Natural Language Processing with Deep Learning

Practical Tips for NLP Projects (NLP Data)
Finding Research Topics

Two basic starting points, for all of science:

- [Nails] Start with a (domain) problem of interest and try to find good/better ways to address it than are currently known/used
- [Hammers] Start with a technical approach of interest, and work out good ways to extend or improve it or new ways to apply it
Project types

This is not an exhaustive list, but most projects are one of

1. Find an application/task of interest and explore how to approach/solve it effectively, usually applying an existing neural network model
2. Implement a complex neural architecture and demonstrate its performance on some data
3. Come up with a new or variant neural network model and explore its empirical success
4. Analysis project. Analyze the behavior of a model: how it represents linguistic knowledge or what kinds of phenomena it can handle or errors that it makes
5. Rare theoretical project: Show some interesting, non-trivial properties of a model type, data, or a data representation

Gated LSTM
Thy youth ’s time and face his form shall cover?
Now all fresh beauty, my love there
Will ever Time to greet, forget each, like ever decease,
But in a best at worship his glory die.

Figure 1: Architecture of the Gated LSTM
We implemented and optimized Differentiable Neural Computers (DNCs) as described in the Oct. 2016 DNC paper [1] on the bAbI dataset [25] and on copy tasks that were described in the Neural Turning Machine paper [12]. This paper will give the reader a better understanding of this new and promising architecture through the documentation of the approach in our DNC implementation and our experience of the challenges of optimizing DNCs. Given how recently the
We present two improvements to the well-known Recurrent Neural Network Language Models (RNNLM). First, we use the word embedding matrix to project the RNN output onto the output space and already achieve a large reduction in the number of free parameters while still improving performance. Second, instead of merely minimizing the standard cross entropy loss between the prediction distribution and the "one-hot" target distribution, we minimize an additional loss term which takes into account the inherent metric similarity between the target word and other words. We show with experiments on the Penn Treebank Dataset that our proposed model (1) achieves significantly lower average word perplexity than previous models with the same network size and (2) achieves the new state of the art by using much fewer parameters than used in the previous best work.
Word2Bits - Quantized Word Vectors

Maximilian Lam
maxlam@stanford.edu

Abstract

Word vectors require significant amounts of memory and storage, posing issues to resource limited devices like mobile phones and GPUs. We show that high quality quantized word vectors using 1-2 bits per parameter can be learned by introducing a quantization function into Word2Vec. We furthermore show that training with the quantization function acts as a regularizer. We train word vectors on English Wikipedia (2017) and evaluate them on standard word similarity and analogy tasks and on question answering (SQuAD). Our quantized word vectors not only take 8-16x less space than full precision (32 bit) word vectors but also outperform them on word similarity tasks and question answering.
How to find an interesting place to start?

• Look at ACL anthology for NLP papers:
  • https://aclanthology.info

• Also look at the online proceedings of major ML conferences:
  • NeurIPS, ICML, ICLR

• Look at online preprint servers, especially:
  • https://arxiv.org

• Even better: look for an interesting problem in the world
How to find an interesting place to start?

Arxiv Sanity Preserver by Stanford grad Andrej Karpathy  http://www.arxiv-sanity.com

Top papers mentioned on Twitter over last day:

Shaping the Narrative Arc: An Information-Theoretic Approach to Collaborative Dialogue
Kory W. Mathewson, Pablo Samuel Castro, Colin Cherry, George Foster, Marc G. Bellemare
1/31/2019  cs.HC I cs.AI I cs.CL I cs.LG
20 pages, 9 figures

We consider the problem of designing an artificial agent capable of interacting with humans in collaborative dialogue to produce creative, engaging narratives. In this task, the goal is to establish universe details, and to collaborate on an interesting story in that universe, through a series of natural dialogue exchanges. Our model can augment any probabilistic conversational agent by allowing it to reason about universe information established and what potential next utterances might reveal. Ideally, with each utterance, agents would reveal just enough information to add specificity and reduce ambiguity without limiting the conversation. We empirically show that our model allows control over the rate at which the agent reveals information and that doing so significantly improves accuracy in predicting the next line of dialogues from movies. We close with a case-study with four professional theatre performers, who preferred interactions with our model-augmented agent over an unaugmented agent.

Learning and Evaluating General Linguistic Intelligence
Dani Yogatama, Cyprien de Masson d'Autume, Jerome Connor, Tomas Kocisky, Mike Chrzanowski, Lingpeng Kong, Angeliki Lazzarou, Wang Ling, Lei Yu, Chris Dyer, Phil Blunsom
1/31/2019  cs.LG I cs.CL I stat.ML
Want to beat the state of the art on something?

Great new site – a much needed resource for this – lots of NLP tasks

- Not always correct, though

https://paperswithcode.com/sota
Must-haves (for most* custom final projects)

• Suitable data
  • Usually aiming at: 10,000+ labeled examples by milestone
• Feasible task
• Automatic evaluation metric
• NLP is central to the project
Finding data

• Some people collect their own data for a project
  • You may have a project that uses “unsupervised” data
  • You can annotate a small amount of data
  • You can find a website that effectively provides annotations, such as likes, stars, ratings, etc.
    • Let’s you learn about real word challenges of applying ML/NLP!

• Some people have existing data from a research project or company
  • Fine to use providing one if you can provide data samples for submission, report, etc.

• Most people make use an existing, curated dataset built by previous researchers
  • You get a fast start and there is obvious prior work and baselines
Linguistic Data Consortium

- https://catalog.ldc.upenn.edu/

- Treebanks, named entities, coreference data, lots of newswire, lots of speech with transcription, parallel MT data
  - Look at their catalog
Machine translation

- http://statmt.org
- Look in particular at the various WMT shared tasks

Statistical Machine Translation

This website is dedicated to research in statistical machine translation, i.e. the translation of text from one human language to another by a computer that learned how to translate from vast amounts of translated text.

Introduction to Statistical MT Research

- The Mathematics of Statistical Machine Translation by Brown, Della Petra, Della Pietra, and Mercer
- Statistical MT Handbook by Kevin Knight
- SMT Tutorial (2003) by Kevin Knight and Philipp Koehn
- ESSLLI Summer Course on SMT (2005), day 1, 2, 3, 4, 5 by Chris Callison-Burch and Philipp Koehn.
- MT Archive by John Hutchins, electronic repository and bibliography of articles, books and papers on topics in machine translation and computer-based translation tools
Dependency parsing: Universal Dependencies

- https://universaldependencies.org

Universal Dependencies

Universal Dependencies (UD) is a framework for cross-linguistically consistent grammatical annotation and an open community effort with over 200 contributors producing more than 100 treebanks in over 70 languages.

- **Short introduction to UD**
- **UD annotation guidelines**
- **More information on UD:**
  - **How to contribute to UD**
  - **Tools for working with UD**
  - **Discussion on UD**
  - **UD-related events**
- **Query UD treebanks online:**
  - **SETS treebank search** maintained by the University of Turku
  - **PML Tree Query** maintained by the Charles University in Prague
  - **Kontext** maintained by the Charles University in Prague
  - **Grew-match** maintained by Inria in Nancy
- **Download UD treebanks**

If you want to receive news about Universal Dependencies, you can subscribe to the **UD mailing list**. If you want to discuss individual annotation questions, use the **Github issue tracker**.
Many, many more

• There are now many other datasets available online for all sorts of purposes
  • Look at Kaggle
  • Look at research papers
  • Look at lists of datasets
    • https://machinelearningmastery.com/datasets-natural-language-processing/
    • https://github.com/niderhoff/nlp-datasets
Doing your research example:
Straightforward Class Project: Apply NNets to Task

1. Define Task:
   • Example: **Summarization**

2. Define Dataset
   1. Search for academic datasets
      • They already have baselines
      • E.g.: Newsroom Summarization Dataset: [https://summari.es](https://summari.es)
   2. Define your own data (harder, need new baselines)
      • Allows connection to your research
      • A fresh problem provides fresh opportunities!
      • Be creative: Twitter, Blogs, News, etc. There are lots of neat websites which provide creative opportunities for new tasks
3. **Dataset hygiene**
   - Right at the beginning, separate off devtest and test splits
   - Discussed more next

4. **Define your metric(s)**
   - Search online for well established metrics on this task
   - Summarization: Rouge (Recall-Oriented Understudy for Gisting Evaluation) which defines \( n \)-gram overlap to human summaries
   - Human evaluation is still much better for summarization; you may be able to do a small scale human eval
Straightforward Class Project: Apply NNets to Task

5. Establish a baseline
   - Implement the simplest model first (often logistic regression on unigrams and bigrams or averaging word vectors)
   - For summarization: See LEAD-3 baseline
   - Compute metrics on train AND dev
   - Analyze errors
   - If metrics are amazing and no errors:
     - Done! Problem was too easy. Need to restart. 😊/😢

6. Implement existing neural net model
   - Compute metric on train and dev
   - Analyze output and errors
   - Minimum bar for this class
7. Always be close to your data! (Except for the final test set!)
   - Visualize the dataset
   - Collect summary statistics
   - Look at errors
   - Analyze how different hyperparameters affect performance

8. Try out different models and model variants
   Aim to iterate quickly via having a good experimental setup
   - Fixed window neural model
   - Recurrent neural network
   - Recursive neural network
   - Convolutional neural network
   - Attention-based model
   - ...

Straightforward Class Project: Apply NNets to Task
Many publicly available datasets are released with a train/dev/test structure. **We're all on the honor system to do test-set runs only when development is complete.**

Splits like this presuppose a fairly large dataset.

If there is no dev set or you want a separate tune set, then you create one by splitting the training data, though you have to weigh its size/usefulness against the reduction in train-set size.

Having a fixed test set ensures that all systems are assessed against the same gold data. This is generally good, but it is problematic where the test set turns out to have unusual properties that distort progress on the task.
Training models and pots of data

- When training, models **overfit** to what you are training on
  - The model correctly describes what happened to occur in particular data you trained on, but the patterns are not general enough patterns to be likely to apply to new data
- The way to monitor and avoid problematic overfitting is using **independent** validation and test sets ...
Training models and pots of data

• You build (estimate/train) a model on a **training set**.

• Often, you then set further hyperparameters on another, independent set of data, the **tuning set**
  • The tuning set is the training set for the hyperparameters!

• You measure progress as you go on a **dev set** (development test set or validation set)
  • If you do that a lot you overfit to the dev set so it can be good to have a second dev set, the **dev2 set**

• **Only at the end**, you evaluate and present final numbers on a **test set**
  • Use the final test set **extremely** few times ... ideally only once
Training models and pots of data

- The **train**, **tune**, **dev**, and **test** sets need to be completely distinct.
- It is invalid to test on material you have trained on.
  - You will get a falsely good performance. We usually overfit on train.
- You need an independent tuning set.
  - The hyperparameters won’t be set right if tune is same as train.
- If you keep running on the same evaluation set, you begin to overfit to that evaluation set.
  - Effectively you are “training” on the evaluation set ... you are learning things that do and don’t work on that particular eval set and using the info.
- To get a valid measure of system performance you need another untrained on, **independent** test set ... hence dev2 and final test.
Getting your neural network to train

• Start with a positive attitude!
  • Neural networks want to learn!
    • If the network isn’t learning, you’re doing something to prevent it from learning successfully

• Realize the grim reality:
  • There are lots of things that can cause neural nets to not learn at all or to not learn very well
    • Finding and fixing them (“debugging and tuning”) can often take more time than implementing your model

• It’s hard to work out what these things are
  • But experience, experimental care, and rules of thumb help!
Models are sensitive to learning rates

- From Andrej Karpathy
Models are sensitive to initialization

- From Michael Nielsen
Training a (gated) RNN

1. Use an LSTM or GRU: *it makes your life so much simpler!*
2. Initialize recurrent matrices to be orthogonal
3. Initialize other matrices with a sensible *(small!)* scale
4. Initialize forget gate bias to 1: *default to remembering*
5. Use adaptive learning rate algorithms: *Adam, AdaDelta, …*
6. Clip the norm of the gradient: *1–5 seems to be a reasonable threshold when used together with Adam or AdaDelta.*
7. Either only dropout vertically or look into using Bayesian Dropout (Gal and Gahramani – not natively in PyTorch)
8. *Be patient! Optimization takes time*

[Saxe et al., ICLR2014; Ba, Kingma, ICLR2015; Zeiler, arXiv2012; Pascaru et al., ICML2013]
Experimental strategy

• Work incrementally!
• Start with a very simple model and get it to work
• Add bells and whistles one-by-one and get the model working with each of them (or abandon them)

• Initially run on a tiny amount of data
  • You will see bugs much more easily on a tiny dataset
  • Something like 8 examples is good
  • Often synthetic data is useful for this
  • Make sure you can get 100% on this data
    • Otherwise your model is definitely either not powerful enough or it is broken
Experimental strategy

• Run your model on a large dataset
  • It should still score close to 100% on the training data after optimization
    • Otherwise, you probably want to consider a more powerful model
    • Overfitting to training data is **not** something to be scared of when doing deep learning
      • These models are usually good at generalizing because of the way distributed representations share statistical strength regardless of overfitting to training data
  • But, still, you now want good generalization performance:
    • Regularize your model until it doesn’t overfit on dev data
      • Strategies like L2 regularization can be useful
      • But normally generous dropout is the secret to success
Details matter!

• Look at your data, collect summary statistics
• Look at your model’s outputs, do error analysis
• Tuning hyperparameters is really important to almost all of the successes of NNets