Generalized Language Models

Jan 31, 2019 by Lilian Weng

As a follow up of word embedding post, we will discuss the models on learning contextualized word vectors as well as the new trend in large unsupervised pre-trained language models which have achieved amazing SOTA results on a variety of language tasks.

Fig. 0: I guess they are Elmo & Bert? (Image source: [here](https://lilianweng.github.io/lil-log/2019/01/31/generalized-lang/))

We have seen amazing progress in NLP in 2018. Large-scale pre-trained language modes like OpenAI GPT and BERT have achieved great performance on a variety of language tasks using generic model architectures. The idea is similar to how ImageNet classification pre-training helps many vision tasks (*). Even better than vision classification pre-training, this simple and powerful approach in NLP does not require labeled data for pre-training, allowing us to experiment with increased training scale, up to our very limit.

(*) Although recently He et al. (2018) found that pre-training might not be necessary for image segmentation task.

In my previous NLP post on word embedding, the introduced embeddings are not context-specific — they are learned based on word concurrency but not sequential context. So in two sentences, “I am eating an apple” and “I have an Apple phone”, two “apple” words refer to very different things but they would still share the same word embedding vector.

Despite this, early adoption of word embeddings in problem-solving is to use them as additional features for an existing task-specific model and in a way the improvement is bounded.

In this post, we will discuss how various approaches were proposed to make embeddings dependent on context, and to make them easier and cheaper to be applied to downstream tasks in general form.

- **CoVe**
  - NMT Recap
  - Use CoVe in Downstream Tasks
- **ELMo**
  - Bidirectional Language Model
  - ELMo Representations
  - Use ELMo in Downstream Tasks
- **Cross-View Training**
  - Model Architecture
  - Multi-Task Learning
  - Use CVT in Downstream Tasks
- **OpenAI GPT**
  - Transformer Decoder as Language Model
CoVe

CoVe (McCann et al. 2017), short for Contextual Word Vectors, is a type of word embeddings learned by an encoder in an attentional seq-to-seq machine translation model. Different from traditional word embeddings introduced here, CoVe word representations are functions of the entire input sentence.

NMT Recap

Here the Neural Machine Translation (NMT) model is composed of a standard, two-layer, bidirectional LSTM encoder and an attentional two-layer unidirectional LSTM decoder. It is pre-trained on the English-German translation task. The encoder learns and optimizes the embedding vectors of English words in order to translate them to German. With the intuition that the encoder should capture high-level semantic and syntactic meanings before transforming words into another language, the encoder output is used to provide contextualized word embeddings for various downstream language tasks.

![Diagram of NMT model](image)

**Fig. 1.** The NMT base model used in CoVe.

- A sequence of $n$ words in source language (English): $x = [x_1, \ldots, x_n]$.
- A sequence of $m$ words in target language (German): $y = [y_1, \ldots, y_m]$.
- The GloVe vectors of source words: $\text{GloVe}(x)$.
- Randomly initialized embedding vectors of target words: $z = [z_1, \ldots, z_m]$.
- The biLSTM encoder outputs a sequence of hidden states: $h = [h_1, \ldots, h_n] = \text{biLSTM}(\text{GloVe}(x))$ and $h_t = [\overrightarrow{h_t}, \overleftarrow{h_t}]$
where the forward LSTM computes \( \overrightarrow{h}_t = \text{LSTM}(x_t, \overrightarrow{h}_{t-1}) \) and the backward computation gives us \( \overleftarrow{h}_t = \text{LSTM}(x_t, \overleftarrow{h}_{t-1}) \).

- The attentional decoder outputs a distribution over words: \( p(y_t \mid H, y_1, \ldots, y_{t-1}) \) where \( H \) is a stack of hidden states \( \{\hat{h}\} \) along the time dimension:

  \[
  \text{decoder hidden state: } s_t = \text{LSTM}([z_{t-1}; \overrightarrow{h}_{t-1}], s_{t-1}) \\
  \text{attention weights: } \alpha_t = \text{softmax}(H(W_1s_t + b_1)) \\
  \text{context-adjusted hidden state: } \hat{h}_t = \text{tanh}(W_2[H^\top \alpha_t; s_t] + b_2) \\
  \text{decoder output: } p(y_t \mid H, y_1, \ldots, y_{t-1}) = \text{softmax}(W_{\text{out}}\hat{h}_t + b_{\text{out}})
  \]

### Use CoVe in Downstream Tasks

The hidden states of NMT encoder are defined as **context vectors** for other language tasks:

\[
\text{CoVe}(x) = \text{biLSTM}(\text{GloVe}(x))
\]

The paper proposed to use the concatenation of GloVe and CoVe for question-answering and classification tasks. GloVe learns from the ratios of global word co-occurrences, so it has no sentence context, while CoVe is generated by processing text sequences is able to capture the contextual information:

\[
v = [\text{GloVe}(x); \text{CoVe}(x)]
\]

Given a downstream task, we first generate the concatenation of Glove + CoVe vectors of input words and then feed them into the task-specific models as additional features.

### Fig. 2. The CoVe embeddings are generated by an encoder trained for machine translation task. The encoder can be plugged into any downstream task-specific model. (Image source: original paper)

**Summary:** The limitation of CoVe is obvious: (1) pre-training is bounded by available datasets on the supervised translation task; (2) the contribution of CoVe to the final performance is constrained by the task-specific model architecture.

In the following sections, we will see that ELMo overcomes issue (1) by unsupervised pre-training and OpenAI GPT & BERT further overcome both problems by unsupervised pre-training + using generative model architecture for different downstream tasks.

### ELMo

**ELMo.** short for **Embeddings from Language Model** (Peters. et al. 2018) learns contextualized word representation by pre-training a language model in an unsupervised way.
Bidirectional Language Model

The bidirectional Language Model (biLM) is the foundation for ELMo. While the input is a sequence of \( n \) tokens, \( (x_1, \ldots, x_n) \), the language model learns to predict the probability of next token given the history.

In the forward pass, the history contains words before the target token:

\[
p(x_1, \ldots, x_n) = \prod_{i=1}^{n} p(x_i \mid x_1, \ldots, x_{i-1})
\]

In the backward pass, the history contains words after the target token:

\[
p(x_1, \ldots, x_n) = \prod_{i=1}^{n} p(x_i \mid x_{i+1}, \ldots, x_n)
\]

The predictions in both directions are modeled by multi-layer LSTMs with hidden states \( \overrightarrow{h_{i, \ell}} \) and \( \overleftarrow{h_{i, \ell}} \) for input token \( x_i \) at the layer level \( \ell = 1, \ldots, \ell_L \). The final layer's hidden state \( \overrightarrow{h_{i,L}} \) is used to output the probabilities over tokens after softmax normalization. They share the embedding layer and the softmax layer, parameterized by \( \Theta_e \) and \( \Theta_s \) respectively.

**Fig. 3.** The biLSTM base model of ELMo. (Image source: recreated based on this.)

The model is trained to minimize the negative log likelihood (= maximize the log likelihood for true words) in both directions:

\[
L = - \sum_{i=1}^{n} \left( \log p(x_i \mid x_1, \ldots, x_{i-1}; \Theta_e, \overrightarrow{\Theta_{LSTM}}, \Theta_s) + \log p(x_i \mid x_{i+1}, \ldots, x_n; \Theta_e, \overleftarrow{\Theta_{LSTM}}, \Theta_s) \right)
\]

**ELMo Representations**

On top of a \( \ell_L \)-layer biLM, ELMo stacks all the hidden states across layers together by learning a task-specific linear combination. The hidden state representation for the token \( x_i \) contains \( 2\ell_L + 1 \) vectors:
\[ R_i = \{ h_{i, \ell} \mid \ell = 0, \ldots, L \} \text{ where } h_{i, \ell} \text{ is the embedding layer output and } h_{i, \ell} = \overrightarrow{h}_{i, \ell} \overleftarrow{h}_{i, \ell}. \]

The weights \( s_{\text{task}} \) in the linear combination are learned for each end task and normalized by softmax. The scaling factor \( \gamma_{\text{task}} \) is used to correct the misalignment between the distribution of biLM hidden states and the distribution of task specific representations.

\[ v_i = f(R_i; \Theta_{\text{task}}) = \gamma_{\text{task}} \sum_{\ell=0}^{L} s_{i, \ell} h_{i, \ell} \]

To evaluate what kind of information is captured by hidden states across different layers, ELMo is applied on semantic-intensive and syntax-intensive tasks respectively using representations in different layers of biLM:

- **Semantic task**: The word sense disambiguation (WSD) task emphasizes the meaning of a word given a context. The biLM top layer is better at this task than the first layer.
- **Syntax task**: The part-of-speech (POS) tagging task aims to infer the grammatical role of a word in one sentence. A higher accuracy can be achieved by using the biLM first layer than the top layer.

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.

**Use ELMo in Downstream Tasks**

Similar to how CoVe can help different downstream tasks, ELMo embedding vectors are included in the input or lower levels of task-specific models. Moreover, for some tasks (i.e., SNLI and SQuAD, but not SRL), adding them into the output level helps too.

The improvements brought up by ELMo are largest for tasks with a small supervised dataset. With ELMo, we can also achieve similar performance with much less labeled data.

**Summary**: The language model pre-training is unsupervised and theoretically the pre-training can be scaled up as much as possible since the unlabeled text corpora are abundant. However, it still has the dependency on task-customized models and thus the improvement is only incremental while searching for a good model architecture for every task remains non-trivial.

**Cross-View Training**

In ELMo the unsupervised pre-training and task-specific learning happen for two independent models in two separate training stages. **Cross-View Training** (abbr. CVT; Clark et al., 2018) combines them into one unified semi-supervised learning procedure where the representation of a biLSTM encoder is improved by both supervised learning with labeled data and unsupervised learning with unlabeled data on auxiliary tasks.

**Model Architecture**

The model consists of a two-layer bidirectional LSTM encoder and a primary prediction module. During training, the model is fed with labeled and unlabeled data batches alternatively.

- **On labeled examples**, all the model parameters are updated by standard supervised learning. The loss is the standard cross entropy.
- **On unlabeled examples**, the primary prediction module still can produce a “soft” target, even though we cannot know exactly how accurate they are. In a couple of auxiliary tasks, the predictor only sees and processes a restricted view of the input, such as only using encoder hidden state representation in one direction. The auxiliary task outputs are expected to match the primary prediction target for a full view of input.

In this way, the encoder is forced to distill the knowledge of the full context into partial representation. At this stage, the biLSTM encoder is backpropagated but the primary prediction module is fixed. The loss is to minimize the distance between
auxiliary and primary predictions.

**Learning on Labeled Examples**

"She lives in Washington."

**Learning on Unlabeled Examples**

"They traveled to Washington by plane."

**Inputs Seen by Auxiliary Prediction Modules**

- Auxiliary 1: They traveled to ____________
- Auxiliary 2: They traveled to Washington ______
- Auxiliary 3: ____________ Washington by plane
- Auxiliary 4: ____________ by plane

**Multi-Task Learning**

When training for multiple tasks simultaneously, CVT adds several extra primary prediction models for additional tasks. They all share the same sentence representation encoder. During supervised training, once one task is randomly selected, parameters in its corresponding predictor and the representation encoder are updated. With unlabeled data samples, the encoder is optimized jointly across all the tasks by minimizing the differences between auxiliary outputs and primary prediction for every task.

The multi-task learning encourages better generality of representation and in the meantime produces a nice side-product: all-tasks-labeled examples from unlabeled data. They are precious data labels considering that cross-task labels are useful but fairly rare.

**Use CVT in Downstream Tasks**

Theoretically the primary prediction module can take any form, generic or task-specific design. The examples presented in the CVT paper include both cases.

In sequential tagging tasks (classification for every token) like NER or POS tagging, the predictor module contains two fully connected layers and a softmax layer on the output to produce a probability distribution over class labels. For each token \( x_i \), we take the corresponding hidden states in two layers, \( h_1^{(i)} \) and \( h_2^{(i)} \):

\[
p_\theta(y_i | x_i) = \text{NN}(h_1^{(i)}) = \text{NN}(h_1^{(i)}; h_2^{(i)}) = \text{softmax}(W \cdot \text{ReLU}(W' \cdot [h_1^{(i)}; h_2^{(i)}]) + b)
\]

The auxiliary tasks are only fed with forward or backward LSTM state in the first layer. Because they only observe partial context either on the left or right, they have to learn like a language model, trying to predict the next token given the context. The `fwd` and `bwd` auxiliary tasks only take one direction. The `future` and `past` tasks take one step further in forward and backward direction, respectively.
\[
\begin{align*}
    p^{\text{fwd}}_\theta(y_i | x_i) &= \text{NN}^{\text{fwd}}(\mathbf{h}^{(i)}) \\
    p^{\text{bwd}}_\theta(y_i | x_i) &= \text{NN}^{\text{bwd}}(\mathbf{h}^{(i)}) \\
    p^{\text{future}}_\theta(y_i | x_i) &= \text{NN}^{\text{future}}(\mathbf{h}^{(i-1)}) \\
    p^{\text{past}}_\theta(y_i | x_i) &= \text{NN}^{\text{past}}(\mathbf{h}^{(i+1)})
\end{align*}
\]

Fig. 5. The sequential tagging task depends on four auxiliary prediction models, their inputs only involving hidden states in one direction: forward, backward, future and past. (Image source: original paper)

Note that if the primary prediction module has dropout, the dropout layer works as usual when training with labeled data, but it is not applied when generating “soft” target for auxiliary tasks during training with unlabeled data.

In the machine translation task, the primary prediction module is replaced with a standard unidirectional LSTM decoder with attention. There are two auxiliary tasks: (1) apply dropout on the attention weight vector by randomly zeroing out some values; (2) predict the future word in the target sequence. The primary prediction for auxiliary tasks to match is the best predicted target sequence produced by running the fixed primary decoder on the input sequence with beam search.

**OpenAI GPT**

Following the similar idea of ELMo, OpenAI GPT, short for Generative Pre-training Transformer (Radford et al., 2018) expands the unsupervised language model to a much larger scale by training on a giant collection of free text corpora. Despite of the similarity, GPT has two major differences from ELMo:

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. The use of contextualized embeddings in downstream tasks are different: ELMo feeds embeddings into models customized for specific tasks as additional features, while GPT fine-tunes the same base model for all end tasks.
   - Generative pre-trained LM + task-specific fine-tuning has been proved to work in ULMFiT, where the fine-tuning happens in all layers gradually. ULMFiT focuses on training techniques for stabilizing the fine-tuning process.

**Transformer Decoder as Language Model**

Compared to the original transformer architecture, the transformer decoder model discards the encoder part, so there is only one
single input sentence rather than two separate source and target sequences.

This model applies multiple transformer blocks over the embeddings of input sequences. Each block contains a masked multi-headed self-attention layer and a pointwise feed-forward layer. The final output produces a distribution over target tokens after softmax normalization.

Fig. 6. The transformer decoder model architecture in OpenAI GPT.

The loss is the negative log-likelihood, same as ELMo, but without backward computation. Let’s say, the context window of the size $k$ is located before the target word and the loss would look like:

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

**Supervised Fine-Tuning**

The most substantial upgrade that OpenAI GPT proposed is to get rid of the task-specific model and use the pre-trained language model directly!

Let’s take classification as an example. Say, in the labeled dataset, each input has $n$ tokens, $\mathbf{x} = (x_1, \ldots, x_n)$, and one label $y$. GPT first processes the input sequence $\mathbf{x}$ through the pre-trained transformer decoder and the last layer output for the last token $x_n$ is $h_L^{(n)}$. Then with only one new trainable weight matrix $\mathbf{W}_y$, it can predict a distribution over class labels.

$$
P(y \mid x_1, \ldots, x_n) = \text{softmax}(h_L^{(n)} \mathbf{W}_y)
$$

The loss is to minimize the negative log-likelihood for true labels. In addition, adding the LM loss as an auxiliary loss is found to be beneficial, because:
• (1) it helps accelerate convergence during training and
• (2) it is expected to improve the generalization of the supervised model.

\[
L_{cb} = \sum_{(x,y) \in D} \log P(y \mid x_1, \ldots, x_n) = \sum_{(x,y) \in D} \log \text{softmax}(h_L^{(n)}(x)W_y) \\
L_{LM} = -\sum_i \log p(x_i \mid x_{i-k}, \ldots, x_{i-1}) \\
\mathcal{L} = L_{cb} + \lambda L_{LM}
\]

With similar designs, no customized model structure is needed for other end tasks (see Fig. 7). If the task input contains multiple sentences, a special delimiter token (§) is added between each pair of sentences. The embedding for this delimiter token is a new parameter we need to learn, but it should be pretty minimal.

For the sentence similarity task, because the ordering does not matter, both orderings are included. For the multiple choice task, the context is paired with every answer candidate.

**Fig. 7. Training objects in slightly modified GPT transformer models for downstream tasks. (Image source: original paper)**

**Summary:** It is super neat and encouraging to see that such a general framework is capable to beat SOTA on most language tasks at that time (June 2018). At the first stage, generative pre-training of 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.

One limitation of GPT is its uni-directional nature — the model is only trained to predict the future left-to-right context.

**BERT**

**BERT**, short for **Bidirectional Encoder Representations from Transformers** (Devlin, et al., 2019) is a direct descendant to GPT: train a large language model on free text and then fine-tune on specific tasks without customized network architectures.

Compared to GPT, the largest difference and improvement of BERT is to make training bi-directional. The model learns to predict both context on the left and right. The paper according to the ablation study claimed that:
“bidirectional nature of our model is the single most important new contribution”

Auxiliary Tasks

The model architecture of BERT is a multi-layer bidirectional Transformer encoder.

![Transformer Encoder](image)

Fig. 8. Recap of Transformer Encoder model architecture. (Image source: Transformer paper)

To encourage the bi-directional prediction and sentence-level understanding, BERT applied two auxiliary tasks in addition to the basic language task (that is, to predict the next token given context).

Task 1: Mask language model (MLM)

From Wikipedia: “A cloze test (also cloze deletion test) is an exercise test or assessment consisting of a portion of language with certain items, words, or signs removed (cloze text), where the participant is asked to replace the missing language item ... The exercise was first described by WL Taylor in 1953.”

It is unsurprising to believe that a representation that learns the context around a word rather than just after the word is able to better capture its meaning, both syntactically and semantically. BERT encourages the model to do so by training on the “mask language model” task:

1. Randomly mask 15% of tokens in each sequence. Because if we only replace masked tokens with a special placeholder [MASK], the special token would never be encountered during fine-tuning. Hence, 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. The model only predicts the missing words, but it has no information on which words have been replaced or which words should be predicted. The output size is only 15% of the input size.

Task 2: Next sentence prediction

Motivated by the fact that many downstream tasks involve the understanding of relationships between sentences (i.e. QA, NLI), BERT added another auxiliary task on training a binary classifier for telling whether one sentence is the next sentence of the other:

1. Sample sentence pairs (A, B) so that:
   - (a) 50% of the time, B follows A;
2. The model processes both sentences and output a binary label indicating whether B is the next sentence of A.

The training data for both auxiliary tasks above can be trivially generated from any monolingual corpus. Hence the scale of training is unbounded. The training loss is the sum of the mean masked LM likelihood and mean next sentence prediction likelihood.

![Comparison of BERT, OpenAI GPT and ELMo model architectures.](original paper)

**Input Embedding**

The input embedding is the sum of three parts:

1. **WordPiece tokenization embeddings**: The WordPiece model was originally proposed for Japanese or Korean segmentation problem. Instead of using naturally split English word, they can be further divided into smaller sub-word units so that it is more effective to handle rare or unknown words. Please read linked papers for the optimal way to split words if interested.

2. **Segment embeddings**: If the input contains two sentences, they have sentence A embeddings and sentence B embeddings respectively and they are separated by a special character [SEP]. Only sentence A embeddings are used if the input only contains one sentence.

3. **Position embeddings**: Positional embeddings are learned rather than hard-coded.

![Figure 10: BERT input representation.](original paper)

Note that the first token is always forced to be [CLS] — a placeholder that will be used later for prediction in downstream tasks.

**Use BERT in Downstream Tasks**

BERT fine-tuning requires only a few new parameters added, just like OpenAI GPT.

For classification tasks, we get the prediction by taking the final hidden state of the special first token [CLS], \( \mathbf{h}_L^{[CLS]} \), and multiplying it with a small weight matrix, \( \text{softmax}(\mathbf{h}_L^{[CLS]} \mathbf{W}_{cls}) \).
For QA tasks like SQuAD, we need to predict the text span in the given paragraph for an given question. BERT predicts two probability distributions of every token, being the start and the end of the text span. Only two new small matrices, $W_s$ and $W_e$, are newly learned during fine-tuning and $\text{softmax}(h^i_L W_s)$ and $\text{softmax}(h^i_L W_e)$ define two probability distributions.

Overall the add-on part for end task fine-tuning is very minimal — one or two weight matrices to convert the Transformer hidden states to an interpretable format. Check the paper for implementation details for other cases.

Fig. 11. Training objects in slightly modified BERT models for downstream tasks. (Image source: original paper)

A summary table compares differences between fine-tuning of OpenAI GPT and BERT.

<table>
<thead>
<tr>
<th>OpenAI GPT</th>
<th>BERT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Special char</td>
<td>[SEP] and [CLS] are only introduced at fine-tuning stage.</td>
</tr>
<tr>
<td>[SEP] and [CLS] and sentence A/B embeddings are learned at the pre-training stage.</td>
<td></td>
</tr>
</tbody>
</table>
Summary

<table>
<thead>
<tr>
<th>Base model</th>
<th>pre-training</th>
<th>Downstream tasks</th>
<th>Downstream model</th>
<th>Fine-tuning</th>
</tr>
</thead>
<tbody>
<tr>
<td>CoVe</td>
<td>seq2seq NMT model</td>
<td>supervised</td>
<td>feature-based</td>
<td>task-specific</td>
</tr>
<tr>
<td>ELMo</td>
<td>two-layer bilSTM</td>
<td>unsupervised</td>
<td>feature-based</td>
<td>task-specific</td>
</tr>
<tr>
<td>CVT</td>
<td>two-layer bilSTM</td>
<td>semi-supervised</td>
<td>model-based</td>
<td>task-specific / task-agnostic</td>
</tr>
<tr>
<td>ULMFT</td>
<td>AWD-LSTM</td>
<td>unsupervised</td>
<td>model-based</td>
<td>task-agnostic</td>
</tr>
<tr>
<td>OpenAI GPT</td>
<td>Transformer decoder</td>
<td>unsupervised</td>
<td>model-based</td>
<td>task-agnostic</td>
</tr>
<tr>
<td>BERT</td>
<td>Transformer encoder</td>
<td>unsupervised</td>
<td>model-based</td>
<td>task-agnostic</td>
</tr>
</tbody>
</table>

Metric: Perplexity

Perplexity is often used as an intrinsic evaluation metric for gauging how well a language model can capture the real word distribution conditioned on the context.

A *perplexity* of a discrete probability distribution $p$ is defined as the exponentiation of the entropy:

$$2^H(p) = 2^{-\sum_x p(x) \log_2 p(x)}$$

Given a sentence with $N$ words, $s = (w_1, \ldots, w_N)$, the entropy looks as follows, simply assuming that each word has the same frequency, $\frac{1}{N}$:

$$H(s) = -\sum_{i=1}^{N} P(w_i) \log_2 p(w_i) = -\sum_{i=1}^{N} \frac{1}{N} \log_2 p(w_i)$$

The perplexity for the sentence becomes:

$$2^H(s) = 2^{-\frac{1}{N} \sum_{i=1}^{N} \log_2 p(w_i)} = (2^{\sum_{i=1}^{N} \log_2 p(w_i)})^{-\frac{1}{N}} = (p(w_1) \cdots p(w_N))^{-\frac{1}{N}}$$

A good language model should predict high word probabilities. Therefore, the smaller perplexity the better.

Common Tasks and Datasets

Question-Answering

- **SQUAD (Stanford Question Answering Dataset):** A reading comprehension dataset consisting of questions posed on a set of Wikipedia articles, where the answer to every question is a span of text.
- **RACE (ReAding Comprehension from Examinations):** A large-scale reading comprehension dataset with more than 28,000 passages and nearly 100,000 questions. The dataset is collected from English examinations in China, which are designed for middle school and high school students.
Commonsense Reasoning

- **Story Cloze Test**: A commonsense reasoning framework for evaluating story understanding and generation. The test requires a system to choose the correct ending to multi-sentence stories from two options.
- **SWAG** (Situations With Adversarial Generations): multiple choices; contains 113k sentence-pair completion examples that evaluate grounded common-sense inference

Natural Language Inference (NLI): also known as **Text Entailment**, an exercise to discern in logic whether one sentence can be inferred from another.

- **RTE** (Recognizing Textual Entailment): A set of datasets initiated by text entailment challenges.
- **SNLI** (Stanford Natural Language Inference): A collection of 570k human-written English sentence pairs manually labeled for balanced classification with the labels entailment, contradiction, and neutral.
- **MNLI** (Multi-Genre NLI): Similar to SNLI but with a more diverse variety of text styles and topics, collected from transcribed speech, popular fiction, and government reports.
- **QNLI** (Question NLI): Converted from SQuAD dataset to be a binary classification task over pairs of (question, sentence).
- **SciTail**: An entailment dataset created from multiple-choice science exams and web sentences.

Named Entity Recognition (NER): labels sequences of words in a text which are the names of things, such as person and company names, or gene and protein names

- **ConLL 2003 NER task**: consists of newswire from the Reuters, concentrating on four types of named entities: persons, locations, organizations and names of miscellaneous entities.
- **OntoNotes 0.5**: This corpus contains text in English, Arabic and Chinese tagged with four different entity types (PER, LOC, ORG, Misc).
- **Reuters Corpus**: A large collection of Reuters News stories.
- **Fine-Grained NER (FGN)**

Sentiment Analysis

- **SST** (Stanford Sentiment Treebank)
- **IMDb**: A large dataset of movie reviews with binary sentiment classification labels.

Semantic Role Labeling (SRL): models the predicate-argument structure of a sentence, and is often described as answering "Who did what to whom?"

- **CoNLL-2004 & CoNLL-2005**

Sentence similarity: also known as **paraphrase detection**

- **MRPC** (Microsoft Paraphrase Corpus): It contains pairs of sentences extracted from news sources on the web, with annotations indicating whether each pair is semantically equivalent.
- **QQP** (Quora Question Pairs) STS Benchmark: Semantic Textual Similarity

Sentence Acceptability: a task to annotate sentences for grammatical acceptability.

- **CoLA** (Corpus of Linguistic Acceptability): a binary single-sentence classification task

Text Chunking: To divide a text in syntactically correlated parts of words.

- **CoNLL-2000**

Part-of-Speech (POS) Tagging: tag parts of speech to each token, such as noun, verb, adjective, etc.; the Wall Street Journal portion of the Penn Treebank (Marcus et al., 1993).

Machine Translation: See **Standard NLP page**.

- **WMT 2015 English-Czech data (Large)**
- **WMT 2014 English-German data (Medium)**
IWSLT 2015 English-Vietnamese data (Small)

Coreference Resolution: cluster mentions in text that refer to the same underlying real world entities.

- CoNLL-2012

GLUE multi-task benchmark: https://gluebenchmark.com

Unsupervised pre-training dataset

- Books corpus: The corpus contains “over 7,000 unique unpublished books from a variety of genres including Adventure, Fantasy, and Romance.”
- 1B Word Language Model Benchmark
- English Wikipedia: ~2500M words

Reference


