Neural Network-based Language Models Implementation on Goodreads dataset

Babaei Elham, Moret Chiara, Moustafa Fatma

Course of AA 2021-2022 - Deep Learning

1 A breief recap of RNNs

RNNs(Recurrent Neural Networks) have internal memory, thus they can remember important features about the input they received. This enables them to be more precise in prediction of what is coming next. They are highly used with sequential data like text, speech, time series, audio, video and more. Unlike the feed forward neural networks, in RNNs the data cycles in a loop. Inputs coming from previous steps are considered when calculating current input. RNNs can be employed for a variety of tasks: binary classification, text generation, sentiment classification, machine translation and many more.

BackPropagation Through Time Backpropagation through time is actually a specific application of backpropagation in RNNs [Wer90]. It requires us to expand the computational graph of an RNN one time step at a time to obtain the dependencies among model variables and parameters. Then, based on the chain rule, we apply backpropagation to compute and store gradients. Since sequences can be rather long, the dependency can be rather lengthy. For instance, for a sequence of 1000 characters, the first token could potentially have significant influence on the token at the final position. This is not really computationally feasible (it takes too long and requires too much memory) and it requires over 1000 matrix products before we would arrive at that very elusive gradient. This is a process fraught with computational and statistical uncertainty.

- Backpropagation through time is merely an application of backpropagation to sequence models with a hidden state.
- Truncation is needed for computational convenience and numerical stability, such as regular truncation and randomized truncation.
- High powers of matrices can lead to divergent or vanishing eigenvalues. This manifests itself in the form of exploding or vanishing gradients.
- For efficient computation, intermediate values are cached during backpropagation through time.

Disadvantages of RNN:

- Problem of long term dependencies:vanishing gradients or exploding gradients. Solution: gated cells e.g: LSTM, GRU, etc.
- Problem of Encoding bottleneck, slow and no parallelization, not long memory. Solution: ATTENTION !!!

2 LSTM

LSTM or Long Short Term Memory are networks which improve on RNNs; standard RNNs have a short-term memory. During the Backpropagation Through Time in RNN's, long-term gradients can either explode (meaning they tend to infinity) or vanish (they tend to 0).

The problem of exploding gradients can be solved by truncation of such gradients.

The case of vanishing gradients can be solved by combining LSTM with RNNs. Vanishing gradients will cause the model to stop learning, or results will take too long. LSTM overcomes this simply by storing vital early information in a memory cell and skipping useless information. In LSTM, we extend memory which could be considered as a gated cell that decides whether to store information or delete it. There are three types of gates in LSTM:

- **Input gates**: determine whether or not to read data into cell
- **Forget gates**: delete the unimportant information and reset cell content
- **Output gates**: impact the output at the current timestep

Gates in an LSTM are represented by sigmoids, therefore their ouput ranges in $(0, 1)$.

2.1 Architecture

The architecture of LSTM is shown in figure 1.The hidden layer output of LSTM includes the hidden state and the memory cell. Only the hidden state is passed into the output layer. The memory cell, represented in green, is entirely internal.

The horizontal line running through top of the diagram is the cell state, in which the LSTM removes or adds information, and is regulated by gates, represented along the bottom. Gates are composed of a sigmoid neural net layer and pointwise multiplication. The sigmoid layer outputs numbers between 0 and 1, describing how much of each component should be let through. A value of 0 means "let nothing through," while a value of 1 means "let everything through!".

Let's explore what happens as information passes through each LSTM cell. We will denote with h_{t-1} the output or control signal of the cell, which roughly corresponds to the short term memory coming from the previous cell; with C_t the cell state which carries long-term memory; with x_t the input; with b_k any bias term.

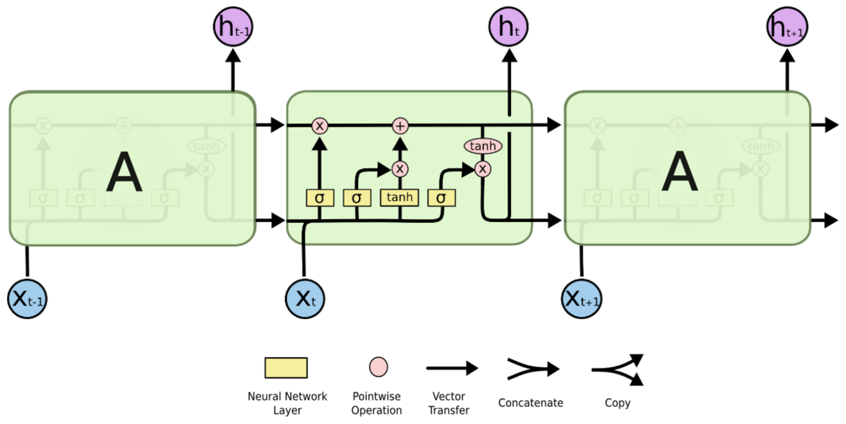


Figure 1: Diagram of the architecture of LSTM models

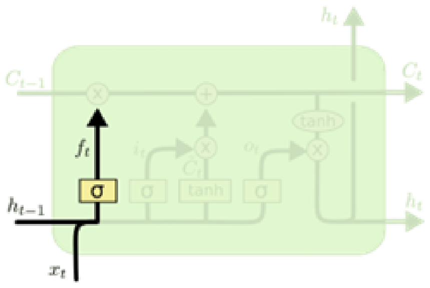


Figure 2: Step 1

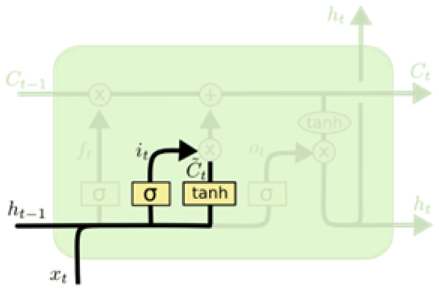


Figure 3: Step 2

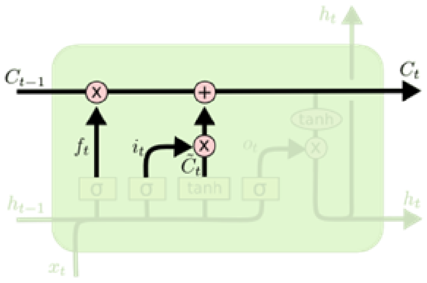


Figure 4: Step 3

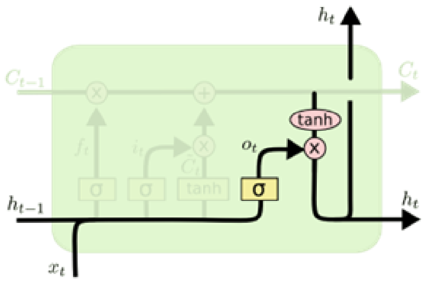


Figure 5: Step 4

1. (Figure 2) Decide what information to delete from the cell state via the **forget gate** sigmoid layer:

   $$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

   The gate inspects $h_t - 1$ context and x_t input and outputs a number between 0 and 1 in cell state C_{t-1} to indicate whether to forget this or keep this respectively

2. (Figure 3) New information to be stored in the cell state goes through a sigmoid layer, which decides which values to update, and a tanh layer, which creates a new candidate value $\tilde{C}_t$:

   $$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

   $$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C))$$

3. (Figure 4) The old cell state C_{t-1} is updated to new cell state C_t. We multiply the old state by f_t, forgetting the things we decided to forget earlier. Then we add $i_t * \tilde{C}_t$: this is the new candidate value, scaled by how much we decided to update each state value.

   $$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

4. (Figure 5) The output is determined based on the h_{t-1}, by running a sigmoid layer deciding what parts of cell state to output. It is also controlled by C_t through a *tanh* function, which makes the values range from -1 to 1. The two contributions are multiplied.

   $$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$

   $$h_t = o_t * \tanh(C_t)$$

3 BiLSTM

A Bi-directional LSTM or BiLSTM is a combination of two LSTM Networks. One runs forward, from left to right, meaning past to future; the other runs backward, from right to left, meaning future to past. Thus the entire essence/context of the sentence is captured. Input flow in both directions allows the network to preserve the future and the past information.

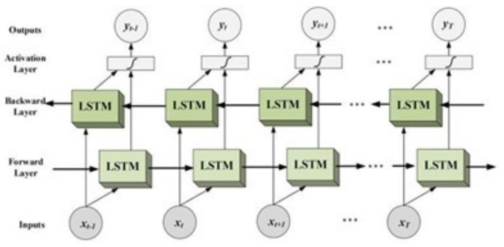


Figure 6: Bi-directional LSTM architecture

This kind of network can be used in text classification, speech recognition and forecasting models. The main reason is that every component of an input sequence has information from both the past and present. For this reason, BiLSTM can produce a more meaningful output, combining LSTM layers from both directions. BiLSTM will have a different output for every component (word) of the sequence (sentence).

4 Word Embeddings: word2vec and GloVe

We input a list of words to the trained language model and predict the following word. The output is not the word but the words with different probabilities.

- Look up embeddings
- Calculate prediction
- Project to output vocabulary

Word embeddings Word embeddings is the technique to create a numerical representation of a word. We don't want to just read or process words, but rather build a relationship between each word in a sentence or document with the other words. They capture the context of previous sentence or paragraph along with semantic and syntactic properties and similarities. They are vectors in space that cluster together similar words.
Word embeddings are either frequency based or prediction based. Frequency based: focuses on how much a word occurs in a given text

1. Word count
2. TF-IDF term frequency-inverse document frequency
3. Co-occurrence

Prediction based:

1. CBOW: continuous bag of word
2. Skip-gram

We use embeddings that are pretrained on vast amounts of text data instead of training them alongside the model on what is usually a small dataset. Therefore, it is possible to download a list of words and their embeddings generated by pre-training with word2vec or GloVe.

Word2vec Use a vector (list of numbers) to properly represent words in a way that capture semantic meaning related relations. We want to find if two words are similar or opposite. We can use cosine similarity metric. If the cosine of the dot product of 2 vectors representing 2 words is 0 then the words overlap(similar). If the cosine of the dot product of the 2 vectors representing 2 words is 1 then the words are independent. Similar words have similar vector representations that is captured by the cosine similarity. We can also find if a

pair of words have a similar relationship that another 2 pair of words have. Also we can get syntactic or grammatically based relationships.
It uses 2 models: CBOW and skip-gram.
CBOW We input various words (n words before target and n words after) and try to predict the target word. The target word supposedly is closely related to the context of the input words. CBOW is fast and finds a better numerical representation of frequent words and is less complex.
Skip gram We input one word and try to predict its closely related context. We predict the surroundings of a given word. It could represent rare words efficiently and is more complex than CBOW.
Word2vec can capture semantic relations among words: Cairo to Egypt is Rome to Italy. It is also suitable for semantic analysis.

GloVe Global Vector for word representation is an unsupervised learning technique to obtain vector representation for words. It is an extension of word2vec. Training is performed on aggregated global word to word co-occurrence statistics from a corpus. The resulting representation showcases interesting linear substructure of the word vector space.
Word-word cooccurrence: A word that is likely to occur with another word more than other words.
Word2vec can only capture the local context of word; it uses neighboring words to capture context. GloVe considers the entire corpus and creates a large matrix that captures the co-occurrence of words within corpus. It combines advantages of 2 word vector learning methods:

1. Matrix factorization on word context matrix
2. Local context window like skip gram model

It has a simpler least square cost of error function that reduces the computational cost of training the model. Disadvantages of techniques like word2vec or GloVe is that we have the same vector for the same word regardless of their meaning in the sentence. Many words have multiple meanings depending on where it is used. So why not include its context when doing the embedding. This will allow to capture the word meaning in that context as well as other contextual information. CONTEXTUALIZED WORD EMBEDDINGS...

5 ELMo

ELMo, or Embeddings from Language Models, is a type of deep contextualized word representation.

In NLP tasks, we face the problem of word embeddings being applied in a context-free manner. This is solved by training contextual representations on text instead. The representation differs from traditional word type embeddings in that each token is assigned a representation that is a function of the entire input sentence. We use vectors derived from a bidirectional language model (biLM), a bidirectional LSTM that is trained with a coupled language model (LM) objective trained on a large text corpus. Therefore, the same word can have different word vectors under different contexts supporting polysemy, meaning that a word can have different meanings or senses.

ELMo representations are:

- **Contextual**: the representation for each word depends on the entire context in which it is used
- **Deep**: the word representations combine all layers of a deep pre-trained neural network. A linear combination of vectors stacked above each input layer is learned instead of just using the top LSTM layer.
- **Character based**: ELMo representations are purely character based, allowing the network to use morphological clues to form robust representations for out-of-vocabulary tokens unseen in training.

5.1 Architecture of ELMo

ELMo word vectors are computed on top of a two-layer bidirectional language model. This biLM model has two layers stacked together. Each layer has two passes — forward pass and backward pass.

The raw word vectors act as inputs to the first layer of biLM. The forward pass contains information about a certain word and the context (other words) before that word; the backward pass contains information about the word and the context after it.

This pair of information terms, from the forward and backward pass, form the intermediate word vectors, which are fed into the next layer of biLM. The final ELMo representation is the weighted sum of the raw word vectors and the two intermediate word vectors.

To train ELMo:

- Train Separate Left-to-Right and Right-to-Left Language Models; this uses all layers of language models
- Apply as "Pre-trained Embeddings", learned task-specific combinations of layers
- Once the training is completed, we can use these pre-trained embeddings and apply them on similar data; this technique is called a *transfer learning*

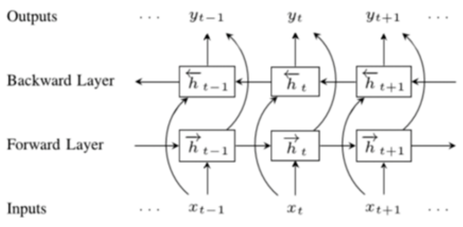


Figure 7: The two LSTM in ELMo

The biLM produces $2L + 1$ intermediate representations:

$$\begin{aligned} R_k &= \{x_k^{LM}, \overrightarrow{h}_{k,j}^{LM}, \overleftarrow{h}_{k,j}^{LM} | j = 1, \ldots, L\} \\ &= \{h_{k,j}^{LM} | j = 0, \ldots, L\} \end{aligned} \tag{1}$$

where $h_{k,j}^{LM}$ is the token layer and $h_{k,j}^{LM} = [\overrightarrow{h}_{k,j}^{LM} + \overleftarrow{h}_{k,j}^{LM}]$, for each biLSTM layer.

ELMo is a task-specific combination of these features:

$$ELMo_k^{task} = E(R_k; \Theta^{task}) = \gamma^{task} \sum_{i=0}^{L} s_j^{task} h_{k,j}^{LM}$$

where s^{task} are softmax-normalized weights and γ^{task} is a scaling parameter.

Unlike traditional word embeddings, the ELMo vector assigned to a token or word is actually a function of the entire sentence containing that word. ELMo and BERT embeddings are context-dependent, i.e. these models output different vector representation (embeddings) for the same word depending on the context in which it is used.

5.2 Applications

There are a plethora of applications for ELMo; here we enumerate a couple of them.

Word sense disambiguation : given a sentence, we can use the biLM representations to predict the sense of a target word using a simple 1-nearest neighbor approach.

POS tagging : To examine whether the biLM captures basic syntax, we use the context representations as input to a linear classifier that predicts POS tags.

6 A brief recap of Attention

Human attention is a limited, valuable, and scarce resource. Subjects selectively direct attention using both the nonvolitional and volitional cues. The former is based on saliency and the latter is task-dependent. Attention mechanisms are different from fully-connected layers or pooling layers due to inclusion of the volitional cues. Attention mechanisms bias selection over values (sensory inputs) via attention pooling, which incorporates queries (volitional cues) and keys (nonvolitional cues). Keys and values are paired. We can visualize attention weights between queries and keys

Steps of attention:

1. Encode position information
2. Extract query, key, value for search
3. Compute attention weighting
4. Extract features with high attention

7 A brief recap of Transformers

A transformer model is a neural network that learns context and thus meaning by tracking relationships in sequential data like the words in this sentence.

Transformer models apply an evolving set of mathematical techniques, called attention or self-attention, to detect subtle ways even distant data elements in a series influence and depend on each other.

First described in a 2017 paper from Google, transformers are among the newest and one of the most powerful classes of models invented to date. They're driving a wave of advances in machine learning some have dubbed transformer AI.

Stanford researchers called transformers "foundation models" in an August 2021 paper because they see them driving a paradigm shift in AI. The "sheer scale and scope of foundation models over the last few years have stretched our imagination of what is possible," they wrote.

8 BERT

BERT was introduced for the first time in 2019, with the aim of producing a powerful model for Language Modeling and other natural language processing tasks. It builds on the powerful (language) Transformer architecture, and it implements bidirectionality in the model. BERT stands for Bidirectional Encoder Representation of Transformers, and it has two distinct phases:

- A pre-training phase
- A fine-tuning phase

Bidrectionality is injected into the pre-training phase.

BERT was not the first model which tried to improve on the standard left-to-right context of traditional encoder architectures: among others we highlight ELMo (see section 5), which is often compared to BERT. In addition to being just as expensive, this approach is strictly less powerful than a deep bidirectional model, since BERT can use both left and right context at every layer, not just posthumously.

8.1 Architecture

As an input, BERT requires a *sequence*, which can be either a sentence or a pair of sentences sampled such that the combined length is ≤ 512 tokens; this is crucial in allowing BERT to handle a variety of downstream tasks (e.g. Question Answering). The first token of a sequence is always the token [CLS]. The final hidden vector `C` of this token is also used for representation of the sequence for classification tasks (see section 8.3).

Each sentence in the sequence ends with the token [SEP]. This helps separate sentences, along with a (learned) segment embedding which denotes which sen-

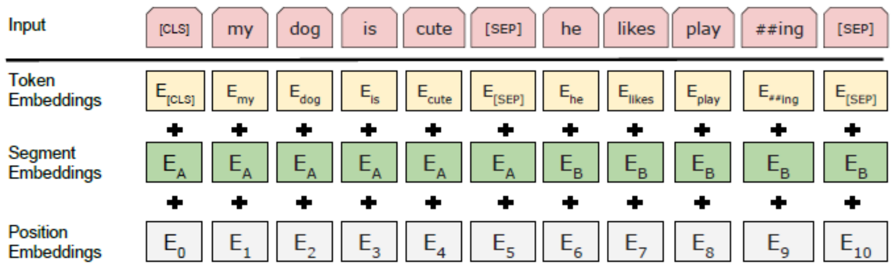


Figure 8: BERT embedding layer

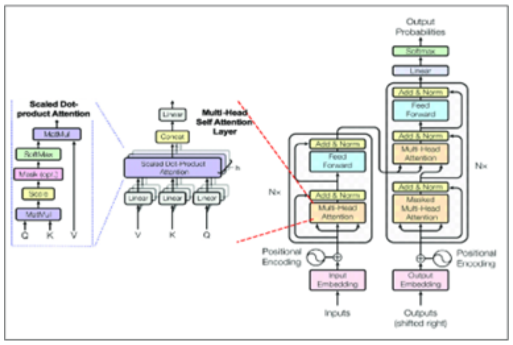


Figure 9: Traditional transformer architecture

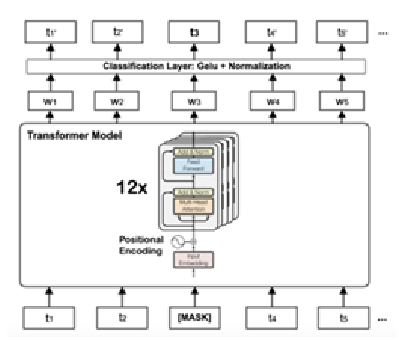


Figure 10: BERT architecture

tence each token belongs to. Overall, the input sequence undergoes 3 different embeddings:

segment embedding discussed above

position embedding denotes the absolute position of the word in the sentence, and is therefore an absolute position embedding. [Dev+18] made it fully learnable, however we note that there are several handcrafted PEs (Positional Embeddings) available

token embedding returns the position of the token in the dictionary

Segment and position embeddings are necessary to preserve the ordering, as input tokens are fed simultaneously into BERT; all of the parameters for the embeddings are transferred form pre-training to fine tuning

The architecture of BERT is almost identical to that of the now ubiquitous transformer architecture (figure 9), but it does away with the decoder part.

The paper presents two versions of BERT:

- BERT$_{\text{BASE}}$: Layers=12, Hidden size=768, Attention heads=12, Total Parameters=110M

- BERT$_{\text{LARGE}}$: Layers =24, Hidden size =1024, Attention heads =16, Total Parameters=340M

8.2 Pre-training phase

During the pre-training phase the network is fed large datasets (BooksCorpus -800M words [Zhu+15] - and English Wikipedia -2,500M words). The network is then trained for two different unsupervised tasks

MLM or Masked Language Modeling or Cloze task Part of the input tokens are replaced with an auxiliary [MASK] token, in a process called *masking*; we then want to train the model to predict the original input. Masking is what allows for bidirectionality: if we didn't mask the input then each word would be able to see itself, and therefore trivially predict the output in a layered setting.

When pre-training BERT 15% of the tokens are randomly selected for prediction; of these (1) 80% are replaced with the mask token, (2) 10% are replaced with a random token and (3) 10% remain unchanged. This masking scheme is important as there is no [MASK] token in fine-tuning, which creates a discrepancy between the two phases; this way the transformer encoder does not know which words it will be asked to predict or which have been replaced by random words, so it is forced to keep a distributional contextual representation of every input token.

This masking is static: the random masking and replacement is performed once at the beginning of training, though in practice data is often replicated so the same sentence may appear multiple times with different masking patterns.

NSP or Next Sentence Predictor The model is given two sentences A and B; B is the sentence following A, and is labeled as `IsNext`, with probability $p = 0.5$, else it's a random sentence and it's labeled as `NotNext`. The model learns if B follows A or not: this is a binary classification task. As mentioned the sequence of sentences is represented by `C`, the final hidden vector of the token [CLS], $\texttt{C} \in \{0, 1\}$. According to the original paper [Dev+18], the NSP task is necessary for understanding relationships between sentences, not captured by LM and required for many downstream tasks such as Natural Language Inference. In section 9 we see that this claim has been disputed.

The training loss is therefore the sum of the mean MLM likelihood and the mean NSP likelihood.

8.3 Fine-tuning phase

The fine-tuning phase inherits the pre-trained model parameters as initialized values of its own parameters: all hyperparameters with the except size, learning rate, and number of training epochs. A simple classification layer is added on top of the pre-trained model, and all parameters are jointly updated. A pooling layer takes the output representation corresponding to the first token and uses it for downstream tasks, applying a linear transformation over it. Downstream tasks (tasks for the fine-tuning phase) can be of many kinds; the sentence-pair input structure can be leveraged for a number of them (e.g. sentence pairs in paraphrasing, question-passage pairs in question answering).

Fine-tuning is much faster than pre-training, even for large datasets.

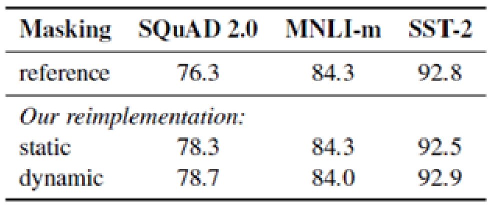
An alternative to fine-tuning is a feature-based approach, used for example by ELMo: instead of transferring pre-trained parameter values to downstream tasks, feature-based networks work by extracting representations as input features for the downstream task without touching the original models. [Dev+18] actually produced a feature-based BERT; the best performance was obtained by extracting the concatenation of the last four hidden layers parameters, and feeding it to a randomly initialized 2-layer 768-dimensional BiLSTM before the classification layer; this performed competitively with ELMo and, while not as performing as the fine-tuning approach, can be an alternative for task that benefit from such an approach.

8.4 Implementation and pre-training parameters

BERT is pre-trained with batch size of 256 sequences (128,000 tokens/batch) for approximately 40 epochs. Adam is used, with learning rate $\lambda = 1e-4$, $\beta_1 = 09$, $\beta_2 = 0.999$, L_2 weight decay of 0.01, learning rate warm up over the first 10,000 steps, and linear decay. On all layers dropout with $p = 0.1$ is implemented, and `GELU` activation is employed (figure 10).

9 RoBERTa

RoBERTa stand for Robustly optimized BERT approach; it is a model based on BERT (BERT$_{\text{LARGE}}$ specifically), and it claims to outperform the original BERT architecture through a series of simple but significant design choices when it comes to the pre-training phase; the base architecture is largely unchanged. These changes are: dynamic masking, removal of the NSP task, larger training batches and a larger vocabulary.



Masking	SQuAD 2.0	MNLI-m	SST-2
reference	76.3	84.3	92.8
Our reimplementation:			
static	78.3	84.3	92.5
dynamic	78.7	84.0	92.9

Figure 11: Static vs dynamic masking performance, from [Liu+19a]

Dynamic masking (vs static masking) The masking is generated every time a sentence is fed to the model;

[Liu+19a] claims a slight improvement with respect to static masking implemented by [Dev+18], as shown in figure 11.

No NSP task It has been observed the NSP loss may not be as fundamental to generalization as was postulated in [Dev+18]; the authors of the roBERTa paper show that removing the NSP loss can slightly improve downstream task performance, though their results on QNLI (the most significantly impacted, according to [Dev+18]) are not reported.

Larger batches Batch size was increased from 256 to 2K sequences, decreasing the number of steps from 1M to 125K; this has roughly the same computational cost as the original implementation, and provides a few advantages. Large batch training has been shown to improve optimization speed and end-task performance; the learning rate, however, has to be increased appropriately ([Ott+18]) . In addition, large batches are much easier to parallelize.

Overall the effect is significant enough that the model is also trained with a batch size of 8K for 31K steps, producing a better performance the same computational cost.

Larger BPE vocabulary When tokenizing and creating the embedding we reference a vocabulary of tokens. BPE (Byte-Pair Encoding) is the tokenization algorithm leveraged by BERT: instead of full words, BPE relies on subwords units extracted by statistical analysis of the training corpus. BERT uses a BPE vocabulary of 30K tokens using Unicode characters as subword units, and relies on WordPiece (a variant of BPE) for tokenization. RoBERTa uses BPE over raw bytes instead of Unicode characters; this produces a much larger vocabulary, which is then trimmed to 50K. This change adds 15M parameters to the model (20M for BERT$_{\text{LARGE}}$), and the model can perform slightly worse on certain task; the upside is that this is a universal encoding scheme.

10 DeBERTa

DeBERTa (Decoder Enhanced BERT with disentangled Attention) was developed after RoBERTa, which it mostly uses as a starting point. It employs three different techniques: a disentangled attention mechanism, an enhanced mask decoder, and an adversarial training method.

Disentangled attention mechanism BERT represents the tokens in a single vector, which encodes both content and positional information through the embedding; deBERTa on the other hand represents a token in position i in the sentence using two vectors, $\{H_i\}$ and $\{P_{i|j}\}$, which represent its content and relative position with the token at position j respectively.

Attention weights among words are computed using disentangled matrices based on their contents and relative positions:

$$\begin{aligned} A_{i,j} &= \{H_i, P_{i|j}\} \times \{H_j, P_{j|i}^T\} \\ &= H_i H_j^T + H_i P_{j|i}^T + P_{i|j} H_j + P_{i|j} P_{j|i}^T \end{aligned} \tag{2}$$

The attention weight of a word pair can therefore be computed as a sum of four attention scores: *content-to-content*, *content-to-position*, *position-to-content*, and *position-to-position.* [Liu+19a] highlight the importance of both *content-to-position* and *position-to-content* scores, as the content of the token influences it's relative position in the sentence and vice versa; they also argue that using relative positional embedding it's possible to capture *content-to-position* information, but not *position-to-content.* The *position-to-position* term is removed as it's redundant: the model uses relative positional embeddings (as opposed to the absolute positional embedding used in the original BERT formulation).

Let's see how to compute these factors in greater detail. We recall that the standard (single-head) self-attention operation, with attention matrix A, is computed as

$$Q = HU_q; K = HU_k; V = HU_v; A = \frac{QK^T}{\sqrt(d)}$$
$$H_o = softmax(AV)$$

with H matrix of input hidden vectors; H_o output of self-attention; U_q, U_k, U_v projection matrices; Q, K, V matrices of queries, keys and values respectively.

In addition to H, we now consider the matrix P of the relative position embedding. We also define the relative distance $\delta(i, j) \in [0, 2k)$ from token i to token j:

$$\delta(i,j) = \begin{cases} 0 & \text{for } i - j \leq -k \\ 2k - 1 & \text{for } i - j \geq k \\ i - j + k & \text{otherwise} \end{cases} \tag{3}$$

We can then represent the disentangled self-attention operation, with self attention matrix, as:

$$Q^c = HU_{q,c}; K^c = HU_{k,c}; V^c = HU_{v,c}; Q^r = PU_{q,r}; K^r = PU_{k,r}$$

$$\tilde{A}_{i,j} = \underbrace{Q_i^c K_j^{cT}}_{\text{content-to-content}} + \underbrace{Q_i^c K_{\delta(i,j)}^{rT}}_{\text{content-to-position}} + \underbrace{K_j^c Q_{\delta(j,i)}^{rT}}_{\text{position-to-content}}$$

$$H_o = softmax(\frac{\tilde{A}}{\sqrt{3d}})V^c$$

$\tilde{A}_{i,j}$ is the element of attention matrix $\tilde{A}$, representing the attention score from token i to token j. Q^c, K^c and V^c are the projected *content* vectors generated using projection matrices $U_{q,c}$, $U_{k,c}$, $U_{v,c}$ respectively: Q^c content query, K^c content key, V^c content value; Q_r and K_r are the projected relative *position* vectors generated using projection matrices $U_{q,r}$ and $U_{k,r}$ respectively: Q^r position query, K^r position key. The subscript indices are row ideces. Note that in the position-to-content term we use $Q^{rT}_{\delta(j,i)}$ rather than $Q^{rT}_{\delta(i,j)}$; this is because for a given position i, the position-to-content term computes the attention weight of the key content at j with respect to the query position at i, note vice versa. . Note that this disentangled attention is in principle very computationally expensive; [Liu+19a] therefore provide an algorithm that efficiently computes the matrices involved. This algorithm removes the need to allocate memory to store a relative position embedding for each query and thus reduce the space complexity to $O(kd)$ from the original $O(N^2d)$.

Enhanced mask decoder BERT encapsulates the information regarding the absolute position of the token in the sentence thanks to the embedding layer. In DeBERTa, this information is not incorporated at the beginning, as there is no absolute embedding; it's incorporated instead right after all the transformer layers and before the *softmax* layer for masked token prediction, as shown in Figure 12, through a component called Enhanced Mask Decoder (EMD).

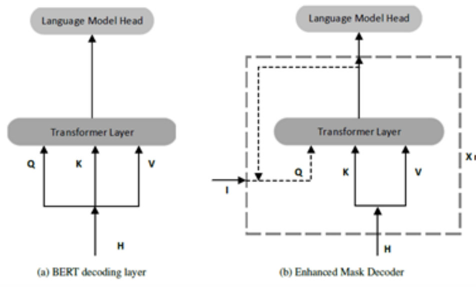


Figure 12: Enhanced Mask Decoder in the RoBERTa architecture

In this way, DeBERTa captures the relative positions in all the Transformer layers and only uses absolute positions as complementary information when decoding the masked words. A comparison between the two approaches shows that EMD works much better.

SiFT or Scale-invariant fine-tuning Scale-invariant fine-tuning is a new virtual adversarial training algorithm introduced by the paper ([He+20]).

The aim of adversarial training is improving the model robustness to adversarial examples, which are examples created by small perturbations to the embedded input token; this is essentially a form of regularization. The model is regularized so that, when given a task-specific example, it produces the same output distribution as is produced on an adversarial perturbation of that same example.

These algorithms are subject to instability, however, as the size of the model grows, because the variance of the embedding grows. To improve on this issue SiFT first normalizes the word embedding vectors into stochastic vectors, and then applies the perturbation to the normalized embedding vectors. The authors find that the normalization substantially improves the performance of the fine-

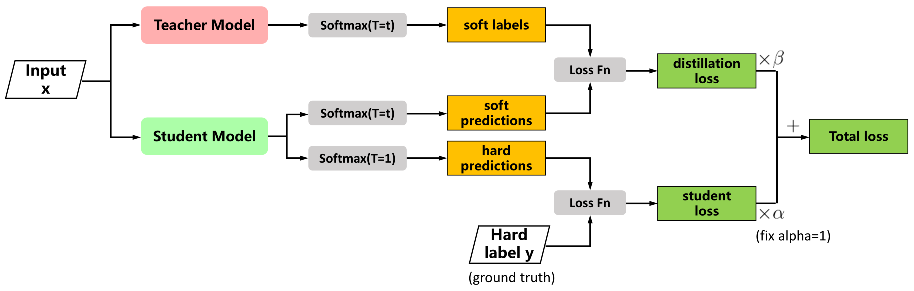


Figure 13: Distillation and student loss

tuned models.

Following RoBERTa, DeBERTa trains with large batches and implements dynamic masking.

11 DistilBERT

It has been shown that often in the field of Machine Learning bigger models are more accurate, however there is a trade-off between accuracy and cost. Quoting [XWD20], *when performing downstream tasks, ask: Is the extra iota of performance worth the significant costs of compute?* Though BERT is not a huge model by modern standards, there is value in proposing a lighter model. We explore distilBERT, one of the most popular lightweight BERT implementations.

DistilBERT leverages a technique called Knowledge distillation, which allows it to learn from the larger BERT.

11.1 Knowledge distillation

Knowledge distillation is a form of model compression through which a small "student" model is trained to mimic a pre-trained larger "teacher" model or ensemble of models, rather than trained on it's own. To do this we introduce a special loss function: the **distillation loss**. The distillation loss measures the difference between the class probabilities predicted by the teacher model and the probabilities predicted by the student. These probabilities are usually the output of a *softmax* function (the last layer of the teacher network); the standard *softmax* however leaves behind information regarding which classes are more probable: in many cases the probability distribution has the correct class at a very high probability with all other class probabilities very close to 0, and therefore it mirrors the ground truth labels already provided in the dataset. To improve information retention we introduce a more general *softmax-temperature* definition, where the probability of class c_i is

$$p_{i,T} = \frac{\exp(\frac{z_i}{T})}{\sum_j \exp(\frac{z_j}{T})}$$

where z_i are the logits and T is is the temperature, a parameter directly proportional to how soft the probability is, meaning classes similar to the predicted class get higher probabilities. The distillation loss is therefore defined as the cross-entropy loss between the outputs $t_{i,T}$ of the theacher models and the outputs $s_{i,T}$ of the student models, both obtained with softmax-temperature with temperature $T = t > 1$:

$$L_d = \sum_i t_{i,t} \dot{\log}(s_{i,t})$$

In addition to the distillation loss the small model is also trained on a **student loss**: the cross-entropy loss between the outputs s_i of the student model obtained with a standard softmax ($T = 1$) and the labels y_i

$$L_s = \sum_i y_i \log(s_i)$$

The overall loss on which the student model is trained is therefore

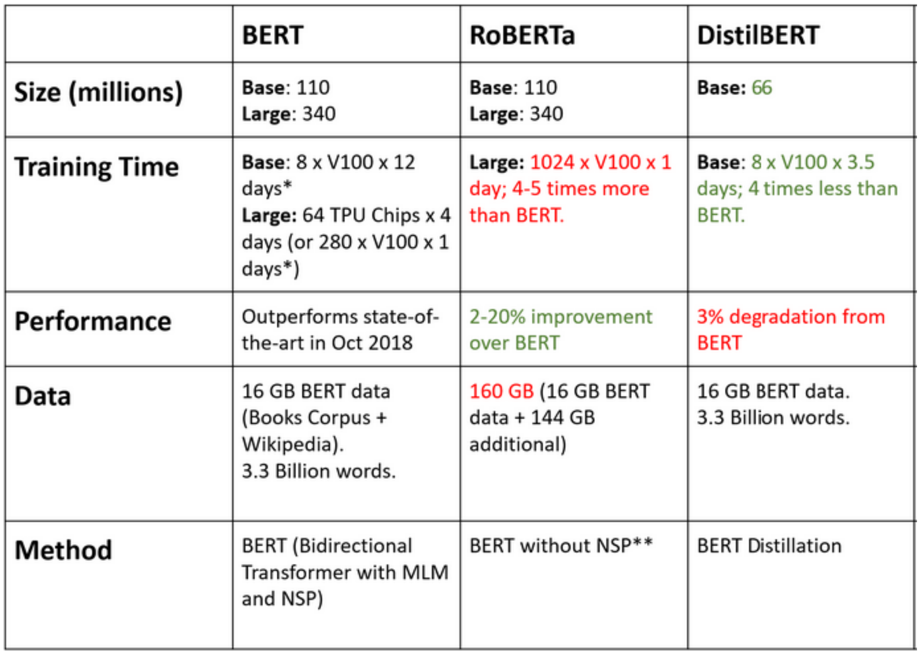
$$L = \alpha L_d + \beta L_s \qquad \alpha, \beta \in \mathbb{R}$$

DistilBERT is trained on a **triple loss**: a linear combination of the distillation loss, the masked language training loss and a cosine embedding loss (L_{cos}), the purpose of which is to align the directions of the student and teacher hidden states vectors.

Architecture The authors find experimentally that number of layers has the greatest impact on computational efficiency; therefore layers are decreased by a factor of 2 with respect to BERT. Moreover the token_type embeddings and the pooler layer are removed.

The initialized values are also important when training with Knowledge Distillation: distilBERT is initialized by taking every other layer from BERT.As a matter of fact, ablation studies show that random weight initialization has the greatest negative impact on student performance, followed by removal of distillation loss; the student loss has the least impact. Following considerations illustrated by [Liu+19a], the model is trained with very large batches, using dynamic masking and without the NSP task as an objective.

DistilBERT has 40% fewer parameters than BERT$_{\text{BASE}}$, only 66M (instead of 110M), and is 60% faster (71% faster on mobile devices). Despite this, it outperforms ELMo on the GLUE benchmark and retains 97% of the capabilities of BERT. It also achieves comparable scores to BERT with respect to some sample downstream tasks.



	BERT	RoBERTa	DistilBERT
Size (millions)	**Base**: 110 **Large**: 340	**Base**: 110 **Large**: 340	**Base:** 66
Training Time	**Base**: 8 x V100 x 12 days* **Large:** 64 TPU Chips x 4 days (or 280 x V100 x 1 days*)	**Large:** 1024 x V100 x 1 day; 4-5 times more than BERT.	**Base**: 8 x V100 x 3.5 days; 4 times less than BERT.
Performance	Outperforms state-of-the-art in Oct 2018	2-20% improvement over BERT	3% degradation from BERT
Data	16 GB BERT data (Books Corpus + Wikipedia). 3.3 Billion words.	160 GB (16 GB BERT data + 144 GB additional)	16 GB BERT data. 3.3 Billion words.
Method	BERT (Bidirectional Transformer with MLM and NSP)	BERT without NSP**	BERT Distillation

Figure 14: performance comparison of BERT vs RoBERTa vs DistilBERT

References

[Wer90] P.J. Werbos. "Backpropagation through time: what it does and how to do it". In: *Proceedings of the IEEE* 78.10 (1990), pp. 1550–1560. DOI: `10.1109/5.58337`.

[Kar15] Andrej Karpathy. *The Unreasonable Effectiveness of Recurrent Neural Networks*. 2015. URL: `http://karpathy.github.io/2015/05/21/rnn-effectiveness/` (visited on 06/24/2022).

[Zhu+15] Yukun Zhu et al. *Aligning Books and Movies: Towards Story-like Visual Explanations by Watching Movies and Reading Books*. 2015. DOI: `10.48550/ARXIV.1506.06724`. URL: `https://arxiv.org/abs/1506.06724`.

[Vas+17] Ashish Vaswani et al. *Attention Is All You Need*. 2017. DOI: `10.48550/ARXIV.1706.03762`. URL: `https://arxiv.org/abs/1706.03762`.

[Dev+18] Jacob Devlin et al. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2018. DOI: `10.48550/ARXIV.1810.04805`. URL: `https://arxiv.org/abs/1810.04805`.

[Ili+18] Suzana Ilić et al. *Deep contextualized word representations for detecting sarcasm and irony*. 2018. DOI: `10.48550/ARXIV.1809.09795`. URL: `https://arxiv.org/abs/1809.09795`.

[Ott+18] Myle Ott et al. *Scaling Neural Machine Translation.* 2018. DOI: 10.48550/ARXIV.1806.00187. URL: https://arxiv.org/abs/1806.00187.

[Agu+19] Gustavo Aguilar et al. *Knowledge Distillation from Internal Representations.* 2019. DOI: 10.48550/ARXIV.1910.03723. URL: https://arxiv.org/abs/1910.03723.

[Liu+19a] Yinhan Liu et al. *RoBERTa: A Robustly Optimized BERT Pretraining Approach.* 2019. DOI: 10.48550/ARXIV.1907.11692. URL: https://arxiv.org/abs/1907.11692.

[Liu+19b] Yong Liu et al. *An Evaluation of Transfer Learning for Classifying Sales Engagement Emails at Large Scale.* 2019. DOI: 10.48550/ARXIV.1905.01971. URL: https://arxiv.org/abs/1905.01971.

[San+19] Victor Sanh et al. *DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter.* 2019. DOI: 10.48550/ARXIV.1910.01108. URL: https://arxiv.org/abs/1910.01108.

[Wan+19] Mengting Wan et al. *Fine-Grained Spoiler Detection from Large-Scale Review Corpora.* 2019. DOI: 10.48550/ARXIV.1905.13416. URL: https://arxiv.org/abs/1905.13416.

[Hat20] Hans Ole Hatzel. "Using Neural Language Models to Detect Spoilers". MA thesis. Universitat Hamburg, 2020, p. 78.

[He+20] Pengcheng He et al. *DeBERTa: Decoding-enhanced BERT with Disentangled Attention.* 2020. DOI: 10.48550/ARXIV.2006.03654. URL: https://arxiv.org/abs/2006.03654.

[Min+20] Shervin Minaee et al. *Deep Learning Based Text Classification: A Comprehensive Review.* 2020. DOI: 10.48550/ARXIV.2004.03705. URL: https://arxiv.org/abs/2004.03705.

[SM20] Harsh Singh and Qusay Mahmoud. "NLP-Based Approach for Predicting HMI State Sequences Towards Monitoring Operator Situational Awareness". In: *Sensors* 20 (June 2020), p. 3228. DOI: 10.3390/s20113228.

[XWD20] Patrick Xia, Shijie Wu, and Benjamin Durme. "Which *BERT? A Survey Organizing Contextualized Encoders". In: Jan. 2020, pp. 7516–7533. DOI: 10.18653/v1/2020.emnlp-main.608.

[Zha+20] Aston Zhang et al. *Dive into Deep Learning.* https://d2l.ai. 2020.

[BHS21] Allen Bao, Marshall Ho, and Saarthak Sangamnerkar. *Spoiler Alert: Using Natural Language Processing to Detect Spoilers in Book Reviews.* 2021. DOI: 10.48550/ARXIV.2102.03882. URL: https://arxiv.org/abs/2102.03882.

[CCG21] Roberto Cahuantzi, Xinye Chen, and Stefan Güttel. *A comparison of LSTM and GRU networks for learning symbolic sequences*. 2021. DOI: 10.48550/ARXIV.2107.02248. URL: https://arxiv.org/abs/2107.02248.

[Cha+21] Buru Chang et al. *"Killing Me" Is Not a Spoiler: Spoiler Detection Model using Graph Neural Networks with Dependency Relation-Aware Attention Mechanism*. 2021. DOI: 10.48550/ARXIV.2101.05972. URL: https://arxiv.org/abs/2101.05972.

[KBA21] Usama Khalid, Mirza Omer Beg, and Muhammad Umair Arshad. *RUBERT: A Bilingual Roman Urdu BERT Using Cross Lingual Transfer Learning*. 2021. DOI: 10.48550/ARXIV.2102.11278. URL: https://arxiv.org/abs/2102.11278.

[Sin21] Aastha Singh. *Evolving with BERT: Introduction to RoBERTa*. 2021. URL: https://medium.com/analytics-vidhya/evolving-with-bert-introduction-to-roberta-5174ec0e7c82 (visited on 06/20/2022).

[Wan+21] Benyou Wang et al. "On Position Embeddings in {BERT}". In: *International Conference on Learning Representations*. 2021. URL: https://openreview.net/forum?id=onxoVA9FxMw.

[Sin+22] Arjun Singh et al. "Evolving Long Short-Term Memory Network-Based Text Classification". In: *Computational Intelligence and Neuroscience* 2022 (2022), p. 11. DOI: https://doi.org/10.1155/2022/4725639.