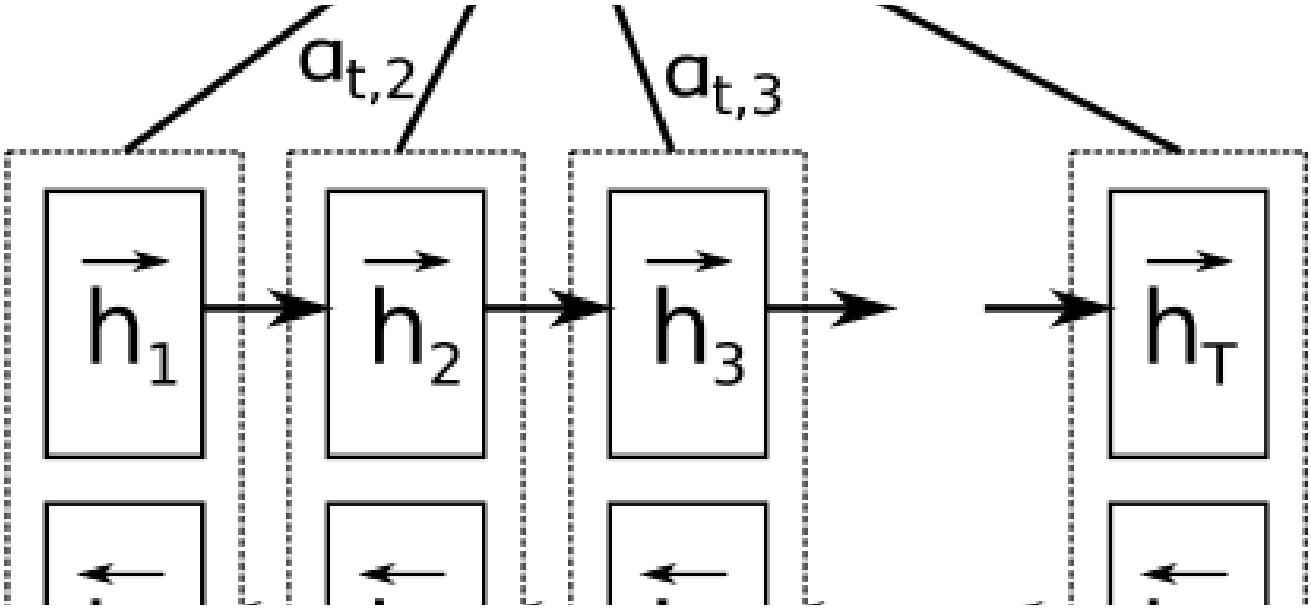
25 JULY 2017 / NATURAL LANGUAGE PROCESSING

Deep Learning for NLP Best Practices



This post gives an overview of best practices relevant for most tasks in natural language processing.

Update July 26, 2017: For additional context, the HackerNews discussion about this post.

Table of contents:

Introduction

This post is a collection of best practices for using neural networks in Natural Language Processing. It will be updated periodically as new insights become available and in order to keep track of our evolving understanding of Deep Learning for NLP.

There has been a running joke in the NLP community that an LSTM with attention will yield state-of-the-art performance on any task. While this has been true over the course of the last two years, the

NLP community is slowly moving away from this now standard baseline and towards more interesting models.

However, we as a community do not want to spend the next two years independently (re-)discovering the *next* LSTM with attention. We do not want to reinvent tricks or methods that have already been shown to work. While many existing Deep Learning libraries already encode best practices for working with neural networks in general, such as initialization schemes, many other details, particularly task or domain-specific considerations, are left to the practitioner.

This post is not meant to keep track of the state-of-the-art, but rather to collect best practices that are relevant for a wide range of tasks. In other words, rather than describing one particular architecture, this post aims to collect the features that underly successful architectures. While many of these features will be most useful for pushing the state-of-the-art, I hope that wider knowledge of them will lead to stronger evaluations, more meaningful comparison to baselines, and inspiration by shaping our intuition of what works.

I assume you are familiar with neural networks as applied to NLP (if not, I recommend Yoav Goldberg's excellent primer [1]) and are interested in NLP in general or in a particular task. The main goal of this article is to get you up to speed with the relevant best practices so you can make meaningful contributions as soon as possible.

I will first give an overview of best practices that are relevant for most tasks. I will then outline practices that are relevant for the most common tasks, in particular classification, sequence labelling, natural language generation, and neural machine translation.

Disclaimer: Treating something as *best practice* is notoriously

difficult: Best according to what? What if there are better alternatives? This post is based on my (necessarily incomplete) understanding and experience. In the following, I will only discuss practices that have been reported to be beneficial independently by *at least* two different groups. I will try to give at least two references for each best practice.

Best practices

Word embeddings

Word embeddings are arguably the most widely known best practice in the recent history of NLP. It is well-known that using pre-trained embeddings helps (Kim, 2014) [2]. The optimal dimensionality of word embeddings is mostly task-dependent: a smaller dimensionality works better for more syntactic tasks such as named entity recognition (Melamud et al., 2016) [3] or part-of-speech (POS) tagging (Plank et al., 2016) [4], while a larger dimensionality is more useful for more semantic tasks such as sentiment analysis (Ruder et al., 2016) [5].

Depth

While we will not reach the depths of computer vision for a while, neural networks in NLP have become progressively deeper. State-of-the-art approaches now regularly use deep Bi-LSTMs, typically consisting of 3-4 layers, e.g. for POS tagging (Plank et al., 2016) and semantic role labelling (He et al., 2017) [6]. Models for some tasks can be even deeper, cf. Google's NMT model with 8 encoder and 8 decoder layers (Wu et al., 2016) [7]. In most cases, however,

performance improvements of making the model deeper than 2 layers are minimal (Reimers & Gurevych, 2017) [8].

These observations hold for most sequence tagging and structured prediction problems. For classification, deep or very deep models perform well only with character-level input and shallow word-level models are still the state-of-the-art (Zhang et al., 2015; Conneau et al., 2016; Le et al., 2017) [9] [10] [11].

Layer connections

For training deep neural networks, some tricks are essential to avoid the vanishing gradient problem. Different layers and connections have been proposed. Here, we will discuss three: i) highway layers, ii) residual connections, and iii) dense connections.

Highway layers Highway layers (Srivastava et al., 2015) [12] are inspired by the gates of an LSTM. First let us assume a one-layer MLP, which applies an affine transformation followed by a non-linearity g to its input $\mathbf{x}$:

$$\mathbf{h} = g(\mathbf{W}\mathbf{x} + \mathbf{b})$$

A highway layer then computes the following function instead:

$$\mathbf{h} = \mathbf{t} \odot g(\mathbf{W}\mathbf{x} + \mathbf{b}) + (1 - \mathbf{t}) \odot \mathbf{x}$$

where $\odot$ is elementwise multiplication, $\mathbf{t} = \sigma(\mathbf{W}_T\mathbf{x} + \mathbf{b}_T)$ is called the *transform* gate, and $(1 - \mathbf{t})$ is called the *carry* gate. As we can see, highway layers are similar to the gates of an LSTM in that they adaptively *carry* some dimensions of the input directly to the output.

Highway layers have been used pre-dominantly to achieve state-of-the-art results for language modelling (Kim et al., 2016; Jozefowicz et al., 2016; Zilly et al., 2017) [13] [14] [15], but have also been used for other tasks such as speech recognition (Zhang et al., 2016) [16]. Sristava's page contains more information and code regarding highway layers.

Residual connections Residual connections (He et al., 2016) [17] have been first proposed for computer vision and were the main factor for winning ImageNet 2016. Residual connections are even more straightforward than highway layers and learn the following function:

$$\mathbf{h} = g(\mathbf{W}\mathbf{x} + \mathbf{b}) + \mathbf{x}$$

which simply adds the input of the current layer to its output via a short-cut connection. This simple modification mitigates the vanishing gradient problem, as the model can default to using the identity function if the layer is not beneficial.

Dense connections Rather than just adding layers from each layer to the next, dense connections (Huang et al., 2017) [18] (best paper award at CVPR 2017) add direct connections from each layer to all subsequent layers. Let us augment the layer output h and layer input x with indices l indicating the current layer. Dense connections then feed the concatenated output from all previous layers as input to the current layer:

$$\mathbf{h}^l = g(\mathbf{W}[\mathbf{x}^1; \ldots; \mathbf{x}^l] + \mathbf{b})$$

where $[\cdot;\cdot]$ represents concatenation. Dense connections have been successfully used in computer vision. They have also found to be useful for Multi-Task Learning of different NLP tasks (Ruder et al.,

2017) [19], while a residual variant that uses summation has been shown to consistently outperform residual connections for neural machine translation (Britz et al., 2017) [20].

Dropout

While batch normalisation in computer vision has made other regularizers obsolete in most applications, dropout (Srivasta et al., 2014) [21] is still the go-to regularizer for deep neural networks in NLP. A dropout rate of 0.5 has been shown to be effective in most scenarios (Kim, 2014). In recent years, variations of dropout such as adaptive (Ba & Frey, 2013) [22] and evolutional dropout (Li et al., 2016) [23] have been proposed, but none of these have found wide adoption in the community. The main problem hindering dropout in NLP has been that it could not be applied to recurrent connections, as the aggregating dropout masks would effectively zero out embeddings over time.

Recurrent dropout Recurrent dropout (Gal & Ghahramani, 2016) [24] addresses this issue by applying the same dropout mask across timesteps at layer l. This avoids amplifying the dropout noise along the sequence and leads to effective regularization for sequence models. Recurrent dropout has been used for instance to achieve state-of-the-art results in semantic role labelling (He et al., 2017) and language modelling (Melis et al., 2017) [25].

Multi-task learning

If additional data is available, multi-task learning (MTL) can often be used to improve performance on the target task. Have a look this blog post for more information on MTL.

Auxiliary objectives We can often find auxiliary objectives that are useful for the task we care about (Ruder, 2017) [26]. While we can already predict surrounding words in order to pre-train word embeddings (Mikolov et al., 2013), we can also use this as an auxiliary objective during training (Rei, 2017) [27]. A similar objective has also been used by (Ramachandran et al., 2016) [28] for sequence-to-sequence models.

Task-specific layers While the standard approach to MTL for NLP is hard parameter sharing, it is beneficial to allow the model to learn task-specific layers. This can be done by placing the output layer of one task at a lower level (Søgaard & Goldberg, 2016) [29]. Another way is to induce private and shared subspaces (Liu et al., 2017; Ruder et al., 2017) [30] [19:1].

Attention

Attention is most commonly used in sequence-to-sequence models to attend to encoder states, but can also be used in any sequence model to look back at past states. Using attention, we obtain a context vector $\mathbf{c}_i$ based on hidden states $\mathbf{s}_1, \ldots, \mathbf{s}_m$ that can be used together with the current hidden state $\mathbf{h}_i$ for prediction. The context vector $\mathbf{c}_i$ at position is calculated as an average of the previous states weighted with the attention scores $\mathbf{a}_i$:

$$\mathbf{c}_i = \sum_j a_{ij} \mathbf{s}_j$$
$$\mathbf{a}_i = \text{softmax}(f_{att}(\mathbf{h}_i, \mathbf{s}_j))$$

The attention function $f_{att}(\mathbf{h}_i, \mathbf{s}_j)$ calculates an unnormalized alignment score between the current hidden state $\mathbf{h}_i$ and the previous hidden state $\mathbf{s}_j$. In the following, we will discuss four

attention variants: i) additive attention, ii) multiplicative attention, iii) self-attention, and iv) key-value attention.

Additive attention The original attention mechanism (Bahdanau et al., 2015) [31] uses a one-hidden layer feed-forward network to calculate the attention alignment:

$$f_{att}(\mathbf{h}_i, \mathbf{s}_j) = \mathbf{v}_a{}^\top \tanh(\mathbf{W}_a[\mathbf{h}_i; \mathbf{s}_j])$$

where $\mathbf{v}_a$ and $\mathbf{W}_a$ are learned attention parameters. Analogously, we can also use matrices $\mathbf{W}_1$ and $\mathbf{W}_2$ to learn separate transformations for $\mathbf{h}_i$ and $\mathbf{s}_j$ respectively, which are then summed:

$$f_{att}(\mathbf{h}_i, \mathbf{s}_j) = \mathbf{v}_a{}^\top \tanh(\mathbf{W}_1\mathbf{h}_i + \mathbf{W}_2\mathbf{s}_j)$$

Multiplicative attention Multiplicative attention (Luong et al., 2015) [32] simplifies the attention operation by calculating the following function:

$$f_{att}(h_i, s_j) = h_i^\top \mathbf{W}_a s_j$$

Additive and multiplicative attention are similar in complexity, although multiplicative attention is faster and more space-efficient in practice as it can be implemented more efficiently using matrix multiplication. Both variants perform similar for small dimensionality d_h of the decoder states, but additive attention performs better for larger dimensions. One way to mitigate this is to scale $f_{att}(\mathbf{h}_i, \mathbf{s}_j)$ by $1/\sqrt{d_h}$ (Vaswani et al., 2017) [33].

Attention cannot only be used to attend to encoder or previous hidden states, but also to obtain a distribution over other features, such as the word embeddings of a text as used for reading comprehension (Kadlec et al., 2017) [34]. However, attention is not

directly applicable to classification tasks that do not require additional information, such as sentiment analysis. In such models, the final hidden state of an LSTM or an aggregation function such as max pooling or averaging is often used to obtain a sentence representation.

Self-attention Without any additional information, however, we can still extract relevant aspects from the sentence by allowing it to attend to itself using self-attention (Lin et al., 2017) [35]. Self-attention, also called intra-attention has been used successfully in a variety of tasks including reading comprehension (Cheng et al., 2016) [36], textual entailment (Parikh et al., 2016) [37], and abstractive summarization (Paulus et al., 2017) [38].

We can simplify additive attention to compute the unnormalized alignment score for each hidden state $\mathbf{h}_i$:

$$f_{att}(\mathbf{h}_i) = \mathbf{v}_a{}^\top \tanh(\mathbf{W}_a \mathbf{h}_i)$$

In matrix form, for hidden states $\mathbf{H} = \mathbf{h}_1, \ldots, \mathbf{h}_n$ we can calculate the attention vector $\mathbf{a}$ and the final sentence representation $\mathbf{c}$ as follows:

$$\mathbf{a} = \text{softmax}(\mathbf{v}_a \tanh(\mathbf{W}_a \mathbf{H}^\top))$$
$$\mathbf{c} = \mathbf{H}\mathbf{a}^\top$$

Rather than only extracting one vector, we can perform several hops of attention by using a matrix $\mathbf{V}_a$ instead of $\mathbf{v}_a$, which allows us to extract an attention matrix $\mathbf{A}$:

$$\mathbf{A} = \text{softmax}(\mathbf{V}_a \tanh(\mathbf{W}_a \mathbf{H}^\top))$$
$$\mathbf{C} = \mathbf{A}\mathbf{H}$$

In practice, we enforce the following orthogonality constraint to penalize redundancy and encourage diversity in the attention vectors in the form of the squared Frobenius norm:

$$\Omega = |(\mathbf{A}\mathbf{A}^\top - \mathbf{I}|_F^2$$

A similar multi-head attention is also used by Vaswani et al. (2017).

Key-value attention Finally, key-value attention (Daniluk et al., 2017) [39] is a recent attention variant that separates form from function by keeping separate vectors for the attention calculation. It has also been found useful for different document modelling tasks (Liu & Lapata, 2017) [40]. Specifically, key-value attention splits each hidden vector $\mathbf{h}_i$ into a key $\mathbf{k}_i$ and a value $\mathbf{v}_i$: $[\mathbf{k}_i; \mathbf{v}_i] = \mathbf{h}_i$. The keys are used for calculating the attention distribution $\mathbf{a}_i$ using additive attention:

$$\mathbf{a}_i = \text{softmax}(\mathbf{v}_a{}^\top \tanh(\mathbf{W}_1[\mathbf{k}_{i-L}; \ldots; \mathbf{k}_{i-1}] + (\mathbf{W}_2\mathbf{k}_i)\mathbf{1}^\top))$$

where L is the length of the attention window and $\mathbf{1}$ is a vector of ones. The values are then used to obtain the context representation $\mathbf{c}_i$:

$$\mathbf{c}_i = [\mathbf{v}_{i-L}; \ldots; \mathbf{v}_{i-1}]\mathbf{a}^\top$$

The context $\mathbf{c}_i$ is used together with the current value $\mathbf{v}_i$ for prediction.

Optimization

The optimization algorithm and scheme is often one of the parts of the model that is used as-is and treated as a black-box. Sometimes, even slight changes to the algorithm, e.g. reducing the β_2 value in

Adam (Dozat & Manning, 2017) [41] can make a large difference to the optimization behaviour.

Optimization algorithm Adam (Kingma & Ba, 2015) [42] is one of the most popular and widely used optimization algorithms and often the go-to optimizer for NLP researchers. It is often thought that Adam clearly outperforms vanilla stochastic gradient descent (SGD). However, while it converges much faster than SGD, it has been observed that SGD with learning rate annealing slightly outperforms Adam (Wu et al., 2016). Recent work furthermore shows that SGD with properly tuned momentum outperforms Adam (Zhang et al., 2017) [43].

Optimization scheme While Adam internally tunes the learning rate for every parameter (Ruder, 2016) [44], we can explicitly use SGD-style annealing with Adam. In particular, we can perform learning rate annealing with restarts: We set a learning rate and train the model until convergence. We then halve the learning rate and restart by loading the previous best model. In Adam's case, this causes the optimizer to forget its per-parameter learning rates and start fresh. Denkowski & Neubig (2017) [45] show that Adam with 2 restarts and learning rate annealing is faster and performs better than SGD with annealing.

Ensembling

Combining multiple models into an ensemble by averaging their predictions is a proven strategy to improve model performance. While predicting with an ensemble is expensive at test time, recent advances in distillation allow us to compress an expensive ensemble into a much smaller model (Hinton et al., 2015; Kuncoro et al., 2016; Kim & Rush, 2016) [46] [47] [48].

Ensembling is an important way to ensure that results are still reliable if the diversity of the evaluated models increases (Denkowski & Neubig, 2017). While ensembling different checkpoints of a model has been shown to be effective (Jean et al., 2015; Sennrich et al., 2016) [49] [50], it comes at the cost of model diversity. Cyclical learning rates can help to mitigate this effect (Huang et al., 2017) [51]. However, if resources are available, we prefer to ensemble multiple independently trained models to maximize model diversity.

Hyperparameter optimization

Rather than pre-defining or using off-the-shelf hyperparameters, simply tuning the hyperparameters of our model can yield significant improvements over baselines. Recent advances in Bayesian Optimization have made it an ideal tool for the black-box optimization of hyperparameters in neural networks (Snoek et al., 2012) [52] and far more efficient than the widely used grid search. Automatic tuning of hyperparameters of an LSTM has led to state-of-the-art results in language modeling, outperforming models that are far more complex (Melis et al., 2017).

LSTM tricks

Learning the initial state We generally initialize the initial LSTM states with a 0 vector. Instead of fixing the initial state, we can learn it like any other parameter, which can improve performance and is also recommended by Hinton. Refer to this blog post for a Tensorflow implementation.

Tying input and output embeddings Input and output embeddings account for the largest number of parameters in the LSTM model. If the LSTM predicts words as in language modelling, input and output parameters can be shared (Inan et al., 2016; Press & Wolf, 2017) [53] [54]. This is particularly useful on small datasets that do not allow to learn a large number of parameters.

Gradient norm clipping One way to decrease the risk of exploding gradients is to clip their maximum value (Mikolov, 2012) [55]. This, however, does not improve performance consistently (Reimers & Gurevych, 2017). Rather than clipping each gradient independently, clipping the global norm of the gradient (Pascanu et al., 2013) [56] yields more significant improvements (a Tensorflow implementation can be found here).

Down-projection To reduce the number of output parameters further, the hidden state of the LSTM can be projected to a smaller size. This is useful particularly for tasks with a large number of outputs, such as language modelling (Melis et al., 2017).

Task-specific best practices

In the following, we will discuss task-specific best practices. Most of these perform best for a particular type of task. Some of them might still be applied to other tasks, but should be validated before. We will discuss the following tasks: classification, sequence labelling, natural language generation (NLG), and -- as a special case of NLG -- neural machine translation.

Classification

More so than for sequence tasks, where CNNs have only recently found application due to more efficient convolutional operations, CNNs have been popular for classification tasks in NLP. The following best practices relate to CNNs and capture some of their optimal hyperparameter choices.

CNN filters Combining filter sizes near the optimal filter size, e.g. (3,4,5) performs best (Kim, 2014; Kim et al., 2016). The optimal number of feature maps is in the range of 50-600 (Zhang & Wallace, 2015) [57].

Aggregation function 1-max-pooling outperforms average-pooling and k-max pooling (Zhang & Wallace, 2015).

Sequence labelling

Sequence labelling is ubiquitous in NLP. While many of the existing best practices are with regard to a particular part of the model architecture, the following guidelines discuss choices for the model's output and prediction stage.

Tagging scheme For some tasks, which can assign labels to segments of texts, different tagging schemes are possible. These are: *BIO*, which marks the first token in a segment with a *B-* tag, all remaining tokens in the span with an _I-_tag, and tokens outside of segments with an *O-* tag; *IOB*, which is similar to BIO, but only uses *B-* if the previous token is of the same class but not part of the segment; and *IOBES*, which in addition distinguishes between single-token entities (*S-*) and the last token in a segment (*E-*). Using IOBES and BIO yield similar performance (Lample et al., 2017)

CRF output layer If there are any dependencies between outputs, such as in named entity recognition the final softmax layer can be replaced with a linear-chain conditional random field (CRF). This has been shown to yield consistent improvements for tasks that require the modelling of constraints (Huang et al., 2015; Max & Hovy, 2016; Lample et al., 2016) [58] [59] [60].

Constrained decoding Rather than using a CRF output layer, constrained decoding can be used as an alternative approach to reject erroneous sequences, i.e. such that do not produce valid BIO transitions (He et al., 2017). Constrained decoding has the advantage that arbitrary constraints can be enforced this way, e.g. task-specific or syntactic constraints.

Natural language generation

Most of the existing best practices can be applied to natural language generation (NLG). In fact, many of the tips presented so far stem from advances in language modelling, the most prototypical NLP task.

Modelling coverage Repetition is a big problem in many NLG tasks as current models do not have a good way of remembering what outputs they already produced. Modelling coverage explicitly in the model is a good way of addressing this issue. A checklist can be used if it is known in advances, which entities should be mentioned in the output, e.g. ingredients in recipes (Kiddon et al., 2016) [61]. If attention is used, we can keep track of a coverage vector $\mathbf{c}_i$, which is the sum of attention distributions $\mathbf{a}_t$ over previous time steps (Tu et al., 2016; See et al., 2017) [62] [63]:

$$\mathbf{c}_i = \sum_{t=0}^{i-1} \mathbf{a}_t$$

$$\mathbf{c}_i = \sum_{t=1} \mathbf{a}_t$$

This vector captures how much attention we have paid to all words in the source. We can now condition additive attention additionally on this coverage vector in order to encourage our model not to attend to the same words repeatedly:

$$f_{att}(\mathbf{h}_i, \mathbf{s}_j, \mathbf{c}_i) = \mathbf{v}_a{}^\top \tanh(\mathbf{W}_1 \mathbf{h}_i + \mathbf{W}_2 \mathbf{s}_j + \mathbf{W}_3 \mathbf{c}_i)$$

In addition, we can add an auxiliary loss that captures the task-specific attention behaviour that we would like to elicit: For NMT, we would like to have a roughly one-to-one alignment; we thus penalize the model if the final coverage vector is more or less than one at every index (Tu et al., 2016). For summarization, we only want to penalize the model if it repeatedly attends to the same location (See et al., 2017).

Neural machine translation

While neural machine translation (NMT) is an instance of NLG, NMT receives so much attention that many methods have been developed specifically for the task. Similarly, many best practices or hyperparameter choices apply exclusively to it.

Embedding dimensionality 2048-dimensional embeddings yield the best performance, but only do so by a small margin. Even 128-dimensional embeddings perform surprisingly well and converge almost twice as quickly (Britz et al., 2017).

Encoder and decoder depth The encoder does not need to be deeper than $2-4$ layers. Deeper models outperform shallower ones,

but more than 4 layers is not necessary for the decoder (Britz et al.,

2017).

Directionality Bidirectional encoders outperform unidirectional ones by a small margin. Sutskever et al., (2014) [64] proposed to reverse the source sequence to reduce the number of long-term dependencies. Reversing the source sequence in unidirectional encoders outperforms its non-reversed counter-part (Britz et al., 2017).

Beam search strategy Medium beam sizes around 10 with length normalization penalty of 1.0 (Wu et al., 2016) yield the best performance (Britz et al., 2017).

Sub-word translation Senrich et al. (2016) [65] proposed to split words into sub-words based on a byte-pair encoding (BPE). BPE iteratively merges frequent symbol pairs, which eventually results in frequent character n-grams being merged into a single symbol, thereby effectively eliminating out-of-vocabulary-words. While it was originally meant to handle rare words, a model with sub-word units outperforms full-word systems across the board, with 32,000 being an effective vocabulary size for sub-word units (Denkowski & Neubig, 2017).

Conclusion

I hope this post was helpful in kick-starting your learning of a new NLP task. Even if you have already been familiar with most of these, I hope that you still learnt something new or refreshed your knowledge of useful tips.

I am sure that I have forgotten many best practices that deserve to be on this list. Similarly, there are many tasks such as parsing,

information extraction, etc., which I do not know enough about to give recommendations. If you have a best practice that should be on this list, do let me know in the comments below. Please provide at least one reference and your handle for attribution. If this gets very collaborative, I might open a GitHub repository rather than collecting feedback here (I won't be able to accept PRs submitted directly to the generated HTML source of this article).

Credit for the cover image goes to Bahdanau et al. (2015).

---

1. Goldberg, Y. (2016). A Primer on Neural Network Models for Natural Language Processing. Journal of Artificial Intelligence Research, 57, 345–420. https://doi.org/10.1613/jair.4992 ↩
2. Kim, Y. (2014). Convolutional Neural Networks for Sentence Classification. Proceedings of the Conference on Empirical Methods in Natural Language Processing, 1746–1751. Retrieved from http://arxiv.org/abs/1408.5882 ↩
3. Melamud, O., McClosky, D., Patwardhan, S., & Bansal, M. (2016). The Role of Context Types and Dimensionality in Learning Word Embeddings. In Proceedings of NAACL-HLT 2016 (pp. 1030–1040). Retrieved from http://arxiv.org/abs/1601.00893 ↩
4. Plank, B., Søgaard, A., & Goldberg, Y. (2016). Multilingual Part-of-Speech Tagging with Bidirectional Long Short-Term Memory Models and Auxiliary Loss. In Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics. ↩
5. Ruder, S., Ghaffari, P., & Breslin, J. G. (2016). A Hierarchical Model of Reviews for Aspect-based Sentiment Analysis. Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing (EMNLP-16), 999–1005. Retrieved from http://arxiv.org/abs/1609.02745 ↩
6. He, L., Lee, K., Lewis, M., & Zettlemoyer, L. (2017). Deep Semantic Role Labeling: What Works and What's Next. ACL. ↩
7. Wu, Y., Schuster, M., Chen, Z., Le, Q. V, Norouzi, M., Macherey, W., … Dean, J. (2016). Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. arXiv Preprint arXiv:1609.08144. ↩
8. Reimers, N., & Gurevych, I. (2017). Optimal Hyperparameters for Deep LSTM-

Networks for Sequence Labeling Tasks. In arXiv preprint arXiv:1707.06799: Retrieved from https://arxiv.org/pdf/1707.06799.pdf ↩

9. Zhang, X., Zhao, J., & LeCun, Y. (2015). Character-level Convolutional Networks for Text Classification. Advances in Neural Information Processing Systems, 649–657. Retrieved from http://arxiv.org/abs/1509.01626 ↩
10. Conneau, A., Schwenk, H., Barrault, L., & Lecun, Y. (2016). Very Deep Convolutional Networks for Natural Language Processing. arXiv Preprint arXiv:1606.01781. Retrieved from http://arxiv.org/abs/1606.01781 ↩
11. Le, H. T., Cerisara, C., & Denis, A. (2017). Do Convolutional Networks need to be Deep for Text Classification ? In arXiv preprint arXiv:1707.04108. ↩
12. Srivastava, R. K., Greff, K., & Schmidhuber, J. (2015). Training Very Deep Networks. In Advances in Neural Information Processing Systems. ↩
13. Kim, Y., Jernite, Y., Sontag, D., & Rush, A. M. (2016). Character-Aware Neural Language Models. AAAI. Retrieved from http://arxiv.org/abs/1508.06615 ↩
14. Jozefowicz, R., Vinyals, O., Schuster, M., Shazeer, N., & Wu, Y. (2016). Exploring the Limits of Language Modeling. arXiv Preprint arXiv:1602.02410. Retrieved from http://arxiv.org/abs/1602.02410 ↩
15. Zilly, J. G., Srivastava, R. K., Koutnik, J., & Schmidhuber, J. (2017). Recurrent Highway Networks. In International Conference on Machine Learning (ICML 2017). ↩
16. Zhang, Y., Chen, G., Yu, D., Yao, K., Kudanpur, S., & Glass, J. (2016). Highway Long Short-Term Memory RNNS for Distant Speech Recognition. In 2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). ↩
17. He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep Residual Learning for Image Recognition. In CVPR. ↩
18. Huang, G., Weinberger, K. Q., & Maaten, L. Van Der. (2016). Densely Connected Convolutional Networks. CVPR 2017. ↩
19. Ruder, S., Bingel, J., Augenstein, I., & Søgaard, A. (2017). Sluice networks: Learning what to share between loosely related tasks. arXiv Preprint arXiv:1705.08142. Retrieved from http://arxiv.org/abs/1705.08142 ↩ ↩
20. Britz, D., Goldie, A., Luong, T., & Le, Q. (2017). Massive Exploration of Neural Machine Translation Architectures. In arXiv preprint arXiv:1703.03906. ↩
21. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A Simple Way to Prevent Neural Networks from Overfitting. Journal of Machine Learning Research, 15, 1929–1958.

http://www.cs.cmu.edu/~rsalakhu/papers/srivastava14a.pdf ↩

22. Ba, J., & Frey, B. (2013). Adaptive dropout for training deep neural networks. In Advances in Neural Information Processing Systems. Retrieved from file:///Files/A5/A51D0755-5CEF-4772-942D-C5B8157FBE5E.pdf ↩
23. Li, Z., Gong, B., & Yang, T. (2016). Improved Dropout for Shallow and Deep Learning. In Advances in Neural Information Processing Systems 29 (NIPS 2016). Retrieved from http://arxiv.org/abs/1602.02220 ↩
24. Gal, Y., & Ghahramani, Z. (2016). A Theoretically Grounded Application of Dropout in Recurrent Neural Networks. In Advances in Neural Information Processing Systems. Retrieved from http://arxiv.org/abs/1512.05287 ↩
25. Melis, G., Dyer, C., & Blunsom, P. (2017). On the State of the Art of Evaluation in Neural Language Models. ↩
26. Ruder, S. (2017). An Overview of Multi-Task Learning in Deep Neural Networks. In arXiv preprint arXiv:1706.05098. ↩
27. Rei, M. (2017). Semi-supervised Multitask Learning for Sequence Labeling. In Proceedings of ACL 2017. ↩
28. Ramachandran, P., Liu, P. J., & Le, Q. V. (2016). Unsupervised Pretrainig for Sequence to Sequence Learning. arXiv Preprint arXiv:1611.02683. ↩
29. Søgaard, A., & Goldberg, Y. (2016). Deep multi-task learning with low level tasks supervised at lower layers. Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, 231–235. ↩
30. Liu, P., Qiu, X., & Huang, X. (2017). Adversarial Multi-task Learning for Text Classification. In ACL 2017. Retrieved from http://arxiv.org/abs/1704.05742 ↩
31. Bahdanau, D., Cho, K., & Bengio, Y.. Neural Machine Translation by Jointly Learning to Align and Translate. ICLR 2015. https://doi.org/10.1146/annurev.neuro.26.041002.131047 ↩
32. Luong, M.-T., Pham, H., & Manning, C. D. (2015). Effective Approaches to Attention-based Neural Machine Translation. EMNLP 2015. Retrieved from http://arxiv.org/abs/1508.04025 ↩
33. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., … Polosukhin, I. (2017). Attention Is All You Need. arXiv Preprint arXiv:1706.03762. ↩
34. Kadlec, R., Schmid, M., Bajgar, O., & Kleindienst, J. (2016). Text Understanding with the Attention Sum Reader Network. In Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics. ↩
35. Lin, Z., Feng, M., Santos, C. N. dos, Yu, M., Xiang, B., Zhou, B., & Bengio, Y.

(2017). A Structured Self-Attentive Sentence Embedding. In ICLR 2017. ↩

36. Cheng, J., Dong, L., & Lapata, M. (2016). Long Short-Term Memory-Networks for Machine Reading. arXiv Preprint arXiv:1601.06733. Retrieved from http://arxiv.org/abs/1601.06733 ↩
37. Parikh, A. P., Täckström, O., Das, D., & Uszkoreit, J. (2016). A Decomposable Attention Model for Natural Language Inference. In Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing. Retrieved from http://arxiv.org/abs/1606.01933 ↩
38. Paulus, R., Xiong, C., & Socher, R. (2017). A Deep Reinforced Model for Abstractive Summarization. In arXiv preprint arXiv:1705.04304,. Retrieved from http://arxiv.org/abs/1705.04304 ↩
39. Daniluk, M., Rockt, T., Welbl, J., & Riedel, S. (2017). Frustratingly Short Attention Spans in Neural Language Modeling. In ICLR 2017. ↩
40. Liu, Y., & Lapata, M. (2017). Learning Structured Text Representations. In arXiv preprint arXiv:1705.09207. Retrieved from http://arxiv.org/abs/1705.09207 ↩
41. Dozat, T., & Manning, C. D. (2017). Deep Biaffine Attention for Neural Dependency Parsing. In ICLR 2017. Retrieved from http://arxiv.org/abs/1611.01734 ↩
42. Kingma, D. P., & Ba, J. L. (2015). Adam: a Method for Stochastic Optimization. International Conference on Learning Representations. ↩
43. Zhang, J., Mitliagkas, I., & Ré, C. (2017). YellowFin and the Art of Momentum Tuning. arXiv preprint arXiv:1706.03471. ↩
44. Ruder, S. (2016). An overview of gradient descent optimization. arXiv Preprint arXiv:1609.04747. ↩
45. Denkowski, M., & Neubig, G. (2017). Stronger Baselines for Trustable Results in Neural Machine Translation. ↩
46. Hinton, G., Vinyals, O., & Dean, J. (2015). Distilling the Knowledge in a Neural Network. arXiv Preprint arXiv:1503.02531. https://doi.org/10.1063/1.4931082 ↩
47. Kuncoro, A., Ballesteros, M., Kong, L., Dyer, C., & Smith, N. A. (2016). Distilling an Ensemble of Greedy Dependency Parsers into One MST Parser. Empirical Methods in Natural Language Processing. ↩
48. Kim, Y., & Rush, A. M. (2016). Sequence-Level Knowledge Distillation. Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing (EMNLP-16). ↩

49. Jean, S., Cho, K., Memisevic, R., & Bengio, Y. (2015). On Using Very Large Target Vocabulary for Neural Machine Translation. Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers), 1–10. Retrieved from http://www.aclweb.org/anthology/P15-1001 ↩

50. Sennrich, R., Haddow, B., & Birch, A. (2016). Edinburgh Neural Machine Translation Systems for WMT 16. In Proceedings of the First Conference on Machine Translation (WMT 2016). Retrieved from http://arxiv.org/abs/1606.02891 ↩

51. Huang, G., Li, Y., Pleiss, G., Liu, Z., Hopcroft, J. E., & Weinberger, K. Q. (2017). Snapshot Ensembles: Train 1, get M for free. In ICLR 2017. ↩

52. Snoek, J., Larochelle, H., & Adams, R. P. (2012). Practical Bayesian Optimization of Machine Learning Algorithms. Neural Information Processing Systems Conference (NIPS 2012). https://papers.nips.cc/paper/4522-practical-bayesian-optimization-of-machine-learning-algorithms.pdf ↩

53. Inan, H., Khosravi, K., & Socher, R. (2016). Tying Word Vectors and Word Classifiers: A Loss Framework for Language Modeling. arXiv Preprint arXiv:1611.01462. ↩

54. Press, O., & Wolf, L. (2017). Using the Output Embedding to Improve Language Models. Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers, 2, 157--163. ↩

55. Mikolov, T. (2012). Statistical language models based on neural networks (Doctoral dissertation, PhD thesis, Brno University of Technology). ↩

56. Pascanu, R., Mikolov, T., & Bengio, Y. (2013). On the difficulty of training recurrent neural networks. International Conference on Machine Learning, (2), 1310–1318. https://doi.org/10.1109/72.279181 ↩

57. Zhang, Y., & Wallace, B. (2015). A Sensitivity Analysis of (and Practitioners' Guide to) Convolutional Neural Networks for Sentence Classification. arXiv Preprint arXiv:1510.03820, (1). Retrieved from http://arxiv.org/abs/1510.03820 ↩

58. Huang, Z., Xu, W., & Yu, K. (2015). Bidirectional LSTM-CRF Models for Sequence Tagging. arXiv preprint arXiv:1508.01991. ↩

59. Ma, X., & Hovy, E. (2016). End-to-end Sequence Labeling via Bi-directional

LSTM-CNNs-CRF. arXiv Preprint arXiv:1603.01354. ↩

60. Lample, G., Ballesteros, M., Subramanian, S., Kawakami, K., & Dyer, C. (2016). Neural Architectures for Named Entity Recognition. NAACL-HLT 2016. ↩
61. Kiddon, C., Zettlemoyer, L., & Choi, Y. (2016). Globally Coherent Text Generation with Neural Checklist Models. Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing (EMNLP2016), 329–339. ↩
62. Tu, Z., Lu, Z., Liu, Y., Liu, X., & Li, H. (2016). Modeling Coverage for Neural Machine Translation. Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics. https://doi.org/10.1145/2856767.2856776 ↩
63. See, A., Liu, P. J., & Manning, C. D. (2017). Get To The Point: Summarization with Pointer-Generator Networks. In ACL 2017. ↩
64. Sutskever, I., Vinyals, O., & Le, Q. V. (2014). Sequence to sequence learning with neural networks. Advances in Neural Information Processing Systems, 9. Retrieved from http://arxiv.org/abs/1409.3215\nhttp://papers.nips.cc/paper/5346-sequence-to-sequence-learning-with-neural-networks ↩
65. Sennrich, R., Haddow, B., & Birch, A. (2016). Neural Machine Translation of Rare Words with Subword Units. In Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (ACL 2016). Retrieved from http://arxiv.org/abs/1508.07909 ↩

Sebastian Ruder
Read more posts by this author.

Read More

— Sebastian Ruder —
natural language processing

AAAI 2019 Highlights: Dialogue, reproducibility, and more

The 4 Biggest Open Problems in NLP

10 Exciting Ideas of 2018 in NLP

See all 22 posts →

19 Comments Blog 5 Nirmal Singhania

Recommend 6 Tweet Share Sort by Best

Join the discussion…

Jae Duk Seo • 9 months ago

another informative post, thank you

^ v • Reply • Share ›

Sebastian Ruder Mod ➚ Jae Duk Seo • 9 months ago

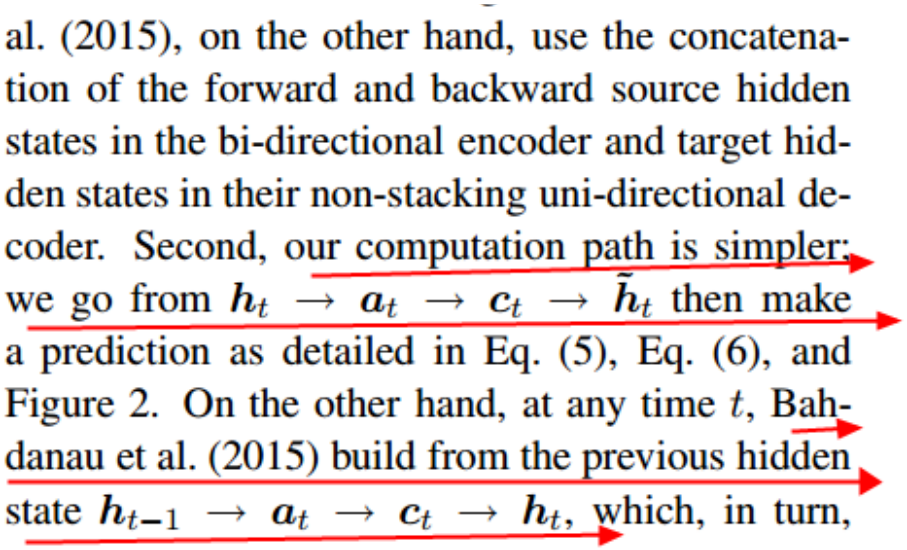
Thanks! :)

^ v • Reply • Share ›

Fisher • a year ago

Hi Sebastian, thx for the post! I've read it many times and benefit a lot. While this time, when comparing your formula of "Bahdanau's attention" with the author's original one, I found a slight difference. The original one uses h(i-1) to get attention weight, while your formula uses hi. This difference is also mentioned in Luong's paper.

Comparison to (Bahdanau et al., 2015) – While our global attention approach is similar in spirit to the model proposed by Bahdanau et al. (2015), there are several key differences which reflect how we have both simplified and generalized from the original model. First, we simply use hidden states at the top LSTM layers in both the encoder and decoder as illustrated in Figure 2. Bahdanau et


al. (2015), on the other hand, use the concatenation of the forward and backward source hidden states in the bi-directional encoder and target hidden states in their non-stacking uni-directional decoder. Second, our computation path is simpler; we go from $\boldsymbol{h}_t \rightarrow \boldsymbol{a}_t \rightarrow \boldsymbol{c}_t \rightarrow \tilde{\boldsymbol{h}}_t$ then make a prediction as detailed in Eq. (5), Eq. (6), and Figure 2. On the other hand, at any time t, Bahdanau et al. (2015) build from the previous hidden state $\boldsymbol{h}_{t-1} \rightarrow \boldsymbol{a}_t \rightarrow \boldsymbol{c}_t \rightarrow \boldsymbol{h}_t$, which, in turn,

^ v • Reply • Share ›

Sebastian Ruder Mod ➚ Fisher • 9 months ago

Hi Fisher,

That's a good point and a very good observation. In doing research for this post, I've encountered several different formulations of attention. I felt that the one used by Luong was slightly more common, so I've gone with that in this post.

^ v • Reply • Share ›

Zichao Wang • a year ago

Thanks for the post! I have a question on the use of word embeddings for natural language generation task with a seq2seq model during inference, i.e. generating tokens with a learnt model and new input sequences.

I know that during training, we feed to the encoder word embedding vectors corresponding to the words in the source, and feed to the decoder word embedding vectors corresponding to the words in the target (assume using teacher forcing).

Now, during testing, in the source sequence we might have words that do not appear in any source sequences in the training set, i.e. out of the training source vocabulary words. In this case, should I set those words to UNK during inference, or should I find the corresponding word embedding vector associated to these words, if they appear in the pre-trained word embeddings, and use them to feed my encoder?

I'm not sure if I articulated my question well enough; I'll clarify it if something I said above isn't clear to you. Thanks in advance!

^ v • Reply • Share ›

Sebastian Ruder Mod ➚ Zichao Wang • a year ago

Hi Zhichao, thanks for the question! I actually touched upon this here: http://ruder.io/word-embedd...

In brief, you would usually use UNK but for some tasks, e.g. reading comprehension setting it to a pre-trained vector might work better.

1 ^ v • Reply • Share ›

Zichao Wang ➚ Sebastian Ruder • a year ago

thanks! will try it out shortly.

^ | v • Reply • Share ›

Sebastian Ruder Mod ➚ Zichao Wang • a year ago

Great! Let me know how it works for you.

^ | v • Reply • Share ›

Stefan M • a year ago

I am about to start with my thesis (I am new to the NLP field) and all of your blog posts were like nitro booster and made me learn much faster! Thanks a lot for the posts Sebastian, keep up the good work! P.S Could you please let me know how do you find out about newest research and what are your resources?

^ | v • Reply • Share ›

Sebastian Ruder Mod ➚ Stefan M • a year ago

Hey Stefan, I'm glad the articles were helpful. Good luck with your thesis! :)

^ | v • Reply • Share ›

Tu Cao Trang Duong • a year ago

Thanks for the updating on this informative post. (Y)

^ | v • Reply • Share ›

Sebastian Ruder Mod ➚ Tu Cao Trang Duong • a year ago

Thank you! :)

^ | v • Reply • Share ›

Marzieh Saeidi • 2 years ago

Thanks for the very informative post which covered most practical aspects of using NNs for NLP. One thing that I would like to know more about is what people commonly do when using an RNN with a long sequence. Truncated backpropagation is suggested but so far, I have seen it mostly for language modeling. Do people use it for classification of long documents as well and how does that work?

^ | v • Reply • Share ›

Sebastian Ruder Mod ➚ Marzieh Saeidi • 2 years ago

Hey Marzieh, thanks! Great question! I agree that I've mostly seen truncated backpropagation through time (TBPTT) through time mainly for language models, where your input for many of the main datasets is one long sequence of text. TBPTT makes sense here in order to split up the text into smaller sequences. For classification or sequence labeling tasks, you normally don't do TBPTT as far as I'm aware. Instead, if you deal with sentences, you typically unroll the RNN over the entire sequence. With documents, this is a bit trickier; I don't think TBPTT would make sense here, either, as using TBPTT would require you to split up the document into many smaller parts. You might split it up into sentences instead and run a separate LSTM over sentences. That's assuming that you want to use an LSTM for document classification; often a

simpler model, e.g. BOW will do just as well.

Reply • Share ›

Marzieh Saeidi ➚ Sebastian Ruder • 2 years ago

Thank you for your response Sebastian.

Reply • Share ›

Yookoon Park • 2 years ago

Thanks for the nice post. One thing is that, to my knowledge, DenseNet concatenates previous layer outputs not sum them. Is this right?

Reply • Share ›

Sebastian Ruder Mod ➚ Yookoon Park • 2 years ago

Hey Yookoon, you're right! I had the dense residual connection used by Britz et al. (2017) [27] in mind when writing this. I've corrected this now.

Reply • Share ›