
Knowledge extraction from medical literature using Recurrent Neural Networks

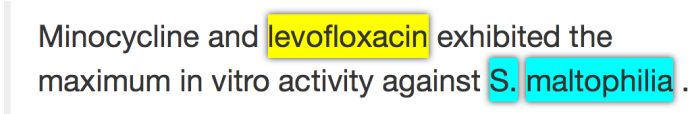
Abhimanyu Banerjee
Department of Physics
Stanford University
manyu@stanford.edu

Abstract

The problem of extracting knowledge relationships from unstructured text has proved a challenge for NLP. We focus on extracting relationship information between drugs targeting bacteria from medical literature. Deep learning techniques have proved most fruitful of late in learning relationships from NLP tasks. We use a recurrent neural network architecture (LSTM) and use this to train on labeled sentences to decide whether a given relationship exists

1 Introduction

Extracting knowledge and summarizing knowledge from reading unstructured text remains one of the large challenges in NLP. In this project I have focused on extracting medical relationships from bio-medical literature. A complete repository of relationships such as gene-gene, gene-drug, bacteria-drug will be extremely helpful for better understanding drug response[3]. The number of known and curated gene-gene relations is growing exponentially and is cataloged in databases such as BioGRID and ChEA. Medical literature itself is growing every year at a rapid rate and curating it by humans is too slow, so it would be really useful if we had a tool that could automatically curate these relationships for us. To be a little more concrete, I have focused on extracting relationships between drugs targeting bacteria. If we are given a sentence with a drug and a bacteria, we want to be able to say whether the drug has any action in targeting the bacteria. Deciding this is a challenge as context matters a lot. Below are two examples to illustrate this point.



Minocycline and levofloxacin exhibited the maximum in vitro activity against S. maltophilia.

Figure 1: Drug(Levifloxacin) targets bacteria(S.maltophilia) is a positive example

The first example is a clear sentence from which we can read and say that Levifloxacin definitely targets the bacteria S.maltophilia. The second example is vague and there is no clear evidence of Ipipenem acting on Escherichia Coli. We want to learn examples such as the first and pick out such relationships(ie. Levifloxacin acts on S.maltophilia) and ignore examples such as the second as it does not reveal anything insightful.

Deep learning approaches have been applied to several NLP tasks such as language modeling[4], sequence to sequence learning[5] with great successes. The natural architecture for learning on sequences is a recurrent neural network (RNN) or some variant of it. We use an LSTM architecture to learn on our data.

054 Our aims were : (i) to evaluate the activity of
055 temocillin in a urinary tract infection (UTI) model
056 due to ESBL-producing **Escherichia coli** and
057 compare it with that of **imipenem** ; and (ii) to
058 define in vivo susceptibility breakpoints .
059
060
061
062

063 Figure 2: Iipenem and Escherichia Coli. It is not clear what their relationship is . A negative
064 example

065 2 Dataset and pre-processing data

066 Since there is not dataset of labeled sentences with drugs targeting bacteria, I had to create my own
067 dataset for this purpose. I used the tool called [6] **ddlite** developed by Chris Re's group which is
068 useful in rapid prototyping and extracting relations. Ddlite was very useful in setting up the dataset.
069 I first downloaded a corpus of 14388 articles containing bacteria and drug mention keywords from
070 Pubmed central. After downloading this, I extracted all sentences that contain both a drug name
071 and a bacteria name from a dictionary match . I got a total of 7001 such sentences. We call such a
072 sentence as a "relation mention" . The "relation-mentions" now need to be labeled with a positive
073 or a negative label depending on whether they exhibit a target sort of relationship between the drug
074 and the bacteria or not.

075 To do this, we must write rules which we call "Labeling functions " in ddlite. Each Labeling
076 function is a rule that assigns the sentence a +1 label if the relation exhibited meets the condition
077 of a positive target relation, labels -1 if the sentence meets the condition of a negative relation and
078 0 if the labeling function is not conclusive and can not decide. These rules have to be designed well
079 to pick out good examples as we do not want to add noisy labels which will affect our final training.
080 We also want large "coverage" ie. our rules should be able to give a +1 or a -1 label to a large
081 fraction of our data set. We do not want too many 0 labels as these are useless for training.
082
083

084 I have written several rules(59 of them) that do labeling and got a coverage of about 60% , ie. I
085 am able to label 60% of these sentences with a positive or a negative label. Some examples of
086 labeling functions are: If we find the words "target","degrade","infect","conductive" in the sentence
087 between the drug and bacteria , mark these sentences with a +1. Some examples of negative labeling
088 functions: If the bacteria and drug are too far apart in the sentence, separated by > 20 tokens, mark
089 it with a -1 as it is unlikely they have any relation. Often also things like chemical elements and
090 nucleotides are mentioned as drugs, mark these as negative examples as well.

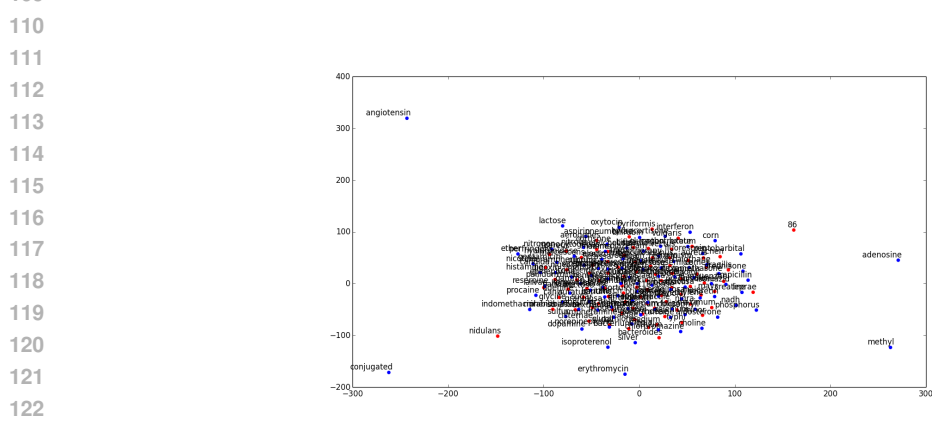
091 Finally it may happen that a sentence may have several different labels, because of different labeling
092 functions clashing. In this case we take a majority vote and assign a single label to the sentence.
093 After doing this I finally generated a computer labeled dataset of 2157 sentences. Of this 906 have
094 the label +1 and 1251 have the label -1 .

095 We hope that this dataset is good enough for training a deep learning model that would capture
096 language features and from that learn what a positive or a negative example would look like. I split
097 this dataset to 1600 for training, 300 for testing hyper-parameters and 257 for my development set.
098

099 2.1 pre training word-vectors

100 I created a vocabulary of word embedding trained specifically for this task on Medical literature.
101 I used a subset of the Medline corpus containing several thousand medical abstracts. The size of
102 this corpus in all was 1.5GB and I got it from the lab I work in (Russ Altman's lab). The word
103 vectors were trained using Tensorflows version of word2vec with skipgram and throwing out ultra
104 rare tokens which occur < 5 times in the corpus. The dimension of the word embeddings is 128.
105 Since the word embeddings generated contain both bacteria and drugs, I first did an initial experi-
106 ment to see if there is any clustering of concepts ie. do well formed concepts such as a bacteria and
107 a drug emerge from these word embeddings and can we visualize them? I plotted the 2D PCA of

108 the top 100 most frequently occurring drugs and bacteria for this purpose.
109
110



112
113
114
115
116
117
118
119
120
121
122
123
124 Figure 3: 2D PCA of top 100 most frequent bacteria and drugs. Bacteria are in red and drugs are in blue
125
126

127 The word embedding do not cluster well on meaning. As you can see from the figure, the clusters
128 overlap a lot . The data-set for training word vectors is probably not large enough to have formed
129 these well developed concepts.
130

131 3 Approach 132

133 We use the recurrent neural network architecture framework because this is what is quite natural
134 when you have sequential data. RNNs are very successful in learning on large sequences and mod-
135 eling the sequences. We use a popular version of the RNN called as the LSTM (long short term
136 memory)
137

138 3.1 Model-LSTM recurrent neural networks: 139

140 The LSTM (long short term memory) are a modified kind of recurrent neural networks. They were
141 introduced by Hochreiter and Schmidhuber (1997)[7] , and have been successfully used by many
142 people in following work . They work tremendously well on a large variety of problems, and are
143 now widely used. Vanilla RNN's can learn long term dependencies in principle but do not work well
144 in practice. This is because they suffer from the problem of vanishing and exploding gradients. The
145 LSTM solves this problem elegantly by defining a cell state that is a linear combination of a new
146 state and the previous state. This allows it to remember information across several time steps and in
147 practice has much better performance. The wonderful review article by Christopher Olah explains
148 the concepts behind LSTM quite well [2]. The basic structure of the LSTM is shown in the picture
below in figure 4:

149 The state of the LSTM is referred to by the symbol C_t This is updated at every step according to the
150 update rules. The final hidden state h_t is got from the cell state.
151

152 3.2 The LSTM update equations: 153

154 The LSTM has 3 gates , a forget gate , an input gate and an output gate. The forget gate controls
155 how much of the previous state we want to keep, the input gate regulates how important the current
156 input information is and the output gate regulates the output. It is best understood by the equations:

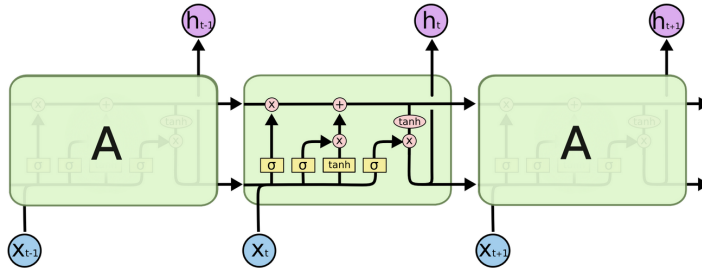
$$157 f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{1}$$

158 Here f_t is the forget gate, $W_f \in \mathbb{R}^{(n*n)}$ and $b_f \in \mathbb{R}^{(n)}$.

159 Similarly we also have an input gate and an update state:

$$160 i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \tag{2}$$

$$161 \tilde{C}_t = Tanh(W_c[h_{t-1}, x_t] + b_c) \tag{3}$$



The repeating module in an LSTM contains four interacting layers.

Figure 4: An unrolled LSTM recurrent neural network

Here as before $W_i \in \mathbb{R}^{(n*n)}$, $W_c \in \mathbb{R}^{(n*n)}$, $b_i \in \mathbb{R}^{(n)}$, $b_c \in \mathbb{R}^{(n)}$ The input and forget gates act to determine how much of the old state to forget and how much of the new state to use to develop the output state .

$$C_t = f_t \circ C_{t-1} + i_t \circ \tilde{C}_t \tag{4}$$

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \tag{5}$$

$$h_t = o_t \circ \text{Tanh}(C_t) \tag{6}$$

Finally the output gate acts on the final state to produce the current hidden state. The output gate controls what it decides is important to outputted to the hidden state. Of course $W_o \in \mathbb{R}^{(n*n)}$ and $b_o \in \mathbb{R}^{(n)}$

It is the final hidden state that we are interested in. The complicated dynamics of creating this state allows the LSTM to solve the vanishing, exploding gradient problems and learn well over long time steps.

We finally feed the final hidden state to a softmax layer (with two output states) and train the neural network with the Cross Entropy cost for the Softmax layer.

4 Experiment:

We train the LSTM using the cross entropy cost of the final hidden state. I modified a version of the LSTM code available to train MNIST [8]. Since tensorflow requires you to enter the number of steps in an RNN from before, you need to pad the sentences to a fixed length. What this means is that a special 'PAD' symbol must be introduced in the embedding which is a zero vector. All shorter sentences than the padded length must have the 'PAD' symbol at the end to make it of the fixed length. Larger sentences will get cut off. The average length of a sentence in my dataset was 38 tokens. The padded length is a hyper parameter that must be varied to get optimum performance.

4.1 Values of the hyper parameters used and tuned

Each word in the dictionary has a fixed length of 128 .
 The dimension of the hidden state is 200 .Performance did not change with changing it to 256
 Total epochs : 15 or 20
 Number of steps is varied between 10 to 150. The average sentence length is 38
 Learning rate : .001
 Batch size : 30

I used the Adam optimizer to optimize.

216 5 Results:

217

218

219

220

221

222

223

224

225

226

227

228

229

230

231

232

233

234

235

236

237

238

239

240

241

242

243

244

245

246

247

248

249

250

251

252

253

254

255

256

257

258

259

260

261

262

263

264

265

266

267

268

269

Since my training set is small (1800) sentences, the model overfits on the training data. The model is quite sensitive to the length of the padded sentence used (the number of steps) . I got the best performance for number of steps equal to 50 with a classification accuracy of 65% on the **dev. set**. I have plotted the performance of the classification algorithm on the dev. set as a function of the number of steps used in figure5.

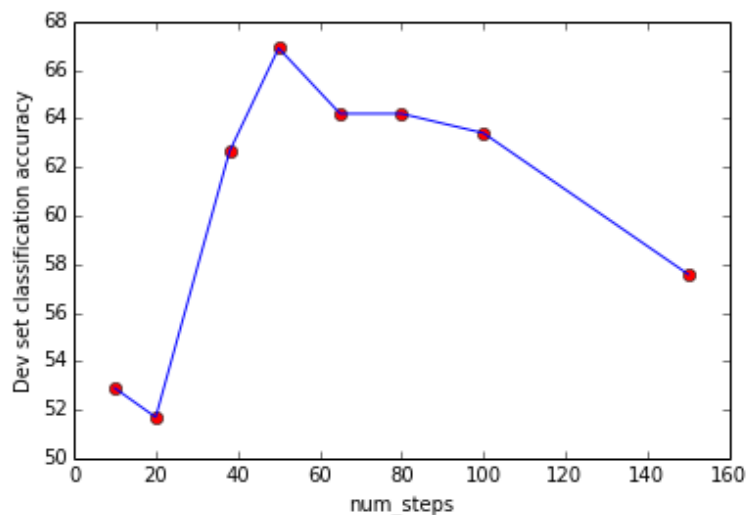


Figure 5: Classification accuracy on the dev set as a function of the number of steps used

It is interesting that we have a peak performance near the average sentence length of my data. Very small lengths are expected to be bad as we cut off too much information. Very long sentence lengths get confused on the shorter sentences as they have too many trailing zeros from the padding. Due to this they get stuck in local minima that they can't come out of and do not have good performance. I have also plotted the dev set accuracy as a function of number of training epochs. This turned out to be quite instructive and we can see how the sentences which are padded to larger lengths get stuck in local minima of the cost during training and the accuracy does not change much(unless it jumps abruptly out of the minima)

The shorter padded sentence length of 65 shows increasing accuracy (on dev set) with training epochs. The large padded length of 150 shows no improvement from one epoch to the next. It is stuck in local minimas. However it jumps out of one local minima only to be stuck in another.

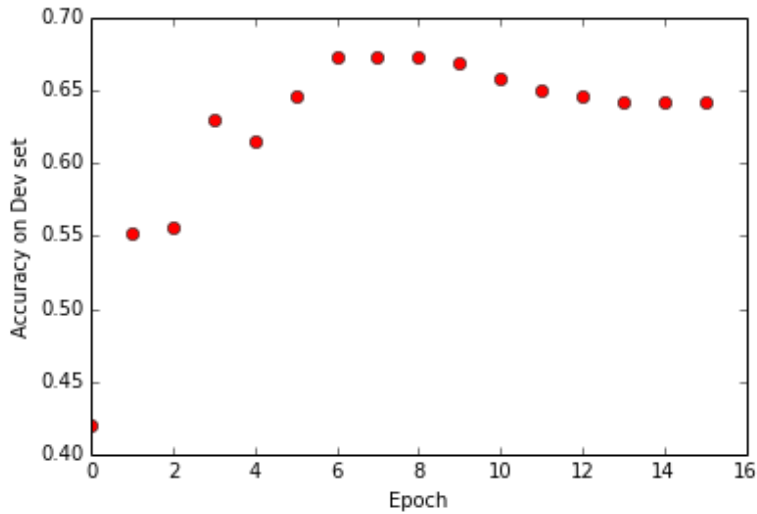
265 6 Conclusions

Our LSTM model is clearly able to learn as we have about 65% classification on the dev.set . However we are limited by our data which is computer generated so, we do not know whether it is learning actual relationships or just fitting the rules I have defined with my labeling functions. It will be amazing to have a good human labeled dataset for this purpose.

266 Acknowledgments

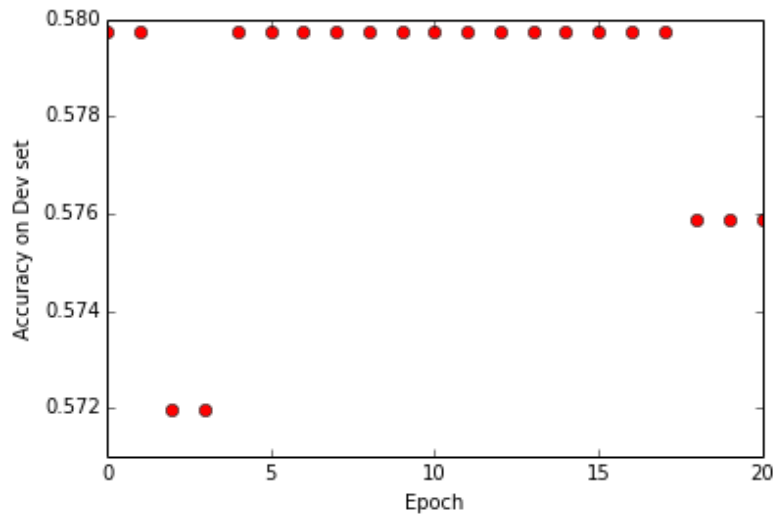
I really wish to thank my friend Raunaq for helping out with the project and homework. This course would not have been as much fun without his help. I also want to thank Emily Mallory for helping me learn ddLite and being a mentor. Thanks to Yuhao Zhang for giving me the medline data to train word vectors and being there to discuss different deep learning models.

270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288



289 Figure 6: Classification accuracy on the dev set as a function training epoch and sentence length=
290 65

291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311



312 Figure 7: Classification accuracy on the dev set as a function training epoch and sentence length=
313 150. Note how the accuracy randomly jumps from one local minima to another

314
315
316
317
318
319
320
321
322
323

References

[1] Wojciech Zaremba & Ilya Sutskever & Oriol Vinyals (2014) Recurrent Neural Network Regularization *arXiv:1409.2329*
[2] Christopher Olah -Understanding LSTM Networks <http://colah.github.io/posts/2015-08-Understanding-LSTMs/>
[3] Emily Mallory & Ce Zhang & Chris Re & Russ Altman(2015) Large-scale extraction of gene interactions from full-text literature using DeepDive *Bioinformatics first published online September 3, 2015* doi:10.1093/bioinformatics/btv476

324 [4]Tomas Mikolov & Martin Karafiat & Lukas Burget & Jan "Honza" Cernocky & Sanjeev Khudanpur - Re-
325 current neural network based language model *INTERSPEECH. Vol. 2. 2010*
326
327 [5]Sutskever Ilya & Oriol Vinyals and Quoc V. Le. "Sequence to sequence learning with neural net-
328 works." *Advances in neural information processing systems. 2014.*
329 [6]DDLITE by Chris Re et. al <https://github.com/HazyResearch/ddlite>
330 [7]Hochreiter, Sepp, & Jrgen Schmidhuber. "Long short-term memory." *Neural computation 9.8 (1997): 1735-*
331 *1780.*
332 [8]<https://github.com/aymericdamien/TensorFlow-Examples>
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377