Stanford CS224N NLP with Deep Learning

Winter 2021

Lecture 5 – Language Models and RNNs

Dependency Parsing (contd.)

The problems with the feature-dependent approach of transition-based parsing are as follows:

- the features are *sparse*
- the features are *incomplete*
- the computation of the features is expensive

A modification to the approach is to create a dense representation of the configuration which can be trained to include the relevant features. Distributed representations are the main step towards this.

Secondly, using softmax classifiers helps the algorithm considerably, as it allows the network to learn nonlinear functions. The hidden layers thus move around the inputs in an intermediate vector space, allowing a linear softmax to classify them.

Another, totally separate, approach to dependency parsing is the graph-based method. A graph-based system computes a score for every possible head for each word (using contextual representations of each word) and uses these scores to find an MST that represents the best parse. However, this approach is quadratic in the length of the sentence.

Neural Networks

Regularisation

Regularisation must be included in the complete loss function. We make use of L2 regularisation, adding the term

$$\lambda \sum_k \theta_k^2$$

to $J(\theta)$, which will penalise unnecessary parameters having nonzero values.

Regularisation was thought to prevent overfitting on the trainning data, but lately it is considered to merely produce models that generalise well.

Dropout

Dropout is a practice in which we zero out some (randomly selected) fixed percentage of the inputs to each neuron during training, and halve the weights of the model during testing.
This prevents feature co-adaptation (?).

Nonlinearities

Nonlinear functions intervening between the linear layers of a neural network are essential, since without them, the entire model collapses to behave like a single layer.

Common nonlinear functions introduced between layers include the sigmoid

$$\sigma(z) = \frac{1}{1 + e^{-z}},$$

the hyperbolic tangent

$$\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}},$$

the hard tanh,

$$\text{hardTanh}(z) = \begin{cases} -1 & x < -1 \\ x & -1 \leq x \leq 1 \\ 1 & x > 1, \end{cases}$$

and the reLU (the default for modern NNs)

$$\text{rect}(z) = \max(z, 0).$$

Note that the tanh function is simply a rescaled and shifted sigmoid:

$$\tanh(z) = 2\sigma(2z) - 1.$$

Parameter Initialisation

The weights of a NN must be initialised to small random values (not null matrices).

Hidden layer biases should be initialised to zero and output biases to the mean target value (or its inverse sigmoid).

All other weights can be initialised from Uniform$(-r, r)$.

Xavier initialisation uses a distribution with variance

$$\text{var}(W_i) = \frac{2}{n_{\text{in}} + n_{\text{out}}}.$$

Optimisers

Stochastic gradient descent (SGD) usually performs well, although it requires hand-tuning the learning rate.
Better results can usually be obtained by decreasing learning rates across epochs, usually exponentially

$$\beta = \beta_0 e^{-kt}.$$

"Adaptive" optimisers, like Adam, are more suited to complex nets.

Language Modelling

Language modelling is the task of predicting what word is likely to come after a given sequence of words, or more formally, predicting

$$P(x^{t+1} \mid x^1 \cdots x^t),$$

for any $x^{t+1} \in V$. A system that carries this out is called a *language model*.

n-gram Language Models

The classical method for building language models was to use n-grams, or chunks of n words. This was simply a statistical method that used n-gram counts to identify the probability distribution of x_n.
The fundamental assumption underlying n-gram models is the Markov assumption – the probability distribution of x_n depends only on $x_1, \ldots, x_{n-1}$.

A simple n-gram-based method would be to use relative co-occurrance counts:

$$P(w \mid \text{students opened their}) = \frac{\text{count(students opened their } w\text{)}}{\text{count(students opened their)}}.$$

This approach, however, runs into the problem of sparsity – many n-grams do not in fact occur in the training data.

Smoothing is a way to solve the 0-probability problem (where the numerator count is 0). We add a small δ to the count for all words in the vocabulary.
Backoff is a way to solve the undefined-probability problem (where the denominator count is 0). We make use of the $(n-1)$-gram count(opened their) instead.

n-gram models also have an issue of storage – the counts of all n-grams in the corpus need to be stored.

Neural Language Models

A simple approach would be a window-based neural model, like what we have seen for NER in the past. We take the representations of $(n-1)$ previous words,

concatenate them, and pass them through a softmax classifier to obtain an output distribution. While this solved the memory and sparsity problems that n-gram models faced, it continued to face the obstacle of *limited context* – the probabilities outputted continued to depend only on a fixed number of previous words.
Furthermore, the weight matrix W “treats” different words in the input completely differently; there is no notion of order in the processing.

This motivated the development of an architecture (more accurately, a family of architectures) capable of processing *arbitrary-length* inputs.
The basic idea behind this family of architectures, called recurrent neural networks (or RNNs), is to feed the hidden state back into itself to process together with the next input.

A basic RNN simply applies the weight matrix, bias and nonlinearity to the initial hidden state $h^{(0)}$ and subsequent inputs $e^{(t)}$:

$$h^{(t)} = \sigma(W_h h^{(t-1)} + W_e e^{(t)} + b_1).$$

The final distribution is obtained by a softmax layer:

$$y = \text{softmax}(U h^{(t)} + b_2).$$

This method has the advantage of processing arbitrarily long inputs, using information from several steps back, without increasing model size. However, the computation tends to be slow, and the persistence of information over long distance presents a challenge.