5. Deep Learning Methods

One of the limiting factors of classical models is their reliance on explicit feature extraction procedures. Good feature engineering requires extensive domain knowledge, which in turn, makes it difficult to generalise approaches to new tasks. Furthermore, manual feature crafting does not utilise the full potential of large amounts of training data because of how features are predefined rather than discovered.

As a consequence, the development of word embeddings marks the beginning of a paradigm shift towards approaches able to leverage vast amounts of data. Deep learning approaches have gained popularity thanks to their ability to capture meaningful latent information in the training data (depicted in Figure 2); in the case of NLP, such architectures are able to create semantically meaningful representations of text. Recently developed contextualized versions of word embeddings have obtained outstanding results in classification, even when paired with very simple classifiers. An informal yet intuitive explanation of this result is that understanding the content of a body of text is the most important step in the classification pipeline, much like a person would likely be able to label a piece of text if they understood what it meant.

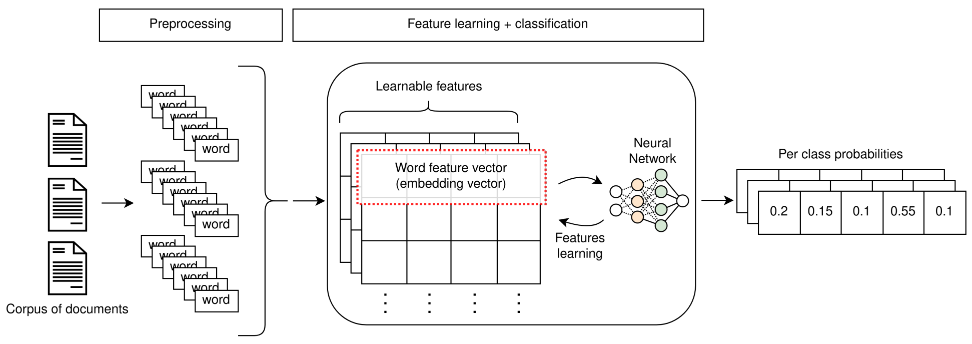

Figure 2. Overview of the training procedure used with deep learning methods.

5.1. Multilayer Perceptrons

Simpler architectures, such as multilayer perceptrons (MLPs), bridge the gap between shallow and deep methods. These are some of the most basic neural network designs, yet they are the foundation of the first word embedding techniques, while also obtaining excellent results as stand-alone classifiers. MLP models such as these usually treat input text as an unordered bag-of-words, where input words are represented through some feature extraction technique (like TF-IDF or word embeddings). For example, deep averaging networks (DAN) [55] are able to perform comparably or better than much more sophisticated methods [3], despite ignoring the syntactic ordering of words. Paragraph-Vec [56] tries instead to incorporate such ordering and the contextual information of paragraphs by utilising an approach that is comparable with CBOW [57], leading to better performances than previous methods.

5.2. Recurrent Neural Networks

The most influential architectures of this period of time were, however, recurrent neural networks (RNNs). RNNs are a popular choice for any type of sequential data; these architectures are aimed at extracting information regarding the structure of sentences, thus capturing latent relationships between context words. The input of an RNN model is usually a sentence represented by a sequence of word embeddings, with its words entering the model one at a time. Long short-term memory networks (LSTMs) are the most popular variant of RNNs, as they address the gradient vanishing or exploding issues faced by standard RNN architectures [58]. Many TC approaches using LSTMs have been proposed, and we mention a few. Tree-LSTM [59] extends LSTMs to tree structures, rather than sequential chain structures, arguing that trees provide a more suitable representation for phrases. TopicRNN [60] integrates the capabilities of latent topic models to overcome the difficulties of RNNs on long-range dependencies. Both of these approaches exhibit improvements on baselines when applied to TC tasks (in particular, sentiment analysis). Howard and Ruder [61] propose the Universal Language Model Fine-tuning (ULMFiT), a recurrent architecture based on an LSTM network trained using discriminative fine-tuning, which allows the tuning of LSTMs using different learning rates in each layer. The Disconnected Recurrent Neural Network (DRNN), introduced by Wang [62], demonstrates the benefits of boosting the feature extraction capabilities of RNNs by incorporating position–invariance—a property attributed to CNNs—into a network based on gated recurrent units (GRU) [63]. In both cases, the proposed enhancements allow surpassing state-of-the-art results on several benchmark datasets. The introduction of bidirectionality in RNNs has also been proven beneficial [64], and has been applied to LSTMs, with notable results such as ELMo [65], a language modelling approach that relies on bidirectional LSTMs, and is one of the first milestones in the development of contextualized word embeddings.

5.3. Convolutional Neural Networks

Convolutional neural networks (CNNs) are commonly associated with computer vision applications, yet they have also seen applications in the context of NLP and TC specifically [66]. The easiest way to understand these approaches in this context is to examine their input, which also relies on word embeddings. While, generally speaking, RNNs input the words of a sentence sequentially, sentences in CNNs are instead presented as a matrix in which each row is the embedding of a word (therefore, the number of columns corresponds to the size of the embeddings). To make a comparison, in image-related tasks, convolutional filters usually slide over local patches of an image across two directions. Instead, filters in text-related tasks are most commonly made as wide as the embedding size, so that this operation only moves in directions that make sense sentence-wise, always considering the entire embedding for each word. In general, the main upside associated with CNNs is their speed and how efficient their latent representations are. Conversely, other properties that could be exploited while working with images, such as location invariance and local compositionality [67,68], make little sense when analysing text. Many approaches have been proposed, one of the most popular being TextCNN [69], a comparatively simple CNN-based model with a one layer convolution structure that is placed on top of word embeddings. More recently, the interest in CNN applications to TC tasks has been renewed thanks to the introduction of temporal convolutional networks (TCN) [70–72], which enhances CNNs with the ability to capture high-level temporal information. For instance, Conneau et al. [73] propose the Very Deep CNN classifier, a 29-layer CNN with skip connections, alternating temporal convolutional blocks to max pooling operations. They achieve state-of-the-art performance on reported TL and SA datasets. Duque et al. [74] modify the VDCNN architecture to lessen the resource requirements and fit mobile platform constraints, while retaining most of its performance.

5.4. Deep Language Models for Classification

In this section, we outline the theoretical notions that lead to the development of current deep language model-based approaches. Then, we proceed to discuss some of these models in more detail. Currently, very few other models are able to compete performance-wise with such classifiers (among them, we discuss the particularly interesting graph-based models).

5.4.1. RNN Encoder–Decoders

For many years, networks based on RNN-like architectures have dominated sequence transduction approaches. Researchers started pushing the boundaries of TC through recurrent language models (evolutions of classic word embedding techniques) and RNN-based encoder–decoder architectures [63,75] (Figure 3).

RNN encoder–decoder architectures are particularly important in this regard. As an example, consider a machine translation task; the input sequence may be a sentence in English, and the output sequences its translation in a different language. In this case, each word of the input sentence is fed to the encoder sequentially, such that at time step t, the model receives the new input word and the hidden state at time step $t-1$. The input is hence consumed sequentially, and the dependence on the output of the previous step should, in theory, allow RNNs to learn long- and short-term dependencies between words. The output of the encoder is the compressed representation of the input sequence, called "context". The decoder then proceeds to interpret the context and produce a new sequence of words (e.g., a translation in a different language) in a sequential manner, where every word depends on the output of the previous time step. With encoder–decoder networks, semantically and syntactically meaningful information is implicitly captured during the encoding phase (the context), which can then potentially be used for tasks such as text classification.

The main issue with this approach is that the encoder needs to compress all relevant information in a fixed-length vector. This was found to be problematic, especially for longer sentences, and it was reported that the performance of a basic encoder–decoder deteriorates rapidly as the length of an input sentence increases [76]. In addition, recurrent models have inherent limitations due to their sequential nature. Sequentiality precludes parallelisation, which means higher computational complexity. Longer sentences can run into memory constraints and, more crucially, are seen as RNNs' true bottleneck because of how the network tends to forget earlier parts of the sequence, making for an incomplete representation (an issue linked to the vanishing gradient problem) [77].

One of the solutions devised to solve the limitations of recurrent architectures was that of the attention mechanism [76,78]. This mechanism would eventually become an integral part of the Transformer architecture, which represents one of the most important milestones in the field of NLP. We denote the particularly effective language models of this era as "Deep" (deep language models, DLM for short), such as to distinguish them from earlier approaches and highlight their reliance on deep architectures. As a matter of fact, depth in these architectures has been proven to be extremely beneficial to their performance, while LSTM-based models were unable to gain much from a large increase in their size [79].

5.4.2. The Attention Mechanism

Attention was initially utilised as an enhancement to various architectures, and was meant to allow the learning process to focus on more relevant parts of input sentences (i.e., give them "attention"). As previously stated, seq2seq tasks have been traditionally approached with RNN-based encoder–decoder architectures, where both the encoder and the decoder are stacked RNN layers.

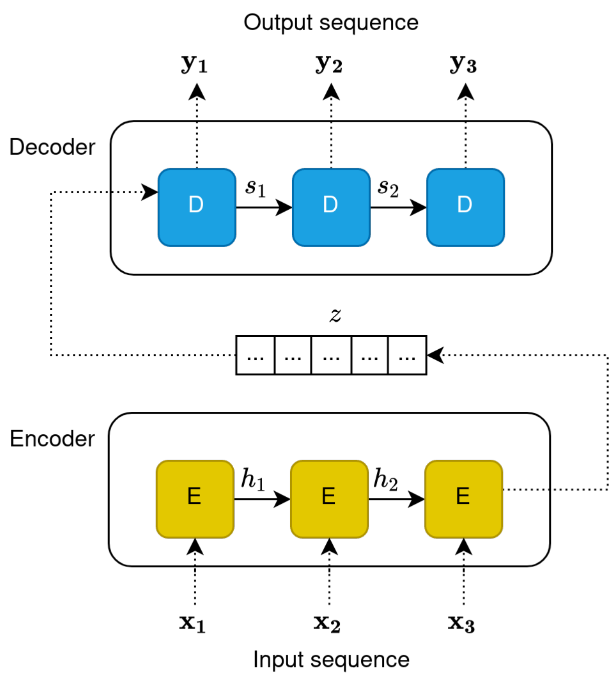

Figure 3. Unfolding of encoder–decoder architecture based on RNN.

The concept of attention was introduced by Bahdanau et al. [76] to mitigate this problem in neural machine translation tasks. The authors argued that, by propagating information about the complete input sequence, the decoder can discern between input words and learn which of them are relevant for generating the next target word. Formally, the attention mechanism relies on enriching the input context (z_i) of each decoder unit, with the hidden state of the encoder h_i (also called "annotation") that carries information on the whole input sequence. Figure 4 highlights the architecture of a simple encoder–decoder recurrent model enhanced with the attention mechanism. Equation (1) describes the computation of the attention score between the input word at position j and the generated word i. This score can be seen as an alignment score, measuring how well the two words match. In the equation, s_{i-1} denotes the previously generated word.

$$\alpha_{ij} = softmax(attention(s_{i-1}, h_j)) \tag{1}$$

The context of each decoder unit is then computed as in Equation (2), by multiplying the annotations of every word position (h_j) with the attention score generated by the attention model.

$$z_i = \sum_{j=1}^{N} \alpha_{ij} h_j \tag{2}$$

The mechanism described here is generally addressed as "additive attention". While there are different approaches to the integration of the attention mechanism in seq2seq architectures (content-based, dot-product, etc.), the goal remains to learn an alignment score that measures the relative importance between words in the input and output sequence.

Applications of attentive neural networks are now widespread, even beyond the boundaries of natural language processing, where attention initially proved its value. Notable examples of application in the text classification domain are hierarchical attention networks (HAN) [71,80]. These methods rely on the application of attention both at the word level, while encoding document sentences, and at the sentence level, roughly encoding the importance of each sentence with respect to the target sequence. Recently, Abreu et al. [71] have proposed a hybrid model (HAHNN) resorting to attention, along with temporal convolutional layers and GRUs.

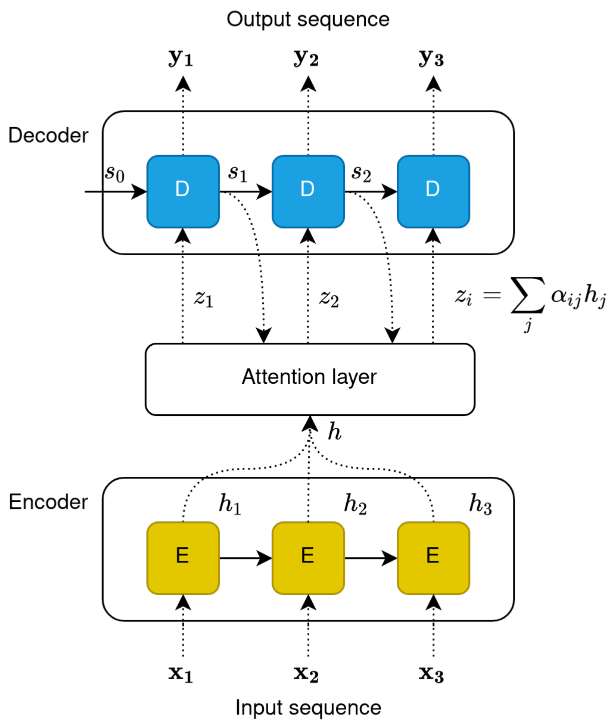

Figure 4. Unfolding of encoder–decoder architecture with attention mechanism.

However, attention has been since used as a strong foundational basis rather than just an added augmentation. The Transformer architecture is based on this idea, retaining a well-known encoder–decoder structure, but making no use of recurrence; instead, dependencies are drawn between input and output through the attention mechanism alone [81]. Transformers have been shown to lead to better results, while also gaining much in speed because of them being highly parallelisable.

5.4.3. The Transformer Architecture

Vaswani et al. [81] introduced the Transformer architecture, a novel encoder–decoder model that allows one to process all input tokens (e.g., words) simultaneously, rather than sequentially. Input sequences in Transformers are presented as a bag of tokens, without any notion of order. To learn dependencies between tokens, the Transformer relies on what is defined as a "self-attention" mechanism. Additionally, a special encoding step performed before the first layer of the encoder ensures that the embeddings for the same word appearing at a different position in the sentence will have a different representation. This step is called positional encoding, and its purpose is injecting information about the relative positioning between words, which would otherwise be lost.

The key component of this architecture is the self-attention layer, which intuitively allows the encoder to look at other words in the input sentence whenever processing one of its words. Stacking multiple layers of this type creates a multi-head attention (MHA) layer, as shown in Figure 5. The individual outputs are then condensed in a single matrix by concatenating the head outputs and passing the result through a linear layer.

Encoder

In the encoding part, the input embeddings are multiplied by three separate weight matrices (Equation (3)) to generate different word representations, which the authors name Q (queries), K (keys) and V (values), following the naming convention used in information retrieval.

$$Q = X \cdot W_Q, \qquad K = X \cdot W_K, \qquad V = X \cdot W_V \tag{3}$$

$W_Q, W_K, W_V \in \mathbb{R}^{dim \times d_k}$ are the learned weight matrices, $X \in \mathbb{R}^{N \times dim}$ are the word embeddings for the input sequence and $Q, V, K \in \mathbb{R}^{N \times d_k}$ contain the word representations. The product of the query and key matrices (Q and K) is taken to compute the self-attention score. With respect to the general attention mechanism introduced in the previous section, this can be seen as the alignment score between each word and the other word in the sentence, which depends on the relative importance between them. Moreover, the authors decided to use scaled dot-product attention instead of additive attention, mainly for efficiency reasons. In order to improve gradient stability, the score is divided by the square root of the representation length for each word, and a softmax is used to obtain a probability distribution. Eventually, the representation of each word is obtained by multiplying the scaled term with the V matrix containing the input representation. This operation is defined in Equation (4).

$$Z = \text{Attention}(Q, K, V) = softmax\left(\frac{Q \cdot K^T}{\sqrt{d_k}}\right) \cdot V \tag{4}$$

The Transformer multi-head self-attention layer performs these operations multiple times in parallel. According to the authors, this was done to extend the diversity of representation sub-spaces that the model can attend to. The output of the attention heads is concatenated and passed through a linear layer to obtain the final representation that condenses information from all the attention heads. This representation is then summed with residual input, normalised and passed to a feed-forward linear layer.

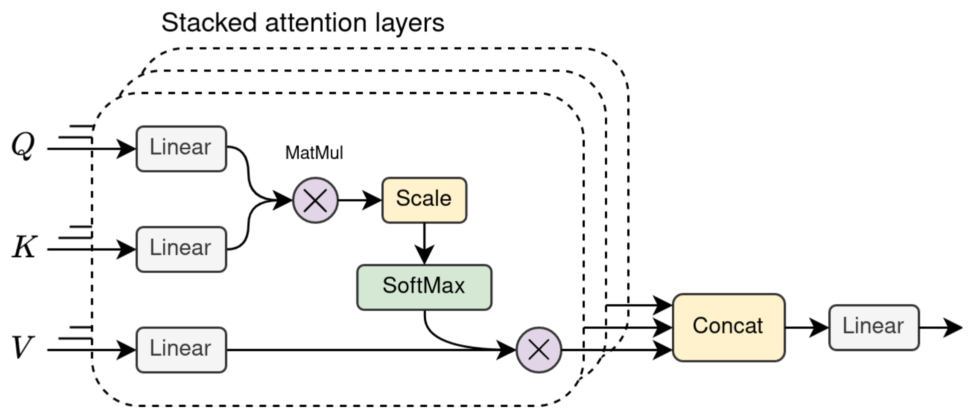

Figure 5. The multi-head attention layer used in the Transformer architecture.

Decoder

During the decoding phase, every decoder layer receives the output of the encoder (the K and V matrices) and the output of the previous decoder layer (or the target sequence for the first one). The key difference between the encoder and decoder layers is the presence, in the latter, of an additional MHA layer, applied to the K and V matrices from the encoder. Additionally, self-attention layers are modified into what are defined as “Masked” self-attention layers. During training, the Transformer decoder is initially fed the real target sequence—this is done to remove the necessity of having to be trained in an autoregressive fashion, hence becoming subject to the same bottleneck as RNNs (a procedure called “teacher forcing”). On the other hand, at inference time, the decoder is indeed autoregressive, since there are no target data and the generation of the next word necessarily depends only on the previously generated sequence. While predicting the token at position i, the masked MH self-attention layer ensures that only the self-attention scores between words at position $[1, i-1]$ are used. This is done by adding a factor M to the word embeddings in Equation (5). M is set to -inf for masked positions, and 0 otherwise. The exponential in the softmax operation will zero the attention scores for masked tokens.

$$Z = \text{MaskedAttention}(Q, K, V) = softmax\left(\frac{Q \cdot K^T + M}{\sqrt{d_k}}\right) \cdot V \tag{5}$$

This allows one to train the model in a parallel fashion, while ensuring that the model behaves consistently between training and inference time (without "cheating" at training time by looking ahead). The additional MHA layer of the decoder works like the one previously described, with the only exception being that the query matrix Q is the output of the preceding masked MHA layer. Residual connections and layer normalisation are also used in the decoder after each attention and feed-forward layer.

5.4.4. BERT and GPT

Recent DLMs are built upon and expand the Transformer architecture. The performance achievable on various NLP tasks, such as TC subtypes, has been boosted by what is considered the new standard in NLP transfer learning. This consists of utilising a pre-trained language model developed on generic textual data, which are subsequently specialised for the task at hand through a "fine-tuning" process.

The fine-tuning process consists of the adaptation of a model to a downstream task (i.e., classification). Classifiers are often just a small segment of the overall algorithm, yet its training procedure usually affects the pre-trained representation of the DLM as well. This is meant to specialise the overarching pipeline on the domain of the task, and usually leads to better results. Technically speaking, the internal representation of the pre-trained language model receives minimal changes from the fine-tuning process, which is a desirable outcome, as otherwise the DLM would incur an excessive loss of generality.

GPT

Although the original Transformer architecture is well suited for language modelling, researchers have argued that, for some tasks, it may learn redundant information in its encoding and decoding phases. It has been suggested that limiting the architecture to either encoders or decoders may result in equivalent performance and lighter models [82]. According to this principle, the Generative Pre-trained Transformer (GPT) [83] utilises a decoder-only architecture, which stacks multiple Transformer–decoder layers. The decoder block of a Transformer is autoregressive (AR), as it defines the conditional probability distribution of a target sequence, given the previous one. Simply put, GPT's pre-training task is that of next word prediction. Precisely because it is an autoregressive model, the GPT architecture produces a language model of which the predictions are only conditioned by either its left or right context (GPT's original architecture is left-to-right). This means that it is not bidirectional (a property that, for instance, BiLSTMs possess).

The decoder-only architecture has the additional benefit of performing better than traditional encoder–decoder models when dealing with longer sequences of text, making it particularly suitable for generative tasks (e.g., abstractive text summarisation and generative question answering). The authors of GPT provide adaptations for it to be used in a variety of downstream tasks, including classification.

BERT

Bidirectional Encoder Representations from Transformers (BERT) [16] closely followed the GPT model, and traded its auto-regressive properties in exchange for bidirectional conditioning. BERT's architecture is, in contrast with GPT, a multi-layer bidirectional Transformer encoder. BERT's training objective is also different, as it follows a "masked language model" (MLM) procedure. A percentage of the input is masked and, rather than predicting the next word, the network tries to predict masked tokens. This training strategy is crucial to achieving bidirectional conditioning, and allows the model to learn the relationship between masked words and their left and right context [84]. BERT is also trained on a secondary task, namely next sentence prediction (NSP), in which it is presented with two sentences and must guess in a binary fashion whether the second sentence follows the first. This task is meant to allow the model to better learn sentence relationships.

Most interestingly, the adaptation of BERT to downstream tasks is very simple. In fact, outstanding results have been obtained in classification by simply fine-tuning a model that passes the representations obtained by the encoders through a single-layer, feed-forward neural network [16].

5.4.5. Recent Transformer Language Models

Since the NLP landscape is constantly evolving, we deem it useful to provide a selection of a few recently proposed language models, and outline their main contributions and differences with previous methods. Most recent models build on the Transformer architecture or introduce variations on the BERT and GPT training approaches. Many are general language models jointly trained on multiple NLP tasks, and are evaluated on the GLUE [85] multi-task benchmark.

Direct Successors

BERT and GPT were just the starting point for the development of many variations and improvements, of which we cite the most influential. Robustly Optimised BERT Pretraining Approach (RoBERTa) [13] was introduced by Liu et al. as a successor of BERT. It improves the training procedure by introducing dynamic masking (masked tokens are changed during training epochs), while also removing the next sentence prediction task. The GPT model has also had successors, the most popular being the GPT-2 [12]. The development of GPT-2 models was mostly in terms of data utilised and the scale of the models (a trend that continued in GPT-3 [86]).

XLNet

As with GPT, XLNet [19] is an autoregressive model. More specifically, it is a generalised autoregressive pre-training approach, which calculates the probability of a word token conditioned on all permutations of word tokens in a sentence. To maintain information about the original relative ordering of words in the midst of all the permutations, XLNet borrows the idea of attention masking, by which, during context calculations, it masks tokens outside of the current context. As an example, consider an input sequence with four tokens and its permutation $P([1,2,3,4]) = [3,2,4,1]$. The first element has no contextual knowledge, and, to reflect this, the attention mask should forbid looking at the other sequence positions (producing mask $[0,0,0,0]$), while the second element should know about the first (namely, that it should be in position 3, giving mask $[0,0,1,0]$). XLNet also introduces a two-stream self-attention schema to allow position-aware word prediction. In other words, this grants the model the ability for token probabilities to be conditioned on their position index without trivially knowing if the word should be part of the sentence or not. XLNet's approach allows it to overcome the lack of bidirectionality of previous autoregressive models. Furthermore, this approach addresses the downsides of masked language modelling (namely, the discrepancy between pre-training and fine-tuning tasks), as it does not rely on data corruption. XLNet has achieved state-of-the-art performance on many TC benchmarks in English, as will be showcased in Section 6.3.

T5

The Text-To-Text Transfer Transformer (T5) [87] is a unified text-to-text model trained to solve a variety of NLP tasks. While general multi-task language models are mostly trained using task-specific architectural components and loss functions, T5's authors build a unified learning framework that casts every NLP problem as a text-to-text problem. This allows them to use the exact same model, loss function and hyperparameters to produce a single, unified multi-task model. The architecture of T5 closely follows the original Transformer architecture. Differences from the original implementation are limited to the layer normalisation (simplified by not using additive bias), the usage of dropout on the linear and attention layers, and a relative positional encoding strategy. Here, word embeddings are altered depending on the alignment between the "key" and "query" matrices computed by the multi-head attention layer, instead of using a fixed embedding for every position. T5 is pre-trained on the C4 English corpus—the Colossal Clean Crawled Corpus, derived from the Common Crawl—using a masked language modelling objective (similarly to BERT) that masks 15% of the tokens in every sentence. Fine-tuning is performed on all tasks at once: all datasets—including the ones for translation, question answering, and classification—are concatenated in a single dataset using a sampling strategy to ensure a balance between samples from each tasks. Differently from BERT, which has an encoder-only architecture, T5 uses the same causal masking strategy used in the Transformer decoder, where the attention output is masked to prevent the model from attending to subsequent positions. T5 has had a recent successor, T0 [88], which utilises the same model, but pushes the boundaries of transfer learning towards zero-shot generalisation further.

DeBERTa

The Decoding-Enhanced BERT with Disentangled Attention [89] is another language model based on the Transformer architecture. As with BERT, the model is pre-trained using a MLM objective, and it is fine-tuned using adversarial training. This strategy improves its generalisation ability and its robustness to adversarial examples. The architecture introduces a disentangled attention layer. Unlike the Transformer, where positional encoding is added to the word content embeddings to create the input representation, DeBERTa encodes words and positions separately. Attention scores for every position are computed using disentangled matrices and explicitly depend on the word content and the word relative position in the sentence. The decoder is also modified to consider word absolute positions when predicting a masked token.

ByT5

ByT5 is an adaptation of the T5 model able to process raw bytes of text, instead of tokens. Models like BERT rely on a separate tokenisation step to chunk documents into a vocabulary of sub-words. This translates into heightened memory constraints, since large vocabularies require massive embedding matrices with a large number of parameters. Additionally, the authors argue that sub-word tokenisation is not robust in handling typos and lexical variants, and reducing all unknown words to the same out-of-vocabulary token prevents the model from distinguishing between different OOV situations. Instead of processing words or sub-words, ByT5 is fed UTF-8 bytes without any preprocessing operation. To represent byte-level embeddings, the model only needs a dense matrix of 256 embeddings with additional embeddings for the special tokens used to pad sequences and to signal the end of a sentence. In this case, there is no need to handle the OOV case, since all possible bytes are represented. Furthermore, removing the dependency on tokenisation allows for the simpler training of the model on multilingual corpora, since there is no need for language-specific tokenisation strategies. ByT5 uses a slightly modified T5 architecture with a heavier encoder, which is three times the size of the decoder. The authors empirically find that a bigger encoder works better for byte-level language models. The model is pre-trained with a "span corruption" self-supervised objective, where the model learns to reconstruct sequences of 20 bytes that are masked in the input sentences.

ERNIE 3.0

ERNIE [90] enhances the language models pre-training phase integrating knowledge-graph information. As with other language models, ERNIE is trained using different unsupervised tasks, including MLM, sentence reordering, and sentence distance. The latter is a variation of the NSP task, where the model is asked to predict whether two sentences are adjacent, within the same document (but not adjacent), or from different documents altogether. In order to integrate knowledge graph information in the training phase, the tasks of knowledge masked language modelling and universal knowledge–text prediction are added. The former objective is used to let the model learn higher-level entity representations, for instance by masking the name of a character (e.g., "Harry Potter"). The latter strategy extends the former by explicitly embedding a knowledge graph. The model is provided with a knowledge graph representation of the sentence (in the shape of a triple) and a sentence, both with masked tokens, and must use this information to fill in the blanks. According to the authors, these tasks allow the model to gain a lexical understanding of words, while traditional language models only capture more global syntactical and semantic knowledge. ERNIE is composed of two modules: a universal representation module, meant to capture shared word representations, and a task-specific module pair, one for natural language understanding tasks, and the other for natural language generation tasks. Both modules use the Transformer-XL architecture [91], which differs from the original Transformer, mainly for the addition of an auxiliary recurrence module to aid in the modelling of long sequences.

FLAN

The Fine-tuned Language Net (FLAN) [92] explores the usage of instruction fine-tuning to enhance the zero-shot generalisation ability of a general pre-trained language model (similarly to T0). Zero-shot learning aims at giving models the ability to solve novel tasks—unseen during training—at inference time. To do so, the authors gathered data from 62 NLP datasets and 12 different tasks—including language inference, translation, question answering, text classification—and aggregated them in a mixed multi-task dataset. A maximum of 30,000 samples are selected from each dataset in order to limit task imbalance. From the final dataset, samples are enriched with instruction templates, defined as descriptions of the task that the model should solve expressed in natural language. For instance, the sentence "classify this movie review as positive or negative" can be used to ask the model to perform binary classification. To increase diversity, 10 different templates are manually created for each dataset, some of them asking the model to perform collateral-derived tasks (e.g., asking for a movie classification from an SA task). FLAN uses a Transformer decoder-only architecture, similarly to GPT, with 137 billion parameters. The model is pre-trained on a collection of English documents that includes a wide variety of textual data, ranging from Wikipedia articles to computer code. This pre-trained model is then used for the instruction-tuning procedure.

5.4.6. Graph Neural Networks

Recent developments in the field of AI are exploring the application of neural networks to graph representations [93]. In this context, graph neural networks (GNNs) [94,95] are architectures that utilise such structures to capture the dependencies and relations between nodes. Well-established approaches are being generalised to arbitrarily structured graphs, most notably CNNs [96,97].

Successful Approaches

Representations are based on heterogeneous graphs in which nodes are both words and documents have seen wide success; TC is thus cast as a node classification task for document nodes. TextGCN [98] utilises a graph convolutional network (GCN) on text mapped onto a graph structure. Words are connected to other words as well as documents, but there are no inter-document relations (documents are able to indirectly exchange information through multiple convolutional layers). Weights of document–word edges are based on word occurrence measurements (usually TF-IDF), while word–word edges are weighted on word co-occurrence in the whole corpus (usually variations of point-wise mutual information [99]). The training procedure learns word and document embeddings, and is therefore connected to text embedding techniques (such embeddings are learnt simultaneously in this case). On this account, it is easy to see how graph architectures can also be integrated with deep language models. BertGCN [100], for example, trains a GCN jointly with a BERT-like model, in order to leverage the advantages of both pre-trained language models and graph-based approaches. Document nodes are initialised through BERT-style embeddings and updated iteratively by the GCN layers. Another example is the MPAD (Message Passing Attention Network for Document Classification) [101], which proposes the application of the message passing framework to TC. The nodes represent unique words and the directed edges encode the text flow and word co-occurrence statistics. Nodes and edges information is iteratively aggregated and combined using GRUs to update word representations.

Transductive Nature

Many GNN architectures include unlabelled test document nodes in the training procedure, making them inherently transductive. Transductive learning [102] can be useful because of how it can allow label influence to be propagated to unlabelled test data during training, notably removing the need for modelling the relation between data and target labels. Often, the data required for comparable performances are fewer than that for traditional inductive learning approaches. However, this also has the downside of not being able to quickly generate predictions for unseen test documents that were not included in the training procedure (i.e., online testing). Furthermore, building a GNN for a large-scale text corpus can be costly due to memory limitations [103].

Weaknesses and Solutions

Reducing the modelling cost has been an important topic of discussion. Huang et al. [103], for example, change the model training strategy, opting to build graphs for each input text and utilizing global parameter sharing to reduce the burden, while maintaining global information. Methods such as SGC [104] work instead on reducing model complexity; in particular, SGC repeatedly removes nonlinearities between consecutive layers in deep GCNs, collapsing the resulting function into a single linear transformation. The authors argue that the expressive power of GCNs originates from graph propagation rather than nonlinear feature extraction, and exhibit comparable or even superior performances to other methods. Related to such nonlinearities is the phenomenon of over-smoothing, which suggests that increasing the number of layers in GCNs causes the node representations to converge to a same value (and become hence indistinguishable) [105]. While removing such non-linearities may certainly help in this regard, recent research has developed techniques aimed directly at combatting this phenomenon. For example, GCNII [106] extends the vanilla GCN model with initial residual and identity mapping. Initial residual connections are related to the idea of residual connections in residual networks, but modify this concept by adding a skip connection to the input layer instead (i.e., the initial representation), while identity mapping (also borrowing from residual networks) adds an identity matrix to the weight matrix. They show great improvements on the over-smoothing problems and state-of-the-art results in TC tasks. SSGC [107] also addresses this problem, by combining techniques from previous works—namely, SGC and a specialised graph convolution filter [108,109].

Summary

While this section is merely a brief introduction to GNNs with a few notable examples, they are among the few architectures able to compete with deep language models in recent years. They are of particular interest both, because of their excellent performance—sometimes even achieving state-of-the-art results—but also because of how they can perform quite well in low label rate datasets [98], a characteristic which can be attributed to their transductive nature.

5.4.7. Contextualisation of Word Embeddings

As anticipated previously, recent language models have the extremely valuable capability of incorporating context into their representations of text. This means that these representations, in principle, are effectively able to understand and distinguish polysemous words by looking at the body of text that they appear in, as well as containing complex characteristics of word use (its syntax and semantics). A practical difference with the earlier word embeddings described in Section 3.3 is that, while older embeddings were "static" and could therefore be extracted as individual entities, contextualised embeddings require their originating language model to be generated. Such language models are only able to understand words in context—they are not meant or suited for isolated words.

Contextualised embeddings have been proven to be "very contextual", meaning that they can adapt well to the different semantic significance of words [110]. Furthermore, researchers have extracted word representations from the lower hidden states of BERT-like language models, and managed to create static embeddings of much higher quality than the ones created by static embedding procedures, further proving the benefits of this learning process.

Contextualised embeddings were not a novelty introduced by Transformers; multiple research contributions pushed on their development [111,112]. As a prime example, ELMo [65] introduced a bidirectional LSTM architecture trained for language modelling (i.e., next word prediction) that also produced contextualised embeddings. Due to its recurrent structure, however, it struggled with long-term dependencies.

5.4.8. Challenges of Language Models

One of the main challenges posed by very deep language models is their size. This depth incurs a huge number of parameters that must be loaded into memory, an operation that is not exclusive to the training phase. In practice, this might be an issue when pre-trained models are used for inference under low-latency constraints, or fine-tuning in lower resource settings.

Solutions to this problem have been devised in order to create more compact models (i.e., with fewer parameters). A successful approach to this end is knowledge distillation [113]. In this process, the larger, original model is utilised as a "teacher" in the creation of the more manageable "student" sub-model. The key idea of distillation is that it should be easier to train a student model to mimic the output distribution (i.e., the knowledge) of a bigger teacher model, with enough capacity to learn a concise knowledge representation from raw data. Thus, the probability scores assigned by the teacher to input samples are used as predictive targets for the student, allowing it to encode the teacher knowledge in a compressed form, without having to learn it from scratch. DistilBERT [114] leverages this to reduce the size of BERT models by up to 40%, while retaining 97% of its effectiveness. Much in the same way, TinyBERT [115] is also based on knowledge distillation, but extends it to the task-specific learning stage. This model is also able to retain most of its teacher's performance.

On the other hand, seeing that such approaches still require a teacher model, there have been methods that propose tackling the issue from a different angle. ALBERT [116], for example, introduces parameter reduction techniques to reduce memory consumption and increase the training speed of BERT models. ELECTRA [117] introduces token detection, a more sample-efficient pre-training task, in place of masked language modelling. This task corrupts the input by replacing tokens with plausible alternatives rather than masks, and changing the learning task to a discriminative one, where the model must identify whether each token in the corrupted input had been artificially replaced or not. In particular, this reframing has proven to be effective in models with fewer parameters, increasing their viability.

In summary, downsized models such as these are essential in practical applications of DLMs. It is worth noting that, whenever these alternatives are not available, it may be possible to perform a fine-tuning procedure without involving the language model in the learning process (i.e., "freezing" the base model's weights). This would be similar to utilising the word embeddings contained in the LM in their agnostic state (contextualised, but not specialised). The resource requirements are hence reduced, though this shrinks the learning capacity of the system.

6. Experimental Performance Analysis

We report, in this section, the performance metrics of a selection of machine learning models on a number of TC tasks among those introduced in Section 1.1. Moreover, we introduce two new multilabel datasets and showcase results for a selection of TC approaches on them. To test models on a TL task, we introduce a new public dataset based on Wikipedia articles labelled with their topics. We also perform tests on the RCV1 dataset as a NC representative, which is available freely for research purposes upon request.

6.1. Datasets

We researched the most popular TC datasets utilised in recent works. We focused on the document-level tasks; hence, we excluded from our search the NLI, SP, and NER tasks. The list of datasets is shown in Table 3, and we refer to the reviews by Li et al. [1], Minaee et al. [3] for a description of each one.

Most datasets with document-level annotations have a single target label for each sample, and are hence used in binary or multiclass classification tasks. For the development of models for TL and NC tasks, we found a limiting unavailability of multilabel datasets. Despite this, real-world applications of supervised topic extraction methods for documents and news are likely to require the assignment of multiple topics to pieces of text, rather than just one. For the evaluation of methods on TL tasks, Wikipedia articles provide a source of semi-structured text labelled with multiple categories (topics) assigned by contributors. Such data can be extracted by accessing the DBpedia https://www.dbpedia.org (accessed on 28 December 2021) project, or from the articles' raw files available in Wikipedia dumps https://dumps.wikimedia.org (accessed on 28 December 2021). Although these corpora have already seen use in various works [118,119], there is no consistent set of annotations to be used as reference for a multilabel formulation. For news classification, we synthesised a new multilabel dataset from RCV1 [120], a collection of English news articles from the Reuters agency. Statistics for both datasets are appended to Table 3.

6.1.1. New Datasets Distillation

EnWiki-100

The synthesised EnWiki-100 dataset contains the text of more than 300,000 English Wikipedia pages, along with a variable number of topics related to the page content. While categories assigned to Wikipedia pages by contributors are often used as predictive targets, we found that these frequently contain too generic or too specific categories that are not informative of the page's main topics. Therefore, extracted articles are annotated with the name of their respective Wikipedia portals. In support of this decision, one should consider that English Wikipedia contains roughly 500 portals, while categories are more than 500,000. Portals themselves are stated to be "entry points" for articles belonging to the same broad subject [121]. Consequently, they are better suited as targets for a TL task.

Articles are extracted from the September 2021 Wikipedia dump, using a customised version of the WikiExtractor https://github.com/attardi/wikiextractor (accessed on 28 December 2021) tool, modified for the extraction of the portal names from the page metadata. Only the 100 most populous portals are kept for the final dataset. In addition, article frequency has been limited to a maximum of 50,000 per label, such as to reduce the size of the already large dataset, as well as to limit class imbalance.

RCV1-57

The RCV1-57 dataset is extracted from the RCV1 (version 1) collection of English news. Articles are labelled with a variety of tags that describe their content in different manners. The most consistent and appropriate were topic codes, which describe the general subject of the news piece. Such codes have a hierarchical structure, with the higher levels being more abstract, and the lowest being the most specific. We decide to cut off the codes at the second level of specificity, since they were the most complete and descriptive overall. We clean up labels by stripping tags from other levels, and discard topics with fewer than 500 representatives. If articles contain only scrapped topics, they are also discarded.

Table 3. English TC datasets.

Name	Task	Classes	Multilabel	Samples
AG News [122]	NC	4	✗	127,600
20 News (20 Newsgroup) [123]	NC	20	✗	18,846
R52 [124]	NC	52	✗	9100
R8 [124]	NC	8	✗	7674
Yelp2 (Yelp Polarity) [125,126]	SA	2	✗	598,000
Yelp5 (Yelp Full) [125,126]	SA	5	✗	700,000
Amz2 (Amazon Polarity) [125]	SA	2	✗	4,000,000
Amz5 (Amazon Full) [125]	SA	5	✗	3,650,000
IMDb [127]	SA	2	✗	50,000
IMDb10 [127]	SA	10	✗	50,000
MR (Movie Review) [128]	SA	2	✗	10,662
Yah!A (Yahoo! Answers) [125]	TL	10	✗	1,450,000
TREC [129]	TL	6	✗	5952
Ohs (Ohsumed) [130,131]	TL	23	✓	13,929
DBP (DBpedia-14) [125]	TL	14	✗	630,000
EnWiki-100 [1]	TL	100	✓	329,600
RCV1-57 [1]	NC	57	✓	758,100

[1] Datasets presented in this paper.

6.2. Evaluation Metrics

Accuracy is the most adopted evaluation metric for TC tasks. This metric is a simple and interpretable way of measuring the overall fraction of correct predictions. It is defined as the number of correct predictions over the total number of samples (N), as in Equation (6). The equations use standard notations for evaluation metrics, namely true positive (TP), true negative (TN), false positive (FP), and false negative (FN).

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{N}} \tag{6}$$

In a multilabel setting, the definition of this metric can change depending on the definition of true positives and negatives. A prediction may be considered correct when the predicted labels exactly match the ground truth (referred to as "subset accuracy"). Alternatively, predictions can be flattened and reduced to a single-label task before the accuracy computation.

Precision and recall are other popular metrics, especially for multilabel classification. The former is the fraction of correct predictions among all the ones that have been predicted as true (TP + TN), while the latter is the fraction of correct predictions over all the ones that should have been predicted (TP + FN). *F-score*, and in particular F_1*-score*, is a popular combination of these two; it is defined as the harmonic mean of precision and recall, as in Equation (7).

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \qquad \text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \qquad F_1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{7}$$

For multilabel and multiclass problems, these metrics can be computed separately for each class and then averaged, obtaining the macro-averaged metrics. In such a case, every class contributes equally to the final score, and hence provides a more challenging metric for unbalanced datasets. On the other hand, a micro-average reduction strategy is used when computing the metric globally with no weighting.

6.3. Quantitative Results

The accuracy score for several previously introduced algorithms on prominent benchmarks are shown in Tables 4 and 5. We report only accuracy, since it is the most consistently used metric across different studies. GLUE and RACE benchmarks are popular choices for the evaluation of NLP methods on a variety of tasks, hence we report results on these datasets in Table 6, whenever available. We evaluated a selection of deep and shallow methods on our new synthesised datasets. Since our datasets are multilabel, we report the F_1-*score* with macro-averaging across all categories, along with the subset accuracy, computed on the test set. Tables 4 and 5 showcases our results, averaged on four different evaluation splits. The following section provides additional details on the training procedure.

Table 4. Test accuracy score (%) on SA benchmark datasets (best results in bold).

Model	Yelp2	Yelp5	Amz2	Amz5	IMDb	IMDb10	MR
XLNet [19]	**98.63**	72.95	**97.89**	**68.33**	96.80	-	-
BERT-base [100,132,133]	98.13	70.80	-	-	95.63	54.20	85.70
BERT-large [132–134]	98.19	71.38	97.37	65.83	95.79	**55.60**	-
L-MIXED [135]	-	-	-	-	95.68	-	-
DNC + CUW [136]	96.40	65.60	-	-	91.30	-	-
SVDCNN [74]	95.26	63.20	-	-	-	-	-
ULMFiT (small-data) [137]	97.10	67.60	96.10	64.10	-	-	-
BERT-large-UDA (ft) [134]	97.95	67.92	96.50	62.88	95.80	-	-
USE_T + CNN [138]	-	-	-	-	-	-	81.59
MPAD [101]	-	66.80	-	-	91.87	-	-
STM + TSED + PT + 2L [139]	-	-	-	-	-	-	80.09
VLAWE [140]	-	-	-	-	-	-	93.30
TM [141]	-	-	-	-	-	-	77.51
ErnieDoc-large [142]	-	-	-	-	97.10	-	-
BigBird [143]	-	72.16	-	-	95.20	-	-
NB-weight-BON + CS [144]	-	-	-	-	**97.42**	-	-
HAHNN (CNN) [71]	-	**73.28**	-	-	92.26	-	-
HAHNN (TCN) [71]	-	72.63	-	-	95.17	-	-
RoBERTa + ODPT [145]	-	-	-	-	96.60	-	-
HAN [80]	-	71.00	-	63.60	-	49.40	-
SSGC [107]	-	-	-	-	-	-	76.70
SGCN [104]	-	-	-	-	-	-	75.90
TextGCN [98]	-	-	-	-	-	-	76.74
GCN [98]	-	-	-	-	-	-	76.30
RMDL (15 m) [146]	-	-	-	-	90.79	-	-
GraphStar [147]	-	-	-	-	-	-	76.60

Table 4. *Cont.*

Model	Yelp2	Yelp5	Amz2	Amz5	IMDb	IMDb10	MR
BertGCN [100]	-	-	-	-	-	-	86.00
RoBERTaGCN [100]	-	-	-	-	-	-	89.70
DRNN [62]	97.27	69.15	96.49	64.43	-	-	-
ULMFiT [61]	97.84	70.02	-	-	95.40	-	-
DPCNN [148]	97.36	69.42	96.68	65.19	-	-	-
LSTM-reg [149]	-	68.70	-	-	-	52.80	-
KD-LSTM-reg [133]	-	69.40	-	-	-	53.70	-
CCCapsNet [150]	96.48	65.85	94.96	60.95	-	-	-
CharCNN [125]	95.40	62.05	95.07	59.57	-	-	-
EFL [151]	-	64.90	-	-	96.10	-	**92.50**
byte mLSTM [152]	-	-	-	-	92.20	-	86.80
BLSTM-2DCNN [153]	-	-	-	-	-	-	82.30
oh-2LSTMp [154]	-	-	-	-	91.86	-	-
VDCNN [73]	95.72	64.72	95.72	63.00	-	-	-
RoBERTa [100,143]	-	71.75	-	-	95.00	-	89.40

Table 5. Test accuracy score (%) on NC and TL benchmark datasets (best results in bold).

	NC				TL			
Model	**AG**	**20N**	**R52**	**R8**	**Yah!A**	**TREC**	**Ohs**	**DBP**
XLNet [19]	**95.55**	-	-	-	-	-	-	**99.40**
BERT-base [100,132]	95.20	85.30	96.40	97.80	**78.14**	**98.20**	70.50	99.35
BERT-large [132]	95.34	-	-	-	-	-	-	99.39
L-MIXED [135]	95.05	-	-	-	-	-	-	99.30
DNC + CUW [136]	93.90	-	-	-	74.30	-	-	99.00
SVDCNN [74]	90.55	-	-	-	-	-	-	-
ULMFiT (small-data) [137]	93.70	-	-	-	74.30	-	-	99.20
USE_T + CNN [138]	-	-	-	-	-	97.44	-	-
MPAD [101]	-	-	-	97.57	-	95.60	-	-
STM + TSED + PT + 2L [139]	-	-	-	-	-	93.48	-	-
DELTA [155]	-	-	-	-	75.10	92.20	-	
VLAWE [140]	-	-	-	89.30	-	94.20	-	-
TM [141]	-	-	89.14	97.50	-	90.04	-	-
HAN [80]	-	-	-	-	75.80	-	-	-
SSGC [107]	-	88.60	94.50	97.40	-	-	68.50	-
SGCN [104]	-	88.50	94.00	97.20	-	-	68.50	-
TextGCN [98]	-	86.34	93.56	97.07	-	-	68.36	-
GCN [98]	-	87.90	93.80	97.00	-	-	68.20	-
NABoE-full	-	86.80	-	97.10	-	-	-	-
RMDL (15 m) [146]	-	87.91	-	-	-	-	-	-

Table 5. *Cont.*

	NC					TL		
Model	**AG**	**20N**	**R52**	**R8**	**Yah!A**	**TREC**	**Ohs**	**DBP**
GraphStar [147]	-	86.90	95.00	97.40	-	-	64.20	-
SCDV-MS [156]	-	86.19	-	-	-	-	-	-
STC-Q [157]	-	87.30	-	-	-	-	-	-
BertGCN [100]	-	89.30	**96.60**	98.10	-	-	**72.80**	-
RoBERTaGCN [100]	-	**89.50**	96.10	**98.20**	-	-	**72.80**	-
RepSet	-	77.02	-	96.85	-	-	66.12	-
Rel-RWMD kNN	-	74.78	-	95.61	-	-	58.74	-
DRNN [62]	94.47	-	-	-	-	-	-	99.19
ULMFiT [61]	94.99	-	-	-	-	96.40	-	99.20
DPCNN [148]	93.13	-	-	-	76.10	-	-	99.12
CCCapsNet [150]	92.39	-	-	-	73.85	-	-	98.72
CharCNN [125]	91.45	-	-	-	71.20	-	-	98.63
EFL [151]	86.10	-	-	-	-	80.90	-	-
byte mLSTM [152]	-	-	-	-	-	90.40	-	-
BLSTM-2DCNN [153]	-	-	-	-	-	96.10	-	-
oh-2LSTMp [154]	-	86.68	-	-	-	-	-	-
VDCNN [73]	91.33	-	-	-	73.43	-	-	98.71
RoBERTa [100]	-	83.80	96.20	97.80	-	-	70.70	-

Table 6. GLUE and RACE score updated to latest results (best ones in bold).

	GLUE		RACE
Model	**Reported**	**Latest**	**Latest**
BERT (base) [16]	79.60	-	65.00
BERT (large)	82.10	80.50	67.90
RoBERTa [13]	88.50	88.10	83.20
XLNet (large) [13,19]	88.40	88.28	81.75
ALBERT (ensemble) [116]	89.40	88.16	**89.40**
ELECTRA (large) [117]	89.40	89.40	-
ELECTRA (base) [117]	85.10	-	-
T5 [87]	90.30	90.30	87.10
GPT [83]	72.80	-	59.00
DeBERTa (large) [89]	**90.00**	90.80	-
ERNIE [90]	-	**91.10**	-

6.3.1. Custom Experimental Setup

We selected seven algorithms (Table 7) that either were recently, or have been, state-of-the-art approaches in TC tasks, and we test their performances on the EnWiki-100 and RCV1-57 datasets to showcase their performances on new multilabel datasets. As strong baseline representatives for classical methods, we present the results of Naïve Bayes and linear support vector classifiers, both using TF-IDF features. These methods are utilised in a one-vs.-rest ensemble fashion, as is common in multilabel applications. We then present the results of FastText [53], a re-implementation of XML-CNN [118,133] and a BiLSTM-based classifier as representatives of those methods bridging the gap between earlier methods and DLMs. Lastly, we fine-tuned and tested Transformer-based language models utilising the pre-trained open-source models, published in the Hugging Face library [158].

Datasets are split into training, validation, and test set: 40% of the data are reserved for testing, and 20% of the remaining samples are used for validation. Splits are produced in a way to preserve the distribution of labels, through a stratification strategy [159,160]. Final metrics reported in Table 7 are obtained by averaging results on the test set over all runs (four runs of each method, except for XLM-R, due to computational complexity). Hyperparameters for the Naïve Bayes and SVM classifiers are tuned using a grid search with 10 fold cross-validation. FastText hyperparameters are selected using the auto-tuning procedure provided in the Python package. Due to resource constraints, only a limited tuning of the most impactful parameters is performed on the bidirectional LSTM and Transformer-based models.

Table 7. Test set Macro-F_1 and Subset Accuracy (%) on synthesized datasets (best results in bold).

	***F_1*-Score**		**Accuracy**	
Model	**EnWiki-100**	**RCV1-57**	**EnWiki-100**	**RCV1-57**
Naïve Bayes (OVA)	63.64	56.30	39.24	47.27
Linear SVM (OVA)	80.29	71.73	66.93	67.67
FastText Classifier	74.45	69.65	68.23	67.02
BiLSTM (GloVe)	81.22	76.62	68.05	72.48
XML-CNN (FastText)	78.19	73.02	66.64	70.98
BERT (base)	**85.52**	**78.07**	**75.28**	73.48
XLM-R (base)	84.60	77.19	74.25	**74.01**

6.3.2. Discussion on Results

Tables 4 and 5 show the dominance of Transformer-based language models in all TC tasks. Specifically, BERT and XLNet reach state-of-the-art performances on most datasets. Moreover, TC using graph representation for documents, paired with graph convolutional networks, proves to be an effective approach for the extraction of useful features, especially when node embeddings are initialized with contextual representations generated by BERT-like models (BertGCN, RoBERTaGCN). Notably, Thongtan and Phienthrakul [144] propose to learn document-level embeddings by minimising cosine similarity, and combining features extracted from a Naïve Bayes model before the classification step. The model architecture is very simple in comparison with BERT and other deeper language models, but manages to achieve the best accuracy on a binary SA dataset. The authors argue that, for this kind of task, tailored document representations are more important than the choice of classifier. Abreu et al. [71] achieve state-of-the-art results on the Yelp5 dataset, by using a CNN to extract features from word vectors and using the attention mechanism twice, first at the word level, to generate a sentence representation, and then at the sentence level, to weight the importance of sentences in the document embedding used for classification. This model makes use of recurrent blocks (GRU) and reiterates the importance of the attention mechanism to enhance seq2seq architectures.

The latest trend for the evaluation of deep language models is the leveraging of general multi-task benchmarks. Table 6 showcases the results of two of them. These benchmarks measure the model ability to generalise to multiple tasks, and are not specific for TC. The reported score averages the performance metrics of the model across all tasks.

Finally, Table 7 lists the results that we obtained on the new synthesised multilabel datasets. Since these are new datasets, we use them to test the classification performance of a few notable language models, within the limits of our resources. Unsurprisingly, BERT and XML-R achieved the best results over the two new corpora, with accuracy values aligned with the respective scores of similar multilabel TL datasets (e.g., Ohsumed). We clarify that, since these datasets are different, metrics should not be directly compared, but we expected classification accuracy to fall in a range of values that are reasonably comparable to the other TL datasets. Interestingly, XML-R performed only slightly worse than BERT, despite being a multilingual model which was not specifically trained to understand the English language. On the one side, this is interesting, as the curse of multilinguality—which states that the per-language capacity decreases as the number of languages grows—suggests that BERT should be able to understand much better the single language that it was trained on. On the other hand, one should consider that the smaller XML-R model has more than 2.5 times the parameters of the BERT base, and the two models are pre-trained on different corpora. XML-R is pre-trained on the Common Crawl in 100 languages, while our BERT model is pre-trained on the combination of BookCorpus and Wikipedia dumps. While the former corpus amounts to 2.5 TB of data, the latter does not exceed one hundred GB. The BiLSTM also performed better than other non-Transformer methods, as expected, since its architecture makes it suitable to model dependencies between adjacent chunks of text.

7. Future Research Directions

Despite achieving state-of-the-art results throughout NLP literature, large-scale lan -guage models are not infallible. For example, recent studies [161,162] reported that BERT models show considerable performance pitfalls when faced with adversarial text examples generated by adversarial networks. These text sequences are subtly altered with word-level replacements or sentence rephrasings that do not change their meaning (as far as human understanding is concerned), but are enough to fool these language models. This phenomenon raises questions about the ability of these models to make decisions based on the actual meaning carried by a portion of text. Particularly interesting is therefore the development of robustness benchmarks such as AdvGLUE [163], which aim to quantita-tively and thoroughly explore the vulnerabilities of modern large-scale language models. Such benchmarks are carefully filtered and validated by human annotators, such as to ensure that adversarial examples are valid and unambiguous. Benchmarks like these will be fundamental in the development of more sophisticated adversarial attacks, as well as more robust language models able to withstand them.

The current trend of proposing deeper and deeper models with an ever-increasing number of parameters has led some scientists to warn against the creation of so-called "stochastic parrots" that effectively emulate language understanding by memorising large training datasets, but with very limited actual generalisation ability, let alone the ability to comprehend human language. Furthermore, the black-box nature of deep neural networks adds to these concerns, as learnt features are hardly interpretable, while some computational linguists also bring legitimate concern about harmful and dangerous biases learnt from the data [79]. As a consequence, future research should put the robustness of large-scale language models under mindful scrutiny and provide tools to lessen the interpretability issues which currently afflict deep learning. The former problem is addressed by Wang et al. [164], who propose a framework for a more robust fine-tuning of BERT language models.

Finally, recent developments have gone against the trend of scaling larger and larger models, proposing that smaller generative LMs can perform competitively with massively larger models when augmented with search/query information from a retrieval database [165,166]. For example, Retrieval-Enhanced Transformer (RETRO) performs on par with GPT-3, despite being 4% its size. Pursuing more reasonably sized models is certainly a research direction that will be worth exploring.