Transfer Learning in NLP (ULMFiT)

Adrian Brasoveanu, 11/14/2019

[based on slides by Sebastian Ruder]
What is transfer learning?

(a) Traditional Machine Learning

(b) Transfer Learning

Pan and Yang (2010)
Why transfer learning in NLP?

- Many NLP tasks share common knowledge about language, e.g., cooccurrence/distributional similarities, meaning representations
- Tasks can inform each other, e.g., building a language model for English can help with subsequent text classification
- Annotated data is (relatively) rare, unlabeled data is (usually) not; leverage unlabeled data to extract as much information as possible from annotated data
- Empirically, transfer learning has resulted in SOTA for many supervised NLP tasks (classification, Q&A etc.) – see GLUE and Super-GLUE tasks
  - [https://super.gluebenchmark.com/leaderboard](https://super.gluebenchmark.com/leaderboard)
Types of transfer learning in NLP

- **Transfer learning**
  - **Inductive transfer learning**
    - Different tasks; labeled data in target domain
    - Tasks learned simultaneously
    - Tasks learned sequentially
  - **Transductive transfer learning**
    - Same task; labeled data only in source domain
    - Different languages
  - **Multi-task learning**
    - Different domains
  - **Domain adaptation**
    - Cross-lingual learning

We will focus on this

Ruder (2019)
Sequential transfer learning

Learn on one task / dataset, then transfer to another task / dataset

Pretraining

word2vec
GloVe
skip-thought
InferSent
ELMo
ULMFiT
GPT
BERT

Adaptation

classification
sequence labeling
Q&A
....
Example: word vectors

Word embedding methods (e.g. word2vec) learn one vector per word:

\[
\text{cat} = [0.1, -0.2, 0.4, ...] \\
\text{dog} = [0.2, -0.1, 0.7, ...]
\]
Example: word vectors

Word embedding methods (e.g. word2vec) learn one vector per word:

\[ \text{cat} = [0.1, -0.2, 0.4, ...] \]
\[ \text{dog} = [0.2, -0.1, 0.7, ...] \]
Example: word vectors

Word embedding methods (e.g. word2vec) learn one vector per word:

cat = [0.1, -0.2, 0.4, ...]
dog = [0.2, -0.1, 0.7, ...]

I love my cat and dog . }-> “positive"
Why embed words?

- Embeddings are themselves parameters, can be learned
- Much lower dimensional space than sparse one-hot encodings: easier to compute with
- We can generalize across words via their continuous dense word embeddings, as opposed to categorical one-hot encodings; mitigates data sparsity
- We can share word representations across tasks
Latent Semantic Analysis (LSA)—SVD of term-document matrix, (Deerwester et al., 1990)

Brown clusters, hard hierarchical clustering based on n-gram LMs, (Brown et al. 1992)

Latent Dirichlet Allocation (LDA)—Documents are mixtures of topics and topics are mixtures of words (Blei et al., 2003)
Many successful pretraining approaches are based on language modeling.

Informally, a LM learns $P_\theta(\text{text})$ or $P_\theta(\text{text} \mid \text{some other text})$.

LMs don’t require human annotation.

Many languages have enough text to learn high capacity model.

Versatile—can learn both sentence and word representations with a variety of objective functions.
Word vector pretraining

n-gram neural language model
(Bengio et al. 2003)

Supervised multitask word embeddings
(Collobert and Weston, 2008)
word2vec

Efficient algorithm + large scale training $\rightarrow$ high quality word vectors

(Mikolov et al., 2013)

See also:
Pennington et al. (2014): GloVe
Bojanowski et al. (2017): fastText
Pretrain AWD-LSTM LM, fine-tune LM in two stages with different adaptation techniques

SOTA for six classification datasets

(Howard and Ruder, ACL 2018)
Pretraining reduces need for annotated data

(Howard and Ruder, ACL 2018)
Why does language modeling work so well?

- Language modeling is a very difficult task, even for humans.
- Language models are expected to compress any possible context into a vector that generalizes over possible completions.
  - “They walked down the street to ???”
- To have any chance at solving this task, a model is forced to learn syntax, semantics, encode facts about the world, etc.
- Given enough data, a huge model, and enough compute, can do a reasonable job!
- Empirically works better than translation, autoencoding: “Language Modeling Teaches You More Syntax than Translation Does” (Zhang et al. 2018)
Go through one example of sequential transfer learning for a classification task in full detail.

- Data: the IMDB movie reviews (Maas et al. 2011)
- Transfer learning following the ULMFiT model (Howard and Ruder 2018)
- We use the AWD-LSTM model (Merity et al 2017), a vanilla LSTM with four kinds of dropout regularization
- Small, simple model: word embedding size of 300, 2 LSTM layers with 300 hidden size per layer, BPTT (back-propagation through time) of size 70
- We pretrain a language model on Wikitext-103 (Merity et al. 2017b): 28595 preprocessed Wikipedia articles, a total of 103 million words

Pretrained model is small and simple (no attention, skip connections etc.) and the pretraining corpus is of modest size.
We finetune the pretrained AWD-LSTM model using discriminative (Yosinski et al. 2014) and slanted triangular (Smith 2017; Howard and Ruder 2018) learning rates.

Discriminative learning rates help us avoid catastrophic forgetting.

Slanted triangular learning rates improve (rate of) convergence in training.

We do the same kind of minimal preprocessing as in Howard and Ruder (2018).

Notebooks based on the Fastai course, v3, part 2: https://course.fast.ai/part2

- We start with an overview of the preprocessing steps
- We introduce the AWD-LSTM model
- We show how to pretrain the AWD-LSTM model on Wikipedia
- We do ULMFiT-style transfer learning for IMDB sentiment classification