Natural Language Processing (2024)

L2 (advanced): Under the hood of the CBOW classifier

Marco Kuhlmann

In Lecture 2.3 you learned about the CBOW classifier. This classifier is easy to implement in PyTorch with its automatic differentiation magic; but it is easy also to forget about what is going on under the hood. Your task in this lab is to implement the CBOW classifier without any magic, using only a library for vector operations (NumPy).

Problem description

To get a Pass on this lab, you must submit a complete implementation of a CBOW classifier that, over 10 different random seeds, achieves an average accuracy of at least 30% on the provided data. Your code must report the accuracy when run from the command line, in the same way as the starter code does.

Detailed information

Make sure that you understand the CBOW classifier (Lecture 2.03) before starting with this lab. It is also a good idea to revisit the notebook with the PyTorch implementation of the CBOW classifier (in the course GitLab).

Structure of the starter code

The starter code for this lab consists of a single Python file, `cbow-starter.py`. This file contains two unfinished code blocks: one class for the CBOW model, and one function for training. In addition, the code provides some helper functions, including an efficient NumPy implementation of the softmax function. You are free to add your own code to the starter code, but you are not allowed to import any libraries other than NumPy (which is already imported).

Data representation

A batch of training data is represented as a pair of NumPy arrays $(\boldsymbol{X}, \boldsymbol{y})$ where $\boldsymbol{X}$ is a matrix containing the word ids of each review in the batch and $\boldsymbol{y}$ is a vector containing the gold-standard classes for the reviews. Let us write B for the total number of reviews in the batch and L for the maximal length of each review; then $\boldsymbol{X}$ has shape $B \times L$, and $\boldsymbol{y}$ has length B. Just as the notebook, the starter code caps the length of each review at $L = 20$ and right-pads shorter reviews with zeroes. (The code is set up in such a way that no real word gets word id 0.)

Learning the parameters

Recall that the CBOW model has three trainable parameters: a matrix $\boldsymbol{E}$ holding the embeddings, a matrix $\boldsymbol{W}$ holding the weights of the linear layer, and a vector $\boldsymbol{b}$ holding the biases. All of these parameters are initialized randomly. For training you will have to implement a simple version of minibatch stochastic gradient descent that updates the parameters. The necessary updates can be computed by backpropagating cross-entropy loss through the network structure. The following paragraphs derive the relevant equations.

Updating the linear layer Given a batch of data $\boldsymbol{X}$, let us write $\boldsymbol{E}[\boldsymbol{X}]$ to denote the three-dimensional tensor obtained by replacing each word id in $\boldsymbol{X}$ with its corresponding word embedding in $\boldsymbol{E}$. Writing E for the dimensionality of the embeddings, $\boldsymbol{E}[X]$ has shape $B \times L \times E$. The input to the linear layer is the element-wise mean of the word embeddings in each row of $\boldsymbol{E}[\boldsymbol{X}]$; this is a $(B \times E)$-matrix that we will denote by $\bar{\boldsymbol{E}}[\boldsymbol{X}]$. Backpropagation yields the following error value and updates:

$$\boldsymbol{\delta}_1 = \operatorname{softmax}(\bar{\boldsymbol{E}}[\boldsymbol{X}]\boldsymbol{W} + \boldsymbol{b}) - \boldsymbol{y}^* , \quad \boldsymbol{W} \coloneqq \boldsymbol{W} - \frac{\eta}{B} \cdot (\bar{\boldsymbol{E}}[\boldsymbol{X}])^\top \boldsymbol{\delta}_1 , \quad \boldsymbol{b} \coloneqq \boldsymbol{b} - \frac{\eta}{B} \cdot \mathbf{1}^\top \boldsymbol{\delta}_1$$

In these equations, η denotes the learning rate, $\boldsymbol{y}^*$ denotes the one-hot-encoded version of $\boldsymbol{y}$ (a matrix of shape $B \times K$), $\mathbf{1}$ denotes the column vector of ones of length B, and $\boldsymbol{A}^\top$ and denotes the transpose of the matrix $\boldsymbol{A}$. Note that these updates are the same as in softmax regression.

Updating the embeddings For the embeddings we get the error value

$$\boldsymbol{\delta}_2 = \frac{1}{L} \cdot \boldsymbol{\delta}_1 \boldsymbol{W}^\top$$

This is a matrix of shape $B \times E$; for each of the B reviews in the batch, it holds the update that should be applied to each of the L embedding vectors in that review.

Initialization and hyperparameters

We suggest that you initialize the embeddings with random samples drawn from a normal distribution with mean zero and scale $\sigma = 0.1$, and the weights and biases with random samples drawn from a uniform distribution over the interval $(-\sqrt{k}, \sqrt{k})$, where $k = 1/E$. The following values are good starting points for the hyperparameters: embedding width 30, 20 epochs of training, batch size 24, learning rate $\eta = 0.1$. If these values do not get you to the 30% threshold, you can see whether you can improve the model's accuracy by tuning the hyperparameters.