ML Theory Journal

Aritz Bercher

February 13, 2019

Abstract

In this journal, I will try to gather good references (mainly webpages) about some Machine Learning (and in particular Deep Learning and Natural Language Processing) algorithms as well as some personal thoughts on the subject. The aim of this journal is to keep gather in a same place the *theory* of Machine Learning

1 Useful Resources

- I found this blog by Tobias Sterbak, which seem to cover a lot of advanced topic in machine learning/deep learning, natural language processing, and computer vision with both theory and implementation aspects treated:
  `https://www.depends-on-the-definition.com/classify-toxic-comments-on-wikipedia/`
- There is this coursera mook about NLP:
  `https://www.coursera.org/learn/language-processing`

2 Journal

04.11.17 ~ Principal Component Analysis (PCA) and Singular Value Decomposition

PCA comes again and again, so I think I should try to learn it properly. I quickly reviewed what I had already seen in the first lecture of CIL. This being said I'm not use that it is exactly the PCA, since it's called SVD (Singular Value Decomposition). I read this very interesting page which gives a good explanation of the two different quantities and their relations:
Stack Exchange: PCA and SVD

21.02.18 ~ Skip-gram model for words embedding

I read these two pages of a same tutorial on the **Skip-gram** model:
`http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/`
`http://mccormickml.com/2017/01/11/word2vec-tutorial-part-2-negative-sampling/`
Edit (27.07.18): The skip-gram model is also presented in the coursera mook on NLP,

in a quite different way.
Edit (21.10.18): Basically, the skip-gram model does the following: it builds a neural network to predict for a given word, its context. The input is a one-hot encoded vector representing a word and the output is a vector of probability of size equal to the size of the vocabulary (let's call it n) indicating the probability of having any word as neighbor. The embeddings are obtained by taking the intermediary layer (which has no activation function and can be seen as matrix of size $n \times 300$ (if the embedding dimension is 300)).

14.03.18 ~ Recurrent Neural Networks (RNN), Long Short Term Memory (LSTM)
I read this introduction to **Recurrent Neural Networks in NLP**, and their applications to **Language Modeling and Generating Test**, **Machine Translation**, **Speech Recognition**, **Generating Image Descriptions** (when coupled with some CNN):
`http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-`
I also read this page on **Long Short Term Memory (LSTM)** networks:
`https://colah.github.io/posts/2015-08-Understanding-LSTMs/`
which is a very successful special case of RNN.

17.03.18 ~ Bag of words model
I read an introduction to the **Bag of word model (BOW)** here:
`https://machinelearningmastery.com/gentle-introduction-bag-words-model/`
I also read the beginning of the "Example: Logistic Regression Bag-of-Words classifier" here:
`http://pytorch.org/tutorials/beginner/nlp/deep_learning_tutorial.html`

21.03.18 ~ Character embedding vs Word embedding
I read this post on Quora which compares word embedding and **character embedding**:
Quora: How does character embedding work in comparison to word embedding?

22.03.18 ~ More on character embedding, GloVe
I read the beginning of this **article on character embedding**:
`http://minimaxir.com/2017/04/char-embeddings/`
but I didn't find it so good when it comes to explaining what character embedding is. But the little implementation with Keras seems like a good exercise. Maybe I could reproduce something like this with the dataset of fast ai.
I read this page about *GloVe*:
`https://nlp.stanford.edu/projects/glove/`
I think it is just a library doing this word embedding. It looks like the fundamental ideas are the same as for Word2Vec, i.e. train a model to predict co-occurrence of words (and then use the weights obtained as your embeddings).

The I read this page:
analyticsvidhya.com: Word representations-text classification using Fasttext (an NLP library from facebook)
The introduction explains quite well what **character embedding** is and what are its advantages compared to classical (i.e. Word2Vec already mentioned above) word embedding techniques. Then it dives into how to use this library. Maybe I should try to use it on the data set from fastai.

23.03.18 ∼ Continuous Bag of Words
I learned a bit about the **continuous bag of word** model is used which differs from the skip-gram model (presented in this page that I had already read) as explained in this page from Quora:
Quora: What are the continous bag of words and skip-gram architectures?

02.04.18 ∼ Constituency parser, Conditional Random Fields (CRS), Viterbi algorithm
I learned a bit about **constituency parser** here:
Stackoverflow: Difference between constituency parser and dependency parser
I read this introduction to **Conditional Random Fields (CRF)**:
`http://blog.echen.me/2012/01/03/introduction-to-conditional-random-fields/`
But this page doesn't mention any neural network.
I read the description of the **Viterbi algorithm** on wikipedia:
Wiki: Viterbi algorithm
From what I understood, it is just a kind of "dynamic programming" algorithm where instead of computing the probability of every possible sequence, we first find for each possible stae the most likely sequence of length 2 finishing by this state, then use it in order to find the most likely sequence of length 3 finishing by every possible state and so on.
Edit (30.04.18): In this paper it is explained (unfortunately not in a detail fashion) how to combine a (Bi)-LSTM network with a CRF:
`https://arxiv.org/pdf/1508.01991.pdf`
I also read this page which gives **an intuition of the Viterbi algorithm**:
Quora: What is an intuitive explanation of the Viterbi algorithm?
Edit (02.05.18): I think I understood how one can mix the LSTMs and the CRFs. It is kind of explained in this tutorial:
`https://pytorch.org/tutorials/beginner/nlp/advanced_tutorial.html`
Here are the main points: let say we want to use LSTMs without anything else to predict what should be the labels/tags of each word in a sentence. For each input (each word) the LSTM would output a distribution over the possible labels, and we would take its argmax as the predicted label (for a given word). Let say the words are $x_1, x_2, \dots$ and the labels can be taken from a set $S = \{s_1, \dots, s_N\}$. Now, we can see each entry of this vector of probability (of length N) as one of the weighted "feature function" of this

explanation of CRF. For an word/input x_i, entry 2 of the "hidden state" (I don't really like this terminology because with LSTM, we never know if we are talking of h_i or C_i (following the terminology introduced in this tutorial) but here I mean h_i) outputed by the LSTM is seen as a feature function giving the likelihood of seeing x_i if the tag/label is s_2. Then we add some "transition scores" which are additional feature functions independent from the LSTM and which allows to use the power of usual CRF models.
Edit (03.05.18): Actually, the idea goes beyond superposing the two processes. As I realized in the pytorch tutorial mentioned above, the transition scores/probabilities, are themselves some tensor which are optimized over, when doing backpropagation.
Edit (12.10.18): I guess that both the hidden states produced by the lstm and the transition functions/scores/probabilities can be seen as feature functions, but (and this is where my guess is) the hidden states make use of the information regarding the position in the sentence, whereas the transition functions look at the neighbouring words of the given word. But since the transition functions are also learned, I guess that these special "CRF" are merely an LSTM with an additional layer of weights on top of it.

30.04.18 $\sim$ Chat bots: Tai, Xiaoice, Named entity recognition, A nice blog about NLP
It seems that some fairly advanced chatbots already exist:
Wiki: Xiaoice
Wiki: Tay
I learned a bit about **Named Entity Recognition (NER)** here:
Wiki: Named Entity Recognition
I found this **nice blog (mainly) about NLP**:
`https://www.depends-on-the-definition.com/about/`

01.05.18 $\sim$ TF-IBF, and Latent-Dirichelet-Allocation
Roman mentioned **tf-idf**, **Latent-Dirichelet-Allocation**, HMM, as a classical alternative to NN models, which should be used as benchmarks to measure the success of the NN models.
From the wikipédia page for tf-idf:
Wiki: tf-idf
this tf-idf seems to be used in order to find among a given set of documents, which ones are relevant for a specific query ("the brown cow") for instance. It gives a score to each document.
Edit (25.05.18): Actually TF-IDF is quite well explained in the first week of the NLP coursera course. It's simply a way to improve a bag-of-word model, where instead of entering just 0 or 1 in each column to indicate if a word is present or not, one puts a score which indicates both how much frequent the word is in the given document/sample, how rare the word is in the total corpus of documents/samples.

The **Latent Dirichelet Allocation** is a bit more complicated. From the wikipedia page:
"In natural language processing, latent Dirichlet allocation (LDA) is a generative statistical model that allows sets of observations to be explained by unobserved groups that explain why some parts of the data are similar. For example, if observations are words collected into documents, it posits that each document is a mixture of a small number of topics and that each word's creation is attributable to one of the document's topics."
The math behind it is explained on the wikipedia page. From what I saw, it seems to be a bit similar to the bayesian nets studied in the course of Krauser in the sense that we make some (smart) assumptions on the distributions of the words in a corpus of documents, and then using algorithm to estimate the parameters of the distributions. But the wikipedia page is very long and technical. Maybe the coursera course on NLP would be better.

06.05.18 ~ Statistical Parsing
I read this wiki article about **Statistical Parsing**:
Wiki: Statistical Parsing

25.05.18 ~ Useful metrics for binary and multi-label classification, ROC curve
In the notebook of the exercise of the first week, I (re)discovered the notion of **F1-score**:
Wiki: F1 score which is a useful metric for binary or multi-label classification.
I also came back on the notion of **ROC curve**. I was confuse at first because, I had the impression that if we were doing binary classification (I guess we can generalize this discussion easily to multi-label classification) and we have just the predicted class of a model for a given set of data, there is only one True Positive Rate (TPR) and one False Positive Rate (FPR) to compute. But actually, we don't use the predicted classes but the *scores* output by the model, i.e. for each sample the probability to belong to each one of the two classes. Then we can consider for each $c \in [0, 1]$, the samples for which the probability to be in the second class is above c, to be "positive" and the other samples to be "negative", and then compare to the true labels. And this way we can compute this ROC curve.

29.05.18 ~ More about NLP
In the NLP coursera course, I was introduced to various models, some using Neural networks, some using classical algorithms.
In particular, I learned that the way to evaluate language models is based on the notion of **Perplexity** (c.f. journal NLP coursera for more infos).

30.05.18 ~ Dropouts for regularization of LSTMs
I read this article (I have it on my computer):

https://arxiv.org/pdf/1409.2329.pdf
explaining how one can use dropouts (i.e. randomly put some coefficients of the NN to 0 during the different training phases) to **avoid over-fitting**. In the case of RNN, one has to be careful not to affect the coefficients which manage the transmission of the memory.

31.05.18 ~ Evolutionary learning: an alternative to gradient descent
I read this article that Mike recommended to me:
http://togelius.blogspot.com/2018/05/empiricism-and-limits-of-gradient.html
It presents some alternative technique to gradient descent to come up with a model achieving good performance. The basic idea is to start with a bunch of different models, test them, throw away the one performing the worst, modify randomly the others and combine them, and then repeat the procedure. The advantages are that you don't need differentiability of the loss function, and according to the author it is more likely to learn "something which isn't in the data" like a mathematical formula. At the end of the article there is a list of recent paper on the topic by some AI labs like Uber AI, Sentient Technologies, DeepMind, and OpenAI.

01.06.18 ~ Gradient clipping
I learned about **Gradient Clipping** there:
hackernoon: Gradient clipping
It seems to come down to putting a bound on the value of the norm of the gradient to avoid NaN appearing. It seems to be useful for RNN:
Stack Overflow: Why do we clip by global norm to obtain gradients while performing RNN?

03.06.18 ~ Truncated Back Propagation Throw Time (TBPTT) for RNN
I read this article which explains some tricks used to change a bit the update scheme of the parameters when using RNN:
https://machinelearningmastery.com/gentle-introduction-backpropagation-time/
It is used in the RNN TF tutorial.

10.06.18 ~ permutation importance: a criterion to assess importance of features
I read this article about **permutation importance**:
http://parrt.cs.usfca.edu/doc/rf-importance/index.html
which explains that there is a better criterion that the "impurity decrease" called "permutation importance" to **judge how much a feature is useful**. This is particularly easy to use in the random forest case (because we have some OOB samples), but it can always be used if one has a validation set. I think it is implemented in scikit learn random forest estimator.

14.06.18 ~ Encoder-decoder

I read this Quora page about **encoder decoder Network**:
`https://www.quora.com/What-is-an-Encoder-Decoder-in-Deep-Learning`
because I wanted to understand something in the "lesson4-imdb" notebook of fastai (see entry of the 15.06.18 of the fastai journal, for details).
Edit (12.10.18): From what I understand an encoder-decoder is just a way to go from one initial representation which is the input of the encoder (for instance a sequence of bag of word vectors) to an intermediate one which is both the output of the encoder and the input of the decoder (for instance a vector) and then to a third one (which is for instance a sequence of words). Autoencoders are just NN which are typically stacks fully connected layers which have to predict their (cathegorical) input. By keeping only the first layers, we build a useful way to represent the data in a low dimensional space (and we build this way an encoder). One can have a CNN as encoder and RNN as decoder as explained in the link above.

15.06.18 ~ Simple but efficient models for sentiment analysis
I read this article:
`https://www.aclweb.org/anthology/P12-2018`
(I have it on my computer) which shows that some simple methods can be used efficiently for sentiment analysis on texts. It provides comparison tables for different common data set. I discovered it in the "nlp" dataset from fastai.

16.06.18 ~ Stop words
Stop words are words which are removed from a text before it is further processed. There are some lists of them (for instance in the "english" file of my NLTK package, or on this page) but they are not the same. Plus sometimes, it could actually harm the model to remove some common words (typically negations) which have a strong impact on the meaning of a sentence, if one wants to perform sentiment analysis. Some people suggest not to remove them at all:
Why sentiment words are in stopwords list?
Is there a stopword list specifically designed for sentiment analysis?

27.06.18 ~ Hyperparameter tuning for RNN models, measures against overfitting
There is an interesting section in this tutorial:
`https://www.oreilly.com/learning/perform-sentiment-analysis-with-lstms-using-tensorflow`
about how to **tune your hyperparameters** for a RNN.

In the same tutorial, the author explains how one should fight against overfitting: "The basic idea is that we train the model on our training set, while also measuring its

performance on the test set every now and again. Once the test error stops its steady decrease and begins to increase instead, you'll know to stop training, since this is a sign that the network has begun to overfit."

14.07.18 ~ Convolutional Neural Network for text classification
I read this paper about the use of **CNN for text classification**:
`https://arxiv.org/abs/1412.1058`
There are two different ways to do it. Either do the convolution directly on texts converted to sequence of one-hot-encoded vectors representing the words, or something a bit more subtle where each local region where a convolutional kernel would be applied is encoded as a bag of words.

16.04.18 ~ Difference Valdiation and Test set
I read this page which clarified what was the distinction between the **test set and the validation set**:
Stack Exchange: What is the difference between test and validation set

06.08.18 ~ Multi-label and multiclass classification
I learned a bit about multi-label and multiclass classification:
`https://en.wikipedia.org/wiki/Multiclass_classification#One-vs.-rest`
`https://en.wikipedia.org/wiki/Multi-label_classification`
which are not exactly the same thing. But the approaches are similar. Interestingly, the canonical methods for each one (One-vs-rest and binary relevance methods) are implemented in the same python class in Scikit-Learn: sklearn.multiclass.OneVsRestClassifier.

21.08.18 ~ Tuning a model
In order to tune a deep learning model, here are the recommendations from Roman:
"I start with a model config that I expect to overfit and slowly reduce nn complexity until validation performance stops improving. As for non nn parameters as learning rate, I pick them after I settle on network topology, I also pick them manually with some sort of coordinate gradient descent (picking them on by one) or using the defaults if I think the parameter is not important".

This link also gives some instructions to tune an XGBoost model:
`https://machinelearningmastery.com/xgboost-python-mini-course/`

12.10.2018 ~ StarSpace

This paper explains how **StarSpace** is working:
`https://arxiv.org/abs/1709.03856`
I had encountered it already when doing duplicate detection in the NLP coursera course.

It seems to be a way to build some custom embeddings. I found again a reference to it on this page explaining how the TensorFlow embedding pipeline of RasaNLU works.

21.10.18 ~ Parallel SGD computation with Hogwild and others

I read the beginning of this page concerning **asynchronous stochastic gradient descent algorithm** performed with **Hogwild**:
`https://srome.github.io/Async-SGD-in-Python-Implementing-Hogwild!/`
which offers also some sample code in python to implement it. At the end of the article, the author points out to other methods.

01.11.18 ~ Neural Turing Machines

I read this page about **Neural Turing Machines (NTM)**:
`https://blog.acolyer.org/2016/03/09/neural-turing-machines/`
and even if I didn't understand every detail, it was interesting. One has to see how far these Machines can learn. The article focuses on some very basic tasks like copying. This topic was also mentioned in the course about Synthesis and other topics of ETH. This page:
`https://medium.com/snips-ai/ntm-lasagne-a-library-for-neural-turing-machines-in-lasagne-`
gives much more insight.

13.02.19 ~ Levenshtein distance

I watched this video introducing the **Levenshtein distance** which gives a **distance** between two finite **sequences**:
`https://stackabuse.com/levenshtein-distance-and-text-similarity-in-python/`