Deep Learning for NLP Summary

Daniel Saggau

2/4/2021

topics

Motivation word embeddings

Deep Learning models understand numerical representations Texts are sequences of symbolic units Step 1 in any NLP deep learning model: Turn symbols (e.g., words) into vectors

word embeddings are dense l= sparse), trainable word vectors Allows us to calculate similarities between words, e.g., with cosine similarity:

Word embeddings can be trained from scratch on supervised data: Initialize W randomly Use as input layer to some model (CNN, RNN, . . .) During training, back-propagate gradients from the model into the embedding layer, and update W Result: Words that play similar roles in the training task get similar embeddings For example: If our training task is sentiment analysis, we might expect (awesome,great) ~1

Supervised training sets tend to be small (labeled data is expensive) Many words that are relevant at test time will be infrequent or unseen at training time The embeddings of infrequent words may have low quality, unseen words have no embeddings at all. We have more unlabeled than labeled data. So let's pretrain embeddings on the unlabeled data first.

Word2Vec

Mikolov et al. (2013): Efficient estimation of word representations in vector space.

distributional hypothesis: "a word is characterized by the company it keeps"'

- Skigram

The skipgram task is to maximize the likelihood of the context words, given their center word: Expressed as a negative log likelihood loss:

- CBOW Task

The continuous bag of words (CBOW) task is to maximize the likelihood of the center words, given their context words: Expressed as a negative log likelihood loss:

- Naive softmax model
- Hierachical softmax model
- Negative sampling model

Instead of predicting a probability distribution over whole vocabulary, predict binary probabilities for a small number of word pairs. This is not a language model anymore. . . . . . but since we only care about the word vectors, and not the skip gram/CBOW predictions, that's not really a problem.

- FastText

Even if we train Word2Vec on a very large corpus, we will still encounter unknown words at test time

Extension of Word2Vec by **Bojanowski et al. (2017):** Enriching Word Vectors with Subword Information.

Design choice: Fine-tune embeddings on task or freeze them?

- Pro fine-tuning: Can learn/strengthen features that are important for the task
- Pro freezing: We might overfit on training set and mess up the relationships between seen and unseen words
- Both options are popular

Applications of pre-trained word embeddings

CNN

An architecture is an abstract design for a neural network

Examples of architectures: * Fully Connected Neural Networks * Convolutional Neural Networks (CNNs) * Recurrent Neural Networks (RNNs) * Transformers (self-attention)

The choice of architecture is often informed by assumptions about the data, based on our domain knowledge

translation invariance: The features that make up a meaningful object do not depend on that object's absolute position in the input.

sequentiality: NLP: Sentences should be processed left-to-right or right-to-left. This one is falling out of fashion, since people are replacing recurrent neural networks with self-attention networks.

Are these assumptions true?

Of course not. But they are a good way of thinking about why certain architectures are popular for certain problems. Also, for limiting the search space when deciding on an architecture for a given project.

Convolutational layers

Technique from Computer Vision (esp. object recognition), by LeCun et al., 1998 Adapted for NLP (e.g., Kalchbrenner et al., 2014) Filter banks with trainable filters So what is a filter? What is a filter bank?

1-D convolution

Let's say that we want to train a linear regression model to predict some variable of interest from each datapoint. Since the raw data is noisy, it makes sense to smoothe it with our rolling average filter first: $y = wh + b; h = (f \times x)jjjj$ But maybe we should weight the datapoints in our 7-day window differently? Maybe we should give a higher weight to datapoints close to the current day j? We can manually engineer a filter that we think is better, for example: # TODO Alternative: Let the model choose the optimal filter parameters, based on the training data. How? Gradient descent!

Bias and nonlinearities: * In a real CNN, we usually add a trainable scalar bias b, and we apply a nonlinearity g (such as the rectified linear unit): K #TODO Edge cases: * Possibel strategies for when the filter overlaps with the edges (beginning/end) of x: * Outputs where filter overlaps with edge are undefined, i.e., h has fewer dimensions than $x(K - 1$ fewer, to be exact) * Pad x with zeros before applying the filter * Pad x with some other relevant value, such as the overall average, the first/last value, … Stride: * The stride of a convolutional layer is the "step size" with which the filter is moved over the input * We apply f to the j'th window of x only if j is divisible by the stride. * In NLP, the stride is usually 1.

Convolution with more than one axis

Extending the convolution operation to tensors with more than one axis is straightforward. Let $X \in RJ1Ö...ÖJL$ be a tensor that has L axes and let $F \in RK1Ö...ÖKL$ be a filter. * The dimensionalities of the filter axes are called filter sizes or kernel sizes ("kernel width", "kernel height", etc.) * From now on, we assume that the filter is applied to a symmetric window around position j, not just to positions to the left of j. (The rolling average was a special scenario, #TODO are not available on day j.) Then the output $H = F \times X$ is a tensor with L axes, where

Channels

So far, we have assumed that each position in the input (each day or each pixel) contains a single scalar. Now, we assume that our input has M features (“channels”) per position, i.e., there is an additional feature axis:$X \in RJ1Ö...ÖJLÖM$ Example: * M = 3 channels per pixel in an RGB (red/green/blue) image * In NLP: dimensionality of pretrained word embeddings Then our filter gets an additional feature axis, which has the same dimensionality as the feature axis of X:

TODO

During convolution, we simply sum over this new axis as well

TODO

Filter banks

A filter bank is a tensor that consists of N filters, which each have the same shape. The filters of a filter bank are applied to X independently, and their outputs are stacked (they form a new axis in the output). So let this be our input and filter bank:

Then our output is a tensor of shape $H \in RJ1Ö...ÖJL, N$ * (assuming that we are padding the first L axes of X to preserve their dimensionality in H) where:

Assumptions behind convolution Translation invariance: Same object in different positions. * Parameter sharing: Filters are shared between all positions. Locality: Meaningful objects form coherent areas * In a single convolutional layer, information can travel no further than a few positions (depending on filter size) Hierarchy of features from simple to complex: * In computer vision, we often apply many convolutional layers one after another * With every layer, the information travels further, and the feature vectors become more complex * Pixels → edges → shapes → small objects → bigger objects

Pooling layers

Pooling layers are parameter-less Divide axes of H (excluding the filter/feature axis) into a coarse-grained “grid”. Aggregate each grid cell with some operator, such as the average or maximum. Example (shown without a feature axis): When pooling, we lose fine-grained information about exact positions In return, pooling allows us to aggregate features from a local neighborhood into a single feature. * For example, if we have a neighborhood where many “dog features” were detected (meaning, shapes that look like parts of a dog), we want to aggregate that information into a single “dog neuron”

In computer vision, we often apply pooling between convolutional layers. Repeated pooling has the effect of reducing tensor sizes.

CNNs in NLP

Images are 2D, but text is a 1D sequence (of words, n-grams, etc). Words are usually represented by M-dimensional word embeddings (e.g., from Word2Vec) So on text, we do 1-D convolution with M input features: * Input matrix: $XinRJÖM$ * Filter bank of $Nfilters : FinRKÖMÖN; K < J$ * Output matrix: $HinRJÖN$ * (assuming that we padded X with zeros along its first axis) Usually, CNNs in NLP are not as deep as CNNs in Computer Vision – often just one convolutional layer.

Pooling is less frequently used in NLP. If the task is a word-level task (i.e., we need one output vector per word), we can simply use the output of the final convolutional layer – no pooling needed. If the task is some sentence-level task (i.e., we need a single vector to represent the entire sentence), we usually do a “global” pooling step over the entire sentence after the last convolutional layer:

RNN

Assumption: Text is written sequentially, so our model should read it sequentially “RNN”: class of Neural Network architectures that process text sequentially (left-to-right or right-to-left) Generally speaking:

- Internal “state” h
- RNN consumes one input $x(j)$ per time step j
- Update function: $h(j) = f(x(j), h(j-1); \theta)$
- where parameters θ are shared across all time steps

Vanilla RNN

TODO

Backpropagation through time RNNs are trained via “backpropagation through time” To understand how this works, imagine the RNN as a Feed-Forward Net (FFN), whose depth is equal to the sentence length For now, let’s pretend that every time step (every layer) has its own (j) dummy parameters θ , which are identical copies of theta

LSTM

Attention

Transformers

Hyperparameter Optimization

Transfer Learning

BERT

Alternatives to BERT

T5

XLNet

Benchmarks

GPT: Ethical considerations In general, a machine does not know (and probably does not care) what consequences its words will have in the real world. * Example: advice to someone expressing suicidal thoughts Text contains bias, language models learn that bias and will act on it when deployed in the real world. * Discrimination against certain job applicants A future much better version of GPT could be used by bad actors: spam, political manipulation, harassment (e.g., on social media), academic fraud etc. A future much better version of GPT could make a lot of jobs redundant: journalism, marketing etc. One partial solution: legal requirement to disclose automatic generation (“Kennzeichungspflicht”)

Multilingual models