Assignment 1: *n*-gram LM & Neural Classifier

Release date: 01/27 (Tue)
Due date: 02/10 (Tue) 05:59PM
Latest possible due date with late days: 02/13 (Fri) 05:59PM

This assignment consists of two parts. The starting code for the assignment can be found in this repository - https://github.com/akshat57/cs288-sp26-a1

NOTE : You are only allowed to make 5 submissions to gradescope. No further submissions will be allowed and the score of your final submission will be kept, so use them sparingly.

Part 1: Train an n-gram language model (30 points)

In Part 1, you will implement and train an n-gram LM and a neural n-gram LM, using wikitext102.

Complete the following notebook https://github.com/akshat57/cs288-sp26-a1/blob/main/Part1.ipynb, and submit the following files on gradescope:

Deliverables:

- Part1.ipynb
- bigram predictions.npy (15 points)
- neural trigram predictions.npy (15 points)

Refer to the following resources for help with this assignment:

- PyTorch: https://pytorch.org/tutorials/beginner/deep_learning_60min_blitz.html
- N-Gram Language Models: https://web.stanford.edu/~jurafsky/slp3/3.pdf
- Neural Language Models: https://web.stanford.edu/~jurafsky/slp3/7.pdf

Part 2: Train a perceptron classifier (70 points)

SUBMISSION NOTE: Submit the final code for part 2 in a zip file (Part2.zip) containing the final code of the final submission. We are looking for your final codebase used to generate gradescope submission for MLP.

You should create two separate files, one for each of the two datasets with the final architecture and hyperparameters baked in. We will *only* run the following commands to evaluate your code:

```
python3 perceptron_newsgroups.py
python3 perceptron_sst.py
python3 multilayer_perceptron_newsgroups.py
python3 multilayer_perceptron_sst.py
```

We will randomly select submissions and run your codebase with only the above two commands. We expect the scores to be within 1% of the gradescope scores.

Use of pre-trained embeddings is NOT allowed for this assignment.

Starter Code : https://github.com/akshat57/cs288-sp26-a1/tree/main/Part2

Deliverables:

- Perceptron_newsgroups_test_predictions.csv (15 points)
- Perceptron_sst2_test_predictions.csv (15 points)
- Mlp_newsgroups_test_predictions.csv (20 points)
- Mlp_sst2_test_predictions.csv (20 points)
- Part2.zip (codebase)

Expected Scores:

1. The target scores on the test set is 70% for perceptron and 75% for MLP.
2. Note that there will be a big gap (11-13%) between the dev accuracy and the test accuracy on gradescope (test accuracy will be lower)

In Part 2, you will (1) implement and train a simple linear perceptron model from scratch, and (2) implement a neural bag-of-words (NBOW) multilayer perceptron (MLP).

The starter repository contains training and development data for the Stanford Sentiment Treebank (sst2) and 20 Newsgroups (newsgroups) datasets, in the data/ directory. You will also find the test data for these two datasets without the labels.

The starter repository contains data and evaluation utilities in utils.py, feature extractors in featurize.py, and finally classification models in perceptron.py and multilayer_perceptron.py. We recommend filling in #TODO in this order. There are corresponding unit tests.

Model 1: Perceptron (30 points)

Implement a simple linear perceptron model from scratch. Remember, instead of having to compute the derivative over the entire training set, the perceptron simply picks examples in sequence, and tries to classify them given the current weight vector. If it gets them right, it simply moves on to the next examples, otherwise it updates the weight vector with the difference of the feature counts in the correct labels and in the prediction. You are required to only use simple Python code for this task, without any other packages/libraries. An efficient implementation with bag-of-words features only should take <1 seconds/epoch for the sentiment analysis dataset on a modern laptop.

For the perceptron model, you need to develop features to train your models with. At the least, you must experiment with bag-of-words features and design at least two feature sets beyond that for each dataset that improve your performance (a feature set can be counted for both, if it's used this way with positive impact on performance).

Features are individual measurable properties or characteristics of the example used as input to a model. Some examples that are not considered as additional feature sets: bag-of-words unigrams or higher order n-grams (you already have this), preprocessing to remove words, or filtering the vocabulary (to remove rare words). These are all potentially useful techniques, which you may want to experiment with, but they are not considered different feature sets. Please try to think why the above are not considered features.

1. Bag of words, n-grams. Start with a bag-of-words classifier. You may want to experiment with using binary (presence/absence) features instead of count features and excluding stopwords (e.g., the, is, a) to improve your accuracy.[1] How does performance change as you vary the size of n used in your word n-grams? Does it make sense to filter out some features based on counts in the training data?
2. Your own features. Develop and use task-specific features of your own design in order to improve your accuracy.

Model 2: Multilayer Perceptron (40 points)

Implement a neural bag-of-words (NBOW) multilayer perceptron (MLP). You will use PyTorch for this model, but no other packages. Auxiliary packages to PyTorch (e.g., for text processing) are also not allowed). The allowed PyTorch package is specified in the requirements.txt file in the assignment repository. One big advantage of deep learning frameworks, such as PyTorch, is in abstracting away the details of error backpropagation, allowing you to focus on designing your network's architecture.[2]

Matrix operations in deep learning frameworks are very similar to NumPy. Your MLP must have multiple hidden layers, although the exact number is for you to determine through your

[1] The file stopwords.txt in the starter repository provides some English stop words. You may extend this list through your analysis.
[2] If you want to learn about backpropagation, see this.

data-driven development process. Hint: your feature vectors will probably be sparse, so it is more efficient (and even critical) to use embedding look-up tables rather than large matrix multiplications. An efficient implementation with a moderately sized MLP should take <5 seconds/epoch on a modern laptop.

When designing your MLP, experiment with different activation functions. Activation functions add nonlinearity to your MLP, allowing it to capture more complex aspects of your training data. You should use at least ReLU, sigmoid, and tanh activation functions.

In your final layer, you will want to transform the output of your network to a probability distribution (using the softmax function) and compare this distribution to your training labels.

Finally, you should experiment with different optimizers and learning rates to see if they allow faster training and/or better results. The standard approach is to use gradient descent, but PyTorch, for example, comes with others built in, such as AdaGrad and ADAM.

You will need to explicitly experiment with batching your MLP computation. Please conduct explicit experiments on a GPU to time your learning and inference speed with and without batching. Measure your speed in how many wall-time seconds it takes you to process 1,000 examples. You can conduct these experiments with a subset of the data. The goal is to measure computation speed, not model accuracy. Batching is expected to have no impact on the computation itself (i.e., the computation should be identical regardless of the batch size), but show speedups up to a certain batch size. You just need to make sure you get enough measurements to reliably estimate your speed, ideally averaging across many measurements and reporting both average and standard deviation. Your benchmarking must be consistent between no batching (batch size = 1) and batching. Experiment with multiple batch sizes.
This experiment is only required for the MLP.

Additional Details:

Sentiment Analysis: Sentiment classification is the task of determining the sentiment - often positive, negative or neutral - expressed in a given text. In general, this requires using a variety of cues such as the presence of emotionally charged words such as “vile” or “amazing,” while taking into account the full context of word use or phenomena like negation or sarcasm. In this assignment, you will write classifiers for the Stanford Sentiment Treebank dataset, which contains snippets taken from Rotten Tomatoes movie reviews, where the sentiment is aligned directly with the review score. You will train classifiers on a filtered version of the dataset containing only full sentences and with neutral reviews removed, reducing the task to binary classification of positive and negative sentiment.

Newsgroup Classification: The 20 Newsgroups dataset contains 18,846 newsgroup documents written on a variety of topics. You will use the text of each document to predict its newsgroup. Unlike the binary sentiment analysis task, each document could belong to one of

twenty newsgroups. Additionally, many of these newsgroups share similar themes, such as computing, science, or politics (see Table 1 for the full list). However, the distributions of words across each of these newsgroups are also fairly distinctive. For example, a document that uses the names of weapons will likely be in talk.politics.guns, a document mentioning “computer” will probably be in a comp.* group, and if a document uses the word “ice,” it was likely written for rec.sport.hockey rather than talk.politics.mideast.

Third-party Packages: This assignment emphasizes building from scratch when possible. The two models allow different third-party packages. Please follow these directions carefully. There is a heavy penalty for using packages beyond these specified below.

Development Environment and Third-party Tools: All allowed third-party tools are specified in the `requirements.txt` file in the assignment repository. The goal of the assignment is to gain experience with specific methods, and therefore using third-party tools and frameworks beyond these specified is not allowed. Please follow these directions carefully. We will penalize heavily for using packages beyond these specified below. Please consult the course generative AI policy with regard to using such tools.

You may only `import` packages that are specified in the `requirements.txt` file or that come with Python’s Standard Library. The version of Python allowed for use is 3.10.x. Do not use older or newer version. We strongly recommend working within a fresh virtual environment for each assignment. For example, you can create a virtual environment using conda and install the required packages:

```
conda create -n cs288a1 python=3.10
conda activate cs288a1
python -m pip install -r requirements.txt
```

Acknowledgement

Part 1 is taken and adapted from Dan Klein’s previous CS 288 offering. Part 2 is taken and adapted from Yoav Artzi (Cornell University)’s CS 5740.