Neural Language Models

John Lyons

Technische Hochschule Rosenheim, Rosenheim 83024, Germany
john.lyons.93@hotmail.com

Abstract. Language models are very common nowadays. They are used for translation, sentiment analysis, in chatbot environments and many other tasks. The history of language models is characterized by rule-based or pure statistical approaches (Bahl et al., 1983). New architectures, such as RNNs make it possible to process sequences in neural networks. Furthermore, trainable word-embeddings, such as proposed by Mikolov et al. (2013) and Pennington et al. (2014), improve neural language models performance resulting in RNN based approaches being equal to the older pure phrase-based or statistical language models. The *transformer* architecture, which is based on the *attention mechanism*, is introduced by Vaswani et al. (2017) and drastically speeds up the development of language models as training and inference time can be further optimized using concurrency. Popular language models based on the *transformer* architecture, such as *BERT* or *GPT-3*, reach SOTA benchmark results and are being widely used.

Keywords: Neural Language Models · Attention · BERT · GPT · Transformer · RNN

1 Introduction

These days language models are very common in various application areas. For example, chatbots are using language models in order to detect the user's intent and serve with FAQ-style answers. This saves cost as no human support client is needed, and provides other advantages, such as constant performance and availability throughout the whole day.
In addition to that, search engines, such as google, use language models to improve question-style search queries. Instead of only using keywords or algorithm based techniques, languages models allow for more complex queries and are able to find answers in documents given a question.
Other than that, there is still the field where language models are being used for entertainment purposes such as conversations between human and a chatbot. *Tay.ai*, a twitter chatbot developed by the Microsoft team, was such an experiment in the year 2019 which unfortunately did not end very well. Shortly after the initial release people began to abuse the bot's ability to learn abusive and racist content. Therefore, Microsoft felt the need to dispose the chatbot from twitter.
The next section deals with the fundamental ideas of language models and shows how neural language model became as popular as they are now.

History of Language Models

There exist various types of language models which can be roughly structured as rule based, stochastic and neural language models.
Specifically in the beginning of the evolution of language models phrase-based and stochastic approaches were used. When targeting the task of forming, translating or validating a sentence the idea is to split sentences into words and assign each word to a probability distribution based on its predecessors. A similar approach is used when dealing with phrases. Let the sentence be denoted as the ordered set $w_1..w_N$ and the sentence length is N. (Bahl et al., 1983)

$$Pr(w_1...w_N) = \prod_{n=1}^{N} Pr(w_1...w_{N-1}) \tag{1}$$

Then the equation 1 (showed by Bahl et al. (1983)) shows the probability of the existence of the corresponding sentence. Each word depends on all the previous words. As such, this is a markov chain problem and is computationally expensive, especially when the sentence is very long.
Because of that Bahl et al. (1983) proposes to use n-grams which essentially allows to reduce the markov chain depth to n. Now, each word only depends on the last n words. Therefore, defining probabilities for upcoming words is now limited to a fixed predecessor length. The most common n-grams are bigrams and trigrams. The equivalent of equation 1 using $n = 3$, hence called trigram, is depicted in equation 2. (Bahl et al., 1983)

$$Pr(w_n|w_1...w_{n-1}) = Pr(w_n|w_{n-2}w_{n-1}) \tag{2}$$

The probability of the existence of word w_n based on all of its predecessors is the same as based on only two previous words. Therefore, the computation of these probabilities is more efficient and realistic than the first approach. n-grams were already mentioned back in 1948 by Shannon (1948) which shows that the idea already exists for a long time.
However, training a language model with the above approach has its drawbacks. For example, there are many bi- or trigrams which do not appear in training data. As such, after training, the probabilities of words appearing directly after those unseen bi- or trigrams would be zero. As a result, those probabilities have to be adjusted in order to make other word combinations, which are used in natural language and are grammatically correct, legitimate. (Bahl et al., 1983)
Nevertheless, researchers were working on probabilistic language models until the rise of computer power allowed to try out different architectures. In 1982 John Hopfield introduced RNNs which made it possible to process sequences, such as time-based data, speech or text. This further led to more research in that particular area. As a consequence, LSTM and GRU cells came out and improved some of the characteristics of RNNs. Furthermore, Bengio et al. (2001) ultimately introduced the first neural language model based on above-mentioned architectures. They especially showed that converting words to vectors results in improved language processing performance.

An important fundamental for neural language models is to transform words to vectors. Mikolov et al. (2013) propose a novel approach on how to train above-mentioned word-embeddings. Pennington et al. (2014) take the approach further and introduce different techniques for training which lead to better results.
By using those word-embeddings, neural based language models reached performance of the older statistical or rule-based language models. Some scientific work has been put into improving characteristics of RNN based SEQ2SEQ models. For example, Li et al. (2016) use RL to train a policy that encourages longevity of a conversation. But even then the drawbacks of RNN based approaches, such as the problem of "forgetting" when reading long sequences or the nature of sequential computation, which is slow, made other ideas arrive.
As a result Vaswani et al. (2017) came up with the *transformer* architecture which is solely based on the new, so-called *attention mechanism*, introduced by Bahdanau et al. (2016). The *transformer* architecture allows to parallelize the training for one particular sentence because each word can be processed concurrently. As a consequence, training with a large parameter count can be sped up and thus many new language models, which achieve SOTA benchmark results, are based on the *transformer* architecture.
The following section briefly explains the fundamentals of modern neural language model architectures.

2 Architectures

RNN

Similar to a usual neural network the recurrent neural network uses cells which have weights, take an input and produce an output. However, the RNN cell can be viewed as a function over time because the output depends on prior outputs of the function. (Géron, 2019)

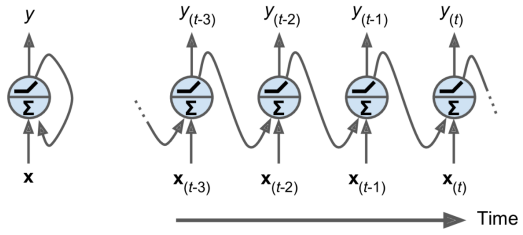

Fig. 1: Recurrent neural cell. Source: Géron (2019)

Fig. 1 depicts the RNN cell on the left-hand side whereas the function is repeatedly drawn over time on the right-hand side. The next output of the RNN cell depends on the output of the previous cells and therefore of all previous

inputs. Unlike a layer with traditional neural cells, the RNN layer has separate weights for the input of the previous output. Let W_x denote the matrix which holds the weights for the input of each RNN cell, W_y denote the weights for the input of each previous output and $Y_{(t-1)}$ be all the outputs of the layer's previous calculation for each time-step t. $X_{(t)}$ is the matrix which holds the mini-batch and b is the bias. Then equation 3 shows how the output is calculated. (Géron, 2019)

$$Y_{(t)} = \phi(X_{(t)}W_x + Y_{(t-1)}W_y + b) \tag{3}$$

Therefore, the difference to the classic neural cell, whose output is calculated with $\phi(XW + b)$, is only the weighted addition of previous outputs. Though, depending on the complexity of the RNN cell, the hidden state (which is used as input for the next time-step) can be different from the output at a corresponding time-step. (Géron, 2019)
Strictly speaking, the RNN cell has a memory which is, as we know today, very limited. However, it is possible to accumulate information of short sequences into an RNN cell. As such, it is also possible to extract information out of an RNN cell, either by providing additional input, the already produced output at a timestep t or by just providing zero input and solely relying on the memory of the cell. (Géron, 2019) (Bengio et al., 2001)

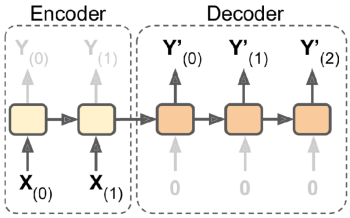

Fig. 2: Encoder-Decoder stack with RNN-like cells. Source: Géron (2019)

Fig. 2 shows how a sequence can be converted to another sequence. As already discussed, a sentence can be split into words and those words can be transformed to vector space. As such, each word vector can be fed to the encoder which effectively results in one vector representing a sentence. Furthermore, the decoder can produce a sequence of vectors (representing words) and thus, for example, translate the sentence. (Géron, 2019)
However, there are some issues. First, long sequences can be difficult to encode, because of the already-mentioned problem of short memory. Solutions, such as LSTM or GRU cells exist but are not further discussed in this paper. Second, the sequential nature of the RNN cells leads to the problem that each word of a sentence has to be put into the network one-by-one. This leads to longer training and inference duration because concurrency is not as extensively used as it is by

other methods. (Géron, 2019) (Vaswani et al., 2017)
As a result Vaswani et al. (2017) introduced the popular *transformer* architecture, which makes heavy use of the *attention mechanism*. The next section deals with this architecture.

Transformer

The idea of *attention* was first introduced by Bahdanau et al. (2016). They showed that it is possible to let the decoder unit focus on only the relevant words of the input sequence.
Because a classic RNN encoder-decoder stack accumulates all information into a single dense vector, the decoder cannot focus on words which occurred early in the sequence (when the sequence is long). The *attention mechanism* circumvents this issue by not only providing the last hidden state output to the decoder but all the hidden states. Not only that, but the decoder also weights all the encoder hidden state outputs so at each time step the decoder knows which parts of the input sequence are important for generating the next word. The resulting weighted context vector for the output position i is thus calculated by equation 4, in which the corresponding weight for output position i and input position j is denoted by $a_{(i,j)}$. (Bahdanau et al., 2016)

$$c_i = \sum_{j=1}^{T_x} a_{ij} h_j \tag{4}$$

The a_{ij} weights are trained by a special *attention layer* which is further described by Bahdanau et al. (2016). Given the context vector the decoder has now better contextual information about the surrounding source words at the input position j.
The *transformer* architecture introduced by Vaswani et al. (2017) takes this approach further by slightly modifying the *attention mechanism* and solely relying on it, thus not being depended on RNNs.

Fig. 3 depicts the *transformer* architecture. What is special about this architecture is that every tokenized word of the input sequence is used concurrently. That leads to the problem, that the encoder and decoder has to somehow know the order of the words. This is solved by adding special positional encoding vectors to each of the tokenized input word vectors and as such the model can learn the order of words. (Vaswani et al., 2017)
The *attention mechanism* used is called *Scaled Dot-Product Attention* and is shown in equation 5.

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{5}$$

Unlike in the before-mentioned *attention mechanism*, introduced by Bahdanau et al. (2016), the newly introduced version transforms each tokenized input word vector using specially trained weights denoted as W^Q, W^K and W^V.

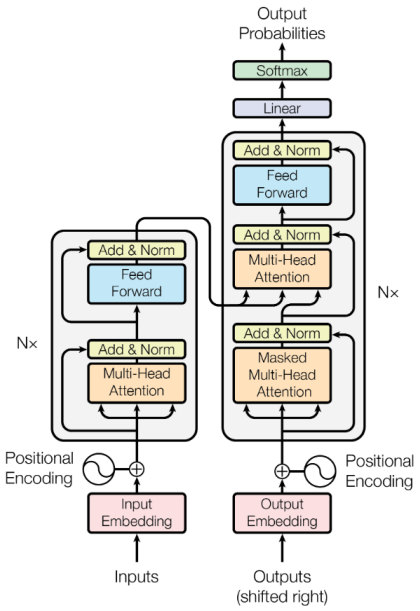

Fig. 3: Transformer architecture. Source: Vaswani et al. (2017)

This results in the matrices Q, K and V respectively (the transformed words). Following that, the result is scaled and the $softmax$ function is used. The multiplication with V effectively mixes each word vector representation with other word vector representations. As a result, the proposed *attention mechanism* is able to relate each word of a sentence proportionally to all the other words in the sentence. The resulting vectors generated by the *attention layer* are further processed by *feed-forwarrd layers*. *Multi-head Attention* denotes the ability of the layer to have multiple attention calculations with different weights W_i^Q, W_i^K and W_i^V. As a consequence, the model has the ability to train completely different attentions. (Vaswani et al., 2017) (Alammar, 2022)
The decoder on the right side uses a *Masked Multi-Head Attention layer* which further allows to mask all not-yet generated words, as the input for the decoder is not only the "weighted word vectors" but also the already generated target sequence. (Vaswani et al., 2017)
Other than that, the architecture uses residual connections in order to compensate for vanishing gradients. Both the encoder and the decoder are stacked multiple times. At the end of the decoding process a linear layer with a $softmax$ output-function is used, which ultimately returns the probabilities of the next word to generate. (Vaswani et al., 2017)

3 Popular Neural Language Models

Most popular language models achieve high performance on various benchmarks. For example, the *GLUE benchmark* is nowadays often used in order measure the quality of a language model. It contains multiple tasks, such as measuring the similarity between two sentences or classifying a sentence as grammatically correct. (Wang et al., 2019)
Therefore, not only one but many different tasks are taken into account. That means, in order to reach a good score, the language model has to perform well on many of the provided tasks.
The following section presents some of the today most popular neural language models.

BERT

A recent advancement in machine learning and natural language processing is *BERT*, which stands for "Bidirectional Encoder Representations from Transformers". It is based on the encoder unit from the *transformer* architecture with slight differences as far as attention head count and other hyperparameters are concerned. (Devlin et al., 2019)
The main idea is to provide an encoder which can be further used for many downstream tasks such as *sentiment analysis*, *extractive question answering*, *named entity recognition*, *intent classification* and more. This is achieved by constructing task specific neural network layers, so called "heads", on top of the *BERT* encoder. (Devlin et al., 2019)

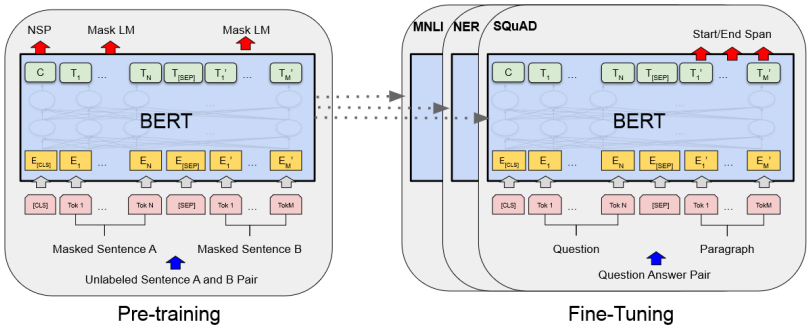

Fig. 4: Pre-training and fine-tuning with different LM heads. Source: Devlin et al. (2019)

Fig. 4 shows how a *BERT* model can have a head used for *extractive question answering*. Both the question and the paragraph are concatenated and a separator-token is inserted in between them. The input is further encoded by the encoder unit, and the custom top layers output the indexes of the answer

within the given paragraph. Similarly, a different head can be used to output classes and therefore solve an intent classification problem or other tasks at all. (Devlin et al., 2019)
The base *BERT* encoder comes pre-trained in various parameter sizes. For example, the biggest model has 330 million whereas the base model has 110 million parameters. It was trained in an unsupervised fashion on a large amount of training data using a *masked language modeling* and a *next sentence prediction* technique which are further described by Devlin et al. (2019). The various models are downloadable for free however the parameter count may limit the usage (especially when training) on low-hardware computers or computers without graphics card acceleration. (Devlin et al., 2019)
Therefore, after downloading the encoder model of choice, the only thing that has to be done after constructing a task corresponding head, is to fine-tune the resulting language model in a supervised manner. That way it can adapt to the specific task and domain.
Using *BERT* with sufficient pre-training and fine-tuning Devlin et al. (2019) push many scores higher than any language model did before. They reach a *GLUE* score of 80.5% and thus beat the previous best result by 7.7%. Nowadays, *BERT* is often used in intent classification and question answering tasks. (Devlin et al., 2019)

GPT

Currently, the most popular text-generation language models are OpenAI's *GPT-2* and *GPT-3* models. As far as parameter count is concerned, *GPT-3* has around 175 billion parameters, which, by the time of its release, was 10x more than any other previous non-sparse language model. The largest and newest models are not downloadable for free. Instead, they are accessible via an API or a virtual playground, which can be visited in a web-browser. (Brown et al., 2020)
The architecture of *GPT*-like models is very similar to only the decoder part of the *transformer* architecture. The biggest difference between all the GPT versions is the parameter count of the models which has a big influence on the performance. (Brown et al., 2020)
These language models make also heavy use of unsupervised pre-training. For example, *GPT-3* was pre-trained on over 410 billion of tokens of the *Common Crawl* dataset and other resources, such as Wikipedia. (Brown et al., 2020)
Unlike *BERT*, no fine-tuning on domain specific tasks is necessary. Instead, a specially constructed prompt is prepended to each input sequence. For example, as far as text-summarization is concerned, an example text including the summarization is prepended to the real target text. Thereafter, the model completes the output similarly to the prompt example which essentially results in task-specific completion. (Brown et al., 2020)
Because of the text-generative nature, GPT language models shine at writing stories or having a conversation with a chat-partner. Brown et al. (2020) show that *GPT-3* is able to generate text, such as news, so that people find it difficult to tell whether it was written by a human or by a machine. However, in a

few-shot learning environment *GPT-3* almost reaches SOTA fine-tuned model or human performance on various benchmarks. Therefore, it can be also used for many downstream tasks such as text-summarization or question answering. (Brown et al., 2020)

Other

Because the new *GPT-3* models are very big in parameter size and only available through an external API, motivation of the open-source community of having a language model of comparable performance lead to the development of *GPT-Neo* (2.7M parameters) and *GPT-J* (6M parameters). Even though being freely available, they have drawbacks, such as the fact that they are trained on a more verbose dataset (*the pile*, etc.).
Furthermore, *GPT-3* is heavily biased by its training data. For example, achieving high-performance on German tasks is not always possible. Therefore, "Aleph Alpha" is training a European GPT-like language model, and it is currently available as beta.
Speaking of parameter size, it is not always beneficial for a production environment if the model size is big. This is the case, because a large number of parameters slows down the inference time, which may be a problem if the model must predict fast. This lead to the development of such called "distilled" language models, *DistilGPT2* being such an example. The goal is to construct the same architecture but to heavily reduce parameter size and train the model with knowledge distillation techniques. (Li et al., 2021)
Other than that, Raffel et al. (2020) show that combining the paradigms of the *BERT* encoder and *GPT* decoder can lead to a very promising text-to-text language model that can perform various tasks based on the prefix in the input sequence. For example, prepending the input paraphrase with "Summarize: " will lead to a text-summarization whereas prepending the input sequence with "Translate: " will translate the sentence. The currently available T5 models yield promising results. Raffel et al. (2020)
Even though other SOTA language models, such as *Turing-NLG* or *XLNet*, exist and can outperform existing solutions, such as *BERT*, they are not discussed any further in this paper as this would go out of scope.

4 Conclusion & future thoughts

History has shown that it takes some time until new approaches beat previously established solutions. However, beginning with the reveal of the *transformer* architecture, new neural language models start to beat benchmarks one after another. Care has to be taken when comparing architectures, because the constant grow in parameter size essentially leads to stronger performance. As a consequence, more money allows for better models because training large parameter models requires a lot of hardware resources, data and time.
However, new solutions, such as proposed by Borgeaud et al. (2022) arrive and

allow for almost equal performance compared to *GPT-3* with 25x fewer parameters.
Not only that, but text-generation language models are evolving so quickly that it is now possible to process pictures and text at once.

A picture of

a bear in a taxi.

Q: Where is the treasure buried?

The treasure is buried in the old oak tree.

Fig. 5: A language model can generate text if it is given a picture. Source: Aleph Alpha

For example, "Aleph Alpha" has trained such a model and an example can be seen in fig. 5. Given a picture and an input text, the model can complete the prompt. The picture on the right-hand side shows that the language model recognizes the context as a problem of finding a treasure. Furthermore, it locates the treasure and answers according to the question. This behaviour goes far beyond regular object detection.
Summing it up, with a high probability it is possible to tell that next-generation language models will further surprise the human-being, be it by beating previous benchmarks, by understanding complex contexts, such as in the previous example or by completely new skills.

Bibliography

Alammar, J. (2022). The Illustrated Transformer the illustrated transformer. https://jalammar.github.io/illustrated-transformer/. Accessed: 2022-01-19.

Bahdanau, D., K. Cho, and Y. Bengio (2016). Neural machine translation by jointly learning to align and translate.

Bahl, L. R., F. Jelinek, and R. L. Mercer (1983). A maximum likelihood approach to continuous speech recognition. *IEEE transactions on pattern analysis and machine intelligence* (2), 179–190.

Bengio, Y., R. Ducharme, and P. Vincent (2001). A neural probabilistic language model. In T. Leen, T. Dietterich, and V. Tresp (Eds.), *Advances in Neural Information Processing Systems*, Volume 13. MIT Press.

Borgeaud, S., A. Mensch, J. Hoffmann, T. Cai, E. Rutherford, K. Millican, G. van den Driessche, J.-B. Lespiau, B. Damoc, A. Clark, D. de Las Casas, A. Guy, J. Menick, R. Ring, T. Hennigan, S. Huang, L. Maggiore, C. Jones, A. Cassirer, A. Brock, M. Paganini, G. Irving, O. Vinyals, S. Osindero, K. Simonyan, J. W. Rae, E. Elsen, and L. Sifre (2022). Improving language models by retrieving from trillions of tokens.

Brown, T. B., B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei (2020). Language models are few-shot learners.

Devlin, J., M.-W. Chang, K. Lee, and K. Toutanova (2019). Bert: Pre-training of deep bidirectional transformers for language understanding.

Géron, A. (2019). *Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent* (2nd ed. ed.). O'Reilly Media.

Li, J., W. Monroe, A. Ritter, M. Galley, J. Gao, and D. Jurafsky (2016). Deep reinforcement learning for dialogue generation.

Li, T., Y. E. Mesbahi, I. Kobyzev, A. Rashid, A. Mahmud, N. Anchuri, H. Hajimolahoseini, Y. Liu, and M. Rezagholizadeh (2021). A short study on compressing decoder-based language models.

Mikolov, T., K. Chen, G. Corrado, and J. Dean (2013). Efficient estimation of word representations in vector space.

Pennington, J., R. Socher, and C. D. Manning (2014). Glove: Global vectors for word representation. In *EMNLP*, Volume 14, pp. 1532–1543.

Raffel, C., N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu (2020). Exploring the limits of transfer learning with a unified text-to-text transformer.

Shannon, C. E. (1948). A mathematical theory of communication. *The Bell System Technical Journal 27*, 379–423.

Vaswani, A., N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin (2017). Attention is all you need.

Wang, A., A. Singh, J. Michael, F. Hill, O. Levy, and S. R. Bowman (2019). Glue: A multi-task benchmark and analysis platform for natural language understanding.