Raw Afternotes - Deep Learning for NLP

Ca' Foscari University of Venice
Department of Environmental Sciences, Informatics and Statistics

DEEP LEARNING FOR NATURAL LANGUAGE PROCESSING [CM0624]
Academic Year 2023 - 2024

Student Zuliani Riccardo 875532
Notes These notes do not provide any images, they were made to have a logical ladder

Contents

1 Langauge Model **1**

2 NL Model **3**
2.1 Matrix Factorization Models 5

3 RNN **8**
3.1 Back Propagation Through Time 8
3.2 RNN and long term dependencies 8
3.3 Long - Short Term Memory Networks 9
3.4 Gated Recurrent Unit 9
3.5 Bidirectional RNN and LSTM 10
3.5.1 BI-RNN 10
3.5.2 BI-LSTM 10
3.6 Sequence 2 Sequence 10

4 Self-Supervised Learning **12**

5 Attention Mechanism **13**
5.1 Neural Turing Machine 13
5.2 Additive Attention 15
5.2.1 Self-Attention 16
5.2.2 Attention Mechanism 17

6 Transformers **18**
6.1 Encoder Part 18
6.2 Decoder Part 19
6.3 Transformer Drawbacks 19

7 Contextualized Word Embedding **20**
7.1 ELMo 20
7.2 BERT 20
7.3 GPT 21
7.3.1 GPT-1 21
7.3.2 Tokenization 22
7.3.3 Byte Paring Encoding 22
7.3.4 Word Piece Encoding 22
7.3.5 GPT-2 22
7.3.6 GPT-3 23
7.3.7 RoBERTa 23

7.3.8 BART . . . 23
7.3.9 T5 . . . 24
7.3.10 GPT-3.5 . . . 24
7.3.11 Istruction-GPT . . . 25

8 Graph and NLP **26**
8.1 Theoretical Foundations of GNN . . . 26
8.2 Convolutional GNN . . . 27
8.3 Graph Attention Networks . . . 28
8.4 Message Passing Neural Network . . . 28
8.5 Graph (NN) in NLP . . . 29
8.6 Knowledge Graphs in NLP . . . 29
8.7 Knowledge Graphs (NN) . . . 30

9 Vision and Language **31**
9.1 ViT . . . 31
9.2 CLIP . . . 32
9.3 Image Captioning . . . 32
9.3.1 Image Captioning using Spartial Features . . . 32
9.3.2 Image Captioning using Transformers . . . 33
9.4 Diffusion Models . . . 33
9.4.1 DDPM: Denoising Diffusion Probabilistic Models . . . 33

10 NLP Tasks **34**
10.1 Part of speech Tagging . . . 34
10.2 Name Entity Recognition . . . 35
10.3 Co-reference Resolution . . . 35
10.3.1 Detect the Mention . . . 35
10.3.2 Clustering Based Approach . . . 36

Chapter 1

Langauge Model

- **Probabilistic Language Modeling**
  - **Goal**: compute the probability of a sentence or sequence of words
  - **Related task**: probability of upcoming words
  - **A model**: that compute either of these $p(w)$ or $p(w_n|w_1, w_2, ..., w_{n-1})$ is a LM or language model
  - How to compute $p(w)$? $\rightarrow$ chin rule of probability
  - How to estimates these probabilities?
    - count / count minus the specific word
    - *Any issues?*
      - Word that cannot appear
      - Very expensive
      - Infinite number of prefix so a very sparse prob
- Unigram Model
- Text generation with a LM
  - Given the prefix how to pick the next given word? Randomly, sampling word from the distribution, pick word with higher conditional probability
  - Once I choose the word there is no way of making an correction
- N-grams model
  - We can extend to trigram, 4-grams, 5-grams
  - insufficient model of language since it has long dependencies
- 2-gram model
  -
$$p(w_i, w_{i-1}) = \frac{count(w_{i-1}, w_i)}{count(w_{i-1})}$$
- Estimating probabilities
  - Instead of doing multiplication we can use a summation by the logarithm

- **How good is our LM**
  - Consider the perplexity: the best ML is the one that best predict an unseen test set
  - Perplexity is the inverse probability of the test set, normalized by the number of words
  - Minimizing the perplexitu is the same as maximising the probability
  - Lower perplexity = better model
  - Bigram with zero probability: 0 prob to the test set so we cant compute the perplexity
  - How we deal with n grams of zero probabilities?
    - Add one estimation, called laplace smoothing, consist on pretend that we saw each word one more time that we did (add one to the count)
    - One hot encoding, but dot product is zero, vectors are orthogonal

Chapter 2

NL Model

- Represent words with low dimensional vectors called embeddings
- Given the embedding through a NN, softmax layer convert a vector into a probability distribution over the entire vocabulary
- **Fixed windows approach**
  - words OHE
  - concatenated word embeddings
  - hidden layer
  - output soft max
    - Advantages:
      - No sparsity problem
      - Model is size O(n)
    - Disadvantages:
      - Fixed window is too small
      - Enlarging window enlarged W
      - Location of word matters we wold like something more general
- Relations
  - Synonyms
  - Similarity
  - Relatedness
  - Anatomy
  - Connotation
- **Word meaning**
  - Defining a meaning by linguistic distribution
  - Each word is a vector called embedding
  - Vectors since
    - With a word, a feature is a word identity

    * With **embeddings**, feature is a word vector, thus we can generalize similar but unseen words

- **OHE:**
  - Sentence are represented as a 0/1 vector of dimension the the length of the vocabulary
  - Similarity via dot product not work since vector are orthogonal
  - RMSE works fine
- Distributional encoding (bag of words): words that occur in similar context has similar meaning
- TF-IDF
- **Similarity measures**:
  - Dot product: tends to be higher when the two vectors have large values in the same dimension, so we normalize it by the length
  - Cosine similarity
- **CBOW**: given a window of words of length $2m+1$ define a probabilistic model for predicting the middle word

  $$P(x_0|x_{-m}, ..., x_{-1}, x_1, ..., x_m)$$

  Train the model minimizing the loss

  $$L = -\sum \log P(x_{-m}, ..., x_{-1}, x_1, ..., x_m)$$

  - Map all the context words into the n dimensional space
  - Average these vectors to get a context vector
  - Get the score vector from the output
  - Use the score to compute probability via softmax
- **Skip-Gram:** given a window of words of length $2m + 1$ define a probabilistic model for predicting each context word

  $$P(x_{context}|x_0)$$

  - Map the center words into the n-dimensional space
  - For the i-th compute the score for a word occupying that position
  - Normalize the score for each position to get a probability
    * Problem on the loss function since the sum requires us to iterate over the entire vocabulary for each example.
    * The complexity comes from the output layer where we apply the softmax function to get the probability tat each vocabulary word appear in the neighbour of the input word

    - Convert the problem into a binary classification problem saying if two words are neighbour or not
    - To train this we have also to sample some negative example, that's why it is called negative sampling
    - **Negative Sampling**
        - Predicting neighbouring word
        - Each training sample to update only a small percentage of weights in the word embedding matrix
    - **Subsampling frequent word:** to decrease the number of training examples, clean the dataset

2.1 Matrix Factorization Models

- Co-occurrences can be computed in two way:
  - **Full document** co-occurrences
  - **Windows-based** co-occurrences, capture both semantic and syntactic information
- **Problem** with co-occurrences matrix
  - Increase in size with vocabulary
  - Very high dimensional
  - Models are less robust
- Idea: store the important info in a fixed small number of dimension: dense vector
- **Singular Value Decomposition**
  - Approximate the full matrix by only considering the leftmost k terms in the diagonal matrix (the k largest singular values)
  - Explain matrix computation
- **Latent Semantic Analysis**
  - Assume that words that are close in meaning will occur in similar pieces of text
  - U-matrix maps words to k-concepts
  - V-matrix maps k-concepts to documents
- **Point Wise Mutual Information**
  - How to represent co-occurrence of words and contexts
  - Binary feature or frequency count
  - Sometimes low co-occurrences are very informative and high co-occurrences are not

  - We need to identify when co-occurrences are higher than we would expect by chance
  - **Point wise mutual information**
  - Problem
  - Positive point wise mutual information

- **Brown Cluster**
  - Input: large corpus of words
  - Otput 1: a partition of words into words clusters
  - Output 2 (general): a hiererchical word clustering
  - Intuition: similar word appear in similar context. Similar word have similar distribution of words to their immediate left and right
  - Formualtion of $Quality(C)$
  - **First Algorithm**
    * Start with $|V|$ clusters
    * We want to get k final clusters
    * We run $|V| - k$ merge steps
      · Pick two clusters c_i and c_j and merge them into a single cluster
      · Continue piking two clusters st the Quality(C) after the merge step is maximized
  - **Second Algorithm**
    * Take the top m most frequent words, put each into its own cluster $c_1, c_2, ..., c_m$
    * For $i = (m + 1)...|V|$
      · Create a cluster c_{m+1} for the i'-th most frequent word. Now we have $m + 1$ clusters
      · Choose two clusters from $c_1, ..., c_{m+1}$ to be merged, pick the merge that gives a maximum value for Quality(C). Now we are back to m clusters

- **Word2Vec SkipGram Negative Sampling - PPMI Matrix Formulation**
  - Skipgram negative sampling is implicitly factorizing a specific matrix of this kind
  - Two points to note:
    1. The entries in the matrix are a shifted positively pointwise mutual information between the word and its context word
    2. The matrix factorization method is not truncated SVD, it minimize the objective function to compute the factorized matricies

- **Global Vector (GloVE)**
  - Pick the best of cont-based and prediction-based models

  - It construct an explicit word-context matrix
  - The optimization objective is weighted least-squares loss assigning more weight to the correct reconstruction of frequent items
  - When using the same word and context vocabularies, the GloVE model represent each word as the sum of it corresponding word and context embedding vectors.
  - Given two words i and j that occour in text, their co-occurrence probability is defined as the probability of seeing word i in the context of word j
  - Formal notation

- **GloVe VS SkipGram**
  - Skipgram: capture co-occurrence one window at a time
  - GloVE: captures the count of the overall statistics of how often words appear.
    * Appropiate scaling and objective gives count-based models the proprieties and performnce of predict-based models

- **FastText**
  - **Limitation of word2vec**
    * Out of vocabulary words: an embedding is created for each word, hence it can't handle words outside its training set.
    * Morphology: like "eat" and "eaten" are different
  - Each word w is represented as a bag of character n-Gram
  - Add <> to indicate the beginning and the end of words
  - Include the word w itself in the set of n-Gram
  - *Heuristics*:
    * N-Gram between 3 and 6 characters
    * Short n-Gram(n = 4): capture syntactic info
    * Long n-Gram(n = 6): capture semantic info
  - **Loss discussion**
  - Sharing representation across words allowing to learn reliable representation of rare words

Chapter 3

RNN

- $A_t = \theta_c A_{t-1} + \theta_x x_t \quad h_t = \theta_n A_t$
- Type of RNN

3.1 Back Propagation Through Time

- Parameters are shared and the derivative are accumulated
- The weights of the unrolled RNN are the same

3.2 RNN and long term dependencies

- The closer I move to the end of the sequence the more I have to add and multiply the weights. During training the gradient could vanish or explode, I will not learn long term dependencies
- If weights are large: exploading gradient
- If the weights are small: vanishing gradient
- **Fixing Exploading Gradients**
  - **Gradient Clipping**
  - **Truncated BPTT**
- **Fixing Vanishing Gradient: adding non-linearity:** *thanh*, like doing an additional power iteration
- RNN exmple: sentiment analysis
- RNN as LM

3.3 Long - Short Term Memory Networks

- LSTM were created with the following goals:
  - Long-term dependencies
  - Incorporation of feedback connections
  - Preserving valuable info from prior data
  - Proficiency in handling data sequences
- The output of an LSTM depends on
  - Current long-term memory, cell state
  - The previous hidden state, short term memory
  - Input data
- Three gates that regulates the amount of long and short term memories allowed
  - **Forget Gate**: how much of the long-term memory will be remember
    - How much we will preserve from the long-term memory
    - Weight removal of part of the long-term memory
  - **Input Gate**: how we update the long-term memory
    - % memory to remember
    - Potential memory to remember based on short term and input data
    - How much of the new memory will be added to the long-term memory
  - **Output Gate**: update the short-term memory
    - Potential long-term memory to become sort
    - % of short to remember
    - new short memory
- Symbols meaning:
  - $tanh \rightarrow$ potential amount of memory to retain
  - $\sigma \rightarrow$ percentage of memory to remember/become

3.4 Gated Recurrent Unit

- Combines forget and input gates, into update gate
- Merges the cell state and the hidden state
- Has fewer parameters than LSTM
- No need cell layer
- Calculations within each iteration ensure that h_t values being passed along, either retain a high amount of old information or are jump-started with a high amount of new information
- **GRU VS LSTM**
  - GRU significantly fewer parameters and train faster

3.5 Bidirectional RNN and LSTM

3.5.1 BI-RNN

- Combine two RNN:
  - First move forward through time from the start of the sequence
  - Second move backword through time from the end of the sequence
- Two RNN stacked on top of each other
- The one that process the input in its original order and the one that process reversed input sequence
- Output computed based on the hidden state of both RNNs
- The weight of the forward are different from the backword RNN

3.5.2 BI-LSTM

- Inputs flows in both directions, capable of using both sides, a side for LSTM
- **Output** is combined in several ways: *average, sum, concatenation, multiplication*
- Every component has information from both past and present
- Bi-LSTM can produce more meaningful output
- Much slower model
- Reqire more time for training

3.6 Sequence 2 Sequence

- Used to convert one sequence into an other
  - Sequences might have different length
  - Based on encoder-decorder architecture
  - Combines two RNNs
- **Given an input/source sentence**
  - Start and end token of the sentence
  - Each word to the embedding layer and then into the encoder
- **At each time step:**
  - Input o the encoder is the previous hidden state and the embedding of the current word
  - Encoder output a hidden state which is the vector representation of the sentence so far

  - Once the final word x_t has been passed into the RNN, we use the final hidden state ans context vector

- **Now turn of the encoder**: append the $< SOS >$ token to the start target sequence
- **At each time step:**
  - Input: embedding of the current token and the previous hidden state
  - Initial decoder hidden state is the context vector
- **Points worth nothing:**
  - Input/source encoder embedding and output/target decoder embedding are two different embedding layer with different parameters
  - Linear layer to convert decoder's hidden state into one of the possible words by softmax
  - The predicted word is used as input for the next loop
- **Training:** once we have our predictor target sequence we compare with the actual target, compute the loss, back-propagation
- **How to select the right word during training**
  - Gready search: highest softmax probability
  - Exhaustive search: generate all possible sequence
  - Teacher forcing: Use ground truth as input instead of the model output from a prior time stamp
  - Beam Search: on each step of decoder keep track of the k-most probable partial translations

Chapter 4

Self-Supervised Learning

- Acquiring large scale dataset with curated humman-annotations is hard and expensive
- The set of categories to learn need to be prefedined
- **Self-Supervised Learning (SSL)**: a smart way of doing supervised learning, the data and not the human provides annotations
- We am to learn good data representation
- Need an automated way to obtain a training signal from the data itself, without human supervision
- One solution is self-prediction: try to predict the properties of the data or hide part of the data and try to reconstruct it
- Pretext and downstream tasks
  - Define a pretext task and train the network for that
  - These features are used for a different downstream task where labels are available
- Self-Supervised Learning in NLP:
  - Center word prediction
  - Neighbour word prediction
  - Neighbour sentence prediction
  - Auto-regressive modelling
  - Masked language modelling
  - Next sentence prediction
  - Sentence order prediction
  - Sentence permutation
  - Document rotation

Chapter 5

Attention Mechanism

5.1 Neural Turing Machine

- Goal: taking input and output and learn algorithms that map from one to the other
- We feed random input and the corresponding expected outputs from the algorithms we intend to learn
- NTM learns to read and write data from the external memory at different time steps to some a given task
- **Controller:** responsible for making the interface between the:
  - Input sequence
  - Output representation
  - Memory through read and write heads
- **Memory**; contains vectors
- NTM lears
  - When to write to memory
  - Which memory cell to read from
  - How to convert result of read into final output
  - When to stop writing: avoiding drifting
- **In a NTM:**
  - Controller: LSTM or RNN
  - The hidden state to the read heads
  - Read head retrieve the information from the memory
  - The retrieve data is passed to the controller again
  - The controller send the hidden state to the write head that decide what to store
- **Direct access is not differentiable, so we use attention**

  - I'm not directly accessing a memory locations
  - I'm accessing all memory locations with a certain prob dist that represent the attention scores
  - Multiply the vector of probability distribution times the matrix that contain all the records

- **Blurry Operations**: interact to a greater or lesser degree with all the elements in memory rather than addressing a single one or few elements directly.
  - Degree determined by attention focus
    * Constraints each read and write operation to interact with a small portion of the memory while ignoring the rest
    * The proportion of memory taken into attentional focus is determined by normalized weights emitted by the heads

- **Memory Addressing**
  - *Content Based - Addressing*
    * Cosine similarity between the embeddings form the lstm and the embedding in the memory
    * Softmax between the temperature of a cosine similarity and the other
  - *Local Based - Addressing*
    * Interpolation
      · The new weight account from both the content based addressing and the focus in the last time stamp
    * Convolutional Shift
      · It smoothly shift the weights left to right based on a parameter
      · This allow NTM to perform basic algorithm like copy and sort
      · Very close to the head shifting in a classical turing machine
    * Sharpening
      · If we index N memory locations from 0 to N-1, the rotation applied to the weights by the parameter can be expressed as a circular convolution where all index arithmetic are computed modulo N
      · It blurs out data, how to fix, sharpening, highlight the most relevant

- **Read operations**: is a weighted sum, in the most desiderable situation, the weights are OHE vector, 0s and one 1

- **Write Operation**:
  - Combination of erase and add operations
  - The process is composed with the previous state and new input, Implements a similar mechanism as the forget and input gates in LSTM

- **Controller unrolled**

  - Once the head has updated its weight vector it is ready to operate on the memory depending on the head type:
    * Read head: it outputs a weighted combination on the memory locations
    * If it is a write head the content of the memory is modified according to weights with an erase and an add vector

5.2 Additive Attention

- **Bottleneck Problem**
  - NN needs to be able to compress all the necessarily information of a source sentence into a fixed length vector
  - Difficult to cope with long sentences especially those that are longer than the sentences in the training corpus
  - Attention mechanism helps to look at all hidden states from encoder sequence for making predictions
  - How we decide which states are more or less useful for every prediction at every time step of the decoder
  - Use a NN to learn which hidden encoder states to attend to and by how much
- **Alignment Computation**
  - How important is the embedding h_j compared to what I have in the hidden state of the decoder at the previous state
  - a is a FFNN
- **Attention Component**
  - Score converted into a prob dist via softmax
  - a_{ij} importance of annotation h_j wrt the previous hidden state s_{i-1} in deciding the next state s_i and generating y_i
- **Context vector**
  - Depends on a sequence of annotations to which the decoder map an input sequence
  - Each annotations contains informations about the whole input sequence with a strong focus on the parts surrounding the i-th word of the input sequence
- **Attention is great indeed it**
  - Significantly improves NTM performance
  - Solves the bottleneck problem
  - Helps with vanishing gradient problem

  - Provide some interpretability

- **General DL Framework**
  - Given a set of vector values and a vector query, attention is a technique to compute a weighted sum of the values dependent on the query

- **Intuition**
  - Weighted sum is a selective summary of the information contained in the values, where the query determines which values to focus on
  - Is a way to obtain a fixed size representation of an arbitrary set of representations

- **RNN Problems**
  - Hard to learn long distance dependencies
  - Linear order of words
  - Embedding combined with all the previous one and at a certain point we completely loose information of the initial input sequence
  - Forward and backword passes have un-parallelizable operations
    * Future RNN hidden states can't be computed in full before past hidden states have been computed

5.2.1 Self-Attention

- Attention treats each word's embedding representation as a query to access and incorporate information from a set of values

- **Attention as a soft-lookup method**
  - Attention as performing fuzzy lookup in a key value store
  - In a look-up table we have a table of keys that map to values. The query matches one of the keys returning its value
  - In attention the query matches all keys softly, to a weighted between 0 and 1. The keys values are multiplied by the weights and summed

- **Self-Attention Mechanism on the same sequence**
  - Attention operates on queries, keys and values
  - In self-attention the queries, keys and values are drawn from the same source
  - Formulas

5.2.2 Attention Mechanism

- Self attention, can we use it as our main building block? Can it be a drop-in replacement for recurrence?
- NO! it has few issues
  1. **Doesn't have inherit notion of order**
     - **Solution**: add positional representation to the inputs
     - We need to encode the order of the sentence in our keys, queries and values
     - **Sinusoidal Representation:** concatenate sinusoidal functions of varying periods
       - Why:
         - Periodicity
         - Constrained values
         - Extrapolation for long sequences
       - Pros: periodicity and extrapolation
       - Cons: not learnable, extrapolation doesn't really work
     - **Learned absolute position representation**
       - Let all p_i be learnable parameters
       - Pro: flexibility: each position gets to be learned to fit the data
       - Con: definitely can't extrapolate to indices outside $1, ..., T$
  2. **No non-linearities for DL. It's all just weighted average**
     - **Solution**: easy fix: apply the same FFNN to each self-attention output
     - So instead of passing directly the soft-attention result to the next step we send it in a FFNN
  3. **Need to ensure we don't "look at the future" when predicting a sequence**
     - **Solution**: Masking
       - Parallelize operations while not looking at the future
       - Keeps info about the future from *"leaking"* to the past
     - Self attention in decoders, we need to ensure we can't peek at the future
     - At every timestamp we could change the set of keys and queries to include only past words
     - Enable parllelization to future words by setting attention scores to $-\infty$

Chapter 6

Transformers

It is an **encoder - decoder** model with attention and without recursion.

6.1 Encoder Part

- **Input Embedding**: convert a token into vector
- **Positional Encoding**: add location of the word within the sentence via sinusoidal representation
- **Multi-head Attention**
  - Self-attention, keys, queries and vales comes from the same source
  - Query - key dot product in one matrix multiplication $XQ(XK)^T \in \mathbb{R}^{n \times n}$
  - Softmax and compute the weighted average with an other matrix multiplication
    $$softmax(XQ(XK)^T) \cdot XV \in \mathbb{R}^{n \times d}$$
  - What if we want to look in multiple places in the sentence at once
    * We define multiple attention heads through multiple Q,K,V matrcies, h for example
    * Each attention head perform attention independently
    * Then outputs of all heads are combined
    * Each head gets to "look" at different things, and construct value vectors differently
    * It's not really more costly, explain
- **Add & Norm**: scaled product attention is a final variation to aid transformer training
  - When dimensionality becomes large, dot products between vectors tend to become large
  - We define the attention scores by $\sqrt{\frac{d}{h}}$ to stop the scores becoming large just as a function of $\frac{d}{h}$
- **Encoder Tricks**

  - **Residual Connections**
    - Help the model to train better
    - Though to make the loss landscape considerably smoother
  - **Layer Normalization**
    - Help the model to train faster
    - Cut down uninformative variations in hidden vectors values by normalizing to unit mean and sd within each layer
    - Z score (how many sd our point is far from the mean) $\times$ gain + bias

6.2 Decoder Part

- **Cross-Attention**
  - Keys and values are drawn from the encoder
  - Queries are drawn from the decoder
  - H vector of concatenation of encoder output
  - Z vector of concatenation of input decoder
  - Query and Z times Keys and H: $ZQK^TH^T \in \mathbb{R}^{T\times T}$
  - Softmax and compute the weighted average with H and Values

$$softmax(ZQK^TH^T) \cdot HV \in \mathbb{R}^{T\times d}$$

- **Masked Self-Attention**
  - We have h attention mask for h heads

6.3 Transformer Drawbacks

- **Positional Representation**
- **Quadratic Compute in Self-Attention**
  - Highly parallelizable
  - However the total # operations grows as $O(T^2d)$, where T is the sequence length and d is the dictionary

Chapter 7

Contextualized Word Embedding

7.1 ELMo

Replace static Embeddings with context-dependent embeddings

- Each token's representation is a function of the entire input sequence computed by a deep bidirectional LM
- Return for each token a linear combination of its representation across layers
- Different layer capture different information
- Each layer of this language model computes a vector representation for each token
- For each task: train task dependent softmax weights to combine the layer wise representation into a single vector for each token jointly with a task specific mode that uses those vectors
- *Forward LM:* deep LSTM that goes form the start to the end to predict the token based on the prefix
- *BackwordLM:* deep LSTM that goes form the end to the start to predict the token based on the suffix
- Train these LMs jointly with the same parameters for the token representation ad the softmaz layer
- How to use it
  - Embedding into choosen architecture
  - Frozen Embeddings
  - Fine Fine-Tuning, bee careful for the catastrophic forgetting

7.2 BERT

Bidirectional Encoder Representation from Transformer

- Fed the transformer encoder with a sequence

- Get the embeddings for each words
- **Goal** Understand laguage
- 12 or 24 encoders layer (Small or Large)
- Pre-Training phase:
  - Pretrain to understand the language
    - masked Language Models (MLM)
    - Next Sentence Predictions (NSP)
    - Both tasks are jointly trained
  - Finetune on specific tasks
    - Softmax on final layer of [CLS] token
    - Softmax to the tokens in the sequences

    CLS question [SEP] answer passage [SEP], lean to predict start and end token on answer token
- Explain training-eval phase
- Mask out k% of the input words and then predict the masked words, k = 15 usually
  - Too little masking: too expensive to train
  - Too much masking: not enought context
- Problem: mask token never seen at fine-tuning
  - 15 % of words to predict, we replace them with
  - 80 % with [mask]
  - 10 % random word
  - 10 % keep the same

7.3 GPT

GPT: Generative Pre-Trained Transformer

- Feed the transformer decoder with a sentence predict the next word
- Only a single Masked Multi-Head Attention
- Language model

7.3.1 GPT-1

- Auto-Regressive 12 layer transformer decoder
- Pre-trained on raw text as language model
- Fine-tuned on labeled data: classification, entailment, similarity, multiple choice

7.3.2 Tokenization

- What happens when we encounter a word at test time that we're never seen in our training data?
  - in Word level tokenization no way of assigning index, no word emebdding, can't process that input sequence
- Solution: low-frequency word in training with a special $< UNK >$ token, used to handle unseen words
- Problem: unseen word are all the same
- Word-level tokenization treats different forms of the same word
  - Open, opened, opens
  - Separate types and embeddings
  - This can be problematic when training over smaller dataset
  - Re-learn the meaning of words

7.3.3 Byte Paring Encoding

- From base vocabulary
- Count up frequency of each character pair in the data, choose the most occured
- Select it and merge the characters together into one symbol, Add this new symbol to the vocab then retokenize the data
- Keep repeating this process, untill we stop

7.3.4 Word Piece Encoding

- Good balance between flexibility of single characters and the efficiency of full words for decoding, sidestep the needs of special treatment of unknown words
- Pretrained model performs the word segmentation
- Special encoding for out-of-vocabulary words, explain

7.3.5 GPT-2

- Larger, more parameters, more vocabularies
- **Emergent zero-shot learning**
  - Ability to do many tasks with no examples and no gradient updates
  - Specifying the right sequence prediction problem
  - Comparing probabilities of sequences
  - We can get interesting zero shot learning behaviour if we are creative enough with how we specify our task

7.3.6 GPT-3

- Specify a task by simply prepending example of the task before our example
- Called also in-context learning, to stress that no gradients are performed when learning a new task
- What are the limits of promting
  - Some tasks seems too hard even for LMs to learn through promting alone, like math operations
  - Solution: **Chain-of-Though Promting**: helps the system explaining to get the answer

7.3.7 RoBERTa

Same model as BERT but better tuned

- Better HP
- Larger model
- Big bathces with more data
- Removing NSP
- Trained on longer sequence
- **Static Masking (BERT)**
  - Precompute the mask before training
  - Masks are constant at each iteration
- **VS Dynamic Masking (RoBERTa)**
  - Change mask on the fly
  - Change it at every training iteration, results in a more data and generalization capabilities

7.3.8 BART

- BERT: encoder, good for analysis tasks, trained with MLM
- GPT: decoder, good for LM
- **BART**: both encoder and decoder
- **Pre-Training Tasks**
  - Token masking, deletion, filling
  - Sentence permutation
  - Text infilling
  - Document rotation

7.3.9 T5

- Treat every NLP tasks as a "txt-to-text" problem, same model, same HP, same loss function
- How: adding a task specific prefix to the input sequence, and pretrain the model to get task specific output
- T5 uses an encoder decoder arch. with the differences:
  - Layernorm applied before each attention and FF transformation
  - No additive baias is used for layer norm
  - A simpler positional embeddings is used, it uses a scalar to the corresponding logit used to compute attention weights
  - Dropout is applied through the network

7.3.10 GPT-3.5

- LM $\neq$ Assisting users
- Pre-training can improve NLP app by serving as parameter initialisation
- From LM to assistant
  - **Instruction Fine-tuning**: collect examples of pair across many tasks and fine-tune a LM, but has a lot of limitations:
    - Expensive to collect ground truth
    - P1: task like open-encoded creative generation has no right answer
    - P2: LM penalize all token-level mistakes equally but some errors are worse than other
    - Miss-match between LM objective and human preferences
  - **Reinforcement Learning for Human Feedback (RLHF)**
    - Training a LM on some task
    - Obtain human reward, higher is better
    - We want to maximise the reward of samples from our LM
    - Gradient ascend
    - Reward is not differentiable
    - Policy gradient, tool for estimating and optimizing this objective
    - Log derivate trick
    - Plug all back
    - Put the gradient inside the expectation, we can approximate this objective with montecarlo samples, performing m-samples and then compute the average
    - I have the gradient of my model's parameter wrt the samples, I can get an update rule so gradient Ascent
      - If R is ++: gradient step to maximise the sample prob
      - If R is –: gradient step to minimise the sample prob

  - Now for any arbitrary non differentiable reward function R(s) we can train our LM to maximise the expected reward
  - *Problem 1*: human in the loop is expensive, solution model their preferences as a separate NLP problem
  - *Problem 2*: human judgements are noisy and misscalibrated, solution ask for pairwise comparison

7.3.11 Istruction-GPT

- Collect demonstration data and train a supervised policy
- Collect comparison data and train a reward model
- Optimize a policy against the reward model using reinforcement learning (RLHF)

Chapter 8

Graph and NLP

8.1 Theoretical Foundations of GNN

- Convolutional neural networks respect translational invariance
- Patterns are interesting irrespective of where they are in the image
- **Locality**: neighbouring pixels relate much more strongly than distant ones
- What about arbitrary graphs?
  - The nodes of a graph are not assumed to be in any order
  - We would like to get the same results for two isomorphic graphs
  - **Permutation invariance and equivariance**
- Assuming the graph has no edges
- $x_i \in \mathbb{R}$ be the feature of node i

$$X = (x_1, ..., x_n)^T$$

- We have specified a node ordering!
- We would like the result of any neural networks to not depend on this
- It will be useful to think about the operations that change the node order, like **Permutations**
- Each permutation defines an $n \times n$ matrix, row with one 1 and all zeros, col i (which is referring to the row) goes to the position in wich is present the 1
- **Permutation Invariance**
  - Design functions $f(X)$ over sets that will not depend on the order
  - Thus applying a permutation matrix shouldn't modify the result!
  - $f(X)$ is permutation invariant if, for all permutation matrices P st:

$$f(PX) = f(x)$$

  - Generic form description (see notes)

- **Permutation Equivariance**
  - We want to still be able to identify node outputs, which a permutation-invariant aggregator would destroy
  - Seek functions that don't change the node order
  - $f(X)$ is permutation equivariant if, for all permutation matrices P

$$f(PX) = Pf(X)$$

  - Each node's row is unchanged by f
  - Equivariant set functions as transforming each node input x_i into a latent vector h

- We can represent these edges with an adjacency matrix, A
- Node permutations now also accordingly act on the edges
- We need to appropriately permute both rows and columns of A, When applying a permutation matrix P, this amounts to PAP^T

$$\text{Invariance} \quad f(PX) = f(X) \quad f(PX, PAP^T) = f(X, A)$$

$$\text{Equivariance} \quad f(PX) = Pf(X) \quad f(PX, PAP^T) = Pf(X, A)$$

- On sets, we enforced equivariance by applying functions to every node in isolation
- **Node's neighbourhood**
  - (1-hop) neighbourhood $N_i = \{j : (i, j) \in \epsilon \wedge (j, i) \in \epsilon\}$
  - $X_{N_i} = \{x_j :\in N_i\}$
- **Permutation Equivariance Fucntions**

8.2 Convolutional GNN

- The goal is to learn a function of signals/features on a graph $G = (V, E)$ which takes as input:
  - A feature description x_i for every node i summarized in a $N \times D$ feature matrix X
  - A representative description of the graph structure in matrix form, adjacency matrix A
  - And produces a node-level output Z
- **Limitations**
  - Initial node features are not considered

    - Add self-loops in A!
  - The matrix A is not normalized, hence using it in the multiplication will change the scale
    - Use the normalized A!
  - **These two fixes will lead to the GCN formulation**

8.3 Graph Attention Networks

- Features of neighbours aggregated with implicit weights (via attention)
- Useful as "middle ground" w.r.t. capacity and scale
- Attention coefficients a_{ij} implicitly defined via self-attention over node features
- Alpha is the attention mechanism
- Alpha's parameters trained jointly with the remaining architecture
- Works in both inductive and transductive settings
- Attention Score
- Multi-Head Attention

8.4 Message Passing Neural Network

- Compute arbitrary vectors ("messages") to be sent across edges
- Two Phases
  - Message Passing
  - Readout
- **Extending Message Passing Framework**
  - What about edges?
  - Compare different types of inductive biases
  - Propose a general and comprehensive GN formulation
- **Type of Baiases**
  - Fully Connected
  - Convolutional
  - Recurrent
  - Graph Network
- **GNN: a General Formualtion**

8.5 Graph (NN) in NLP

- Represent natural language as a graph
  - Dependency graphs
  - Text graph containing multiple hierarchies of elements
- **Graph Construction from Text**
  - Different NLP tasks require different aspects of text
  - Static vs dynamic construction
  - Goal: good downstream task performance
- **Static Graph Construction**
  - Problem Setting
    - Input: raw text
    - Output: graph
  - Graph structure learned during preprocessing by argumenting text with domain knowledge
  - **Types:**
    - Dependency Graph
    - Constituency Graph
    - Abstract Meaning Representation Graph
    - Information Extraction (IE) Graph
    - Knowledge Graphs Q&A
    - Co-occurrence Graph
    - SQL Graph
    - Dynamic Graph Construction
- **Dynamic Graph Construction**
  - Problem Setting:
    - Input: raw text
    - Output: graph
  - Graph structure (adjacency matrix) learning on the fly, joint with graph representation learning

8.6 Knowledge Graphs in NLP

- **Tasks**
  - Link prediction / triple classification
    - Link prediction
      - Learning to rank problem

            - Information retrevial metrics
            - No ground truth negativities
        - Triple Classification
            - Binary classification task and metrics
            - Test set require positive and ground truth negatives
    - Collective node classification/ link-based clustering
    - Entity matching

- **Transductive Link Prediction**: When both subject and object off the predicted link occur in the training knowledge graph
- **Inductive Link Prediction**: when subject or object of the predicted link do not occur in the training knowledge graph
- **Representation Learning**
    - Learning representation of nodes and edges
    - Our goal is Knowledge Graph Embeddings
        - Automatic, supervised learning of embeddings, projections of entities and relations into a continuous low-dimensional space
        - Scoring Function: f assign a score to a triple (s, p, o), high score = triple is very likely to be factually correct
        - Translation-based Scoring Functions: $f_{TransE} = ||(e_s + r_p) - e_o||_n$
        - Loss function
            - Pairwise Margin-Based Hinge loss
            - Negative Log-Likehood / Cross Entropy

8.7 Knowledge Graphs (NN)

Multi Relational Graphs, see notes.

Chapter 9

Vision and Language

9.1 ViT

ViT: Vision Transformer

- N input patches each shape for example 3x16x16
- Linear projection to D-dimensional vector
- Add positional embedding: learn D-dim vector per position and a special extra input, the classification token like the [CLS] token
- Transformer Encoder, like NLP Transformer of
- Output vectors, last output is the linear projection to C-dim vector o predicted class scores
- **ViT vs CNN**
  - IN most CNNs decrease resolution and increase channels as you go deeper in the network *(hierarchical architecture)*
  - In ViT all blocks have the same resolution and number of channels *(isotopic architecture)*
- **Pros**:
  - Can learn global features of images
  - ViTs are not as sensitive to data argumentation as CNNs, thus they can be trained on smaller dataset
  - ViTs can be used for a variety of image classification tasks
- **Cons**:
  - Computationally expensive, large number of parameters
  - Are not efficients as CNNs at processing images since they need to attend to every part of the image, even if it is not important for the task

9.2 CLIP

CLIP: Contrastive Language-Image Pre-Trained

- Motivation: instead of using a set of labels, get supervision from NL
- Result: robust zero-shot inference, multimodel feature space
- Series of text description → textual embedding
- Series of images → vision emebdding
- Maximise the diagonal similarities, minimize the rest
- Compute the cosine similarities of these embedding scaled by a temperature parameter and normalized into a prob dist via softmax
- Use symmetric cross entropy loss
- **Zero-shot image Classification**
  - Image, encode it into a feature space
  - Pick all the classes and create a text with prefix: "A photo of a {class name}"
  - Convert the text using text encoder
  - Compute cosine similarity
  - Pick the top similar pair of embedding
  - Zero shot image classification

9.3 Image Captioning

- Image into a CNN
- Output used as context vector to my sequence-to-sequence model (RNN)
- Auto-regressive
- I continue until I obtain the token $< END >$

9.3.1 Image Captioning using Spartial Features

- **Problem**: input is "bottleneck" through c, model needs to encode everything it wants to say within c
- **Solution**: Attention idea, new context vector at every time step, each one will attend to different image regions
  - Each time step of **decoder** it uses a different context vector that looks at different parts of the input image
  - This entire process is differentiable
  - Model chooses its own weights. No attention supervision is required

9.3.2 Image Captioning using Transformers

- No recurrence at all
- We don't need convolutions
- Transformers from pixels to language

9.4 Diffusion Models

- **Forward diffusion process**: at each step add some gaussian noise until we destroy the image structure. At each iteration we can decide how much noise we want tot add.
- **Reverse denoising process**: at each step generate a bit cleaner version of the image, gradually reconstruct the image
- Sampling from a gaussian dist with mean and std determined by a NN
  - Gaussian are convinient since we can jump to arbitrary time step in the process

9.4.1 DDPM: Denoising Diffusion Probabilistic Models

- **Challenges**
  - Training and sampling in high-res space consume a lot of resources
  - Images are high-dimensional
- **Remedies**
  - Downsampling images and next up-sampling generations
  - Do diffusion in latent space
    - Compressed space so it has lower resolution
    - It is perceptual compression, model doesn't need to learn imperceptible details
    - It is a projection onto lower dimensional space
  - We can condition the LDMs either via concatenation or by a more general cross-attention mechanism
  - The denoising step is done in the latent space

Chapter 10

NLP Tasks

10.1 Part of speech Tagging

- Input: a sequence of word tokens
- Output: a sequence of part-of-speech tags t
- **Why part of speech tagging**
  - Can be useful for other NLP tasks: parsing, MT, sentiment, text-to-speech
  - Linguistic or language-analytic computational tasks
- **A limited number of tags for word class**
  - Distributed criteria: same context, same syntatic functions
  - Morphological criteria
  - Not about meaning
- **Open-class parts of speech**
  - Nouns
  - Verbs
  - Adjective
  - Adverbs
  - Preposition
  - Determiners
  - Pronouns
- **Why tagging is hard**
  - If every word by spelling was a candidate for just one tag, positional tagging would be trivial
  - This won't always work
    - Roughly 15% of word types are ambiguous, but those 15% tend to be very common

        * So around the 60% of word tokens are ambiguous

- Positional tagging algorithm
  - Hidden markov model
  - RNNs, transformers
  - LLM

10.2 Name Entity Recognition

- Anything that can be referred to with a proper name, most common 4 tags:
  1. PER: person
  2. LOC: location
  3. ORG: organization
  4. GPE: geo-political entity
- Tings that helps us to understand the context of the text
- Find spans of text that constitute proper names, tag the type of the entity
- **Why NER is hard?**
  - Segmentation, we have to find and segment entities
  - Type Ambiguity
- **BIO Tagging**: B → beginning, I → inside, O → outside

10.3 Co-reference Resolution

Identify all mentions that refer to the same entity in the text

10.3.1 Detect the Mention

- Part of the text that can receive mentioning. Pronouns, named entities, noun phrases
- Making all pronouns, named entities and noun phrases as mentions cause over-generation of mentions
  - Could train a classifier to filter out spurious mentions
  - Keep all mentions as "candidate mentions"
  - We can build a model that begins with all spans and jointly does mention detection and co-reference resolution end-to-end in one model
- **Mention Pairs Model**
  - Train a binary classifier that assign every pair of mentions a probability of being co-referent. Positive near 1, negative near 0

- **Mention airs Model at Test Time**
  - Co-reference resolution is a clustering task, only scoring pairs, what do
  - Pick some threshold and add co-reference links between mention pairs where $p(m_i, m_j)$ is above the threshold
  - Take transitive closure to get the clustering, add missing link, but not all, explain why

10.3.2 Clustering Based Approach

- Start with each mention in its own singleton cluster
- Merge a pair of clusters at each step. Use a model to score which cluster merges are good
- Training phase
  1. Produce a vector for each pair of mentions
  2. Poolig operation over matrix of mention pair representation to get cluster-pair representation
  3. Score the candidate cluster, merge by taking the dot product of the representation with a weight vector
- Current candidate cluster depend on the previous one, use reinforcement learning to train the model
- **End-to End Models**
  - Use an LSTM, attention
  - Do mention detection and co-reference end-to-end
  - No mention detection step, instead consider every span of text as candidate mention
  - Embed the words in the document using a word embedding matrix and a character level CNN
  - Run Bi-LSTM over the document
  - Represent each span of text i going from start(i) to end(i) as a vector