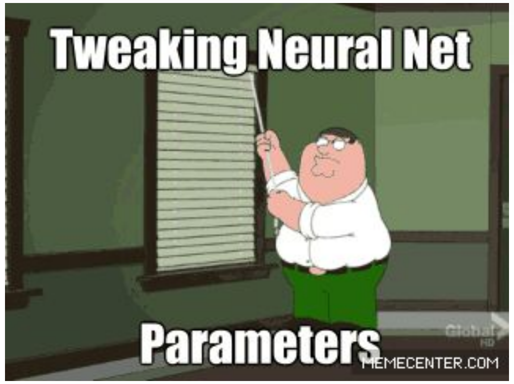
#ml-papers December 2019

Semi-supervised Sequence Learning

Section 1: Introduction



- Recurrent neural networks (RNN's) are powerful tools for modelling sequential data, but training them by back propagation through time is difficult
  - rarely used for nlp (text classification etc)
- Thesis: It is possible to use unsupervised learning with more unlabeled data to improve supervised learning.
- Dataset sources
  - Newsgroups , DBpedia
  - IMDB and Rotten Tomatoes
- Presented two approaches
  - Next step prediction model i.e recurrent language model in NLP as unsupervised method
  - Use of sequence encoder
    - Uses a RNN to read a long input sequence into a single vector. Later, use the same vector is recreate the original sequence
  - Weights obtained from these 2 pre-training steps can be used to initialize for standard LSTM RNN to improve training and generalization
- Conducted experiments such as data classification on Newsgroups , DBpedia and sentiment analysis on IMDB and Rotten Tomatoes
- Result :
  - <u>Important</u> : Using more unlabeled data from related tasks in the pre-training can improve the generalization of a subsequent supervised model.
  - Long short term memory (LSTMs) pretrained by recurrent language model or sequence encoders are usually better than LSTMs initialized randomly.
  - With sequence autoencoders, and outside unlabeled data LSTMs are able to match or surpass previously reported results

Section 2: Sequence autoencoders and recurrent language models

Inspiration:

seq2seq

enoder: use a recurrent network to read in an input sequence into a hidden state;
decoder: use this hidden state as input for decoder recurrent network

pre-training step:

option1: recurrent language model

option2: sequence autoencoder

unsupervised version of seq2seq (replace output seq with input seq-- recreate orig. seq)
the weights for decoder network and encoder network are the same
helpful for limited labeled data

whats' next

weights are used as initialization of supervised network
(1. network memorizes input sequence 2. gradients have shortcuts)

Section 3: Overview of methods

- LSTM recurrent network
  - compare basic LSTM with LSTM initialized with sequence autoencoder (language model LSTM or LM-LSTM vs sequence autoencoder or SA-LSTM)
- Predict document labels from previous time step in most experiments
- Also tried linear label gain, where document label is at each time step and increase weights linearly with each step

Section 4.0: Experiments - General Setup

- Follow LSTM recipes from this paper, section 3.4 ie: *"Although LSTMs tend to not suffer from the vanishing gradient problem, they can have exploding gradients. Thus we enforced a hard constraint on the norm of the gradient [10, 25] by scaling it when its norm exceeded a threshold."*
- tasks are text classification and sentiment analysis (a subset of text classification).. I think they should rename the paper to match this test setup it's misleading to imply these results would apply to all sequence learning
- autoencoders are trained without windowing, so they have tc reproduce the whole document as 1 sequence which is challenging for long documents
- cap their backprop @ 400 time steps to speed up training
- early stopping & dropout tuned on a validation set taken from the train set

"SA-LSTMs surpassed reported results for all datasets" -> not just their results, but research baselines for these problems!

Table 1: A summary of the error rates of SA-LSTMs and previous best reported resul

Dataset	SA-LSTM	Previous best result
IMDB	7.24%	7.42%
Rotten Tomatoes	16.7%	18.5%
20 Newsgroups	15.6%	17.1%
DBpedia	1.19%	1.74%

Section 4.1: Experiments - Sentiment Analysis on IMBD

- IMBD movie dataset: 25k labeled, 50k unlabeled docs in training, 25k in test. 15% of labeled train was used as validation; average doc had 241 words, max had 2.5k.. so very very long.

- able to reach 86.5% accuracy (5% lower than baselines) with their vanilla LSTM setup by fiddling with HPPs

- they found the approach to be very unstable, tweaking the LSTM hyperparameters just a bit lead to garbage models.. makes sense because docs are very very long; in contrast to this they said the SA-LSTM (sequence-autoencoder initialized) was very stable to deviations in hyperparameters, very interesting concept because a lot DL gains attributed to research are actually just better HPP tuning

- included a baseline where the LSTM was initialized with w2v embeddings which I was very happy about

Table 2: Performance of models on the IMDB sentiment classification task.

Model	Test error rate
LSTM with tuning and dropout	13.50%
LSTM initialized with word2vec embeddings	10.00%
LM-LSTM (see Section 2)	7.64%
SA-LSTM (see Figure 1)	7.24%
SA-LSTM with linear gain (see Section 3)	9.17%
SA-LSTM with joint training (see Section 3)	14.70%
Full+Unlabeled+BoW [21]	11.11%
WRRBM + BoW (bnc) [21]	10.77%
NBSVM-bi (Naïve Bayes SVM with bigrams) [35]	8.78%
seq2-bown-CNN (ConvNet with dynamic pooling) [11]	7.67%
Paragraph Vectors [18]	7.42%

Section 4.2: Experiments - Sentiment Analysis on Rotten Tomatoes

- 11k docs split 80/10/10, average length 22 words, max is 52 so its smaller than IMBD in documents && words/doc

- much easier problem: LSTMs train much faster and the gaps between different baselines are smaller, however due to the small training set size the scores are actually worse & models are very prone to overfitting

- they use unlabeled data from IMBD & 7.9M amazon movie reviews (why?) to train the autoencoder & also use word2vec embeddings trained on google news (were the previous ones trained on a larger corp?)

- good results but a bit less convincing than previous table; in particular would be interested to see how their SA-LSTM performed if it was only augmented with google news unsupervised data, or how the nonstatic CNN with w2v on IMBD&Amazon performed)

Table 4: Performance of models on the Rotten Tomatoes sentiment classification task.

Model	Test error rate
LSTM with tuning and dropout	20.3%
LSTM with linear gain	21.1%
LM-LSTM	21.7%
SA-LSTM	20.3%
LSTM with word vectors from word2vec Google News	20.5%
SA-LSTM with unlabeled data from IMDB	18.6%
SA-LSTM with unlabeled data from Amazon reviews	16.7%
MV-RNN [28]	21.0%
NBSVM-bi [35]	20.6%
CNN-rand [12]	23.5%
CNN-non-static (ConvNet with vectors from word2vec Google News) [12]	18.5%

they mention in their analysis that they're comparing methods that only use labeled data against methods that use labeled & non-labeled so it's unclear if the lift is due to approach or more data; they ran some experiments where they gave the SA-LSTM less labeled data to try to find a relationship between labeled <-> unlabeled data.. and were able to find this effect (although they don't mention what the ratios were)

Section 4.3-4.5: Experiments

Text classification experiments with 20 newsgroups

- Is it possible to use SA-LSTMs for tasks that have a substantial number of words?
- 70% input embedding dropout and 75% word dropout, SA-LSTM achieves 15.6% test set error

Character-level document classification experiments with DBpedia

- This linear gain method works well and achieves 1.32% test set error, which is better than SA-LSTM.
- Combining SA-LSTM and the linear gain method achieves 1.19% test set error

Object classification experiments with CIFAR-10

- We attempt to see if our pre-training methods extend to non-textual data
- Our 2-layer pretrained LM-LSTM is able to beat a baseline convolutional DBN model despite not using any convolutions and outperforms the non pre-trained LSTM.

Table 6: Performance of models on the 20 newsgroups classification task.

Model	Test error rate
LSTM	18.0%
LSTM with linear gain	71.6%
LM-LSTM	15.3%
SA-LSTM	15.6%
Hybrid Class RBM [17]	23.8%
RBM-MLP [5]	20.5%
SVM + Bag-of-words [2]	17.1%
Naïve Bayes [2]	19.0%

Table 7: Performance of models on the DBpedia character level classification task.

Model	Test error rate
LSTM	13.64%
LSTM with linear gain	1.32%
LM-LSTM	1.50%
SA-LSTM	2.34%
SA-LSTM with linear gain	1.23%
SA-LSTM with 3 layers and linear gain	1.19%
SA-LSTM (word-level)	1.40%
Bag-of-words	3.57%
Small ConvNet	1.98%
Large ConvNet	1.73%