Alma Mater Studiorum
Università di Bologna
Campus di Cesena

DIPARTIMENTO DI INFORMATICA – SCIENZA E INGEGNERIA
Corso di Laurea in Ingegneria e Scienze Informatiche

TRADUZIONI AUTOMATICHE

Relatore:
Prof. Giovanni Delnevo

Presentata da:
Alessandro Pioggia

Sessione III
Anno Accademico 2021-2022

Desidero dedicare la tesi a coloro che lo hanno reso possibile.

Introduzione

Questa è l'introduzione.

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Contesto

1.1 Neural machine translation - NMT

La neural machine translation (in italiano traduzione automatica neurale (NMT)) è un approccio alla traduzione automatica che utilizza una rete neurale per prevedere la probabilità di una sequenza di parole, modellando in genere intere frasi in un unico modello integrato. La creazione del modello sfrutta interamente il machine learning, dunque è in grado di apprendere autonomamente sulla base dei training alla quale prende parte.

Differenze dai modelli tradizionali

- Rispetto ai modelli di traduzione tradizionali, chiamati SMT(statical machine translation), sfruttano una quantità infinitesimale di memoria;
- grazie all'utilizzo del deep learning nella creazione della rete neurale, è possibile optare per un addestramento del tipo end-to-end, questo permette di semplificare la pipeline di sviluppo, con conseguente aumento di prestazioni non indifferente. Per andare più nello specifico:
  - una pipeline tradizionale ha questo aspetto:
    voce(input) → riduzione del rumore (noise) → estrazione dei fonemi → composizione delle parole → trascrizione

– una pipeline che sfrutta il deep learning si presenta invece così: voce(input) $\rightarrow DNN \rightarrow$ trascrizione

Osservando le pipeline proposte, è possibile osservare che l'approccio tradizionale porta con sè una complessità non indifferente, ogni singolo layer citato deve essere ottimizzato separatamente, attraverso un preciso criterio. La pipeline creata a partire da una rete neurale invece, semplifica la comunicazione fra i due agenti(input $\rightarrow$ output desiderato).
Nota: DNN = deep neural network

1.1.1 RNN

Prima di descrivere in maniera analitica e precisa la struttura che permette la costruzione di un modello di encoding-decoding è necessario definire le RNN, o Recurrent Neural Networks. Le RNN sono delle reti neurali molto interessanti, perché a differenza dalle tradizionali straight forward neural networks, in cui l'informazione può andare solo in un verso (in avanti), in questo tipo di reti i nodi possono essere interconnessi ai livelli precedenti, ammettendo dei loop. Il fatto che i neuroni (o nodi, intesi come elementi che compongono la rete neurale) possano propagare il segnale in tutte le direzioni, permette di introdurre intrinsecamente il concetto di memoria. Questo perché, l'output di un neurone può ipoteticamente influenzare sè stesso in un diverso istante temporale (rispetto al presente). In particolare, una RNN richiede, oltre all'input tradizionale, lo hidden state, che non è altro che l'output prodotto dallo stesso neurone al passo precedente.

Perché sfruttare le RNN? Le RNN, grazie alla memoria, possono operare su dati dinamici mentre le SNN (straight forward neural networks) operano unicamente su dati statici. Grazie a questa proprietà, le reti neurali ricorrenti, sono in grado di trattare delle sequenze temporali, variabili nel tempo. Un esempio di applicazione può convenire nella interpretazione del linguaggio dei segni. In questo ambito applicativo non è richiesto solo di ri-

conoscere la mano, bensì di comprendere la sequenza di movimenti per poter dedurre ed interpretare il significato dei gesti. Quest'ultima constatazione ci rende in grado di percepire l'importanza di questo tipo di rete neurale nell'ambito delle traduzioni automatiche, in quanto si ha a che fare con sequenze di parole, non necessariamente definite in maniera statica che, per dare un senso logico alla frase, necessitano un meccanismo che sfrutti la memoria (e non solo). Per concludere l'approfondimento, considero fondamentale precisare che, by default, la memoria di un neurone è piuttosto breve, dunque gli output generati dai primi layer, difficilmente condizioneranno quelli presenti in livelli più avanzati. A questo proposito sono state studiate delle reti neurali a lunga memoria, quali le LSTM (Long Short Term Memory) e le GRU (Gated Recurrent Units).

1.1.2 Transformers

Nell'ambito del NMT, il modello maggiormente utilizzato è il transformer, che negli ultimi anni ha preso il posto delle reti neurali RNN, descritte nella sezione precedente.

Caratteristiche e peculiarità

La tipologia di modelli citati in questa sezione sono nati con l'obiettivo di evitare la ricorsione, facendo spazio a delle tecnologie che garantissero un tipo di computazione che ammettesse il parallelismo. Le caratteristiche principali sono le seguenti:

- data una sequenza di dati in input, a differenza delle reti RNN, che operano word-by-word (ossia hanno la necessità di analizzare ogni singolo elemento della struttura), il transformers sono in grado di processare l'input **"as a whole"**, ossia come un unico grande dato;
- l'introduzione della **self-attention**, che consente un notevole miglioramento nella predittività, ha dato il nome all'articolo che ha presentato questa tipologia di modelli, ovvero "Attention is all you need";

- il **positional embedding**: si tratta di una tecnica che, insieme all'attention, ha come obiettivo quello di eliminare i processi che sfruttano la ricorsione. In particolare, l'idea è di assegnare dei pesi legati alla posizione di una parola all'interno della frase, in modo da verificare quali siano le tuple di parole che hanno più probabilità di essere accostate.

Il primo punto è fondamentale perché consente al modello di non dipendere da neuroni di layer lontani, processando l'intero input in una sola volta, viene eliminato il concetto di perdita di memoria. Il tutto può essere chiarito attraverso un semplice esempio:
Consideriamo un ipotetico articolo, supponendo che i puntini rappresentino il testo di intermezzo:

La nazionale italiana di basket ..... Pozzecco ha deciso di intervenire in sala stampa, ringraziando lo staff.

Dato questo input, il transformer, analizza l'articolo nel suo intero, mentre un modello tradizionale considera parola per parola, ovvero: {0:La, 1:nazionale, 2: italiana, 3: di, 4: Basket, ..., 1200:Pozzecco }. A giudicare da questo esempio si può osservare che è impossibile mantenere il contesto analizzando una parola per iterazione, specialmente se si ha necessità trovare un nesso fra due parole con indici molto distanti.

Descrizione tecnica dettagliata del modello

Si tratta di un sistema di encoding-decoding. In linea generale, l'encoder mappa l'input ricevuto in un vettore continuo, che contiene tutte le informazioni apprese dai dati ricevuti in ingresso. Il decoder a partire dal vettore continuo, passo per passo, genera un unico output. Il singolo output generato è al contempo condizionato dagli ulteriori vettori continui dati in pasto al decoder in precedenza. Questo passaggio avviene ricorsivamente, fino a quando il decoder non incontra il token <end>, rappresentante la end of sentence (in italiano, fine della frase).

Input embedding

Il primo passo è passare il nostro input al word embedded layer, il quale può essere considerato come una lookup table che contiene dei fattori di rappresentazione per ogni parola. In particolare, ogni parola viene mappata in un vettore avente valori continui, che la rappresentano, dal momento che il sistema apprende attraverso pattern numerici.

Positional encoding

In seguito alla definizione dell'embedded layer è necessario passargli delle informazioni circa la collocazione di ogni vettore all'interno della rete neurale, dal momento che non è più possibile sfruttare la ricorsione a questo scopo. Per fare ciò è stata studiata una brillante soluzione, che verte sull'utilizzo di queste due formule:

- $P(pos, 2i+1) = \cos(\frac{pos}{10.000^{2i/dmodel}})$
- $P(pos, 2i) = \sin(\frac{pos}{10.000^{2i/dmodel}})$

Per ogni step di esecuzione di indice pari, genera un vettore di valori continui attraverso la funzione coseno. Per quelli di indice dispari invece, il vettore viene generato sfruttando il seno. Ad ogni vettore risultante viene sommato l'embedding vector corrispondente (ottenuto al primo step). Per la generazione si utilizzano le funzioni seno e coseno per via delle loro proprietà lineari, che l'intelligenza artificiale è in grado di apprendere con maggiore facilità.

Encoding layer

Come introdotto inizialmente, l'encoder mappa l'input ricevuto in un vettore numerico contenente valori continui, rappresentanti le informazioni apprese dai dati ricevuti in ingresso. Contiene due sottomoduli:

- il primo è una unità dedicata alla self- attention chiamata **multi head attention layer**. Andando nello specifico, ;

- l'altro è una vera e propria **rete neurale feed forward.**

I due layer sono collegati fra loro non direttamente, fra loro si interpone un layer intermedio che si occupa della normalizzazione dell'output.

Multi-head attention layer

L'unità citata nel titolo, implementa uno specifico meccanismo di attention chiamato self-attention. Questa tecnologia consente ad un modello, attraverso principi combinatori accurati, di associare le parole presenti nell'input. Ad esempio, supponendo di avere la frase "ciao", "come", "stai", l'unità potrebbe ipoteticamente correlare : "ciao", "come" oppure "stai", "ciao". Attraverso questo tipo di associazioni, il modello può ad esempio interpretare la tipologia di frase, rendendosi conto in questo caso, di avere a che fare con una domanda. Queste intuizioni, permettono alla rete neurale di strutturare un possibile output.

Perché il meccanismo funzioni l'input deve essere passato come parametro a tre distinti layer, fra loro connessi, i quali genereranno i seguenti vettori continui:

- query vector;
- key vector;
- value vector.

In particolare il query vector contiene la richiesta, generalmente in formato json, che verrà etichettata dal key vector. Questa operazione è molto simile a quella che viene effettuata durante una richiesta ad un motore di ricerca, come ad esempio quello di youtube. Quando viene effettuata la ricerca, google etichetta il testo inserito dall'utente, in particolare associa la ricerca a descrizione, autore o titolo del video. Finita l'etichettatura, interroga il database, cercando i migliori match, ovvero i video strettamente compatibili con la richiesta.

Il vettore delle query dunque, viene moltiplicato per il vettore delle chiavi (o key vector), ottenendo come risultato la matrice degli scores, o punteggi. Le celle della matrice risultante contengono un valore numerico, il quale indica quanto forte sia il collegamento fra una parola e l'altra, ovvero l'attenzione che occorre porre. Quindi, semplificando il concetto, all'interno di un arco temporale ogni parola avrà a disposizione un punteggio in relazione ad ogni altra parola presente. Il focus aumenta con l'aumentare del valore contenuto nella cella.

81	18	91
509	1000	230
300	200	3300

Tabella 1.1: Ipotetica matrice degli scores, per semplicità utilizzerò la sua nomenclatura originale, ovvero score matrix.

Osservando la matrice in formato tabellare sovrastante, si nota subito che l'ordine di grandezza del contenuto delle celle necessita una sorta di normalizzazione, per evitare che, la moltiplicazione con altre matrici, generi valori troppo grandi. A questo proposito si effettuano le seguenti operazioni:

- $\frac{scorematrix}{\sqrt{d_k}} \rightarrow$ divido la matrice per la dimensione delle query e delle keys;
- $softmax(x)_i = \frac{exp(x_i)}{\sum_j exp(x_j)} \rightarrow$ in questo passaggio, viene presa come input la matrice ottenuta al passo precedente, x un elemento arbitrario. La matrice risultante chiamata sealed matrix, conterrà per ogni cella un valore probabilistico, il quale range va da 0 ad 1. Alle probabilità più alte verrà attribuita una maggiore attenzione (da qui, il termine attention), mentre quelle inferiori alla soglia verranno ignorate, perché non utili. This allows the model to be more confident on which words to attend to;

- $sealedmatrix \cdot valuematrix = output \rightarrow$ moltiplico la matrice ottenuta al passaggio precedente, che può essere considerata una matrice di attention, per il vettore dei valori definito inizialmente. L'output, come già accennato, oscurerà le parole con un punteggio softmax più basso, dando la priorità alle altre.

- infine nell'ultimo passaggio, l'*output* ottenuto al passaggio precedente viene passato in input ad un layer lineare, che sarà in grado di processarlo.

Perché si parli di multi-headed attention, è necessario che, i vettori presi originariamente in input (query, key e value vectors), vengano scissi n volte e ogni tripletta di vettori ottenuta, attraversi lo stesso processo di self-attention individualmente. La scissione dei vari array, si basa su criteri prevalentemente dimensionali, in quanto, a partire dal vettore originale, lungo $length$, si ricavano n array, che rappresentano una divisione dello stesso in n sottoarray. Ogni singolo processo di self-attention prodotto prenderà il nome di head, da questo deriva il nome di multi-headed attention layer.

Il seguente esempio può chiarire il concetto riguardante la scissione:

- array originali: $q = \{"hi", "how", "are", "you", "my", "friend"\}$, $k = \{1, 2, 3, 4, 5, 6\}$, $v = \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6\}$ con singolo layer di self attention h, potrebbe diventare:

  - $\{q_1 = \{"hi", how"\}, k_1 = \{1, 2\}, v_1 = \{0.1, 0.2\}\} \rightarrow head_1$
  - $\{q_2 = \{"are", "you"\}, k_2 = \{3, 4\}, v_2 = \{0.3, 0.4, \}\} \rightarrow head_2$
  - $\{q_3 = \{"my", "friend"\}, k_3 = \{5, 6\}, v_3 = \{0.5, 0.6\}\} \rightarrow head_3$

  Abbiamo dunque ottenuto tre tuple di array, di cardinalità 3 e ad ogni tupla è associata una unità individuale di self-attention.

Gli output prodotti da ogni singolo processo derivante dalla scissione, al termine della operazione verranno concatenati in un unico oggetto, che sarà a tutti gli effetti l'input del layer lineare, introdotto nell'ultimo punto della sottosezione precedente. Il fatto di avere a disposizione un input generato da n diversi processi di self-attention, fa in modo che, ogni head unit impari attraverso processi logici diversi e concatenandoli, si è in grado di offrire al modello un potere di rappresentazione più spiccato.

Per riassumere Il multi-headed-attention è un modulo, compreso in una rete di trasformatori, che riceve in input i pesi relativi all'attention (o attention weights) e produce un output, che contiene informazioni codificate. Le informazioni dicono al modello con quale frequenza e modalità le parole di una sequenza, devono essere associate.

Residual connections

Le residual connections (o connessioni residue), sono appartenenti ad un tipo di rete neurale chiamata **residual neural networks**. Si tratta di una rete la quale, attraverso le proprie connessioni, sia in grado di operare non considerando nel processo i layer intermedi. Una domanda che ci si potrebbe porre è la seguente: "a cosa serve saltare dei layer nell'esecuzione? non è controproducente?". Questo tipo di domanda è a mio avviso, molto opportuna, in quanto è vero che la logica generale sia contro intuitiva, allo stesso tempo però si può osservare che, lo skip di layer:

- risolve il noto ed importante problema dei **vanishing gradients**, ottimizzando la rete neurale. Nell'apprendimento automatico, il problema del Vanishing gradient si incontra quando si addestrano reti neurali con metodi di apprendimento basati sul gradiente e sulla back-propagation. In tali metodi, durante ogni iterazione dell'addestramento, ciascuno dei pesi della rete neurale riceve un aggiornamento proporzionale alla derivata parziale della funzione di errore rispetto al peso corrente. Il pro-

blema è che in alcuni casi il gradiente sarà di dimensioni infinitesimali, impedendo al peso di cambiare valore in maniera ottimale;

- **mitiga il problema di degradazione, meglio noto come accuracy problem (problema di precisione)**. La degradazione si verifica quando si ha a che fare con una rete con un grande numero di layer, perché è formalmente dimostrato che, aumentare il numero di strati, comporta un margine di errore più ampio.

Nel trasformer, l'importanza delle connessioni residue si può percepire dopo aver studiato ed analizzato con precisione tutti i singoli passaggi, indicati nel prossimo paragrafo.

Si parte unendo l'output generato dal **multi-level attention layer** all'input originale. Il risultato ottenuto sarà poi normalizzato, passando attraverso il normalization layer. Supponendo che $multiLevelHeadOutput$ sia l'output ottenuto dal multi-headed layer e che $input$ sia l'input originale,dato in pasto allo stesso, il tutto può essere formalizzato attraverso la formula:

- $LayerNorm(ResidualConnection = multiLevelHeadOutput, input)$

Ciò che ottengo dalla normalizzazione, passa attraverso una rete neurale feed-forward, che dunque non ammette backtracking e non ha memoria, nemmeno a breve termine. La rete neurale artificiale citata, è composta da 2 layer lineari, come quelli visti in precedenza, con la differenza che fra essi, viene collocato un layer ReLu. La funzione ReLu è una funzione di attivazione non lineare, molto nota in ambito deep learning. Si tratta di una sigla che sta per Rectified Linear Unit, semplificabile in "rettificatore". La particolarità della funzione è che attiva **solo un** neurone alla volta e non gli è concesso attivarli tutti contemporaneamente.

Il valore di uscita dalla rete feed forward, verrà a sua volta legato all'input originale, formando un'ulteriore connessione residua, che verrà poi normalizzata da un layer di normalizzazione. Incorporare al modello una

rete feed-forward serve per elaborare ulteriormente l'output della attention, dandogli potenzialmente una rappresentazione più ricca.

Questo ultimo tassello conclude la logica procedurale del modello encoder, il quale scopo, come già spiegato nell'introduzione del capitolo, è di offrire al decoder delle indicazioni da seguire, circa le parole alla quale deve essere dedicata una maggiore attenzione. Come con l'attention è possibile creare uno stack di differenti encoder, dove ogni singolo elemento della pila ha possibilità di imparare diverse rappresentazioni dell'attention. Quest'ultimo aspetto è molto interessante, perché aumenta il potere predittivo del rete del modello.

Decoder

Il ruolo del decoder consiste nel generare sequenze di testo, la sua struttura è molto simile a quella dell'encoder. In particolare, ha a sua disposizione:

- 2 multi-headed attention layers;
- un feed fordward layer, ovvero una rete neurale senza backtracking;
- un sub-layer di normalizzazione per ogni layer padre presente, oltre alla presenza di residual connections derivanti dall'utilizzo di una residual neural network (rnn);

Si tratta a tutti gli effetti di un modello autoregressivo (in inglese autoregressive model). Il decoder infatti, prende in ingresso gli output generati dall'encoder, oltre alle informazioni sull'attention degli stessi. Il layer elabora richieste fino a quando non viene generato l'¡end¿ token, il quale indica la fine della frase.

Modello autoregressivo

Un modello autoregressivo si fonda sulla misurazione della correlazione fra osservazioni, effettuate ad un istante temporale antecedente rispetto al

tempo di misurazione, allo scopo di predirre il valore che la variabile considerata assumerà al prossimo step (il nostro output). Nel caso in cui la variabile di input e output varino nella stessa direzione, ovvero incrementando o diminuendo insieme, si otterrà una correlazione positiva. Al contrario, se la variazione segue il senso opposto, la correlazione sarà negativa. All'aumentare del valore di correlazione, che sia negativo o positivo, aumenta la probabilità che attraverso l'input la macchina sia in grado di prevedere il futuro. Dal momento che la correlazione è fra la variabile al presente e sè stessa al passo precedente, si parla di autocorrelazione.

Passaggi effettuati nella fase di decoding

Embedding dell'output e positional encoding

L'input passa attraverso due layer, si tratta dell'embedding layer e il positional encoding layer. Il procedimento denota la posizione delle parole analizzate rispetto alla frase presa come riferimento.

Decoder multi headed attention 1

L'output ottenuto dal passaggio precedente vengono dati in pasto al primo dei due multi headed attention layer. In questa fase, il componente ha lo scopo di estrarre i punteggi relativi all'attention dell'input. Non bisogna confonderlo con il multi-headed attention layer presente nella struttura dell'encoder, questo perché, l'autoregressione richiede soluzioni leggermente diverse. In particolare, dal momento che processa l'input parola per parola è ritenuto necessario evitare che token futuri vengano condizionati. Ad esempio, supponendo che il modello stia analizzando la parola "tutto", della frase { "Io", "tutto", "bene"; "tu"?}, esso non deve avere accesso alle parole successive, perché generate in un istante futuro. Detto ciò, quindi, è importante fare in modo che la computazione di una parola, possa essere condizionata solo dalle precedenti (generate anteriormente), non dalle successive (generate

successivamente). Il metodo che rende possibile tutto ciò prende il nome di masking, e segue il seguente funzionamento:

- per evitare che il decoder prenda in considerazione token futuri, viene applicata una "look ahead mask" prima di calcolare il softmax della matrice dei valori e dopo la normalizzazione degli score (osservare la sezione della attention per chiarire). La maschera è una matrice delle stesse dimensioni della sealed matrix, ovvero la matrice degli score in seguito alla normalizzazione. Le celle sopra la diagonale assumono valore - ∞, mentre le altre valore 0. Il risultato è la matrice chiamata **masked scores**.

- il motivo per il quale si applica questo tipo di maschera, è legato al fatto che, applicando il softmax (visto nei capitoli precedenti), i valori con -∞ verranno sostituiti da zeri. La matrice risultante valorizza il punteggio di attention che verrà attribuito ad ogni token, a questo proposito, abbiamo che azzerando la parte sopra alla diagonale della matrice, stiamo implicitamente escludendo i token futuri dal calcolo.

0.7	*	*		0	$-\infty$	$-\infty$		0.7	$-\infty$	$-\infty$
0.5	0.1	*	+	0	0	$-\infty$	=	0.5	0.1	$-\infty$
0.3	0.2	0.2		0	0	0		0.3	0.2	0.2

Tabella 1.2: L'esempio sovrastante ci mostra l'applicazione della maschera alla matrice degli score (post normalizzazione). In particolare, alla matrice degli score viene sommata la maschera (matrice centrale), che va ad incidere solo sui valori situati sopra la diagonale. Gli asterischi presenti nella matrice di sinistra, vanno ad indicare valori randomici, non è importante conoscerli, dal momento che diventeranno pari a - ∞

.

$softmax($

0.7	*	*
0.5	0.1	*
0.3	0.2	0.2

$) =$

0.7	0	0
0.5	0.1	0
0.3	0.2	0.2

Tabella 1.3: Applicazione della funzione softmax alla matrice ottenuta in seguito al masking

.

Multi headed attention layer 2

Riceve in input il vettori delle query e values derivanti dall'encoder, mentre il vettore degli scores è l'output ottenuto dal primo layer di multi head attention.

Feed forward linear layer (Classifier)

L'output ottenuto in seguito all'elaborazione delle informazioni a carico del secondo layer di attention, passa attraverso una rete feed forward di tipo classifier.

1.2 Seconda Sezione

Ora vediamo un elenco puntato:

- primo oggetto
- secondo oggetto

1.3 Altra Sezione

Vediamo un elenco descrittivo:

OGGETTO1 prima descrizione;

OGGETTO2 seconda descrizione;

OGGETTO3 terza descrizione.

1.3.1 Altra SottoSezione

SottoSottoSezione

Questa sottosottosezione non viene numerata, ma è solo scritta in grassetto.

1.4 Altra Sezione

Vediamo la creazione di una tabella; la tabella 1.4 (richiamo il nome della tabella utilizzando la label che ho messo sotto): la facciamo di tre righe e tre colonne, la prima colonna "incolonnata" a destra (r) e le altre centrate (c):

$(1,1)$	$(1,2)$	$(1,3)$
$(2,1)$	$(2,2)$	$(2,3)$
$(3,1)$	$(3,2)$	$(3,3)$

Tabella 1.4: legenda tabella

1.5 Altra Sezione

1.5.1 Listati dei programmi

Primo Listato

```
        In questo ambiente     posso scrivere      come voglio,
lasciare gli spazi che voglio e non % commentare quando voglio
e ci sarà scritto tutto.
Quando lo uso è meglio che disattivi il Wrap del WinEdt
```

Capitolo 2

Secondo capitolo

Questo è il secondo capitolo.

2.1 Prima Sezione

Questa è la prima sezione.

2.2 Seconda Sezione

Questa è la seconda sezione.

Capitolo 3

Terzo capitolo

Questo è il terzo capitolo.

3.1 Prima Sezione

Questa è la prima sezione.

3.2 Seconda Sezione

Questa è la seconda sezione.

3.3 Terza Sezione

Questa è la terza sezione.

Conclusioni

Queste sono le conclusioni.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Quisque a magna quis nunc venenatis vestibulum. Curabitur commodo efficitur ipsum, non ullamcorper tellus. Duis dictum commodo nisi nec venenatis. Donec euismod pulvinar finibus. Suspendisse lorem mi, suscipit quis faucibus ut, luctus in justo. Cras pulvinar arcu ut ullamcorper pulvinar. Aliquam dictum tortor quis diam luctus, quis tristique tortor ultrices. Integer et lacus a velit efficitur convallis. Morbi enim erat, fermentum vel nulla id, viverra vehicula nisi. Integer non auctor leo, eu convallis massa. Cras eu cursus ligula. Nunc non purus et sem vehicula viverra ut nec nibh.

Quisque posuere purus quis eros auctor efficitur. Etiam mattis vitae nulla et blandit. Nulla a orci magna. Cras ac elit enim. Vestibulum nec nisl metus. Mauris congue velit nec malesuada scelerisque. Sed dignissim, enim vitae semper fermentum, mauris leo vestibulum nisl, in malesuada nibh felis nec dui.

Nullam sit amet tellus eget mi varius commodo. Vestibulum sit amet egestas odio. Nam in ullamcorper quam, nec efficitur augue. Curabitur eget elit in leo eleifend tempor vel lobortis lorem. Duis neque dui, tempus eu sollicitudin ac, lobortis sit amet odio. Morbi eleifend, tellus a varius consequat, enim erat sagittis justo, ac rutrum ipsum augue in leo. Suspendisse non mi ante.

Praesent sed pretium dui, id volutpat tortor. Suspendisse tortor lorem,

vestibulum vitae ullamcorper vitae, tincidunt nec leo. Proin interdum congue blandit. Ut bibendum sagittis leo, nec venenatis urna mollis id. Donec nec erat non justo maximus venenatis. In mollis elit eu odio maximus porta. Vestibulum varius turpis sit amet orci blandit, vitae volutpat erat viverra.

Suspendisse nunc urna, elementum ut purus a, sagittis porta velit. Integer ultricies convallis tortor id pellentesque. Duis et sem a mi bibendum congue. Morbi ut tellus cursus, laoreet ipsum rutrum, condimentum felis. Proin velit mi, ultricies a urna nec, facilisis pretium mi. Pellentesque tristique interdum purus, a facilisis mi tempor quis. Sed finibus venenatis ligula porttitor porttitor. Suspendisse cursus lorem nec velit commodo fringilla. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Quisque a magna quis nunc venenatis vestibulum. Curabitur commodo efficitur ipsum, non ullamcorper tellus. Duis dictum commodo nisi nec venenatis. Donec euismod pulvinar finibus. Suspendisse lorem mi, suscipit quis faucibus ut, luctus in justo. Cras pulvinar arcu ut ullamcorper pulvinar. Aliquam dictum tortor quis diam luctus, quis tristique tortor ultrices. Integer et lacus a velit efficitur convallis. Morbi enim erat, fermentum vel nulla id, viverra vehicula nisi. Integer non auctor leo, eu convallis massa. Cras eu cursus ligula. Nunc non purus et sem vehicula viverra ut nec nibh.

Quisque posuere purus quis eros auctor efficitur. Etiam mattis vitae nulla et blandit. Nulla a orci magna. Cras ac elit enim. Vestibulum nec nisl metus. Mauris congue velit nec malesuada scelerisque. Sed dignissim, enim vitae semper fermentum, mauris leo vestibulum nisl, in malesuada nibh felis nec dui.

Nullam sit amet tellus eget mi varius commodo. Vestibulum sit amet egestas odio. Nam in ullamcorper quam, nec efficitur augue. Curabitur eget elit in leo eleifend tempor vel lobortis lorem. Duis neque dui, tempus eu sollicitudin ac, lobortis sit amet odio. Morbi eleifend, tellus a varius consequat, enim erat sagittis justo, ac rutrum ipsum augue in leo. Suspendisse non mi ante.

Praesent sed pretium dui, id volutpat tortor. Suspendisse tortor lorem,

vestibulum vitae ullamcorper vitae, tincidunt nec leo. Proin interdum congue blandit. Ut bibendum sagittis leo, nec venenatis urna mollis id. Donec nec erat non justo maximus venenatis. In mollis elit eu odio maximus porta. Vestibulum varius turpis sit amet orci blandit, vitae volutpat erat viverra.

Suspendisse nunc urna, elementum ut purus a, sagittis porta velit. Integer ultricies convallis tortor id pellentesque. Duis et sem a mi bibendum congue. Morbi ut tellus cursus, laoreet ipsum rutrum, condimentum felis. Proin velit mi, ultricies a urna nec, facilisis pretium mi. Pellentesque tristique interdum purus, a facilisis mi tempor quis. Sed finibus venenatis ligula porttitor porttitor. Suspendisse cursus lorem nec velit commodo fringilla.

Bibliografia

[1] Latex.

Ringraziamenti

Qui possiamo ringraziare il mondo intero!!!!!!!!!!
Ovviamente solo se uno vuole, non è obbligatorio.