Lessons Learned from Applying Deep Learning for NLP Without Big Data

yonatan hadar Follow

Oct 24, 2018 · 10 min read

As a data scientist, one of your most important skills should be choosing the right modeling techniques and algorithms for your problems. A few months ago I was trying to solve a text classification problem of classifying which news articles will be relevant for my customers.

I had only a few thousand labeled examples so I started with simple classic machine learning modeling methods like Logistic regression on TF-IDF. These models that usually works well on text classification of long documents like news articles performed only slightly better than random on this task.

After investigating the mistakes of my model I found that bag of words representation is just not enough for this task and I need a model the will use a deeper semantic understanding of the documents.

Deep learning models have presented very good performance on complex tasks that require deep understanding of text like translation, question answering, summarization, natural language inference etc. so it seemed like an excellent approach but deep learning is usually trained on hundreds of thousands or even millions of labeled data points and I had a much smaller dataset.

Usually, we need big datasets for deep learning to avoid over-fitting. Deep neural networks have many many parameters thus usually if they don't have enough data, they tend to memorize the training set and perform poorly on the test set. To avoid this phenomenon without big data we need to use special techniques.

In this post I will show some methods I found on articles, blogs, forums, Kaggle, and more resources or developed by myself in order to make deep learning work better on my task without big data. Many of these methods are based on best practices that are widely used in computer vision.

A small disclaimer: I am not a deep learning expert and this project was one of my first big projects with deep learning. Everything on this post is based

on my experience and possibly will behave differently on your problems.

. . .

Regularization

Regularization methods are methods that are used inside the machine learning model in different ways in order to avoid over-fitting, these methods have a strong theoretical background and work in a generic manner for most problems.

L1 and L2 Regularization

These methods are probably the oldest and are used for many years in many machine learning models. In this method, we add the weight size to the model's loss function that we are trying to minimize. This way the model will try to make the weights small and weights that don't help the model significantly will be reduced to zero and won't affect the model. This way we have a much smaller number of weights that the model can use to memorize the training set. for more explanation, you can read this post.

Dropout

Dropout is another newer regularization method that suggests that during training time, every node (neuron) in the neural network will be dropped (weights will be set to zero) in a probability of P. This way, the network can't rely on specific neurons or interaction of neurons and must learn every pattern in different parts of the network. This makes the model focus on important patterns that generalize to new data.

Early stopping

Early stopping is an easy regularization method, just monitor your validation set performance and if you see that the validation performance stops improving, stop the training. This method is very important without big data because the model tends to start over-fitting after 5–10 epochs or even earlier.

Small number of parameters

If you don’t have a large dataset you should be very careful with the number of layers and the number of neurons in each layer. Also, special layers like Convolutional layers have fewer parameters than fully connected layers so it’s very helpful to use them when they fit your problem.

. . .

Data augmentation

Data augmentation is a method to create more training data by changing training data in a way that the label doesn’t change. In computer vision, many image transformations are used to augment a dataset like flipping, cropping, scaling, rotating and more.

Image data augmentation example

These transformations are good for image data but will not work for text, for example flipping a sentence like “dogs love I” is not a valid sentence and using it will make the model learn junk. Here are a few text data augmentation methods:

Synonym replacement

In this method, we replace random words in our text with their synonyms, for example, we will change the sentence “ I like this movie very much” to “ I love this film very much” which still has the same meaning and probably the same label. This method didn’t work for me because synonyms have very similar word vectors, so the model sees both sentences as almost the same sentence and not as an augmentation.

Back translation

In this method, we take our text, translate it to an intermediate language with machine translation and then translate it back to English. This method was used successfully in the Kaggle toxic comments challenge. For example, if we translate “ I like this movie very much” to Russian, we get “Мне очень

нравится этот фильм”, when we translate back to English we get “I really like this movie”. The back translation method gives us both synonym replacement like the first method but also it can add or remove words and paraphrase the sentence while preserving the same meaning.

Document cropping

News articles are very long and while looking at the data I saw that I don’t need all the article in order to classify the document. Moreover, I saw that the articles main idea usually returns a few times. This made me think on cropping the article to several sub-documents as a data augmentation, this way I will have much more data. First, I tried to sample a few sentences from the documents and create 10 new documents. This created documents without a logical relation between sentences, so I got a bad classifier. My second attempt was splitting every article into pieces of 5 consecutive sentences. This method worked very nicely and gave me a nice boost in performance.

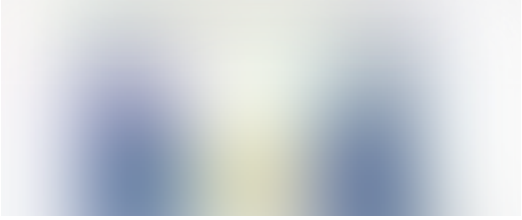
Generative adversarial network

GANs are one of the most exciting recent advances in data science, they are usually used as a generative model for image creation. This blog post explains how to use GANs for data augmentation for image data but it probably can be used for text also.

. . .

Transfer learning

Transfer learning is the use of weights from a network that was trained on another problem, usually with a big dataset, on your problem. Transfer learning is sometimes used as a weight initialization to some of the layers and sometimes as a feature extractor that we don't train anymore. In computer vision, starting with a pre-trained Imagenet model is a very common practice to solve problems but NLP doesn't have a very big dataset like Imagenet that can be used for transfer learning.



Pre-trained word vectors

NLP deep learning architectures usually start with an embedding layer that converts One hot encoded words to numerical vector representation. We can train the embedding layer from scratch but we can also use pre-trained word vectors like Word2Vec, FastText or Glove that were trained on tons of data using unsupervised learning methods or train on data from our domain. Pre-trained word vectors are very effective because they give the model context for words that are based on a lot of data and reduces the number of parameters of the model which reduces the chances for overfitting significantly. You can read more about word embedding here.

Pre-trained sentence vectors

We can change the input for the model from words to sentences, this way we can have smaller models with a smaller number of parameters that will still be expressive enough. In order to do this, we can use pre-trained sentence encoders like Facebook's InferSent or Google's Universal sentence encoder. We can also train a sentence encoder on unlabeled data from our domain using methods like skip-thought vectors or language models. You can learn more about unsupervised sentence vectors from my previous blog post.

Pre-trained language models

Recent papers like ULMFIT, Open-AI transformer, and BERT got amazing results for many NLP tasks by pre-training a language model on a very large corpus. Language model is a task of predicting the next word in a sentence

using the previous words. For me, this kind of pre-training didn't really help in getting better results, but the articles have shown a few methods to help with better fine tuning that I haven't tried. This is a great blog about pre-trained language models.

Pre-training with unsupervised or self-supervised learning

If we have a large dataset of unlabeled data from our domain we can use an unsupervised method like Autoencoders or masked language models to pre-train our models using only the text itself. Another option that worked better for me was using self-supervision. Self-supervised models are models that the label is extracted automatically without human annotation. A great example is the Deepmoji project. In Deepmoji, the authors trained a model to predict emojis from tweets, after getting good results in the emoji prediction they used their network to pre-train a tweeter sentiment analysis model that got state of the art results. The emoji prediction and sentiment analysis are obviously very related thus it performed very well as a pre-training task. Self-supervised tasks for news data can be predicting the headline, the newspaper, number of comments, number of retweets and

more. Self-supervised can be a very good way to pre-train, but it is usually hard to tell what proxy label will be connected to your real label.

Pre-training with other in company networks

In a lot of companies many machine learning models are built on the same dataset or similar datasets for different tasks. For example, for tweets, we can predict its topic, sentiment, number of retweets, and more. Pre-training your network with a network that is already in use can be the best thing that can be done and for my task, it gave a nice boost in performance.

. . .

Features engineering

I know that deep learning “killed” feature engineering and that it is a little old fashion to do it. But when you don’t have big data, helping the network to learn complex pattern with feature engineering can give a big boost in performance. For example, in my classification of news articles the author,

the newspaper, the number of comments, tags, and more features can help in predicting our label.

Multimodal architecture

We can use multimodal architecture to combine document level features into our model. In multimodal we build two different networks, one for the text and one for features, merge their output layers (without softmax) and add a few more layers. These models are pretty hard to train because the features usually have a stronger signal than the text so the network learns mostly the feature effect. This is a great Keras tutorial on multimodal networks. This approach improved my performance by less than one percent.

Word level features

Another type of feature engineering is word level features like Part of speech tagging, semantic role labeling, entity extraction and more. we can combine one hot encoded representation or an embedding of the word feature with the embedding of the word and use it as an input to the model. We can also use other word features in this method, for example in a sentiment analysis task we can take a sentiment dictionary and add another dimension to the embedding with 1 for words that we have in our dictionary and 0 for other words, this way the model can easily learn some

of the words that it needs to focus on. On my task, I added dimensions with certain entities that were important and this gave me a nice boost in performance.

Pre-processing as feature engineering

The last feature engineering method is pre-processing the input text in a way that it will be easier for the model to learn. One example is special “stemming”, if sports is not important for our label we can change football, baseball, and tennis to the word sport, this will help the network learn that the difference between sports is not important and can reduce the number of parameters in the network. Another example is using automatic summary. As I said earlier neural networks don’t perform very well on long texts so we can run an automatic summary algorithm like “text rank” on our text and give the network only the important sentences.

. . .

My model

In my case, after trying different combinations of the methods I discussed in this post, the best model was a Hierarchical Attention Network from this article with dropout and early stopping as regularization and document cropping as data augmentation. I used pre-trained word vectors and pre-training on another task that my company did for this customer on the same data. As feature engineering, I added entity word level feature to the word embedding. These changes from the basic model gave me a boost of almost 10% accuracy that made my model from slightly better than random to a model with valuable business impact.

Deep learning with small data is still in its early stages as a research field but it looks like it's gaining more popularity especially with pre-trained language models and I hope that researchers and practitioners will find more methods that will make deep learning valuable to every dataset.

Hope you enjoyed my post, and you're more than welcomed to read and follow my blog.

Machine Learning | Deep Learning | Artificial Intelligence | Towards Data Science | NLP

WRITTEN BY

yonatan hadar

Follow

Towards Data Science

A Medium publication sharing concepts, ideas, and codes.

Follow

See responses (12)

More From Medium

More from Towards Data Science

3 Insane Secret Weapons for Python

Preston Badeer in Towards Data Science
Apr 20 · 7 min read ★

3.2K

More from Towards Data Science

If I had to start learning data science again, how would I do it?

Santiago Víquez Segura in Towards Data Science
Apr 24 · 5 min read

1.99K

More from Towards Data Science

RIP correlation. Introducing the Predictive Power Score

Florian Wetschoreck in Towards Data Science
Apr 23 · 13 min read ★

2.2K