NLP_Kaggle

February 24, 2024

1 NLP Cheatsheet: Master NLP

Since Kaggle has been recently awash of NLP competitions, I told myself that it would be a great opportunity to **share my knowledge by posting questions** (more or less advanced) on NLP topics.

It is a great way for you guys to test your knowledge of the field as well as to **understand better the underlying algorithms** of the famous frameworks that we use on a daily basis to model a problem.

This notebook is mostly a repertory of questions that can shape your learning of the field.

As a tip, I would recommend learning the structure (to have a structrued view of NLP) using this notebook and to look for details when needed using this notebook.

I will continually update this notebook regularly with questions that you can include in your favorite **note-taking** or **spaced-repetition** software (Anki is a must for this matter).

If you like this work, don't forget to upvote the notebook as it encourages me to do it for many other fields. In particular, I plan on doing the same for Transformers, geometric deep learning and computer vision in the near future.

If you want to have a more **holistic** approach to NLP, check out my other NLP notebook: https://www.kaggle.com/rftexas/ml-cheatsheet-a-mind-map-for-nlp

Also this notebook mostly summarizes those amazing resources: - https://nlpoverview.com/#d-attention-mechanism - https://medium.com/saarthi-ai/transformers-attention-based-seq2seq-machine-translation-a28940aaa4fe - http://web.stanford.edu/class/cs224n/ (almost all the illustrations are extracted from this amazing course) - https://lilianweng.github.io/lil-log/2020/04/07/the-transformer-family.html - https://lilianweng.github.io/lil-log/2019/01/31/generalized-language-models.html

2 Table of contents

3 1. Learning representations that convey semantic and syntactic information

A central problem in NLP is how to represent words as input to any of our models. We need some notions of similarity and distance between words. Intuitively, we do feel that queen is closer to king than cat for instance.

Thus, we want to encode word tokens into some vectors that represent points in some 'word space'.

Objective: Finding a N-dimensional space (where N is between 100 and 1000) that is sufficient to encode all semantics of our language. In this word space, each dimension would encode some meaning that we transfer using speech (e.g. tense, count, gender…)

- **What are the four main steps of text processing?**

  1) **Loading**: Load text as string into memory
  2) **Cleaning**: Cleaning the text, correct misspellings, remove emojis
  3) **Tokenization**: Split strings into tokens, where a token could be a word or a character
  4) **Vectorization**: Map all the tokens in data into vectors for ease of feeding into models

- **What does bag of words refer to?**

  When we put all the words of a document in a *'bucket'* we call such a bucket a bag of words. Simply put, a bag of words is a set of all the words in a document.

- **What are stop words?**

  Stop words are essentially high-frequency generic words that do not convey context-specific sense. E.g.: 'the', 'of', …

- **What is TF-IDF vectorizer?**

  TF-IDF stands for term frequency - inverse data frequency. TF is the number of times a word appears in a document divided by the total number of words in the document. IDF is the log of the numbers of documents divided by the number of documents that contain the word w. TF-IDF is the product of those two quantities.

- **What is a word embedding?**

  Word embedding is a dense representation of words in the form of numeric vectors.

3.1 1.1. One-hot vector, a naïve approach (denotational semantics)

- **What is one-hot word representation?**

  Every word is represented as a V-dimensional vector (where V is the size of the vocabulary), with all 0s and 1 at the index of that word in the sorted English language.

- **What is denotational semantics?**

  The concept of representing an idea as a symbol. It is sparse and cannot capture similarity.

- **What is the issue of one-hot encoding?**

  No notion of similarity (no cosine similarity for instance).

3.2 1.2. SVD-based methods (distributional semantics)

Objective: Finding word embeddings that captures some notion of similarity.

- **What is distributional semantics?**

  The concept of representing the meaning of a word based on the context in which it usually appears. It is dense and can better capture similarity.

- **How do SVD-based methods work?**

  Loop over a massive dataset and accumulate word co-occurence counts in a matrix X. Then use SVD where the word vectors are the columns of U.

- **What is latent semantic analysis (LSA)?**

  LSA is a technique of analyzing relationships between a set of documents and the terms they contain by producing a set of concepts related to the documents and terms. To do so, we build a matrix whose coefficients are word counts per document and use SVD to reduce the number of rows while preserving the similarity structure among columns. To compare documents, we compute the cosine similarity between the column vectors representing them.

3.2.1 1.2.1. Word-document matrix

- **What is the assumption of a word-document matrix?**

  We make the bold conjecture that words that are related will often appear in the same documents. For instance, *banks*, *bonds* and *stocks* are probably likely to appear together. But *banks*, *octopus* and *hockey* would probably not consistently appear together.

- **How is a word-document matrix built?**

  Loop over billions of documents and for each time word i appears in document j add one to entry Xij.

- **What is the issue of a word-document matrix?**

  A very large matrix that scales with the number of documents.

3.2.2 1.2.2. Window-based co-occurrence matrix

- **What is a window-based co-occurrence matrix?**

  A matrix stores co-occurrences of words thereby becoming an affinity matrix. In this method, we count the number of times each word appears inside a window of a particular size around the word of interest.

- **How to extract word vectors from a word-word co-occurrence matrix?**

  Generate a VxV co-occurrence matrix X. Apply SVD on X. Select the first k columns of U to get a k-dimensional word vectors. The ratio of the variances indicates the amount of variance captured by the first k dimensions.

- **What are the problems faced with such a method?**
  - The dimensions of the matrix change very often
  - The matrix is sparse
  - Very high dimensional
  - Quadratic cost to train

3.3 1.3. Word2Vec

- **What is an iteration-based model?**

A model that is able to learn one iteration at a time and eventually be able to encode the probability of a word given its context.

- **What is Word2Vec?**

  A model whose parameters are the word vectors. Train the model on a certain objective. At every iteration, we run our model, evaluate the errors and backpropagate the gradients in the model.

- **What are the initial embeddings of Word2Vec model?**

  The embedding matrix is initialized randomly using a Normal or uniform distribution. Then, the embedding of word i in the vocabulary is the row i of the embedding matrix.

- **What are the two algorithms used by Word2Vec? Explain how they work.**

  Continuous bag-of-words (CBOW)

  Skip-gram

- **What are the two training methods used?**

  Hierarchical softmax

  Negative sampling

- **What is the advantage of Word2Vec over SVD-based methods?**

  Much faster to compute and capture complex linguistic patterns beyond word similarity

- **What is the limitation of Word2Vec?**

  Fails to make use of global co-occurrence statistics. It only relies on local statistics (words in the neighborhood of word i).

  E.g.: The cat sat on the mat. Word2Vec doesn't capture if *the* is a special word in the context of cat or just a stop word.

3.4 1.4. Global vectors for word representations (GloVe)

- **What is GloVe?**

  GloVe aims to combine the SVD-based approach and the context-based skip-gram model.

- **How to build a co-occurence matrix for GloVe? What can we calculate with such a matrix?**

  Let X be a word-word co-occurence matrix (coefficients are the number of times word i appears in the context of word j). With this matrix, we can compute the probability of word i appearing in the context of word j: $Pij = Xij \ / \ Xi$

- **How is GloVe built?**

  After building the co-occurence matrix, GloVe computes the ratios of co-occurrence probabilities (non-zero). The intuition is that the word meanings are capture by the ratios of co-occurrence probabilities rather than the probabilities themselves. The global vector models the relationship between two words regarding to the third context word as:

$$F(w_i, w_j, \tilde{w}_k) = \frac{p_{\text{co}}(\tilde{w}_k|w_i)}{p_{\text{co}}(\tilde{w}_k|w_j)}$$

F is designed to be a function of the linear difference between two words *wi* and *wj*. It is an exponential function.

- **What are the pros of GloVe?**

  The GloVe model efficiently leverages global statistical information by training only on non-zero elements in a word-word co-occurence matrix, and produces a vector space with meaningful substructure.

- **What is window classification and why is it important?**

  Natural languages tend to use the same word for very different meanings and we typically need to know the context of the word usage to discriminate between meanings.

  E.g.: 'to sanction' means depending on the context 'to permit' or 'to punish'

  A sequence is a central word vector preceded and succeeded by context word vectors. The number of words in the context is also known as the context window size and varies depending on the problem being solved.

- **How do window size relate to performance?**

  Generally, narrower window size lead to better performance in syntactic tests while wider windows lead to better performance in semantic tests.

4 2. How to create language models?

- **What are language models?**

  Language models compute the probability of occurrence of a number of words in particular sequence.

4.1 2.1. N-gram language models

Objective: Compute the probability of occurrence of a number of words in particular sequence looking at the n previous words.

- **What are the 2 limitations of n-gram language models?**
  - **Sparsity problems**: if n-gram never appears in corpus, then probability is 0.
  - **Storage problems**: as n increases or the corpus size increases, the model size increases as well.

Since we only consider the n previous words in the sequence, we cannot take advantage of the full information conveyed in a sequence.

4.2 2.2. Recurrent neural networks (RNN)

Objective: Condition the language model on all previous words in the corpus.

- **How is organized RNN?**

  A RNN is organized in a series of hidden layers holding a number of neurons, each of which performs a linear matrix operation on its inputs followed by a non-linear operation.

  At each time-step, there are two inputs to the hidden layer: the output of the previous layer *h(t-1)*, and the input at that timestep *x(t)*.

$$h_t = \sigma(W^{(hh)}h_{t-1} + W^{(hx)}x_t)$$

$$\hat{y_t} = softmax(W^{(S)}h_t)$$

  After this operation, the output h(t) is multiplied by a weight matrix and run through a softmax over the entire vocabulary, which outputs the probabilities of the next word. The word with the highest probability is picked.

- **How does a RNN solve the curse of dimensionality problem incurred by n-gram language models?**

  It is solved since the weight matrices are applied at every step of the network. Hence the model parameters don't grow proportionally to the input sequence size. The number of parameters is independent of the sequence length.

- **What is the loss function of a RNN?**

  Cross-entropy summed over a corpus of size T and a vocabulary of size V.

- **What is the perplexity of a RNN and what does it mean to have a low perplexity?**

  The perplexity of a RNN is 2 to the negative log probability of the cross entropy loss function.

  Perplexity is a measure of confusion where lower values imply more confidence in predicting the next word in the sequence.

  A perplexity for a language model of 247 means that the model is as confused/perplex as if it has to choose uniformly and independently among 247 possibilities for each word.

- **What are advantages of RNN?**

  - They can process input sequences of any length
  - The model size does not increase for longer input sequence lengths
  - The same weights are applied at every time step of the input, so there is symmetry in how inputs are processed

- **Give an example of the vanishing gradient problem in RNN and explain it.**

  S1: 'Jane walked into the room. John walked in too. Jane said hi to *?*'.

  S2: 'Jane walked into the room. John walked in too. It was late in the day, and everyone was walking home after a long day at work. Jane said hi to *?*'.

  In both cases, the RNN should predict John as an answer. However, in practice, it turns out the RNN is more likely to predict John in sentence 1 than in sentence 2. Indeed, during back-propagation, the contribution of gradient values **gradually vanishes as they propagate**

to earlier timesteps. Thus for long sentences, the RNN is less likely to recall information introduced in the earliest part of a sentence.

- **How to solve vanishing gradient problem?**

  Technique 1: Instead of initializing W randomly, start off from an identity matrix initialization.

  Technique 2: Use ReLU as an activation function since the derivative of the gradient is either 0 or 1. This way, gradients would flow through the neurons whose derivatives is 1 without getting attenuated while propagating back through time-steps.

- **What are exploding gradients and give a technique on how to solve them?**

  The explosion occurs through exponential growth by repeatedly multiplying gradients through the network layers that have values larger than 1.0.

  A technique to solve exploding gradients is gradient clipping. Gradient clipping is a simple heuristic invented by Thomas Mikolov to counter the effect of exploding gradient. That is, whenever the gradient reach a certain threshold, they are set back to a small number.

4.3 2.3. Deep bidirectional RNN

Assumption: It is possible to predict a word by looking at future words. We should be able to create more accurate and contextual representation of a word by concatenating forward and backward hidden layers.

- **What is a deep bidirectional RNN?**

  It is possible to make predictions based on future words in the sequence. The RNN should read forward and backward at the same time to make an accurate prediction at time t.

  This network maintains two hidden layers, one for the left-to-right propagation and another for the right-to-left propagation.

$$\overrightarrow{h_t} = f(\overrightarrow{W}x_t + \overrightarrow{V}\overrightarrow{h_{t-1}} + \vec{b})$$

$$\overleftarrow{h_t} = f(\overleftarrow{W}x_t + \overleftarrow{V}\overleftarrow{h_{t-1}} + \overleftarrow{b})$$

$$\hat{y_t} = g(U[\overrightarrow{h_t}; \overleftarrow{h_t}] + c)$$

- **What is a limitation of deep bidirectional RNN?**

  Exploding and vanishing gradients.

4.4 2.4. GRU and LSTM cells

- **Why do we need GRU?**

  Although RNNs can theoretically capture long-term dependencies, they are very hard to actually train to do this. GRU are designed in a manner to have more persistent memory thereby making it easier for RNNs to capture long-term dependencies.

- **What are the 4 gates of a GRU cell?**
  - New memory generation
  - Reset gate
  - Update gate
  - Hidden state
- **What are the new memory stage and reset gate in a GRU?**

  The reset signal is responsible for determining how important h(t-1) is to the summarization of h(t). The reset gate has the ability to completely diminish past hidden state if it finds that h(t-1) is irrelevant to the computation of the new memory.

  $$r_t = \sigma(W^{(r)}x_t + U^{(r)}h_{t-1})$$

  This stage is the one who knows the recipe of combining a newly observed word with the past hidden state h(t-1) to summarize this new word in light of the contextual past.

  $$h_t = tanh(r_t \circ Uh_{t-1} + Wx_t)$$
- **What is the update gate in a GRU?**

  The update gate is responsible for how much of h(t-1) should be carried forward to the next state.

  $$z_t = \sigma(W^{(z)}x_t + U^{(z)}h_{t-1})$$
- **Give the equation of the new hidden state in a GRU cell.**

  $$h_t = (1 - z_t) \circ \hat{h}_t + z_t \circ h_{t-1}$$

  where z is the update gate, h hat the new memory and h(t-1) the previous hidden state.
- **What are the 6 memory cells of LSTM?**
  - Input gate
  - Forget gate
  - Output/exposure gate
  - New memory cell
  - Final memory cell
- **What is the difference between GRU and LSTM cells?**

  The GRU controls the flow of information like the LSTM unit, but without having a memory unit. It just exposes the full hidden content without any control.

  GRU performance is on par with LSTM, but computationally more efficient (less complex structure as pointed out).

5 3. How to deal with sequential output with Seq2Seq models?

So far, we have predicted a single output: an NER label for a word, the single most likely next word in a sentence given the past few. However, there's a whole class of NLP tasks that rely on sequential output.

E.g.: translation, conversation, summarization

5.1 3.1. Seq2Seq models

- **What is the advantage of Seq2Seq models over regular RNNs?**

  Seq2Seq models can generate arbitrary output sequences after seeing the entire input. They can even focus in on specific parts of the input automatically to help generate a useful translation.

  E.g.: 1 encoder and 1 decoder (both LSTM or bi-LSTM)

- **What does an encoder do in a Seq2Seq model?**

  It reads the input sequence and generate a fixed-dimensional context vector C for the sequence.

  The encoder stacks multiple RNNs on top of each other to effectively compress an arbitrary-length sequence into a fixed-size vector. The final layer's LSTM hidden state will be used as a context vector.

- **What does a decoder do in a Seq2Seq model?**

  It uses the context vector as a *'seed'* from which to generate an output sequence.

- **In which order does an encoder read a sentence?**

  Reverse

- **How does a decoder work in a Seq2Seq model?**

  We initialize the hidden state of our first layer with the context vector, we then pass an token appended to the end of the input. We then run the three-stacked RNN, following up with a softmax on the final layer's output to generate the first word.

5.2 3.2. Attention mechanism

- **Why do we need attention mechanism?**

  Some words convey more information than others, hence requires more attention. Thus, there is a flaw in using the final RNN hidden state as a context vector: different parts of the input have different levels of significance. Moreover, different parts of the output may even consider different parts of the input *'important'*.

  E.g.: The ball is on the field.

  Most important words: ball, on, field.

- **What is self-attention?**

  The motivation for self-attention is that as the model processes each word (each position in the sequence), self attention allows it to look at other positions in the input sequence for clues

that can help lead to a better encoding for this word. **Simply put, self-attention allows each word in the sentence to look at other words to better know which words contribute for the current word.**

For example, when the model processing the word *it* in the right column in the above figure, the self-attention look at other words for better encode (understand) the current word it. This is the magic behind the scenes how BERT **can understand each word based on its context** (sentence). One word **can have different meanings in different sentences** (context), and self-attention can **encode** (understand) each word **based on context words for the current word.**

- **Explain how self-attention works mathematically and programmatically.**

  We define three matrices: query, key and value. Intuitively, we can think of query as what kind of information we are looking for, the key represents the relevance to the query, and the value represents the actual content of the input.

  The three matrices are the product of the input and a custom weight matrix to add more parameters to the model which enhances its ability to generalize.

  Each row in the input matrix X represents a word embedding vector. The following equation is called the scaled dot-product attention.

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V$$

  Computing the dot product between Q and K gives similarity scores between rows (word embedding vectors). So every line represents the similarity between one token and other tokens.

  Using softmax allows to normalize the matrix.

  V still represents the sentence with all the embedding vectors stacked. Hence multiplying V with the attention matrix acts as a weighted sum to give more importance to some vectors than others.

  The attention matrix acts as a filtering matrix, which can make the value matrix pay more attention to those important words.

- **How does attention work?**

  We compute an attention vector. It is a vector of weights that are used to compute the context vector as a weighted average of the hidden states generated by the encoder at time steps *1, …, n.*

  The decoder network is provided with a look at the entire input sequence at every decoding step since it is fed with the context vector. The decoder can then decide what input words are important.

- **What is global attention?**

  Instead of giving to the decoder's LSTM cells a single context vector which is a weighted average of all the hidden states of the encoder (attention). LSTM cells in the decoder are fed with the **concatenation** of the encoder hidden states at time i and the context vector.

- **Give 4 algorithms that can be used by decoders to search translations.**

  Exhaustive search, ancestral sampling, greedy search, beam search

- **How does beam search work?**

  We maintain K candidates at each time step. To do so, we compute H(t+1) by expanding H(t) and keeping the best K candidates (the ones with the highest probability).

6 4. How to deal with large vocabulary?

Seq2Seq models have a hard time dealing with large vocabulary size. These models predict the next word in the sequence by computing a target probabilistic distribution over the entire vocabulary using softmax. **Softmax is expensive to compute and its complexity scales proportionally to the vocabulary size**.

6.1 4.1. Scaling softmax

- **Give two techniques to scale softmax**.

  Noise contrastive estimation and hierarchical softmax

- **What is the issue with both techniques?**

  Both methods save computation during training step (when target word is known). At test time, one still has to compute the probability of all words in the vocabulary in order to make predictions.

6.2 4.2. Word and character-based models

Phonology posits a small set or sets of distinctive, categorical units that we call **phonemes**. Let's try to change our perspective: instead of working with words or even splitting words into characters, let's now consider **n-gram characters**.

- **Name three families of techniques to deal with unknown or rare words.**

  Word segmentation (adaptation of Byte-pair encoding), character-based model, hybrid model

6.2.1 4.2.1. Word segmentation

- **How does word segmentation solve the issue related to large vocabulary?**

  Word segmentation represents unknown or rare words as a sequence of subword units.

- **What is Byte-pair encoding?**

  Byte-pair encoding is a lossless compression algorithm. In NLP, it is adapted to transform rare words into character tokens.

  For instance: athazagoraphobia becomes ['_ath', 'az', 'agor', 'aphobia']

  It follows the following procedure:

  1. Represent each word in the corpus as a combination of the characters along with the special end of word token

2. Iteratively count character pairs in all tokens of the vocabulary
3. Merge every occurrence of the most frequent pair, add the new character n-gram to the vocabulary
4. Repeat step 3 until the desired number of merge operations are completed or the desired vocabulary size is achieved

   {'l o w ': 5, 'l o w e r ': 2, 'n e w e s t ': 6, 'w i d e s t ': 3}

   {'l o w ': 5, 'l o w e r ': 2, 'n e w es t ': 6, 'w i d es t ': 3}

   {'l o w ': 5, 'l o w e r ': 2, 'n e w est ': 6, 'w i d est ': 3}

- **How does word segmentation work?**

  Start with a vocabulary of characters and keep extending the vocabulary with most frequent n-gram pairs in the data set. This process is repeated until all n-gram pairs are selected or vocabulary size reaches some threshold.

- **What is the difference between WordPiece and SentencePiece?**

  WordPiece is an algorithm that creates smaller units than words (sub-word level).

  SentencePiece - created by Google - is an algorithm that combines sub-word level tokenization (BPE) as well as unigram tokenization.

6.2.2 4.2.2. Character-based models

- **How are character-level embeddings generated?**

  Character-level embeddings use one-dimensional CNN to find numeric representation of words by looking at their character-level compositions.

6.2.3 4.2.3. FastText embeddings

- **Give a brief explanation of FastText embeddings.**

  Simply put, FastText is a word2vec-like algorithm that leverages n-gram word pieces to encode a large and open vocabulary and to capture word morphology.

- **How are words represented in FastText?**

  They are represented as char n-grams augmented with boundary symbols and as whole word.

  E.g.: where = <wh, whe, her, ere, re>, < where >

- **How are FastText embeddings generated?**

  A bi-LSTM is trained to compute embeddings.

6.2.4 4.2.4. Hybrid NMT

- **How does a hybrid Neural Machine Translation work?**

  The system translates mostly at word-level and consults the character components for rare words.

- **Give the structure of a hybrid NMT.**

  A word-based translation as a backbone

  A source character-based representation

  A target character-level generation

- **What is the purpose of a word-based translation as a backbone in a hybrid NMT?**

  This is the core of a hybrid NMT (LSTM encoder-decoder) that translates at the word level. We use to represent OOV words.

- **What is the purpose of a source character-based representation model in a hybrid NMT?**

  A deep LSTM model that learns over characters of rare words and use the final hidden state of the LSTM as the representation for the rare word.

- **What is the purpose of a target character-level generation model?**

  The goal is to create a coherent representation that handles unlimited output vocabulary. To do so, we have a separate deep LSTM that *'translates'* at the character-level given the current word-level state.

7 5. How to create contextual word embeddings?

So far, we have always the same representation for a word type regardless of the context in which a word token occurs. Word embeddings are context-free with GloVe, Word2Vec or FastText. Those language models are trained to predict the next word but they are producing context-specific word representations at each position.

E.g.: I ate an apple. I have an Apple iPhone.

In both cases, the word *'apple'* shares the same word embedding. However, they mean two radically different things.

7.1 5.1. ELMo (Embeddings from language model)

- **What is ELMo?**

  ELMo learns contextualized word representation by pre-training a language model in an unsupervised way.

- **How is ELMo trained?**

  The bidirectional language model is the foundation of ELMo. While the input is a sequence of **n** tokens, the language model learns to predict the probability of next token given the history.

  Forward pass:

$$p(x_1, \dots, x_n) = \prod_{i=1}^{n} p(x_i \mid x_1, \dots, x_{i-1})$$

Backward pass:

$$p(x_1, \dots, x_n) = \prod_{i=1}^{n} p(x_i \mid x_{i+1}, \dots, x_n)$$

The model is trained to minimize the negative log-likelihood in both directions:

$$\mathcal{L} = -\sum_{i=1}^{n} \Big(\log p(x_i \mid x_1, \dots, x_{i-1}; \Theta_e, \overrightarrow{\Theta}_{\text{LSTM}}, \Theta_s) + \log p(x_i \mid x_{i+1}, \dots, x_n; \Theta_e, \overleftarrow{\Theta}_{\text{LSTM}}, \Theta_s) \Big)$$

- **How does ELMo learn task-specific representations?**

  On top of a L-layer biLM, ELMo stacks all the hidden states across layers together by learning a task-specific linear combination.

  The weights, s_task, in the linear combination are learned for each end task and normalized by softmax. The scaling factor task is used to correct the misalignment between the distribution of biLM hidden states and the distribution of task specific representations.

$$v_i = f(R_i; \Theta^{\text{task}}) = \gamma^{\text{task}} \sum_{\ell=0}^{L} s_i^{\text{task}} \mathbf{h}_{i,\ell}$$

- **To which tasks correspond which layers?**

  The comparison study indicates that syntactic information is better represented at lower layers while semantic information is captured by higher layers. Because different layers tend to carry different type of information, **stacking them together helps**.

7.2 5.2. ULMFiT

- **What is innovative with ULMFiT?**

  ULMFiT is the first model to introduce the idea of generative pretrained language models that is fine-tuned for a specific task.

- **What are the three steps to achieve good transfer learning results?**

  1. General LM pre-training (already done by fast.ai researchers) on Wikipedia
  2. Target task LM fine-tuning: finetuning LM on a specific vocabulary
  3. Train a target task classifier with 2 fully-connected layers

- **What are the two techniques used when fine-tuning LM?**

  1. **Discriminative fine-tuning**: tune each layer with different learning rates
  2. **Slanted triangular learning rates**: triangular learning rate schedule

- **What are the two techniques used when fine-tuning the classifier?**

  1. **Concat pooling** extracts max-polling and mean-pooling over the history of hidden states and concatenates them with the final hidden state.

2. **Gradual unfreezing** extracts max-polling and mean-pooling over the history of hidden states and concatenates them with the final hidden state.

7.3 5.3. Transformers

7.3.1 5.3.1. OpenAI GPT

- **What does GPT stand for?**

  Generative Pre-training transformer

- **What are the two major difference between ELMo and OpenAI GPT?**

  1. The model architectures are different: ELMo uses a shallow concatenation of independently trained left-to-right and right-to-left multi-layer LSTMs, while GPT is a multi-layer transformer decoder.
  2. ELMo feeds embeddings into models customized for specific tasks as additional features, while GPT fine-tunes the same base model for all end tasks.

- **What is the major upgrade brought by OpenAI GPT?**

  To get rid of the task-specific model and use the pre-trained language model directly.

  Hence we don't need new a new design for specific tasks. We just need to modify the input sequence by adding custom tags. At the first stage, generative pre-training on a language model can absorb as much free text as possible. Then at the second stage, the model is fine-tuned on specific tasks with a small labeled dataset and a minimal set of new parameters to learn.

- **What is the loss of OpenAI GPT?**

  The loss is the negative log-likelihood for true labels to which we add the LM loss as an auxiliary loss.

$$\mathcal{L}_{\text{cls}} = \sum_{(\mathbf{x},y)\in\mathcal{D}} \log P(y \mid x_1, \dots, x_n) = \sum_{(\mathbf{x},y)\in\mathcal{D}} \log \text{softmax}(\mathbf{h}_L^{(n)}(\mathbf{x})\mathbf{W}_y)$$

$$\mathcal{L}_{\text{LM}} = -\sum_i \log p(x_i \mid x_{i-k}, \dots, x_{i-1})$$

$$\mathcal{L} = \mathcal{L}_{\text{cls}} + \lambda \mathcal{L}_{\text{LM}}$$

- **What is a limitation of OpenAI GPT?**

  Its unidirectional nature.

7.3.2 5.3.2. BERT

- **What is the biggest difference between OpenAI GPT and BERT? Which limitation does it solve?**

  OpenAI is unidirectional. BERT is bidirectional.

- **What is the base component of BERT?**

  The model architecture of BERT is a multi-layer bidirectional Transformer encoder. It is composed of Multi-headed self attention, feed-forward layers, layer norm and residuals and positional embeddings.

- **What are the tasks on which BERT is trained?**

  1. **Mask language model (MLM)**: Randomly mask 15% of tokens in each sequence.

     BERT employed several heuristic tricks:

     - (a) with 80% probability, replace the chosen words with `[MASK]`;
     - (b) with 10% probability, replace with a random word;
     - (c) with 10% probability, keep it the same.

  2. **Next sentence prediction**: tell whether one sentence is the next sentence of the other. It is used to learn relationships between sentences.

- **Give the 3 components that constitutes BERT input embedding.**

  1. WordPiece embeddings (cf. previously)
  2. Segment embeddings
  3. Position embeddings (are learned)

- **What is the structure of BERT base?**

  12 layers, 768-dimensional output hidden state, 12 heads

- **What is the structure of BERT large?**

  24 layers, 1024-dimensional output hidden state, 16 heads

- **What is the [SEP] token used for?**

  [SEP] token is used when building a sequence from multiple sequences

  E.g.: two sequences for sequence classification or for a text and a question for question answering

- **What is the [PAD] token used for?**

  The token used for padding, for example when batching sequences of different lengths

- **What is the [CLS] token used for?**

  The classifier token which is used when doing sequence classification. It is the first token of the sequence when built with special tokens.

7.3.3 5.3.3. RoBERTa

- **What does RoBERTa stand for?**

  Robustly optimized BERT approach

7.3.4 5.3.4. ALBERT

- **What are the 3 innovations brought by ALBERT?**

  1. Factorized Embedding parametrization
  2. Cross-layer parameter sharing
  3. Sentence order prediction

- **What is factorized embedding parametrization?**

  In BERT, the WordPiece tokenization embedding size $\boldsymbol{E}$ is configured to be the same as the hidden state size $\boldsymbol{H}$. That is saying, if we want to increase the model size (larger $\boldsymbol{H}$), we need to learn a larger tokenization embedding too, which is expensive because it depends on the vocabulary size ($\boldsymbol{V}$).

  Conceptually, because the tokenization embedding is expected to learn *context-independent* representation and the hidden states are *context-dependent*, it makes sense to separate the size of the hidden layers from the size of vocabulary embedding. Using factorized embedding parameterization, the large vocabulary embedding matrix of size $\boldsymbol{V \times H}$ is decomposed into two small matrices of size $\boldsymbol{V \times E}$ and $\boldsymbol{E \times H}$.

7.3.5 5.3.5. ELECTRA

7.3.6 5.3.6. DistilBERT

- **How does distillation work?**

  - Train "Teacher": Use SOTA pre-training + fine-tuning technique to train model with maximum accuracy
  - Label a large amount of unlabeled input examples with Teacher
  - Train "Student": much smaller model which is trained to mimic Teacher output
  - Student objective is typically MSE or cross-entropy

7.3.7 5.3.7. XLNet

- **What are two innovations of XLNet?**

  1. Relative position embeddings
  2. Permutation language modelling

8 6. Miscellaneous

- **Explain what is self-supervised learning.**

  Given a task and enough labels, supervised learning can solve it really well. Good performance usually requires a decent amount of labels, but collecting manual labels is expensive (i.e. ImageNet) and hard to be scaled up. Considering the amount of unlabelled data (e.g. free text, all the images on the Internet) is substantially more than a limited number of human curated labelled datasets, it is kinda wasteful not to use them. However, **unsupervised learning is not easy and usually works much less efficiently than supervised learning.**

Self-supervised learning consists in getting labels for free for unlabelled data and train unsupervised dataset in a supervised manner.

- **What is a problem induced by ReLU activations?**

  Exploding gradients, dying ReLU, mean and variance of activations is not 0 or 1

- **How to solve exploding gradients?**

  Gradient clipping

- **What is dying ReLU?**

  No learning if the activation is 0

- **How to solve dying ReLU?**

  Parametric ReLU

- **What is the difference between learning latent features using SVD and getting embedding vectors from a neural network?**

  SVD uses linear combination of inputs while a neural network uses non-linear combination.

- **Give the time complexity of LSTM.**

$$seqlength * hiddensize^2$$

- **Give the time complexity of a transformer.**

$$seqlength^2 * hiddensize$$

- **How is AdamW different from Adam?**

  AdamW is Adam with L2 regularization on weight as models with smaller weights generalise better

- **What is the difference between LayerNorm and BatchNorm?**

  BatchNorm: compute the mean and variance at each layer for every minibatch

  LayerNorm: compute the mean and variance for every single sample for each layer independently

- **What changes would you make to your deep learning code if you knew there are errors in your training data?**

  Label smoothing where the smoothening values is based on % error.

  Use class weights to modify the loss