Let the Algorithm Speak: How to Use Neural Networks for Automatic Item Generation in Psychological Scale Development

Supplementary Information 1: Using Statistical Language Models for Automatic Item Generation

April 2022

Statistical language modeling. The key idea of obtaining a numerical representation for a language, which can be viewed as a symbolic sequence, is to define an appropriate probability distribution of symbols (say *words*[1]) in a sequence (say a *sentence*) of that language (say *English*[2]) (Bengio et al., 2003). Let $\{w_1, w_2, \ldots, w_k\}$ indicate a sequence of words, then the joint distribution of the words sequence can be written as the product of conditional distributions of a word given all preceding words in the sequence:

$$P(w_1, w_2, \ldots, w_k) = P(w_1)P(w_2|w_1)P(w_3|w_1, w_2) \cdots P(w_k|w_1, \ldots, w_{k-1}). \tag{1}$$

The task of learning this joint distribution, which we refer to as the "statistical language model", can be termed as *training* the model. After the training stage, we can perform typical tasks of probabilistic modeling. This includes finding the most likely sequence conditioned on an "opening" set of words, or stochastically generating new sentences based on this model. This can be referred to as *sampling* from the model. Clearly, our primary interest is in the sampling step, for the purpose of automatic item generation. For clarity of exposition, we describe both steps in Sec. 1 and Sec. 2 with only the most important technical details, and refer the reader to the cited literature for more. Those interested in only applying GPT-2 appropriately for automatic item generation may skip directly to Sec. 2.

1 Training a statistical language model

Training data. To learn the language model, we first require a large corpus of text to serve as *training data*, which we believe to be representative of the language, or of a more specific domain of interest. For instance, if we would like a language model that can generate children's stories, we may learn it on a corpus of children's literature. Since word patterns and associations should transfer, at large, across domains, we can use a large but general text corpus to first *pre-train* the model, and then, if required, *fine-tune* it on a specialized text corpus or for a specific language task. This two-step training methodology was used in the first version of GPT (Radford et al., 2018), whose pre-training was performed on a corpus of 7,000 books (Zhu et al., 2015). If the pre-training corpus is sufficiently large, then the language models can learn many tasks without

[1]Symbols can be characters, or any arbitrary linguistic unit. GPT-2 uses byte-pair encoding (Sennrich et al., 2016) to yield its symbol vocabulary that interpolate between character and word levels.

[2]We remark that "language" could also refer to a combination of languages, which can be useful for multilingual tasks like language translation.

the need for any fine-tuning. The second iteration of GPT, i.e GPT-2 (Radford et al., 2019), used a new corpus of millions of webpages for its pre-training step, which allowed it to learn very rich word associations for a variety of downstream tasks.

1.1 The curse of dimensionality in language modeling

Given a text corpus, we can learn the language model in Eq. 1 in multiple ways, the simplest of which is maximum likelihood estimation (MLE). Observing the right hand side of Eq. 1, one notes that simply counting the number of occurrences of a word, conditioned on preceding words, yields the full joint distribution. Consider the sentence "`I am a human`", then $P(\texttt{I})$ is estimated as the proportion of words in the corpus that are "`I`", $P(\texttt{am}|\texttt{I})$ is estimated as the proportion of words following "`I`" that are "`am`", $P(\texttt{a}|\texttt{I}, \texttt{am})$ is estimated as the proportion of words following "`I am`" that are "`a`" and so on. However, due to the exponential number of word combinations in the size of the vocabulary, the MLE quickly runs into the "curse of dimensionality" as many of these counts will be simply zero.

n-gram language models. One way to reduce the dimensionality is to note that words that are closer will be statistically more dependent on each other. In particular, we can assume that the conditional distribution of w_k can only depend on a *context* set of $n-1$ words preceding it:

$$P(w_k|w_1, w_2, \ldots, w_{k-1}) \approx P(w_k|w_{k-n+1}, w_{k-n+2}, \ldots, w_{k-1}) = P(w_k|W_k^n), \tag{2}$$

where n, the context size, is much smaller than typical text lengths, and we define $W_k^n \triangleq \{w_{k-n+1}, w_{k-n+2}, \ldots, w_{k-1}\}$ to indicate the context set. Evidently, a small n makes learning easier but we lose on the expressivity of the model. If $n = 1$, then this yields a *unigram* or "bag-of-words" model, where all word associations are lost and the joint distribution in Eq. 1 is given by product of marginal distributions. This model would assign equal probabilities to any word-level permutations of a sentence—say to "`I am a human`" and "`a I human am`"—which is unlikely to be helpful for generating coherent text. Using a larger value of n, yielding so-called *n-gram* models, would be an improvement: say a *bigram* model with $n = 2$ would learn that $P(\texttt{am}|\texttt{I}) > P(\texttt{human}|\texttt{I})$. However, assigning non-zero probabilities to unseen word combinations would have to fall back on using smaller context sizes (Jelinek and Mercer, 1980).

Continuous word representations. As highlighted by the examples above, the key issue is that words (and more generally symbols) are intrinsically *discrete*, which makes generalization to unseen "out-of-sample" examples very difficult. Therefore, any "non-parametrized" language model will inevitably confront the curse of dimensionality. On the other hand, functions in continuous spaces can be "smooth" and are expected to generalize more easily. The solution then is to consider a parametrized probability model in Eq. 2

$$P(w_k|W_k^n) = f(W_k^n; \boldsymbol{\theta}), \tag{3}$$

such that parameters $\boldsymbol{\theta}$ characterize both the probability function, and the words in a *continuous* representation (Bengio et al., 2003). In particular, words can be represented as points in a d-dimensional vector space $\mathbb{R}^d$, such that words possessing semantic and/or syntactic similarity are somehow *closer* in $\mathbb{R}^d$. For example, if the word "`person`" is close to "`human`" in $\mathbb{R}^d$, then such a model will be able to assign a probability to "`I am a person`", even if it only observed "`I am a human`" during training.

1.2 Neural language models

Neural networks are highly expressive universal function approximators (Hornik et al., 1989), and have been the driving force behind the machine learning revolution (LeCun et al., 2015).

They can be used to model the distribution of words in Eq. 3 very effectively, termed as *neural language models*. Depending on their architecture, different neural networks can encode an arbitrary function $f(x;\boldsymbol{\theta})$ in different ways. A "feedforward neural network" (FFN) with l layers approximates $f(x;\boldsymbol{\theta})$ by iteratively composing simpler non-linear (and non-polynomial; Leshno et al., 1993) functions $g_1, g_2, \ldots, g_l$ with parameters $\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \ldots, \boldsymbol{\theta}_l$:

$$f(x;\boldsymbol{\theta}) \approx g_l(g_{l-1}(\cdots g_2(g_1(x;\boldsymbol{\theta}_1);\boldsymbol{\theta}_2)\cdots;\boldsymbol{\theta}_{l-1});\boldsymbol{\theta}_l). \tag{4}$$

If x indicates the context set W_k^n, then the first "input" layer g_1 will typically be a linear projection of discrete context words onto a continuous space using $\boldsymbol{\theta}_1$, which may (Bengio et al., 2003) or may not be followed by a non-linearity (Mikolov, Chen, et al., 2013). Since we want the output to be a distribution of w_k over the vocabulary, the final "output" layer g_l will typically be a linear projection from a continuous space to discrete words using $\boldsymbol{\theta}_l$, followed by the softmax function, thus indicating a multinomial distribution over words in the vocabulary. The parameters $\boldsymbol{\theta}_1$ and $\boldsymbol{\theta}_l$ are referred to as the *input* and *output* word embeddings respectively, and effectively encode the similarity between words (Mnih and Teh, 2012), as described in Sec. 1.1. Thus, even a two-layer FFN with no linearity can encode meaningful word relationships, such as that of capital cities to countries (Mikolov, Sutskever, et al., 2013), allowing the language model to generalize. The ability of a network to learn complicated functional relationships tends to increase with depth, i.e. with larger number of layers l (Telgarsky, 2016). Thus using "deep" neural networks, by adding more "hidden" layers in the middle, will allow the model to learn richer and more contextualized word and phrase relationships.

Recurrent neural network language models. Since the context size n determines the "width" of the input layer, it must be fixed *ad hoc* before training an FFN language model (Mikolov et al., 2010). But, a flexible use of longer contexts is evident in human language. For instance, if a character of a story is introduced early in the text, humans are capable of tracking any following pronouns with the said character, even after parsing many sentences. Thus, any n-gram model with small n may hinder more complex representations of language. To get around this problem, neural network architectures with time-delay connections, termed as "recurrent neural networks" (RNN; Elman, 1990), have been applied to language modeling. A typical RNN language model takes as input only one word at a time, modeling the probability of the next word w_k given the previous word w_{k-1} and a "memory" z_k of the entire context seen yet. That is, it attempts to model the conditional distribution $P(w_k|W_k^k)$ (as per Eq. 1) using a function $f(W_k^k;\boldsymbol{\theta})$. The memory state, generated as one of the outputs at each step alongside f, is fed back as an input to the next "layer" of the network:

$$z_k = g_1(w_k, z_{k-1};\boldsymbol{\theta}_1), \tag{5a}$$
$$f(W_k^k;\boldsymbol{\theta}) \approx g_2(z_k;\boldsymbol{\theta}_2). \tag{5b}$$

The function g_1 may be as simple as a linear projection of previous word w_k and memory z_{k-1} using $\boldsymbol{\theta}_1$, followed by a sigmoidal non-linearity[3] (Mikolov et al., 2010). More complicated recurrent architectures that use long-short term memory for g_1 (Graves, 2012; Hochreiter and Schmidhuber, 1997), and its variations (Cho et al., 2014), are able to learn richer representations for z_k by exploiting long-range word dependencies. Work in neural language translation also introduced an "attention" mechanism that allowed a word to automatically "attend to" different and selective parts of the sentence, including those further away, when generating a translation (Bahdanau et al., 2015). Until recently, these advances had firmly established RNN-based architectures as the state-of-the-art in various language modeling tasks. However, with growing size of text corpora, their drawbacks became more apparent. Since RNNs operate serially—one

[3]Note that the output of g_2 must be a valid distribution over the vocabulary. Typically, g_2 takes a linear projection of z_k using $\boldsymbol{\theta}_2$, and applies the softmax function to generate a valid distribution.

word at a time—they cannot be parallelized easily, which becomes a computational challenge especially when training on longer texts due to memory constraints (Vaswani et al., 2017).

The transformer language model. The current state-of-the-art neural language models, including GPT-2, are based on a deep network architecture called the *transformer* (Vaswani et al., 2017), that gets rid off recurrent connections and returns to a feedforward architecture akin to Eq. 4, but with significant differences.

1. Being an FFN, the context size must be fixed in a transformer too. But a much *larger* context size is used here, typically 512 as in GPT (Radford et al., 2018), or 1024 as in GPT-2 (Radford et al., 2019), which allows for longer dependencies than older FFN-based models.

2. The transformer can go very *deep*, with the architectures used in GPT-2 spanning 12–48 layers from the smallest to the largest model (Radford et al., 2019), offering them much higher learning capacity.

3. While the transformers ditch recurrence, they retain and enhance the use of *attention* in RNN-based language models by introducing a "multi-head" attention mechanism. Moreover, the network consists of *fully connected* layers, which allows a word to attend to *all* other words in a sentence, jointly across *multiple* word representations (Vaswani et al., 2017). Say, one attention head may correspond to word tenses, while another to parts-of-speech.

4. Since the architecture has no recurrence and only fully-connected layers, to incorporate information on the *ordering* of words the transformer adds "positional" embeddings to the input embeddings. For example, if every dimension of the positional embedding is a sinusoid whose frequencies form a geometric progression, then the model can learn to attend by *relative* word positions (Vaswani et al., 2017).

Altogether, having a wider, deeper, fully-connected feedforward architecture with multi-headed attention, alongside positional encodings for the words, allows for a very rich language model. While the number of parameters is quite large, training here is parallelizable, which allows one to learn a language model on very large text corpora relatively quickly when compared to RNN-based models.

1.3 The Generative Pre-trained Transformer-2 (GPT-2)

The Generative Pre-trained Transformer-2 is, as the name suggests, a neural language model based on the transformer architecture[4] described in Sec. 1.2. It exploits the scalability of the transformer by training with a large context size of $n = 1024$, on a very large corpus of 8 million web-scraped documents amounting to 40 GB of text, collected with an emphasis on document quality (Radford et al., 2019). It uses a context size of 1024 words, and uses four architectures spanning varying sizes: from $l = 12$ layers ($|\boldsymbol{\theta}| = 117$ million parameters and $d = 768$ word embedding dimensions) to $l = 48$ layers ($|\boldsymbol{\theta}| = 1.5$ billion, $d = 1600$) (Radford et al., 2019).

[4]The transformer was designed with the objective of language *translation*, and thus originally has an "encoder" component for the input language and "decoder" component for the output language. Since GPT-2 is pre-trained with a language modeling objective (Eq. 1), it drops the encoder part of the transformer entirely (Radford et al., 2018). Thus, strictly speaking, GPT-2 is based on the *transformer decoder* architecture (Liu et al., 2018).

Training objective. Since GPT-2 aims to learn a statistical language model, its training objective is to maximize the probability of word sequences seen in the training corpus $\mathcal{W} \triangleq \{w_1, w_2, \ldots, w_k\}$. Using Eqs. 1 and 2, this is equivalent to maximizing the following log-likelihood function:

$$\mathcal{L}(\boldsymbol{\theta}; \mathcal{W}) \triangleq \sum_{i=1}^{k} \log P(w_i | W_i^n; \boldsymbol{\theta}), \tag{6}$$

with respect to $\boldsymbol{\theta}$:

$$\hat{\boldsymbol{\theta}} = \arg\max_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}; \mathcal{W}). \tag{7}$$

Let $\mathcal{V} \triangleq \{v_1, v_2, \ldots, v_m\}$ be the vocabulary of all words, then the final softmax layer of the network is a probability vector of size m. Denoting the jth component of the network's output by f_j, and the indicator function as $[\cdot]$, we can rewrite Eq. 6 using Eq. 3 as:

$$\mathcal{L}(\boldsymbol{\theta}; \mathcal{W}) = \sum_{i=1}^{k} \sum_{j=1}^{m} [w_i = v_j] \log f_j(W_i^n; \boldsymbol{\theta}), \tag{8}$$

where f is the function modeled by the neural network, as exemplified in Eq. 4. Maximizing Eq. 8 with respect to $\boldsymbol{\theta}$ is the network training objective (Eq. 7). Evidently, finding the optimal value $\hat{\boldsymbol{\theta}}$ in a space of close to a billion parameters is a computational challenge, which further highlights the utility of "pre-training" a network language model like the GPT-2 once, and not having to "fine-tune" it for a new task. Gradient descent methods, in particular *stochastic* gradient descent (SGD; Robbins and Monro, 1951) that approximates gradients using data subsamples, have been crucial to the success with high-dimensional data and parameters in contemporary machine learning. GPT-2 uses the "Adam" optimization scheme (Kingma and Ba, 2015), a computationally efficient algorithm for SGD, to maximize the log-likelihood in Eq. 8.

2 Sampling from a statistical language model

Once a statistical language model like the GPT-2 is trained, generating text from it involves exploring different sampling strategies. We refer the reader to Holtzman et al., 2020 for a detailed discussion on sampling strategies, but also provide a self-contained discussion in this section.

Let $\{c_1, c_2, \ldots, c_q\}$ be a conditioning text of q words, also termed as the "prefix", which will be used to generate the rest of the words $\{c_{q+1}, c_{q+2}, \ldots, c_{q+r}\}$, yielding the full text $\{c_1, c_2, \ldots, c_q, c_{q+1}, c_{q+2}, \ldots, c_{q+r}\}$. As before, let $C_i^n \triangleq \{c_{i-n+1}, c_{i-n+2}, \ldots, c_{i-1}\}$ denote the context set of $n-1$ words preceding c_i. We remark that the prefix can be empty, in which case the sampling will have no prior knowledge to condition on. However, since we use GPT-2 off-the-shelf, i.e. without any fine-tuning, the only way to have it generate contextually relevant text is to provide a good context. For instance, to generate a story about a medical incident, one might provide the prefix "`Yesterday, I had to go to the doctor's because`".

Having provided some context, the statistical language model can now be used to generate new text. Let $\mathcal{V} \triangleq \{v_1, v_2, \ldots, v_m\}$ be the vocabulary of all words of size m, and denote the jth component of the network's output by f_j. From Eq. 3, we get for $q < i \leq q + r$, the probability of c_i to be v_j as:

$$P(c_i = v_j | C_k^n; \hat{\boldsymbol{\theta}}) = f_j(C_i^n; \hat{\boldsymbol{\theta}}). \tag{9}$$

For c_1, the entire context set C_1^n is provided by the prefix[5]. Using some sampling strategy that makes use of Eq. 9, we can generate a value for c_1 from $\mathcal{V}$, which can now feature in the

[5]We remark that the prefix size can be shorter than the context size n, in which case the prefix is simply "padded up" with empty tokens.

context set for c_2. Thus, we can successively apply a sampling strategy to generate the text, one word at a time. The choice of sampling strategy can make all the difference between generating meaningful versus incoherent text. Below, we discuss some strategies, and describe the one used in our experiments.

2.1 Sampling strategies

Greedy decoding. Since the language model is trained on the objective to maximize the conditional likelihood of a word given preceding words, one simple strategy is to generate the word which offers the largest probability:

$$c_i = \arg\max_{v \in \mathcal{V}} P(v | C_i^n; \hat{\boldsymbol{\theta}}). \tag{10}$$

Using Eq. 10 is often referred to as "greedy decoding", as it deterministically picks the most probable next word, given the context yet. Unfortunately, strategies that aim to maximize the probability tend to generate highly degenerate and repetitive text, even when using a state-of-the-art language model like the GPT-2 (Holtzman et al., 2020).

Pure sampling. Alternatively, one could pursue a purely stochastic strategy, and generate the next word by sampling the next word from a size m categorical distribution whose probability vector is given by the language model:

$$c_i \sim \text{Categorical}\left(\left\{P(c_i = v_j | C_i^n; \hat{\boldsymbol{\theta}})\right\}_{j=1}^{m}\right). \tag{11}$$

This is referred to as a "pure sampling" strategy, and may be interpreted as the other extreme of greedy decoding. While this stochastic strategy produces a diverse sequence of words, thus getting around the problem of generating repetitive and dull text, the text it generates now risks being incoherent. The reason comes down to a long "unreliable" tail of the distribution, wherein *many* words tend to have a roughly similar *low* probability (Holtzman et al., 2020).

Top-k sampling. One solution around this problem is to consider only the top "most probable" words in the vocabulary to sample from (Fan et al., 2018), such that it becomes unlikely to sample a word from the "unreliable" tail of low probabilities. Formally, consider the probability of the next word to be in a vocabulary subset $\mathcal{V}' \subset \mathcal{V}$:

$$P(c_i \in \mathcal{V}' | C_i^n; \hat{\boldsymbol{\theta}}) = \sum_{v \in \mathcal{V}'} P(c_i = v | C_i^n; \hat{\boldsymbol{\theta}}). \tag{12}$$

Define the truncated distribution:

$$\widetilde{P}_{\mathcal{V}'}(c_i = v_j | C_i^n; \hat{\boldsymbol{\theta}}) = \begin{cases} \frac{P(c_i = v_j | C_i^n; \hat{\boldsymbol{\theta}})}{P(c_i \in \mathcal{V}' | C_i^n; \hat{\boldsymbol{\theta}})} & \text{if } v_j \in \mathcal{V}', \\ 0 & \text{otherwise.} \end{cases} \tag{13}$$

Let $\mathcal{V}_k \subset \mathcal{V}$ be a vocabulary subset of size k (where $1 \leq k \leq m$) that maximizes $P(c_i \in \mathcal{V}_k | C_i^n; \hat{\boldsymbol{\theta}})$ over the set of all subsets of $\mathcal{V}$ of size k. Then using the truncated distribution $\widetilde{P}_{\mathcal{V}_k}(c_i = v_j | C_i^n; \hat{\boldsymbol{\theta}})$ from Eq. 13 instead of $P(c_i = v_j | C_i^n; \hat{\boldsymbol{\theta}})$ in Eq. 11 for sampling the next word is referred to as "top-k sampling". We remark that setting $k = 1$ will imply greedy decoding, whereas setting $k = m$ (the vocabulary size) will imply pure sampling. Therefore, top-k sampling can be seen as an interpolation between the two extremes of pure determinism and pure stochasticity, and generates considerably higher quality text than either (Holtzman et al., 2020). However, the exact choice of k can be crucial to the sampling performance. The shape of the distribution will directly impact performance for any given k. In particular, a flat distribution will have its probability mass "spread out", and a small k will exclude many equally valid words from being sampled. A peaked distribution will have its probability mass "concentrated", and a large k may inadvertently increase probability mass on low-probability words.

Nucleus sampling. To get around the issues of choosing k, we can directly work with the *shape* of the distribution, and pick a suitable k_i for every word c_i depending on the spread of $P(c_i|C_i^n;\hat{\boldsymbol{\theta}})$. By picking a "nucleus" of k_i words on which most probability mass is concentrated, say no less than p (where $0 < p < 1$), top-k_i sampling should yield a good performance. More formally, let $\mathcal{V}^p \subset \mathcal{V}$ be a vocabulary subset of smallest size over all subsets of $\mathcal{V}$ with the property $P(c_i \in \mathcal{V}^p|C_i^n;\hat{\boldsymbol{\theta}}) \geq p$. Using the truncated distribution $\widetilde{P}_{\mathcal{V}^p}(c_i = v_j|C_i^n;\hat{\boldsymbol{\theta}})$ from Eq. 13 instead of $P(c_i = v_j|C_i^n;\hat{\boldsymbol{\theta}})$ in Eq. 11 for sampling the next word is referred to as "nucleus sampling". This implies sampling from the most probable words whose cumulative probability mass exceeds p (Holtzman et al., 2020). Larger (smaller) values of p will be closer to pure sampling (greedy decoding), and picking a suitable value can strike the right balance between coherency and diversity.

Temperature sampling. An alternative solution around the problem of picking a suitable k is to reshape the distribution in a manner that preserves the relative ordering of word probabilities, but sharpens up the distribution. One way to do this is to take a power of word probabilities, and renormalizing them to 1 so as to render them as a valid distribution:

$$\overline{P}_t(c_i = v_j|C_i^n;\hat{\boldsymbol{\theta}}) = \frac{\exp\left(\log\left(P(c_i = v_j|C_i^n;\hat{\boldsymbol{\theta}})\right)/t\right)}{\sum_{j=1}^{m}\exp\left(\log\left(P(c_i = v_j|C_i^n;\hat{\boldsymbol{\theta}})\right)/t\right)}. \tag{14}$$

Using $\overline{P}_t(c_i = v_j|C_i^n;\hat{\boldsymbol{\theta}})$ from Eq. 14 instead of $P(c_i = v_j|C_i^n;\hat{\boldsymbol{\theta}})$ in Eq. 11 for sampling the next word is referred to as "temperature sampling" (Ackley et al., 1985; Holtzman et al., 2020), where the temperature t controls the "sharpness". In particular, $t = 0$ will yield greedy decoding, while $t = 1$ yields pure sampling. Picking $0 \leq t < 1$ while sharpen the distribution toward higher probability words, while choosing $t > 1$ while flatten it, with the extreme of $t = \infty$ yielding a uniform distribution over the entire vocabulary. Thus, a good selection of t to balance diversity and quality will be somewhere close to but smaller than 1.

2.2 Recommendations for automatic item generation

The ultimate goal for us is not to "fine-tune" the GPT-2 to generate items for psychometric tests, rather to make use of effective "sampling" strategies, with the right conditioning prefix, to generate new items of high coherence, and yet exhibiting some novelty. Sec. 2.1 describes the advantage of using nucleus sampling, perhaps in conjunction with temperature-based reshaping, to generate items from GPT-2. This requires choosing a suitable value of p (for nucleus sampling) and t (for temperature sampling).

One way of doing this is to pick some "default" values, generate text, and see for oneself if the items are of appropriate quality as per expert judgement. Holtzman et al., 2020 note that setting $t < 0.9$ induces repetition, and setting $p = 0.95$ yields textual diversity (measured via "perplexity") comparable to human-generated text. These are the default values we use in our experiments. If the generated text seems low on creativity, then increasing p and/or t can encourage a creative flair. On the other hand, if the text seems diverse but incoherent, then reducing p and/or t can encourage higher quality.

References

Ackley, D. H., Hinton, G. E., & Sejnowski, T. J. (1985). A learning algorithm for boltzmann machines. *Cognitive Science*, *9*(1), 147–169. https://doi.org/10.1016/S0364-0213(85)80012-4

Bahdanau, D., Cho, K., & Bengio, Y. (2015). Neural machine translation by jointly learning to align and translate. *CoRR*, *abs/1409.0473*.

Bengio, Y., Ducharme, R., Vincent, P., & Janvin, C. (2003). A Neural Probabilistic Language Model. *Journal of Machine Learning Research*, *3*, 1137–1155.

Cho, K., van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014). Learning phrase representations using RNN encoder–decoder for statistical machine translation. *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 1724–1734. https://doi.org/10.3115/v1/D14-1179

Elman, J. L. (1990). Finding structure in time. *Cognitive Science*, *14*(2), 179–211. https://doi.org/10.1207/s15516709cog1402_1

Fan, A., Lewis, M., & Dauphin, Y. (2018). *Hierarchical Neural Story Generation* [Preprint]. https://arxiv.org/abs/1805.04833

Graves, A. (2012). Supervised sequence labelling with recurrent neural networks. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-24797-2

Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, *9*(8), 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735

Holtzman, A., Buys, J., Du, L., Forbes, M., & Choi, Y. (2020). The Curious Case of Neural Text Degeneration. *International Conference on Learning Representations.* https://openreview.net/pdf?id=rygGQyrFvH

Hornik, K., Stinchcombe, M., & White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*, *2*(5), 359–366. https://doi.org/10.1016/0893-6080(89)90020-8

Jelinek, F., & Mercer, R. (1980). Interpolated estimation of Markov source parameters from sparse data. *Proc. Workshop on Pattern Recognition in Practice.*

Kingma, D. P., & Ba, J. (2015). Adam: A method for stochastic optimization. *CoRR*, *abs/1412.6980.*

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, *521*(7553), 436–444. https://doi.org/10.1038/nature14539

Leshno, M., Lin, V. Y., Pinkus, A., & Schocken, S. (1993). Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural Networks*, *6*(6), 861–867. https://doi.org/10.1016/S0893-6080(05)80131-5

Liu, P. J., Saleh, M., Pot, E., Goodrich, B., Sepassi, R., Kaiser, L., & Shazeer, N. (2018). Generating Wikipedia by Summarizing Long Sequences. *International Conference on Learning Representations.* https://openreview.net/pdf?id=Hyg0vbWC-

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

Mikolov, T., Karafiát, M., Burget, L., Cernocký, J. H., & Khudanpur, S. (2010). Recurrent neural network based language model. *INTERSPEECH.*

Mikolov, T., Sutskever, I., Chen, K., Corrado, G., & Dean, J. (2013). Distributed representations of words and phrases and their compositionality. *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*, 3111–3119.

Mnih, A., & Teh, Y. W. (2012). A fast and simple algorithm for training neural probabilistic language models. *In Proceedings of the International Conference on Machine Learning.*

Radford, A., Narasimhan, K., Salimans, T., & Sutskever, I. (2018). *Improving Language Understanding by Generative Pre-training* [Preprint]. https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., & Sutskever, I. (2019). *Language Models are Unsupervised Multitask Learners* [Preprint]. https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf

Robbins, H., & Monro, S. (1951). A stochastic approximation method. *The Annals of Mathematical Statistics*, *22*(3), 400–407. http://www.jstor.org/stable/2236626

Sennrich, R., Haddow, B., & Birch, A. (2016). Neural machine translation of rare words with subword units. *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 1715–1725. https://doi.org/10.18653/v1/P16-1162

Telgarsky, M. (2016). Benefits of depth in neural networks. *29th Annual Conference on Learning Theory*, *49*, 1517–1539. https://proceedings.mlr.press/v49/telgarsky16.html

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is All You Need. *Proceedings of the 31st International Conference on Neural Information Processing Systems*, 6000–6010.

Zhu, Y., Kiros, R., Zemel, R., Salakhutdinov, R., Urtasun, R., Torralba, A., & Fidler, S. (2015). Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. *2015 IEEE International Conference on Computer Vision (ICCV)*, 19–27. https://doi.org/10.1109/ICCV.2015.11