. Il recensore in questo caso potrebbe aver apprezzato alcune delle feature del prodotto e altre no, quindi non possiamo considerarla solamente una ed estendere la classificazione positiva/negativa all'intero documento.

La soluzione a questo problema è che dato un documento non devo trovare solamente una quintupla ma ne devo trovare tutte, una volta che ne ho trovate tutte allora qualsiasi tipo di analisi diventa possibile.

Quando abbiamo trovato le varie quintuple che quindi rappresentano ognuna una certa opinione, è necessario dare un senso a queste opinioni. Il processo di prendere tante opinioni e poi dargli un senso viene chiamato "Opinion Summary". Ad esempio alcuni fatti potrebbero avere un peso più alto degli altri, inoltre alcune opinioni potrebbero essere ripetute e per evitare ripetizioni dovremmo eliminarle (dobbiamo considerare un solo fatto, indipendentemente da quante opinioni compaiono).

Un'altra cosa che possiamo considerare è il feature based summary, abbiamo il nostro testo e potremmo essere interessati ad alcune specifiche feature del prodotto, ad esempio se consideriamo una recensione di un cellulare potremmo considerare come feature lo schermo e la batteria, avremmo quindi due feature separate e per ciascuna di esse andiamo a classificare le varie frasi come positive o negative dando pesi differenti a ciascuna di queste frasi.

Una volta che abbiamo fatto questa divisione in base alle feature possiamo anche procedere con un confronto tra vari dispositivi, potremmo infatti considerare più di una recensione (ognuna relativa ad un prodotto

differenti) e poi per ogni recensione estrarre le varie feature con le frasi positive e negative.

Queste analisi delle feature possono anche essere ripetute nel tempo se con il passare del tempo si aggiungono altri documenti da analizzare. In questo modo possiamo andare a capire in che modo sono cambiate nel tempo le opinioni degli utenti riguardo ad un certo dispositivo.

Il campo della sentiment analysis è in continua espansione e infatti ci sono ancora tante problematiche che devono essere affrontate e risolte, ad esempio il problema delle co-reference resolution, dell'estrazione di relazioni tra le varie parole e frasi e il match dei sinonimi. Ci sono in ogni caso problemi più complessi e altri più semplici da risolvere, i tweet sono più facili da analizzare perché sono frasi più corte rispetto ad esempio ad una recensione. I testi più complessi sono le discussioni nei forum o nei blog perché ci sono spesso varie entità di cui si parla e spesso abbiamo anche confronti e sarcasmo.

11.1.4 Aspect based Sentiment Analysis

In questa specifica area di interesse tornano utili le dependency parsing che possono essere utilizzate per riconoscere le strutture dipendenti tra loro in una frase e anche per task come machine translation o score classification. Fino ad ora abbiamo parlato di feature based sentiment analysis in cui consideriamo un oggetto e tutte le varie feature che lo compongono (telefono-schermo-batteria) andando poi a fare un riassunto di tutte le feature e delle corrispondenti quintuple, ora possiamo arrivare a parlare di aspect based sentiment analysis.

Noi vorremmo capire di quale aspetto di sta parlando e quale sentimento viene espresso, per le recensioni dei film per esempio possiamo immaginare che abbiamo un singolo oggetto del nostro discorso e quindi ogni recensione si riferirà a quell'oggetto, l'utente che scrive la recensione esprimerà dei pensieri riguardo ai vari aspetti di questa entità. Quindi è più semplice rispetto ad un testo in cui abbiamo varie entity (vari

prodotti o varie parti del prodotto) perchè qua la entity è una sola.

In ogni caso anche qua è necessario saper trovare la entity di cui si sta parlando, sotto alcuni aspetti è simile a quello che si fa con il NER ma non è proprio del tutto identico.

Oltre ad estrarre la entity di cui stiamo parlando nel testo dobbiamo anche cercare di estrarre le feature che stiamo considerando, ci sono vari metodi che possiamo utilizzare:

- Possiamo usare nomi che compaiono più frequentemente di altri.
- Dipendenze presenti all'interno del testo.
- Possiamo anche utilizzare metodi standard dell'information extraction come ad esempio Rule Based, il supervised learning o anche gli Hidden Markov Models.

Double Propagation

Questo metodo cerca di sfruttare la struttura delle frasi e in modo particolare il fatto che all'interno delle frasi sono presenti delle relazioni. Le relazioni all'interno delle frasi le troviamo tra le opinioni e le varie feature e possono essere utilizzate per estrarre proprio queste feature, ad esempio la frase "Questa fotocamera scatta belle foto" contiene "belle foto" che è una opinione che è chiaramente legata con la fotocamera.

L'algoritmo viene utilizzato quindi per estrarre le features, in particolare abbiamo in input un set di parole e poi viene creato un albero delle dipendenze. Noi possiamo navigare all'interno di questo albero basandoci sulle regole della dependency grammar.

Una cosa che dobbiamo saper fare è il raggruppamento delle feature che sono sinonimi tra loro, ci sono varie possibili tecniche per risolvere questo problema.

Un altro problema da considerare è il coreference Resolution per la sentiment analysis che è differente rispetto al classico problema della

coreference resolution. In questo caso diventa importante saper distinguere i vari soggetti di cui si parla all'interno della frase e capire quando questi sono legati tra loro. Ad esempio consideriamo la frase: "Ho comprato una canon S500 ieri, è molto bella e ho scattato alcune foto la scorsa notte. Sono ottime". Qua dobbiamo saper distinguere solamente due soggetti ed è importante saperli distinguere, abbiamo un primo soggetto che è la Canon S500 che è anche legata alla parte di frase "È molto bella", allo stesso tempo abbiamo anche il soggetto "alcune foto", a questo si lega l'opinione "sono ottime".

Per quanto riguarda la coreference resolution dobbiamo capire se due parole si riferiscono allo stesso soggetto, abbiamo una possibile soluzione proposta dal Prof. Attardi nel 2010 che fa uso del parsing e della clusterizzazione. Il metodo in questione è formato da due stage:

- Nel primo stage utilizziamo il dependency tree per identificare le varie menzioni.
- Nel secondo stage invece utilizziamo un classificatore per classificare queste menzioni.

Per la fase di training vengono creati esempi positivi ed esempi negativi a partire dai dati di training, si prendono le varie menzioni e si creano da queste istanze positive e negative accoppiandole con tutte le istanze precedenti. Da ogni coppia di menzioni vengono poi estratte alcune features che sono lessicali, che possono dipendere dalla distanza tra le varie feature (simile all'edit distance) o che possono essere anche sintattiche.

Un esempio che possiamo considerare: abbiamo una frase in cui nominiamo "Barack Obama" e poi nominiamo semplicemente "Obama", si tratta chiaramente della stessa persona. Usando questo approccio è stato creato un classificatore che funziona abbastanza bene. L'anno successivo è stata trovata una soluzione alternativa che si basa sulla ricerca di sostantivi nelle frasi, sull'identificazione delle varie proprietà di queste frasi e poi sulla clusterizzazione in varie iterazioni deterministiche.

Chapter 12

Lesson 11 - Sentiment Classification

Cominciamo a parlare di sentiment classification, supponiamo di avere il nostro testo ma di non avere degli esempi di training labellati. Questo vuol dire che stiamo considerando il caso dell'unsupervised learning dove la conoscenza riguardo ad un task viene solitamente aggiunta utilizzando risorse lessicali o euristiche oppure si utilizzano delle caratteristiche condivise e ricorrenti delle frasi per eseguire la classificazione. Sono stati sviluppati anche dei modelli di Neural network in cui possiamo eseguire la sentiment classification senza avere una conoscenza esplicita riguardo al task.

L'alternativa all'unsupervised learning è il supervised learning che è comunemente utilizzato e che ci permette di avere a disposizione un training set con esempi labellati che descrivono la corretta classificazione dei vari documenti che stiamo considerando. Partendo da questi dati di training possiamo poi generare un modello con una classification function.

12.1 Sentiment Analysis con i Tweet

Ora consideriamo il paper "Building the State of the Art in Sentiment Analysis of Tweets" [2]. È un lavoro del 2013 in cui si è arrivati allo stato dell'arte nella sentiment analysis effettuata con i tweet.

L'obiettivo del primo dei due task che sono stati implementati è quello di classificare come positiva, negativa o neutra una certa frase.

Ci sono varie possibili applicazioni della sentiment analysis, possiamo utilizzarla per tracciare le opinioni delle persone riguardo politica, film ma anche riguardo ad aziende. Riuscire a capire l'opinione delle persone è qualcosa che è importante durante una conversazione, abbiamo da capire se la persona è irritata o meno e in caso dobbiamo poter cambiare la nostra eventuale risposta. Non è un task semplice da effettuare anche perchè in alcuni casi, come ad esempio con i tweet, è anche possibile trovare del sarcasmo all'interno delle frasi.

Un'altra delle difficoltà che possiamo avere durante lo svolgimento della sentiment analysis è che non si hanno troppi dati a disposizione. Ad esempio nel caso che stiamo considerando dell'analisi dei tweet abbiamo solamente 3045 tweet per il training, 375 per il dev set e 1572 per il test set.

C'è un approccio tipico che può essere adottato per svolgere questo tipo di task.

Possiamo utilizzare una specifica risorsa come il sentiment lexicon che è una lista di parole con associato il sentimento corrispondente e un grado di sicurezza per quella associazione. Ad esempio l'utilizzo della parola spettacolare potrebbe essere associato al 90% con una sensazione di positività. Nel caso della sentiment analysis dei tweet è stato creato un lexicon con circa 14.000 parole. Questo che hanno creato è, in ogni caso, un lexicon molto piccolo e quindi possiamo provare ad espanderlo utilizzando un generatore automatico. Ad esempio potremmo tenere in considerazione anche le emoji che sono un metodo per esprimere un certo sentimento, possiamo assumere che se in un tweet troviamo una emoji

positiva allora avremo anche parole positive e viceversa. Assumendo di conoscere alcune parole a cui abbiamo assegnato una classificazione positiva o negativa, possiamo poi riuscire a creare un set di tweet che contengono queste parole. In questo caso sono state considerate 32 parole positive e 36 negative e da queste sono stati estratti 775 tweets. I tweet sono poi positivi se hanno al loro interno parole positive e negativi se hanno parole negative, per ogni tweet è possibile calcolare uno score che sarà > 0 se è positivo e < 0 se negativo.

Ci sono ovviamente delle parole che possono essere utilizzate in entrambi i contesti, se il significato della parola non è eccessivamente spostato verso una delle due classi, le differenze tra le due classificazioni saranno minime. Le parole che vengono utilizzate poi non sono solamente unigram ma sono anche bigram.

Dopo aver fatto questa prima fase di preprocessing possiamo passare alla fase in cui creiamo il classificatore, per fare il classificatore dobbiamo utilizzare delle feature che possiamo estrarre dal testo:

- Il sentiment Lexicon può essere preso come una feature.
- Dato il sentiment Lexicon possiamo considerare quante parole positive compaiono in un tweet e tra queste possiamo considerare la massima positiva.
- Possiamo usare le parole unigram come feature.
- Così come possiamo usare le parole bigram come feature.
- In alcuni casi le parole devono essere tagliate.
- Possiamo utilizzare il pos tagging per capire come è formata la frase che stiamo considerando.
- Sono importanti anche le negazioni.

- Possiamo anche creare dei cluster di parole.
- Ci sono anche alcuni modi speciali per scrivere alcune parole, ad esempio se scriviamo con il caps lock oppure se utilizziamo delle parole allungate per stressare ancora di più il concetto.

Tra tutte queste feature che possiamo prendere in considerazione durante l'analisi dei tweet si è cercato di capire quali fossero le feature che contribuiscono maggiormente allo score finale.

Il primo task da implementare per quel che riguarda i tweet era a livello di Messaggi, il secondo task a livello di termini presenti nel messaggio.

Per quanto riguarda questo secondo task loro hanno utilizzato lo stesso approccio utilizzato per il primo task che abbiamo descritto fino ad ora.

Con questa soluzione sono riusciti ad ottenere lo stato dell'arte nel 2013 per quel che riguarda la sentiment analysis dei tweet.

Lo sforzo maggiore ce l'abbiamo all'inizio quando deve essere creato il lexicon, questo ha sia una parte automatica sia una parte manuale. Poi successivamente un'altra parte complessa consiste nel selezionare le feature che possono aiutare.

12.2 Deep Learning per la sentiment analysis

Anche per la sentiment analysis sono state testate delle neural network e in particolare si parla di Convolutional Neural Network. Le CNN sono state utilizzate inizialmente per l'image processing, l'idea è di suddividere l'immagine in blocchi più piccoli e poi per ogni blocco si svolge un'analisi. La convoluzione ha delle sovrapposizioni ed è utile per rappresentare aspetti dell'immagine che sono dipendenti dalla posizione perché la CNN può rilevare queste dipendenze.

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

Anche in NLP è possibile utilizzare queste CNN. Ogni layer di questa CNN è detto Convolutional layer ed è composto da un set di filtri che combina una parte dei dati in input in un valore in output. Ognuno di questi filtri viene fatto "scorrere" su tutti i dati in input. Ogni filtro è specializzato nel riconoscimento di determinate feature.

Le CNN possono essere utilizzate anche per il sentiment classification:

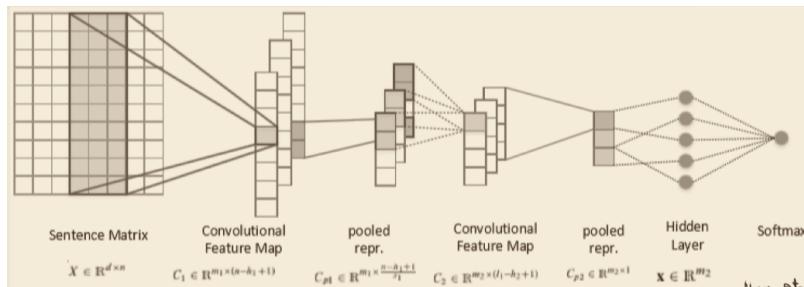
- Abbiamo un testo da analizzare in input e la prima cosa che facciamo è un embedding layer, creiamo una matrice (con una riga per ogni parola) in cui memorizziamo gli embedding per ognuna delle possibili parole.
- Abbiamo poi il Convolutional layer con ReLU come activation function. In questo layer abbiamo in particolare due filtri che vengono utilizzati sulla embedding matrix, ognuno dei filtri considera due righe della matrice alla volta e produce ogni volta un input. Il numero 2 delle righe che consideriamo ogni volta è variabile.
- Partendo dalla rappresentazione del convolutional layer con ReLU possiamo fare lo squeeze ed arrivare ad un vettore. Su questo applichiamo il Max Pool Layer.
- Possiamo anche utilizzare un layer per il dropout e un layer lineare con activation function tanh.
- Alla fine abbiamo bisogno del layer softmax che si prende in input l'output del layer precedente e va a produrre una classificazione positiva o negativa per la frase che abbiamo preso in considerazione.

Tutto questo sistema non funziona troppo bene perchè non abbiamo grandi quantità di dati quindi dobbiamo trovare un modo differente che sia utilizzabile.

Training degli embedding

Quello che possiamo notare è che il word embedding che utilizziamo in questa CNN non sa nulla riguardo ai sentiment che vengono espressi nella frase, l'idea quindi è di creare dei sentiment-aware word embedding. Come si può fare? Possiamo usare una CNN per ridefinire gli embedding, prendiamo i nostri dati non labellati e poi un insieme di tweet che trattiamo come esempi positivi (contengono emoji positive).

Usiamo l'insieme dei tweet come distantly supervised labels per eseguire la labellizzazione dei dati presenti nel nostro dataset iniziale. Quando utilizziamo una Neural Network dove sui pesi della rete e sui word embedding viene fatto il training per catturare gli aspetti relativi ai sentimenti espressi nella frase. Non è una labellizzazione perfetta per i dati che vogliamo usare in input e trasformare in embedding ma è già qualcosa per iniziare.



Una volta che abbiamo prodotto questo embedding "pre-trained" possiamo andare avanti ed utilizzare il sistema della CNN vista in precedenza.

Un altro metodo per il training degli embedding

Un altro metodo che possiamo utilizzare per eseguire il training degli embedding consiste in un processo simile a quello che abbiamo per creare i normali word embedding. In questo caso però consideriamo due aspetti:

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

- Dobbiamo decidere se una parola è nel contesto corretto.
- Dobbiamo capire se la parola è in un tweet positivo o se è in un tweet negativo, possiamo anche usare emoji positive o negative per capirlo.

Ensemble of Classifier

Quando lavoriamo con neural network applicate al NLP è anche possibile utilizzare più di una neural network contemporaneamente andando a combinare gli output che otteniamo.

Le due CNN potrebbero essere inizializzate con word embedding differenti e potrebbero anche utilizzare dei dati di training leggermente differenti. Altre differenze potrebbero stare nella scelta dei parametri come ad esempio il numero di filtri da usare nella convoluzione.

12.3 Conclusione

Abbiamo parlato di sentiment Analysis considerando alcuni possibili approcci che sono stati utilizzati nel corso degli anni.

Con le CNN abbiamo il problema che servono molti dati di training e quindi è stata introdotta l'idea di utilizzare degli embedding pre-trained. Quindi è stata utilizzato un training in due fasi (abbiamo dei dati iniziali e poi eseguiamo un pre-train usiando altri dati che sono labellati basandoci ad esempio sulla positività/negatività della frase in base alla presenza di emoji), questa combinazione ci permette di raggiungere una quantità di dati che ci permettono di lavorare in modo accettabile.

Questi esperimenti mostrano ancora una volta che la capacità di avere tanti dati permette di creare lexicon specifici e di avere quindi sistemi che funzionano bene.

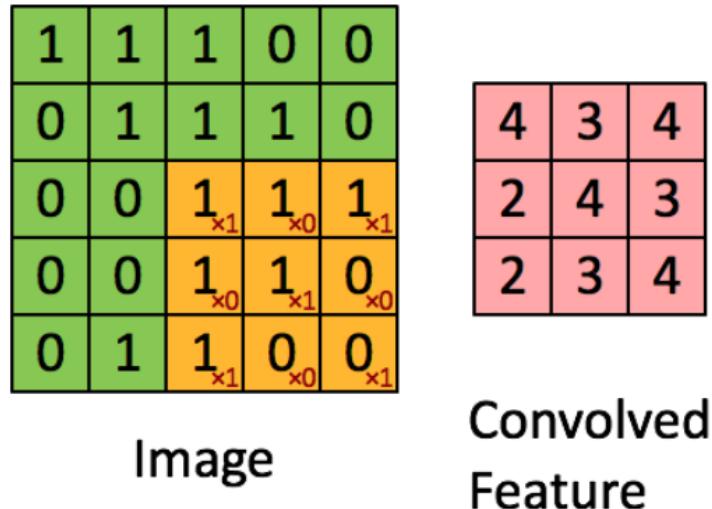
Chapter 13

Lesson 12 - CNN per NLP

13.1 Introduzione

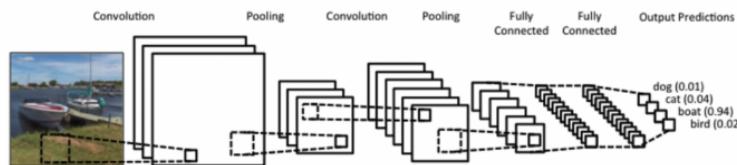
Le CNN sono tipicamente utilizzate in Computer Vision ma negli ultimi anni sono state applicate anche a problemi di Natural Language Processing.

Quello che vogliamo fare con le immagini è estrarre una feature che può essere posizionata in varie possibili posizioni dell'immagine. Noi non sappiamo dove si trova la feature, vogliamo semplicemente estrarla in modo automatico. Quello che facciamo è il processo di convoluzione che tenta di catturare queste features nell'immagine andandole poi ad esemplificare in una rappresentazione più "profonda" dell'immagine stessa.



Come possiamo vedere dalla immagine sopra, noi abbiamo un input (in questo caso una immagine) e poi abbiamo un filtro che passa su tutta l'immagine andando a produrre una matrice più piccola che mi rappresenta le feature dell'immagine. In questo caso abbiamo una matrice iniziale di dimensione 5*5 e usando un filtro di dimensione 3*3 avremo una matrice finale (convolved feature) di dimensione 3*3.

Più formalmente possiamo dire che nelle CNN utilizziamo la convoluzione nell'input layer per produrre l'output e il risultato è che ogni regione dell'input è connessa con un neurone dell'output.



Nella foto qua sopra possiamo vedere una struttura tipica di una CNN.

La cosa bella è che durante la fase di training la CNN impara automaticamente i valori del filtro che verrà utilizzato basandoci sul task che

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

vogliamo effettuare.

Quello che abbiamo in una CNN è una serie di convolution layer, in ognuno di questi andiamo a riconoscere alcune delle feature dell'immagine fino a quando si arriva al layer finale di output in cui classifichiamo l'immagine basandoci su tutte le feature che abbiamo estratto.

Ci sono due aspetti da considerare, uno è la Location Invariance e uno è la Compositionality:

- Ammettiamo di voler capire se in una immagine c'è un elefante o no, non ci interessa dove l'elefante si trova e se è capovolto o ruotato. Con il filtro che passa su tutta l'immagine possiamo riuscire a capire se l'elefante è presente indipendentemente dalla posizione, questa è la proprietà di Local Invariance.
- La seconda feature è la compositionality, abbiamo che il filtro che passa sull'immagine è in grado di comporre le feature da un basso livello ad una rappresentazione di più alto livello.

13.2 CNN per NLP

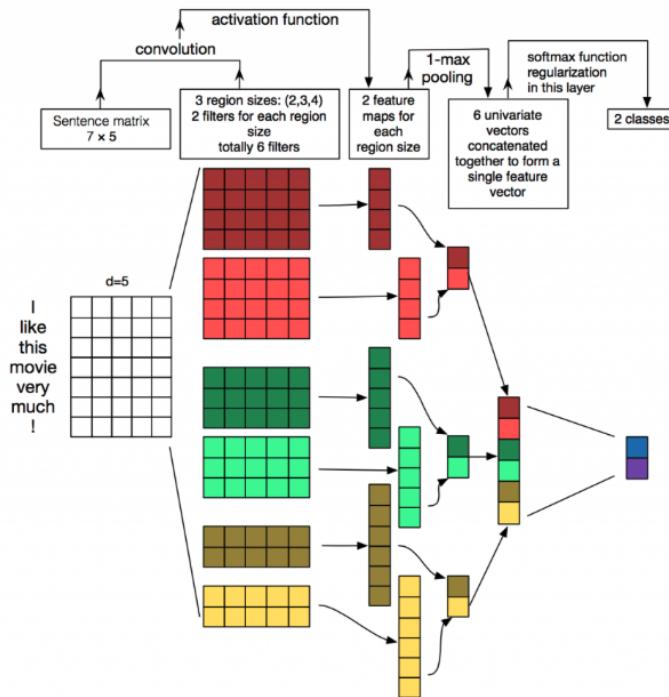
Da qualche anno le CNN sono utilizzate anche per il NLP, l'idea basilare è che abbiamo il nostro set di dati che possiamo rappresentare come una matrice in cui per ogni riga abbiamo una parola o magari anche un carattere. In ognuna delle righe abbiamo un word embedding (word2Vec o Glove per esempio) ma potrebbe anche essere un one-hot vector (per esempio per 10 parole con un embedding di dimensione 100 avremmo una matrice di dimensione 10*100).

Data questa matrice in input possiamo considerare dei trigram e poi eseguire la convoluzione su questi trigram. La cosa che va sottolineata è che questi trigram potrebbero anche essere non grammaticalmente corretti o linguisticamente motivati ma non ci interessa. In pratica quello che si fa è creare dei filtri che come larghezza hanno la larghezza della

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

matrice dei nostri dati e come altezza una dimensione compresa tra 2 e 5 in generale.

In generale possiamo dire che una CNN per il NLP potrebbe avere una struttura di questo genere:



Nella foto sopra abbiamo una matrice in input di dimensione $7 * 5$ e utilizziamo 3 differenti dimensioni per i filtri, per ogni dimensione abbiamo 2 filtri. Quindi in tutto applichiamo la convoluzione 6 volte, una per ogni filtro e otteniamo per ognuno di questi filtri un feature vector. Qua poi abbiamo anche un layer di max pooling ovvero prendiamo il massimo da ciascuno di questi filtri e poi alla fine abbiamo il layer con softmax che prende in input le varie feature che abbiamo prodotto e produce l'output.

Abbiamo parlato di Location Invariance e Local Compositionality per le immagini, qua queste proprietà chiaramente sono importanti perché

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

nel testo la posizione delle parole conta e non possiamo disinteressarcene. Per questa ragione le CNN non sono una soluzione a tutti i problemi di NLP, ci sono le RNN che funzionano meglio ad esempio, in alcuni casi però per alcuni utilizzi risultano comunque utili.

La cosa positiva è che sono veloci e che sono in grado di fare automaticamente il learning dei filtri che vengono utilizzati, il motivo è che le CNN sono pensate per le immagini e quindi abbiamo implementazioni che sfruttano anche le GPU, utilizzando n-gram di dimensione 3 le prestazioni sono ottime, poi iniziano a degradare a mano a mano che si alza il numero degli n-gram, anche Google non va oltre a 5.

Un esempio di convoluzione con NLP

Possiamo considerare una matrice di embedding in cui abbiamo quindi una serie di parole con i corrispondenti embedding.

Not	0.2	0.1	-0.3	0.4
going	0.5	0.2	-0.3	-0.1
to	-0.1	-0.3	-0.2	0.4
the	0.3	-0.3	0.1	0.1
beach	0.2	-0.3	0.4	0.2
tomorrow	0.1	0.2	-0.1	-0.1
:-(-0.4	-0.4	0.2	0.3

La convoluzione prende le parole che sono consecutive e le moltiplica per il filtro, in questo caso moltiplichiamo per un filtro che è una matrice $3 * 4$.

3	1	2	-3
-1	2	1	-3
1	1	-1	1

Il filtro viene mosso sulla matrice e ad ogni iterazione dobbiamo eseguire la moltiplicazione. Per produrre una convoluzione finale che non sia lineare dobbiamo introdurre anche del bias:

w1,w2,w3	-1.0
w2,w3,w4	-0.5
w3,w4,w5	-3.6
w4,w5,w6	-0.2
w5,w6,w7	0.3

Il risultato finale che otteniamo è il seguente:

0.0	0.50
0.5	0.38
-2.6	0.93
0.8	0.31
1.3	0.21

Come possiamo vedere qua abbiamo una matrice iniziale che ha 7 righe ma poi alla fine la feature matrix che produciamo ha solamente 5 righe. Il motivo è legato al fatto che quando scorriamo il filtro abbiamo delle righe che vengono considerate 3 volte. Una possibilità è aggiungere del padding in modo da ottenere una feature matrix che abbia lo stesso numero di righe della matrice originale.

0	0.0	0.0	0.0	0.0
Not	0.2	0.1	-0.3	0.4
going	0.5	0.2	-0.3	-0.1
to	-0.1	-0.3	-0.2	0.4
the	0.3	-0.3	0.1	0.1
beach	0.2	-0.3	0.4	0.2
tomorrow	0.1	0.2	-0.1	-0.1
:-)	-0.4	-0.4	0.2	0.3
0	0.0	0.0	0.0	0.0

Fino ad ora abbiamo sempre applicato un singolo filtro, quello che possiamo fare però è applicarne più di uno alla volta.

3	1	2	-3	1	0	0	1	1	-1	2	-1
-1	2	1	-3	1	0	-1	-1	1	0	-1	3
1	1	-1	1	0	1	0	1	0	2	2	1

Ad esempio potremmo applicare 3 filtri con all'interno dei pesi differenti e in questo modo invece di produrre un singolo feature vectore ne produciamo 3.

0,w1,w2	-0.6	0.2	1.4
w1,w2,w3	-1.0	1.6	-1.0
w2,w3,w4	-0.5	-0.1	0.8
w3,w4,w5	-3.6	0.3	0.3
w4,w5,w6	-0.2	0.1	1.2
w5,w6,w7	0.3	0.6	0.9
w6,w7,0	-0.5	-0.9	0.1

Notare che la convoluzione che effettuiamo consiste in:

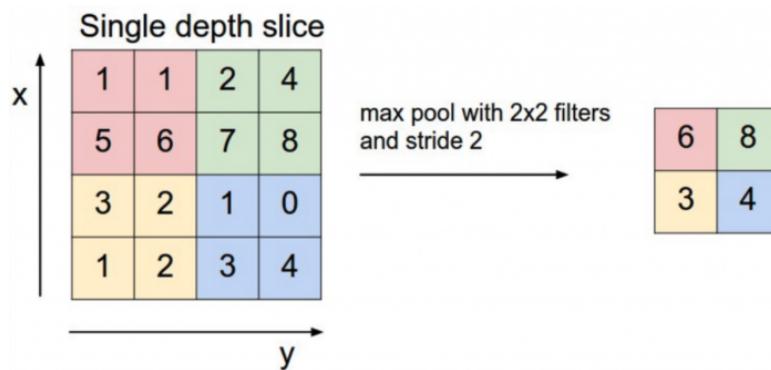
- Passare il filtro sopra alla matrice, calcolare la moltiplicazione di ogni elemento della matrice per l'elemento del filtro.
- Sommare tutti i risultati di questa moltiplicazione ottenendo un solo valore finale.

Un'altra cosa simile al padding che possiamo modificare è lo stride, quando muoviamo il filtro sopra alla matrice originale possiamo decidere di nuoverci di 2 righe alla volta e non di una sola riga alla volta.

La dimensione dello stride, così come la dimensione del padding è uno degli iperparametri delle CNN.

13.2.1 Pooling Layer

Il pooling layer ci permette di applicare una operazione di max al risultato che otteniamo con ognuno dei filtri. Non è necessario eseguire il pooling sulla matrice completa, posso lavorare solamente su una finestra 2×2 all'interno della matrice.



Come possiamo vedere dalla foto abbiamo questo pooling che viene effettuato su ognuna delle parti di questa matrice e abbiamo che da ogni parte della matrice noi consideriamo il valore massimo.

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

Perchè utilizziamo il pooling:

- Supponiamo di avere 1000 filtri, a seconda della dimensione del filtro e anche in base alla dimensione dei dati in input abbiamo una dimensione dell'output che è variabile. Applicando il MaxPooling noi andiamo a prendere il massimo di ogni parte della matrice e questo fa sì che si ottenga sempre una matrice di output di dimensione costante indipendentemente dalle dimensioni dei filtri e dell'input.
- Il pooling riduce anche la dimensione dell'output e dovrebbe mantenere le informazioni salienti che sono contenute all'interno del feature vector. In particolare la convoluzione dovrebbe evidenziare alcune feature, ad esempio se evidenzia la presenza di una parola magari assegnerà un valore maggiore in un certo elemento della matrice perchè ha trovato quella parola e altrove assegnerà un valore minore. Quindi se noi facciamo il pooling riusciamo ad ottenere una matrice che comunque mantiene le informazioni essenziali e più importanti.
- Quando si parla di immagini abbiamo anche che il Pooling ci permette di essere invarianti rispetto alla rotazione e alla traslazione delle immagini.

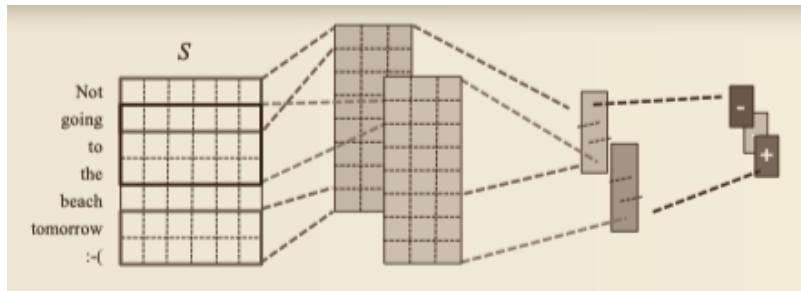
13.2.2 Channels

Un concetto importante quando si parla di CNN è quello dei channel, quando dobbiamo lavorare su una immagine a colori abbiamo 3 canali che corrispondono a RGB, se invece abbiamo una immagine in bianco e nero avremo solamente 1 canale. Avere più di un canale vuol dire avere più di una matrice in input. Anche in NLP abbiamo una situazione simile con i canali, in particolare potremmo avere più di un canale perchè magari

stiamo utilizzando più di un singolo word embedding o perchè magari stiamo utilizzando varie lingue per il nostro testo.

13.2.3 Possibili utilizzi delle CNN in NLP

Possiamo utilizzare le CNN in NLP ad esempio per la Sentiment Classification ma anche per la rilevazione dell'argomento principale di una certa frase o per la rilevazione dello spam. In particolare sono state effettuate delle ricerche per usare una CNN con un singolo layer per classificare le frasi semplicemente come positive o negative. La struttura di una rete di questo genere potrebbe essere la seguente:



- Avremmo un layer con gli embedding.
- Un layer per la convoluzione che usa ReLU come activation function, abbiamo vari filtri da utilizzare.
- Abbiamo un layer per il max pooling.
- Un layer per il dropout.
- Un layer con tanh come activation function.
- Il layer con la softmax che produce l'output.

Possiamo subito notare che operazioni come la convolution o come il pooling potrebbero portare ad una perdita delle informazioni presenti all'interno delle frasi soprattutto l'ordine delle parole.

Fino ad ora abbiamo sempre parlato di CNN applicate alle parole, c'è però da dire che in alcune ricerche sono state applicate le CNN anche direttamente alle parole e non alle frasi.

13.3 Implementazione di una CNN per NLP

Per implementare una CNN da utilizzare per la classificazione del testo usiamo Keras.

La prima cosa che viene fatta è settare alcuni parametri che poi utilizzeremo nella definizione della CNN, in particolare limitiamo la dimensione dei vocaboli presenti all'interno del vocabolario e la lunghezza delle frasi che prendiamo in considerazione. Fissiamo inoltre a 100 il numero di filtri che vogliamo utilizzare:

```
max_words = 5000 # vocabulary size
 maxlen = 80      # max sentence length
batch_size = 32
embedding_dims = 50
filters = 100     # the number of output filters in the convolution
kernel_size = 3   # length of the 1D convolution window
hidden_dims = 250
epochs = 3
```

Una volta scaricati i dati che dobbiamo utilizzare e fatto un minimo di preprocessing (divisione in train e test set e riduzione della dimensione di ogni frase a frasi di massimo 80 parole) possiamo iniziare a creare la struttura della nostra CNN.

```
model.add(Embedding(max_words,
 embedding_dims,
 input_length=maxlen))

model.add(Dropout(0.2))
```

Il primo layer che aggiungiamo è il layer con gli embedding, per creare il layer con gli embedding dobbiamo passare come parametri il numero

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

di parole che stiamo considerando, la dimensione degli embedding e la lunghezza massima delle frasi.

Oltre al primo layer con gli embedding aggiungiamo poi un secondo Layer che è per il Dropout, questo ci serve per ridurre l'overfitting.

Una volta finita questa prima parte di architettura possiamo creare il layer di convoluzione.

```
model.add(Conv1D(filters,           # output size
                 kernel_size,      # width of convolution window
                 padding='valid',   # "valid" means "no padding"
                 activation='relu', # Rectified Linear Unit
                 strides=1))       # consecutive, no gaps
```

Con questo codice qua noi a aggiungiamo il layer di convoluzione e come parametri passiamo il numero di filtri che vorremmo utilizzare (che equivale alla dimensione dell'output di questo layer), la dimensione del filtro (in questo caso è 3 quindi consideriamo 3 parole alla volta) e poi fissiamo anche l'activation function che in questo caso è ReLU.

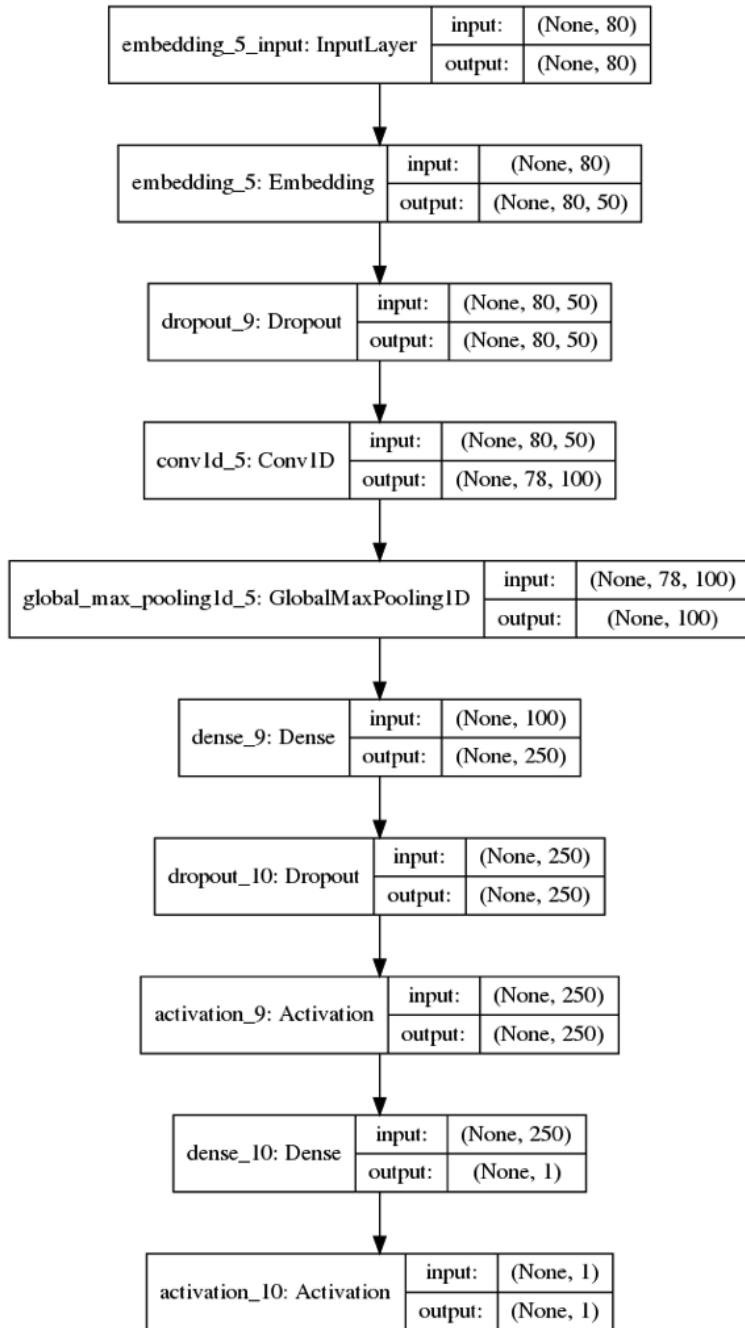
Poi possiamo aggiungere altri layer:

```
model.add(GlobalMaxPooling1D())
model.add(Dense(hidden_dims))
model.add(Dropout(0.2))
model.add(Activation('relu'))
model.add(Dense(1))
model.add(Activation('sigmoid'))
```

qua aggiungiamo un layer con il MaxPooling, poi il dense layer in cui passiamo come parametro il numero di hidden dimension che vogliamo utilizzare (in questo caso 250) e di nuovo aggiungiamo un layer con l'activation function ReLU. Poi un altro layer di Dense, in questo caso l'output è 1 e poi aggiungiamo alla fine la sigmoid function per fare la previsione.

Il modello viene compilato usando la binary cross entropy come funzione per la loss.

La struttura del modello è la seguente:



Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

Notare che gli embedding non cambiano durante l'esecuzione del modello e che non viene eseguito il training degli embedding.

In particolare sempre rimanendo in tema embedding se utilizziamo degli embedding pre-trained ci troviamo nella situazione in cui analizzare un testo che parla di notizie è differente rispetto all'analisi di un testo che parla di sport e quindi delle parole che in un campo potrebbero essere simili sono considerate diverse tra loro nell'altro campo.

In generale quello che possiamo dire è che conviene utilizzare degli embedding pre-trained perchè sono stati creati considerando molte più parole di quante tu ne possa avere nel tuo dataset. Quando abbiamo un training set piccolo non abbiamo bisogno di fare il training dei word vectors e quindi usiamo direttamente gli embedding, se invece abbiamo un training set più grande possiamo anche eseguire un tuning di questi word-vector andandoli ad aggiornare.

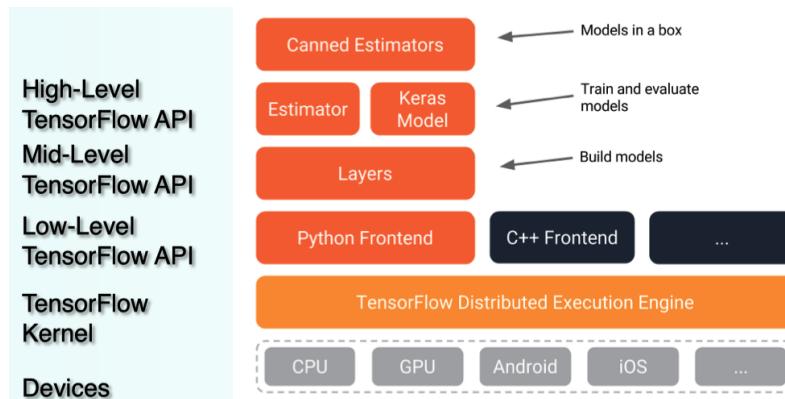
Il layer del Dropout lo inseriamo perchè è una delle tecniche che vengono utilizzate per eseguire la regolarizzazione. L'idea del dropout è quella di non considerare in ognuna delle iterazioni del training del modello una certa feature in modo che si eviti che alcune feature siano co-esistenti in ogni esecuzione del training.

Chapter 14

Lesson 13 - Introduction to TensorFlow

14.1 Introduction

TensorFlow è uno dei Framework open source più popolari che esistono per il Machine Learning. La parte principale di TensorFlow è scritta in C++ e questo permette di utilizzare il più possibile le capacità hardware della macchina su cui stiamo facendo girare il framework.



La cosa bella è che possiamo creare il nostro modello in modo del

tutto trasparente rispetto all'hardware che abbiamo sotto. All'interno dell'architettura di TensorFlow abbiamo anche un Frontend scritto in Python che mi permette di fare un binding con le librerie del C++.

Abbiamo due modelli di esecuzione con TensorFlow:

- Graph Execution: questo è il primo modello d'esecuzione che è stato pensato con la versione 1 di TensorFlow. L'idea qua è che costruiamo un computational graph che può essere poi eseguito. I benefici di utilizzare questo tipo di modello di esecuzione sono legati in modo particolare alle performances, possiamo infatti distribuire facilmente il lavoro in varie CPU e varie GPU, questo perché possiamo facilmente suddividere le varie parti del grafo. Un altro beneficio di questo modello riguarda poi il fatto che in alcuni casi possiamo eseguire solamente i sottografi che realmente ci interessano, dividendo poi la computazione in parti più piccole riusciamo anche a facilitare l'auto differenziazione. Un'ultima motivazione che ci fa apprezzare questo modello di esecuzione è che risulta una rappresentazione naturale dei modelli di machine learning che spesso vengono rappresentati su grafi. Questo tipo di esecuzione è anche più adatto se vogliamo utilizzarlo in produzione. Si tratta però di un modello di esecuzione non troppo intuitivo, ad esempio Pytorch è più intuitivo. In questa soluzione abbiamo due fasi: nella prima assembliamo il grafo e nella seconda eseguiamo le operazioni all'interno del grafo.
- Eager Execution: qua per definire il nostro modello possiamo scrivere del codice Python che valuta le varie operazioni immediatamente senza dover creare un grafo. All'interno di questo modello vengono utilizzati i tensori, ogni operazione che lavora con i tensori produce poi in output dei tensori. I benefici di utilizzare questo tipo di modello di esecuzione riguardano soprattutto la facilità d'uso, abbiamo una interfaccia più intuitiva, poi è più facile eseguire il debug.

14.2 TensorFlow Computation Graph

All'interno del computation graph di TensorFlow abbiamo i seguenti componenti:

- PlaceHolders: nel grafo (non in tensorflow) sono variabili che utilizziamo al posto degli input per inserire dati all'interno del grafo.
- Variabili: sono le variabili del modello che dobbiamo ottimizzare per avere migliori performances.
- Model: l'output complessivo del grafo, una funzione matematica che calcola l'output basandosi su placeholders e variabili del modello.
- Loss measure: è la funzione che utilizziamo per l'ottimizzazione del modello.
- Optimization method: il metodo che viene utilizzato per il tuning delle variabili del modello.
- Session: si tratta di un runtime dove vengono eseguite le operazioni dei vari nodi e dove vengono valutati i tensori, questa la troviamo solamente nella esecuzione su grafo e non esiste nel modello eager.

All'interno di TensorFlow i dati vengono rappresentati tramite dei tensori, un tensore di dimensione 0 è uno scalare, di dimensione 1 è un vettore e di dimensione 2 è una matrice (ad esempio rappresenta l'embedding).

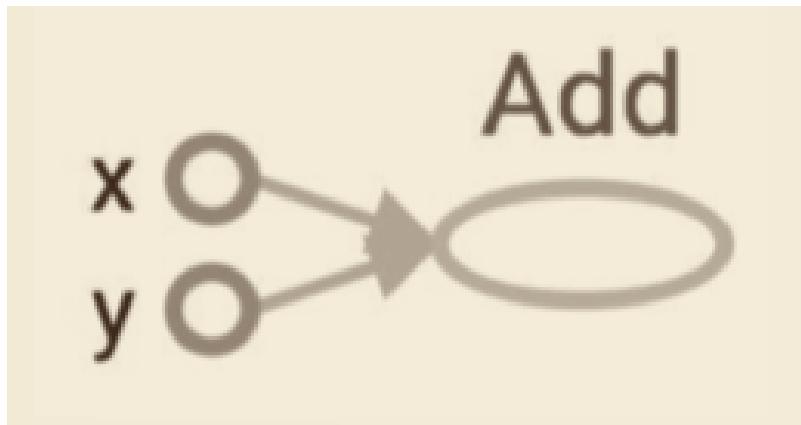
Supponiamo di avere un codice come questo qua sotto:

```
import tensorflow as tf
a = tf.add(3, 5)
```

Quello che succede è che Tensorflow mi fa una conversione di questo codice in un grafo in cui eseguiamo la somma dei due valori in input.

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

Dietro le quinte vengono create due variabili x e y e il nodo in se per sé rappresenta l'operazione di addizione di x e di y .



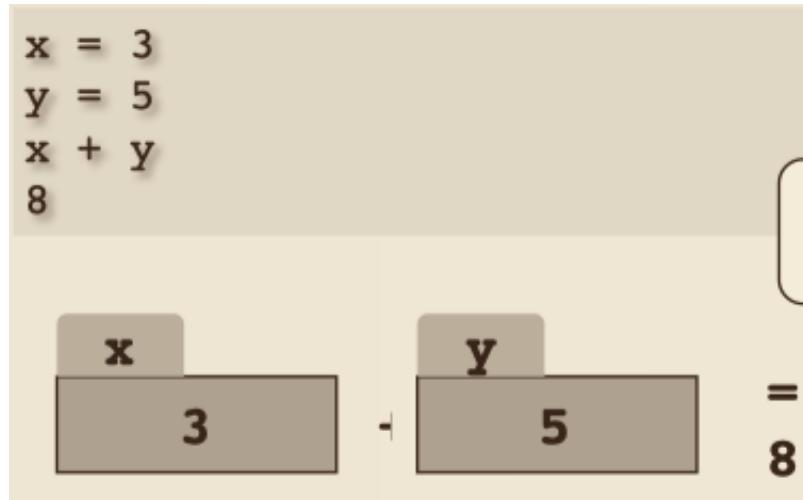
Qua non dobbiamo aspettarci di ottenere un numero alla fine dell'operazione, quello che otterremo sarà un tensore.

14.3 Automatic Differentiation

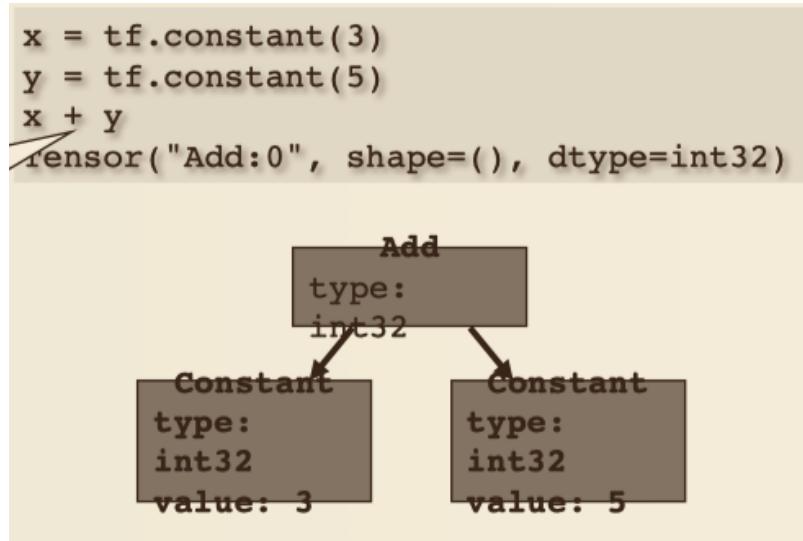
Abbiamo vari modi per eseguire la differenziazione, possiamo definire manualmente l'operazione ad esempio, oppure possiamo fare una approssimazione. Quello che fa TensorFlow è supportare l'automatic differentiation e in particolare implementa una reverse mode automatic differentiation, non si tratta di una approssimazione ma si tratta di una differenziazione corretta. Questa automatic differentiation viene utilizzata per la backpropagation, quindi nel backpropagation graph che viene prodotto automaticamente da TensorFlow, dato che il grafo è formato da sottografi che vengono distribuiti nei vari dispositivi su cui facciamo girare il nostro modello, anche il calcolo del gradiente verrà distribuito per essere eseguito su vari dispositivi.

Consideriamo il caso della classica programmazione, in questo caso in una variabile memorizziamo dei valori e le operazioni calcolano dei

valori. Possiamo vedere dalla immagine qua sotto quello che succede:



Se invece utilizziamo il grafo per rappresentare le varie operazioni abbiamo i valori che devono essere sommati che vengono memorizzati in una variabile che rappresenta solamente se stessa e non il valore in se per se.



Quando poi dobbiamo rappresentare l'operazione matematica utilizziamo l'overloading (a seconda del tipo dei parametri che passiamo abbiamo qualcosa di differente nell'esecuzione della funzione) e passiamo all'operatore di addizione le due variabili create in precedenza. Viene quindi generato un grafo in cui, nei nodi abbiamo i valori con il loro tipo (oggetti che vengono sommati) e poi viene effettuata la somma. Il tipo delle due variabili che sommiamo è un tensore che (a parte le costanti) non hanno un valore vero e proprio ma possiamo comunque valutarle per produrre un valore, la valutazione richiede una Session che contenga la memoria per i valori associati alle variabili. I tipi di tensori che possiamo avere sono:

- tf.variable
- tf.constant
- tf.sparsetensor

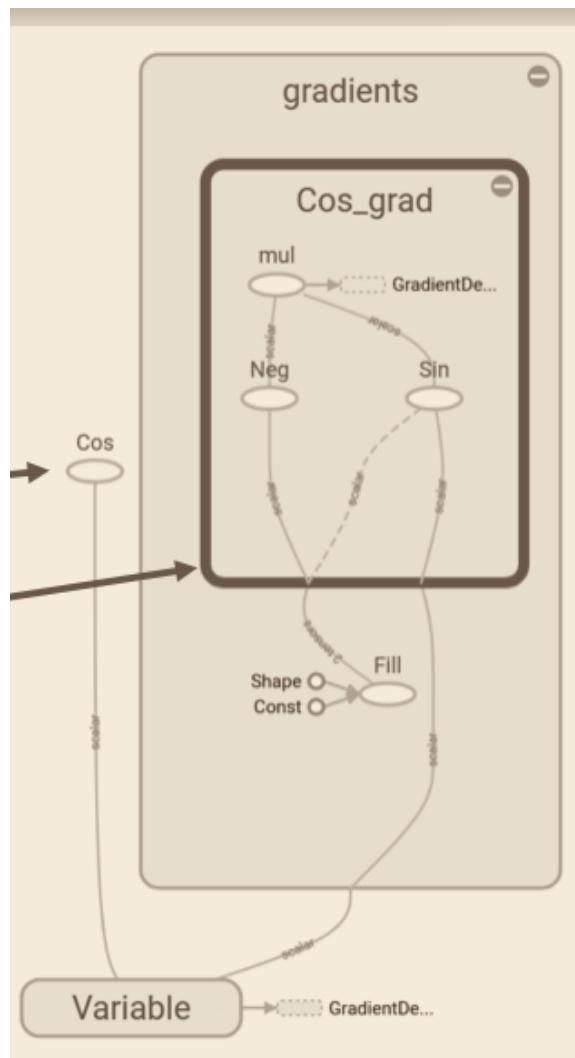
Un esempio:

```
a = tf.constant(10)
b = tf.constant(7)
a+b
<tf.Tensor, id = 2, shape=(), ... >
(a+b).numpy() # if you want to evaluate the sum you can use this numpy() method
17
```

Vediamo ora di capire in che modo funziona il meccanismo di Automatic Differentiation, consideriamo il seguente codice:

```
x = tf.Variable(initial_value = 3.0)
y = tf.cos(x)
optimizer = tf.train.GradientDescentOptimizer()
```

che produce il seguente differentiation graph:

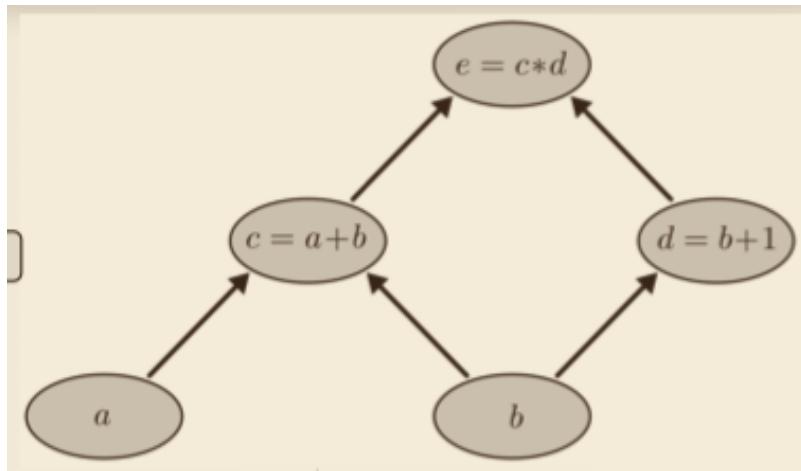


Nel codice abbiamo l'esecuzione del calcolo dell'operazione di $\cos(s)$ e deve essere calcolata la derivata di questa operazione che è $-\sin(x)$, nella derivata abbiamo una moltiplicazione tra -1 e $\sin(x)$. Tutto quello che vediamo nella figura sopra è il grafo che viene prodotto per il calcolo del gradiente, possiamo notare che per il calcolo del $\cos()$ e per il calcolo del $\sin()$ abbiamo in comune la stessa variabile che infatti è connessa ad entrambi i nodi.

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

L'auto differentiation viene calcolata utilizzando la Reverse Accumulation Auto Differentiation. Quello che si fa è partire con una espressione iniziale della derivata e poi andiamo a suddividerla in derivate più piccole applicando la chain rule dall'esterno all'interno, in pratica attraversiamo il grafo in ordine inverso rispetto a quando abbiamo effettuato l'operazione di forward in cui abbiamo calcolato i valori dei vari nodi.

Supponiamo di essere in una situazione di questo genere:



Vogliamo eseguire l'operazione di backward dal nodo e fino ad arrivare al nodo a , quindi partiamo con il calcolo della derivata $\frac{\nabla e}{\nabla a}$, questo possiamo dividerlo in una moltiplicazione utilizzando la chain rule e quindi diventa: $\frac{\nabla e}{\nabla a} = \frac{\nabla e}{\nabla c} \frac{\nabla c}{\nabla a}$.

Ognuno dei nodi di questo grafo ha due metodi:

- Metodo Forward
- Metodo Backward, questo viene ottenuto con la chain rule.

Supponiamo di avere un grafo in cui abbiamo solamente due variabili x e y che vengono moltiplicate per formare z . Il nodo che produce la z avrà il seguente codice:

```

def forward:
    z = x*y
    self.x = x
    self.y = y
    return z

def backward:
    # Qua noi dobbiamo calcolare la derivata rispetto
    # a x e a y
    # Memorizziamo la derivata all'interno del nodo stesso
    dx = self.y
    dy = self.x
    return [dx, dy]

```

Ovviamente Tensorflow ci permette di eseguire tutta questa operazione di automatic differentiation in modo del tutto automatica e trasparente, non dobbiamo scriverci noi direttamente l'operazione del calcolo del seno e del calcolo della derivata.

Come ottenere il gradiente

Per memorizzare quello che succede al gradiente all'interno del grafo dobbiamo utilizzare GradientTape, allo stesso modo lo utilizziamo per produrre poi il gradiente.

```

def grad(x):
    with tf.GradientTape() as Tape:
        return Tape.gradient(square(x), x)

square(3).numpy() # x^2 = 9
grad(3).numpy() # 2*x

```

Il gradiente in questo caso è $2 * x$, noi non lo vediamo ma possiamo applicare la funzione numpy() per ottenere poi l'output.

Oltre al calcolo del gradiente fornito da TensorFlow di cui abbiamo parlato fino ad ora abbiamo anche dei casi in cui possiamo fare meglio di quello che ci viene offerto dall'Automatic Differentiation. Ad esempio le librerie sanno come eseguire le derivate del logaritmo ma l'output non sarà stabile e quindi produrrà dei numeri che non sono troppo semplici da utilizzare per alcuni valori. Utilizzando l'annotazione

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

`@tf.e.custom_gradient` abbiamo la possibilità di rendere la computazione del gradiente più stabile andandola a definire in modo che produca un valore adatto alle nostre esigenze e non "NaN" come potrebbe avvenire altrimenti.

Session

L'oggetto Session di TensorFlow incapsula l'ambiente in cui le operazioni vengono eseguite e in cui gli oggetti tensori vengono valutati. Le sessioni allocano anche memoria per memorizzare i valori correnti delle variabili. Se eseguiamo una operazione e vogliamo ottenere il valore corrispondente possiamo assegnare l'operazione alla sessione e poi da li ottenere il valore.

Subgraph

Dato che TensorFlow ci permette di creare un grafo partendo dal codice che scriviamo, possiamo anche creare dei sottografi andando poi a capire quali sono le parti che realmente sono utili e quali sono quelle che invece non sono troppo utili. Grazie a questa suddivisione in sottografi è possibile eseguire in parallelo ognuno di questi sottografi andandoli ad assegnare alle varie CPU o GPU in modo da sfruttare al massimo il parallelismo. Se non vogliamo fare automaticamente questa suddivisione in varie GPU possiamo fare tutto anche automaticamente:

```
# Create a graph.
with tf.device('/gpu:2'):
    a = tf.constant([1.0, 2.0, 3.0, 4.0, 5.0, 6.0], name='a')
    b = tf.constant([1.0, 2.0, 3.0, 4.0, 5.0, 6.0], name='b')
    c = tf.multiply(a, b)
    # Create a session with log_device_placement set to True.
    sess = tf.Session(config=tf.ConfigProto(log_device_placement=True))
    # Run the op.
    print(sess.run(c))
```

L'esecuzione di programmi come questo, che vengono poi tradotti in grafi, può anche essere visualizzata utilizzando TensorBoard.

Managing the training set

TensorFlow ci offre anche la possibilità di gestire il training set e distribuirlo utilizzando *tf.data*. Tramite *tf.data* possiamo rappresentare il nostro dataset e inserire i dati all'interno. Vediamo un esempio:

```
tf.data.Dataset.from_tensor_slices((features, labels))
dataset = tf.data.Dataset.from_tensor_slices((data[:,0], data[:,1]))
print(dataset.output_types) # >> (tf.float32, tf.float32)
print(dataset.output_shapes) # >> (TensorShape([]), TensorShape([]))
```

Nel codice sopra quello che facciamo è prendere i dati che abbiamo a disposizione e poi fare una prima divisione tra feature e labels andando poi ad assegnare i dati all'interno del dataset. La stessa funzione *tf.data* ci permette anche di caricare i dati a partire da file. Una volta che abbiamo creato il dataset possiamo anche iterarlo e questo lo possiamo fare utilizzando la funzione *tf.data.Iterator*.

Chapter 15

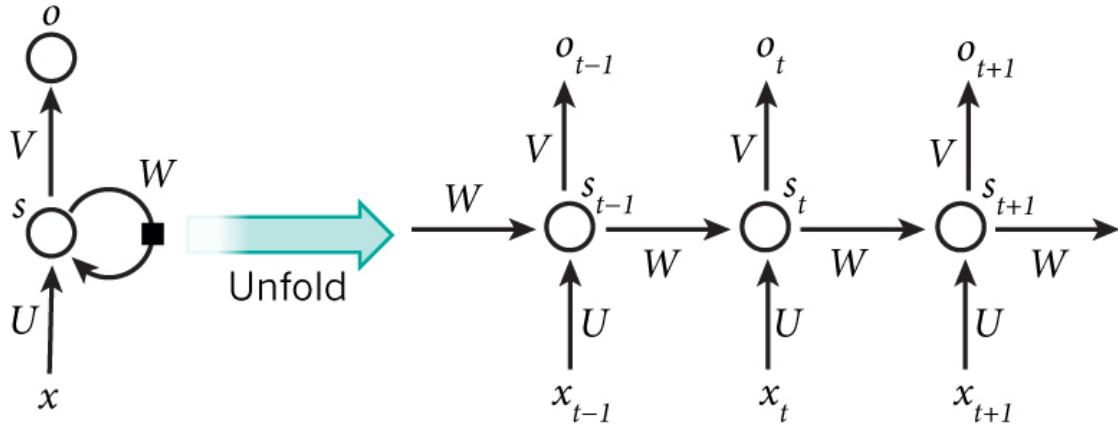
Lesson 14 - Recurrent Neural Networks

15.1 Introduzione alle RNN

Per capire il motivo per cui sono state inventate le RNN possiamo pensare ad una situazione più umana, quando leggiamo qualcosa noi capiamo ogni parola basandoci sul modo in cui abbiamo compreso le precedenti parole. Se prendiamo una normale neural network noi assumiamo che tutti gli input e tutti gli output sono indipendenti tra loro, se vogliamo prevedere la prossima parola presente all'interno di una frase, questo ci riuscirà meglio se conosciamo le parole precedenti.

Per questa ragione sono state pensate le Recurrent Neural Network (RNN). Queste NN sono chiamate Recurrent perchè eseguono la stessa operazione su tutti gli elementi di una sequenza con gli output di ognuna di queste operazioni che dipendono dalle precedenti operazioni.

Un altro modo di pensare le RNN è che hanno una memoria che cattura le informazioni riguardo a quello che è stato calcolato fino a quel momento.



Nella immagine sopra possiamo vedere come appare una RNN, a sinistra abbiamo la rappresentazione compatta in cui abbiamo un input x che viene moltiplicato per un peso U e poi tramite una funzione viene unito al peso W per produrre un output o . Nell'immagine a destra possiamo vedere la stessa RNN ma esplosa. In particolare supponiamo di avere una frase di 5 parole, avremo una RNN con 5 layer, un layer per ogni parola. In particolare i componenti della RNN saranno i seguenti:

- x_t è l'input allo step t quindi vuol dire che corrisponde ad una delle 5 parole della frase.
- Abbiamo l'hidden State S_t al tempo t , questo viene calcolato in base all'input corrente e in base all'input che arriva dallo step precedente. Quindi possiamo scrivere il valore dell'hidden state come:

$$s_t = f(Ux_t + Ws_{t-1})$$

La funzione f solitamente è una funzione non lineare come ad esempio ReLU o tanh. In teoria in s_t abbiamo informazioni su tutto quello che è successo nei precedenti time stamp, in pratica è più complesso perché non riusciamo a tenere in considerazione proprio tutto.

- o_t è l'output dello step t , se ad esempio vogliamo prevedere se la parola che stiamo considerando è adatta all'interno della frase dovremmo generare una probabilità quindi utilizzeremmo la softmax. In particolare questo output lo possiamo generare come:

$$o_t = \text{softmax}(Vs_t)$$

È importante notare che l'output allo step t viene calcolato solamente in base allo stato al tempo t ovvero in base a s_t .

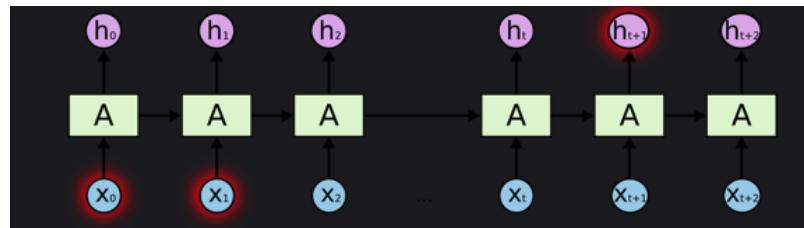
Una cosa che vale la pena sottolineare è che a differenza delle classiche Neural Network dove abbiamo tanti parametri, qua abbiamo solamente tre parametri che vengono condivisi tra tutti i layer. Come possiamo vedere dalla foto infatti abbiamo solamente tre parametri che sono U, V, W e in tutti i layer vengono utilizzati per produrre l'output.

Notare anche che nell'immagine sopra produciamo un output in ognuno dei vari layer ma in realtà non sempre è necessario, in alcuni casi può bastare anche solamente un output.

Le Recurrent Neural Network possono avere grande successo in task di NLP, hanno dei vantaggi e degli svantaggi:

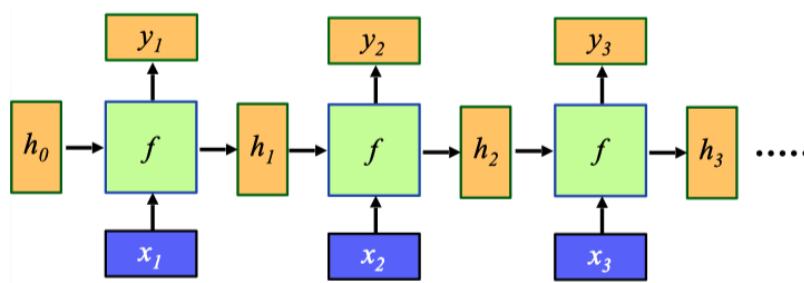
- Vantaggi:
 - Possono processare un input di qualsiasi lunghezza
 - La dimensione del modello non aumenta se abbiamo input più grandi
 - La computazione in un certo step t riesce (in teoria) ad essere influenzata dalla computazione che svolgiamo negli step precedenti.
 - I pesi sono condivisi tra tutti i layer.
- Svantaggi

- Le sequenze (e i layer) vanno ad aumentare e quindi abbiamo problemi con le performance di queste RNN perchè la back-propagation dovrebbe andare dall'ultimo layer fino al primo per ogni elemento della sequenza e quindi questo non è molto semplice.
- Un secondo problema riguarda il fatto che a mano a mano che le RNN crescono diventano sempre meno propense a considerare le informazioni dei layer più vecchi. In teoria questo problema non dovrebbe esistere ma in pratica c'è.



Questo problema è stato risolto con le LSTM.

Consideriamo la seguente RNN e facciamo alcune considerazioni:



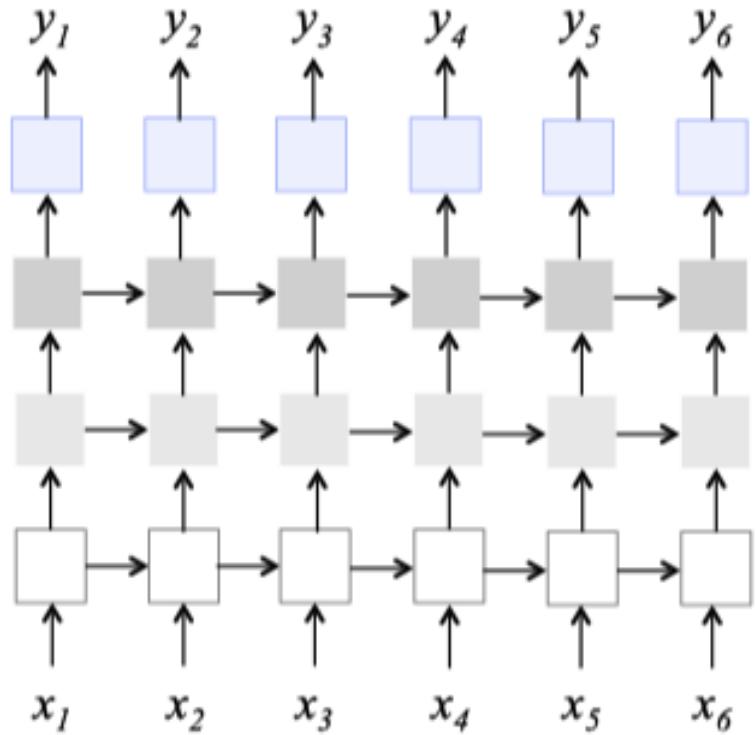
- Abbiamo una RNN in cui ognuno degli hidden state calcola una funzione $f(h, x)$ ovvero una funzione che dipende dall'output del precedente layer e dall'input corrente.

- Notare che la funzione f che viene calcolata in ognuno di questi hidden state è sempre la stessa in ciascun layer. Se avessimo funzioni differenti non avremmo più una RNN ma avremmo una normale feedforward Neural Network e non farebbe differenza avere più layers.
- Notare anche che gli output di tutti i livelli devono essere della stessa dimensione.

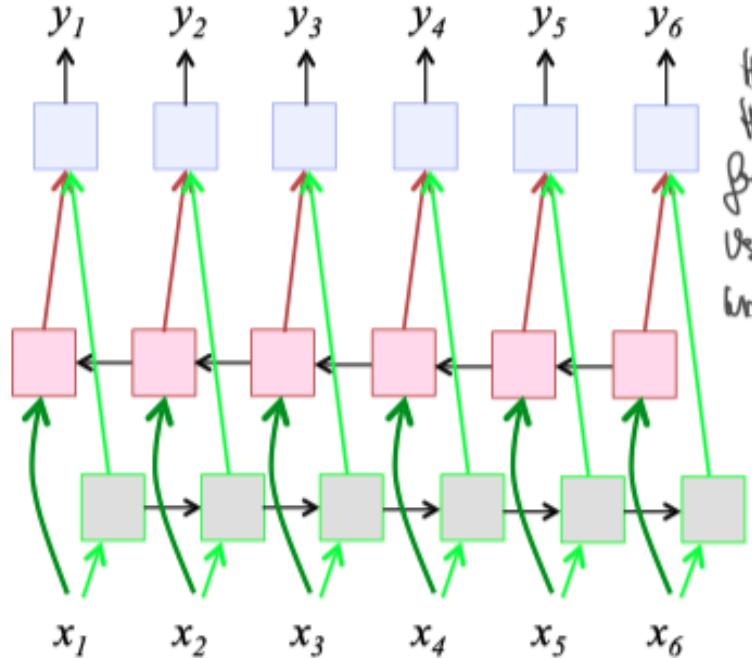
15.1.1 Tipologie di RNN

Possono esistere varie tipologie differenti di RNN, vediamone alcune:

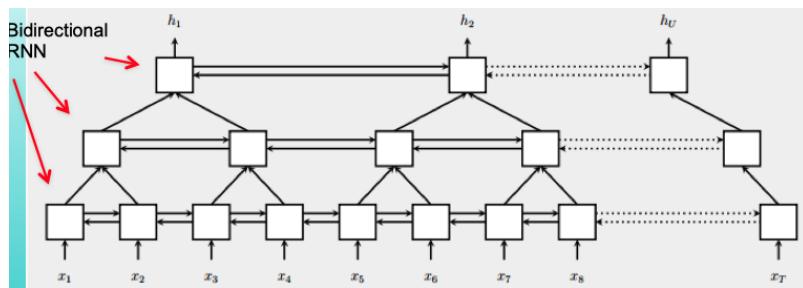
- Una prima versione di RNN è quella in cui abbiamo più di un singolo layer (verticalmente). Quello che succede è che ogni hidden state è collegato con l'hidden state dello stato successivo (tempo $t + 1$) e con l'altro layer dello stesso tempo t . Ad ognuno di questi stati poi corrisponde un output come nelle classiche RNN.



- Abbiamo RNN Bidirezionali in cui possiamo lavorare con lo stesso set di input in entrambe le direzioni, sia da sinistra a destra sia da destra a sinistra. Questo può aiutare a mitigare il non utilizzo di informazioni che arrivano dagli step successivi rispetto a quello in cui stiamo producendo l'output. Usando due RNN come in questo caso possiamo ottenere informazioni da entrambi i livelli e produrre quindi un output più "informato". Questa tipologia di RNN è utilizzato in particolare nello speech recognition.

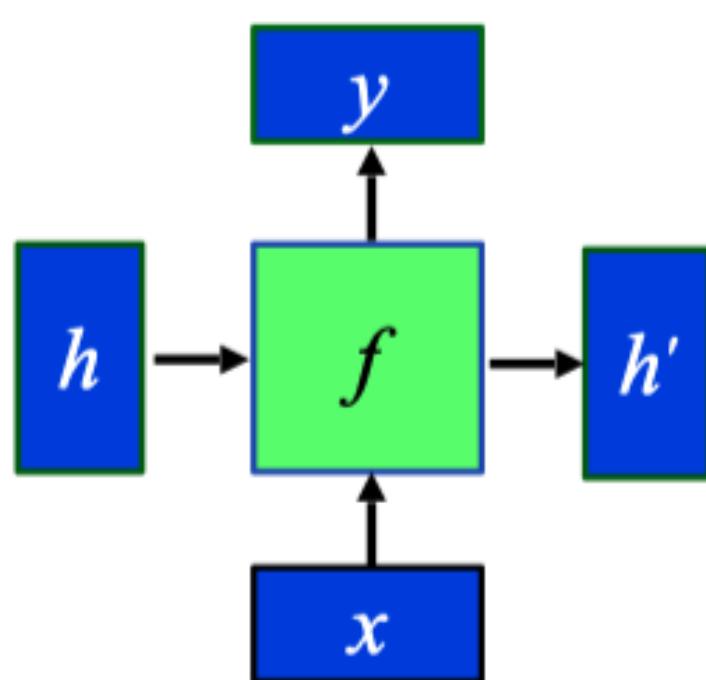


- Pyramid RNN: sono RNN che diminuiscono moltissimo il tempo necessario per il training, abbiamo vari layer (verticalmente) e abbiamo un collegamento bidirezionale.



- Vanilla RNN: in questo caso abbiamo una RNN che esegue una trasformazione molto semplice, queste sono le RNN più semplici in assoluto.

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.



In particolare gli output vengono definiti in questo modo:

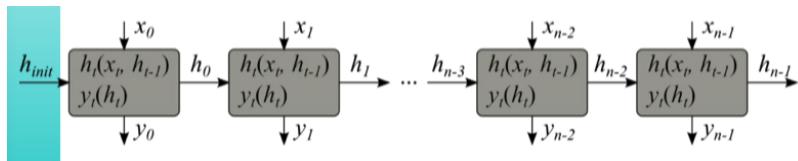
$$h' = \sigma(W^h h + W^i x + b)$$

nella formula abbiamo utilizzato la sigmoid function e all'interno abbiamo un hidden vector, usiamo il nostro input x , e il bias b , i pesi che utilizziamo sono diversi tra hidden vector e input.

$$y = \sigma(W^o h')$$

Per quanto riguarda la definizione dell'output, utilizziamo una matrice di pesi differente per calcolare l'output y applicandola all'output h' . Qua possiamo notare che la y viene calcolata a partire dalla h' . Possiamo vedere questa Vanilla RNN anche esplosa:

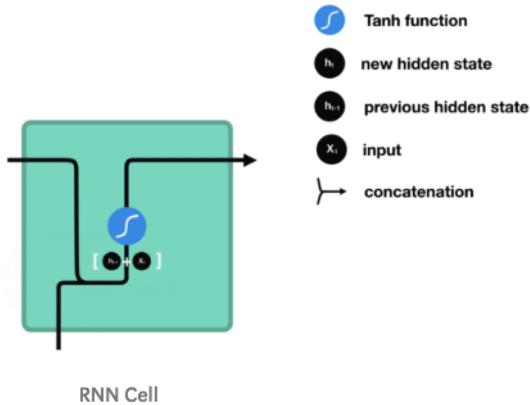
Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.



15.1.2 Training di RNN

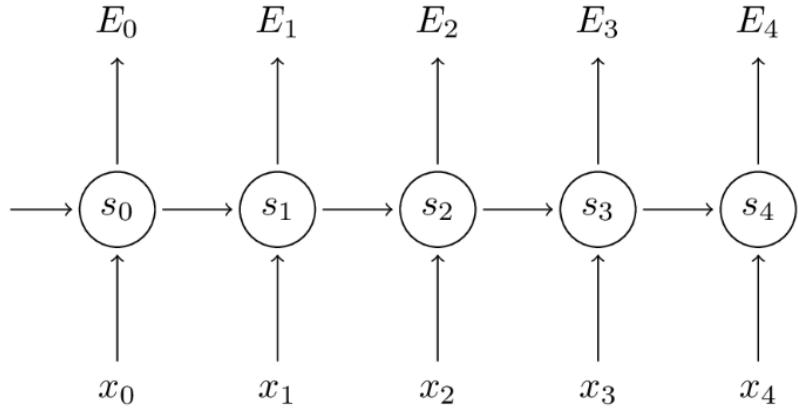
Prima di utilizzare una RNN dobbiamo eseguire il training, l'obiettivo è di ottenere i parametri per la nostra RNN andando ad ottimizzare una cost function.

Per eseguire l'ottimizzazione della cost function e per trovare i parametri migliori per la RNN utilizziamo un metodo differente che è chiamato backpropagation Through Time che si differenza dalla classica backpropagation.



Per capire come funziona la backpropagation Through Time possiamo considerare l'esempio della seguente immagine:

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.



Riscriviamo la formula dell'hidden state e dell'output di questa RNN:

$$s_t = \tanh(Ux_t + Ws_{t-1})$$

$$y_t^* = \text{softmax}(Vs_t)$$

L'activation function \tanh è usata per regolare i valori che passano attraverso la rete, questa riduce il valore che viene dato in input tra -1 e 1. In questo modo riusciamo ad evitare che il valore che passa attraverso la rete esploda, immaginiamo di moltiplicare per 3, di unità in unità avremmo ogni volta un valore più grande e questi questo porterebbe ad una esplosione dei valori. Usando la \tanh invece evitiamo questo problema perchè l'output sarà sempre tra -1 e 1.

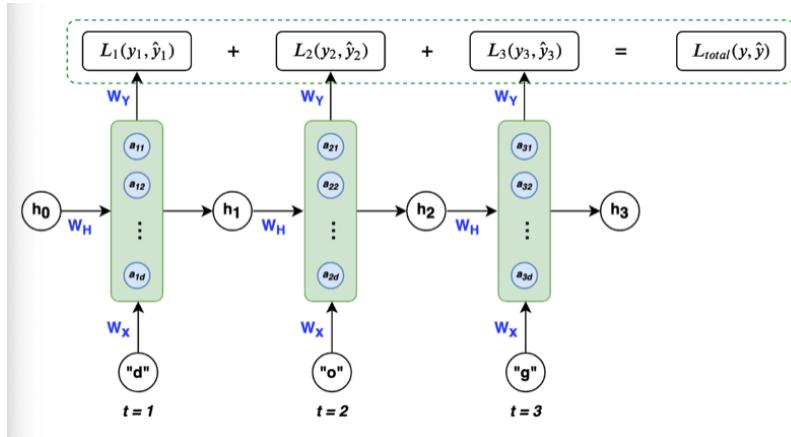
Possiamo anche definire la nostra loss utilizzando la cross entropy:

$$E(y, y^*) = \sum_t E_t(y_t, y_t^*)$$

dove indichiamo con y^* la nostra previsione e con y l'output reale. In questo caso noi stiamo considerando la loss complessiva considerando tutti i timestamp. In ognuno dei timestamp poi possiamo anche produrre una loss relativa a quello specifico timestamp.

Il nostro obiettivo qua è calcolare il gradiente dell'errore rispetto ai parametri del nostro modello che chiameremo $\Theta = U, V, W$.

Vediamo come funziona la Backpropagation Through Time, considerando la seguente architettura:



Abbiamo tre matrici di pesi e quindi qua dobbiamo calcolare il gradiente rispetto a queste tre matrici e dobbiamo aggiornarle basandoci sulle informazioni che ci dà il gradiente.

Vediamo il caso in cui vogliamo calcolare il gradiente rispetto a W_Y , la loss function la possiamo calcolare in questo modo e abbiamo la seguente catena di dipendenze rispetto alla W_Y :

$$L_t = -y_t \log(y_t^*) \rightarrow y_t^* = \text{softmax}(z_t) \rightarrow z_t = W_Y h_t$$

In particolare la chain rule rispetto alla W_Y diventa la seguente:

$$\frac{\nabla L_t}{\nabla W_Y} = \frac{\nabla L_t}{\nabla y_t^*} \frac{\nabla y_t^*}{\nabla z_t} \frac{\nabla z_t}{\nabla W_Y}$$

questa sopra è la formula della chain rule in cui consideriamo uno specifico tempo t . Ora quando calcoliamo il gradiente rispetto a W_Y dobbiamo considerare tutti i possibili t e per ognuno applicare la chain rule. Quindi otteniamo una cosa di questo genere:

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

$$\frac{\partial L_{total}}{\partial W_Y} = \frac{\partial L_1}{\partial \hat{y}_1} \frac{\partial \hat{y}_1}{\partial z_1} \frac{\partial z_1}{\partial W_Y} + \frac{\partial L_2}{\partial \hat{y}_2} \frac{\partial \hat{y}_2}{\partial z_2} \frac{\partial z_2}{\partial W_Y} + \frac{\partial L_3}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial z_3} \frac{\partial z_3}{\partial W_Y} = \sum_{t=1}^n \frac{\partial L_t}{\partial W_Y}$$

Per gli altri parametri il ragionamento è simile, avremmo infatti:

$$\begin{aligned}
 & \text{Function Dependencies with respect to } W_X \\
 L_t &= -y_t \log(\hat{y}_t) \rightarrow \hat{y}_t = \text{softmax}(z_t) \rightarrow z_t = W_Y h_t \rightarrow h_t = \tanh(W_H h_{t-1} + W_X X_t) \quad (1) \\
 & \text{Chain rule with respect to } W_X \\
 \frac{\partial L_t}{\partial W_X} &= \frac{\partial L_t}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial z_t} \frac{\partial z_t}{\partial h_t} \frac{\partial h_t}{\partial W_X} \text{ but note, that within } h_t, h_{t-1} \text{ also contains } W_X, \text{ thus we need to recursively chain rule } h_{t-1} \text{ until we reach } h_0 \quad (2) \\
 \frac{\partial L_t}{\partial W_X} &= \frac{\partial L_t}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial z_t} \frac{\partial z_t}{\partial h_t} \frac{\partial h_t}{\partial W_X} + \frac{\partial L_t}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial z_t} \frac{\partial z_t}{\partial h_t} \frac{\partial h_t}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial W_X} + \dots + \frac{\partial L_t}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial z_t} \frac{\partial z_t}{\partial h_t} \frac{\partial h_t}{\partial h_{t-n}} \frac{\partial h_{t-n}}{\partial W_X} = \sum_{k=0}^n \frac{\partial L_t}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial z_t} \frac{\partial z_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial W_X} \quad (3) \\
 & \text{Gradient for } W_X \\
 \frac{\partial L_{total}}{\partial W_X} &= \sum_{t=1}^n \sum_{k=0}^n \frac{\partial L_t}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial z_t} \frac{\partial z_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial W_X} \quad (4)
 \end{aligned}$$

Notare che qua abbiamo la W_X anche all'interno della h_{t-1} quindi quando facciamo la backpropagation questa cosa la dobbiamo considerare e infatti dobbiamo arrivare al $t = 0$, ecco perchè nel calcolo del gradiente abbiamo due sommatorie, una per muovermi su tutti i possibili t e una per muovermi dalla h_{t-1} fino alla h_0 .

15.1.3 Vanishing Gradient

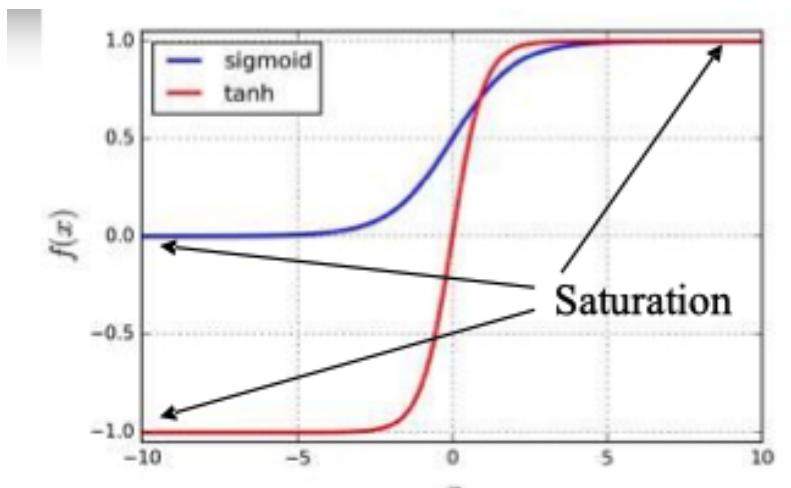
Il problema del Vanishing Gradient non è un problema esclusivo delle RNN, in generale possiamo dire che è un problema di tutte le Neural Network. Nel caso delle RNN il problema si sente di più perchè per sua natura questa è una NN che va più in profondità per cercare di valutare tutti i possibili legami tra le varie parole di una frase.

Il problema si verifica in particolare quando nella matrice dei pesi di partenza abbiamo valori molto piccoli mentre è almeno limitato quando i valori della matrice di partenza sono più grandi. Il problema del Vanishing gradient capita quando abbiamo che il calcolo del gradiente a mano a mano che andiamo indietro nei vari layer diventa sempre più piccolo

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

fino a che non arriviamo ad avere un gradiente pari a 0 che non consente quindi di comprendere la correlazione tra le parole e non permette neanche di aggiornare i pesi a mano a mano che andiamo avanti.

Il problema è più chiaro se consideriamo le activation function che utilizziamo, sia la sigmoid che la tanh infatti quando trovano un valore che è minore di 0 tendono ad assegnare un valore che è sempre lo stesso ovvero 0 (nel caso della sigmoid succede anche per valori più grandi di 0 quando assegnamo sempre 1).



Questo comporta che quando calcoleremo la derivata avremo la derivata pari a 0 che quindi causerà il problema del vanishing gradient.

Il problema del Vanishing gradient lo possiamo parzialmente risolvere andando a scegliere una differente inizializzazione della matrice dei pesi W oppure cambiando la activation function utilizzando ad esempio ReLU al posto della sigmoid o della tanh.

15.1.4 Exploding Gradient

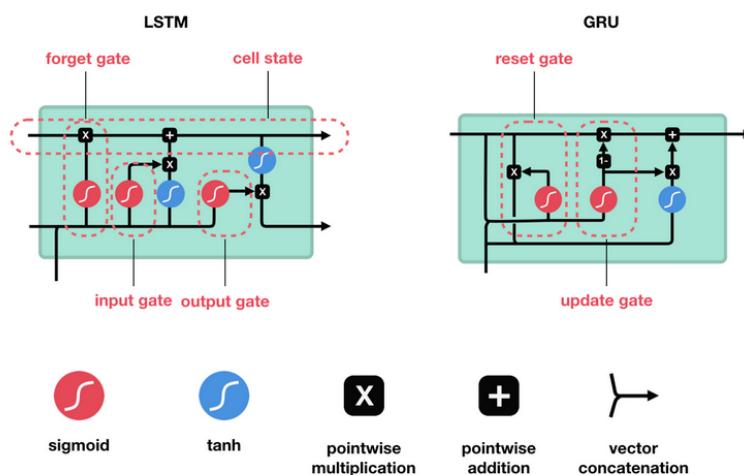
Si tratta del problema opposto del vanishing gradient, qua il gradiente arriva ad avere un valore molto grande e quindi alla fine si arriva ad

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

una situazione in cui il gradiente esplode, otteniamo Nan ad esempio. Per risolvere il problema dell'exploding gradient potremo utilizzare il gradient clipping o potremo ridurre il learning rate.

15.2 LSTM

Come abbiamo detto fino ad ora il problema principale delle RNN consiste nel fatto che non riusciamo a catturare realmente le relazioni tra le varie parole, specialmente tra quelle che sono molto lontane tra loro. Inoltre avere reti così profonde porta anche ad avere il problema del vanishing gradient.



La soluzione a questi problemi sono le LSTM e GRU che hanno un meccanismo interno chiamato gate che permette di regolare il flusso delle informazioni. Questi gates sono in grado di capire quali informazioni sono rilevanti e quali no per la previsione finale.

Ad esempio possiamo avere una certa recensione:

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

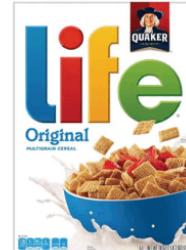
Customers Review 2,491



Thanos

September 2018
Verified Purchase

Amazing! This box of cereal gave me a perfectly balanced breakfast, as all things should be. I only ate half of it but will definitely be buying again!



A Box of Cereal
\$3.99

E la LSTM o GRU vogliono ricordarsi solamente le informazioni che sono rilevanti per il task finale di classificazione (decidere se è positiva o negativa):

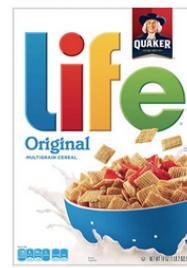
Customers Review 2,491



Thanos

September 2018
Verified Purchase

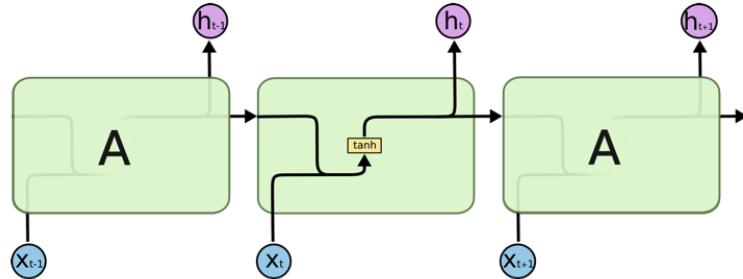
Amazing! This box of cereal gave me a perfectly balanced breakfast, as all things should be. I only ate half of it but will definitely be buying again!



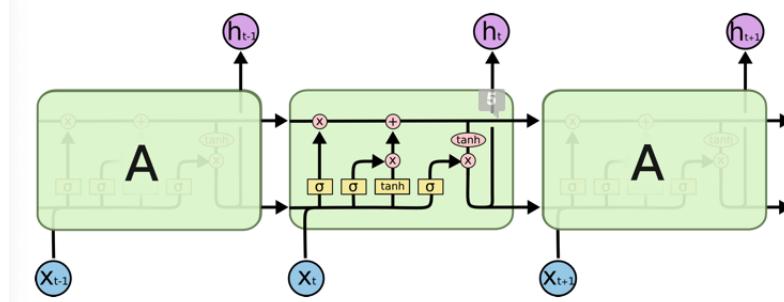
A Box of Cereal
\$3.99

Le LSTM (Long Term Memory Networks) sono delle speciali RNN in grado di eseguire il learning delle dipendenze tra le parole che sono lontane tra loro in un testo. La caratteristica principale di queste LSTM è che riescono a memorizzare informazioni per un periodo molto lungo, cosa che non riusciamo a fare nelle classiche RNN. Nella classica RNN infatti abbiamo una struttura di questo genere, con una catena di moduli che si ripetono e che hanno una struttura interna molto standard e soprattutto molto semplice

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.



La differenza con le LSTM la possiamo vedere anche direttamente considerando solamente la struttura di questi moduli all'interno della catena. Qua all'interno dei moduli abbiamo qualcosa di più complesso, ad esempio abbiamo 4 layer della rete neurale e non solamente uno come nelle RNN.



All'interno di questi schemi utilizziamo la seguente notazione:



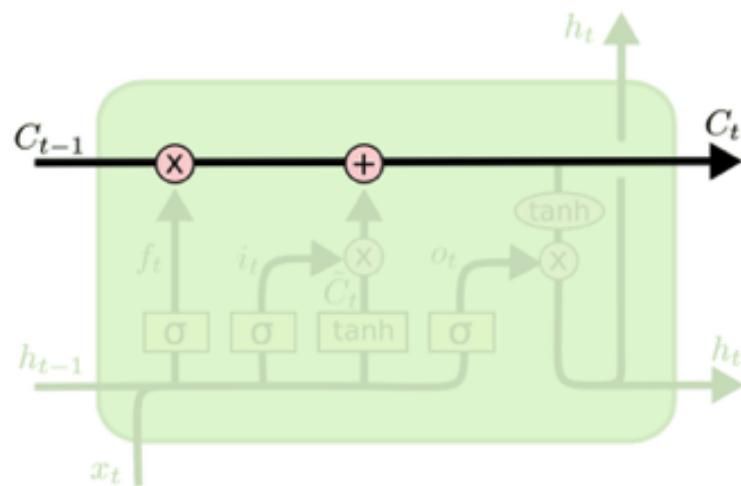
Vediamo nel dettaglio come è fatta la LSTM.

Cell State

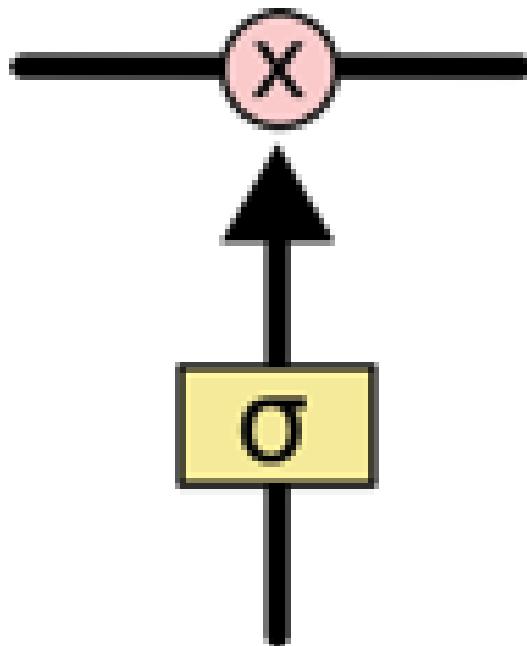
Una parte fondamentale delle LSTM è sicuramente la cell state ovvero la linea orizzontale che attraversa tutta la catena andando ad interagire

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

solamente in alcuni punti dei moduli. È molto utile per fare in modo che le informazioni viaggino lungo questa linea.

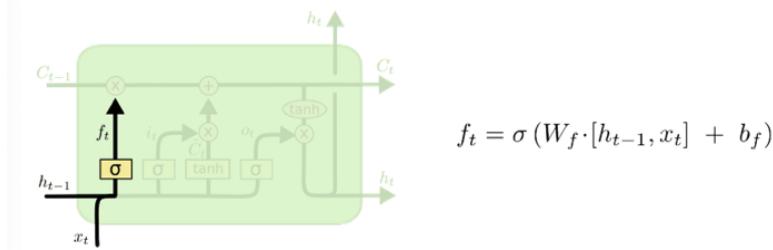


Per modificare le informazioni presenti all'interno di questa linea vengono utilizzati dei Gate ovvero dei componenti in cui abbiamo una sigmoid function. La sigmoid function mi genera un valore compreso tra 0 e 1 che viene moltiplicato per il contenuto della linea.



Forget Gate Layer

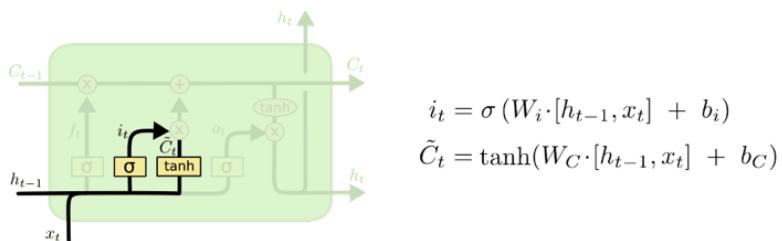
Il forget gate layer è un componente della LSTM che va a considerare l'input x_t e l'input h_{t-1} e utilizza poi la sigmoid function per generare un output compreso tra 0 e 1. Si utilizza la Sigmoid e non la tanh perchè in questo modo l'output è tra 0 e 1 e quindi quando poi moltiplichiamo per questo output, se il valore è 0, avremo 0 come risultato. Questo forget gate layer serve perchè ci permette di capire se vogliamo cancellare quello che c'è nel cell state (questo succede quando la sigmoid genera 0), se vogliamo mantenere tutto (questo succede quando la sigmoid genera 1) o se invece vogliamo mantenere in parte i dati (abbiamo un valore compreso tra 0 e 1, se siamo vicini a 0 manteniamo poco ad esempio).



Ad esempio se nella cell state abbiamo una informazione che mi dice se la parola che stiamo considerando è maschile o femminile e io voglio andare a considerare solamente il caso della parola corrente, andrò a cancellare l'informazione precedente.

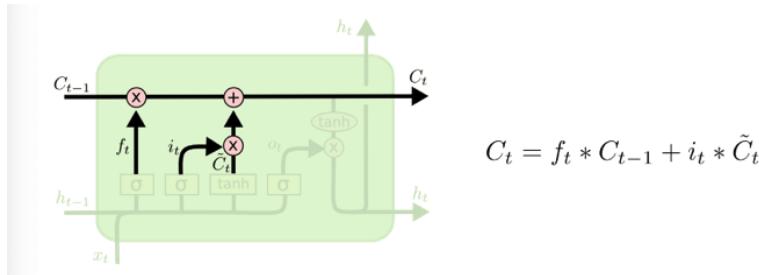
Input Gate Layer

Dopo aver deciso se cancellare o meno le informazioni presenti sulla linea dobbiamo decidere quali sono le informazioni da aggiungere in questo step. Quello che facciamo è andare prima a calcolare una sigmoid function per decidere quale valore aggiornare e poi con un layer tanh generiamo quello che effettivamente dovrebbe essere aggiunto al cell state (la linea).

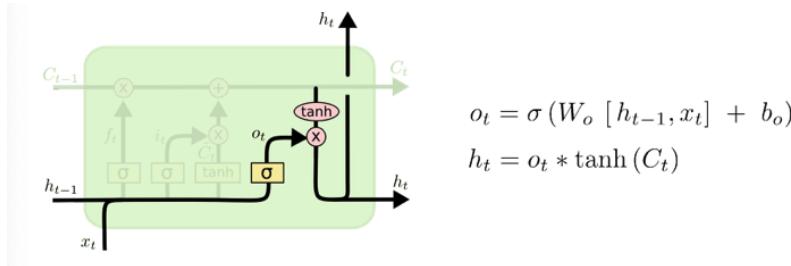


Una volta presa la decisione e calcolata la sigmoid possiamo andare a modificare il contenuto del cell state, in particolare utilizziamo l'informazione calcolata nel forget Gate Layer per azzerare quello che abbiamo deciso di azzerare e poi sommiamo quello che abbiamo deciso di aggiungere al cell state.

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.



Arrivati a questo punto dobbiamo decidere quali sono i dati che verranno mandati in output. Questo dipende dal cell state ma anche da una sigmoid function che mi indica quali dei dati provenienti dalla sigmoid function dovrò considerare e quali saranno azzerati. In pratica qua prendo i dati dal cell state, li passo in una funzione tanh e poi moltiplico per il risultato dell'applicazione della sigmoid.



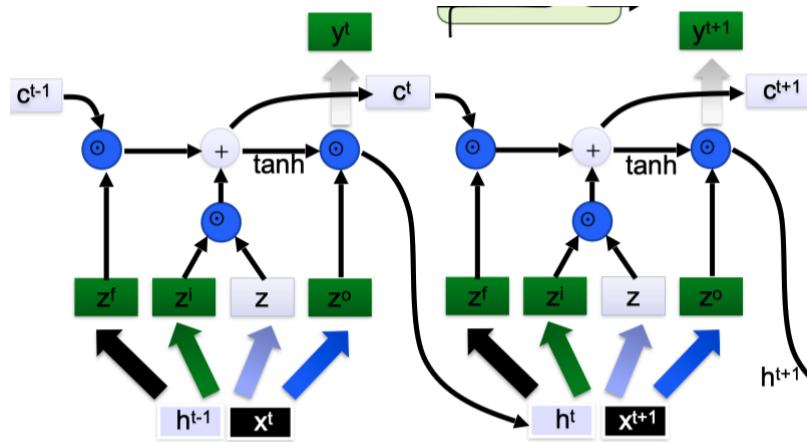
In questo modo produco l'output di questa specifica parte della catena e inviamo l'output anche al prossimo step della catena.

Funzionamento complessivo

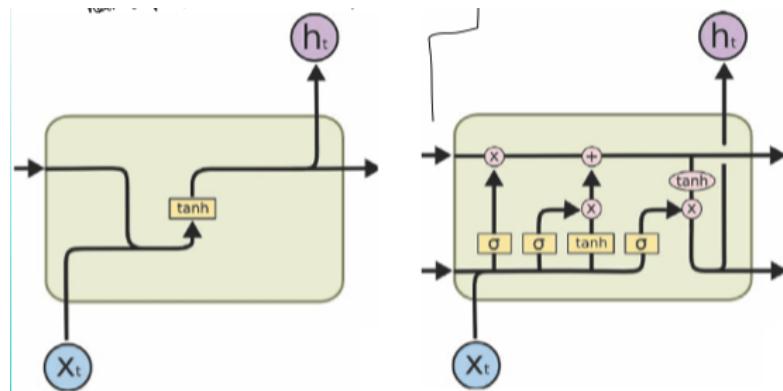
La cosa positiva è che il calcolo di tutti questi componenti interni ai vari step possono essere effettuati in parallelo. Nelle computazioni che facciamo all'interno del modulo abbiamo sempre lo stesso input, quello che cambia è la matrice dei pesi W che utilizziamo per la moltiplicazione.

Possiamo vedere il funzionamento complessivo dell'LSTM in parallelo in questa immagine:

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.



15.2.1 Confronto



Nella foto possiamo vedere il confronto tra le RNN e le LSTM. In particolare le RNN prendono semplicemente l'input corrente e l'input dall'hidden state precedente e poi l'output va allo stato successivo e all'output corrente.

Nel caso della LSTM invece abbiamo due input che arrivano dal precedente stage e a nostra volta in questo stage produciamo due output.

15.3 Implementare il POS Tagger usando RNN

Abbiamo già implementato il POS Tagger utilizzando gli HMM ora vogliamo fare la stessa cosa utilizzando LSTM.

Per farlo utilizzeremo Keras e il brown corpus considerando solamente le frasi della categoria news e lo universal tagset.

Per le frasi presenti all'interno del corpus andiamo a creare una lista di coppie formate dalla parola stessa e dal POS corrispondente. I POS che vengono assegnati alle varie parole li possiamo anche memorizzare in una lista in cui memorizziamo i tag che compaiono.

Quando creiamo le frasi da analizzare dobbiamo trasformarle da frasi con parole a frasi in cui le parole sono sostituite da numeri. Per questo definiamo una funzione che mi permette di assegnare un numero ad ognuna delle parole presenti nelle frasi, in questa fase aggiungiamo anche altri due identificatori che sono 0 per il padding e 1 per le parole che non conosciamo.

Una volta che abbiamo preparato le frasi che vogliamo analizzare per eseguire il tagging delle parti del discorso possiamo fare la divisione tra train e test set e possiamo decidere alcuni parametri della RNN come ad esempio la dimensione degli embedding e delle hidden units della RNN.

I nostri sample di dati hanno una lunghezza uniforme ma in alcuni casi abbiamo del padding che dovrebbe essere ignorato, per ignorare questo padding possiamo utilizzare un parametro all'interno della definizione degli embedding. Il parametro in questione è *mask_zero = true* e permette di ignorare il padding.

15.3.1 Creiamo i layer

```
model.add(Embedding(len(word2index), EMBEDDING_DIM,
input_length=MAX_LENGTH, mask_zero=True))
```

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

```

model.add(Bidirectional(GRU(RNN_HIDDEN_SIZE,
    return_sequences=True, merge_mode='concat'))

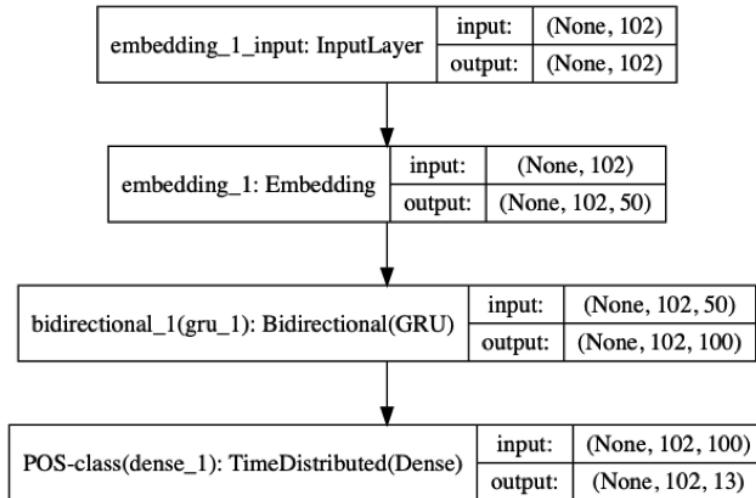
model.add(TimeDistributed(Dense(TAGSET_SIZE, activation='softmax'),
#input_shape=(BATCH_SIZE, MAX_LENGTH, RNN_HIDDEN_SIZE*2), name='POS-class'))

```

Vediamo quali sono i layer della nostra RNN:

- Abbiamo un primo layer per gli embedding, quando lo creiamo dobbiamo passare come parametro *mask_zero = true* e la dimensione che vogliamo per gli embedding.
- Bidirectional Layer: questo lo creiamo passando GRU come parametro. Tra i parametri che passiamo a GRU ci sono *merge_mode = 'concat'* e *return_sequences = true*. Il primo serve per concatenare i risultati che otteniamo e il secondo serve per considerare gli output da ognuna delle celle, vogliamo ricordare tutti gli output. Il bidirectional Layer serve per avere una RNN in cui le informazioni viaggiano da sinistra a destra e viceversa.
- Abbiamo un Dense Time Distributed layer (è un layer fully connected) perchè vogliamo un output per ognuno dei token che abbiamo all'interno delle frasi, quando facciamo questo passaggio passiamo il parametro *RNN_HIDDEN_SIZE * 2* perchè le GRU sono concatenate.

Il modello che otteniamo è il seguente:



In input abbiamo frasi di lunghezza massima uguale a 102, questo lo vediamo perchè nell'input layer abbiamo in input (e in output) (*None*, 102).

Nel layer degli embedding abbiamo una dimensione in output di (*None*, 102, 50) perchè abbiamo la dimensione degli embedding che è 50, per ognuna delle parole presenti nella frase (di massimo 102 parole) abbiamo un embedding di dimensione 50. Quindi qua l'output è una matrice 102 * 50.

Nel layer successivo con il Bidirectional(GRU) abbiamo l'output che è (*None*, 102, 100) perchè abbiamo che la dimensione viene raddoppiata per le bidirectional GRU sono concatenate.

Alla fine otteniamo un output che è di dimensione (*None*, 102, 13) perchè abbiamo 13 possibili classificazioni per le varie parole delle frasi.

Una volta che abbiamo fatto tutto questo possiamo passare ad eseguire il training del nostro modello e possiamo anche testarlo.

15.3.2 GRU

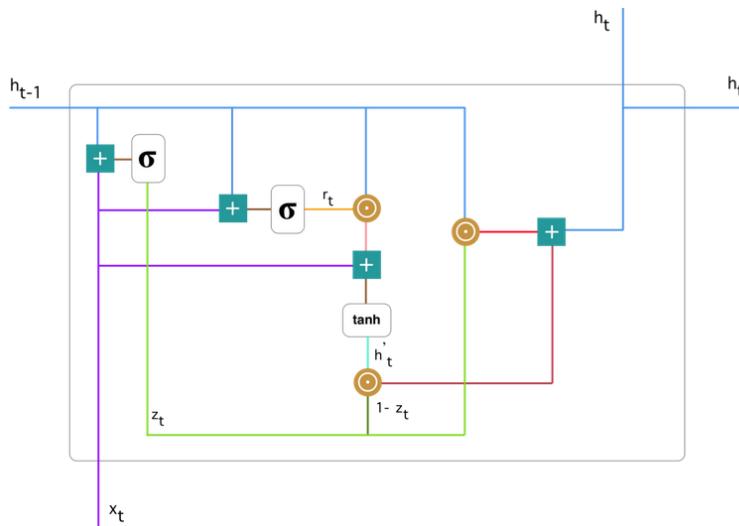
Fino ad ora avevamo visto LSTM che permette di risolvere il problema delle RNN, ora in questo esempio pratico abbiamo visto anche l'utilizzo

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

di GRU che è una versione semplificata delle LSTM. Sono simili ma la struttura interna è più semplice. Tutto questo aiuta per considerare tutta la sequenza quando dobbiamo fare la classificazione.

Rispetto a LSTM, GRU è più recente ed essendo più semplice all'interno ha anche meno parametri per cui va fatto il training quindi potrebbe essere più veloce o potrebbe volere meno dati per generalizzare. In generale le performances sono più o meno uguali ma se abbiamo tanti dati allora LSTM potrebbe funzionare meglio e portarci a migliori risultati.

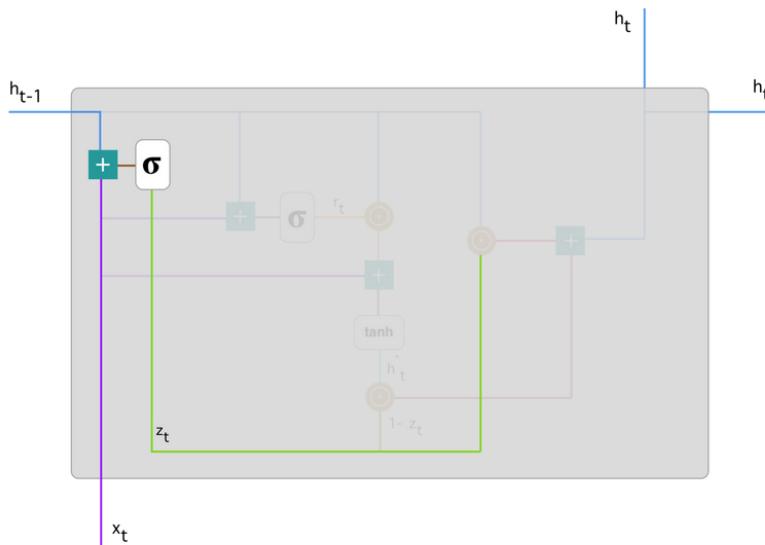
Anche GRU si pone come obiettivo quello di risolvere il problema del vanishing gradient utilizzando gli update gate e i reset gate, la caratteristica interessante è che il training viene fatto per fare in modo che vengano mantenute informazioni di tanto tempo fa senza rimuoverle nel corso del tempo e anche senza rimuovere quelle informazioni che sono irrilevanti per la previsione. Vediamo nel dettaglio come funziona GRU, questo è il tipico schema di funzionamento:



- Update Gate: il primo componente della rete che vediamo è l'update gate, quello che facciamo è prendere x_t che è l'input corrente e h_{t-1} che contiene tutte le informazioni che arrivano dalle precedenti $t-1$

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

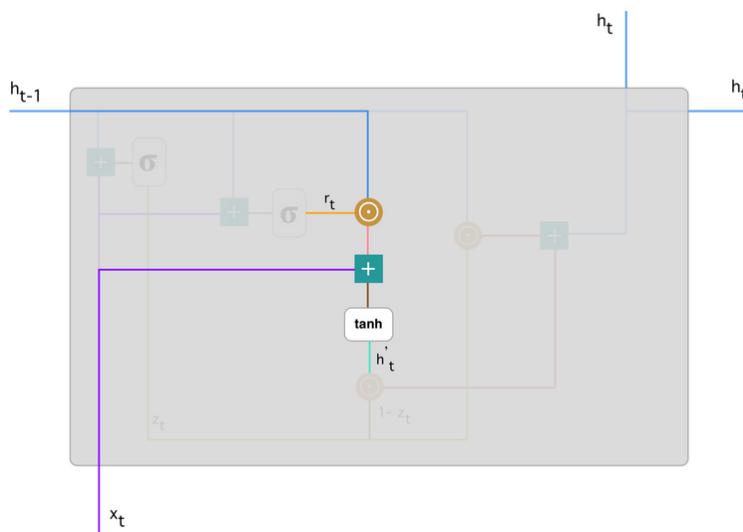
unità. Questi due valori sono moltiplicati per i pesi corrispondenti poi vengono sommati e poi si applica la Sigmoid function.



L'update gate mi serve perchè mi permette di determinare quanto delle informazioni che arrivano dagli step precedenti devono essere inoltrate alla prossima unità. Ad esempio se copiamo tutte le informazioni del passato riusciamo ad evitare il problema del vanishing gradient.

- Reset Gate: questo gate viene utilizzato per decidere quante delle informazioni passate devono essere dimenticate. Il funzionamento è simile all'update gate, anche qua sommiamo x_t che è l'input corrente e h_{t-1} che contiene tutte le informazioni che arrivano dalle precedenti $t - 1$ unità, moltiplichiamo per i pesi corrispondenti e poi applichiamo la sigmoid function.
- Memoria corrente: quello che facciamo qua è prendere l'input dello stato precedente h_{t-1} , moltiplicarlo per il suo peso e poi moltiplica-

carlo per il risultato del Reset Gate. Poi sommiamo questo risultato con l'input x_t che viene moltiplicato a sua volta per un suo peso. Alla somma di questi due valori viene applicata una funzione \tanh che mi restituisce in output h'_t .

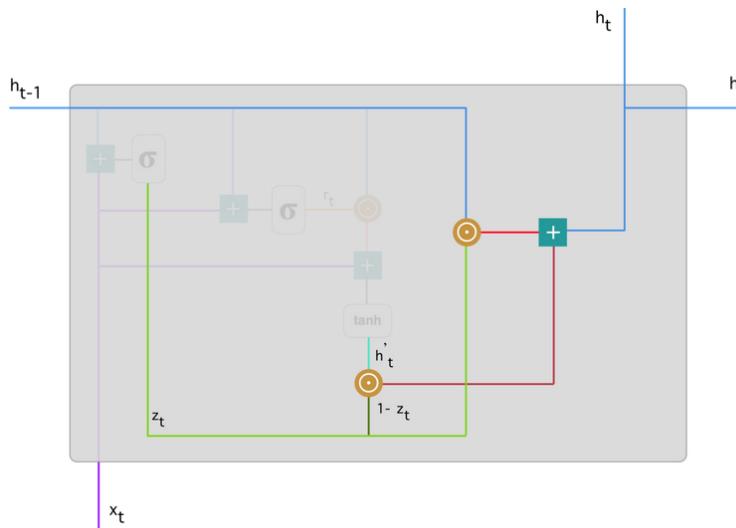


- Come ultimo step dobbiamo calcolare h_t che contiene le informazioni dell'unità corrente, questo valore viene mandato in output e poi viene inviato alla successiva unità. La formula che viene applicata in questo caso è:

$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot h'_t$$

Ovvero prendiamo h_{t-1} e lo moltiplichiamo per z_t che sarebbe

l'output dell'update gate, poi il risultato viene sommato alla moltiplicazione tra $(1 - z_t)$ e h'_t che sarebbe la memoria corrente.



Questa ultima operazione quindi produce l'output che viene inviato alla unità successiva e che viene anche inviato in output.

Avere questo tipo di architettura, con dei dati che vengono inviati a tutte le varie unità permette all'LSTM e anche a GRU di ricordare l'esistenza di feature specifiche anche se queste sono state identificate a molti step di distanza da quella unità. L'aggiunta di questo componente fa sì che si crei un path che mi permette di eseguire facilmente la back-propagation facilmente senza avere problemi con il vanishing gradient.

LSTM e GRU hanno ovviamente anche delle differenze:

- LSTM controlla quanto della memoria corrente deve essere esposta mentre GRU non fa questo controllo ed espone tutto il contenuto della sua memoria senza alcun controllo.

- Un'altra differenza riguarda l'input gate, l'LSTM calcola il nuovo contenuto della memoria senza un controllo sulla quantità di informazioni che arrivano dal precedente time step. GRU invece controlla il flusso di informazioni che arrivano dalla precedente unità quando calcola la nuova activation ma non controlla indipendentemente la quantità di activation che deve essere aggiunta (il controllo è legato all'update gate).

Chapter 16

Lesson 15 - Machine Translation: Introduction

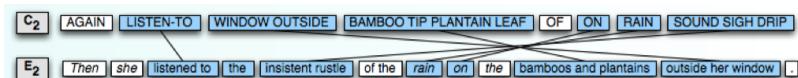
Con Machine Translation intendiamo la traduzione di un testo da un linguaggio ad un altro in modo automatico. L'obiettivo è ottenere una machine translation di alta qualità e questo è divenuto possibile nel corso degli anni, specialmente da quando per questo task si utilizzano le reti neurali. I primi test con la Machine Translation non sono stati così buoni, sono stati effettuati ad esempio da Altavista che però era stata criticata perché i risultati ottenuti non erano buoni e quello che si otteneva era più che altro una traduzione parola per parola del testo.

Con la traduzione parola per parola il risultato che otteniamo non è ottimo, quello che otteniamo è un qualcosa che ci permette di intuire a grandi linee il significato delle frasi senza però avere una correttezza poi nell'effettiva traduzione.

16.1 Machine Translation

I problemi che possono sorgere quando si parla di Machine Translation sono vari:

- Possiamo avere problemi con la segmentazione delle frasi, in particolare abbiamo che in alcuni linguaggi abbiamo una frase unica per indicare la stessa frase che in altri linguaggi separiamo in varie frasi.
- Possiamo avere delle differenze a livello di grammatica, ad esempio nel Cinese non si utilizzano quasi mai verbi al passato, cosa che invece succede in inglese. Sempre prendendo come esempio il cinese, non abbiamo pronomi e articoli che possiamo utilizzare.
- Abbiamo all'interno delle frasi anche delle differenze stilistiche dovute a differenze culturali e a differenze di stile nel modo di scrivere. Questa qua è la parte più complessa da gestire.
- Un'altra cosa che risulta complicata da effettuare è l'allineamento delle frasi che traduciamo. Nella lingua sorgente potremmo avere un ordinamento che sarà differente rispetto alla lingua di destinazione. Ad esempio quando facciamo la traduzione dal cinese all'inglese oltre al riordinare la frase in base alle regole della grammatica inglese dobbiamo anche aggiungere davanti alle parole gli articoli, quando servono.



Attualmente la Machine Translation è arrivata ad un livello in cui alcuni task sono fattibili e i risultati ottenuti sono buoni mentre altri task sono ancora troppo complessi.

- Quando ci basta una traduzione che non sia precisa al 100% la MT può essere una buona soluzione.

- Quando abbiamo la necessità di fare una traduzione ma poi non usiamo direttamente il risultato ma andiamo anche ad fare delle modifiche allora possiamo usare la MT.
- Non riusciamo ancora ad avere risultati eccellenti quando dobbiamo tradurre testi letterari o anche espressioni che sono utilizzate nel quotidiano quando parliamo, in alcuni linguaggi potrebbero esserci infatti dei modi di dire che non riusciamo a tradurre.
- Ci sono poi delle situazioni in cui il risultato che otteniamo diventa determinante, pensiamo ad esempio all'utilizzo della MT in campo medico, è necessario che la traduzione sia perfetta, allo stesso modo quando dobbiamo fare delle chiamate di emergenza.

16.2 Storia della Machine Translation

All'inizio della storia della Machine translation, la traduzione era stata pensata tramite un dizionario, presa una certa parola si andava a scrivere in questo dizionario la parola corrispondente nell'altro linguaggio e in questo modo si andavano a tradurre interi documenti. Il problema chiaramente era legato al fatto che avevamo solamente una traduzione parola per parola del testo visto che per ogni parola si andava a prendere la corrispondente traduzione dal dizionario.

Poi una prima evoluzione c'è stata quando Warren Weaver ha pubblicato un report in cui ha evidenziato il fatto che tutti i linguaggi hanno delle caratteristiche in comune, queste caratteristiche sono invarianti, inoltre nello stesso report suggerì anche che sarebbe stato possibile utilizzare i computer per la machine translation.

Inizialmente il task della Machine Translation fu considerato eccessivamente complesso (fine anni 50), questo perchè sembrava fosse impossibile comprendere il significato di una parola o di una frase e quindi andare di conseguenza a fare la traduzione. La conseguenza di tutto ciò

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

inizialmente è stato il fatto che era necessario includere della conoscenza umana per poter arrivare all'obiettivo della machine translation.

Sempre nello stesso periodo la ricerca nel campo della Machine Translation ebbe un declino perchè pochissime persone erano interessate a lavorare in questo campo. Gli interessi riguardanti la MT erano legati in particolare al fatto che si volevano raccogliere materiali russi e tradurli, ad esempio tutta la letteratura russa era stata tradotta in questo periodo, da umani però, non da macchine.

Negli anni 70 qualcosa cambiò, in Europa si cominciò a parlare di nuovo di Machine Translation e poi negli anni 80 si cominciarono ad utilizzare delle tecniche di AI nel campo della MT. Negli anni 90 vennero pubblicati i primi sistemi commerciali di Machine Translation utilizzando tecniche statistiche nel campo della MT. Queste tecniche vennero utilizzate fino agli anni 2000 quando poi arrivò Google Translate. Tutto cambiò nuovamente nel 2015 quando Google passò all'utilizzo delle reti neurali per le traduzioni su Google Translate. Questo campo continua anche ora e continua la sua crescita e continua a produrre nuovi risultati.

16.3 Somiglianze e variazioni nei linguaggi

Come abbiamo già detto i linguaggi hanno molti aspetti che sono fortemente in comune, altri invece sono in comune ma la somiglianza è meno forte, abbiamo poi degli aspetti dei linguaggi che sono completamente differenti da un linguaggio all'altro. Lo studio di differenze e di similitudini tra i vari linguaggi viene chiamato Typology.

16.3.1 Morphology

La Morphology è uno degli aspetti in cui si differenziano maggiormente i linguaggi.

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

- Definiamo come Morpheme la più piccola unità del linguaggio dotata di significato, concatenando varie morpheme otteniamo una parola
- Stems: gli stems sono la parte base di una parola, ad esempio se consideriamo la parola *hoping* abbiamo che la forma base (stem) è la sola *hope*.
- Affixes: qua facciamo una distinzione tra i prefissi ovvero la parte di parola che mettiamo all'inizio di una parola, ad esempio "Anticalcio" dove Anti è il nostro prefisso, e i suffissi ovvero la parola che mettiamo alla fine della parola, ad esempio "Antidisestablishmentarianism" con -ism che è il nostro suffisso. Sempre in questa sezione abbiamo anche infixes, che rappresentano la parte centrale della parola e circumfixes che sono una parte della parola che comprende la parte iniziale e quella finale.

Tra i vari linguaggi abbiamo delle variazioni in termini di morfologia e in particolare può essere utile fare una distinzione tra:

- Isolating Languages: sono i linguaggi che hanno parole che sono generalmente formate da un singolo morpheme.
- Polysynthetic Languages: una singola parola in questi linguaggi può essere formata da vari morphemes.



e poi tra:

- Agglutinative Languages: in questi linguaggi i morphemen hanno dei confini ben definiti e questo comporta che possiamo segmentare le parole in modo molto semplice.

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

- Fusion Languages: ad esempio il russo, dove abbiamo vari morphemes senza però avere chiaramente un modo per capire dove iniziano e dove finiscono.



16.3.2 Ordinamento della traduzione

Non possiamo limitarci a fare una traduzione della frase che troviamo, è anche importante che si vada a fare un ordinamento della traduzione, da linguaggio a linguaggio cambia il modo in cui ordiniamo la frase, in particolare in linguaggi come ad esempio l'inglese, il francese o il tedesco (eccetto per le frasi subordinate) si utilizza l'idea di SVO (Soggetto, verbo, oggetto). In altri linguaggi come ad esempio il giapponese, il verbo va in fondo alla frase. Questo vuol dire che se traduciamo qualcosa dal giapponese all'inglese non possiamo semplicemente tradurre, bisognerà anche ordinare in modo differente la frase per fare in modo che rispetti l'ordinamento classico usato in inglese.

16.3.3 Segmentation Variation

Un'altra cosa che dobbiamo considerare è che ci sono alcuni linguaggi che non hanno i confini delle parole ben definiti, ad esempio il cinese e il giapponese. Alcuni linguaggi tendono ad avere delle frasi che sono abbastanza lunghe rispetto all'inglese ad esempio.

16.3.4 Inferential Load

Un'altra differenza tra i linguaggi riguarda il modo in cui specifichiamo i vari attori all'interno della frase:

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

- I linguaggi detti *cold* sono linguaggi che non richiedono di esprimere sempre chi sono gli attori all'interno della frase, quindi possiamo in alcuni casi non specificare il soggetto o il complemento oggetto. In questo caso è il lettore che deve fare uno sforzo per comprendere il significato della frase.
- Ci sono poi linguaggi detti "hot" che invece sono più esplicativi e quindi in questo caso dobbiamo sempre andare ad indicare chi ha fatto cosa. Un esempio è l'inglese che richiede sempre la presenza del soggetto all'interno della frase.

16.3.5 Lexical Divergences

Ci sono poi le differenze lessicali:

- In alcuni linguaggi vengono utilizzate più parole per indicare quello che in altri linguaggi indichiamo con una singola parola. Ad esempio in italiano diciamo Informatica e in inglese diciamo Computer Science.
- Ci sono anche differenze in termini di POS, proprio a livello di quale costrutto grammaticale è previsto dopo un certo tipo di parola.
- Ci sono poi dei vincoli grammaticali che sono validi solamente in alcuni linguaggi, ad esempio in inglese abbiamo i pronomi al maschile e al femminile mentre in cinese non c'è questa distinzione.
- Abbiamo anche dei vincoli semantici, ad esempio per tradurre fratello abbiamo una singola parola in inglese mentre abbiamo una distinzione tra fratello più piccolo e fratello più grande in cinese.
- Ci sono alcune parole che possono avere vari significati possibili.

- In alcuni linguaggi potrebbero non essere presenti delle parole che invece vengono utilizzate in altri, ad esempio non c'è un corrispondente per la parola *privacy* in giapponese.
- Ci sono differenze anche per il modo in cui viene utilizzato un verbo, ad esempio potremmo utilizzare un normale verbo al presente in inglese mentre poi andando ad eseguire la traduzione in spagnolo abbiamo che quel verbo viene tradotto con il gerundio.

16.4 Approcci classici per la Machine Translation

Abbiamo tre differenti possibili approcci per la Machine Translation:

- Diretto: con questo approccio di traduzione diretta abbiamo un input e procediamo parola per parola eseguendo la traduzione nel linguaggio di destinazione. La conoscenza in questo caso viene fornita sotto forma di un grande dizionario bilingua. Non vengono utilizzate delle strutture intermedie in questo sistema ad eccezione della morfologia che sfruttiamo per capire la forma delle varie parole (dobbiamo capire se la parola è al dativo, genitivo...). Dopo aver completato la traduzione parola per parola andiamo a fare un ordinamento della frase. Vediamo un esempio, qua ad esempio lo studio della morfologia ci indica che abbiamo un verbo al passato, alla fine otteniamo la frase tradotta e riordinata.

Input:	Mary didn't slap the green witch
After 1: Morphology	Mary DO-PAST not slap the green witch
After 2: Lexical Transfer	Maria PAST no dar una bofetada a la verde bruja
After 3: Local reordering	Maria no dar PAST una bofetada a la bruja verde
After 4: Morphology	Maria no dió una bofetada a la bruja verde

L'intuizione che ha guidato la nascita di questo metodo è che noi quando traduciamo andiamo a trasformare in modo incrementale

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro *Speech and Language Processing* e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

la frase scritta nel linguaggio di origine nella frase scritta nel linguaggio di destinazione, questo approccio non viene chiaramente più utilizzato ma la sua intuizione di base rimane comunque utilizzata.



Questo metodo diretto ha chiaramente dei problemi, quando dobbiamo fare l'ordinamento delle frasi è difficile riuscire ad ordinare delle frasi in cui ci sono da fare ordinamenti a lunga distanza perché non abbiamo alcuna conoscenza riguardo alle strutture grammaticali nel linguaggio di origine e di destinazione. Il metodo è anche poco affidabile e non troppo potente, il numero di regole aumenta a mano a mano che andiamo avanti ed è richiesto molto contesto. I pro di questo metodo è che è veloce perché facciamo semplicemente una scansione della frase, è semplice da utilizzare ed il costo è basso.

- Transfer: sappiamo che i linguaggi si differenziano da un punto di vista strutturale, una strategia per la MT consiste nell'utilizzare la traduzione per ridurre queste differenze alterando la struttura dell'input in modo che sia simile alla struttura del linguaggio di destinazione. Questo processo può essere svolto applicando la contrastive knowledge ovvero lo svolgiamo quando conosciamo le differenze tra i due linguaggi. I sistemi che utilizzano questo sistema sono chiamati transfer model. Questo transfer model si divide in due fasi, una prima fase di parsing del linguaggio di origine che poi è seguita da una fase di generazione che crea la frase in output.

Più in generale per questo modello la machine translation prevede tre fasi:

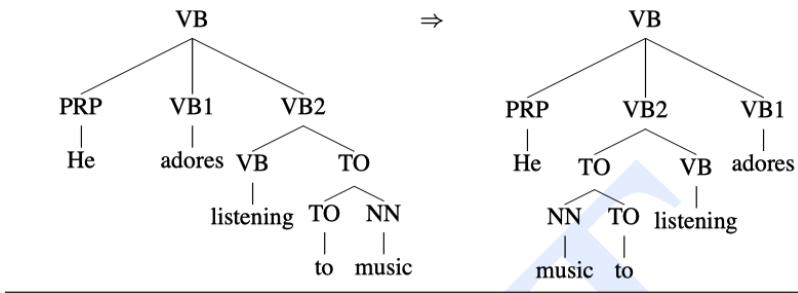
- Analisi

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

- Transfer: la fase di transfer riduce il gap tra l'output del parsing della frase di origine e l'input della fase di generazione.
- Generazione

Una volta che abbiamo finito il parsing della frase nel linguaggio di origine, abbiamo bisogno di regole per il "Syntactic Transfer" e il "Semantic Transfer":

- La Syntactic Transfer rule mi dice come deve essere modificato l'albero della frase nella lingua originale per essere il più simile possibile all'albero del parsing della frase nel linguaggio destinazione. Queste trasformazioni eseguono un mapping da una struttura ad albero all'altra. Un esempio è la trasformazione che possiamo fare dalla organizzazione SVO dell'inglese alla SOV del giapponese:



- La semantic transfer rule: consiste nell'eseguire una "semantic role labeling", anche in questo caso partiamo con la situazione della lingua di origine e cerchiamo di fare una traduzione nella lingua di destinazione.

English to Spanish:		
1.	$NP \rightarrow Adjective_1 Noun_2$	$\Rightarrow NP \rightarrow Noun_2 Adjective_1$
Chinese to English:		
2.	$VP \rightarrow PP[+Goal] V$	$\Rightarrow VP \rightarrow V PP[+Goal]$
English to Japanese:		
3.	$VP \rightarrow V NP$	$\Rightarrow VP \rightarrow NP V$
4.	$PP \rightarrow P NP$	$\Rightarrow PP \rightarrow NP P$
5.	$NP \rightarrow NP_1 Rel. Clause_2$	$\Rightarrow NP \rightarrow Rel. Clause_2 NP_1$

L'idea non è male e per un certo periodo è stata anche utilizzata.

Oltre alle trasformazioni sintattiche, un sistema transfer based deve avere anche delle transfer rule lessicali. Nelle transfer rule lessicali si utilizza un dizionario bilingua come nel caso della direct MT. Il dizionario in se per se può essere utilizzato per gestire problemi di ambiguità lessicale. Ad esempio la parola home inglese ha varie possibili traduzioni in tedesco a seconda del tipo di casa che stiamo indicando. In molti casi il lexical transfer è troppo complesso da gestire con un singolo dizionario, in questi casi possiamo fare una disambiguazione durante l'analisi del linguaggio originale utilizzando delle tecniche di disambiguazione del senso della frase.

Un'altra possibilità che abbiamo è quella di combinare gli approcci di direct e transfer. In questo caso andiamo a combinare dei dizionari bilingua molto grandi con dei parser e dei tagger. Un esempio di questa combinazione è il Sytran system che include 3 fasi:

- Analisi: in questa prima fase si effettua l'analisi morfologica, il POS tagging e un minimo di dependency parsing.
- Trasferimento: in questo caso viene effettuata una traduzione delle varie parole cercando di disambiguare il senso delle parole.
- Sintesi: si utilizza il dizionario bilingua per eseguire anche una traduzione "lessicale", qua eseguiamo anche un ordinamento delle frasi.

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

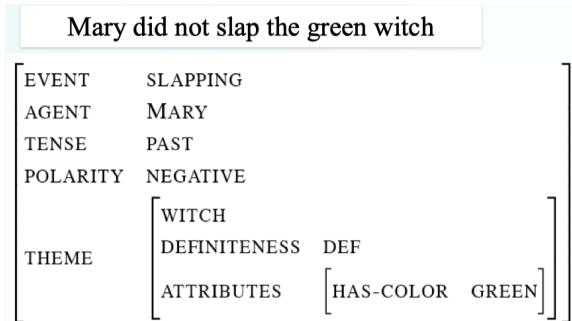
Anche l'approccio transfer ha dei problemi perchè per ogni coppia di linguaggio (lingua sorgente da cui tradurre e lingua di destinazione della traduzione) dovremmo creare un set di regole per la traduzione del parsing che è $O(N^2)$. Si tratta chiaramente di un sistema che è difficile da creare ma anche da mantenere.

- Interlingua: dato il problema che abbiamo con il modello di transfer (abbiamo bisogno di troppe regole), l'approccio interlingua consiste nel trovare una sorta di linguaggio universale, ogni linguaggio viene tradotto in un altro linguaggio intermedio e poi da questo viene tradotto in quello di destinazione. In questo modo invece di avere bisogno di $O(N^2)$ trasfer rule mi basterà solamente un numero proporzionale al numero di linguaggi che il sistema può gestire.

Questo schema che abbiamo creato suppone l'esistenza di un interlingua, l'idea è che nell'interlingua andiamo a rappresentare le frasi che rappresentano lo stesso concetto e hanno quindi lo stesso significato indipendentemente dal linguaggio in cui sono scritte.

In questo modello viene effettuata come prima cosa una analisi semantica approfondita della frase in input che viene tradotta nell'interlingua, poi dall'interlingua si genera il linguaggio in output Y. Cosa possiamo utilizzare come interlingua? Abbiamo varie possibilità, una è la decomposizione della frase in modo semantico in delle primitive, possiamo utilizzare delle tecniche di analisi semantica e avremo anche bisogno di eseguire una disambiguazione

Quello che otteniamo è una cosa di questo genere:



L'idea è che alcuni dei lavori di Machine Translation sono parte dei task di NLP che abbiamo visto fino ad ora, ad esempio possiamo utilizzare il NLP per disambiguare le varie parole.

Il metodo Interlingual ci permette di evitare il problema di avere N^2 regole e abbiamo inoltre delle regole che sono semplici da scrivere. Il conto è che la semantica è difficile da comprendere e che potremmo avere perdita di informazioni interessanti. Un altro problema che possiamo avere consiste nel fatto che nella traduzione di alcuni linguaggi avremo bisogno di integrare dei concetti che magari nella lingua di origine non sono presenti, ad esempio per tradurre dal giapponese al cinese avremo bisogno di concetti come fratello minore e fratello maggiore che normalmente non sono presenti. Allo stesso tempo utilizzando le stesse regole per tradurre ad esempio dal tedesco all'inglese, dovremmo utilizzare vari concetti che non hanno alcun senso di esistere e che rendono la disambiguation più complessa.

Chapter 17

Lesson 16 - Statistical Machine Translation

I tre sistemi classici che abbiamo visto fino ad ora ci forniscono una risposta al problema di come affrontare la traduzione e ci forniscono quindi una rappresentazione ma non si concentrano poi sul risultato.

Guardando il problema dal punto di vista del risultato ci sono varie considerazioni che possiamo fare. È praticamente impossibile, partendo da una frase in un linguaggio, trovare una traduzione letterale e che voglia significare la stessa identica cosa in un linguaggio differente. Ad esempio ci sono dei concetti che esistono solamente in alcune lingue mentre non sono esistenti in altre e questo ci porta ad avere quindi o delle traduzioni che sono molto lunghe e cercano di spiegare nel dettaglio la frase di origine dando quindi un'idea del concetto non esistente nel linguaggio target oppure ci portano a delle traduzioni che nel linguaggio di destinazione non sono chiare. Quindi, una traduzione che sia allo stesso tempo naturale e fedele (Faithful) all'originale è praticamente impossibile, quello che si fa in pratica è produrre una traduzione che sia tollerabile da entrambi i punti di vista naturalezza e fedeltà.

Partendo da quest'ultima affermazione possiamo iniziare a parlare di

Statistical Machine Translation, la nostra traduzione può essere model-lata come il prodotto di un output che massimizza allo stesso tempo la Faithfulness e la Fluency. L'obiettivo è di combinare queste due metriche per cercare di ottenere la migliore traduzione possibile.

- Faithfulness: questa metrica mi permette di giudicare quanto la traduzione che abbiamo ottenuto è simile alla frase originale. Oltre a questo ci permette anche di misurare se le conclusioni a cui giunge l'utente che legge la frase tradotta sono le stesse a cui giunge leggendo la frase originale.
- Fluency: questa metrica misura quanto è "grezza" la traduzione e quanto la frase che otteniamo è naturale nel linguaggio comune.

17.1 Faithfulness e Fluency

In modo più formale possiamo rappresentare queste due metriche in una singola formula che vorremmo massimizzare:

$$T^* = \operatorname{argmax}_T fluency(T)Faithfulness(T, S)$$

la formula ci dice che noi vorremmo ottenere il più alto valore della T^* andando quindi a massimizzare il prodotto delle due metriche. Nell'argmax abbiamo la T come pedice perchè dobbiamo considerare il fatto che vanno considerate tutte le possibili traduzioni T della frase S originale. Una volta che abbiamo calcolato il prodotto di fluency e Faithfulness di tutte le varie frasi T rispetto alla frase originale S allora possiamo scegliere la migliore T.

Questo risultato è stato raggiunto dall'IBM e inizialmente era stato pensato per il task di speech recognition. Questo dimostra che gli stessi risultati ottenuti nel campo della speech recognition possono poi essere utilizzati anche in altri campi, come ad esempio la machine translation.

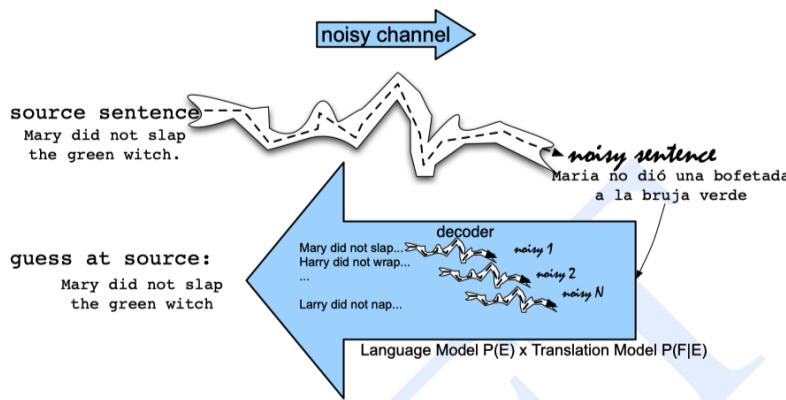
Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

17.1.1 Formulazione

Abbiamo detto che la formula per trovare la migliore T è la seguente:

$$T^* = \operatorname{argmax}_T \text{fluency}(T) \text{Faithfulness}(T, S)$$

questa formula è praticamente la Bayes Rule, dobbiamo considerare infatti le due probabilità e massimizzarle. Questa trasformazione della formulazione la otteniamo se pensiamo alla similitudine con il noisy channel model. Il noisy channel model è un tool importante nel campo della speech recognition e nel campo della machine translation. L'intuizione del noisy channel model è di trattare le parole che sono state scritte male come parole correttamente scritte che però sono state distorte perché sono passate attraverso un canale di comunicazione che introduce il rumore (in termini di sostituzione di lettere).



Questo rumore mi rende più complessa la comprensione del testo. Il noisy channel model è un caso speciale della Bayesian Inference, abbiamo una osservazione O e il nostro obiettivo è quello di trovare la parola w che ha generato l'osservazione. Quindi qua il nostro obiettivo è trovare la w^* tale che $w^* = \operatorname{argmax}_w P(w|O)$.

In questi esempi consideriamo E come la lingua inglese e F come la lingua di partenza della frase che vogliamo tradurre. Il nostro obiettivo è eseguire una traduzione da F ad E.

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

In una visione probabilistica la migliore frase E che possiamo produrre è quella tale per cui la $P(E|F)$ è più alta. Possiamo quindi andare ad utilizzare la Bayes Rule per calcolare:

$$E = \operatorname{argmax}_E P(E|F) = \operatorname{argmax}_E \frac{P(F|E)P(E)}{P(F)}$$

dato che il denominatore non contiene la E che stiamo cercando di massimizzare allora vuol dire che possiamo riscrivere la nostra formula come:

$$E^* = \operatorname{argmax}_E P(F|E)P(E)$$

Stiamo in pratica cercando la migliore traduzione inglese E^* . L'equazione risultante mi indica che abbiamo bisogno di tre componenti all'interno di questa formula:

- Per la $P(F|E)$, che rappresenta la Faithfulness mi serve un translation model.
- Per la $P(E)$, che rappresenta la Fluency mi serve un language model. Per quanto riguarda il language model in Statistical MT utilizziamo gli stessi language model basati su N-gram che vengono utilizzati anche nello speech recognition e in altre applicazioni.
- Ho poi necessità di un decoder che, data la F, produce la E più probabile.

17.1.2 Fluency $P(E)$

Con fluency $P(T)$ intendiamo la probabilità di una certa frase T, supponiamo di avere due frasi, se vogliamo trovare quella con una Fluency maggiore vuol dire che stiamo cercando la frase che suona meglio e che quindi ha un senso maggiore.

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

Come possiamo fare a capire quale delle due frasi suona meglio? Dovremmo essere in grado di calcolare questa proprietà per entrambe le frasi. Questo lo possiamo fare utilizzando un language model che ci fornisce la probabilità di una frase e ci permette di confrontare le probabilità di due frasi per capire quale delle due è migliori.

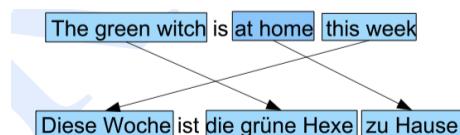
Dato il language model consideriamo ogni coppia di parole che sono presenti all'interno della frase e calcoliamo la probabilità che una certa parola compaia dopo un'altra.

Esistono anche metodi più complessi per raggiungere lo stesso scopo, questo metodo ha una caratteristica positiva perché ha una conoscenza di una singola lingua.

17.1.3 Faithfulness $P(F|E)$: Phrase-Based Translation Model

L'obiettivo della Faithfulness è calcolare la probabilità $P(F|E)$ che mi indica quanto è probabile che la frase E in inglese mi generi la frase F nella lingua originale.

Inizialmente l'idea era quella di andare a vedere parola per parola la traduzione e in questo modo si andava poi a decidere una probabilità per quella traduzione, con la statistical MT invece le cose sono differenti. Non ci basiamo semplicemente sulla traduzione delle singole parole ma utilizziamo una Machine Translation che considera la traduzione delle frasi, quindi non calcoliamo la probabilità di traduzione delle singole parole ma calcoliamo la probabilità di tradurre intere frasi.



Tutto questo sistema si basa su tre step fondamentali:

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

- Prima di tutto dobbiamo creare i gruppi di parole, quindi prendiamo la nostra frase completa E e andiamo a creare i vari sottogruppi e_1, \dots, e_i .
- Poi traduciamo ognuna di queste frasi che abbiamo creato nella lingua di destinazione F.
- Ognuna di queste frasi può essere optionalmente riordinata.

La Phrase-Based Translation si basa su due probabilità:

- Translation probability $\phi(f_i|e_i)$, questa indica la probabilità di generare la frase f_i partendo dalla frase inglese e_i .
- Distortion probability $d(i)$: Il riordinamento della frase tradotta viene effettuato sulla base della distortion probability. Con distortion ci riferiamo ad una parola che ha una posizione differente nella frase tradotta nel linguaggio F rispetto alla posizione che aveva nella frase nel linguaggio E iniziale. Quindi si tratta in pratica di una misura di distanza. Nel calcolo della distortion probability abbiamo due parametri che sono a_i e b_{i-1} dove la a_i indica la posizione della frase nella lingua F generata dalla frase e_i nella lingua E mentre la b_{i-1} indica la posizione della frase nella lingua F generata dalla frase e_{i-1} nella lingua E.

In pratica alla fine quello che otteniamo è la seguente probabilità:

$$P(F|E) = \prod_{i=1}^I \phi(f_i, e_i) d(a_i - b_{i-1})$$

Esempio

Supponiamo di voler tradurre "Mary did not slap the green witch". Ci troviamo nella seguente situazione:

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

Position	1	2	3	4	5	6
English	Mary	did not	slap	the	green	witch
Spanish	Maria	no	dio una bofetada a	la	bruja	verde
$a_i - b_{i-1}$	1	1	1	1	2	-1

- Dividiamo la frase in 6 (sotto)frasi.
- Cominciamo a tradurre ognuna di queste frasi e per ognuna delle sottofrasi che traduciamo calcoliamo sia translation probability sia la distortion probability.
- In particolare possiamo vedere che le prime 4 frasi si trovano in una posizione che va da 1 a 4 e poi nella traduzione la posizione viene mantenuta. Quando andiamo a calcolare la $a_i - b_{i-1}$ quindi abbiamo che la distanza tra la traduzione attuale e la precedente è sempre 1 e quindi la distortion per queste 4 frasi viene calcolata come $c\alpha^1$.
- Dalla quinta parola in poi le cose cambiano perchè abbiamo una inversione, la parola che originariamente era in posizione 5 finisce in posizione 6 e la 6 finisce in posizione 5. Ora per quanto riguarda la posizione 5 abbiamo che la posizione a_i della frase che creiamo traducendo la e_i è 6 mentre la posizione b_{i-1} che abbiamo dopo la traduzione della e_{i-1} è 4, quindi la $a_i - b_{i-1}$ in questo caso è uguale a 2. Allo stesso modo abbiamo che la frase in posizione 6 va in posizione 5 mentre la frase che abbiamo tradotto prima è andata in posizione 6, quindi abbiamo $a_i = 5$ e $b_{i-1} = 6$, quindi complessivamente abbiamo $a_i - b_{i-1} = 5 - 6 = -1$.

Il calcolo complessivamente diventa:

$$p(F|E) = \varphi(\text{Maria}, \text{Mary}) \alpha^1 \varphi(\text{no}, \text{did not}) \alpha^1 \varphi(\text{slap}, \text{dio una bofetada a}) \alpha^1 \\ \varphi(\text{la}, \text{the}) \alpha^1 \varphi(\text{verde}, \text{green}) \alpha^2 \varphi(\text{bruja}, \text{witch}) \alpha^{-1}$$

17.1.4 Osservazioni su Fluency e Faithfulness

Ci sono alcune cose da considerare riguardo a Fluency e Faithfulness:

- La Faithfulness quando calcola $P(S|T)$ va ad un utilizzare un modello di tipo "bag of word" in cui consideriamo semplicemente la traduzione delle singole parole da una lingua all'altra. L'ordine delle parole in questo caso o comunque il senso della frase non è importante. Questo è strano perché abbiamo detto che l'ordine in una frase è importante, in realtà possiamo delegare il controllo dell'ordine all'interno della frase ad un language model, quindi alla Fluency $P(T)$.
- La Fluency, basandosi sul language model dovrebbe essere in grado di eseguire una bag generation ovvero di mettere in ordine le varie parole all'interno della frase.

Come abbiamo detto l'obiettivo finale è avere una traduzione che sia sia Faithful sia Fluent, per massimizzare entrambe le metriche abbiamo quindi formulato il problema di massimizzazione come:

$$T_{best} = \operatorname{argmax}_T \text{Faithfulness}(T, S) \text{Fluency}(T) = \operatorname{argmax}_T P(F|E)P(E)$$

all'interno della formula come detto il primo componente è il translation Model e il secondo è il language model, abbiamo bisogno di un algoritmo di decoding che ci permetta di trovare la migliore traduzione.

- Abbiamo detto che nella formula dobbiamo utilizzare la $P(E)$, data una frase assegnamo la probabilità a quella frase, lo facciamo con un language model. Se abbiamo una frase che ha senso compiuto in inglese allora assegnamo una probabilità alta, altrimenti se la frase è scritta a caso assegnamo una bassa probabilità. Il language

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

model che viene utilizzato per questo calcolo della probabilità è un language model standard creato con gli N-Gram, dobbiamo fare il training su un corpus molto grande per creare un N-gram model che ci permetta di ottenere una buona stima per le frasi che stiamo considerando.

- Il secondo componente della formula è il Translation Model, con questo secondo componente vogliamo calcolare $P(F|E)$ e vogliamo trovare la probabilità tale che la probabilità sia alta se la traduzione da una lingua all'altra è buona. Quindi data la coppia di stringhe $< f, e >$ assegnamo la probabilità $P(f|e)$.
- Algoritmo di decoding: ci dice quale è la migliore frase che massimizza la probabilità. Dato un language model e un translation model.

17.1.5 Training dei parametri del Phrase Based translation model

I parametri per cui va fatto il training sono i set delle probabilità di traduzione che mi permettono di generare le probabilità $\phi(f_i, e_i)$, inoltre abbiamo anche il parametro *alpha* che viene utilizzato nel calcolo della $P(F|E)$.

Il primo set di parametri, quello che mi permette di generare la probabilità che una certa frase venga tradotta con un'altra frase suppose l'esistenza di un mapping tra le frasi nella lingua E e le frasi nella lingua F. Questo mapping viene chiamato Phrase Alignment.

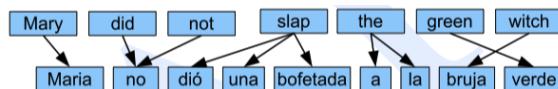
Per calcolare le probabilità di ottenere la frase F dalla frase E, che indichiamo con $\phi(f_i, e_i)$, dobbiamo avere a disposizione un corpus in cui abbiamo il mapping delle frasi tradotte dalla lingua E alla lingua F. Quello che si fa poi è contare, consideriamo per ogni coppia di possibili traduzioni il numero di volte che quel mapping compare all'interno del

corpus normalizzando ovvero contando il numero delle volte che quella stessa frase inglese compare insieme a qualsiasi altra frase f:

$$\phi(\bar{f}, \bar{e}) = \frac{\text{count}(\bar{f}, \bar{e})}{\sum_{\bar{f}} \text{count}(\bar{f}, \bar{e})}$$

Possiamo memorizzare queste probabilità all'interno di una phrase translation table insieme alla corrispondente coppia di frasi a cui la probabilità è associata.

Il problema è che di base non abbiamo un allineamento delle frasi ma possiamo comunque estrarre un allineamento delle frasi partendo dall'allineamento delle parole.



Quella della foto è la rappresentazione grafica degli allineamenti delle parole tra l'inglese e lo spagnolo. Dopo che abbiamo ottenuto l'allineamento delle parole possiamo a questo punto estrarre il phrase alignment.

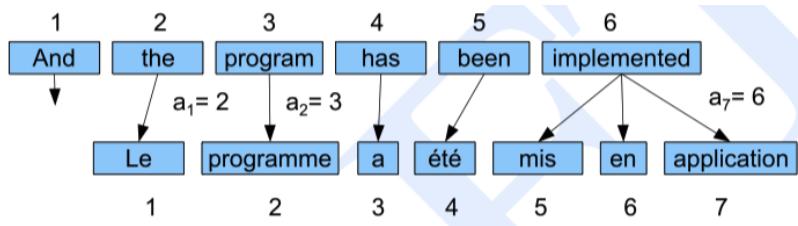
17.2 Allineamento in Machine Translation

Abbiamo già definito il concetto di allineamento delle parole e abbiamo anche visto un esempio di allineamento.

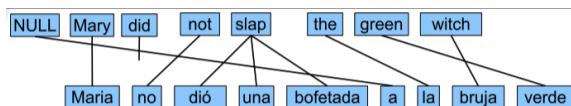
In teoria potremmo seguire qualsiasi regola per eseguire l'allineamento tra le parole di due linguaggi, i metodi che eseguiamo per l'allineamento però necessitano di alcuni vincoli per funzionare:

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

- Ad ogni parola della lingua destinazione corrisponde una sola parola della lingua sorgente.
- Una parola della lingua sorgente può essere collegata con varie parole della lingua destinazione.

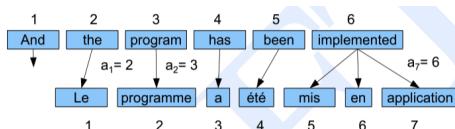


- Una parola della lingua destinazione potrebbe non avere una corrispondenza con la frase originale, in questo caso la colleghiamo ad una parola "finta" che aggiungiamo e che chiamiamo "spurious word".



- È possibile anche ottenere degli allineamenti più complessi ma per i modelli che vedremo (IBM Model 1 ad esempio) non è necessario.

Una volta che abbiamo trovato l'allineamento delle parole, possiamo rappresentare l'allineamento che abbiamo trovato con la sequenza di numeri corrispondenti alla posizione della parola nella frase originale. Quindi ad esempio in questa situazione:



avremmo 2,3,4,5,6,6,6. Questo è un allineamento molto probabile mentre invece ad esempio 3,3,3,3,3,3,3 non è probabile. Notare che dato che abbiamo 7 posizioni nella frase di destinazione e 6 parole nella frase di partenza vuol dire che i possibili assegnamenti sono 6^7 e quindi può diventare computazionalmente difficile andare a considerare tutti i possibili assegnamenti.

17.2.1 IBM Model 1

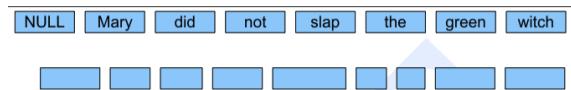
L'IBM Model 1 è un metodo statistico per l'allineamento che viene utilizzato per trovare il miglior allineamento possibile per la coppia F, E . È anche possibile utilizzare questo allineamento come translation model $P(F, E)$, in questo caso la relazione tra allineamento e traduzione può essere espressa come:

$$P(F|E) = \sum_A P(F, A|E)$$

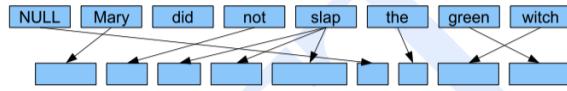
qui nella formula consideriamo tutti i possibili allineamenti di A e poi calcoliamo la probabilità che la frase F sia la traduzione della frase E. Per eseguire il calcolo sopra è necessario avere a disposizione tutti i possibili allineamenti delle due frasi.

L'IBM Model 1, è stato presentato nel 1993 e suppone l'esistenza di una frase inglese $E = e_1, \dots, e_I$ di lunghezza I che dobbiamo tradurre in una frase spagnola. Come funziona questo modello:

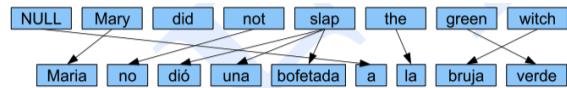
- La prima cosa che dobbiamo fare è scegliere una lunghezza J per la frase spagnola $F = f_1, \dots, f_J$



- Ora scegliamo un allineamento $A = a_1, \dots, a_J$ tra la frase E e la frase F.



- Per ogni posizione j nella frase spagnola scegliamo una parola spagnola che traduce la corrispondente parola inglese iniziale.



Una volta fatti questi passaggi dovremmo assegnare la probabilità $P(F|E)$ di generare la frase nella lingua F data la frase nella lingua E. Indichiamo con e_{a_j} la parola inglese che è allineata con la parola spagnola f_j e indichiamo con $t(f_x, e_y)$ la probabilità che la parola e_y sia tradotta con la parola f_x . Ora supponiamo di trovarci nello step 3 e di avere a disposizione la traduzione nella lingua F dalla lingua E, allora la probabilità della frase tradotta sarà:

$$P(F|E, A) = \prod_{j=1}^J t(f_j | e_{a_j})$$

vuol dire che consideriamo tutte le possibili parole della frase di destinazione e calcoliamo il prodotto delle varie probabilità di ottenere quella parola data la parola inglese corrispondente.

Il modello IBM 1 esegue una semplificazione molto forte perché dice che ogni allineamento ha la stessa probabilità di essere ottenuto, come abbiamo già visto da una sequenza di lunghezza I e una sequenza di lunghezza J abbiamo I^J possibili allineamenti. Il modello fa anche un'altra assunzione ovvero mi dice che la probabilità di scegliere una certa J per la lunghezza della frase di destinazione è una costante piccola

ϵ . Combinando queste due assunzioni riusciamo ad ottenere la probabilità di avere un certo allineamento data la parola inglese:

$$P(A|E) = \frac{\epsilon}{(I+1)^J}$$

Possiamo combinare tutte le varie probabilità calcolate fino ad ora, ricordiamo che noi conosciamo:

$$\begin{aligned} P(F|E, A) &= \prod_{j=1}^J t(f_j|e_{a_j}) \\ P(A|E) &= \frac{\epsilon}{(I+1)^J} \end{aligned}$$

Partendo da $P(F|E, A)$ possiamo usare la Bayes Rule per ottenere:

$$P(F|E, A) = \frac{P(F, E, A)}{P(E, A)} = \frac{P(F, A|E)P(E)}{P(A|E)P(E)} = \frac{P(F, A|E)}{P(A|E)}$$

Riscrivendo otteniamo:

$$P(F, A|E) = P(F|E, A) * P(A|E) = \prod_{j=1}^J t(f_j|e_{a_j}) * \frac{\epsilon}{(I+1)^J}$$

La probabilità $P(F, A|E)$ che abbiamo generato è la probabilità di generare una frase F in spagnolo partendo da una frase in inglese utilizzando un particolare allineamento A.

Dato che il nostro obiettivo è calcolare la probabilità di generare la frase F in spagnolo allora dobbiamo sommare tutte le probabilità che otteniamo al variare dell'allineamento:

$$P(F|E) = \sum_A P(F, A|E) = \sum_A \frac{\epsilon}{(I+1)^J} * \prod_{j=1}^J t(f_j|e_{a_j})$$

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

Questo è il modo in cui riusciamo a calcolare la probabilità di ottenere la frase F partendo dalla frase E, qua viene assegnata una probabilità ad ognuna delle possibili frasi spagnole.

Per ottenere poi il miglior allineamento possibile tra due frasi F ed E dobbiamo trovare un modo di eseguire il decoding utilizzando questo modello probabilistico. Sappiamo che il miglior allineamento per ognuna delle parole è indipendente dal miglior allineamento che abbiamo per le parole intorno. Quindi per trovare il miglior allineamento abbiamo:

$$A^* = \operatorname{argmax}_A P(F, A|E) = \operatorname{argmax}_A \frac{\epsilon}{(I+1)^J} * \prod_{j=1}^J t(f_j | e_{a_j})$$

Nella massimizzazione possiamo rimuovere la prima parte perché la A non è presente mentre invece rimane solamente la seconda:

$$= \operatorname{argmax}_{a_j} t(f_j | e_{a_j})$$

Questo vuol dire che per ogni coppia di parole inglese-spagnolo scegliamo la a_j che mi massimizza la probabilità in modo indipendente. Massimizzando ognuno di questi termini si massimizza anche il prodotto dei termini ovvero:

$$\operatorname{argmax}_A \prod_{j=1}^J t(f_j, e_{a_j})$$

17.3 Training dei modelli di allineamento

Tutti i modelli di traduzione statistica sono allenati utilizzando i seguenti step:

- La prima cosa che si fa è trovare un grande corpus parallelo. Con corpus parallelo intendiamo un corpus che è scritto in due lingue

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

differenti, ad esempio i verbali del parlamento canadese sono scritti parallelamente sia in Inglese che in Francese. Questi corpus paralleli contengono centinaia di migliaia di frasi tradotte e di parole tradotte. In teoria potremmo crearcisi dei corpus paralleli anche partendo dalle traduzioni di testi letterari ma spesso non si usa nel campo della machine translation perché è difficile ottenere legalmente la possibilità di utilizzare questi corpus e soprattutto perché spesso tradurre questo tipo di documenti di letteratura è più complesso.

- Il secondo step che viene effettuato è il sentence segmentation o sentence alignment. L'idea di questo algoritmo è che si prendono i due corpus e in parallelo si vanno ad allineare le frasi che sono l'una la traduzione dell'altra. In particolare un algoritmo molto semplice per raggiungere questo scopo consiste nel considerare la lunghezza delle frasi, se due frasi hanno più o meno la stessa lunghezza allora è probabile che siano l'una la traduzione dell'altra. Questa intuizione la possiamo implementare con un algoritmo di programmazione dinamica, poi esistono anche degli altri algoritmi più complessi che fanno uso anche delle informazioni riguardo all'allineamento delle singole parole. Questi algoritmi vengono eseguiti sui parallel corpus prima di eseguire il training dei modelli di Machine Translation, le frasi che non matchano non vengono considerate. Una volta che abbiamo finito questa prima fase abbiamo in output un insieme di coppie (F_s, E_s) e per ognuna di queste coppie l'obiettivo è eseguire il learning degli allineamenti interni alla frase e la probabilità di questi allineamenti.
- Il terzo step consiste nel trovare gli allineamenti delle parole partendo dall'allineamento delle frasi e da un corpus bilingua. Se noi avessimo le frasi già allineate con le parole all'interno delle frasi già perfettamente allineate sarebbe facile andare a calcolare

la probabilità di ogni singola coppia di parole. Il problema è che noi questo allineamento di parole non lo conosciamo, abbiamo solamente l'allineamento delle frasi. Supponendo di avere a disposizione i parametri t del modello di traduzione (che mi permettono di capire la probabilità con cui traduco una parola con un'altra) possiamo calcolare la probabilità di ogni allineamento di parole, conosciamo infatti la seguente probabilità:

$$P(A|E, F) = \frac{P(A, F|E)}{\sum_A P(A, F|E)}$$

Questa probabilità la riusciamo a calcolare solamente se conosciamo i parametri t del modello, il problema è che noi questi problemi non li conosciamo. Allora possiamo utilizzare l'algoritmo EM, invece di stimare le probabilità di t partendo da un allineamento perfetto che non conosciamo possiamo stimare la t da ogni possibile allineamento e poi possiamo combinare queste stime pesandole con la probabilità di ogni allineamento. Ad esempio se abbiamo due possibili allineamenti con due probabilità, stimeremo i parametri t dai due allineamenti e mixeremo i due allineamenti utilizzando i pesi delle due probabilità. Una volta che abbiamo trovato questi parametri possiamo stimarli nuovamente andando avanti con una fase di expectation e una fase di maximization. Vediamo un esempio di questo metodo: Vogliamo calcolare la probabilità

$$P(A, F|E) = m \prod_{j=1}^J t(f_j|e_{a_j})$$

Nella fase di expectation calcoliamo i parametri t contando e normalizzando, nella fase si maximization invece calcoliamo la maximum likelihood estimation della probabilità t .

17.3.1 Esempio di EM

Vediamo un esempio di EM per l'allineamento delle parole, abbiamo il seguente corpus di training:

green house
casa verde

the house
la casa

Inizialmente non abbiamo idea di come debbano essere allineate le parole e quindi utilizziamo una distribuzione di probabilità uniforme.

$t(\text{casa} \text{green}) = \frac{1}{3}$	$t(\text{verde} \text{green}) = \frac{1}{3}$	$t(\text{la} \text{green}) = \frac{1}{3}$
$t(\text{casa} \text{house}) = \frac{1}{3}$	$t(\text{verde} \text{house}) = \frac{1}{3}$	$t(\text{la} \text{house}) = \frac{1}{3}$
$t(\text{casa} \text{the}) = \frac{1}{3}$	$t(\text{verde} \text{the}) = \frac{1}{3}$	$t(\text{la} \text{the}) = \frac{1}{3}$

Ora inizia lo step Expectation, noi non sappiamo come devono essere allineate le parole quindi proviamo tutti i possibili allineamenti, per ognuno di questi possibili allineamenti, basandoci sulle probabilità in tabella calcoliamo poi la probabilità $P(a, f|e)$:

green casa	house verde	$P(a, f e) = t(\text{casa}, \text{green}) \times t(\text{verde}, \text{house}) = \frac{1}{3} \times \frac{1}{3} = \frac{1}{9}$
green casa	house verde	$P(a, f e) = t(\text{verde}, \text{green}) \times t(\text{casa}, \text{house}) = \frac{1}{3} \times \frac{1}{3} = \frac{1}{9}$
the la	house casa	$P(a, f e) = t(\text{la}, \text{the}) \times t(\text{casa}, \text{house}) = \frac{1}{3} \times \frac{1}{3} = \frac{1}{9}$
the la	house casa	$P(a, f e) = t(\text{casa}, \text{the}) \times t(\text{la}, \text{house}) = \frac{1}{3} \times \frac{1}{3} = \frac{1}{9}$

Dobbiamo poi normalizzare e quindi dividiamo i numeri che abbiamo ottenuto per la somma delle probabilità di avere quella parola associata con un'altra traduzione, questo mi porta ad avere la seguente situazione:

green casa	house verde	$P(a f, e) = \frac{1/9}{2/9} = \frac{1}{2}$
green casa	house verde	$P(a f, e) = \frac{1/9}{2/9} = \frac{1}{2}$
the la	house casa	$P(a f, e) = \frac{1/9}{2/9} = \frac{1}{2}$
the la	house casa	$P(a f, e) = \frac{1/9}{2/9} = \frac{1}{2}$

Questo mi porta alla conclusione dello step E, abbiamo ottenuto i nuovi valori per le probabilità:

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

$tcount(casa green) = \frac{1}{2}$	$tcount(verde green) = \frac{1}{2}$	$tcount(la green) = 0$	$total(green) = 1$
$tcount(casa house) = \frac{1}{2} + \frac{1}{2}$	$tcount(verde house) = \frac{1}{2}$	$tcount(la house) = \frac{1}{2}$	$total(house) = 2$
$tcount(casa the) = \frac{1}{2}$	$tcount(verde the) = 0$	$tcount(la the) = \frac{1}{2}$	$total(the) = 1$

Ora inizia lo step di Maximization, per ognuna delle righe della tabella sopra facciamo la somma, questa deve essere 1, quindi se la somma non è 1 dobbiamo fare la normalizzazione. Per fare la normalizzazione prendiamo il valore della somma e dividiamo tutti i valori di quella riga per il valore somma. In questo caso la prima e la terza riga sono sommate a 1 mentre per la seconda riga la somma è 2, quindi dobbiamo dividere tutti i risultati della seconda riga per 2.

Questo è quello che otteniamo:

$t(casa green) = \frac{1/2}{1} = \frac{1}{2}$	$t(verde green) = \frac{1/2}{1} = \frac{1}{2}$	$t(la green) = \frac{0}{1} = 0$
$t(casa house) = \frac{1}{2} = \frac{1}{2}$	$t(verde house) = \frac{1/2}{2} = \frac{1}{4}$	$t(la house) = \frac{1/2}{2} = \frac{1}{4}$
$t(casa the) = \frac{1/2}{1} = \frac{1}{2}$	$t(verde the) = \frac{0}{1} = 0$	$t(la the) = \frac{1/2}{1} = \frac{1}{2}$

Ora che abbiamo le nuove probabilità per gli allineamenti delle parole possiamo ricominciare con lo step E, quindi calcoliamo nuovamente la probabilità di ognuno degli allineamenti delle frasi e poi ricalcoliamo di nuovo le probabilità degli allineamenti delle parole. Notare che a mano a mano che andiamo avanti la probabilità degli allineamenti più sensati diventa sempre più alta.

Ripetiamo gli step di Expectation e di Maximization fino a che non arriviamo alla convergenza, quando arriviamo alla convergenza le coppie che sono più sensate avranno una probabilità più alta delle altre.

L'algoritmo che abbiamo visto fino ad ora richiede che per eseguire il learning delle probabilità vengano considerati tutti i possibili allineamenti, per una frase molto lunga enumerare tutti i possibili allineamenti può essere molto inefficiente.

Con i successivi Modelli di IBM (Model 2 e Model 3) si è cercato di risolvere questi problemi di difficoltà computazionale dovuta al conteggio di tutte le possibili combinazioni.

17.4 Phrase Based Translation Model

All'interno di un Phrase Based Translation Model abbiamo i seguenti componenti:

- Abbiamo il Phrase Translation Model $\phi(f|e)$.
- Abbiamo il reordering model $\Omega(f|e)$.
- Language Model $P_{LM}(e)$.

Dati questi tre modelli l'obiettivo è utilizzare la Bayes Rule per unirli tutti insieme e ottenere la traduzione con la probabilità più alta:

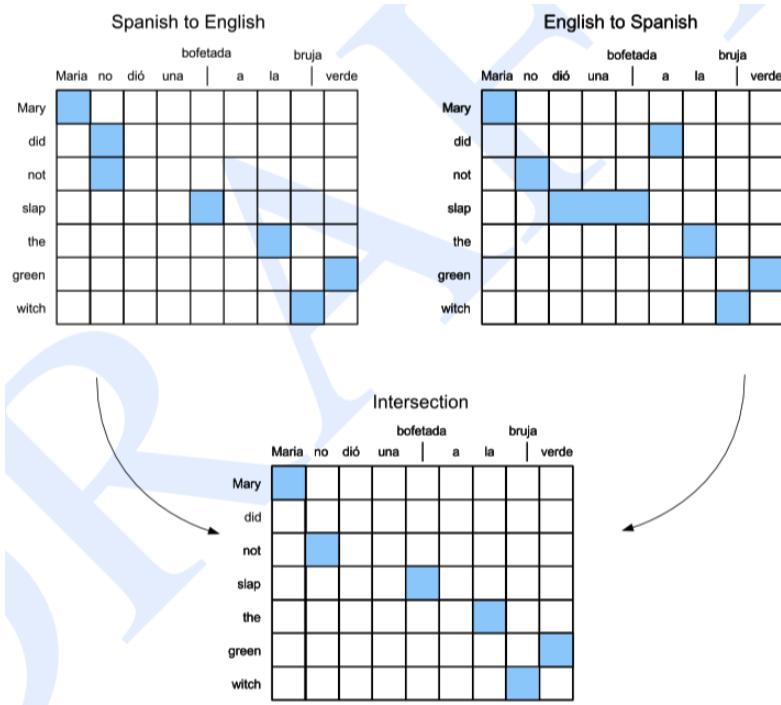
$$\text{argmax}_e P(e|f) = \text{argmax} \phi(f|e) \Omega(f|e) P_{LM}(e) s$$

I benefici del phrase based translation sono vari, possiamo utilizzare all'interno della traduzione un contesto locale ad esempio, poi più dati abbiamo a disposizione e più possiamo imparare.

17.4.1 Phrase Alignment

Quando eseguiamo la traduzione sappiamo che nella traduzione che viene prodotta abbiamo una parola della lingua originale che può essere mappata in più parole della lingua di destinazione. Sappiamo che le parole non vengono mappate uno ad uno nella traduzione. Quello che possiamo fare è estendere il modello per fare in modo di produrre un allineamento frase a frase per ogni coppia di frasi (F,E) tramite un metodo che viene chiamato simmetrizzazione. Viene fatto il training di due "aligners", uno ad esempio dall'inglese allo spagnolo e uno dallo spagnolo all'inglese. Poi facciamo l'allineamento della coppia di frasi (F,E) utilizzando entrambi gli aligners, in questo modo otteniamo due allineamenti. Per combinare i due allineamenti quello che si fa è andare a prendere l'intersezione che

conterrà solamente le zone in cui entrambi gli allineamenti sono "concordi".



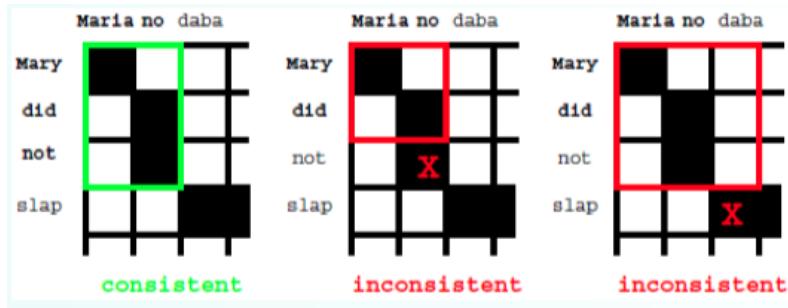
Quello creato dall'intersezione è un allineatore di parole molto preciso.

Oltre all'intersezione possiamo anche calcolare l'unione di questi due allineamenti, all'interno dell'unione ci saranno molte meno parole allineate. Possiamo poi creare un classificatore per selezionare le parole dall'unione in modo che possano essere riportate indietro e aggiunte in modo incrementale anche all'interno dell'intersezione.

	Maria	no	dió	una	bofetada	a	la	bruja	verde
Mary									
did									
not									
slap									
the									
green									
witch									

(Maria, Mary), (no, did not),
 (slap, dió una bofetada), (verde, green),
 (a la, the), (bruja, witch),
 (Maria no, Mary did not),
 (no dió una bofetada, did not slap),
 (dió una bofetada a la, slap the),
 (bruja verde, green witch),
 (a la bruja verde, the green witch),...

Quello nella immagine qua sopra è il risultato che possiamo ottenere, notare che possiamo avere un assegnamento molti ad uno in entrambe le direzioni. Una volta fatto tutto questo possiamo andare a raccogliere le frasi che sono consistenti con l'allineamento delle parole, questo vuol dire che le parole sono allineate solamente tra di loro e non con parole che sono esterne ovvero dato un certo assegnamento, nella stessa colonna non vogliamo altre parole che sono assegnate alla stessa parola che stiamo considerando e lo stesso vale per la riga. Da questo esempio si capisce abbastanza bene:



Una volta che abbiamo ottenuto l'assegnamento delle varie parole poi possiamo ottenere svariati assegnamenti anche per la frase, non è uno singolo l'assegnamento che possiamo avere.

Partendo dal corpus possiamo allineare le varie frasi, possiamo poi calcolare la MLE per stimare la probabilità che ognuna delle frasi venga tradotta nel modo in cui la stiamo traducendo. Quindi per ognuna delle

frasi andiamo a contare il numero di volte che quella frase viene tradotta in quel modo rispetto al numero complessivo di volte che la frase viene tradotta.

$$\phi(\bar{f}, \bar{e}) = \frac{\text{count}(\bar{f}, \bar{e})}{\sum_{\bar{f}} \text{count}(\bar{f}, \bar{e})}$$

Tutte queste frasi tradotte con la loro rispettiva probabilità di essere tradotte in quel modo vengono poi memorizzate all'interno del phrase translation table. Un esempio di phrase translation table è il seguente:

f	e	$\phi(f e)$	$\text{lex}(f e)$	$\phi(e f)$	$\text{lex}(e f)$	Alignments (F-E)
l'it definitivamente correct that , che conferma 0.01369865.5.6952e-07 1.4.68166e-17 0-0-4-1 1-2 73 1						
l'it depends sono solo 0.00107991.2.85796e-05 1.5.63319e-11 1-0-2-1 926 1						
l'it does not bode at all , che non lascia presagire nulla di 1.2.58282e-09 0.5.1.60281e-13 0-0-1-1 2-2 3-2 4-3 4-4 4-5 5-5 6-5 1 2						
l'it does not bode at all , che non lascia presagire nulla 1.2.20735e-08 0.5.1.60281e-13 0-0-1-1 2-2 3-2 4-3 4-4 4-5 5-5 6-5 1 2						
l'it does not , che non 0.00034448 0.000259527 1.1.6381e-08 0-0-1-1 2-2 3-2 2990 1						
l'it has become , che è diventata una 0.2 6.20953e-07 1.3.72893e-08 0-0-1-1 2-2 3-2 3-4-4 5 1						
l'it has become , che è diventata 0.0588235 2.67744e-06 1.7.12714e-08 0-0-1-1 2-2 3-3 17 1						
l'it has not been implemented yet e non è ancora stato applicato 1.3.69476e-07 1.1.233e-10 0-0-3-1 2-2 6-3 4-4 5-5 1 1						
l'it has not e non è 0.000823045 0.000251293 1.4.8038e-08 0-0-3-1 1-2 2-2 1215 1						
l'it has 5.86751e-05 0.755472 0.5.1.82129e-05 0-0 17043 2						
l'it has , che è 0.000127307 7.00774e-05 0.5.1.41342e-07 0-0-1-1 2-2 2-2 7855 2						
'access to cheap medicines di accedere a farmaci a basso prezzo 1.2.18888e-07 1.5.95807e-07 0-0-2-0-1-1 2-2 4-3 4-3-5 3-6 1 1						
'access to credit l'accesso al credito da parte 1.5.13669e-09 0.333333 2.9826e-05 0-0-1-0-1-2-2 3-3 1 3						
'access to credit l'accesso al credito da 1.2.01676e-06 0.333333 2.9826e-05 0-0-1-0-1-2-2 3-3 1 3						
'access to credit l'accesso al credito 0.030303 0.000396314 0.333333 2.9826e-05 0-0-1-0-1 2-2 3-3 33 3						
'access to satellite TV , canali censurati 0.166667 2.4734e-10 0.5.1.35061e-17 0-1 6 2						

All'interno abbiamo la frase originale e quella tradotta, abbiamo le varie probabilità relative a quella frase e abbiamo anche il modo in cui le parole all'interno delle due frasi sono allineate nella traduzione finale.

17.5 Decoding per il Phrase Based Statistical MT

Quello che abbiamo fatto fino ad ora sono gli allineamenti tra le frasi da tradurre e abbiamo una stima per ogni traduzione.

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

Lo step di traduzione è chiamato decoding e il compito di questo step è quindi quello di prendere una frase spagnola F e di produrre una frase inglese E cercando di massimizzare il seguente prodotto:

$$E = \operatorname{argmax} P(F|E)P(E)$$

dove $P(F|E)$ è il translation model e $P(E)$ è il language model. Trovare la frase che massimizza questo problema consiste nell'eseguire una ricerca, i decoders in questo campo si basano sulla best-first search ovvero una ricerca euristica e informata che selezionano un nodo nello spazio e poi esplorano i vicini basandosi su una funzione di valutazione $f(n)$. Questa tipologia di decoders si basa su una specifica categoria di ricerche best first che sono chiamate A^* .

In generale l'algoritmo, chiamato stack decoding, si occupa di mantenere una priority queue con una traduzione parziale delle ipotesi insieme agli score di queste varie ipotesi.

Per limitare la ricerca ed evitare di cercare attraverso lo spazio di tutte le possibili frasi inglesi, vogliamo solamente considerare le frasi che includono parole o frasi che sono possibili traduzioni della frase originale F. Per trovare le parole o le frasi che sono possibili traduzioni andiamo a guardare nel phrase translation table di cui abbiamo parlato prima.

Per ognuna delle traduzioni che sceglieremo dobbiamo associare una probabilità.

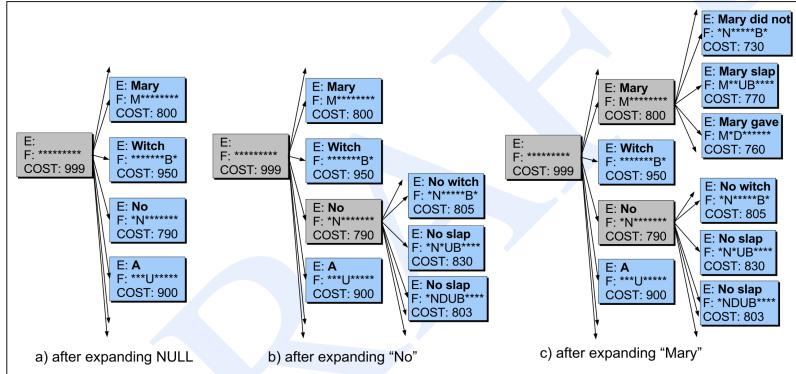
Maria	no	dió	una	bofetada	a	la	bruja	verde
Mary	not	give	a	slap	to	the	witch	green
	did not			a slap	to		green witch	
	no		slap		to the			
	did not give				to			
					the			
			slap			the witch		



Ora quello che si fa è andare all'interno di questo spazio delle ipotesi da sinistra a destra per scegliere la migliore traduzione. In particolare si

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

parte da un "null state" e poi prendiamo la prima parola della frase in spagnolo e espandiamo tutte le possibili ipotesi considerando per ognuna la probabilità della traduzione. Vediamo visivamente l'esempio:



Nell'esempio abbiamo che per prima cosa dal null state scegliamo la prima parola della nostra traduzione, ad ognuna di queste parole corrisponde un costo che dipende dalla probabilità di scegliere quella frase. Quello che si fa è mantenere alcuni nodi nel search space (sono quelli celesti) mentre facciamo diventare grigi quelli che abbiamo già espanso. La prima cosa che facciamo qua è espandere NO perché vediamo che ha il costo minore, poi però dobbiamo espandere il nodo successivo e vediamo che il costo minore ce l'ha Mary, quindi espandiamo quella parte dell'albero e poi andiamo avanti in quel ramo.

Il costo è la probabilità totale della frase di essere stata tradotta fino a quel punto, è il prodotto delle probabilità della traduzione, della distorsione e del language model, lo possiamo esprimere come:

$$cost(E, F) = \prod_{i \in S} \phi(f_i, e_i) d(a_i - b_{i-1}) P(E)$$

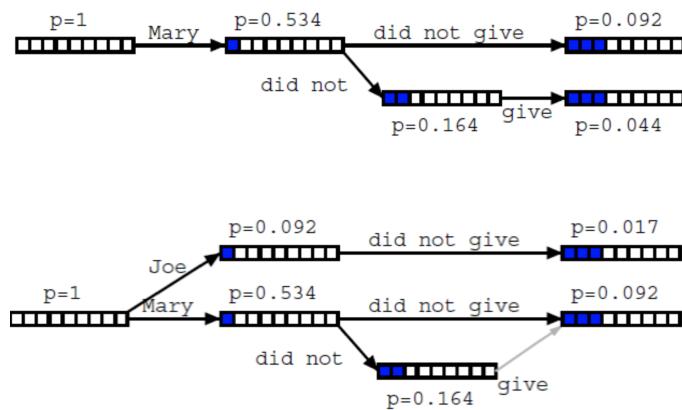
Se lo vogliamo vedere dal punto di vista delle probabilità espandiamo ogni volta i vari nodi e ogni volta moltiplichiamo la probabilità dei nodi e alla fine otterremo la probabilità della sequenza completa.

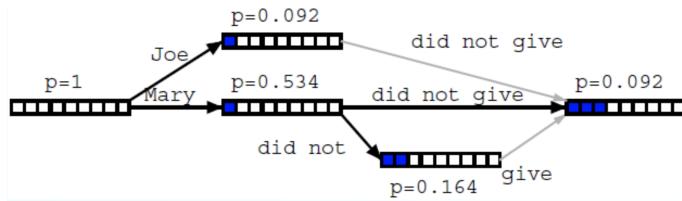
Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

Non sempre si ottiene la risposta corretta.

Il problema di questo metodo è che è esponenziale in profondità (il numero delle ipotesi è esponenziale rispetto alla lunghezza della frase che stiamo considerando). Il problema del decoding quindi è NP-Completo. Quindi dobbiamo trovare un modo per controllarlo perché altrimenti diventa impossibile eseguire una generazione di questo genere per trovare la traduzione migliore. Non possiamo considerare tutte le possibili frasi e dobbiamo quindi ridurre lo spazio in cui facciamo la ricerca.

Una possibilità che abbiamo per evitare di arrivare ad una esplosione esponenziale del numero nodi da considerare è la recombining hypotheses che viene utilizzata praticamente da ogni decoders. In Machine Translation quello che succede è che possiamo eseguire il merge di qualsiasi coppia di ipotesi che sono sufficientemente simili, ad esempio devono coprire le stesse parole di origine o avere la stessa lunghezza della traduzione. Non è necessario avere un match completo. Capita infatti che ad un certo punto facciamo una divisione dei nodi in due rami che però alla fine mi portano alla stessa traduzione, l'idea quindi è di combinare questi percorsi.





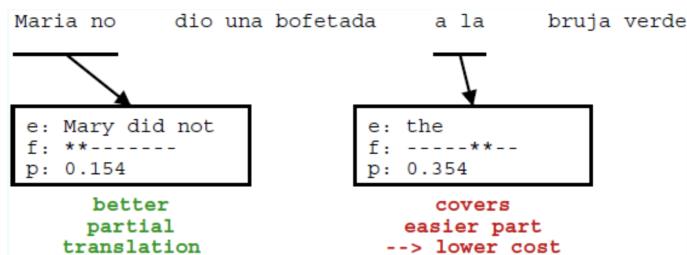
Questo mi permette di andare avanti nell'espansione andando a produrre alternative e combinandole poi in modo che alla fine si riduca il numero di nodi che devono essere espansi.

Eseguire solamente la recombining hypotheses non è sufficiente, quindi possiamo fare una ulteriore ottimizzazione. Possiamo diminuire il numero di stati da controllare con il beam-search pruning ovvero ad ogni iterazione andiamo a mantenere solamente gli stati più promettenti (ovvero quelli con i costi più bassi) eliminando quelli con i costi più alti. Quello che facciamo è creare una priority queue di dimensione fissata N, ad ogni iterazione prendiamo sempre i migliori N nodi e li mettiamo nella priority queue e poi espandiamo i nodi che stanno nella priority queue. Ad ogni iterazione poi prendiamo i migliori N e togliamo quelli che non sono più nei migliori N.

Nel caso della Machine Translation in realtà non manteniamo solamente un singolo stack con N posizioni ma manteniamo M stack in cui inseriamo N nodi più promettenti. La motivazione è che altrimenti non riusciremmo a fare una comparazione tra le ipotesi che traducono varie parole della frase originale tutte insieme. Quindi questi stack sono suddivisi in base al numero di parole che sono tradotte dalla lingua originale all'inglese. In questo modo quando espanderemo una ipotesi scegliendo la frase da tradurre inseriremo un nuovo stato all'interno dello stack corretto basandoci sul numero di parole che sono state coperte da quella traduzione.

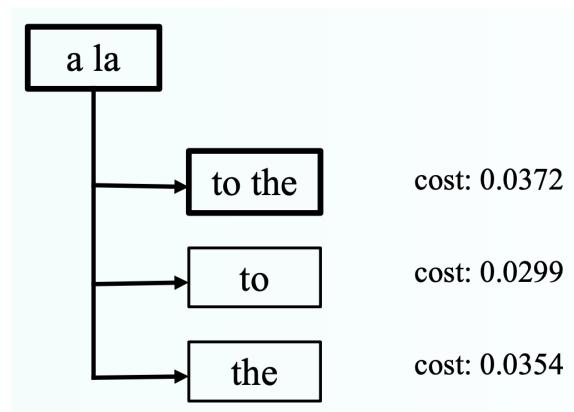
Confronto delle ipotesi

Una volta che abbiamo fatto le traduzioni e una volta che abbiamo all'interno degli stack le ipotesi di traduzione divise in base al numero di parole della frase originale che stiamo traducendo possiamo fare delle comparazioni. Per fare questa ricombinazione potremmo avere varie traduzioni di varie lunghezze, l'idea è di preferire le frasi che coprono la parte più semplice della frase, a queste viene assegnata una probabilità maggiore e quindi un costo minore.

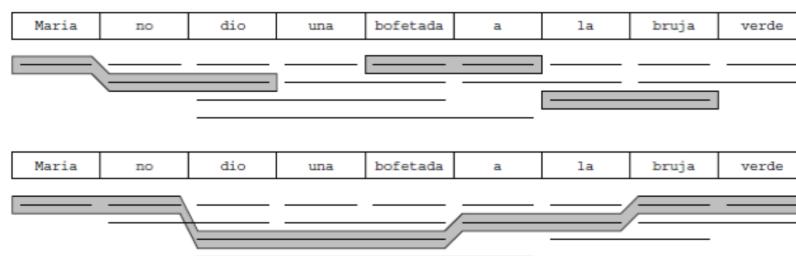


Eseguire questa stima del costo della traduzione diventa via via più complicato, per ogni opzione di traduzione dobbiamo andarci a controllare il costo del language model ignorando al momento il costo del riordino delle frasi. Quello che possiamo fare è considerare il costo futuro anche delle parti che non abbiamo ancora considerato. L'idea è di fare una stima per la traduzione della parte restante dell'input. Il costo futuro deve essere stimato per la parte mancante della frase basandoci sul costo del transation model, sul costo del language model andando per ora ad ignorare il distortion cost che al momento sarebbe troppo costoso da calcolare.

Una volta che abbiamo eseguito la stima delle possibili future traduzioni possiamo andare a scegliere quella che ha il costo minore.



Quello che otteniamo è un percorso di questo genere e alla fine riusciamo a creare la frase completa che abbiamo ottenuto con la traduzione. Questo percorso lo otteniamo con la programmazione dinamica ad esempio.



17.6 Valutare un sistema di Machine Translation

Come abbiamo già visto la valutazione di un sistema di Machine Translation viene effettuata considerando due metriche che sono la Fluency e la Faithfulness (Fidelity). Tutti i modelli devono essere valutati per essere in grado di comprendere se gli output che otteniamo sono abbastanza buoni o no.

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

La frase che consideriamo come ottimale è quella che viene prodotta con la traduzione svolta da una persona reale, quello che vogliamo fare è calcolare quanto la traduzione che abbiamo effettuato con il nostro modello è vicina alla traduzione di riferimento (umana).

La persona che deve valutare la traduzione deve considerare due metriche in particolare che sono:

- Fluency: possiamo chiedere alla persona quanto è leggibile, chiara e comprensibile la traduzione che abbiamo effettuato. Per giudicare una traduzione è possibile prendere la persona e farle giudicare la frase con un punteggio da 1 a 5 dividendo magari la Fluency in altre sotto-metriche come ad esempio la chiarezza o la naturalezza. Una alternativa consiste invece nel giudicare la frase nel complesso.
- Fidelity: per questa seconda dimensione che l'utente può giudicare abbiamo varie metriche. Due metriche tipiche che possono essere prese in considerazione sono l'Adequacy e l'informativeness.
 - L'adequacy di una traduzione controlla se la frase tradotta contiene le stesse informazioni contenute nella frase originale. Per misurare l'adequacy usiamo anche qua una scala.
 - Informativeness: controlla se ci sono sufficienti informazioni all'interno della frase tradotta, ad esempio possiamo fornire delle domande alla persona che deve fare la traduzione e poi vediamo se data solamente la frase tradotta è in grado di rispondere.

Ci sono anche altre metriche che possono essere utilizzate per giudicare la corretta qualità della traduzione combinando Fluency e Fidelity, ad esempio una valutazione tipica che viene utilizzata è l'edit cost ovvero misuriamo il numero di parole e il tempo necessario ad una persona per correggere l'output della traduzione in modo da farlo arrivare ad un livello accettabile. Per questa metrica possiamo utilizzare l'edit distance

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

tra la frase tradotta e la frase aggiustata e contiamo il tempo necessario per correggere gli errori.

17.6.1 Automatic Evaluation della Machine Translation

Una cosa che possiamo fare è considerare più di una traduzione di riferimento "umana" e poi andare a fare un confronto dell'output della Machine Translation con queste traduzioni di riferimento utilizzando dei sistemi automatizzati che non richiedano alla persona di dare uno score alle traduzioni.

Ci sono varie metriche che sono state proposte per calcolare lo score delle frasi che vengono tradotte, ad esempio Bleau, NIST, TER, Precision/Recall e Meteor. L'intuizione di queste metriche è che vogliamo che un buon sistema di Machine Translation sia in grado di produrre un output che è simile alla traduzione umana.

Questi sistemi in generale non richiedono per una frase una singola traduzione umana ma varie possibili traduzioni umane di riferimento perchè per calcolare lo score non ci fidiamo di una singola traduzione.

Quando vogliamo valutare il modello prendiamo la frase da tradurre che troviamo nel test set e produciamo l'output della MT e poi confrontiamo con la traduzione di riferimento assegnando uno score che è tanto più alto in base a quanto le due frasi sono simili tra loro.

BLEU

In BLEU eseguiamo il ranking dei vari output della Machine Translation considerando una media pesata del numero degli N-Gram che sono presenti sia nella traduzione umana che in quella della Machine Translation.

Bleu si basa sulla precision, nello specifico quello che facciamo è prendere la prima possibile frase prodotta in output dal MT, dividerla in unigram e calcolare quanti unigram sono presenti all'interno delle frasi

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

che prendiamo come riferimento. Il conteggio che otteniamo lo dobbiamo dividere per il numero di unigram che sono presenti nella frase originale. In questo caso ad esempio otteniamo 7/10.

Cand 1: Mary no slap the witch green.

Cand 2: Mary did not give a smack to a green witch.

Ref 1: Mary did not slap the green witch.

Ref 2: Mary did not smack the green witch.

Ref 3: Mary did not hit a green sorceress.

Poi facciamo la stessa cosa dividendo però la frase in bigram.

Cand 1: Mary no slap the witch green.

Cand 2: Mary did not give a smack to a green witch.

Ref 1: Mary did not slap the green witch.

Ref 2: Mary did not smack the green witch.

Ref 3: Mary did not hit a green sorceress.

Se abbiamo due frasi ripetiamo la stessa cosa su entrambe le frasi.

Una volta che abbiamo ripetuto il conteggio con le varie frasi ottenute dalla machine translation possiamo procedere con un calcolo complessivo dello score delle due frasi. Per questo andiamo a fare la somma di tutto quello che abbiamo contato dividendo per la somma di tutti i possibili unigram:

$$p_n = \frac{\sum_{C \in corpus} \sum_{n-gram \in C} \text{count}_{clip}(n - \text{gram})}{\sum_{C \in corpus} \sum_{n-gram \in C} \text{count}(n - \text{gram})}$$

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

Solitamente questo calcolo viene eseguito considerando unigram, bigram, trigram e al massimo fourgram. Inoltre si tende anche a penalizzare le traduzioni che sono troppo corte, possiamo infatti avere una frase che magari è molto corta nella traduzione e quindi ha più match di una frase che magari è più lunga ed ha anche più senso ma magari non ha troppi match in confronto alla lunghezza.

Abbiamo detto che la BLEU è una metrica che si basa sulla precision, per gestire anche questa penalizzazione si potrebbe introdurre la recall ma sarebbe troppo complicato quindi si preferisce inserire un termine che fa da penalty e che definiamo in questo modo:

$$BP = \begin{cases} 1 & \text{if } c > r \\ e^{(1-r/c)} & \text{if } c \leq r \end{cases}$$

Questo penalty è definito basandoci sulla lunghezza r della frase di riferimento e su c che è la lunghezza della frase tradotta. Se abbiamo la frase tradotta che è più lunga di quella di riferimento il peso sarà 1 altrimenti avremo un penalty che sarà esponenziale e che dipenderà da quanto la frase è breve.

Lo score complessivo per ognuna delle traduzioni prodotte dalla MT viena calcolato con la formula:

$$BLEU = BP * p$$

dove la p è lo score che abbiamo calcolato prima considerando unigram e bigram mentre BP è il penalty che abbiamo visto ora.

In questo score che produciamo l'obiettivo è trovare tante più sovrapposizioni possibili tra la frase target e la frase tradotta dal modello, non consideriamo invece l'ordine della frase tradotta.

Problemi di BLEU

BLEU possiamo utilizzarlo per valutare velocemente i miglioramenti che facciamo nel nostro modello rispetto e in che modo migliora la traduzione che produciamo con il nostro modello rispetto alla traduzione target. Ovviamente questo sistema ha alcune limitazioni che sono importanti da considerare:

- BLEU si focalizza su informazioni locali, ad esempio se facciamo qualche modifica alla frase che abbiamo prodotto e ad esempio la shiftiamo un minimo per produrre una traduzione che è quasi identica, BLEU ci potrebbe fornire uno score che è identico alla traduzione originale mentre invece lo score fornito da un umano sarebbe più basso.
- BLEU non funziona bene quando viene utilizzato per stimare le performance di modelli molto differenti tra di loro. In particolare la stima prodotta da BLEU per sistemi che utilizzano solamente la statistical Machine translation sarà differente rispetto alla stima che viene prodotta quando usiamo BLEU con sistemi che invece svolgono la Machine Translation introducendo anche la traduzione "umana". Se invece confrontiamo differenti phrase based MT allora la stima sarà buona.

17.7 Syntactic Models for MT

I primi sistemi di MT che abbiamo visto erano basati sulla traduzione delle singole parole, poi si è espansa questa prima idea andando ad utilizzare non più le singole parole ma le frasi in modo da considerare un contesto maggiore. Più recentemente si è passati dalla semplice traduzione delle frasi all'utilizzo di strutture gerarchiche in cui si utilizza la sintassi della frase. L'idea è di assegnare una struttura ad albero per la sintassi

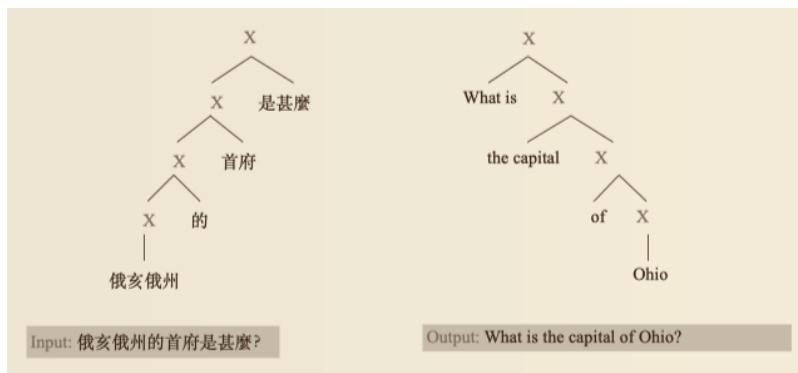
Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

di ogni coppia di frasi in linguaggi differenti con l'obiettivo di tradurre le frasi applicando una operazione di riordino dell'albero.

Il modello matematico che sta dietro a questa struttura sintattica parallela è il transduction grammar, lo possiamo vedere come una implementazione del syntactic transfer system.

Da un punto di vista generativo possiamo vedere la transduction grammar come la generazione di coppie di frasi allineate tra i due linguaggi.

Quello che abbiamo in pratica è una struttura gerarchica di frasi con la loro traduzione e la corrispondenza tra le sotto parti delle frasi, per ogni coppia di linguaggi abbiamo due alberi come quello della immagine:



Qua noi abbiamo un nodo all'interno dell'albero a sinistra che rappresenta la struttura della sintassi nella frase scritta in cinese e prendiamo la corrispondente regola nell'albero a destra che invece corrisponde all'albero sintattico della frase inglese.

Possiamo utilizzare una metrica per calcolare la derivazione di questi alberi e per selezionare quella che ci offre lo score più alto.

Il syntax based è una buona idea ma sfortunatamente non ci permette di avere una migliore traduzione ed è difficile da gestire.

Chapter 18

Lesson 17 - Neural Machine Translation

Fino ad ora abbiamo visto solamente la Statistical Machine Translation. La SMT è stato un importante campo di ricerca per molti anni. Presenta però dei problemi perchè i sistemi sono complessi, hanno tanti sotto componenti per cui va fatto il training. Ci sono da gestire molti dettagli nei modelli, c'è da fare molto lavoro di feature engineering in modo da essere in grado di catturare tutti i dettagli di ciascun linguaggio.

E poi soprattutto ci sono da mantenere delle risorse esterne e differenti che aumentano e si moltiplicano, in particolare se abbiamo la traduzione dall'inglese al francese ci servono delle tabelle con le frasi in inglese e la traduzione in francese e questo va fatto per tutti i vari linguaggi.

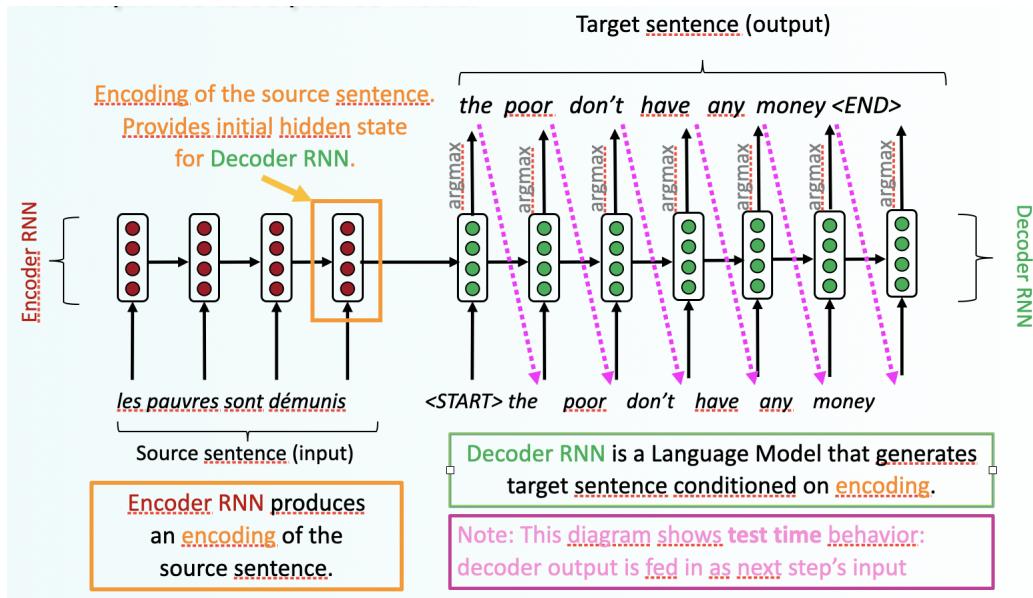
In generale per la SMT abbiamo bisogno anche di un lavoro umano e la quantità di questo lavoro umano aumenta se vogliamo ottenere traduzioni di un buon livello.

Nel 2014 però c'è stata una rivoluzione nel campo della Machine Translation. Questa rivoluzione è stata l'introduzione della Neural Machine Translation ovvero si è passati all'utilizzo di una (singola) neural

network per eseguire la traduzione da un testo all'altro. L'architettura utilizzata nella Neural Machine Translation è chiamata sequence-to-sequence o seq2seq e utilizza due RNN. Il nome sequence to sequence viene utilizzato perchè in pratica quello che si fa è prendere una sequenza in input e la si traduce in una sequenza in output tradotta.

18.1 Sequence To Sequence Model

Vediamo lo schema del Sequence To Sequence Model:



- Si prende la frase che vogliamo tradurre, in questo caso è una frase in francese e la diamo in input ad una RNN che è chiamata Encoder. La RNN che utilizziamo in questo esempio è una RNN unidirezionale in cui le informazioni viaggiano da sinistra a destra ma possiamo tranquillamente utilizzare anche una RNN bidirezionale o una multi layer. All'interno di questa RNN in realtà non inseriamo il testo vero e proprio ma sotto forma di embedding. Gli

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

embedding che inseriamo possono essere sia scaricati da internet pre-trained (Glove o Word2Vec) oppure possiamo eseguire il training da 0.

- All'interno della RNN ognuno degli stati produce un output e lo passa allo stato successivo. L'idea di questo Encoder è quella di utilizzare la RNN per produrre un encoding della sequenza di input. Questo encoding viene prodotto dall'ultimo degli stati della RNN. L'encoding che viene prodotto rappresenta poi il primo hidden state del Decoder della RNN.
- La seconda parte del sequence-to-sequence è una seconda RNN che chiamiamo Decoder che effettua la traduzione in inglese. La RNN del decoder è un conditional language model perchè produce una frase di output condizionata rispetto all'hidden state che riceviamo in input dalla Encoder RNN.
- Il decoder funziona in questo modo, il primo stato della RNN prende in input l'hidden state dall'Encoder e prende in input un token *<START>*. Questo primo stato produce un output che in questo caso è *the*. L'output viene prodotto eseguendo la traduzione della prima parola e prendendo poi la traduzione che ha la probabilità maggiore. In pratica questo primo stato della RNN produce una distribuzione di probabilità e noi calcoliamo l'argmax. Per tutte le parole successive il funzionamento è simile. Prendiamo in input l'output dello stato precedente e poi prendiamo come input l'hidden state e produciamo in output la traduzione. Anche per la traduzione abbiamo bisogno di avere un word embedding per tutte le parole. Quindi mi servono 2 word embedding, uno per la lingua in input e uno per il vocabolario della lingua output.

Quello che vediamo in questa immagine è quello che succede nel momento in cui testiamo il nostro modello, non quello che succede quando

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

dobbiamo fare il training del nostro modello.

Il Modello sequence-to-sequence è un esempio di Conditional Language model, è un language model perchè il decoder esegue la previsione della parola successiva della frase target, è condizionale perchè le previsioni che facciamo dipendono anche dalla frase di origine che utilizziamo come input e che viene rappresentata sotto forma di encoding.

In formula il Neural machine translation calcola direttamente $P(y|x)$:

$$P(y|x) = P(y_1|x)P(y_2|y_1, x)P(y_3|y_2, y_1, x)\dots P(y_t|y_{t-1}\dots y_1, x)$$

possiamo dividere la probabilità in un prodotto di probabilità per ognuna delle parole all'interno della traduzione. All'interno della probabilità la y è la frase di output mentre la x è la frase in input. In SMT non andavamo ad eseguire direttamente il learning di questa probabilità ma la spezzavamo in parti più piccole e semplici. Qua invece quello che succede è che eseguiamo direttamente il learning di $P(y|x)$ e questo è positivo perchè è più facile eseguire il learning e ottimizzare tutto insieme.

Come si esegue il training?

Per eseguire il training del sequence-to-sequence dobbiamo avere un grande corpus parallelo, nei classici modelli con le NN può diventare un problema se abbiamo dataset troppo grandi perchè potrebbe formarsi un plateau, nel caso della Neural Machine Translation invece questo non è un problema anzi ci fornisce la possibilità di utilizzare una maggiore quantità di dati.

Il sequence To sequence model non viene utilizzato solamente per la Machine translation, ha utilizzi anche nei seguenti campi:

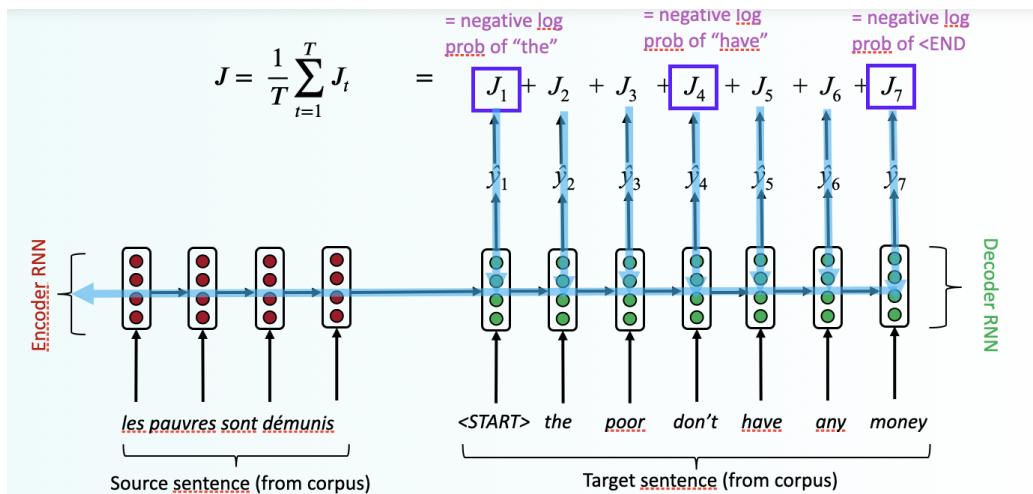
- Viene utilizzato per riassumere dei testi, il nostro seq2seq prende in input un testo lungo e restituisce in output un testo breve.
- Viene utilizzato per i dialoghi.

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

- Viene utilizzato per il parsing, abbiamo la sequenza in input e abbiamo l'output che è la frase di input ma parsata.
- Può essere utilizzato anche per la code generation, da una frase in natural language ad un codice Python ad esempio.

18.1.1 Eseguire il training del Sequence To Sequence Model

Il training del sequence to sequence model lo possiamo rappresentare nella seguente immagine:



Eseguiamo il training fornendo frasi di esempio al nostro modello, gli permettiamo di lavorare e di produrre degli output, quindi di produrre delle parole che corrispondono alla traduzione di quelle che abbiamo dato in input. Per ognuna di queste parole che produciamo in input viene prodotta una loss, consideriamo infatti l'errore che stiamo facendo con la nostra traduzione. Queste loss vengono sommate tutte insieme. La loss che utilizziamo è la cross entropy o la negative Log Likelihood. L'encoder e il decoder sono parte della stessa rete quindi viene prodotta

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

una singola loss e viene propagata poi tramite backpropagation per eseguire un aggiornamento dei pesi sia dell'encoder sia del decoder.

Si preferisce fare il training dell'intero modello tutto insieme e non fare separatamente il training di encoder e decoder in modo da avere l'ottimo di tutto il modello e non l'ottimo di uno e dell'altro che poi magari mettendoli insieme non si otterrebbe l'ottimo per tutto il sistema. C'è però anche la nozione di pre-trained, potremmo ad esempio essere interessati ad eseguire il learning del Decoder come un language model a se stante e poi lo utilizziamo per eseguire l'inizializzazione del Decoder e poi comunque facciamo il training.

Questa è una soluzione molto semplice e potrebbe però essere molto costosa.

La differenza fondamentale tra questo schema per il training e lo schema che abbiamo visto in precedenza per il test del modello è che nel precedente schema avevamo che l'output di ciascuno degli stati della RNN (con output intendo la parola tradotta) veniva dato in input allo stato successivo della RNN. Qua invece, dato che facciamo il training, non prendiamo in input l'output dello stato precedente ma prendiamo in input la parola target reale e produciamo l'output della rete. In pratica visivamente lo vediamo perchè l'output dello stato s_t non è connesso con l'input dello stato s_{t+1} .

La lunghezza della frase in input e della traduzione non sono fissate. Da un punto di vista più pratico possiamo prendere ciascuna delle frasi che abbiamo in input, aggiungiamo del padding in modo che la lunghezza della frase sia sempre la stessa (magari mettiamo in padding considerando la frase più lunga che abbiamo nel corpus) e poi quando facciamo il training facciamo in modo di non considerare il padding come input.

18.1.2 Beam Search Decoding

Quello che abbiamo visto fino ad ora è chiamato Greedy Decoding perchè nel decoder quello che facciamo è generare ad ogni step del decoder una parola, facciamo la nostra scelta e poi non torniamo indietro e non abbiamo modo di cambiare la scelta fatta. Questo vuol dire che localmente abbiamo fatto la scelta ottima ma globalmente la traduzione potrebbe non essere quella ottima, quindi in pratica non calcoliamo realmente l'argmax di $P(y|x)$.

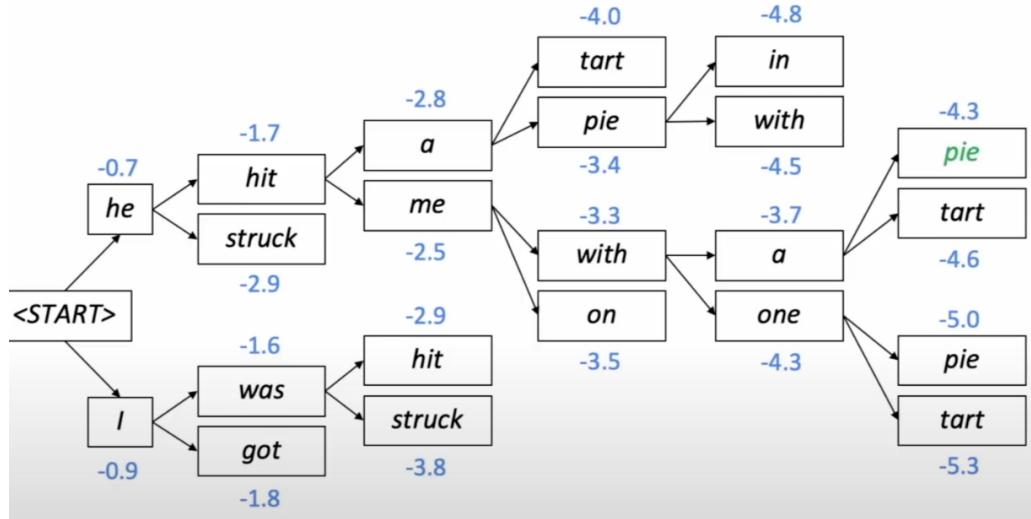
Possiamo migliorare questo tipo di traduzione utilizzando la Beam Search. L'idea della Beam Search è che invece di selezionare solamente la soluzione migliore, consideriamo varie possibili traduzioni, non possiamo considerarle tutte quindi ci serve un modo per fare un pruning e mantenere solamente i percorsi migliori. La Beam Search comunque non garantisce la scelta della soluzione migliore ma è semplice da sviluppare.

Idealmente il nostro obiettivo sarebbe quello di massimizzare questa probabilità, per avere una massimizzazione reale dovremmo andare ad enumerare tutte le possibili y e questo sarebbe troppo costoso perchè avremmo una complessità di $O(V^T)$ dove V è la dimensione del nostro dizionario e T è la lunghezza della frase tradotta.

$$P(y|x) = P(y_1|x)P(y_2|y_1, x)P(y_3|y_2, y_1, x)\dots P(y_t|y_{t-1}\dots y_1, x)$$

Con la Beam Search in ogni step del decoder teniamo traccia delle K traduzioni più probabili (la k è un numero tra 5 e 10 in genere). Non abbiamo comunque la garanzia di trovare la migliore soluzione possibile ma nella maggior parte dei casi ci riusciamo, è molto efficiente e fissiamo la quantità di spazio necessaria in modo da non avere una esplosione in uno spazio esponenziale più grande.

Vediamo un esempio di come funziona la Beam Search:



Quando noi consideriamo ad ogni step del Decoding 2 alternative ($K=2$), ad esempio partendo da $<start>$ abbiamo generato una distribuzione di probabilità e prendiamo le due traduzioni che hanno una probabilità più alta. Poi andiamo avanti in modo ricorsivo e quindi generiamo la parola successiva partendo dalle due che abbiamo ora e delle 4 che abbiamo ottenuto prendiamo le 2 con la probabilità più alta rimuovendo le 2 con la probabilità più bassa. Andiamo avanti espandendo le due probabilità più alte fino a che non abbiamo ottenuto la traduzione complessiva della frase.

Per ognuna di queste traduzioni viene prodotto uno score che dipende dalla probabilità delle varie singole traduzioni delle parole, lo score che produciamo è il logaritmo della somma di tutte queste singole probabilità ed è un numero negativo perché le probabilità sono comprese tra 0 e 1. L'obiettivo è massimizzare questo score.

Per trovare la traduzione complessiva prendiamo poi la traduzione che ha prodotto lo score più alto e torniamo indietro al token start considerando tutte le parole nel mezzo.

Un problema che possiamo avere con la Beam Search consiste nel capire quando dobbiamo fermarci con la traduzione. Quando usiamo la

Greedy decoding ci fermiamo nel momento in cui troviamo l'END token e a quel punto abbiamo ottenuto la traduzione che ci interessava. In Beam search è differente:

- Eseguendo la Beam Search non sappiamo quando troveremo l'END token, in un percorso magari lo troviamo prima e in uno dopo. Da un punto di vista pratico quello che si fa è prendere ogni volta le frasi in cui abbiamo trovato END e continuare ad espandere le altre K migliori.
- Per capire quando ci dobbiamo fermare consideriamo un numero di ipotesi N che dobbiamo avere con il token END. In alternativa possiamo fissare un tempo T a cui dobbiamo fermarci.

Alla fine della Beam Search abbiamo una serie di possibili ipotesi (varie traduzioni) con associato il loro score. Il problema in questo caso è che si tende a scegliere la traduzione più breve perché quelle più lunghe hanno uno score più basso dato che vanno avanti con le espansioni per più iterazioni. Per risolvere questo problema ed evitare di scegliere sempre l'ipotesi più breve possiamo prendere tutte le ipotesi e dividere per la lunghezza della frase tradotta in modo da fare la normalizzazione e scegliere quella che realmente ha lo score più alto.

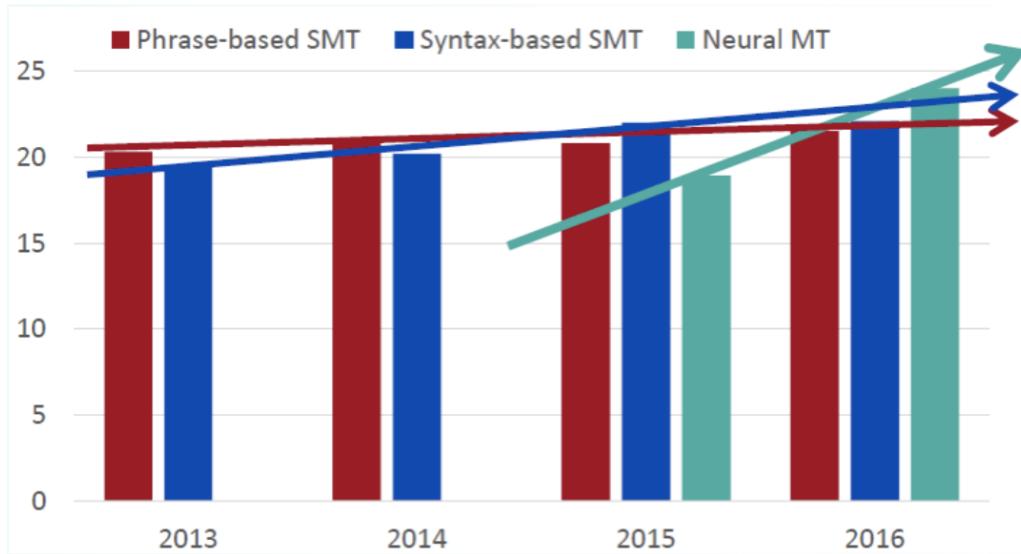
18.1.3 Pro e contro di NMT e di SMT

- I sistemi NMT hanno delle performances migliori, dal punto di vista della Fluency, poi utilizzano meglio il contesto in cui si trova la frase e utilizzano meglio le somiglianze tra le frasi nel senso che se fanno il training su una frase e poi ne trovano una simile, sono in grado di generalizzare meglio e di ottenere una frase tradotta per questa frase simile che sarà migliore rispetto a quella che possiamo ottenere in una SMT. In un certo senso le NMT svolgono un lavoro migliore perché generano le parti della traduzione facendo

una scelta che dipende da quello che hanno visto fino a quel momento. Questo è il motivo per cui la traduzione è più Fluent, le parole sono selezionate per produrre una migliore traduzione.

- Abbiamo anche un miglioramento per quel che riguarda l'ottimizzazione della NN, perchè abbiamo una singola Neural Network da ottimizzare, non abbiamo varie parti da ottimizzare in modo autonomo.
- È necessario meno sforzo "umano" in particolare non abbiamo feature engineering. Inoltre possiamo utilizzare gli stessi metodi più o meno per ogni coppia di linguaggi.
- Un problema è che le NMT sono più complicate da interpretare e anche più difficili da debuggare, se abbiamo un errore è complicato capire dove si trova quel bug e fixarlo. Può capitare che se abbiamo fatto un errore dovremo poi eseguire di nuovo il training del sistema con più dati, non abbiamo una regola precisa per eseguire la traduzione, per esempio non abbiamo la regola che ogni frase deve avere un soggetto. In alcuni casi può essere un problema non avere un modo per aggiungere delle regole in più o dei vincoli all'interno della traduzione.

Nella seguente immagine possiamo vedere in che modo si stanno evolvendo i vari sistemi di Machine Translation nel corso del tempo. La metrica che viene utilizzata in questo grafico è il BLEU score che va a fare una comparazione tra la frase che si è ottenuta con la traduzione e la frase di riferimento (c'è sempre da ricordare che il BLEU score mi fornisce uno score buono anche se la frase che abbiamo non è ordinata).



Dal grafico quello che possiamo vedere è che la SMT ha avuto un piccolo miglioramento in termini di performances nel corso degli anni e lo stesso vale anche per la sintax bases SMT. La NMT invece è partita più tardi ma ha avuto un incremento delle performance molto veloce.

Come possiamo vedere quindi la NMT è stato un grande successo per la Machine Translation, anni e anni di ricerche nel campo della SMT sono state in pratica spazzate via dal lavoro di chi ha studiato la NMT in qualche mese. Nonostante i progressi effettuati la NMT non ha comunque ancora raggiunto lo scopo finale, ci sono ancora delle difficoltà che devono essere risolte:

- Non si sa ancora come gestire le parole che non sono presenti all'interno del dizionario.
- Se dobbiamo tradurre un documento da un certo dominio ad un altro non sempre è facile riuscire a catturare le espressioni in entrambi i domini.
- Non sempre si riesce a mantenere un contesto lungo una sequenza di parole troppo lunga.

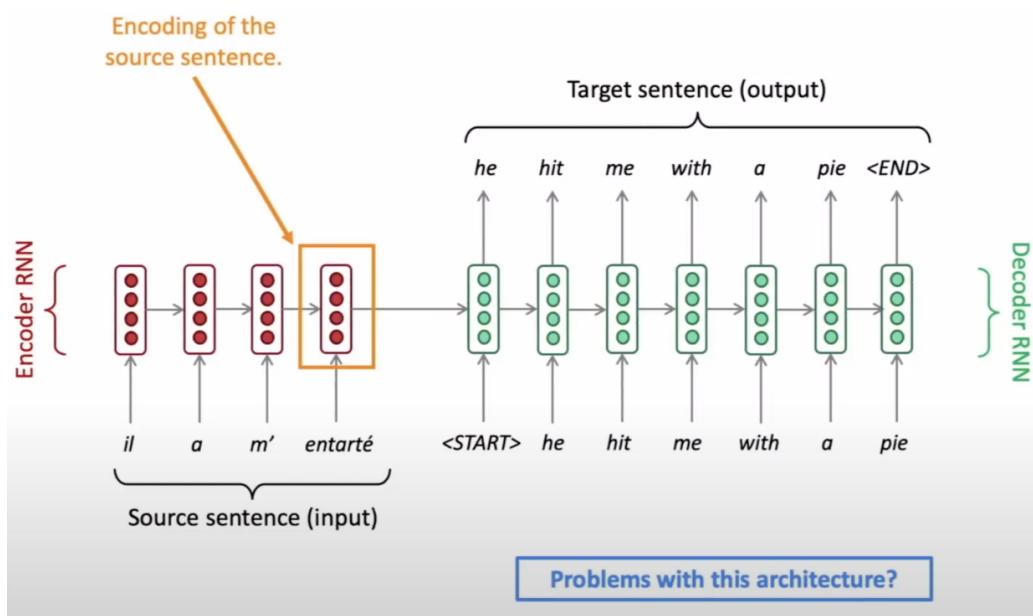
Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

- Spesso non sono presenti corpus troppo grandi per eseguire il training di sistemi come questi.

Per queste ragioni la ricerca continua e sono state apportate alcune migliorie a questa Seq2Seq Vanilla che abbiamo visto fino ad ora, la più grande è l'Attention.

18.2 Attention

Consideriamo la struttura che abbiamo visto in precedenza:



Questa struttura comporta un problema, può funzionare se abbiamo abbastanza memoria ma in generale eseguendo questa procedura non riusciamo a ricordarci tutto quello che abbiamo in input, specialmente se le sequenze sono particolarmente lunghe.

Abbiamo quindi quella che viene chiamata Information Bottleneck ovvero stiamo forzando tutta l'informazione in input ad essere compressa

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

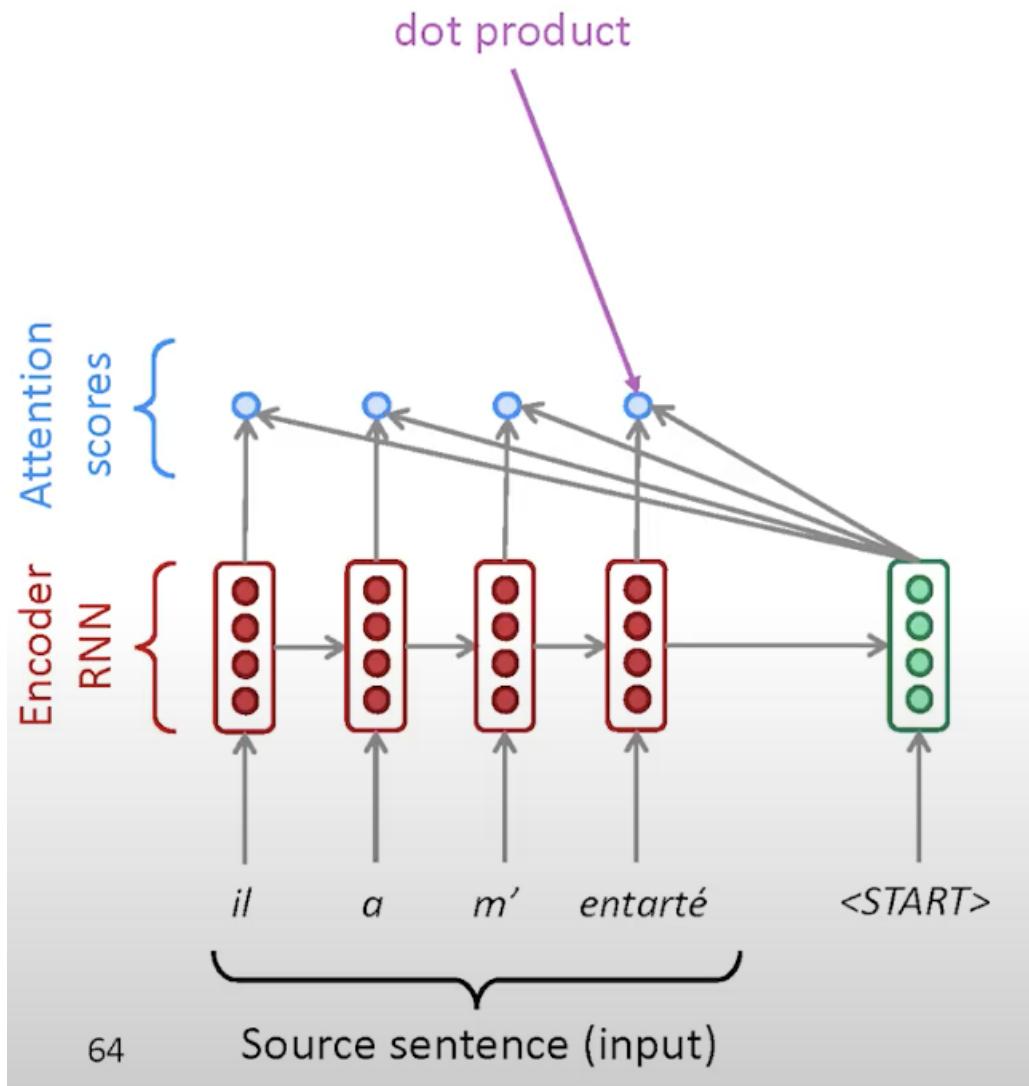
all'interno di un singolo vettore di output del nostro Encoder perchè il Decoder prende in input un singolo vettore di encoding.

La soluzione a questo problema di Information Bottleneck è quella che viene chiamata Attention, l'idea è di connettere ogni step del decoder con una connessione diretta all'encoder in modo da focalizzarci su una parte in particolare della sequenza di origine per capire quale parte della frase potrebbe essere utile.

18.2.1 Funzionamento dell'Attention

Vediamo come funziona l'attention:

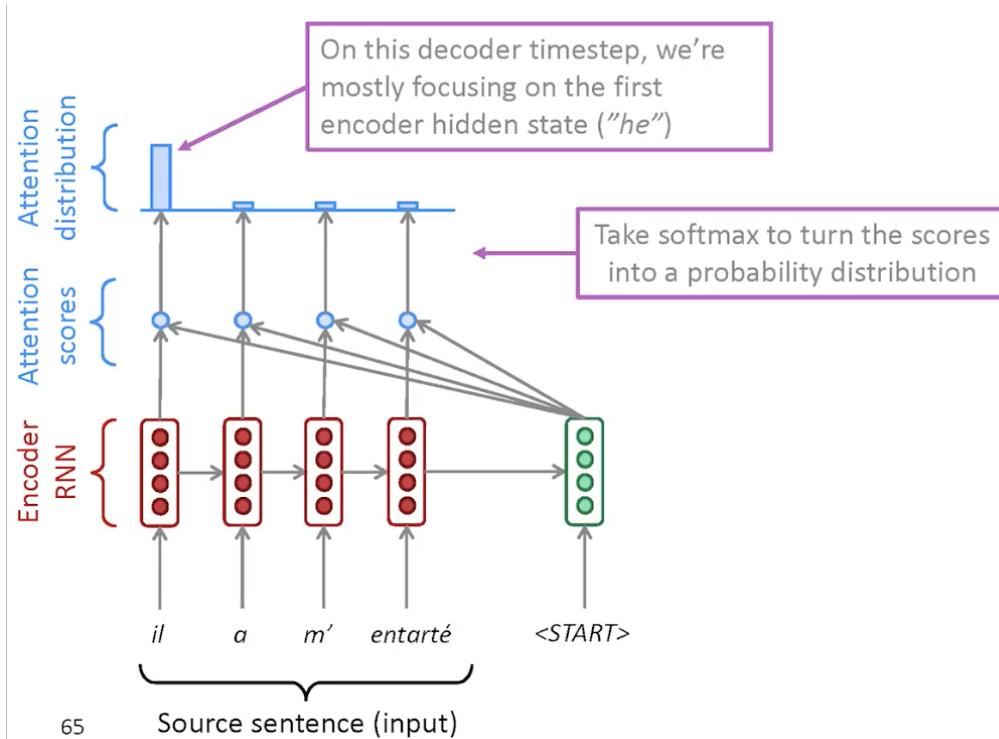
- Per ogni parola in input prendiamo il vettore prodotto dall'Encoder e poi eseguiamo il prodotto con il vettore in input del Decoder producendo quello che viene chiamato Attention Score per ognuna delle parole in input.



- L'attention score che abbiamo prodotto fino ad ora viene dato in input alla softmax function che mi produce una probabilità per ognuno degli attention score, otteniamo quindi una distribuzione di probabilità (vogliamo che le probabilità si sommino a 1). Questa viene chiamata Attention distribution, il significato è che in questo caso, dato che stiamo considerando la prima parola della frase, la

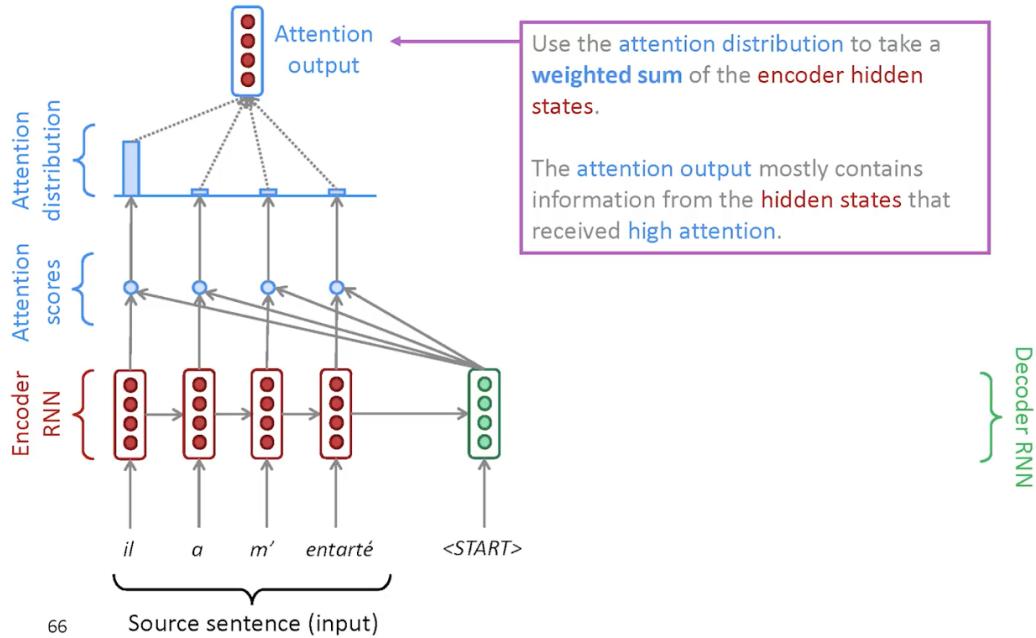
Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

maggior parte della probability mass è nella prima parola della frase.

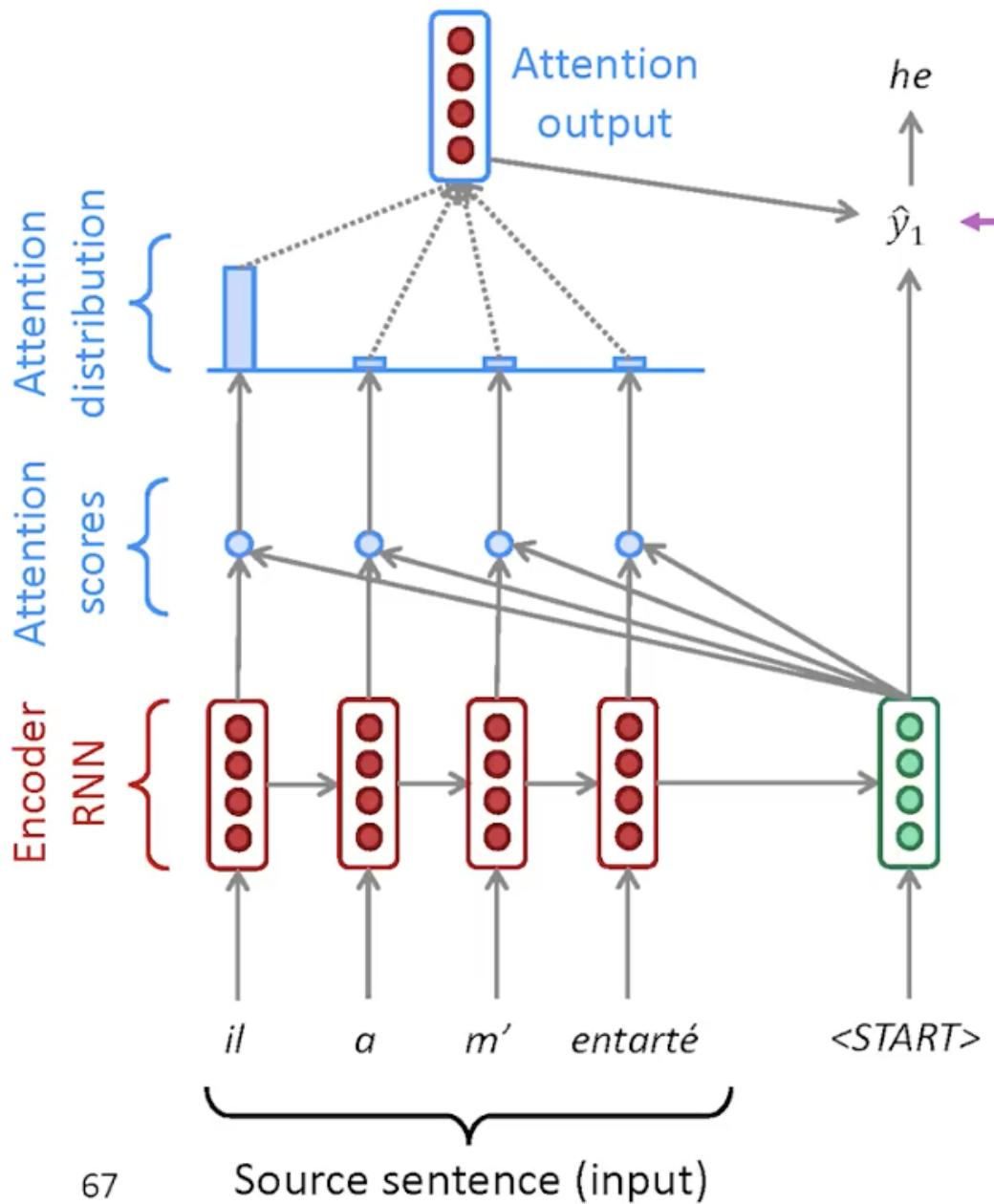


Questo si traduce nel fatto che dato che ci troviamo nel primo step del decoding, saremo focalizzati in modo particolare sul primo hidden state dell'encoder.

- Una volta che abbiamo ottenuto questa attention distribution possiamo calcolare una somma pesata delle varie distribuzioni pensando in base alla probability mass, in questo caso ad esempio il vettore dell'attention output conterrà in modo particolare informazioni riguardanti il primo hidden state perché è quello che ha la probability mass più alta.



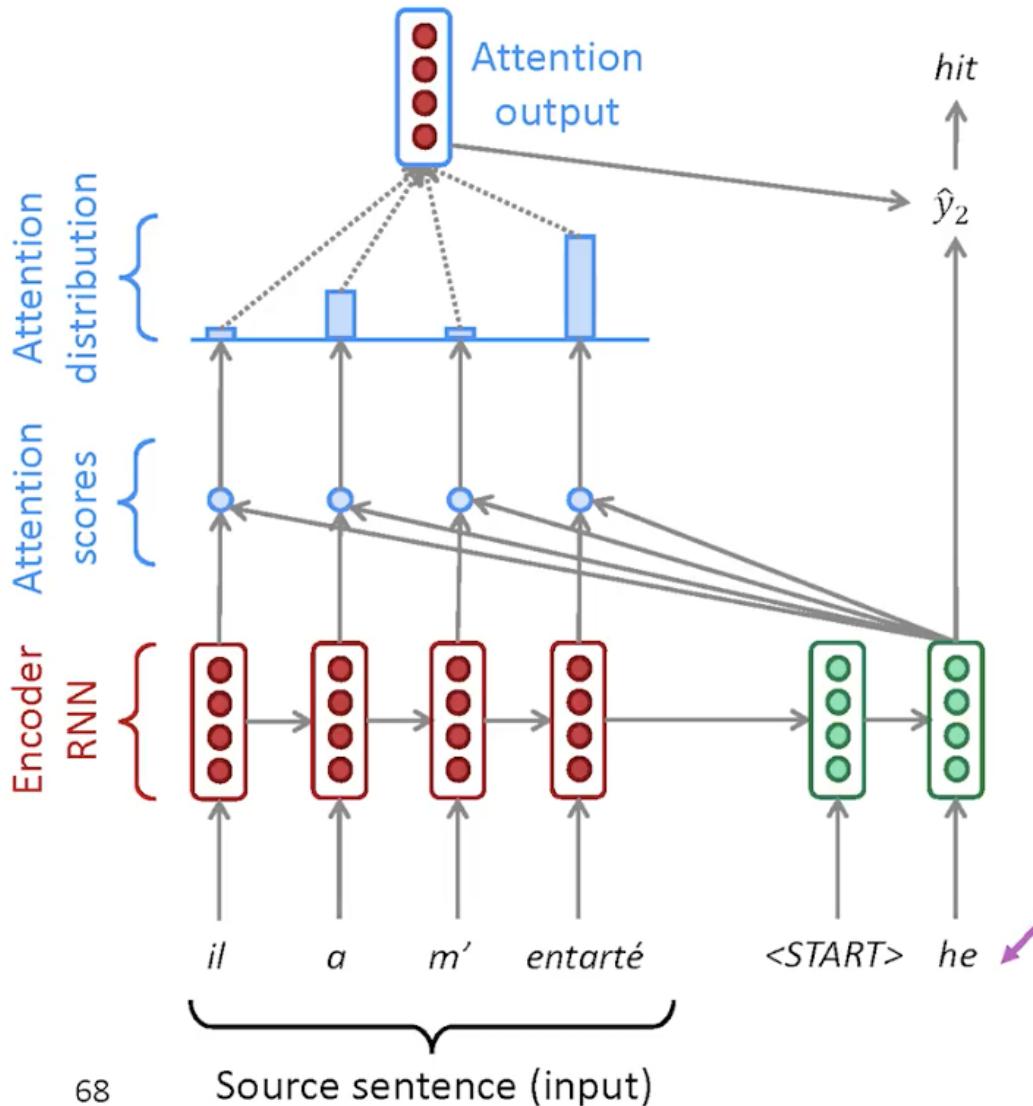
- Dopo che abbiamo calcolato l'attention output vogliamo fare in modo che questo vettore influenzi l'output del decoder. Quello che si fa quindi è concatenare l'attention output con l'hidden state del decoder in modo da produrre l'output migliore per quella parola.



- Per lo step successivo il funzionamento è lo stesso, vediamo che ora la probability mass è più sulla seconda parola che sulle altre perché

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

ci troviamo nel secondo step del decoding.



Una cosa da notare è che in alcuni casi può essere utile, per avere più informazioni sull'attention dello step precedente, integrare l'attention output all'interno dell'input dello step successivo del decoder.

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

- Eseguiamo il calcolo dell'attention su ogni step e andiamo avanti fino a che non abbiamo prodotto tutta la frase.

18.2.2 Equazioni dell'attention

Vediamo in modo formale come funziona l'attention:

- Per produrre l'attention score abbiamo detto che consideriamo ad ogni timestamp gli hidden state dell'encoder h_1, \dots, h_n e consideriamo l'hidden state del decoder s_t . Quello che facciamo è il prodotto di questi vettori per ottenere un altro vettore di dimensione uguale al numero di hidden state dell'encoder.

$$e^t = [s_t^T h_1, \dots, s_t^T h_n]$$

- Ora che abbiamo il vettore e^t possiamo calcolare la softmax su questo vettore in modo da ottenere una distribuzione di probabilità:

$$\alpha^t = \text{softmax}(e^t)$$

- Ora utilizziamo la α^t per eseguire la somma pesata dell'hidden state dell'encoder in modo da ottenere l'output dell'attention al tempo t:

$$a^t = \sum_{i=1}^N \alpha_i^t h_i$$

- Infine la a^t viene concatenata alla s_t in modo da poterla utilizzare all'interno del decoder.

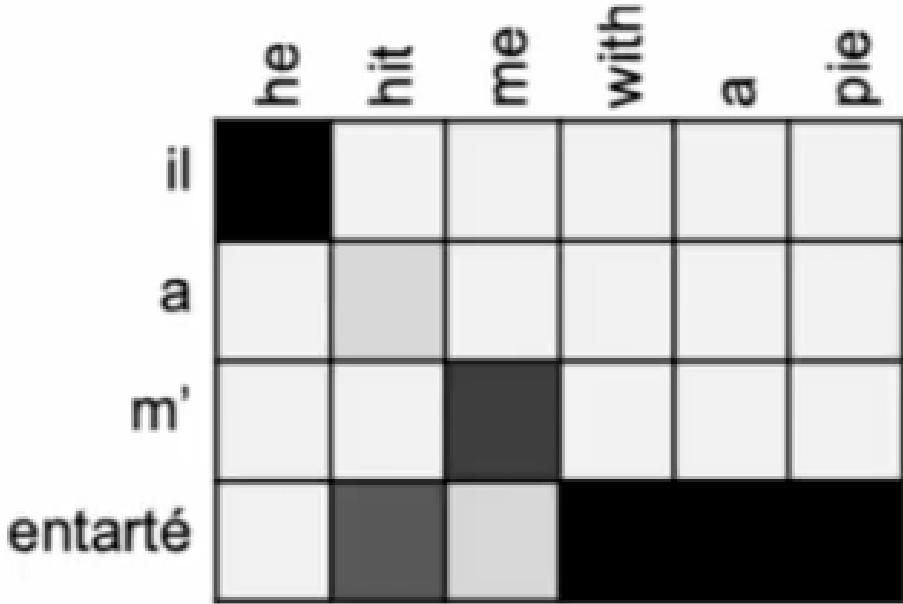
18.2.3 Considerazioni sull'Attention

L'attention è stata utilissima per migliorare in modo eccezionale le performances della NMT. Risulta molto utile perchè permette di focalizzarci su alcune parti in particolare della frase di origine.

L'attention risolve anche il bottleneck problem perchè permette al decoder di guardare direttamente alla parola di origine andando a bypassare la bottleneck e il singolo vettore dato in output dall'encoder.

L'attention permette di risolvere il problema del Vanishing gradient, specialmente se la frase da tradurre è molto lunga è utile utilizzarla. In particolare aiuta la connessione diretta tra decoder ed encoder e quindi è come una connessione "shortcut" che permette di far passare il gradiente ed evitare quindi il vanishing gradient.

Tramite l'attention riusciamo anche ad avere una migliore interpretabilità della nostra rete, quando la sequenza in input è particolarmente lunga infatti è più complesso ottenere informazioni dalla rete. Ora grazie all'attention riusciamo sempre ad avere un grafico che mi mostra per ogni parola tradotta quale è la corrispondente parte della frase originale che è più importante per quella parte di traduzione.



Questo ci consente anche di ottenere "gratuitamente" gli allineamenti delle frasi, in pratica la rete esegue il learning degli allineamenti da sola. Questa è una delle cose più importanti e interessanti dell'Attention, esegue il learning di questo tipo di strutture in modo non supervisionato.

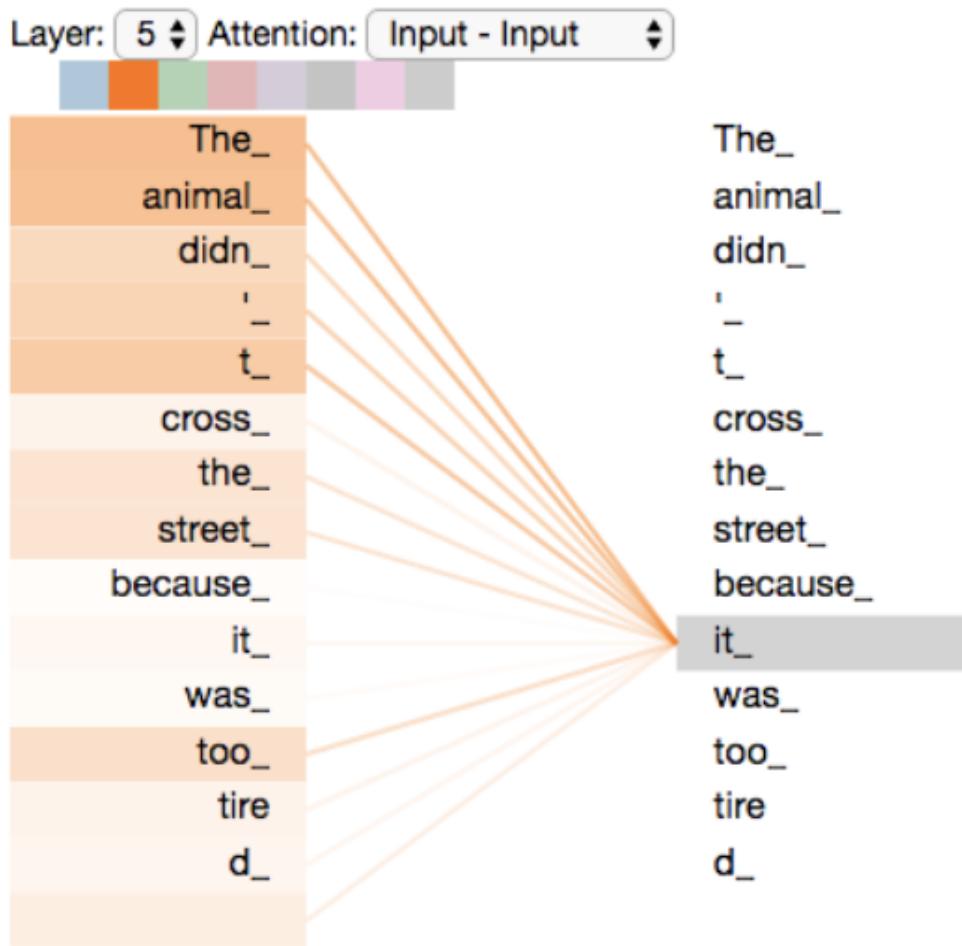
18.2.4 Altri utilizzi dell'attention

L'attention non viene utilizzata solamente per la Machine Translation, abbiamo anche altri possibili utilizzi nel campo della NLP.

Contestualizzare

Abbiamo sempre parlato di word embedding, ad ogni parola abbiamo sempre associato un singolo vettore anche se quella parola può avere molti significati. L'attention ci permette di introdurre all'interno della

rappresentazione della parola il contesto in cui quella parola compare consentendo quindi di osservare anche altre parole della frase.



In questo esempio sopra abbiamo che *it* viene associato alle varie parole di sinistra, ad alcune in modo più netto ad altre in modo meno netto. In questo caso ad esempio si vede che *it* è associato alla parola animale.

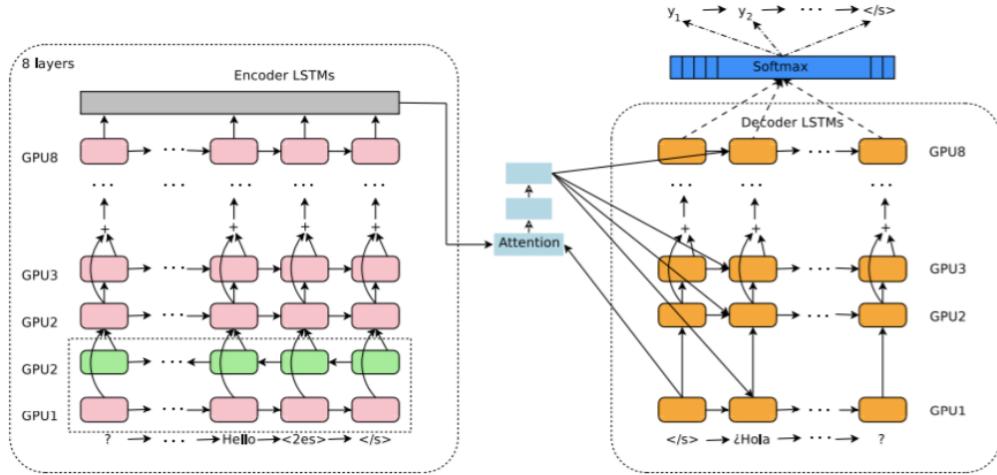
Transformer Models

Esistono anche dei language Models che incorporano l'attention all'interno della generazione del modello. Un esempio è Bert, poi abbiamo anche XLNET, RoBERTa e altri prodotti da varie aziende. Questi modelli diventano solitamente sempre più grandi, acquisiscono nuovi layer e necessitano sempre di più tempo per eseguire il training.

Model	Dimensioni	Parametri
BERT	24-layer, 1024-hidden, 16-heads.	340M
GPT-2	48-layer, 1600-hidden, 25-heads	1558M
Transformer XL	18-layer, 1024-hidden, 16-heads.,	257M
XLM	12-layer, 1024-hidden, 8-heads	
RoBERTa	24-layer, 1024-hidden, 16-heads	355M
DistilBERT	6-layer, 768-hidden, 12-heads	134M
CTRL	48-layer, 1280-hidden, 16-heads	1.6B
alBERT	12 repeating layer, 128 embedding, 4096-hidden, 64-heads	223M
T5	24-layers, 1024-hidden-state, 65536 feed-forward hidden-state, 128-heads	11B

Struttura della NMT di Google

Questa che vediamo nella immagine è la struttura corrente della Neural Machine Translation utilizzata da Google.

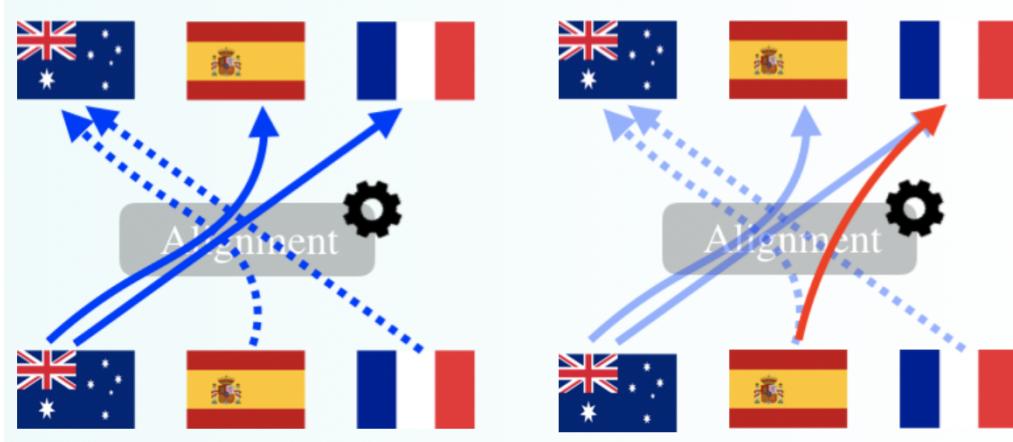


Nella parte dell'encoder abbiamo 8 layer, ognuno di questi viene dato in pasto ad una GPU per eseguire il training. Viene utilizzata l'attention per fare in modo che nella RNN del Decoder si abbiano maggiori informazioni sul contesto delle singole parole. Eseguire il training di questa architettura ha richiesto 3 settimane utilizzando 100 TPU, quindi è un task abbastanza grande e complesso.

18.3 Multilingual Translation

C'è un'altra ricerca recente che vale la pena tenere in considerazione. Consideriamo il caso della traduzione mono-lingua, quindi abbiamo una lingua di partenza e abbiamo una lingua di destinazione. Per ogni coppia di linguaggi abbiamo un encoder e un decoder.

Può essere possibile utilizzare l'encoder di un linguaggio in un altro linguaggio in modo da fare la traduzione di lingue per cui non ho già un encoder? Ad esempio considerando questa immagine, noi qua abbiamo l'encoder per tradurre da inglese a francese e da inglese a spagnolo e viceversa.



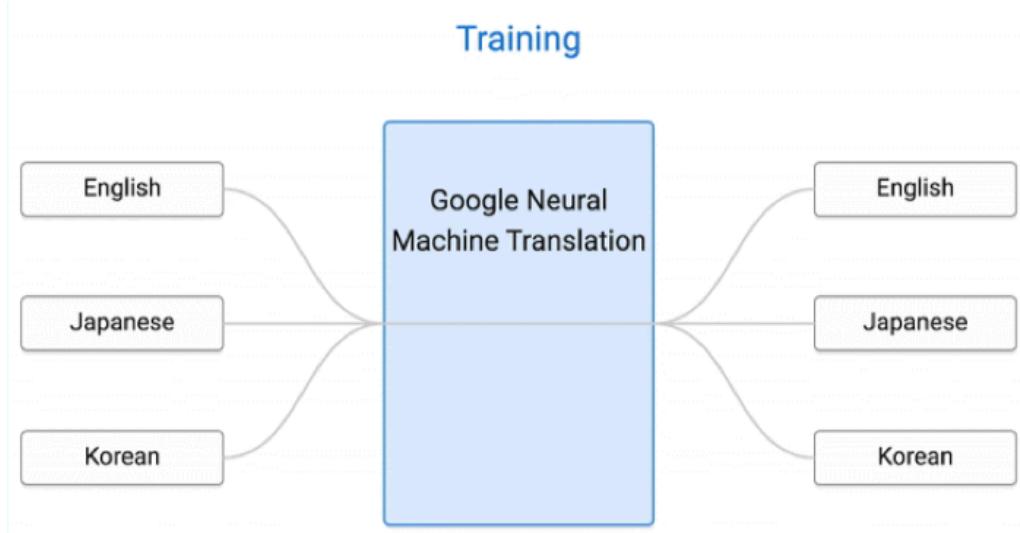
Non sappiamo però come tradurre il francese in spagnolo perchè non abbiamo fatto il training dell'encoder e del decoder corrispondente. Potremmo però utilizzare un encoder intermedio.

Ad esempio, se abbiamo l'encoder dell'inglese possiamo tradurre prima lo spagnolo in inglese e poi dall'inglese facciamo la traduzione in francese. In questo modo riusciamo a tradurre lo spagnolo in francese anche se non abbiamo una connessione diretta e anche se non abbiamo fatto il training dell'encoder, questo metodo è chiamato zero shot translation.

In questo modo il linguaggio di input della nostra traduzione diventa praticamente irrilevante, l'unica cosa che conta è il linguaggio di destinazione.

18.3.1 Zero-shot Translation

Abbiamo parlato della Zero-shot Translation, quello che succede durante il training di questo modello è che dobbiamo creare un encoder che sia comune a tutti i linguaggi poi consideriamo sempre in input il linguaggio target di destinazione.



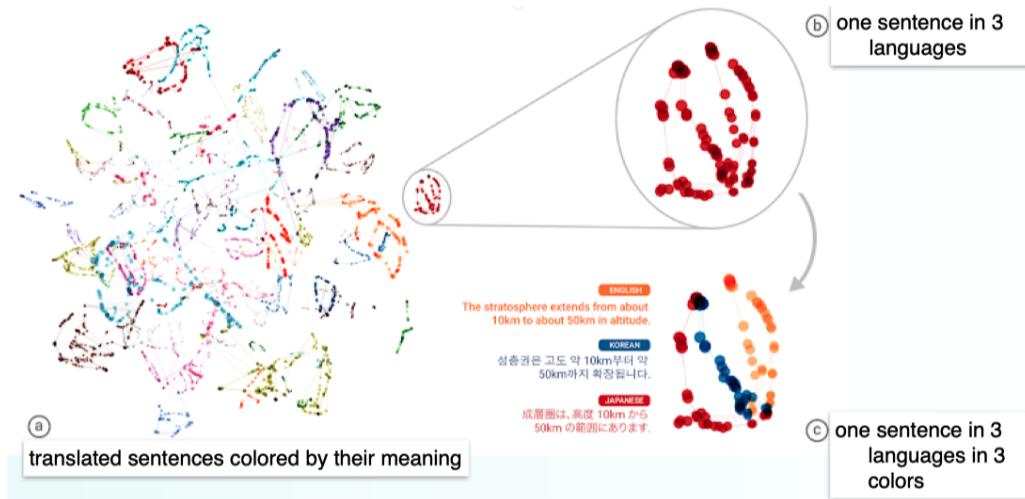
Si tratta quindi di un modello che acquisisce informazioni tutte insieme, possiamo imparare in che modo deve essere fatta la traduzione dall’inglese al giapponese e poi possiamo applicare lo stesso modello per eseguire la traduzione dal Koreano all’inglese semplicemente sfruttando l’encoding in comune.

Un effetto dell’utilizzo di questo zero-shot translation è che non abbiamo la necessità di avere dei corpus per ogni coppia di testi che vogliamo tradurre.

18.3.2 Visualizzare la traduzione

Una cosa che possiamo fare una volta che abbiamo creato il nostro modello è visualizzare il risultato della traduzione. Le frasi che abbiamo tradotto vengono colorate in base al loro significato e poi vengono rappresentate all’interno di un grafico. Possiamo vedere dal grafico che le parole che hanno significati simili sono tutte vicine tra loro, se eseguiamo uno zoom poi vediamo che anche le frasi tradotte nei 3 linguaggi differenti hanno un significato simile tra loro dato che si trovano vicine tra loro nel grafico.

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.



18.4 Conclusione

La Neural Machine Translation non è in grado di comprendere, per come l'abbiamo vista ora, il significato delle frasi, si limita solamente ad assemblare le frasi in modo che abbiano senso. Allo stesso tempo non si occupa neanche di comprendere la grammatica delle frasi.

Nonostante queste limitazioni la NMT funziona molto bene e inoltre fornisce anche la possibilità di svolgere la traduzione multilingua utilizzando lo zero-shot learning.

I modelli che abbiamo visto fino ad ora nel campo della NMT e soprattutto l'Attention hanno ispirato anche altre ricerche in NLP e in particolare il Transformer Model che viene utilizzato in vari task oltre alla MT.

Chapter 19

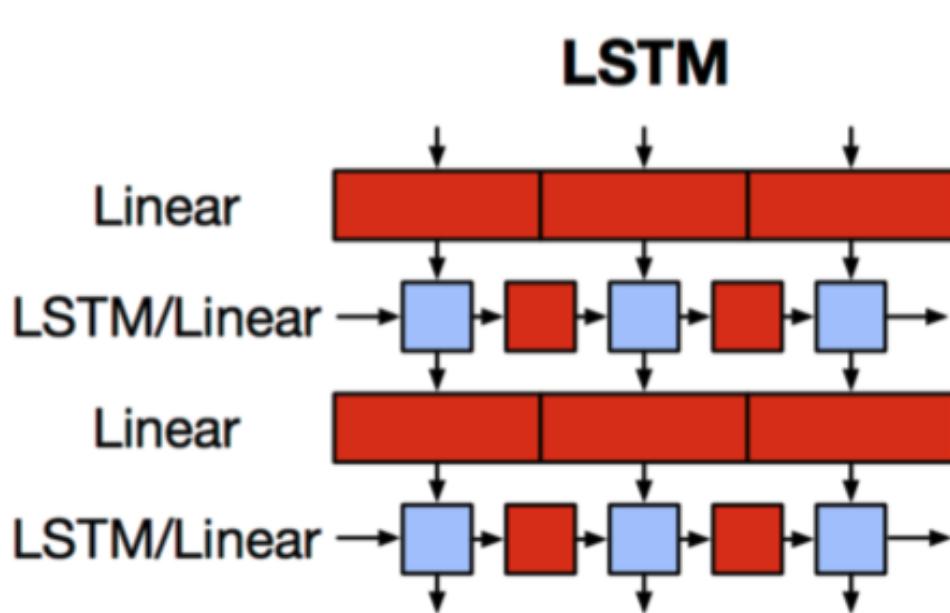
Transformer e Bert

19.1 Introduzione

Fino ad ora abbiamo visto il Seq2Seq Model con l'utilizzo dell'Attention che permette di ottenere un buon risultato nel task di traduzione. L'attention ci permette di utilizzare nel decoder una informazione riguardante il contesto delle varie parole risolvendo quindi il problema delle sequenze di testo troppo lunghe (Information bottleneck) e del vanishing gradient. La natura sequenziale di questi sistemi però ne limita il parallelismo e questo può rendere il tutto molto lento.

Ora consideriamo un modello differente che è il Transformer che utilizza l'attention per migliorare le performance del training dei modelli di MT. Per alcuni task specifici il transformer è migliore della Google Neural Machine, soprattutto se consideriamo il fatto che è pensato per essere parallelizzato.

La motivazione per cui è stato sviluppato questo Transformer è la parallelizzazione. Noi siamo in una situazione come questa:



La LSTM passa lungo le varie parole da sinistra a destra, ognuno di questi layer è sequenziale. Abbiamo visto che in questo modello sequenziale possiamo integrarci l'attention in modo da gestire le dipendenze che si trovano a distanze molto lunghe, l'attention ci garantisce l'accesso ad ogni stato, magari potremmo semplicemente utilizzare l'attention senza la necessità di avere la RNN.

19.2 Il transformer

L'idea del Transformer, presentata nel paper Attention is all you need [3] è che possiamo liberarci completamente della LSTM utilizzando l'attention per gestire completamente le dipendenze tra input e output.

Nel paper viene descritta l'architettura di questo transformer che non utilizza alcuna RNN e quindi può andare completamente in parallelo

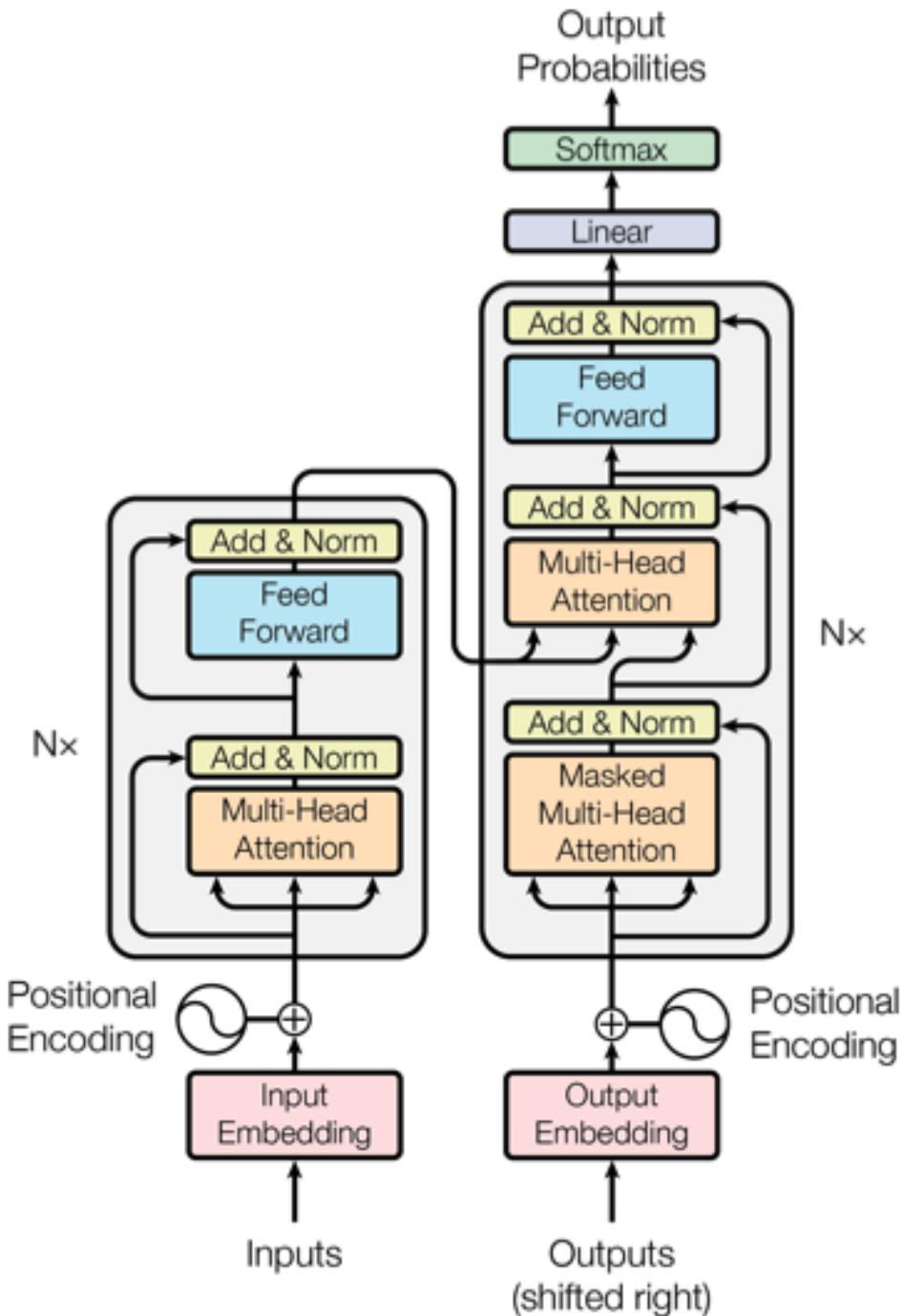


Figure 1: The Transformer - model architecture.

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

L'encoder fa il mapping di una sequenza di input (x_1, \dots, x_n) in una sequenza intermedia z_1, \dots, z_n e poi il decoder, presa in input questa sequenza z genera la sequenza di output y_1, \dots, y_n . Il modello è auto regressive e consuma l'output precedentemente generato come input aggiuntivo nel momento in cui genera il prossimo testo. Il transformer segue in generale una architettura che utilizza sia nel decoder che nell'encoder dei layer fully connected con self-attention.

Vediamo i dettagli dell'architettura:

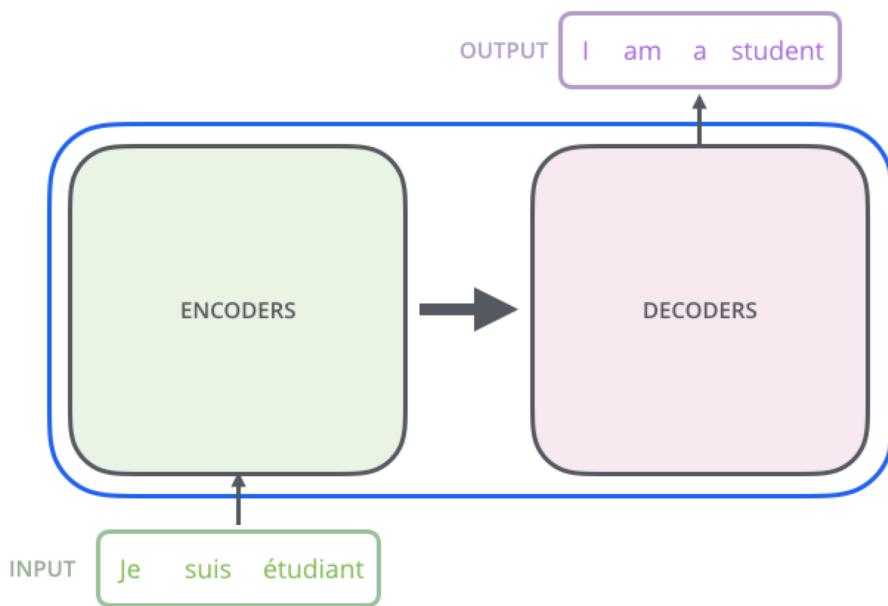
- Encoder: è composto da uno stack di 6 layer identici composti da due sotto layer. Il primo è un layer multi head con self-attention e il secondo è un fully connected feed-forward network. L'output di ognuno di questi due layer è un layer di normalizzazione.
- Decoder: anche questo è composto da uno stack di 6 layer identici, ciascuno di questi è diviso in tre parti, due sono quelle che abbiamo anche nell'encoder e una è invece quella che calcola la multi-head attention rispetto all'output dello stack di encoder.

19.2.1 Transformer Model

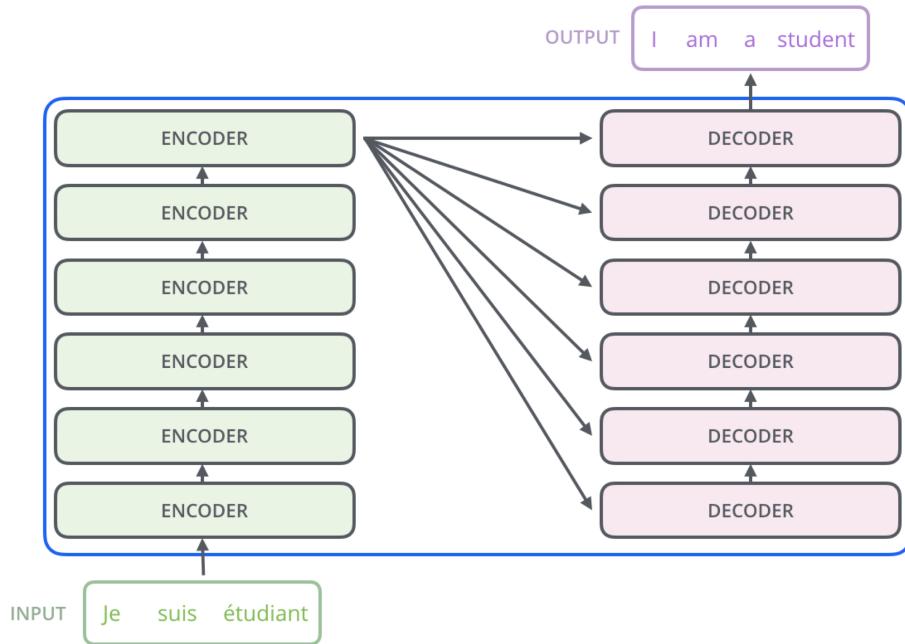
Il transformer prende in input una frase che vogliamo tradurre e la trasforma nella sequenza di output. Nell'encoder prendiamo la sequenza (x_1, \dots, x_n) e la rappresentiamo con una rappresentazione intermedia $z = z_1, \dots, z_n$, il decoder poi genera la sequenza di output y_1, \dots, y_n un elemento alla volta. Ad ogni step il decoder è auto regressive perchè prende come input la rappresentazione intermedia, in pratica utilizza il suo stesso output (l'output dell'encoder) per produrre l'output complessivo.



Entrando in questa black box che è il Transformer troviamo due parti principali che sono l'encoder e il decoder, sappiamo che i dati originali passano nell'encoder e poi l'output che produce l'encoder finisce nel decoder che produce l'output del modello.

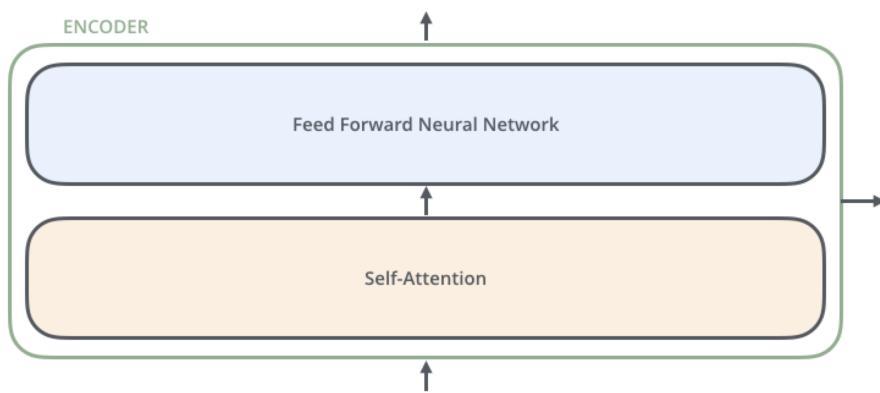


Esplodendo ancora di più l'architettura vediamo che all'interno dell'encoder e del decoder abbiamo vari layer, il paper [3] ne mette 6, sia nell'encoder sia nel decoder.



Non c'è nulla di magico nella scelta del numero 6 però è importante che decoder ed encoder abbiano lo stesso numero di layer. L'ultimo layer dell'encoder è quello che produce l'output, questo output deve essere poi dato in pasto a tutti i layer del decoder, non solamente al primo.

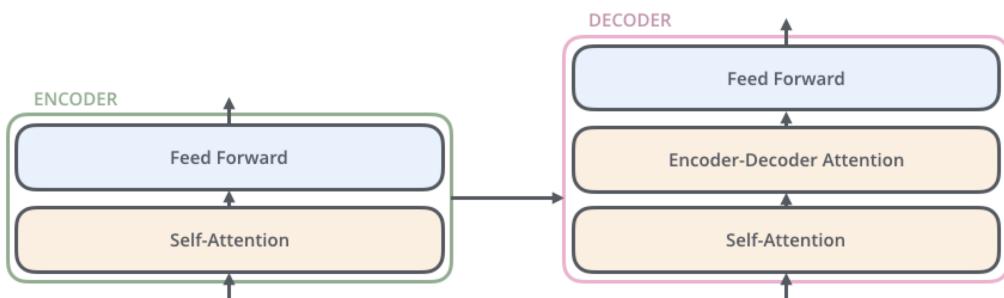
I 6 layer degli encoders sono tutti uguali tra di loro, non condividono dei pesi tra loro e ognuno viene suddiviso in due sublayers.



Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

Il primo di questi sublayers è il Self-Attention Layer che ci permette di considerare le altre parole nel testo mentre ne stiamo codificando una specifica, (vedremo più avanti una rappresentazione dettagliata del self attention). Non si tratta di un meccanismo simile a quello delle RNN perchè nella LSTM noi riceviamo i dati in input anche dalle parole precedenti, quindi riceviamo una piccola parte di informazioni, qui con la self attention possiamo conoscere il contesto completo con informazioni riguardanti tutte le relazioni delle parole.

L'output del layer di self attention viene inviato ad un layer con una feed forward neural network, la stessa NN viene applicata in ogni layer dell'encoder.



Il decoder contiene all'interno una struttura che è simile a quella dell'encoder, abbiamo il layer della self attention e del feed forward e poi abbiamo anche un layer intermedio "Encoder-Decoder Attention" che prende in considerazione i suggerimenti dell'encoder e cerca quindi di focalizzarsi sulle parti più rilevanti della frase in quello specifico momento. L'output dei primi due layer del decoder viene combinato insieme e viene dato in pasto al layer finale che è una Feed Forward Neural Network.

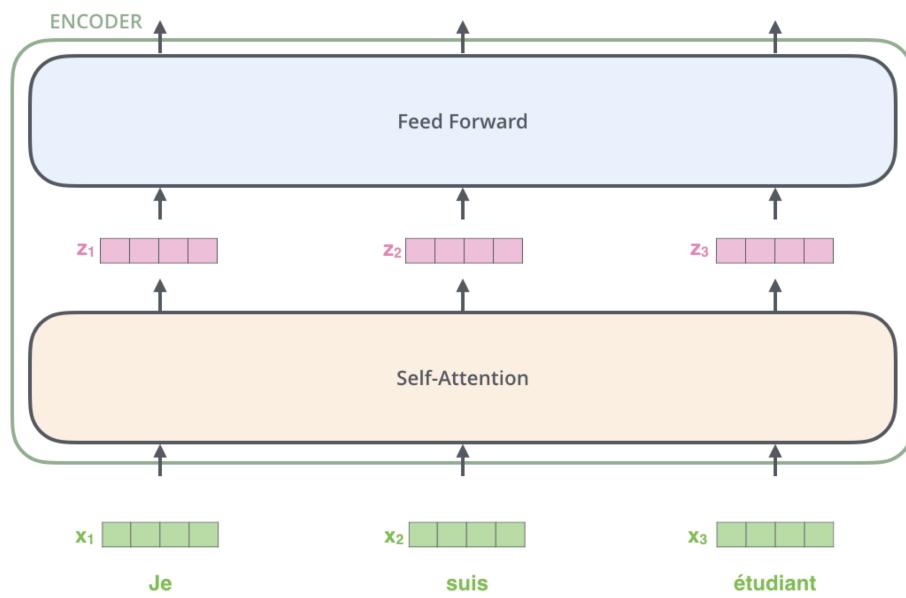
19.2.2 Word Embedding

L'input dell'encoder non è direttamente rappresentato come le stringhe da tradurre ma sono word embedding ovvero vettori.

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

L'astrazione che viene utilizzata è che l'encoder riceve una lista di vettori (embedding) di dimensione 512, nel primo layer dell'encoder questo è direttamente l'insieme degli embedding delle parole mentre nei layer successivi saranno dei vettori che sono gli output dei layer precedenti. La dimensione della lista degli embedding che vengono dati in input è un iperparametro che possiamo settare, in pratica potrebbe essere la lunghezza della frase più lunga all'interno del nostro training set.

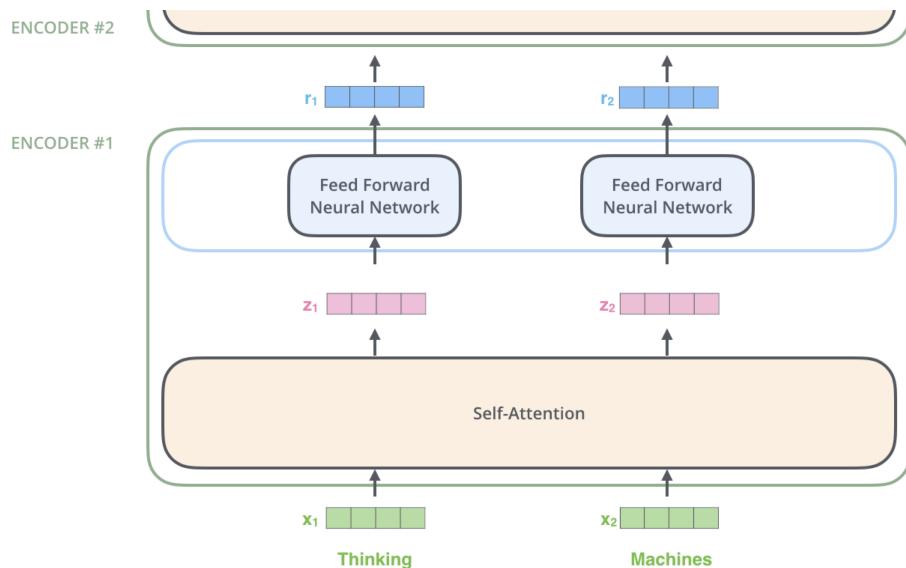
Dopo che sono state inserite sotto forma di embedding, le parole viaggiano all'interno dei vari layer dell'encoder seguendo ognuna il suo percorso.



Questo "seguire ognuno i suo percorso" è già una proprietà chiave del trasformer, nei layer di self attention abbiamo relazioni tra i vari percorsi che stiamo considerando mentre nel layer feed forward le dipendenze non ci sono più e per questa ragione possiamo eseguire questa seconda parte dell'encoder in parallelo ad esempio su differenti GPU. Quindi graficamente questo lo possiamo rappresentare in questo modo, con più Feed Forward NN che lavorano in parallelo sui vari output del self attention

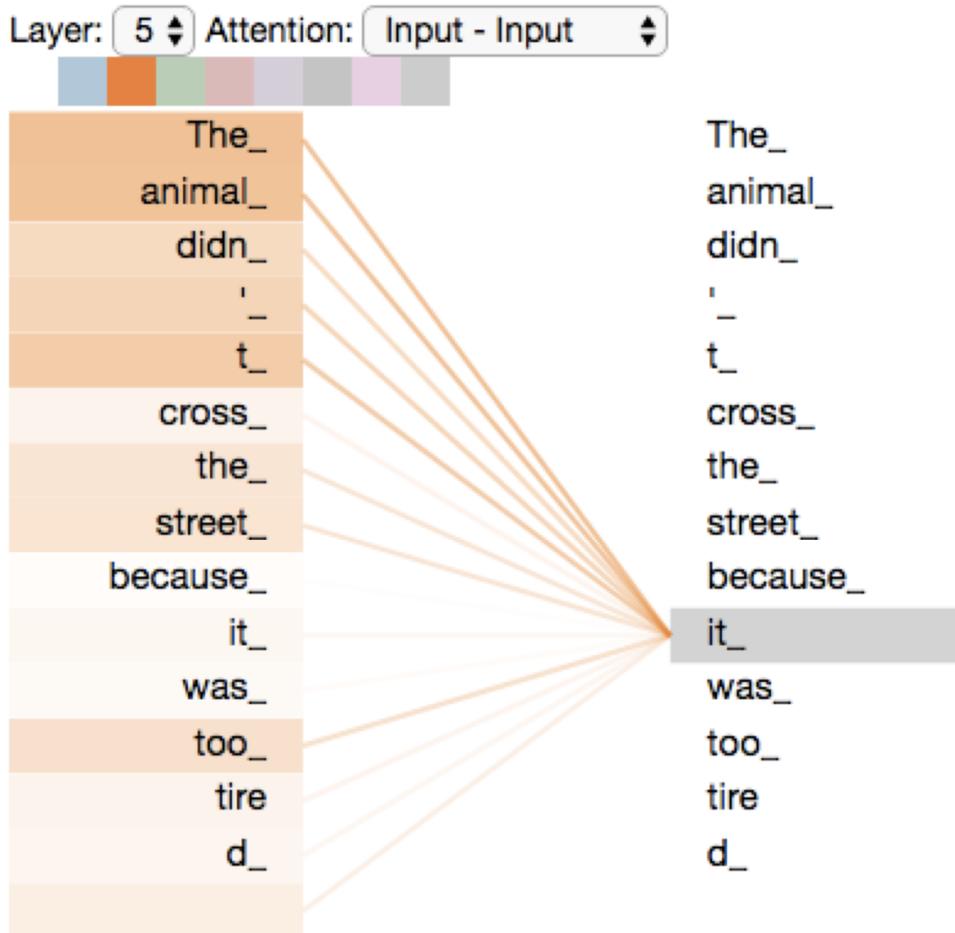
Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

(che a loro volta rappresentano le parole in input e che all'interno del self attention layer sono legate tra loro):



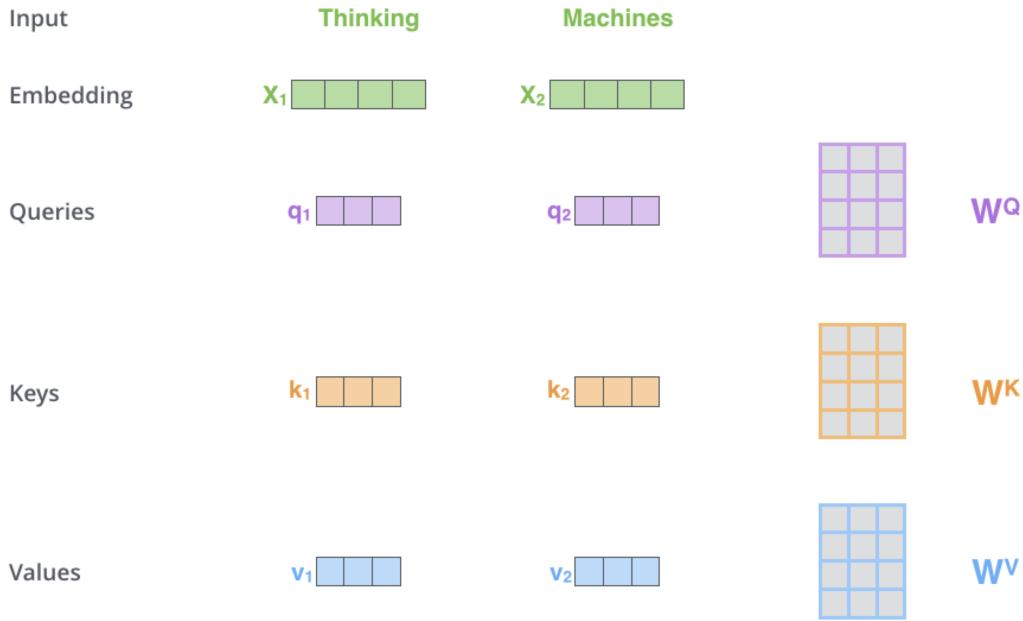
19.3 Self Attention

L'idea della self attention è quella di considerare le informazioni dalle altre parole presenti nella frase che stiamo dando in input cercando di comprendere meglio il significato della parola che stiamo processando. Ad esempio nella seguente immagine stiamo processando la parola "it", la self attention mi permette di controllare tutte le parole presenti nella frase di input e poi per ognuna di queste mi fornisce uno score che mi indica quali sono le parole che sono più rilevanti rispetto a quella che stiamo prendendo in considerazione. In questo esempio abbiamo che "it" è molto legata con "The" e con "animal".



Quindi in pratica quello che abbiamo in output è un vettore con i pesi che ci permettono di capire quali sono le parole più legate alla parola che stiamo considerando.

Vediamo il funzionamento nel dettaglio, partiamo come prima cosa dal calcolo della self attention.



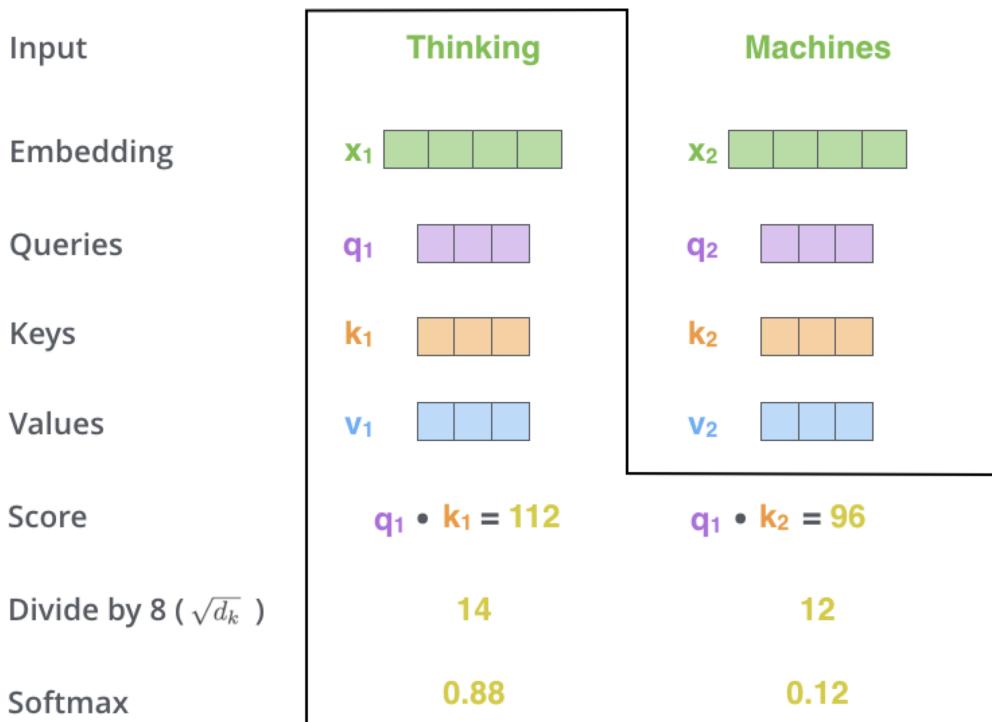
Supponiamo di avere come input del nostro layer di self attention due parole che vengono trasformate in embedding di dimensione 512 (basandoci sul paper). La prima cosa che si fa è creare 3 vettori partendo dai vettori di embedding che abbiamo in input, i tre vettori sono Query vector, Key Vector e Value Vector. Per ottenere questi vettori prendiamo delle matrici W che abbiamo creato durante il training e le moltiplichiamo per il word embedding che abbiamo in input. Il risultato sono questi tre vettori che sono di dimensione minore rispetto agli embedding, questi hanno lunghezza 64 contro 512, non devono necessariamente essere più piccoli degli embedding originali, nel paper lo fanno.

Questi tre vettori che abbiamo ottenuto, Queries, Keys e Values sono delle astrazioni e sono utili per calcolare e ragionare riguardo all'attention.

Il secondo step che abbiamo per il calcolo della attention consiste nel calcolo di uno score. Questo score ci permette di capire come le parole sono legate tra loro. Lo score lo calcoliamo per ogni parola che abbiamo all'interno della nostra frase rispetto a tutte le altre parole della frase. Il calcolo consiste in una moltiplicazione del vettore Queries della

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

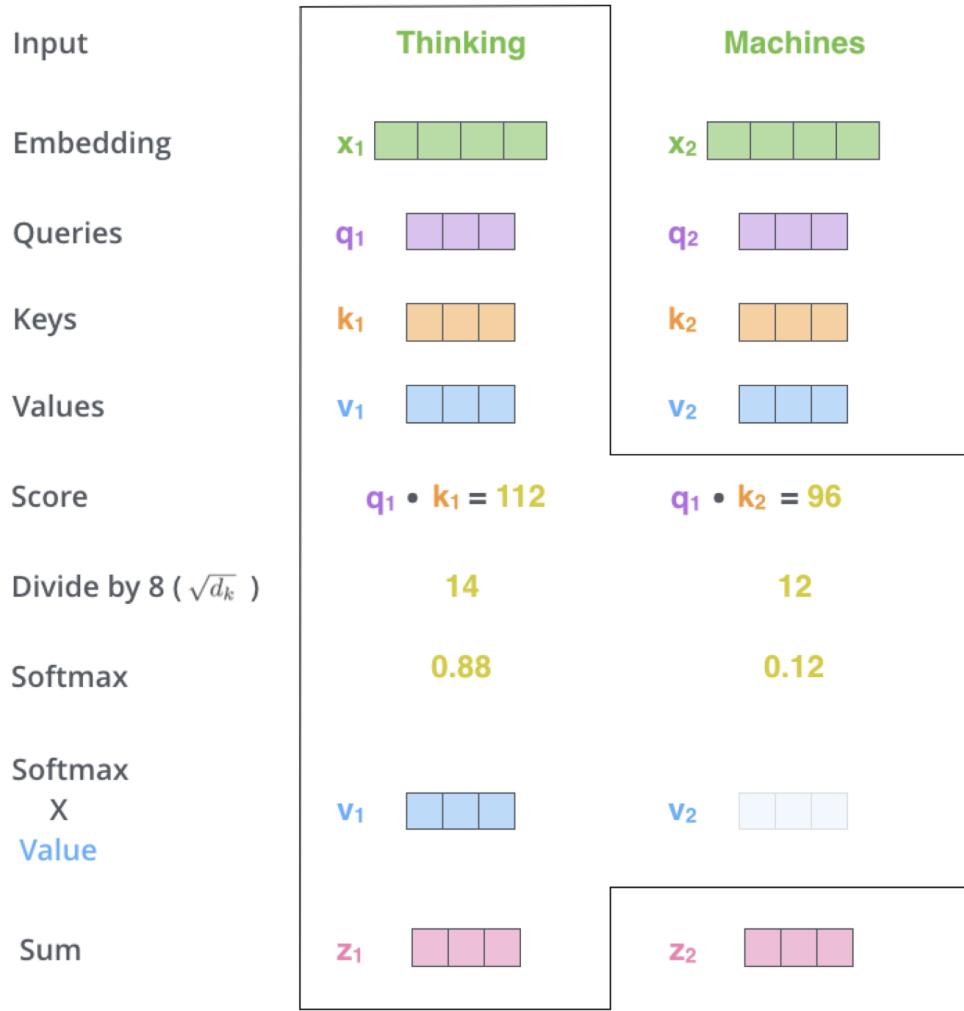
parola su cui siamo focalizzati con il vettore Keys di tutte le altre parole. Ad esempio se consideriamo la parola "Thinking" per calcolare lo score dobbiamo calcolare $q_1 * k_1$ per avere lo score rispetto alla stessa parola "Thinking" e poi $q_1 * k_2$ per avere lo score rispetto a Machines. Quello che otteniamo da questo calcolo dello score consiste in un valore numerico che mi indica quando è rilevante la parola rispetto alle altre.



Dopo il calcolo dello score possiamo fare una normalizzazione, in questo caso si divide lo score per 8 (8 è stato scelto perchè è la radice quadrata della lunghezza del vettore Queries). Per fare in modo che questo score venga trasformato in una probabilità dobbiamo applicare la softmax allo score normalizzato, in questo caso abbiamo che la parola thinking è più rilevante rispetto a se stessa (88%) rispetto all'altra (12%).

L'ultimo step del calcolo della self attention consiste nella produzione dell'output, quello che si fa qua è andare a fare una moltiplicazione tra

l'array Values e gli output della softmax e poi si sommano i risultati che abbiamo ottenuto, in questo modo otteniamo un output z_i per ognuna delle parole che abbiamo all'interno della nostra frase.

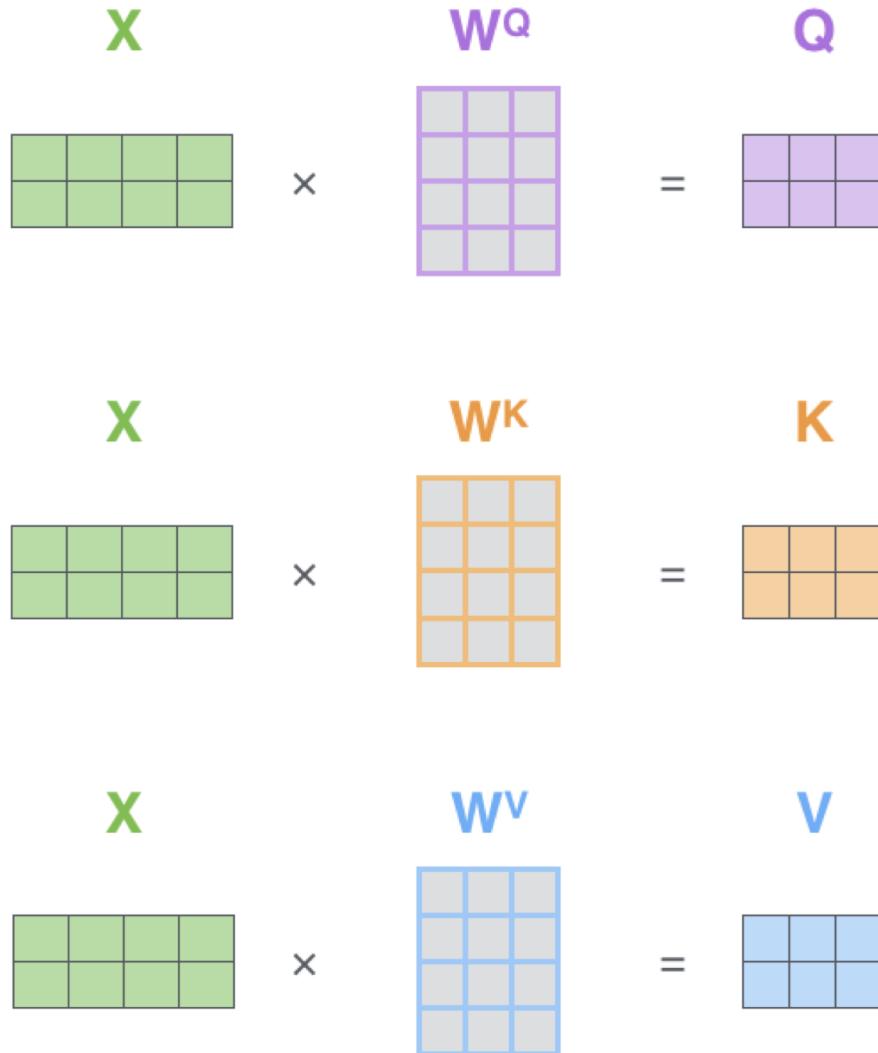


Come possiamo vedere dall'immagine sopra abbiamo un output per ogni colonna, quindi un output per ognuna delle parole che abbiamo nella frase e ognuna di queste colonne è indipendente dalle altre eccetto che per la parte della softmax. I vettori z_i che abbiamo ottenuto sono ora quelli che manderemo come input alla feed forward neural network.

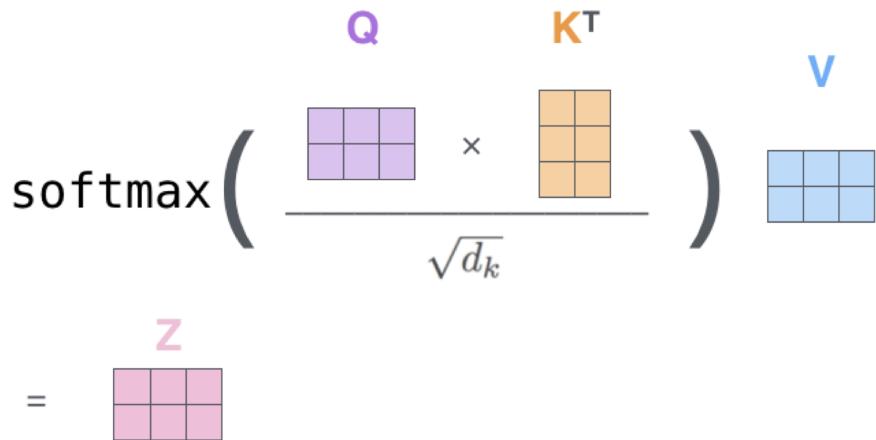
Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

Fino ad ora in questa operazione di calcolo della Self attention abbiamo sempre parlato di moltiplicazioni tra vettori, per fare in modo di lavorare in parallelo e quindi di avere delle migliori performances abbiamo una alternativa, lavorare con le matrici.

Quello che si fa è calcolare come abbiamo fatto in precedenza Queries, Values e Keys ma in questo caso lo facciamo prendendo i due embedding e inserendoli all'interno di una matrice X che poi viene moltiplicata per le varie matrici con i pesi. In questo modo riusciamo a creare non dei vettori di Queries, Keys e Values ma delle matrici.



Avere queste matrici ci consente di parallelizzare il lavoro che dobbiamo svolgere tra le varie GPU ma non solo, quello che prima facevamo in vari step ora tramite le moltiplicazioni di matrici diventa un unico step che mi porta alla generazione dell'output del layer di self attention.



19.3.1 Perchè usare la Self Attention?

La Self attention ci consente di mantenere una conoscenza riguardante sequenze di testo molto lunghe e soprattutto di mantenere le dipendenze tra le parti del testo molto lontane tra loro. Il layer di self attention connette tutte queste posizioni utilizzando un numero costante di operazioni sequenziali, quindi abbiamo che il costo delle operazioni sequenziali è $O(1)$ mentre con la RNN è $O(N)$. Quindi la Self-Attention è migliore dal punto di vista computazionale.

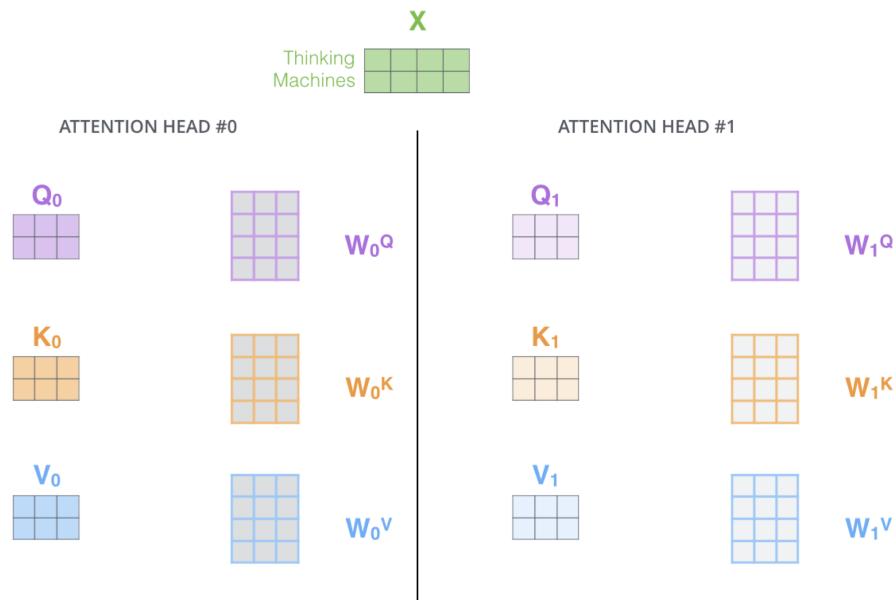
19.3.2 Multi Head Attention

Quella che abbiamo visto fino ad ora è la più semplice computazione che possiamo avere per l'attention, nel paper però ne viene indicata un'altra che viene chiamata Multi-Head. Questa Multi-Head Attention migliora le performances in due modi:

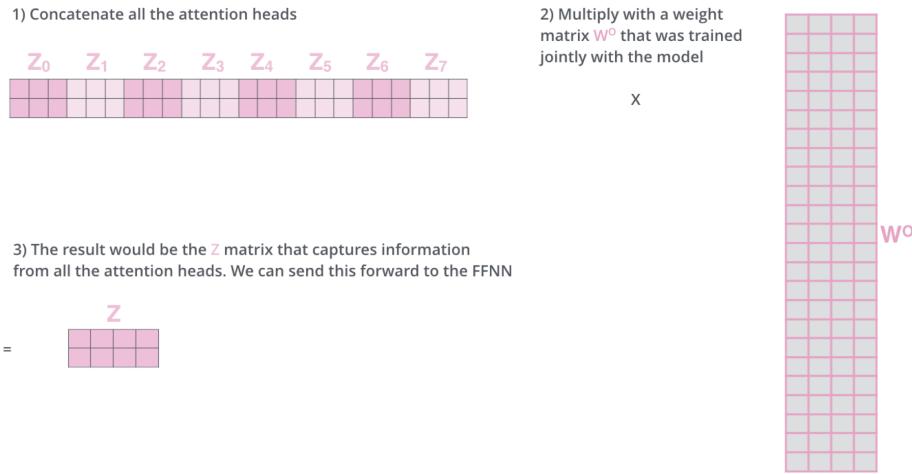
- Permette di espandere la capacità del modello di focalizzarsi su differenti posizioni della frase in input, nel nostro primo esempio con la Self Attention abbiamo visto che l'output z_1 era dominato

dalla stessa parola su cui eravamo focalizzati e questo potrebbe non essere troppo utile.

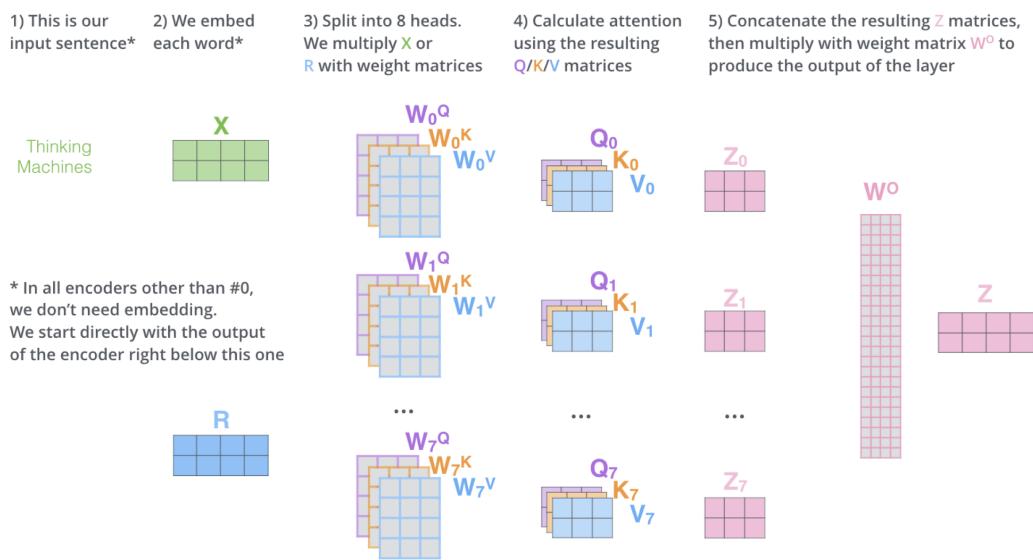
- Aggiungiamo all'attention layer varie representation subspaces ovvero utilizziamo non un singolo set di Query/Value/Key ma vari set. Questi set sono inizialmente random, poi dopo il training ci permettono di proiettare l'embedding in input in un differente subspace. Nel caso del paper crea 8 differenti Attention Head e quindi 8 differenti matrici per la generazione delle z :



Il problema che abbiamo con questo metodo è che noi nella classica self attention producevamo un singolo output z per ognuno degli embedding in input e questo andava in input alla Feed Forward NN, qua avremmo 8 z da dare in input e non va bene. La soluzione quindi è concatenare tutte le z che abbiamo e poi moltiplicarle per una matrice di pesi W_0 in modo da ottenere una singola z finale da passare come input alla Feed Forward NN.

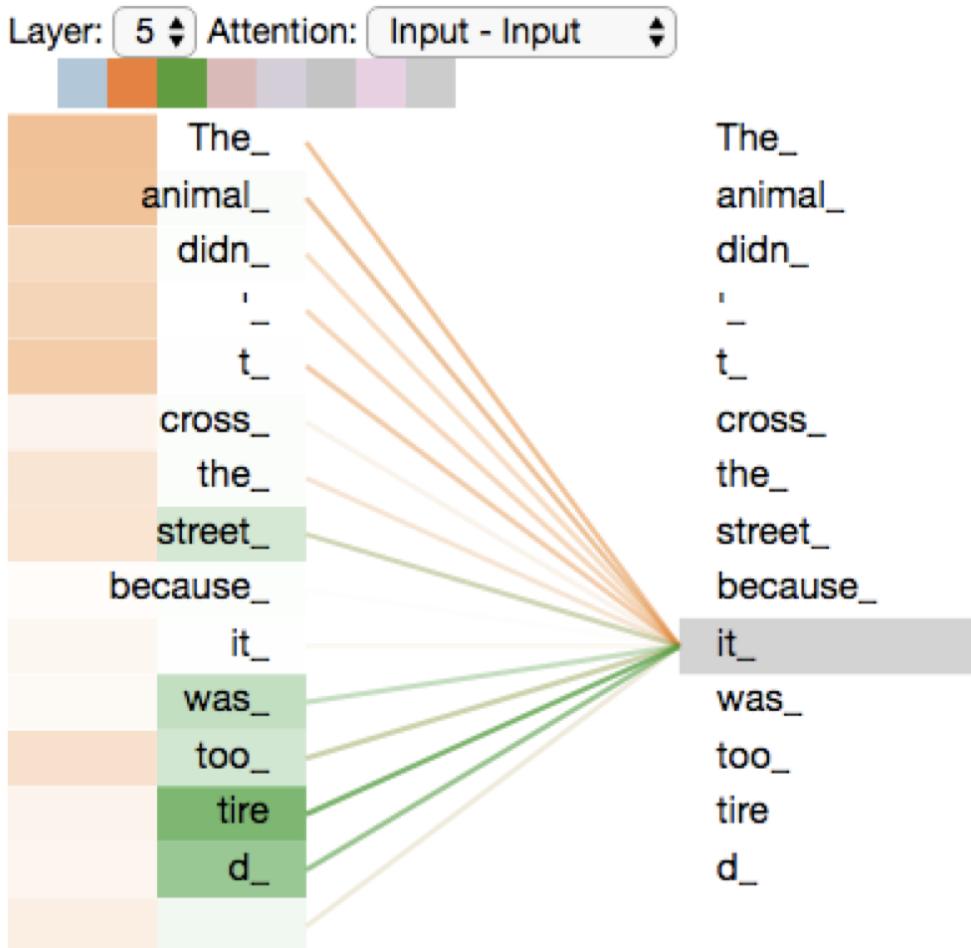


Questo quindi è il Multi Head Attention, possiamo vederlo tutto insieme in un unico schema:



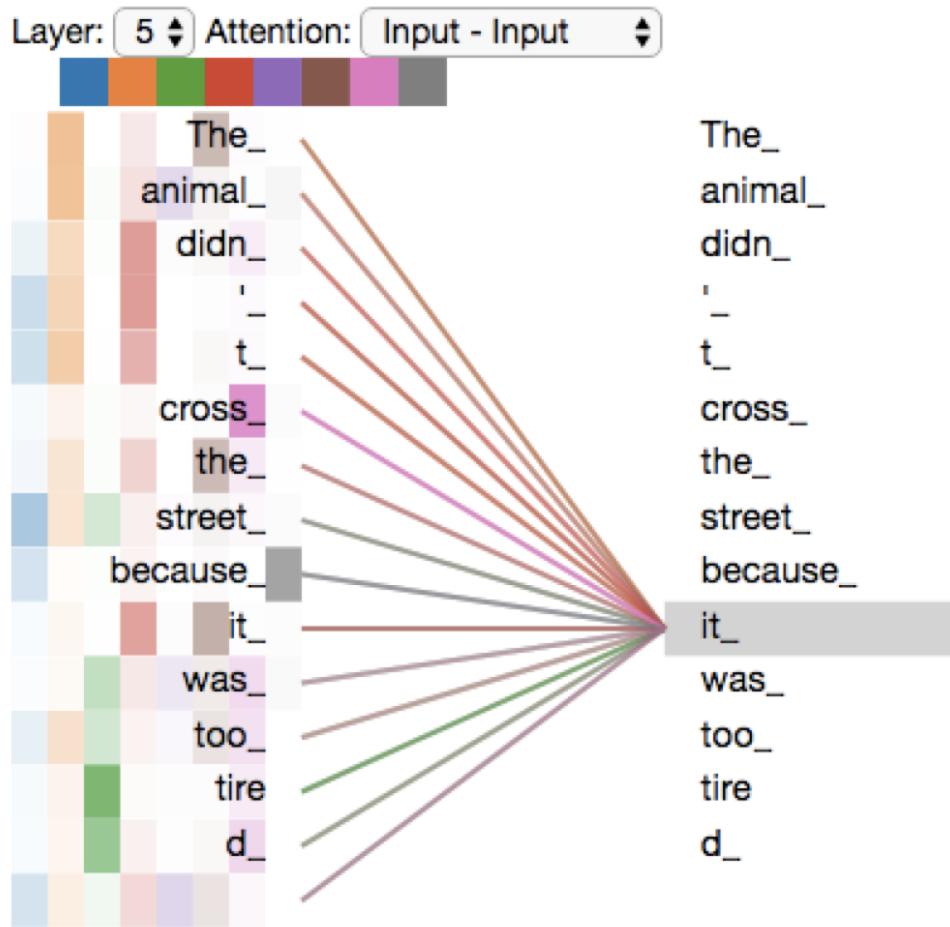
Ora lo stesso esempio di prima con l'output dell'attention layer lo possiamo vedere in modo differente:

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.



In precedenza avevamo un'unica Head e quindi la nostra parola su cui eravamo focalizzati "it" era legata in modo particolare ad animal, in questo secondo esempio abbiamo invece 2 Heads e quindi abbiamo che un attention head è focalizzato su "The animal" come prima, l'altro su "Tired". Le parole all'interno di questa tabella sono tutte con l'underscore perché abbiamo eseguito una tokenizzazione e le parole che hanno la stessa struttura vengono "tagliate" in modo da ridurre il numero di parole nel dizionario. Chiaramente se utilizziamo le 8 Heads che vengono utilizzate nel paper originale dell'attention abbiamo una situazione che

diventa più complessa da comprendere, per ogni parola vengono infatte trovate le 8 altre parole che sono maggiormente legate:

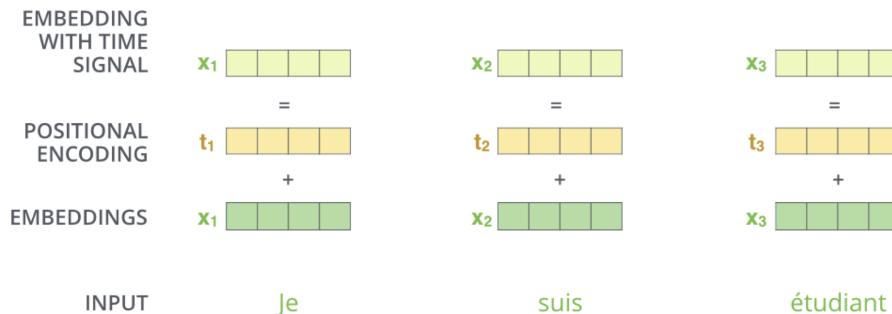


19.4 Positional Encoding

Il modello di cui abbiamo parlato fino ad ora non tiene conto dell'ordine delle parole nella sequenza di input e questo può essere un problema. Una soluzione a questo problema consiste nel gestire anche la posizione delle parole, per farlo introduciamo un nuovo vettore che segue un pat-

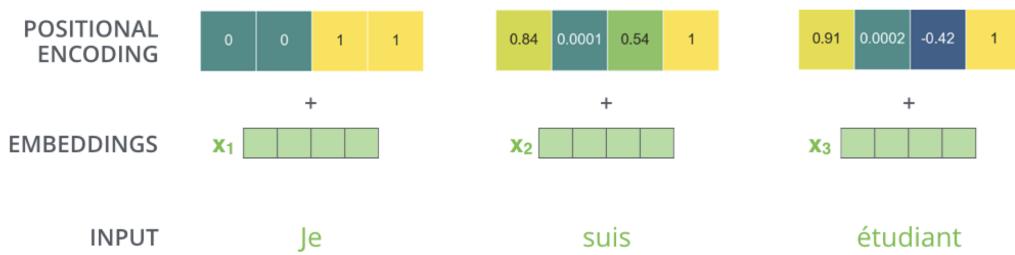
Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

tern che viene "imparato" dal modello durante il training e che aiuta a determinare la posizione di ogni parola o la distanza tra le varie parole presenti all'interno della sequenza.



Questo nuovo vettore di Positional Encoding viene sommato al vettore degli embedding in modo da produrre un vettore finale che contiene non solo gli embedding ma anche l'encoding delle posizioni delle parole all'interno delle frasi.

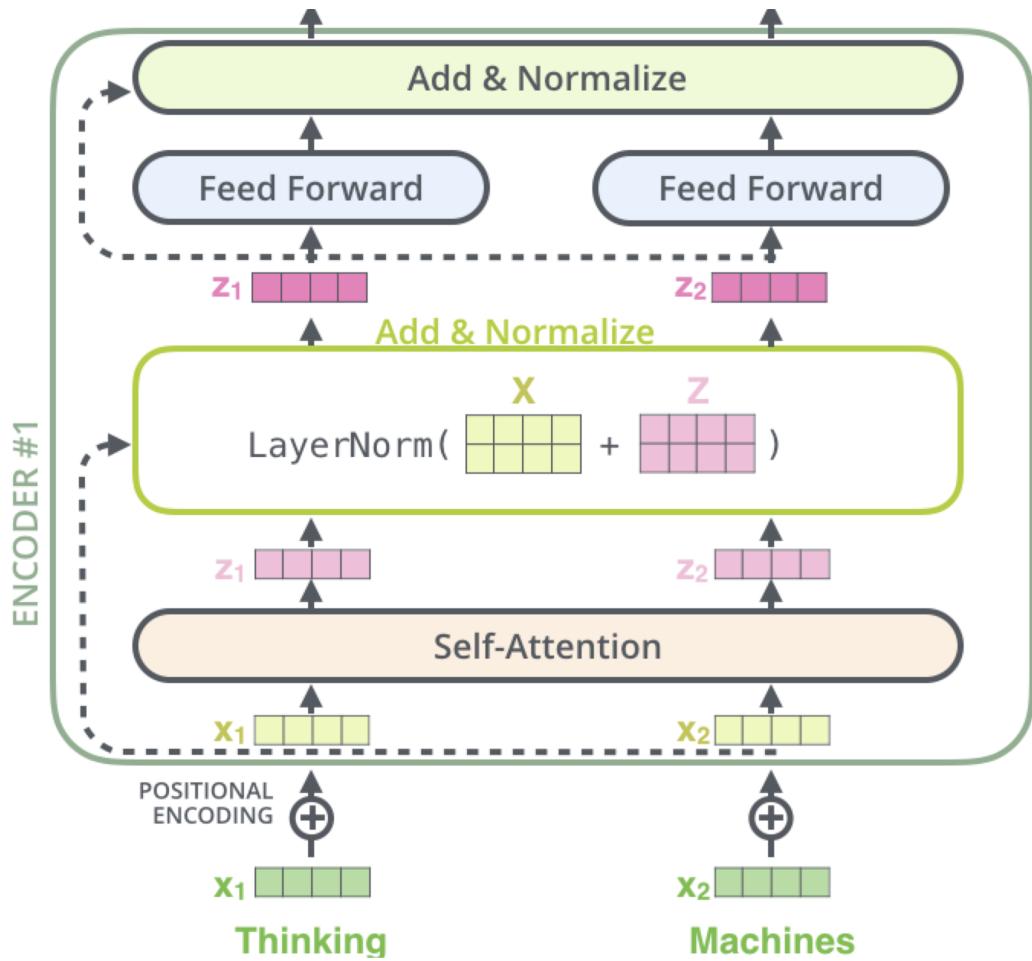
Consideriamo il caso in cui abbiamo degli embedding di dimensione 4, come nella immagine, all'interno dei vettori del Positional Encoding abbiamo dei valori che dipendono da una formula descritta nel paper.



Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

19.5 Residual Connections e Layer Normalization

Continuando con la struttura dell'encoder, abbiamo visto che all'interno abbiamo anche un layer di normalizzazione.



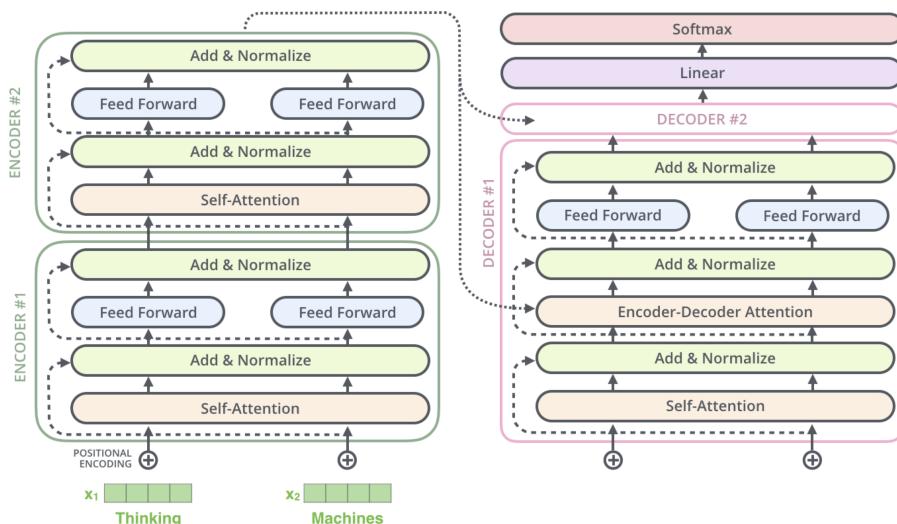
All'interno dell'encoder abbiamo una struttura come quella dell'immagine sopra, abbiamo la residual connections che è il collegamento tra il vettore in input e il layer di normalizzazione. All'interno del layer di normalizzazione viene eseguita la somma tra la matrice con gli embedding di

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

input e la matrice Z che è l'output del layer di Self Attention. Poi con il risultato della somma eseguiamo la normalizzazione e otteniamo il reale output del layer di self attention che daremo poi in input al layer di feed forward neural network.

Il funzionamento è il medesimo anche per il layer di feed forward, attorno ad esso abbiamo il collegamento del residual e poi abbiamo la somma delle matrici e la normalizzazione per produrre l'output.

Possiamo avere una vista complessiva del transformer per sottolineare che anche nel decoder avremo una struttura identica con residual connections e normalizzazione.



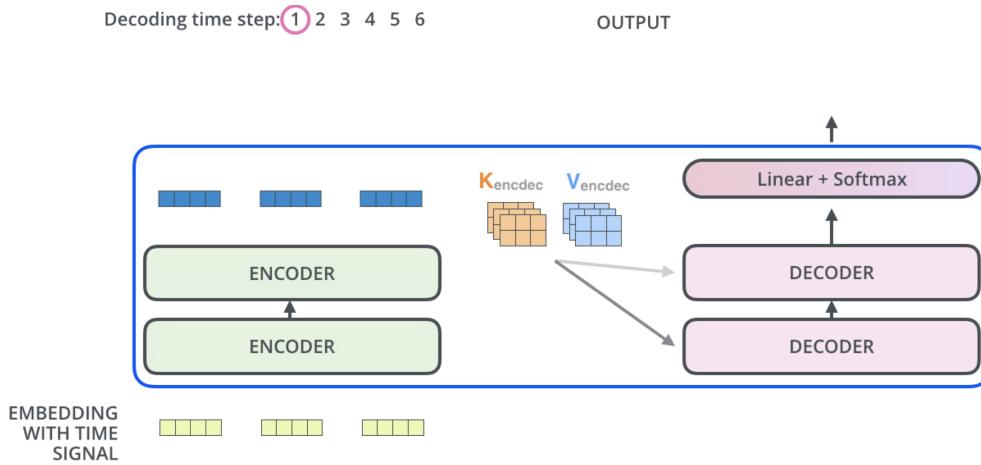
Nel decoder in alto abbiamo anche la softmax, questa viene utilizzata per produrre una distribuzione di probabilità dell'output in modo poi da calcolare l'argmax scegliendo l'output più probabile.

19.6 Il decoder

Ora che abbiamo visto il funzionamento dell'Encoder possiamo passare al funzionamento del Decoder.

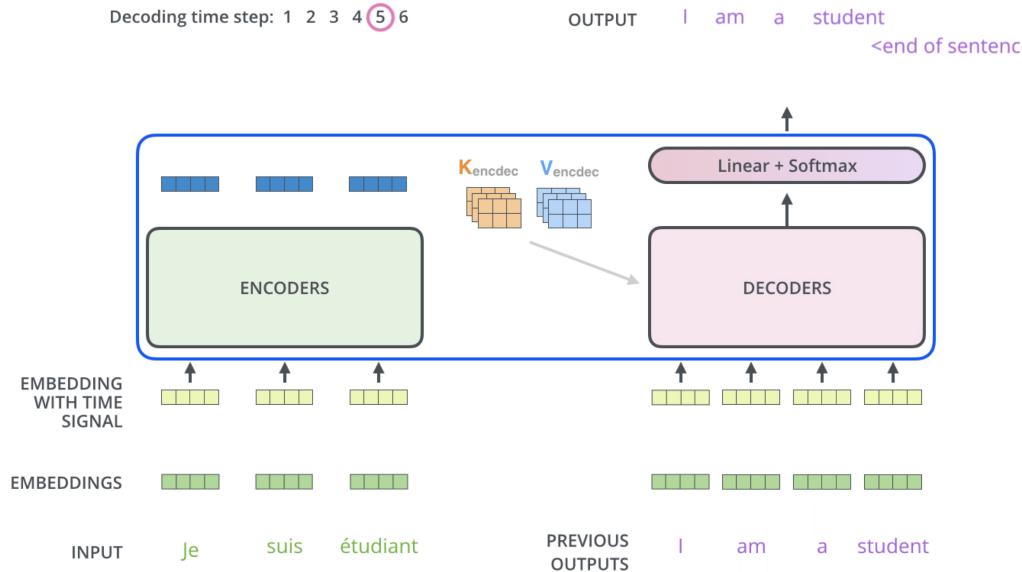
Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

L'output che viene prodotto dall'encoder non viene direttamente dato in input ai decoder così come è, prima viene trasformato in un set di attention vectors che sono K e V.



Questi dati che arrivano ai decoder vengono utilizzati all'interno del layer "Encoder-Decoder attention" che permette al decoder di focalizzarsi su alcuni punti in particolare della sequenza di input.

Oltre all'input dagli encoder, il decoder riceve anche un input dall'output precedente del decoder. Quindi in pratica nell'immagine sotto abbiamo che viene ricevuto l'input dall'encoder ma anche l'output relativo ad "I" che è stato l'output del primo time step del Decoder. Il decoder prende questi due input e produce in output "am" che è l'output del secondo time step.



Anche all'interno dei decoder abbiamo un layer di self attention che però funziona diversamente rispetto al layer di self attention dell'encoder. Questo layer infatti è autorizzato a considerare solamente le parole precedenti a quelle che stiamo considerando nella sequenza di output e non le successive, questo lo facciamo mascherando le posizioni successive settandole a meno infinito. Il layer di encoder-decoder attention funziona come un MultiHeaded self-attention tranne per il fatto che crea la matrice Query dal layer inferiore e prende invece le matrici Keys e Values dall'output dell'encoder.

19.6.1 Output del Decoder

Lo stack di decoder di cui abbiamo parlato fino ad ora produce un vettore di float, come possiamo fare a trasformarli in parole? Alla fine dello stack di decoder abbiamo due altri layer:

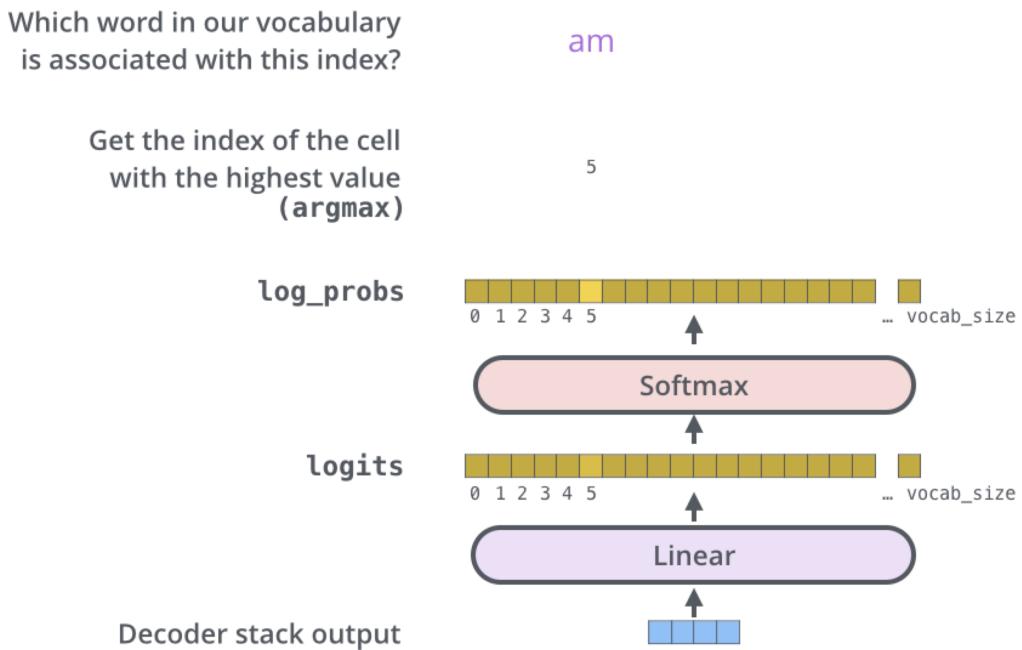
- Linear Layer: è una fully connected Neural Network che proietta i vettori prodotti dallo stack di decoder in un vettore molto più

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

grande chiamato logits vector. Assumiamo che il nostro modello abbiamo un dizionario di 10.000 parole inglese, l'output del linear layer sarà un vettore di 10.000 elementi, ognuno relativo ad una parola e conterrà lo score di ognuna di queste parole.

- Il layer finale è la softmax che prende in input il logits vector e lo converte in probabilità, la somma degli elementi presenti nel vettore output del Transformer dovrà essere 1 ovviamente. Di questo vettore di probabilità prendiamo il valore maggiore e scegliamo quindi la parola corrispondente producendo il nostro output.

Possiamo rappresentare in una immagine tutto questo:



Come abbiamo già visto nella Neural Machine Translation e in particolare nel momento in cui facciamo il decoding della sequenza producendo l'output, considerare solamente il massimo dell'output e tramite questo scegliere quindi la parola da dare in output non è una scelta ottima perchè avremmo un ottimo locale ma non globale. Una aggiunta da

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

integrare nel modello consiste nell'utilizzare la Beam Search per esplorare le possibili alternative che abbiamo, quindi non scegliamo semplicemente la parola con la probabilità più alta ma ne scegliamo varie e poi da queste continuiamo l'espansione per fare in modo di scegliere un output che sia il migliore ma globalmente.

19.7 Training

Ora che abbiamo visto come funziona il forward pass del nostro modello vediamo un po' come funziona il backward pass e come viene eseguito il training. Durante il training il nostro modello funziona esattamente come nel forward pass, la sola differenza è che in questo caso il nostro output è già labellato e quindi possiamo eseguire un confronto con i dati che mandiamo in output.

Per capire meglio consideriamo un esempio e assumiamo di avere un vocabolario con 6 parole, ad ogni parola associamo un indice e creiamo quindi il nostro dizionario.

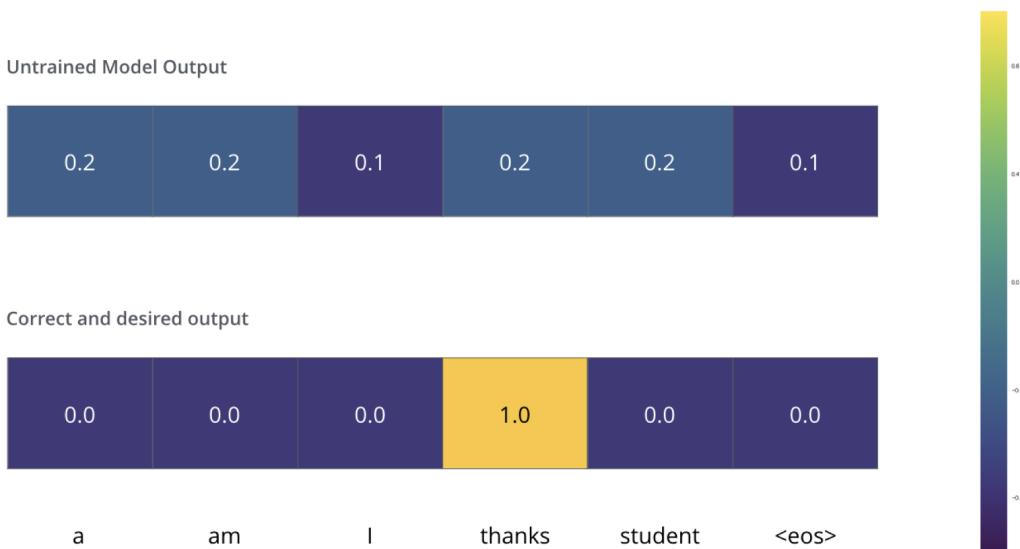
Output Vocabulary						
WORD	a	am	I	thanks	student	<eos>
INDEX	0	1	2	3	4	5

Quando vogliamo indicare una certa parola presente all'interno del nostro dizionario possiamo utilizzare il one-hot encoding, creiamo un vettore della stessa lunghezza del dizionario e poi assegnamo 1 nella posizione della parola che stiamo considerando e 0 in tutte le altre posizioni.

One-hot encoding of the word "am"



Supponiamo di produrre il primo output del nostro modello, dato che tutti i pesi presenti all'interno del modello sono inizializzati random anche l'output sarà del tutto random e non sarà simile al nostro output corretto (stiamo traducendo grazie in thanks in questo caso).



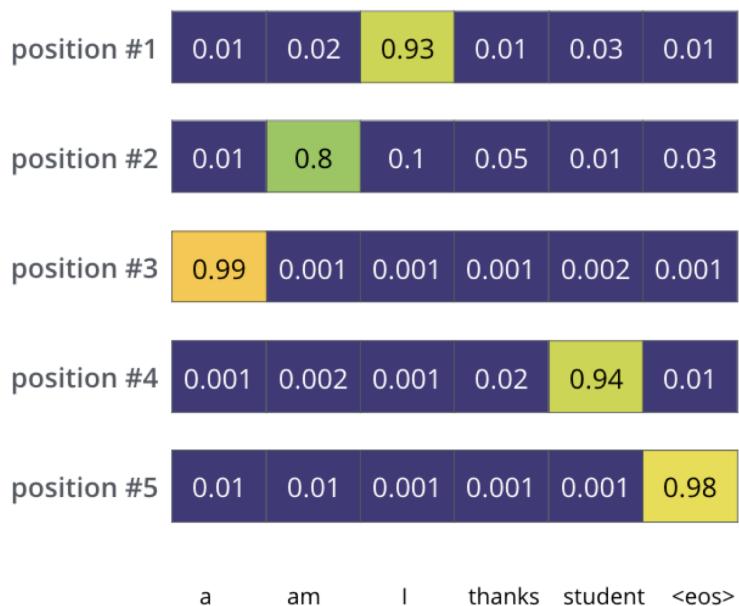
Quello che possiamo fare in questo caso è comparare le due probability distributions ed eseguire poi la backpropagation. Per fare il confronto tra le due probability distribution possiamo utilizzare la cross entropy e in pratica il gradiente di questa cross entropy è la differenza tra l'expected distribution e la predicted distribution.

Il nostro obiettivo, dopo aver fatto la backpropagation è ottenere un modello che produce un output di questo genere ovvero una probability distribution per ogni parola in output ad ogni iterazione.

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

Trained Model Outputs

Output Vocabulary: a am I thanks student <eos>



Idealmente scegliamo il valore massimo di ognuna di queste probability distribution (greedy decoding) oppure, come abbiamo visto, utilizziamo la beam search (il numero di elementi nella frontiera della beam search è a sua volta un iperparametro).

19.8 Conclusione - Machine Translation

È interessante capire quanto il Transformer funziona bene rispetto ad esempio al Sequence2Sequence di cui abbiamo parlato in precedenza.

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [18]	23.75			
Deep-Att + PosUnk [39]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [38]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [9]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [32]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [39]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [38]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [9]	26.36	41.29	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	$3.3 \cdot 10^{18}$	
Transformer (big)	28.4	41.8	$2.3 \cdot 10^{19}$	

All'interno di questa tabella troviamo il BLEU score che otteniamo con i due modelli e abbiamo già qua un miglioramento in termini di performance del Transformer rispetto agli altri modelli. Nella tabella poi abbiamo anche il costo per eseguire il training, vediamo che il Transformer riduce la complessità in termini di tempo perchè sfrutta la parallelizzazione nell'encoder mentre invece gli altri modelli utilizzando le RNN lavorano in modo sequenziale. Dalla tabella vediamo anche che è più complessa la traduzione dall'inglese al tedesco rispetto a quella dall'inglese al francese.

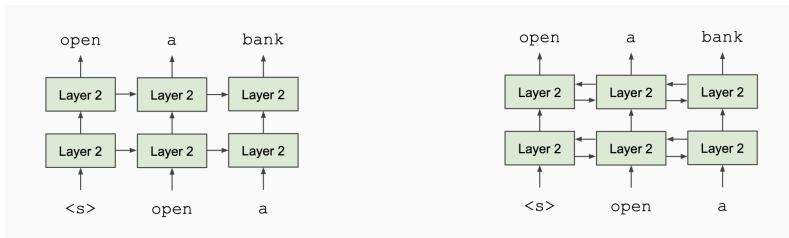
19.9 Bert

Dopo aver utilizzato il transformer per la traduzione si è pensato di utilizzarlo anche in altri campi differenti dalla Machine Translation.

19.9.1 Problemi con i modelli precedenti

I language model che abbiamo visto fino ad ora utilizzano solamente il contesto a sinistra o a destra ma in realtà la comprensione del linguaggio dovrebbe essere bidirezionale. Perchè solo destra o sinistra?

- Perchè abbiamo bisogno di avere una direzionalità per creare una probability distribution, in molti casi questo non ci interessa perchè per il task che stiamo considerando non è necessario.
- Le parole possono vedere se stesse in un encoder bidirezionale.



Abbiamo utilizzato il transformer nel campo della Machine Translation e abbiamo usato gli embedding per fornire una rappresentazione delle parole sotto forma di vettori. Per alcuni task questa rappresentazione è utile ma per task in cui abbiamo bisogno del senso della parola non è per niente utile. Il significato della parola infatti non è un qualcosa di direttamente collegato alla parola ma dipende dal contesto in cui la parola è stata utilizzata.

19.9.2 Una possibile soluzione

La soluzione che è stata pensata per risolvere i problemi che si avevano con i modelli precedenti consiste nell'utilizzare un "Masked" Language Model. Invece di prevedere solamente la prossima parola il language model va a nascondere un certo numero di parole della frase in input e poi esegue la previsione anche della parola nascosta. Tipicamente invece di considerare una singola parola alla volta consideriamo varie posizioni in una frase ed eliminiamo una parola sostituendola con una "Maschera". Tipicamente il numero di parole che sostituiamo corrisponde al 15% delle parole delle frasi. Questo viene chiamato masked language model,

l'obiettivo è di trovare il numero più alto di parole all'interno del contesto.

Quando scegliamo la quantità di parole da mascherare bisogna stare attenti perchè non possiamo mascherarle poche, altrimenti avremmo troppo training da effettuare ma neanche troppe perchè altrimenti non avremmo abbastanza contesto.

Bert introduce il concetto di Bidirectional Embedding, consideriamo le parole in entrambe le direzioni utilizzando il transformer model. Ogni parola viene considerata in entrambi le direzioni con tutte le altre parole, le informazioni riguardanti la posizione della parola vengono rappresentate esplicitamente.

Quando utilizziamo il Masked Language Model in realtà non viene utilizzata sempre la sostituzione delle parole random con MASK, supponendo di avere il 15% delle parole da prevedere ci sono vari casi:

- Nell'80% dei casi viene sostituita la parola con MASK.
- Nel 10% dei casi viene eseguita una sostituzione della parola con una parola random.
- Nel 10% dei casi manteniamo invece la stessa parola

Il secondo task che svolgiamo è la previsione della prossima frase, questo task ci genera degli esempi positivi e degli esempi negativi. Oltre a fare il learning della prossima possibile parola e della parola mancante in un contesto possiamo anche provare ad eseguire il learning delle relazioni tra le varie frasi. In particolare data una certa frase possiamo cercare di fare una previsione per capire quale sarà la frase successiva a quella che stiamo considerando.

Sentence A = The man went to the store.
Sentence B = He bought a gallon of milk.
Label = IsNextSentence

Sentence A = The man went to the store.
Sentence B = Penguins are flightless.
Label = NotNextSentence

Per eseguire il training in modo da poter fare questo tipo di previsioni consideriamo due frasi e per ogni coppia abbiamo un label che mi indica se le due frasi sono correlate tra loro e possono trovarsi vicino oppure no.

19.9.3 Rappresentazione dell'input

L'input che viene dato in pasto a Bert utilizza il WordPiece embedding. Nel caso del NMT abbiamo milioni di parole all'interno del nostro dizionario ma se poi troviamo una parola che non abbiamo mai visto prima allora vuol dire che avremo dei problemi. Il Wordpiece prova a risolvere questo problema al livello delle subword, le parole vengono suddivise utilizzando la lemmatization, questo consente di ridurre la quantità di parole all'interno del vocabolario che ora diventano 30.000. Si tratta di un modello molto piccolo se confrontato con gli altri, l'obiettivo è quello di avere un bilanciamento tra la composizione delle subword e il costo del grande vocabolario.

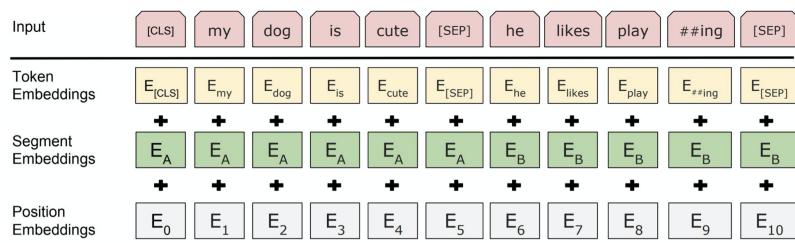
Il risultato della normalizzazione che otteniamo con Wordpiece è che le parole comuni sono nel vocabolario così come sono, le parole che sono meno comuni invece sono divise in modo da creare nuove parole. Questo permette di rivelare la parola originale utilizzando la concatenazione, ad esempio partendo dalla parola *hypatia* otterremo la seguente suddivisione *h##yp##ati##a*.

L'obiettivo è creare un wordpiece Model che permette di creare tutti i wordpieces facendo in modo che il corpus risultato sia minimale nel numero di wordpieces. Se usiamo Bert possiamo utilizzare il Tokenizzatore di Bert per ottenere la rappresentazione tokenizzata delle parole. Il codice che può essere utilizzato è il seguente:

```
from transformer import BertModel, BertTokenizer
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
model = BertModel.from_pretrained('bert-base-uncased')
wps_ids = tokenizer.encode(("Hypatia_was_a_mathematician"))
wordpieces = tokenizer.convert_ids_to_tokens(wps_ids)
['[CLS]', 'h', '#yp', '#ati', '#a', 'was', 'a', 'mathematician', '[SEP]']
```

Nel codice utilizziamo "bert-base-uncased" che è un modello pre-trained da utilizzare per Bert, anche per la tokenizzazione utilizziamo un modello pre-trained ovvero bert-base-uncased. Nell'output abbiamo un token CLS per l'inizio della frase e un token SEP per la fine della frase, poi all'interno della frase abbiamo la parola hypatia che viene suddivisa, le altre invece sono parole originali perchè sono comuni.

Tornando all'input che viene fornito a Bert, abbiamo che questo è formato da varie parole che sono suddivise in tre embedding.



Le tre parti sono:

- Token embedding
- Segment Embedding: permette di distinguere la prima frase dalla seconda
- Position Embedding: per ognuna delle posizioni all'interno delle frasi abbiamo una posizione differente e rappresentiamo con un embedding ciascuna di queste posizioni.

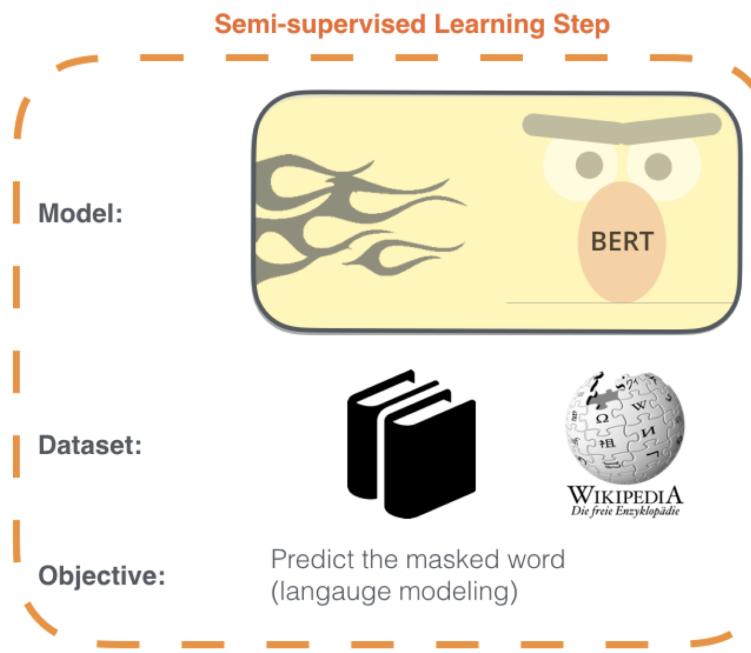
Ogni token che arriva in input a Bert è la somma di questi tre embedding.

19.9.4 Two Step Development

Ora abbiamo visto in che modo i dati devono essere forniti a Bert, ora cosa possiamo fare? Dobbiamo considerare che Bert è formato da due fasi per quel che riguarda il training:

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

- Semi-supervised Learning Step: prima di tutto dobbiamo creare il modello di Bert, abbiamo il nostro dataset di testo preso da Wikipedia e da libri e l'obiettivo è ottenere un modello che sia in grado di predire le Masked Words nel modo corretto. Alla fine quello che otteniamo sono dei pesi per ognuno dei layer del nostro modello di Bert.

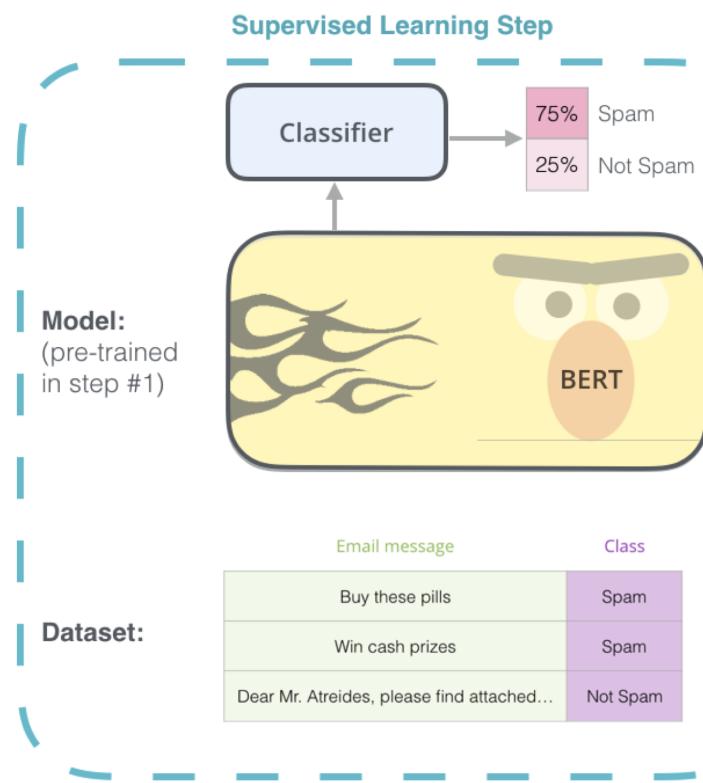


Questa prima fase è come detto, una fase di Semi-supervised Learning e quando abbiamo finito otteniamo un modello pre-trained (allenato su dati non annotati) che possiamo utilizzare per il tuning durante la fase successiva. Di questa prima fase noi non ce ne dobbiamo preoccupare perché abbiamo già modelli pre-trained disponibili.

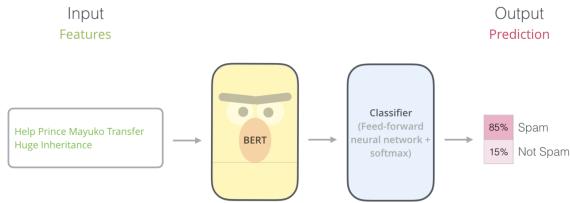
- Supervised Learning Step: Quello di cui ci dobbiamo preoccupare è la seconda fase, qua noi abbiamo il nostro modello pre-trained e lo vogliamo utilizzare per classificare delle frasi per capire se

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

sono spam o no (abbiamo dei dati labellati). Possiamo utilizzare un classificatore binario con un layer di softmax per produrre una distribuzione di probabilità delle due classi. Il nostro compito sarà quello di implementare il classificatore.



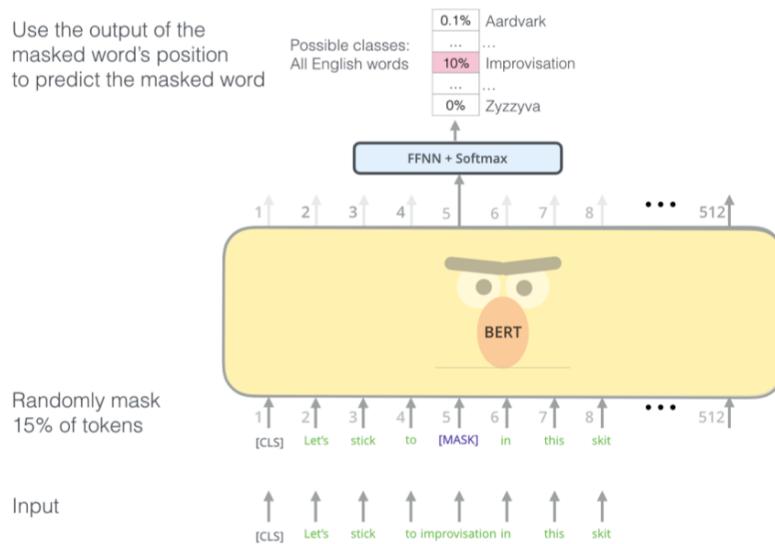
L'obiettivo di Bert è produrre una migliore rappresentazione dell'input che non tenga in considerazione le singole parole. In questa fase dobbiamo principalmente utilizzare il training del classificatore con pochi cambiamenti che riguardano il Bert Model, questo processo di training è chiamato Fine-Tuning.



Pre-Trained Tasks

Abbiamo detto che dobbiamo effettuare una fase di semi-supervised Learning in cui possiamo andare a fare il learning del modello. In questa fase, per ottenere il pre-trained model dovremmo eseguire vari task:

- Masked Language Model: il modello dovrebbe essere in grado di riconoscere la parola corretta in ogni posizione. Per ogni parola dovremmo avere una probabilità che quella parola si trovi in quella posizione. In particolare il transformer utilizza la softmax che produce una distribuzione per ogni possibile parola. Il training viene effettuato nascondendo una parte delle parole a caso e poi facendo la previsione di questi token.

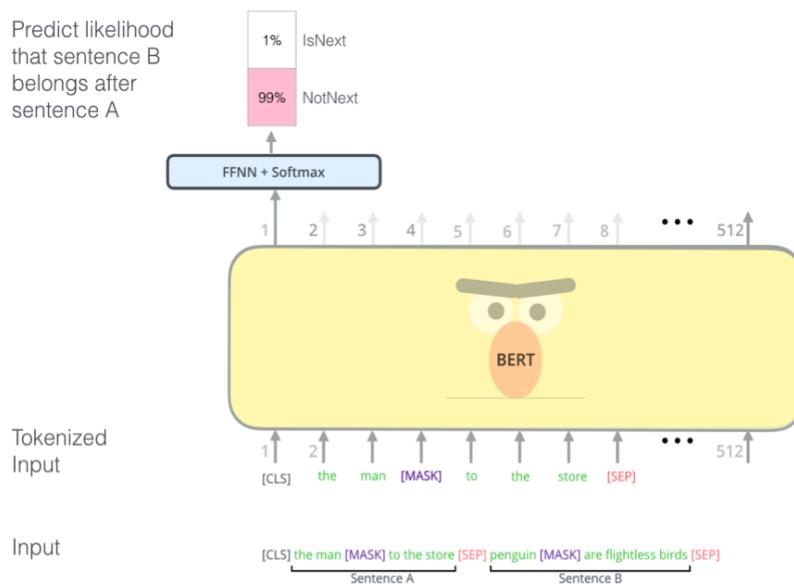


Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

Per quanto riguarda il Masked LM possiamo vedere nella immagine sopra quello che succede, noi abbiamo la nostra frase in input, scegliamo random alcune parole e mettiamo il token MASK. Queste parole in input sono prese e in parallelo vengono date in input al transformer, poi una volta uscite dal transformer diventano l'input di una Feed Forward Neural Network con Softmax e produciamo la distribuzione di probabilità per la parola nascosta.

Noi utilizziamo un grande numero di frasi per eseguire il training e poi sostituiamo solamente il 15% delle parole presenti all'interno con MASK. Dobbiamo imparare come sostituire questi token.

- Previsione della prossima frase: per eseguire il training di un modello che sia in grado di capire le relazioni tra le parole dobbiamo essere in grado di fare una previsione della prossima frase. Per ogni frase abbiamo il 50% di probabilità che quella frase è quella che effettivamente è corretta. Le frasi sono scelte a caso.



Nella immagine sopra possiamo vedere come funziona la previsione

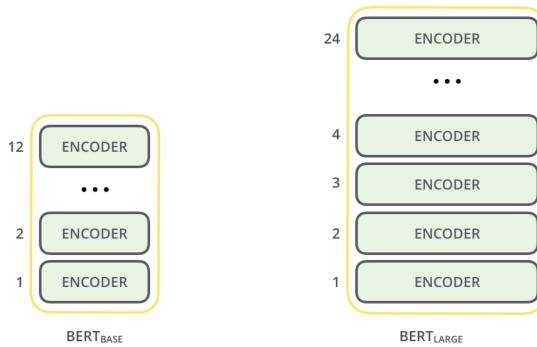
Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

della prossima frase, in particolare qua abbiamo due frasi in input vengono tokenizzate, notare che anche qua abbiamo MASK al posto di alcune parole. Per distinguere le frasi utilizziamo il token SEP. Una volta uscite dal transformer le frasi finiscono in una Feed Forward Neural Network in cui abbiamo anche un layer di softmax che produce la probabilità che le due frasi siano consecutive.

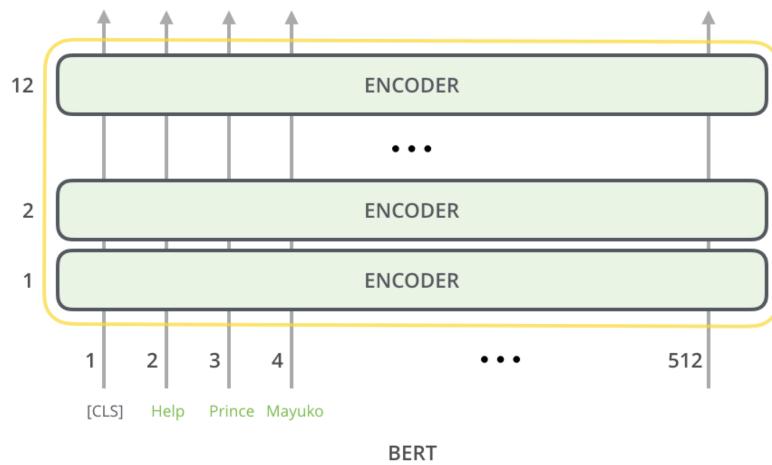
19.9.5 Architettura del Modello

Non abbiamo una singola architettura per Bert, dipende tutto dalla nostra scelta, se vogliamo avere più accuracy ad esempio e dipende anche dalla macchina che stiamo utilizzando. In pratica Bert è uno stack di Transformer Encoder messi uno sopra all'altro, qua i livelli di encoder vengono chiamati Transformer Block. Nel paper in particolare vengono presentati due modelli per Bert:

- Bert Base: è comparabile in dimensione al transformer di OpenAI, abbiamo 12 livelli di encoder. Qua abbiamo 768 hidden units nella Feed Forward Neural Network e 12 attention Heads, quindi vuol dire che abbiamo 12 differenti rappresentazioni per le parole.
- Bert Large: è un modello con 24 livelli di encoder che arriva allo stato dell'arte in termini di risultati ottenuti. Qua abbiamo anche 1024 hidden units nella Feed Forward Neural Network e 16 attention Heads.



Quello che succede è molto simile a quello che abbiamo nel transformer, in particolare i dati in input che finiscono all'interno del nostro modello passano attraverso una serie di layer di Encoder con self-attention. Ogni layer passa il risultato che ha ottenuto all'interno di una Feed Forward Neural Network il cui output finisce al layer superiore.

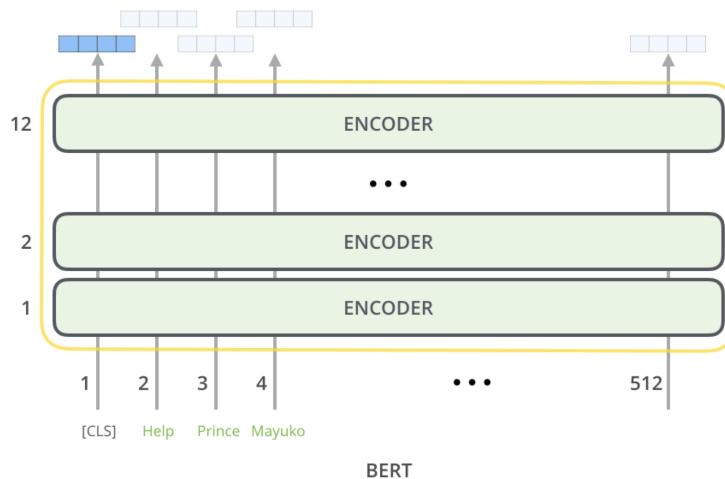


In termini di architettura fino a questo punto è identico al transformer.

Ognuna delle posizioni delle parole in input manda in output un vettore che sarà di dimensione 768 nel caso di Bert Base. All'interno di questo vettore troveremo l'embedding di quella parola prodotto da Bert. Il vettore che abbiamo ottenuto può diventare poi l'input del classificatore,

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

nel paper si ottengono buoni risultati utilizzando una Feed Forward NN come classificatore con un singolo layer. In questa fase abbiamo anche la softmax che ci produce l'output che stiamo cercando. Se abbiamo più di due sole classi possiamo modificare la softmax e andare quindi ad avere più neuroni in output che poi passano attraverso la softmax.



Quello che abbiamo visto fino ad ora ci permette di creare il nostro pre-trained model, la fase successiva è la fase di Fine-Tuning che però è semplice rispetto alla prima fase. Una volta che abbiamo il nostro pre-trained model possiamo utilizzarlo per vari task, diamo in input i dati di quel task e gli output e poi eseguiamo il tuning dei parametri. Confrontato con la fase di Pre-Training, il Fine Tuning è un processo relativamente poco costoso.

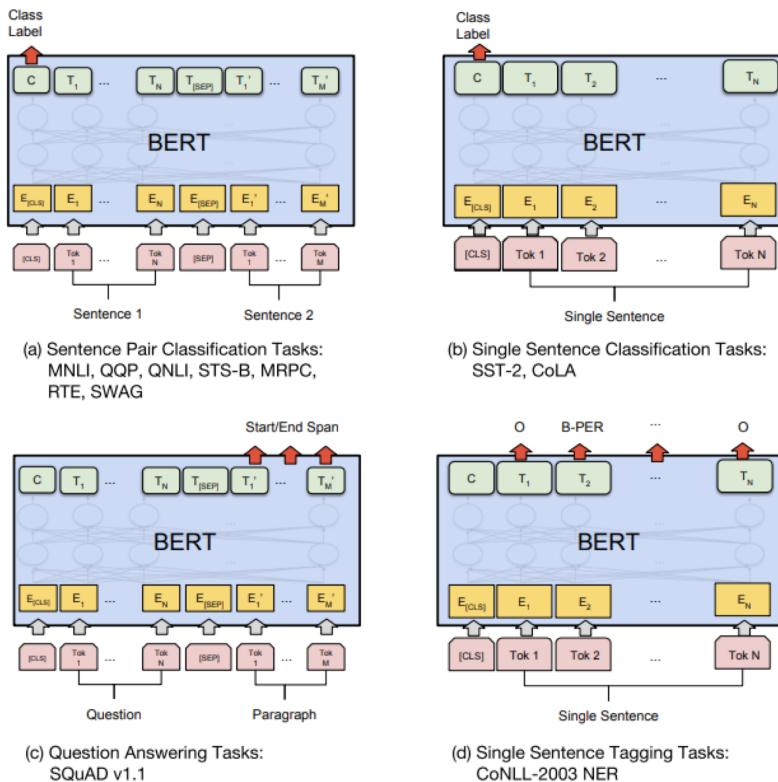
Task Specific Models

Nel paper di Bert vengono mostrati vari possibili utilizzi di Bert in task differenti.

I modelli specifici per un certo task sono formati incorporando Bert con un layer addizionale in output o la softmax in modo che solamente

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

un minimo numero di parametri dovranno essere imparati da 0, non dovremo rifare di nuovo il training di Bert.



I dati in input variano in base al task che dobbiamo svolgere, ad esempio nella immagine sopra abbiamo:

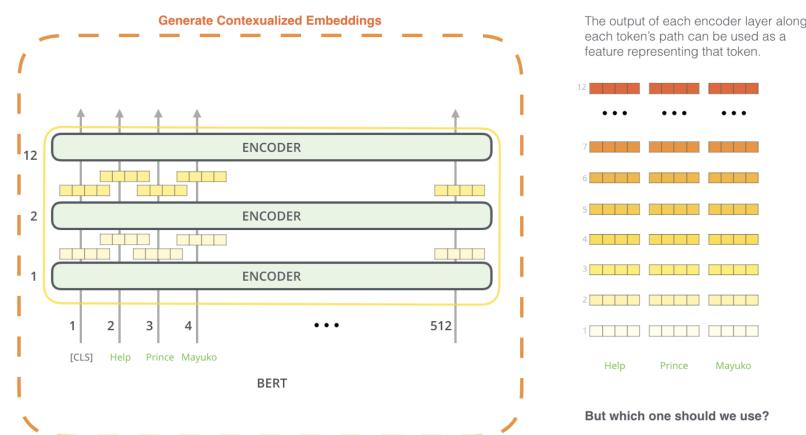
- Il caso a: è il caso in cui vogliamo fare la classificazione di due frasi contemporaneamente, qua noi diamo in input due frasi a Bert.
- Nel caso b invece diamo in input una singola frase e poi in output abbiamo la classificazione.
- Nel caso c abbiamo un problema di question answering, quindi abbiamo in input (distanziate con un token) la domanda e un paragrafo in cui si trova la risposta.

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

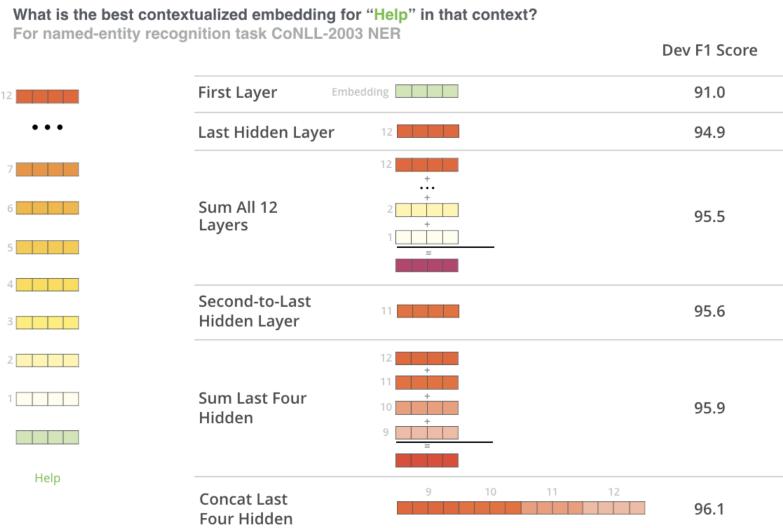
- Nel caso d invece abbiamo il Named Entity Recognition, per ogni token in input ci aspettiamo un output.

Bert per Feature Extraction

L'approccio Fine-Tuning non è il solo che possiamo utilizzare con Bert, volendo possiamo anche utilizzare la versione pre-trained di Bert per creare dei word embedding contestualizzati, poi possiamo utilizzare questi word embedding all'interno del nostro modello. Questo processo ci porta a dei risultati che sono non troppo lontani da quelli che possiamo ottenere con il fine-tuning di Bert.



Ognuno di questi layer di Bert ci produce una feature che rappresenta la frase, in particolare per ogni parola nella frase abbiamo un vettore. Per quanto riguarda il tipo di vettore che deve essere utilizzato come contextualized Embedding ad ognuno di questi livelli, questo dipende dal tipo di task che stiamo cercando di risolvere.



19.9.6 Conclusioni su Bert

Tutto questo funziona?

Possiamo considerare GLUE per il benchmark, in particolare si considerano 10 task (tra queste abbiamo un task di QA in cui abbiamo domande piccole e non possiamo utilizzare direttamente il paragrafo per la risposta ma dobbiamo capire se abbiamo una contraddizione o meno e poi abbiamo CoLa che è un task in cui cerchiamo di distinguere i comportamenti accettabili da quelli non accettabili).

System	MNLI-(m/mm)	QQP	QNLI	SST-2	CoLA	STS-B	MRPC	RTE	Average
Pre-OpenAI SOTA	392k	363k	108k	67k	8.5k	5.7k	3.5k	2.5k	-
BiLSTM+ELMo+Attn	80.6/80.1	66.1	82.3	93.2	35.0	81.0	86.0	61.7	74.0
OpenAI GPT	76.4/76.1	64.8	79.8	90.4	36.0	73.3	84.9	56.8	71.0
BERT _{BASE}	82.1/81.4	70.3	87.4	91.3	45.4	80.0	82.3	56.0	75.1
BERT _{LARGE}	84.6/83.4	71.2	90.5	93.5	52.1	85.8	88.9	66.4	79.6
	86.7/85.9	72.1	92.7	94.9	60.5	86.5	89.3	70.1	82.1

Nella tabella possiamo vedere i risultati che otteniamo, Bert sorpassa tutte le altre soluzioni, in alcuni casi non vediamo troppe differenze ma in alcuni task dove si partiva da score molto bassi Bert dimostra tutta la sua potenza.

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

Un altro dataset per calcolare le performances è Squad 2.0, in questo caso Bert supera lo score che viene ottenuto da persone reali. Questo in realtà è un task semplice perchè il modello sa che la risposta è sempre presente all'interno del paragrafo. Per questa ragione la seconda versione di Squad ha tolto le informazioni riguardanti la risposta dal paragrafo, in questo caso abbiamo che il risultato di Bert viene superato dal risultato umano.

Dimensione del modello

Abbiamo detto che abbiamo due modelli di Bert, uno Base e uno Large, solitamente quando aumentiamo la dimensione del modello riusciamo a migliorare l'accuracy. In particolare in NLP la dimensione del modello è importante, una cosa che possiamo notare è che l'accuracy non smette mai di crescere all'aumentare del numero di parametri presenti all'interno del modello e questo risultato è simile a quello che abbiamo anche nella Neural machine translation. La cosa che possiamo notare però è che l'aumento che otteniamo è piccolo rispetto al costo di aumentare la complessità del modello

19.10 Evoluzioni di Bert

19.10.1 RoBERTa

RoBERTa è l'estensione di Bert proposta da Facebook, l'idea è stata quella di eseguire il training di Bert andando però ad utilizzare più epoch. I ricercatori hanno infatti mostrato che già utilizzare più epoch sugli stessi dati aiuta, così come aiuta aggiungere altri dati. Con RoBERTa si ottiene un miglioramento in termini di score rispetto ai risultati che abbiamo con il Bert classico in versione large.

19.10.2 XLNet

Questa è un'altra soluzione che è stata presentata da Google, è chiamata XLNet, loro hanno introdotto la relative position embedding. La relative attention è differente rispetto alla absolute attention, consideriamo una frase che è "John ate a hot dog":

- Absolute attention: con la absolute attention consideriamo quanto è legato dog rispetto ad hot e consideriamo quanto la parola dog in quarta posizione è legata rispetto alla parola in terza posizione.
- Relative attention: con la relative consideriamo quanto è legato dog rispetto ad hot in qualsiasi posizione e poi consideriamo quanto è legato dog rispetto alla sua parola precedente.

La seconda innovazione che abbiamo con XLNet è il Permutation language Model. In un language model classico abbiamo che ogni parola viene prevista sulla base delle parole che si trovano alla sua sinistra. Qua invece andiamo a prendere le varie frasi di training ed eseguiamo una permutazione random, è equivalente ad eseguire il masking ma avremo in questo modo molte più previsioni per ogni frase.

XLNet utilizza inoltre più dati e anche un modello più grande ma comunque è stato dimostrato che si ottengono dei risultati migliori di Bert anche a parità di dati e di dimensione del modello.

Model	MNLI	QNLI	QQP	RTE	SST-2	MRPC	CoLA	STS-B
<i>Single-task single models on dev</i>								
BERT [2]	86.6/-	92.3	91.3	70.4	93.2	88.0	60.6	90.0
RoBERTa [21]	90.2/90.2	94.7	92.2	86.6	96.4	90.9	68.0	92.4
XLNet	90.8/90.8	94.9	92.3	85.9	97.0	90.8	69.0	92.5

19.10.3 ALBERT

Con ALBert si sono posti l'obiettivo di creare un modello molto più semplice di quello che abbiamo con Bert. Teniamo a mente che in Bert

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

base abbiamo 768 hidden units mentre nel Large Bert ne abbiamo 1024. Supponiamo di avere un numero di vocaboli all'interno del dizionario che è $V = 100.000$. Con il classico Bert la matrice degli embedding sarebbe $1024 * 100.000$, in ALBert invece si utilizza una fattorizzazione degli embedding che vengono decomposte in due matrici più piccole. In particolare abbiamo degli embedding più piccoli (128) e poi viene fatta una proiezione nella dimensione del transformer (1024).



Anche se abbiamo reso il modello più semplice otteniamo comunque un risultato competitivo e abbiamo un numero di parametri che è diminuito molto.

19.10.4 T5

L'ultima innovazione in questo campo è il modello T5 che è un modello enorme, il training viene effettuato su un dataset gigantesco. Il modello è anche molto complicato ed attualmente è quello che raggiunge lo score più alto dopo l'uomo.

Il modello T5 funziona con 11 miliardi di parametri (Bert Large ne ha 300 milioni) e viene eseguito il training su 120 miliardi di parole.

19.11 Distillation

Abbiamo parlato fino ad ora di modelli molto grandi che necessitano di molta memoria per essere utilizzati. Come può fare una azienda ad utilizzare un modello pre-trained così grande e costoso? Si applica la

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

distillation ovvero si prende il modello originale e non si considerano alcune parti lasciando però il modello finale simile a quello originale.

L'idea è di avere un modello "Teacher" su cui facciamo il training cercando di ottenere l'accuracy più alta possibile. Poi si fa la labellizzazione di tanti esempi di training non ancora visti tramite il modello "Teacher". Poi si esegue il training di un modello "Student" che è un modello più piccolo, i parametri di questo modello vengono allenati per cercare di avere un output che è il più simile possibile all'output del "Teacher".

Il processo di Distillation può essere molto utile perchè replichiamo il modello "Teacher" utilizzando però molti meno parametri e quindi diminuendo la complessità. Questo metodo di Distillation funziona sorprendentemente bene, questo è probabilmente legato alla capacità di rappresentare il language model, inoltre dobbiamo considerare che quando eseguiamo il training stiamo utilizzando tantissimi dati e probabilmente vengono catturate numerose features che sono utili per altri task di NLP e non per quello che stiamo effettuando in quel momento. Quindi la distillation probabilmente ci permette di focalizzarci solamente su alcune delle feature del modello. In generale comunque per eseguire la distillation mi serve un modello che deve essere necessariamente più grande.

19.12 Conclusioni

I modelli pre-trained bidirectional language model funzionano incredibilmente bene ma sono anche molto costosi. I miglioramenti di questi modelli ci portano ad avere modelli ancora più costosi e che richiedono più dati. Fortunatamente noi possiamo scaricare questi modelli dal web e possiamo utilizzarli, se abbiamo bisogno di utilizzarli per un nostro uso specifico però dobbiamo crearceli il nostro modello e potrebbe essere un problema il training, stesso discorso se vogliamo una lingua del modello differente da quella che viene fornita online. Ad esempio il modello potrebbe essere in inglese ma noi lo vogliamo in italiano.

Per fare il training dobbiamo considerare che abbiamo bisogno di grandi dataset, assumendo di avere questi dataset abbiamo bisogno anche di macchine molto potenti, quindi avremo bisogno di un cluster di macchine. Ad esempio un paio di mesi fa (oggi è il 4 Maggio 2020) Facebook ha provato a creare un language model enorme, utilizzava 500 GPU per il training ma hanno dovuto fermarsi perchè era finita la memoria.

Se facciamo un errore mentre facciamo il training può essere un grave problema perchè dobbiamo ricominciare dall'inizio.

Il consiglio quindi è quello di partire con un dataset piccolo e poi estendere.

Chapter 20

Syntax Probe

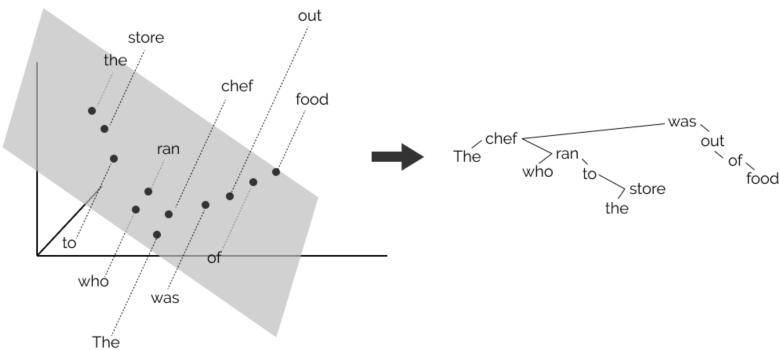
Maggiori informazioni su questo argomento¹.

Bert e gli altri sistemi di cui abbiamo parlato fino ad ora non conoscono niente riguardo alla sintassi della frase che stiamo considerando, come fanno a funzionare comunque?

20.1 Introduzione

Nel linguaggio umano il significato delle frasi viene ricostruito compiendo piccoli gruppi di parole e unendoli tra di loro ottenendo poi gruppi di dati via via più grandi e più complessi. L'ordine con cui questi chunks di dati vengono combinati crea una struttura ad albero che viene chiamata parse tree.

¹Più informazioni su questo argomento: <https://nlp.stanford.edu/~johnhew/structural-probe.html>.

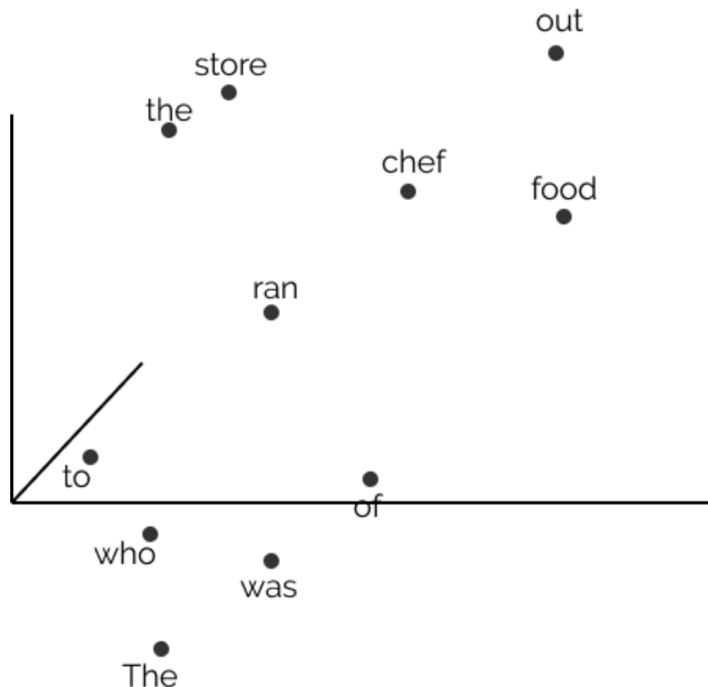


Nella figura, sulla destra, possiamo vedere il parse tree corrispondente alla frase "The chef who ran to the store was out of food". Questa frase ha due possibili interpretazioni, lo chef potrebbe aver finito il cibo oppure il negozio, la nostra conoscenza ci permette di disambiguare e di capire quali sono le relazioni tra le parole. Per disambiguare dobbiamo possiamo utilizzare ad esempio un parser che può comprendere come suddividere la frase e in particolare noi abbiamo visto i parser transition based e sequence based.

Il fatto è che negli ultimi anni nel campo dell'NLP queste frasi non sono state rappresentate come alberi di parole ma sono state rappresentate come insiemi di vettori, ogni vettore è un embedding delle varie parole.

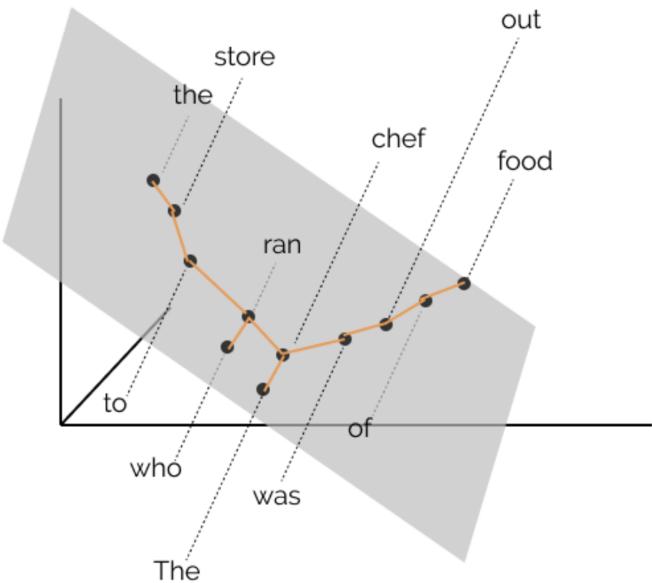
$$\begin{array}{cccccccccccc}
 \text{The} & \text{chef} & \text{who} & \text{ran} & \text{to} & \text{the} & \text{store} & \text{was} & \text{out} & \text{of} & \text{food} \\
 \begin{bmatrix} .4 \\ -.2 \\ .3 \end{bmatrix} & \begin{bmatrix} .1 \\ .9 \\ -.2 \end{bmatrix} & \begin{bmatrix} .3 \\ -.4 \\ .2 \end{bmatrix} & \begin{bmatrix} .7 \\ -.4 \\ 0 \end{bmatrix} & \begin{bmatrix} .4 \\ 0 \\ -.5 \end{bmatrix} & \begin{bmatrix} .1 \\ -.6 \\ .2 \end{bmatrix} & \begin{bmatrix} .3 \\ .1 \\ -.6 \end{bmatrix} & \begin{bmatrix} .1 \\ .9 \\ -.8 \end{bmatrix} & \begin{bmatrix} .3 \\ .1 \\ .8 \end{bmatrix} & \begin{bmatrix} -.8 \\ .3 \\ -.6 \end{bmatrix} & \begin{bmatrix} 0 \\ .7 \\ -.9 \end{bmatrix}
 \end{array}$$

Normalmente infatti per ogni parola abbiamo un vettore e questo possiamo anche rappresentarlo in uno spazio, all'interno di questo spazio il vettore può essere ovunque, potrebbe essere distribuito in modo che siano tutti staccati tra loro o magari vicini se le parole sono simili.



Questi due metodi di rappresentazione, la struttura ad albero e i vettori, possono essere combinate insieme? Vorremmo trovare un metodo che, preso un vector space, ci permetta di trovare la struttura ad albero corrispondente. La seguente immagine ci fornisce una intuizione di come dovrebbe funzionare un tool di questo genere:

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.



Abbiamo all'interno del piano una serie di parole e queste sono unite da un percorso. Per ottenere tutto ciò dobbiamo trovare la corretta trasformazione lineare che vogliamo effettuare, fare la proiezione nel piano e poi vediamo che le parole che sono vicine tra loro sono quelle che sono sintatticamente connesse.

Fino ad ora abbiamo parlato di word embedding e ci sono state nel corso degli anni varie proposte:

- Originariamente i word embedding mappavano ogni parola presente nel testo in un vettore, quindi in pratica rappresentavano una funzione del tipo:

$$f_{vocab} = v - > h$$

dove la v rappresenta la singola parola e la h rappresenta il vettore.

- È stato proposto poi un metodo differente chiamato Subword method in cui viene considerata anche la sequenza di caratteri all'interno

della parola. Questo metodo, chiamato subword viene definito come:

$$f_{subword} = (v, (c_1, \dots, c_t)) - > h$$

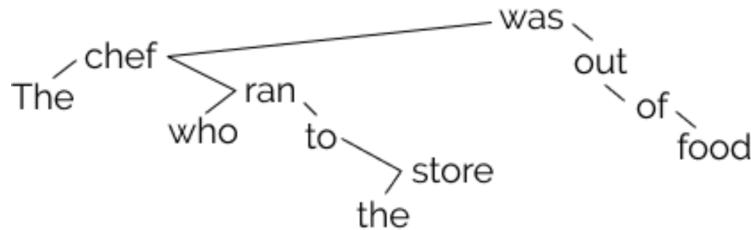
- Poi è arrivata l'idea che l'embedding della parola non dovrebbe dipendere solamente dalla singola parola ma anche da quello che gli sta intorno. Quindi supponiamo di avere un testo w_1, \dots, w_n prendiamo tutto il contesto e creiamo l'embedding:

$$f_{contextual} = (w_1, \dots, w_N) - > (h_1, \dots, h_N)$$

Per comprendere meglio il significato del testo però, oltre al comprendere la parola all'interno del contesto potrebbe essere utile andare a considerare la sintassi. In particolare, data la frase che stavamo considerando in precedenza potremmo avere due continuazioni differenti:

1. Because there was no food to be found, the chef went to the next store.
2. After stocking up on ingredients, the chef returned to the restaurant.

La seconda ipotesi sembra più adatta alla nostra situazione, c'è però da considerare che la frase "the store was out of food" è una sottostringa della frase originale, conoscendo il fatto che "the chef was" e non "the store was" potrebbe essere utile per disambiguare. Per fare in modo che questo funzioni e che si possano distinguere le interpretazioni sarebbe interessante saper comprendere che in realtà il verbo was è riferito allo chef e non allo store. Questa intuizione la possiamo codificare nel seguente dependency parse tree:

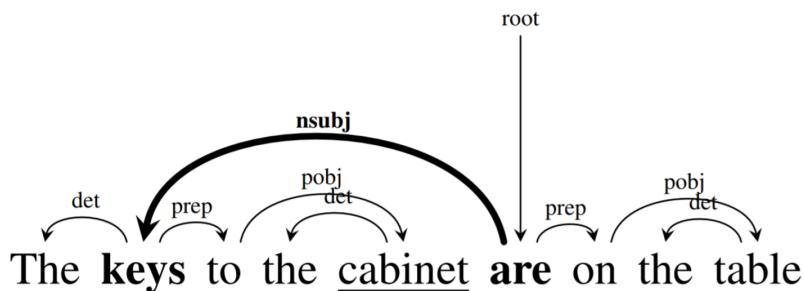


Con una interpretazione differente invece andremmo a connettere "was" con "store".

20.2 Observational vs constructive evidence

All'interno dell'analisi del testo possiamo fare una distinzione tra gli "observational studies" e i "constructive studies":

- Observational Evidence: un observational network valuta il modello considerando il task per cui è stato ottimizzato, solitamente gli input vengono anche scelti a mano per cercare di capire se un certo comportamento è osservato o meno.

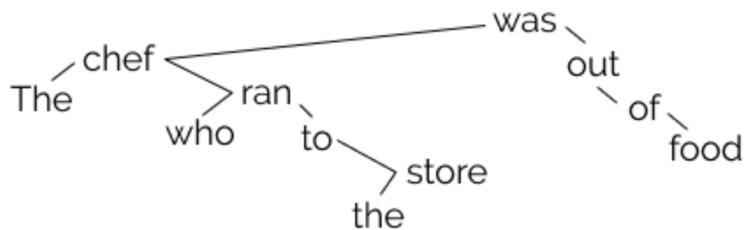


Un esempio di questo studio è quello che vediamo nella foto in cui viene testato se il modello assegna una maggiore probabilità alle forme verbali che sono concordi con il soggetto della frase.

- Constructive Evidence: questo specifica un set di funzioni Θ tramite il quale il modello può codificare i fenomeni di interesse (sintassi, part of speech, ecc) e spesso viene fatto il training sui dati supervisionati per scegliere quali sono i componenti del set di funzioni $\Theta^* \in \Theta$ che meglio rappresentano questi fenomeni. Questo viene chiamato probe classifier o classificatore diagnostico. Un esempio recente è uno studio in cui si è cercato di comprendere cosa i modelli di neural machine translation imparano dalla morfologia. Il loro fenomeno di interesse in questo caso è un set di feature morfologiche, il task in questo caso è di classificazione, per ciascuna delle parole in ogni frase possiamo classificare quali sono le feature morfologiche che valgono.

Questa constructive evidence è utile perché, se le nostre ipotesi sono corrette allora Θ^* ci permette di estrarre il fenomeno di interesse dal modello. Nonostante questo non abbiamo comunque una risposta alla domanda iniziale ovvero "Il mio modello ha eseguito il learning del fenomeno X?", abbiamo solamente una risposta alla domanda "Il mio modello codifica il fenomeno X in un modo che sia trattabile dal mio probe?"

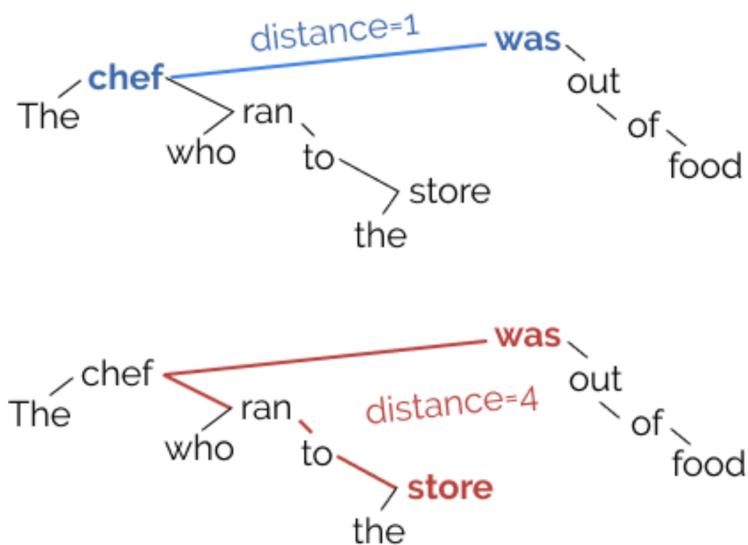
L'idea comunque è che all'interno di ogni frase ci sia un latent parse tree a cui la rete neurale non ha accesso, ogni parola presente all'interno della frase ha un corrispondente nodo all'interno del parse tree. Un esempio di parse tree è il seguente:



Quando ci troviamo nello spazio vettoriale le informazioni riguardanti

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

il significato delle parole vengono rappresentate in termini di distanze, possiamo pensare ad una misurazione che consiste nel contare quanto due parole sono lontane all'interno della rappresentazione vettoriale. Possiamo estendere questo ragionamento anche ai parse tree, in particolare in questo caso per capire quanto due parole sono distanti non consideriamo la distanza in uno spazio vettoriale ma la distanza in termini di percorso all'interno dell'albero.

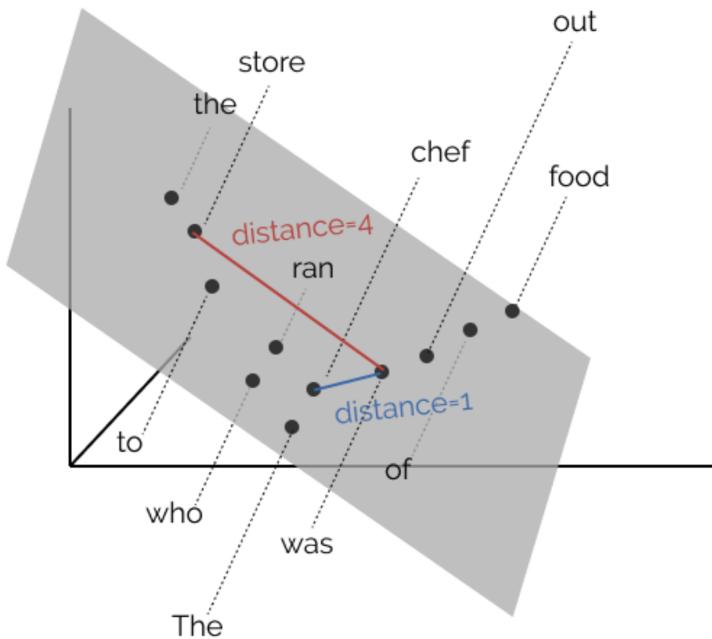


Una volta che abbiamo tutte le distanze tra le varie parole della frase che stiamo considerando (sono N^2) distanze allora l'embedding dell'albero si riduce all'embedding della distance metric definita dall'albero.

Se la distanza che abbiamo all'interno del latent parse tree dovesse essere la stessa che abbiamo tra i word vector costruiti dalla rete neurale allora vorrebbe dire che la rete ha praticamente imparato come calcolare l'embedding dell'albero. In realtà i requisiti che abbiamo sono più rilassati, la neural network infatti non deve codificare solamente informazioni riguardanti la distanza ma anche tante altre informazioni. Questo ci porta a formulare la syntax distance hypothesis:

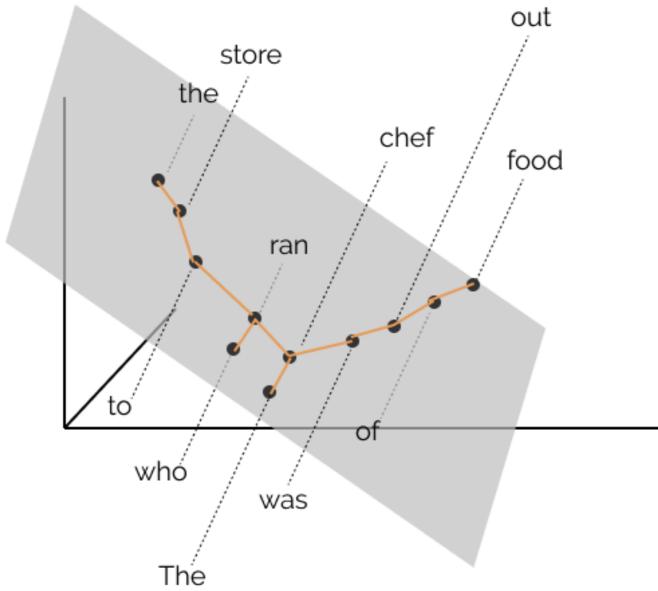
Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

L'ipotesi ci dice che esiste una trasformazione lineare B della rappresentazione delle parole nel piano tramite la quale il vettore delle distanze codifica il parse tree. In pratica cerchiamo la trasformazione B che fa sì che le parole siano più vicine nello spazio se queste sono vicine tra loro all'interno del parse tree. Per il calcolo della distanza useremo la distanza L2. Qua nella immagine possiamo vedere come le parole vengono mappate nel piano, in questo caso vediamo che ad esempio tra le parole store e was abbiamo una distanza maggiore di quella che abbiamo tra was e chef.



In particolare se riusciamo a calcolare tutte queste distanze possiamo recuperare lo stesso albero che ci interessa, più vicine sono le parole nel piano e più vicine sono all'interno del parse tree, se due parole sono vicine tra loro allora sono connesse, quello che otteniamo è il minimum spanning tree. Una volta che abbiamo trovato la trasformazione lineare per i nostri dati e una volta che abbiamo anche calcolato il minimum spanning tree riusciamo ad ottenere il parse tree che ci interessa.

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.



Quella che abbiamo ottenuto è una rappresentazione molto simile al parse tree originale che avevamo.

20.2.1 Trovare una distance metric per il parse tree

Le nostre possibili distanze per la codifica dell'albero sono parametrizzate tramite una trasformazione lineare $B \in R^{k*n}$.

$$\|h_i - h_j\|_B^2 = (B(h_i - h_j)^T)(B(h_i - h_j))$$

In questa formula abbiamo Bh che è la trasformazione lineare della rappresentazione del mondo, la distanza nello spazio è la distanza tra gli embedding. Questa distanza tra i due vettori possiamo riscriverla considerando la distanza L2 nello spazio vettoriale originale parametrizzato dalla matrice positive semi-definite $A = B^T B$ ovvero:

$$\|h_i - h_j\|_A^2 = (h_i - h_j)^T A (h_i - h_j)$$

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

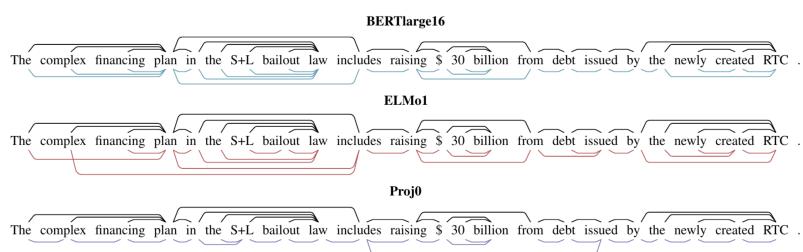
L'idea è di scegliere la B per fare in modo di minimizzare la differenza tra le parole all'interno del parse tree rispetto allo spazio in cui facciamo la proiezione.

$$\min_B \sum_{\ell} \frac{1}{|s_{\ell}|^2} \sum_{i,j} (d(w_i, w_j) - \|B(h_i - h_j)\|^2)$$

Le parole che sono vicine nel parse tree dovrebbero essere vicine anche nel projected space. Mappiamo i vettori nel piano e consideriamo la distanza al quadrato, l'obiettivo sarebbe quello di avere due distanze uguali in modo che con la sottrazione si azzeri. In questa formula dobbiamo trovare la B che approssima meglio, possiamo fare eseguendo il training con un corpus e con una collezione di parse tree. Possiamo produrre un valore per ogni frase e poi possiamo usare la trasformazione B per calcolare la distanza.

20.2.2 Reconstructed trees and depth

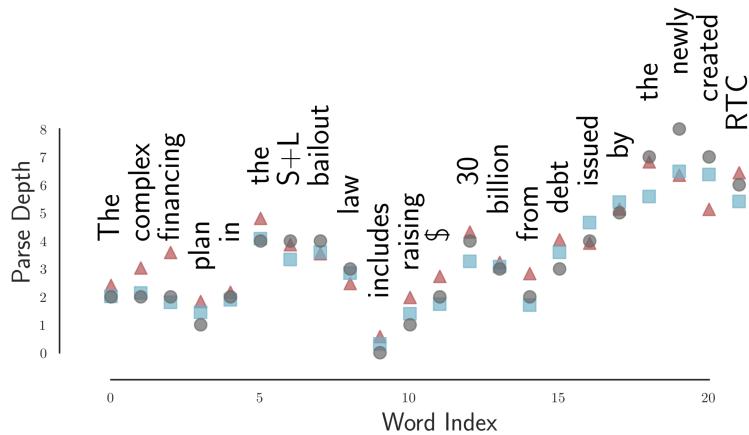
Sorprendentemente questo sistema funziona, cercando di minimizzare la loss possiamo veramente trovare la B che può essere utilizzata nella trasformazione per prendere la frase e per creare il minimum spanning tree.



Nella immagine sopra possiamo vedere come il minimum spanning tree (quello colorato in blu, rosso e viola) sia veramente vicino all'albero

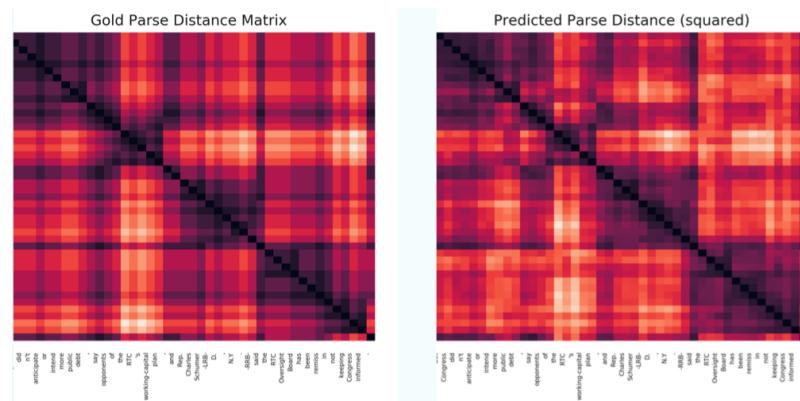
Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

originale (quello nero). Un altro confronto che possiamo fare riguarda la profondità della parola all'interno dell'albero, in questo caso abbiamo il puntino grigio che indica la profondità nel parse tree ottimo mentre gli altri segni indicano la profondità nei vari alberi che otteniamo.



Come possiamo vedere dall'immagine non c'è molta differenza.

Un'altra metrica che possiamo considerare per capire la distanza tra il parse tree ottimale e quello trasformato è una distance matrix. All'interno della matrice abbiamo dei colori che sono più scuri quando le parole sono vicine tra loro mentre sono più chiari quando sono più lontane:



Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

Anche in questo caso non troviamo troppa differenza tra le due matrici.

20.3 Conclusioni

Fino ad ora abbiamo parlato di ricostruire la sintassi della frase perché potrebbe essere nascosta all'interno. Possiamo utilizzare anche una metrica semantica? Si potrebbe fare e questo potrebbe essere utile in determinati task di question answering. Anche questa è una questione interessante da considerare.

In generale possiamo dire di aver trovato qualcosa che funziona sorprendentemente bene, usiamo una trasformazione lineare per ricostruire il parse tree, questo vuol dire che potremmo anche trovare delle strutture che sono nascoste all'interno dei nostri dati, potrebbero esserci delle rappresentazioni nascoste.

20.4 Bert Sentiment - Notebook

20.4.1 Introduzione

Nel notebook "Bert-Sentiment" vogliamo implementare la "State-of-the-Art" Sentiment Classification utilizzando Bert. Utilizziamo le seguenti tecnologie:

- Transfer Learning: abbiamo dimostrato che questo sistema può permettere di raggiungere buone performances con una quantità di dati non troppo alta. L'idea è di avere un modello che viene trainato su una quantità di dati molto alta e quello che otteniamo è il pre-trained model poi questo lo utilizziamo nella seconda fase della classificazione.

- Bert: Utilizziamo il modello di Bert che è stato trainato con le informazioni prese da Wikipedia da Google. Questo modello prova a fare una previsione di alcune parole che sono state nascoste da una frase. Il modello combina il task di previsione con una architettura con il Transformer che crea una rete neurale che usa la self attention per identificare le relazioni di una parola con tutte le altre parole nella frase.
- Huggingface Transformer: Huggingface ha rilasciato un wrapper code che semplifica l'utilizzo di Bert e degli altri transformer model.

Nel nostro notebook lavoriamo con una parte dei dati del dataset *"yelp_polarity_reviews"*, consideriamo solamente una parte per fare in modo di velocizzare il training. All'interno del dataset abbiamo:

20.4.2 Training del modello

Per fare il training del modello è stato dimostrato che batch piccoli possono migliorare la generalizzazione del nostro modello, quindi nel nostro caso utilizziamo 32 come dimensione del nostro batch.

Prima del training una cosa che dobbiamo fare è la conversione delle parole in numeri in modo che possano essere poi processati dal Bert Transfer Learning. Questo processo è chiamato tokenizzazione e Huggingface include un tokenizer che possiamo utilizzare all'interno di questo processo.

```
#tokenizer = DistilBertTokenizer.from_pretrained('distilbert-base-cased')
tokenizer = BertTokenizer.from_pretrained('bert-base-cased')
```

Noi qua stiamo creando il nostro BertTokenizer utilizzando come modello pre-trained il "bert-base-cased". Nella riga commentata stiamo utilizzando non direttamente il BertTokenizer ma la versione "distillata", in pratica come abbiamo visto è una versione più semplice del modello.

Allo stesso modo possiamo utilizzare il modello pre-trained di Bert per la classificazione:

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

```
#model = TFAutoModelForSequenceClassification.from_pretrained('distilbert-base-uncased-finetuned-sentiment')
model = TFBertForSequenceClassification.from_pretrained('bert-base-cased')
```

Quello che possiamo fare è andare a prendere il tensori del nostro dataset e vogliamo tokenizzarli per trasformarli in indici.

Il modello che stiamo utilizzando qua per le nostre previsioni è il seguente:

Model: "tf_bert_for_sequence_classification"		
Layer (type)	Output Shape	Param #
bert (TFBertMainLayer)	multiple	108310272
dropout_37 (Dropout)	multiple	0
classifier (Dense)	multiple	1538

Total params: 108,311,810
Trainable params: 108,311,810
Non-trainable params: 0

Abbiamo il primo livello che è Bert che crea gli embedding dipendenti dal contesto, poi abbiamo un layer con il dropout e poi abbiamo il layer con il classificatore che prende i dati in output di Bert e mi classifica le varie frasi. Il modello viene fittato su i nostri dati di training:

```
history = model.fit(train_dataset, epochs=1)
```

L'output del fit del modello ci mostra che in questo momento il modello ha una accuracy di circa il 90% che, considerando anche il fatto che usiamo solamente una epoch è veramente un buon risultato. Va anche considerato il fatto che le API di Yelp in alcuni casi le API di Yelp ci offrono solamente parte della recensione scritta dall'utente quindi può diventare ancora più complesso comprendere bene la polarità della frase.

20.4.3 Valutare i risultati del modello

Una volta che è stato effettuato il training del modello possiamo valutarne le performances. Possiamo anche visualizzare le performances

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

del modello in termini di quanti true positive e false negative troviamo. Questo tipo di valutazione viene solitamente chiamata confusion matrix. Per creare la confusion matrix si deve:

- Prendere l'output del modello non normalizzato e comprimerlo in probabilità, questo viene fatto con la softmax.
- Una volta che per ogni previsione abbiamo la probabilità dobbiamo identificare la previsione che ha la probabilità maggiore e questo lo facciamo con la funzione argmax.
- Poi identifichiamo true positive, true negative, false negative e false positive e visualizziamo la confusion matrix utilizzando seaborn e matplotlib.

Chapter 21

Pytorch

21.1 Introduzione

PyTorch è una libreria che permette di sostituire NumPy per astrarre ancora di più la scrittura del codice e per utilizzare la potenza delle GPU. Si tratta di una piattaforma che fornisce maggiore flessibilità e velocità nella scrittura del codice. Partiamo dalle basi, in Numpy possiamo definirci delle matrici usando il codice:

```
zero_np = np.zeros([2, 3])
zero_np
```

La stessa cosa la possiamo fare anche in PyTorch e qua questo tipo di struttura dati prende il nome di Tensor:

```
zero_pt = torch.zeros([2,3])
zero_pt
```

È anche possibile passare da un numpy array ad un tensore di PyTorch utilizzando la funzione *torch.tensor*, il passaggio inverso da tensore a numpy array lo possiamo fare con la funzione *numpy()*.

```
zero_pt_from_np = torch.tensor(zero_np)
zero_np_from_pt = zero_pt.numpy()
```

Abbiamo parlato di GPU e di come PyTorch sia pensato per essere utilizzato con le GPU, se vogliamo che il nostro codice venga paralleliz-

zato dobbiamo inserire questa riga nel notebook, il device che utilizziamo poi lo utilizzeremo in seguito:

```
device = 'cuda' if torch.cuda.is_available() else 'cpu'
```

Fino ad ora abbiamo parlato di tensori e abbiamo visto come crearne, quello che distingue un tensore in cui devono solamente finire dei dati da un tensore in cui invece possiamo inserire dei parametri che potranno essere modificati durante il training è che questo secondo tipo di tensori richiede la computazione del suo gradiente in modo da poter aggiornare il valore associato. per farlo, nel momento in cui definiamo un tensore x possiamo chiamare la funzione:

```
x.requires_grad_()
```

In questo modo se definiamo che questo tensore richiede il gradiente vuol dire che memorizzerà le varie operazioni che vengono eseguite in modo tale che alla fine possa essere chiamata la funzione *backward()* per calcolare in automatico il gradiente (il gradiente del tensore viene memorizzato all'interno dell'attributo *.grad*).

21.1.1 Autograd

Autograd è il pacchetto di PyTorch destinato all'automatic differentiation, grazie a questo pacchetto non dobbiamo preoccuparci delle derivate parziali e della chain rule. Per calcolare il gradiente l'unica cosa che facciamo quindi è chiamare il metodo *backward()*. Vediamo un esempio:

```
x = torch.ones(2, 2, requires_grad=True)
y = x + 2
z = y * y * 3
print(z)
z.backward(torch.ones([2,2]))
x.grad
```

Noi qua definiamo varie variabili e nella prima che definiamo, la x , indiciamo anche che vogliamo che venga calcolato il gradiente. Possiamo eseguire l'operazione forward che mi porta al calcolo della z e poi pos-

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

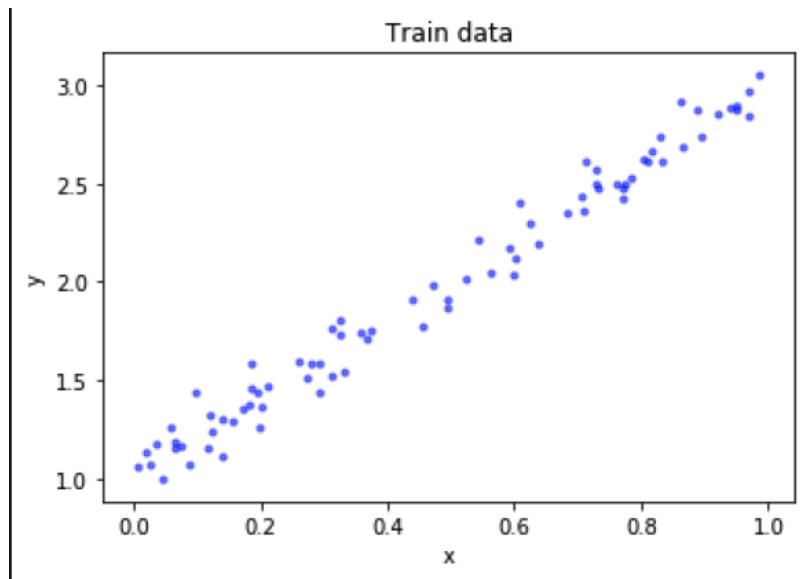
siamo eseguire l'operazione di backward. Per quanto riguarda il calcolo della z otteniamo:

```
tensor([[27., 27.], [27., 27.]], grad_fn=<MulBackward0>)
```

Per quanto riguarda il calcolo della backward, quello che succede è che dobbiamo calcolare la derivata della z rispetto alla x e questo lo facciamo con la chain rule ovvero calcoliamo prima la derivata di z rispetto a y e poi la derivata di y rispetto a x .

21.1.2 Training Data

Per fare una prima prova generiamo dei dati di training e facciamo in modo di posizionarli tutti attorno ad una linea immaginaria in modo tale da poter poi svolgere un problema di regressione.



21.1.3 Regression

Il processo della regressione qua in questo caso è tipico e abbiamo un loop con un certo numero di epoch e calcoliamo il gradient descent. Per un

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

certo numero di epoch, che in questo caso è 1000 partiamo da un valore random per a e b e poi produciamo un nostro output per ognuna delle x che abbiamo in input. Otteniamo quindi un vettore di y e sottraiamo il vettore delle y corrette e otteniamo la loss. Poi calcoliamo il gradiente chiedendo a Pytorch di calcolarlo utilizzando la funzione *backward()*. Quando chiamiamo questa funzione backward andiamo a memorizzare in *a.grad* e *b.grad* il gradiente che viene calcolato. Solitamente per modificare i pesi non usiamo tutto il gradiente ma solamente una porzione del gradiente che è *lr*, il learning rate, a e b vengono aggiornate in base al learning rate. Dopo aver fatto questo abbiamo calcolato la nuova a, rimettiamo il gradiente a 0 e finiamo l'iterazione per la prima epoch, poi iniziamo l'iterazione dell'epoch successiva. L'output che produciamo è una stima per a e per b.

```

lr = 1e-1
n_epochs = 1000

torch.manual_seed(42)
a = torch.randn(1, requires_grad=True, dtype=torch.float, device=device)
b = torch.randn(1, requires_grad=True, dtype=torch.float, device=device)

for epoch in range(n_epochs):
    yhat = a + b * x_train_pt
    error = y_train_pt - yhat
    loss = (error ** 2).mean()

    # We just tell PyTorch to work its way BACKWARDS from the specified loss!
    loss.backward()

    # We need to use NO_GRAD to keep the update out of the gradient computation
    # Why is that? It boils down to the DYNAMIC GRAPH that PyTorch uses...
    with torch.no_grad():
        a -= lr * a.grad
        b -= lr * b.grad

    # PyTorch is "clingy" to its computed gradients, we need to tell it to let it go...
    a.grad.zero_()
    b.grad.zero_()

print(a, b)

```

Quello che stiamo facendo qua è fare molte di queste operazioni

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

manualmente utilizzando i "building block" di pytorch. Possiamo rendere il tutto più automatico utilizzando ad esempio un Optimizer. Nel codice qua sotto l'ottimizzatore che utilizziamo è lo Stochastic Gradient Descent, qua il codice è semplificato perchè non dobbiamo fare manualmente la parte di aggiornamento della a e della b e la parte di azzeramento del gradiente perchè ci basterà chiamare *optimizer.step()* e *optimizer.zero_grad()*.

```

torch.manual_seed(42)
a = torch.randn(1, requires_grad=True, dtype=torch.float, device=device)
b = torch.randn(1, requires_grad=True, dtype=torch.float, device=device)
print(a, b)

lr = 1e-1
n_epochs = 1000

# Defines a SGD optimizer to update the parameters
optimizer = optim.SGD([a, b], lr=lr)

for epoch in range(n_epochs):
    yhat = a + b * x_train_pt
    error = y_train_pt - yhat
    loss = (error ** 2).mean()

    loss.backward()

    # Gradient
    # with torch.no_grad():
    #     a -= lr * a.grad
    #     b -= lr * b.grad
    optimizer.step()

    # No more telling PyTorch to let gradients go!
    # a.grad.zero_()
    # b.grad.zero_()
    optimizer.zero_grad()

print(a, b)

```

Modello

In PyTorch un modello viene rappresentato con una classe Python che eredita dalla classe nn.Module, i metodi fondamentali da implementare

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

sono:

- init
- forward, questo metodo lo definiamo ma in realtà non lo chiamiamo, chiameremo direttamente il modello.

Qua quindi ci definiamo una classe per la Linear Regression, l'architettura dei layer in PyTorch consiste in classi come questa che hanno due funzioni, una è la forward che dato l'input produce l'output. C'è anche una backward function che possiamo scrivere noi oppure Pytorch la può produrre da solo per noi.

```
class LinearRegression(nn.Module):
    def __init__(self):
        super().__init__()
        # To make "a" and "b" real parameters of the model, we need to wrap them with nn.Parameter
        self.a = nn.Parameter(torch.randn(1, requires_grad=True, dtype=torch.float))
        self.b = nn.Parameter(torch.randn(1, requires_grad=True, dtype=torch.float))

    def forward(self, x):
        # Computes the outputs / predictions
        return self.a + self.b * x
```

Ora che abbiamo creato la classe possiamo eseguire il training, anche qua abbiamo un for per le varie epoch, qua quello che si fa è chiamare le funzioni sul modello. Quindi chiamiamo nel for la funzione `.train()`, poi calcoliamo l'output del modello e lo memorizziamo in `yhat` e poi calcoliamo la loss. Sulla loss che abbiamo calcolato possiamo chiamare la funzione backward e poi fare gli step di aggiornamento dei pesi utilizzando l'optimizer.

All'interno del codice chiamiamo anche la funzione `model.train()` che serve per informare il modello che ci stiamo spostando in training mode.

Notare che quando calcoliamo l'output del modello noi non chiamiamo la funzione forward ma andiamo ad utilizzare `model(x_train_pt)`, ovvero usiamo l'oggetto model e gli passiamo i nostri dati di training.

```
torch.manual_seed(42)
```

```

# Now we can create a model and send it at once to the device
model = LinearRegression().to(device)
# We can also inspect its parameters using its state_dict
print(model.state_dict())

lr = 1e-1
n_epochs = 1000

loss_fn = nn.MSELoss(reduction='mean')
optimizer = optim.SGD(model.parameters(), lr=lr)

for epoch in range(n_epochs):
    # What is this???
    model.train()

    yhat = model(x_train_pt)

    loss = loss_fn(y_train_pt, yhat)
    loss.backward()
    optimizer.step()
    optimizer.zero_grad()

print(model.state_dict())

```

Layers

Possiamo anche creare un modello annidato nel senso che definiamo la nostra classe LayerLinearRegression che rappresenta un modello e all'interno utilizziamo poi il layer nn.Linear al posto dei parametri custom che avevamo in precedenza. Questo livello lineare all'interno prende un input e produce un output e viene chiamato per eseguire la previsione.

```

class LayerLinearRegression(nn.Module):
    def __init__(self):
        super().__init__()
        # Instead of our custom parameters, we use a Linear layer with single input and sing
        self.linear = nn.Linear(1, 1)

    def forward(self, x):
        # Now it only takes a call to the layer to make predictions
        return self.linear(x)

```

Sequential

Anche in PyTorch, così come avviene con Keras, possiamo utilizzare un modello sequenziale per definire in modo semplice un modello. In questo caso possiamo creare il modello Sequenziale passando come singolo argomento un Linear Layer che utilizzeremo per fare il training della nostra linear regression. Il modello sarebbe così:

```
model = nn.Sequential(nn.Linear(1, 1)).to(device)
```

21.2 Tutorial

Vediamo ora un esempio più realistico per capire come utilizzare effettivamente Pytorch in qualcosa di reale nel campo del Machine Learning. Vedremo l'utilizzo di Pytorch con il dataset MNist, creeremo una neural network per il learning delle lettere scritte a mano.

Per prima cosa coi creiamo un nostro dataset, abbiamo in input delle immagini e gli output sono i label associati alle varie immagini.

```
class MNIST(Dataset):
    """
    A customized data loader for MNIST.

    def __init__(self,
                 root,
                 transform=None,
                 preload=False):
        """ Initialize the MNIST dataset

        Args:
            - root: root directory of the dataset
            - transform: a custom transform function
            - preload: if preload the dataset into memory
        """
        self.images = None
        self.labels = None
        self.filenames = []
        self.root = root
        self.transform = transform

        # read filenames
```

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

```

for i in range(10):
    filenames = glob.glob(osp.join(root, str(i), '*.png'))
    for fn in filenames:
        self.filenames.append((fn, i)) # (filename, label) pair

# if preload dataset into memory
if preload:
    self._preload()

self.len = len(self.filenames)

def _preload(self):
    """
    Preload dataset to memory
    """
    self.labels = []
    self.images = []
    for image_fn, label in self.filenames:
        # load images
        image = Image.open(image_fn)
        self.images.append(image.copy())
        # avoid too many opened files bug
        image.close()
        self.labels.append(label)

# probably the most important to customize.
def __getitem__(self, index):
    """
    Get a sample from the dataset
    """
    if self.images is not None:
        # If dataset is preloaded
        image = self.images[index]
        label = self.labels[index]
    else:
        # If on-demand data loading
        image_fn, label = self.filenames[index]
        image = Image.open(image_fn)

    # May use transform function to transform samples
    # e.g., random crop, whitening
    if self.transform is not None:
        image = self.transform(image)
    # return image and label
    return image, label

def __len__(self):
    """
    Total number of samples in the dataset

```

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

```
"""
return self.len
```

Il metodo più importante per la customizzazione del dataset è quello che ci permette di accedere un elemento all'interno del dataset. Questo è il metodo `__getitem__` e viene chiamato quando utilizziamo la bracket notation `list[3]`, in particolare `3` è il parametro che viene passato al metodo `__getitem__` e rappresenta la posizione che vogliamo accedere. Questo metodo controlla la lista delle immagini, trasforma i dati in un tensore e poi mi restituisce l'immagine e il tensore.

Presi i nostri dati di training li passiamo alla classe MNIST che abbiamo definito e che li memorizzerà.

```
trainset = MNIST(
    root='data/mnist_png/training',
    preload=True, transform=transforms.ToTensor(),
)

# Use the torch dataloader to iterate through the dataset
# We want the dataset to be shuffled during training.
trainset_loader = DataLoader(trainset, batch_size=64, shuffle=True, num_workers=1)

# Load the testset
testset = MNIST(
    root='data/mnist_png/testing',
    preload=True, transform=transforms.ToTensor(),
)
# Use the torch dataloader to iterate through the dataset
testset_loader = DataLoader(testset, batch_size=1000, shuffle=False, num_workers=1)
```

In questo modo produciamo un dataset per i nostri dati di training e uno per i dati di test. Notare che prima di fare altro, tramite PyTorch possiamo andare ad controllare se il computer su cui vogliamo eseguire il training ha una GPU adeguata o meno.

```
# Use GPU if available, otherwise stick with cpu
use_cuda = torch.cuda.is_available()
torch.manual_seed(123)
device = torch.device("cuda" if use_cuda else "cpu")
print(device)
```

A questo punto noi vogliamo definire una Convolutional Neural Network e lo facciamo definendo una classe:

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

```

class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()

        # Conv2d(in_channels, out_channels, kernel_size, stride=1, padding=0,
        #         dilation=1, groups=1, bias=True, padding_mode='zeros')
        self.conv1 = nn.Conv2d(1, 10, kernel_size=5)
        self.conv2 = nn.Conv2d(10, 20, kernel_size=5)
        self.conv2_drop = nn.Dropout2d()

        # Linear(in_features, out_features, bias=True)
        self.fc1 = nn.Linear(320, 50)
        self.fc2 = nn.Linear(50, 10)

        # MaxPool2d(kernel_size, stride=None, padding=0, dilation=1, return_indices=False, ceil_mode=False)
        self.max_pool = nn.MaxPool2d(2)
        # ReLU(inplace=False)
        self.relu = nn.ReLU()

    def forward(self, x):
        # Note: the following two ways for max pooling / relu are equivalent.
        # 1) with torch.nn.functional:
        x = F.relu(F.max_pool2d(self.conv1(x), 2))
        # 2) with torch.nn:
        x = self.relu(self.max_pool(self.conv2_drop(self.conv2(x))))
        x = x.view(-1, 320)
        x = self.fc1(x)
        x = F.dropout(x, training=self.training)
        x = self.fc2(x)
        return F.log_softmax(x, dim=1)

```

Qua nel metodo init stiamo semplicemente creando i layer che vogliamo utilizzare all'interno della rete ma al momento non è importante l'ordine.

Dove è importante l'ordine è il metodo forward in cui andiamo a specificare cosa vogliamo fare all'interno della nostra rete neurale. Qua noi andiamo a scrivere solamente la forward mentre la backward viene creata in automatico da Pytorch.

Il modello che utilizzeremo sarà una istanza di questa classe Net che abbiamo definito, quando utilizziamo un oggetto direttamente senza utilizzare (Abbiamo un call method nella classe Net che chiama il forward method, questo metodo viene ereditato dalla nn.Module).

```

model = Net().to(device)
optimizer = optim.SGD(model.parameters(), lr=0.001, momentum=0.9)

```

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

Ora abbiamo il dataset, il modello e possiamo eseguire il training, scriviamo una funzione per eseguire il training, la funzione train prende come parametro il nostro modello che è una istanza della classe che abbiamo definito. Andiamo in train mode e ripetiamo un numero di step di training che dipende dal numero di epochs che abbiamo scelto. Prendiamo i dati in batches utilizzando il *trainset_loader* che crea questi set di dati, Pytorch lavora sempre utilizzando batch di dati, mai con il dato singolo.

```
from time import time
def train(model, epoch, log_interval=100):
    model.train()  # set training mode
    iteration = 0
    for ep in range(epoch):
        start = time()
        for batch_idx, (data, target) in enumerate(trainset_loader):
            # bring data to the computing device, e.g. GPU
            data, target = data.to(device), target.to(device)

            # forward pass
            output = model(data)
            # compute loss: negative log-likelihood
            loss = F.nll_loss(output, target)

            # backward pass
            # clear the gradients of all tensors being optimized.
            optimizer.zero_grad()
            # accumulate (i.e. add) the gradients from this forward pass
            loss.backward()
            # performs a single optimization step (parameter update)
            optimizer.step()

            if iteration % log_interval == 0:
                print('Train_Epoch:{}[{}]/{}({:.0f}%) \t Loss: {:.6f}'.format(
                    ep, batch_idx * len(data), len(trainset_loader.dataset),
                    100. * batch_idx / len(trainset_loader), loss.item()))
            iteration += 1

        end = time()
        print('{:.2f}s'.format(end - start))
        test(model) # evaluate at the end of epoch
```

Il modello viene usato come una funzione, infatti per eseguire il forward pass gli passiamo i nostri dati tramite la funzione *model(data)* e

questa produce un output. All'interno della funzione train del nostro modello viene utilizzato anche l'optimizer che avevamo definito in precedenza, viene utilizzato lo stochastic gradient descent, questo capisce in automatico quali sono i parametri che devono essere ottimizzati.

Quando vogliamo valutare il nostro modello non vogliamo calcolare il gradiente, quindi andiamo a specificare `torch.no_grad()`:

```
def test(model):
    model.eval()    # set evaluation mode
    test_loss = 0
    correct = 0
    with torch.no_grad():
        for data, target in testset_loader:
            data, target = data.to(device), target.to(device)
            output = model(data)
            test_loss += F.nll_loss(output, target, size_average=False).item() # sum up batch loss
            prob, predicted = output.max(1)
            correct += (predicted == target).sum().item()

    test_loss /= len(testset_loader.dataset)
    print('\nTest set: Average loss: {:.4f}, Accuracy: {} / {} ({:.0f}%)\n'.format(
        test_loss, correct, len(testset_loader.dataset),
        100. * correct / len(testset_loader.dataset)))
```

Dalla rete noi otteniamo 10 output perchè otteniamo la distribuzione di probabilità sui 10 possibili caratteri che abbiamo nel dataset. In particolare noi dobbiamo andare a prendere il massimo di queste probabilità. Per ognuno dei batches che stiamo considerando andiamo a calcolare quanti sono le immagini che vengono classificate correttamente e quelle che invece sono classificate male. Poi sommiamo alla fine tutti i risultati che abbiamo ottenuto per cercare di ottenere una valutazione complessiva su tutti i dati di test.

Quando abbiamo completato il training vorremmo utilizzare il nostro modello ma chiaramente non possiamo eseguire nuovamente il training ogni volta che lo vogliamo utilizzare inoltre, se utilizziamo un modello particolarmente grande e complesso da trainare magari vorremmo avere dei checkpoint in modo da poter tornare indietro a quel checkpoint quando qualcosa dovesse andare male.

Se abbiamo il controllo sul processo di training possiamo quindi creare

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

dei checkpoint e questo è utile anche quando vogliamo salvare un modello che apparentemente sta funzionando bene sul validation set e magari andando avanti con altri dati potremmo peggiorare, quindi mettiamo un checkpoint che tra l'altro ci permette di avere il nostro modello salvato anche per quando ci si spegne all'improvviso il computer e magari non vogliamo rifare il training da 0.

21.2.1 Fine Tuning

In alcuni casi potrebbe essere interessante ed utile partire da un modello su cui è già stato fatto il training e andare a eseguire il training nuovamente però con dati nuovi in modo da avere già dei pesi di partenza che magari devono essere modificati di poco. Questa operazione si chiama finetuning e la possiamo fare ad esempio anche se abbiamo salvato un modello tramite un checkpoint:

```
# What's in a state dict?
print(model.state_dict().keys())
```

Tramite la funzione sopra possiamo vedere cosa abbiamo salvato all'interno del nostro modello con il checkpoint:

```
odict_keys(['conv1.weight', 'conv1.bias', 'conv2.weight', 'conv2.bias', 'fc1.weight', 'fc1.bias'])
```

Possiamo decidere che vogliamo eseguire di nuovo il training di alcuni di questi layer mentre altri layer vanno bene così come sono.

Possiamo anche utilizzare dei pesi pre-trained all'interno del modello che definiamo:

```
class SmallNet(nn.Module):
    def __init__(self):
        super(SmallNet, self).__init__()
        # same conv layers
        self.conv1 = nn.Conv2d(1, 10, kernel_size=5)
        self.conv2 = nn.Conv2d(10, 20, kernel_size=5)
        self.conv2_drop = nn.Dropout2d()
        # fewer FC layers
        self.fc1 = nn.Linear(320, 10)

    def forward(self, x):
```

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

```

x = F.relu(F.max_pool2d(self.conv1(x), 2))
x = F.relu(F.max_pool2d(self.conv2_drop(self.conv2(x)), 2))
x = x.view(-1, 320)
x = self.fc1(x)
return F.log_softmax(x, dim=1)

model = SmallNet().to(device)
optimizer = optim.SGD(model.parameters(), lr=0.001, momentum=0.9)

```

21.2.2 Clean Code

Possiamo ottenere un codice più pulito se utilizziamo un sequence model andando a mettere uno sopra all'altro tutti i layer che abbiamo nominato in precedenza. In questo modo abbiamo una descrizione più semplice e pulita del nostro modello.

```

class NetSeq(nn.Module):
    def __init__(self):
        super(NetSeq, self).__init__()

        # conv layers: feature extractor
        self.conv_layers = nn.Sequential(
            nn.Conv2d(1, 10, kernel_size=5),
            nn.MaxPool2d(2),
            nn.ReLU(),
            nn.Conv2d(10, 20, kernel_size=5),
            nn.Dropout2d(),
            nn.MaxPool2d(2),
            nn.ReLU()
        )

        # fc layers: classifier
        self.fc_layers = nn.Sequential(
            nn.Linear(320, 50),
            nn.ReLU(),
            nn.Dropout(),
            nn.Linear(50, 10),
        )

    def forward(self, x):
        x = self.conv_layers(x)
        x = x.view(-1, 320)
        x = self.fc_layers(x)
        return F.log_softmax(x, dim=1)

```

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

```
model = NetSeq().to(device)
optimizer = optim.SGD(model.parameters(), lr=0.001, momentum=0.9)
```

Chapter 22

Question Answering

Il Question Answering è un altro dei campi in cui l'utilizzo delle Neural Network ha portato grandi novità e miglioramenti. Il QA ha anche molti possibili utilizzi commerciali e questo è uno dei motivi per cui c'è molto interesse in questo campo negli ultimi anni.

22.1 Introduzione

Ci sono tantissime persone che cercano sul web delle risposte a delle domande, possiamo ad esempio andare su google e fare una ricerca e google mi restituirà la risposta alla mia domanda.



Notare che in alcuni casi le risposte che fornisce Google, in particolare le più dirette (ad esempio chi è il regista di un film), provengono dal knowledge Graph di Google. Non funziona così con tutte le risposte che può fornirci Google, ad esempio nell'immagine sopra viene estratta la risposta presa direttamente da Wikipedia.

Le motivazioni per il question answering:

- Quando cerchiamo qualcosa non è troppo utile sapere solamente che quella risposta è presente all'interno di alcuni documenti che sono più rilevanti di altri perché magari quei documenti potrebbero essere troppo lunghi e non posso leggerli tutti.
- Noi vorremmo dei sistemi in grado di rispondere direttamente alle domande, specialmente se siamo in mobilità, non avrebbe senso infatti ricevere una lista di documenti rilevanti in quel caso.

Tipicamente per rispondere alle domande ci si concentra su due percorsi:

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

- Trovare il documento che sia interessante per il nostro caso e che quindi possa contenere la risposta alla nostra domanda. Per questo primo step possiamo utilizzare tecniche di information retrieval, anche tecniche tradizionali che sono scalabili anche con miliardi di documenti.
- Poi si deve andare all'interno di questo documento per trovare esattamente la frase che contiene la risposta alla nostra domanda. Questo problema viene chiamato **Reading Comprehension** perché consiste nel leggere il documento e comprenderne il contenuto.

Nel question answering abbiamo una domanda che è scritta in linguaggio naturale, non abbiamo solamente una keyword che vogliamo cercare. È differente dall'information retrieval, bisogna andare uno step oltre in questo caso. Possiamo utilizzare i vari task di AI e NLP per eseguire il task di question answering, questo task infatti dovrebbe essere capace di fare varie cose per arrivare all'obiettivo finale, ad esempio deve poter fare il POS tagging o il NER o anche una traduzione. In generale quindi il question answering è un task molto complesso.

22.1.1 Open Domain QA

Oltre al QA in cui abbiamo semplicemente una domanda e un paragrafo in cui troviamo la risposta esiste anche un Open Domain QA.

Ci interessiamo dell'Open Domain QA perché se restringiamo il contesto in cui siamo interessati allora diventa tutto più semplice. Storicamente già dal 1964 sono state fatte delle prove in questo campo di QA, si prendeva un certo documento e poi si andava a rispondere alla domanda basandosi su un dependency parsing di domanda e risposta. Poi si è iniziato a produrre sistemi migliori per l'information retrieval, e nel 1999 la community di information retrieval è iniziata a diventare sempre più importante nel campo del QA. Spostandosi dall'information retrieval

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

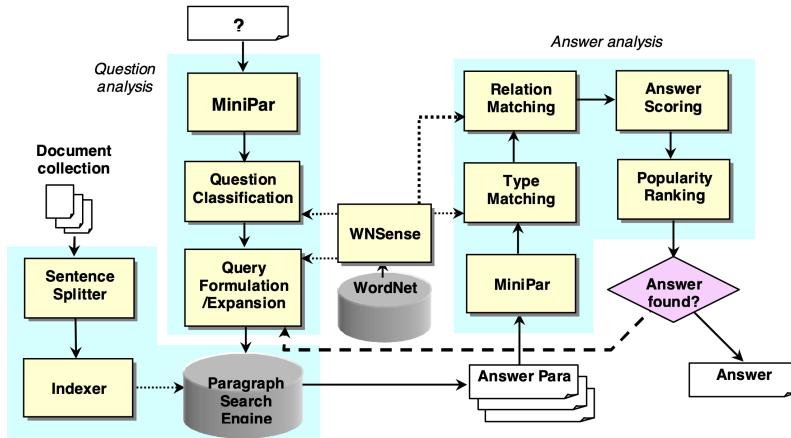
al QA vuol dire che ci serve una capacità di Natural language processing e non ci basta più fare solamente un match dei nostri dati. Intorno al 2002 un team dell'IBM portò risultati molto interessanti e l'IBM decise di continuare a investire molto nel campo del QA creando un team per creare sistemi capaci di rispondere alle domande in real time. Questo ha portato al sistema che viene chiamato Watson che è stato in grado nel 2011 di battere il campione umano di un quiz. Watson all'epoca non utilizzava le tecnologie del Deep Learning che sono state rilasciate qualche anno dopo. Negli anni successivi e in particolare nel 2016 è stato introdotto un sistema che utilizza il deep learning, quindi anche in questo caso c'è stata un rivoluzione nel campo del QA grazie all'introduzione del deep learning.

22.2 Trec QA (2001-2006)

PiQASso è un sistema di QA prodotto all'UniPi, si basa su una combinazione di tecniche di Information retrieval e una serie di semantic filters per selezionare i paragrafi che contengono risposte interessanti.

La struttura del sistema è la seguente ed è composta da due componenti fondamentali che sono:

- Un componente per l'indicizzazione e la parte di retrieval.
- Un componente per il sistema di QA



Per quanto riguarda la parte di Question Analysis abbiamo la domanda che viene analizzata con un dependency parsing che produce un parse tree che la rappresenta. La domanda a cui dobbiamo rispondere può essere di vario tipo, possiamo rispondere "Chi?", "Dove?" o altro, in base ad ognuna di queste specifiche keyword possiamo rispondere in modo differente.

Per quanto riguarda la preparazione della risposta, dobbiamo preparare la collezione di documenti. Invece di restituire documenti vogliamo dei paragrafi perché le risposte le troviamo in paragrafi. Quindi il documento viene diviso in paragrafi e poi viene prodotto un indice. Questi paragrafi che produciamo vengono memorizzati all'interno del Paragraph Search Engine.

Abbiamo poi un paragraph retrieval system che presi i paragrafi va a fare un match per capire dove si trova la risposta. Da questo match possiamo estrarre dei candidate answer e poi dobbiamo capire quali di questi candidate answer sono migliori. Questo ci fornisce uno score per le varie risposte e alla fine ne estraiamo una.

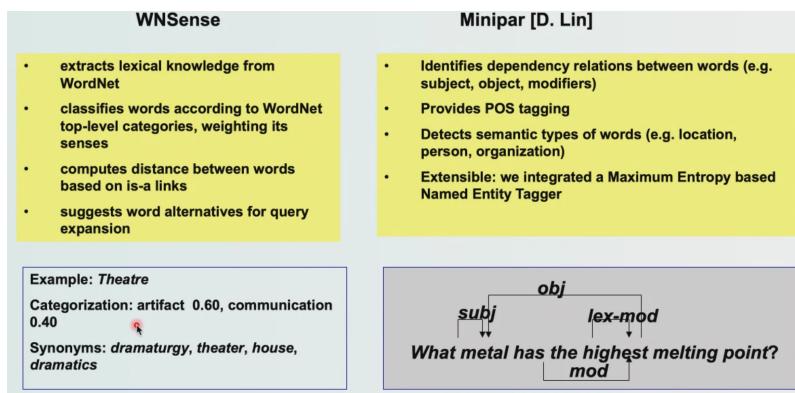
I tool linguistici che sono stati utilizzati sono:

- Un dependency parser chiamato Minipar che utilizziamo per il parsing. Utilizzando Minipar si va a produrre un dependency tree che

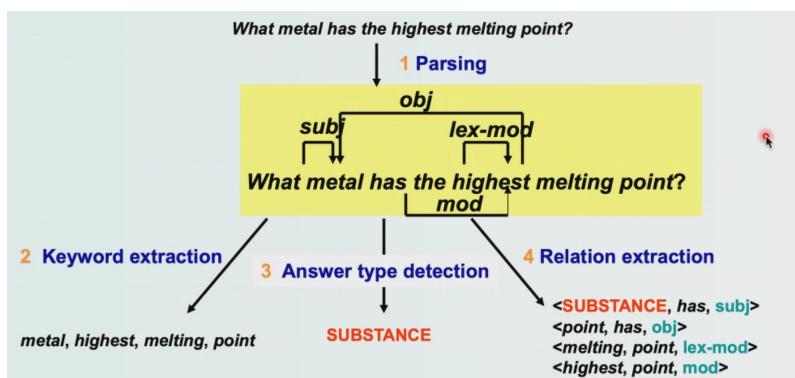
Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

rappresenta le relazioni presenti all'interno della frase. Questa dependency relation è una relazione tra una parola chiamata head e una serie di modifier. Ognuno di questi modifier è collegato (modifica) solamente un head.

- WNSense che è una interfaccia per accedere a WordNet in modo da estrarre la conoscenza lessicale.
- Un Named Entity Tagger.



Vediamo un esempio di domanda:

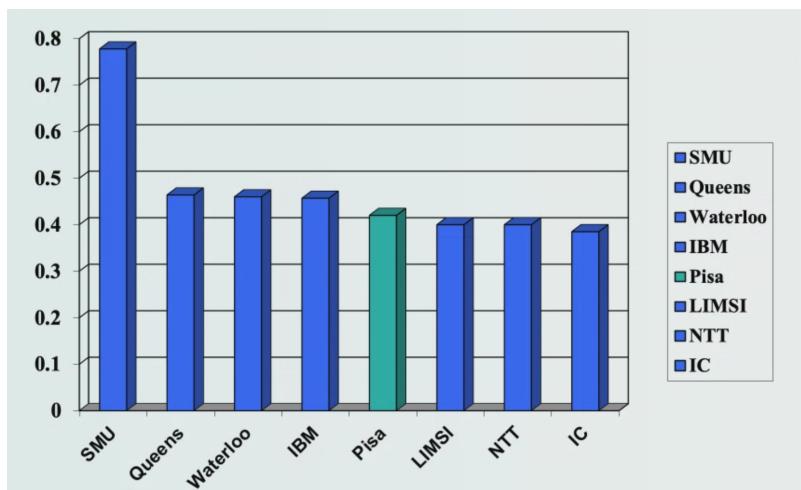


- Abbiamo la nostra domanda che viene parsata andando a produrre il parse tree.

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

- Poi ci focalizziamo sulla ricerca di alcune parole chiavi all'interno della frase e questo lo facciamo tramite il POS Tagging.
- Si cerca di capire tramite delle euristiche il tipo di domanda a cui vogliamo rispondere.
- Poi si estraggono le relazioni dal parse tree.
- Alla fine si identifica la risposta alla domanda.

Questo è come si è posizionato il team di Pisa nella competizione:



Quello che colpisce è il sistema dell'università del Texas SMU, questo ha avuto uno score di circa 76% ed era assurdo all'epoca.

Il sistema di Pisa però era abbastanza lento, per rispondere alla domanda era necessario qualche minuto, quindi bene per una competizione ma non utilizzabile nella vita reale.

22.3 Answer Selection

Nel campo del QA un compito abbastanza importante consiste nella selezione della risposta più probabile in un set di candidati. Sono state pre-

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

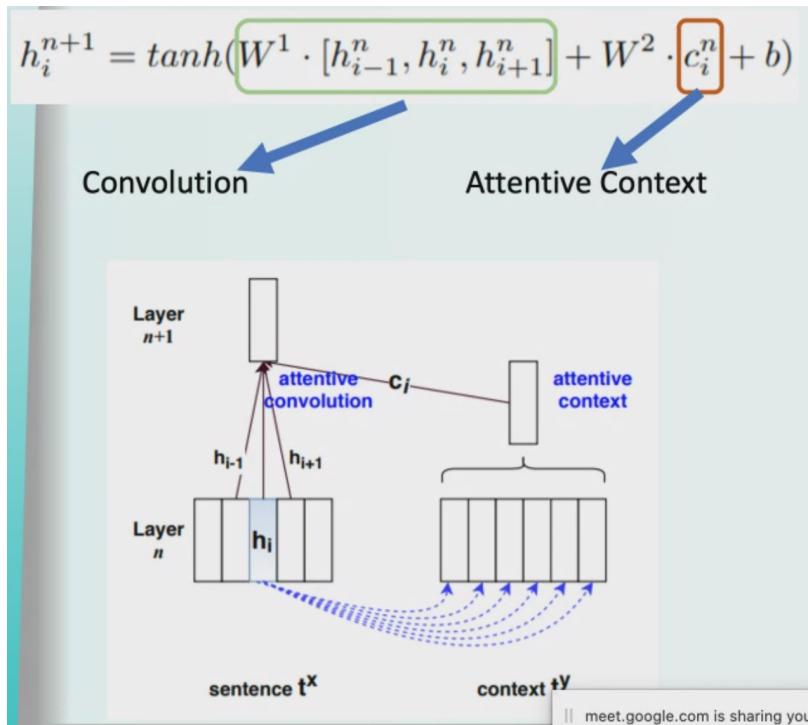
sentate varie possibili soluzioni per questo problema, abbiamo l'Attentive Pooling che utilizza LSTM e CNN, abbiamo la Lexical Decomposition e Composition e l'Attentive Convolution.

Tra le varie possibili soluzioni vediamo quella prodotta per la Challenge di Fujitsu due anni fa da altri studenti UniPi.

Il task da risolvere in questa challenge consisteva nello sviluppo di un sistema di QA sul dataset SelQA, si tratta di un dataset e di domande che richiedono qualcosa di più della semplice Information Retrieval per rispondere.

22.3.1 L'approccio utilizzato

Per risolvere la challenge è utilizzato un meccanismo di attention per la selezione delle risposte corrette. Il punto di partenza è stato utilizzare questo meccanismo all'interno di una neural network. Per ogni parola presente all'interno della frase andiamo a considerare il suo contesto e produciamo un "attentive context". Questo attentive context viene dato in input al layer successivo pesandolo e sommangolo alle parole presenti all'interno della frase.

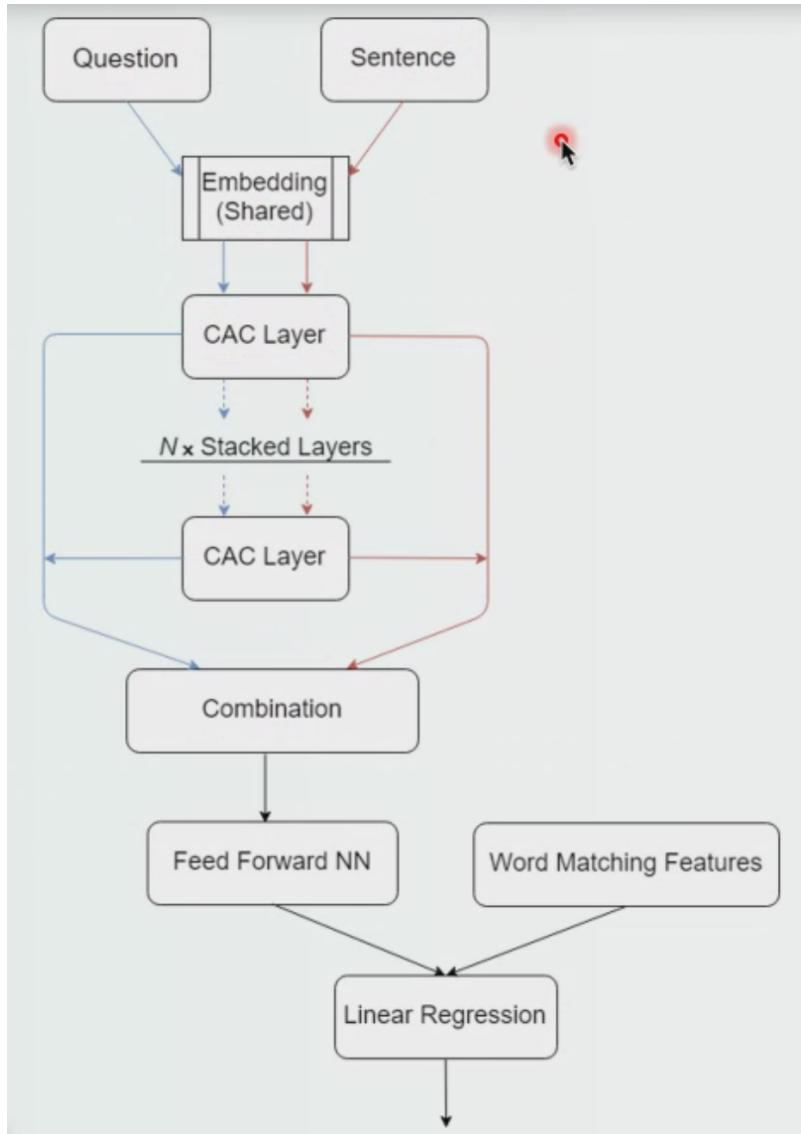


In particolare la formula vera e propria che viene utilizzata per produrre l'input del livello successivo è:

$$h_i^{n+1} = \tanh(W^1 \cdot [h_{i-1}^n, h_i^n, h_{i+1}^n] + W^2 \cdot c_i^n + b)$$

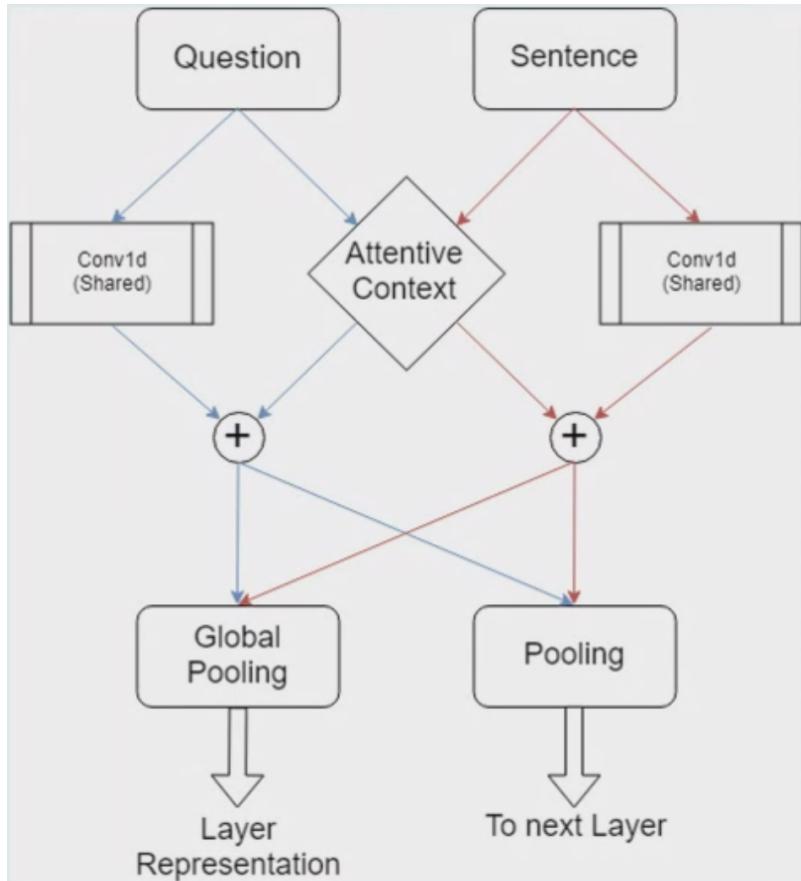
Dove la c_i^n è l'attentive context e la parte $W^1 * [h_{i-1}^n, h_i^n, h_{i+1}^n]$ rappresenta invece la convoluzione.

Questo era lo starting point, poi hanno prodotto un approccio più complesso chiamato Stacked Cross Attentive Convolutional Layers.



In particolare all'interno del Cross Attentive Convolution Layer abbiamo che viene applicata la Light Attentive Convolution in entrambe le direzioni nel senso che domanda e risposta si conoscono a vicenda. Da ognuno di questo layer estraiamo una layer representation.

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.



Ognuna di queste fa una combinazione di quello che arriva in input e poi produce un output che viene combinato e concatenato. Abbiamo due output da ogni cross attentive convolution che sono il global pooling e l'individual pooling che vanno a finire nel prossimo layer. Alla fine tutti questi risultati intermedi vengono combinati e abbiamo alla fine una Feed Forward NN che produce uno score finale per la frase rispetto alla domanda.

Questi sono i risultati che hanno ottenuto:

WikiQA

	Validation MRR	Test MRR
Cross-Attentive CNN	72.07 %	72.12 %
CNN SelQA	70.18 %	70.82 %
RNN SelQA	68.31 %	68.92 %
Yang et al.	-	66.52 %
Wang et al.	-	72.26 %

SelQA

	Validation MRR	Test MRR
Cross-Attentive CNN	91.37 %	90.61 %
CNN SelQA	86.67 %	85.68 %
RNN SelQA	88.25 %	87.59 %

I risultati sono migliori rispetto agli altri che abbiamo con altri sistemi, quindi l'attentive mechanism è risultato molto interessante in questo caso. Quello che il sistema fa è, dato un sistema di possibili domande utilizza la rete e per ogni frase viene prodotto un possibile candidate answer e uno score per quel candidate answer. Alla fine scegliamo poi la frase associata allo score più alto. Questo fa molto bene in termini di question selection.

È anche interessante capire cosa fa il sistema in base al tipo di argomento che abbiamo all'interno della domanda:

	Country	Food	Art	Movies	Music	History	Science	Sport	Travel	TV
Dev	94.22	91.08	88.58	91.36	89.43	93.41	93.93	88.93	90.75	90.12
Test	95.32	89.1	86.56	88.86	87.81	94.52	90.95	89.57	91.51	83.41

Come si può vedere in base al tipo di argomento di cui si parla il sistema si comporta in modo differente.

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

22.4 Reading Comprehension

Con l'introduzione del Deep Learning c'è stata una rivoluzione anche nel campo del QA. Per eseguire il training di un sistema di questo genere abbiamo però bisogno di dataset molto grandi. Nel 2015 poi sono stati introdotti i primi dataset per il QA e nel 2016 un dataset chiamato SQuAD è stato prodotto da Stanford. L'idea di Bruges nel 2013 è stata quella di introdurre un task di machine comprehension è quella di misurare il contenuto di un certo paragrafo, se siamo in grado di leggere e comprendere il contenuto dovremmo essere in grado di rispondere a qualsiasi domanda che un'altra persona può chiederci riguardo a quella domanda, se la macchina è in grado di rispondere nello stesso modo alla domanda allora questa potrebbe essere considerata una buona metrica per valutare il funzionamento di un sistema di questo genere.

Nel caso della comprensione del documento, possiamo fare tante cose senza neanche capire il significato delle parole perché assembliamo le varie parole e le varie frasi e possiamo ottenere ad esempio delle traduzioni anche fatte bene però rimane il problema del non comprendere il significato. Per comprendere anche il significato dobbiamo inventare un test e il Machine Comprehension è uno di questi possibili test. Qua vediamo un esempio dell'MCTest Reading Comprehension:

Passage (P) + Question (Q) → Answer (A)

P Alyssa got to the beach after a long trip. She's from Charlotte. She traveled from Atlanta. She's now in Miami. She went to Miami to visit some friends. But she wanted some time to herself at the beach, so she went there first. After going swimming and laying out, she went to her friend Ellen's house. Ellen greeted Alyssa and they both had some lemonade to drink. Alyssa called her friends Kristin and Rachel to meet at Ellen's house.....

Q Why did Alyssa go to Miami? A To visit some friends

Abbiamo in basso a sinistra la nostra domanda, abbiamo il nostro paragrafo da cui possiamo estrarre la risposta che sta in basso a destra. Non è ovvio il modo in cui possiamo estrarre questa risposta.

In generale per il task del QA come abbiamo visto nelle competizioni precedenti, possiamo dividere il problema in subtask:

- Question analysis
- Query generation and retrieval: si prende dalle informazioni che abbiamo selezionato. Questo produce una lista di candidate answer.
- Poi facciamo una selezione di risposte e scegliamo quella che corrisponde di più alla domanda. Abbiamo la domanda e vogliamo fare un rank.
- Per selezionare la risposta potrebbe non essere ovvio perché magari non c'è il nome della persona che ci interessa ma troviamo un pronome come ad esempio "lei", anche il verbo che usiamo nella domanda potrebbe essere diverso rispetto al verbo che troviamo all'interno del paragrafo, in ogni caso queste cose dovremmo essere in grado di comprenderle e di risolvere i problemi.

22.5 SQuAD

Un altro punto importante nella storia del QA è stato il rilascio di SQuAD.

Question: Which team won Super Bowl 50?

Passage

Super Bowl 50 was an American football game to determine the champion of the National Football League (NFL) for the 2015 season. The American Football Conference (AFC) champion Denver Broncos defeated the National Football Conference (NFC) champion Carolina Panthers 24–10 to earn their third Super Bowl title. The game was played on February 7, 2016, at Levi's Stadium in the San Francisco Bay Area at Santa Clara, California.

100k examples

Answer must be a span in the passage

A.k.a. extractive question answering

Un problema della prima versione di SQuAD è che per ogni domanda era presente una risposta all'interno del paragrafo, questo "problema" è stato però risolto con SQuAD 2.0 in cui per alcune delle domande presenti all'interno del Corpus non sono presenti delle risposte.

22.6 Conclusione

QA è un task abbastanza complesso, dobbiamo prendere i vari documenti, trovare candidati, selezionare il candidato e restituire la risposta.

Vedremo poi in che modo i QA componenti possono fare parte di Dialog Systems ovvero di Chatbot che possono partecipare ad una discussione con un umano.

In un chatbot e in un dialog system non abbiamo una collezione fissata di domande.

È un sistema più dinamico che deve rappresentare anche una memoria di quello che ho detto per produrre una risposta.

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

Chapter 23

Reading Comprehension

All'interno del campo del Question Answering abbiamo vari possibili task:

- Abbiamo l'Open Domain Question Answering, per rispondere alle domande non ci basta la conoscenza dei documenti che abbiamo analizzato ma dobbiamo basarci anche su una memoria esterna che consiste in documenti molto grandi che possono essere processati per creare un sistema di Information Retrieval.
- Answer Selection: l'obiettivo è scegliere la migliore risposta possibile all'interno di una lista di possibili risposte.
- Reading Comprehension: vogliamo trovare la risposta alla nostra domanda all'interno di un paragrafo che viene fornito senza fare preprocessing e senza avere memoria.
- Dialog Systems: dobbiamo fornire risposte alle domande e dobbiamo anche tenere memoria del dialogo che abbiamo avuto e delle precedenti domande e risposte.

23.1 Reading Comprehension

Avevamo già introdotto, parlando del QA, il task di reading comprehension, molte persone hanno lavorato su questo tipo di task per molti anni cercando di trovare delle possibili soluzioni al problema. La svolta c'è stata nel 2015 quando è stato introdotto il Deep Learning. I primi dataset sono stati creati da DeepMind utilizzando articoli della CNN e del Daily Mail, poi è stato sviluppato SQuAD da alcuni ricercatori di Stanford. SQuAD è migliore del primo dataset che abbiamo nominato, partendo da questo poi ne sono stati sviluppati anche altri come ad esempio MS Marco.

Negli ultimi anni, grazie alla disponibilità di grandi dataset per fare il training c'è stato molto interesse in questo campo della reading comprehension.

Vediamo un primo esempio considerando un test di Machine Comprehension, abbiamo a disposizione una domanda e un paragrafo che contiene la risposta a quella domanda. L'obiettivo è trovare all'interno del paragrafo la risposta alla mia domanda, in questo caso dovremmo trovare la frase "To visit some friends".

Passage (P)	+	Question (Q)	→	Answer (A)
<p>(P) Alyssa got to the beach after a long trip. She's from Charlotte. She travelled from Atlanta. She's now in Miami. She went to Miami to visit some friends. But she wanted some time to herself at the beach, so she went there first. After going swimming and laying out, she went to her friend Ellen's house. Ellen greeted Alyssa and they both had some lemonade to drink. Alyssa called her friends Kristin and Rachel to meet at Ellen's house.....</p>				
(Q)	Why did Alyssa go to Miami?			(A) To visit some friends

23.1.1 SQuAD 1.1

Per 4 o 5 anni lo SQuAD è stato il benchmark per i sistemi di Reading Comprehension, conteneva anche articoli di Wikipedia annotati con do-

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

mande e risposte. In questo caso noi consideriamo un esempio relativo al Superbowl, abbiamo un paragrafo e una domanda. La risposta alla nostra domanda è sempre contenuta all'interno del paragrafo e in particolare sarà una sottosequenza del paragrafo che spesso viene chiamata span.

Question: Which team won Super Bowl 50?

Passage

Super Bowl 50 was an American football game to determine the champion of the National Football League (NFL) for the 2015 season. The American Football Conference (AFC) champion Denver Broncos defeated the National Football Conference (NFC) champion Carolina Panthers 24–10 to earn their third Super Bowl title. The game was played on February 7, 2016, at Levi's Stadium in the San Francisco Bay Area at Santa Clara, California.

In questo caso la risposta alla nostra domanda sarà "Denver Broncos".

Nella prima versione di SQuAD erano presenti 100.000 esempi.

In generale anche se noi rispondiamo con lo span possono esserci delle variazioni perchè non sappiamo mai quanto può essere lungo lo span che dobbiamo considerare:

Private schools, also known as independent schools, non-governmental, or nonstate schools, are not administered by local, state or national governments; thus, they retain the right to select their students and are funded in whole or in part by charging their students tuition, rather than relying on mandatory taxation through public (government) funding; at some private schools students may be able to get a scholarship, which makes the cost cheaper, depending on a talent the student may have (e.g. sport scholarship, art scholarship, academic scholarship), financial need, or tax credit scholarships that might be available.

Along with non-governmental and nonstate schools, what is another name for private schools?

Gold answers: ① independent ② independent schools ③ independent schools

Along with sport and art, what is a type of talent scholarship?

Gold answers: ① academic ② academic ③ academic

Rather than taxation, what are private schools largely funded by?

Gold answers: ① tuition ② charging their students tuition ③ tuition

Nell'esempio della frase sopra abbiamo varie possibili risposte per la nostra domanda che sono tutte corrette se leggiamo il paragrafo.

Valutazione di SQuAD 1.1

Nell'esempio di prima abbiamo ottenuto 3 risposte, consideriamo queste tre "gold answers" e vogliamo capire quale di queste è la risposta migliore che possiamo dare. Questo sistema viene valutato utilizzando due metriche considerando sempre per ogni domanda 3 gold answers:

- Exact Match: se la risposta alla domanda che viene data dal sistema è una delle tre "gold answers" allora otteniamo un punto mentre se non è una di queste abbiamo 0 punti. L'accuracy che otteniamo qua è la percentuale delle risposte che otteniamo correttamente.
- La seconda metrica è l'F1, qua noi andiamo a fare una match a livello di parole per le varie possibili risposte. Abbiamo la nostra risposta e le gold answers del sistema che vengono considerate come bag of word e consideriamo:

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

- La precision: la percentuale di parole della nostra risposta che stanno anche nel gold span.
- La recall: la percentuale di parole nel gold span che sono anche nella risposta data dal nostro sistema.
- F1 Score: calcoliamo la media armonica di Recall e Precision e questo ci fornisce uno score.

Per ogni domanda calcoliamo l’F1 Score e prendiamo quello maggiore, poi dato che abbiamo un insieme di domande, andiamo a fare una media di questo F1 Score con tutte le domande e in questo modo otteniamo il nostro F1 Score complessivo.

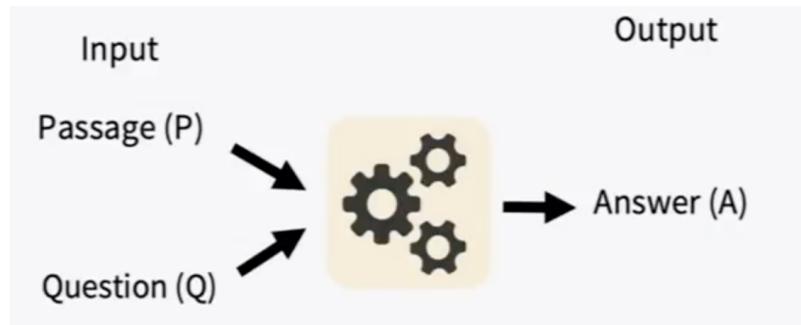
Entrambe le metriche ignorano la presenza di articoli e di punteggiatura all’interno delle risposte.

Tra i due sistemi l’F1 Score è più affidabile perchè se usiamo l’Exact Match abbiamo comunque solamente 3 gold answers che non è un sample esageratamente grande.

Nel corso degli anni sono stati sviluppati sistemi sempre più complessi e migliori in grado di ottenere ottimi risultati con lo SQuAD Dataset, in particolare un modello prodotto da Google che utilizza Bert è stato in grado di ottenere risultati migliori dell’uomo con lo SQuAD Dataset e le metriche appena descritte.

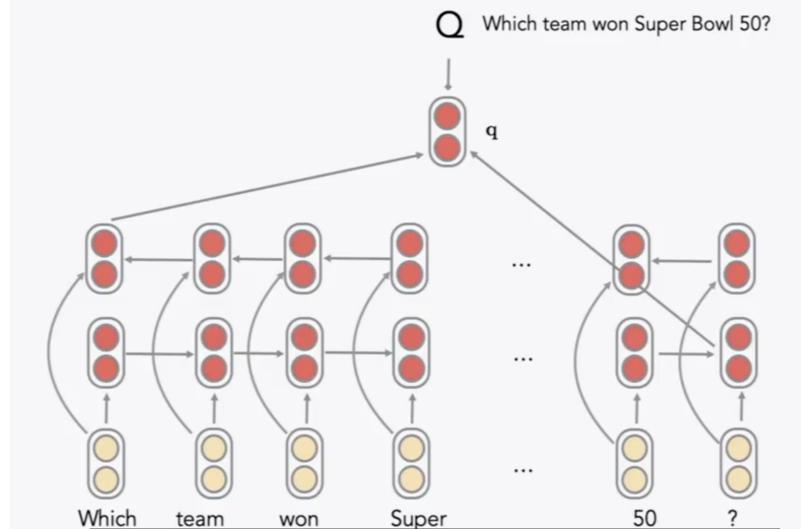
23.2 Stanford Attentive Reader

Lo Stanford Attentive Reader è il più semplice sistema di Neural QA che funziona in modo decente, ha dimostrato che con una semplice architettura è possibile ottenere buoni risultati per la reading comprehension e per la question answering. È stata chiamata Stanford Attentive Reader perchè utilizza l’attention all’interno del task.



Come funziona?

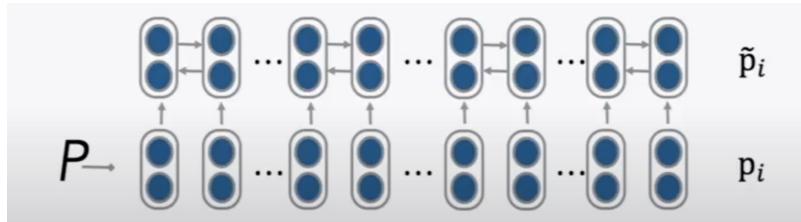
Prima di tutto abbiamo una domanda che viene data in input, quello che facciamo è creare una rappresentazione di questa domanda sotto forma di vettore. Per creare questa rappresentazione per ogni parola all'interno della parola creiamo un word embedding poi usiamo l'LSTM sulla domanda, prima da sinistra a destra e poi da destra a sinistra e poi alla fine prendiamo lo stato N-esimo delle due LSTM e li concateniamo in un vettore finale che sarà la rappresentazione della nostra domanda.



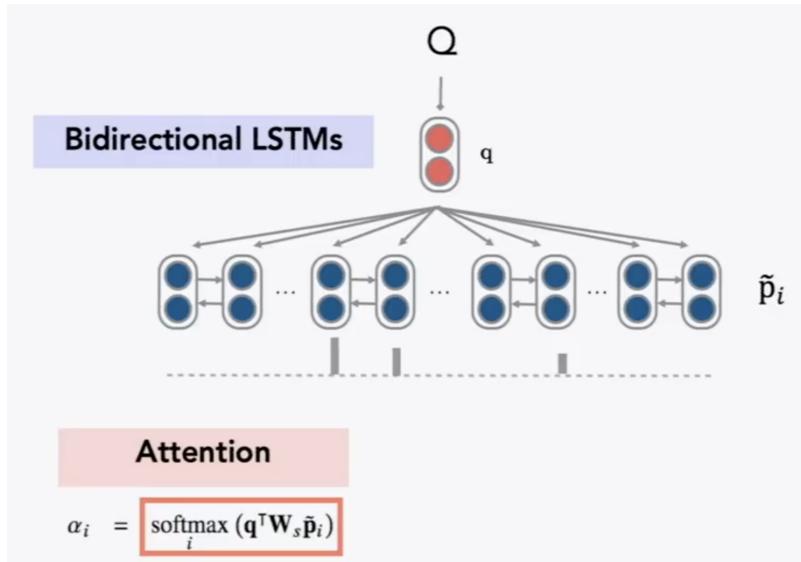
Una volta che abbiamo creato la rappresentazione della domanda possiamo passare al paragrafo, per il paragrafo facciamo la stessa cosa, ab-

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

biamo gli embedding per ognuna delle parole poi utilizziamo una LSTM bidirezionale, poi però dobbiamo fare un lavoro aggiuntivo perchè dobbiamo trovare la risposta all'interno del paragrafo.



Quello che faremo è fare la Question Representation per capire dove si trova la risposta nel paragrafo utilizzando l'attention. Questo è un utilizzo differente rispetto all'attention della Machine Translation. Quello che facciamo in questo caso è calcolare un attention score considerando sia la domanda sia la rappresentazione del paragrafo che otteniamo con la bidirectional LSTM.



Il calcolo di questo attention score viene effettuato in questo modo:

$$\alpha_i = softmax_i(q^T W_s p_i)$$

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

dove la q^T è il vettore relativo alla domanda, la p_i è il vettore per una particolare posizione all'interno del paragrafo, abbiamo una matrice W_s su cui facciamo il learning e poi mettiamo tutto questo nella softmax che ci restituisce la probabilità delle varie differenti parole presenti all'interno del paragrafo. Da questo punto abbiamo una serie di matrici W_s che mi indicano l'attention weight per le varie posizioni all'interno dei paragrafi. Il calcolo della α_i ci permette di prevedere dove si trova lo "start token" ovvero dove inizia la risposta alla domanda. Per quanto riguarda l'end token ovvero il punto in cui finisce la risposta facciamo una cosa simile:

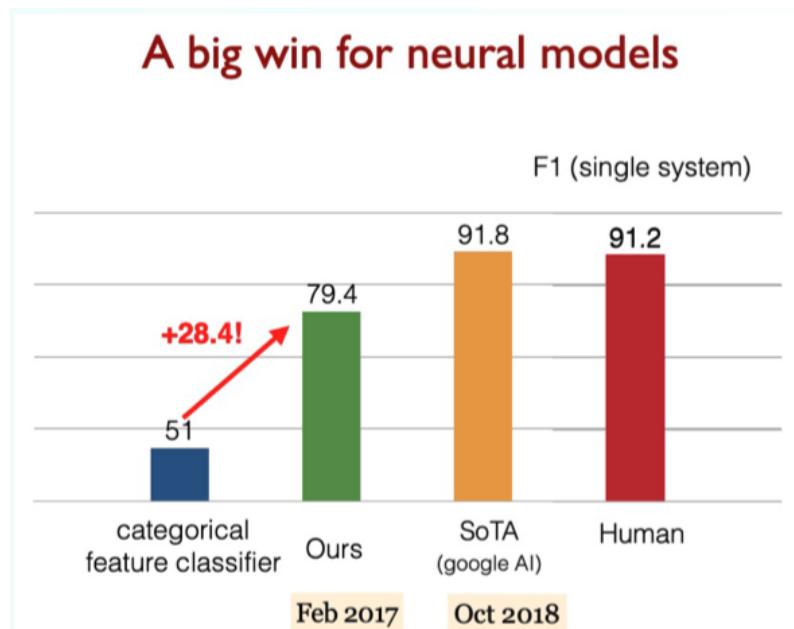
$$\alpha'_i = \text{softmax}_i(q^T W_s p_i)$$

Noi qua consideriamo quindi l'inizio e la fine della domanda, ma quello che c'è al centro ha comunque importanza? Perchè magari potremmo avere nel mezzo la parte più importante della risposta. In realtà quello che facciamo con il nostro modello è considerare solamente gli estremi perchè se nel mezzo c'è qualcosa di importante per la risposta, sarà compito della Bidirectional LSTM spingerlo verso gli estremi. Per questa ragione questa semplice bi-linear attention sarà in grado di ottenere uno score alto all'inizio dello span.

Un'altra cosa interessante è che usiamo la stessa formula per l'inizio e per la fine senza fare una distinzione, non imponiamo niente, è compito della neural network imparare in modo differente le due matrici W_s in modo che siano in grado di riuscire a identificare lo starting point e l'end point.

Cosa dovrebbe produrre questa rete? Deve produrre lo span nel paragrafo che stiamo considerando, per ogni posizione nel paragrafo viene prodotta la probabilità che la risposta comincia da lì e che la risposta finisce in quel punto.

Questo sistema, che comunque non è troppo complesso, ha dimostrato che grazie all'introduzione delle Neural Network e dell'attention è stato possibile fare un grande salto in termini di risultati ottenuti rispetto ai modelli classici.



Quelli nella immagine sopra sono alcuni sistemi ma nel corso degli anni ci sono stati altri sistemi ed altre evoluzioni, siamo passati con il tempo ad approcci più sofisticati andando ad utilizzare in modi differenti l'attention.

23.3 SQuAD 2.0

Abbiamo visto le performances ottenute dai vari modelli con il dataset SQuAD 1.1, dopo questo primo dataset ne è stato introdotto un altro, lo SQuAD 2.0.

SQuAD 2.0 è stata la seconda release di questo dataset, il primo era troppo semplice perchè per ogni domanda era sempre presente la risposta all'interno del dataset, questo non richiedeva di capire davvero la domanda e rendeva tutto molto più semplice, bastava semplicemente trovare le risposta e tutto poi si riduceva ad un task di ranking delle possibili risposte.

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

In SQuAD 2.0 sono state apportate delle modifiche, la metà delle domande hanno una risposta e la metà delle domande non hanno una risposta, questo è più complesso, il sistema che noi abbiamo visto restituisce sempre un ranking per le varie parole che sono presenti all'interno del paragrafo e quindi restituisce sempre una risposta, però se ci vuole indicare che la risposta non c'è il sistema dovrebbe essere più smart.

Con le varie metriche indicate prima otteniamo 1 punto se ad una domanda che non ha risposta rispondiamo direttamente "No answer" mentre invece prendiamo 0 punti se rispondiamo in modo differente.

Quello che hanno fatto i vari sistemi è inserire una threshold che permette di restituire "No answer" nel momento in cui non c'è una risposta alla domanda.

In molti casi però capire se una domanda ha una risposta all'interno del paragrafo oppure no non è semplice, questo è un esempio di domanda che però non ha una risposta all'interno del paragrafo:

Genghis Khan united the Mongol and Turkic tribes of the steppes and became Great Khan in 1206. He and his successors expanded the Mongol empire across Asia. Under the reign of Genghis' third son, Ögedei Khan, the Mongols destroyed the weakened Jin dynasty in 1234, conquering most of northern China. Ögedei offered his nephew Kublai a position in Xingzhou, Hebei. Kublai was unable to read Chinese but had several Han Chinese teachers attached to him since his early years by his mother Sorghaghtani. He sought the counsel of Chinese Buddhist and Confucian advisers. Möngke Khan succeeded Ögedei's son, Güyük, as Great Khan in 1251. He

When did Genghis Khan kill Great Khan?

Gold Answers: <No Answer>

Prediction: 1234 [from Microsoft nInet]

In questo caso non abbiamo la risposta, all'interno di un sistema però quello che succede è che, anche se alla fine otteniamo un risultato molto alto, il sistema non riesce a capire bene il linguaggio, quindi, dato che nella domanda sembra che stiamo chiedendo una data e dato che nel paragrafo ci sono tre date (1206, 1234, 1251) e visto che stiamo cercando

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

quando una persona ha ucciso un'altra possiamo mettere insieme tutte queste informazioni, vediamo che vicino alla data 1234 c'è la parola destroyed che è simile a kill e restituiamo 1234 come risposta. Questa però non è la risposta vera ma questo è un tipico comportamento di un sistema di questo genere perchè non capiscono davvero cosa c'è scritto nel testo.

The Yuan dynasty is considered both a successor to the Mongol Empire and an imperial Chinese dynasty. It was the khanate ruled by the successors of Möngke Khan after the division of the Mongol Empire. In official Chinese histories, the Yuan dynasty bore the Mandate of Heaven, following the Song dynasty and preceding the Ming dynasty. The dynasty was established by Kublai Khan, yet he placed his grandfather Genghis Khan on the imperial records as the official founder of the

What dynasty came before the Yuan?

Gold Answers: (1) Song dynasty (2) Mongol Empire
(3) the Song dynasty

Prediction: Ming dynasty [BERT (single model) (Google AI)]

Con lo SQuAD 2.0 non c'è stato ancora un modello che è riuscito ad ottenere lo score umano ma comunque non sono troppo lontani nelle classifiche.

Da alcuni punti di vista questi sistemi funzionano veramente bene ma allo stesso tempo continuano a fare degli errori di Natural Language Understanding davvero basilari, come possiamo vedere in questo esempio:

The Yuan dynasty is considered both a successor to the Mongol Empire and an imperial Chinese dynasty. It was the khanate ruled by the successors of Möngke Khan after the division of the Mongol Empire. In official Chinese histories, the Yuan dynasty bore the Mandate of Heaven, following the Song dynasty and preceding the Ming dynasty. The dynasty was established by Kublai Khan, yet he placed his grandfather Genghis Khan on the imperial records as the official founder of the

What dynasty came before the Yuan?

Gold Answers: ① Song dynasty ② Mongol Empire
③ the Song dynasty

Prediction: Ming dynasty [BERT (single model) (Google AI)]

Quella dell'esempio è una risposta molto semplice per un umano ma come possiamo vedere in realtà viene fatta una previsione che non è neanche tra le tre gold answers.

23.3.1 Limitazioni di SQuAD

Lo SQuAD Dataset è stato utile nel corso degli anni ma continua ad avere molte limitazioni:

- Possiamo avere solamente delle span-based answers ovvero possiamo rispondere solamente restituendo uno span del paragrafo, non possiamo rispondere ad esempio si o no oppure contando qualcosa oppure a domande più complesse la cui risposta non è contenuta nel paragrafo.
- Il modo in cui SQuAD è stato costruito, per garantire una semplicità della costruzione, è un problema. Infatti per ogni paragrafo che è stato considerato, sono state formulate delle domande sapendo che la risposta si sarebbe trovata in quel paragrafo, nella versione 2.0 poi sono state formulate domande anche sapendo che la risposta non sarebbe stata all'interno del paragrafo. In ogni caso le persone che facevano queste domande rimanevano in qualche modo legate

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

al paragrafo, questo vuol dire che le domande sono sovrapposte sia in termini di parole utilizzate sia in termini di strutture sintattiche con il paragrafo e questo rende il task di QA esageratamente semplice. Nel mondo reale quello che succede è che una persona scrive una domanda in un motore di ricerca e la domanda la scrive in modo completamente diverso da come la risposta sarà presente all'interno dell'articolo che gli verrà proposto.

- Questo task di QA è in realtà più un "trova la parte della frase che matcha meglio la domanda e restituisce", non c'è niente di più complesso.

Nonostante questi svantaggi SQuAD è comunque il dataset più completo per il QA ed è anche stato un punto di partenza per creare sistemi migliori.

23.4 Open Domain Question Answering

Fino ad ora abbiamo parlato di QA andando ad utilizzare delle Neural Network, per i tipi di domande viste fino ad ora, quindi con la risposta presente in un determinato paragrafo, questo era sufficiente per ottenere dei buoni risultati. Il problema è che le Neural Networks possono memorizzare la loro "memoria" solamente all'interno dei loro parametri, questi parametri possono essere tanti e potrebbero dipendere da miliardi di documenti come ad esempio in Bert dove accumuliamo informazioni da tantissimi documenti ma non manteniamo una struttura dei dati all'interno e non si capisce neanche il senso delle frasi che vengono memorizzate. Un primo problema è che una volta che abbiamo fatto il training poi non possiamo modificare questi parametri a mano a mano che arrivano nuovi documenti.

Oltre ai classici sistemi di QA che abbiamo visto fino ad ora possiamo avere anche dei sistemi di Open Domain QA che devono combinare:

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

- L'accesso ad una memoria esterna (quindi deve esserci un modo per accedere ai vari documenti per estrarre informazioni)
- Reading Comprehension

23.4.1 DrQA

DrQA è un sistema prodotto da Stanford in collaborazione con alcuni ricercatori di Facebook che permette di lavorare nel campo dell'Open Domain Question Answering.

L'idea dei ricercatori è stata quella di utilizzare Wikipedia come unica fonte di informazioni permettendo alle persone di fare qualsiasi tipo di domanda cercando sempre di dare la risposta.

Supponiamo di inserire una domanda all'interno del sistema, DrQA è composto da due moduli:



- Document Retriever: è un componente di information retrieval che scansiona tutti gli articoli di Wikipedia cercando di trovare quello che più è relativo alla domanda che abbiamo chiesto.
- Document Reader: questo componente "legge" l'articolo e produce la risposta alla domanda.

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

23.4.2 Document Retriever

Questo non è niente di speciale, è un tipico sistema di Information Retrieval, viene prodotto un inverted index utilizzando il tf.idf e possiamo eseguire il ranking dei documenti per cercare di capire quali sono i documenti più importanti per la nostra query. Per la ricerca all'interno di wikipedia abbiamo anche un sistema che semplifica la ricerca ed è basato su Elastic Search.

Dataset	Wiki Search	Doc. Retriever	
		plain	+bigrams
SQuAD	62.7	76.1	77.8
CuratedTREC	81.0	85.2	86.0
WebQuestions	73.7	75.5	74.4
WikiMovies	61.7	54.4	70.3

Quindi otteniamo un buon sistema di QA composto da semplici componenti, un motore di information retrieval e un semplice ma efficiente sistema di reading comprehension che utilizza le NN.

23.5 Stato dell'arte del QA nel 2020

Oggi nel 2020 lo stato dell'arte della question answer qual è?

23.5.1 La competizione

C'è stata la competizione Tensorflow 2.0 QA, il task consisteva nel selezionare una parte del testo da un articolo di Wikipedia che veniva fornito, quindi la parte di information retrieval era già stata fatta e noi dobbiamo solamente fare l'estrazione delle informazioni. Rispetto a SQuAD

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

abbiamo delle cose nuove, abbiamo delle domande che richiedono risposte brevi e delle domande che invece richiedono delle risposte più lunghe.

Per quanto riguarda la valutazione si è utilizzato lo score F1 ma anche un match tra la risposta predetta dal modello e la risposta corretta del test set.

Questa è stata la leaderbord di questa competizione su Kaggle.

#	Δ pub	Team Name	Notebook	Team Members	Score
1	▲ 3	Guanshuo Xu		 0.71705	0.71705
2	▼ 1	DeepThought	</> submit_full	 0.71684	0.71684
3	▲ 8	in pytorch we trust		 0.70436	0.70436
4	▲ 3	toxu		 0.70025	0.70025
5	▼ 3	bestfitting		 0.69687	0.69687
6	▼ 1	prvi		 0.69474	0.69474

Gli organizzatori avevano anche fornito una architettura che poteva essere utilizzata come partenza per creare il QA system.

L'architettura è definita in modo più formale nel paper [1]. Il sistema utilizza BERT, dato che limita la dimensione dell'input, i documenti in input vengono spartiti e quindi otteniamo delle sequenze di lunghezza massima 512. Facendo questo split per ogni domanda otteniamo tanti split e magari uno di questi conterrà la risposta mentre tutti gli altri non la conterranno. Per il training gli split sono utilizzati per produrre l'input di Bert.

Nel modello abbiamo una quadrupla (c, s, e, t) che viene utilizzata per il training del sistema:

- c è il contesto di 512 parole che stiamo considerando.
- s rappresenta l'indice "start" della risposta
- e rappresenta l'indice "end" della risposta

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

- t rappresenta il tipo della risposta, quindi breve, lunga, si, no oppure niente risposta.

Per quanto riguarda la previsione che viene effettuata si vanno a calcolare 3 probabilità:

- $P_{start}(s|c)$ per ogni parola nel testo si considera la probabilità che quella parola sia quella con cui comincia la risposta dato il contesto in cui si trova quella parola.
- $P_{end}(s|c)$ per ogni parola nel testo si considera la probabilità che quella parola sia quella con cui termina la risposta.
- $p_{type}(s|c)$ questa indica la probabilità del tipo di risposta che stiamo fornendo.

Queste tre probabilità sono calcolate dall'output Layer di Bert, vengono tutte stimate utilizzando la softmax.

Una volta che abbiamo fatto il training del sistema dobbiamo prendere una decisione riguardante una nuova domanda, dato un nuovo documento e dato il nuovo contesto c dove compare la parola. Per capire quale risposta è migliore per una certa domanda calcoliamo lo score per ognuno delle coppie di posizioni all'interno del documento, per ogni possibile spans di differenti lunghezze.

$$\begin{aligned} g(c, s, e) = & f_{\text{start}}(s, c; \theta) + f_{\text{end}}(s, c; \theta) \\ & - f_{\text{start}}(s = [\text{CLS}], c; \theta) - f_{\text{end}}(e = [\text{CLS}], c; \theta) \end{aligned}$$

Viene calcolato uno score g che è una combinazione di f_{start} e f_{end} . Questo è lo score che otteniamo e quello che facciamo è selezionare quello più alto. Questa neural network che veniva fornita come partenza è un baseline system molto semplice da capire e da implementare.

23.5.2 Il vincitore

Cosa ha fatto il vincitore? Quanto era sofisticato il sistema?

Le regole della competizione richiedono che venga fornito il codice per la soluzione quindi possiamo vedere cosa ha fatto il vincitore.

- L'architettura riproduce l'architettura della baseline.
- Rispetto alla baseline ci sono modi differenti per fare il negative sampling.
- Dobbiamo avere dei negative examples ma non troppi e non troppo pochi

Come ha vinto? La maggior parte dei suoi risultati erano 0.66 e quindi non era troppo alto per vincere, quello che ha fatto è stato modificare il threshold nelle ultime submission che ha fatto, il threshold indica quando una risposta è buona o no. Solamente giocando con questo threshold è riusito a vincere la competizione.

Questo è sorprendente ma non porta nessun miglioramento significativo per la scienza, è stato solamente fortunato nel trovare un threshold adatto.

Chapter 24

Machine Reading

In questo caso la knowledge base è lo stesso passaggio che stiamo considerando, non abbiamo dati esterni.

- Span selection: se c'è una risposta sarà nel paragrafo che stiamo considerando.
- Cloze: abbiamo del testo con alcune parole rimosse e dobbiamo riempire gli spazi vuoti andando ad inserire la parola corretta.

Per lavorare con questo tipo di dati sono stati prodotti vari dataset:

- MCTest, ci sono 500 paragrafi e abbiamo 2000 domande che riguardano qualcosa che sta scritto in quel passaggio.
- RACE: qua abbiamo tanti più passaggi con molte più domande e sono delle domande che sono utilizzate normalmente per dei test di comprensione dell'inglese. Anche qua rispondiamo alla domanda con una risposta tra quelle proposte.

Per quanto riguarda la selezione dello Span abbiamo:

- SQuAD: ci sono 500 paragrafi e 100.000 domande riguardanti testi presi da Wikipedia.

- TriviaQA:

Cloze Questions:

Le Cloze Questions sono prese da un dataset della CNN e da un dataset del Daily Mail, sono stati creati da riassunti di articoli e l'obiettivo è comprendere l'entità che viene nascosta all'interno degli articoli. Ad esempio:

Original Version	Anonymised Version
<p>Context</p> <p>The BBC producer allegedly struck by Jeremy Clarkson will not press charges against the "Top Gear" host, his lawyer said Friday. Clarkson, who hosted one of the most-watched television shows in the world, was dropped by the BBC Wednesday after an internal investigation by the British broadcaster found he had subjected producer Oisin Tymon "to an unprovoked physical and verbal attack." ...</p>	<p>the <i>ent381</i> producer allegedly struck by <i>ent212</i> will not press charges against the "<i>ent153</i>" host , his lawyer said friday . <i>ent212</i> , who hosted one of the most - watched television shows in the world , was dropped by the <i>ent381</i> wednesday after an internal investigation by the <i>ent180</i> broadcaster found he had subjected producer <i>ent193</i> " to an unprovoked physical and verbal attack . " ...</p>

Quando sostituiamo con delle chiavi anonime alcune delle entità che sono presenti all'interno del testo. La cosa che possiamo notare è che se una entità compare più di una volta utilizziamo la stessa chiave per rappresentarla.

Il sistema utilizzando questo sistema diventa più generale. Una cosa interessante è che prendere un paragrafo e rimuovere alcune parole è simile ad un sistema che abbiamo già visto in precedenza, è Bert.

Possiamo utilizzare un sistema simile a quello che abbiamo in Bert per risolvere il problema di Cloze.

Abbiamo la frase originale in cui mascheriamo alcune parole, Bert dovrebbe essere capace di fare la previsione semplicemente applicando quello che abbiamo imparato.

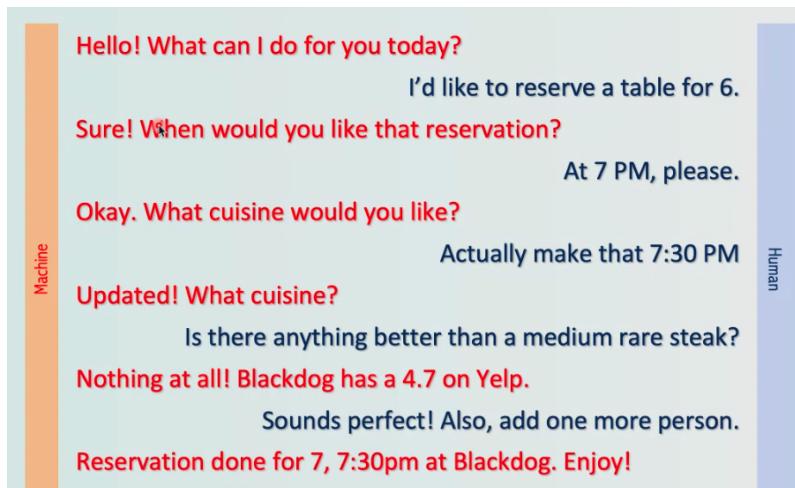
La teoria dietro Bert è utilizzare una grande quantità di testo da cui possiamo prendere la conoscenza che ci serve.

Machine Reading: Per alcune domande non abbiamo le risposte presenti come span all'interno del documento che abbiamo letto prima. Possiamo pensare a due tipi differenti di task che dobbiamo eseguire per gestire questo tipo di QA:

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

- Inference QA:
- Reading Comprehension:

Perchè è importante fare delle multi step reasoning? Ad esempio per i Dialogue Systems, abbiamo un esempio di Dialogue System, in questo caso abbiamo un dialogo per quanto riguarda la prenotazione di un ristorante. Il sistema dovrebbe essere in grado di capire se sto rispondendo ad una domanda oppure se sto cambiando la risposta ad una precedente domanda. Ad esempio qua noi rispondiamo 7PM poi 7:30PM e il sistema lo capisce e aggiorna e ci chiede di nuovo la stessa domanda a cui non avevamo risposto. Se la macchina è abbastanza intelligente può rispondere anche alle altre domande che facciamo dopo e in particolare qua ci risponde "Nothing at all!!" quando chiediamo cosa c'è di meglio. Alla fine poi gli chiediamo anche di aggiungere una nuova persona nella prenotazione del ristorante e la macchina risponde con un riassunto.



Tra le frasi che scriviamo al dialogue system ci sono delle relazioni quando cambiamo idea.

Questo è un task che non è per niente facile.

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

24.1 bAbl Dataset

Per cercare di migliorare questo tipo di task vogliamo utilizzare un dataset che è stato introdotto, si chiama bAbl dataset ed è formato da una collezione di 20 task, per ogni task (tipo di domanda) ci focalizziamo su una certa abilità.

Vediamo degli esempi, possiamo raggruppare in task dai più semplici ai più complessi:

(T1) Single supporting fact	(T2) Two supporting facts
<ul style="list-style-type: none"> Questions where a single supporting fact, previously given, provides the answer. Simplest case of this: asking for the location of a person. <p>John is in the playground. Bob is in the office. Where is John? A: playground</p>	<ul style="list-style-type: none"> Harder task: questions where two supporting statements have to be chained to answer the question. <p>John is in the playground. Bob is in the office. John picked up the football. Bob went to the kitchen. Where is the football? A: playground Where was Bob before the kitchen? A: office</p>

Ad esempio qua chiediamo la posizione di una persona dopo che abbiamo indicato dove si trovano varie persone. Si tratta solamente di estrarre la posizione corretta della persona a cui siamo interessati. Two supporting facts: in questo caso la situazione è più complessa, per rispondere a questa c'è da ricordarsi i due fatti riguardanti John per concludere che il calcio è nel playground. A mano a mano che andiamo avanti i task sono via via più complessi, anche noi se rispondiamo dobbiamo ragionarci un po' prima di dare una risposta. Un altro tipo di task è la relazione tra due oggetti, qua abbiamo 3 oggetti nello spazio e dobbiamo capire quali sono le relazioni spaziali tra loro.

(T3) Three supporting facts	(T4) Two argument relations
<ul style="list-style-type: none"> Similarly, one^scan make a task with three supporting facts The first three statements are all required to answer this <p>John picked up the apple. John went to the office. John went to the kitchen. John dropped the apple. Where was the apple before the kitchen? A:office</p>	<ul style="list-style-type: none"> To answer questions the ability to differentiate and recognize subjects and objects is crucial. We consider the extreme case: sentences feature re-ordered words: <p>The office is north of the bedroom. The bedroom is north of the bathroom. What is north of the bedroom? A:office What is the bedroom north of? A:bathroom</p>

Qualcosa di ancora più difficile è il conteggio, dobbiamo essere in grado di trovare le possibili risposte e anche di contarle.

(T3) Three supporting facts	(T4) Two argument relations
<ul style="list-style-type: none"> Similarly, one^scan make a task with three supporting facts The first three statements are all required to answer this <p>John picked up the apple. John went to the office. John went to the kitchen. John dropped the apple. Where was the apple before the kitchen? A:office</p>	<ul style="list-style-type: none"> To answer questions the ability to differentiate and recognize subjects and objects is crucial. We consider the extreme case: sentences feature re-ordered words: <p>The office is north of the bedroom. The bedroom is north of the bathroom. What is north of the bedroom? A:office What is the bedroom north of? A:bathroom</p>

Un altro task complesso è quello del ragionamento riguardante la dimensione e anche la capacità di rispondere sì o no. La transitività non è una relazione semplice da gestire nella logica.

(T6) Yes/No questions	(T7) Counting
<ul style="list-style-type: none"> This task tests, in the simplest case possible (with a single supporting fact) the ability of a model to answer true/false type questions: 	<ul style="list-style-type: none"> This task tests the ability of the QA system to perform simple counting operations, by asking about the number of objects with a certain property

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

Anche il path finding è complicato perché dobbiamo considerare varie possibilità e capire quale delle varie alternative è quella più probabile. Un altro task complesso è quello che consiste nel comprendere le motivazioni di una qualche azione.

(T19) Path finding	(T20) Agent's motivation
<ul style="list-style-type: none"> the goal is to find the path between locations This task is difficult because it effectively involves search <p>The kitchen is north of the hallway. The den is east of the hallway. How do you go from den to kitchen? A: west,north</p>	<p>John is hungry. John goes to the kitchen. John grapples the apple there. Daniel is hungry. Why did John go to the kitchen? A: hungry Where does Daniel go? A: kitchen</p>

In generale i vari task che abbiamo da risolvere sono i seguenti:

Type	Task number	Difficulty
Single Supporting Fact	1	Easy
Two or Three Supporting Facts	2-3	Hard
Two or Three Argument Relations	4-5	Medium
Yes/No Questions	6	Easy
Counting and Lists/Sets	7-8	Medium
Simple Negation and Indefinite Knowledge	9-10	Hard
Basic Coreference, Conjunctions and Compound Coreference	11-12-13	Medium
Time Reasoning	14	Medium
Basic Deduction and Induction	15-16	Medium
Positional and Size Reasoning	17-18	Hard
Path Finding	10	Hard
Agent's Motivations	20	Easy

Multi Step Reasoning.

24.2 Memory Networks

Il problema che vogliamo affrontare oggi è come possiamo rispondere a queste domande, per rispondere abbiamo la necessità di ricordare quello

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

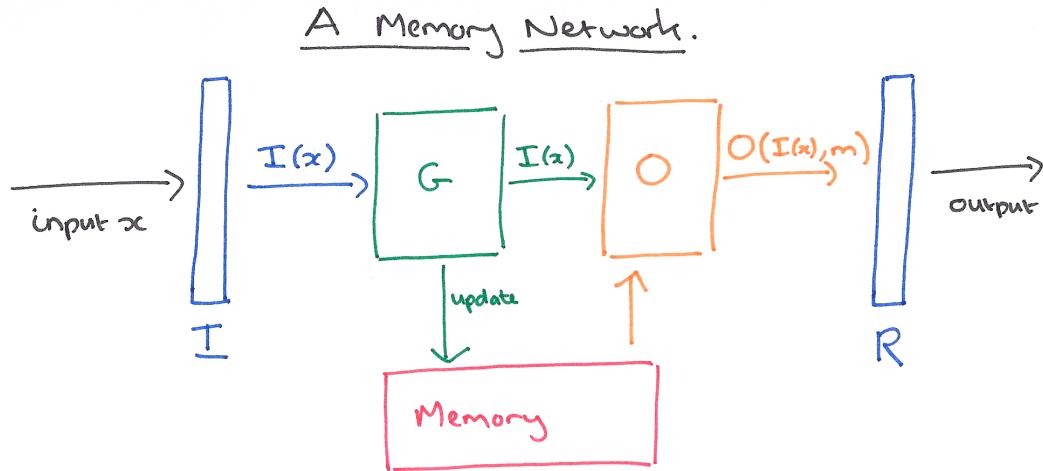
che è stato detto in precedenza e poi dobbiamo essere capaci di selezionare quello che può essere utile per rispondere alla domanda. I modelli di Machine Learning quindi hanno bisogno di memoria, le RNN hanno una memoria breve ma non ci sono meccanismi per compartimentalizzare la memoria, viene memorizzata in modo.

Un suggerimento è aggiungere memoria alla rete (MemNN),

La prima proposta è quella dei Memory Network di Facebook Ai, consiste in 4 parti, è il meccanismo per rappresentare gli input e manipolare gli input, poi c'è una generalizzazione, poi c'è una Output Feature Map e la Response che è l'ultimo stage che prende la risposta prodotta dallo stage precedente e la trasforma in una risposta per questa domanda.

Vediamo la MemNN:

- C'è una memoria m indicizzata come un array di oggetti, ad esempio un vettore di stringhe;
- C'è una Input Feature Map I che converte l'input in una rappresentazione interna delle feature di quell'input;
- Poi c'è un componente G di generalizzazione che aggiorna la vecchia memoria considerando il nuovo input, lo chiamano generalizzazione perchè c'è l'opportunità per la rete di comprimere e generalizzare la sua memoria in questo stage per utilizzi futuri;
- C'è una feature map di output O che produce un nuovo output dato l'input corrente e la memoria corrente;
- C'è un componente R di risposta che converte l'output nel formato desiderato, ad esempio una risposta testuale.



Se ad esempio vogliamo avere una memoria molto grande e memorizzare tutta Wikipedia, la funzione H potrebbe essere prodotta per fare in modo di memorizzare i dati per entità o per topics. Se la memoria diventa troppo grande poi c'è una funzione forgetting che potrà essere implementata. Quando i componenti della rete I, G, O e R sono reti neurali allora il risultato è che la memory network diventa una Memory Neural Network.

Funzionamento basilare dei Memory Networks

La versione più semplice dei Memory Neural Networks funziona in questo modo:

- Al layer di input I arriva una frase alla volta, questa frase mi indica informazioni da memorizzare, ad esempio possono essere frasi prese da Wikipedia, poi possono essere domande relative a quello che abbiamo memorizzato;
- G prende questa frase e la memorizza nel prossimo slot di memoria disponibile, assumiamo di avere più slot che frasi da inserire;

- Il layer di output O serve per trovare le K memories che sono utili per il task che vogliamo risolvere. Per capire quali sono le K memories più utili consideriamo la frase in input e poi calcoliamo il match tra le varie frasi che abbiamo in memoria e prendiamo le migliori K ;
- L'output del layer O è una lista di frasi che contengono la frase originale e le K frasi di supporto che vengono trovate. Una versione più sofisticata può usare una RNN per eseguire la generazione delle frasi.

Quando l'input che arriva ad I è solamente uno stream di parole e non è già diviso in testi e domande a cui rispondere allora va anche aggiunto un layer che serve per segmentare.

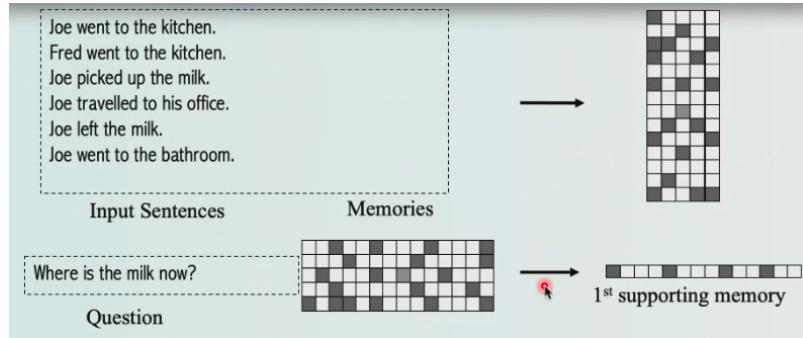
Altre informazioni sui Memory Networks

Una rappresentazione semplice per la domanda è la rappresentazione Bag of Word che consiste solamente nel contare il numero di occorrenze di ognuna di queste parole. Nella maggior parte dei casi avremo degli 0 tranne nei punti in cui abbiamo una parola.

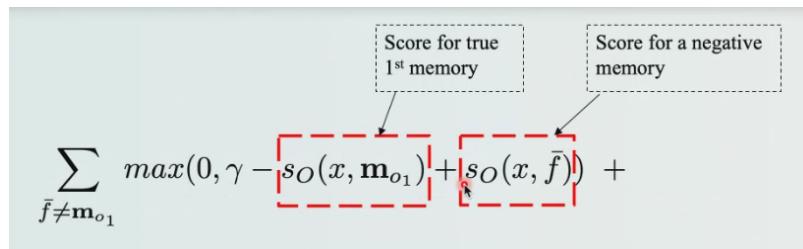
Dopo il 5 input la memory viene aggiornata. Dopo dobbiamo portare qualcosa fuori dalla memoria, utilizziamo un mapping che calcola uno score di tutta la memoria rispetto all'input. Poi il secondo step è simile, prendiamo l'output dallo stage precedente e svolgiamo il secondo step in modo simile, abbiamo la combinazione dell'input e dell'output dallo stage precedente e produciamo un secondo e finale score. Lo score finale misura in qualche modo quale frase del passato è più rilevante per la domanda corrente. Nello step 4 consideriamo anche un altro scoring function.

La scoring function per x e y viene ottenuta moltiplicando per U che è una matrice.

Qua cerchiamo di estrarre dalle varie frasi la risposta alla nostra domanda.



Questa è la struttura generale del nostro sistema per cui dobbiamo fare il training:



Poi facciamo la stessa cosa per il secondo layer che funziona esattamente nello stesso modo e poi la stessa cosa la facciamo anche per lo score della risposta.

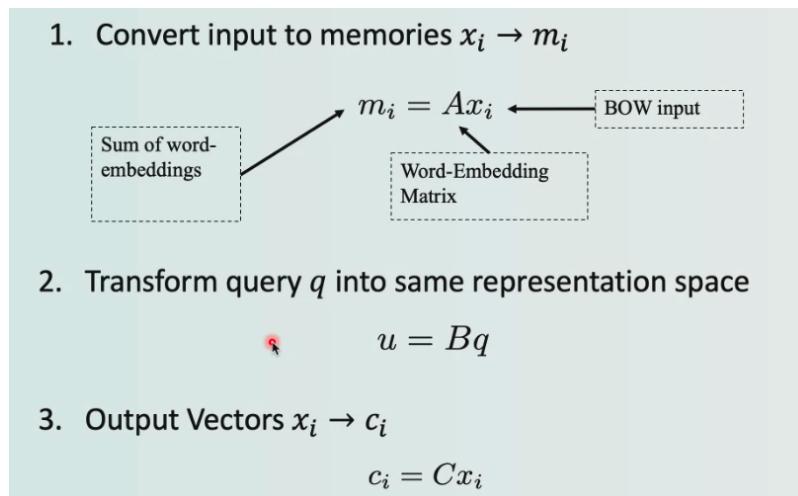
$$\begin{aligned} & \sum_{\bar{f} \neq \mathbf{m}_{o_1}} \max(0, \gamma - s_O(x, \mathbf{m}_{o_1}) + s_O(x, \bar{f})) + \\ & \sum_{\bar{f}' \neq \mathbf{m}_{o_2}} \max(0, \gamma - s_O([x, \mathbf{m}_{o_1}], \mathbf{m}_{o_2}) + s_O([x, \mathbf{m}_{o_1}], \bar{f}')) + \\ & \sum_{\bar{r} \neq r} \max(0, \gamma - s_R([x, \mathbf{m}_{o_1}, \mathbf{m}_{o_2}], r) + s_R([x, \mathbf{m}_{o_1}, \mathbf{m}_{o_2}], \bar{r})) \end{aligned}$$

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

Come funziona tutto questo?

- Abbiamo una rappresentazione con Bow

Un altro approccio che può funzionare meglio è quello dell'End to end memory networks. È una soluzione simile alla precedente, non usiamo la rappresentazione Bow ma usiamo i word embedding.



Poi usiamo la softmax per eseguire lo score della memoria rispetto alla query. È simile all'attention.

Risultati sul babl dataset:

Story (1: 1 supporting fact)	Support	Hop 1	Hop 2	Hop 3
Daniel went to the bathroom.		0.00	0.00	0.03
Mary travelled to the hallway.		0.00	0.00	0.00
John went to the bedroom.		0.37	0.02	0.00
John travelled to the bathroom.	yes	0.60	0.98	0.96
Mary went to the office.		0.01	0.00	0.00
Where is John? Answer: bathroom Prediction: bathroom				

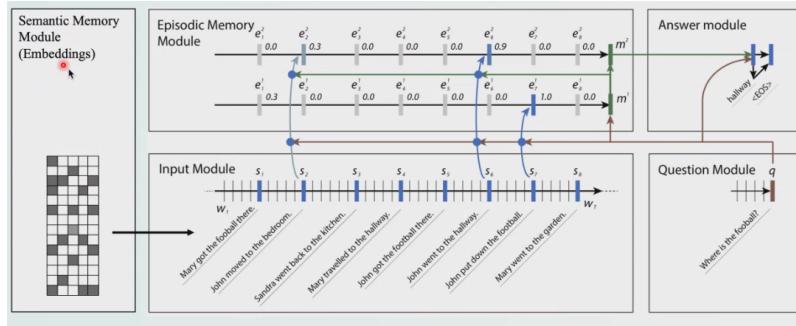
Abbiamo l'esempio con 5 fatti e il sistema è in grado di evidenziare le frasi più rilevanti dando uno score nei vari Hop. In pratica sono i tre

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

layer di analisi e a mano a mano che andiamo in profondità siamo sempre più confidenti. Questa richiede più induzione.

Story (16: basic induction)	Support	Hop 1	Hop 2	Hop 3
Brian is a frog.	yes	0.00	0.98	0.00
Lily is gray.		0.07	0.00	0.00
Brian is yellow.	yes	0.07	0.00	1.00
Julius is green.		0.06	0.00	0.00
Greg is a frog.	yes	0.76	0.02	0.00
What color is Greg? Answer: yellow		Prediction: yellow		

Qua abbiamo dei fatti che sono già conosciuti e abbiamo la nostra matrice di embedding, poi abbiamo una RNN che viene utilizzata per produrre una rappresentazione finale per la frase. La RNN prende una sequenza e produce una rappresentazione combinando le informazioni.



Questo è quello che accade qua, abbiamo varie rappresentazioni e poi abbiamo l'episodic memory module che evidenzia quelle che sono più rilevanti per una certa domanda.

Questa è l'accuracy che otteniamo con le Dynamic Memory Network:

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

Classification accuracy	Positional Reasoning (1K)	Positional Reasoning (10K)
Dynamic Memory Network	59.6	-
Neural Reasoner	66.4	97.9

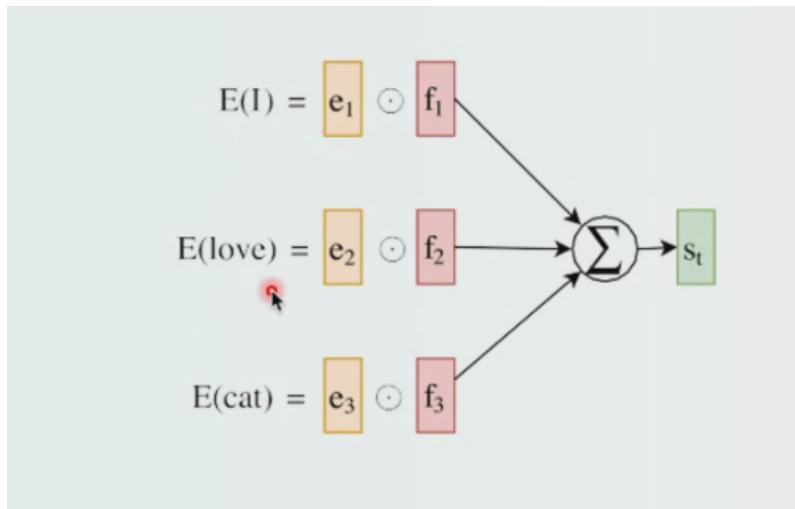
Classification accuracy	Path Finding (1K)	Path Finding (10K)
Dynamic Memory Network	34.5	-
Neural Reasoner	17.3	87.0

Riassunto dei 20 task con confronto con il MemNN

Task	MemNN	DMN	Task	MemNN	DMN
1: Single Supporting Fact	100	100	11: Basic Coreference	100	99.9
2: Two Supporting Facts	100	98.2	12: Conjunction	100	100
3: Three Supporting Facts	100	95.2	13: Compound Coreference	100	99.8
4: Two Argument Relations	100	100	14: Time Reasoning	99	100
5: Three Argument Relations	98	99.3	15: Basic Deduction	100	100
6: Yes/No Questions	100	100	16: Basic Induction	100	99.4
7: Counting	85	96.9	17: Positional Reasoning	65	59.6
8: Lists/Sets	91	96.5	18: Size Reasoning	95	95.3
9: Simple Negation	100	100	19: Path Finding	36	34.5
10: Indefinite Knowledge	98	97.5	20: Agent's Motivations	100	100
			Mean Accuracy (%)	93.3	93.36

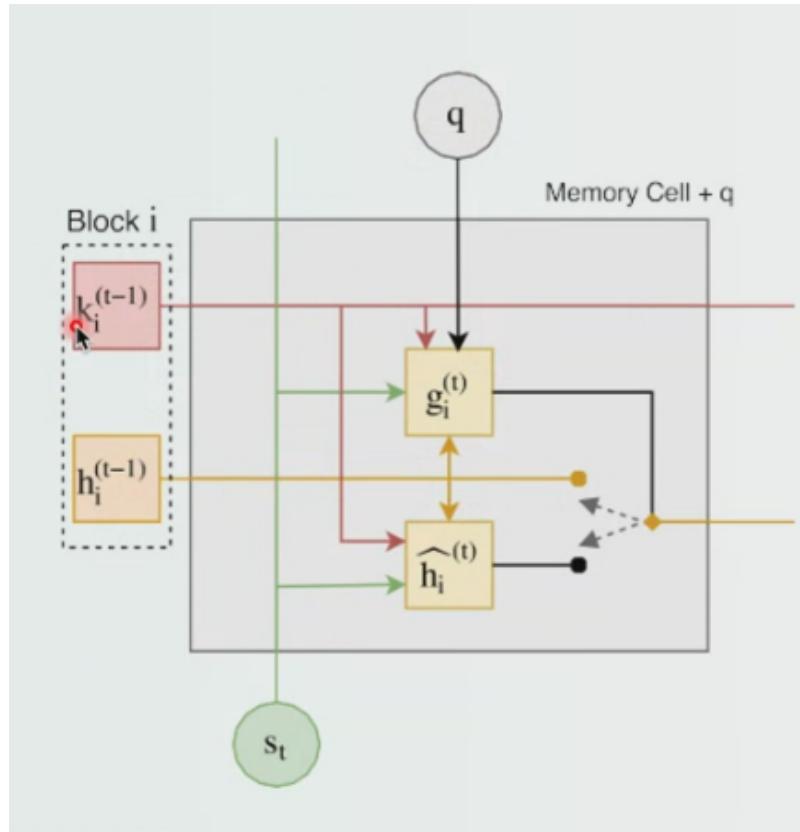
abbiamo che il DMN è migliore in molti task, specialmente in quelli più complessi, per gli altri sono quasi sempre migliori. In media l'accuracy non è così tanto maggiore.

Similarities	Differences
<ul style="list-style-type: none"> • MemNets and DMNs have input, scoring, attention and response mechanisms 	<ul style="list-style-type: none"> • For input representations MemNets use bag of word, nonlinear or linear embeddings that explicitly encode position • MemNets iteratively run functions for attention and response • DMNs shows that neural sequence models can be used for input representation, attention and response mechanisms \Leftrightarrow naturally captures position and temporality • Enables broader range of applications

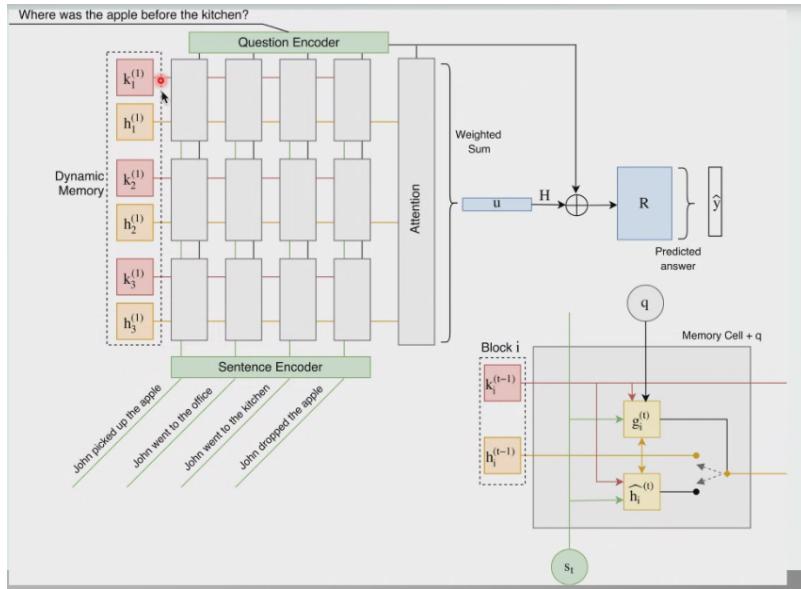


In questo caso abbiamo gli embedding per ognuno di queste parole, poi queste vengono aggiunte tutte insieme per produrre la rappresentazione finale della nostra frase facendo una somma pesata.

Alla fine abbiamo una dynamic memory che è composta da un set di blocchi, ogni entità corrisponde ad una chiave che identifica l'entity e di un hidden state che memorizza informazioni al riguardo.



Una volta che abbiamo deciso in che modo possiamo utilizzare l'embedding matrix per generare l'output. Il final score è un altro linear layer che prende l'output e abbiamo la matrice finale.

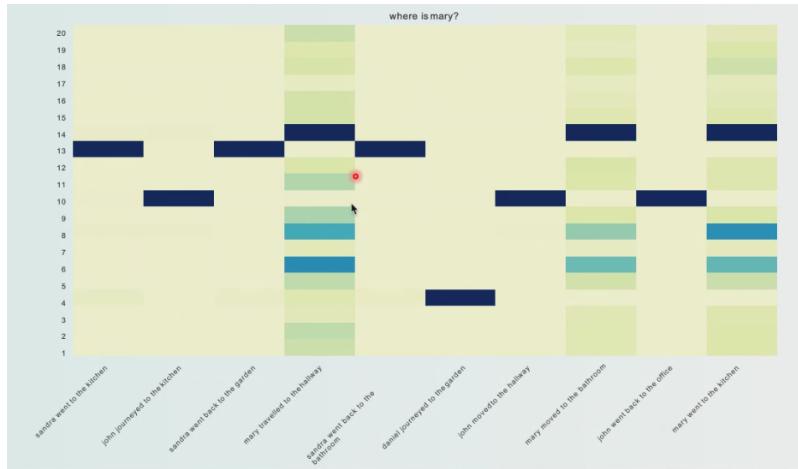


Come funziona tutto questo? Molto bene, provato su CNN dataset e su bAbI dataset.

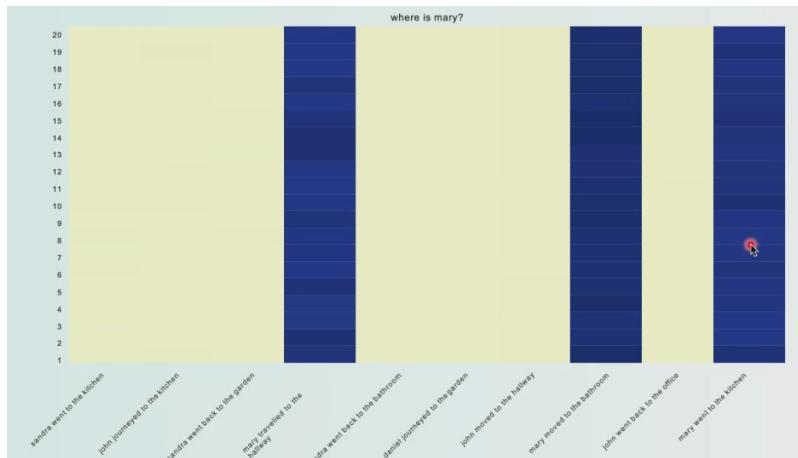
<i>Failed Tasks (>5%):</i>	20	20	11	15	8
<i>Mean Error:</i>	65.9	50.8	13.9	29.6	18.6
	n-gram	LSTM	MemN2N	REN	QDREN

Abbiamo un error rate del 30%. Le memory end to end sono lo stato dell'arte ora. Il numero di task che falliamo è solamente 8, da 11 che erano prima. L'errore medio è leggermente più alto ma in molti casi il sistema funziona molto meglio rispetto al MemN2N.

Lo stesso sistema è applicato ad un altro test che è quello del dataset della CNN. Vediamo come funziona nella selezione delle parti corrette.



Questo è come estrae le risposte, qui in particolare abbiamo la funzione g che è alta per esempio in alcune domande in cui Mary è presente perché la domanda riguarda Mary.



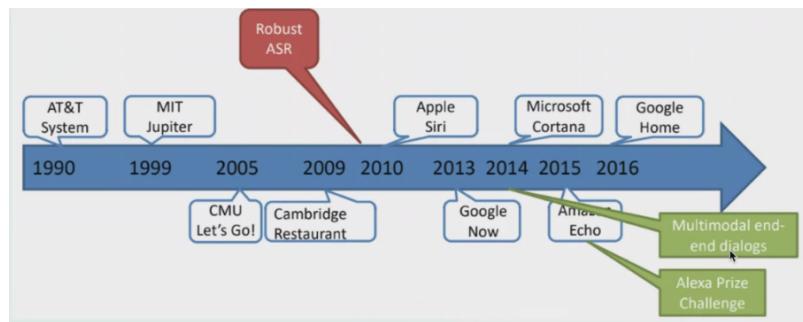
Abbiamo visto come utilizzare le Neural Network con la memoria in modo da memorizzare delle sequenze di fatti che sono stati menzionati, ci sono ancora ricerche in questo campo e questo approccio di entity network potrebbe essere molto promettente. Non ci interessa solamente capire dove si trova Mary ma anche la posizione che è la proprietà che la associa nella struttura che associa.

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

Chapter 25

chatbot

I chatbot sono difficili perchè comprendono tutte le tecnologie che abbiamo visto fino ad ora, molti vorrebbero avere dei chatbot funzionanti.



La storia dei chatbot parte abbastanza da lontano, già negli anni 90 furono effettuati i primi studi, poi nel corso degli anni le ricerche sono continue ma nei primi anni era tutto molto complesso perchè non era ancora possibile eseguire uno speech recognition in automatico. Poi nel 2005 circa sono cominciati ad arrivare i primi sistemi per lo speech recognition. Dal 2010 poi sono stati sviluppati vari sistemi di Dialog System come Siri, che è stato il primo, poi Google Now, Cortana e poi questi sono stati messi in dispositivi dedicati esclusivamente ad essere assistenti personali con cui si può parlare. Ad esempio il Google

Home o Amazon Alexa o l'Homepod di Apple. Per eseguire il training di questi sistemi Amazon ad esempio ha avviato una challenge per cercare di migliorare la conversazione con Alexa.

Ai giorni d'oggi abbiamo assistenti personali che sono in produzione e vengono venduti e milioni di persone li utilizzano.

25.1 Storia

I primissimi approcci in questo campo sono stati prodotti all'MIT con ELIZA che era un sistema che cercava di utilizzare dei template predefiniti per cercare di rispondere alle domande che venivano inviate da un utente. Era veramente interessante all'epoca, non comprendeva bene quello che l'utente scriveva, cercava il più possibile di dare una risposta sensata. Ora come ora ci stanno tantissimi framework che permettono di creare dei chatbot.

Come funziona questo approccio con template e regole?

Venivano utilizzate delle regole definite per generare delle risposte. L'idea era di fare un match tra un pattern fissato e la possibile risposta che il sistema era in grado di dare. Ad esempio al pattern "What is your name?" si poteva rispondere con il template "My name is Alice", alcuni pattern erano più complessi e potevano lasciare un po' più di libertà, ad esempio il pattern "I like *", con l'asterisco permettiamo di eseguire il match con qualsiasi parola.

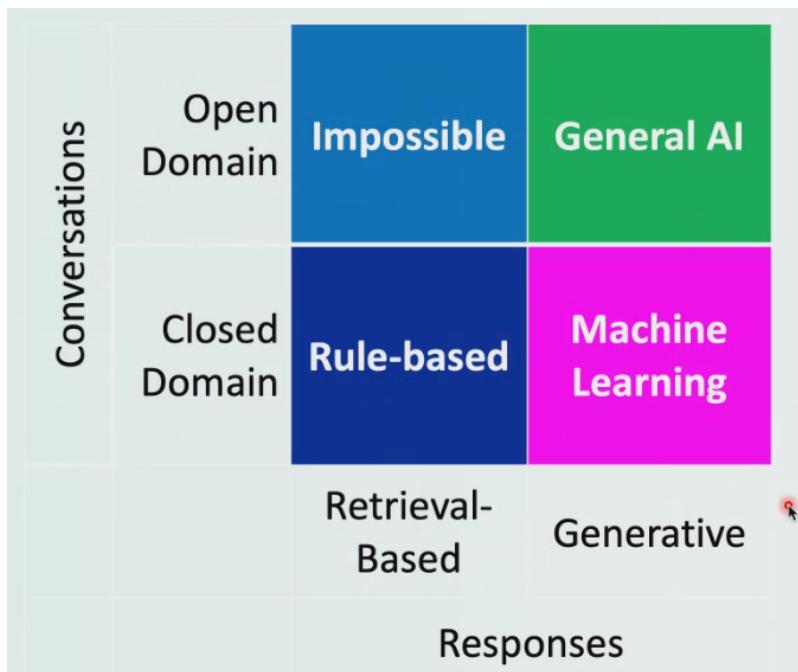
All'epoca la cosa era interessante perché le persone pensavano di stare a parlare con un agente intelligente. Ancora oggi questa tecnologia viene utilizzata.

Vediamo ora la classificazione dei vari chatbot e delle tecniche per crearli, la prima distinzione è tra Open e Closed domain chatbots. Gli Open Domain Chatbots sono più complessi, non abbiamo una intenzione durante la conversazione, l'utente può provare a continuare la conversazione aperta per tempo e il chatbot dovrebbe cercare di mantenere

aperta la conversazione. È tipo una conversazione su Twitter o Reddit.

Nel closed domain invece abbiamo un numero fissato di input e di output che sono fissati in partenza. L'utente in questo caso ha uno scopo che deve essere raggiunto, il successo dell'interazione è se riusciamo a raggiungere lo scopo di questa interazione. Un esempio di questo sono i Technical Customer Support o gli assistenti di shopping.

C'è un trade off tra Open e Closed Domain che è presente nelle slide, la difficoltà e la complessità dei vari chatbot dipende dal tipo di dominio, verticalmente dividiamo in closed domain e in Open domain e orizzontalmente consideriamo due tipi di chatbot, quelli che sono retrieval based e quelli che sono generative. I Retrieval prendono le risposte da un set di risposte possibili, funzionano bene nel caso del closed domain ma è impossibile usarli nel caso dell'open Domain. Nel caso generative non abbiamo delle risposte già pronte, le risposte devono essere generate in base alla conversazione che stiamo avendo, l'approccio in questo caso consiste nell'utilizzo di tecniche di machine learning, funzionano bene nel caso del Closed Domain, se andiamo in un Open Domain il chatbot necessita di essere più intelligenza per essere utilizzato, siamo ancora lontani dall'avere queste capacità oggi.



Abbiamo due paradigmi:

- Retrieval Based: abbiamo una collezione di risposte predefinite e alcune euristiche per poter scegliere la risposta appropriata basandoci sugli input e sul contesto. Il retrieval based può essere anche il più semplice, potrei fare solamente pattern matching oppure potrebbe essere più complesso se vado ad utilizzare classificatori che sfruttano il Machine Learning. Con questo approccio non generiamo dei dati, usiamo solamente delle risposte da set fissati di dati. Con questo approccio non faremo degli errori grammaticali perchè prendiamo solamente delle risposte predefinite ma non saremo in grado di gestire situazioni o argomenti che non abbiamo mai visto in precedenza e che non sono presenti nel database. Il retrieval Based non fa neanche un buon lavoro per mantenere il contesto, tipicamente non possiamo memorizzare il contesto della conversazione avvenuta fino ad ora.

- Generative: le risposte in questo caso sono generate da 0, non utilizziamo delle risposte predefinite, i modelli generativi sono basati su tecniche di transducer tipo il machine translation, prendiamo in questo caso un input che è la domanda che viene fatta dall'utente e produciamo un output che è la risposta alla domanda. Abbiamo ancora il problema di come sfruttare il contesto in cui avviene il dialogo, abbiamo necessità di capacità per capire il contesto (visto nella lezione precedente). Con questo sistema possiamo memorizzare un contesto e possiamo considerare quello che è successo nel passato durante la conversazione. Chiaramente sono più complesse per quel che riguarda il training e possono produrre anche degli errori grammaticali, specialmente nelle frasi più lunghe. Per eseguire il training di questi sistemi poi possiamo anche avere necessità di grandi quantità di dati.

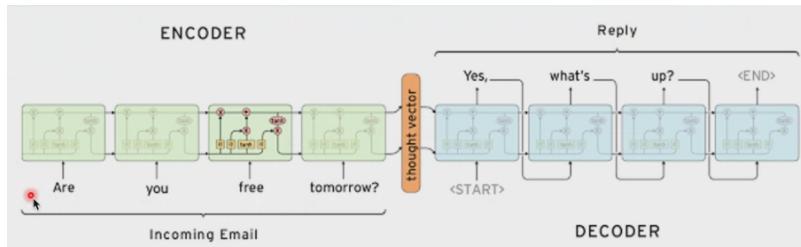
Un'altra cosa che possiamo distinguere nei chatbot è la lunghezza della conversazione che possiamo avere. Ci sono infatti chatbot che possono avere una conversazione breve e quelli che invece possono avere una conversazione più lunga. Chiaramente i chatbot che fanno conversazioni brevi sono quelli più semplici, l'obiettivo in questo caso è creare una singola risposta per il singolo input che riceviamo, ad esempio quando rispondiamo ad una domanda specifica che arriva da un utente con una risposta che deve essere il più appropriata possibile. I chatbot che permettono una conversazione più lunga sono più complessi, qua c'è l'alexa challenge che chiede di creare un dialogue system che deve essere in grado di mantenere l'utente in una conversazione per 20 minuti.

25.2 Generation Based Models

I generation based models utilizzano tecniche del Natural Language Understanding, quello di cui abbiamo parlato fino ad ora è il transducer model, è un sequence to sequence model che prende in input una frase

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

e produce un output che è un'altra frase. Ad esempio possiamo ricevere in input la frase "Are you free tomorrow?", questa viene analizzata dal transducer tramite un LSTM o una RNN. Poi l'encoder produce un output che viene inviato al decoder che a sua volta produce in output la risposta che deve essere data dal sistema.

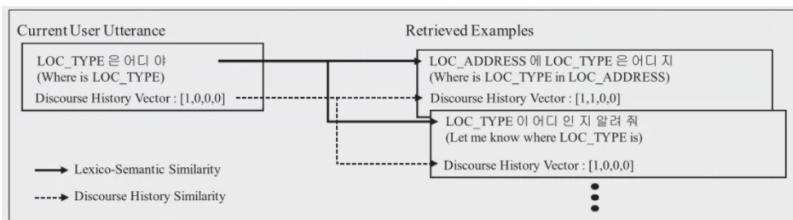


Questo è esattamente quello che facciamo con la Neural Machine Translation e anche qua possiamo introdurre l'attention per avere dei risultati che sono migliori. L'attention mi aggiunge un livello di complessità.

Come succede anche con altri task di traduzione, la produzione della risposta può essere effettuata tramite un sistema di encoder-decoders.

25.3 Retrieval Based Models

Possono essere sviluppati utilizzando dei Template. Consideriamo un esempio di Retrieval Based Chat:



L'idea è che data una frase da un utente possiamo cercare quella più simile nel database e poi la restituiamo. Se la domanda è del tipo "Where

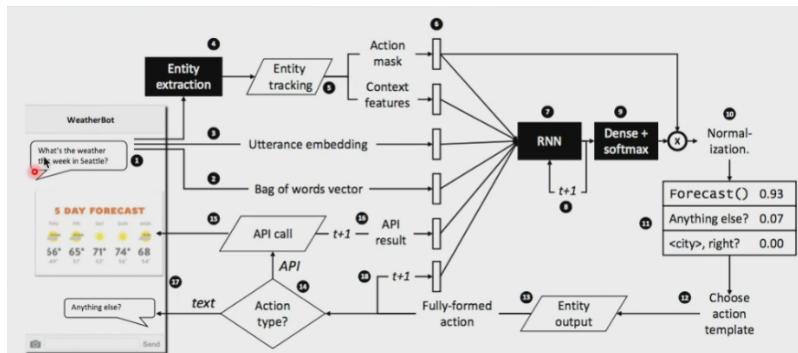
Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

is *LOC_TYPE?*", il pre-processing ci dice che dobbiamo trovare qualcosa del tipo *LOC_ADDRESS*, quindi troviamo il match e poi restituiamo la risposta. Ad esempio potremmo chiedere "Where is Rome?" e con il pattern matching troveremmo la risposta più simile che è "Rome is in Italy".

Una idea per utilizzare le Neural Network per questo task è stata presentata nel 2014, invece di svolgere un match esatto come si faceva in precedenza, con la neural response retrieval è possibile utilizzare un approccio soft che utilizza una NN per trovare similarità tra i vari pattern e mi dice quali sono le risposte più simili e i pattern più simili senza fare un match parola per parola.

25.4 Dialog Control

Qua in questo caso abbiamo una domanda e questa è un tipo di conversazione che necessita di retrieving, dobbiamo chiamare una API per svolgere il task e poi possiamo mostrare il meteo per i prossimi giorni. Poi il bot mi può chiedere se voglio altro.



Arriva la domanda e poi viene trasformata in una rappresentazione con word embedding e poi le entità vengono estratte. Le entità sono poi passate ad un entity tracking e poi finiscono nella rappresentazione dell'interazione corrente. Tutto questo viene mandato in input ad una

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

RNN che poi produce un output che viene inviato ad un layer di Dense e di Softmax, questo viene combinato con l'output della precedente domanda e poi si va a produrre una possibile risposta. Quella nell'immagine sopra è la struttura completa che viene utilizzata, la NN viene sfruttata per capire quale azione devo effettuare selezionando il pattern più adatto per questa occasione. È una combinazione di retrieval e di action based in una NN che è in grado di memorizzare il passato della conversazione.

25.5 Challenges

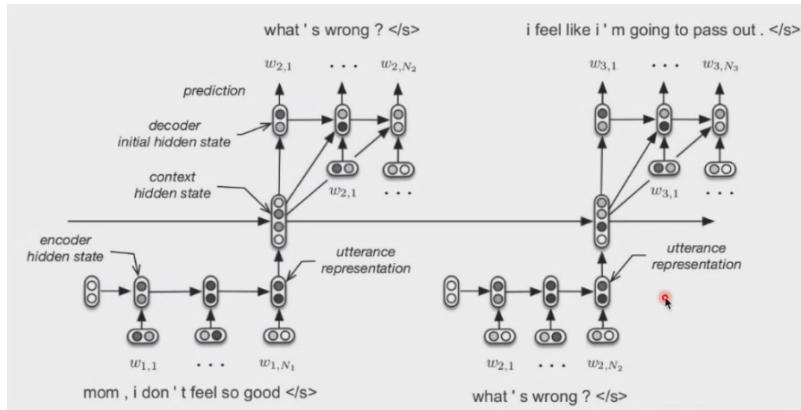
Incorporare il contesto è una dei problemi più grandi che abbiamo con i chatbot, in una conversazione le persone tengono traccia nella loro mente degli argomenti di cui si sta parlando.

Sordoni ha proposto di considerare un contesto in più nel passato concatenandolo con quello attuale. Una soluzione semplice è rendere l'input più grande considerando come input tutta la storia passata. Possiamo farlo concatenando tutto il passato e sperando che la NN abbia imparato e possa capire cosa è successo nel passato. Questo è un esempio di conversazione tra bot e umano:

Machine: describe your problem Human: i am having problems accessing the vpn Machine: could you please let me know what are the operating systems you are using ? Human: i am using linux Machine: ok it was in the ssl vpn on which you stated it was broken Machine: please check if you are logged in to corp network Machine: what is the error that you are running please Human: i am seeing an error related to vpn Machine: what is the error message that you are getting when connecting to vpn using network connect ? Human: connection refused or something like that Machine: may i know the version of network connect you connect ?	Human: i am not sure i know that Machine: is the network connect prompting for any user certificate Machine: ? Human: yes Machine: are you at home or at work at the moment ? Machine: office ? Human: i am at home Machine: try this Machine: goto <URL> page and enter your name and password Human: done Machine: try logging in with <NAME> and check if its working fine now Human: yes , now it works ! Machine: great Machine: anything else that i can help Human: no , that was very useful Machine: you are other than <NAME> :) Machine: have a nice night
--	---

Questa è una conversazione che ha avuto successo, la conversazione è buona perchè riusciamo a mantenere una cronologia di quello che abbiamo detto in passato. Questa è una possibile soluzione per gestire questa comunicazione:

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.



Qua ad esempio abbiamo che l'input del secondo stage è l'output del primo stage, e produciamo anche un output. Anche in questo caso abbiamo una hierarchical encoding.

25.6 Diversity

Un problema che vogliamo risolvere con i chatbot è fornire non sempre la stessa risposta ad una certa domanda ma variare perchè altrimenti la comunicazione diventa noiosa nel corso del tempo.

25.6.1 Coherent Personality

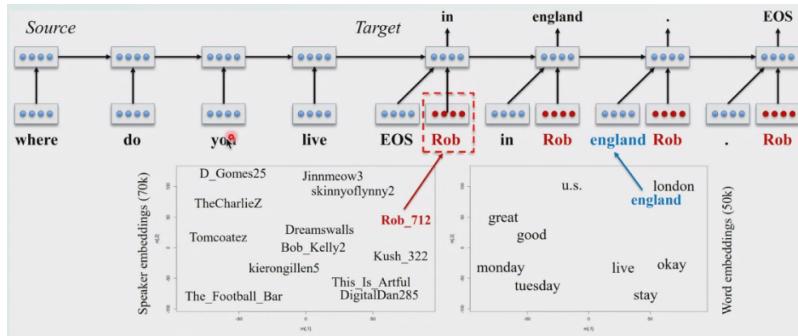
Vogliamo che sia mantenuta anche una consistenza tra le varie risposte che vengono date dal chatbot, per fare in modo che sembra che stiamo parlando sempre con la stessa persona.

Un altro problema è che vorremmo che il chatbot sembri una persona vera, in questo caso possiamo selezionare la risposta corretta data la personalità della persona che il chatbot sta impersonando. Ad esempio qua abbiamo delle risposte differenti perchè il chatbot sta impersonando differenti persone.

Alt	Realization	Extra
5	Err... it seems to me that Le Marais isn't as bad as the others.	1.83
4	Right, I mean, Le Marais is the only restaurant that is any good.	2.83
8	Ok, I mean, Le Marais is a quite french, kosher and steak house place, you know and the atmosphere isn't nasty, it has nice atmosphere. It has friendly service. It seems to me that the service is nice. It isn't as bad as the others, is it?	5.17
9	Well, it seems to me that I am sure you would like Le Marais. It has good food, the food is sort of rather tasty, the ambience is nice, the atmosphere isn't sort of nasty, it features rather friendly servers and its price is around 44 dollars.	5.83
3	I am sure you would like Le Marais, you know. The atmosphere is acceptable, the servers are nice and it's a french, kosher and steak house place. Actually, the food is good, even if its price is 44 dollars.	6.00
10	It seems to me that Le Marais isn't as bad as the others. It's a french, kosher and steak house place. It has friendly servers, you know but it's somewhat expensive, you know!	6.17
2	Basically, actually, I am sure you would like Le Marais. It features friendly service and acceptable atmosphere and it's a french, kosher and steak house place. Even if its price is 44 dollars, it just has really good food, nice food.	6.17

25.6.2 Speaker Embeddings

Possiamo imparare dei fatti associati con lo speaker specifico con cui stiamo parlando, ad esempio possiamo avere un embedding differente per ogni tipo di persona che stiamo considerando. Possiamo incorporare delle informazioni riguardanti lo speaker.



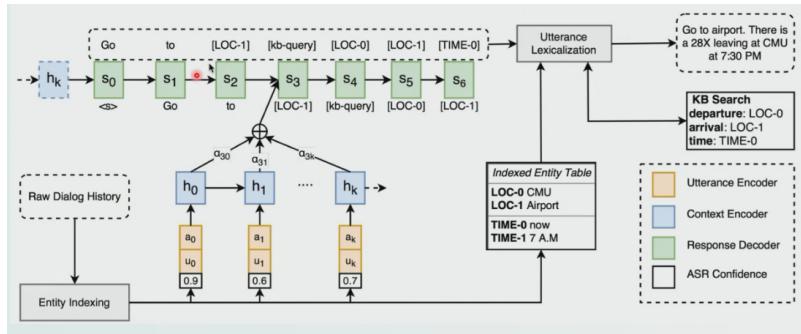
25.7 Evaluation

Abbiamo visto che per le traduzioni viene utilizzato il BLEU score, questo è utile se dobbiamo eseguire il ranking delle traduzioni. Ma BLEU non funziona bene se dobbiamo considerare i dialoghi. Quindi una proposta per la valutazione è stata DeltaBLEU che comprende una valutazione umana e aumenta i pesi in base a cosa è stato considerato buono e cosa non è stato considerato buono dall'umano.

Possiamo applicare il ML anche a questo

25.8 Advanced Features

Un aspetto importante delle chatbot è la possibilità di rispondere con delle informazioni che arrivano da altre fonti, questo è importante per dialoghi task oriented. Questo è un esempio di una soluzione che utilizza la RNN per selezionare la risposta estraendo delle informazioni da una KB. Le interazioni sono sulla destra, l'obiettivo è trovare un modo per andare all'aeroporto e trovare poi il bus. Dai dati che sono contenuti nella KB possiamo poi estrarre le informazioni che sono richieste dall'utente.



Per quanto riguarda l’architettura, abbiamo in input la nostra frase e questi dati vanno in una RNN, l’input corrente viene esteso anche con le informazioni del dialogo utilizzando le informazioni che sono state menzionate in precedenza. Poi l’output qua è il set di slot che devono essere riempiti per creare la query ed estrarre le informazioni. Questo comporta una KB search utilizzando il tempo e la localizzazione estratta dalla prima fase del lavoro.

25.8.1 Alexa Prize Socialbot

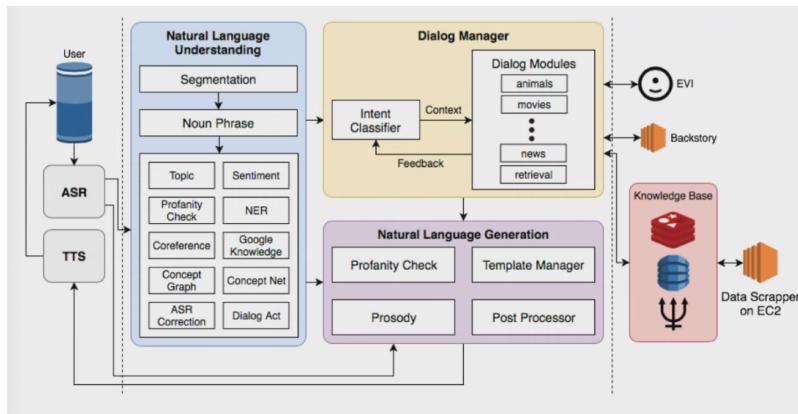
Consideriamo ora l’Alexa Prize, abbiamo visto alcuni sistemi sperimentali, prototipi, abbiamo visto la capacità di memorizzare informazioni dal passato, estrarre info dalla KB, mantenere dati della persona che sta parlando. Tutti questi sono degli aspetti che un chatbot dovrebbe garantire. L’idea dell’alexia prize è di creare un prodotto che sia utilizzabile in un contesto reale e non solamente in un contesto di ricerca.

Si voleva avere un sistema in grado di mantenere una conversazione per quanto più tempo possibile, almeno 20 minuti. Presentiamo in modo veloce la soluzione vincente che è stata chiamata Gunrock ed è stata presentata 2 anni fa, si tratta di un SocialBot che dovrebbe essere in grado di parlare di qualsiasi cosa. Durante le semifinali del prize ci sono state 40k conversazioni con questo socialbot.

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

Architettura del sistema Gunrock

Il vantaggio di questo prize è che il sistema viene testato su tutti i dispositivi Amazon Alexa, ci sono stati 1000 dialoghi al mese per i 5 mesi precedenti alla fase finale del torneo. Questa è l'architettura per Gunrock.



Sulla parte sinistra abbiamo l'echo dove la persona parla, l'utente parla e abbiamo un sistema di speech recognition fornito direttamente da Amazon, l'output che è testo viene inviato ad un sistema di NLU, quindi tutto quello che abbiamo studiato durante il corso, c'è sentiment analysis, topic recognition, parsing con i parse tree e poi abbiamo una coreference solution, cerchiamo di studiare le informazioni che arrivano da un concept graph. Tutto questo lo facciamo eseguendo una segmentazione della frase perchè magari all'interno ci sono altre sotto frasi che possono essere utili. Poi l'output di questo va al dialog Manager in cui cerchiamo di capire cosa chiede l'utente con questa domanda e questo poi viene inviato ad un modello in grado di classificare il tipo di richiesta che è stata fatta. Dati i vari possibili argomenti della conversazione, ad esempio potrebbe chiedere qualcosa riguardo allo stesso Alexa e in questo caso possiamo invocare le api EVI di amazon, altrimenti se chiede informazioni possiamo andare a chiedere ad alcune KB che magari sono

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

specifiche per film o libri o qualsiasi altro. Poi vengono estratte le informazioni da qua e finiscono di nuovo nel dialog manager e poi viene passato l'output al Natural Language Generation. Qua abbiamo vari stage di post processing per rendere più fluent la frase e anche un profanity check, questo deve funzionare in una vita reale e quindi le persona potrebbero offendersi se diciamo qualcosa di offensivo, quindi usiamo questo profanity check che viene fornito direttamente da amazon. Poi possiamo rendere le frasi ripetute in modo umano e quindi abbiamo la parte di prosody.

Questa è l'architettura generale del sistema, ci sono un sacco di problemi che devono essere ancora risolti, ci sono problemi anche nell'automatic speech recognition poi possiamo fare un errore nel boundary detection tra le varie frasi. Poi possono esserci problemi con NLU quindi ambiguità ad esempio.

Qua introduciamo anche una idea di classificazione dei dialoghi, in questa taxonomy la richiesta viene divisa in categorie, abbiamo l'initiative e la responsive, la domanda può essere open ended e chiedere factual question o opinion, abbiamo tanti tipi differenti di domande che provano a rappresentare la domanda che viene fatta dall'utente.

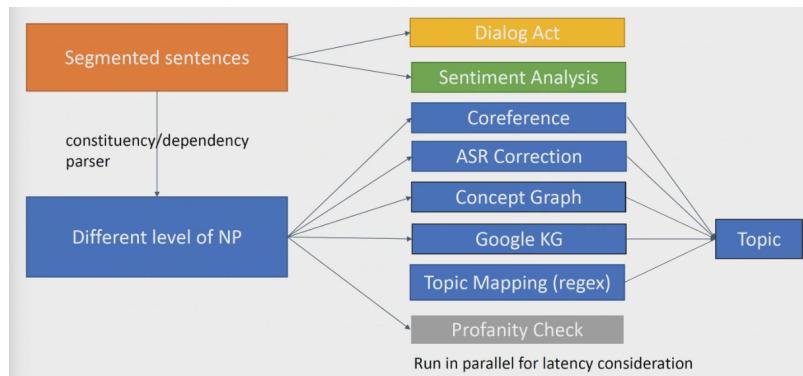
Dialog Act Tag	Description	Example
open_question_factual	factual question	How old is Tom Cruise; How's the weather today
open_question_opinion	opinionated question	What's your favorite book; what do you think of disney movies
yes_no_question	yes or no questions	Do you like pizza; did you watch the game last night
pos_answer	positive answers	yes; sure; i think so; why not
neg_answer	negative response to a previous question	no; not really; nothing right now, it was alright

Quindi abbiamo vari esempi di dialog act, per esempio in una frase "How old is Tom Cruise" abbiamo che stiamo chiedendo di alcuni fatti.

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

Poi ci sono anche domande che aspettano risposte positive/negative.

Per decidere il tipo di domanda, in NLP segue una pipeline molto sofisticata, prendo l'input e la divido in frasi, poi si fa il dependency parsing.



Poi si applicano dei tool per NLU, ad esempio il coreference resolution è importante e poi usano anche un concept graph per correlare le entità con la conoscenza in un concept graph. Poi si usa la topic mapping. Tutto questo viene inviato in input alla selezione del topic.

Il modello è un Bert based che viene utilizzato come pre trained. I risultati:

	Pre(%)	Rec(%)	F1(%)
LSTM-text	75.94	75.91	75.51
LSTM-DA	75.83	73.48	73.77
BERT-text	79.57	79.31	79.11
BERT-DA	79.29	76.12	76.87
BERT-no_context	73.88	70.43	71.30
BERT-DA+text	79.79	79.47	79.28
BERT_F-text	79.83	79.64	79.40
BERT_F-DA	79.30	76.15	76.89
BERT_F-DA+text	79.93	79.61	79.44
BERT-SwDA	79.26	76.43	78.98
BERT-SwDA_F	79.58	79.76	79.28

Quello che funziona meglio è quello di Bert che include il dialog act component.

Qui abbiamo una conversazione tipo:

USR: Hey Alexa, let's chat.
SYS: Hi, this is an Alexa Prize Socialbot. I'm glad to meet you. How are you doing today?
USR: I'm okay. How are you?
SYS: I'm good! So, tell me more about yourself! Do you like movies?
USR: I do like movies.
SYS: Ouu (Filler), I'm excited to talk about movies. So. Umm (Filler), have you seen any movies lately?
USR: A Star is Born.
SYS: Ah, (Filler) I've seen A Star is Born too! Hmm (Filler). What would you rate this movie on a scale from 1 to 10?
USR: A ten.
SYS: <say-as interpret-as= 'interjection'> Wow! </say-as> (Speechcon)! A perfect 10! It really is an exceptional movie! Here is something I just learned. Some of the scenes were filmed at the Coachella Festival in 2017, where Lady Gaga was the first female headliner in ten years. Festival-goers paid ten dollars to see the scenes being filmed, although recording devices were locked away to prevent any footage being leaked. Any thoughts?

Qualsiasi utente può iniziare la conversazione con la prima frase che abbiamo nell'immagine.

I chatbot vengono usati molto in molte aree anche in education, healthcare, quindi hanno molte possibili applicazioni, possono essere de-

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

ployate su smart speaker, web ma lo stato dell'arte non è ancora adatto a fornire delle soluzioni perfette per i chatbot. Vedremo poi in seguito altri dettagli al riguardo.

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

Chapter 26

DialogFlow e Facebook Blender

26.1 Template Based with DialogFlow

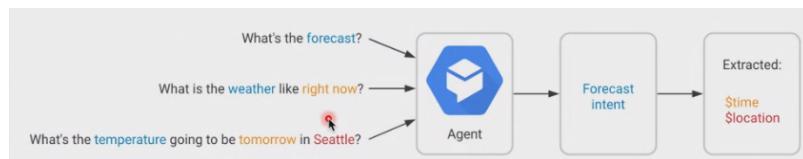
DialogFlow è un sistema che permette di creare delle Actions per il Google Assistant e ci permette di eseguire delle operazioni basilari di NLP, è in grado in automatico di fornirci lo speech recognition e anche il text to speech.

Quali sono i concetti chiavi in DialogFlow?

- Agent: un agent è l'elemento che gestisce la conversazione con l'utente, possiamo scegliere l'agente con cui vogliamo parlare e possiamo parlare all'assistente tramite l'agente che gestirà la conversazione. L'agent cerca di capire cosa vuole fare l'utente, questo è rappresentato da un Intent.
- Intent: è l'obiettivo dell'utente quando parla con l'assistente vocale. Un agent durante il dialogo dovrebbe essere in grado di gestire più di un singolo intent. Quello che fa DialogFlow è cercare un match tra quello che dice l'utente con un certo Intent.

- Entities: rappresentano qualcosa di rilevante che possono essere considerati come parametri per il task, possiamo avere per esempio delle espressioni che menzionano una data o una posizione. Le varie entities sono riconosciute da DialogFlow e per ognuna viene riconosciuto lo specifico tipo, l'estrazione delle entities dalle frasi viene svolta da DialogFlow.
- Context: serve per memorizzare il contesto della conversazione e ci permette di ritrovare quello che è stato detto in precedenza in modo da andare avanti nella conversazione.
- Fulfillment: in alcuni casi vogliamo che l'agente faccia qualcosa per noi, questa è l'azione che deve essere svolta che viene delegata ad un servizio dal web (tramite un web hook ad esempio che è una callback che riceve una richiesta, esegue l'azione e restituisce la risposta all'agente che poi invierà la risposta all'utente).

Un intent ha la seguente struttura:

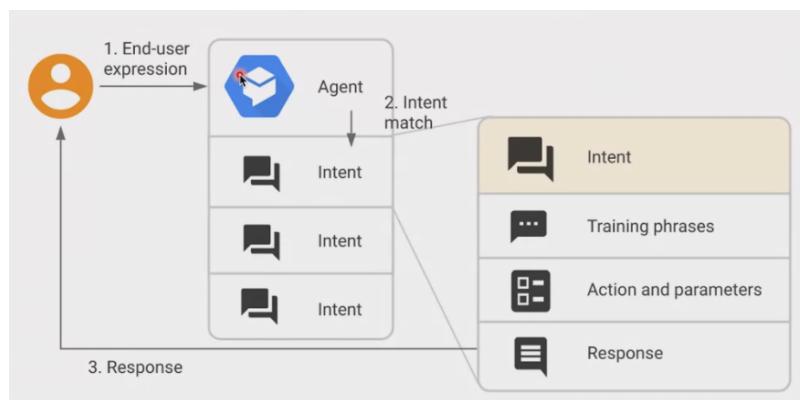


- Riceve in input la richiesta dell'utente.
- L'agente deve essere in grado di capire cosa è stato richiesto. Per un basilare intent ci sono vari modi differenti per formulare la domanda, all'interno della frase possiamo inserire dei parametri, più o meno a seconda della domanda che stiamo facendo.
- L'agente calcola alcune informazioni dal meteo per esempio e recupera dei dati relativi al tempo e alla localizzazione. Per ogni intent, c'è una action corrispondente.

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

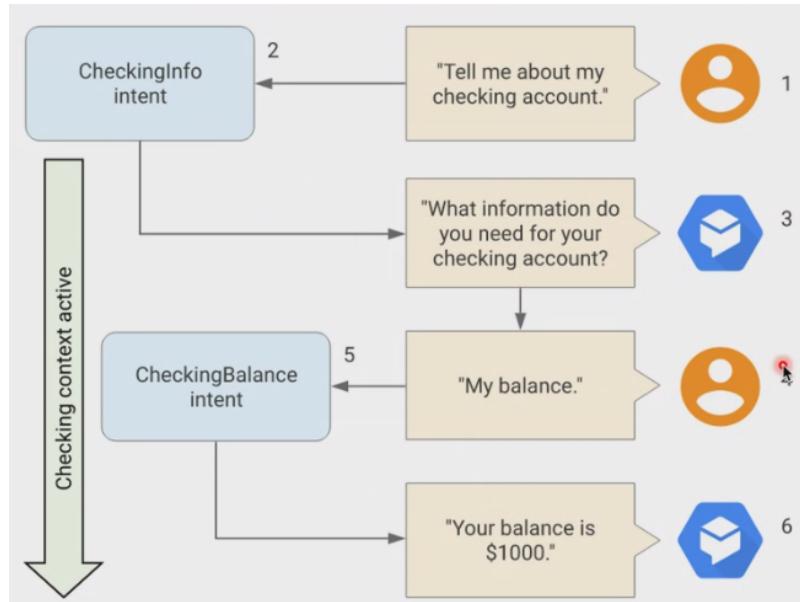
- L'agent poi produce una risposta quando completa l'azione che doveva essere svolta.
- Follow-up intents: un intent può avere una struttura gerarchica, quindi un follow up intent può essere un "figlio" della domanda che abbiamo fatto in precedenza.

Il sistema funziona in questo modo:



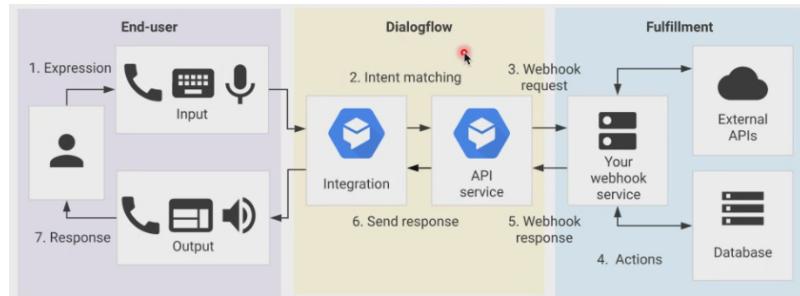
Abbiamo l'utente che esegue una domanda all'agent, questa viene matchata con l'intent e una volta che l'intent viene matchato. All'interno dell'intent abbiamo una training phase.

Il contesto è importante nel dialogo, ad esempio qua possiamo vedere un esempio:



Qua chiediamo una prima domanda che chiede informazioni ad un certo intent, poi facciamo all'utente una domanda in modo che lui possa richiedere una informazione specifica, ad esempio qua siamo interessati al nostro balance, qui poi viene aperto un altro intent in cui andiamo a recuperare il balance che stavamo cercando e restituiamo la risposta all'utente.

Follow-up intent Quando l'azione che richiediamo deve andare da un servizio esterno per ottenere la risposta che stiamo cercando.



Qua ad esempio l'utente chiede una domanda, poi dopo DialogFlow

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

deve andare a fare una chiamata a delle API che vanno a fare una richiesta ad un servizio esterno che è completamente separato.

Questa è più o meno tutta l'architettura che viene utilizzata.

26.1.1 Esempio di utilizzo di DialogFlow

- Per prima cosa dobbiamo creare il nostro Agent, vogliamo farne uno che mi permetta di prenotare un tavolo al ristorante.
- Poi dobbiamo creare un nostro intent, qua ad esempio abbiamo creato il nostro Welcome Intent.
- Una volta che abbiamo creato l'intent possiamo andare ad aggiungere delle frasi di training. Abbiamo tre frasi, se l'utente dice una di queste frasi, non deve esserci un match perfetto

The screenshot shows the DialogFlow interface for creating a 'Default Welcome Intent'. At the top, there's a title bar with the intent name 'Default Welcome Intent' and a 'SAVE' button. Below the title, there are three main sections:

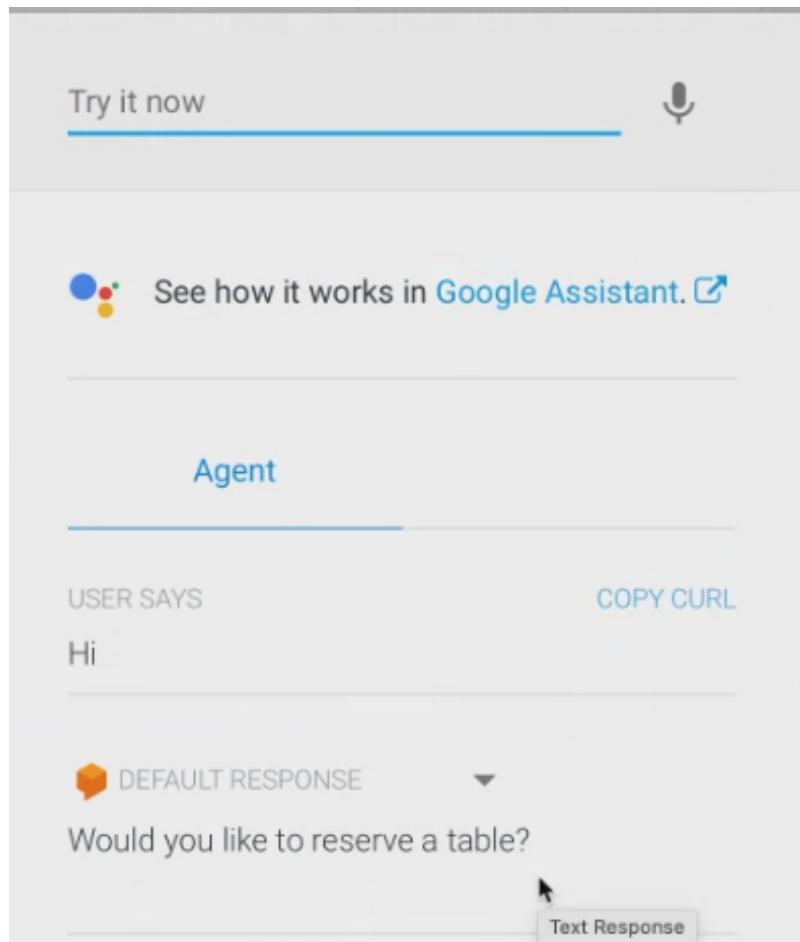
- Contexts**: A dropdown menu.
- Events**: A section containing a single event entry: 'Welcome' with an 'X' icon and a link to 'Add event'.
- Training phrases**: A section with a search bar 'Search training phrase' and two entries:
 - 'How are you today?' with a timestamp '2023-09-18T10:00:00Z'.
 - 'How are you today?' with a timestamp '2023-09-18T10:00:00Z'.
 Below these entries is a table with columns 'PARAMETER NAME', 'ENTITY', and 'RESOLVED VALUE'. It shows one row for the first entry with values 'date-time', '@sys.date-time', and 'today'.

Da queste frasi vengono anche estratte automaticamente delle en-

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

tity, ad esempio dalla prima frase viene estratto il parametro "today" che si riferisce al tempo.

- Poi possiamo aggiungere una risposta alle domande che abbiamo inserito in precedenza.
- Dalla console che c'è nella pagina di Dialog Flow possiamo provare il nostro intent.



- Creiamo un altro intent, questo è quello per fare la prenotazione.

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

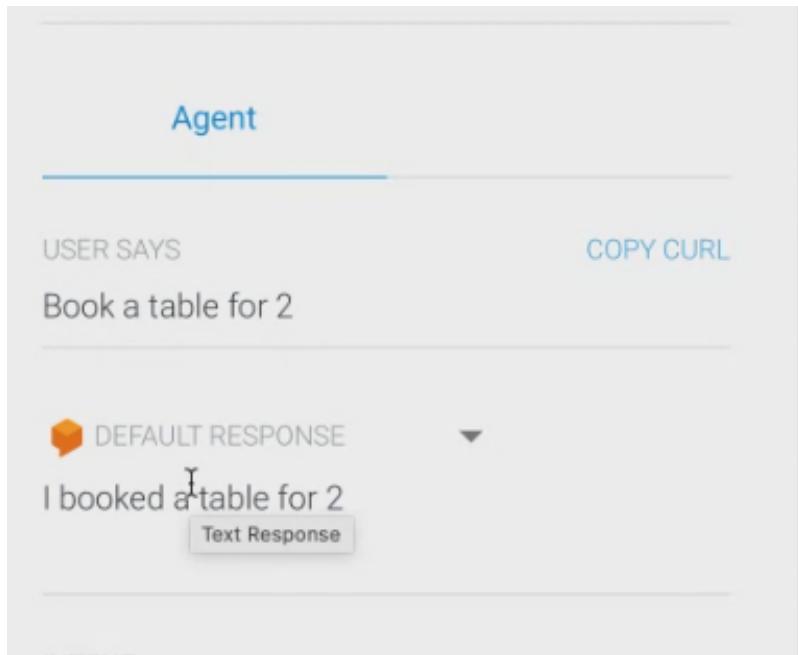


- Anche qua dobbiamo aggiungere le nostre training phrases, ad esempio possiamo aggiungere "Book a table" oppure "Make Me a reservation" e "Reserve a Table".
- Poi dobbiamo aggiungere una risposta, quindi ad esempio possiamo fargli dire "I booked a table for you" o comunque qualcosa di simile.
- Possiamo anche fare qualcosa di più sofisticato, ad esempio possiamo fargli prenotare un tavolo per 3 persone. Produciamo una nuova frase che ha un parametro, ovvero una entity.

PARAMETER NAME	ENTITY	RESOLVED VALUE
number	@sys.number	3

Il parametro lo possiamo personalizzare e possiamo indicare che quel parametro per noi indica il numero di ospiti per cui vogliamo prenotare.

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.



In questo modo è stato in grado di estrarre il numero di ospiti e lo usa anche all'interno della risposta.

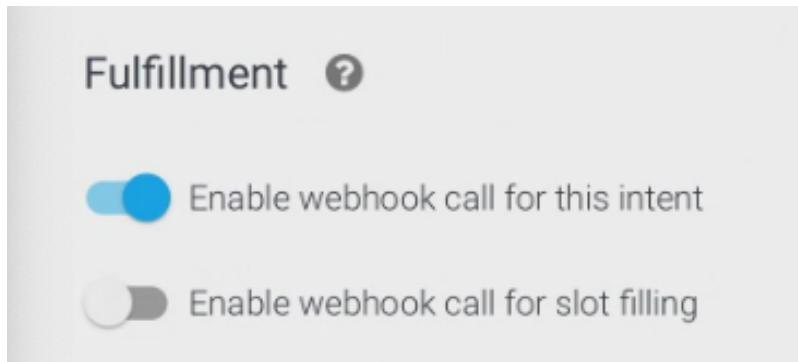
- Supponiamo di non inserire nella domanda il numero di ospiti, abbiamo bisogno di una richiesta da parte dell'assistente che mi deve chiedere per quante persone prenotare. Quindi dobbiamo aggiungere un prompt.

The screenshot shows the 'Action and parameters' configuration screen. At the top, it says 'Action and parameters'. Below that, there is a search bar with the placeholder 'Enter action name'. Underneath the search bar is a table with columns: REQUIRED, PARAMETER NAME, ENTITY, VALUE, IS LIST, and PROMPTS. There are two rows in the table. The first row has 'REQUIRED' checked, 'PARAMETER NAME' set to 'guests', 'ENTITY' set to '@sys.number', 'VALUE' set to '\$guests', 'IS LIST' unchecked, and 'PROMPTS' set to 'How many people...'. The second row has 'REQUIRED' unchecked, 'PARAMETER NAME' set to 'Enter name', 'ENTITY' set to 'Enter entity', 'VALUE' set to 'Enter value', 'IS LIST' unchecked, and 'PROMPTS' set to '-'.

- L'ultimo step consiste nel connettere l'agente con un fulfillment

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

intent. È una richiesta che va ad un servizio esterno.



Fatto questo passaggio di attivazione possiamo passare alla tab Fulfillment e poi scrivere il codice per eseguire la nostra richiesta. In questo codice eseguiamo una chiamata e questo mi restituisce la risposta che dobbiamo inviare all'utente. Un esempio del codice che possiamo usare qua:

```
function createBooking(agent) {
  let guests = agent.parameters.guests;
  let time = new Date(agent.parameters.time);
  let date = new Date(agent.parameters.date);
  let bookingDate = new Date(date);
  bookingDate.setHours(time.getHours()); bookingDate.setMinutes(time.getMinutes());
  let now = new Date();
  if (guests < 1){
    agent.add('You need to reserve a table for at least one person. Please try again!');
  } else if (bookingDate < now) {
    agent.add('You can't make a reservation in the past. Please try again!');
  } else if (bookingDate.getFullYear() > now.getFullYear()) {
    agent.add('You can't make a reservation for ${bookingDate.getFullYear()} yet. Please choose a date in ${now.getFullYear()}.');
  } else {
    let timezone = parseInt(agent.parameters.time.toString().slice(19,22));
    bookingDate.setHours(bookingDate.getHours() + timezone);
    agent.add(`You have successfully booked a table for ${guests} guests on ${bookingDate.toString().slice(0,21)}`);
    agent.add('See you at the restaurant!');
    agent.add('Have a wonderful day!');
  }
}
```

Qua utilizziamo Node.js per invocare una certa funzione, la funzione viene invocata e possiamo passare il numero di parametri della richiesta e poi vengono fatti dei check, ad esempio vogliamo che venga fatta una prenotazione per più di 0 persone. In base

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

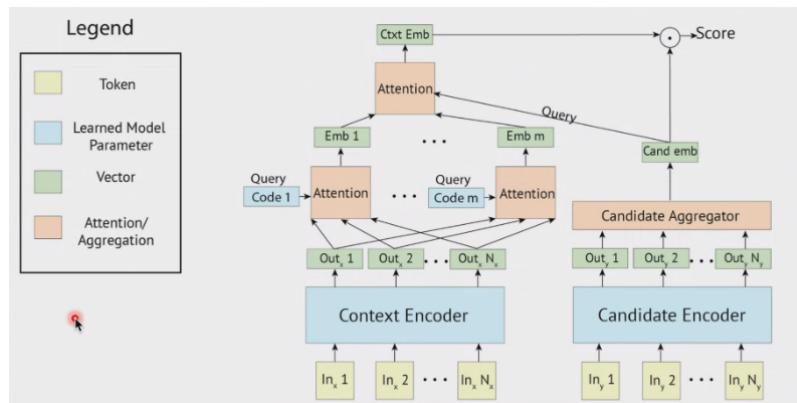
al tipo di errore che riceviamo poi possiamo restituire una frase differente all'agent in modo che mostri l'errore corretto. Se invece non ci sono errori passiamo all'agent una frase in cui confermiamo la prenotazione.

26.2 Facebook BlenderBot

Si tratta di un sistema che è basato sul Transformer, è stato creato da Facebook, sono stati creati modelli di vari dimensioni, il più grande ha 9.4 milioni di parametri. Possiamo avere una conversazione con questo bot e sembra molto naturale la conversazione.

Per eseguire il training di questo sistema sono stati utilizzati 1.5 miliardi di esempi di training e questo richiedeva grande potenza di calcolo. La neural network era troppo grande per entrare in un singolo dispositivo quindi sono state utilizzate delle tecniche di parallelismo per eseguire il training in parallelo. Queste tecniche hanno permesso di dividere la NN in varie macchine cercando comunque di massimizzare l'efficienza.

Attualmente questa soluzione è lo stato dell'arte nel campo dei chatbot, questa qua sotto è una overview dell'architettura del sistema, quello che si ha è un approccio encoder-decoder.



Abbiamo la frase precedente che viene data in input ad un context

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

encoder che produce in output una rappresentazione, poi si utilizza un attention mechanism che guarda a tutto l'output della rappresentazione e con questo cerchiamo di capire quale parte della precedente domanda è più rilevante per la domanda corrente, quindi si mette tutto insieme e pronunciamo un embedding finale. Abbiamo un encoder nella seconda colonna che invece riguarda la domanda attuale che abbiamo fatto al sistema. Anche qua abbiamo un encoder che quindi mi crea una rappresentazione della frase che abbiamo in input, questo candidate embedding viene combinato poi con l'attention del contesto per creare uno score.

26.2.1 Skills

Una volta che abbiamo capito quali sono le parti importanti della domanda, possiamo utilizzare un classico Seq2Seq Transformer per generare una risposta. Il training della neural network viene effettuato tipicamente cercando di minimizzare la perplexity.

Quando abbiamo parlato di QA abbiamo parlato di un sistema in grado di prendere dati da Wikipedia e abbiamo visto come utilizza l'inverted index per cercare di ottenere dei candidati per una risposta e poi prende la risposta da questi candidati. Anche qua utilizziamo Wikipedia ed estraiamo le nostre risposte.

Quindi abbiamo un transformer model e poi andiamo a scegliere la risposta migliore e poi generiamo la risposta. Il sistema è stato valutato e confrontato con altri sistemi che attualmente sono lo stato dell'arte come ad esempio Meena di Google. In alcuni casi Meena è migliore in termini di score che è stato raggiunto, in altri casi invece è Blender che funziona meglio.

26.2.2 Conclusione

Non siamo ancora nella posizione di poterci fidare dei chatbot per qualsiasi cosa.

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

Chapter 27

Trends and Future of NLP

27.1 Introduzione

Abbiamo visto grandi cambiamenti nel campo della NLP, gli studi sono partiti negli anni 40 con tante aspettative ma in realtà è stato molto più complesso del previsto. Le cose sono cominciate a cambiare negli anni 90 con l'introduzione di metodi statistici e poi nel 2010 con l'introduzione del Deep Learning. Un'altra rivoluzione c'è stata poi nel 2015 nella Machine Translation con il transformer e Bert che poi hanno permesso di applicare queste stesse tecniche anche ad altri task.

In generale il deep learning è diventato uno dei principali topic nel campo del Natural Language Processing e il numero di paper in cui vengono utilizzate queste tecniche sono cresciuti in modo esponenziale.

Cosa ha reso possibile tutto questo? L'idea che ha permesso di utilizzare il deep learning è stata l'introduzione degli embedding per rappresentare le parole presenti in un testo. Questo ha reso possibile eseguire il training di questi word embedding da testi non labellati, la cosa importante è avere dei dati. Una volta che abbiamo poi il language model possiamo utilizzarlo come pre trained stage in un modello e ne abbiamo visti vari di modelli, ciascuno in grado di ottenere dei risultati migliori

rispetto a quelli che avevamo avuto con le tecniche precedenti.

Poi l'introduzione dell'attention è stata anch'essa fondamentale per permetterci di focalizzarci sulla parte della traduzione più importante. Un'altra cosa che abbiamo visto è l'integrazione della Neural Network con della conoscenza esterna che è fuori dalla neural network.

Negli ultimi anni in particolare abbiamo avuto le seguenti novità:

- Attention
- Seq2Seq
- Large Scale QA e comprensione del dataset
- Sono nate librerie e framework che permettono più facilmente di creare i nostri sistemi, sono in particolare Tensorflow e PyTorch.

27.1.1 Applicazioni

Per tutti questi sistemi ci sono varie possibili applicazioni, ad esempio una di queste è prendere una fotografia ed utilizzare una LSTM per aggiungere una descrizione all'immagine, un'altra consiste nel prendere in input una immagine e poi andare ad estrarre i contenuti dell'immagine nominandoli, ovviamente per questo servono molti dati di training. Per questo ultimo sistema in particolare la cosa sorprendente è che funziona end to end nel senso che non abbiamo bisogno di fare preprocessing dell'immagine a mano, viene fatto tutto dal sistema. Questo è molto interessante perché riusciamo quindi a rimuovere molte difficoltà nel riconoscimento delle immagini.

27.1.2 Attention

Il meccanismo dell'attention è stato applicato alla NMT ed è particolarmente utile anche in Aspect Based Sentiment e in Topic Classification.

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

Con la Aspect Based sentiment analysis in particolare non consideriamo solamente se l'intero tweet è positivo o no ma anche le varie parti della parola che devono essere estratte per capire lo stato d'animo della persona rispetto a quella specifica parte della frase.

27.2 Deep NLP

Tramite le Neural Network abbiamo ottenuto grandi risultati nel campo della Machine Translation e anche nel campo del Language Modeling o del Question Answering. In molti task le neural network hanno quindi grande successo. Quello che è veramente impressionante è il modo in cui funzionano i sistemi, perché sembra tutto magico, possiamo interpretare questo come l'abilità di far emergere delle proprietà dei dati. L'altra cosa sorprendente è che oltre a funzionare funzionano anche bene.

Una cosa in cui non funzionano troppo bene invece è la memorizzazione di relazione tra certi dati, ad esempio se chiediamo dove è nato Dante non è detto che un modello che utilizza Bert o Elmo sia in grado di rispondermi correttamente. Nonostante questo sono in grado di catturare alcune relazioni, ad esempio se utilizziamo il corpus Google-RE e Bert possiamo essere in grado di memorizzare una quantità di fatti e relazioni anche sorprendente.

Un modello come Bert che è in grado di fare molte cose deve aver accumulato molte informazioni dal nostro testo. È anche in grado di considerare la struttura grammaticale di un testo? Bert è in grado di capire queste relazioni che sono nascoste nei vettori che rappresentano le parole presenti all'interno del testo? Si ci sono delle ricerche che sono state effettuate, in pratica Bert codifica anche un undirected parse tree.

In generale i neural model sono complessi, apparentemente possono gestire tante informazioni e possono essere usati per molti task, sia direttamente sia tramite pre-training. Sono sorprendenti perché non sappiamo bene come funzionano, le persone stanno continuando a studiare questi

modelli e in particolare stanno cercando anche di capire quali sono le similitudini con il cervello umano. Stanno cercando di capire come mai piccoli cambiamenti nel modello possono portare grandi cambiamenti nei risultati e in generale come funzionano under the hood. Questi metodi che utilizzano il Deep Learning comunque possono essere integrati all'interno di progetti NLP, questa è una possibilità affascinante per il futuro.

27.3 Inference from Memory

Un'altra cosa che richiede miglioramenti nei tool per eseguire Natural Language Processing o NLU è il Memory Systems. I sistemi sono in grado di produrre risposte dalla conoscenza ma non sono in grado di fare inferenza. I modelli sono molto ricchi, ma ci sono ancora alcune difficoltà, ad esempio ci sono difficoltà nel QA nel momento in cui vogliamo contare qualcosa. Sono anche modelli che funzionano bene se utilizziamo un singolo task ma poi se abbiamo un altro task ripartono da 0, in questo modo non riusciremo mai arrivare ad un modello generale per NLP. Sarebbe utile saper gestire vari task contemporaneamente in modo da poter utilizzare a seconda del tipo di problema che vogliamo utilizzare.

Deep Sequence Models: sono stati studiati da Yoav Goldberg, il parser è davvero fondamentale per capire cosa realmente una frase vuole dire e per portare a termine un task di NLP? Non ha dimostrato che porta significativi benefici utilizzando un parser, LSTM o altri meccanismi in grado di eseguire delle analisi potrebbero non essere necessari....

Ci sono tre task che sono particolarmente importanti, NLP Complete Super Tasks, questi task sono sufficientemente sofisticati per sistemi NLP:

- Language Modeling è un NLP-Complete Super Tasks
- Question Answering

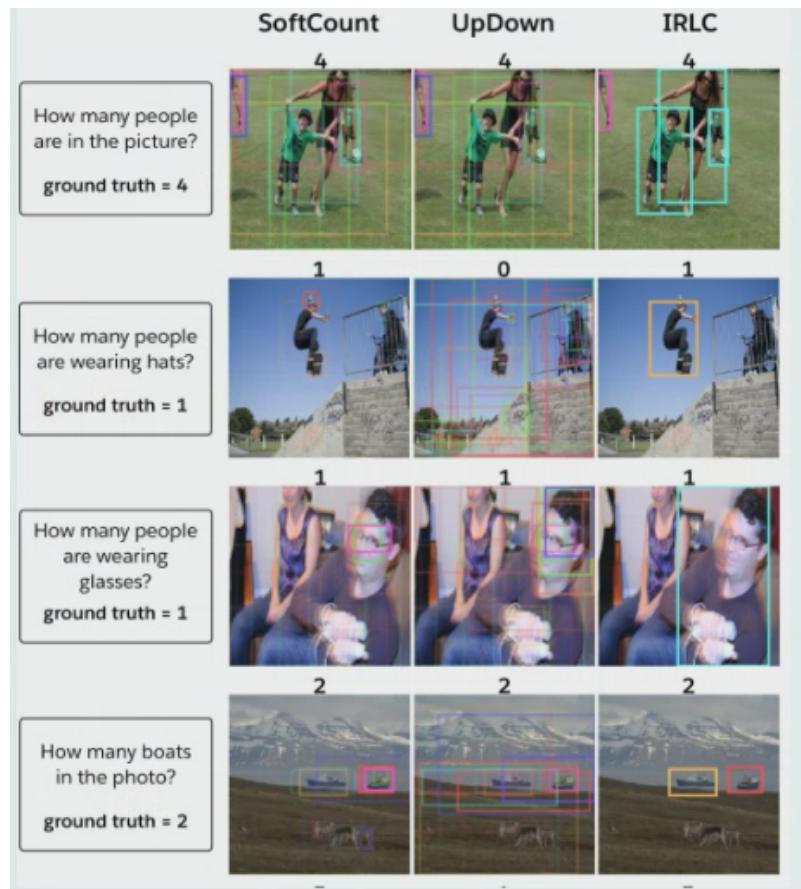
Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

- Counting

Per il QA è sicuramente NLP Complete perchè possiamo fare una qualsiasi domanda al nostro sistema, rispondere alla domanda richiede l'esecuzione di Named Entity Recognition.

Seq2SQL: Abbiamo una domanda e estraiamo delle feature che ci interessano e poi la trasformiamo in una query SQL per estrarre la domanda e per poi presentarla.

Un altro task difficile è il counting, ad esempio potremmo avere delle fotografie e vorremmo essere in grado di capire quante persone ci sono nella foto, il conteggio non è molto semplice.



Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

L'altra cosa che deve essere superata è la ricerca della corretta architettura da utilizzare.

Problemi con l'inferenza: Dobbiamo andare dal Machine Learning al Machine Reasoning, apparentemente questo è un qualcosa di molto complicato.

27.4 Limiti di Deep NLP

Abbiamo visto che siamo in grado di fare molte cose allo stato dell'arte ma ci sono ancora difficoltà con il Multitask learn, il learning da pochi esempi. Un grande problema è quello di avere dei dataset che non sono mai abbastanza per fare il training. Dato che ci sono centinaia di linguaggi sarebbe necessario avere la stessa quantità di dati labellati per ciascun linguaggio, una soluzione possibile consiste nell'utilizzare degli unlabeled data.

Un trend che abbiamo visto è che si stanno ottenendo modelli via via più grandi.

Ci sono anche problemi etici che sono molto rilevanti e in particolare sull'impatto che può avere l'utilizzo di tecniche di questo genere.

Bert è stato veramente un punto di svolta, è la morte dell'architecture engineering? NO perchè comunque possiamo ancora migliorare l'architettura di Bert.

Nell'industria NLP è stata utilizzata in molte applicazioni, smartphone, smartspeaker, nel campo medico.

27.5 Conclusione

Il deep learning ha offerto grandi potenzialità producendo sistemi che funzionano molto bene in vari campi. Nel futuro abbiamo bisogno di creare sistemi che possano combianre l'internal memory con materiale che

Questi riassunti sono stati prodotti da Luca Corbucci tenendo conto delle lezioni, delle slide, del libro Speech and Language Processing e di vari paper e articoli. Questo materiale non deve essere venduto e non sostituisce il libro di testo o le lezioni.

proviene dal web. Vogliamo muoverci dal Machine learning al machine reasoning e in particolare vogliamo essere in grado di eseguire l'inferenza. Sappiamo che non è un task semplice ma necessitiamo di sistemi che siano in grado di eseguire inferenza e di produrre risultati che non sono già presenti all'interno del dataset. NLP al momento sta avendo un grande impatto e alcuni problemi come il bias e la sicurezza stanno acquisendo una importanza sempre maggiore.

Bibliography

- [1] Chris Alberti, Kenton Lee, and Michael Collins. A bert baseline for the natural questions, 2019.
- [2] Saif Mohammad, Svetlana Kiritchenko, and Xiaodan Zhu. NRC-canada: Building the state-of-the-art in sentiment analysis of tweets. In *Second Joint Conference on Lexical and Computational Semantics (*SEM), Volume 2: Proceedings of the Seventh International Workshop on Semantic Evaluation (SemEval 2013)*, pages 321–327, Atlanta, Georgia, USA, June 2013. Association for Computational Linguistics.
- [3] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.