Deep Learning with Python Ch. 11 Natural Language

Deep Learning for Text

- We will just look at a few big ideas in this area.
- bags versus sequences
- various questions to ask
  - text classification, content filtering, sentiment analysis, language modeling, translation, summarization

Human Language

- CS people distinguish betweeen what we call natural languages (what everyone else calls languages or perhaps human languages) and programming languages.
- Programming languages are designed from scratch to be able to unambibuously describe computation. There are various goals in this process, but the big ones are being relatively easy for humans to use, clear, understandable syntax, and (without fail) a lack of ambiguity.
- Human languages, on the other hand, are constantly evolving, were not designed directly by any human or group of humans (with the possible exception of Esperanto and Klingon) and are meant for much more open-ended communication.

Natural Language Processing

- NLP has been part of AI from the beginning. AI has helped us to understand how difficult some aspects of language understanding is and how inherently ambiguous human languages generally are.
- Most early (meaning at least through the 80s) AI language-related systems were hand-crafted and usually involved both CS people and applied linguists. (For example DecTalk, which we discussed earlier in this class.)
- But in the last 30 years there has been a push to use data to learn language-related rules. A famous quote from an early speech recognition researcher: "Every time I fire a linguist, the performance of the speech recognizer goes up."

Problem Areas

- What is the topic of this text? (text classification)
- Is this text appropriate? (content filtering)
- Is this text positive or negative? (sentiment analysis)
- What should/could the next word be in this sequence (language modeling)
- How would you say this in Swahili? (translation)
- How would you summarize this article or pick out the important facts? (information extraction or summarization)

Natural Language Understanding

- In the last slide, note the complete lack of:
  - What does the sentence mean?
  - Was the speaker being serious or are they joking?
  - Is that duck as in run for cover or duck as in quack?
  - How could I construct a sentence to respond to the previous sentence in an appropriate way?

NLP toolsets

- Early data-driven NLP work (meaning the 90s and into the 2000s) used decision trees and logistic regression. We will briefly look at decision trees later this semester.
- Then **recurrent neural networks** became popular. A recurrent network isn't just an acyclic graph like we have assumed so far. It has inputs to layers coming from layers further along in problem solving or even from the layer itself. This supports a form of memory and allows sequences in the input to be noticed.
- In general RNNs have an issue with something called the vanishing gradiant problem, where, older recurrent inputs gradually vanish in the weights. This is addressed by the Long Short-Term Memory (LSTM) algorithm, which has a way to carry information across many time steps. In particular bidirectional LSTM models have been quite successful.
- Even more recently the Transformer architecture has become the new standard.

Preparing text data

- Figure 11.1 shows clearly the steps involved in transforming an input text string into something more standardized. We remove capitalization and punctuation (usually), pick out the individual words, turn them into integer keys for a dictionary of all the words (or the most frequent words) in the string, then use one-hot encoding the represent them.
- One possibility is to encode n-grams, rather than just individual words. Page 314 shows how this works.
- Not surprisingly, Keras has tools for automating this process.

Two approaches for groups of words

- Representing individual words is fairly non-controversial.
- But how do we encode the way words are woven into sentences?
- But notice that word orders vary significantly across languages and usually we can transform a sentence into an essentially equivalent sentence while changing word order.
- The simplest solution is to ignore word order and use a **bag-of-words** representation. For many problems, this is exactly what you want.
- Note that the bag-of-words approach can easily be extended to bigrams or trigrams (or beyond). Just put each bigram in your dictionary. The text shows this process improving the performance of the IMDB problem from earlier.

Sequences of words

- More recently, with the growth of RNNs and Transformers, it has been possible to work with sequences of words. Just plain using sequences doesn't help much with the IMDB model.
- But **word embeddings** are a really interesting idea. They are vector representations of words that map human language into a structured geometric space where words with similar semantic meanings are embedded into similar word vectors. Word2Vec from Google is a famous example of this.
- Instead of a binary, sparse, high-dimensional vector (as from one-hot encoding) we get a low-dimensional, floating point vector. See Figure 11.2 for an example. Figure 11.3 shows the idea of what a word-embedding space looks like.

Word embeddings

- Where do word embeddings come from?
  - Learn them jointly with your main task, starting with random word vectors and learn them as you learn weights.
  - Use pretrained word embeddings. There are lots of examples out there now.

Transformers

- The last big topic in this chapter is the Transformer architecture. This is a revolutionary idea that I don't really understand. It sounds cool.